SDA 6000

Teletext Decoder with Embedded 16-bit Controller M2

ICs for Consumers

Never stop thinking.

Edition 2000-06-15

**Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
D-81541 München, Germany**

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide (see address list).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Users Manual, Version 2.1, June 2000

SDA 6000

Teletext Decoder with Embedded 16-bit Controller

ICs for Consumers

Never stop thinking.

SDA 6000

Revision History:	**Current Version: 2000-06-15**
Previous Version:	08.99
Page	Subjects (major changes since last revision)
	Complete Update of Controller & Peripheral Spec --> Detailed Version
	ASC: Autobaud Detection Feature included
	IC: New Description
	GPT: New Description
	IIC changed to I^2C

For questions on technology, delivery and prices please contact the Infineon Technologies Offices in Germany or the Infineon Technologies Companies and Representatives worldwide:
see our webpage at http://www.infineon.com

Contents
Overview

Table of Contents

Table of Contents **Page**

Table of Contents **Page**

Table of Contents **Page**

Table of Contents

List of Figures

List of Figures

List of Figures

Preface

M2 is a 16-bit controller based on Infineon's C16x core with embedded teletext and graphic controller functions. M2 can be used for a wide range of TV and OSD applications. This document provides complete reference information on the hardware of M2.

Organization of this Document

This Users Manual is divided into 14 chapters. It is organized as follows:

- Chapter 1, Overview
  Gives a general description of the product and lists the key features.
- Chapter 2, Pin Description
  Lists pin locations with associated signals, categorizes signals according to function, and describes signals.
- Chapter 3, Architectural Overview
  Gives an overview on the hardware architecture and explains the dataflow within M2.
- Chapter 4, C16X Microcontroller
  Gives a detailed explanation of the 16-bit µC architecture.
- Chapter 5, Interrupt and Trap Functions,
  Explains the powerful C166 Interrupt facilities.
- Chapter 6, System Control & Configuration
  Describes how to configure and control the complete µC system and the Power Management Unit.
- Chapter 7, Peripherals
  Describes the peripherals (serial buses and timers modules) of the micro.
- Chapter 8 & 9, Clock System & Sync System
  Describes how clocks & syncs for the display generator are generated.
- Chapter 10 & 11, Display Generator and D/A Converter
  Explains the architecture and programming possibilities of the unit which generates the RGB signals.
- Chapter 12, Acquisition and Slicer
  Describes features and functionality of the data caption unit.
- Chapter 13, Register Overview
  Summarizes all HW-registers of M2.
- Chapter 14, Electrical Characteristics
  Lists all important AC and DC values and the maximum operating conditions of M2.

Related Documentation

For easier understanding of this specification it is recommended to read the documentation listed in the following table. Moreover it gives an overview of the software drivers which are available for M2.

Document Name	Document Purpose
Appl. Note “Initialization and Bootstraploader of M2”	System integration support

1 Overview

M2 is designed to provide absolute top performance for a wide spectrum of teletext and graphic applications in standard and high end TV-sets and VCRs. M2 contains a data caption unit, a display unit and a high performance Infineon C16x based microcontroller (so that M2 becomes a one chip TV-controller) an up to level 3.5 teletext decoder and display processor with enhanced graphic accelerator capabilities. It is not only optimized for teletext usage but also, due to its extremely efficient architecture, can be used as a universal graphic engine.

M2 is able to support a wide range of standards like PAL, NTSC or applications like Teletext, VPS, WSS, Chinatext, Closed Caption and EPG (Electronic Program Guide). With the support of a huge number of variable character sets and graphic capabilities a wide range of OSD applications are also open for M2.

A new flexible data caption system enables M2 to slice most data, making the IC an universal data decoder. The digital slicer concept contains measurement circuitries that help identify bad signal conditions and therefore support the automatic compensation of the most common signal disturbances. M2's enhanced data caption control logic allows individual programming, which means that every line can carry an individual service to be sliced and stored in the memory.

The display generation of M2 is based on frame buffer technology. A frame buffer concept displays information which is individually stored for each pixel, allowing greater flexibility with screen menus. Proportional fonts, asian characters and even HTML browsers are just some examples of applications that can now be supported.

Thus, with the M2, the process of generation and display of on-screen graphics is split up into two independent tasks. The generation of the image in the frame buffer is supported by a hardware graphics accelerator which frees the CPU from power intensive address calculations. The graphics accelerator 'prints' the characters, at the desired 'screen' position, into the frame buffer memory based on a display list provided by the software.

The second part of the display generator (the screen refresh unit) then reads the frame buffer according to the programmed display mode and screen refresh rate and converts the pixel information into an analog RGB signal.

Furthermore, M2 has implemented an RGB-DAC for a maximum color resolution of state-of-the-art up to 65536 colors, so that the complete graphic functionality is implemented as a system on chip. The screen resolution is programmable up to SVGA, to cover today's and tomorrow's applications, only limited by the available memory (64 Mbit) and the maximum pixel clock frequency (50 MHz).

The memory architecture is based on the concept of a unified memory - placing program code, variables, application data, bitmaps and data captured from the analog TV signal's vertical blanking interval (VBI) in the same physical memory. M2's external bus interface

supports SDRAMs as well as ROMs or FLASH ROMs. The organization of the memory is linear, so that it is easy to program the chip for graphic purposes.

The SW development environment “MATE” is available to simplify and speed up the development of the software and displayed information. **MATE** stands for: **M**2 **A**dvanced **T**ool **E**nvironment. Using MATE, two primary goals are achieved: shorter Time-to-Market and improved SW qualitiy. In detail:

- Re-usability
- Target independent development
- Verification and validation before targeting
- General test concept
- Documentation
- Graphical interface design for non-programmers
- Modular and open tool chain, configurable by customer

MATE uses available Infineon C166 microcontroller family standard tools as well as a dedicated M2 tools.

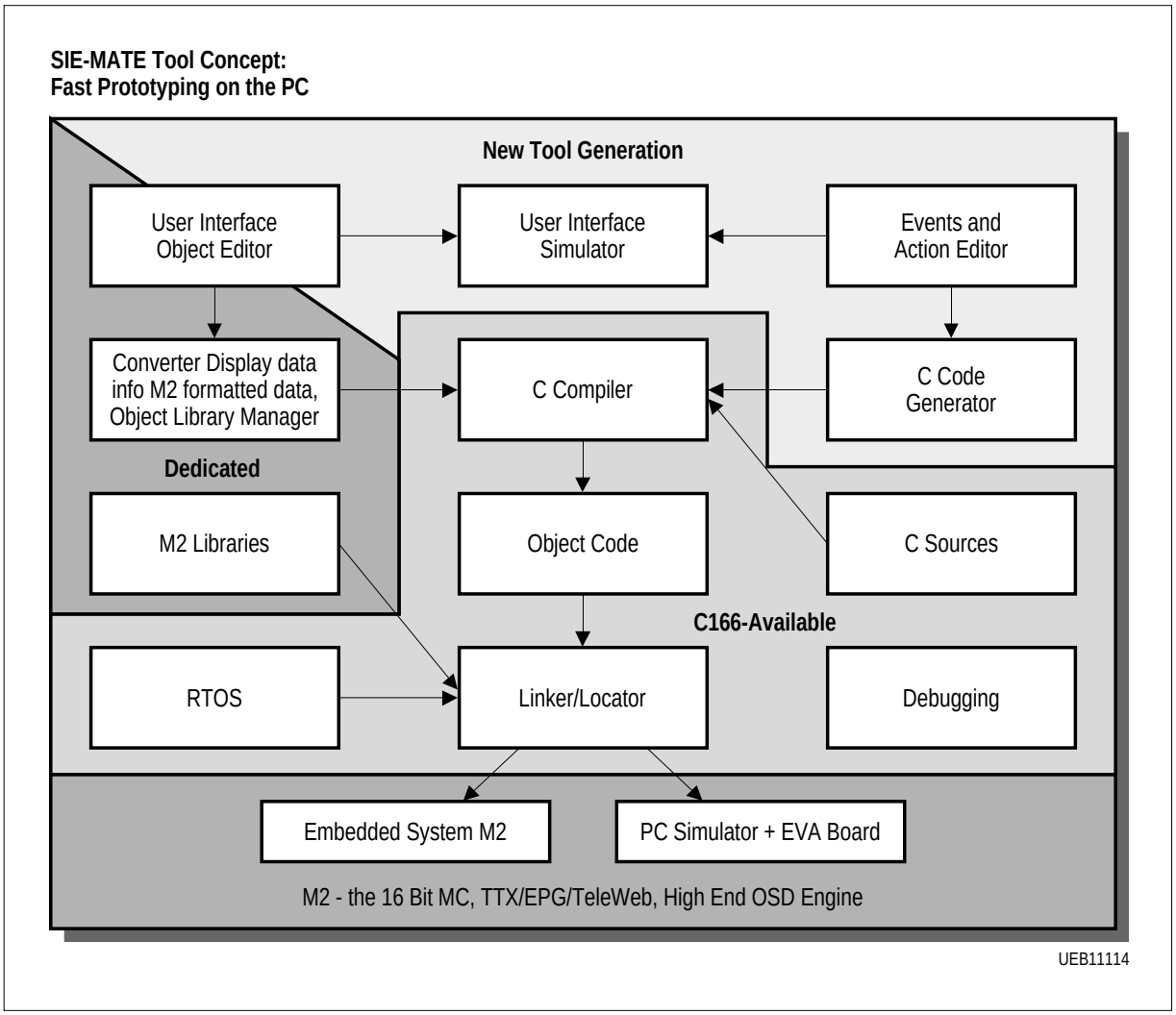


Figure 1-1 M2 Tool Flow

Standard Tool Chain

For the M2 software development (documentation, coding, debugging and test) the Infineon C166 microcontroller family standard tools can be used: These are ASCII editor, structogram editor, compiler, assembler, linker. Debugging is supported by low-priced ROM-Monitor debuggers or the OCDS (On Chip Debug Support) debugger.

M2 Dedicated Tools

Special tools are primarily available for platform independent M2 software development and secondly to generate data and control code for the M2 graphical user interface (GDI) without having knowledge of M2 hardware. These are:

- Display Generator Simulator
- Teletext Data Slicer Simulator
- GDI (Graphical Device Interface)
- Teletext Decoder and Display Software for Level 1.5 and Level 2.5
- Mate Display Builder for management, editing, handling and generation of all necessary data to display OSD's
- Evaluation Board Simulator to connect a C166 EVA Board to the M2 simulation

The M2 software is written in ANSI-C to fulfil the platform independent development. The ported software is code and runtime optimized. The layers of the modular architecture are separated by application program interfaces which ensure independent handling of the modules.

Teletext Decoder with Embedded 16-bit Controller M2

SDA 6000

Version 2.0

CMOS

1.1 Features

P-MQFP-128-2

General

- Level 1.5, 2.5, 3.5 WST Display Compatible
- Fast External Bus Interface for SDRAM (Up to 8 MByte) and ROM or Flash-ROM (Up to 4 MByte)
- Embedded General Purpose 16 Bit CPU (Also used as TV-System Controller, C16x Compatible)
- Display Generation Based on Pixel Memory
- Program Code also Executable From External SDRAM
- Embedded Refresh Controller for External SDRAM
- Enhanced Programmable Low Power Modes
- Single 6 MHz Crystal Oscillator
- Multinorm H/V-Display Synchronization in Master or Slave Mode
- Free Programmable Pixel Clock from 10 MHz to 50 MHz
- Pixel Clock Independent from CPU Clock
- 3 × 6 Bits RGB-DACs On-Chip
- Supply Voltage 2.5 and 3.3 V
- P-MQFP-128 Package

Microcontroller Features

- 16-bit C166-CPU Kernel (C16x Compatible)
- 60 ns Instruction Cycle Time
- 2 KBytes Dual Ported IRAM
- 2 KBytes XRAM On-chip
- General Purpose Timer Units (GPT1 and GPT2).
- Asynchronous/Synchronous Serial Interface (ASC0) with IrDA Support. Full-duplex Asynchronous Up To 2 MBaud or Half-duplex Synchronous up to 4.1 MBaud.

Type	Package
SDA 6000	P-MQFP-128-2

- High-speed Synchronous Serial Interface (SSC). Full- and Half-duplex synchronous up to 16.5 Mbaud
- 3 Independent, HW-supported Multi Master/Slave I^2C Channels at 400 Kbit/s
- 16-Bit Watchdog Timer (WDT)
- Real Time Clock (RTC)
- On Chip Debug Support (OCDS)
- 4-Channel 8-bit A/D Converter
- 42 Multiple Purpose Ports
- 8 External Interrupts
- 33 Interrupt Nodes

Display Features

- OSD size from 0 to 2046 (0 to 1023) pixels in horizontal (vertical) direction
- Frame Buffer Based Display
- 2 HW Display Layers
- Support of Double Page Level 2.5 TTX in 100 Hz Systems
- Support of Transparency for both Layers Pixel by Pixel
- User Programmable Pixel Frequency from 10.0 MHz to 50 MHz
- Up to 65536 Displayable Colors in one Frame
- DMA Functionality
- Graphic Accelerator Functions (Draw Lines, Draw and Fill Rectangle, etc.)
- 1, 2, 4 or 8-bit Bitmaps (up to 256 out of 4096 colors)
- 12 bit/16 bit RGB Mode for Display of up to 65535 Colors
- HW-support for Proportional Characters
- HW-support for Italic Characters
- User Definable Character Fonts
- Fast Blanking and Contrast Reduction Output

Acquisition Features

- Two Independent Data Slicers (One Multistandard Slicer + one WSS-only Slicer)
- Parallel Multi-norm Slicing (TTX, VPS, WSS, CC, G+)
- Four Different Framing Codes Available
- Data Caption only Limited by available Memory
- Programmable VBI-buffer
- Full Channel Data Slicing Supported
- Fully Digital Signal Processing
- Noise Measurement and Controlled Noise Compensation
- Attenuation Measurement and Compensation
- Group Delay Measurement and Compensation
- Exact Decoding of Echo Disturbed Signals

1.2 Logic Symbol

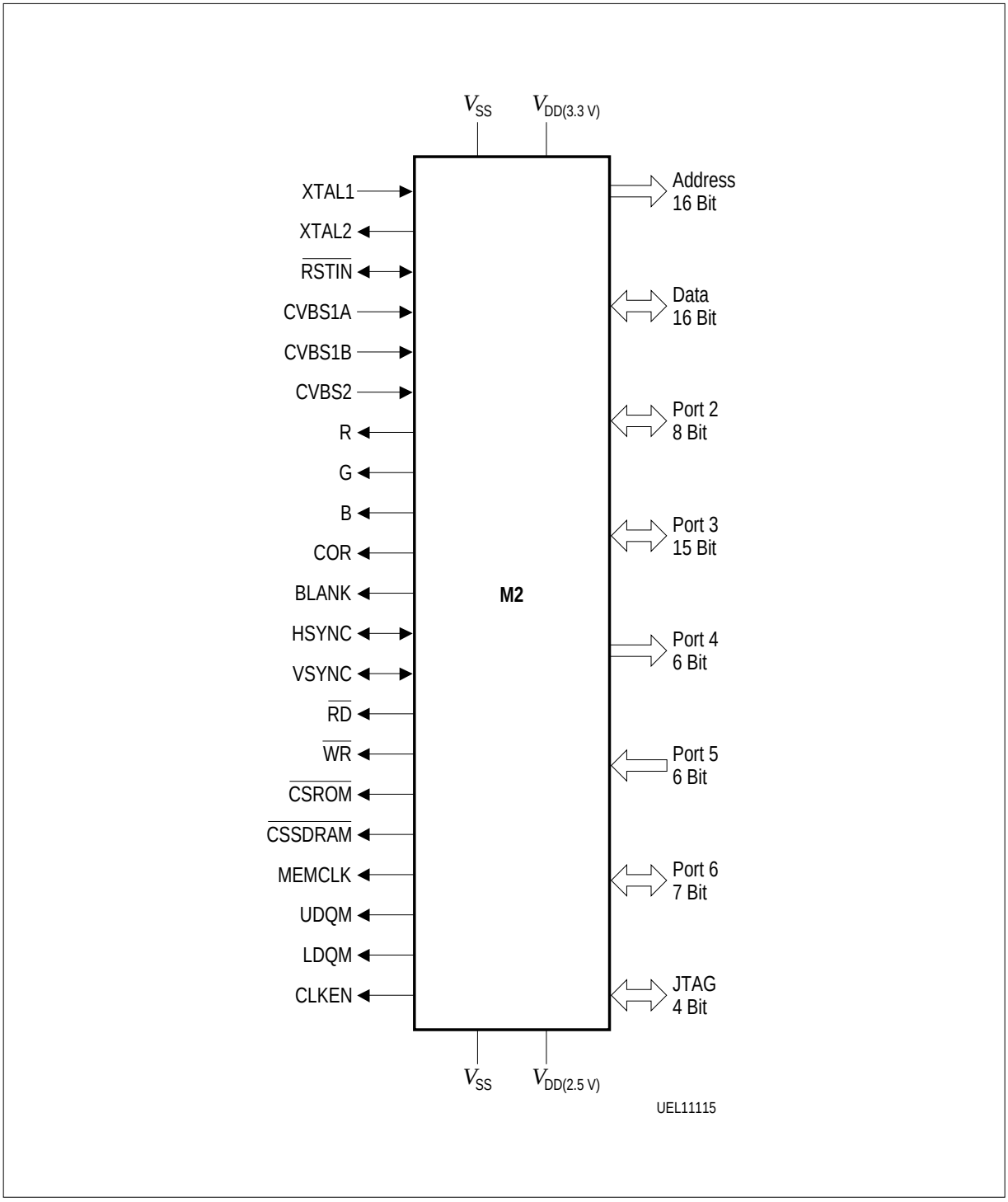


Figure 1-2 Logic Symbol

Pin Description

2 Pin Descriptions

2.1 Pin Diagram (top view)

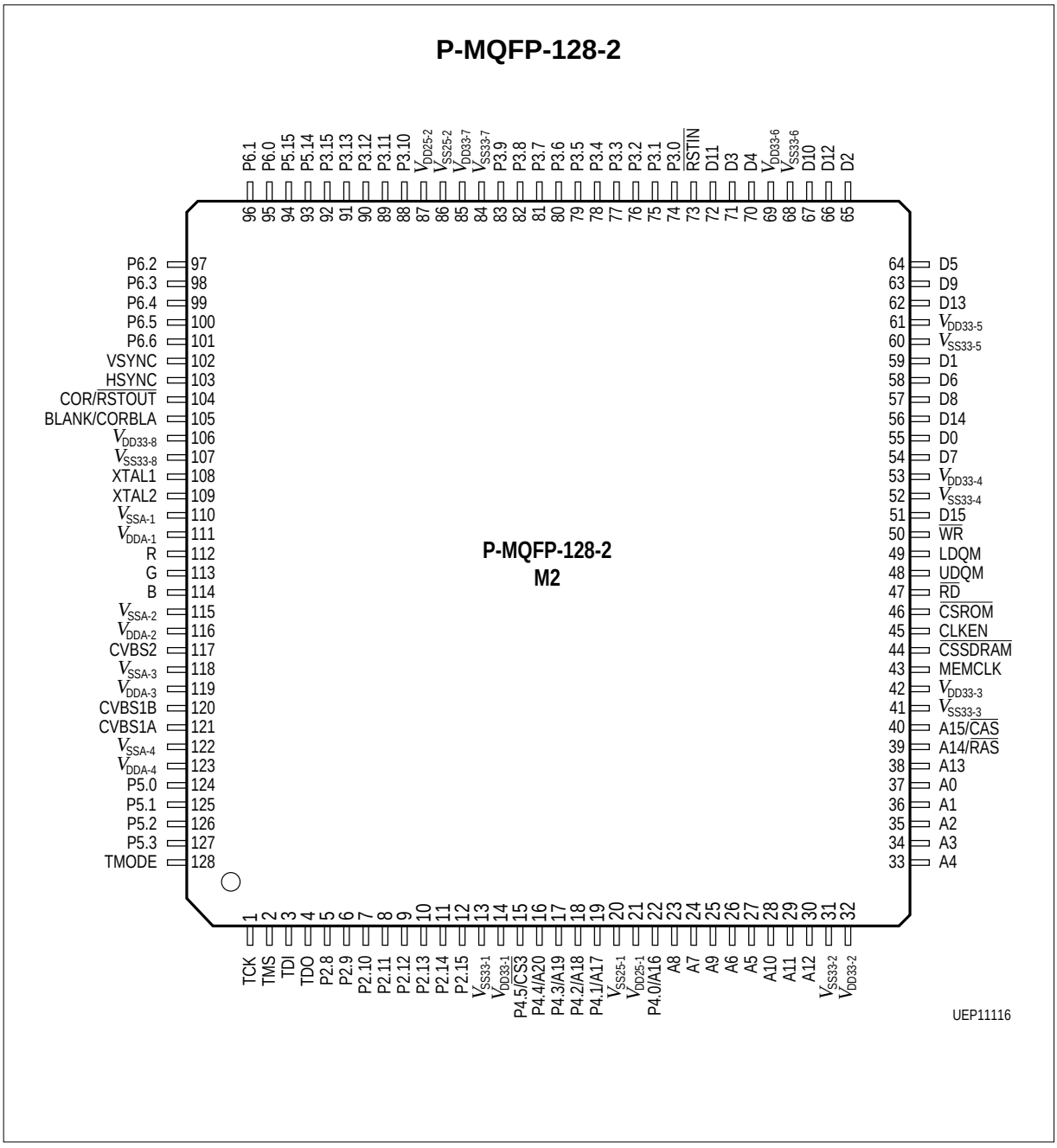


Figure 2-1 Pin Configuration

2.2 Pin Definitions and Functions

Table 2-1 Pin Definition and Functions

Pin No.	Pin Name	Second Function	Dir.	Function
37	A0	R0/C0	O	Address bit (All addresses are word addresses)/SDRAM Address bit
36	A1	R1/C1	O	Address bit/SDRAM address bit
35	A2	R2/C2	O	Address bit/SDRAM address bit
34	A3	R3/C3	O	Address bit/SDRAM address bit
33	A4	R4/C4	O	Address bit/SDRAM address bit
27	A5	R5/C5	O	Address bit/SDRAM address bit
26	A6	R6/C6	O	Address bit/SDRAM address bit
24	A7	R7/C7	O	Address bit/SDRAM address bit
23	A8	R8	O	Address bit/SDRAM address bit
25	A9	R9	O	Address bit/SDRAM address bit
28	A10	R10	O	Address bit/SDRAM address bit
29	A11	R11	O	Address bit/SDRAM address bit
30	A12	R12	O	Address bit/SDRAM address bit
38	A13	R13	O	Address bit/SDRAM address bit
39	A14	$\overline{RAS}$	O	Address bit/ Row address strobe for SDRAM access
40	A15	$\overline{CAS}$	O	Address bit/ Column address strobe for SDRAM access
55	D0	–	I/O	Data bit
59	D1	–	I/O	Data bit
65	D2	–	I/O	Data bit
71	D3	–	I/O	Data bit
70	D4	–	I/O	Data bit
64	D5	–	I/O	Data bit
58	D6	–	I/O	Data bit
54	D7	–	I/O	Data bit
57	D8	–	I/O	Data bit
63	D9	–	I/O	Data bit

Table 2-1 Pin Definition and Functions (cont'd)

Pin No.	Pin Name	Second Function	Dir.	Function
67	D10	–	I/O	Data bit
72	D11	–	I/O	Data bit
66	D12	–	I/O	Data bit
62	D13	–	I/O	Data bit
56	D14	–	I/O	Data bit
51	D15	–	I/O	Data bit
47	$\overline{RD}$	–	O	External memory read strobe for ROM. $\overline{RD}$ is activated for every external instruction or data read access.
46	$\overline{CSROM}$	–	O	Chip select signal for ROM device
44	$\overline{CSSDRAM}$	–	O	Chip select signal for SDRAM device
43	MEMCLK	–	O	Clock for SDRAM
45	CLKEN	–	O	Enable for memory clock
50	$\overline{WR}$	–	O	Memory write strobe
22	P4.0	A16	O	General purpose output port/Address bit
19	P4.1	A17	O	General purpose output port/Address bit
18	P4.2	A18	O	General purpose output port/Address bit
17	P4.3	A19	O	General purpose output port/Address bit
16	P4.4	A20	O	General purpose output port/Address bit
15	P4.5	CS3	O	General purpose output port/Chip select signal for second external static memory
49	LDQM	–	O	Write disable for low byte
48	UDQM	–	O	Write disable for high byte
109	XTAL2	–	O	Output of the oscillator amplifier circuit
108	XTAL1	–	I	Input of the oscillator amplifier circuit
73	$\overline{RSTIN}$	–	I	Reset input pin
121	CVBS1A	–	I	CVBS signal inputs for full service data slicing
120	CVBS1B	–	I	Ground for CVBS1A (differential input)
117	CVBS2	–	I	CVBS signal inputs for WSS data slicing
112	R	–	O	Analog output for red channel

Table 2-1 Pin Definition and Functions (cont'd)

Pin No.	Pin Name	Second Function	Dir.	Function
113	G	–	O	Analog output for green channel
114	B	–	O	Analog output for blue channel
104	COR	RSTOUT	O	Output for contrast reduction/Reset output
105	BLANK	CORBLA	O	Fast blanking signal/Three-level signal for contrast reduction + fast blanking
103	HSYNC	–	I/O	Horizontal sync In/output
102	VSYNC	VCS	I/O	Vertical sync In/output/Composite sync output
5	P2.8	EX0IN	I/O	General purpose I/O port/External interrupt 0
6	P2.9	EX1IN	I/O	General purpose I/O port/External interrupt 1
7	P2.10	EX2IN	I/O	General purpose I/O port/External interrupt 2
8	P2.11	EX3IN	I/O	General purpose I/O port/External interrupt 3
9	P2.12	EX4IN	I/O	General purpose I/O port/External interrupt 4
10	P2.13	EX5IN	I/O	General purpose I/O port/External interrupt 5
11	P2.14	EX6IN	I/O	General purpose I/O port/External interrupt 6
12	P2.15	EX7IN	I/O	General purpose I/O port/External interrupt 7
74	P3.0	SCL0	I/O	General purpose I/O port/I^2C Bus clock line 0
75	P3.1	SDA0	I/O	General purpose I/O port/I^2C Bus data line 0
76	P3.2	CAPIN	I/O	General purpose I/O port/GPT2 register CAPREL
77	P3.3	T3OUT	I/O	General purpose I/O port/GPT1 timer T3 toggle
78	P3.4	T3EUD	I/O	General purpose I/O port/GPT1 timer T3 ext. up/down
79	P3.5	T4IN	I/O	General purpose I/O port/GPT1 timer T4 input for count/gate/reload/capture
80	P3.6	T3IN	I/O	General purpose I/O port/GPT1 timer T3 count/gate input
81	P3.7	T2IN	I/O	General purpose I/O port/GPT1 timer T2 input for count/gate/reload/capture
82	P3.8	MRST	I/O	General purpose I/O port/SSC master-receiver/slave-transmit I/O

Table 2-1 Pin Definition and Functions (cont'd)

Pin No.	Pin Name	Second Function	Dir.	Function
83	P3.9	MTSR	I/O	General purpose I/O port/SSC master-transmit/slave-receiver O/I
88	P3.10	TxD0	I/O	General purpose I/O port/ASC0 clock/data output
89	P3.11	RxD0	I/O	General purpose I/O port/ASC0 data input (asynchronous) or I/O (synchronous.)
90	P3.12	–	I/O	General purpose I/O port
91	P3.13	SCLK	I/O	General purpose I/O port/SSC master clock output/slave clock input
92	P3.15	–	I/O	General purpose I/O port
124	P5.0	AN.0	I	General purpose I/O port/Analog input for A/D-converter
125	P5.1	AN.1	I	General purpose I/O port/Analog input for A/D-converter
126	P5.2	AN.2	I	General purpose I/O port/Analog input for A/D-converter
127	P5.3	AN.3	I	General purpose I/O port/Analog input for A/D-converter
93	P5.14	T4EUD	I/O	General purpose I/O port/GPT1 timer T4 ext.up/down ctrl. input
94	P5.15	T2EUD	I/O	General purpose I/O port/GPT1 timer T2 ext.up/down ctrl. input
95	P6.0	TRIG_IN	I/O	General purpose I/O port/Trigger input-signal for 'On Chip Debug System' (OCDS)
96	P6.1	TRIG_OUT	I/O	General purpose I/O port/Trigger output-signal for 'On Chip Debug System' (OCDS)
97	P6.2	–	I/O	General purpose I/O port
98	P6.3	SCL1	I/O	General purpose I/O port/I^2C bus clock line 1
99	P6.4	SDA1	I/O	General purpose I/O port/I^2C bus data line 1
100	P6.5	–	I/O	General purpose I/O port
101	P6.6	SDA2	I/O	General purpose I/O port/I^2C bus data line 2
1	TCK	–	I	Clock for JTAG interface
3	TDI	–	I	Data input for JTAG interface

Table 2-1 Pin Definition and Functions (cont'd)

Pin No.	Pin Name	Second Function	Dir.	Function
4	TDO	–	O	Data output for JTAG interface
2	TMS	–	I	Control signal for JTAG interface
128	TMODE	–	I	Testmode pin[1)]
110	V_{SSA-1}	–	S	Analog ground
111	V_{DDA-1}	–	S	Analog power (for PLL and DAC) (2.5 V)
115, 118, 122	V_{SSA2-4}	–	S	Analog ground
116, 119, 123	V_{DDA2-4}	–	S	Analog power (for ADCs) (2.5 V)
20, 86	$V_{SS25\ 1-2}$	–	S	Digital ground (for digital core)
21, 87	$V_{DD25\ 1-2}$	–	S	Digital power (for digital core) (2.5 V)
13, 31, 41, 52, 60, 68, 84, 107	$V_{SS33\ 1-8}$	–	S	Digital ground for pads
14, 32, 42, 53, 61, 69, 85, 106	$V_{DD33\ 1-8}$	–	S	Digital power (for pads) (3.3 V)

1) (Must be kept to "0" in application.)

Architectural Overview

3 Architectural Overview

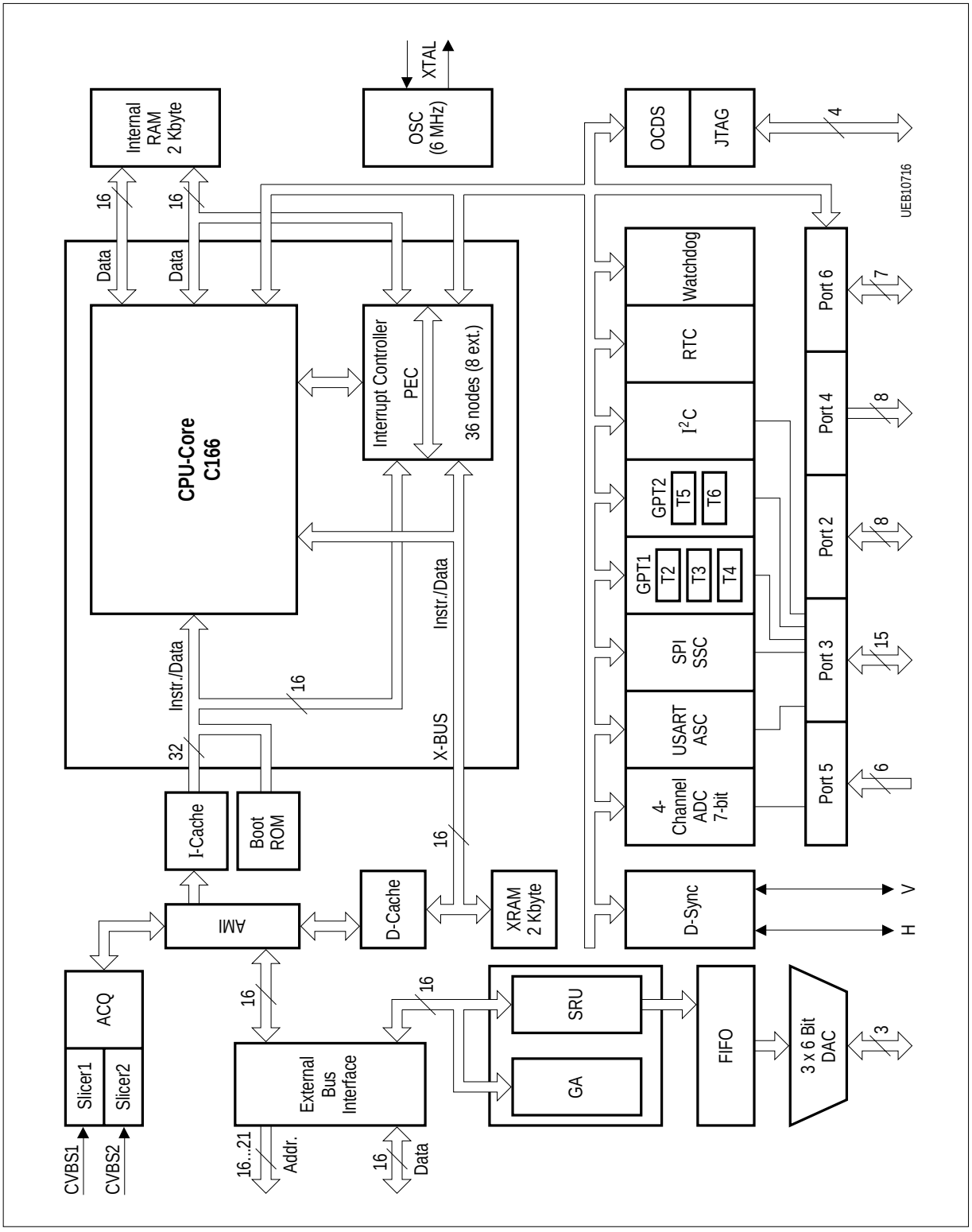


Figure 3-1 M2 Top Level Block Diagram

The architecture of M2 comprises of a 16-bit microcontroller which is derived from the well known Infineon Technologies C16x controller family. Due to the core philosophy of M2, the architecture of the CPU core is the same as described in other Infineon Technologies C16x derivatives.

The **CPU,** with its peripherals, can be used on one hand to perform all TV controlling tasks, and on the other hand to process the data, sliced by the slicer, and the acquisition unit according to the TTX standard. Furthermore it is used to generate an “instruction list” for the graphic accelerator which supports the CPU by generating the display.

M2 has integrated two digital **slicers** for two independent CVBS signals. One slicer is used to capture the data (e.g. Teletext or EPG) from the main channel, the other slicer can be used to slice the WSS information from a different channel, which is helpful e.g. to support PIP applications in 16:9 TVs. Both slicers separate the data from the analog signal and perform the bit synchronization and framing code selection before the data is stored in a programmable VBI buffer in the external RAM. Capturing and storing the raw data in the RAM does not need any CPU power.

M2’s display concept has improved in comparison to the common known state of the art Teletext-ICs. The display concept is based on a pixel orientated attribute definition instead of the former character orientated attribute definition.

For the processing of this new pixel based attribute definition the display generator architecture is divided in two subblocks: the graphic accelerator (GA) and the screen refresh unit (SRU).

The **graphic accelerator** is used to modify the frame buffer. From an abstract point of view, the graphic accelerator is a DMA which is optimized for OSD functionality, so e.g. bitmaps can be copied to the frame buffer. The graphic accelerator is used to draw rectangles, parallelograms, horizontal, vertical and diagonal lines. The user does not need to access the graphic accelerator directly, thanks to an easy to handle SW-GDI function which is available with the M2 hardware.

The DMA functionality of the display generator (DG) supports the pixel transfer between any address of entire external memory. The teletext and graphic capabilities can be used simultaneously, so that M2 can combine teletext information with e.g. background images and advanced high resolution OSD graphics.

M2 uses the frame buffer located in external memory so every bitmap can be placed at any location on the screen. The contents of the frame buffer does not have to be set up in real time. The duration of the set up of the screen depends on the contents of the displayed information.

M2 supports two hardware display layers. To refresh the screen the M2 reads and mixes two independent pixel sources simultaneously.

Different formats of the pixels which are part of different applications (e.g. Teletext formats, 12-bit RGB or 16-bit RGB values) can be stored in the same frame buffer at the same time.

The screen refresh unit is used to read the frame buffer pixel by pixel in real time and to process the transparency and RGB data. A color look up table (CLUT) can be used to get the RGB data of the current pixel. Afterwards the RGB data is transferred to the D/A converter. The blank signal and contrast reduction signal (COR) is also processed for each pixel by the SRU and transferred to the corresponding output pins.

The pixel, line and field frequencies are widely programmable so that the **sync system** can be used from low end 50 Hz to high end 100 HZ TV applications as well as for any other standard.

The on chip **clock system** provides the M2 with its basic clock signals. Independent clock domains are provided for the embedded controller, the bus interface and the display system. The pixel clock can vary between 10 MHz and 50 MHz.

Due to the unified memory architecture of M2, a new **bus** concept is implemented. An arbiter handles the bus requests from the different request sources. These are:

- Slicer 1 requests (normally used as a TTX slicer)
- Slicer 2 requests (used as a WSS slicer)
- Graphic accelerator requests
- Screen refresh unit requests
- Data requests from the CPU via XBUS
- Instruction requests via the CPU program bus

For exploiting the full computational power of the controller core the code of time critical routines can be stored in one bank of the external SDRAM separated from all display information (frame buffer, character set etc.). An **instruction cache** (I-CACHE) is used for buffering instruction words in order to minimize the probability of wait states to occur when the microcontroller is interfering with the display generator (DG) for access rights to the external memory devices. The **data cache** (D-CACHE) serves for operand reads and writes via the XBUS from/to external memory devices.

The **external bus interface** (EBI) features interleaved access cycles to one or two static external memory devices (ROM, Flash-ROM or SRAM) with a total maximum size of 4 MByte and one PC100 compliant (Intel standard) SDRAM device (16 MBit organized as 2 memory banks or 64 MBit organized as 4 memory banks).

For TV controlling tasks M2 provides three serial interfaces (I^2C, ASC, SSC), two general purpose timers, (GPT1, GPT2), a real time clock (RTC), a watch dog timer (WDT), an A/D converter and eight external interrupts.

4 C16X Microcontroller

4.1 Overview

M2's microcontroller and its peripherals are based on a Cell-Based Core (**CBC**) which is compatible to the well known C166 architecture.

In M2, the CPU and its peripherals are generally clocked with 33.33 MHz which results in an instruction cycle time of 60 ns. The implementation of the microcontroller within M2 deviates from other known C16x derivates since the controller's XBUS is not used as the external bus. All external access cycles of the microcontroller, the display generator and the acquisition unit are performed via a high performance time interlocking SDRAM bus. The external bus interface (EBI) manages the arbitration procedure for access cycles to the external synchronous DRAM in parallel to an external static memory (ROM or FLASH; for more details refer to **Chapter 4.4**).

Due to the realtime critical bus bandwidth requirements of the display generator, unpredictable wait-states for the controller may occur. These wait-states do not destroy the overall average system performance, because they are mostly buffered by the CPU related instruction and data buffers. Nevertheless they can influence, for example, the worst disconnection response time.

Emulation is now performed by an on-chip debug module which can be accessed by a JTAG interface.

The following microcontroller peripherals are implemented:

- 2 KByte IRAM (System RAM)
- 2 KByte XRAM (XBUS located)
- 32 Interrupt Nodes
- General Purpose Timer Units (GPT1 and GPT2)
- Real Time Clock (RTC)
- Asynchronous/Synchronous Serial Interface (ASC0)
- High-Speed Synchronous Serial Interface (SSC)
- I^2C Bus Interface (I^2C)
- 4-Channel 8-bit A/D Converter (ADC)
- Watchdog Timer (WDT)
- On-Chip Debug Support Module (OCDS)
- 42 Multiple Purpose Ports

Central Processing Unit

The CPU executes the C166 instruction set (with the extensions of the C167 products). Its main features are the following:

- 4-stage pipeline (Fetch, Decode, Execute and Write-Back).
- 16 $\times$ 16-bit General Purpose Registers
- 16-bit Arithmetic and Logic Unit

- Barrel shifter
- Bit processing capability
- Hardware support for multiply and divide instructions

Internal RAM (IRAM)

The internal dual-port RAM is the physical support for the General Purpose Registers, the system stack and the PEC pointers. Due to its close connections with the CPU, the internal RAM provides fast access to these resources. As the GPR bank can be mapped anywhere in the internal RAM through a base pointer (Context Pointer CP), fast context switching is allowed. The internal RAM is mapped in the memory space of the CPU and can be used also to store user variables or code.

Interrupt Controller

Up to 32 interrupt sources can be managed by the Interrupt Controller through a multiple priority system which provides the user with the ability to customize the interrupt handling.

The interrupt system of M2 includes a Peripheral Event Controller (PEC). This processor performs single-cycle interrupt-driven byte or word transfers between any two locations in the entire memory space of M2.

In M2, the PEC functionalities are extended by the External PEC which allows an external device to trigger a PEC transfer while providing the source and destination pointers. New features also include the packet transfer mode and the channel link mode.

Besides user interrupts, the Interrupt Controller provides mechanisms to process exceptions or error conditions, so-called “hardware traps”, that arise during program execution.

System Control Unit

M2’s System Control Unit (CSCU) is used to control system specific tasks such as reset control or power management within an on-chip system built around the core. The power management features of the CSCU provide effective means to realize standby conditions for the system with an optimum balance between power reduction, peripheral operation and system functionality. The CSCU also provides an interface to the Clock Generation Unit (CGU) and is able to control the operation of the Real Time Clock (RTC).

The CSCU includes the following functions:

- System configuration control
- Reset sequence control
- External interrupt and frequency output control
- Watchdog timer module
- General XBUS peripherals control
- Power management additional to the standard Idle and Power Down modes

- Control interface for Clock Generation Unit
- Identification register block for chip and CSCU identification

OCDS

The On-Chip Debug System allows the detection of specific events during user program execution through software and hardware breakpoints. An additional communication module allows communication between the OCDS and an external debugger, through a standard JTAG port. This communication is performed in parallel to program execution.

4.2 Memory Organization

In normal operation mode the memory space of the CPU is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same memory areas, i. e. external memory, internal controller memory (IRAM), the address areas for integrated XBUS peripherals (I^2C, internal XBUS memory (XRAM)) and the special function register areas (SFR, ESFR) are mapped into one common address space of 16 MBytes. This address space is arranged as 256 segments of 64 KBytes each and each segment is again subdivided into four data pages of 16 KBytes each.

All internal memory areas and the address space of the integrated XBUS peripherals are mapped to segment 0. Code and data may be stored in any part of the memory, except for the SFR blocks, which can not be used for instructions. Despite this equivalence of code and data, proper partitioning is necessary to make use of the full bandwidth of the memory system.

The integrated C16x controller communicates via 2 busses with the memory interface. In normal operation mode access to segments 00_H to 41_H (excluding internal memory areas) is mapped to the read only program memory bus (PMBUS), whereas access to segments 42_H to FF_H is mapped to the XBUS. In bootstrap loader mode (BSLMode) instruction fetches to external memory areas via PMBUS are redirected to the internal bootstrap loader ROM (BSLROM). Operand (data) accesses remain unchanged.

The PMBUS is connected to the instruction cache (ICACHE) which operates as read-ahead FIFO (see **Figure 4-1**). The data cache (DCACHE) which is connected to the XBUS holds a maximum of 4 words corresponding to one SDRAM burst. Accesses of DCACHE, ICACHE and the acquisition unit (ACQ) are joined within the acquisition memory interface (AMI) and directed to the external bus interface (EBI). Redirections via ESFR REDIR (instruction fetches only) and ESFR REDIR1 are done in the AMI (see **Chapter 4.5.1**). The EBI joins AMI and display generator (DG) accesses and reads data from or writes data to the external static and dynamic memory devices (see **Chapter 4.5** for further information). In case of cache miss wait states are inserted until the data is ready. IRAM, XRAM and the special function register areas can be accessed without wait states.

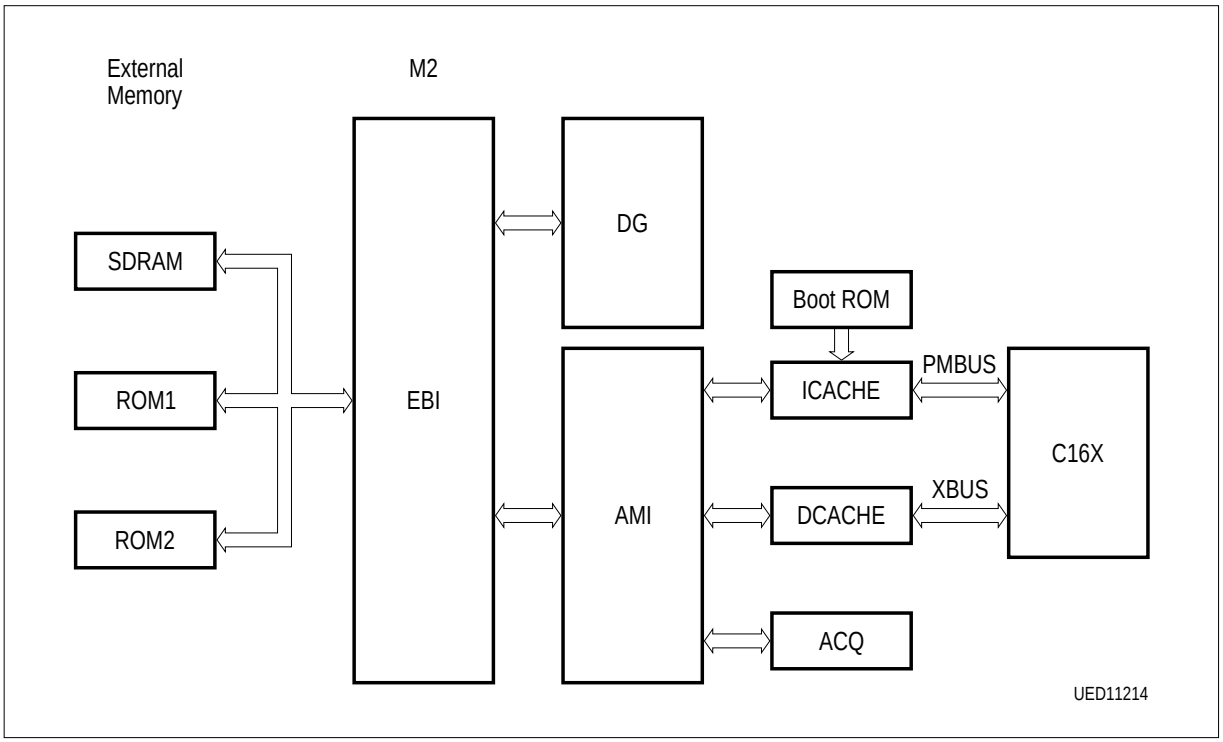


Figure 4-1 M2 Memory Path Block Diagram

All memory locations are byte and word readable. The internal memories (IRAM, XRAM) and the external dynamic memory (SDRAM) are byte and word writable, but external static memory is only word writable. Bytes are stored at even or odd byte addresses. Words are stored in ascending memory locations, with the low byte at an even byte address being followed by the high byte at the next odd byte address. Double words (instructions only) are stored in ascending memory locations as two subsequent words. Single bits are always stored in the specific bit position at a word address. Bit position 0 is the least significant bit of the byte at an even byte address and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported by a part of the special function registers, a part of the IRAM and the general purpose registers (GPRs).

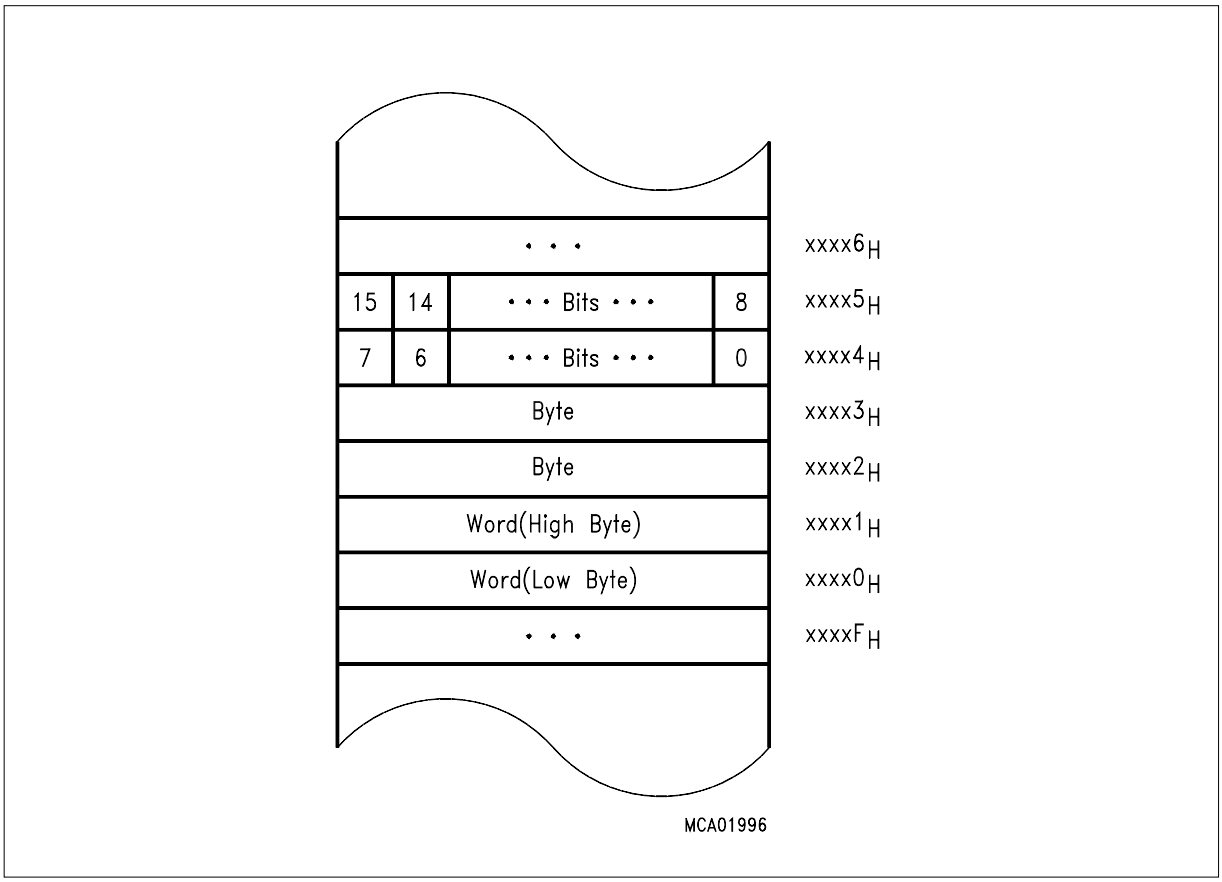


Figure 4-2 Storage of Words, Byte and Bits in a Byte Organized Memory

Note: Byte units forming a single word or a double word must always be stored within the same physical (internal, external, ROM, RAM) and organizational (page, segment) memory area.

4.3 On-Chip Microcontroller RAM and SFR Area

The IRAM/SFR area is located within data page 3, and provides access to 2 KByte of dual ported IRAM and two 512 Byte blocks of Special Function Registers (SFRs).
The internal RAM (IRAM) serves several purposes:

- System Stack (Programmable Size)
- General Purpose Register Banks (GPRs)
- Source and Destination Pointers for the Peripheral Event Controller (PEC)
- Variable and other data storage, or
- Code storage

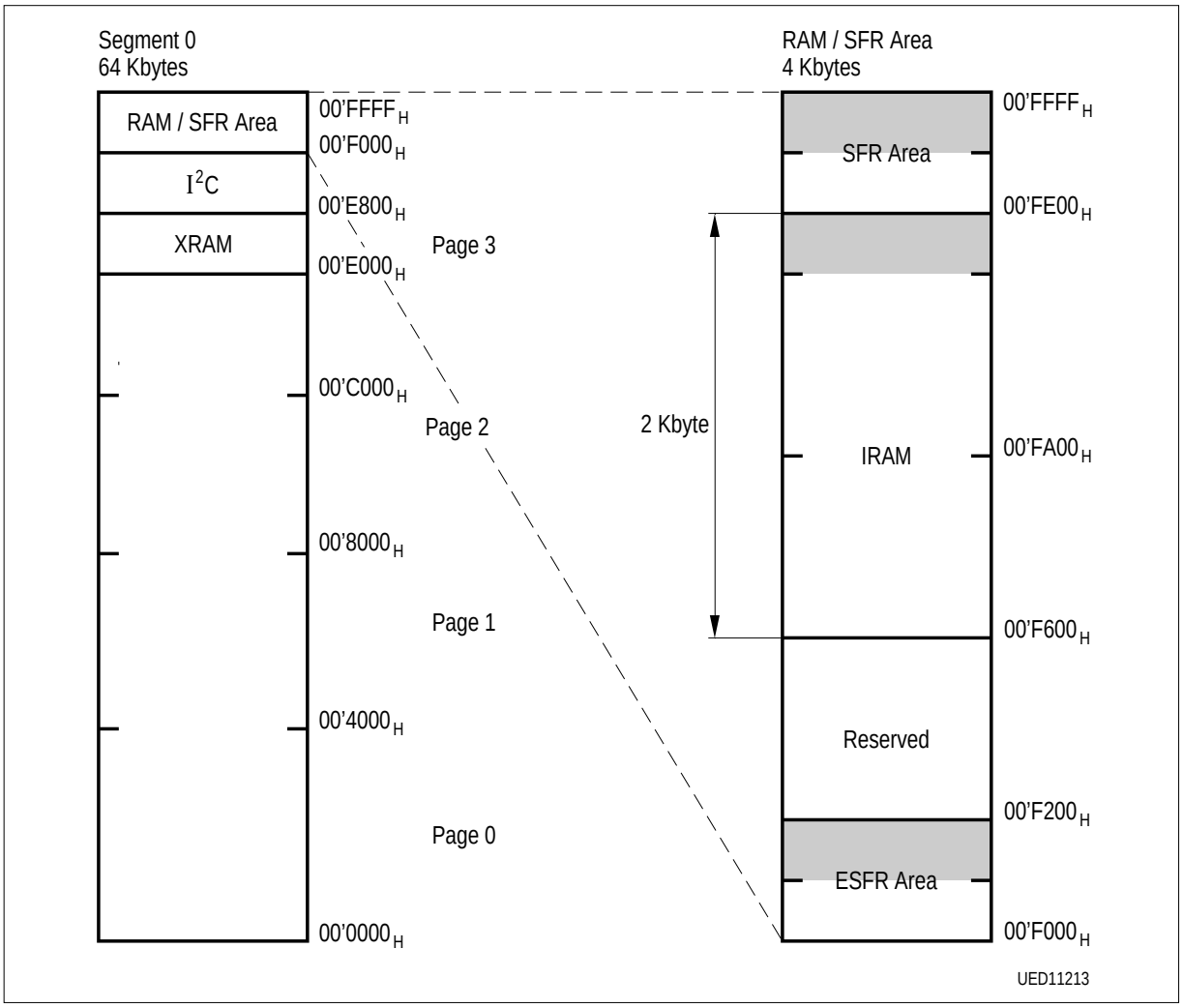


Figure 4-3 Internal RAM Areas and SFR Areas

*Note: The upper 256 bytes of SFR area, ESFR area and IRAM are bit-addressable (see shaded blocks in **Figure 4-3**).*

Code accesses are always made through even byte addresses. The highest possible code storage location in the IRAM is either 00'FDFE$_H$ for single word instructions, or 00'FDFC$_H$ for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossings from IRAM to the SFR area are not supported and cause erroneous results.

Any word and byte data in the IRAM can be accessed via indirect or long 16-bit addressing modes if the selected DPP register points to page 3. Any word data access is made through an even byte address. The highest possible word data storage location in the IRAM is 00'FDFE$_H$. For PEC data transfers, the IRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The upper 256 Byte of the IRAM (00'FD00$_H$ through 00'FDFF$_H$) and the GPRs of the current bank are provided for single bit storage, and thus they are bit addressable.

4.3.1 System Stack

The system stack may be defined within the IRAM. The size of the system stack is controlled by bit field STKSZ in the SYSCON register (see table below).

<STKSZ>	Stack Size (Words)	Internal RAM Addresses (Words)
0 0 0 $_B$	256	00'FBFE$_H$ … 00'FA00$_H$ (Default after Reset)
0 0 1 $_B$	128	00'FBFE$_H$ … 00'FB00$_H$
0 1 0 $_B$	64	00'FBFE$_H$ … 00'FB80$_H$
0 1 1 $_B$	32	00'FBFE$_H$ … 00'FBC0$_H$
1 0 0 $_B$	512	00'FBFE$_H$ … 00'F800$_H$
1 0 1 $_B$	–	Reserved. Do not use this combination.
1 1 0 $_B$	–	Reserved. Do not use this combination.
1 1 1 $_B$	1024	00'FDFE$_H$ … 00'F600$_H$ (Note: No circular stack)

For all system stack operations, the IRAM is accessed via the Stack Pointer (SP) register. The stack grows downward from higher to lower RAM address locations. Only word accesses are supported by the system stack. A stack overflow (STKOV) and a stack underflow (STKUN) register are provided to control the lower and upper limits of the selected stack area. These two stack boundary registers can be used not only for protection against data destruction, but also to implement flushing and filling a circular stack with a hardware supported system stack (except for option '111').

4.3.2 General Purpose Registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words within the IRAM. The Context Pointer (CP) register determines the base address of the currently active register bank. This register bank may consist of up to 16 word GPRs (R0, R1, …, R15) and/or of up to 16 byte GPRs (RL0, RH0, …, RL7, RH7). The sixteen byte GPRs are mapped onto the first eight word GPRs (see **Table 4-1**).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 Byte. The GPRs are accessed via short 2-, 4- or 8-bit addressing modes using the Context Pointer (CP) register as a base address (independent of the current DPP register contents). In addition, each bit in the currently active register bank can be accessed individually.

Table 4-1 Mapping of General Purpose Registers to RAM Addresses

Internal RAM Address	Byte Registers		Word Register
<CP> + $1E_H$	–		R15
<CP> + $1C_H$	–		R14
<CP> + $1A_H$	–		R13
<CP> + 18_H	–		R12
<CP> + 16_H	–		R11
<CP> + 14_H	–		R10
<CP> + 12_H	–		R9
<CP> + 10_H	–		R8
<CP> + $0E_H$	RH7	RL7	R7
<CP> + $0C_H$	RH6	RL6	R6
<CP> + $0A_H$	RH5	RL5	R5
<CP> + 08_H	RH4	RL4	R4
<CP> + 06_H	RH3	RL3	R3
<CP> + 04_H	RH2	RL2	R2
<CP> + 02_H	RH1	RL1	R1
<CP> + 00_H	RH0	RL0	R0

M2 supports fast register bank (context) switching. Multiple register banks can physically exist within the IRAM at the same time. However, only the register bank selected by the Context Pointer register (CP) is active at a given time. Selecting a new active register bank is simply done by updating the CP register. A particular Switch Context (SCXT) instruction performs register bank switching and automatically saves the previous context. The number of implemented register banks (arbitrary sizes) is only limited by the size of the available internal RAM.

4.3.3 PEC Source and Destination Pointers

The 16 word locations in the IRAM from 00'$FCE0_H$ to 00'$FCFE_H$ (just below the bit-addressable section) are provided as source and destination offset address pointers for data transfers on the eight PEC channels. Each channel uses a pair of pointers stored in two subsequent word locations, with the source pointer (SRCPx) on the lower and the destination pointer (DSTPx) on the higher word address (x = 7 … 0). In M2, these pointers are used to specify the address offset within the segment, and the destination / source segment numbers are specified in designated SFRs (see **Chapter 5.2**).

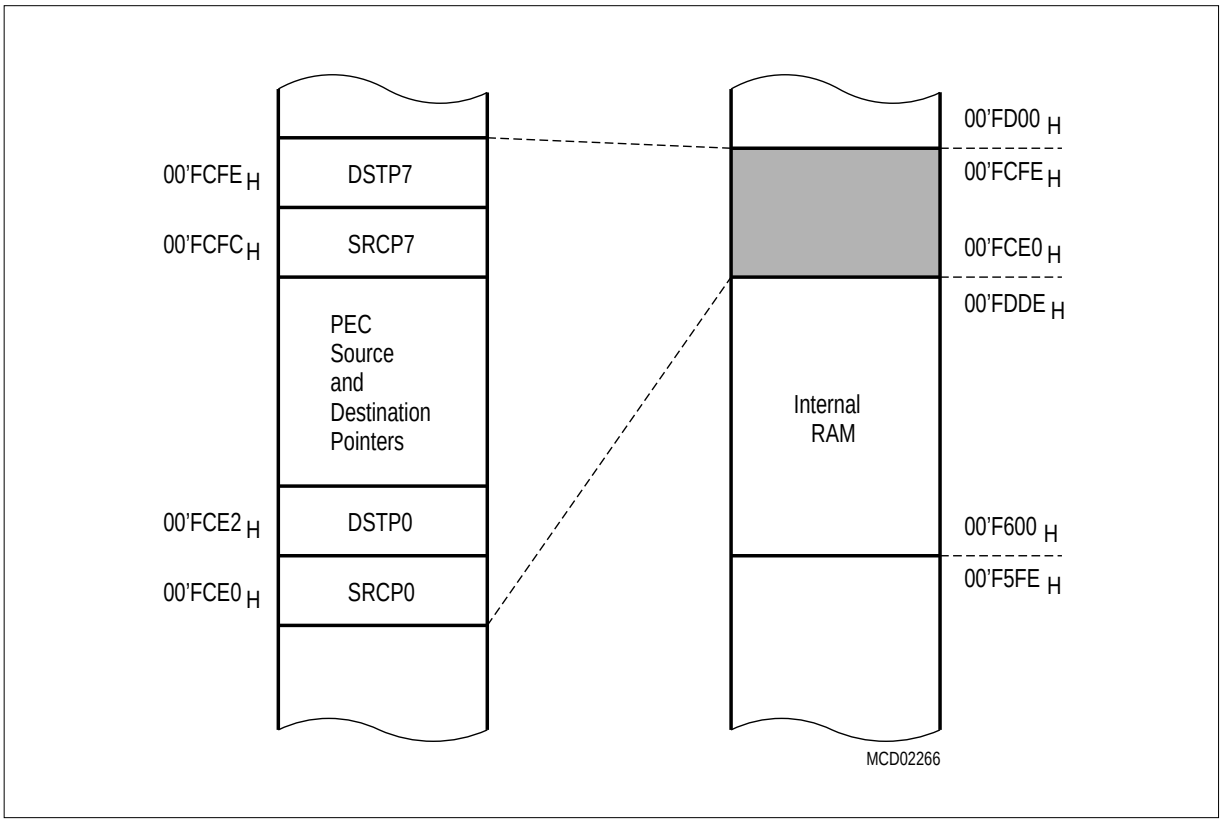


Figure 4-4 Location of the PEC Pointers

Whenever a PEC data transfer is performed, the pair of source and destination pointers, which is selected by the specified PEC channel number, is accessed independent of the current DPP register contents; the locations referred to by these pointers are also accessed independent of the current DPP register contents. If a PEC channel is not used, the corresponding pointer locations are available and can be used for word or byte data storage.

4.3.4 Special Function Registers

The so-called Special Function Registers (SFRs) are provided to control internal functions of M2 (CPU, bus interface, Interrupt Controller, OCDS) or peripherals connected to the Peripheral Bus. These SFRs are arranged within two areas of 512 Bytes each. The first register block, the SFR area, is located in the 512 Bytes above the internal RAM ($00'FFFF_H$ … $00'FE00_H$); the second register block, the Extended SFR (ESFR) area, is located in the 512 Bytes below the IRAM ($00'F1FF_H$ … $00'F000_H$).

Special function registers can be addressed via indirect and long 16-bit addressing modes. Using an 8-bit offset together with an implicit base address allows word SFRs and their respective low bytes to be addressed. However, this **does not work** for the respective high bytes!

Note: Writing to any byte of an SFR causes the non-addressed complementary byte to be cleared!

The upper half of each register block is bit-addressable, so the respective control/status bits can directly be modified or checked using bit addressing.

When accessing registers in the ESFR area using 8-bit addresses or direct bit addressing, an Extend Register (EXTR) instruction is first required to switch the short addressing mechanism from the standard SFR area to the Extended SFR area. This is not required for 16-bit and indirect addresses. The GPRs R15 … R0 are duplicated, i.e. they are accessible within both register blocks via short 2-, 4- or 8-bit addresses without switching.

```
ESFR_SWITCH_EXAMPLE:
EXTR   #4                      ;Switch to ESFR area for next 4 instr.
MOV    ODP2, #data16           ;ODP2 uses 8-bit reg addressing
BFLDL  DP6, #mask, #data8      ;Bit addressing for bit fields
BSET   DP1H.7                  ;Bit addressing for single bits
MOV    T8REL, R1               ;T8REL uses 16-bit mem address,
                               ;R1 is duplicated into the ESFR space
                               ;(EXTR is not required for this access)

;----  ;-------------------    ;The scope of the EXTR #4 instruction…
                               ; … ends here!

MOV    T8REL, R1               ;T8REL uses 16-bit mem address,
                               ;R1 is accessed via the SFR space
```

In order to minimize the use of the EXTR instructions the ESFR area primarily holds registers which are required mainly for initialization and mode selection. Registers that need to be accessed frequently are allocated, wherever possible, to the standard SFR area.

Note: The tools are equipped to monitor accesses to the ESFR area and will automatically insert EXTR instructions or issue a warning in case of missing or excessive EXTR instructions.

4.4 External Memory

M2 provides an external bus interface (EBI) to access an external SDRAM, together with an external static memory device (ROM or SRAM). To optimize the overall system performance, access to both memory types is interlocked. Because of high performance requirements M2 provides only one bus type (Demultiplexed 16-bit Bus). Depending on the reset configuration (refer to **Chapter 6.1**) an external ROM/SRAM size from 128 KByte up to 4 MByte can be chosen. Although external addresses (represented by pins A0 … A20) are always word addresses, byte accesses to the SDRAM are possible by using mask signals LDQM and UDQM.

4.4.1 SDRAM

PC SDRAM compliant (Intel standard) memory devices with 2 or 8 MByte and a minimum clock period of 10 ns (latency 3) may be connected to M2's external memory bus.

Supported data organizations are given below:

Memory Size	# SDRAM Banks	# Bank Addresses	# Row Addresses	# Column Addresses
2 MByte	2	1	11	8
8 MByte	4	2	12	8

The external SDRAM connected to M2 is a multifunctional, byte or word addressable device which can be used for frame buffers, character sets, pixel graphics, acquisitions, microcontroller workspace and any other data storage purposes.

Using a 100 MHz external memory bus the theoretical optimum memory bandwidth is limited to 200 MByte/s. In order to keep the sustainable memory bandwidth as close to the optimum as possible, the bank oriented architecture of SDRAM devices has to be exploited. Basically, display related information should be separated from controller related data items.

The following allocation is recommended for a 2 bank, 2 MByte device:

- **"Display Bank"**: Both Frame Buffers, Character Set, Pixel Graphic, Graphic Accelerator Instructions (GAI), Application Data (i.e. TTX, EPG, …)
- **"Controller Bank"**: Instruction Code, VBI-buffer, Application Data (i.e. TTX, EPG, …)

The suggested allocation leads to best performance results since it reduces the number of time consuming row commands on the SDRAM.

4.4.2 External Static Memory Devices

M2 supports access to external ROM, Flash ROM and SRAM devices which provide a read cycle time t_{RC} < 120 ns. Only 16-bit word access is supported. The maximum memory size is limited by the number of external address lines. Up to 21 external address lines are configurable, thus devices providing up to 4 MByte of static external memory can be connected to M2.

4.5 External Bus Interface (EBI).

The EBI handles access channels to four SDRAM banks within one SDRAM device and up to two static memory devices at 100 MHz. (For lower requirements the clock frequency can be reduced to 66 MHz, refer to **Chapter 8**). A maximum of three external memory devices is supported.

Figure 4-5 shows the possible configurations.

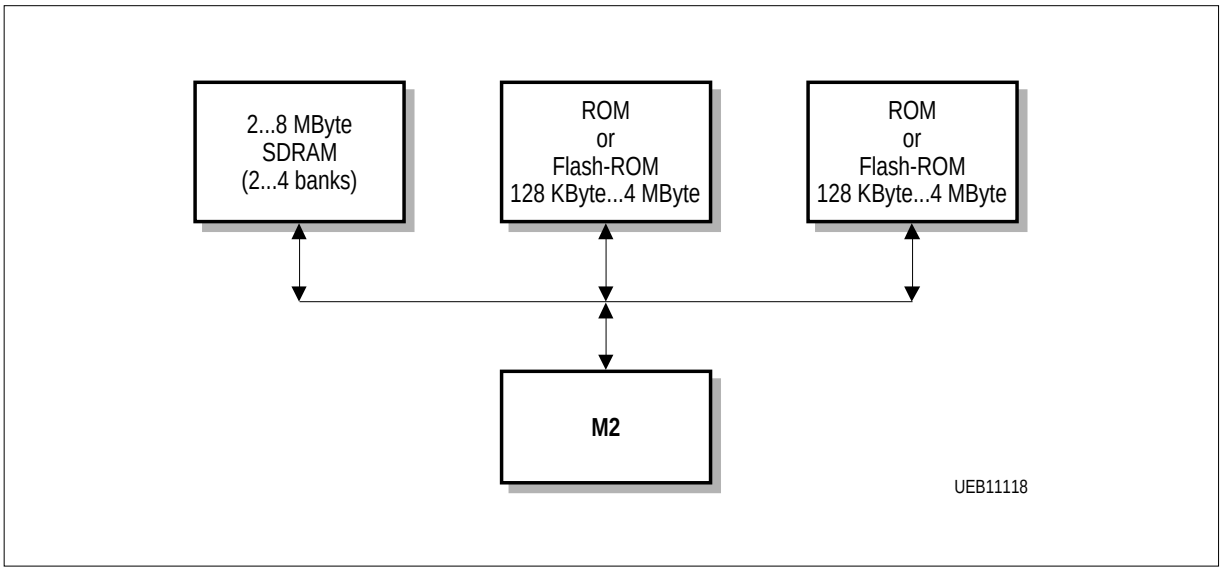


Figure 4-5 External Memory Configuration

The interlocking execution of access cycles to different memory modules is supported. All external SDRAM access cycles must be executed with a pre-defined burst length BL = 4 and latency 3. Write access cycles, which modify less than four SDRAM locations, are achieved by activating mask control signals L/UDQM.

The integrated refresh controller of the EBI checks for the compliance of refresh periods and executes refresh operations on the SDRAM devices. The configuration of different external SDRAM types can be controlled by a special SW driver as well as refresh modes and power down features. The microcontroller and the acquisition unit use a common interface to the EBI. A separate connection to the EBI is provided for the display generator. The EBI performs an arbitration procedure for granting right access to either of the request sources. But granting right access to one source does not exclude requests initiated by the other source from being served. A maximum of two access requests from a source may be served consecutively if the other source is addressing an SDRAM location. Up to four consecutive access cycles from the same source are served if the other source is addressing an external ROM device.

The following figures show typical timing diagrams that may be observed on the external bus. The first figure presents the interlocked execution of access cycles to the external ROM and a SDRAM device. The other figure resumes the situation when both sources address locations are in different SDRAM banks. Detailed timings and the specification of setup and hold conditions can be found in **Chapter 14**.

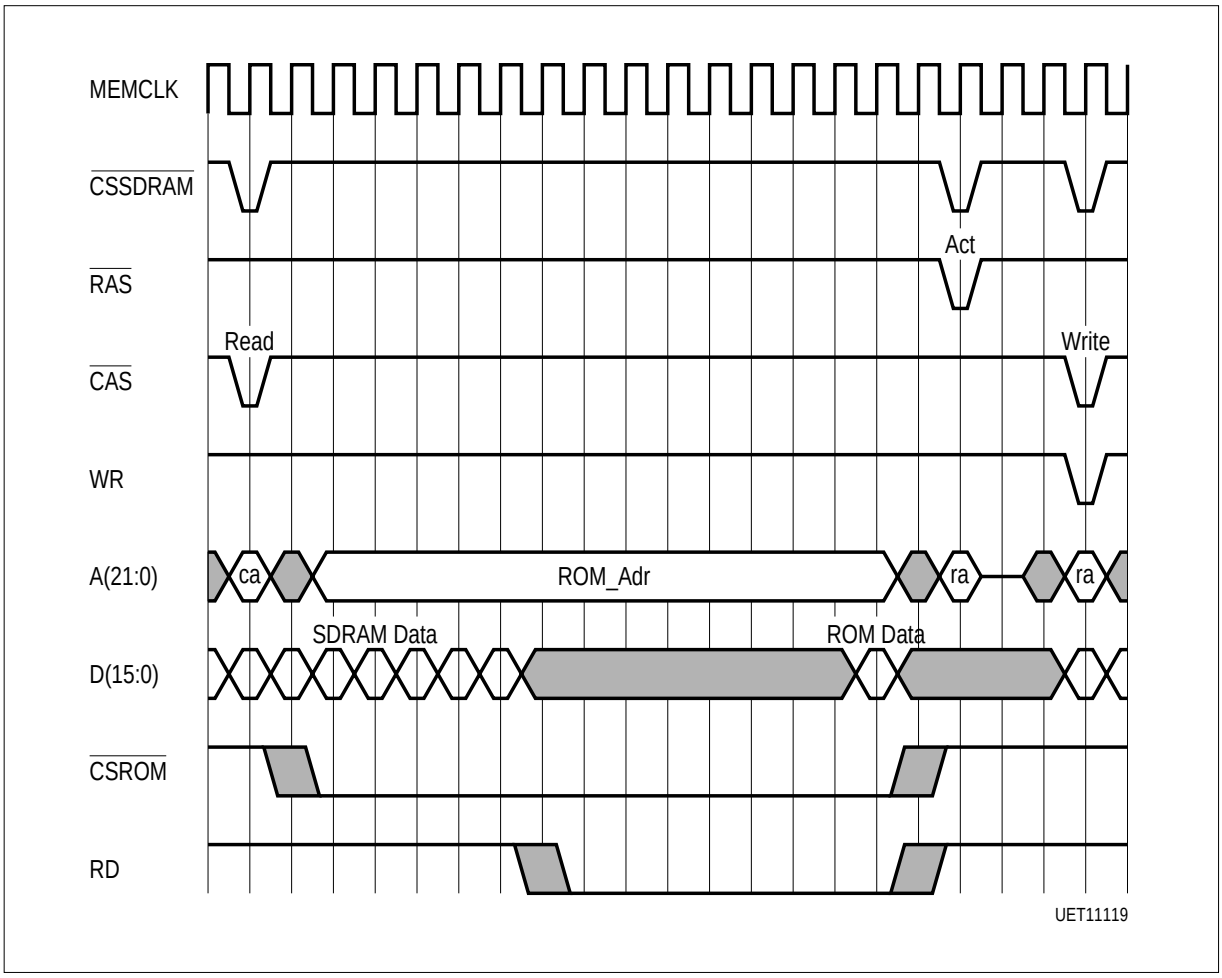


Figure 4-6 Interlocked Access Cycles to ROM and SDRAM

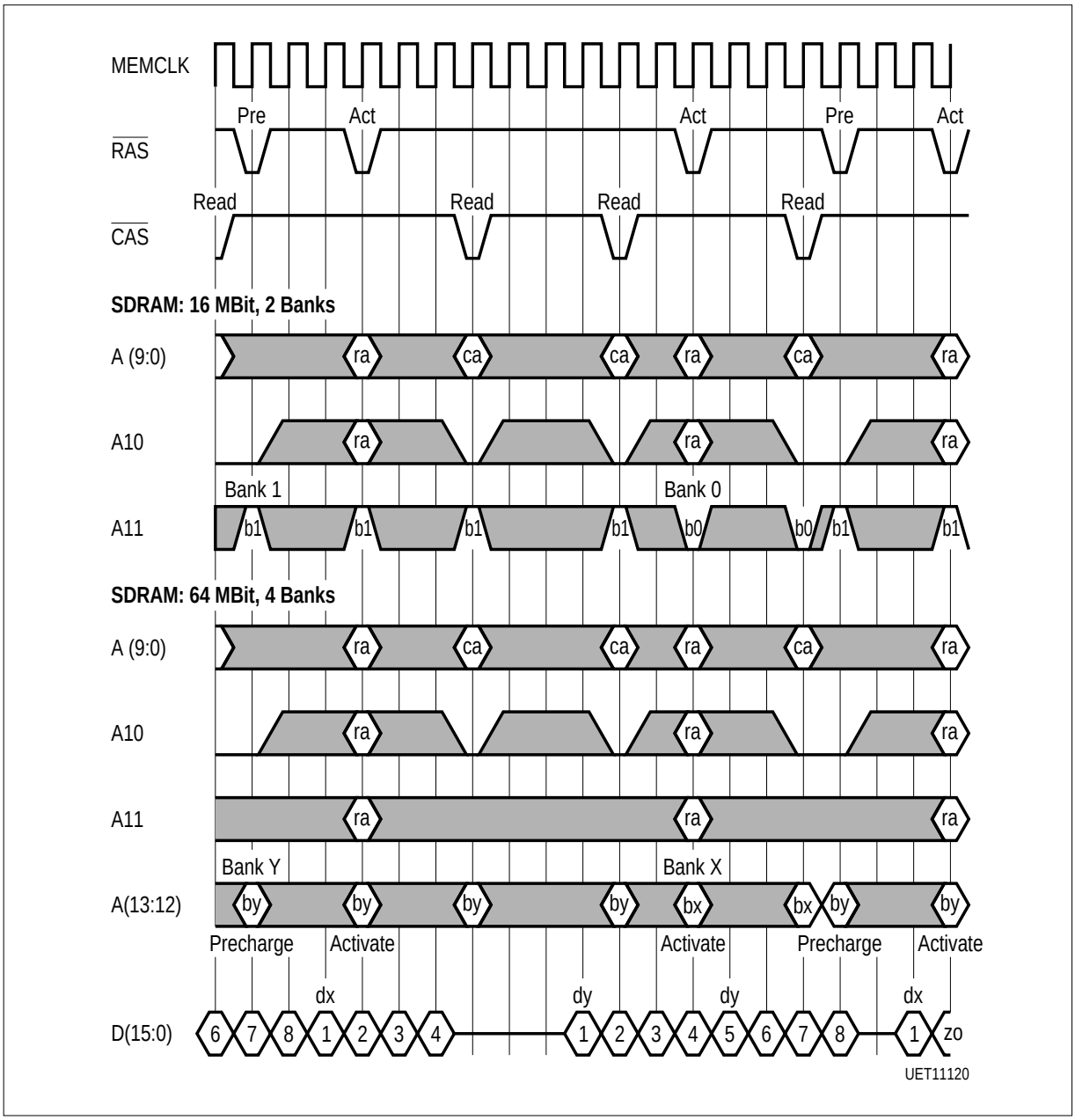


Figure 4-7 Interlocked Access Cycles to two SDRAM Banks

4.5.1 Memory Mapping

Mapping of memory locations from the address space of the C16x to physical addresses (and chip selects) occurs in 4 steps (all addresses shown below are byte addresses unless otherwise noted). **Figure 4-8** gives a coarse depiction (without redirection) of the mapping process in normal operation mode.

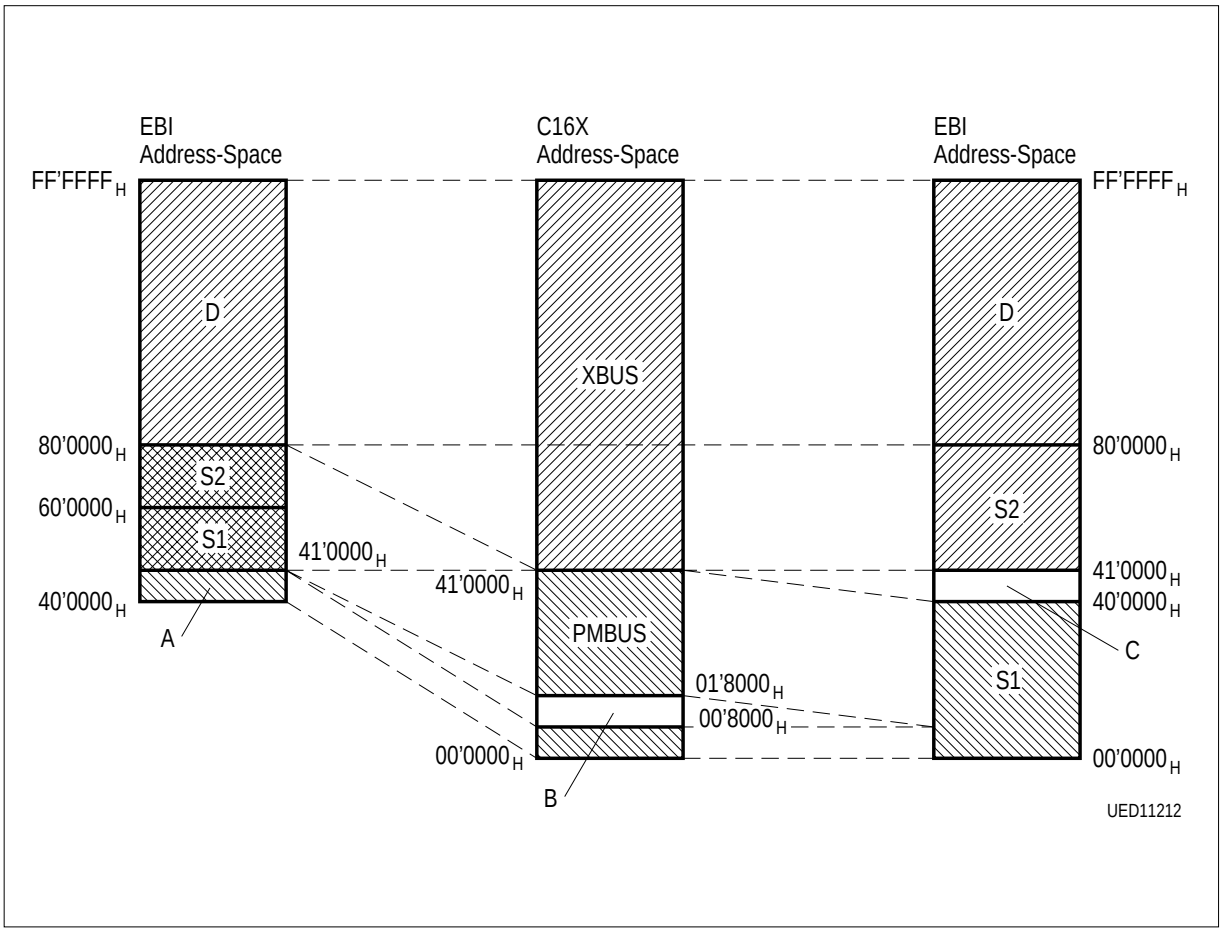


Figure 4-8 Memory Mapping
from C16x address-space to EBI address-space for 1 64MBit SDRAM (D) and 2 16MBit static memory devices (S1, S2) is shown on the left (XBUS/ PMBUS overlap). The right part shows the mapping for 1 64MBit SDRAM (D) and 2 32MBit static memory devices (S1, S2) (no XBUS/PMBUS overlap). Redirection via REDIR/REDIR1 is not shown. The exclusion of the address space for internal memories B leads to 1 segment in physical memory that is either addressable only via PMBUS A or not addressable at all C. To get access to these segments use REDIR1.

a) Internal mapping of the C16x:
Access to segments 0 to 64 selects the PMBUS. The address range ($00'0000_H$ ... $40'FFFF_H$) is mapped to the range $00'0000_H$ ... $3F'FFFF_H$ shown to the ICACHE, thus omitting the internal memories, the special function register areas and the XBUS peripheral address space contained the range $00'8000_H$ ... $01'7FFF_H$:

- *C16x addresses $00'0000_H$ ... $00'7FFF_H$ are mapped to $00'0000_H$ ... $00'7FFF_H$*
- *C16x addresses $01'8000_H$ ... $40'FFFF_H$ are mapped to $00'8000_H$ ... $3F'FFFF_H$*

Access to segments 65 to 255 selects the XBUS. This address range (41'0000$_H$ ... FF'FFFF$_H$) is not remapped by the C16x.

b) Mapping by caches:
In normal operation mode the address requested by the controller is not altered by ICACHE and DCACHE. In bootstrap loader mode, instruction fetches via the ICACHE are mapped to the boot ROM by the ICACHE:

- *Address is mapped to address mod 40$_H$*

Operand reads via ICACHE and all accesses via DCACHE are passed unaltered to the AMI.

c) Mapping and redirection in AMI:
Address mapping in AMIdepends on the settings of the ESFRs REDIR and REDIR1 as well as on the total amount of static memory. The mapping is done in the following order:

1) Check for redirection via REDIR1: If the requested address lies in segment 255 the segment address is replaced by the low byte of REDIR1.

2) Check for redirection via REDIR: If the address resulting from step1 lies in the address range specified by REDIR, the address is shifted to SDRAM area:

- *Address is mapped to 80'0000$_H$ + address – REDIR_LOWER $\times$ 16 kBytes*

3) If the total amount of static memeory in 4 MBytes or less (i.e. SALSEL $\neq$ "111" or SALSEL = "111" and no second device present) and the address resulting from step 2 is below 4 MBytes the address is shifted by 4 MBytes:

- *Address is mapped to address + 40'0000$_H$*

This means that the address ranges 02'0000$_H$ ... 40'FFFF$_H$ (PMBUS) and 41'0000$_H$ ... 7F'FFFF$_H$ (XBUS) in the address space of the C16x are mapped to the same physical memory. The overlap allows to make use of 2 independent busses for code (PMBUS) and data (XBUS) for fast parallel access.

If the total amount of static memory is 8 MBytes (i.e. SALSEL = "111" and a second device is present) no further mapping occurs.

d) Mapping to addresses of specific physical devices by EBI:

1) dynamic memory:

- *addresses 80'0000$_H$ ... FF'FFFF$_H$ (9F'FFFF$_H$) access the 64MBit (16 MBit) SDRAM ($\overline{CSSDRAM}$)*

2) static memory:
In the total amount of static memory is 4 MBytes or less (see above) the requests of the AMI or the DG in the address range 40'0000$_H$ ... 7F'FFFF$_H$ are passed to the external static memory devices according to the SALSEL and CSENA settings (see **Chapter 6.1**). The address range 00'0000$_H$ ... 3F'FFFF$_H$ must not be used. If the total amount of the static memory is 8 MBytes there is no reserved address range:

- *Addresses $00'0000_H$ ... $3F'FFFF_H$ access the first static memory device ($\overline{CSROM}$)*
- *Addresses $40'0000_H$ ... $7F'FFFF_H$ access the second static memory device ($\overline{CS3}$)*

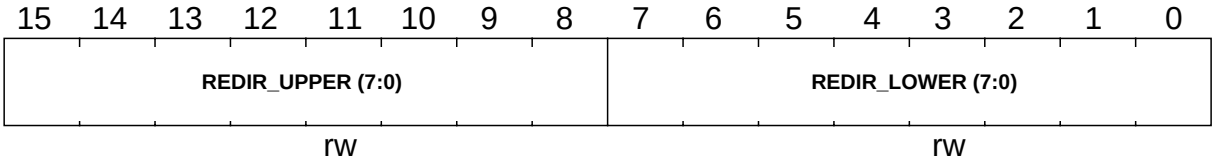
  The addresses shown on the external address lines of M2 are word oriented, starting at 0 for each device.

The External Bus Interface (EBI) provides a special mode, the “EBI direct mode”, where the control pins to the SDRAM device are directly controlled by the CPU while the HW finite state machines of the EBI are bypassed. “EBI direct mode” is used for accomplishing operations on the SDRAM such as the execution of the requisite initialization sequence, power down mode entry/exit etc. When executing a direct mode command the EBI shifts the contents of register EBIDIR into the SDRAM control lines. The INFINEON SDRAM driver (refer to document list) provides appropriate functions for executing operations in direct mode.

4.5.2 Register Description

Access cycles to addresses specified by bit fields REDIR_LOWER and REDIR_UPPER are redirected by hardware to the SDRAM area. The area to which the specified range is mapped, starts at the base address ($80'0000_H$) of the SDRAM. The range for redirection is selected in groups of 16 KByte. Using the REDIR register, access cycles to ROM located routines may be redirected to copies of these routines in the SDRAM.

REDIR **Reset Value: 0000_H**



Bit	Function
REDIR_LOWER (7:0)	Base address of selected range in the ROM Area Bitfield REDIR_LOWER specifies the MSBs of the base address of the selected range.
REDIR_UPPER (7:0)	Upper address of selected range in the ROM Area (exclusive) Bitfield REDIR_UPPER specifies the MSBs of the first no longer redirected 16 KBytes.

To gain access to memory areas covered by the read only PMBUS or to areas not accessible at all (see **Figure 4-8**) accesses to segment 255 can be redirected to any other segment in EBI address space.

REDIR1 **Reset Value: 00FF$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								REDIR1_SEG (7:0)							
												rw			

Bit	Function
REDIR1_SEG (7:0)	For access to segment 255, the segment part of the address is replaced by REDIR1_SEG.

The configuration of the “External Bus Interface” and its operation mode is defined with the EBICON register.

EBICON **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	REFEN	SDRSZE	EDMA	EDMR
												rw	r	r	rw

Bit	Function
EDMR	**EBI Direct Mode Request Flag** ‘0’: EBI direct mode is disabled ‘1’: EBI direct mode is enabled *Note: This bit is only used for EBI direct mode.*
EDMA	**EBI Direct Mode Acknowledge Flag** ‘0’: The EBI has not (yet) entered direct mode ‘1’: The EBI has entered direct mode *Note: This bit is only used for EBI direct mode.*
SDRSZE	**SDRAM Size** ‘0’: 16 MBit ($2 \times 2048 \times 256 \times 2B$), 2 banks (bank = adr_11) ‘1’: 64 MBit ($4 \times 4096 \times 256 \times 2B$), 4 banks (bank = adr_13:12)
REFEN	**Refresh Controller Enable Bit** ‘0’: Refresh controller for SDRAM is disabled ‘1’: Refresh controller for SDRAM is enabled

The EDMR request flag and the hardware controlled acknowledge flag EDMA are used during “EBI direct mode” for implementing a four-phase handshake which guarantees that each direct mode command is executed exactly once by the EBI.

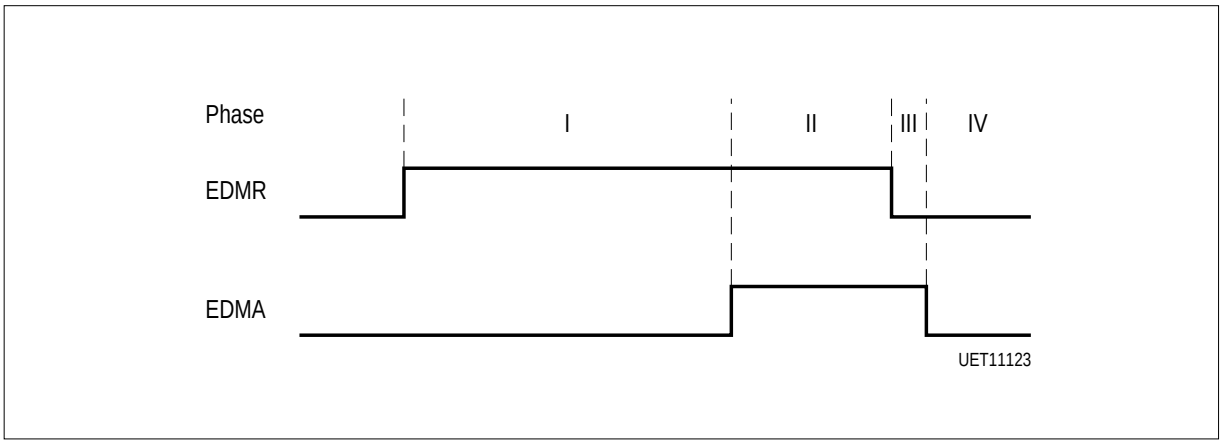


Figure 4-9 Four-Phase Handshake

- Phase I: The controller requests a direct mode command which has not yet been executed by the EBI. The controller must not reset the EDMR bit until the EBI acknowledges the EDMA bit (EDMA polling required).
- Phase II: The EBI acknowledges the EDMA bit after executing the requested direct mode command. The EBI will not reset the EDMA bit before the requested EDMR bit is reset.
- Phase III: The requested EDMR bit is reset while the acknowledged EDMA bit is still valid. This phase will only take one EBI clock period (no EDMA polling required).
- Phase IV: EBI is waiting for the next direct mode request.

When executing a direct mode command the EBI shifts the contents of register EBIDIR into the external control pins of the SDRAM.

EBIDIR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	CLK EN	CS _N	RAS _N	CAS _N	WR _N	ADR _10
										rw	rw	rw	rw	rw	rw

Bit	Function
ADR_10	Control Bit for Address Pin A10 in Direct Mode
WR_N	Control Bit for Pin WR in Direct Mode
CAS_N	Control Bit for Pin $\overline{\text{CAS}}$ (A15) in Direct Mode
RAS_N	Control Bit for Pin $\overline{\text{RAS}}$ (A14) in Direct Mode
CS_N	Control Bit for Pin $\overline{\text{CSSDRAM}}$ in Direct Mode
CLKEN	Control Bit for Pin CLKEN in Direct Mode

The setting required for initiating a certain command on the SDRAM has to be written to the EBIDIR register before the direct mode request, the EDMR bit in the EBICON register is asserted.

The following table shows the commands that may be executed in direct mode along with the associated settings of the EBIDIR register.

	CLKEN n-1	CLKEN n	CS_N	RAS_N	CAS_N	WE_N	ADR_10
Device deselect *DSEL*	1	dc[1)]	1	dc	dc	dc	dc
No Operation *NOP*	1	dc	0	1	1	1	dc
Precharge all banks *PALL*	1	dc	0	0	1	0	1
Auto refresh *CBR*	1	1	0	0	0	1	dc
Self refresh entry *SELFRSH*	1	0	0	0	0	1	dc
Self refresh exit *SELFRSHX*	0	1	1	dc	dc	dc	dc
Power down entry *PWRDN*	1	0	dc	dc	dc	dc	dc
Power down exit *PWRDNX*	0	1	1	dc	dc	dc	dc
Mode register set *MRS*	1	dc	0	0	0	0	0

1) dc = don't care

The MRS command (mode register set) is used to program the SDRAM for the desired operating mode. When executing the MRS command, address lines A11-A10 encode the operating mode. M2 then issues a hardwired pattern that sets the SDRAM to latency mode *3*, wrap type *linear* and burst length *4*.

For the correct handling of access cycles the user has to provide the EBI with information about the external memory configuration and memory sizes. The combination of reset configuration and the SDRSZE bit of the EBICON register includes all the information needed. Based on these inputs the EBI constructs its internal address map for allocating ROM devices and SDRAM banks.

The external memory configuration is defined with bit CSENA of the RP0H register (refer to **Chapter 6.1**). The memory configuration controls the correct behavior of pin $\overline{CS3}$.

Bit	Function
CSENA	**Chip Select Enable** '0': $\overline{CS3}$ is active for 2nd ROM device '1': $\overline{CS3}$ is inactive

The allocation of address ranges for the SDRAM banks is controlled through the SDRSZE bit.

SDRSZE = '0' (16 MBit): Address Ranges of Banks

Bank	Address Range
Bank1	$80'0000_H$ - $8F'FFFF_H$
Bank2	$90'0000_H$ - $9F'FFFF_H$

SDRSZE = '1' (64 MBit): Address Ranges of Banks

Bank	Address Range
Bank1	$80'0000_H$ - $9F'FFFF_H$
Bank2	$A0'0000_H$ - $BF'FFFF_H$
Bank3	$C0'0000_H$ - $DF'FFFF_H$
Bank4	$E0'0000_H$ - $FF'FFFF_H$

If a second ROM device is enabled, its base address depends on the **maximum** size of **both** ROM devices as defined within bit field SALSEL of register RP0H during reset.

Base Address of 2nd ROM Device

SALSEL	Physical Base Address
'111'	2nd ROM device not possible
'110'	$20'0000_H$
'101'	$10'0000_H$
'100'	$08'0000_H$
'011'	$04'0000_H$
others	$02'0000_H$

4.5.3 Crossing Memory Boundaries

The address space of M2 is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these

blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

Memory Areas are partitions of the address space that represent different kinds of memory (if provided at all). These memory areas are the internal RAM/SFR area, the program memory (if available), the on-chip X-Peripherals (if integrated) and the external memory.

Accessing subsequent data locations that belong to different memory areas is no problem. However, when executing code, the different memory areas must be switched explicitly via branch instructions. Sequential boundary crossing is not supported and leads to erroneous results.

Note: Changing from the external memory area to the internal RAM/SFR area takes place within segment 0.

Segments are contiguous blocks of 64 KByte each. They are referenced via the code segment pointer CSP for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.
During code fetching segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS and RETS will do this.
In larger sequential programs make sure that the highest used code location of a segment contains an unconditional branch instruction to the respective following segment, to prevent the prefetcher from trying to leave the current segment.

Data Pages are contiguous blocks of 16 KByte each. They are referenced via the data page pointers DPP3…0 and via an explicit data page number for data accesses overriding the standard DPP scheme. Each DPP register can select one of the 1024 possible data pages. The DPP register that is used for the current access is selected via the two upper bits of the 16-bit data address. Subsequent 16-bit data addresses that cross the 16 KByte data page boundaries will therefore use different data page pointers, while the physical locations need not be subsequent within memory.

4.6 Central Processing Unit

Basic tasks of the CPU are to fetch and decode instructions, to supply operands for the arithmetic and logic unit (ALU), to perform operations on these operands in the ALU, and to store the previously calculated results.

Since a four stage pipeline is implemented in M2, up to four instructions can be processed in parallel. Most instructions of M2 are executed in one machine cycles (2 CPU clock cycles) due to this parallelism. This chapter describes how the pipeline works for sequential and branch instructions in general, and which hardware provisions have been made, in particular, to speed up the execution of jump instructions. The description of the general instruction timing includes standard and exceptional timing.

For instruction and operand fetches, the CPU is connected to the different areas (external memory, program memory, internal dual-port RAM or (E)SFR area) either

internally or through the interfaces of the CPU (XBUS, program memory bus or peripheral bus). Where the program memory bus and the peripheral bus are tightly coupled to the CPU, XBUS accesses are performed, if possible, in parallel while the CPU continues operating. If data is required but not yet available or if a new XBUS access is requested by the CPU before a previous access has been completed, the CPU will be held until the request can be satisfied.

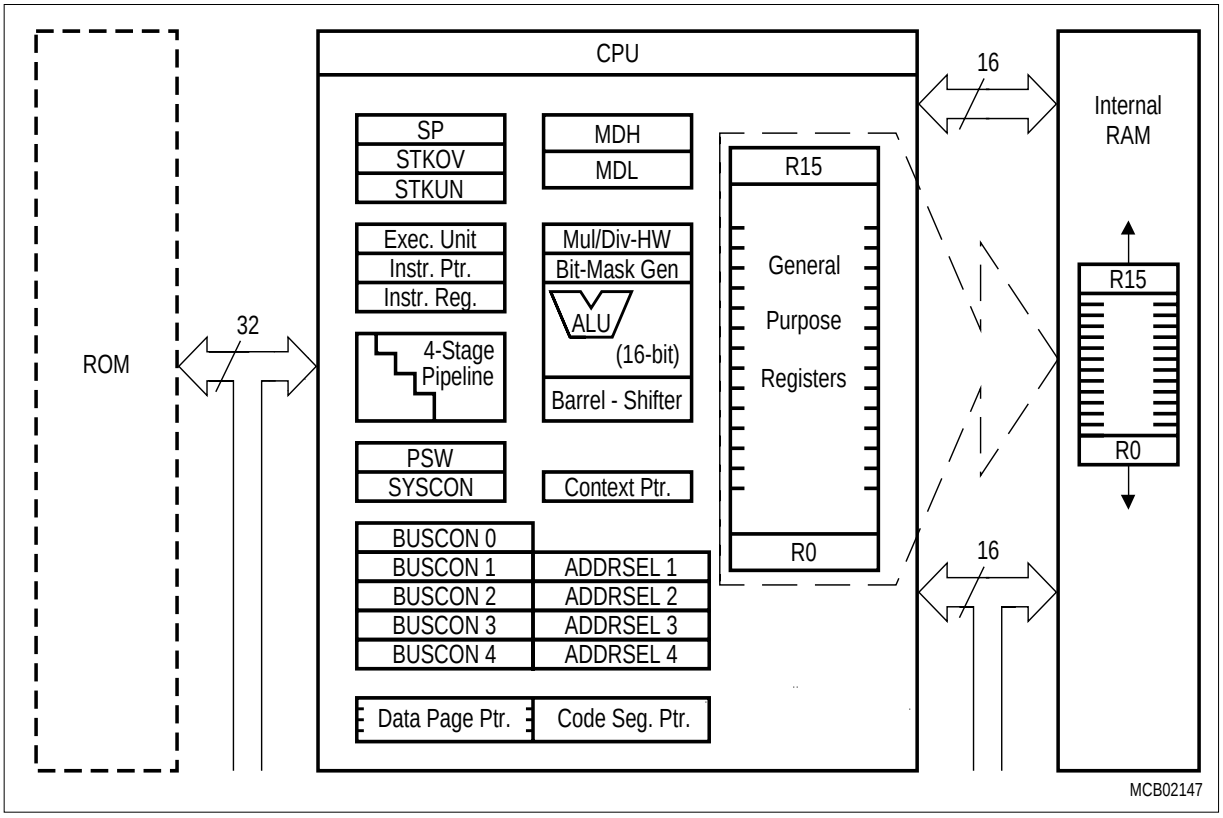


Figure 4-10 CPU Block Diagram

Peripheral units are connected to the CPU by the peripheral bus or the XBUS and can work practically independent of the CPU. Data and control information is interchanged between the CPU and these peripherals by Special Function Registers (SFRs) or external memory locations, depending on to which bus they are connected. Whenever peripherals need a non-deterministic CPU action, the Interrupt Controller compares all pending peripheral service requests with each other and prioritizes one of them. If the priority of the current CPU operation is lower than the priority of the selected peripheral request, an interrupt will occur.

Basically, there are two types of interrupt processing:

- **Standard interrupt processing** forces the CPU to save the current program status and the return address to the stack before branching to the interrupt vector jump table.

- **PEC interrupt processing** steals just one machine cycle from the current CPU activity to perform a single data transfer via the on-chip Peripheral Event Controller (PEC).

System errors detected during program execution (so-called hardware traps) are also processed as standard interrupts with a very high priority.

Besides its normal operation there are the following particular CPU states:

- **Reset state:** Any reset (hardware, software, watchdog) forces the CPU into a predefined active state.
- **IDLE state:** The clock signal to the CPU itself is switched off, while the clocks for the peripherals keep running.
- **POWER DOWN state:** All of the on-chip clocks are switched off.

A transition into an active CPU state is forced by an interrupt (if in IDLE mode) or by a reset (if in POWER DOWN mode).
The IDLE, POWER DOWN and RESET states can be entered by particular system control instructions.

A set of Special Function Registers is dedicated to the functions of the CPU core:

- General System Configuration: **SYSCON (RP0H)**
- CPU Status Indication and Control: **PSW**
- Code Access Control: **IP, CSP**
- Data Paging Control: **DPP0, DPP1, DPP2, DPP3**
- GPRs Access Control: **CP**
- System Stack Access Control: **SP, STKUN, STKOV**
- Multiply and Divide Support: **MDL, MDH, MDC**
- ALU Constants Support: **ZEROS, ONES**

4.6.1 Instruction Pipelining

The instruction pipeline of the CPU separates instruction processing into four stages, and each one has an individual task:

1st –>FETCH:

In this stage the instruction selected by the Instruction Pointer (IP) and the Code Segment Pointer (CSP) is fetched from either the program memory, internal RAM, or external memory.

2nd –>DECODE:

In this stage the instructions are decoded and, if required, the operand addresses are calculated and the respective operands are fetched. For all instructions, which implicitly access the system stack, the SP register is either decremented or incremented, as specified. For branch instructions the Instruction Pointer and the Code Segment Pointer are updated to the desired branch target address (provided that the branch is taken).

3rd –>EXECUTE:

In this stage an operation is performed on the previously fetched operands in the ALU. In addition, the condition flags in the PSW register are updated, as specified by the instruction. All explicit writes to the SFR memory space and all auto-increment or auto-decrement writes to GPRs used as indirect address pointers are also performed during the execute stage of an instruction.

4th –>WRITE BACK:

In this stage all external operands and the remaining operands within the internal RAM space are written back.

A particularity of the CPU are the so-called imported instructions. These imported instructions are internally generated by the machine to provide the time needed to process instructions which cannot be processed within one machine cycle. They are automatically imported into the decoding stage of the pipeline, and then they pass through the remaining stages like all standard instructions. Program interrupts are also performed by means of imported instructions. Although these internally imported instructions will not be noticed in reality, they are introduced here to ease the explanation of the pipeline in the following:

Sequential Instruction Processing

Each single instruction has to pass through each of the four pipeline stages regardless of whether all possible stage operations are performed or not. Since passing through one pipeline stage takes at least one machine cycle, any isolated instruction takes at least four machine cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to four instructions. Thus, most of the instructions seem to be processed during one machine cycle as soon as the pipeline has been filled once after reset (see **Figure 4-11**).

Instruction pipelining increases the average instruction throughput considered over a certain period of time. In the following, any execution time specification of an instruction always refers to the average execution time due to pipelined parallel instruction processing.

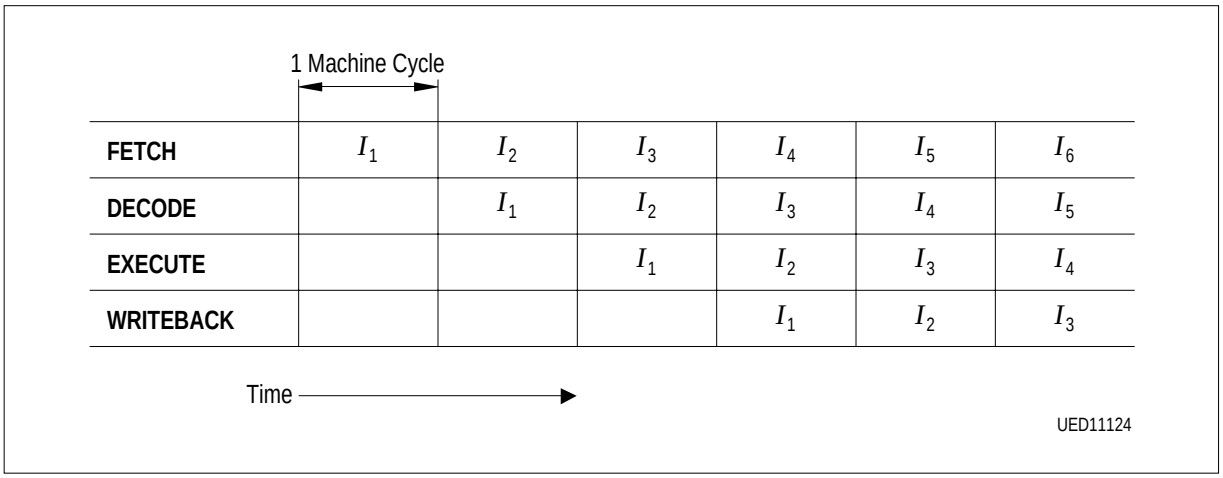


Figure 4-11 Sequential Instruction Pipelining

Standard Branch Instruction Processing

Instruction pipelining helps to speed up sequential program processing. When a branch is taken, the instruction which has been fetched in advance is usually not the instruction which must be decoded next. Thus, at least one additional machine cycle is normally required to fetch the branch target instruction. This extra machine cycle is provided by means of an imported instruction (see **Figure 4-12**).

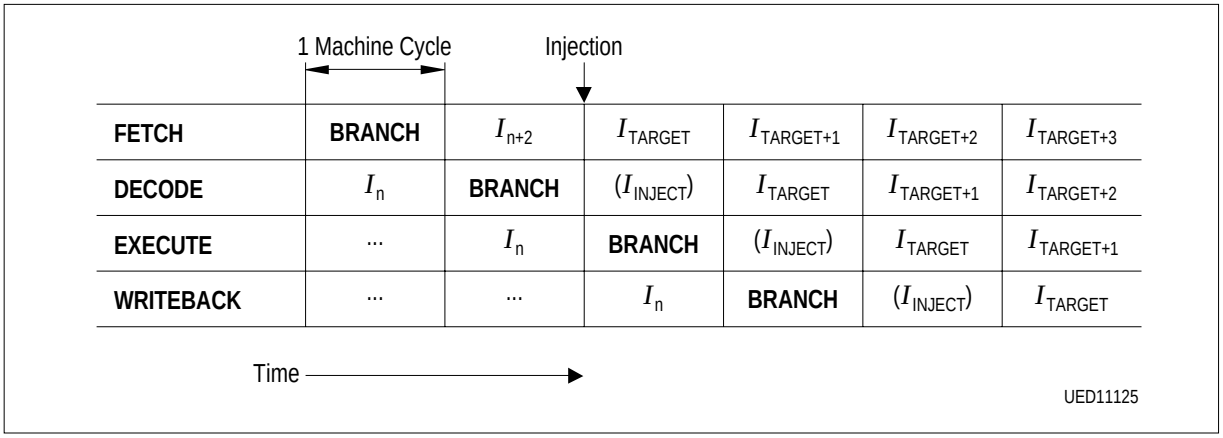


Figure 4-12 Standard Branch Instruction Pipelining

If a conditional branch is not taken, there is no deviation from the sequential program flow, and thus no extra time is required. In this case, the instruction following the branch instruction will enter the decoding stage of the pipeline at the beginning of the next machine cycle after decoding the conditional branch instruction.

Cache Jump Instruction Processing

The CPU incorporates a jump cache to optimize conditional jumps, which are processed repeatedly within a loop. Whenever a jump on cache is taken, the extra time to fetch the branch target instruction can be saved, therefore causing the corresponding cache jump instruction to need only one machine cycle.

This performance is achieved by the following mechanism:
Whenever a cache jump instruction passes through the decode stage of the pipeline for the first time (and provided that the jump condition is met), the jump target instruction is fetched as usual, causing a time delay of one machine cycle. In contrast to standard branch instructions, however, the target instruction of a cache jump instruction (JMPA, JMPR, JB, JBC, JNB, JNBS) is additionally stored in the cache after having been fetched.

After repeatedly following each execution of the same cache jump instruction, the jump target instruction is not fetched from program memory but taken from the cache and immediately imported into the decoding stage of the pipeline (see **Figure 4-13**).

A time saving jump on cache is always taken after the second and any further occurrence of the same cache jump instruction, unless an instruction, which has the fundamental capability of changing the CSP register contents (JMPS, CALLS, RETS, TRAP, RETI), or any standard interrupt has been processed during the period of time between two following occurrences of the same cache jump instruction.

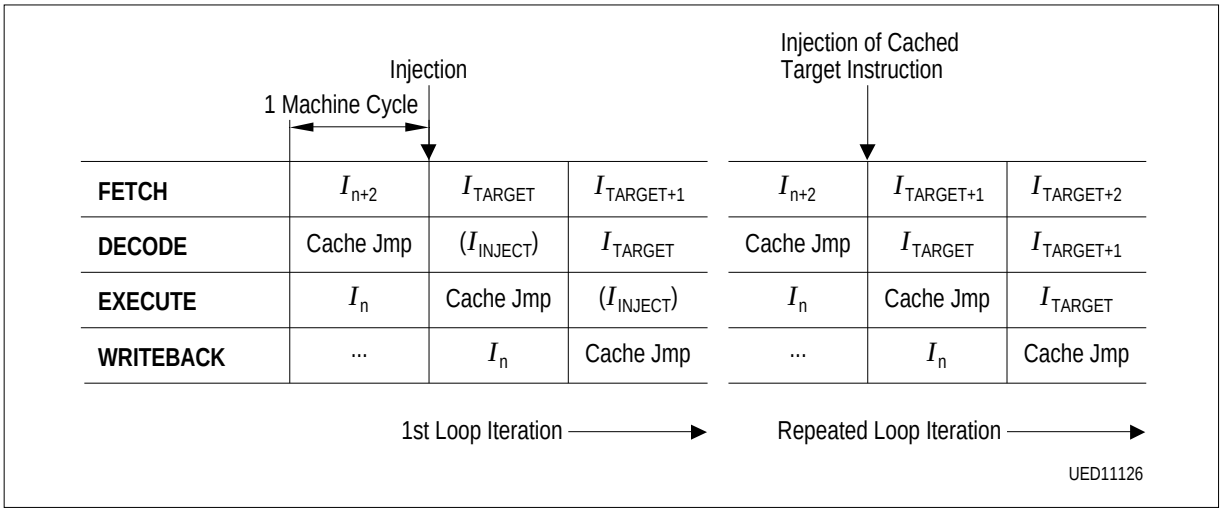


Figure 4-13 Cache Jump Instruction Pipelining

Particular Pipeline Effects

Since up to four different instructions are processed simultaneously, additional hardware has been used in the CPU to consider all causal dependencies which may exist on instructions in different pipeline stages without a loss of performance. This extra hardware (i.e. for “forwarding” operand read and write values) resolves most of the

possible conflicts (e.g. multiple usage of buses) in a time optimized way and thus usually avoids the pipeline being noticed by the user. However, there are some very rare cases, where the CPU, being a pipelined machine, requires attention by the programmer. In these cases the delays caused by pipeline conflicts can be used for other instructions in order to optimize performance.

Context Pointer Updating

An instruction which calculates a physical GPR operand address via the CP register, is mostly not capable of using a new CP value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new CP value is used, at least one instruction must be inserted between a CP-changing and a subsequent GPR-using instruction, as shown in the following example:

```
In      : SCXT CP, #0FC00h      ; select a new context
In+1    : ....                  ; must not be an instruction using a GPR
In+2    : MOV   R0, #dataX      ; write to GPR 0 in the new context
```

Data Page Pointer Updating

An instruction, which calculates a physical operand address via a particular DPPn (n = 0 to 3) register, is mostly not capable of using a new DPPn register value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new DPPn register value is used, at least one instruction must be inserted between a DPPn-changing instruction and a subsequent instruction which implicitly uses DPPn via a long or indirect addressing mode, as shown in the following example:

```
In      : MOV   DPP0, #4        ; select data page 4 via DPP0
In+1    : ....                  ; must not be an instruction using DPP0
In+2    : MOV   DPP0:0000H, R1; move contents of R1 to address location
                                  01'0000H
                                ; (in data page 4) supposed segmentation
                                  is enabled
```

Explicit Stack Pointer Updating

None of the RET, RETI, RETS, RETP or POP instructions are capable of correctly using a new SP register value, which is to be updated by an immediately preceding instruction. Thus, in order to use the new SP register value without erroneously performed stack accesses, at least one instruction must be inserted between an explicitly SP-writing and any subsequent just mentioned implicitly SP-using instructions, as shown in the following example:

```
In      : MOV   SP, #0FA40H; select a new top of stack
In+1    : ....              ; must not be an instruction popping
                              operands
                            ; from the system stack
```

```
I_n+2       : POP   R0          ; pop word value from new top of stack into
                                  R0
```

Note: Conflicts with instructions writing to the stack (PUSH, CALL, SCXT) are solved internally by the CPU logic.

Controlling Interrupts

Software modifications (implicit or explicit) of the PSW are done in the execute phase of the respective instructions. In order to maintain fast interrupt responses, however, the current interrupt prioritization round does not consider these changes, i.e. an interrupt request may be acknowledged after the instruction that disables interrupts via IEN or ILVL or after the subsequent instructions. Timecritical instruction sequences therefore should not begin directly after the instruction disabling interrupts, as shown in the following example:

```
INT_OFF:   BCLR IEN          ; globally disable interrupts
           I_N-1             ; non-critical instruction
CRIT_1ST:  I_N               ; begin of uninterruptable critical sequence
           ...
CRIT_LAST:I_N+x              ; end of uninterruptable critical sequence
INT_ON:    BSET IEN          ; globally re-enable interrupts
```

Note: The described delay of 1 instruction also applies for enabling the interrupts system i.e. no interrupt requests are acknowledged until the instruction after the enabling instruction.

Changing the System Configuration

The instruction following an instruction that changes the system configuration via register SYSCON (e.g. the mapping of the program memory, segmentation, stack size) cannot use the new resources (e.g. program memory or stack). In these cases an instruction that does not access these resources should be inserted. Code accesses to the new program memory area are only possible after an absolute branch to this area.

Note: As a rule, instructions that change program memory mapping should be executed from internal RAM or external / XBUS memory.

BUSCON/ADDRSEL and XBCON/XADRS

The instruction following an instruction that changes the properties of an external / XBUS address area cannot access operands within the new area. In these cases an instruction that does not access this address area should be inserted. Code accesses to the new address area should be made after an absolute branch to this area.

Note: As a rule, instructions that change external bus properties should not be executed from the respective external memory area.

Timing

Instruction pipelining reduces the average instruction processing time on a wide scale (usually from four to one machine cycles). However, there are some rare cases where a particular pipeline situation causes the processing time for a single instruction to be extended either by a half or by one machine cycle. Although this additional time represents only a tiny part of the total program execution time, it might be of interest to avoid these pipeline-caused time delays in time critical program modules.

Besides a general execution time description, the following section provides some hints on how to optimize time-critical program parts with regard to such pipeline-caused timing particularities.

4.6.2 Bit-Handling and Bit-Protection

The CPU provides several mechanisms to manipulate bits. These mechanisms either manipulate software flags within the internal RAM, control on-chip peripherals via control bits in their respective SFRs or control IO functions via port pins.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV, BMOVN explicitly set or clear specific bits. The instructions BFLDL and BFLDH allow the manipulation of up to 8 bits of a specific byte at one time. The instructions JBC and JNBS implicitly clear or set the specified bit when the jump is taken. The instructions JB and JNB (also conditional jump instructions that refer to flags) evaluate the specified bit to determine if the jump is to be taken.

Note: Bit operations on undefined bit locations will always read a bit value of ‘0’, while the write access will not effect the respective bit location.

All instructions that manipulate single bits or bit groups internally use a read-modify-write sequence that accesses the whole word, which contains the specified bit(s).

This method has several consequences:

- Bits can only be modified within the internal address areas, i.e. internal RAM and SFRs. External locations cannot be used with bit instructions.

The upper 256 bytes of the SFR area, the ESFR area and the internal RAM are bit-addressable (see **Chapter 4.2**), i.e. those register bits located within the respective sections can be directly manipulated using bit instructions. The other SFRs must be byte/word accessed.

Note: All GPRs are bit-addressable independent of the allocation of the register bank via the context pointer CP. Even GPRs which are allocated to not bit-addressable RAM locations provide this feature.

- The read-modify-write approach may be critical with hardware-effected bits. In these cases the hardware may change specific bits while the read-modify-write operation is in progress, where the writeback would overwrite the new bit value generated by the hardware. The solution is either the implemented hardware protection (see below) or

realization through special programming (see **“Particular Pipeline Effects” on page 4-29**).

Protected bits are not changed during the read-modify-write sequence, i.e. when hardware sets e.g. an interrupt request flag between the read and the write of the read-modify-write sequence. The hardware protection logic guarantees that only the intended bit(s) is/are effected by the write-back operation.

Note: If a conflict occurs between a bit manipulation generated by hardware and an intended software access the software access has priority and determines the final value of the respective bit.

4.6.3 Instruction State Times

Basically, the time needed to execute an instruction depends on where the instruction is fetched from, and where possible operands are read from or written to. The fastest processing mode of M2 is the execution of a program fetched from the program memory. In this case most of the instructions can be processed within just one machine cycle, which is also the general minimum execution time.

This section summarizes the execution times in a very condensed way. A detailed description of the execution times for the various instructions and the specific exceptions can be found in the **“C16x Family Instruction Set Manual”**.

The table below shows the minimum execution times required to process an M2 instruction fetched from the program memory, the internal RAM or from external / XBUS memory. These execution times apply to most of the M2 instructions - except some of the branches, the multiplication, the division and a special move instruction. In case of program execution from the program memory there is no execution time dependency on the instruction length except for some special branch situations. The numbers in the table are in units of CPU clock cycles and assume no wait-states.

Table 4-2 Minimum Execution Times

	Instruction Fetch		Word Operand Access	
Memory Area	**Word Instruction**	**Doubleword Instruction**	**Read from**	**Write to**
Internal code memory	2	2	2	–
Internal RAM	6	8	0/1	0
16-bit Demux Bus	2	4	2	2
16-bit Mux Bus	3	6	3	3

Execution from the internal RAM provides flexibility in terms of loadable and modifiable code on the account of execution time.

Execution from external memory strongly depends on the selected bus mode and the programming of the bus cycles (wait-states).

The operand and instruction accesses listed below can extend the execution time of an instruction:

- Internal code memory operand reads (same for byte and word operand reads)
- Internal RAM operand reads via indirect addressing modes
- Internal SFR operand reads immediately after writing
- External operand reads
- External operand writes
- Jumps to non-aligned double word instructions in the program memory space
- Testing Branch Conditions immediately after PSW writes

4.6.4 CPU Special Function Registers

The CPU requires a set of Special Function Registers (SFRs) to maintain the system state information, to supply the ALU with register-addressable constants and to control system and bus configuration, multiply and divide ALU operations, code memory segmentation, data memory paging, and accesses to the General Purpose Registers and the System Stack.

The access mechanism for these SFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can simply be controlled by means of any instruction, which is capable of addressing the SFR memory space, a lot of flexibility has been gained, without the need to create a set of system-specific instructions.

Note, however, that there are user access restrictions for some of the CPU core SFRs to ensure proper processor operations. The instruction pointer IP and code segment pointer CSP cannot be accessed directly. They can only be changed indirectly via branch instructions.

The PSW, SP, and MDC registers can not only be modified explicitly by the programmer, but also implicitly by the CPU during normal instruction processing. Note that any explicit write request (via software) to an SFR supersedes a simultaneous modification by hardware of the same register.

Note: Any write operation to a single byte of an SFR clears the non-addressed complementary byte within the specified SFR.
Non-implemented (reserved) SFR bits cannot be modified, and will always supply a read value of '0'.

System Configuration Register SYSCON

This bit-addressable register provides general system configuration and control functions. The reset value for register SYSCON depends on the state of the PORT4 pins during reset.

SYSCON **Reset Value: 0400_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ(2..0)			ROM S1	SGT DIS	ROM EN	-	-	-	CS CFG	-	-	RSO EN	XPEN	-	-
rw			rw	rw	rw				rw			rw	rw		

Bit	Function
XPEN	**XBUS Peripheral Enable Bit** '0': Accesses to the on-chip X-Peripherals and their functions are disabled. '1': The on-chip X-Peripherals are enabled and can be accessed.
RSOEN	**Reset Output Enable Bit** '0': The contrast reduction signal is driven on pin 104. '1': The reset output signal is driven on pin 104, i.e. pin 104 is pulled low during internal reset sequence. *Note: Refer also to **Chapter 6.7***
CSCFG	**Select Line Configuration Control** '0': Latched select line mode for X-Peripherals. '1': Select lines for access cycles via XBUS are directly derived from the address lines. *Note: CSCFG = '1' is recommended. The effect of the switch is not visible at an external interface.*
ROMEN	**PM-Bus Enable Bit** '0': PM-Bus disabled: accesses to the ROM area use the XBUS. '1': PM-Bus enabled: PM-Bus enabled for access cycles to the ROM area. *Note: The recommended value ROMEN = '1' is set by hardware during reset.*
SGTDIS	**Segmentation Disable/Enable Control** '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit). '1': Segmentation disabled (Only IP is saved/restored).

Bit	Function
ROMS1	**Internal ROM Mapping** '0': External ROM area mapped to segment 0 ($00'0000_H$ … $00'7FFF_H$) '1': External ROM area mapped to segment 1 ($01'0000_H$ … $01'7FFF_H$). *Note: ROMS1 = '0' is recommended.*
STKSZ (2 … 0)	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words.

Note: Register SYSCON cannot be changed after execution of the EINIT instruction.

Segmentation Disable/Enable Control (SGTDIS)

Bit SGTDIS allows to select either the segmented or non-segmented memory mode.
In non-segmented memory mode (SGTDIS = '1') it is assumed that the code address space is restricted to 64 KBytes (segment 0) and thus 16 bits are sufficient to represent all code addresses. For implicit stack operations (CALL or RET) the CSP register is totally ignored and only the IP is saved to and restored from the stack.
In segmented memory mode (SGTDIS = '0') it is assumed that the whole address space is available for instructions. For implicit stack operations (CALL or RET) the CSP register and the IP are saved to and restored from the stack. After reset the segmented memory mode is selected.

Note: Bit SGTDIS controls if the CSP register is pushed onto the system stack in addition to the IP register before an interrupt service routine is entered, and it is repopped when the interrupt service routine is left again.

System Stack Size (STKSZ)

This bitfield defines the size of the physical system stack, which is located in the internal RAM of M2. An area of 32 … 512 words or all of the internal RAM may be dedicated to the system stack. A so-called "circular stack" mechanism allows the use of a bigger virtual stack than this dedicated RAM area.

The Processor Status Word PSW

This bit-addressable register reflects the current state of the microcontroller. Two groups of bits represent the current ALU status, and the current CPU interrupt status. A separate bit (USR0) within register PSW is provided as a general purpose user flag.

PSW **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL(3..0)				IEN	HLD EN	-	-	-	USR0	MUL IP	E	Z	V	C	N
rw				rw	rw				rw	rw	rw	rw	rw	rw	rw

Bit	Function
N	**Negative Result** Set, when the result of an ALU operation is negative.
C	**Carry Flag** Set, when the result of an ALU operation produces a carry bit.
V	**Overflow Result** Set, when the result of an ALU operation produces an overflow.
Z	**Zero Flag** Set, when the result of an ALU operation is zero.
E	**End of Table Flag** Set, when the source operand of an instruction is 8000_H or 80_H.
MULIP	**Multiplication/Division In Progress** '0': There is no multiplication/division in progress. '1': A multiplication/division has been interrupted.
USR0	**User General Purpose Flag** May be used by the application software.
HLDEN, ILVL, IEN	**Interrupt and EBC Control Fields** Define the response to interrupt requests and enable external bus arbitration.

ALU Status (N, C, V, Z, E, MULIP)

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status after the last recently performed ALU operation. They are set by most of the instructions due to specific rules, which depend on the ALU or data movement operation performed by an instruction.

After execution of an instruction which explicitly updates the PSW register, the condition flags cannot be interpreted as described in the following, because any explicit write to the PSW register supersedes the condition flag values, which are implicitly generated by the CPU. Explicitly reading the PSW register provides a read value that represents the state of the PSW register after execution of the immediately preceding instruction.

Note: After reset, all of the ALU status bits are cleared.

- **N-Flag:** For most of the ALU operations, the N-flag is set to ‘1’ if the most significant bit of the result contains a ‘1’, otherwise it is cleared. In the case of integer operations the N-flag can be interpreted as the sign bit of the result (negative: N = ‘1’, positive: N = ‘0’). Negative numbers are always represented as the 2’s complement of the corresponding positive number. The range of signed numbers extends from ‘$–8000_H$’ to ‘$+7FFF_H$’ for the word data type, or from ‘$–80_H$’ to ‘$+7F_H$’ for the byte data type. For Boolean bit operations with only one operand, the N-flag represents the previous state of the specified bit. For Boolean bit operations with two operands, the N-flag represents the logical XORing of the two specified bits.

- **C-Flag:** After an addition, the C-flag indicates that a carry from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison, the C-flag indicates a borrow, which represents the logical negation of a carry for the addition.
  This means that the C-flag is set to ‘1’ if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction, which is performed internally by the ALU as a 2’s complement addition, and the C-flag is cleared when this complement addition causes a carry.
  The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a carry.
  For shift and rotate operations the C-flag represents the value of the bit last shifted out. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritized ALU operation because a ‘1’ is never shifted out of the MSB during the normalization of an operand.
  For Boolean bit operations with only one operand the C-flag is always cleared. For Boolean bit operations with two operands the C-flag represents the logical ANDing of the two specified bits.

- **V-Flag:** For addition, subtraction and 2’s complementation the V-flag is always set to ‘1’, if the result overflows the maximum range of signed numbers, which are representable by either 16 bits for word operations (‘$–8000_H$’ to ‘$+7FFF_H$’), or by 8 bits for byte operations (‘$–80_H$’ to ‘$+7F_H$’), otherwise the V-flag is cleared. Note that the result of an integer addition, integer subtraction, or 2’s complement is not valid if the V-flag indicates an arithmetic overflow.
  For multiplication and division the V-flag is set to ‘1’ if the result cannot be represented in a word data type, otherwise it is cleared. Note that a division by zero will always cause an overflow. In contrast to the result of a division, the result of a multiplication is valid regardless of whether the V-flag is set to ‘1’ or not.
  Since logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.
  The V-flag is also used as a “Sticky Bit” for rotate right and shift right operations. By only using the C-flag, a rounding error caused by a shift right operation can be estimated up to a quantity of one half of the LSB of the result. In conjunction with the

V-flag, the C-flag allows the evaluation of the rounding error with a finer resolution (see **Table 4-3**).
For Boolean bit operations with only one operand the V-flag is always cleared. For Boolean bit operations with two operands the V-flag represents the logical ORing of the two specified bits.

Table 4-3 Shift Right Rounding Error Evaluation

C-Flag	V-Flag	Rounding Error Quantity
0	0	- No rounding error -
0	1	0 < Rounding error < $^1/_2$ LSB
1	0	Rounding error = $^1/_2$ LSB
1	1	Rounding error > $^1/_2$ LSB

- **Z-Flag:** The Z-flag is normally set to ‘1’ if the result of an ALU operation equals zero, otherwise it is cleared.
  For the addition and subtraction with carry, the Z-flag is only set to ‘1’ if the Z-flag already contains a ‘1’ and the result of the current ALU operation additionally equals zero. This mechanism is provided for the support of multiple precision calculations.
  For Boolean bit operations with only one operand the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands the Z-flag represents the logical NORing of the two specified bits. For the prioritized ALU operation the Z-flag indicates whether the second operand is zero or not.

- **E-Flag:** The E-flag can be altered by instructions, which perform ALU or data movement operations. The E-flag is cleared by those instructions which cannot be reasonably used for table search operations. In all other cases the E-flag is set depending on the value of the source operand to signify whether the end of a search table is reached or not. If the value of the source operand of an instruction equals the lowest negative number, which is representable by the data format of the corresponding instruction (‘8000_H’ for the word data type, or ‘80_H’ for the byte data type), the E-flag is set to ‘1’, otherwise it is cleared.

- **MULIP-Flag:** The MULIP-flag will be set to ‘1’ by hardware upon entry to an interrupt service routine, when a multiply or divide ALU operation was interrupted before completion. Depending on the state of the MULIP bit, the hardware decides whether multiplication or division must be continued or not after the end of an interrupt service. The MULIP bit is overwritten with the contents of the stacked MULIP-flag when the return-from-interrupt-instruction (RETI) is executed. This normally means that the MULIP-flag is cleared again after that.

Note: The MULIP flag is a part of the task environment. When the interrupting service routine does not return to the interrupted multiply/divide instruction (i.e. as in the

case of a task scheduler that switches between independent tasks), the MULIP flag must be saved as part of the task environment and must be updated accordingly for the new task before this task is entered.

CPU Interrupt Status (IEN, ILVL)

The Interrupt Enable bit allows for global enabling (IEN = ‘1’) or disabling (IEN = ‘0’) of interrupts. The four-bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity. The interrupt level is updated by hardware upon entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. In case an interrupt level ‘15’ has been assigned to the CPU, it has the highest possible priority, and thus the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details please refer to **Chapter 5**.

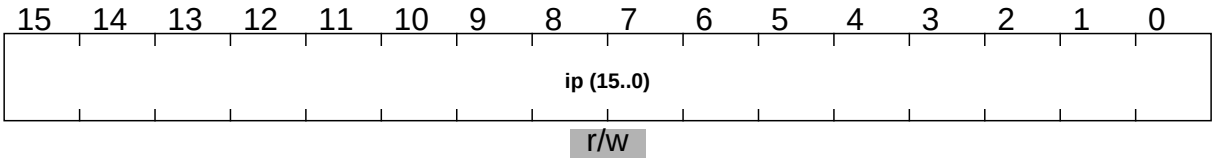
After reset all interrupts are globally disabled, and the lowest priority (ILVL = 0) is assigned to the initial CPU activity.

The Instruction Pointer IP

This register determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register. The IP register is not mapped into the M2’s address space, and thus it is not directly accessible by the programmer. The IP can, however, be modified indirectly via the stack by means of a return instruction.

The IP register is implicitly updated by the CPU for branch instructions and after instruction fetch operations.

IP **Reset Value: 0000_H**



Bit	Function
ip(15 … 0)	Specifies the intra segment offset, from where the current instruction is to be fetched. IP refers to the current segment <SEGNR>.

The Code Segment Pointer CSP

This non-bit addressable register selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up to 256 segments of 64 KBytes each, while the upper 8 bits are reserved for future use.

CSP **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	SEGNR(7..0)							
-	-	-	-	-	-	-	-	r							

Bit	Function
SEGNR (7 … 0)	**Segment Number** Specifies the code segment, from where the current instruction is to be fetched. SEGNR is ignored when segmentation is disabled.

Code memory addresses are generated by directly extending the 16-bit contents of the IP register by the contents of the CSP register, as shown in **Figure 4-14**.

In case of the segmented memory mode the selected number of segment address bits (via bit field SALSEL) of the CSP register is output on the respective segment address pins of Port 4 for all external code accesses. For non-segmented memory mode the content of this register is not significant, because all code accesses are automatically restricted to segment 0.

Note: The CSP register can only be read but not written by data operations. It is, however, modified either directly by means of the JMPS and CALLS instructions, or indirectly via the stack by means of the RETS and RETI instructions.
Upon the acceptance of an interrupt or the execution of a software TRAP instruction, the CSP register is automatically set to zero.

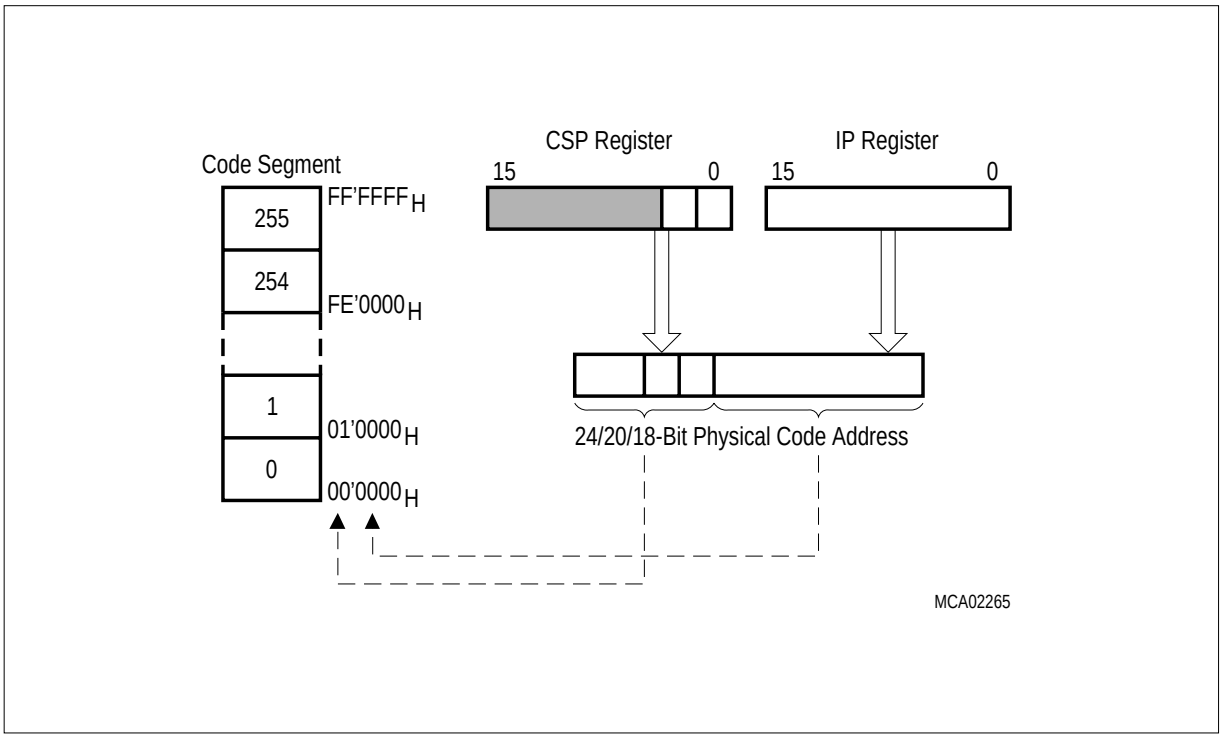


Figure 4-14 Addressing via the Code Segment Pointer

Note: When segmentation is disabled, the IP value is used directly as the 16-bit address.

The Data Page Pointers DPP0, DPP1, DPP2, DPP3

These four non-bit addressable registers select up to four different data pages being active simultaneously at run-time. The lower 10 bits of each DPP register select one of the 1024 possible 16-Kbyte data pages while the upper 6 bits are reserved for future use. The DPP registers allow access to the entire memory space in pages of 16 Kbytes each.

DPP0 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP0PN									
-	-	-	-	-	-	rw									

DPP1 **Reset Value: 0001_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP1PN									
-	-	-	-	-	-	rw									

DPP2 **Reset Value: 0002_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP2PN									
-	-	-	-	-	-	rw									

DPP3 **Reset Value: 0003_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP3PN									
-	-	-	-	-	-	rw									

Bit	Function
DPPxPN	**Data Page Number of DPPx** Specifies the data page selected via DPPx. Only the least significant two bits of DPPx are significant, when segmentation is disabled.

The DPP registers are implicitly used whenever data accesses to any memory location are made via indirect or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers). After reset, the Data Page Pointers are initialized in a way that all indirect or direct long 16-bit addresses result in identical 18-bit addresses. This allows access to data pages 3 … 0 within segment 0 as shown in the figure below. If the user does not want to use any data paging, no further action is required.

Data paging is performed by concatenating the lower 14 bits of an indirect or direct long 16-bit address with the contents of the DPP register selected by the upper two bits of the 16-bit address. The contents of the selected DPP register specify one of the 1024 possible data pages. This data page base address, together with the 14-bit page offset, forms the physical 24-bit address (selectable part is driven to the address pins).

In case of non-segmented memory mode, only the two least significant bits of the implicitly selected DPP register are used to generate the physical address. Thus, extreme care should be taken when changing the content of a DPP register if a non-segmented memory model is selected, because otherwise unexpected results could occur.

In case of the segmented memory mode the selected number of segment address bits (via bit field SALSEL) of the respective DPP register is output on the respective segment address pins of Port 4 for all external data accesses.

A DPP register can be updated via any instruction, which is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a new DPP value is not yet usable for the operand address calculation of the instruction immediately following the updating of the DPP register by the instruction.

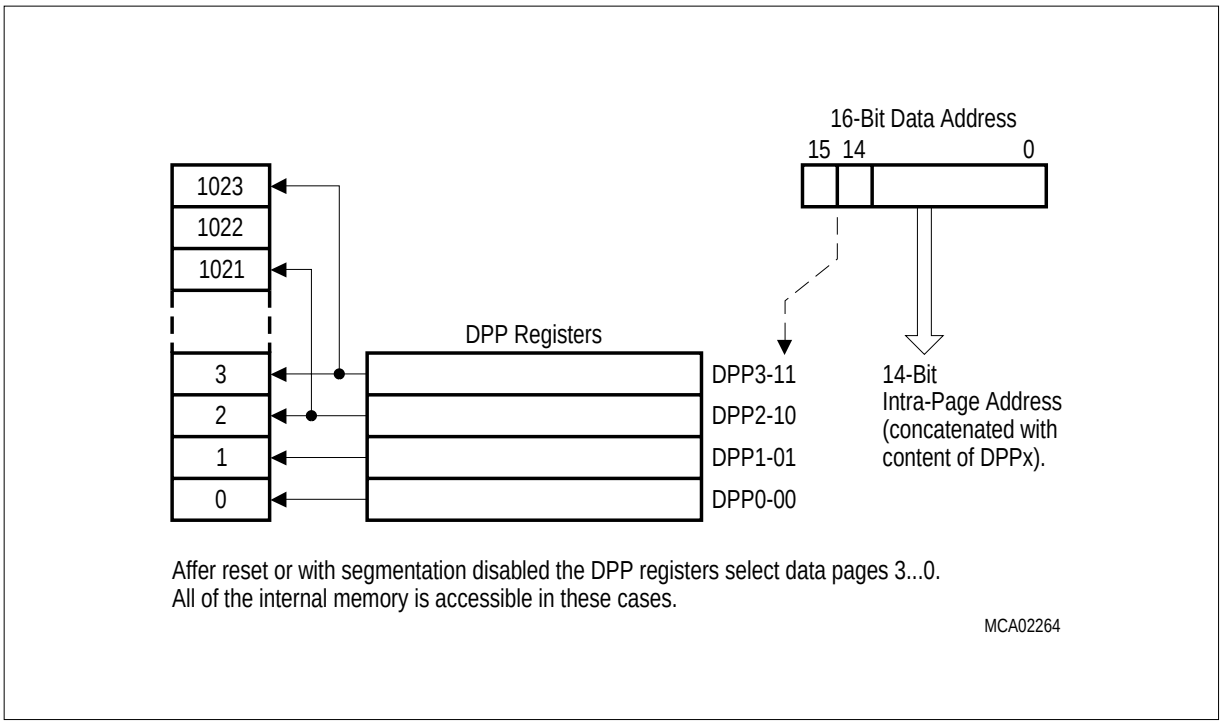


Figure 4-15 Addressing via the Data Page Pointers

The Context Pointer CP

This non-bit addressable register is used to select the current register context. This means that the CP register value determines the address of the first General Purpose Register (GPR) within the current register bank of up to 16 word and/or byte GPRs.

CP **Reset Value: $FC00_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	cp											0
r	r	r	r	rw											r

Bit	Function
cp	**Modifiable Portion of Register CP** Specifies the (word) base address of the current register bank. When writing a value to register CP with bits CP.11 … CP.9 = ‘000’, bits CP.11 … CP.10 are set to ‘11’ by hardware, in all other cases all bits of bit field “cp” receive the written value.

Note: It is the user's responsibility that the physical GPR address specified via CP register plus short GPR address must always be an internal RAM location. If this condition is not met, unexpected results may occur.

- *Do not set CP below the IRAM start address, i.e. 00'FA00$_H$/00'F600$_H$/00'F200$_H$ (1/2/3KB)*
- *Do not set CP above 00'FDFE$_H$*
- *Be careful when using the upper GPRs with CP above 00'FDE0$_H$*

The CP register can be updated via any instruction which is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a new CP value is not yet usable for GPR address calculations of the instruction immediately following the instruction updating the CP register.

The Switch Context instruction (SCXT) allows the saving of the contents of the CP register onto the stack and updating of it with a new value in just one machine cycle.

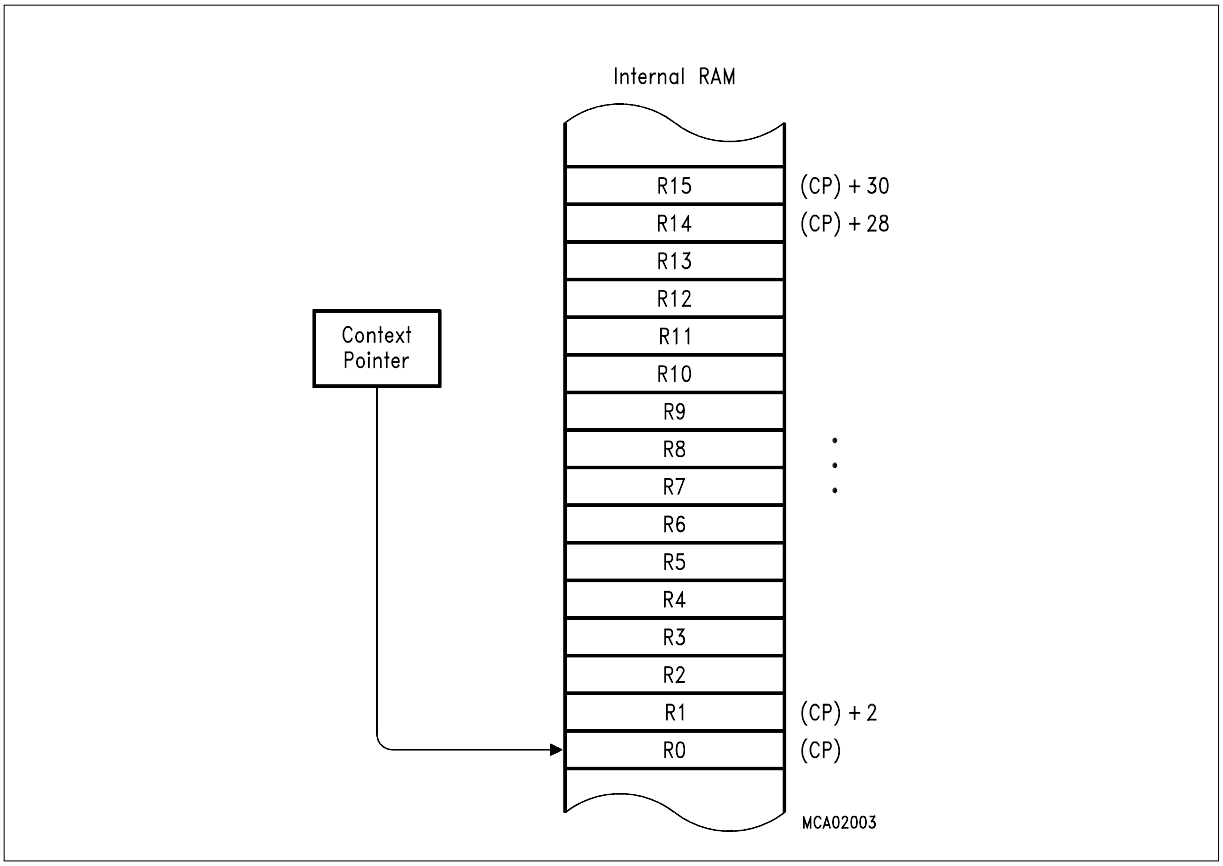


Figure 4-16 Register Bank Selection via Register CP

Several addressing modes use the CP register implicitly for address calculations. The addressing modes mentioned below are described in the **"C16x Family Instruction Set Manual"**.

Short 4-Bit GPR Addresses (mnemonic: Rw or Rb) specify an address relative to the memory location specified by the contents of the CP register, i.e. the base of the current register bank.
Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short 4-bit GPR address is sometimes multiplied by two before it is added to the contents of the CP register (see **Figure 4-17**). Thus, in this way, both byte and word GPR accesses are possible.

GPRs used as indirect address pointers are always word accessed. For some instructions only the first four GPRs can be used as indirect address pointers. These GPRs are specified by short 2-bit GPR addresses. The respective physical address calculation is identical to that for the short 4-bit GPR addresses.

Short 8-Bit Register Addresses (mnemonic: reg or bitoff), within a range from $F0_H$ to FF_H, interpret the four least significant bits as short 4-bit GPR addresses, while the four most significant bits are ignored. The respective physical GPR address calculation is identical to that for the short 4-bit GPR addresses. For single bit accesses on a GPR, the GPR's word address is calculated as described above, but the position of the bit within the word is specified by a separate additional 4-bit value.

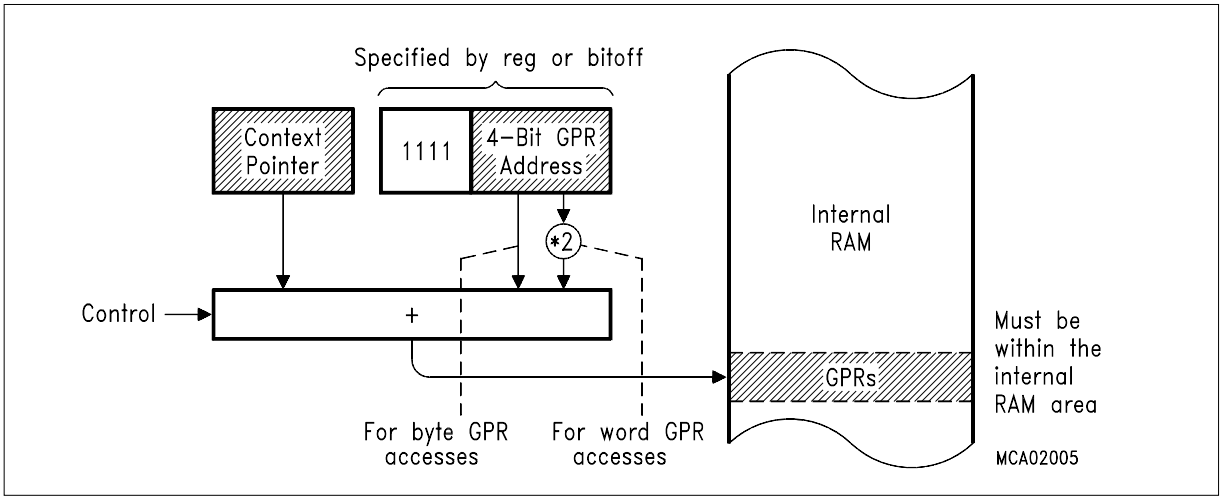


Figure 4-17 Implicit CP Use by Short GPR Addressing Modes

The Stack Pointer SP

This non-bit addressable register is used to point to the top of the internal system stack (TOS). The SP register is pre-decremented whenever data is to be pushed onto the stack, and it is post-incremented whenever data is to be popped from the stack. Thus, the system stack grows from higher to lower memory locations.

Since the least significant bit of the SP register is tied to ‘0’, and bits 15 through 12 are tied to ‘1’ by hardware, the SP register can only contain values from $F000_H$ to $FFFE_H$. This allows access to a physical stack within the internal RAM of the M2. A virtual stack

(usually bigger) can be realized via software. This mechanism is supported by the STKOV and STKUN registers (see respective descriptions below).

The SP register can be updated via any instruction, which is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a POP or RETURN instruction must not immediately follow an instruction updating the SP register.

SP **Reset Value: $FC00_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	sp											0
r	r	r	r	rw											r

Bit	Function
sp	**Modifiable portion of register SP** Specifies the top of the internal system stack.

The Stack Overflow Pointer STKOV

This non-bit addressable register is compared against the SP register after each operation, which pushes data onto the system stack (e.g. PUSH and CALL instructions or interrupts) and after each subtraction from the SP register. If the contents of the SP register are less than the content of the STKOV register, a stack overflow hardware trap will occur.

Since the least significant bit of register STKOV is tied to ‘0’ and bits 15 through 12 are tied to ‘1’ by hardware, the STKOV register can only contain values from $F000_H$ to $FFFE_H$.

STKOV **Reset Value: $FA00_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	stkov											0
r	r	r	r	rw											r

Bit	Function
stkov	**Modifiable portion of register STKOV** Specifies the lower limit of the internal system stack.

The Stack Overflow Trap (entered when (SP) < (STKOV)) may be used in two different ways:

- **Fatal error indication** treats the stack overflow as a system error through the associated trap service routine. Under these circumstances data in the bottom of the

stack may have been overwritten by the status information stacked upon the stack overflow trap service.

- **Automatic system stack flushing** allows the use of the system stack as a “Stack Cache” for a bigger external user stack. In this case the STKOV register should be initialized to a value which represents the desired lowest Top of Stack address plus 12 according to the selected maximum stack size. This takes into consideration the worst case that could occur, when a stack overflow condition is only detected during entry into an interrupt service routine. Then, six additional stack word locations are required to push IP, PSW, and CSP for both the interrupt service routine and the hardware trap service routine.

The Stack Underflow Pointer STKUN

This non-bit addressable register is compared against the SP register after each operation, which pops data from the system stack (e.g. POP and RET instructions) and after each addition to the SP register. If the content of the SP register is greater than the content of the STKUN register, a stack underflow hardware trap will occur.

Since the least significant bit of register STKUN is tied to ‘0’ and bits 15 through 12 are tied to ‘1’ by hardware, the STKUN register can only contain values from $F000_H$ to $FFFE_H$.

STKUN **Reset Value: $FC00_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	stkun											0
r	r	r	r	rw											r

Bit	Function
stkun	**Modifiable portion of register STKUN** Specifies the upper limit of the internal system stack.

The Stack Underflow Trap (entered when (SP) > (STKUN)) may be used in two different ways:

- **Fatal error indication** treats the stack underflow as a system error through the associated trap service routine.
- **Automatic system stack refilling** allows the use of the system stack as a “Stack Cache” for a bigger external user stack. In this case the STKUN register should be initialized to a value which represents the desired highest Bottom of Stack address.

Scope of Stack Limit Control

The stack limit control, realized by the register pair STKOV and STKUN, detects cases where the stack pointer SP is moved outside the defined stack area either by ADD or

SUB instructions or by PUSH or POP operations (explicit or implicit, i.e. CALL or RET instructions).

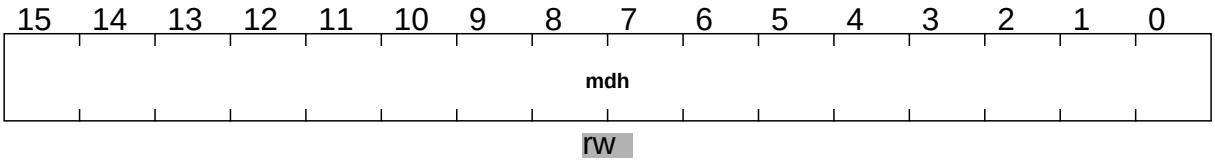
This control mechanism is not triggered, i.e. no stack trap is generated, when

- the stack pointer SP is directly updated via MOV instructions
- the limits of the stack area (STKOV, STKUN) are changed, so that SP is outside of the new limits.

The Multiply/Divide High Register MDH

This register is a part of the 32-bit multiply/divide register, which is implicitly used by the CPU, when it performs a multiplication or a division. After a multiplication, this non-bit addressable register represents the high order 16 bits of the 32-bit result. For long division, the MDH register must be loaded with the high order 16 bits of the 32-bit dividend before the division is started. After any division, the MDH register represents the 16-bit remainder.

MDH **Reset Value: 0000_H**



Bit	Function
mdh	Specifies the high order 16 bits of the 32-bit multiply and divide register MD.

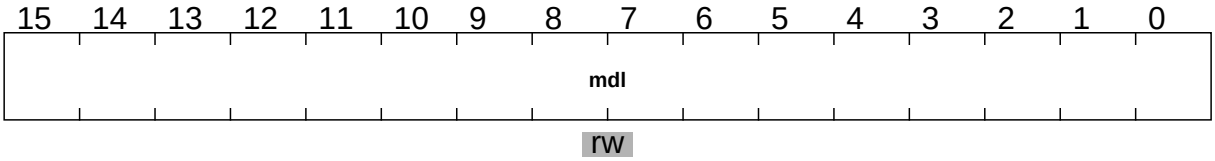
Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to ‘1’.

When multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, the MDH register must be saved along with the MDL and MDC registers to avoid erroneous results.

The Multiply/Divide Low Register MDL

This register is a part of the 32-bit multiply/divide register which is implicitly used by the CPU when it performs a multiplication or a division. After multiplication, this non-bit addressable register represents the low order 16 bits of the 32-bit result. For long division, the MDL register must be loaded with the low order 16 bits of the 32-bit dividend before the division is started. After any division, the MDL register represents the 16-bit quotient.

MDL **Reset Value: 0000_H**



Bit	Function
mdl	Specifies the low order 16 bits of the 32-bit multiply and divide register MD.

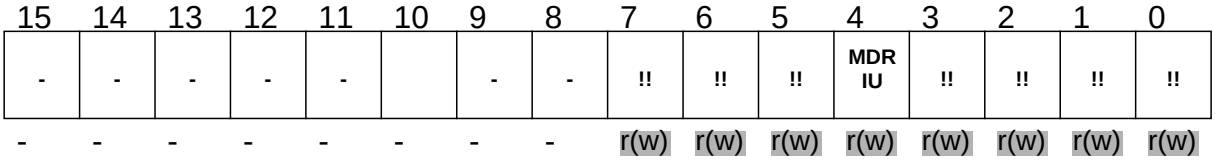
Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to ‘1’. The MDRIU flag is cleared whenever the MDL register is read via software.

When multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, the MDL register must be saved along with the MDH and MDC registers to avoid erroneous results.

The Multiply/Divide Control Register MDC

This bit addressable 16-bit register is implicitly used by the CPU when it performs multiplication or division. It is used to store the required control information for the corresponding multiply or divide operations. The MDC register is updated by hardware during each single cycle of a multiply or divide instruction.

MDC **Reset Value: 0000_H**



Bit	Function
MDRIU	**Multiply/Divide Register In Use** ‘0’: Cleared, when register MDL is read via software. ‘1’: Set when the MDL or MDH register is written via software, or when a multiply or divide instruction is executed.
!!	**Internal Machine Status** The multiply/divide unit uses these bits to control internal operations. Never modify these bits without saving and restoring the MDC register.

When division or multiplication is interrupted before its completion and the multiply/divide unit is required, the MDC register must first be saved along with the MDH and MDL

registers (to be able to restart the interrupted operation later), and then it must be cleared to prepare it for the new calculation. After the completion of the new division or multiplication the state of the interrupted multiply or divide operation must be restored.

The MDRIU flag is the only portion of the MDC register which might be of interest to the user. The remaining portions of the MDC register are reserved for dedicated use by the hardware, and should never be modified by the user in any way other than described above. Otherwise a correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

The Constant Zeros Register ZEROS

All bits of this bit-addressable register are fixed to ‘0’ by hardware. This register can be read only. The ZEROS register can be used as a register-addressable constant of all zeros, i.e. for bit manipulation or mask generation. It can be accessed via any instruction which is capable of addressing an SFR.

ZEROS **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

The Constant Ones Register ONES

All bits of this bit-addressable register are fixed to ‘1’ by hardware. This register can be read only. The ONES register can be used as a register-addressable constant of all ones, i.e. for bit manipulation or mask generation. It can be accessed via any instruction which is capable of addressing an SFR.

ONES ($FF1E_H$ / $8F_H$) **Reset Value: $FFFF_H$**

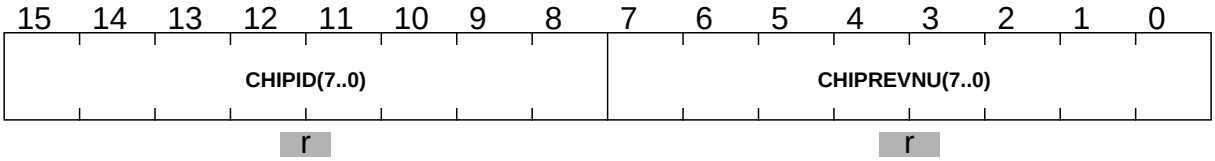
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Note: Register SYSCON cannot be changed after execution of the EINIT instruction.

Identification Register Block

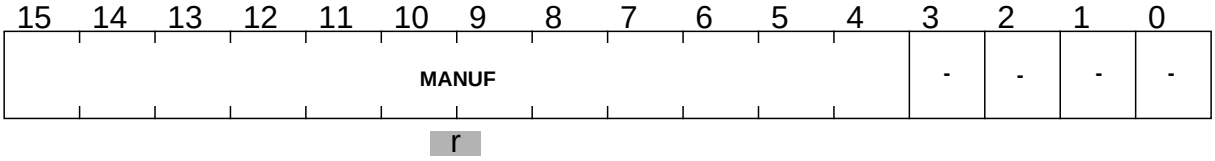
All new derivatives of 16-bit microcontrollers provide a set of identification registers that offer information on the HW-status of the chip.The ID registers are read only registers. They are placed in the extended SFR area.

IDCHIP



Bit	Function
CHIPREVNU (7 … 0)	**Device Revision Code** Identifies the device step where the first release is marked '01_H'.
CHIPID (7 … 0)	**Device Identification** Identifies the device name.

IDMANUF



Bit	Function
MANUF	**JEDEC Normalized Manufacturer Code** $0C1_H$: Infineon Technologies

Interrupt and Trap Function

5 Interrupt and Trap Functions

The C166 architecture supports several mechanisms for fast and flexible response to service requests that can be generated from various sources either by the CPU itself or external, i.e. peripherals connected to the XBUS or the PD bus.

These mechanisms include:

Normal Interrupt Processing

The CPU temporarily suspends the current program execution and branches to an interrupt service routine in order to service an interrupt requesting device. The current program status (IP, PSW, also CSP in segmentation mode) is saved on the internal system stack. A prioritization scheme with 16 priority levels allows the user to specify the order in which multiple interrupt requests are to be handled.

Interrupt Processing via the Peripheral Event Controller (PEC)

A faster alternative to normal software controlled interrupt processing is to service an interrupt requesting device with the integrated Peripheral Event Controller (PEC). Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two locations in the whole memory space through one of nine programmable PEC Service Channels. During a PEC transfer the normal program execution of the CPU is halted for just 1 instruction cycle. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing. PEC transfers share the 2 highest priority levels.

M2 enhances the functionalities of the original C166 PEC with the following features:

- PEC range extended to the entire memory space,
- new chaining mechanism between pairs of PEC channels.

Trap Functions

Trap functions are activated in response to special conditions that occur during the execution of instructions. Several hardware trap functions are provided for handling erroneous conditions and exceptions that arise during the execution of an instruction. Hardware traps always have highest priority and cause immediate system reaction. The software trap function is invoked by the TRAP instruction, which generates a software interrupt for a specified interrupt vector. For all types of traps the current program status is saved on the system stack.

5.1 Interrupt System Structure

M2 provides up to 33 separate interrupt nodes that may be assigned to 16 priority levels. Each node is associated with an interrupt input line in the Interrupt System Interface of the CPU. In order to support modular and consistent software design techniques, all

interrupt nodes are supplied with a separate interrupt control register and interrupt vector. The control register contains the interrupt request flag, the interrupt enable bit, and the interrupt priority of the associated node.

The C166 architecture provides a vectored interrupt system. In this system specific vector locations in the memory space are reserved for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location that is associated with the respective interrupt source. This allows direct identification of the source that caused the request. The only exceptions are the class B hardware traps, which all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) can then be used to determine which exception caused the trap. For the special software TRAP instruction, the vector address is specified by the operand field of the instruction, which is a seven bit trap number.

The reserved vector locations build a jump table in the low end of the address space (segment 0). The jump table is made up of the appropriate jump instructions that transfer control to the interrupt or trap service routines, which may be located anywhere within the address space. The entries of the jump table are located at the lowest addresses in code segment 0 of the address space. Each entry occupies 2 words, except for the reset vector and the hardware trap vectors which occupy 4 or 8 words.

5.1.1 Interrupt Allocation Table

M2 provides 33 separate interrupt nodes that may be assigned to 16 priority levels. In addition to the standard peripheral and external interrupts, there are some teletext related interrupts which support the realtime processing of the sliced data and the generation of the graphical data. Its fast external interrupt inputs are sampled every 3 ns and are even able to recognize very short external signals.

The **Table 5-1** lists all sources that are capable of requesting interrupt or PEC service in M2, the associated interrupt vectors, their locations and the associated trap numbers. It also lists the mnemonics of the affected interrupt request flags and their corresponding interrupt enable flags. The mnemonics are composed of a part that specifies the respective source, followed by a part that specifies their function (IR = Interrupt Request flag, IE = Interrupt Enable flag).

Table 5-1 Interrupt Allocation Table

Source of Interrupt or PEC Service Request	Interrupt Control Register	Address of Control Register	Interrupt Vector Location	Trap Number
External Interrupt 0	EX0IC	$00'FF88_H$	$00'0060_H$	$18_H/24_D$
External Interrupt 1	EX1IC	$00'FF8A_H$	$00'0064_H$	$19_H/25_D$
External Interrupt 2	EX2IC	$00'FF8C_H$	$00'0068_H$	$1A_H/26_D$
External Interrupt 3	EX3IC	$00'FF8E_H$	$00'006C_H$	$1B_H/27_D$

Table 5-1 Interrupt Allocation Table (cont'd)

Source of Interrupt or PEC Service Request	Interrupt Control Register	Address of Control Register	Interrupt Vector Location	Trap Number
External Interrupt 4	EX4IC	$00'FF90_H$	$00'0070_H$	$1C_H/28_D$
External Interrupt 5	EX5IC	$00'FF92_H$	$00'0074_H$	$1D_H/29_D$
External Interrupt 6	EX6IC	$00'FF94_H$	$00'0078_H$	$1E_H/30_D$
External Interrupt 7	EX7IC	$00'FF96_H$	$00'007C_H$	$1F_H/31_D$
GPT1 Timer 2	T2IC	$00'FF60_H$	$00'0088_H$	$22_H/34_D$
GPT1 Timer 3	T3IC	$00'FF62_H$	$00'008C_H$	$23_H/35_D$
GPT1 Timer 4	T4IC	$00'FF64_H$	$00'0090_H$	$24_H/36_D$
GPT2 Timer 5	T5IC	$00'FF66_H$	$00'0094_H$	$25_H/37_D$
GPT2 Timer 6	T6IC	$00'FF68_H$	$00'0098_H$	$26_H/38_D$
GPT2 CAPREL Register	CRIC	$00'FF6A_H$	$00'009C_H$	$27_H/39_D$
A/D1 Conversion Complete	ADC1IC	$00'FF98_H$	$00'00A0_H$	$28_H/40_D$
A/D2 Conversion Complete	ADC2IC	$00'FF9A_H$	$00'00A4_H$	$29_H/40_D$
ASC0 Transmit	S0TIC	$00'FF6C_H$	$00'00A8_H$	$2A_H/42_D$
ASC0 Transmit Buffer	S0TBIC	$00'F19C_H$	$00'011C_H$	$47_H/71_D$
ASC0 Receive	S0RIC	$00'FF6E_H$	$00'00AC_H$	$2B_H/43_D$
ASC0 Autobaud Detection Start	ABSTAIC	$00'FF9E_H$	$00'0084_H$	$21_H/33_D$
ASC0 Autobaud Detection Stop	ABSTOIC	$00'F17A_H$	$00'00F4_H$	$3D_H/61_D$
ASC0 Error	S0EIC	$00'FF70_H$	$00'00B0_H$	$2C_H/44_D$
SSC Transmit	SSCTIC	$00'FF72_H$	$00'00B4_H$	$2D_H/45_D$
SSC Receive	SSCRIC	$00'FF74_H$	$00'00B8_H$	$2E_H/46_D$
SSC Error	SSCEIC	$00'FF76_H$	$00'00BC_H$	$2FH/47_D$
I^2C Data Transfer Event	I^2CTIC	$00'F194_H$	$00'0118_H$	$46_H/70_D$
I^2C Protocol Event	I^2CPIC	$00'F18C_H$	$00'0114_H$	$45_H/69_D$
I^2C Transmission End Event	I^2CTEIC	$00'F184_H$	$00'0110_H$	$44_H/68_D$
ADC Wake Up	ADWIC	$00'F178_H$	$00'00F0_H$	$3C_H/60_D$
ACQ Interrupt *	ACQIC	$00'F176_H$	$00'00EC_H$	$3B_H/59_D$
Display Vertical Sync *	VSDISIC	$00'F174_H$	$00'00E8_H$	$3A_H/58_D$
Display Horizontal Sync *	HSDISIC	$00'F172_H$	$00'00E4_H$	$39_H/57_D$
Graphic Acc. Finished *	GAFIC	$00'FF9C_H$	$00'0080_H$	$20_H/32_D$

Table 5-1 Interrupt Allocation Table (cont'd)

Source of Interrupt or PEC Service Request	Interrupt Control Register	Address of Control Register	Interrupt Vector Location	Trap Number
Realtime Clock	RTCIC	$00'F19E_H$	$00'010C_H$	$43_H/67_D$
PECC Link IRQ	PECCLIC	$00'F180_H$	$00'004C_H$	$4C_H/76_D$

Note: Each entry of the interrupt vector table provides room for two word instructions or one doubleword instruction. The respective vector location results from multiplying the trap number by 4 (4 bytes per entry).

*Note: * = Interrupts relevant for acquisition and graphic support.*

5.1.2 Hardware Traps

The **Table 5-2** lists the vector locations for hardware traps and the corresponding status flags in the TFR register. It also lists the priorities of trap service for cases where more than one trap condition might be detected within the same instruction. After any reset, program execution starts at the reset vector at location $00'0000_H$. Reset conditions have priority over every other system activity and therefore have the highest priority (trap priority IV).

Software traps may be initiated to any vector location between $00'0000_H$ and $00'01FC_H$. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bit field ILVL in register PSW. This means that routines entered via the software TRAP instruction can be interrupted by all hardware traps or higher level interrupt requests.

Table 5-2

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Priority
Reset Functions:					
Hardware Reset	–	RESET	$00'0000_H$	00_H	IV
Software Reset		RESET	$00'0000_H$	00_H	IV
Watchdog Timer Overflow		RESET	$00'0000_H$	00_H	IV
Debug Hardware Trap	DEBUG	DTRAP	$00'0020_H$	08_H	III
Class A Hardware Traps:					
Non-Maskable Interrupt	NMI	NMITRAP	$00'0008_H$	02_H	II
Stack Overflow	STKOF	STOTRAP	$00'0010_H$	04_H	II
Stack Underflow	STKUF	STUTRAP	$00'0018_H$	06_H	II

Table 5-2 (cont'd)

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Priority
Class B Hardware Traps:					
Undefined Opcode	UNDOPC	BTRAP	$00'0028_H$	$0A_H$	I
Protected Instruction Fault	PRTFLT	BTRAP	$00'0028_H$	$0A_H$	I
Illegal Word Operand Access	ILLOPA	BTRAP	$00'0028_H$	$0A_H$	I
Illegal Instruction Access	ILLINA	BTRAP	$00'0028_H$	$0A_H$	I
Illegal External Bus Access	ILLBUS	BTRAP	$00'0028_H$	$0A_H$	I
Reserved	–	–	$[2C_H – 3C_H]$	$[0B_H – 0F_H]$	–
Software Traps TRAP Instruction	–	–	Any $[00'0000_H – 00'01FC_H]$ in steps of 4_H	Any $[00_H – 7F_H]$	Current CPU Priority

Normal Interrupt Processing and PEC Service

During each instruction cycle one out of all sources which require PEC or interrupt processing is selected according to its interrupt priority. This prioritization of interrupts and PEC requests is programmable in two levels. Each requesting source can be assigned to a specific priority. A second level (called "group priority") allows the specification of an internal order for simultaneous requests from a group of different sources on the same priority level. At the end of each instruction cycle the one source request with the highest current priority will be determined by the interrupt system. This request will then be serviced if its priority is higher than the current CPU priority in the PSW register.

Interrupt System Register

Description Interrupt processing is controlled globally by the PSW register through a general interrupt enable bit (IEN) and the CPU priority field (ILVL). Additionally the different interrupt sources are controlled individually by their specific interrupt control registers (… IC). Thus, the acceptance of requests by the CPU is determined by both the individual interrupt control registers and the PSW. PEC services are controlled by the respective PECCx register and the source and destination pointers, which specify the task of the respective PEC service channel.

Interrupt Control Registers

All interrupt control registers are organized identically. The lower 8 bits of an interrupt control register contain the complete interrupt status information of the associated source which is required during one round of prioritization; the upper 8 bits of the respective register are reserved. All interrupt control registers are bit-addressable and all bits can be read or written via software. This allows each interrupt source to be programmed or modified with just one instruction. When accessing interrupt control registers through instructions which operate on word data types, their upper 8 bits (15 … 8) will return zeros when read, and will discard written data.

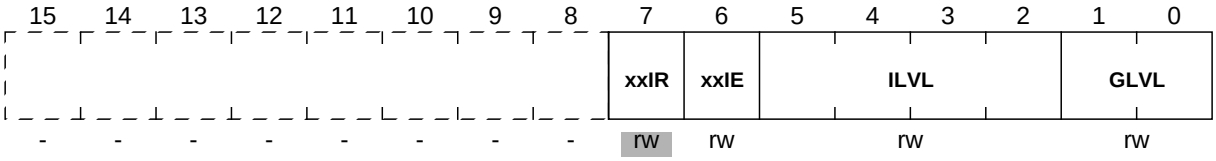
Note: The layout of the Interrupt Control registers shown below applies to each xxIC register, where xx stands for the mnemonic for the respective source.

Interrupt Node Sharing

The interrupt controller of M2 can be configured to control up to 33 different sources. If there is a need for a greater number of interrupt sources to be managed, interrupt requests may share the same interrupt node. In this case, all the sources on the same node share the priority level defined by the corresponding Interrupt Control register xxIC and may be globally enabled/disabled by the IE bit of this register.

Arbitration between sources connected to the same node must be performed by the interrupt handler associated with this node. For low rate requests, the software overhead is not critical.

xxIC **Reset Value: - - 00_H**



Bit	Function
GLVL	**Group Level** Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
ILVL	**Interrupt Priority Level** Defines the priority level for the arbitration of requests. F_H: Highest priority level 0_H: Lowest priority level
xxIE	**Interrupt Enable Control Bit** (individually enables/disables a specific source) '0': Interrupt request is disabled '1': Interrupt Request is enabled
xxIR	**Interrupt Request Flag** '0': No request pending '1': This source has raised an interrupt request

The **Interrupt Request Flag** is set by hardware whenever a service request from the respective source occurs. It is cleared automatically upon entry into the interrupt service routine or upon a PEC service. In the case of PEC service, the Interrupt Request flag remains set if the COUNT field in register PECCx of the selected PEC channel decrements to zero. This allows a normal CPU interrupt to respond to a completed PEC block transfer.

Note: Modifying the Interrupt Request flag via software causes the same effect as if it had been set or cleared by hardware.

Interrupt Priority Level and Group Level

The four bits of an ILVL bit field specify the priority level of a service request for the arbitration of simultaneous requests. The priority increases with the numerical value of ILVL, so 0000_B is the lowest and 1111_B is the highest priority level.

When more than one interrupt request on a specific level becomes active at the same time, the values in the respective GLVL bit fields are used for second level arbitration to select one request for servicing. Again the group priority increases with the numerical value of GLVL, so 00_B is the lowest and 11_B is the highest group priority.

Note: All interrupt request sources that are enabled and programmed to the same priority level must always be programmed to different group priorities. Otherwise an incorrect interrupt vector will be generated.

Upon entry into the interrupt service routine the priority level of the source that won the arbitration and who's priority level is higher than the current CPU level, is copied into the ILVL bit field of register PSW after pushing the old PSW contents onto the stack.

The interrupt system of M2 allows nesting of up to 15 interrupt service routines of different priority levels (level 0 cannot be arbitrated).

Interrupt requests that are programmed to priority levels 15 or 14 (i.e., ILVL = $111X_B$) will be serviced by the PEC unless the COUNT field of the associated PECC register contains zero. In this case the request will instead be serviced by normal interrupt processing. Interrupt requests that are programmed to priority levels 13 through 1 will always be serviced by normal interrupt processing.

Note: Priority level 0000_B is the default level of the CPU. Therefore a request on level 0 will never be serviced, because it can never interrupt the CPU. However, an enabled interrupt request on level 0000_B will terminate the Idle mode and reactivate the CPU.

For interrupt requests which are to be serviced by the PEC, the associated PEC channel number is derived from the respective ILVL (LSB) and GLVL (see **Figure 5-1**). So programming a source to priority level 15 (ILVL = 1111_B) selects the PEC channel group 7 … 4, programming a source to priority level 14 (ILVL = 1110_B) selects the PEC channel group 3 … 0. The actual PEC channel number is then determined by the GLVL group priority field.

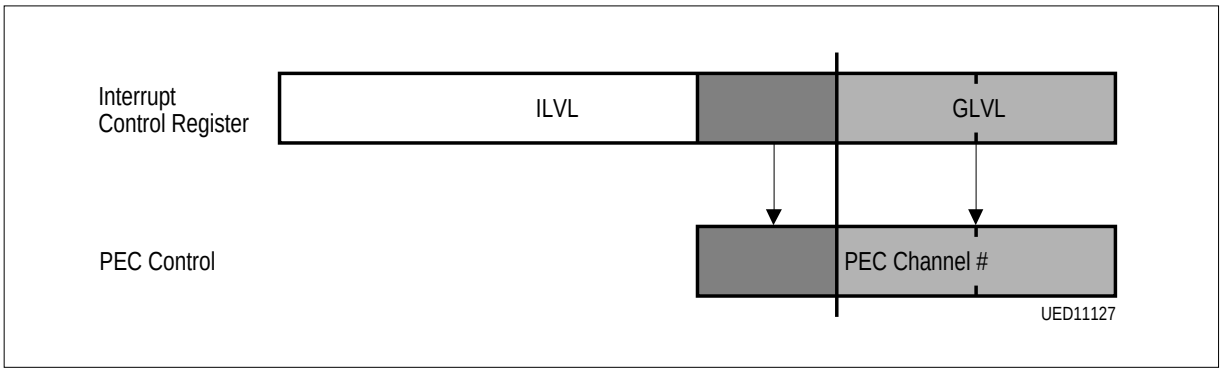


Figure 5-1 Priority Levels and PEC Channels

Simultaneous requests for PEC channels are prioritized according to the PEC channel number, where channel 0 has lowest and channel 8 has highest priority.

Note: All sources that request PEC service must be programmed to different PEC channels. Otherwise an incorrect PEC channel may be activated.

The table below shows a few examples of each action executed with each particular programming of an interrupt control register.

Priority Level		Type of Service	
ILVL	**GLVL**	**COUNT = 00_H**	**COUNT ≠ 00_H**
1 1 1 **1**	**1 1**	CPU interrupt, level 15, group priority 3	PEC service, channel 7
1 1 1 **1**	**1 0**	CPU interrupt, level 15, group priority 2	PEC service, channel 6
1 1 1 **0**	**1 0**	CPU interrupt, level 14, group priority 2	PEC service, channel 2
1 1 0 1	1 0	CPU interrupt, level 13, group priority 2	CPU interrupt, level 13, group priority 2
0 0 0 1	1 1	CPU interrupt, level 1, group priority 3	CPU interrupt, level 1, group priority 3
0 0 0 1	0 0	CPU interrupt, level 1, group priority 0	CPU interrupt, level 1, group priority 0
0 0 0 0	X X	No service!	No service!

Note: All requests on levels 13 … 1 cannot initiate PEC transfers. They are always serviced by an interrupt service routine. No PECC register is associated and no COUNT field is checked.

Interrupt Control Functions in the PSW

The Processor Status Word (PSW) is functionally divided into 2 parts: the lower byte of the PSW basically represents the arithmetic status of the CPU, the upper byte of the PSW controls the interrupt system of M2 and the arbitration mechanism for the external bus interface.

Note: Pipeline effects have to be considered when enabling/disabling interrupt requests via modifications of register PSW.

PSW **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	HLD EN	-	-	-	USR0	MUL IP	E	Z	V	C	N
rw				rw	rw	-	-	-	rw	rw	rw	rw	rw	rw	rw

Bit	Function
N, C, V, Z, E, MULIP, USR0	**CPU status flags** (Described in **Chapter 4.6**) Define the current status of the CPU (ALU, multiplication unit).
HLDEN	**HOLD Enable** (Enables External Bus Arbitration) 0: Bus arbitration disabled, P6.7 ... P6.5 may be used for general purpose IO 1: Bus arbitration enabled, P6.7 ... P6.5 serve as $\overline{\text{BREQ}}$, $\overline{\text{HLDA}}$, $\overline{\text{HOLD}}$, resp.
ILVL	**CPU Priority Level** Defines the current priority level for the CPU F_H: Highest priority level 0_H: Lowest priority level
IEN	**Interrupt Enable Control Bit** (globally enables/disables interrupt requests) ‘0’: Interrupt requests are disabled ‘1’: Interrupt requests are enabled

CPU Priority ILVL defines the current level for the operation of the CPU. This bit field reflects the priority level of the routine that is currently being executed. Upon entry into an interrupt service routine, this bit field is updated with the priority level of the request that is being serviced. The PSW is saved on the system stack before. The CPU level determines the minimum interrupt priority level that will be serviced. Any request on the same or lower level will not be acknowledged.
The current CPU priority level may be adjusted via software, to control which interrupt request sources will be acknowledged.

PEC transfers do not really interrupt the CPU, but rather “steal” a single cycle, so PEC services do not influence the ILVL field in the PSW.

Hardware traps switch the CPU level to maximum priority (i.e. 15) so no interrupt or PEC requests will be acknowledged while an exception trap service routine is being executed.

Note: The TRAP instruction does not change the CPU level, so software invoked trap service routines may be interrupted by higher requests.

Interrupt Enable bit IEN globally enables or disables PEC operations and the acceptance of interrupts by the CPU. When IEN is cleared, no new interrupt requests

are accepted by the CPU. However requests that have already entered the pipeline at that time will be processed. When IEN is set to ‘1’, all interrupt sources, which have been individually enabled by the interrupt enable bits in their associated control registers, are globally enabled.

Note: Traps are non-maskable and are therefore not affected by the IEN bit.

5.2 Operation of the PEC Channels

M2’s Peripheral Event Controller (PEC) provides 8 PEC service channels, which move a single byte or word between two locations in the entire memory space. Packet transfers are provided with channels 0 and 1. This is the fastest possible interrupt response and in many cases is sufficient to service the respective peripheral request (e.g. serial channels, etc.). Each channel is controlled by a dedicated PEC Channel Counter/Control register (PECCx) and a pair of pointers for the source (SRCPx) and destination (DSTPx) of the data transfer.

The PECC registers control the action that is performed by the respective PEC channel.

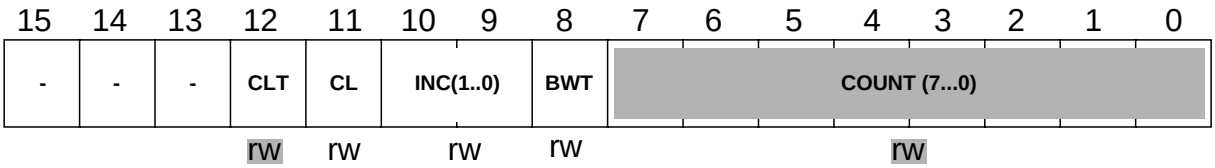
Compared to existing C16x architectures, the PEC transfer function is enhanced by extended functionality. The extended PEC functions are defined as follows:

- Source pointer and destination pointer are extended to 24-bit pointer, thus enabling PEC controlled data transfers between any two locations within the total address space. Both 8-bit segment numbers of every source/destination pointer pair are defined in one 16-bit SFR register; thus, 8 PEC segment number registers are available for the 8 PEC channels.
- For every two channels a chaining feature is provided. When enabled in the PEC control register, a termination interrupt of one channel will automatically switch transfer control to the other channel of the channel pair.

Extended PEC Channel Control

The PEC control registers with the extended functionality and their application for new PEC control are defined as follows:

PECCx **Reset Value: 0000_H**



Bit	Function
COUNT (7 … 0)	**PEC Transfer Count** Counts PEC transfers (bytes or words) and influences the channel's action.
BWT	**Byte / Word Transfer Selection** 0: Transfer a Word. 1: Transfer a Byte.
INC(1 … 0)	**Increment Control** (Modification of SRCPx or DSTPx) 0 0: Pointers are not modified. 0 1: Increment DSTPx by 1 or 2. 1 0: Increment SRCPx by 1 or 2. 1 1: Reserved. Do not use this combination.
CL	**Channel Link Control** 0: PEC channels work independent 1: Pairs of channels are linked together
CLT	**Channel Link Toggle State** 0: Even numbered PEC channel of linked channels active 1: Odd numbered PEC channel of linked channels active

Table 5-3 PEC Control Register Addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PECC0	$FEC0_H$ / 60_H	SFR	PECC4	$FEC8_H$ / 64_H	SFR
PECC1	$FEC2_H$ / 61_H	SFR	PECC5	$FECA_H$ / 65_H	SFR
PECC2	$FEC4_H$ / 62_H	SFR	PECC6	$FECC_H$ / 66_H	SFR
PECC3	$FEC6_H$ / 63_H	SFR	PECC7	$FECE_H$ / 67_H	SFR

Byte/Word Transfer bit BWT controls, if a byte or a word is moved during a PEC service cycle. This selection controls the transferred data size and the increment step for the modified pointer.

Increment Control Field INC controls, if one of the PEC pointers is incremented after the PEC transfer. However, it is not possible to increment both pointers. If the pointers are not modified (INC = '00') the respective channel will always move data from the same source to the same destination.

Note: The reserved combination '11' is changed to '10' by hardware. However, it is not recommended to use this combination.

The PEC Transfer Count Field COUNT controls the action of a respective PEC channel, where the content of bit field COUNT, at the time the request is activated, selects the action. COUNT may allow a specified number of PEC transfers, unlimited transfers or no PEC service at all.

The table below summarizes, how the COUNT field itself, the interrupt requests flag IR and the PEC channel action depends on the previous content of COUNT.

Previous COUNT	Modified COUNT	IR after PEC service	Action of PEC Channel and Comments
FF_H	FF_H	'0'	Move a Byte / Word Continuous transfer mode, i.e. COUNT is not modified
FE_H … 02_H	FD_H … 01_H	'0'	Move a Byte / Word and decrement COUNT
01_H	00_H	'1'	Move a Byte / Word Leave request flag set, which triggers another request
00_H	00_H	('1')	**No action!** Activate interrupt service routine rather than PEC channel.

The PEC transfer counter allows the servicing of a specified number of requests by the respective PEC channel, and then (when COUNT reaches 00_H) activates the interrupt service routine, which is associated with the priority level. After each PEC transfer the COUNT field is decremented and the request flag is cleared to indicate that the request has been serviced.

Continuous transfers are selected by the value FF_H in bit field COUNT. In this case COUNT is not modified and the respective PEC channel services any request until it is disabled again.

When COUNT is decremented from 01_H to 00_H after a transfer the request flag is not cleared, which generates another request from the same source. When COUNT already contains the value 00_H, the respective PEC channel remains idle and the associated interrupt service routine is activated instead. This provides a choice if a level 15 or 14 request is to be serviced by the PEC or by the interrupt service routine.

Note: PEC transfers are only executed if their priority level is higher than the CPU level, i.e. only PEC channels 7 … 4 are processed, while the CPU executes on level 14. All interrupt request sources that are enabled and programmed for PEC service should use different channels. Otherwise only one transfer will be performed for all simultaneous requests. When COUNT is decremented to 00_H, and the CPU is interrupted, an incorrect interrupt vector will be generated.

Channel Link Mode for Data Chaining

Data chaining with linked PEC channels is enabled if the channel link control bit in PECCx register is set to ‘1’ either in one or both PEC channel control registers of a channel pair. In this case, two PEC channels are linked together and handle chained block transfers alternatively to each other. The whole data transfer is divided into several block transfers where each block is controlled by one PEC channel of a channel pair. When a data block is completely transferred a **channel link interrupt** is generated and the PEC service request processing is automatically switched to the “other” PEC channel of the channel-pair. Thus, PEC service requests addressed to a linked PEC channel are either handled by linked PEC channel A or by linked PEC channel B. This channel toggle allows the setting up of shadow and multiple buffers for PEC transfers by changing pointer and count values of one channel while the other channel is active. The following table lists the channels that can be linked together, and the channel numbers to address the linked channels.

Linked PEC Channels		Linked PEC Channel
PEC Channel A	**PEC Channel B**	
channel 0	channel 1	channel 0
channel 2	channel 3	channel 2
channel 4	channel 5	channel 4
channel 6	channel 7	channel 6

For each pair of linked channels an internal channel flag, the channel link toggle flag CLT, identifies which of the two PEC channels will serve the next PEC request. The CLT flag is indicated in both PECCx registers of the two linked PEC channels, where the CLT bit in channel B is always inverse to the CLT bit in channel A. The very first transfer is always started with the channel A if the CLT bit is not otherwise programmed before. The CLT bit is only valid in the case of linked PEC channels, indicated by the CL bits of linked channels. If linking is not enabled, the CLT bit of both channels is always zero.

The internal channel link flag CLT toggles, and the other channel begins servicing with the next request if the “old” channel stops the service (COUNT = 0), and if the new

channel has in its PEC control register the enabled CL flag and if its transfer count is more than zero.

Note: With the last transfer of a block transfer (COUNT = 0), the channel link control flag CL of that channel is cleared in its PECCx register. If the CL channel link flag of the new (chained) PEC control register is found to be zero, the whole data transfer is finished and the channel link interrupt is coincidentally a termination interrupt. The channel link mode is finished and the internal channel toggle flag is cleared after the last transfer of the block, if the CL flags of both pair channels are cleared.

Additional Interrupt Request Node for Channel Link Interrupts

The PEC unit has one dedicated service request node (trap number) for all channel link interrupts. This service request node requests CPU interrupt service in case of one or more channel link request flags and the respective **enable control bit being** set in the channel link interrupt subnode control register (CLISNC). These flags indicate a channel link interrupt condition of linked PEC channels (A and B channels) which requires support from the CPU. The following channel link interrupt conditions requesting CPU service are possible:

- In single transfer mode a COUNT value change from 01_H to 00_H in a linked PEC channel and CL flag is set in the respective PEC control register.

In this case the CPU service is requested to update the PEC control and pointer registers while the next block transfer is executed (the whole transfer is divided into separately controlled block transfers). The last block transfer is determined by the missing link bit in the new (linked) PEC control register. If a new service request hits a linked channel with count equal to zero and channel link flag disabled, a standard interrupt, as known from standard PEC channels, is performed.

The channel link interrupt subnode register CLISNC is defined as follows:

CLISNC **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	C6 IR	C6 IE	-	-	C4 IR	C4 IE	-	-	C2 IR	C2 IE	-	-	C0 IR	C0 IE
		rw	rw			rw	rw			rw	rw			rw	rw

Bit	Function
xxIE	**PEC Channel Link Interrupt Enable Control Bit** (individually enables/disables a specific channel pair interrupt request) '0': PEC interrupt request is disabled '1': PEC interrupt request is enabled
xxIR	**PEC Channel Service Request Flag** '0': No channel link service request pending '1': This source (channel pair) has raised an request to service a PEC channel after channel linking

The source and destination pointers specifiy the locations between which the data is to be moved. PEC transfers can be performed between any locations in the entire memory space of the M2. For each of the 8 PEC channels, the source and destination addresses are specified by a 8-bit segment number and a 16-bit offset. The source and destination segment numbers, respectively PECSSN and PECDSN, are stored in a SFR associated with each channel (PECSNx, see description below). The offset pointers for the source and destination address do not reside in specific SFRs, but are mapped into the internal RAM of the M2 just below the bit-addressable area (see **Figure 5-2**).

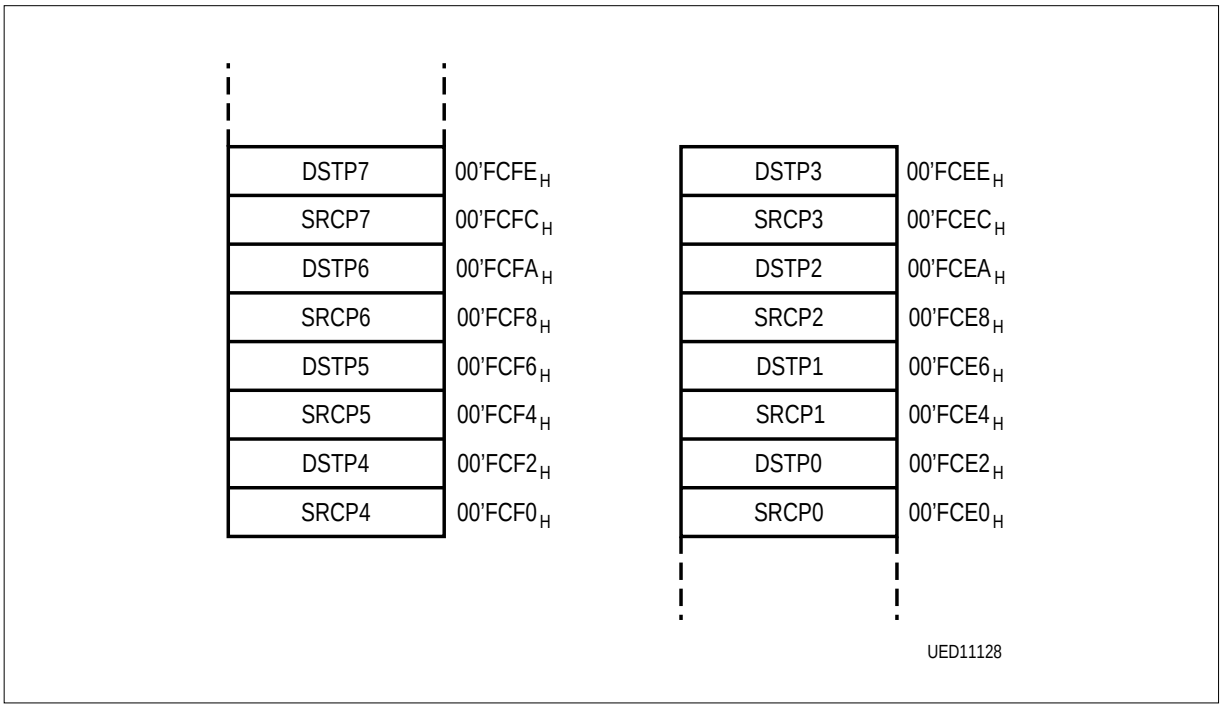
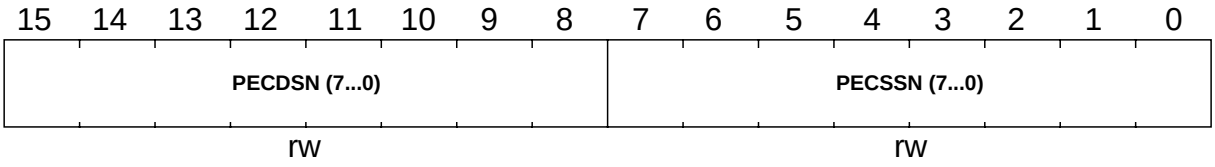


Figure 5-2 Mapping of PEC Offset Pointers into the Internal RAM

The pointer locations for inactive PEC channels may be used for general data storage. Only the required pointers occupy RAM locations.

PECSNx **Reset Value: 0000_H**



Bit	Function
PECSSN (7 … 0)	**PEC Source Segment Number** 8-bit Segment Number (address bits A23 … 16) used for addressing the source of the respective PEC transfer.
PECDSN (7 … 0)	**PEC Destination Segment Number** 8-bit Segment Number (address bits A23 … 16) used for addressing the destination of the respective PEC transfer.

Table 5-4 PEC Segment Number Register Addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PECSN0	$FED0_H$ / 68_H	SFR	PECSN4	$FED8_H$ / $6C_H$	SFR
PECSN1	$FED2_H$ / 69_H	SFR	PECSN5	$FEDA_H$ / $6D_H$	SFR
PECSN2	$FED4_H$ / $6A_H$	SFR	PECSN6	$FEDC_H$ / $6E_H$	SFR
PECSN3	$FED6_H$ / $6B_H$	SFR	PECSN7	$FEDE_H$ / $6F_H$	SFR

If a word data transfer is selected for a specific PEC channel (i.e. BWT = ‘0’), the respective source and destination pointers must both contain a valid word address which points to an even byte boundary. Otherwise the Illegal Word Access trap will be invoked when this channel is used.

5.2.1 Prioritization of Interrupt and PEC Service Requests

Interrupt and PEC service requests from all sources can be enabled so they are arbitrated and serviced (if they win), or they may be disabled so their requests are disregarded and not serviced.

Enabling and disabling interrupt requests may be done via three mechanisms:

Control Bits allow the switching of each individual source to “ON” or “OFF” so that it may generate a request or not. The control bits (xxIE) are located in the respective interrupt control registers. All interrupt requests can generally be enabled or disabled via the IEN bit in register PSW. This control bit is the “main switch” that selects whether requests from any source are accepted or not.
For a specific request to be arbitrated the respective source’s enable bit and the global enable bit must both be set.

The Priority Level automatically selects a certain group of interrupt requests that will be acknowledged, disclosing all other requests. The priority level of the source that won the arbitration is compared with the CPU’s current level and the source is only serviced if its level is higher than the current CPU level. Changing the CPU level to a specific value via software blocks all requests on the same or a lower level. An interrupt source that is assigned to level 0 will be disabled and never be serviced.

The ATOMIC and EXTend instructions automatically disable all interrupt requests for the duration of the following 1 … 4 instructions. This is useful e.g. for semaphore handling and does not require re-enabling the interrupt system after the inseparable instruction sequence.

Interrupt Class Management

An interrupt class covers a set of interrupt sources with the same importance, i.e. the same priority from the system’s viewpoint. Interrupts of the same class must not interrupt each other. M2 supports this function with two features:

Classes with up to 4 members can be established by using the same interrupt priority (ILVL) and assigning a dedicated group level (GLVL) to each member. This functionality is built-in and handled automatically by the interrupt controller.

Classes with more than 4 members can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (4 per ILVL). Each interrupt service routine within this class sets the CPU level to the highest interrupt priority within the class. All requests from the same or any lower level are blocked now, i.e. no request of this class will be accepted.

The example below establishes 3 interrupt classes which cover 2 or 3 interrupt priorities, depending on the number of members in a class. A level 6 interrupt disables all other sources in class 2 by changing the current CPU level to 8, which is the highest priority (ILVL) in class 2. Class 1 requests or PEC requests are still serviced in this case.

The 24 interrupt sources (excluding PEC requests) are assigned to 3 classes of priority rather than to 7 different levels, as the hardware support would do.

Table 5-5 Software controlled Interrupt Classes (Example)

ILVL (Priority	GLVL				Interpretation
	3	2	1	0	
15					PEC service on up to 8 channels
14					
13					
12	X	X	X	X	Interrupt Class 1 5 sources on 2 levels
11	X				
10					
9					
8	X	X	X	X	Interrupt Class 2 9 sources on 3 levels
7	X	X	X	X	
6	X				
5	X	X	X	X	Interrupt Class 3 5 sources on 2 levels
4	X				
3					
2					
1					
0					No service!

5.2.2 Saving the Status during Interrupt Service

Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved along with the location, where the execution of the interrupted task is resumed after returning from the service routine. This return location is specified through the Instruction Pointer (IP) and, in case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register SYSCON controls how the return location is stored.

The system stack first receives the PSW, followed by the IP (unsegmented) or the CSP and then IP (segmented mode). This optimizes the usage of the system stack, if segmentation is disabled.

The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request that is to be serviced, so the CPU now executes on the new level. If multiplication or division was in progress at the time the interrupt request was acknowledged bit MULIP in register PSW is set to ‘1’. In this case the return location that is saved on the stack is not the next instruction in the instruction flow, but rather the multiply or divide instruction itself, as this instruction has been interrupted and will be completed after returning from the service routine.

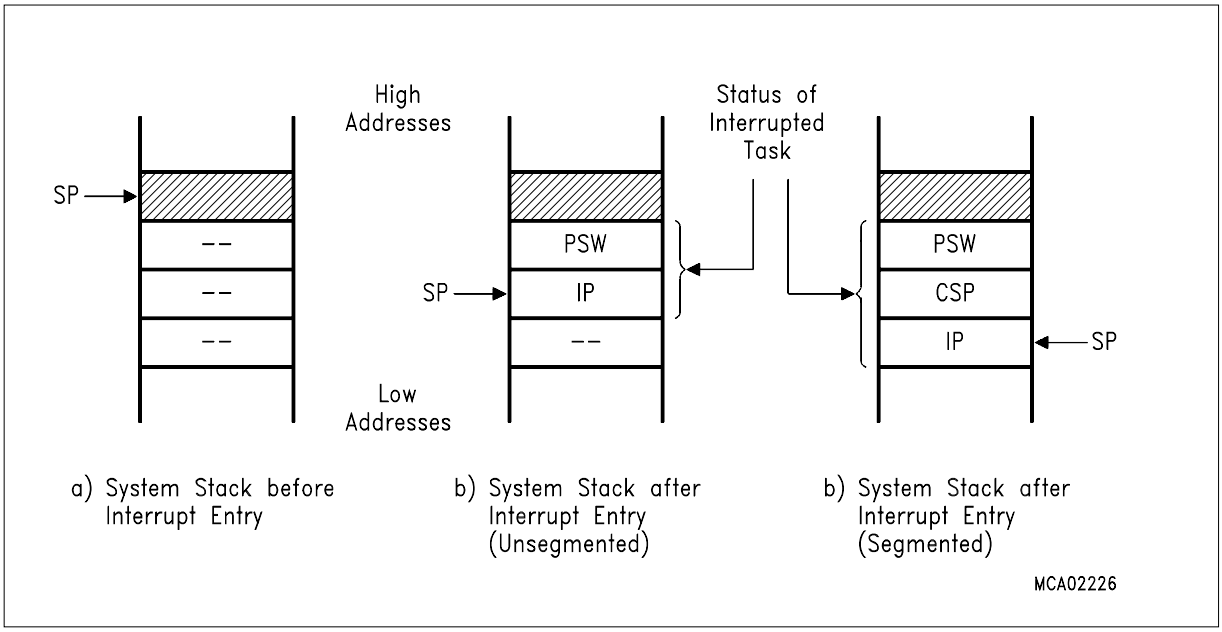


Figure 5-3 Task Status Saved on the System Stack

The interrupt request flag of the source that is being serviced is cleared. The IP is loaded with the vector associated with the requesting source (the CSP is cleared in case of segmentation) and the first instruction of the service routine is fetched from the respective vector location, which is expected to branch to the service routine itself. The data page pointers and the context pointer are not affected.

When the interrupt service routine is left (RETI is executed), the status information is popped from the system stack in reverse order, taking into account the value of bit SGTDIS.

Context Switching

An interrupt service routine usually saves all the registers it uses on the stack, and restores them before returning. The more registers a routine uses, the more time is wasted with saving and restoring. The M2 allows the complete bank of CPU registers (GPRs) to switch with a single instruction, so the service routine executes within its own, separate context.

The instruction “SCXT CP, #New_Bank” pushes the contents of the context pointer (CP) on the system stack and loads CP with the immediate value “New_Bank”, which selects a new register bank. The service routine may now use its “own registers”. This register bank is preserved when the service routine terminates, i.e. its contents are available on the next call.
Before returning (RETI) the previous CP is simply POPped from the system stack, which returns the registers to the original bank.

Note: The first instruction following the SCXT instruction must not use a GPR.

Resources that are used by the interrupting program must eventually be saved and restored, e.g. the DPPs and the registers of the MUL/DIV unit.

5.2.3 Interrupt Response Times

The interrupt response time defines the time from an interrupt request flag of an enabled interrupt source being set until the first instruction (I1) is fetched from the interrupt vector location. The basic interrupt response time for the M2 is 3 instruction cycles.

Pipeline Stage	Cycle 1	Cycle 2	Cycle 3	Cycle 4
FETCH	N	N + 1	N + 2	I1
DECODE	N - 1	N	TRAP (1)	TRAP (2)
EXECUTE	N - 2	N - 1	N	TRAP
WRITEBACK	N - 3	N - 2	N - 1	N

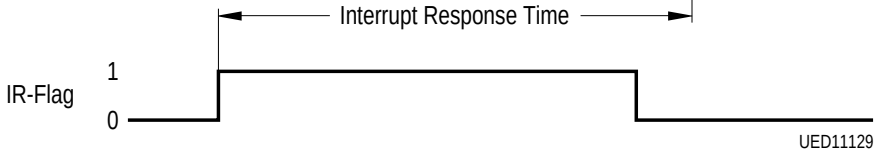


Figure 5-4 Pipeline Diagram for Interrupt Response Time

All instructions in the pipeline, including instruction N (during which the interrupt request flag is set), are completed before entering the service routine. The actual execution time for these instructions (e.g. wait-states) therefore influences the interrupt response time.

In **Figure 5-4** the respective interrupt request flag is set in cycle 1 (the fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a TRAP instruction is injected into the decode stage of the pipeline, replacing instruction N+1 and clearing the source's interrupt request flag to ‘0’. Cycle 4 completes the injected TRAP instruction (save PSW, IP and CSP, if in segmented mode) and fetches the first instruction (I1) from the respective vector location.

All instructions that entered the pipeline, after the setting of the interrupt request flag (N+1, N+2), will be executed after returning from the interrupt service routine.

The minimum interrupt response time is 5 states (10 TCL). This requires program execution from the internal code memory, no external operand read requests and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum interrupt response time under these conditions is 6 state times (12 TCL).

The interrupt response time is increased by all delays of the instructions in the pipeline that are executed before entering the service routine (including N).

- When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, or when instruction N explicitly writes to the PSW or the SP, the minimum interrupt response time may be extended by 1 state time for each of these conditions.
- When instruction N reads an operand from the internal code memory, or when N is a call, return, trap, or MOV Rn, [Rm+ #data16] instruction, the minimum interrupt response time may be extended by 2 state times during internal code memory program execution.
- In case instruction N reads the PSW and instruction N-1 effects the condition flags, the interrupt response time may be extended by 2 state times.

The worst case interrupt response time during internal code memory program execution adds 12 state times (24 TCL).

Any reference to external locations increases the interrupt response time due to pipeline related access priorities. The following conditions have to be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

There are a number of combinations depending on where the instructions, source and destination operands are located. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time, including external accesses, will occur when instructions N, N+1 and N+2 are executed out of external memory, instructions N-1

and N require external operand read accesses, instructions N-3 through N write back external operands, and the interrupt vector also points to an external location. In this case the interrupt response time is the time needed to perform 9 word bus accesses, because instruction I1 cannot be fetched via the external bus until all write, fetch and read requests from preceding instructions in the pipeline are terminated.

- When the interrupt vector, of the example above, is pointing into the internal code memory, the interrupt response time is 7 word bus accesses plus 2 states because the fetching of instruction I1 from internal code memory can start earlier.
- When instructions N, N+1 and N+2 are executed out of the external memory and the interrupt vector points to an external location, but all operands for instructions N-3 through N are in internal memory, then the interrupt response time is the time needed to perform 3 word bus accesses.
- When the interrupt vector, of the example above, is pointing into the internal code memory, the interrupt response time is 1 word bus access plus 4 states.

After an interrupt service routine has been terminated by executing the RETI instruction, and if further interrupts are pending, the next interrupt service routine will not be entered until at least two instruction cycles of the program that was interrupted have been executed. In most cases two instructions will be executed during this time. Only one instruction will typically be executed if the first instruction following the RETI instruction is a branch instruction (without cache hit), if it reads an operand from internal code memory, or if it is executed out of the internal RAM.

Note: A bus access, in this context, includes all delays which can occur during an external bus cycle.

5.2.4 PEC Response Times

The PEC response time defines the time between an interrupt request flag of an enabled interrupt source being set and the PEC data transfer being started. The basic PEC response time for the M2 is 2 instruction cycles.

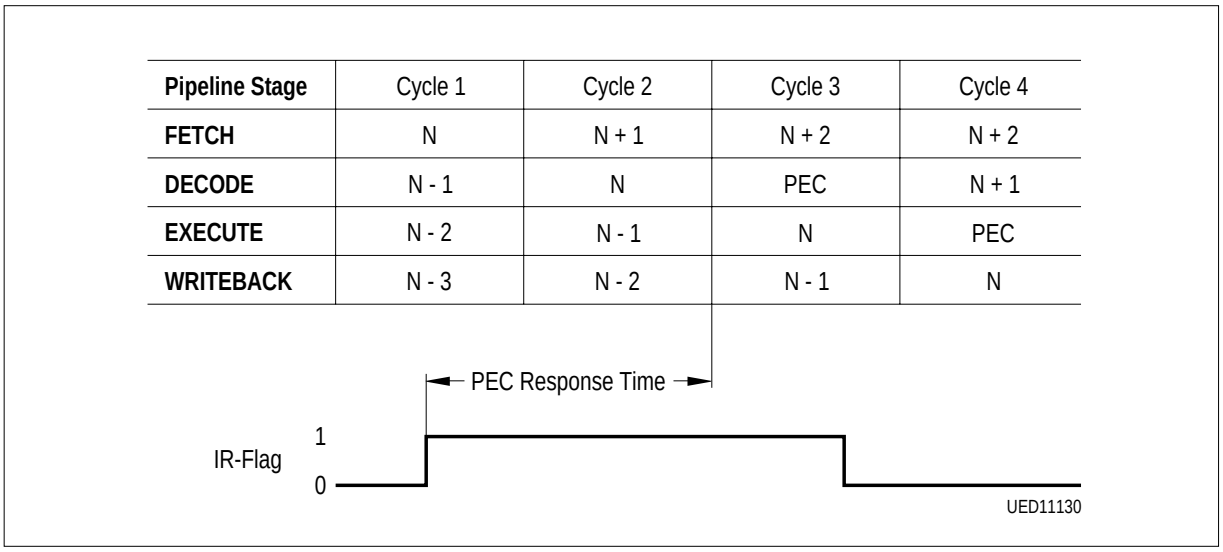


Figure 5-5 Pipeline Diagram for PEC Response Time

In **Figure 5-5**, the respective interrupt request flag is set in cycle 1 (fetching instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a PEC transfer “instruction” is imported into the decode stage of the pipeline, suspending instruction N+1 and clearing the source’s interrupt request flag to ‘0’. Cycle 4 completes the imported PEC transfer and resumes the execution of instruction N+1.

All instructions that entered the pipeline after setting of the interrupt request flag (N+1, N+2) will be executed after the PEC data transfer.

Note: When instruction N reads any of the PEC control registers PECC7 … PECC0, while a PEC request wins the current round of prioritization, the round is repeated and the PEC data transfer is started one cycle later.

The minimum PEC response time is 3 states (6 TCL). This requires program execution from the internal code memory, no external operand read requests and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum PEC response time under these conditions is 4 state times (8 TCL).

The PEC response time is increased by all delays of the instructions in the pipeline which are executed before starting the data transfer (including N).

- When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, the minimum PEC response time may be extended by 1 state time for each of these conditions.
- When instruction N reads an operand from the internal code memory, or when N is a call, return, trap, or MOV Rn, [Rm+ #data16] instruction, the minimum PEC response time may be extended by 2 state times during internal code memory program execution.

- If instruction N reads the PSW and instruction N-1 effects the condition flags, the PEC response time may additionally be extended by 2 state times.

The worst case PEC response time during internal code memory program execution adds to 9 state times (18 TCL).

Any reference to external locations increases the PEC response time due to pipeline related access priorities. The following conditions have to be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

There are a number of combinations depending on where the instructions, source and destination operands are located. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time, including external accesses, will occur when instructions N and N+1 are executed out of external memory, instructions N-1 and N require external operand read accesses and instructions N-3, N-2 and N-1 write back external operands. In this case the PEC response time is the time needed to perform 7 word bus accesses.
- When instructions N and N+1 are executed out of the external memory, but all operands for instructions N-3 through N-1 are in internal memory, then the PEC response time is the time needed to perform 1 word bus access plus 2 state times.

Once a request for PEC service has been acknowledged by the CPU, the execution of the next instruction is delayed by 2 state times plus the additional time it might take to fetch the source operand from internal code memory or external memory and to write the destination operand over the external bus in an external program environment.

Note: A bus access, in this context, includes all delays which can occur during an external bus cycle.

For an EPEC request, the basic response time is 3 instruction cycles. The minimum response time is reached when the request occurs at the end of an instruction cycle. In this case the response time is 5 states (10 TCL). All the conditions described below that may increase the response time apply to the EPEC.

5.2.5 Fast Interrupts

The interrupt inputs are sampled every 8 states (16 TCL), i.e. external events are scanned and detected in timeframes of 16 TCL. M2 provides 8 interrupt inputs that are sampled every 2 TCL, so external events are captured faster than with standard interrupt inputs.

The 8 lines can be programmed individually to this fast interrupt mode, where the trigger transition (rising, falling or both) can also be selected. The External Interrupt Control register EXICON controls this feature for all 8 signals.

EXICON ($F1C0_H$ / $E0_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXI7ES		EXI6ES		EXI5ES		EXI4ES		EXI3ES		EXI2ES		EXI1ES		EXI0ES	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
EXIxES	**External Interrupt x Edge Selection Field (x = 7 … 0)** 0 0: Fast external interrupts disabled: standard mode 0 1: Interrupt on positive edge (rising) 1 0: Interrupt on negative edge (falling) 1 1: Interrupt on any edge (rising or falling)

Note: The fast external interrupt inputs are sampled every 2 TCL. The interrupt request arbitration and processing, however, is executed every 8 TCL.

In Sleep mode, no clock is available for sampling, but interrupt request detection is still possible on fast interrupt request lines using asynchronous logic.

5.3 Trap Functions

Traps interrupt the current execution similar to standard interrupts. However, trap functions offer the possibility to bypass the interrupt system's prioritization process in cases where immediate system reaction is required. Trap functions are not maskable and always have priority over interrupt requests on any priority level.

M2 provides two different kinds of trapping mechanisms. **Hardware traps** are triggered by events that occur during program execution (e.g. illegal access or undefined opcode), **software traps** are initiated via an instruction within the current execution flow.

Software Traps

The TRAP instruction is used to cause a software call to an interrupt service routine. The trap number that is specified in the operand field of the trap instruction determines which vector location in the address range from $00'0000_H$ through $00'01FC_H$ will be branched.

Executing a TRAP instruction causes a similar effect as if an interrupt at the same vector had occurred. PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and a jump is taken to the specified vector location. When segmentation is enabled and a trap is executed, the CSP for the trap service routine is set to code segment 0. No Interrupt Request flags are affected by the TRAP instruction. The

interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

Note: The CPU level in register PSW is not modified by the TRAP instruction, so the service routine is executed on the same priority level from which it was invoked. Therefore, the service routine entered by the TRAP instruction can be interrupted by other traps or higher priority interrupts, other than when triggered by a hardware trap.

Hardware Traps

Hardware traps are issued by faults or specific system states that occur during runtime of a program (not identified at assembly time). A hardware trap may also be triggered intentionally, e.g. to emulate additional instructions by generating an Illegal Opcode trap or to enter the OCDS Software Debug Mode. M2 distinguishes eight different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. Depending on the trap condition, the instruction which caused the trap is either completed or cancelled (i.e. it has no effect on the system state) before the trap handling routine is entered.

Hardware traps are non-maskable and always have priority over every other CPU activity. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced (see **Table 5-1**).

PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and the CPU level in the PSW register is set to the highest possible priority level (i.e. level 15), disabling all interrupts. The CSP is set to code segment zero, if segmentation is enabled. A trap service routine must be terminated with the RETI instruction.

The nine hardware trap functions of M2 are divided into two classes:

Class A traps are

- external Non-Maskable Interrupt (NMI)
- Stack Overflow
- Stack Underflow trap

These traps share the same trap priority, but have an individual vector address.

Class B traps are

- Undefined Opcode
- Protection Fault
- Illegal Word Operand Access
- Illegal Instruction Access
- Illegal External Bus Access Trap

These traps share the same trap priority and vector address.

The **Debug Trap** (see chapter x “OCDS”) is set apart and has its own individual priority and vector address.

The bit-addressable Trap Flag Register (TFR) allows a trap service routine to identify the kind of trap which caused the exception. Each trap function is indicated by a separate request flag. When a hardware trap occurs, the corresponding request flag in the TFR register is set to ‘1’.

TFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	STK OF	STK UF	DE BUG	-	-	-	-	UND OPC	-	-	-	PRT FLT	ILL OPA	ILL INA	ILL BUS
rw	rw	rw	rw	-	-	-	-	rw	-	-	-	rw	rw	rw	rw

Bit	Function
ILLBUS	**Illegal External Bus Access Flag** An external access has been attempted without a defined external bus.
ILLINA	**Illegal Instruction Access Flag** A branch to an odd address has been attempted.
ILLOPA	**Illegal Word Operand Access Flag** A word operand access (read or write) to an odd address has been attempted.
PRTFLT	**Protection Fault Flag** A protected instruction with an illegal format has been detected.
UNDOPC	**Undefined Opcode Flag** The currently decoded instruction has no valid M2 opcode.
DEBUG	**Debug Trap Flag** A debug event programmed to trigger a Debug Trap has been detected by the OCDS.
STKUF	**Stack Underflow Flag** The current stack pointer value exceeds the content of register STKUN.
STKOF	**Stack Overflow Flag** The current stack pointer value falls below the content of register STKOV.

Note: The trap service routine must clear the respective trap flag, otherwise a new trap will be requested after exiting the service routine. Setting a trap request flag by software causes the same effects as if it had been set by hardware.

The reset functions (hardware, software, watchdog) may be regarded as a type of trap. Reset functions have the highest system priority (trap priority IV).

The Debug Trap has the second highest priority (trap priority III) and can interrupt any class A or class B trap. If a class A or class B trap and a Debug Trap occur at the same time, both flags are set in the TFR but the Debug Trap is executed first.

Class A traps have the third highest priority (trap priority II), class B traps are on the 4rd rank so a class A trap can interrupt a class B trap. If more than one class A trap occurs at a time, they are prioritized internally, with the NMI trap on the highest and the stack underflow trap on the lowest priority.

All class B traps have the same trap priority (trap priority I). When several class B traps are activated at a time, the corresponding flags in the TFR register are set and the trap service routine is entered. Since all class B traps have the same vector, the priority of service of simultaneously occurring class B traps is determined by software in the trap service routine.

A class A trap occurring during the execution of a class B trap service routine will be serviced immediately. However, during the execution of a class A trap service routine, any class B trap which occurs will not be serviced until the class A trap service routine is exited with a RETI instruction. In this case, the occurrence of the class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap is lost.

In the case where e.g. an Undefined Opcode trap (class B) occurs simultaneously with an NMI trap (class A), both the NMI and the UNDOPC flag is set, the IP of the instruction with the undefined opcode is pushed onto the system stack, but the NMI trap is executed. After returning from the NMI service routine, the IP is popped from the stack and immediately pushed again because of the pending UNDOPC trap.

Debug Trap

The OCDS may be programmed to trigger a Debug Trap when a debug event (match of data/address comparison, execution of DEBUG instruction, event on *brk_in_n* input) rises. This is normally used to call a monitor routine (software debug mode) for debugging purposes. Normal program execution resumes when a regular RETI instruction is executed, which ends the monitor routine. This trap has the highest priority (except for reset functions) but the monitor routine can reduce its own priority by writing the ILVL field in the PSW.

External NMI Trap

Whenever a high to low transition on the designated *nmi_n* pin (Non-Maskable Interrupt) is detected, the NMI flag in the TFR register is set and the CPU will enter the NMI trap routine. The IP value pushed on the system stack is the address of the instruction following the one after which normal processing was interrupted by the NMI trap.

Note: The nmi_n pin is sampled with every CPU clock cycle to detect transitions.

Stack Overflow Trap

Whenever the stack pointer is decremented to a value which is less than the value in the stack overflow register STKOV, the STKOF flag in the TFR register is set and the CPU

will enter the stack overflow trap routine. Which IP value will be pushed onto the system stack depends on which operation caused the decrement of the SP. When an implicit decrement of the SP is made through a PUSH or CALL instruction, or upon interrupt or trap entry, the IP value pushed is the address of the following instruction. When the SP is decremented by a subtract instruction, the IP value pushed represents the instruction address following the post subtract-instruction command.

To recover from stack overflow it must be ensured that there is enough excess space on the stack to save the current system state twice (PSW, IP, in segmented mode also CSP). Otherwise, a system reset should be generated.

Stack Underflow Trap

Whenever the stack pointer is incremented to a value which is greater than the value in the stack underflow register STKUN, the STKUF flag is set in the TFR register and the CPU will enter the stack underflow trap routine. Again, which IP value will be pushed onto the system stack depends on which operation caused the increment of the SP. When an implicit increment of the SP is made through a POP or return instruction, the IP value pushed is the address of the following instruction. When the SP is incremented by an add instruction, the pushed IP value represents the instruction address following the post add-instruction command.

Undefined Opcode Trap

When the instruction currently decoded by the CPU does not contain a valid M2 opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The IP value pushed onto the system stack is the address of the instruction that caused the trap.

This can be used to emulate unimplemented instructions. The trap service routine can examine the faulting instruction to decode operands for unimplemented opcodes based on the stacked IP. In order to resume processing, the stacked IP value must be incremented by the size of the undefined instruction, which is determined by the user, before a RETI instruction is executed.

Protection Fault Trap

Whenever one of the special protected instructions is executed where the opcode of that instruction is not repeated twice in the second word of the instruction, and the byte following the opcode is not the complement of the opcode, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, and SRVWDT. The IP value pushed onto the system stack for the protection fault trap is the address of the instruction that caused the trap.

Illegal Word Operand Access Trap

Whenever a word operand read or write access is attempted to an odd byte address, the ILLOPA flag in the TFR register is set and the CPU enters the illegal word operand access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

Illegal Instruction Access Trap

Whenever a branch is made to an odd byte address, the ILLINA flag in the TFR register is set, and the CPU enters the illegal instruction access trap routine. The IP value pushed onto the system stack is the illegal odd target address of the branch instruction.

Illegal External Bus Access Trap

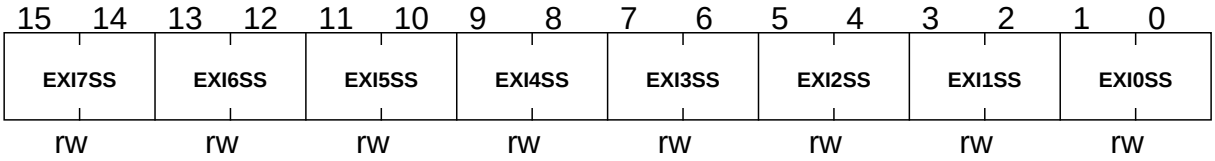
Whenever the CPU requests an external instruction fetch, data read or data write, and no external bus configuration has been specified, the ILLBUS flag in the TFR register is set, and the CPU enters the illegal bus access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

5.3.1 External Interrupt Source Control

Fast external interrupts may also have interrupt sources selected from other peripherals. This function is very advantageous in Slow Down or in Sleep mode if, for example, the A/D converter input shall be used to wakeup the system. The register EXISEL is used to switch alternate interrupt sources to the interrupt controller.

The EXISEL register is defined as follows:

EXISEL **Reset Value: 0000_H**



Bit	Function
EXIxSS	**External Interrupt x Source Selection Field (x = 7 … 0)** 0 0: Input from default pin 0 1: Input from “alternate source” 1 0: Input from default pin ORed with “alternate source” 1 1: Input from default pin ANDed with “alternate source”

Fast Interrupt	Alternate Source (input FEIxIN_B)
0	ADWINT
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

System Control & Configuration

6 System Control & Configuration

M2 has extended features for system level control and configuration. Most of these features are now handled by a new block inside the M2 which is the System Control Unit (SCU). The SCU is used to control system-level tasks such as reset control, clock control and power management. It is implemented to ease compatibility of new M2 based products with already existing C16x derivatives.

M2 provides the following functions for system control and configuration:

- System and Controller Core reset function
- System and Controller Core start-up configuration
- Configuration Registers protection
- Clock Management functions
- Power Management modes (Idle, Sleep, Power Down)
- Watchdog timer
- Identification registers for Core (CPU, SCU, OCDS) and system (manufacturer, chip version, memory) identification

These functions are explained in further details in the following paragraphs.

6.1 System Reset

The internal system reset function provides the initialization of the M2 into a defined default state and is invoked either by asserting a hardware reset signal on pin $\overline{\text{RSTIN}}$ (Hardware Reset Input), upon the execution of the SRST instruction (Software Reset) or by an overflow of the watchdog timer.

Whenever one of these conditions occurs, the CBC is reset into its predefined default state through an internal reset procedure. When a reset, other than a watchdog reset, is initiated, pending internal hold states are cancelled and any external bus cycle is aborted (see description).

After the reset condition is removed, M2 will start program execution from memory location $00'0000_H$ in code segment zero. This start location will typically hold a branch instruction to the start of a software initialization routine for the configuration of peripherals and M2 SFRs.

M2 recognizes the following reset conditions.

Reset Type	Short-cut	Condition
Power-on Reset	PONR	Power-on
Short Hardware Reset	SHWR	16 TCL < t_{RSTIN} ≤ 2048 TCL
Long Hardware Reset	LHWR	t_{RSTIN} > 2048 TCL
Watchdog Timer Reset	WDTR	WDT overflow
Software Reset	SWR	SRST command

Reset conditions are indicated in the WDTCON register.

Hardware Reset

A hardware reset is triggered **asynchronously** by a falling edge of the reset input signal, $\overline{\text{RSTIN}}$. To ensure the recognition of the $\overline{\text{RSTIN}}$ signal, it must be held low for at least 2 CPU clock cycles, assuming the clock input signal is stable. Also, shorter $\overline{\text{RSTIN}}$ pulses may trigger a hardware reset, however, this is not recommended. The internal reset condition is prolonged until one of the following conditions arises:

- the rising edge of the $\overline{\text{RSTIN}}$ signal, or
- the termination of the reset sequence, if $\overline{\text{RSTIN}}$ was deasserted before, or
- the termination of the lengthening conditions.

After termination of the reset state, program execution will start.

Three different kinds of hardware reset conditions are considered:

- **Power-on Reset**
  A complete power-on reset requires an active $\overline{\text{RSTIN}}$ time until a stable clock signal is available. The on-chip oscillator needs about 2 ms to stabilize.
- **Long Hardware Reset**
  A long hardware reset requires an $\overline{\text{RSTIN}}$ active time longer than the duration of the internal reset sequence. The duration of the internal reset sequence is 2056 TCL. After the internal reset sequence has been completed, the $\overline{\text{RSTIN}}$ input is sampled. As long as the reset input is still active the internal reset condition is prolonged. Accordingly, the internal hardware reset (HWRST) is active until the external reset on the $\overline{\text{RSTIN}}$ input becomes inactive.

Note: The hardware reset is also used as a wake up from power down state; in this case the internal system reset will be lengthened (execution of 1. instruction delayed) until the oscillator and PLL have been stabilized.

- **Short Hardware Reset**
  The $\overline{\text{RSTIN}}$ active time of a short hardware reset is between 16 TCL and 2056 TCL. If the $\overline{\text{RSTIN}}$ signal is active for at least 16 TCL clock cycles the internal reset sequence is started (see below). In case of a short HW-reset, the internal HWRST signal is prolonged until the reset sequence is finished.

Software Reset

The reset sequence can be triggered at any time via the protected instruction SRST (Software Reset). This instruction can be executed deliberately within a program, e.g. to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure.

Watchdog Timer Reset

When the watchdog timer is not disabled during the initialization or serviced regularly during program execution it will overflow and trigger the reset sequence. Other than

hardware and software reset the watchdog reset completes a running external bus or X-Bus cycle.

6.1.1 Behavior of I/Os during Reset

During the internal reset sequence all of the M2's I/O pins are configured as inputs by clearing the associated direction registers and switching their pin drivers to the high impedance state. This ensures that the M2 and external devices will not try to drive the same pin to different levels. Outputs are generally driven to their expected inactive state during reset. Output pins driven to '0' are: A13 … A0; MEMCLK, CLKEN, XTAL2, COR, BLANK and TDO. Output pins driven to '1' are: A15 ($\overline{CAS}$),A14 ($\overline{RAS}$), $\overline{RD}$, $\overline{CSROM}$, $\overline{CSDRAM}$, $\overline{WR}$, P4.5($\overline{CS3}$), P4.4 … P4.0, LDQM and UDQM.

6.1.2 Reset Values for the Controller Core Registers

During the reset sequence the registers of the C166 CBC are preset with a default value. Most C166 CBC SFRs are cleared to zero, so the interrupt system is off after reset. A few exceptions to this rule provide a first pre-initialization, which is either fixed or controlled by input pins.

DPP1: 0001_H (points to data page 1)
DPP2: 0002_H (points to data page 2)
DPP3: 0003_H (points to data page 3)
CP: $FC00_H$
STKUN: $FC00_H$
STKOV: $FA00_H$
SP: $FC00_H$
SYSCON: 0400_H (set according to start-up configuration)
BUSCON0: $15B7_H$ (set according to start-up configuration)
ONES: $FFFF_H$ (fixed value)

6.1.3 The Internal RAM after Reset

The contents of the internal RAM are not affected by a reset. However, after a power-on reset, the contents of the internal RAM are undefined. This implies that the GPRs (R15 … R0) and the PEC source and destination pointers (SRCP7 … SRCP0, DSTP7 … DSTP0), which are mapped into the internal RAM, are also unchanged after a warm reset, software reset or watchdog reset, but are undefined after a power-on reset.

6.2 System Start-up Configuration

Although most of the programmable features of the M2 are either selected during the initialization routine or repeatedly during program execution, there are some features

that must be selected earlier, because they are used for the first access of the program execution.

The system start-up configuration is determined by the level on PORT4 at the end of the internal reset sequence. During reset internal pull-up devices are active on PORT4 lines, so their input level is high, if the respective pin is left open, or is low, if the respective pin is connected to an external pull-down device. The value $FFFF_H$ on PORT4 will select the default configuration during reset. If a particular configuration is required, the corresponding line(s) should be driven low according to the coding of the selections, as shown below. Registers SYSCON and BUSCON0 are initialized according to the selected configuration.

Pins that control the operation of the internal control logic and reserved pins are evaluated only during a hardware triggered reset sequence. Pins that influence the configuration of the M2 are evaluated during any reset sequence, e.g. also during software and watchdog timer triggered resets.

The configuration input via PORT4 is latched in register RP0H for subsequent evaluation by software.

RP0H **Reset Value: $00XX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	**1**	**1**	**SALSEL(2..0)**			**1**	**CS ENA**	**1**
														r	

Bit	Function
CSENA	**Chip Select Lines Selection** Number of external ROMs. Description of possible selections: see table below (Start up Configuration)
SALSEL (2 … 0)	**Segment Address Lines Selection** Number of active segment address outputs. Description of possible selections: see table below (Start up Configuration)

Bootstrap Loader Mode

Pin P4.0 (BSL) activates the on-chip bootstrap loader, when low, during reset. The bootstrap loader allows the start code to move into the internal RAM of the M2 via the serial interface ASC0. The M2 will remain in bootstrap loader mode until a hardware reset with P4.0 high or a software reset.

Default: The M2 starts fetching code from location $00'0000_H$, the bootstrap loader is off.

Chip Select Line

Pin P4.1 (CSENA) defines the external memory configuration. When pulled low it enables chip select 3 ($\overline{CS3}$) (a second ROM device is assumed).

RP0H.1 = ‘0’ denotes the memory configuration with only one ROM device while RP0H.1 = ‘1’ indicates availability of the 2nd ROM device.

Note: $\overline{CS3}$ status cannot be changed via software after reset.

Segment Address Lines

The status of Pins P4.5 … P4.3 (SALSEL) during reset defines the number of active segment address lines. This allows the selection which pins of Port 4 drive address lines and which are used for general purpose I/O. The three bits are latched in register RP0H. Depending on the system architecture the required address space is chosen and accessible right from the start, so the initialization routine can directly access all locations without prior programming. The required pins of Port 4 are automatically switched to address output mode.

During runtime the configured number of segment address line can be read from bit field RP0H.5 (= SALSEL.2) … RP0H.3 (= SALSEL.0).

SALSEL	Segment Address Lines	Directly Accessible Address Space
1 1 1	A20, A19, A18, A17, A16	4 MByte (Default without pull-downs)
1 1 0	A19, A18, A17, A16	2 MByte
1 0 1	A18, A17, A16	1 MByte
1 0 0	A17, A16	512 KByte
0 1 1	A16	256 KByte
0 1 0	–	128 KByte
0 0 1	–	128 KByte
0 0 0	–	128 KByte

Default: 5-bit segment address (A20 … A16) allowing access to 4 MByte.

Note: The selected number of segment address lines cannot be changed via software after reset.

6.3 Register Write Protection

The System Control Unit (SCU) provides **two different protection types** of configuration registers:

- Unprotected Registers
- Protectable Registers

The unprotected registers allow the reading and writing (if not read-only) of register values without any restrictions. However, the write access of the protectable registers (security registers) can be programmed for **three different modes of security level**, whereas the read access is always unprotected:

- Write Protected Mode
- Low Protected Mode
- Unprotected Mode

In write protected mode, the registers can not be accessed by a write command. However, in low protected mode, the registers can be written with a special command sequence (see description below). If the registers are set to unprotected mode, all write accesses are possible.

Some register controlled functions and modes which are critical for M2's operation are locked after the execution of EINIT, so these vital system functions cannot be changed inadvertently e.g. by software errors. However, as these security registers also control the power management, they need to be accessed during operation to select the appropriate mode.

The switching between the different security levels is controlled by a state machine. By using a password and command sequence the security levels can be changed. After reset the unprotected mode is always automatically selected. The EINIT command switches the security level automatically to protected mode.

The low protected mode is especially important for a standby state of the application. This mode allows fast accesses within 2 commands to the protected registers without removing the protection completely.

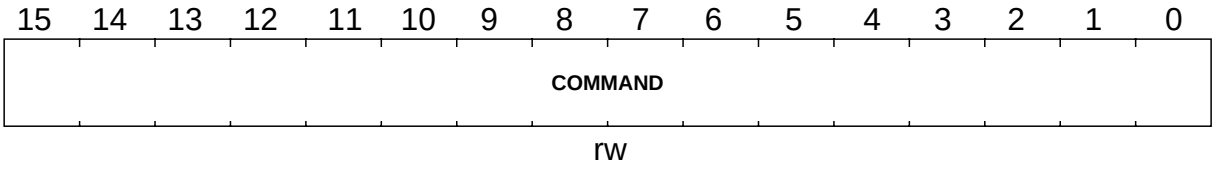
Security Level Switching

Two registers are provided for switching the security level, the security level command register SCUSLC and the security level status register SCUSLS. The security level command register SCUSLC is used to control the state machine for switching the security level. The SCUSLC register is loaded with the different commands of the command sequence necessary to control a change in the security level. It is also used for the one unlock command, which is necessary in the low protected mode to access one protected register. The commands of the (unlock) command sequence are characterized by certain pattern words (such as $AAAA_H$) or by patterns combined with an 8-bit password. For command definition see the following state diagram (**Figure 6-1**).

The new password is defined with command 3 and stored in the according 8-bit field in the SCUSLS register.

The SCUSLC register is defined as follows

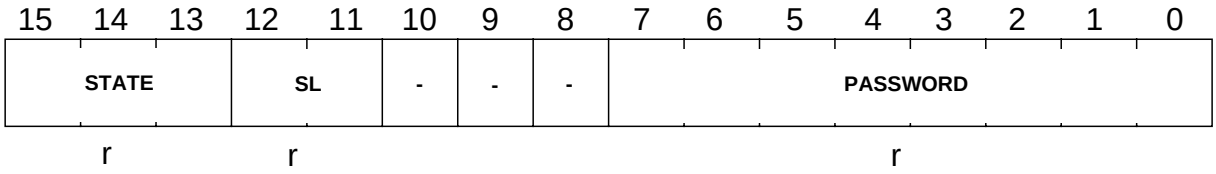
SCUSLC **Reset Value: 0000_H**



Bit	Function
Command	**Code of Command to be Executed** *Command 0*: '$AAAA_H$' *Command 1*: '5554_H' *Command 2*: '96 & inverse password' *Command 3*: '000_b' & new level & '000_b' & new password *Command 4*: '$8E_H$' & inverse password (Command 4 unlocks protected registers for one write access if current security level is in low protected mode.)

The security level status register SCUSLS is a read only register which shows the current password, the actual security level and the state of the switching state machine. The SCUSLS is defined as follows:

SCUSLS **Reset Value: 0000_H**



Bit	Function
PASSWORD	**Current Password**
SL	**Current Security Level** ‘00’: Unprotected Write Mode ‘01’: Low Protected Mode ‘10’: Reserved ‘11’: Write Protected Mode
STATE	**Current State** ‘000’: State 0 = Wait for Command 0. ‘001’: State 1 = Wait for Command 1. ‘010’: State 2 = Wait for Command 2. ‘011’: State 3 = Wait for new security level and new password (Command 3). ‘100’: State 4 = Protected registers are unlocked. Write access to one register is possible (only in low protected mode). ‘101’: Reserved ‘110’: Reserved ‘111’: Reserved

The following registers are defined as protected (security) registers:

- SYSCON1
- SYSCON2
- SYSCON3

The following state diagram shows the state machine for security level switching and for unlock command execution in low protected mode:

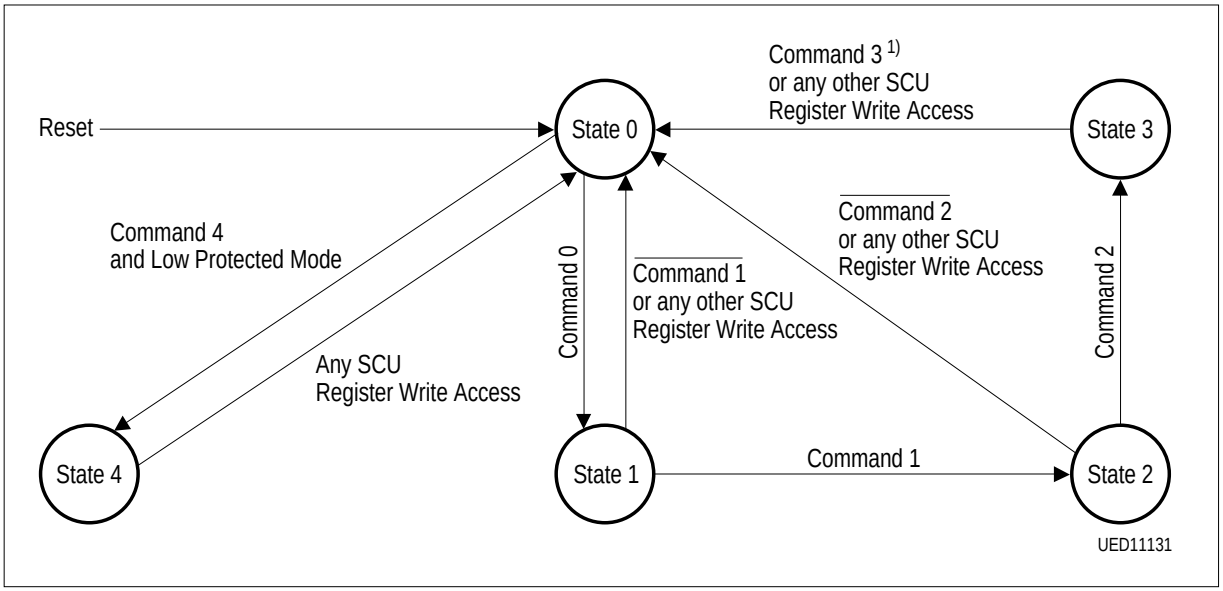


Figure 6-1 State Machine for Security Level Switching

Note: 1) The new security level and password are valid, only if the security level command register is accessed.

c_SCU is an abbreviation for "*write access to any SCU register*", i.e. write access to any of the following registers:

SYSCON1, SYSCON2, SCUSLC, SCUSLS, RP0H, XPERCON, WDT, WDTCON, EXICON, EXISEL, ISNC, FOCON, SYSCON3

Note: The FOCON and SYSCON3 registers do not provide any function within the M2. Nevertheless they are implemented in the system control unit of the embedded microcontroller and write access to the associated addresses will influence the state machine for security level switching.

Write Access in Low Protected Mode

The write access in low protected mode is also done via a command sequence. First the specific command 4 (see **Figure 6-1**) has to be written to register SCUSLC with the current password. After this command, all security registers are unlocked until the next write access to any SCU register is done. Read access is always possible to all registers of the SCU and will not influence the command sequences. In register SCUSCS the actual status of the command state machine can always be read.

It is recommended to use an atomic sequence for all command sequences.

6.4 Power Reduction Modes

Three power reduction modes with different levels of power reduction, which may be entered under software control, have been implemented in M2:

Idle Mode: The CPU is stopped, while the peripherals including watchdog timer continue their operation at low clock frequency.

Sleep Mode: CPU, peripherals and PLL are completely turned off. The controller ADC operates in wake-up mode. The real-time-clock is still in operation.

Power Down Mode: All modules are turned off.

Mode	Normal	Idle	Sleep	Power Down
Oscillator	on	on	on	off
PLL	on	on	off	off
CPU	33 MHz	3 MHz/ stopped	no clock/ stopped	no clock/ stopped
CADC	33 MHz[1)]	3 MHz[2)]	3 MHz[2)]	off
RTC	3 MHz	3 MHz	3 MHz	off
Peripherals	33 MHz	3 MHz	off	off
Bus Interface	100/66 MHz[3)]	off	off	off

1) Conversion mode

2) Wake up mode

3) Depending on value of CLKCON (see **Chapter 8**)

In the table above clocking frequencies have been specified, indicating that power reduction is achieved by means of clock-gating. None of the power supplies are internally switched, neither may voltage be turned off at the supply pins.

M2's power management functions are supplemented by a Real Time Clock (RTC) timer with optional periodic wake-up from idle mode. The periodic wake-up combines the reduced power consumption in power reduction modes with a high level of system availability. External signals and events can be scanned (at a lower rate) by periodically activating the CPU and selected peripherals which then return to power-save mode after a short time. This greatly reduces the system's average power consumption.

Entering and Terminating Idle and Sleep Mode

All three modes are entered by writing register SLEEPCON (see register definition below) and issuing the IDLE instruction. All external bus actions are completed before entry to Idle, Sleep or Power Down mode. Normal operation is resumed after Idle mode upon any interrupt request. To return from Sleep mode, external, wake up- or RTC-

interrupt requests can be used. Power down mode can only be terminated with hardware reset.

To prevent unintentional entry into Idle mode, the IDLE instruction has been implemented as a protected 32-bit instruction.

Mode	Idle	Sleep	Power Down
Entry by …	writing SLEEPCON and issuing IDLE instruction		
Exit by …	any interrupt request	external, wake up- or RTC-IRQ	HW-reset

Idle mode is terminated by interrupt requests from any enabled interrupt source whose individual Interrupt Enable flag was set before the Idle mode was entered, regardless of bit IEN.

For a request selected for CPU interrupt service, the associated interrupt service routine is entered if the priority level of the requesting source is higher than the current CPU priority and the interrupt system is globally enabled. After the RETI (Return from Interrupt) instruction of the interrupt service routine is executed the CPU continues executing the program with the instruction following the IDLE instruction. Otherwise, if the interrupt request cannot be serviced because of a too low priority or a globally disabled interrupt system the CPU immediately resumes normal program execution with the instruction following the IDLE instruction.

For a request which was programmed for PEC service, a PEC data transfer is performed if the priority level of this request is higher than the current CPU priority and the interrupt system is globally enabled. After the PEC data transfer has been completed the CPU remains in Idle mode. Otherwise, if the PEC request cannot be serviced because of a too low priority or a globally disabled interrupt system, the CPU does not remain in Idle mode but continues program execution with the instruction following the IDLE instruction.

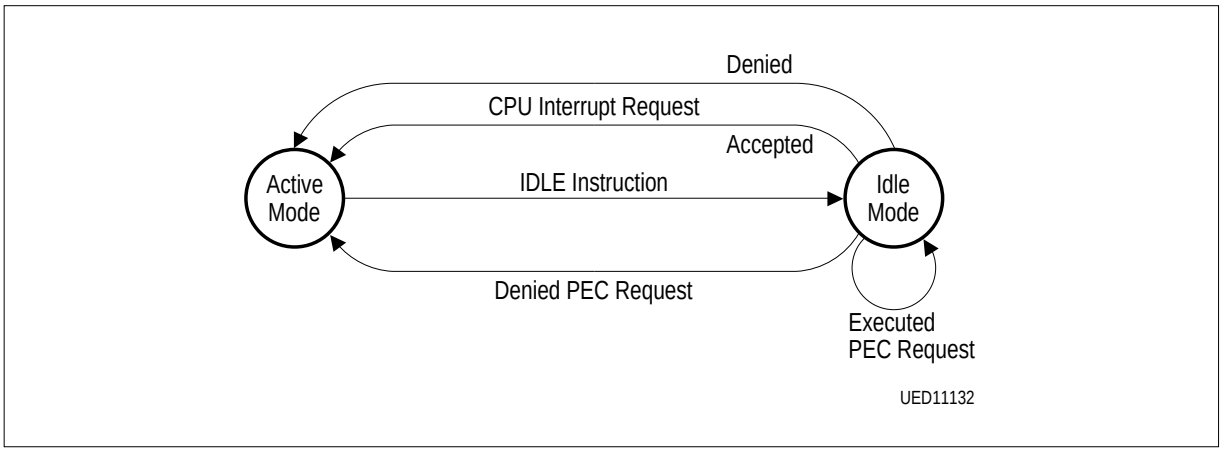


Figure 6-2 Transitions between Idle Mode and Active Mode

Any interrupt request, whose individual Interrupt Enable flag was set before Idle mode was entered, will terminate Idle mode regardless of the current CPU priority. The CPU will **not** go back into Idle mode when a CPU interrupt request is detected, even when the interrupt was not serviced because of a higher CPU priority or a globally disabled interrupt system (IEN = '0'). The CPU will **only** go back into Idle mode when the interrupt system is globally enabled (IEN = '1') **and** a PEC service on a priority level higher than the current CPU level is requested and executed.

Note: An interrupt request which is individually enabled and assigned to priority level 0 will terminate Idle mode. However, the associated interrupt vector will not be accessed.

The watchdog timer may be used to monitor the Idle mode: an internal reset will be generated if no interrupt request occurs before the watchdog timer overflows. To prevent the watchdog timer from overflowing during Idle mode it must be programmed to a reasonable time interval before Idle mode is entered.

Power Down Mode

Clocking of all internal blocks is stopped in Power Down Mode, the contents of the internal RAMs, however, are preserved through the voltage supplied via the V_{DD} pins. The watchdog timer is stopped in Power Down Mode. This mode can only be terminated by an external hardware reset, e.g. by asserting a low level on the $\overline{\text{RSTIN}}$ pin. This reset will initialize all SFRs and ports to their default state, but will not change the contents of the internal RAMs.

SDRAM Refreshing

Before entering into one of the power save modes the external SDRAM must be put into self-refresh-mode by use of register EBIDIR (see **Chapter 4.5**).

Status of Output Pins during Idle and Power Down Mode

During Idle mode the CPU is stopped, while all peripherals continue their operation in the same way previously described. Therefore all ports pins, which are configured as general purpose output pins, output the last data value which was written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral.

Port 4 outputs the segment address for the last access on the pins that were selected during reset, otherwise the output pins of Port 4 represent the port latch data.

During Power Down mode the oscillator and the clocks to the CPU and peripherals are turned off. Like in idle mode, all port pins, which are configured as general purpose output pins, output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

SYSCON1 **Reset Value: 0001_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	SLEEPCON	
														rw	

Bit	Function
SLEEPCON	**Power Save Mode Selection** 00: Idle Mode 01: Sleep Mode 10: reserved 11: Power Down Mode

Note: This register is a protected register; it's security level is automatically set to full write protection after execution of the EINIT instruction.

6.5 Dedicated Pins

M2 has different dedicated Pins than other controllers of C16X family. The following dedicated pins are **not** available: $\overline{\text{ALE}}$, $\overline{\text{READY}}$, $\overline{\text{EA}}$, $\overline{\text{NMI}}$. The following table explains M2 specific dedicated pins.

The external read strobe $\overline{\text{RD}}$ controls the output drivers of external memory or peripherals when M2 reads data from these external devices. During reset an internal pull-up ensures an inactive (high) level on the $\overline{\text{RD}}$ output.

Pin(s)	Function
$\overline{RD}$	External Read Strobe
$\overline{WR}$	Write Enable Strobe for SDRAM
LDQM, UDQM	Byte Mask Signals for SDRAM
XTAL1, XTAL2	Oscillator Input/Output
RSTIN	Reset Input Pin
$\overline{CSDRAM}$, $\overline{CSROM}$, $\overline{CS3}$	Chip Select Signals
MEMCLK, CLKEN	Clock Signals for SDRAM
CVBS1, CVBS2	CVBS Input Signals
R, G, B	Analog RGB Output
CORBLA	Contrast Reduction and Blanking Pin
HSYNC, VSYNC	Sync Inputs/Outputs for the Display

The external write strobe $\overline{WR}$ controls the data transfer from the M2 to an external memory. During reset an internal pull-up ensures an inactive (high) level on the $\overline{WR}$ output.

Byte mask signals **LDQM** and **UDQM** control the byte access to an external SDRAM according to the PC100 specification. These pins are active if either the high byte or the low byte of a 16-bit word are written.

The oscillator input XTAL1 and output XTAL2 connect the internal oscillator to the external crystal. The oscillator provides an inverter and a feedback element. An external TTL clock signal may be fed to the input XTAL1, leaving XTAL2 open.

By using the **$\overline{RSTIN}$** pin M2 can be put into the well defined reset condition either at power-up or upon external events like a hardware failure or manual reset. The internal reset signal can be driven on output pin **$\overline{RSTOUT}$/COR**.

The chipselect signals $\overline{CSDRAM}$, $\overline{CSROM}$, $\overline{CS3}$ are for general control of external memories. During reset an internal pull-up ensures an inactive (high) level on these outputs. $\overline{CS3}$ can be used for a ROM.

Signals MEMCLK, CLKEN are used to provide a clock and an enable signal for an external SDRAM. During reset an internal pull-down ensures an inactive (low) level on these outputs.

CVBS1A, CVBS1B and **CVBS2** can carry two analog composite video signals and act as inputs for the two data slicers. The video signal on channel 1 can be either differential (CVBS1A and CVBS1B) or single-ended (CVBS1A, CVBS1B to ground).

R, G, B are analog outputs from the display generator.

CORBLA is a signal which indicates whether a pixel created by M2 should be displayed or mixed with external video source (blanking function). At the same time this signal

carries information on contrast reduction of this pixel. Alternatively **BLANK** can be generated as a separate output signal. **COR** can be generated separately as well, in which case no $\overline{\text{RSTOUT}}$ is available.

HSYNC and **VSYNC** are bidirectional pins which are used to synchronize M2 to an external video source or to deliver a stable horizontal and vertical sync timing to external components.

6.6 XBUS Configuration

Although the XBUS is not visible at the chip boundary, some registers have to be set to guarantee correct operation. The user has to program the XBUS-registers in the following way:

Register	Value
SYSCON	$E444_H$
XADRS1	$0E03_H$
XADRS2	$0E83_H$
XADRS3 - XADRS6 (not used)	0000_H
ADDRSEL1 - ADDRSEL4 (not used)	0000_H
XBCON1	$05BF_H$
XBCON2	$05BF_H$
XBCON3 - XBCON6 (not used)	0000_H
BUSCON0	$15B7_H$
BUSCON1 - BUSCON4 (not used)	0000_H
XPERCON	0003_H

6.7 Watchdog Timer

The watchdog timer is a 16-bit up counter which can be clocked with the CPU clock (f_{CPU}), either divided by 2 or divided by 128. This 16-bit timer is realized as two concatenated 8-bit timers (see **Figure 6-3**). The upper 8 bits of the watchdog timer can be preset to a user-programmable value via a watchdog service access in order to vary the watchdog expire time. The lower 8 bits are reset on each service access.

The following figure shows the WDT block diagram:

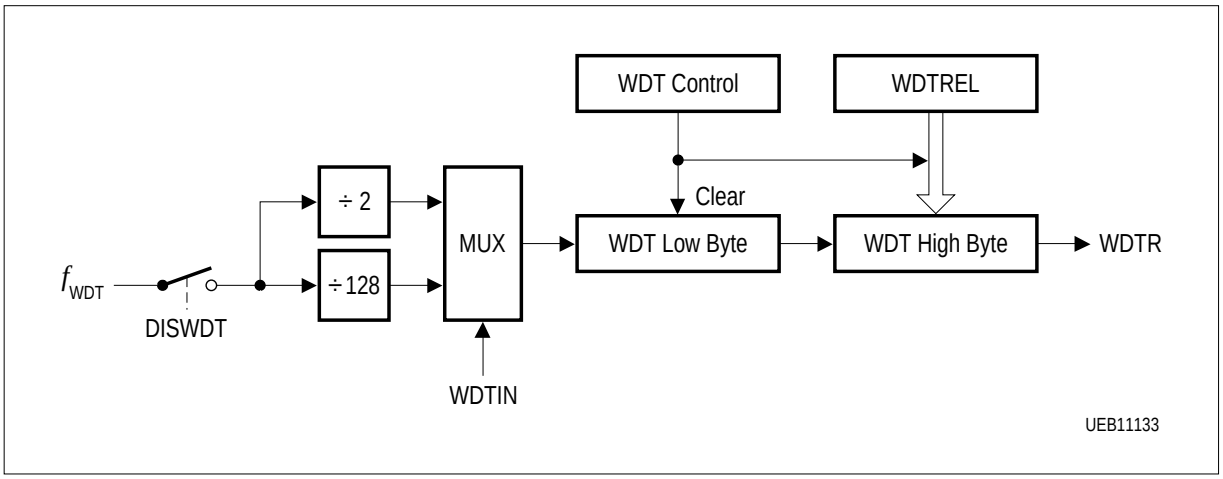


Figure 6-3 WDT Block Diagram

The current count value of the Watchdog Timer is contained in the Watchdog Timer Register WDT, which is a non-bitaddressable read-only register. The operation of the Watchdog Timer is controlled by its bitaddressable Watchdog Timer Control Register WDTCON. This register specifies the reload value for the high byte of the timer and selects the input clock prescaling factor.

After any software reset, external hardware reset (see note), or watchdog timer reset, the watchdog timer is enabled and starts counting up from 0000_H with the frequency $f_{CPU}/2$. The input frequency may be switched to $f_{CPU}/128$ by setting bit WDTIN. The watchdog timer can be disabled via the instruction DISWDT (Disable Watchdog Timer). Instruction DISWDT is a protected 32-bit instruction which will ONLY be executed during the time between a reset and execution of either the EINIT (End of Initialization) or the SRVWDT (Service Watchdog Timer) instruction. Either one of these instructions disables the execution of DISWDT.

When the watchdog timer is not disabled via instruction DISWDT it will continue counting up, even during Idle Mode. If it is not serviced by the SRVWDT instruction by the time the count reaches $FFFF_H$ the watchdog timer will overflow and cause an internal reset. In this case the Watchdog Timer Reset Indication Flag (WDTR) in register WDTCON will be set.

To prevent the watchdog timer from overflowing, it must be serviced periodically by the user software. The watchdog timer is serviced with the instruction SRVWDT, which is a protected 32-bit instruction. Servicing the watchdog timer clears the low byte and reloads the high byte of the watchdog time WDT register with the preset value in bit field WDTREL, which is the high byte of the WDTCON register. Servicing the watchdog timer will also reset the WDTR bit. After being serviced the watchdog timer continues counting up from the value (<WDTREL> $\times$ 28). Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer (e.g. by fetching and executing a bit pattern from a wrong location) is minimized. When instruction

SRVWDT does not match the format for protected instructions, the Protection Fault Trap will be entered, rather than the instruction being executed.

The time period for an overflow of the watchdog timer is programmable in two ways:

- **The input frequency** of the watchdog timer can be selected via bit WDTIN in register WDTCON
  to be either $f_{CPU}/2$ or $f_{CPU}/128$.
- **The reload value** WDTREL for the high byte of WDT can be programmed in register WDTCON.

The period P_{WDT} between servicing the watchdog timer and the next overflow can therefore be determined by the following formula:

$$P_{WDT} = (2^{(1 + <WDTIN> \times 6)} \times (2^{16} - <WDTREL> \times 2^{8}))/f_{CPU} \quad [1]$$

Table 6-1 marks the possible ranges for the watchdog time which can be achieved using a certain CPU clock. Some numbers are rounded to 3 significant digits.

Table 6-1 Watchdog Time Ranges

Reload Value in WDTREL	Prescaler for f_{CPU}			
	2 (WDTIN = '0')		128 (WDTIN = '1')	
	33.33 MHz	3 MHz	33.33 MHz	3 MHz
FF_H	25.6 µs	32.0 µs	1.64 ms	2.05 ms
$7F_H$	3.3 ms	4.13 ms	211 ms	264 ms
00_H	6.55 ms	8.19 ms	419 ms	524 ms

Note: For safety reasons, the user is advised to rewrite WDTCON each time before the watchdog timer is serviced.

Here is the description of the Watchdog timer SFRs.

WDTCON **Reset Value: $00XX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL(7 .. 0)								-	-	-	LHW R	SHW R	SW R	WDT R	WDT IN
rw											r	r	r	r	rw

Bit	Function
WDTIN	**Watchdog Timer Input Frequency Selection** 0: Input frequency is $f_{CPU}/2$ 1: Input frequency is $f_{CPU}/128$
WDTR	**Watchdog Timer Reset Indication Flag** Cleared by a hardware reset or by the SRVWDT instruction.
SWR	**Software Reset Indication Flag**
SHWR	**Short Hardware Reset Indication Flag**
LHWR	**Long Hardware Reset Indication Flag**
WDTREL (7 … 0)	**Watchdog Timer Reload Value** (for the high byte of WDT)

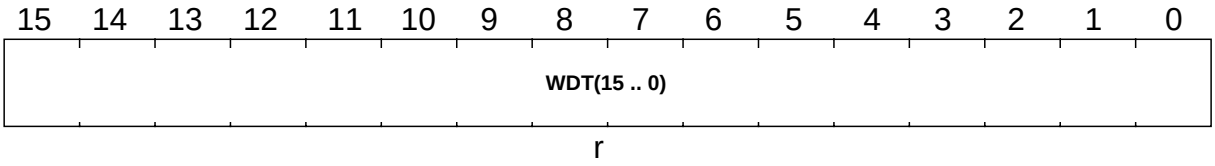
Note: The reset value depends on the reset source (see description below). The execution of EINIT clears the reset indication flags.

Note: When the reset output is enabled pin $\overline{RSTOUT}$ will be pulled low for the duration of the internal reset sequence upon a watchdog timer reset.

WDT Timer Register

The WDT register contains the current count value of the Watchdog Timer.

WDT **Reset Value: 0000_H**



Note: This register in a read-only register. Write access can be performed to this register, during test mode only.

Reset Source Indication

The reset indication flags in register WDTCON provide information on the source of the last reset. As M2 starts executing from location $00'0000_H$ after any possible reset event, the initialization software may check these flags in order to determine if the recent reset event was triggered by an external hardware signal (via $\overline{RSTIN}$), by software itself or by an overflow of the watchdog timer. The initialization (and also the further operation) of the microcontroller system can thus be adapted to the respective circumstances, e.g. a special routine may verify the software integrity after a watchdog timer reset.

The reset indication flags are not mutually exclusive, e.g. more than one flag may be set after reset depending on its source. The table below summarizes the possible combinations:

Table 6-2 Reset Indication Flag Combinations

	Reset Indication Flags			
Reset Source	**LHWR**	**SHWR**	**SWR**	**WDTR**
Long Hardware Reset	X	X	X	–
Short Hardware Reset	*	X	X	–
Software Reset	*	*	X	–
Watchdog Timer Reset	*	*	X	X

*Note: *) When the reset output is enabled, the indicated flags are also set in the respective reset case. The WDTCON reset value will then be different from the table value.*

Note: The listed reset values for WDTCON assume the reserved bits as '0'.

Long Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is still sampled low (active) at the end of a hardware triggered internal reset sequence.

Short Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is sampled high (inactive) at the end of a hardware triggered internal reset sequence.

Software Reset is indicated after a reset triggered by the execution of instruction SRST.

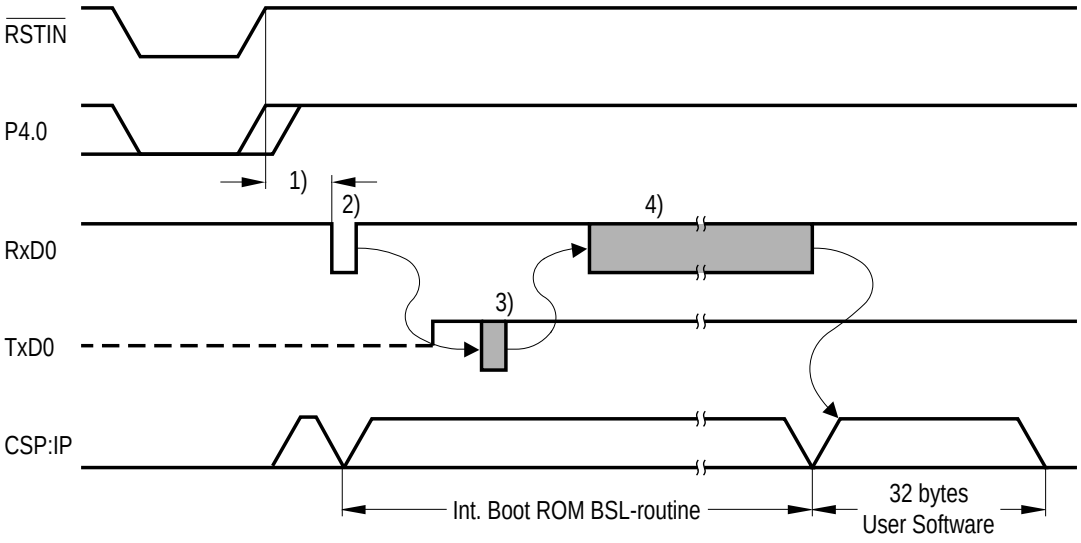
Watchdog Timer Reset is indicated after a reset triggered by an overflow of the watchdog timer.

Note: When reset output is enabled the $\overline{\text{RSTOUT}}$ pin is pulled low for the duration of the internal reset sequence upon any sort of reset.
Therefore a long hardware reset (LHWR) will be recognized in any case.

6.8 Bootstrap Loader

The bootstrap loader of M2 works in the same way as implemented in other C16x derivatives. It provides a mechanism to load the start-up program, which is executed after reset, via a serial interface (ASC). In this case no external (ROM) memory is required.

The bootstrap loader moves code/data into the internal RAM, but it is also possible to transfer data via the serial interface into an external RAM using a second level loader routine.



1) BSL initialization time, > 1.5 µs @ f_{CPU} = 33 MHz
2) Zero byte (1 start bit, eight '0' data bits, 1 stop bit), sent by host
3) Identification byte ($D5_H$) sent by M2
4) 32 bytes of code/data, sent by host

UET11134

Figure 6-4 Bootstrap Loader Sequence

The M2 enters BSL mode, if pin P4.0 is sampled low at the end of a hardware reset. When M2 has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked***):

Watchdog Timer:	***Disabled***
Register SYSCON:	$0C00_H$
Context Pointer CP:	$FA00_H$
Register STKUN:	$FA40_H$
Stack Pointer SP:	$FA40_H$
Register STKOV:	$FA0C_H$ 0 <-> C
Register S0CON:	***8001_H***
Register BUSCON0:	according to start-up config.
Register S0BG:	according to '00' byte
P3.10/TXD0:	'***1***'
DP3.10:	'***1***'

Other than after a normal reset the watchdog timer is disabled, therefore the bootstrap loading sequence is not time limited. Pin TXD0 is configured as output. The configuration (e.g. the accessibility) of the M2's memory areas after reset in Bootstrap-Loader mode differs from the standard case. Accesses to the external ROM area are partly redirected,

while the M2 is in BSL mode. All code fetches to segment 0 are made from the special Boot-ROM, while data accesses read from the external user ROM.

6.9 Identification Registers

A set of 8 identification registers are provided to offer information on the chip as manufacturer, chip type and its memory (EEPROM, OTP, DRAM or Flash memory) properties, and information on the CSCU (type of module, redesign state).

The ID registers are read only registers. These registers are:

- IDMANUF, for manufacturer and department identification,
- IDCHIP, for device identification and revision code,
- IDMEM, for identification of on-chip program memory (type, size),
- IDMEM2, for identification of additional EEPROM, OTP, DRAM or Flash memory,
- IDPROG, for identification of programming/erasing voltage of on-chip program memory,
- IDRT, redesign tracking register for identification of revisions and laser cuts.

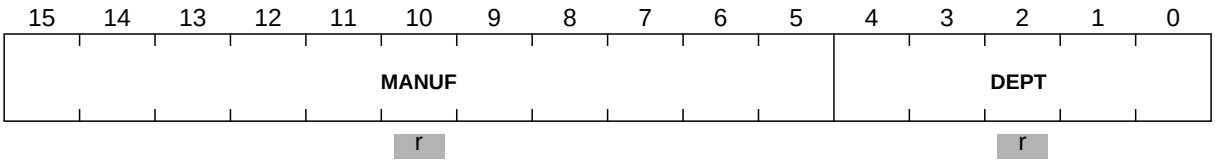
Besides, some other registers provide identification of some parts of the M2. These registers are hardwired inside the M2.

- CPUID, for CPU identification,
- IDSCU, for CSCU identification,
- DIPX version bit field, for OCDS identification.

6.9.1 System Identification

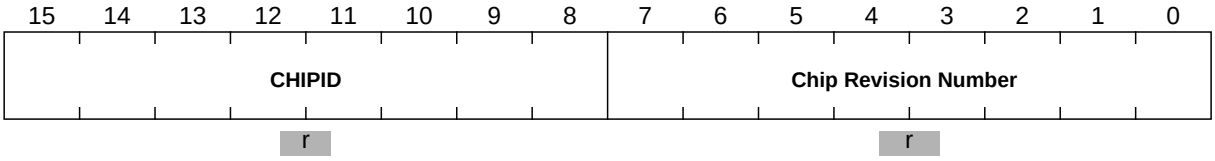
These identific ID register description

IDMANUF **Reset Value: $XXXX_H$**



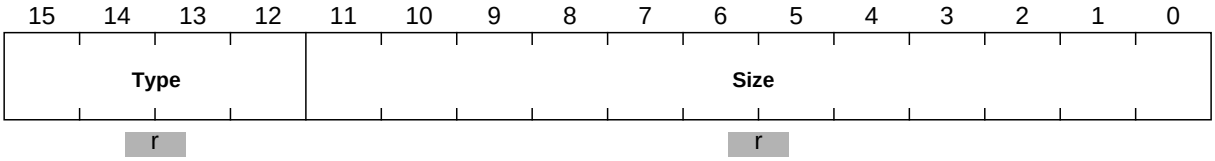
Bit	Function
MANUF	**Manufacturer** This is the JEDEC normalized manufacturer code. $0C1_H$: Infineon Technologies 020_H: SGS-Thomson
DEPT	**Department** Indicates the department within Infineon Technologies. 00_H: HL MC 01_H: HL CAD Macrocells 02_H: HL IT

IDCHIP **Reset Value: $XXXX_H$**

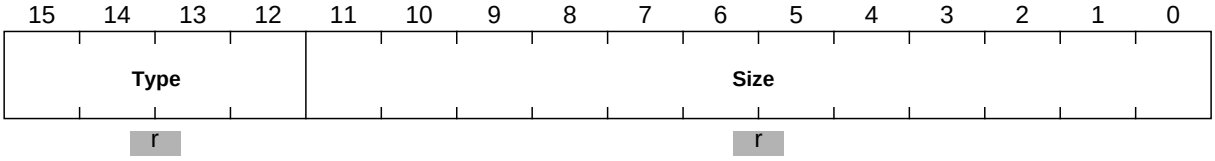


Bit	Function
Revision	**Device Revision Code** Identifies the device step where the first release is marked '01_H'.
CHIPID	**Device Identification** Identifies the device name. Please refer to the association table.

IDMEM **Reset Value: $XXXX_H$**



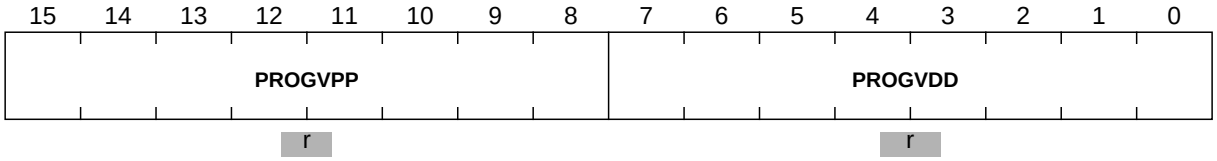
IDMEM2 **Reset Value: $XXXX_H$**



Note: IDMEM2 describes the second block of (program) memory. E.g. a device may contain Flash and EEPROM or DRAM sections. Static RAM modules are not described with ID registers.

Bit	Function
Size	**Size of On-chip Program Memory** The size of the implemented program memory in terms of 4 K blocks, i.e. Memory-size = <Size> × 4 KByte.
Type	**Type of On-chip Program Memory** Identifies the memory type on this silicon. 0_H: ROMless 1_H: Mask ROM 2_H: EPROM 3_H: Flash 4_H: OTP 5_H: EEPROM 6_H: DRAM/SRAM

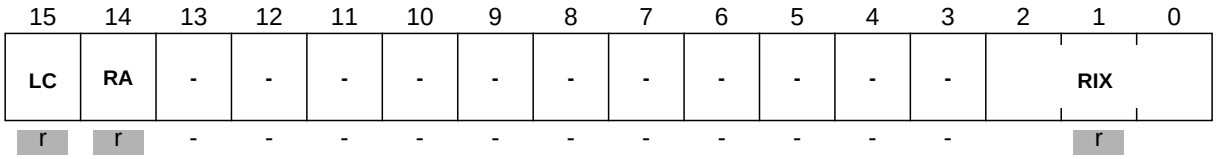
IDPROG **Reset Value: $XXXX_H$**



Bit	Function
PROGVDD	**Programming V_{DD} Voltage[1)]** The voltage of the standard power supply pins required when programming or erasing (if applicable) the on-chip program memory. Formula: V_{DD} = 20 × <PROGVDD>/256 [V]
PROGVPP	**Programming V_{PP} Voltage[1)]** The voltage of the special programming power supply (if existent) required to program or erase (if applicable) the on-chip program memory. Formula: V_{PP} = 20 × <PROGVPP>/256 [V]

Note: [1)] Devices that incorporate memories which cannot be programmed outside the factory will indicate '00_H' in both bit fields.

IDRT **Reset Value: $XXXX_H$**



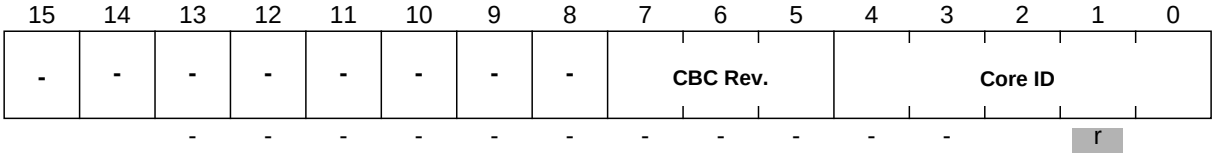
Note: The Redesign Tracing register IDRT is not commonly specified. It is protected against standard read accesses via the function testmode (Testmode A).

Bit	Function
RIX	**Redesign Index** 0: This device is the original “Revision”. else: This device has experienced minor changes that are not reflected to the customer by the “Revision” bit field.
RA	**Redundancy Activation** 0: This device is as it was manufactured. 1: Redundant memory structures have been activated.
LC	**Laser Correction** 0: This device is as it was manufactured. 1: This device has been laser corrected.

6.9.2 CPU Identification

The CPU is identified by the CPUID register. This register is only implemented on STARLIB versions of the M2 Rev. 2.0.

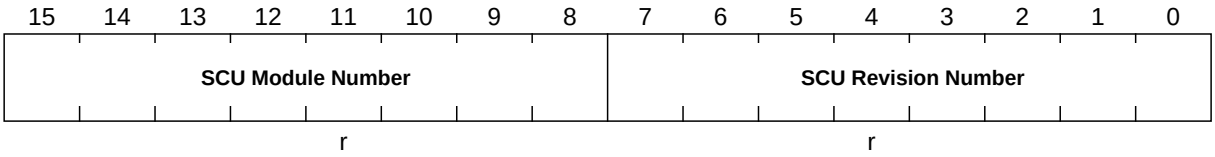
CPUID **Reset Value: 0061_H**



Bit	Function
CBC Rev.	CBC Revision number currently 011_B
Core ID	Core Identification number for the C166 CBC the value is 00001_B

The SCU provides a specific read-only identification register for its own module type and revision identification.

IDSCU **Reset Value: 0000_H**



For Rev. 2.0 derivatives, the current value is 0000_H.

6.10 Parallel Ports

M2 provides up to 30 input/outputs, 6 output and 6 input multiple purpose ports. All port lines are bit-addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of two I/O ports can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. During the internal reset, all port pins are configured as inputs. Most of the port lines have programmable alternate input or output functions associated with them (GPT1/GPT2, external interrupts, I^2C-bus, analog inputs for A/D converter, SSC-interface or ASC-interface). All port lines that are not used for these alternate functions may be used as general purpose I/O lines. Ports of M2 are 3.3 V tolerant and can deliver a 3.3 V output voltage (refer also to **Chapter 14**).

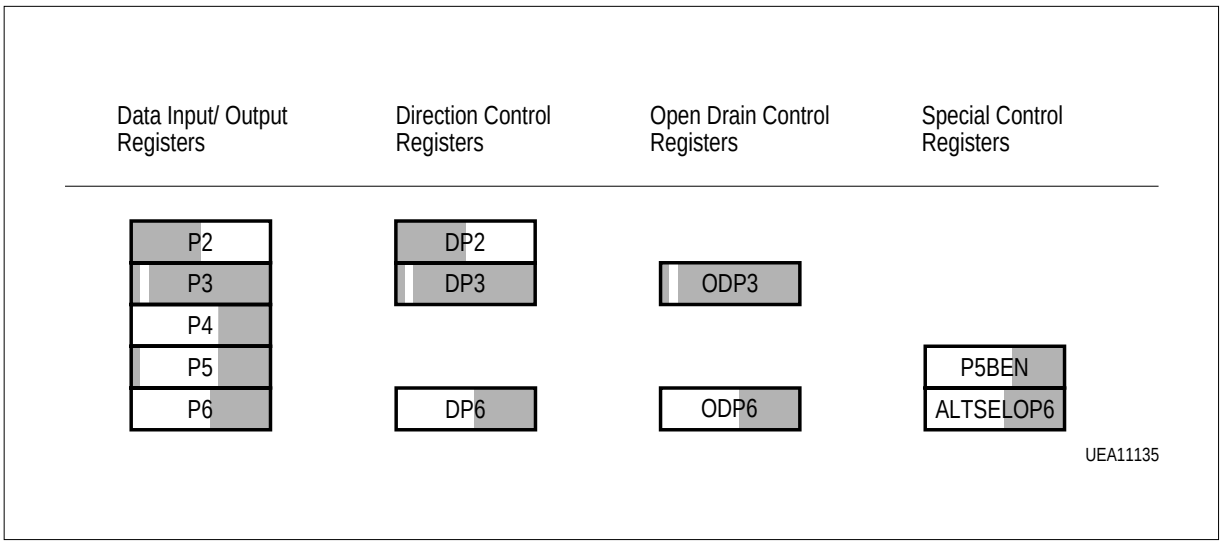


Figure 6-5 Portlogic Register Overview

Port 0

Port 0 does not exist due to the dedicated memory bus structure of M2.

Port 1

Port 1 does not exist due to the dedicated memory bus structure of M2.

Port 2

Port 2 is an 8-bit bidirectional I/O port which can also serve as fast external interrupt input (sample rate is 30 ns).

P2 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P2.15	P2.14	P2.13	P2.12	P2.11	P2.10	P2.9	P2.8	-	-	-	-	-	-	-	-
rw	rw	rw	rw	rw	rw	rw	rw								

Bit	Function
P2.y	Port data register P2 bit y.

DP2 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DP2.15	DP2.14	DP2.13	DP2.12	DP2.11	DP2.10	DP2.9	DP2.8	-	-	-	-	-	-	-	-
rw	rw	rw	rw	rw	rw	rw	rw								

Bit	Function
DP2.y	**Port direction register DP2 bit y** DP2.y = 0: Port line P2.y is an input (high-impedance). DP2.y = 1: Port line P2.y is an output.

Port 3

If this 15-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP3. All port lines can be switched into push/pull or open drain mode via the open drain control register ODP3. Due to pin limitations register bit P3.14 is not connected to any pin.

P3 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P3.15	-	P3.13	P3.12	P3.11	P3.10	P3.9	P3.8	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P3.y	**Port data register P3 bit y.**

Note: Register bit P3.14 is not connected to an I/O pin.

DP3 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DP3.15	-	DP3.13	DP3.12	DP3.11	DP3.10	DP3.9	DP3.8	DP3.7	DP3.6	DP3.5	DP3.4	DP3.3	DP3.2	DP3.1	DP3.0
rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP3.y	**Port direction register DP3 bit y** DP3.y = 0: Port line P3.y is an input (high-impedance). DP3.y = 1: Port line P3.y is an output.

ODP3 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODP3.15	-	ODP3.13	ODP3.12	ODP3.11	ODP3.10	ODP3.9	ODP3.8	ODP3.7	ODP3.6	ODP3.5	ODP3.4	ODP3.3	ODP3.2	ODP3.1	ODP3.0
rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
ODP3.y	**Port 3 Open Drain control register bit y** ODP3.y = 0: Port line P3.y output driver in push/pull mode. ODP3.y = 1: Port line P3.y output driver in open drain mode.

Alternate Functions of Port 3

The pins of Port 3 serve various functions which include external timer control lines and the 3 serial interfaces.

The table below summarizes the alternate functions of Port 3.

Port 3 Pin	Alternate Function	
P3.0	SCL0	I^2C - bus clock line 0
P3.1	SDA0	I^2C - bus data line 0
P3.2	CAPIN	GPT2 capture input
P3.3	T3OUT	Timer 3 Toggle output
P3.4	T3EUD	Timer 3 External Up/Down input
P3.5	T4IN	Timer 4 Count input
P3.6	T3IN	Timer 3 Count input
P3.7	T2IN	Timer 2 Count input
P3.8	MRST	SSC Master receive/Slave transmit
P3.9	MTSR	SSC Master transmit/Slave receive
P3.10	TxD0	ASC0 Transmit data output
P3.11	RxD0	ASC0 Receive data input
P3.12	---	No alternate function
P3.13	SCLK	SSC Shift Clock input/output
P3.14	---	Not implemented. No pin assigned!
P3.15	LED2	No alternate function

Port 4

Port 4 is a 6-bit output port. Because of its high frequency requirements in the alternate function mode, it has different electrical characteristics than the other ports.

During reset, the user specific portion of the system start-up configuration is input via Port 4. The complete configuration (user specific as well as hardwired settings) can be read at runtime from register RP0H. For a detailed description refer to **Chapter 6.1**.

P4L (during reset) **Reset Value: $XXXX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	P4L.5	P4L.4	P4L.3	P4L.2	P4L.1	P4L.0
										r	r	r	r	r	r

Bit	Function
P4L.0	**BSLENA (Boot Strap Load Enable)** P4L.0 = 1: Boot strap loader enabled P4L.0 = 0: Boot strap loader disabled
P4L.1	**CSENA (Chip Select Enable)** P4L.1 = 1: P4.5 configured as general purpose pin P4L.1 = 0: P4.5 configured as $\overline{CS3}$
P4L.2	**Reserved**
P4L.(5 … 3)	**SALSEL(2 … 0) (Select Number of Address Lines)** see explanation below

P4 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
										rw	rw	rw	rw	rw	rw

Bit	Function
P4.y	Port data register P4 bit.y

Alternate Functions of Port 4

During external bus cycles that use segmentation (e.g. an address space above 128 KByte), a number of Port 4 pins may output the segment address lines and activate the third chip select signal (A(20 … 16) and $\overline{CS3}$). The number of pins that is used for segment address output determines the external address space which is directly accessible. The other pins of Port 4 (if any) may be used for general purpose output. If segment address lines are selected, the alternate function of Port 4 may be necessary to access e.g. external memory directly after reset. For this reason Port 4 will be automatically switched to this alternate function.

The number of segment address lines is selected via P4 during reset. The selected value can be read from bit field SALSEL and CSENA of register RP0H e.g. in order to check the configuration during run time.

Port 4 Pin	Altern. Function SALSEL = 111	Altern. Function SALSEL = 110	Altern. Function SALSEL = 101	Altern. Function SALSEL = 100	Altern. Function SALSEL = 011	SALSEL = 010 or 001 or 000
P4.0	A16	A16	A16	A16	A16	Gen. Purp. I/O
P4.1	A17	A17	A17	A17	Gen. Purp. I/O	Gen. Purp. I/O
P4.2	A18	A18	A18	Gen. Purp. I/O	Gen. Purp. I/O	Gen. Purp. I/O
P4.3	A19	A19	Gen. Purp. I/O	Gen. Purp. I/O	Gen. Purp. I/O	Gen. Purp. I/O
P4.4	A20	Gen. Purp. I/O	Gen. Purp. I/O	Gen. Purp. I/O	Gen. Purp. I/O	Gen. Purp. I/O

Port 4 Pin	Altern. Function CSENA = 0	Altern. Function CSENA = 1
P4.5	$\overline{CS3}$	Gen. Purp. I/O

Port 5

Port 5 is a 6-bit input port.

P5 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P5.15	P5.14	-	-	-	-	-	-	-	-	-	-	P5.3	P5.2	P5.1	P5.0
r	r											r	r	r	r

Bit	Function
P5.y	Port data register P5 bit y (read only).

Alternate Functions of Port 5

Port 5.3 … P5.0 is also connected to the input multiplexer of the A/D Converter. These lines can accept analog signals (AN3 … AN0) that can be converted by the ADC. Port 5.15 and 5.14 also serve as external timer control lines for GPT1 and GPT2.

Port 5 Pin	Alternate Function	
P5.0	ANA0	Analog Input 0 (Wake Up Function)
P5.1	ANA1	Analog Input 1
P5.2	ANA2	Analog Input 2
P5.3	ANA3	Analog Input 3
P5.14	T4EUD	Timer 4 External Up/Down Input
P5.15	T2EUD	Timer 2 External Up/Down Input

P5BEN **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	P5BEN.3	P5BEN.2	P5BEN.1	P5BEN.0
												rw	rw	rw	rw

Bit	Function
P5BEN.y	**Input Functionality Control Bit** P5BEN.y = 0: Analog Input Functionality. P5BEN.y = 1: Digital Input Functionality.

Port 6

If this 7-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP6. The port lines can be switched into push/pull or open drain mode via the open drain control register ODP6.

P6 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
									rw	rw	rw	rw	rw	rw	rw

Bit	Function
P6.y	Port data register P6 bit y.

DP6 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	DP6.6	DP6.5	DP6.4	DP6.3	DP6.2	DP6.1	DP6.0
									rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP6.y	**Port direction register DP6 bit y.** DP6.y = 0: Port line P6.y is an input (high-impedance). DP6.y = 1: Port line P6.y is an output.

ODP6 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	ODP6.6	ODP6.5	ODP6.4	ODP6.3	ODP6.2	ODP6.1	ODP6.0
									rw	rw	rw	rw	rw	rw	rw

Bit	Function
ODP6.y	**Port 6 Open Drain control register bit y** ODP6.y = 0: Port line P6.y output driver in push/pull mode ODP6.y = 1: Port line P6.y output driver in open drain mode

Alternate Functions of Port 6

The table below summarizes the alternate functions of Port 6.

Port 6 Pin	Alternate Function	
P6.0	TRIG_IN	Trigger Input for Emulation
P6.1	TRIG_OUT	Trigger output for Emulation
P6.2		no alternate function
P6.3	SCL1	I^2C Bus Clock Line 1
P6.4	SDA1	I^2C Bus Data Line 1
P6.5		no alternate function
P6.6	SDA2	I^2C Bus Data Line 2

ALTSEL0P6 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	SEL P6.6	SEL P6.5	SEL P6.4	SEL P6.3	SEL P6.2	SEL P6.1	SEL P6.0
									rw	rw	rw	rw	rw	rw	rw

Bit	Function
SELP6.y	**Alternate Function Control Bit** SELP6.y = 0: General Purpose Port Functionality enabled for Line P6.y. SELP6.y = 1: Alternate Function enabled for Line P6.y.

Peripherals

7 Peripherals

All of the peripherals described in the following paragraphs are clocked with the same clock as the CPU (f_{hw_clk}). Depending on the mode (normal or Idle), this frequency is 33.33 MHz or 3 MHz.

7.1 General Purpose Timer Unit

The General Purpose Timer Unit (GPT) represents very flexible multifunctional timer structures which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes. They incorporate five 16-bit timers that are grouped into the two timer blocks GPT1 and GPT2. Each timer in each block may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block.

Block 1 contains 3 timers/counters with a maximum resolution of $f_{hw\_clk}/4$. The auxiliary timers of GPT1 may be optionally configured as reload or capture registers for the core timer.

Block 2 contains 2 timers/counters with a maximum resolution of $f_{hw\_clk}/2$. An additional CAPREL register supports capture and reload operations with extended functionality, and its core timer T6 may be concatenated with the timers from the CAPCOM unit (T0 and T1).

The following enumeration summarizes all features to be supported:

Timer Block 1:

- $f_{hw\_clk}/4$ maximum resolution
- 3 independent timers/counters
- Timers/counters can be concatenated
- 4 operating modes (timer, gated timer, counter, incremental)
- Separate interrupt nodes

Timer Block 2:

- $f_{hw\_clk}/2$ maximum resolution
- 2 independent timers/counters
- Timers/counters can be concatenated
- 3 operating modes (timer, gated timer, counter)
- Extended capture/reload functions via 16-bit Capture/Reload register CAPREL
- Separate interrupt nodes

7.1.1 Functional Description of Timer Block 1

All three timers of block 1 (T2, T3, T4) can run in 4 basic modes, which are timer, gated timer, counter and incremental interface mode. All timers can either count up or down.

Each timer has an input line (TxIN) associated with it which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) may be programmed via software or dynamically altered by a signal at an external control input line (TxEUD). An overflow/underflow of core timer T3 is indicated by the output toggle latch T3OTL whose state may be output on related line T3OUT. The auxiliary timers T2 and T4 may additionally be concatenated with the core timer, or used as capture or reload registers for the core timer. Concatenation of T3 with other timers is provided through line T3OTL.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer registers T2, T3, or T4, which are located in the non-bitaddressable SFR space. When any of the timer registers are written to by the CPU in the state immediately before a timer increment, decrement, reload, or capture is to be performed, the CPU write operation has priority in order to guarantee correct results.

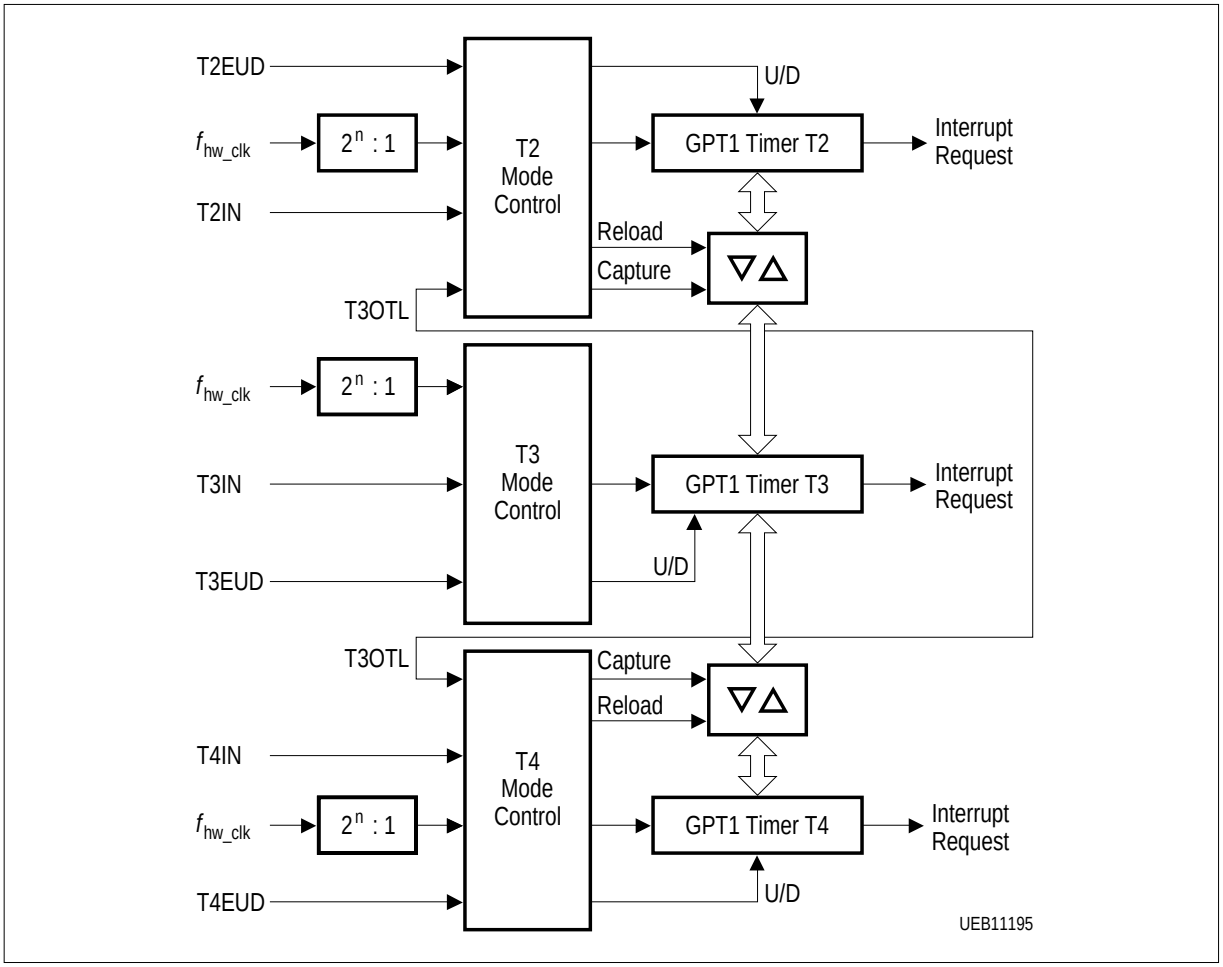


Figure 7-1 Structure of Timer Block 1 Core Timer T3

The operation of the core timer T3 is controlled by its bit-addressable control register T3CON.

Run Control

The timer can be started or stopped by software through bit T3R (Timer T3 Run Bit). Setting bit T3R will start the timer, clearing T3R stops the timer.

In gated timer mode, the timer will only run if T3R is set and the gate is active (high or low, as programmed).

Note: When bit T2RC/T4RC in timer control register T2CON/T4CON is set, T3R will also control (start and stop) auxiliary timer T2/T4.

Count Direction Control

The count direction of the core timer can either be controlled by software or by the external input line T3EUD (Timer T3 External Up/Down Control Input). These options are selected by bits T3UD and T3UDE in control register T3CON. When the up/down control is executed by software (bit T3UDE = ‘0’), the count direction can be altered by setting or clearing bit T3UD. When T3UDE = ‘1’, line T3EUD is selected to be the controlling source of the count direction. However, bit T3UD can still be used to reverse the actual count direction, as shown in the table below. If T3UD = ‘0’ and line T3EUD shows a low level, the timer is counting up. With a high level T3EUD, the timer is counting down. If T3UD = ‘1’, a high level at line T3EUD specifies counting up, and a low level specifies counting down. The count direction can be changed regardless of whether the timer is running or not.

When line T3EUD is used as external count direction control input, its associated port pin must be configured as input.

Table 7-1 Core Timer T3 Count Direction Control

Line T3EUD	Bit T3UDE	Bit T3UD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

Note: The direction control works in the same way for core timer T3 and for auxiliary timers T2 and T4. Therefore the lines and bits are named Tx …

Timer 3 Overflow/Underflow Monitoring

An overflow or underflow of timer T3 will clock the overflow toggle latch T3OTL in control register T3CON. T3OTL can also be set or reset by software. Bit T3OE (Overflow/

Underflow Output Enable) in register T3CON enables the state of T3OTL to be monitored via an external line T3OUT. If this line is linked to an external port pin, which has to be configured as output, T3OTL can be used to control external HW.

In addition, T3OTL can be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4. For this purpose, the state of T3OTL does not have to be available at any port pin, because an internal connection is provided for this option.

Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '000_B'.

In this mode, T3 is clocked with the system clock f_{hw_clk} divided by a programmable prescaler, which is controlled by bit field T3I and BPS1. The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower module clock frequencies, as can be seen from the following formula:

$$f_{T3} = \frac{f_{hw\_clk}}{BPS1 \times 2^{<T3I>}} \qquad r_{T3}\,[ms] = \frac{BPS1 \times 2^{<T3I>}}{f_{hw\_clk}\,[MHz]}$$

Table 7-2 gives an overview for timer resolutions depending on prescaler factors.

Table 7-2

f_{MOD} = 33.33 MHz	Timer Input Selection T2I / T3I / T4I								
	000_B	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
FM =	1	0	0	0	0	0	0	0	0
Prescaler factor	4	8	16	32	64	128	256	512	1024
Input Frequency	2.08 MHz	4.16 MHz	2.08 MHz	1.04 MHz	521.83 kHz	260.41 kHz	130.20 kHz	65.10 kHz	32.55 kHz
Resolution	120 ns	240 ns	480 ns	960 ns	1.92 µs	3.84 µs	7.68 µs	15.36 µs	30.72 µs
Period	7.86 ms	15.72 ms	31.45 ms	62.91 ms	125.82 ms	251.6 ms	503 ms	1 s	2.01 s

This formula also applies to the Gated Timer Mode of T3 and to the auxiliary timers T2 and T4 in timer and gated timer mode.

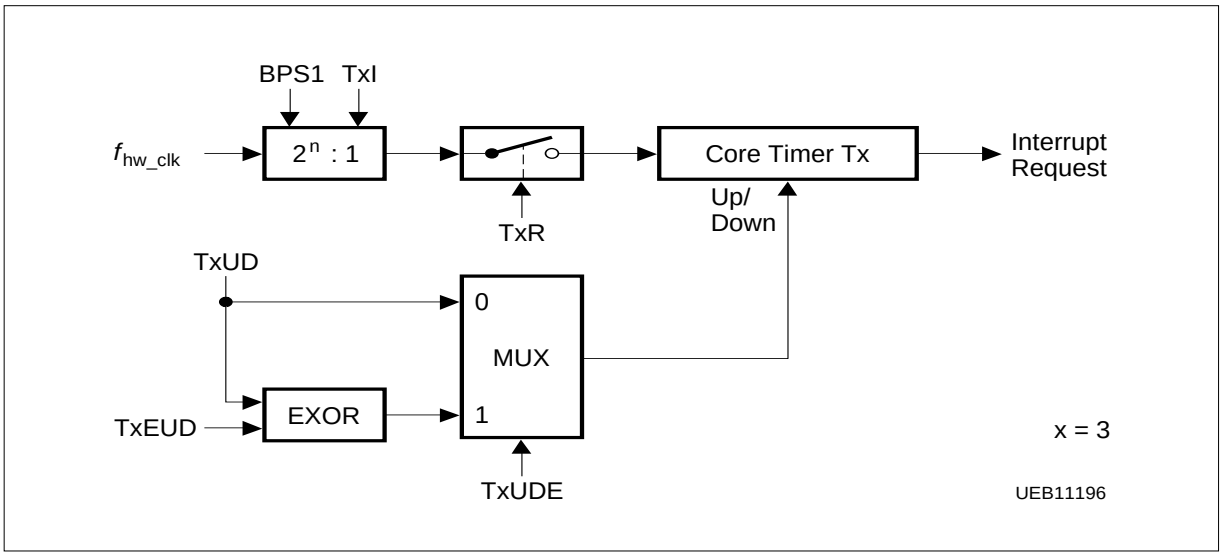


Figure 7-2 Block Diagram of Core Timer T3 in Timer Mode

Timer 3 in Gated Timer Mode

The gated timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '010_B' or '011_B'.

Bit T3M.0 (T3CON.3) selects the active level of the gate input. In gated timer mode the same options are available for the input frequency as for the timer mode. However, the input clock to the timer in this mode is gated by the external input line T3IN (Timer T3 External Input); an associated port pin should be configured as input.

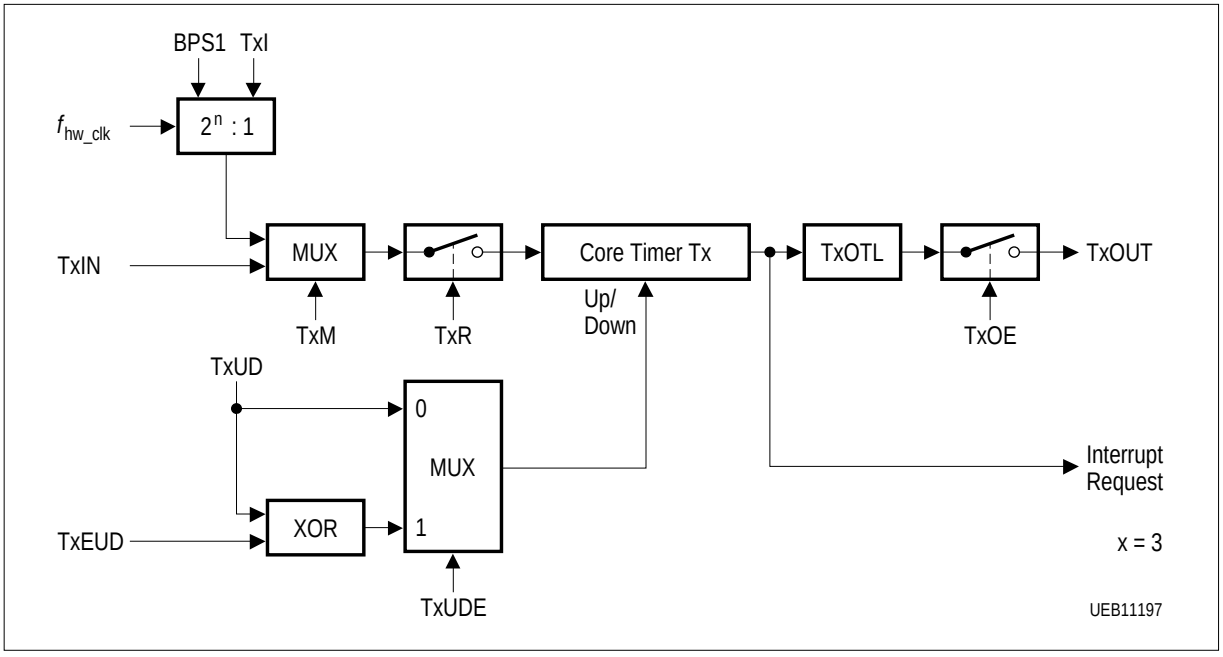


Figure 7-3 Block Diagram of Core Timer T3 in Gated Timer Mode

If T3M = '010_B', the timer is enabled when T3IN shows a low level. A high level at this line stops the timer. If T3M = '011_B', line T3IN must have a high level in order to enable the timer. In addition, the timer can be turned on or off by software using bit T3R. The timer will only run, if T3R is set and the gate is active. It will stop, if either T3R is cleared or the gate is inactive.

Note: A transition of the gate signal at line T3IN does not cause an interrupt request.

Timer 3 in Counter Mode

Counter mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '001_B'. In counter mode, timer T3 is clocked by a transition at the external input line T3IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and negative transition at this line. Bit field T3I in control register T3CON selects the triggering transition (see table below).

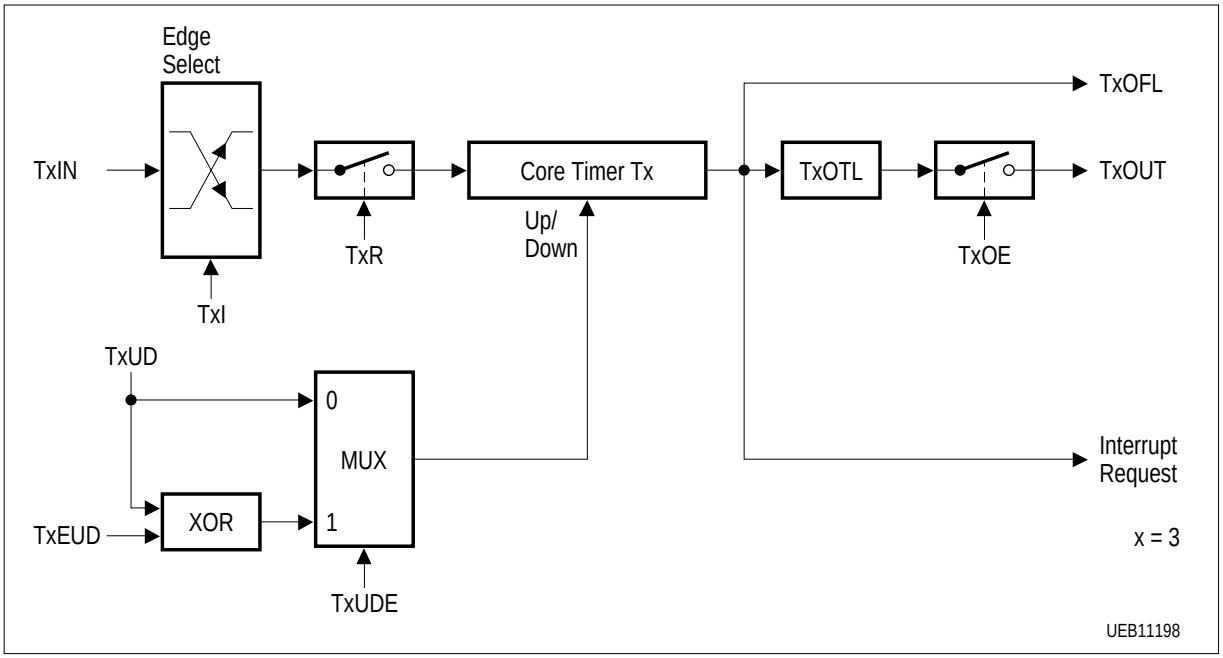


Figure 7-4 Block Diagram of Core Timer T3 in Counter Mode

Table 7-3 Core Timer T3 (Counter Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 is disabled
0 0 1	Positive transition (rising edge) on T3IN
0 1 0	Negative transition (falling edge) on T3IN
0 1 1	Any transition (rising or falling edge) on T3IN
1 X X	Reserved. Do not use this combination

For counter operation, a port pin associated to line T3IN must be configured as input. The maximum input frequency which is allowed in counter mode is $f_{hw\_clk}/8$ (BPS1 = '01'). To ensure that a transition of the count input signal which is applied to T3IN is correctly recognized, its level should be held high or low for at least 4 f_{hw_clk} cycles (BPS1 = '01') before it changes.

Timer 3 in Incremental Interface Mode

Incremental Interface mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '110_B' or '111_B'. In Incremental Interface mode the two inputs associated with timer T3 (T3IN, T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input lines which gives 2-fold or 4-fold the resolution of the encoder input.

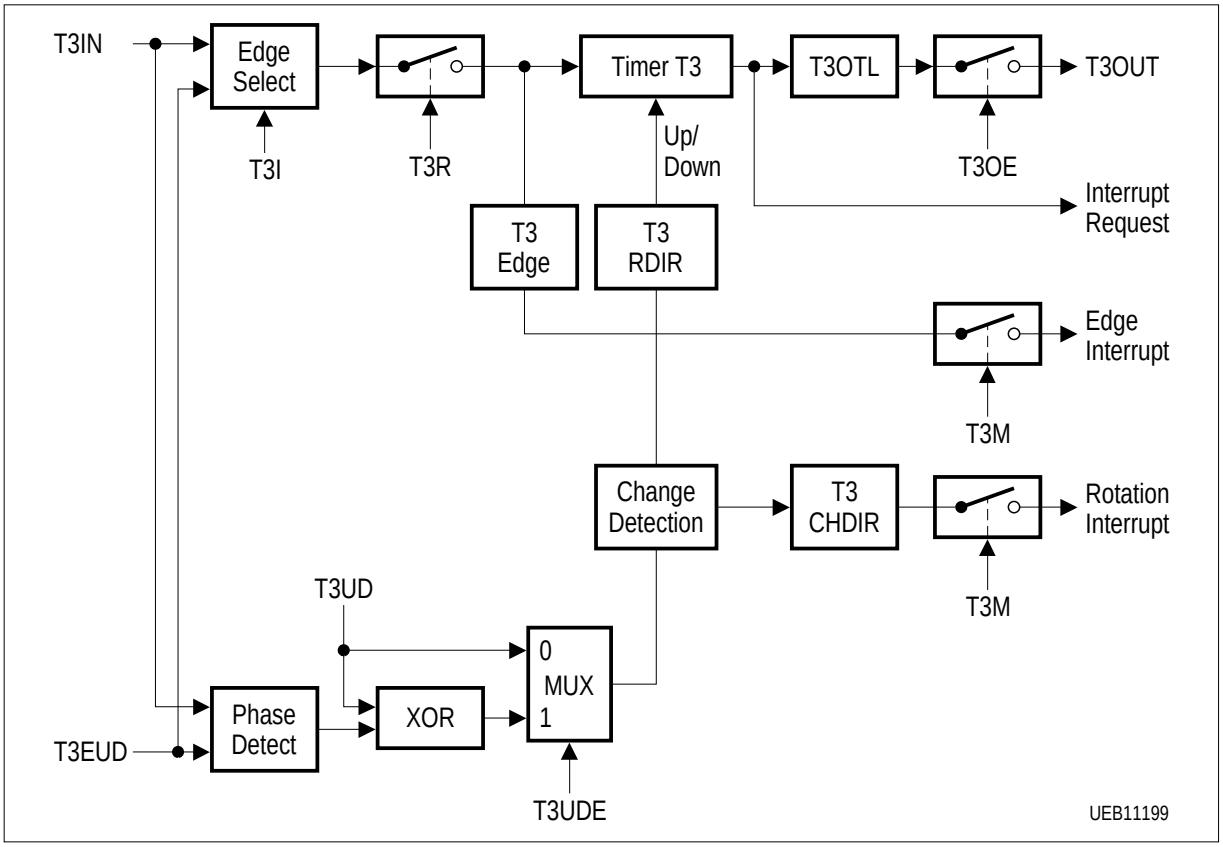


Figure 7-5 Block Diagram of Core Timer T3 in Incremental Interface Mode

The T3I bit field in control register T3CON selects the triggering transitions (see **Table 7-4**). In this mode, the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. Depending on the chosen Incremental Interface Mode, rotation detection '110_B' or edge detection '111_B', an interrupt is generated. For the rotation detection, an interrupt will be generated each time the count direction of timer T3 changes. For the edge detection an interrupt will be

generated each time a count action for timer T3 occurs. Count direction, changes in the count direction and count requests are monitored through the status bits T3RDIR, T3CHDIR and T3EDGE in register T3CON. T3 is modified automatically according to the speed and the direction of the incremental encoder. Therefore, the contents of the T3 timer always represents the encoder's current position.

Table 7-4 Core Timer T3 (Incremental Interface Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 stops.
0 0 1	Any transition (rising or falling edge) on T3IN.
0 1 0	Any transition (rising or falling edge) on T3EUD.
0 1 1	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1 X X	Reserved. Do not use this combination.

The incremental encoder can be connected directly to the microcontroller without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (e.g. A, $\overline{A}$) to digital signals (e.g. A). This greatly increases noise immunity.

Note: The third encoder output T0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3.

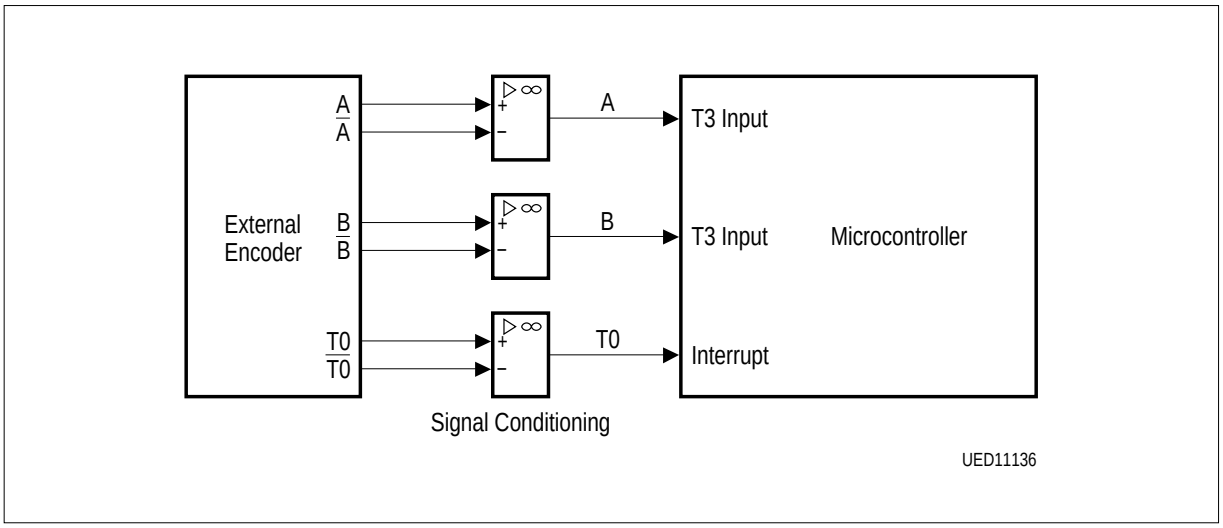


Figure 7-6 Interfacing the Encoder to the Microcontroller

For incremental interface operation the following conditions must be met:

- The T3M bit field must be '110_B' or '111_B'.
- Pins associated with lines T3IN and T3EUD must be configured as input.
- The T3UDE bit must be set to enable automatic direction control.

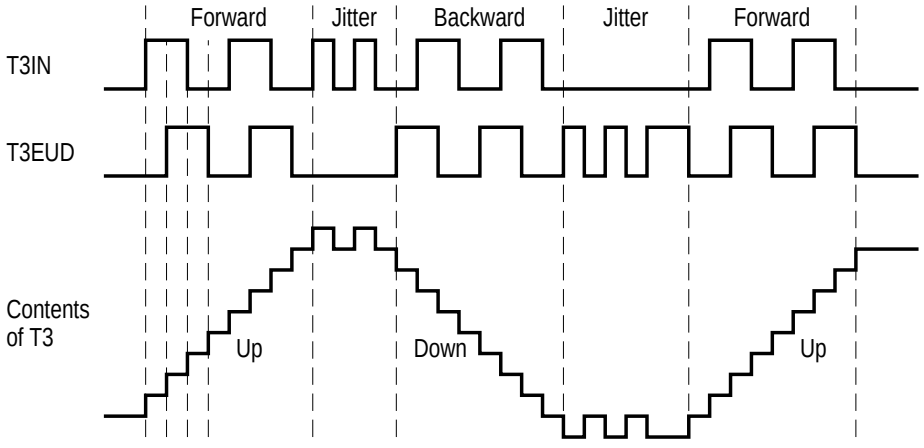
The maximum input frequency which is allowed in incremental interface mode is f_{hw_clk}/ 8 (BPS = 01). To ensure that a transition of any input signal is correctly recognized, its level should be held high or low for at least 4 f_{hw_clk} cycles (BPS = 01) before it changes.

In Incremental Interface Mode the count direction is automatically derived from the sequence in which the input signals change, which corresponds to the rotation direction of the connected sensor. **Table 7-5** summarizes the possible combinations.

Table 7-5 Core Timer T3 (Incremental Interface Mode) Count Direction

Level on Respective other Input	T3IN Input		T3EUD Input	
	Rising ↗	Falling ↘	Rising ↗	Falling ↘
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

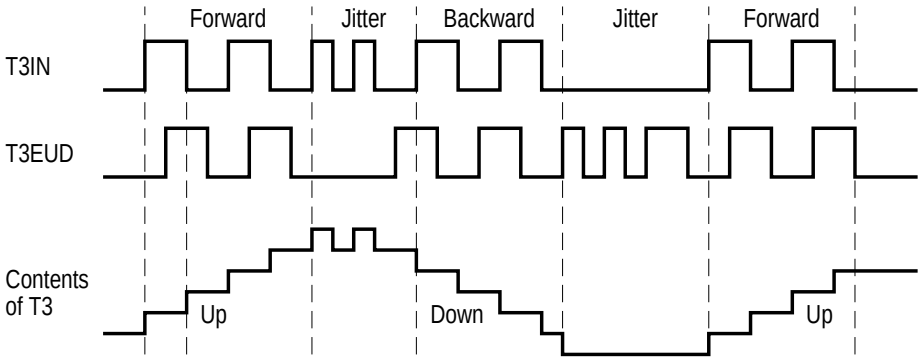
The figures below give examples of T3's operation, visualizing count signal generation and direction control. It also shows how input jitter is compensated, which might occur if the sensor rests near to one of its switching points.



Note: This example shows the timer behavior assuming that T3 counts upon any transition on any input, i.e. T3I = '011_B'.

UET11137

Figure 7-7 Evaluation of the Incremental Encoder Signals



Note: This example shows the timer behavior assuming that T3 counts upon any transition on input T3IN, i.e. T3I = '001_B'.

UET11138

Figure 7-8 Evaluation of the Incremental Encoder Signals

Note: Timer T3, operating in incremental interface mode, automatically provides information on the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods.

Auxiliary Timers T2 and T4

Both auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer, gated timer, counter, or incremental interface mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 4 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer.

The individual configuration for the T2 and T4 timers is determined by their bit-addressable control registers T2CON and T4CON, which are both organized identically. Note that functions which are present in all 3 timers of timer block 1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

Run control for auxiliary timers T2 and T4 can be handled by the associated Run Control Bit T2R, T4R in register T2CON/T4CON. Alternatively, a remote control option (T2RC, T4RC set) may be enabled to start and stop T2/T4 via the run bit T3R of core timer T3.

Timers T2 and T4 in Timer Mode or Gated Timer Mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly with two exceptions:

- There is no TxOUT output line for T2 and T4.
- Overflow/Underflow Monitoring is not supported (no output toggle latch).

Timers T2 and T4 in Counter Mode

In counter mode timers T2 and T4 can be clocked either by a transition at the respective external input line TxIN, or by a transition of timer T3's output toggle latch T3OTL.

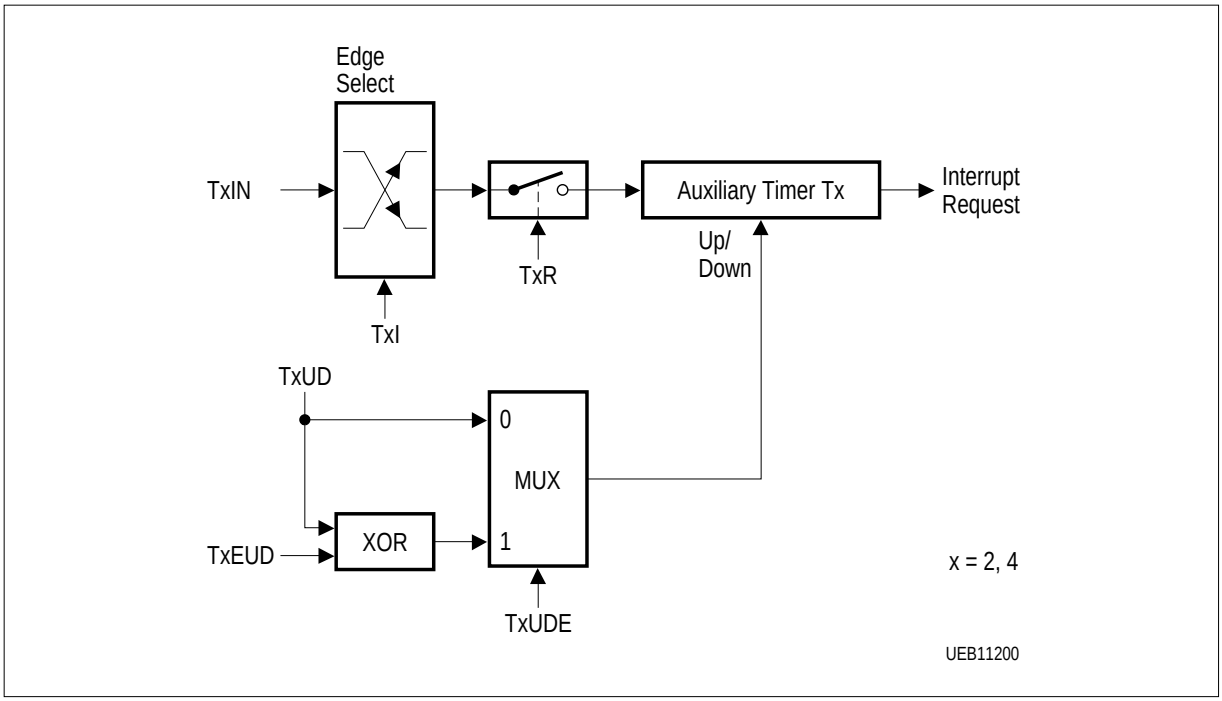


Figure 7-9 Block Diagram of an Auxiliary Timer in Counter Mode

The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and negative transition at either the respective input line, or at the output toggle latch T3OTL.

Bit field TxI in the respective control register TxCON selects the triggering transition (see **Table 7-6**).

Table 7-6 Auxiliary Timer (Counter Mode) Input Edge Selection

T2I/T4I	Triggering Edge for Counter Increment/Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

Note: Only state transitions of T3OTL which are caused by the overflows/underflows of T3 will trigger the counter function of T2/T4. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

For counter operation, an external pin associated to line TxIN must be configured as input. The maximum input frequency which is allowed in counter mode is $f_{hw\_clk}/8$ (BPS1 = ‘01’). To ensure that a transition of the count input signal which is applied to TxIN is correctly recognized, its level should be held for at least 4 f_{hw_clk} cycles (BPS1 = ‘01’) before it changes.

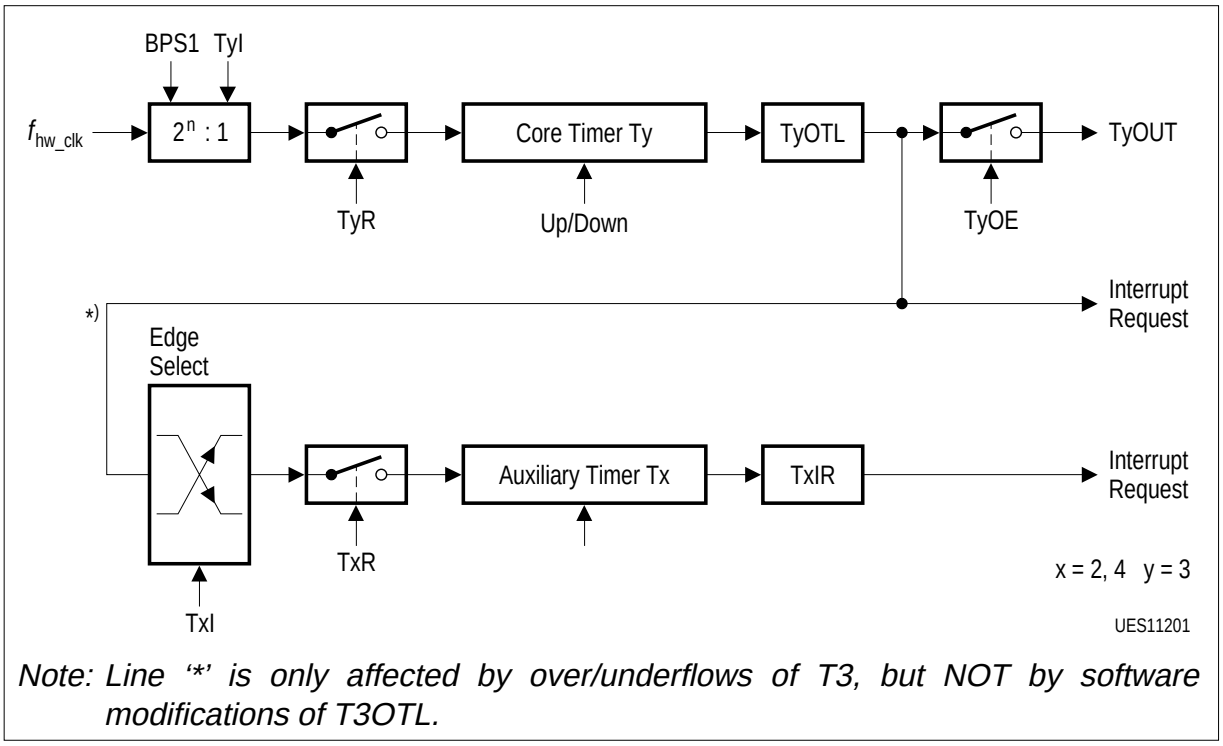
7.1.1.1 Timer Concatenation

Using the output toggle latch T3OTL as a clock source for an auxiliary timer in counter mode concatenates the core timer T3 with the respective auxiliary timer. Depending on which transition of T3OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer/counter.

- **32-bit Timer/Counter**: If both positive and negative transitions of T3OTL are used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter**: If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T3. This configuration forms a 33-bit timer (16-bit core timer + T3OTL + 16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.

In this case T3 can operate in timer, gated timer or counter mode.



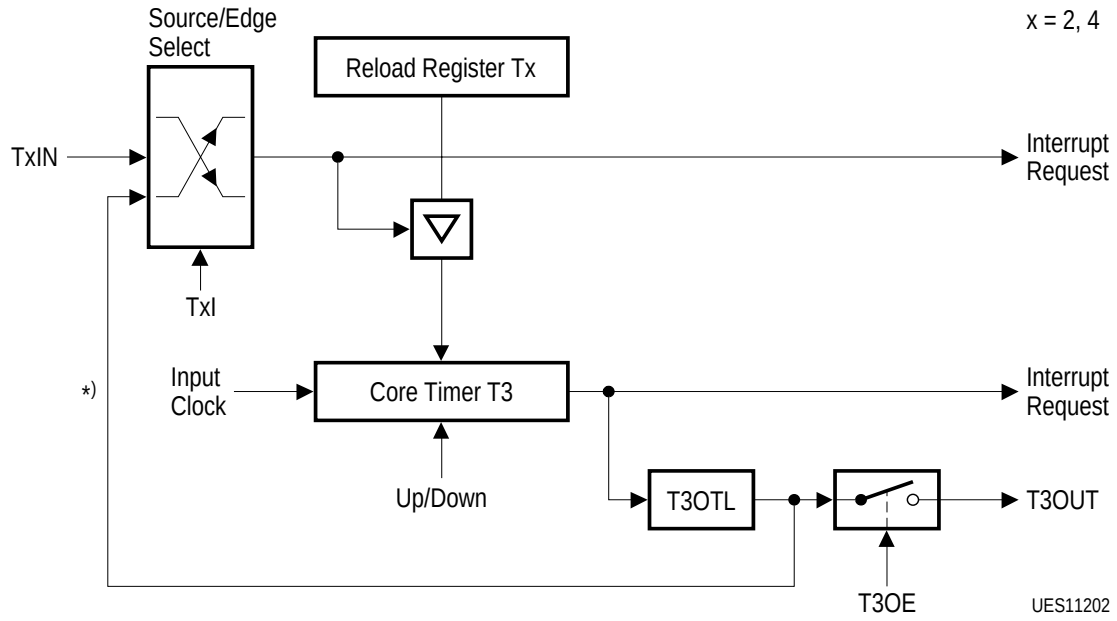
Note: Line '' is only affected by over/underflows of T3, but NOT by software modifications of T3OTL.*

Figure 7-10 Concatenation of Core Timer T3 and an Auxiliary Timer

Auxiliary Timer in Reload Mode

Reload mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '100_B'. In reload mode the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see **Table 7-6**), i.e. a transition of the auxiliary timer's input or the output toggle latch T3OTL may trigger the reload.

Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.



Note: Line '' is only affected by over/underflows of T3, but NOT by software modifications of T3OTL.*

Figure 7-11 GPT1 Auxiliary Timer in Reload Mode

Upon a trigger signal, T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

Note: When a T3OTL transition is selected for the trigger signal, also the interrupt request flag T3IR will be set upon a trigger, indicating T3's overflow or underflow. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

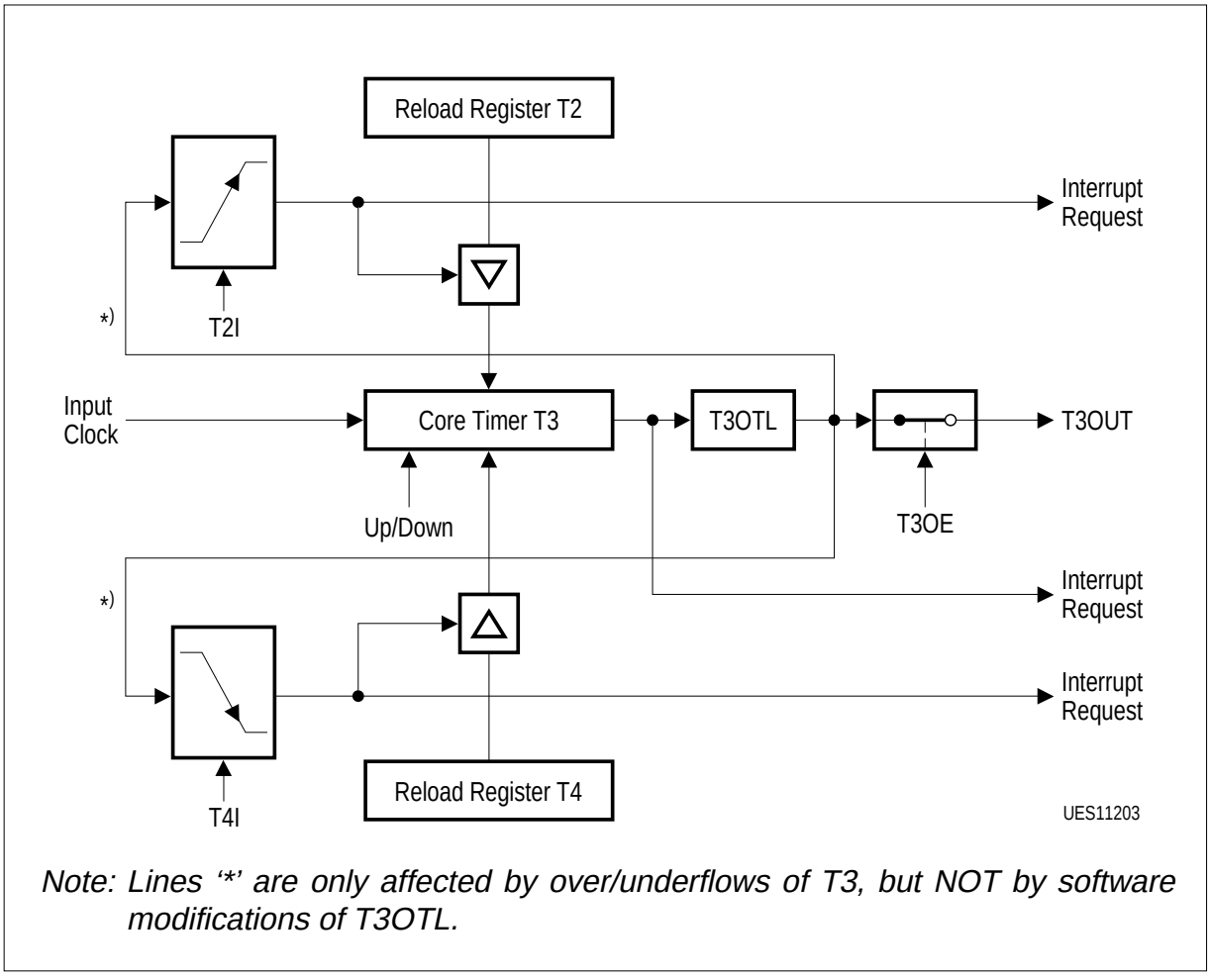
The reload mode triggered by T3OTL can be used in a number of different configurations. Depending on the selected active transition the following functions can be performed:

- If both a positive and a negative transition of T3OTL are selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/ underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this "single-transition" mode for both auxiliary timers allows very flexible pulse width modulation (PWM) to be performed. One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

Figure 7-12 shows an example of the generation of a PWM signal using the alternate reload mechanism. T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions). The PWM signal can be output on line T3OUT if the control bit T3OE is set. With this method the high and low time of the PWM signal can be varied in a wide range.

Note: The output toggle latch T3OTL is accessible via software and may be changed, if required, to modify the PWM signal. However, this will NOT trigger the reloading of T3.

Note: An associated port pin linked to line T3OUT should be configured as output.



Note: Lines '' are only affected by over/underflows of T3, but NOT by software modifications of T3OTL.*

Figure 7-12 GPT1 Timer Reload Configuration for PWM Generation

Note: Although it is possible, selecting the same reload trigger event for both auxiliary timers should be avoided. In this case both reload registers would try to load the core timer at the same time. If this combination is selected, T2 is disregarded and the contents of T4 reloaded.

Auxiliary Timer in Capture Mode

Capture mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '101_B'. In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input line TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bit field TxI are used to select the active transition (see table in the counter mode section), while the most significant bit, TxI.2, is irrelevant for capture mode. It is recommended to keep this bit cleared (TxI.2 = '0').

Note: When programmed for capture mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.

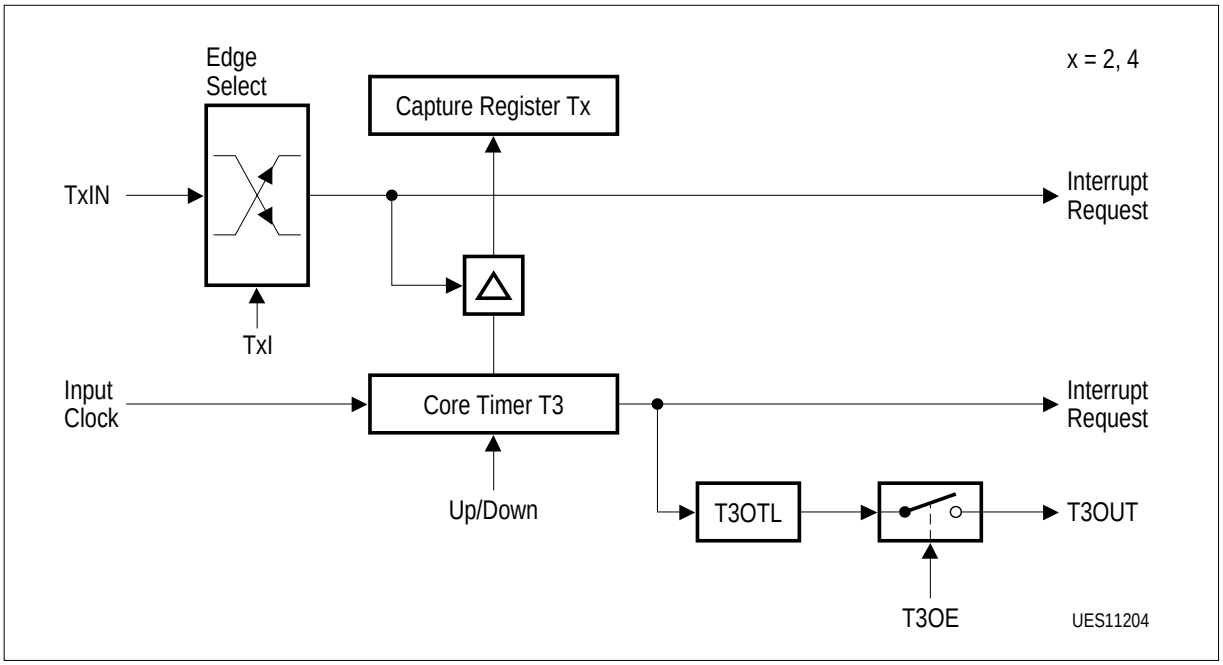


Figure 7-13 Auxiliary Timer of Timer Block 1 in Capture Mode

Upon a trigger (selected transition) at the corresponding input line TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

Note: The direction control for T2IN and for T4IN must be set to 'Input', and the level of the capture trigger signal should be kept high or low for at least 4 f_{hw_clk} (BPS1 = '01') cycles before it changes, to ensure correct edge detection.

7.1.2 Functional Description of Timer Block 2

Timer block 2 includes the two timers T5 (referred to as the auxiliary timer) and T6 (referred to as the core timer), and the 16-bit capture/reload register CAPREL.

The count direction (Up / Down) may be programmed by software. The auxiliary timer T6 may be reloaded with the contents of CAPREL.

The toggle bit (T6OTL) also supports the concatenation of T6 with auxiliary timer T5, while concatenation of T6 with other timers is provided through line T6OFL. Triggered by an external signal, the contents of T5 can be captured in register CAPREL, and T5 may optionally be cleared. Both timer T6 and T5 can count up or down, and the current timer value can be read or modified by the CPU in the non-bitaddressable SFRs T5 and T6.

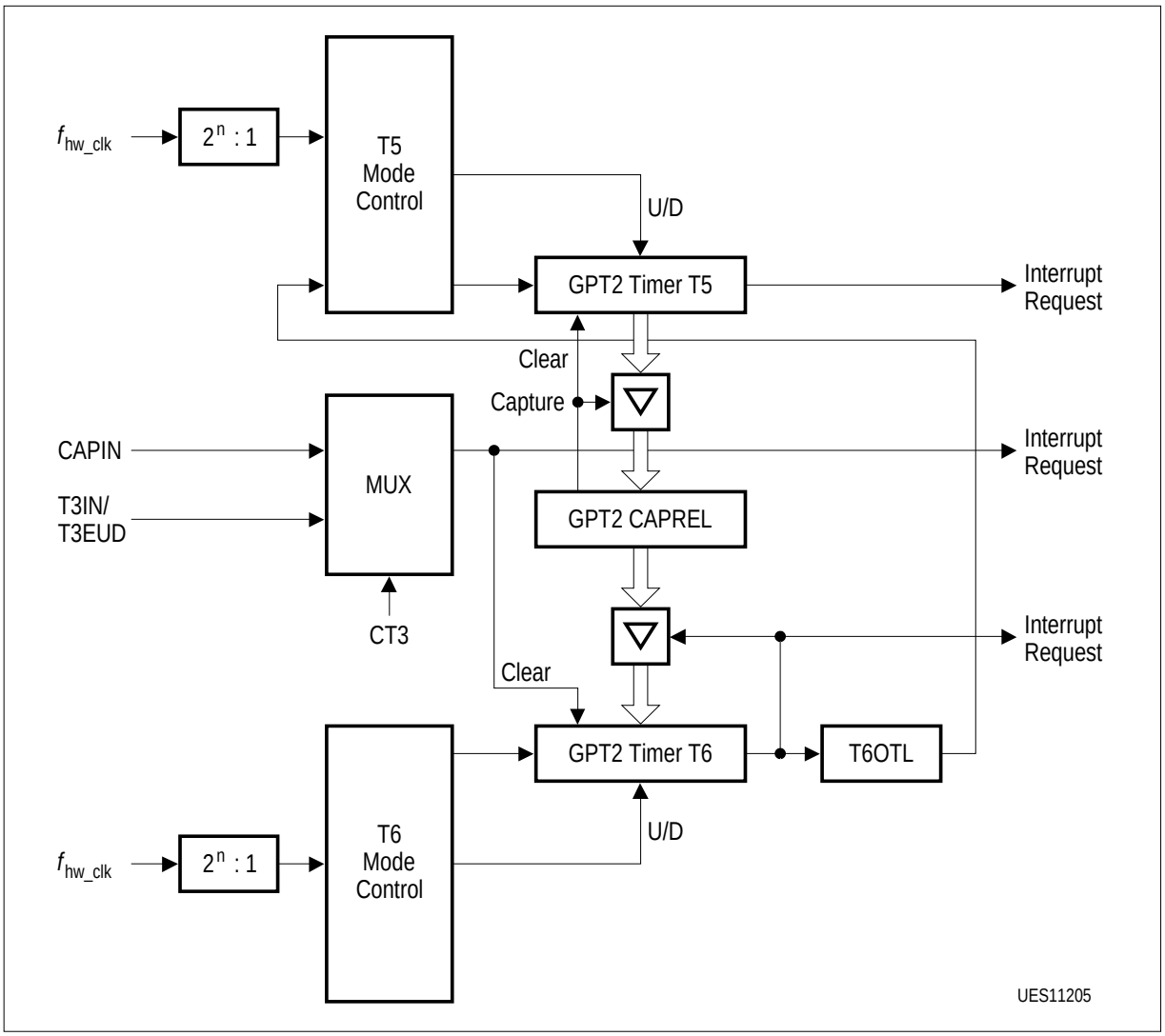


Figure 7-14 Structure of Timer Block 2

7.1.2.1 Core Timer T6

The operation of the core timer T6 is controlled by its bit-addressable control register T6CON.

Timer 6 Run Bit

The timer can be started or stopped by software through bit T6R (Timer T6 Run Bit). Setting bit T6R will start the timer, clearing T6R stops the timer.

Note: When bit T5RC is set, bit T6R will also control (start and stop) auxiliary timer T5.

Count Direction Control

The count direction of the core timer can be controlled by software. The count direction can be changed regardless of whether the timer is running or not.

Table 7-7 Core Timer T6 Count Direction Control

Bit TxUD	Count Direction
0	Count Up
1	Count Down

Note: The direction control works the same for core timer T6 and for auxiliary timer T5. Therefore the lines and bits are named Tx …

Timer 6 Overflow/Underflow Monitoring

An overflow or underflow of timer T6 will clock the toggle latch T6OTL in control register T6CON. T6OTL can also be set or reset by software. T6OTL can be used in conjunction with the timer over/underflows as an input for the counter function of the auxiliary timer T5.

Timer 6 in Timer Mode

Timer mode for the core timer T6 is selected by setting bit field T6M in register T6CON to '000_B'. In this mode, T6 is clocked with the module clock divided by a programmable prescaler, which is selected by bit field T6I. The input frequency f_{T6} for timer T6 and its resolution r_{T6} are scaled linearly with lower clock frequencies f_{hw_clk}, as can be seen from the following formula:

$$f_{T6} = \frac{f_{hw\_clk}}{BPS2 \times 2^{<T6I>}} \qquad r_{T6}\,[ms] = \frac{BPS2 \times 2^{<T6I>}}{f_{hw\_clk}\,[MHz]}$$

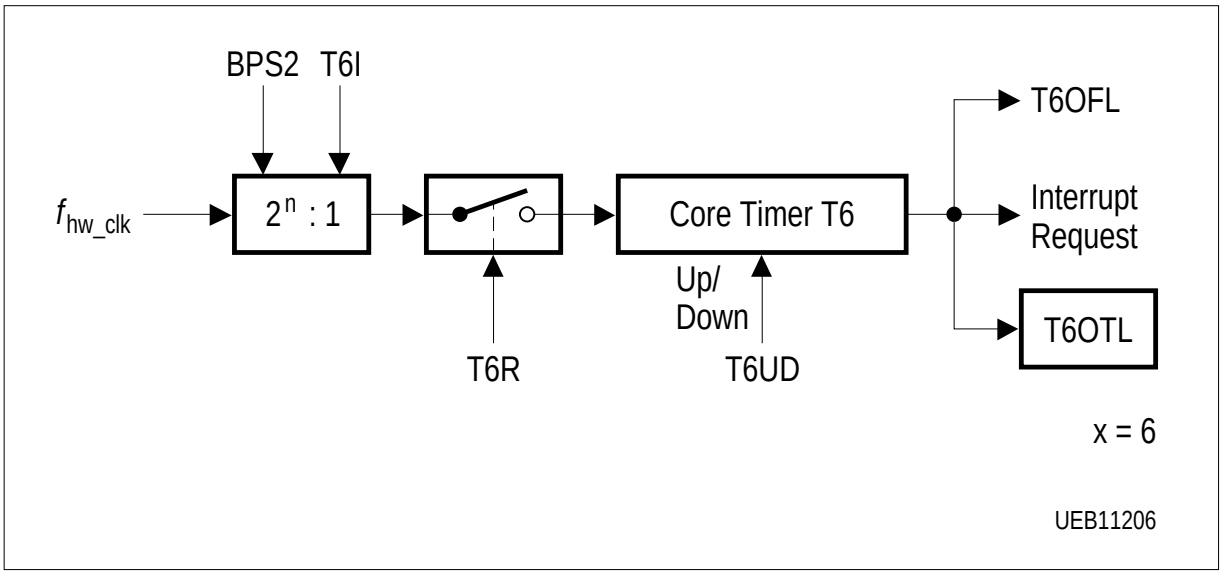


Figure 7-15 Block Diagram of Core Timer T6 in Timer Mode

7.1.2.2 Auxiliary Timer T5

The auxiliary timer T5 can be configured for timer mode using the same options for the timer frequencies and the count signal as the core timer T6. In addition to these 3 counting modes, the auxiliary timer can be concatenated with the core timer.

The individual configuration for timer T5 is determined by its bit-addressable control register T5CON. Note that functions present in both timers of timer block 2 are controlled in the same bit positions and manner in each of the specific control registers.

Run control for auxiliary timer T5 can be handled by the associated Run Control Bit T5R in register T5CON. Alternatively, a remote control option (T5RC is set) may be enabled to start and stop T5 via the run bit T6R of core timer T6.

Note: The auxiliary timer has no overflow/underflow toggle latch. Therefore, an output line for Overflow/Underflow monitoring is not provided.

Count Direction Control for Auxiliary Timer

The count direction of the auxiliary timer can be controlled in the same way as for the core timer T6. The description and the table apply accordingly.

Timer T5 in Timer Mode

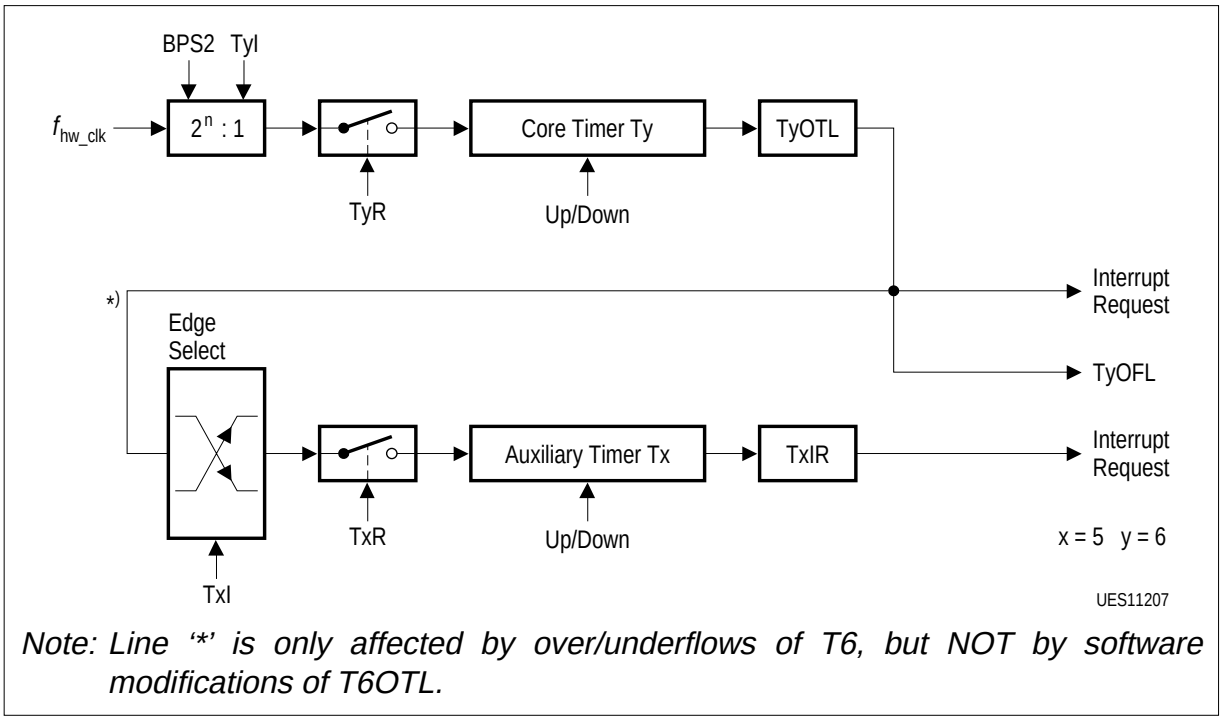
When the auxiliary timer T5 is programmed to timer mode its operation is the same as described for the core timer T6. The descriptions, figures and tables apply accordingly with one exception:

- Overflow/Underflow monitoring is not supported (no output toggle latch).

7.1.2.3 Timer Concatenation

Using the toggle bit T6OTL as a clock source for the auxiliary timer in counter mode concatenates the core timer T6 with the auxiliary timer. Depending on which transition of T6OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer/counter.

- 32-bit Timer/Counter: If both positive and negative transitions of T6OTL are used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T6. Thus, the two timers form a 32-bit timer.
- 33-bit Timer/Counter: If either a positive or a negative transition of T6OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T6. This configuration forms a 33-bit timer (16-bit core timer + T6OTL + 16-bit auxiliary timer). The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations. In this case T6 can operate in timer, gated timer or counter mode.



Note: Line '' is only affected by over/underflows of T6, but NOT by software modifications of T6OTL.*

Figure 7-16 Concatenation of Core Timer T6 and Auxiliary Timer T5

Capture/Reload Register CAPREL in Capture Mode

This 16-bit register can be used as a capture register for the auxiliary timer T5. This mode is selected by setting bit T5SC in control register T5CON. The CT3 bit selects the external input line CAPIN or the input lines of timer T3 as the source for a capture trigger. Either a positive, a negative, or both a positive and a negative transition at line CAPIN can be selected to trigger the capture function, or transitions on input T3IN or input

T3EUD or both inputs T3IN and T3EUD. The active edge is controlled by bit field CI in register T5CON.

The maximum input frequency for the capture trigger signal at CAPIN is $f_{hw\_clk}/2$ (BPS2 = '01'). To ensure that a transition of the capture trigger signal is correctly recognized, its level should be held for at least 2 f_{hw_clk} cycles (BPS2 = '01') before it changes.

When the timer T3 capture trigger is enabled (CT3 is set), register CAPREL captures the contents of T5 when transitions of the selected input(s) occur. These values can be used to measure T3's input signals. This is useful e.g. when T3 operates in incremental interface mode, in order to derive dynamic information (speed acceleration) from the input signals.

When a selected transition at the external input line CAPIN is detected, the contents of the auxiliary timer T5 are latched into register CAPREL, and interrupt request flag CRIR is set. At the same time, timer T5 can be cleared to 0000_H. This option is controlled by bit T5CLR in register T5CON. If T5CLR = '0', the contents of timer T5 are not affected by a capture. If T5CLR = '1', timer T5 is cleared after the current timer value has been latched into register CAPREL.

Note: Bit T5SC only controls whether a capture is performed or not. If T5SC = '0', the input line CAPIN can still be used to clear timer T5 or as an external interrupt input. This interrupt is controlled by the CAPREL interrupt control register CRIC.

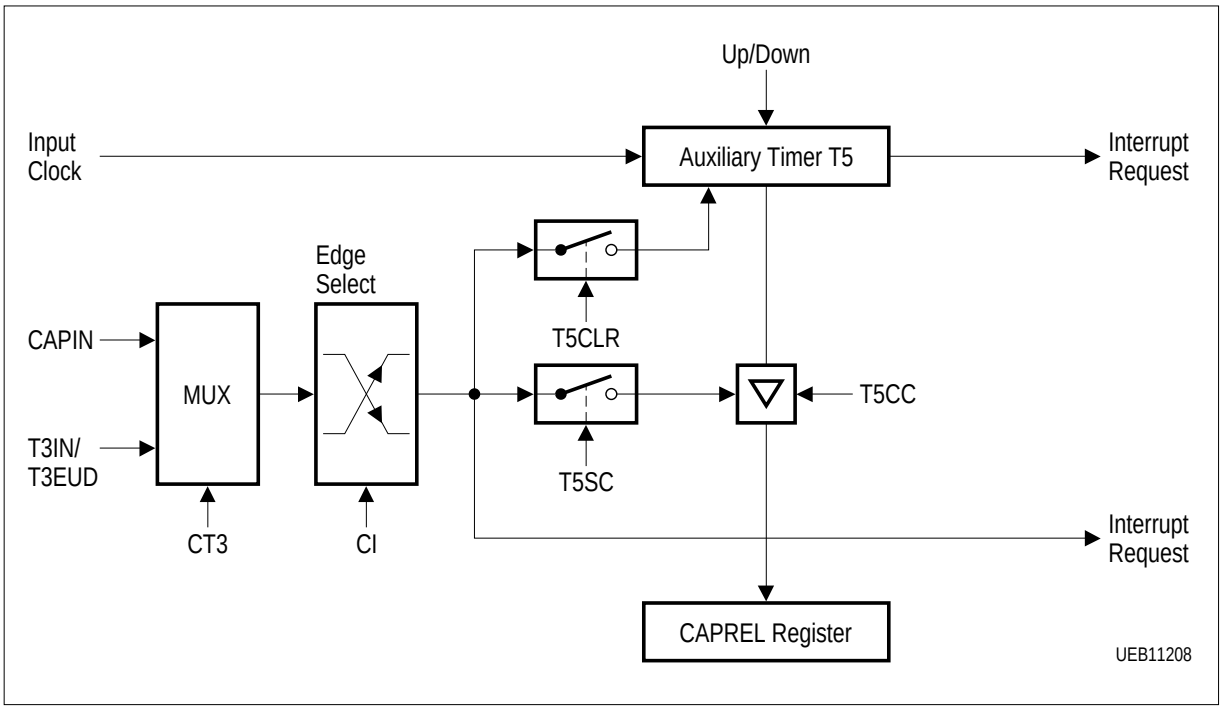


Figure 7-17 Timer Block 2 Register CAPREL in Capture Mode

Timer Block 2 Capture/Reload Register CAPREL in Reload Mode

This 16-bit register can be used as a reload register for the core timer T6. This mode is selected by setting bit T6SR = ‘1’ in register T6CON. The operation causing a reload in this mode is an overflow or underflow of the core timer T6.

When timer T6 overflows from $FFFF_H$ to 0000_H (when counting up) or when it underflows from 0000_H to $FFFF_H$ (when counting down), the value stored in register CAPREL is loaded into timer T6. This will not set the interrupt request flag CRIR associated with the CAPREL register. However, interrupt request flag T6IR will be set indicating the overflow/underflow of T6.

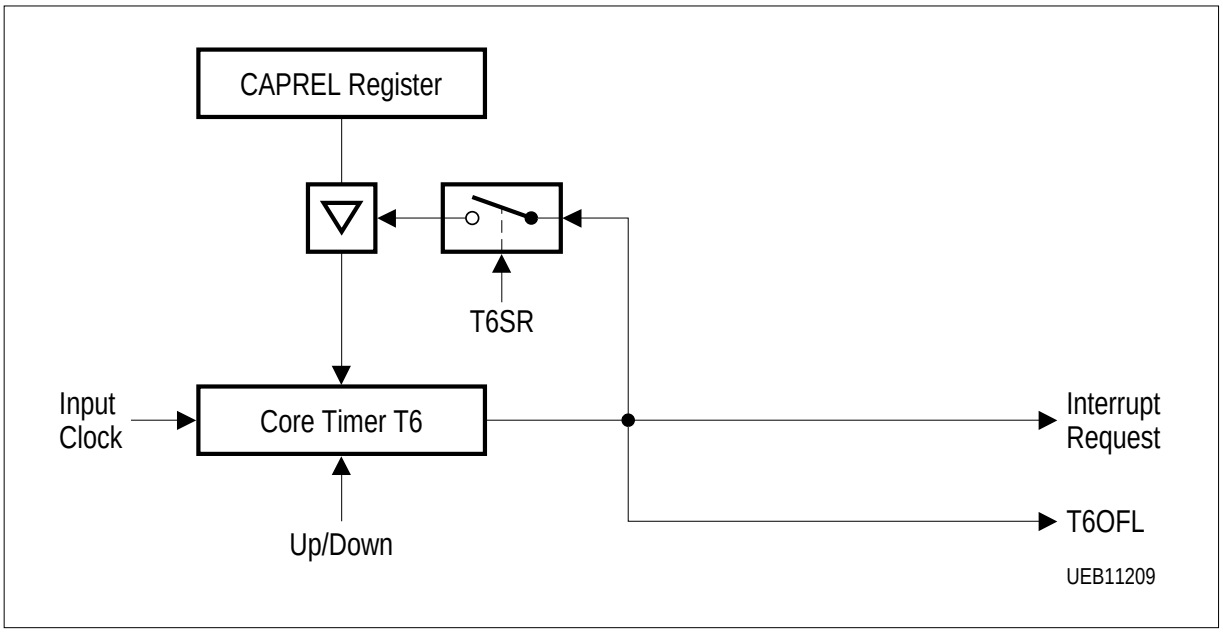


Figure 7-18 Timer Block 2 Register CAPREL in Reload Mode

Timer Block 2 Capture/Reload Register CAPREL in Capture-And-Reload Mode

Since the reload function and the capture function of register CAPREL can be enabled individually by bits T5SC and T6SR, by setting both bits the two functions can be enabled simultaneously. This feature can be used to generate an output frequency that is a multiple of the input frequency.

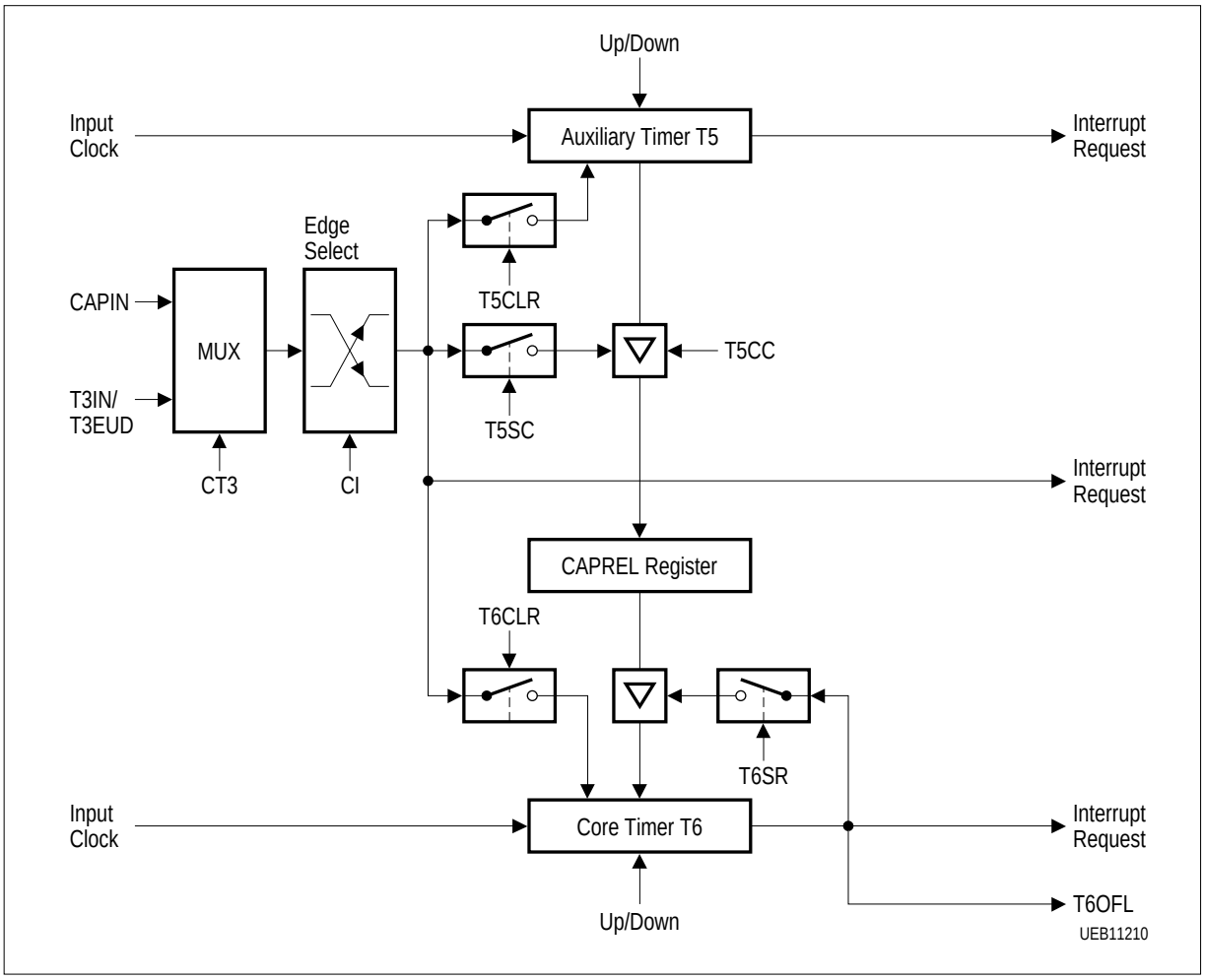


Figure 7-19 Timer Block 2 Register CAPREL in Capture-And-Reload Mode

This combined mode can be used to detect consecutive external events which may occur aperiodically, but where a finer resolution, i.e. more ‘ticks’ within the time between two external events, is required.

For this purpose, the time between the external operations is measured using timer T5 and the CAPREL register. Timer T5 runs in timer mode counting up with a frequency of e.g. $f_{hw\_clk}/32$. The external operations are applied to line CAPIN. When an external operation occurs, the timer T5 contents are latched into register CAPREL, and timer T5 is cleared (T5CLR = ‘1’). Thus, register CAPREL always contains the correct time between two operations, measured in timer T5 increments. Timer T6, which runs in timer mode counting down with a frequency of e.g. $f_{hw\_clk}/4$, uses the value in register CAPREL to perform a reload on underflow. This means that the value in register CAPREL represents the time between two underflows of timer T6, now measured in timer T6 increments. Since timer T6 runs 8 times faster than timer T5, it will underflow 8 times within the time between two external operations. Thus, the underflow signal of timer T6 generates 8 ‘ticks’. Upon each underflow, the interrupt request flag T6IR will be set and

bit T6OTL will be toggled. This signal has 8 times more transitions than the signal which is applied to line CAPIN.

A certain deviation of the output frequency is generated by the fact that timer T5 will count actual time units (e.g. T5 running at 1 MHz will capture the value $64_H/100_D$ for a 10 KHz input signal) while T6OTL will only toggle on an underflow of T6 (i.e. the transition from 0000_H to $FFFF_H$). In the above mentioned example, T6 would count down from 64_H so the underflow would occur after 101 T6 timing ticks. The actual output frequency then is 79.2 KHz instead of the expected 80 KHz.
This can be solved by activating the capture correction (T5CC = '1'). If the capture correction is actived the content of T5 is decremented by 1 before being captured. The deviation described is eliminated (in the example T5 would capture $63_H/99_D$ and the output frequency would be 80 KHz).

The underflow signal of timer T6 can furthermore be used to clock one or more of the CAPCOM unit's timers, which gives the user the possibility to set compare operations based on a finer resolution than that of the external operations. This connection is accomplished through the T6OFL signal.

7.1.3 GPT Registers

All available kernel registers are summarized in **Table 7-8**.

Table 7-8 GPT Register Summary

Name	Reset Value	Description
T2CON	0000_H	Timer 2 Control Register
T3CON	0000_H	Timer 3 Control Register
T4CON	0000_H	Timer 4 Control Register
T5CON	0000_H	Timer 5 Control Register
T6CON	0000_H	Timer 6 Control Register
CAPREL	0000_H	Capture/Reload Register
T2	0000_H	Timer 2 Register
T3	0000_H	Timer 3 Register
T4	0000_H	Timer 4 Register
T5	0000_H	Timer 5 Register
T6	0000_H	Timer 6 Register

Function Control Registers

The operating mode of the core timer T3 is configured and controlled by its bit-addressable control register T3CON.

T3CON

Timer 3 Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T3 RDIR	**T3 CH DIR**	**T3 EDGE**	**T3 BPS1**		**T3 OTL**	**T3 OE**	**T3 UDE**	**T3 UD**	**T3R**	**T3M**			**T3I**		
rh	rwh	rwh	rw		rwh	rw	rw	rw	rw	rw			rw		

Field	Bits	Type	Description
T3I	[2:0]	rw	Timer 3 Input Parameter Selection Timer mode see **Table 7-9** for encoding Gated Timer see **Table 7-9** for encoding Counter mode see **Table 7-10** for encoding Incremental Interface mode see **Table 7-11** for encoding
T3M	[5:3]	rw	Timer 3 Mode Control 000 Timer Mode 001 Counter Mode 010 Gated Timer with Gate active low 011 Gated Timer with Gate active high 100 ***Reserved***. Do not use this combination! 101 ***Reserved***. Do not use this combination! 110 Incremental Interface Mode (Rotation detection) 111 Incremental Interface Mode (Edge detection)
T3R	[6]	rw	Timer 3 Run Bit 0 Timer/Counter 3 stops 1 Timer/Counter 3 runs
T3UD	[7]	rw	**Timer 3 Up/Down Control** (when T3UDE = ‘0’) 0 Counting ‘Up’ 1 Counting ‘Down’
T3UDE	[8]	rw	**Timer 3 External Up/Down Enable** 0 Counting direction is internally controlled by SW 1 Counting direction is externally controlled by line T3EUD

Field	Bits	Type	Description
T3OE	[9]	rw	Overflow/Underflow Output Enable 0 T3 overflow/underflow can not be externally monitored 1 T3 overflow/underflow may be externally monitored via T3OUT
T3OTL	[10]	rwh	Timer 3 Output Toggle Latch Toggles on each overflow/underflow of T3. Can be set or reset by software.
BPS1	[12:11]	rw	Timer Block Prescaler 1 The maximum input frequency 00 For Timer 2/3/4 is $f_{hw\_clk}/8$ 01 For Timer 2/3/4 is $f_{hw\_clk}/4$ 10 For Timer 2/3/4 is $f_{hw\_clk}/32$ 11 For Timer 2/3/4 is $f_{hw\_clk}/16$
T3EDGE	[13]	rwh	Timer 3 Edge Detection The bit is set on each successful edge detection. The bit has to be reset by SW. 0 No count edge was detected 1 A count edge was detected
T3CHDIR	[14]	rwh	Timer 3 Count Direction Change The bit is set on a change of the count direction of timer 3. The bit has to be reset by SW. 0 No change in count direction was detected 1 A change in count direction was detected
T3RDIR	[15]	rh	Timer 3 Rotation Direction 0 Timer 3 counts up 1 Timer 3 counts down

Table 7-9 Timer 3 Input Parameter Selection for Timer Mode and Gated Mode

T3I	Prescaler for f_{hw_clk} (BPS1 = 00)	Prescaler for f_{hw_clk} (BPS1 = 01)	Prescaler for f_{hw_clk} (BPS1 = 10)	Prescaler for f_{hw_clk} (BPS1 = 11)
000	8	4	32	16
001	16	8	64	32
010	32	16	128	64
011	64	32	256	128
100	128	64	512	256
101	256	128	1024	512
110	512	256	2048	1024
111	1024	512	4096	2048

Table 7-10 Timer 3 Input Parameter Selection for Counter Mode

T3I	Triggering Edge for Counter Update
000	None. Counter T3 is disabled
001	Positive transition (raising edge) on T3IN
010	Negative transition (falling edge) on T3IN
011	Any transition (raising or falling edge) on T3IN
1XX	***Reserved***. Do not use this combination!

Table 7-11 Timer 3 Input Parameter Selection for Incremental Interface Mode

T3I	Triggering Edge for Counter Update
000	None. Counter T3 stops
001	Any transition (raising or falling edge) on T3IN
010	Any transition (raising or falling edge) on T3EUD
011	Any transition (raising or falling edge) on T3IN or T3EUD
1XX	***Reserved***. Do not use this combination!

T2CON
T4CON
Timer 2/4 Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Tx RDIR	Tx CH DIR	Tx EDG E	Tx IR DIS	0		Tx RC	Tx UDE	Tx UD	TxR	TxM			TxI		
rh	rwh	rwh	rw	r		rw	rw	rw	rw	rw			rw		

Field	Bits	Type	Description
TxI	[2:0]	rw	Timer x Input Parameter Selection Timer mode see **Table 7-12** for encoding Gated Timer see **Table 7-12** for encoding Counter mode see **Table 7-13** for encoding Incremental Interface mode see **Table 7-14** for encoding
TxM	[5:3]	rw	Timer x Mode Control (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer with Gate active low 011 Gated Timer with Gate active high 100 Reload Mode 101 ***Reserved***. Do not use this combination! 110 Incremental Interface Mode (Rotation detection) 111 Incremental Interface Mode (Edge detection)
TxR	[6]	rw	Timer x Run Bit 0 Timer/Counter x stops 1 Timer/Counter x runs
TxUD	[7]	rw	Timer x Up/Down Control (when TxUDE = ‘0’) 0 Counting ‘Up’ 1 Counting ‘Down’
TxUDE	[8]	rw	**Timer x External Up/Down Enable** 0 Counting direction is internally controlled by SW 1 Counting direction is externally controlled by line TxEUD

Field	Bits	Type	Description
TxRC	[9]	rw	Timer x Remote Control 0 Timer/Counter x is controlled by its own run bit TxR 1 Timer/Counter x is controlled by the run bit of core timer 3
TxIRDIS	[12]	rw	Timer x Interrupt Disable 0 Interrupt generation for TxCHDIR and TxEDGE interrupts in Incremental Interface Mode is enabled 1 Interrupt generation for TxCHDIR and TxEDGE interrupts in Incremental Interface Mode is disabled
TxEDGE	[13]	rwh	Timer x Edge Detection The bit is set on each successful edge detection. The bit has to be reset by SW. 0 No count edge was detected 1 A count edge was detected
TxCHDIR	[14]	rwh	Timer x Count Direction Change The bit is set on a change of the count direction of timer x. The bit has to be reset by SW. 0 No change in count direction was detected 1 A change in count direction was detected
TxRDIR	[15]	rh	Timer x Rotation Direction 0 Timer x counts up 1 Timer x counts down

Table 7-12 Timer x Input Parameter Selection for Timer Mode and Gated Mode

T3I	Prescaler for f_{hw_clk} (BPS1 = 00)	Prescaler for f_{hw_clk} (BPS1 = 01)	Prescaler for f_{hw_clk} (BPS1 = 10)	Prescaler for f_{hw_clk} (BPS1 = 11)
000	8	4	32	16
001	16	8	64	32
010	32	16	128	64
011	64	32	256	128
100	128	64	512	256
101	256	128	1024	512
110	512	256	2048	1024
111	1024	512	4096	2048

Table 7-13 Timer x Input Parameter Selection for Counter Mode

TxI	Triggering Edge for Counter Update
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (raising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (raising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

Table 7-14 Timer x Input Parameter Selection for Incremental Interface Mode

TxI	Triggering Edge for Counter Update
000	None. Counter Tx stops
001	Any transition (raising or falling edge) on TxIN
010	Any transition (raising or falling edge) on TxEUD
011	Any transition (raising or falling edge) on TxIN or TxEUD
1XX	***Reserved***. Do not use this combination!

T6CON
Timer 6 Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T6 SR	T6 CLR	0	BPS2		T6 OTL	0	0	T6 UD	T6R	T6M			T6I		
rw	rw	r	rw		rwh	rw	rw	rw	rw	rw			rw		

Field	Bits	Type	Description
T6I	[2:0]	rw	Timer 6 Input Parameter Selection Timer mode see **Table 7-15** for encoding
T6M	[5:3]	rw	Timer 6 Mode Control (Basic Operating Mode) 000 Timer Mode 001 ***Reserved***. Do not use this combination! 010 ***Reserved***. Do not use this combination! 011 ***Reserved***. Do not use this combination! 1XX ***Reserved***. Do not use this combination!
T6R	[6]	rw	Timer 6 Run Bit 0 Timer/Counter 6 stops 1 Timer/Counter 6 runs
T6UD	[7]	rw	Timer 6 Up/Down Control 0 Counting ‘Up’ 1 Counting ‘Down’
T6OTL	[10]	rwh	Timer 6 Output Toggle Latch Toggles on each overflow/underflow of T6. Can be set or reset by software.
BPS2	[12:11]	rw	Timer Block Prescaler 2 The maximum input frequency 00 For Timer 5/6 is $f_{hw\_clk}/4$ 01 For Timer 5/6 is $f_{hw\_clk}/2$ 10 For Timer 5/6 is $f_{hw\_clk}/16$ 11 For Timer 5/6 is $f_{hw\_clk}/8$
T6CLR	[14]	rw	Timer 6 Clear Bit 0 Timer 6 is not cleared on a capture event 1 Timer 6 is cleared on a capture event
T6SR	[15]	rw	Timer 6 Reload Mode Enable 0 Reload from register CAPREL disabled 1 Reload from register CAPREL enabled

Table 7-15 Timer 6 Input Parameter Selection for Timer Mode and Gated Mode

T6I	Prescaler for f_{hw_clk} (BPS2 = 00)	Prescaler for f_{hw_clk} (BPS2 = 01)	Prescaler for f_{hw_clk} (BPS2 = 10)	Prescaler for f_{hw_clk} (BPS2 = 11)
000	4	2	16	8
001	8	4	32	16
010	16	8	64	32
011	32	16	128	64
100	64	32	256	128
101	128	64	512	256
110	256	128	1024	512
111	512	256	2048	1024

Table 7-16 Timer 6 Input Parameter Selection for Counter Mode

T6I	Triggering Edge for Counter Update
000	None. Counter T6 is disabled
001	***Reserved***. Do not use this combination!
010	***Reserved***. Do not use this combination!
011	***Reserved***. Do not use this combination!
1XX	***Reserved***. Do not use this combination!

T5CON
Timer 5 Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T5 SR	T5 CLR	CI		T5 CC	CT3	T5 RC	0	T5 UD	T5R	T5M			T5I		
rw	rw	rw		rw	rw	rw	rw	rw	rw	rw			rw		

Field	Bits	Type	Description
T5I	[2:0]	rw	Timer 5 Input Parameter Selection Timer mode see **Table 7-17** for encoding Counter mode see **Table 7-18** for encoding
T5M	[5:3]	rw	Timer 5 Mode Control (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 ***Reserved***. Do not use this combination! 011 ***Reserved***. Do not use this combination! 1XX ***Reserved***. Do not use this combination!
T5R	[6]	rw	Timer 5 Run Bit 0 Timer/Counter 5 stops 1 Timer/Counter 5 runs
T5UD	[7]	rw	Timer 5 Up/Down Control 0 Counting ‘Up’ 1 Counting ‘Down’
T5RC	[9]	rw	Timer 5 Remote Control 0 Timer/Counter x is controlled by its own run bit T5R 1 Timer/Counter 5 is controlled by the run bit of core timer 6
CT3	[10]	rw	Timer 3 Capture Trigger Enable 0 Capture trigger from input line CAPIN 1 Capture trigger from T3 input lines
T5CC	[11]	rw	Timer 5 Capture Correction 0 T5 is just captured without any correction 1 T5 is decremented by 1 before being captured

Field	Bits	Type	Description
CI	[13:12]	rw	Register CAPREL Capture Trigger Selection (depending in bit CT3) 00 Capture disabled 01 Positive transition (rising edge) on CAPIN or any transition on T3IN 10 Negative transition (falling edge) on CAPIN or any transition on T3EUD 11 Any transition (rising or falling edge) on CAPIN or any transition on T3IN or T3EUD
T5CLR	[14]	rw	Timer 5 Clear Bit 0 Timer 5 is not cleared on a capture operation 1 Timer 5 is cleared on a capture operation
T5SC	[15]	rw	Timer 5 Capture Mode Enable 0 Capture into register CAPREL disabled 1 Capture into register CAPREL enabled

Table 7-17 Timer 5 Input Parameter Selection for Timer Mode and Gated Mode

T5I	Prescaler for f_{hw_clk} (BPS2 = 00)	Prescaler for f_{hw_clk} (BPS2 = 01)	Prescaler for f_{hw_clk} (BPS2 = 10)	Prescaler for f_{hw_clk} (BPS2 = 11)
000	4	2	16	8
001	8	4	32	16
010	16	8	64	32
011	32	16	128	64
100	64	32	256	128
101	128	64	512	256
110	256	128	1024	512
111	512	256	2048	1024

Table 7-18 Timer 5 Input Parameter Selection for Counter Mode

T5I	Triggering Edge for Counter Update
X00	None. Counter T5 is disabled
001	Reserved. Do not use this combination.
010	Reserved. Do not use this combination.
011	Reserved. Do not use this combination.
101	Positive transition (rising edge) of output toggle latch T6OTL
110	Negative transition (falling edge) of output toggle latch T6OTL
111	Any transition (rising or falling edge) of output toggle latch T6OTL

7.1.4 Interrupts

For a detailed description of the various interrupts see description above. An overview is given with a **Table 7-19**:

Table 7-19 Peripheral Name Interrupt Sources

Interrupt	Interrupt Node	Description
Timer 2 Overflow	T2IC	Interrupt is requested on overflow of timer 2 if counting up.
Timer 2 Underflow	T2IC	Interrupt is requested on underflow of timer 2 if counting down.
Timer 3 Overflow	T3IC	Interrupt is requested on overflow of timer 3 if counting up.
Timer 3 Underflow	T3IC	Interrupt is requested on underflow of timer 3 if counting down.
Timer 4 Overflow	T4IC	Interrupt is requested on overflow of timer 4 if counting up.
Timer 4 Underflow	T4IC	Interrupt is requested on underflow of timer 4 if counting down.
Timer 5 Overflow	T5IC	Interrupt is requested on overflow of timer 5 if counting up.
Timer 5 Underflow	T5IC	Interrupt is requested on underflow of timer 5 if counting down.
Timer 6 Overflow	T6IC	Interrupt is requested on overflow of timer 6 if counting up.
Timer 6 Underflow	T6IC	Interrupt is requested on underflow of timer 6 if counting down.

Table 7-19 Peripheral Name Interrupt Sources (cont'd)

Interrupt	Interrupt Node	Description
Rotation Direction Change Timer 2	T2IC	Interrupt is requested on a change of the count direction in the Incremental Interface Mode (T2I = 110).
Edge Detection Timer 2	T2IC	Interrupt is requested on a successful detected edge resulting in a timer count action (T2I = 111).
Rotation Direction Change Timer 3	T3IC	Interrupt is requested on a change of the count direction in the Incremental Interface Mode (T3I = 110).
Edge Detection Timer 3	T3IC	Interrupt is requested on a successful detected edge resulting in a timer count action (T3I = 111).
Rotation Direction Change Timer 4	T4IC	Interrupt is requested on a change of the count direction in the Incremental Interface Mode (T4I = 110).
Edge Detection Timer 4	T4IC	Interrupt is requested on a successful detected edge resulting in a timer count action (T4I = 111).
Reload Action Timer 2	T2IC	Interrupt is requested on a trigger signal for reloading timer 3 in Reload Mode (T2I = 100).
Reload Action Timer 4	T4IC	Interrupt is requested on a trigger signal for reloading timer 3 in Reload Mode (T4I = 100).
Capture Action Timer 2	T2IC	Interrupt is requested on a trigger signal for a capture action to capture timer 3 in timer 2 Capture Mode (T2I = 101).
Capture Action Timer 4	T4IC	Interrupt is requested on a trigger signal for a capture action to capture timer 3 in timer 4 Capture Mode (T4I = 101).
Capture Action Timer Block 2	CRIC	Interrupt is requested on a trigger signal for a capture action of timer 5 to register CAPREL in Capture Mode (T5SC = 1).

7.2 Real-time Clock

7.2.1 General Description

The Real Time Clock (RTC) module of M2 is basically an independent timer chain and counts time ticks. The base frequency of the RTC can be programmed via a reload counter. The RTC can work fully asynchronous to the system frequency (33.33 MHz) and is optimized on a low power consumption.

The RTC serves different purposes:

- Real-time clock to determine the current time and date
- Cyclic time based interrupt
- Alarm interrupt for wake up on a defined time
- 48-bit timer for long term measurements

The real time clock module provides three different types of registers: a control register for controlling the RTC´s functionality, three data registers for setting the clock divider for RTC base frequency programming and for flexible interrupt generation, and three counter registers that contain the actual time and date. The interrupts are programmed via one interrupt subnode register and via an interrupt node register.

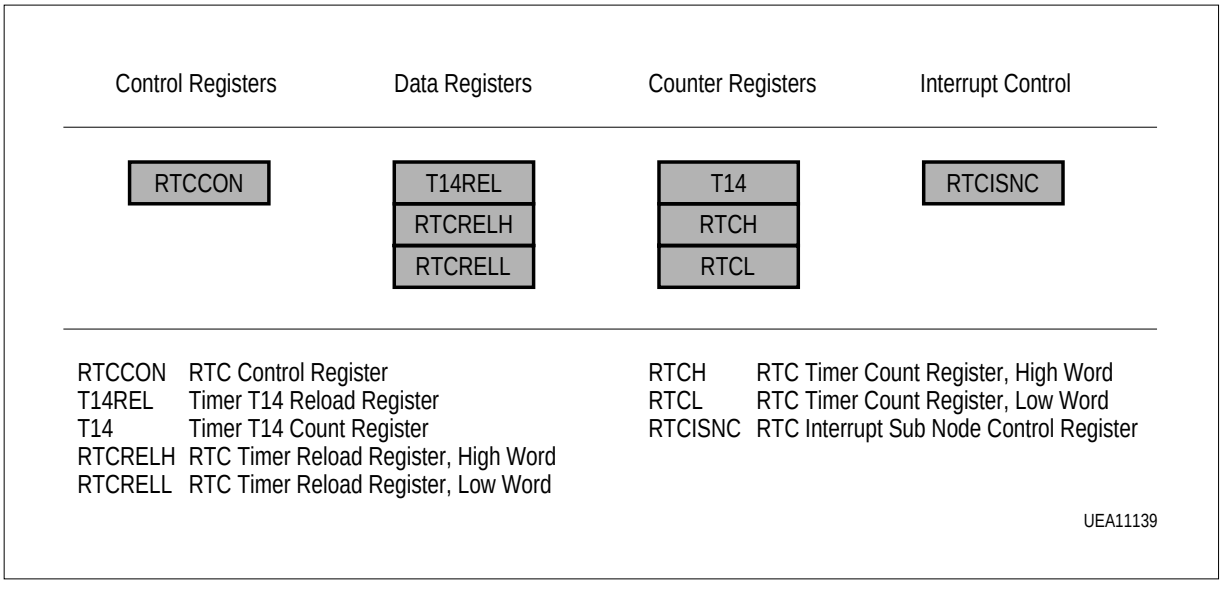


Figure 7-20 RTC Register Overview

The RTC module consists of a chain of 2 divider blocks, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up. Timer T14 is reloaded with the value of register T14REL on every T14 timer overflow. T14REL is transparent during reload state.

Figure 7-21 shows the RTC block diagram:

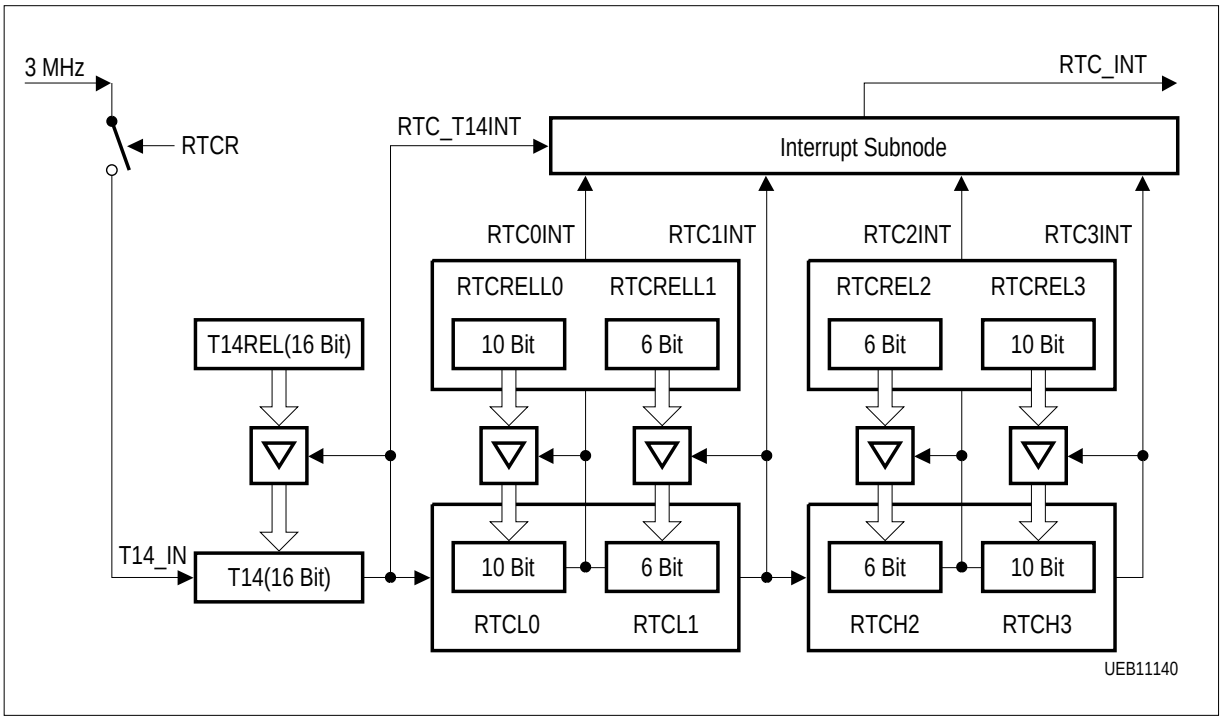


Figure 7-21 RTC Block Diagram

RTC Control

The operating behavior of the RTC module is controlled by the RTCCON register. The RTC starts counting by setting the RTCR run bit. After reset, the run bit is set and the RTC automatically starts operation. Setting bits T14DEC or T14INC allows the T14 timer to be manually and asynchronously decremented or incremented. These bits are cleared by hardware after the decrement/increment operation. The RTC is only reset in case of a hardware reset, so it keeps on running during idle and sleep mode.

Cyclic Interrupt Generation

The RTC module can generate an interrupt request RTC_INT whenever timer T14 overflows and is reloaded. This interrupt request may be used, for example, to provide a system time tick, independent of the CPU clock frequency, without loading the general purpose timers, or to wake up regularly from idle mode. The T14 overflow interrupt (RTC_T14INT) cycle time can be adjusted via the timer T14 reload register T14REL. The 32 bit timer (RTCL and RTCH) can be divided into smaller reloadable timers. Each sub-timer can be programmed for an overflow on different time bases (e.g. second, hour, minute, day). With each timer overflow an RTC interrupt is generated. All these RTC interrupts are ored within the RTC module via the interrupt subnode RTCISNC to one interrupt request RTC_INT. This interrupt RTC_INT is one source of another interrupt subnode ISNC in the interrupt controller. The other interrupt input of subnode ISNC is disconnected in the M2. With typical values of T14REL = $F448_H$, RTCRELL = 1018_H and

RTCRELH = $FA04_H$, counter T14 generates one overflow per millisecond, RTCL0 one per second, RTCL1 one per minute, RTCH2 one per hour and RTCH3 one per day.

48-bit Timer Operation

The concatenation of the 16-bit reload timer T14 and the 32-bit RTC timer can be regarded as a 48-bit timer which counts with the RTC count input frequency (3 MHz). The reload registers T14REL, RTCRELL and RTCRELH should be cleared to get a 48-bit binary timer. However, any other reload values may be used.

Interrupt Subnode RTCISNC

All RTC interrupts are connected to one interrupt node via an interrupt subnode. For this interrupt sharing each interrupt source has, in addition to the node enable and request flag its own enable and request flag located in register RTCISNC. After an RTC interrupt (RTC_INT) is arbitrated, the interrupt service routine has to check all the enabled sources request flags and run the respective software routine. The request flags have to be deleted by software before leaving the interrupt service routine.

Reset Behavior

The RTC registers are only cleared or set by a hardware reset. Bit RTCR is set when the hardware is reset.

7.2.2 Register Description

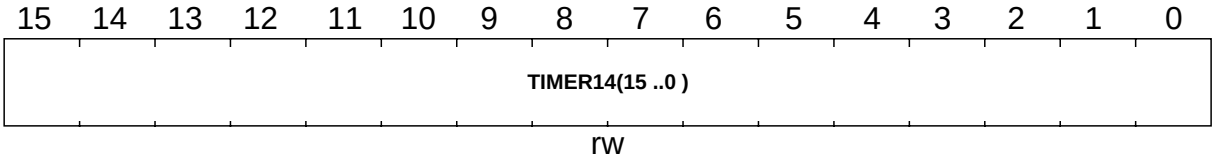
RTCCON **Reset Value: 0003_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	T14 INC	T14 DEC	0	RTC R
												rw	rw		rw

Bit	Function
RTCR	**RTC Run Bit** '0': RTC stops '1': RTC runs
T14DEC	**Decrement T14 Timer Value** Setting this bit to 1 effects a decrement of the T14 timer value. The bit is cleared by hardware after decrementation.
T14INC	**Increment T14 Timer Value** Setting this bit to 1 effects an increment of the T14 timer value. The bit is cleared by hardware after incrementation.

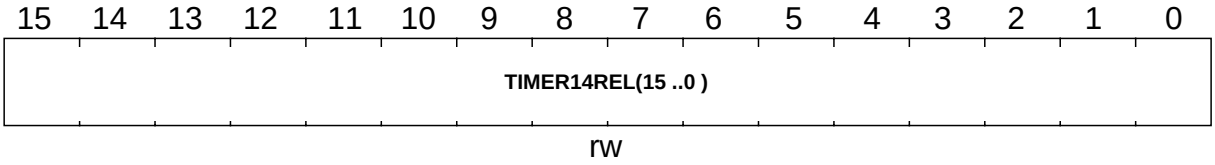
Note: Bit RTCR is set on hardware reset.

T14 **Reset Value: 0000_H**



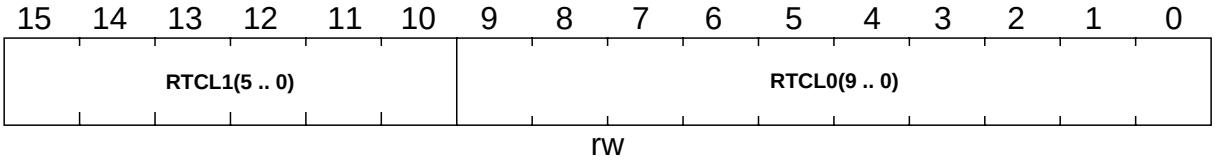
Bit	Function
TIMER14 (15 … 0)	**16 Bit Timer Register** Timer T14 generates the input clock for the RTC register and the periodic interrupt.

T14REL **Reset Value: 0000_H**



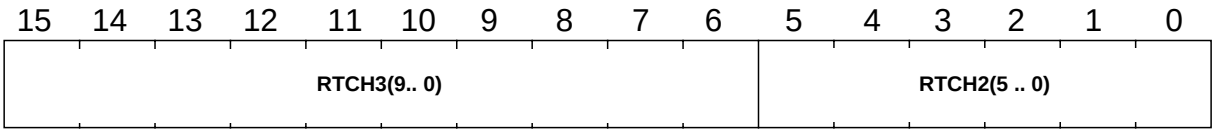
Bit	Function
TIMERREL14 (15 … 0)	**16 Bit Reload Register for Timer 14** Represents the 16 bit reload value for T14

RTCL **Reset Value: 0000_H**



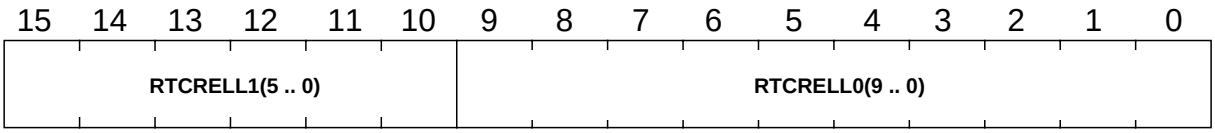
Bit	Function
RTCL1 (5 … 0)	Low Word of 32 Bit Capture Register.
RTCL0 (9 … 0)	

RTCH **Reset Value: 0000 H**



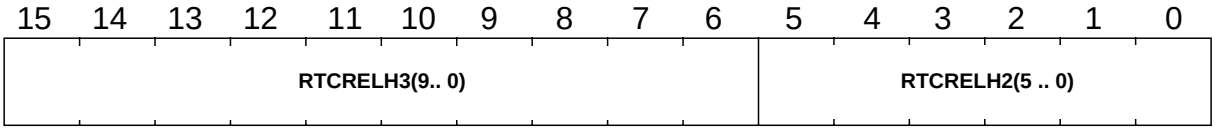
Bit	Function
RTCH3 (9 … 0)	High Word of 32 Bit Capture Register.
RTCH2 (5 … 0)	

RTCRELL **Reset Value: 0000H**



Bit	Function
RTCRELL1 (5 … 0)	Low Word of 32 Bit Reload Register.
RTCRELL0 (9 … 0)	

RTCRELH **Reset Value: 0000H**



Bit	Function
RTCRELH2 (5 … 0)	High Word of 32 Bit Reload Register.
RTCRELH3 (9 … 0)	

RTC Interrupt Subnode Control

RTCISNC **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	RTC 3 IR	RTC 3IE	RTC 2 IR	RTC 2IE	RTC 1 IR	RTC 1 IE	RTC 0 IR	RTC 0IE	T14 IR	T14 IE
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
T14IR	**T14 Overflow Interrupt Request Flag** ‘0’: No request pending. ‘1’: This source has raised an interrupt request.
T14IE	**T14 Overflow Interrupt Enable Control Bit** ‘0’: Interrupt request is disabled. ‘1’: Interrupt request is enabled.
RTCxIR	**RTCx Interrupt Request Flag** ‘0’: No request pending. ‘1’: This source has raised an interrupt request.
RTCxIE	**RTCx Interrupt Enable Control Bit** ‘0’: Interrupt request is disabled. ‘1’: Interrupt request is enabled.

ISNC **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	INT2 IE	INT2 IR	RTC INT IE	RTC INT IR
												rw	rw	rw	rw

Bit	Function
RTCINTIR	**RTC Interrupt Request Flag** '0': No request pending. '1': RTC has raised an interrupt request.
RTCINTIE	**RTC Interrupt Enable Control Bit** '0': Interrupt request is disabled. '1': Interrupt request is enabled.
INT2IR	**Interrupt Request Flag of 2nd Source** disconnected in M2.
INT2IE	**2nd Source Interrupt Enable Control Bit** '0': recommended value.

Note: The interrupt request flags of both RTC interrupt subnodes have to be cleared by software inside the interrupt service routine.

7.3 Asynchronous/Synchronous Serial Interface

The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between M2 and other microcontrollers, microprocessors or external peripherals. It provides the following features:

- Full duplex asynchronous operating modes
  - 8- or 9-bit data frames, LSB first
  - Parity bit generation/checking
  - One or two stop bits
  - Baud rate from 2.0625 MBaud to 0.48 Baud (@ 33 MHz clock)
  - Multiprocessor mode for automatic address/data byte detection
  - Loop-back capability
- Support for IrDA data transmission/reception up to max. 115.2 KBaud
- Autobaud detection unit for asynchronous operating modes
  - Detection of standard baud rates
    1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 Baud
  - Detection of non-standard baud rates
  - Detection of asynchronous modes
    7 bit, even parity; 7 bit, odd parity;
    8 bit, even parity; 8 bit, odd parity; 8 bit, no parity
  - Automatic initialization of control bits and baud rate generator after detection
  - Detection of a serial two-byte ASCII character frame
- Half-duplex 8-bit synchronous operating mode
  - Baud rate from 4.125 MBaud to 420 Baud (@ 33 MHz clock)
- Double buffered transmitter/receiver
- Interrupt generation
  - on a transmitter buffer empty condition
  - on a transmit last bit of a frame condition
  - on a receiver buffer full condition
  - on an error condition (frame, parity, overrun error)
  - on the start and the end of a autobaud detection

Figure 7-22 shows a block diagram of the ASC with its operating modes (asynchronous and synchronous mode.)

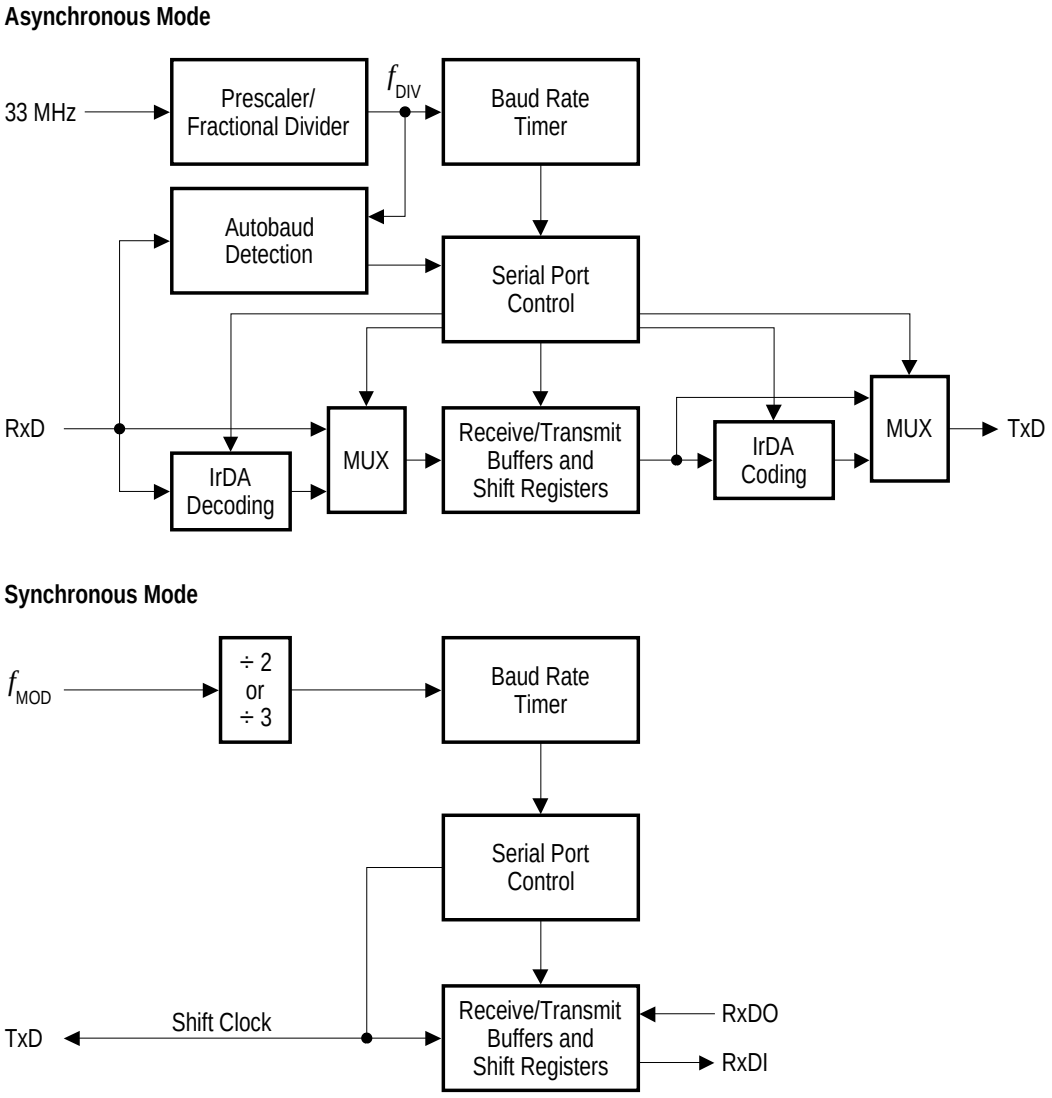


Figure 7-22 Block Diagram of the ASC0

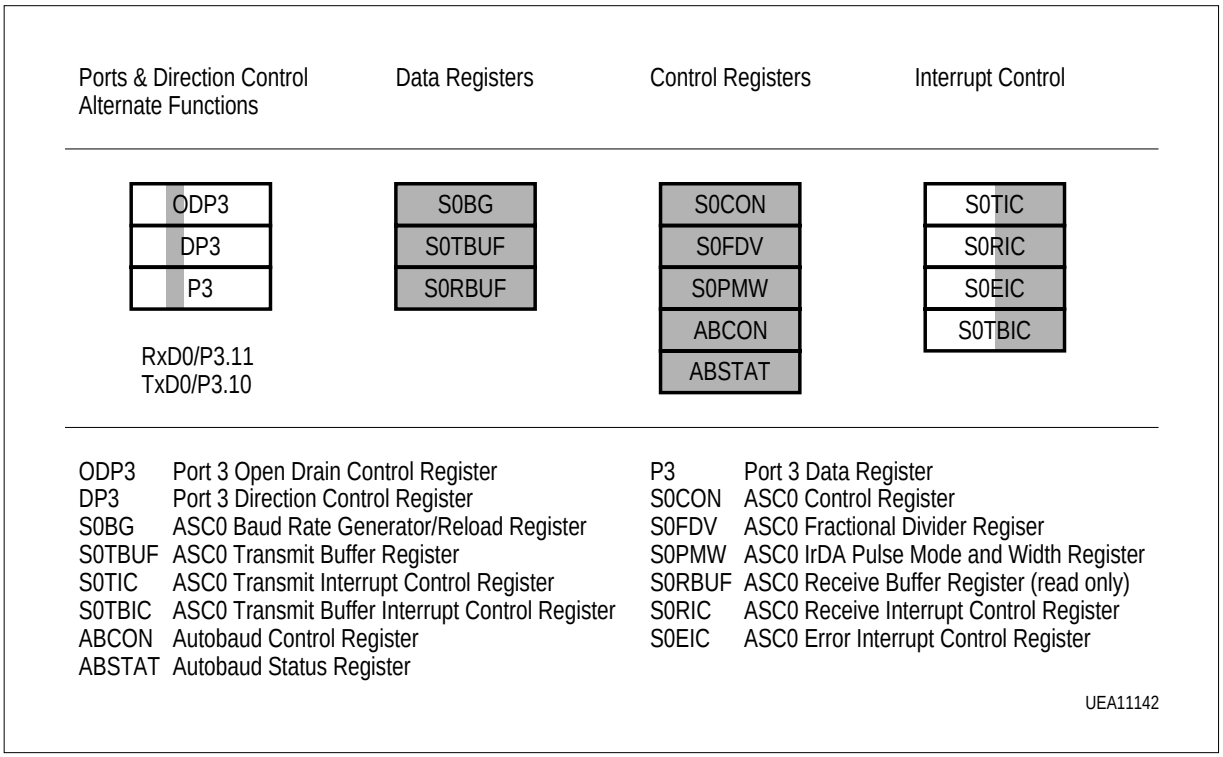


Figure 7-23 ASC Register Overview

The ASC0 supports full-duplex asynchronous communication up to **2.08** MBaud and half-duplex synchronous communication up to **4.16** MBaud (@ 33.33 MHz CPU clock). In synchronous mode, data is transmitted or received synchronous to a shift clock which is generated by the microcontroller. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered. For multiprocessor communication, a mechanism to distinguish address from data bytes is included. Testing is supported by a loop-back option. A 13-bit baud rate timer with a versatile input clock divider circuitry provides the ASC0 with the serial clock signal. In a special asynchronous mode, the ASC0 supports IrDA data transmission up to 115.2 KBaud with fixed or programmable IrDA pulse width.

A transmission is started by writing to the Transmit Buffer register S0TBUF (by way of an instruction or a PEC data transfer). Only the number of data bits which is determined by the selected operating mode, will actually be transmitted, e.g. bits written to positions 9 through 15 of register S0TBUF are always insignificant.
Data transmission is double-buffered, so a new character may be written to the transmit buffer register, before the transmission of the previous character is complete. This allows the transmission of characters back-to-back without gaps.

Data reception is enabled by the receiver enable bit S0REN. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) receive buffer register S0RBUF. Bits in the upper half of S0RBUF which are not valid in the selected operating mode will be read as zeros.
Data reception is double-buffered, so that reception of a second character may begin before the previously received character has been read out of the receive buffer register. In all modes, receive buffer overrun error detection can be selected through bit S0OEN. When enabled, the overrun error status flag S0OE and the error interrupt request line S0EIR will be activated if the receive buffer register has not been read when the reception of a second character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit S0LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output functions of the Port 3 pins are not necessary.

Note: Serial data transmission or reception is only possible when the baud rate generator run bit S0R is set to '1'. Otherwise the serial interface is idle.
Do not program the mode control field S0M in register S0CON to one of the reserved combinations to avoid unpredictable behavior of the serial interface.

7.3.1 Asynchronous Operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same baud rate. Data is transmitted on pin TXD0 and received on pin RXD0. These signals are alternate functions of Port 3 pins. IrDA data transmission/reception supports up to 115.2 KBit/s. **Figure 7-24** shows the block diagram of the ASC0 operating in asynchronous mode.

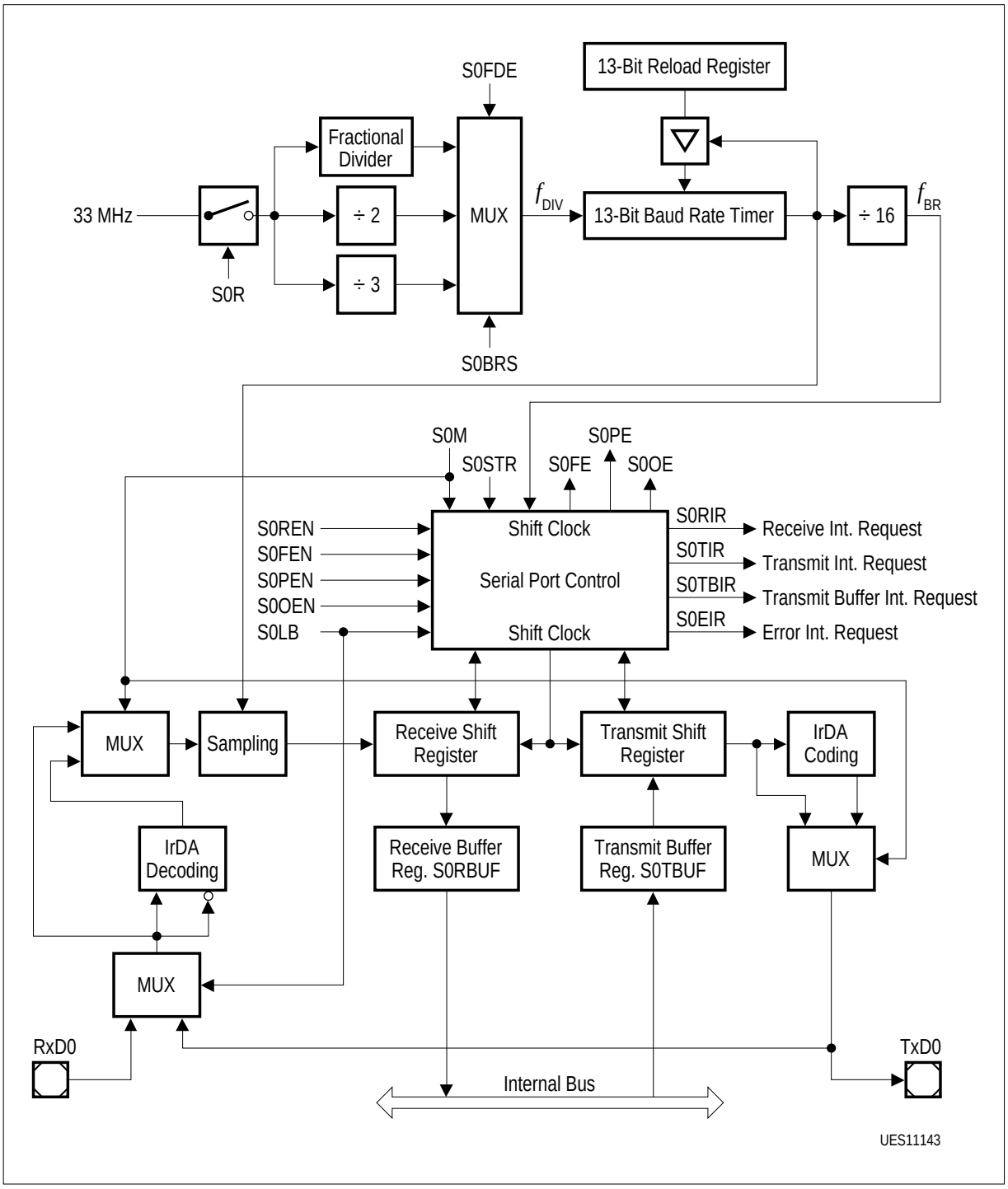


Figure 7-24 Asynchronous Mode of Serial Channel ASC0

7.3.1.1 Asynchronous Data Frames

8-Bit Data Frames

8-bit data frames either consist of 8 data bits D7 … D0 (S0M = '001_B'), or of 7 data bits D6 … D0 plus an automatically generated parity bit (S0M = '011_B'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 8-bit data mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.

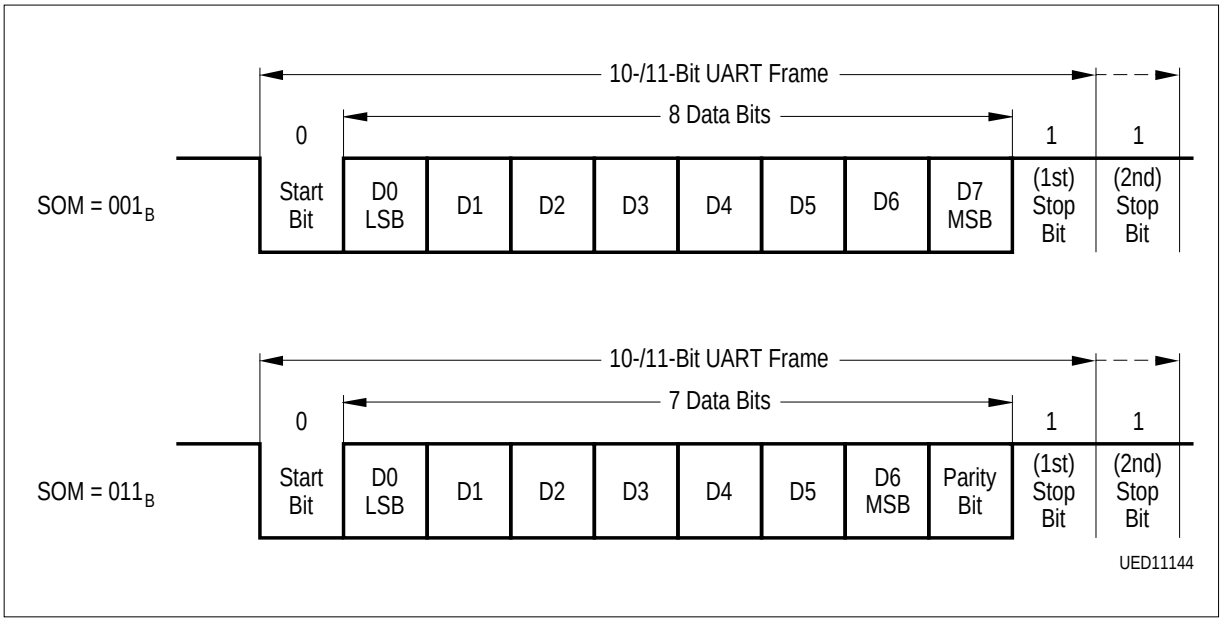


Figure 7-25 Asynchronous 8-Bit Frames

9-Bit Data Frames

9-bit data frames either consist of 9 data bits D8 … D0 (S0M = '100_B'), of 8 data bits D7 … D0 plus an automatically generated parity bit (S0M = '111_B') or of 8 data bits D7 … D0 plus wake-up bit (S0M = '101_B'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 9-bit data and wake-up mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.8.

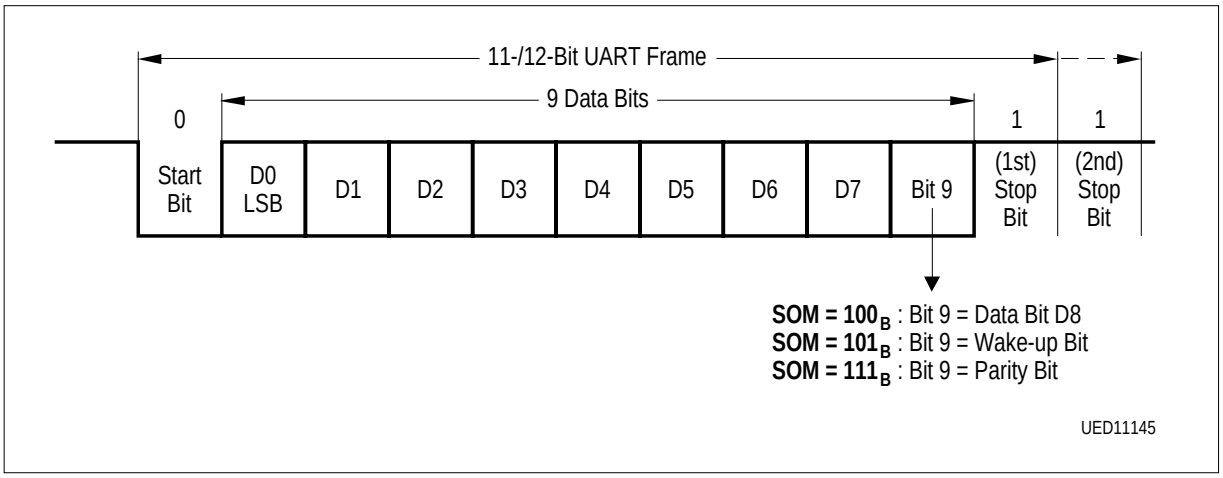


Figure 7-26 Asynchronous 9-Bit Frames

In wake-up mode, received frames are only transferred to the receive buffer register if the 9th bit (the wake-up bit) is ‘1’. If this bit is ‘0’, no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in a multi-processor system:
When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional 9th bit is a ‘1’ for an address byte and a ‘0’ for a data byte, so no slave will be interrupted by a data ‘byte’. An address ‘byte’ will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the 8 LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (e.g. by clearing bit S0M.0), which enables it to also receive further data bytes (having the wake-up bit cleared). The slaves that were not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.

IrDA Frames

The modulation schemes of IrDA is based on standard asynchronous data transmission frames. The asynchronous data format in IrDA mode (S0M = 010_B) is defined as follows:

- 1 start bit/8 data bits/1 stop bit

The coding/decoding of/to the asynchronous data frames is shown in **Figure 7-27**. In general, during the IrDA transmissions, UART frames are encoded into IR frames and vice versa. A low level on the IR frame indicates a ‘LED off’ state. A high level on the IR frame indicates a ‘LED on’ state.

For a ‘0’-bit in the UART frame, a high pulse is generated. For a ‘1’-bit in the UART frame, no pulse is generated. The high pulse starts in the middle of a bit cell and has a fixed width of 3/16 of the bit time. The ASC0 also allows the length of the IrDA high pulse to be programmed. Furthermore, the polarity of the received IrDA pulse can be inverted in IrDA mode. **Figure 7-27** shows the non-inverted IrDA pulse scheme.

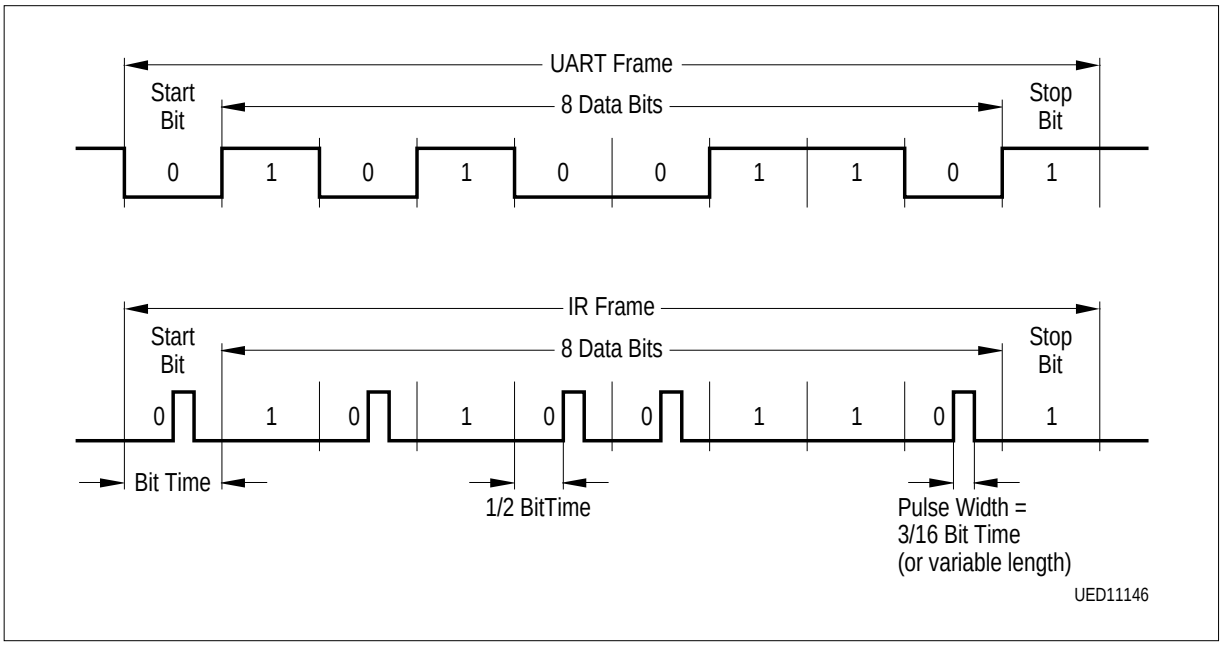


Figure 7-27 IrDA Frame Encoding/Decoding

7.3.1.2 Asynchronous Transmission

Asynchronous transmission begins at the next overflow of the divide-by-16 baud rate timer (transition of the baud rate clock f_{BR}), if bit S0R must be set and data has been loaded into S0TBUF. The transmitted data frame consists of three basic elements:

- the start bit
- the data field (8 or 9 bits, LSB first, including a parity bit, if selected)
- the delimiter (1 or 2 stop bits)

Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request line S0TBIR being activated. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request line S0TIR will be activated before the last bit of a frame is transmitted, e.g. before the first or the second stop bit is shifted out of the transmit shift register.

For alternate data output the transmitter output pin TXD0 must be configured.

7.3.1.3 Asynchronous Reception

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bits S0R and S0REN are set. The receive data input pin RXD0 is sampled at 16 times the rate of the selected baud rate. A majority decision of the 7th, 8th and 9th sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not equal to ‘0’ when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin RXD0. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request line S0RIR is activated after the 9th sample in the last stop bit time slot (as programmed), regardless whether valid stop bits have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input pin.

The receiver input pin RXD0 must be configured for input.

Asynchronous reception is stopped by clearing bit S0REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

Note: In wake-up mode received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is equal to ‘1’. If this bit is equal to ‘0’, no receive interrupt request will be activated and no data transferred.

IrDA Mode

The duration of the IrDA pulse is normally 3/16 of a bit period. The IrDA standard also allows the pulse duration to be independent of the baud rate or bit period. In this case the transmitted pulse always has the width corresponding to the 3/16 pulse width at 115.2 KBaud which is 1.67 µs. Both bit period dependent or fixed IrDA pulse width generation can be selected. The IrDA pulse width mode is selected by bit S0IRPW, which is located in register S0PWM.

In case of a fixed IrDA pulse width generation, the lower 8 bits in register S0PWM are used to adapt the IrDA pulse width to a fixed value of e.g. 1.67 µs. The fixed IrDA pulse width is generated by a programmable timer as shown in **Figure 7-28**.

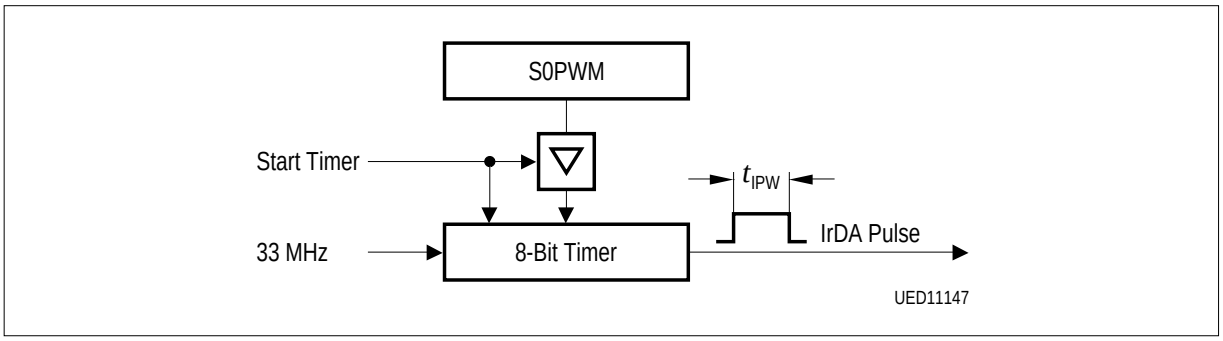


Figure 7-28 Fixed IrDA Pulse Generation

The IrDA pulse width can be calculated according to the formulas given in the following table.

S0PWM	S0IPWM	Formulas	
1 … 255	0	$t_{IPW} = \frac{3}{16 \times 33\ MHz}$	$t_{IPW\ min} = \frac{(S0PWM >> 1)}{33\ MHz}$
	1	$t_{IPW} = \frac{S0PWM}{33\ MHz}$	

The name S0PWM in the formulas represents the contents of the reload register S0PWM (bits S0PW0-7), taken as an unsigned 8-bit integer.

The content of S0PWM further defines the minimum IrDA pulse width ($t_{IPW\ min}$) which is still recognized as a valid IrDA pulse during a receive operation. This function is independent of the selected IrDA pulse width mode (fixed or variable) which is defined by bit S0IRPW in register S0PWM. The minimum IrDA pulse width is calculated by a shift right operation of S0PWM bit 7-0 by one bit divided by the CPU clock (33.33 MHz).

Note: If S0IRPW=0 (fixed IrDA pulse width), SxPWM bit 7-0 must be loaded with a value which assures that $t_{IPW} > t_{IPW\ min}$.

RXD/TXD Data Path Selection in Asynchronous Modes

The data paths for the serial input and output data in asynchronous modes are affected by several control bits in the registers S0CON and S0ABCON, as shown in **Figure 7-29**. The synchronous mode operation is not affected by these data path selection capabilities.

The input signal from RXD passes an inverter which is controlled by bit ABCON_RXINV. The output signal of this inverter is used for the autobaud detection and may bypass the ASC_P logic in the echo mode (controlled by bit ABCON_ABEM). Furthermore, two multiplexers are in the RXD input signal path to provide the loopback mode capability (controlled by bit CON_LB) and the IrDA receive pulse inversion capability (controlled by bit CON_RXDI).

Depending on the asynchronous operating mode (controlled by bitfield CON_M), the ASC output signal or the RXD input signal in echo mode (controlled by bit ABCON_ABEM) is switched to the TXD output by an inverter (controlled by bit ABCON_TXINV).

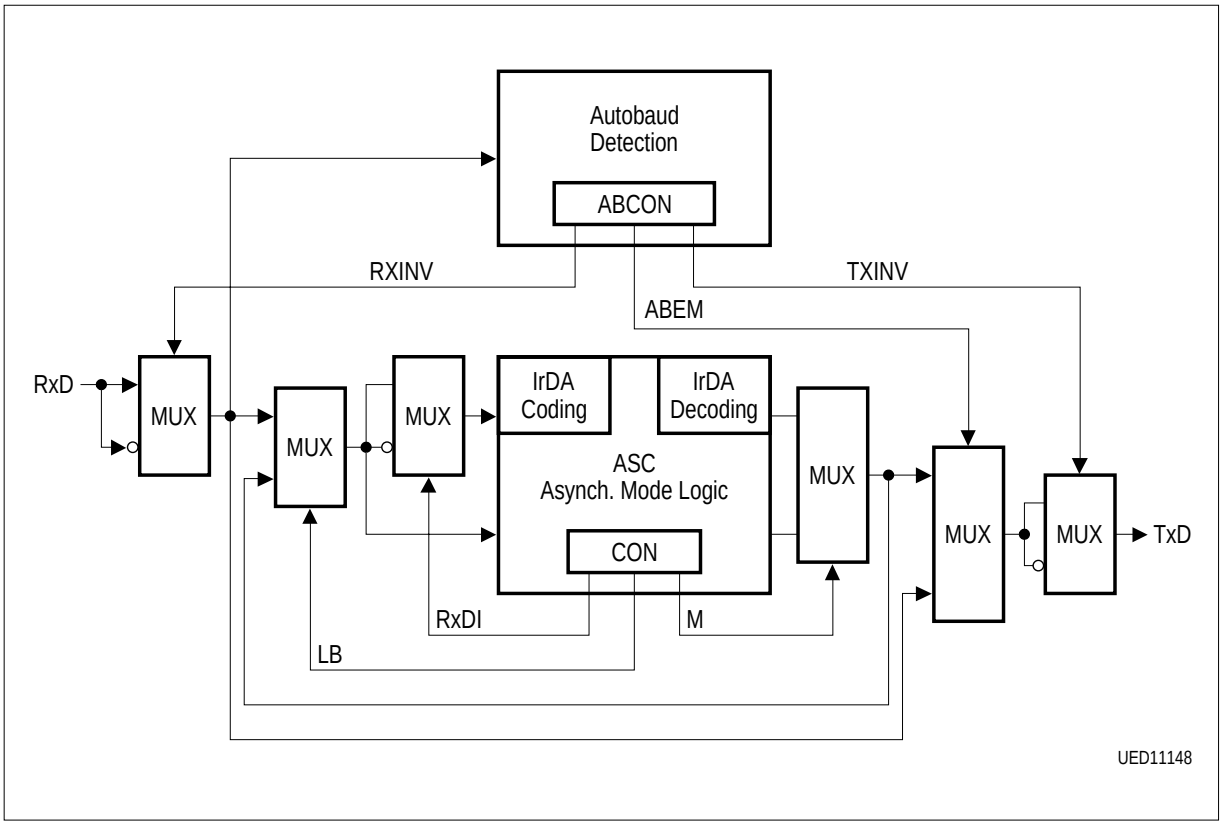


Figure 7-29 RXD/TXD Data Path in Asynchronous Modes

In echo mode, the transmit output signal of the ASC_P3 logic is blocked by the echo mode output multiplexer. **Figure 7-29** also shows that it is not possible to use an IrDA coded receiver input signal for autobaud detection.

7.3.2 Synchronous Operation

Synchronous mode supports half-duplex communication, basically for simple I/O expansion via shift registers. Data is transmitted and received via pin RXD0 while pin TXD0 outputs the shift clock. These signals are alternate functions of port pins. Synchronous mode is selected with S0M = '000_B'.

8 data bits are transmitted or received synchronous to a shift clock, generated by the internal baud rate generator. The shift clock is only active as long as data bits are transmitted or received.

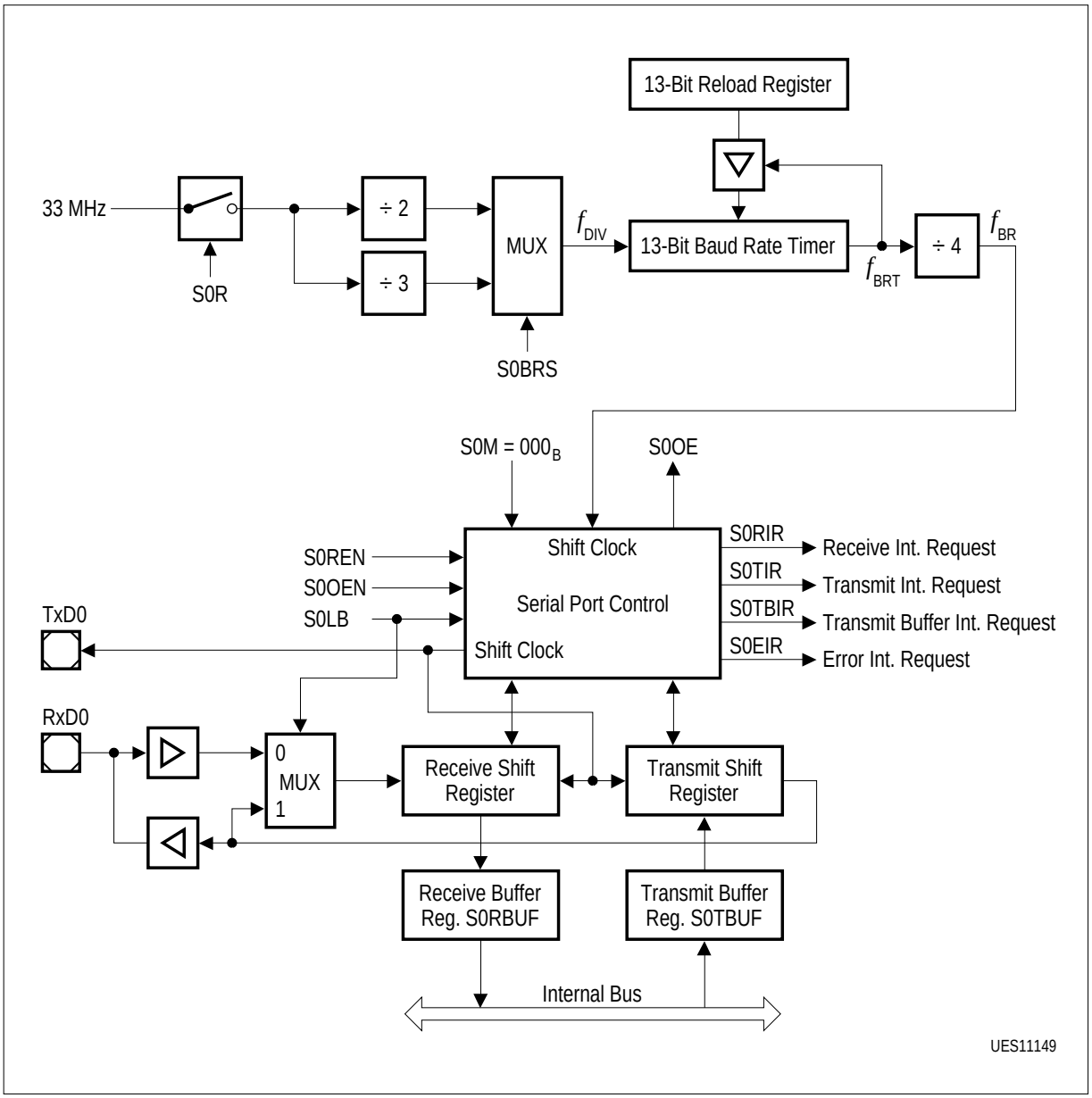


Figure 7-30 Synchronous Mode of Serial Channel ASC0

7.3.2.1 Synchronous Transmission

Synchronous transmission begins within 4 state times after data has been loaded into S0TBUF, provided that S0R is set and S0REN = '0' (half-duplex, no reception). Exception: in loop-back mode (bit S0LB in S0CON set), S0REN must be set to receive the transmitted byte. Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request line S0TBIR being activated. S0TBUF may now be loaded with the next

data, while transmission of the previous one is still going on. The data bits are transmitted synchronous to the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request line S0TIR is activated, and serial data transmission stops.

Pin TXD0 must be configured for alternate data output in order to support the shift clock. Pin RXD0 must also be configured for output during transmission.

7.3.2.2 Synchronous Reception

Synchronous reception is initiated by setting bit S0REN = ‘1’. If bit S0R = 1, the data applied at pin RXD0 is clocked into the receive shift register synchronous to the clock which is output at pin TXD0. After the 8th bit has been moved in, the content of the receive shift register is transferred to the receive data buffer S0RBUF, the receive interrupt request line S0RIR is activated, the receiver enable bit S0REN is reset, and serial data reception stops.

Pin TXD0 must be configured for alternate data output in order to support the shift clock. Pin RXD0 must be configured as an alternate data input.

Synchronous reception is terminated by clearing bit S0REN. A currently received byte is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register at the time the reception of the next byte is complete, both the error interrupt request line S0EIR and the overrun error status flag S0OE will be activated/set, provided the overrun check has been enabled by bit S0OEN.

7.3.2.3 Synchronous Timing

Figure 7-31 shows timing diagrams of the ASC0 synchronous mode data reception and data transmission. In idle state the shift clock is at high level. With the beginning of a synchronous transmission of a data byte, the data is shifted out at pin RXD0 with the falling edge of the shift clock. If a data byte is received through pin RXD0, data is latched with the rising edge of the shift clock.

Between two consecutive receive or transmit data bytes one shift clock cycle (f_{BR}) delay is inserted.

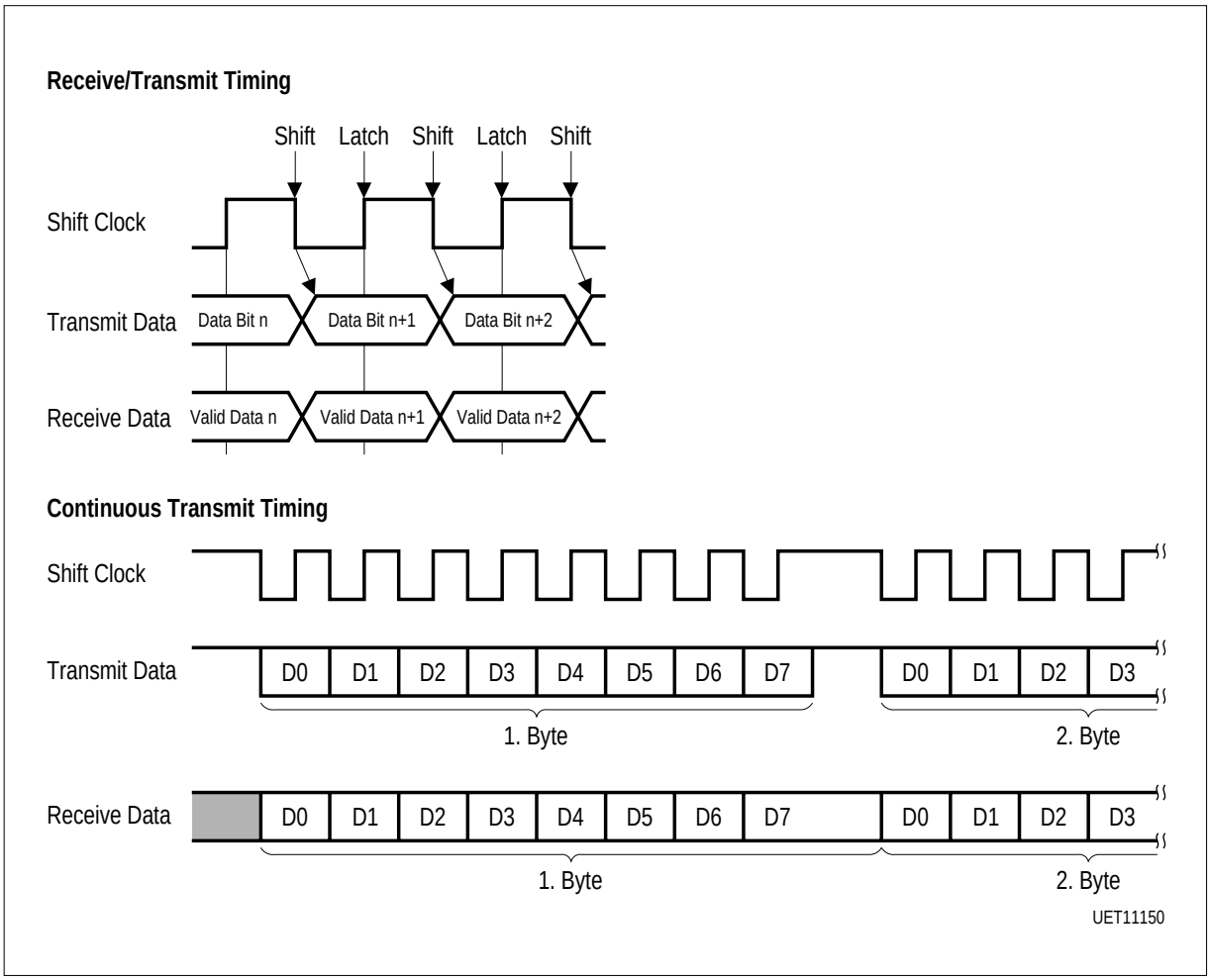


Figure 7-31 ASC0 Synchronous Mode Waveforms

7.3.3 Baud Rate Generation

The serial channel ASC0 has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation to be independent of the GPT timers.

The baud rate generator is clocked with the CPU clock. The baud rate timer is counting downwards and can be started or stopped through the baud rate generator run bit S0R in register S0CON. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock f_{BRT} is again divided according to the operating mode and controlled by the baud rate selection bit S0BRS. If S0BRS = ‘1’, the clock signal is additionally divided to 2/3rd of its frequency (see formulas and table). So the baud rate of ASC0 is determined by the CPU clock, the reload value, the value of S0BRS and the operating mode (asynchronous or synchronous).

Register S0BG is the dual-function Baud Rate Generator/Reload register. Reading S0BG returns the content of the timer (bits 15 … 13 return zero), while writing to S0BG always updates the reload register (bits 15 … 13 are insignificant).

An auto-reload of the timer with the content of the reload register is performed each time S0BG is written to. However, if S0R = ‘0’ at the time the write operation to S0BG is performed, the timer will not be reloaded until the first instruction cycle after S0R = ‘1’. For a clean baud rate initialization, S0BG should only be written if S0R = ‘0’. If S0BG is written with S0R = ‘1’, an unpredicted behavior of the ASC0 may occur during the running of transmit or receive operations.

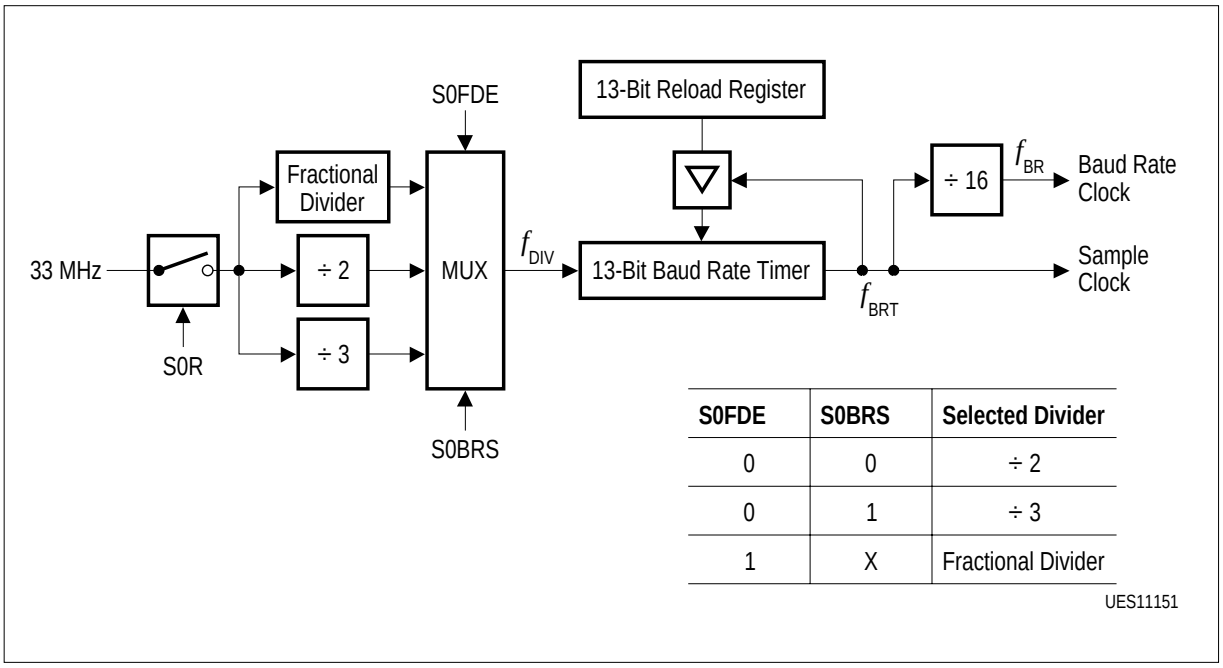
7.3.3.1 Baud Rates in Asynchronous Mode

For asynchronous operation, the baud rate generator provides a clock f_{BRT} with 16 times the rate of the established baud rate. Every received bit is sampled at the 7th, 8th and 9th cycle of this clock. The clock divider circuitry, which generates the input clock for the 13-bit baud rate timer, is extended by a fractional divider circuitry, which allows the adjustment of more accurate baud rates and the extension of the baud rate range.

The baud rate of the baud rate generator depends on the following bits and register values:

- CPU clock
- Selection of the baud rate timer input clock f_{DIV} by bits S0FDE and S0BRS
- If bit S0FDE = 1 (fractional divider): value of register S0FDV
- Value of the 13-bit reload register S0BG

The output clock of the baud rate timer with the reload register is the sample clock in the asynchronous modes of the ASC0. For baud rate calculations, this baud rate clock f_{BR} is derived from the sample clock f_{DIV} by a division of 16.



S0FDE	S0BRS	Selected Divider
0	0	÷ 2
0	1	÷ 3
1	X	Fractional Divider

Figure 7-32 ASC0 Baud Rate Generator Circuitry in Asynchronous Modes

Using the fixed Input Clock Divider

The baud rate for asynchronous operation of serial channel ASC0, when using the fixed input clock divider ratios (S0FDE = 0) and the required reload value for a given baud rate, can be determined by the following formulas:

S0FDE	S0BRS	S0BG	Formula
0	0	0 … 8191	$\text{Baud rate} = \frac{33\ \text{MHz}}{32 \times (\text{S0BG}+1)}$ $\text{S0BG} = \frac{33\ \text{MHz}}{32 \times \text{Baud rate}} - 1$
	1		$\text{Baud rate} = \frac{33\ \text{MHz}}{48 \times (\text{S0BG}+1)}$ $\text{S0BG} = \frac{33\ \text{MHz}}{48 \times \text{Baud rate}} - 1$

S0BG represents the content of the reload register S0BG, taken as an unsigned 13-bit integer.

The maximum baud rate that can be achieved by the asynchronous modes when using the two fixed clock dividers and a CPU clock of 33.33 MHz is 1041.66 KBaud. The table

below lists various commonly used baud rates, together with the required reload values and the deviation errors compared to the intended baud rate.

Baud Rate	S0BRS = '0', f_{MOD} = 33.33 MHz		S0BRS = '1', f_{MOD} = 33.33 MHz	
	Deviation Error	Reload Value	Deviation Error	Reload Value
1041.66 KBaud	–	0000_H	–	–
694.4 KBaud	–	–	–	0000_H
19.2 KBaud	+ 0.4%/– 1.3%	$0035_H/0036_H$	+ 0.5%/– 2.2%	$0023_H/0024_H$
9600 Baud	+ 0.4%/– 0.4%	$006b_H/006C_H$	+ 0.5%/– 0.9%	$0047_H/0048_H$
4800 Baud	+ 0.0%/– 0.4%	$00D8_H/00D9_H$	+ 0.5%/– 0.2%	$008F_H/0090_H$
2400 Baud	+ 0.0%/– 0.2%	$01B1_H/01B2_H$	+ 0.1%/– 0.2%	$0120_H/0121_H$
1200 Baud	+ 0.0%/– 0.1%	$0363_H/0364_H$	+ 0.1%/– 0.0%	$0241_H/0242_H$
110 Baud	+ 0.0%/– 0.0%	$24FC_H/24FD_H$	+ 0.0%/– 0.0%	$18A8_H/18A9_H$

Note: S0FDE must be equal to 0 to achieve the baud rates in the table above. The deviation errors given in the table are rounded. Using a baud rate crystal will provide correct baud rates without deviation errors.

Using the Fractional Divider

When the fractional divider is selected, the input clock f_{DIV} for the baud rate timer is derived from the CPU clock by a programmable divider. If S0FDE = 1, the fractional divider is activated, It divides 33.33 MHz by a fraction of n/512 for any value of n from 0 to 511. If n = 0, the divider ratio is 1 which means that f_{DIV} = 33.33 MHz.

S0FDE	S0BRS	S0BG	S0FDV	Formula
1	–	1 … 8191	1 … 511	$\text{Baud rate} = \frac{\text{S0FDV}}{512} \times \frac{33\text{MHz}}{16 \times (\text{S0BG}+1)}$
			0	$\text{Baud rate} = \frac{33\text{MHz}}{16 \times (\text{S0BG}+1)}$

S0BG represents the content of the reload register S0BG, taken as an unsigned 13-bit integer.
S0FDV represents the content of the fractional divider register taken as an unsigned 9-bit integer.

7.3.3.2 Baud Rates in Synchronous Mode

For synchronous operation, the baud rate generator provides a clock with 4 times the rate of the established baud rate (see **Figure 7-33**).

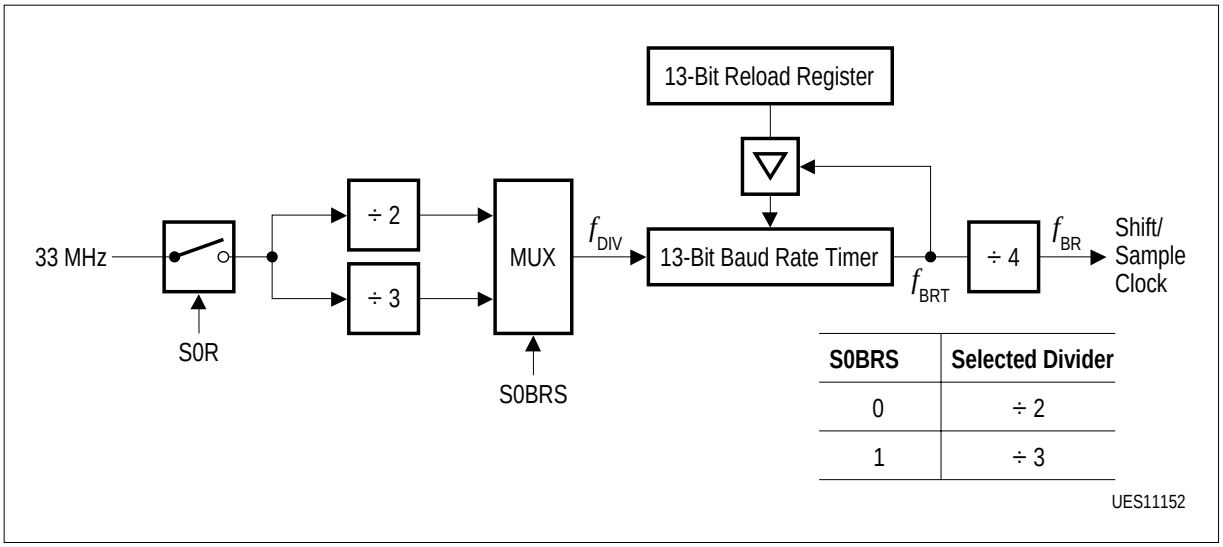


Figure 7-33 ASC0 Baud Rate Generator Circuitry in Synchronous Mode

The baud rate for synchronous operation of serial channel ASC0 can be determined by the formulas as shown in the following table.

S0BRS	S0BG	Formula
0	0 … 8191	$\text{Baud rate} = \frac{33\ \text{MHz}}{8 \times (\text{S0BG}+1)}$ $\quad \text{S0BG} = \frac{33\ \text{MHz}}{8 \times \text{Baud rate}} - 1$
1		$\text{Baud rate} = \frac{33\ \text{MHz}}{12 \times (\text{S0BG}+1)}$ $\quad \text{S0BG} = \frac{33\text{MHz}}{12 \times \text{Baud rate}} - 1$

S0BG represents the content of the reload register, taken as unsigned 13-bit integers.

The maximum baud rate that can be achieved in synchronous mode when using a CPU clock of 33.33 MHz is 4.166 MBaud.

7.3.4 Autobaud Detection

The autobaud detection unit provides an ability to recognize the mode and the baud rate of an asynchronous input signal at RXD. Generally, the baud rates to be recognized should be known by the application. With this knowledge, a set of nine baud rates can always be detected. The autobaud detection unit is not designed to calculate a baud rate of an unknown asynchronous frame.

Figure 7-34 shows how the autobaud detection unit of the ASC is integrated into its asynchronous mode configuration. The RXD data line is an input of the autobaud detection unit. The clock f_{DIV}, which is generated by the fractional divider, is used by the autobaud detection unit as a time base. After successful recognition of the baud rate and asynchronous operating mode of the RXD data input signal, bits in the CON register and the value of the BG register in the baud rate timer are set to the appropriate values, and the ASC_P3 can start immediately with the reception of serial input data.

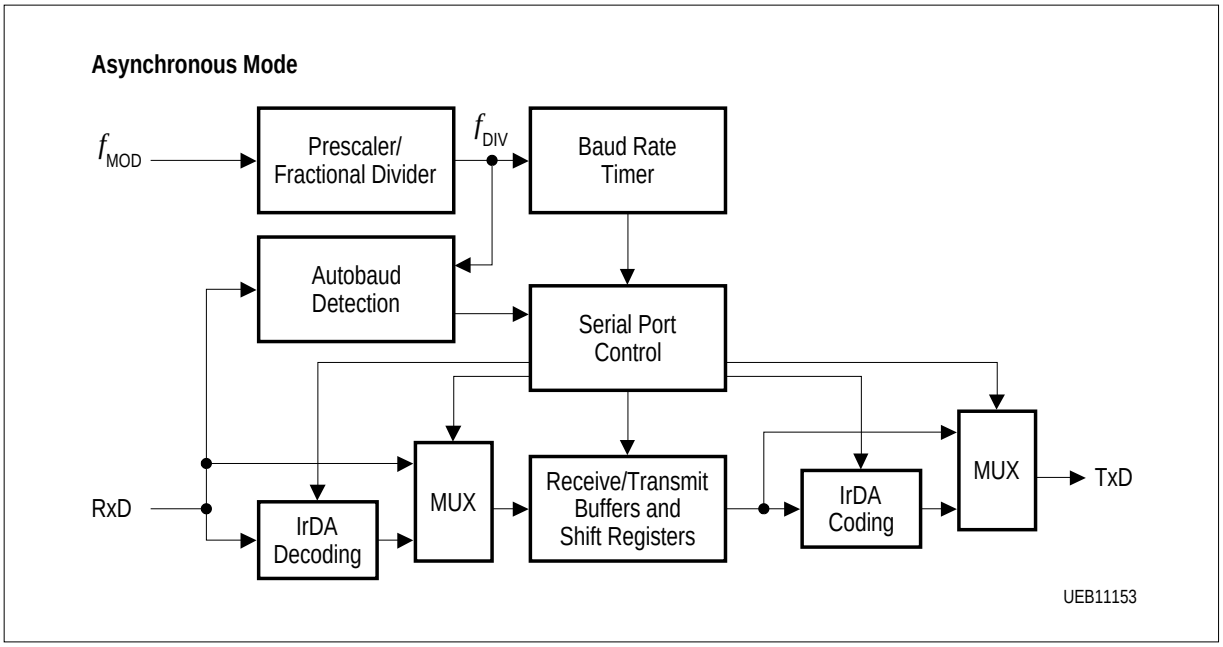


Figure 7-34 ASC_P3 Asynchronous Mode Block Diagram

The following sequence must be generally executed to start the operation of the autobaud detection unit:

- Definition of the baud rates to be detected: standard or non-standard baud rates
- Programming of the Prescaler/Fractional Divider to select a specific value of f_{DIV}
- Starting the Prescaler/Fractional Divider (setting CON_R)
- Preparing the interrupt system of the CPU
- Enabling the autobaud detection (setting ABCON_ EN and the interrupt enable bits in ABCON for interrupt generation, if required)
- Polling interrupt request flag or waiting for the autobaud detection interrupt

7.3.4.1 Serial Frames for Autobaud Detection

The autobaud detection of the ASC_P3 is based on the serial reception of a specific two-byte serial frame. This serial frame is build up by the two ASCII bytes ‘at’ or ‘AT’ (‘aT’ or ‘At’ are not allowed). Both byte combinations can be detected in five types of asynchronous frames. **Figure 7-35** and **Figure 7-36** show the serial frames which are least detected.

Note: Some other two-byte combinations will also be defined.

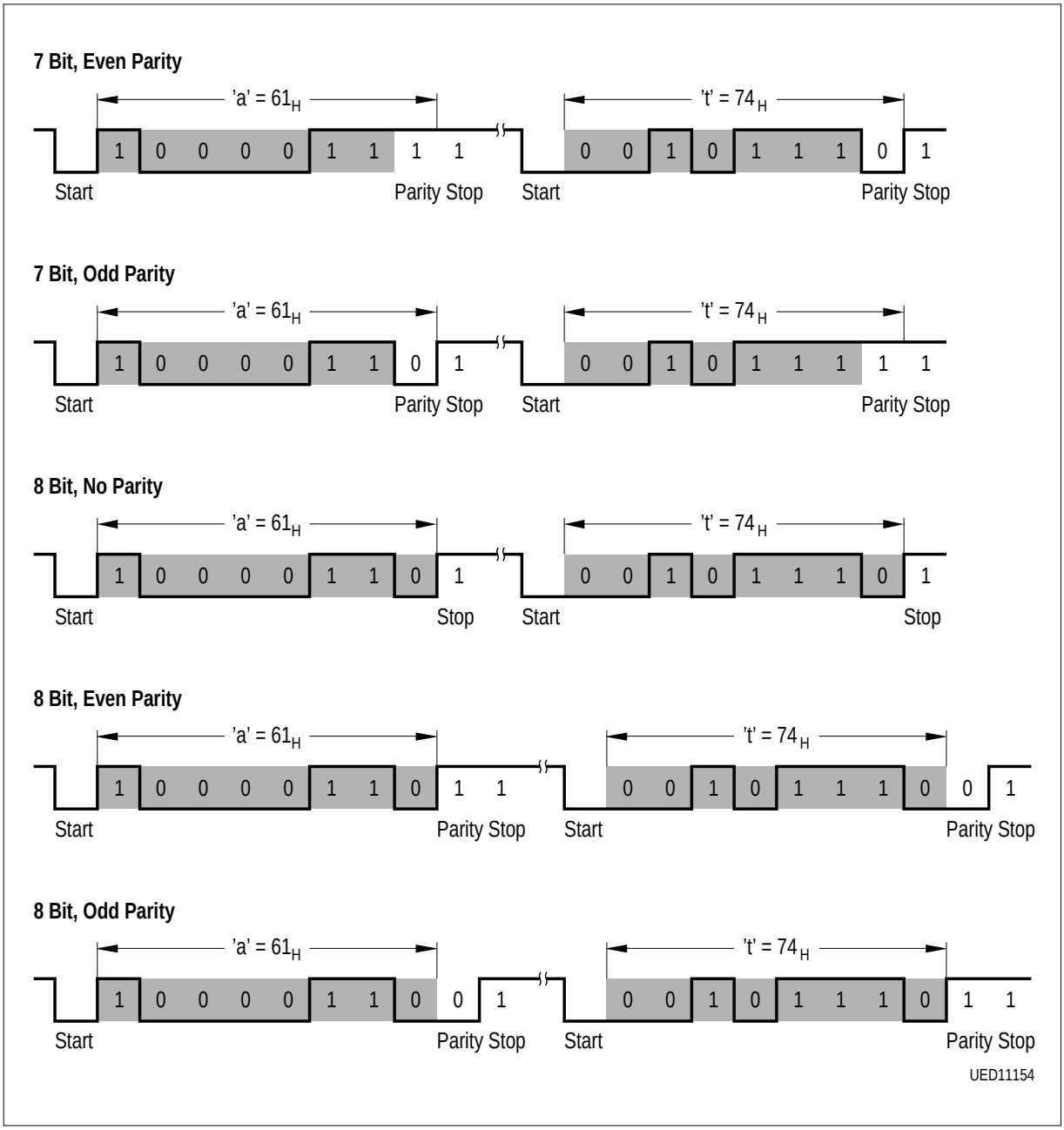


Figure 7-35 Two-Byte Serial Frames with ASCII 'at'

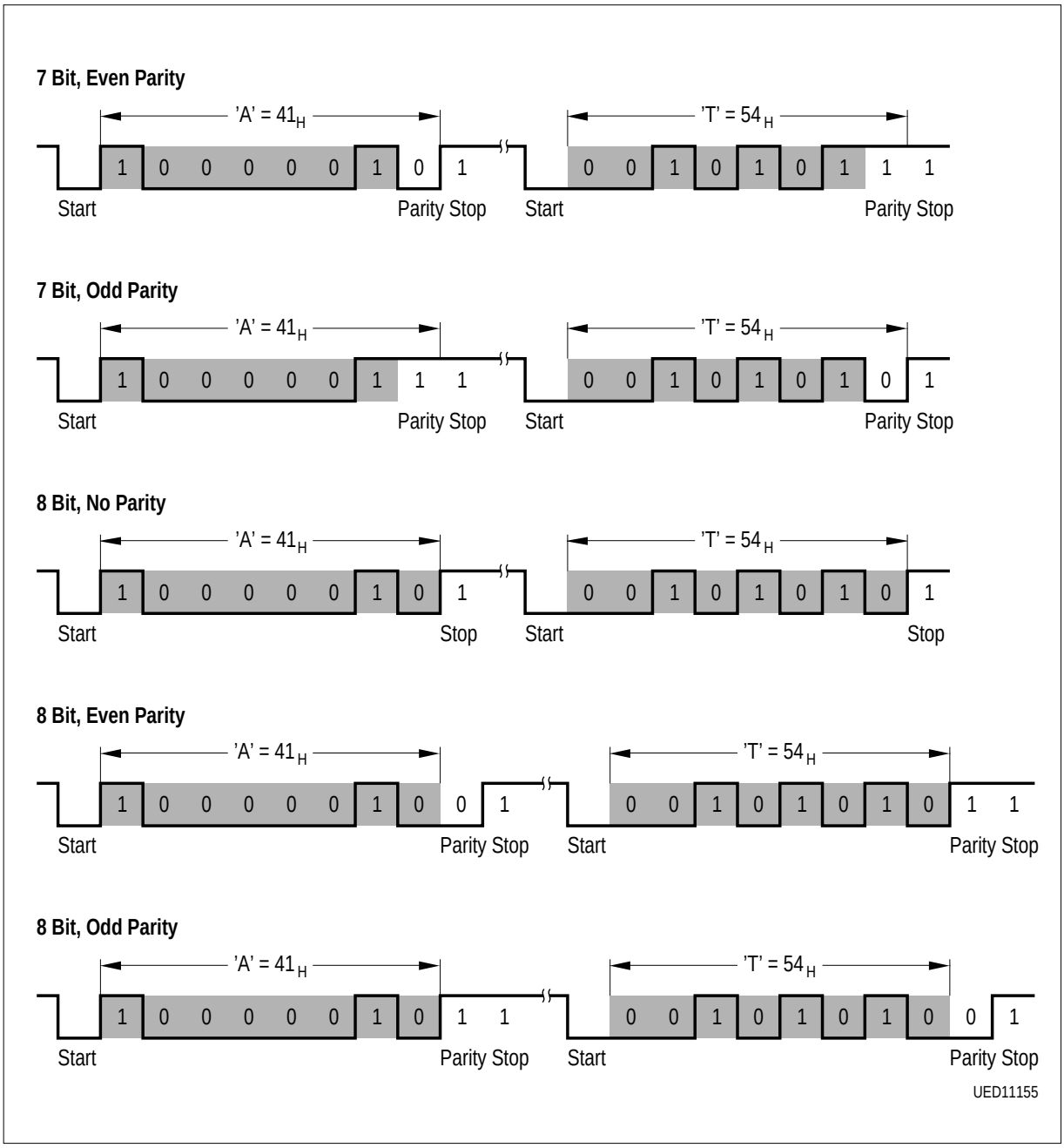


Figure 7-36 Two-Byte Serial Frames with ASCII 'AT'

7.3.4.2 Baud Rate Selection and Calculation

The autobaud detection requires some calculations concerning the programming of the baud rate generator and the baud rates to be detected. Two steps must be considered:

- Defining the baud rate(s) to be detected
- Programming of the baud rate timer prescaler - setup of the clock rate of f_{DIV}

In general, the baud rate generator of the ASC in asynchronous mode is built up of two parts:

- the clock prescaler part which derives f_{DIV} from f_{MOD}
- the baud rate timer part which generates the sample clock f_{BRT} and the baud rate clock f_{BR}

Prior to an autobaud detection, the prescaler part has to be set up by the CPU while the baud rate timer (register BG) is automatically initialized with a 13-bit value (BR_VALUE) after a successfull autobaud detection. For the subsequent calculations, the fractional divider is used (CON_FDE = 1).

Note: It is also possible to use the fixed divide-by-2 or divide-by-3 prescaler. But the fractional divider allows for a more precise adaption of f_{DIV} to the required value.

Standard Baud Rates

For standard baud rate detection the baud rates as shown in **Table 7-20** can be detected. Therefore, the output frequency f_{DIV} of the ASC baud rate generator must be set to a frequency derived from the system, clocked (33 MHz) in a way that it is equal to 11.0592 MHz. The value to be written into register FDV is the nearest integer value which is calculated according the following formula:

$$FDV = \frac{512 \times 11.0592\ MHz}{33\ MHz}$$

Table 7-20 defines the nine standard baud rates (Br0 - Br8) which can be detected for f_{DIV} = 11.0592 MHz.

Table 7-20 Autobaud Detection using Standard Baud Rates (f_{DIV} = 11.0592 MHz)

Baud Rate Numbering	Detectable Standard Baud Rate	Divide Factor d_f	BG is Loaded after Detection with Value
Br0	230.400 kBaud	48	2 = 002_H
Br1	115.200 kBaud	96	5 = 005_H
Br2	57.600 kBaud	192	11 = $00B_H$
Br3	38.400 kBaud	288	17 = 011_H
Br4	19.200 kBaud	576	35 = 023_H
Br5	9600 Baud	1152	71 = 047_H
Br6	4800 Baud	2304	143 = $08F_H$
Br7	2400 Baud	4608	287 = $11F_H$
Br8	1200 Baud	9216	575 = $23F_H$

According to **Table 7-20** a baud rate of 9600 Baud is achieved when register BG is loaded with a value of 047_H, assuming that f_{DIV} has been set to 11.0592 MHz.
Table 7-20 also lists a divide factor d_f which is defined with the following formula:

$$\text{baud rate} = \frac{f_{DIV}}{d_f}$$

This divide factor d_f defines a fixed relationship between the prescaler output frequency f_{DIV} and the baud rate to be detected during the autobaud detection operation. This means, that changing f_{DIV} results in a totally different baud rate table in terms of baud rate values. For the baud rates to be detected, the following relations are always valid:

– Br0 = $f_{DIV}/48_D$, Br1 = $f_{DIV}/96_D$, … up to Br8 = $f_{DIV}/9216_D$,

A requirement for detecting standard baud rates up to 230.400 kBaud, is the f_{DIV} minimum value of 11.0592 MHz. With the value FD_VALUE in register FDV, the fractional divider f_{DIV} is adapted to the system clock frequency 33 MHz. **Table 7-21** defines the deviation of the standard baud rates when using autobaud detection depending on the system f_{MOD}.

Table 7-21 Standard Baud Rates - Deviations and Errors for Autobaud Detection

f_{MOD}	FDV	Error in f_{DIV}
33 MHz	172	+ 0.24%

Note: If the deviation of the baud rate after autobaud detection is too high, the baud rate generator (fractional divider FDV and reload register BG) can be reprogrammed if required to get a more precise baud rate with less error.

Non-Standard Baud Rates

Due to the relationship between Br0 to Br8 in **Table 7-20** concerning the divide factor d_f other baud rates than the standard ones can also be selected. For example, if a baud rate of 50 kBaud has to be detected, Br2 is e.g. defined as baud rate for the 50 kBaud detection. This further results in:

– f_{DIV} = 50 kBaud × d_f@Br2 = 50 kBaud × 192 = 9.6 MHz

Therefore, depending on the system clock frequency, the value of the fractional divider (register FDV must be set according the formula in this example:

$$FDV = \frac{512 \times f_{DIV}}{33\ MHz} \quad \text{with } f_{DIV} = 9.6\ MHz$$

Using this selection (f_{DIV} = 9.6 MHz), the detectable baud rates start at 200 kBaud (Br0) down to 1042 Baud (Br8). **Table 7-22** shows the baud rate table for this example.

Table 7-22 Autobaud Detection using Non-Standard Baud Rates (f_{DIV} = 9.6 MHz)

Baud Rate Numbering	Detectable Non-Standard Baud Rates	Divide Factor d_f	BG is Loaded after Detection with Value
Br0	200.000 kBaud	48	2 = 002_H
Br1	100.000 kBaud	96	5 = 005_H
Br2	50 kBaud	192	11 = $00B_H$
Br3	33.333 kBaud	288	17 = 011_H
Br4	16.667 kBaud	576	35 = 023_H
Br5	8333 Baud	1152	71 = 047_H
Br6	4167 Baud	2304	143 = $08F_H$
Br7	2083 Baud	4608	287 = $11F_H$
Br8	1047 Baud	9216	575 = $23F_H$

7.3.4.3 Overwriting Registers on Successful Autobaud Detection

With a successful autobaud detection, some bits in register CON and BG are automatically set to a value which corresponds to the mode and baud rate of the detected serial frame conditions. In control register CON the mode control bits CON_M

and the parity select bit CON_ODD are overwritten. Register BG is loaded with the 13-bit reload value for the baud rate timer.

Table 7-23 Autobaud Detection Overwrite Values for the CON Register

Detected Parameters		CON_M	CON_ODD	BG_BR_VALUE
Operating Mode	7 bit, even parity 7 bit, odd parity 8 bit, even parity 8 bit, odd parity 8 bit, no parity	0 1 1 0 1 1 1 1 1 1 1 1 0 0 1	0 1 0 1 0	–
Baud rate	Br0 Br1 Br2 Br3 Br4 Br5 Br6 Br7 Br8	–	–	2 = 002_H 5 = 005_H 11 = $00B_H$ 17 = 011_H 35 = 023_H 71 = 047_H 143 = $08F_H$ 287 = $11F_H$ 575 = $23F_H$

7.3.5 ASC Hardware Error Detection Capabilities

To improve the safety of serial data exchange, the serial channel ASC0 provides an error interrupt request flag, which indicates the presence of an error, and three (selectable) error status flags in register S0CON, which indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request line S0EIR will be activated simultaneously with the receive interrupt request line S0RIR if one or more of the following conditions are met:

- The framing error detection enable bit S0FEN is set and any of the expected stop bits are not high, the framing error flag S0FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).
- If the parity error detection enable bit S0PEN is set in the mode where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag S0PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).
- If the overrun error detection enable bit S0OEN is set and the last character received was not read out of the receive buffer by software or PEC transfer at the time the reception of a new frame is complete, the overrun error flag S0OE is set, indicating that the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

7.3.6 Interrupts

Six interrupt sources are provided for serial channel ASC0. Line S0TIC indicates a transmit interrupt, S0TBIC indicates a transmit buffer interrupt, S0RIC indicates a receive interrupt and S0EIC indicates an error interrupt of the serial channel. The autobaud detection unit provides two additional interrupts, the ABSTIR start of autobaud operation interrupt and the ABDETIR autobaud detected interrupt. The interrupt output lines S0TBIR, S0TIR, S0RIR, and S0EIR are activated (active state) for two periods of the module clock f_{MOD} (33.33 MHz).

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags S0FE, S0PE, and S0OE which are located in control register S0CON.

Note: In contrary to the error interrupt request line S0EIR, the error status flags S0FE/ S0PE/S0OE are not reset automatically but must be cleared by software.

For normal operation (e.g. besides the error interrupt) the ASC0 provides three interrupt requests to control data exchange via this serial channel:

- S0TBIR is activated when data is moved from S0TBUF to the transmit shift register.
- S0TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- S0RIR is activated when the received frame is moved to S0RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This has its advantages for the servicing software.

For single transfers it is sufficient to use the transmitter interrupt (S0TIR), which indicates that the previously loaded data, except for the last bit of an asynchronous frame, has been transmitted.

For multiple back-to-back transfers it is necessary to wait to load the last piece of data until the last bit of the previous frame has been transmitted. In asynchronous mode this leaves just one bit-time for the handler to respond to the transmitter interrupt request, in synchronous mode it is impossible.

Using the transmit buffer interrupt (S0TBIR) to reload transmitted data gives enough time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

The ABSTIR start of autobaud operation interrupt is generated whenever the autobaud detection unit is enabled (ABEN, ABDETEN and ABSTEN set), and a start bit has been detected at RXD. In this case ABSTIR is generated during autobaud detection whenever a start bit is detected.

The ABDETIR autobaud detected interrupt is always generated after recognition of the second character of the two-byte frame, after a successful autobaud detection. If ABCON_FCDETEN is set the ABDETIR autobaud detected interrupt is also generated after the recognition of the first character of the two-byte frame.

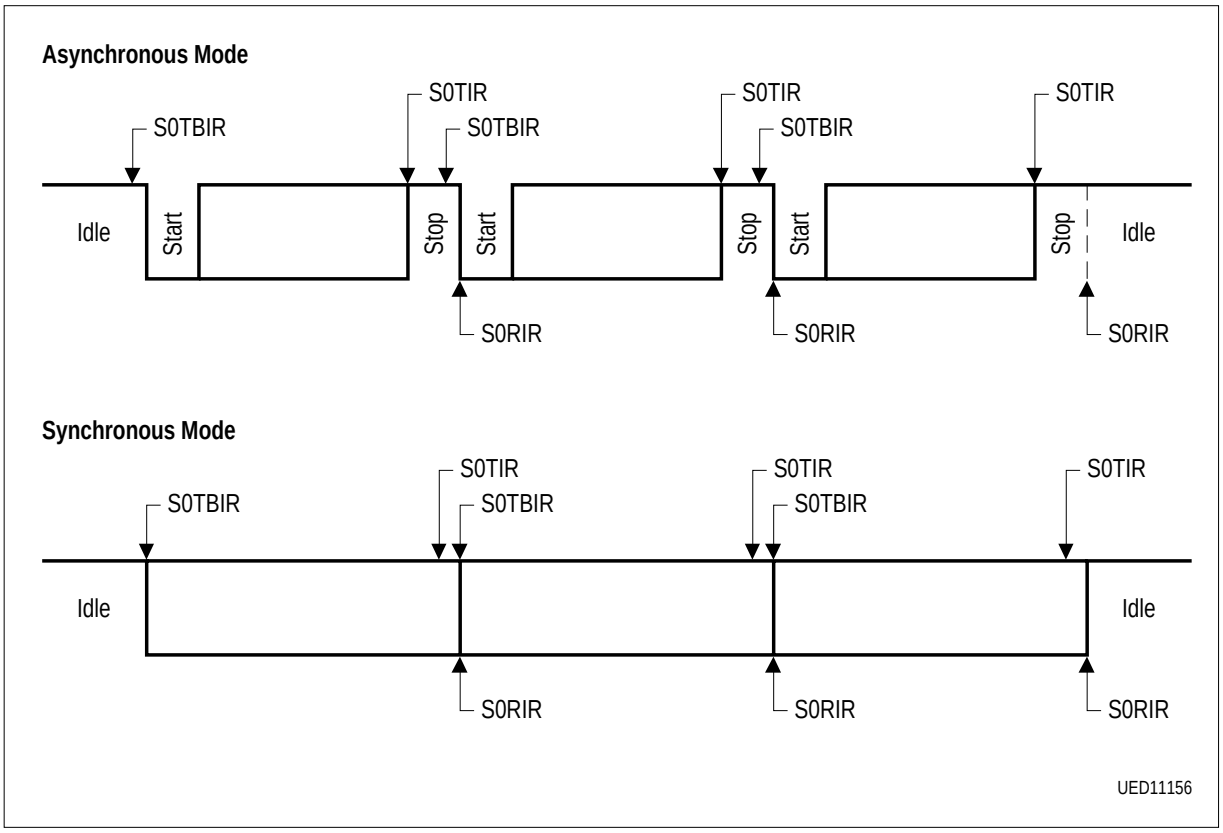


Figure 7-37 ASC0 Interrupt Generation

As shown in Figure 7-37, S0TBIR is an early trigger for the reload routine, while S0TIR indicates the completed transmission. Software using handshake therefore should rely on S0TIR at the end of a data block to make sure that all data has really been transmitted.

7.3.7 Register Description

The operating mode of the serial channel ASC0 is controlled by its control register S0CON. This register contains control bits for mode and error check selection, and status flags for error identification.

S0CON
Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**LB**	**BRS**	**ODD**	**FDE**	**OE**	**FE**	**PE**	**OEN**	**FEN**	**PEN / RXDI**	**REN**	**STP**	**M**		

Field	Bits	Type	Value	Description
M	2-0	rwh		**Mode Selection**
			0 0 0	8-bit data sync. operation
			0 0 1	8-bit data async. operation
			0 1 0	IrDA mode, 8-bit data async. operation
			0 1 1	7-bit data + parity async. operation
			1 0 0	9-bit data async. operation
			1 0 1	8-bit data + wake up bit async. operation
			1 1 0	Reserved. Do not use this combination!
			1 1 1	8-bit data + parity async. operation
				Bits are set/cleared by hardware after a successful autobaud detection operation.
STP	3	rw		**Number of Stop Bit Selection**
			0	One stop bit
			1	Two stop bits
REN	4	rwh		**Receiver Enable Control**
			0	Receiver disabled
			1	Receiver enabled
				Bit can be affected during autobaud detection operation when bit ABEN_AUREN is set.
				Bit is reset by hardware after reception of byte in synchronous mode.
PEN	5	rw		**Parity Check Enable/**
RXDI				**IrDA Input Inverter Enable**
				All asynchronous modes without IrDA mode:
			0	Ignore parity
			1	Check parity
			0	Only in IrDA mode (M = 010):
			1	RXD input is not inverted
				RXD input is inverted
FEN	6	rw		**Framing Check Enable**
				(async. modes only)
			0	Ignore framing errors
			1	Check framing errors
OEN	7	rw		**Overrun Check Enable**
			0	Ignore overrun errors
			1	Check overrun errors

Field	Bits	Type	Value	Description
PE	8	rwh	–	**Parity Error Flag** Set by hardware on a parity error (PEN = ‘1’). Must be reset by software.
FE	9	rwh	–	**Framing Error Flag** Set by hardware on a framing error (FEN = ‘1’). Must be reset by software.
OE	10	rwh	–	**Overrun Error Flag** Set by hardware on an overrun error (OEN = ‘1’). Must be reset by software.
FDE	11	rw	 0 1	**Fractional Divider Enable** Fractional divider disabled Fractional divider is enabled and used as prescaler for baud rate timer (bit BRS is don’t care).
ODD	12	rwh	 0 1	**Parity Selection** Even parity selected (parity bit set on odd number of ‘1’s in data) Odd parity selected (parity bit set on even number of ‘1’s in data) Bit is be set/cleared by hardware after a successful autobaud detection operation.
BRS	13	rw	 0 1	**Baud rate Selection** Baud rate timer prescaler divide-by-2 selected Baud rate timer prescaler divide-by-3 selected BRS is don’t care if FDE = 1 (fractional divider enabled)
LB	14	rw	 0 1	**Loop-back Mode Enable** Loop-back mode disabled Loop-back mode enabled
R	15	rw	 0 1	**Baud rate Generator Run Control** Baud rate generator disabled (ASC_P inactive) Baud rate generator enabled BG should only be written if R = ‘0’.

The autobaud control register ABCON is used to control the autobaud detection operation. It contains its general enable bit, the interrupt enable control bits, and data path control bits

S0ABCON
Autobaud Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**RX INV**	**TX INV**	**ABEM**		**0**	**0**	**0**	**FC DET EN**	**AB DET EN**	**ABS T EN**	**AUR EN**	**AB EN**

Field	Bits	Type	Value	Description
ABEN	0	rwh		**Autobaud Detection Enable**
			0	Autobaud detection is disabled
			1	Autobaud detection is enabled ABEN is reset by hardware after a successful autobaud detection; (with the stop bit detection of the second character). Resetting ABEN by software if it was set aborts the autobaud detection.
AUREN	1	rw		**Automatic Autobaud Control of CON_REN**
			0	CON_REN is not affected during autobaud detection
			1	CON_REN is cleared (receiver disabled) when ABEN and AUREN are set together. CON_REN is set (receiver enabled) after a successful autobaud detection (with the stop bit detection of the second character).
ABSTEN	2	rw		**Start of Autobaud Detection Interrupt Enable**
			0	Start of autobaud detection interrupt disabled
			1	Start of autobaud detection interrupt enabled
ABDETEN	3	rw		**Autobaud Detection Interrupt Enable**
			0	Autobaud detection interrupt disabled
			1	Autobaud detection interrupt enabled

Field	Bits	Type	Value	Description
FCDETEN	4	rw		**First Character of Two-Byte Frame Detected Enable**
			0	Autobaud detection interrupt ABDETIR becomes active after the two-byte frame recognition
			1	Autobaud detection interrupt ABDETIR becomes active after detection of the first and second byte of the two-byte frame.
ABEM	8-9	rw		**Autobaud Echo Mode Enable** In echo mode the serial data at RXD is switched to TXD output.
			0 0	Echo mode disabled
			0 1	Echo mode is enabled during autobaud detection
			1 0	Echo mode is always enabled
			1 1	reserved
TXINV	10	rw		**Transmit Inverter Enable**
			0	Transmit inverter disabled
			1	Transmit inverter enabled
RXINV	11	rw		**Receive Inverter Enable**
			0	Receive inverter disabled
			1	Receive inverter enabled

The autobaud status register ABSTAT indicates the status of the autobaud detection operation.

S0ABSTAT
Autobaud Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**DET WAIT**	**SCC DET**	**SCS DET**	**FCC DET**	**FCS DET**

Field	Bits	Type	Value	Description
FCSDET	0	rwh		**First Character with Small Letter Detected**
			0	no small ‘a’ character detected
			1	small ‘a’ character detected
				Bit is cleared by hardware when ABCON_ABEN is set or if FCCDET or SCSDET or SCCDET is set. Bit can be also cleared by software.
FCCDET	1	rwh		**First Character with Capital Letter Detected**
			0	no capital ‘A’ character detected
			1	capital ‘A’ character detected
				Bit is cleared by hardware when ABCON_ABEN is set or if FCSDET or SCSDET or SCCDET is set. Bit can be also cleared by software.
SCSDET	2	rwh		**Second Character with Small Letter Detected**
			0	no small ‘t’ character detected
			1	small ‘t’ character detected
				Bit is cleared by hardware when ABCON_ABEN is set or if FCSDET or FCCDET or SCCDET is set. Bit can be also cleared by software.

Field	Bits	Type	Value	Description
SCCDET	3	rwh		**Second Character with Capital Letter Detected**
			0	no capital ‘T’ character detected
			1	capital ‘T’ character detected Bit is cleared by hardware when ABCON_ABEN is set or if FCSDET or FCCDET or SCSDET is set. Bit can be also cleared by software.
DETWAIT	4	rwh		**Autobaud Detection is Waiting**
			0	Either character ‘a’, ‘A’, ‘t’, or ‘T’ has been detected.
			1	The autobaud detection unit waits for the first ‘a’ or ‘A’ Bit is cleared when either FCSDET or FCCDET is set (‘a’ or ‘A’ detected). Bit can be also cleared by software. DETWAIT is set by hardware when ABCON_ABEN is set.

Note: SCSDET or SCCDET are set when the second character has been recognized. CON_ABEN is reset, and ABDETIR set, after SCSDET or SCCDET have seen set.

The baud rate timer reload register BG contains the 13-bit reload value for the baud rate timer in asynchronous and sychronous mode.

S0BG
Baud Rate Timer/Reload Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**BR_VALUE**												

Field	Bits	Type	Value	Description
BR_VALUE	12-0	rw	all	**Baud Rate Timer/Reload Register Value** Reading BG returns the 13-bit content of the baud rate timer (bits 15 … 13 return 0); writing BG loads the baud rate timer reload register (bits 15 … 13 are don't care). BG should only be written if CON_R = '0'.

The fractional divider register FDV contains the 9-bit divider value for the fractional divider (asynchronous mode only). It is also used for reference clock generation of the autobaud detection unit.

S0FDV
Fractional Divider Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**FD_VALUE**								

Field	Bits	Type	Value	Description
FD_VALUE	8-0	rw	all	**Fractional Divider Register Value** FDV contains the 9-bit value n of the fractional divider which defines the fractional divider ratio: n/512 n = 0-511). With n = 0, the fractional divider is switched off (input = output frequency, $f_{DIV} = f_{MOD}$).

The IrDA pulse mode and width register PMW contains the 8-bit IrDA pulse width value and the IrDA pulse width mode select bit. This register is only required in the IrDA operating mode.

S0PMW
IrDA Pulse Mode/Width Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**IRPW**	**PW_VALUE**							

Field	Bits	Type	Value	Description
PW_VALUE	7-0	rw	all	**IrDA Pulse Width Value** PW_VALUE is the 8-bit value n, which defines the variable pulse width of an IrDA pulse. Depending on the ASC_P input frequency f_{MOD}, this value can be used to adjust the IrDA pulse width to value which is not equal 3/16 bit time (e.g. 1.6 µs).
IRPW	8	rw	 0 1	**IrDA Pulse Width Mode Control** IrDA pulse width is 3/16 of the bit time IrDA pulse width is defined by PW_VALUE

The transmitter buffer register TBUF contains the transmit data value in asynchronous and synchronous modes.

S0TBUF
Transmitter Buffer Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	TD_VALUE								

Field	Bits	Type	Value	Description
TD_VALUE	8-0	rw	all	**Transmit Data Register Value** TBUF contains the data to be transmitted in asynchronous and synchronous operating mode of the ASC_P. Data transmission is double buffered, therefore, a new value can be written to TBUF before the transmission of the previous value is complete.

The receiver buffer register RBUF contains the receive data value in asynchronous and synchronous modes.

S0RBUF
Transmitter Buffer Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	RD_VALUE								

Field	Bits	Type	Value	Description
RD_VALUE	8-0	rw	all	**Receive Data Register Value** RBUF contains the received data bits and, depending on the selected mode, the parity bit in asynchronous and synchronous operating mode of the ASC_P. In asynchronous operating mode with M = 011 (7-bit data + parity), the received parity bit is written into RD7. In asynchronous operating mode with M = 111 (8-bit data + parity) the received parity bit is written into RD8.

7.4 High Speed Synchronous Serial Interface

- Master and slave mode operation
  - Full-duplex or half-duplex operation
- Flexible data format
  - Programmable number of data bits: 2 to 16 bit
  - Programmable shift direction: LSB or MSB shift first
  - Programmable clock polarity: idle low or high state for the shift clock
  - Programmable clock/data phase: data shift with leading or trailing edge of SCLK
- Baud rate generation from 12.5 MBaud to 190.7 Baud (@ 25 MHz module clock)
- Interrupt generation
  - on a transmitter empty condition
  - on a receiver full condition
  - on an error condition (receive, phase, baud rate, transmit error)
- Three pin interface
  - Flexible SSC0 pin configuration

The High Speed Synchronous Serial Interface SSC0 provides serial communication between M2 and other microcontrollers, microprocessors or external peripherals. It is compatible with the SSC0 of the referred C161RI device with the following extensions:

- Maximum SSC clock in master mode: $f_{SCLK\ max.} \leq 16.5$ MHz.
  Maximum SSC clock in slave mode: $f_{SCLK\ max.} \leq 8.25$ MHz.
- Reload value 0000_H is allowed in master mode.
- Because of the symmetric SSC clock requirement in master mode (50% duty cycle), the divide-by-2 prescaler is required.
- The counter of the baud rate generator is only active (running) during a transmit or receive operation.
- The transmit interrupt becomes active when a transmission starts.

The SSC0 registers can be basically divided into three types of registers as shown in **Figure 7-38**.

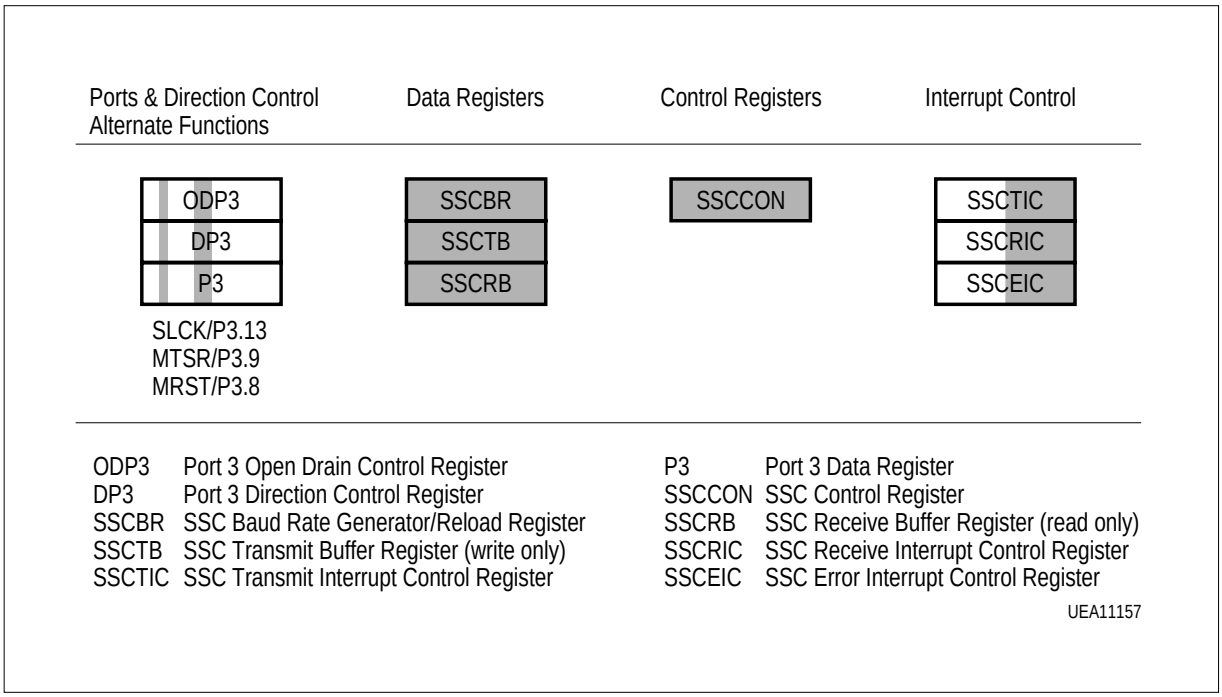


Figure 7-38 SFRs and Port Pins Associated with the SSC0

The SSC0 supports full-duplex and half-duplex synchronous communication up to 16.5 MBaud (@ 33.33 MHz module clock). The serial clock signal can be generated by the SSC0 itself (master mode), or received from an external master (slave mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baud rate generator provides the SSC with a separate serial clock signal.

The high-speed synchronous serial interface can be configured in a very flexible way, so it can be used with other synchronous serial interfaces, serve for master/slave or multimaster interconnections or operate compatible with the popular SPI interface. So it can be used to communicate with shift registers (I/O expansion), peripherals (e.g. EEPROMs etc.) or other controllers (networking). The SSC0 supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR0 (Master Transmit/Slave Receive) and MRST0 (Master Receive/Slave Transmit). The clock signal is output or input on pin SCLK0. These pins are alternate functions of port pins.

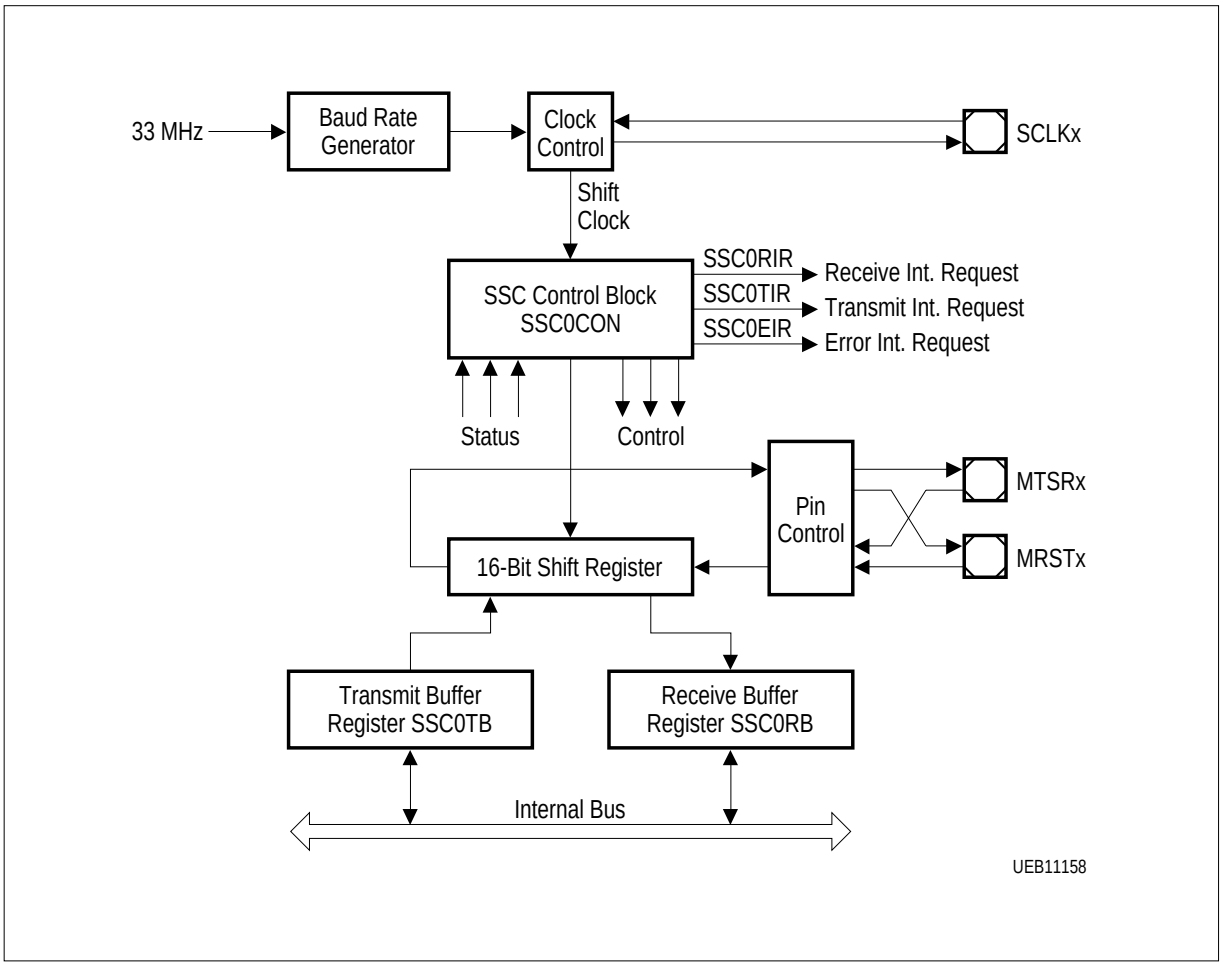


Figure 7-39 Synchronous Serial Channel SSC0 Block Diagram

The operating mode of the serial channel SSC0 is controlled by its bit-addressable control register SSCC0N. This register serves two purposes:

- during programming (SSC0 disabled by SSC0EN = ‘0’) it provides access to a set of control bits
- during operation (SSC enabled by SSC0EN = ‘1’) it provides access to a set of status flags

The shift register of the SSC0 is connected to both the transmit pin and the receive pin via the pin control logic (see block diagram in **Figure 7-39**). Transmission and reception of serial data is synchronized and takes place at the same time, e.g. the same number of transmitted bits is also received. Transmit data is written into the Transmit Buffer SSCTB. It is moved to the shift register as soon as this is empty. An SSC master (SSC0MS = ‘1’) immediately begins transmitting, while an SSC slave (SSC0MS = ‘0’) will wait for an active shift clock. When the transfer starts, the busy flag SSC0BSY is set and a transmit interrupt request line (SSCTIR) will be activated to indicate that SSCTB may be reloaded again. When the programmed number of bits (2 … 16) has been

transferred, the contents of the shift register are moved to the Receive Buffer SSCRB and a receive interrupt request line (SSCRIR) will be activated. If no further transfer is to take place (SSCTB is empty), SSC0BSY will be cleared at the same time. Software should not modify SSC0BSY, as this flag is hardware controlled.

Note: Only one SSC (etc.) can be master at a given time.

The transfer of serial data bits can be programmed in many respects:

- The data width can be chosen from 2 bits to 16 bits
- A transfer may start with the LSB or the MSB
- The shift clock may be idle low or idle high
- The data bits may be shifted with the leading or trailing edge of the clock signal
- The baud rate may be set from 254 Baud up to 16.66 MBaud (@ 33.33 MHz module clock)
- The shift clock can be generated (master) or received (slave)

These features allow the adaptation of the SSC0 to a wide range of applications, where serial data transfer is required.

The Data Width Selection supports the transfer of frames of any data length, from 2-bit ‘characters’ up to 16-bit ‘characters’. Starting with the LSB (SSC0HB = ‘0’) allows communication e.g. with an SSC device in synchronous mode (C166 family) or 8051 like serial interfaces. Starting with the MSB (SSC0HB = ‘1’) allows operation compatible with the SPI interface.
Regardless which data width is selected and whether the MSB or the LSB is transmitted first, the transfer data is always right aligned in registers SSCTB and SSCRB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of SSCTB are ignored, the unselected bits of SSCRB will not be valid and should be ignored by the receiver service routine.

The Clock Control allows the transmit and receive behavior of the SSC0 to be adapted to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out transmit data, while the other clock edge is used to latch in receive data. Bit SSC0PH selects the leading edge or the trailing edge for each function. Bit SSC0PO selects the level of the clock line in the idle state. So for an idle-high clock the leading edge is a falling one, a 1-to-0 transition (see **Figure 7-40**).

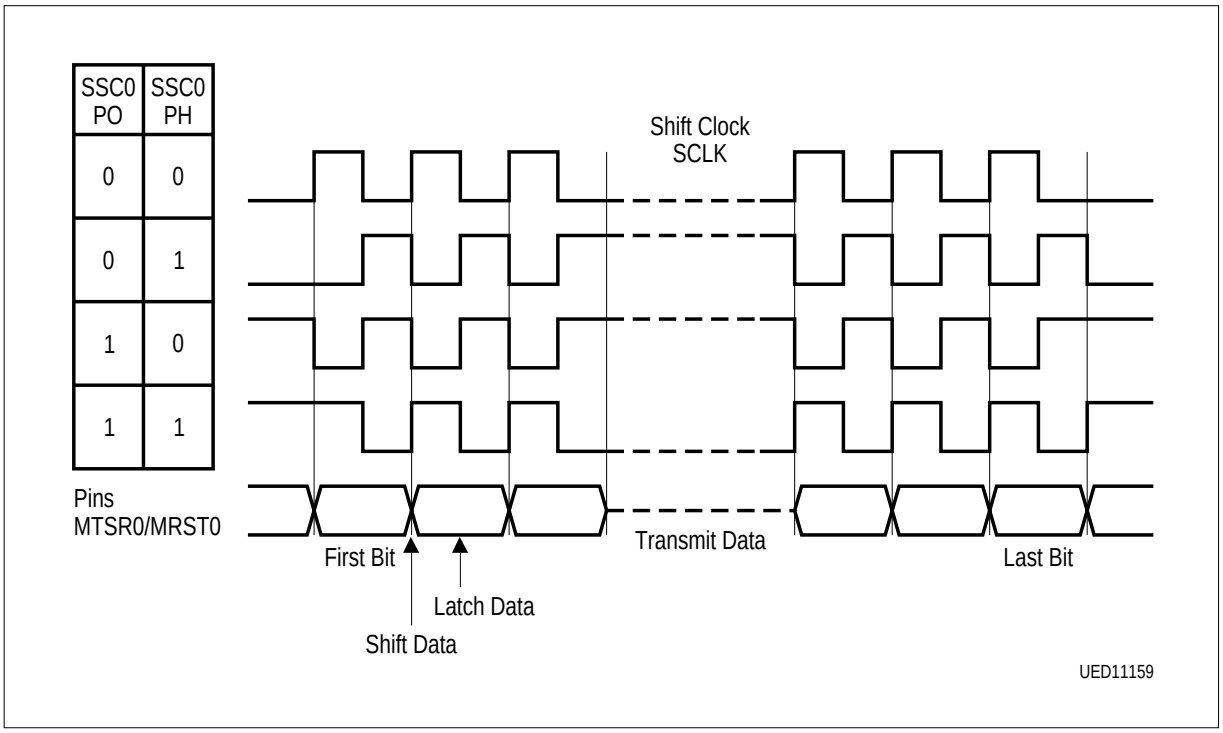


Figure 7-40 Serial Clock Phase and Polarity Options

7.4.1 Full-Duplex Operation

The different devices are connected by three lines. The definition of these lines is always determined by the master: the line connected to the master's data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode. The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register input. The output of the slaves' shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins is determined by the master or slave operation of the individual device.

Note: The shift direction shown in the diagram applies to MSB-first operation as well as to LSB-first operation.

When initializing the devices in this configuration, one device must be selected for master operation while all other devices must be programmed for slave operation. Initialization includes the operating mode of the device's SSC and also the function of the respective port lines.

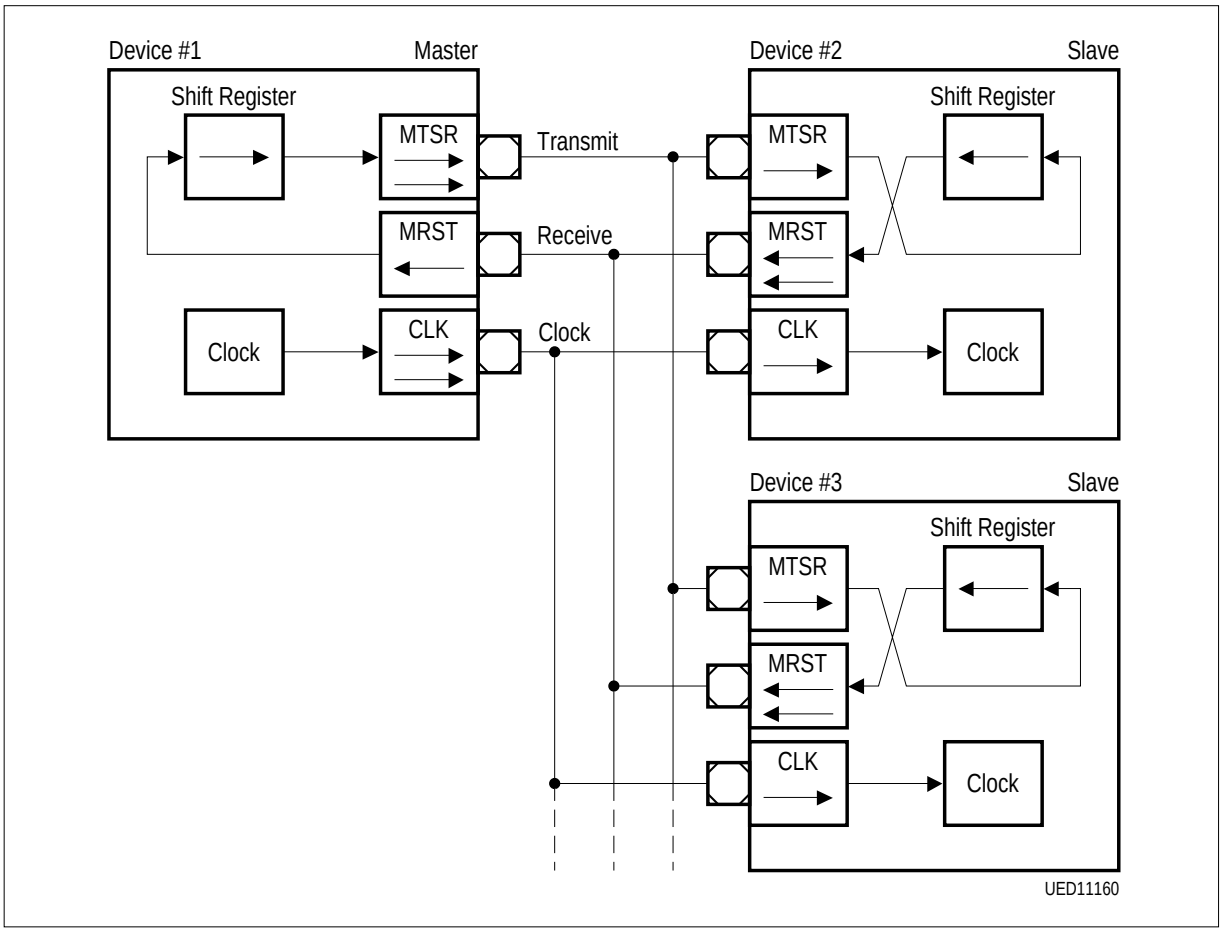


Figure 7-41 SSC0 Full Duplex Configuration

The data output pins MRST of all slave devices are connected together onto the one receive line in this configuration. During a transfer; each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

- **Only one slave drives the line**, e.g. enables the driver of its MRST pin. All the other slaves have to program their MRST pins to input. So only one slave can put its data onto the master's receive line. Only receiving of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output, until it gets a deselection signal or command.
- **The slaves use open drain output on MRST**. This forms a wired-AND connection. The receive line needs an external pullup in this case. Corruption of the data on the receive line sent by the selected slave is avoided, when all slaves which are not selected for transmission to the master only send ones ('1'). Since this high level is not actively driven onto the line, but only held through the pullup device, the selected slave can actively pull this line to a low level when transmitting a zero bit. The master

selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initializations of the SSC0, the serial interfaces can be enabled. In a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either ‘0’ or ‘1’, until the first transfer starts. After a transfer, the alternate data line will always remain at the logic level of the last transmitted data bit.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register SSCTB. This value is copied into the shift register (which is assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock to the baud rate generator (transmission only starts, if SSC0EN = ‘1’). Depending on the selected clock phase, a clock pulse will also be generated on the SCLK line. With the opposite clock edge the master simultaneously latches and shifts in the data detected at its input line MRST. This ‘exchanges’ the transmit data with the receive data. Since the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master’s shift register, shifting out the data contained in the registers, and shifting in the data detected at the input line. After the pre-programmed number of clock pulses (via the data width selection) the data transmitted by the master is contained in all slaves’ shift registers, while the master’s shift register holds the data of the selected slave. In the master and all the slaves, the content of the shift register is copied into the receive buffer SSCRB and the receive interrupt line SSC0RIR is activated.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST, when the content of the transmit buffer is copied into the slave’s shift register. It will not wait for the next clock from the baud rate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock edge generated by the master may be already used to clock in the first data bit. So the slave’s first data bit must already be valid at this time.

*Note: On the SSC0 a transmission **and** a reception always takes place at the same time, regardless whether valid data has been transmitted or received.*

The initialization of the SCLK pin on the master requires some attention in order to avoid undesired clock transitions, which may disturb the other receivers. The state of the internal alternate output lines is ‘1’ as long as the SSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the SSC with an idle-low clock (SSC0PO = ‘0’) will immediately drive the alternate data output and (via the AND) the port pin SCLK low. To avoid this, the following sequence should be used:

- select the clock idle level (SSC0PO = ‘x’)
- load the port output latch with the desired clock idle level
- switch the pin to output
- enable the SSC0 (SSC0EN = ‘1’)
- if SSC0PO = ‘0’: enable alternate data output

The same mechanism as for selecting a slave for transmission (separate select lines or special commands) may also be used to move the role of the master to another device in the network. In this case the previous master and the future master (previous slave) will have to toggle their operating mode (SSC0MS) and the direction of their port pins.

7.4.2 Half Duplex Operation

In a half duplex configuration only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both the MTSR and MRST pins in each device, the clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

Similar to full duplex mode there are **two ways to avoid collisions** on the data exchange line:

- only the transmitting device may enable its transmit pin driver
- the non-transmitting devices use open drain output and only send ones.

Since the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By these means any corruptions on the common data exchange line, where the received data is not equal to the transmitted data are detected.

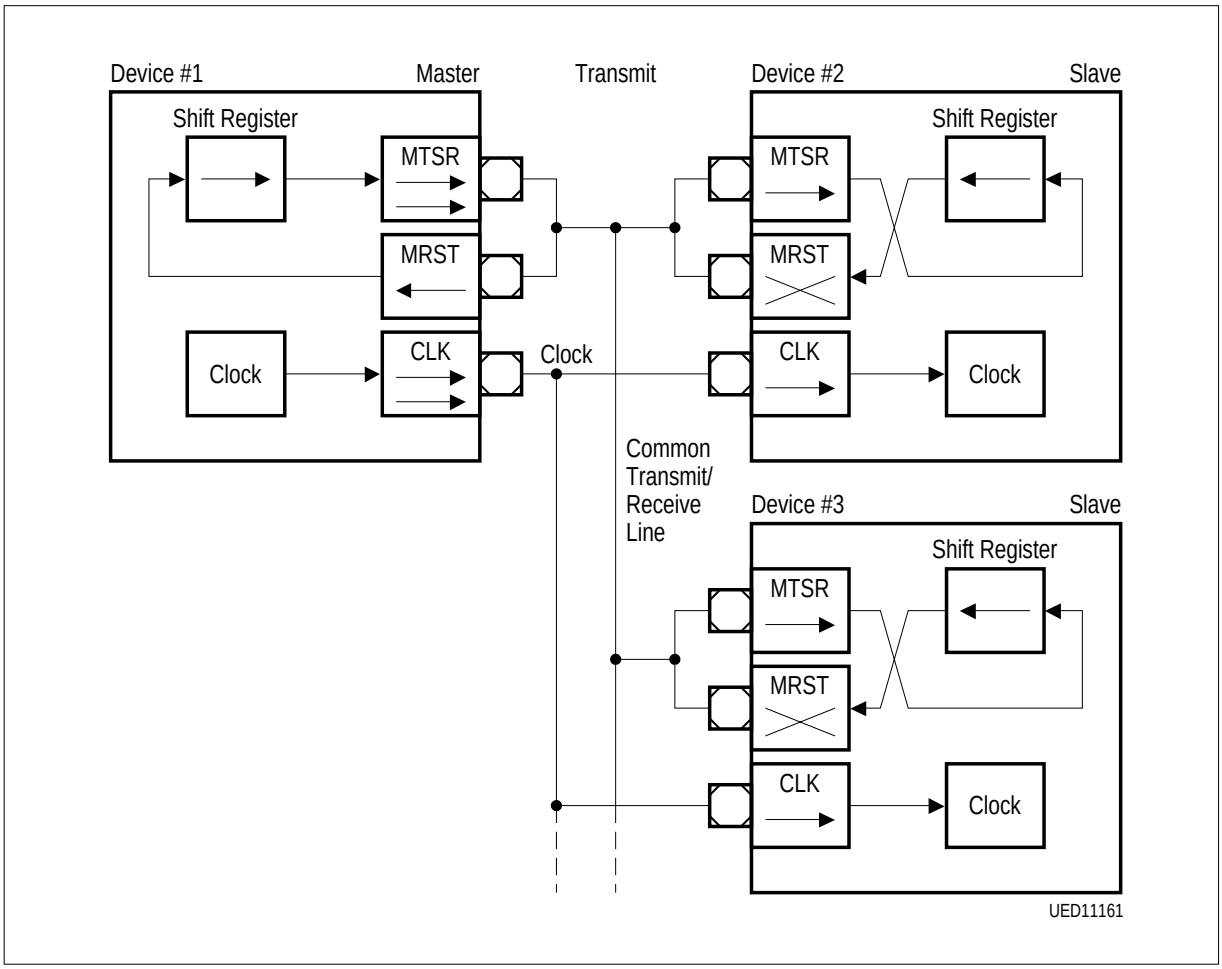


Figure 7-42 SSC Half Duplex Configuration

7.4.3 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCTB is empty and ready to be loaded with the next transmit data. If SSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission starts without any additional delay. On the data line there is no gap between the two successive frames. For example, two byte transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. How long a total data frame length can be depends on the software. This option can also be used e.g. to interface to byte-wide and word-wide devices on the same serial bus.

Note: Of course, this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the SSC0 to reprogram the basic data width on-the-fly.

7.4.4 Port Control

The SSC0 uses three pins to communicate with the external world. Pin SCLK serves as the clock line, while pins MRST (Master Receive/Slave Transmit) and MTSR (Master Transmit/Slave Receive) serve as the serial data input/output lines.

The operation of the SSC0 pins depends on the selected operating mode (master or slave). The direction of the port lines depends on the operating mode. The SSC0 will automatically use the correct alternate input or output line of the ports when switching modes. The direction of the port pins, however, must be programmed by the user. Using the open drain output feature of the port lines helps to avoid bus contention problems and reduces the need for hardwired hand-shaking or slave select lines. In this case it is not always necessary to switch the direction of a port pin.

7.4.5 Baud Rate Generation

The serial channel SSC0 has its own dedicated 16-bit baud rate generator with 16-bit reload capability, allowing baud rate generation independent from the timers. In addition to **Figure 7-39**, **Figure 7-43** shows the baud rate generator of the SSC0 in more detail.

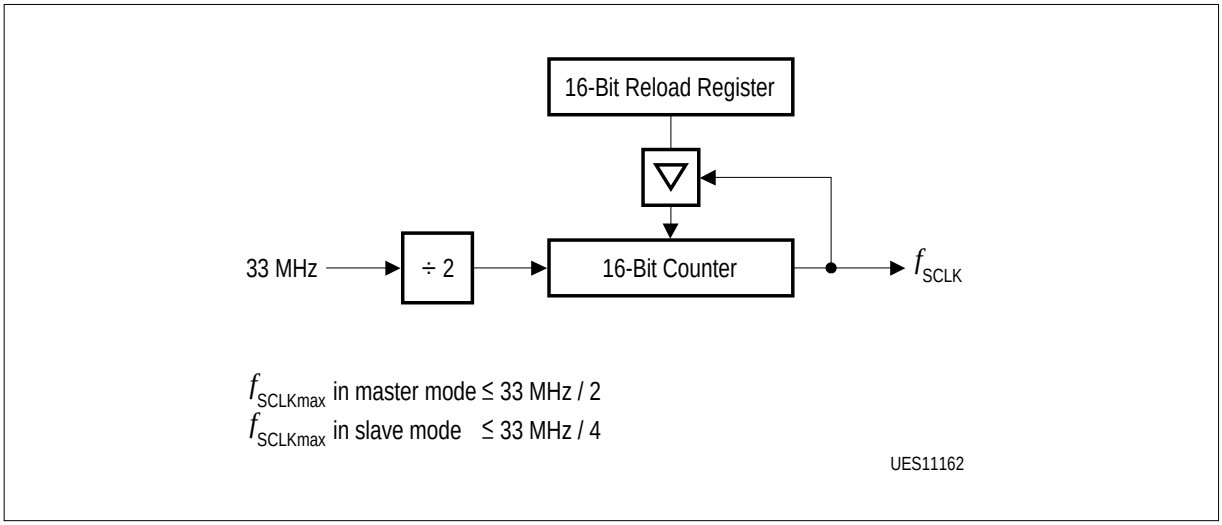


Figure 7-43 SSC0 Baud Rate Generator

The baud rate generator is clocked with the module clock 33.33 MHz. The timer is counting downwards. Register SSCBR is the dual-function Baud Rate Generator/Reload register. Reading SSCBR, while the SSC0 is enabled, returns the content of the timer. Reading SSCBR, while the SSC0 is disabled, returns the programmed reload value. In this mode the desired reload value can be written to SSCBR.

Note: Never write to SSCBR, while the SSC0 is enabled.

The formulas below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baud rate:

$$\text{Baud rate}_{SSC0} = \frac{33\ \text{MHz}}{2 \times (\text{<SSCBR>} + 1)} \qquad \text{<SSCBR>} = \left(\frac{33\ \text{MHz}}{2 \times \text{Baud rate}_{SSC0}} \right) - 1$$

<SSCBR> represents the content of the reload register, taken as an unsigned 16-bit integer while Baud rate$_{SSC}$ is equal to f_{SCLK} as shown in **Figure 7-43**.

The maximum baud rate that can be achieved when using a module clock of 33.33 MHz is 16.6 MBaud in master mode (with <SSCBR> = 0000_H) and 8.33 MBaud in slave mode (with <SSCBR> = 0001_H) lists some possible baud rates together with the required reload values and the resulting bit times, assuming a module clock of 33.33 MHz.

7.4.6 Error Detection Mechanisms

The SSC0 is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes, while Transmit Error and baud rate Error only apply to slave mode. When an error is detected, the respective error flag is set and an error interrupt request will be generated by activating the SSCEIR line (see **Figure 7-44**). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not reset automatically but rather must be cleared by software after servicing. This allows some error conditions to be serviced via interrupt, while the others may be polled by software.

Note: The error interrupt handler must clear the associated (enabled) error-flag(s) to prevent repeated interrupt requests.

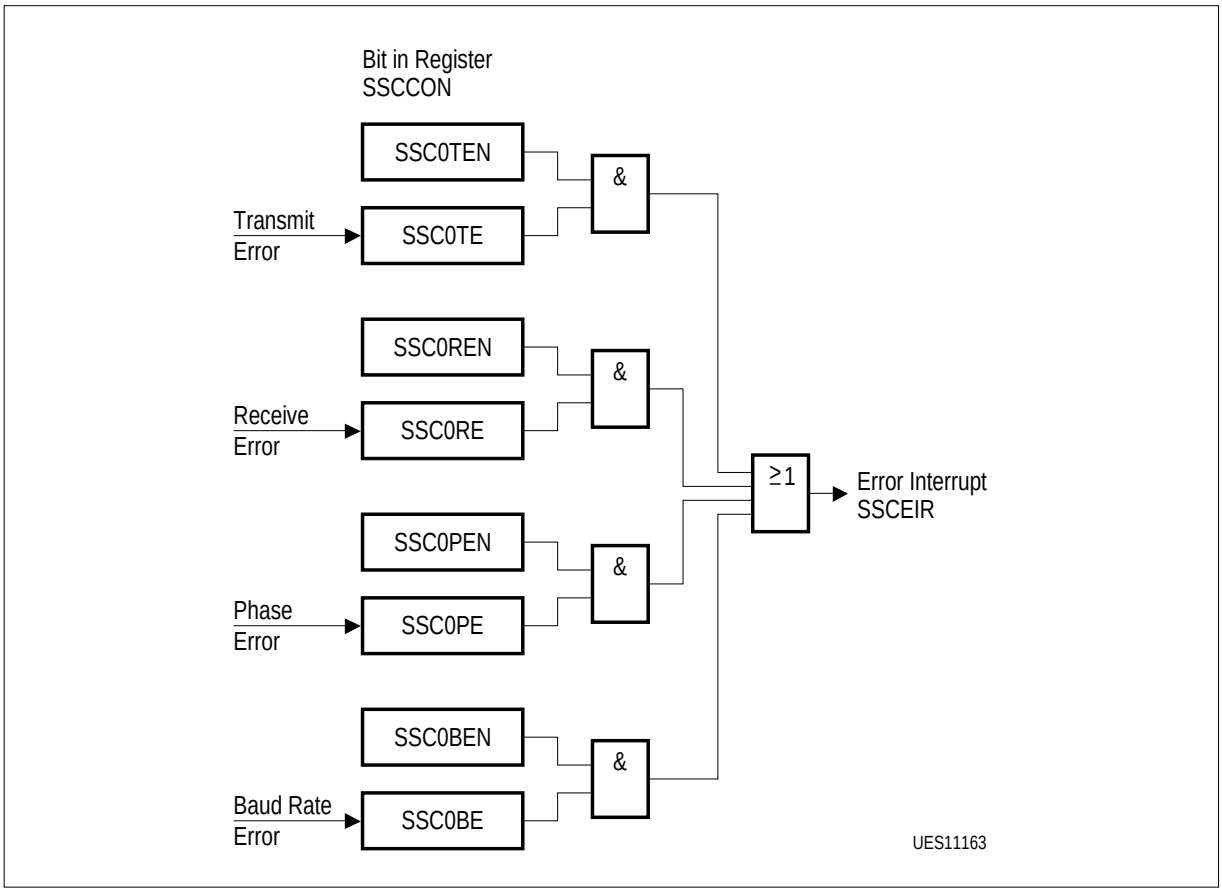


Figure 7-44 SSC0 Error Interrupt Control

A **Receive Error** (Master or Slave mode) is detected, when a new data frame is completely received, but the previous data was not read out of the receive buffer register SSCRB. This condition sets the error flag SSC0RE and, when enabled via SSC0REN, the error interrupt request line SSCEIR. The old data in the receive buffer SSCRB will be overwritten with the new value and is unretrievably lost.

A **Phase Error** (Master or Slave mode) is detected, when the incoming data at pin MRST0 (master mode) or MTSR0 (slave mode), sampled with the same frequency as the module clock, changes between one cycle before and two cycles after the latching edge of the shift clock signal SCLK. This condition sets the error flag SSC0PE and, when enabled via SSC0PEN, the error interrupt request flag SSCEIR.

A **Baud Rate Error** (Slave mode) is detected, when the incoming clock signal deviates from the programmed baud rate by more than 100%, e.g. it is either more than double or less than half the expected baud rate. This condition sets the error flag SSC0BE and, when enabled via SSC0BEN, the error interrupt request line SSCEIR. Using this error detection capability requires that the slave's baud rate generator is programmed to the same baud rate as the master device. This feature detects false additional, or missing pulses on the clock line (within a certain frame).

Note: If this error condition occurs and bit SSC0REN = '1', an automatic reset of the SSC0 will be performed in case of this error. This is done to re-initialize the SSC0, if too few or too many clock pulses have been detected.

A **Transmit Error** (Slave mode) is detected, when a transfer is initiated by the master (shift clock gets active), but the transmit buffer SSCTB of the slave was not updated since the last transfer. This condition sets the error flag SSC0TE and, when enabled via SSC0TEN, the error interrupt request line SSC0EIR. If a transfer starts while the transmit buffer is not updated, the slave will shift out the 'old' contents of the shift register, which is normally the data received during the last transfer. This may lead to the corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration), if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones, e.g. their transmit buffers must be loaded with '$FFFF_H$' prior to any transfer.

Note: A slave with push/pull output drivers, which is not selected for transmission, will normally have its output drivers switched. However, in order to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.

The cause of an error interrupt request (receive, phase, baud rate, transmit error) can be identified by the error status flags in control register SSCCON.

Note: In contrary to the error interrupt request line SSCEIR, the error status flags SSC0TE, SSC0RE, SSC0PE, and SSC0BE, which are located in register SSCCON, are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

7.4.7 Register Description

The operating mode of the serial channel SSC0 is controlled by its control register SSCCON. This register contains control bits for mode and error check selection, and status flags for error identification. Depending on bit SSC0EN, either control functions or status flags and master/slave control is enabled.

SSC0EN = 0: Programming Mode

SSCCON **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC0 EN	SSC0 MS	-	SSC0 AREN	SSC0 BEN	SSC0 PEN	SSC0 REN	SSC0 TEN	SSC0 LB	SSC0 PO	SSC0 PH	SSC0 HB	SSC0BM			
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

Bit	Function
SSC0BM	**SSC0 Data Width Selection** 0: Reserved. Do not use this combination. 1 … 15: Transfer Data Width is 2 … 16 bit (<SSC0BM>+1)
SSC0HB	**SSC0 Heading Control Bit** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
SSC0PH	**SSC0 Clock Phase Control Bit** 0: Shift transmit data on the leading clock edge, latch on trailing edge 1: Latch receive data on leading clock edge, shift on trailing edge
SSC0PO	**SSC0 Clock Polarity Control Bit** 0: Idle clock line is low, leading clock edge is low-to-high transition 1: Idle clock line is high, leading clock edge is high-to-low transition
SSC0LB	**SSC0 Loop Back Bit** 0: Normal output 1: Receive input is connected with transmit output (half duplex mode)
SSC0TEN	**SSC0 Transmit Error Enable Bit** 0: Ignore transmit errors 1: Check transmit errors
SSC0REN	**SSC0 Receive Error Enable Bit** 0: Ignore receive errors 1: Check receive errors

Bit	Function
SSC0PEN	**SSC0 Phase Error Enable Bit** 0: Ignore phase errors 1: Check phase errors
SSC0BEN	**SSC0 Baud Rate Error Enable Bit** 0: Ignore baud rate errors 1: Check baud rate errors
SSC0AREN	**SSC0 Automatic Reset Enable Bit** 0: No additional action upon a baud rate error 1: The SSC is automatically reset upon a baud rate error
SSC0MS	**SSC0 Master Select Bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSC0EN	**SSC0 Enable Bit = ‘0’** Transmission and reception disabled. Access to control bits.

SSC0EN = 1: Operating Mode

SSCCON **Reset Value: 0000_H**

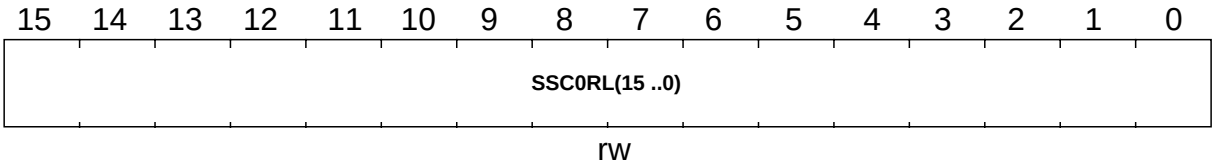
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC0 EN	SSC0 MS	-	SSC0 BSY	SSC0 BE	SSC0 PE	SSC0 RE	SSC0 TE	-	-	-	-	SSC0BC			
rw	rw		r	rw	rw	rw	rw							rw	

Bit	Function
SSC0BC	**SSC0 Bit Count Field** Shift counter is updated with every shifted bit. **Do not write to!!!**
SSC0TE	**SSC0 Transmit Error Flag** 1: Transfer starts with the slave's transmit buffer not being updated
SSC0RE	**SSC0 Receive Error Flag** 1: Reception completed before the receive buffer was read
SSC0PE	**SSC0 Phase Error Flag** 1: Received data changes around sampling clock edge
SSC0BE	**SSC0 Baud Rate Error Flag** 1: More than factor 2 or 0.5 between Slave's actual and expected baud rate
SSC0BSY	**SSC0 Busy Flag** Set while a transfer is in progress. **Do not write to!!!**
SSC0MS	**SSC0 Master Select Bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSC0EN	**SSC0 Enable Bit = '1'** Transmission and reception enabled. Access to status flags and M/S control.

Note: The target of an access to SSCCON (control bits or flags) is determined by the state of SSC0EN prior to the access, i.e. writing $C057_H$ to SSC0CON in programming mode (SSC0EN = '0') will initialize the SSC (SSC0EN was '0') and then switch it on (SSC0EN = '1').
When writing to SSCCON, make sure that zeros are input to reserved locations.

The SSC0 baud rate timer reload register SSCBR contains the 16-bit reload value for the baud rate timer.

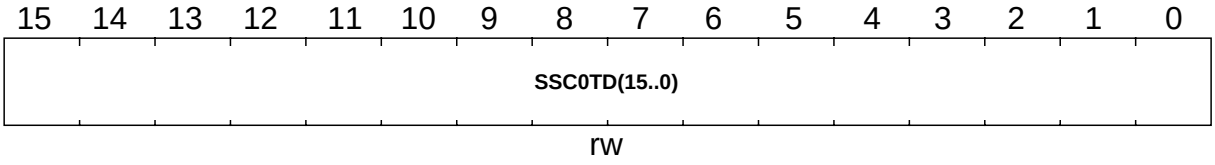
SSCBR **Reset Value: 0000_H**



Bit	Function
SSC0RL (15 … 0)	**Baud Rate Timer/Reload Register Value** Reading SSCBR returns the 16-bit content of the baud rate timer. Writing SSC0BR loads the baud rate timer reload register.

The SSC0 transmitter buffer register SSCTB contains the transmit data value.

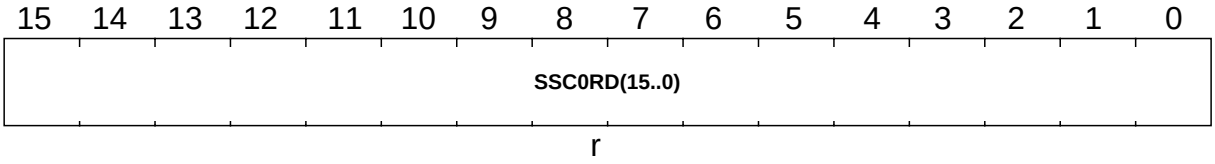
SSCTB **Reset Value: 0000_H**



Bit	Function
SSC0TD (15 … 0)	**Transmit Data Register Value** SSCTB contains the data to be transmitted. Unselected bits of SSC0TB are ignored during transmission.

The SSC0 receiver buffer register SSCRB contains the receive data value.

SSCRB **Reset Value: 0000_H**



Bit	Function
SSC0RD (7 … 0)	**Receive Data Register Value** SSCRB contains the received data bits. Unselected bits of SSC0RB will be not valid and should be ignored

7.5 I^2C-Bus Interface

The on-chip I^2C Bus module connects the M2 to other external controllers and/or peripherals via the two-line serial I^2C interface. The I^2C Bus module provides communication at data rates of up to 400 Kbit/s and features 7-bit as well as 10-bit addressing.

The module can operate in three different modes:

Master mode, where the I^2C controls the bus transactions and provides the clock signal.

Slave mode, where an external master controls the bus transactions and provides the clock signal.

Multimaster mode, where several masters can be connected to the bus, i.e. the I^2C can be master or slave.

The on-chip I^2C bus module allows efficient communication via the common I^2C bus. The module unloads the CPU of low level tasks such as

- (De)Serialization of bus data.
- Generation of start and stop conditions.
- Monitoring the bus lines in slave mode.
- Evaluation of the device address in slave mode.
- Bus access arbitration in multimaster mode.

Features

- Extended buffer allows up to 4 send/receive data bytes to be stored.
- Support of standard 100 KBaud and extended 400 KBaud data rates.
- Operation in 7-bit or 10-bit addressing mode.
- Flexible control via interrupt service routines or by polling.

7.5.1 Operational Overview

Data is transferred by the 2-line I^2C bus (SDA, SCL) using a protocol that ensures reliable and efficient transfers. This protocol clearly distinguishes regular data transfers from defined control signals which control the data transfers.

The following bus conditions are defined:

Bus Idle: SDA and SCL remain high. The I^2C bus is currently not used.

Data Valid: SDA stable during the high phase of SCL. SDA then represents the transferred bit. There is one clock pulse for each transferred bit of data. During data transfers SDA may only change while SCL is low (see below)!

Start Transfer: A falling edge on SDA (↘) while SCL is high indicates a start condition. This start condition initiates a data transfer over the I^2C bus.

Stop Transfer: A rising edge on SDA (↗) while SCL is high indicates a stop condition. This stop condition terminates a data transfer. An arbitrary number of bytes may be transferred between a start condition and a stop condition.

7.5.2 The Physical I^2C-Bus Interface

Communication via the I^2C Bus uses two bidirectional lines, the serial data line SDA and the serial clock line SCL. Each of these two generic interface lines can be connected to a number of IO port lines. These connections can be established and released under software control.

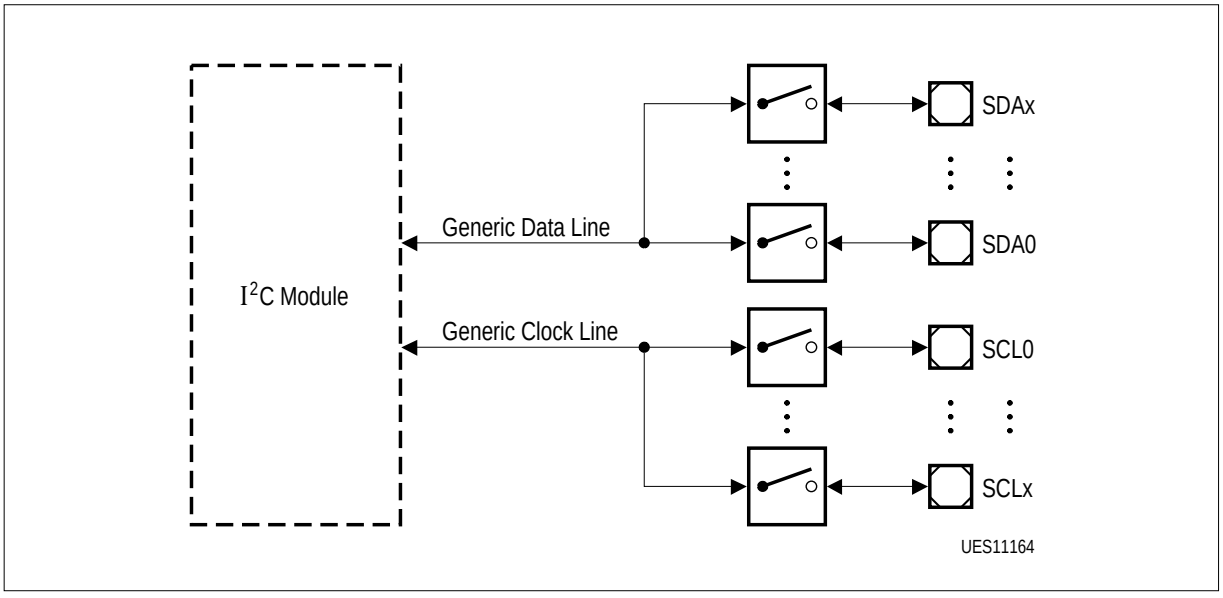


Figure 7-45 I^2C Bus Line Connections

This mechanism allows a number of configurations of the physical I^2C Bus interface:

Physical Channels

Can be selected, so the I^2C module can use electrically separated channels or increase the addressing range by using more data lines.

Note: Baud rate and physical channels should never be changed (via ICCFG) during a transfer.

Channel Switching

The I^2C module can be connected to a specific pair of pins (e.g. SDA0 and SCL0) which then forms a separate I^2C channel to the external system. The channel can be dynamically switched by connecting the module to another pair of pins (e.g. SDA1 and SCL1). This establishes physically separate interface channels.

Broadcasting:

Connecting the module to more than one pair of pins (e.g. SDA0/1 and SCL0/1) allows the transmission of messages over multiple physical channels at the same time. Please note that this configuration is critical when the M2 is a slave. In master mode it cannot be guaranteed that all selected slaves have reached the message.

Register ICCFG selects the bus baud rate as well the activation of SDA and SCL lines. So an external I^2C channel can be established (baud rate and physical lines) with one single register access.

Note: Respective port pin definition

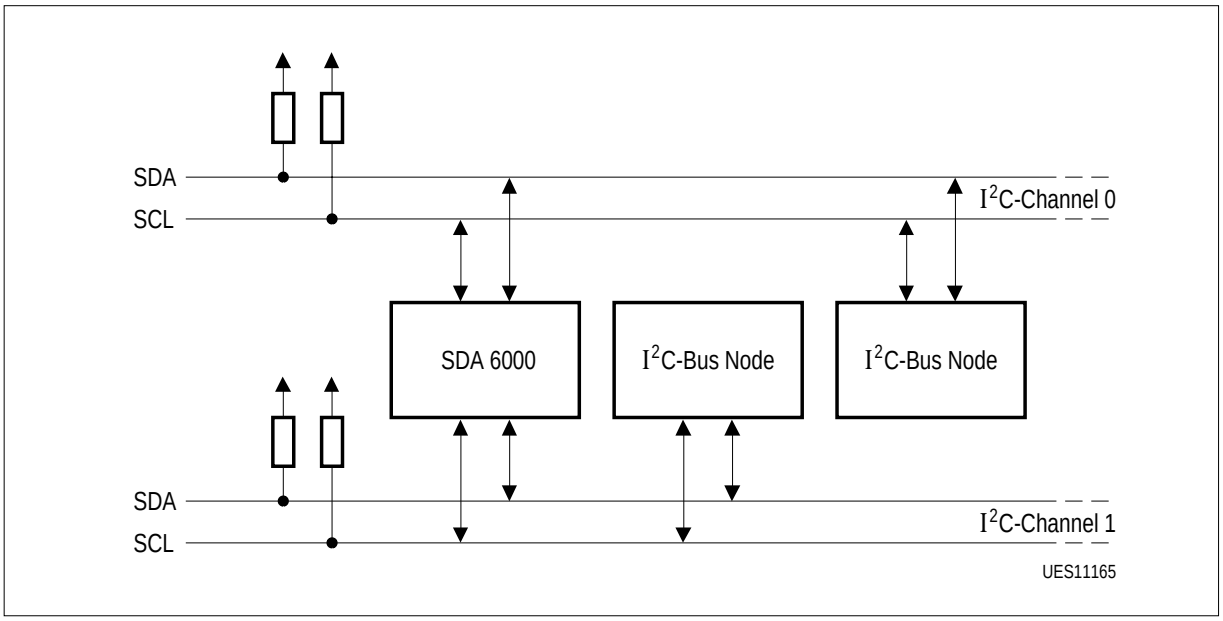


Figure 7-46 Physical Bus Configuration Example

Output Pin Configuration

The pin drivers that are assigned to the I^2C channel(s) provide open drain outputs (i.e. no upper transistor). This ensures that the I^2C module does not put any load on the I^2C bus lines while the M2 is not powered. The I^2C bus lines therefore require external pull-up resistors (approx. 10 KΩ for operation at 100 KBaud, 2 KΩ for operation at 400 KBaud).

All pins of the M2 that are to be used for I^2C bus communication must be switched to output, opendrain and their alternate function must be enabled (by setting the respective port output latch to ‘1’), before any communication can be established.

If not driven by the I^2C module (i.e. the corresponding enable bit in register ICCFG is ‘0’) they then switch off their drivers (i.e. driving ‘1’ to an open drain output). Due to the external pull-up devices the respective bus levels will then be ‘1’ which is idle.

The I^2C module features digital input filters in order to improve the noise output from the external bus lines.

7.5.3 Functional Overview

Operation in Master Mode

If the on-chip I^2C module controls the I^2C bus (i.e. bus master), master mode must be selected via bit field MOD in register ICCON. The physical channel is configured by a control word written to register ICCFG, defining the active interface pins and the used baud rate. More than one SDA and/or SCL line may be active at a time. The address of the remote slave that is to be accessed is written to ICRTB0 … 3. The bus is claimed by setting bit BUM in register ICCON. This generates a start condition on the bus and automatically starts the transmission of the address in ICRTB0. Bit TRX in register ICCON defines the transfer direction (TRX = ‘1’, i.e. transmit, for the slave address). A repeated start condition is generated by setting bit RSC in register ICCON, which automatically starts the transmission of the address previously written to ICRTB0. This may be used to change the transfer direction. RSC is cleared automatically after the repeated start condition has been generated.

The bus is released by clearing bit BUM in register ICCON. This generates a stop condition on the bus.

Note: Between load the address in ICRTB0 an setting bit BUM at least one command (nop) has to be executed.

Operation in Multimaster Mode

If multimaster mode is selected via bit field MOD in register ICCON, the on-chip I^2C module can operate concurrently as a bus master or as a slave. The descriptions of these modes apply accordingly.

Multimaster mode implies that several masters are connected to the same bus. As more than one master may try to claim the bus at a given time, an arbitration is done on the SDA line. When a master device detects a mismatch between the data bit to be sent and the actual level on the SDA (bus) line, it looses the arbitration and automatically switches to slave mode (leaving the other device as the remaining master). This loss of arbitration is indicated by bit AL in register ICST which must be checked by the driver software when operating in multimaster mode. Lost arbitration is also indicated when the software tries to claim the bus (by setting bit BUM) while the I^2C bus is active (indicated by bit BB = ‘1’). Bit AL must be cleared via software.

Operation in Slave Mode

If the on-chip I^2C module is controlled via the I^2C bus by a remote master (i.e. be a bus slave), slave mode must be selected via bit field MOD in register ICCON. The physical channel is configured by a control word written to register ICCFG, defining the active interface pins and the used baud rate. It is recommended to have only one SDA and SCL

line active at a time when operating in slave mode. The address by which the slave module can be selected is written to register ICADR.

The I^2C module is selected by another master when it receives (after a start condition), either its own device address (stored in ICADR) or the general call address (00_H). In this case an interrupt is generated and bit SLA in register ICST is set, indicating the valid selection. The desired transfer mode is then selected via bit TRX (TRX = '0' for reception, TRX = '1' for transmission).

For a transmission the respective data byte is placed into the buffer ICRTB0 … 3 (which automatically sets bit TRX) and the acknowledge behavior is selected via bit ACKDIS.

For a reception the respective data byte is fetched from the buffer ICRTB0 … 3 after IRQD has been activated.

In both cases the data transfer itself is enabled by clearing bits IRQD, IRQP and IRQE which releases the SCL line.

When a stop condition is detected, bit SLA is cleared.

The I^2C bus configuration register ICCFG selects the bus baud rate (partly) as well as the activation of SDA and SCL lines. So an external I^2C channel can be established (baud rate and physical lines) with one single register access.

Systems that utilize several I^2C channels can prepare a set of control words which configure the respective channels. By writing one of these control words to ICCFG the respective channel is selected. Different channels may use different baud rates. Also different operating modes can be selected, e.g. enabling all physical interfaces for a broadcast transmission.

*Note: Refer also to **Chapter 7.5.2**.*

7.5.4 Registers

All available module registers are summarized in the overview table below.

Register Name	Register Description	Address	b/p[1)]	Reset Value
ICCFG	I^2C Configuration Register	$00'E810_H$	b	0000_H
ICCON	I^2C Control Register	$00'E812_H$	b	0000_H
ICST	I^2C Status Register	$00'E814_H$	b	0000_H
ICADR	I^2C Address Register	$00'E816_H$	b	0000_H
ICRTBL	I^2C Receive/Transmit Buffer	$00'E818_H$	–	0000_H
ICRTBH	I^2C Receive/Transmit Buffer	$00'E81A_H$	–	0000_H
IICPISEL[2)]	I^2C Port Input Select Register	$00'E804_H$	b	0000_H

1) **b**: **b**it addressable / **p**: bit **p**rotected

2) Itus currently no function. Should be left on reset value.

I^2C Configuration Register

ICCFG **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRPL								0	0	SCL EN1	SCL EN0	0	SDA EN2	SDA EN1	SDA EN0

Field	Bits	Type	Value	Description
SDAENx (x = 2 … 0)	2 … 0	rw		**Enable Input for Data Pin x** These bits determine to which pins the I^2C data line is connected.
			0	SDA pin x is disconnected.
			1	SDA pin x is connected with I^2C data line.
SCLENx (x = 3 … 0)	5 … 4	rw		**Enable Input for Clock Pin x** These bits determine to which pins the I^2C clock line is connected.
			0	SCL pin x is disconnected.
			1	SCL pin x is connected with I^2C clock line.
BRPL	15 … 8	rw	–	**Baud rate Prescaler Low** Determines the 8 least significant bits of the 10 bit baud rate prescaler. (See BPRH in ICADR.)

I²C Control Register ICCON

Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WM EN	**0**	**0**	**0**	**CI**		**STP**	**IGE**	**TRX**	**INT**	**ACK DIS**	**BU M**	**MOD**		**RSC**	**M10**

Field	Bits	Type	Value	Description
M10	0	rw	 0 1	Address Mode 7-bit addressing using ICA7 … 1. 10-bit addressing using ICA9 … 0.
RSC	1	rwh	 0 1	Repeated Start Condition No operation. Generate a repeated start condition in (multi) master mode. RSC cannot be set in slave mode. Note: RSC is cleared automatically after the repeated start condition has been sent.
MOD	[3:2]	rwh	 00 01 10 11	Basic Operating Mode I²C module is disabled and initialized (Init-Mode). Transmissions under execution will be aborted. Slave mode. 10 Master mode. 11 Multi-Master mode.
BUM	4	rwh	 0 1	Busy Master Clearing bit BUM (↘) generates a stop condition immediately. Setting bit BUM (↗) generates a start condition in (multi)master mode. Note: Setting bit BUM (↗) while BB = ‘1’ generates an arbitration lost situation. In this case BUM is cleared and bit AL is set. BUM can not be set in slave mode.

Field	Bits	Type	Value	Description
ACKDIS	5	rwh		Acknowledge Pulse Disable
			0	An acknowledge pulse is generated for each received frame.
			1	No acknowledge pulse is generated.
				Note: ACKDIS is automatically cleared by a stop condition.
INT	6	rw		Interrupt Delete Select
			0	Interrupt flag IRQD is deleted by read/write to ICRTB0 … 3.
			1	Interrupt flag IRQD is not deleted by read/ write to ICRTB0 … 3.
TRX	7	rwh		Transmit Select
			0	No data is transmitted to the I^2C bus.
			1	Data is transmitted to the I^2C bus.
				Note: TRX is set automatically when writing to the transmit buffer. It is not allowed to delete this bit in the same buscycle. It is automatically cleared after last byte as slave transmitter.
IGE	8	rw		Ignore IRQE
				Ignore IRQE (End of transmission)
			0	interrupt.
			1	The I^2C is stopped at IRQE interrupt.
				The I^2C ignores the IRQE interrupt.
				If RMEN is set, RM is mirrored here.
STP	9	rwh		Stop Master
			0	0 Clearing bit STP generates no stop condition.
			1	1 Setting bit STP generates a stop condition after next transmission. BUM is set to zero. ACKDIS is set to one. STP is automatically cleared by a stop condition. If RMEN is set, RM is mirrored here.

Field	Bits	Type	Value	Description
CI	[10:9]	rw	 00 01 10 11	Length of the Transmit Buffer 1 Byte 2 Bytes 3 Bytes 4 Bytes If RMEN is set, RM is mirrored here.
WMEN	15	rwh	 0 1	Write Mirror Enable write mirror is not active write mirror is active If RMEN is set WMEN can not be set and will remain zero. If WMEN and RMEN are simultaneously set to 1, both will remain what they are. So only one of each can be set to 1.
RM	[15:8]	rw	–	Read Mirror If RMEN is set, RTB0 may be read here. Writing to RM has no effect in this mode.

I²C Status Register ICST

Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RM EN	**0**	**0**	**0**	**0**	**CO**			**IRQ E**	**IRQ P**	**IRQ D**	**BB**	**LRB**	**SLA**	**AL**	**ADR**

Field	Bits	Type	Value	Description
ADR	0	rh	–	Address Bit ADR is set after a start condition in slave mode until the address has been received (1 byte in 7-bit address mode, 2 bytes in 10-bit address mode).
AL	1	rwh	–	Arbitration Lost Bit AL is set when the I²C module has tried to become master on the bus, but has lost the arbitration. Operation is continued until the 9th clock pulse. If multimaster mode is selected the I²C module temporarily switches to slave mode after a lost arbitration. Bit IRQP is set along with bit AL. AL must be cleared via software.
SLA	2	rh	0 1	Slave The I²C module is not selected as a slave, or the module is in master mode. The I²C module has been selected as a slave (device address received).
LRB	3	rh	–	Last Received Bit Bit LRB represents the last bit (i.e. the acknowledge bit) of the last transferred frame. It is automatically set to zero by a write or read access to the buffer ICRTB0 … 3. **Note**: If LRB is high (no acknowledge) in slave mode, TRX bit is set automatically.

Field	Bits	Type	Value	Description
BB	4	rh		Bus Busy
			0	The I^2C bus is idle, i.e. a stop condition has occurred.
			1	The I^2C bus is active, i.e. a start condition has occurred.
				Note: Bit BB is always ‘0’ while the I^2C module is disabled.
IRQD	5	rwh		I^2C Interrupt Request Bit for Data Transfer Events [1)]
			0	No interrupt request pending.
			1	A data transfer event interrupt request is pending.
				IRQD is set after the acknowledge bit of the last byte has been received or transmitted, and is cleared automatically upon a complete read or write access to the buffer(s) ICRTB0 … 3.
				New data transfers will start immediately after clearing IRQD. Do not access any register until next interrupt. If in polling mode and CI is ‘0’ only 8-Bit accesses to the lower byte are allowed.
				Note: If a multi byte write could not be finished in slave mode because of missing acknowledge, then the data interrupt is followed by a end of transmission interrupt. The number of bytes sent can be read from CO. The data interrupt must have higher priority than IRQE.

Field	Bits	Type	Value	Description
IRQP	6	rwh		I^2C Interrupt Request Bit for Protocol Events [1)]
			0	No interrupt request pending.
			1	A protocol event interrupt request is pending. IRQP is set when bit SLA or bit AL is set (↗), and must be cleared via software. If the I^2C has been selected by an other master, the software must look up the required transmission direction by reading the received address and direction bit, stored in ICRTB0. The TRX-bit must correspondingly be set by software.
IRQE	7	rwh		I^2C Interrupt Request Bit for Data Transmission End [1)]
			0	No interrupt request pending.
			1	A receive end event interrupt request is pending (a stop is detected). IRQE is automatically cleared upon a start condition. IRQE is not activated in init-mode. IRQE must always be deleted to continue transmission. **Note:** In slave mode IRQE is set after the transmission is finished. This can also be after a stop or RSC condition. In this case the slave is not selected any more. This bit is also set, if a transmission is stopped by a missing acknowledge. In this case the bit must be cleared by software.

Field	Bits	Type	Value	Description
CO	10..8	rw		**Counter of Transmitted Bytes Since Last Data Interrupt.** If a multi byte transmission could not be finished because of a missing acknowledge, the number of correctly transferred bytes can be read from CO. It is automatically set to zero by the correct number (defined by CI) of write/read accesses to the buffers ICRTB0 … 3.
			000	No Byte
			001	1 Byte
			010	2 Bytes
			011	3 Bytes
			100	4 Bytes
				The number of legal bytes depends on the data buffer size (CI). Writing to this bit field does not affect its content. If WMEN is set, WM is mirrored here.
WM	[15:8]	wh	–	Write Mirror If WMEN is set, RTB0 may be written here. Reading WM will result in zero.

[1] While IRQD, IRQP **or** IRQE is set and the I^2C module is in master mode or has been selected as a slave, the I^2C clock line is held low which prevents further transfers on the I^2C bus.
The clock line of the I^2C bus is released when IRQD, IRQE **and** IRQP are cleared. Only in this case can the next I^2C bus action take place.
Interrupt request bits may be set or cleared via software, e.g. to control the I^2C bus.

I^2C Address Register ICADR

Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRPMOD	**PREDIV**		**0**	**0**	**0**	**ICA9/0**	**ICA8IGE**	**ICA7..1**							**ICA0/0**

Field	Bits	Type	Value	Description
ICA0	0	rw	–	Node Address Bit 0 in 10-Bit Mode (See ICCON bit M10) Access is only possible in 10-bit mode.
–	0	0	0	Reserved read/write 0 if in 7-bit mode.
ICA7 … 1	7..1	rw	–	Node Address in 7-Bit Mode (ICA9 and ICA0 disregarded, ICA8 becomes IGE-bit).
IGE	8	rw	 0 1	Ignore IRQE In 7-bit mode, this bit becomes IGE-bit: Ignore IRQE (End of transmission) interrupt. The I^2C is stopped at IRQE interrupt. The I^2C ignores the IRQE interrupt. Access is only possible in 7-bit mode.
ICA8	8	rw	–	Node Address Bit 8 in 10-Bit Mode Access is only possible in 10-bit mode.
ICA9..0	9..0	rw	–	Node Address in 10-Bit Mode (all bits used). Note: Access is only possible in 10-bit mode.
PREDIV	[14:13]	rw	 00 01 10 11	Pre Divider for Baud Rate Generation pre divider is disabled pre divider factor 8 is enabled pre divider factor 64 is enabled reserved, do not use
BRPMOD	15	rw	 0 1	Baud Rate Prescaler Mode Mode 0 is enabled (by default) Mode 1 is enabled.

Baud Rate Selection

In order to give the user high flexibility in selection of CPU frequency and baud rate, without constraints to baud rate accuracy, a flexible baud rate generator has been

implemented. It uses two different modes and an additional pre divider. Low baud rates may be configured at high precision in mode 0 which is compatible with older versions. High baud rates may be configured precisely in mode 1.

Mode 0: Reciprocal Divider

The resulting baud rate is

$$B_{IIC} = \frac{f_{cpu}}{4 \cdot (BRP + 1)} \qquad BRP = \frac{f_{cpu}}{4 \cdot B_{IIC}} - 1$$

Mode 1: Fractional Divider

The resulting baud rate is

$$B_{IIC} = \frac{f_{cpu} \cdot BRP}{1024} \qquad BRP = \frac{1024 \cdot B_{IIC}}{f_{cpu}}$$

Table 7-24 I^2C-Bus Baud Rate Selection @ 33.33 MHz CPU-Freq.

BRPMOD	BRP @ 100 kBaud			BRP @ 400 kBaud		
	PREDIV = 00_B	PREDIV = 01_B	PREDIV = 10_B	PREDIV = 00_B	PREDIV = 01_B	PREDIV = 10_B
0	52_H	$0A_H$	1_H	14_H	02_H	–
1	03_H	–	–	$0C_H$	02_H	–

I^2C Receive/Transmit Buffer ICRTBH/L

Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICRTB3								ICRTB2							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICRTB1								ICRTB0							

Field	Bits	Type	Value	Description
ICRTBx **x = 3 … 0**	15..0	rwh	–	**Receive/Transmit Buffer** [1) 2)] The buffers contain the data to be sent/ received. The buffer size can be set in bit field CI (from 1 up to 4 bytes). ICRTB0 is sent/ received first.

1) A read respectively a write access (depending on bit TRX) to all bytes (specified in CI) of ICRTB0 … 3 sets CO to 111 (no byte sent/received).

2) If bit INT is set to zero and all bytes (specified in CI) of ICRTB0 … 3 are read/written (depending on bit TRX) IRQD is cleared.

Interrupts

Table 7-25 Interrupt Sources

Interrupt	SRC Register	Description
Data	I^2CTIC	Interrupt is requested after the acknowledge bit of the last byte has been received or transmitted.
Data Error	I^2CTIC	Interrupt is requested if a multi byte write could not be finished in slave mode because of missing acknowledge, then the data interrupt is followed by an end of transmission interrupt.
Protocol: Arbitration Lost	I^2CPIC	Interrupt is requested when the I^2C module has tried to become master on the bus but has lost the arbitration.
Protocol: Slave Mode after Lost Arbitration	I^2CPIC	Interrupt is requested if multimaster mode is selected and the I^2C module temporarily switches to slave mode after a lost arbitration.
Protocol: Slave Mode after Device Address	I^2CPIC	Interrupt is requested if multimaster mode is selected and the I^2C module temporarily switches to slave mode after a lost arbitration.
Data Trans-mission End after Stop Condition	I^2CTEIC	Interrupt is requested after transmission is finished by a stop condition.
Data Trans-mission End after RSC Condition	I^2CTEIC	Interrupt is requested after transmission is finished by a repeated start condition (RSC).
Data Transmission End after missing Acknowledge	I^2CTEIC	Interrupt is also requested if a transmission is stopped by a missing acknowledge.

Synchronization

In Mastermode, the SCL line is controlled by the I^2C Module. Sent and received data is only valid if SCL is high. With SCL going down, all modules are starting to count down their low period. During the low period all connected modules are allowed to hold SCL low. As the physical bus connection is wired-AND, SCL will remain low until the device with the longest low period enters high state. Then the device with the shortest high period will pull SCL low again.

Programming

It is strictly recommended not to write to the I^2C registers when the I^2C is working, except for interrupt handling. This is indicated by the BUM bit (in master mode) and the interrupt flags. All registers can be written in initial mode. In master mode the I^2C is working as long as the BUM bit is set, in slave mode the I^2C is working from receiving a start condition until receiving the next stop condition. Change of transmit direction is possible only after a protocol interrupt (IRQP) or in initialization mode (MOD = 00_B).

Initialization

Before data can be sent or received, data buffer size must be set in the count registers (only necessary if buffer greater than one byte is available). To decide if slave/master or multimaster mode is required, the MOD bits must be programmed.

Repeated Start Condition

The RSC bit must be set to one.

Start Condition

To generate a start condition the I^2C must be in master mode. If the BUM bit is set, a start condition is sent and the transmission started. The slave returns the acknowledge bit, which is indicated by the LRB bit.

Sending Data Bytes

To send bytes it is only necessary to write data bytes to the transmit buffer every time a data interrupt (IRQD) occurs.

Stop Condition

The BUM bit must be set to zero, or the STP bit must be set to one.

Receiving Data Bytes

To receive bytes it is necessary to set the TRX bit to zero. The bytes can be read after every data interrupt (IRQD). After a stop condition (protocol interrupt IRQE), the count

bit field CO must be read in case the buffer size (defined in CI) is greater than one byte, to decide which bytes in the receive buffer were received in the last transmission cycle.

7.6 Analog Digital Converter

M2 includes a four channel 8-bit ADC for control purposes. By means of these four input signals the controller is able to supervise the status of several analog signals and to take action if necessary. As these analog signals are fairly slow (compared to the video input), one SAR-Converter is used. The input is multiplexed to four different analog inputs.
The ADC is running continuously. The four channels are scanned one after another. The conversion results (one byte per channel), for the four channels, are stored in registers ADDAT1 and ADDAT2. After completion of the conversion for the last channel, two interrupt request flags ADC1IR and ADC2IR are generated.

The Peripheral Event Controller (PEC) may be used to automatically store the conversion results into a table in the memory for later evaluation, without requiring the overhead of entering and exiting interrupt routines for each data transfer.

The S&H circuit is open for about 2 µs. New results are available in ADDAT1 and ADDAT2 every 48 µs. The previous conversion results are overwritten unless the contents are transferred to the memory by PEC data transfers or an ordinary interrupt service routine.

If some of the port lines P5.0 to P5.3 are to be used as digital inputs the associated enable bits in register P5BEN have to be enabled.

The input voltage on port 5 should never exceed 2.5 V.

A set of SFRs and port pins provide access to control functions and results of the ADC.

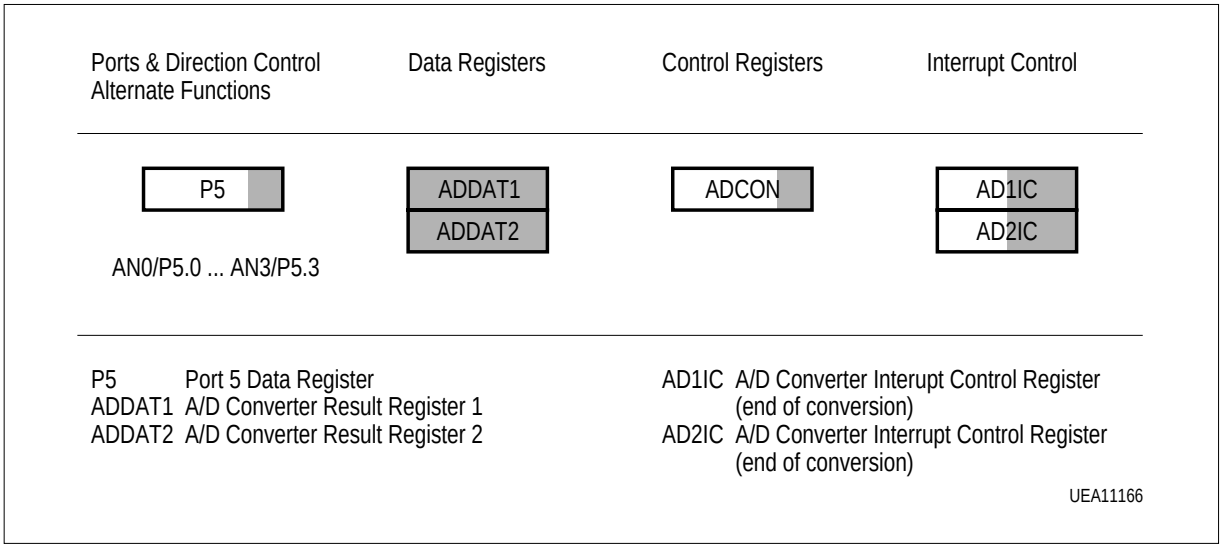


Figure 7-47 SFRs and Port Pins Associated with the A/D Converter

7.6.1 Power Down and Wake Up

As the power consumption of the ADC is quite high it should also be switched off during idle mode. Due to some application requirements it is necessary to include the possibility of generating an interrupt signal (ADWIC) as soon as the CADC (ANA0) input voltage falls below a predefined level. Two different levels are available. The first one corresponds to (fullscale-4 LSB) the second one to (fullscale-16 LSB). The actual level can be selected by a control bit (ADWULE).

7.6.2 Register Description

ADCON **Reset Value: 0000_H**

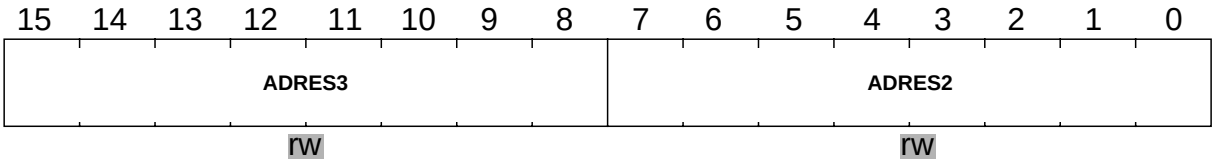
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	FS ADC DIFF	AD-WUL E	-	-	-	-
										rw	rw				

Bit	Function
FSADCDIFF	**Selects FSADC input range** Selects input range of the Full Service ADC: FSADCDIFF = 0: single-ended input FSDACDIFF = 1: differential input
ADWULE	**Defines threshold level for wake up** A special wake up unit has been implemented to allow a system wake-up as soon as the analog input signal on pin ANA0 falls below a predefined level. ADWULE defines this level. ADWULE = 0: threshold level corresponds to fullscale-4LSB. ADWULE = 1: threshold level corresponds to fullscale-16LSB.

ADDAT1 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRES1								ADRES0							
rw								rw							

ADDAT2 **Reset Value: 0000_H**



Bit	Function
$ADRES_i$ (7 … 0)	**A/D Conversion Result (8-bit) of Channel 0 … 3 (ANA 0 … 3)** For each A/D channel two successive 7-bit samples (@33.3 MHZ) are processed, averaged and scaled to 0 - 254.

Clock System

8 Clock System

8.1 General Function

The on-chip clock generator provides M2 with its basic clock signals. Its oscillator can either run with an external crystal and appropriate oscillator circuitry (refer to “Application Diagram”) or it can be driven by an external digital clock signal. For applications with low accuracy requirements (RTC is not used) the external oscillator circuit can also be a ceramic resonator. Depending on the absolute tolerance of the resonator the slicer may not work correctly. Moreover the display timings and baud rate prescaler have to be adapted in an appropriate way. In some applications the timing reference given by the horizontal frequency of the CVBS signal can be used to measure the timing tolerance and to adjust the programming.

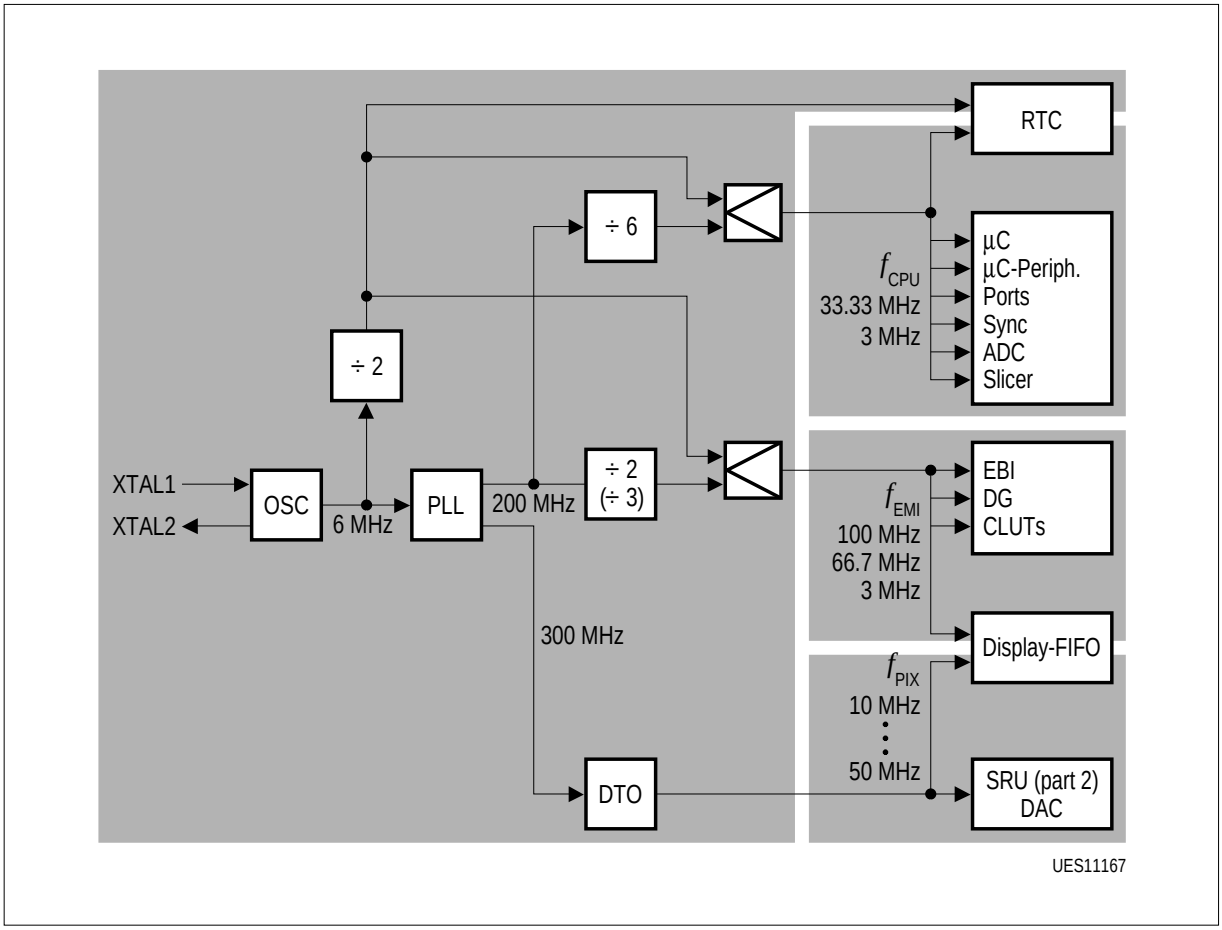


Figure 8-1 Clock System in M2

The on-chip phase locked loop (PLL), which is internally running at 600 MHz, is fed by the oscillator or can be bypassed to reduce the power consumption in idle and sleep mode. If it is not required to wake up immediately from idle mode, the PLL can be

switched off by entering sleep-mode. The same oscillator is used to clock the built-in RTC. (For a further description refer to **Chapter 7.2**.)

From the output frequency of the PLL three clock systems are derived:

One, the 33 MHz system clock (f_{CPU}) supplies the processor, all processor related peripherals, the sync timing logic, the A/D converters and the slicer.

The second clock system (100/66 MHZ) (f_{EBI}) is used to clock the external bus interface, the display generator, the CLUTs and the input part of the display FIFO. This clock starts at 66 MHz after hardware reset. It can be configured to 100 MHz during the initialization sequence.

The frequency of the latter clock system can be changed via bit CLKCON in register SYSCON2. The refresh rate of the external SDRAM is always kept constant, independent of the selected system clock frequency.

The third clock system runs the pixel clock (f_{PIX}), which is programmable in a range of 10 … 50 MHz. It serves the output part of the display FIFO and the D/A converters. The pixel clock is derived from the high frequency output of the PLL and it is phase shifted line by line to the positive edge of the horizontal sync signal (normal polarity). Because the final display clock is derived from a DTO (digital time oscillator) it has no equidistant clock periods although the average frequency is exact. This pixel clock generation system has several advantages:

- The frequency of the pixel clock can be programmed independently from the horizontal line period.
- Since the input of the PLL is already a signal with a high frequency, the resulting pixel frequency has an extremely low jitter.
- The resulting pixel clock follows the edge of the H-sync impulse without any delay and always has the same quality as the sync timing of the deflection controller.

8.2 Register Description

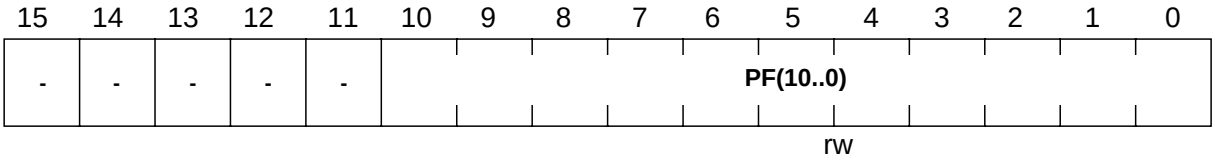
SYSCON2 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	-	-	-	-	-	-	CLK CON	-	-	-	-	-	-	-	-
r							rw								

Bit	Function
CLKCON	**Bus Clock Frequency** 0: f_{EBI} = 66 MHz 1: f_{EBI} = 100 MHz

Note: Register SYSCON2 cannot be changed after execution of the EINIT instruction.

PFR **Reset Value: 0148_H**



Bit	Function
PF (10 … 0)	**Pixel Frequency Factor** This register defines the relation between the output pixel frequency and the frequency of the crystal. The pixel frequency does not depend on the line frequency. It can be calculated by the following formula: $f_{PIX} = PF \times 300\ MHz / 8192$ The pixel frequency can be adjusted in steps of 36.6 KHz. After power-on, this register is set to 328_D. So, the default pixel frequency is set to 12.01 MHz. ***Note: Register values exceeding 1366 generate pixel frequencies which are outside of the specified boundaries.***

Sync System

9 Sync System

9.1 General Description

The display sync system is completely independent of the acquisition sync system (CVBS timing) and can either work as a sync master or as a sync slave system. Any mention of “H/V-Syncs” in this chapter and in **Chapter 10** always refers to display related H/V Syncs and never to CVBS related sync timing.

In sync slave mode, M2 receives the synchronisation information from two independent pins which deliver separate horizontal and vertical signals. Due to the not-line-locked pixel clock generation (refer to **Chapter 8**), it can process any possible horizontal and vertical sync frequency.

In sync master mode, M2 delivers separate horizontal and vertical signals with the same flexibility in the programming of their periods as in sync slave mode.

9.1.1 Screen Resolution

The number of displayable pixels on the screen is defined by the pixel frequency (which is independent of horizontal frequency), the line period and number of lines within a field. The screen is divided into 3 different regions:

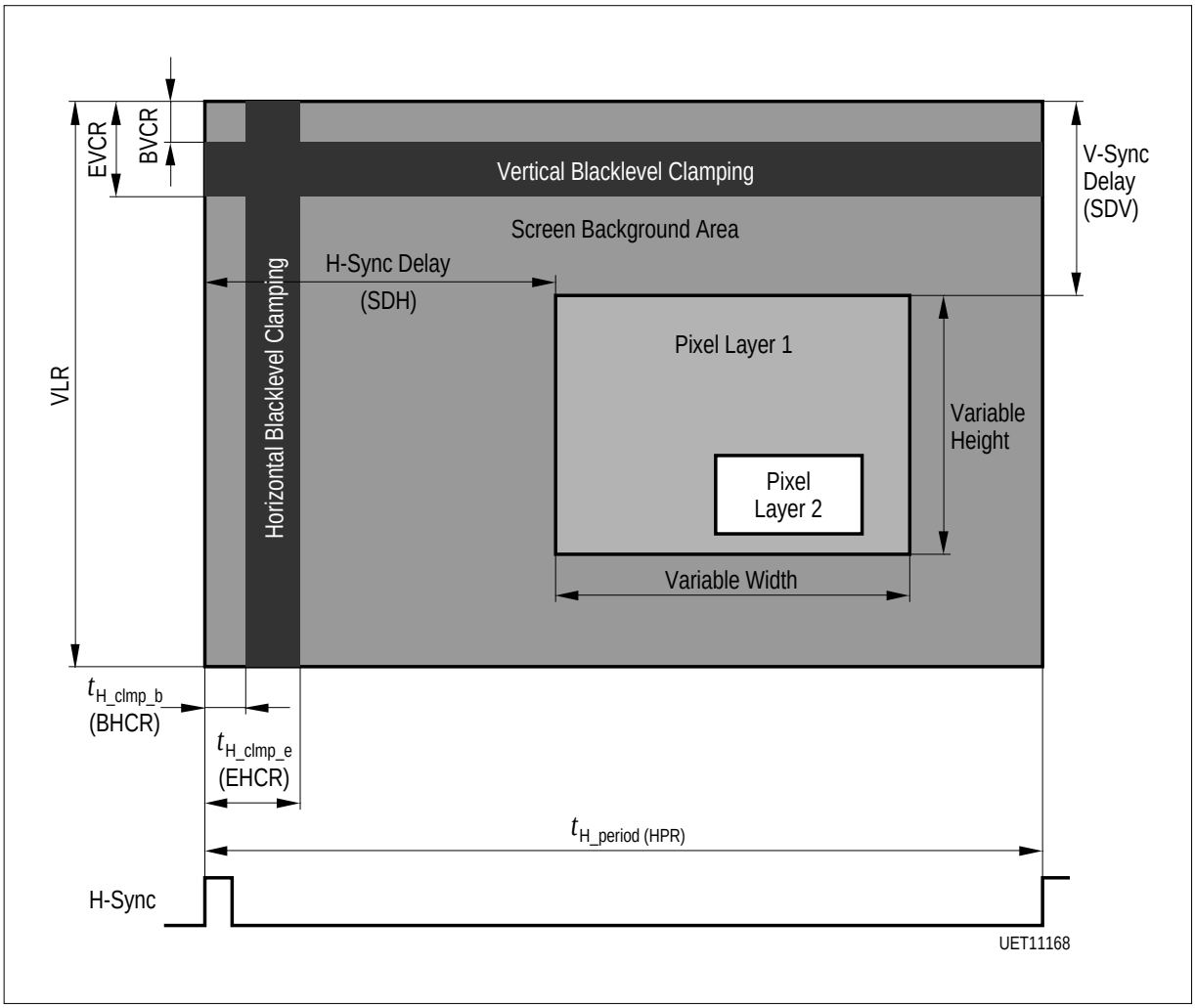


Figure 9-1 M2's Display Timing

Blacklevel Clamping Area

During horizontal and vertical blacklevel clamping, the black value (RGB = 000) is delivered. The blank pin is set to '1' and COR is set to '0' (normal polarity assumed). This area is vertically programmable (in terms of lines) and horizontally in terms of 33.33 MHz clock cycles. These programmings are independent of all other registers.

Screen Background Area

The size of that area is defined by the sync delay registers (SDH and SDV) and the size of pixel layer1. The contents of that area are defined by GA instruction SAR (refer also to **Chapter 10.1**). Pixels within that area are programmable (colour and transparency level), but all have the same value.

Pixel Layer Area

Pixels of this area are freely programmable according to the specifications of the display generator. The information is stored in the frame buffer in the external memory, that means the bigger that area is defined, the more bus performance is needed for SRU. If that area is set to ‘0’ no bus performance is needed. The start position of that area can be shifted in horizontal and vertical direction by programming the horizontal and vertical sync delay registers (SDH and SDV). The size of that area is defined by the instruction FSR in the display generator.

Registers which allow the screen and sync parameters to be set up, are given in the **Table 9-1**.

Table 9-1 Overview on Sync Register Settings

Parameters	Register	Min Value	Max Value	Step	Default
Sync Control Register	SCR	see below			
VL - Lines / Field	VLR	1 line	1024 lines	1 line	625 lines
$T_{h\text{-}period}$ - Horizontal Period	HPR	15 µs	100 µs	30 ns	64 µs
F_{pixel} - Pixel Frequency	PFR	10 MHz	50 MHz	73.25 KHz	12.01 MHz
T_{vsync_delay} - Sync Delay	SDV	4 lines	1024 lines	1 line	32 lines
T_{hsync_delay} - Sync Delay	SDH	32 pixel	2048 pixel	1 pixel	72 pixel
BVCR - Beginning Of Vertical Clamp Phase	BVCR	1 line	1024 lines	1 line	line 1
EVCR - End Of Of Vertical Clamp Phase	EVCR	1 line	1024 lines	1 line	line 5
$T_{h\_clmp\_b}$ - Beginning Of Horizontal Clamp Phase	BHCR	0 µs	163.2 µs	480 ns	0 µs
$T_{h\_clmp\_e}$ - End Of Horizontal Clamp Phase	EHCR	0 µs	163.2 µs	480 ns	4.5 µs

9.1.2 Sync Interrupts

The sync unit delivers interrupts (Horizontal and vertical interrupt) to the controller to support the recognition of the frequencies of an external sync source (e.g. a VGA source via SCART). These interrupts are related to the positive edge of the non delayed horizontal and vertical impulses which can be seen at pins HSYNC and VSYNC.

9.2 Register Description

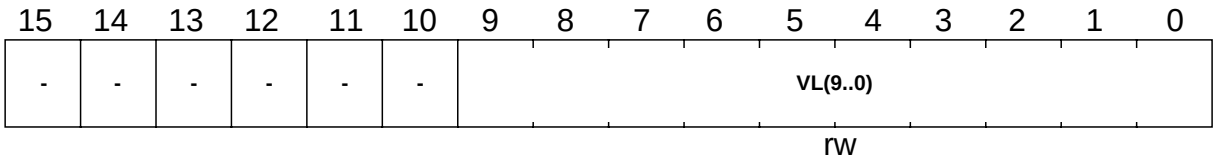
SCR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	COR-BL	VSU(3..0)				BLANKP	CORP	HP	VP	INT	VCS	MAST
				rw	rw				rw	rw	rw	rw	rw	rw	rw

Bit	Function
MAST	**Master / Slave Mode** This bit defines the configuration of the sync system *(master or slave mode)* and also the direction (input/output) of the V, H pins. 0: Slave mode. H, V pins are configured as inputs. 1: Master mode. H, V pins are configured as outputs. *Note: Switching from slave to master mode resets the internal H, V counters, so that the phase shift during the switch can be minimized. In slave mode registers VLR, and HPR are without any use.*
VCS	**Vertical Composite Sync** VCS defines the sync output at pin V (*Master mode only*). 0: At pin V the vertical sync appears. 1: At pin V a composite sync signal (including equalizing pulses, H-Sync and V-Syncs) is generated (VCS). The length of the equalizing pulses have fixed values as described in the timing specifications. *Note: Don't forget to set registers VLR and HPR according to your requirements.*
INT	**Interlace / Non-interlace** M2 can either generate an interlaced or a non-interlaced timing. *(Master mode only).* Interlaced timing can only be created if VLR is an odd number. 0: Interlaced timing is generated. 1: Non-interlaced timing is generated.
VP	**V-Pin Polarity** This bit defines the polarity of the V pin *(master and slave mode).* 0: Normal polarity (active high). 1: Negative polarity.

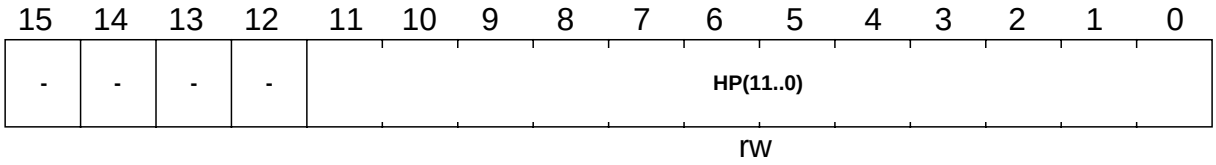
Bit	Function
HP	**H-Pin Polarity** This bit defines the polarity of the H pin. *(Master and slave mode).* 0: Normal polarity (active high). 1: Negative polarity.
CORP	**COR-Pin Polarity** This bit defines the polarity of the COR pin. *(Master and slave mode).* 0: Normal polarity (active high). 1: Negative polarity (not allowed for CORBL = 1).
BLANKP	**BLANK-Pin Polarity** This bit defines the polarity of the BLANK pin. *(Master and slave mode).* 0: Negative polarity (not allowed for CORBL = 1). 1: Normal polarity (active high).
VSU (3 … 0)	**Vertical Set Up Time.** *(Slave mode only)* The vertical sync signal is internally sampled with the next edge of the horizontal sync edge. The phase relation between V and H differs from application to application. To guarantee (vertical) jitter free processing of external sync signals, the vertical sync impulse can be delayed before it is internally processed. The following formula shows how to delay the external V-sync before it is internally latched and processed. $t_{V\_delay} = 3.84\ \mu s \times VSU$
CORBL	**3-Level Contrast Reduction Output** There is one pin each for BLANK and COR. Nevertheless by means of CORBL the user is able to switch the COR signal to a three level signal providing BLANK and contrast reduction information on Pin BLANK simultaneously. 0: Two level signal for contrast reduction. 1: Three level signal Level0: BLANK off; COR off. Level1: BLANK off; COR on. Level2: BLANK on; COR off. *Note: Please refer to **Chapter 14** for the detailed specification of these levels.*

VLR **Reset Value: 0271_H**



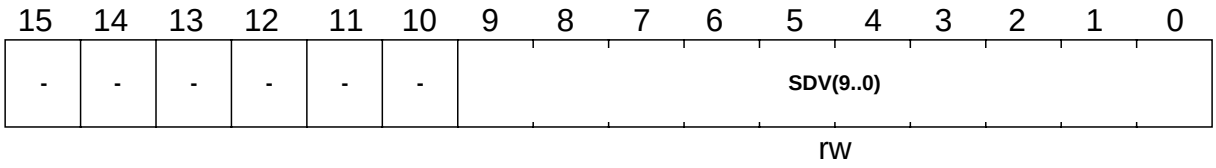
Bit	Function
VLR (9 … 0)	**Amount of Vertical Lines in a Frame.** (*Master mode only*). M2 generates vertical sync impulses in sync master mode. If, for example, a normal PAL timing should be generated, set the register to ‘625d’ and set the interlace bit to ‘0’. The hardware will generate a vertical impulse periodically after 312.5 lines. If a non-interlaced picture with 312 lines should be generated, set this register to ‘312’ and set the interlace bit to ‘1’. The hardware will generate a vertical impulse every 312 lines. Progressive timing can be generated by setting VLR to ‘625’ and interlace to ‘1’.

HPR **Reset Value: 855_H**



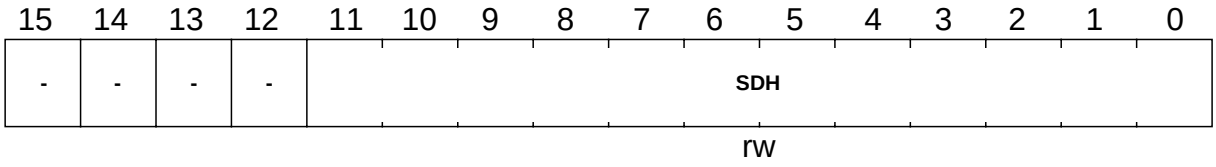
Bit	Function
HPR (11 … 0)	**Horizontal Period factor.** (*Master mode only*) This register allows the period of the horizontal sync signal to be adjusted. The horizontal period is independent of the pixel frequency and can be adjusted with the following resolution: $t_{H\text{-period}} = HP \times 30\ ns$

SDV **Reset Value: 0020_H**



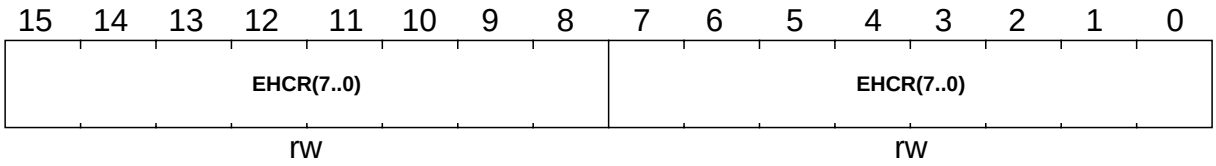
Bit	Function
SDV (9 … 0)	**Vertical Sync Delay.** *(Master and slave mode).* This register defines the delay (in lines) from the vertical sync to the first line of pixel layer 1 on the screen.

SDH **Reset Value: 0020_H**



Bit	Function
SDH (11 … 0)	**Horizontal Sync Delay.** *(Master and slave mode).* This register defines the delay (in pixels) from the horizontal sync to the first pixel of layer 1 on the screen.

HCR **Reset Value: $0A00_H$**



Bit	Function
BHCR (7 … 0)	**Beginning of Horizontal Clamp Phase.** *(Master and slave mode).* This register defines the delay of the horizontal clamp phase from the positive edge of the horizontal sync impulse (normal polarity is assumed). The beginning of the clamp phase can be calculated by the following formula: $t_{H\_clmp\_b} = 480\ ns \times BHC$
EHCR (7 … 0)	**End of Horizontal Clamp Phase.** *(Master and slave mode).* This register defines the end of the horizontal clamp phase from the positive edge of the horizontal sync impulse (at normal polarity). The end of the clamp phase can be calculated by the following formula: $t_{H\_clmp\_e} = 480\ ns \times EHC$ If EHC is smaller than BHC the clamp phase will include the H-sync phase.

The clamp phase area has higher priority than the screen background area or the pixel layer area and can be shifted independent from any other register.

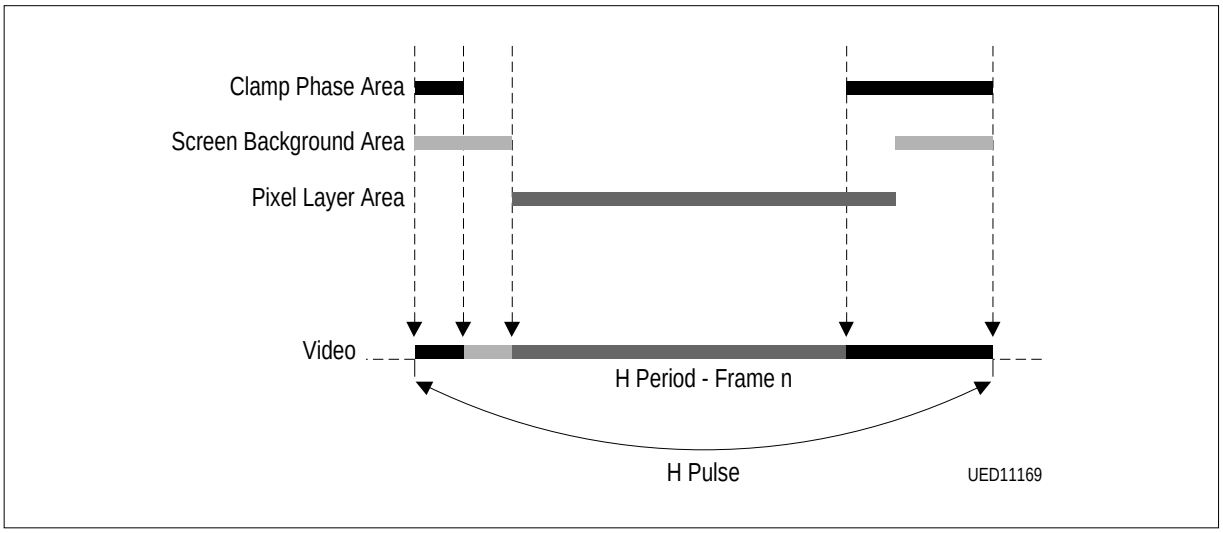


Figure 9-2 Priority of Clamp Phase, Screen Background and Pixel Layer Area

BVCR **Reset Value: 0000_H**

BVCR1								BVCR0							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	BVCR(9..0)									
															rw

Bit	Function
BVCR (9 … 0)	**Beginning of Vertical Clamp Phase.** *(Master and slave mode).* This register defines the beginning of the vertical clamp phase from the positive edge of the vertical sync impulse (at normal polarity) in line count.

EVCR **Reset Value: $000A_H$**

EVCR1								EVCR0							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	EVCR(9..0)									
															rw

Bit	Function
EVCR (9 … 0)	**End of Vertical Clamp Phase.** *(Master and slave mode).* This register defines the end of the vertical clamp phase from the positive edge of the vertical sync impulse (at normal polarity) in line count. *Note: It must be guaranteed that the value EVCR is always smaller than the value of SDV.*

Display Generator

10 Display Generator

10.1 General Description

M2's display concept is based on frame buffer technology, which means that for each pixel displayed on a screen appropriate information is stored in the memory of the so called frame buffer. To relieve the controller from processing time consuming tasks like writing this frame buffer pixel by pixel, a graphic accelerator machine is introduced (GA). The GA reads e.g. bitmap information in various formats together with attributes which define the final behavior of those bitmaps. Depending on these attributes these bitmaps are processed and written into the frame buffer. Due to the processor like architecture of the GA, it is controlled by so called GAIs (graphic accelerator instructions). The screen refresh unit (SRU) reads out the pixel based information (various formats are also available here) and hands them over via a look up table and a FIFO to the D/A converter. The FIFO is used to adapt the variable pixel output frequency to the fixed memory bus frequency. Up to two frame buffers are possible and supported by GA and SRU.

10.2 Screen Alignments

Two HW-layers are supported: layer 1 and layer 2. Layer 2 can be positioned relative to layer 1 (also in negative direction). If layer 2 exceeds the dimensions of layer 2 these exceeding parts are not visible on the screen.

The alignment of the OSD on the screen depends on the configuration of layer 1.

The maximum amount of displayable pixels is 2047 pixels in horizontal direction and 1023 pixels in vertical direction.

To adapt M2 to a wide range of displays in the market the sync-processing can be flexibly configured.

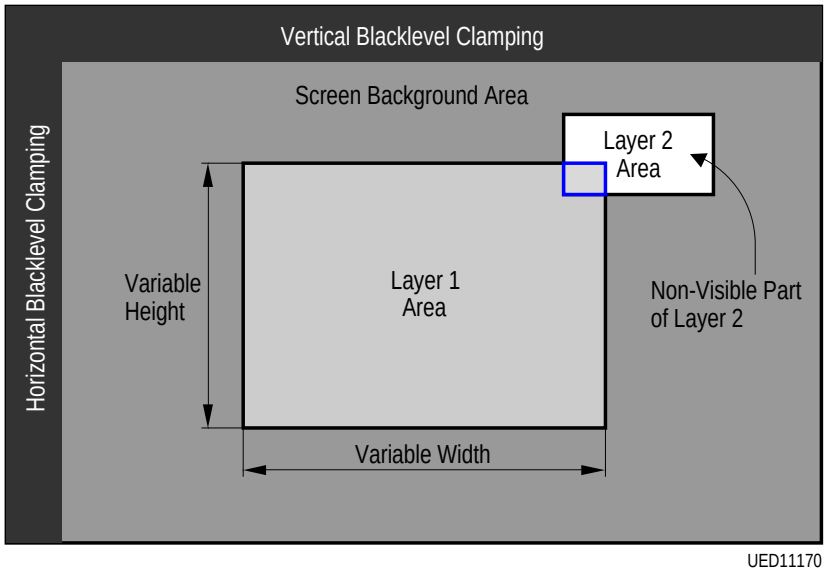


Figure 10-1 Display Regions and Alignments

There are three registers in the synchronization unit which are necessary for OSD setup:

- **SDH:** used to setup the horizontal position of the top left pixel of layer 1.
- **SDV**: used to setup the vertical position of the top left pixel of layer 1.
- **PFR:** used to setup the pixel frequency.

For detailed description of these registers please refer to chapter ‘Display Sync System’ and ‘Clock System’.

In the area which is defined for layer 1 or layer 2 (layer area) each pixel is defined by the attribute definition of the frame buffer. There is no pixel by pixel definition for the blacklevel clamping area and the screen background area. For these areas the colour and transparency is defined as follows:

Transparency level and colour of the screen background area is defined globally for the whole screen by GA instruction SAR.

During the blacklevel clamping area, black value (RGB = ‘000’) is delivered at RGB output. Pin ‘Blank’ is set to ‘1’ and COR-pin is set to ‘0’ (normal polarity is assumed).

10.3 Layer Concept

M2 supports two HW-layers. Frame buffers of layer 1 and layer 2 can be placed at any word aligned position in external memory.

Two different layer modes can be chosen:

- Overlapped layers. Layer 1 and layer 2 are processed in parallel by the screen refresh unit.
- Embedded layers. Layer 1 and layer 2 are alternatively processed.

If the area of layer 2 exceeds the area of layer 1, these parts are not visible on the screen.

Mixing of layer 1 and layer 2 is performed by the display generator. As a result of the RGB output of M2, there is only one RGB stream which contains the information of layer 1 and layer 2. This RGB stream is externally mixed with a video source. For this external mixing there are two output signals (COR and BLANK) available.

Meshed areas:

A special mesh mode is defined to mix the external video with the RGB information from M2 in a chess-pattern-shape. From frame to frame this chess-pattern is inverted. Inverted means, that pixels which have been displayed as video in the previous frame, are displayed as RGB in the following frame.

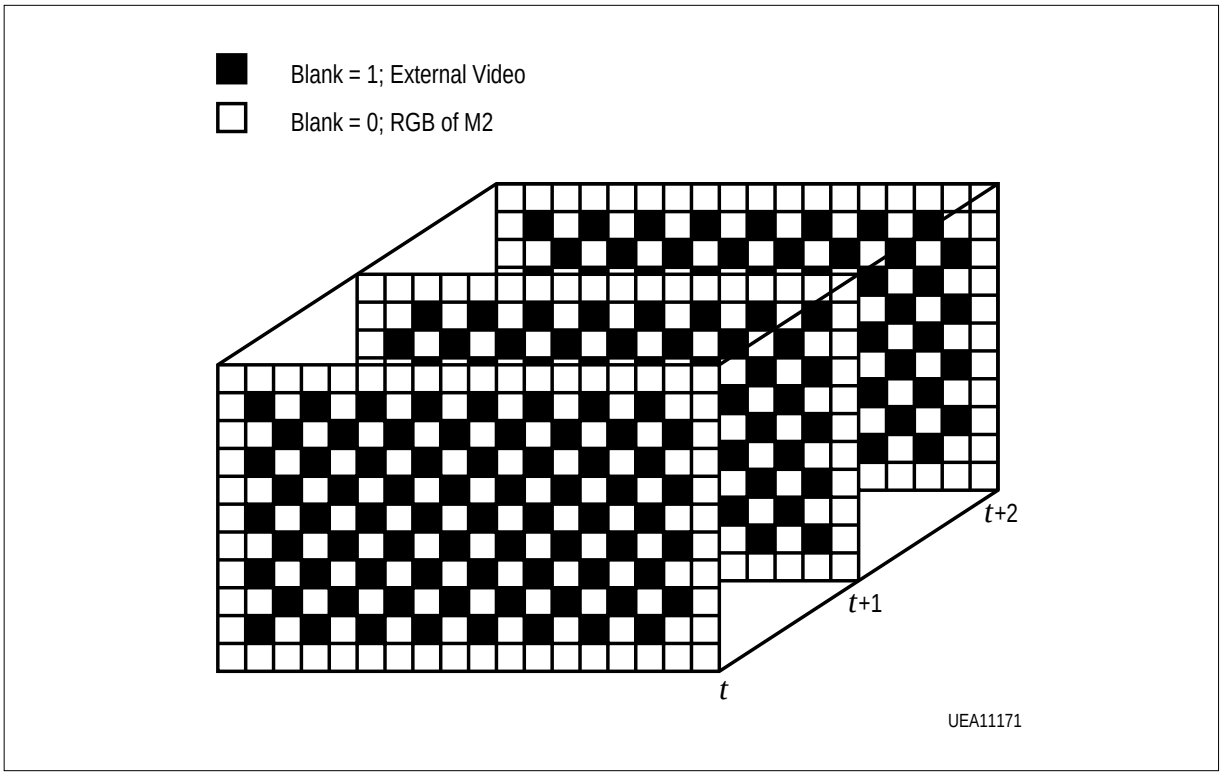


Figure 10-2 Behavior of Blank Pin for Consecutive Frames in ‘Meshed’ Regions

10.3.1 Overlapped Layers

In overlapped layer mode the pixel information of both layers (layer 1, layer 2) is read in parallel to the RAM. This means that for each pixel the individual decision can be made which pixel source (layer 1, layer 2, screen background or video) has the highest priority.

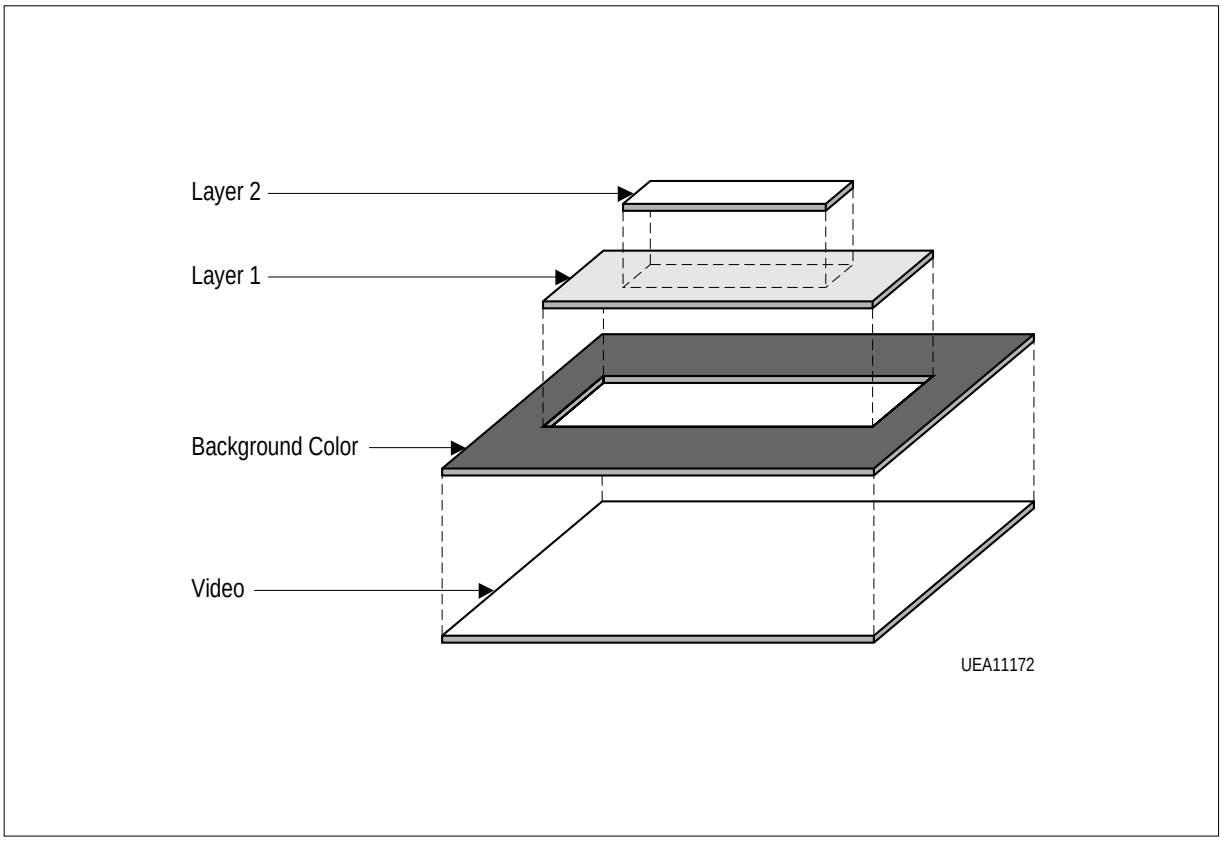


Figure 10-3 Priority of Layers in Overlapped Layer Mode

The transparency between layers is supported. Below layer 2 there is layer 1, below layer 1 there is the screen background colour and below screen background there is video. In overlapped layer mode a transparency hierarchy is defined for layer 2, layer 1, screen background and video. The transparency hierarchy is controlled, for each pixel, by two bits (TR1 … 0) which are defined in the pixel format of the framebuffer and bit SBTL which is defined in instruction SAR. Thus each pixel position can be defined individually as layer 1, layer 2, screen background, video or contrast reduced video.

Depending on the transparency bits of both layers, subsequent signals are switched to the RGB, COR and BLANK (normal polarity assumed) outputs of M2. As a result one of the two layers or the screen background will be visible on the screen. If layer 2 is not available for a pixel, signals COR, BLANK and RGB output depends on layer 1 only.

Table 10-1 Behavior of M2's Outputs in Overlapped Layer Mode

Layer 1		Layer 2		Screen Background	BLANK Pin	COR Pin	RGB Pins	RGB Tube
TR1	TR0	TR1	TR0	SBTL				
0	0	n.a.[*1)]	n.a.	0	0	0	Layer 1	Layer 1
0	1	n.a.	n.a.	0	Meshed	0	Layer 1	Layer 1/ Video
1	0	n.a.	n.a.	0	0	0	Back-ground	Back-ground
1	1	n.a.	n.a.	0	0	0	Back-ground	Back-ground
X[*2)]	X	0	0	0	0	0	Layer 2	Layer 2
X	X	0	1	0	Meshed	0	Layer 2	Layer 2/ Video
0	0	1	X	0	0	0	Layer 1	Layer 1
0	1	1	X	0	Meshed	0	Layer 1	Layer 1/ Video
1	0	1	X	0	0	0	Back-ground	Back-ground
1	1	1	X	0	0	0	Back-ground	Back-ground
0	0	n.a.	n.a.	1	0	0	Layer 1	Layer 1
0	1	n.a.	n.a.	1	Meshed	0	Layer 1	Layer 1/ Video
1	0	n.a.	n.a.	1	1	0	Back-ground	Video
1	1	n.a.	n.a.	1	1	1	Back-ground	Contrast red. Video
X	X	0	0	1	0	0	Layer 2	Layer 2
X	X	0	1	1	Meshed	0	Layer 2	Layer 2/ Video
0	0	1	X	1	0	0	Layer 1	Layer 1
0	1	1	X	1	Meshed	0	Layer 1	Layer 1/ Video

Table 10-1 Behavior of M2's Outputs in Overlapped Layer Mode (cont'd)

Layer 1		Layer 2		Screen Background	BLANK Pin	COR Pin	RGB Pins	RGB Tube
TR1	**TR0**	**TR1**	**TR0**	**SBTL**				
1	0	1	X	1	1	0	Back-ground	Video
1	1	1	X	1	1	1	Back-ground	Contrast red. Video

*1) n.a. = not available

*2) X = don't care

For transparency in 'screen background area' please refer to **Chapter 10.3.3**.

10.3.2 Embedded Layers

Layer 1 and 2 are not read in parallel to the RAM. The SRU reads only one layer at a time. During the area of layer 2 only pixels of layer 2 are read. Otherwise only layer 1 is read. As a result in the area of layer 2, the pixel information for layer 1 is not available. This is why layer 2 is transparent to video and not to layer 1. As a result, in embedded layer mode, transparency between layers is only supported layer-wise not pixel-wise. Also please refer to transparency using overlapped layers (**Chapter 10.3.1**).

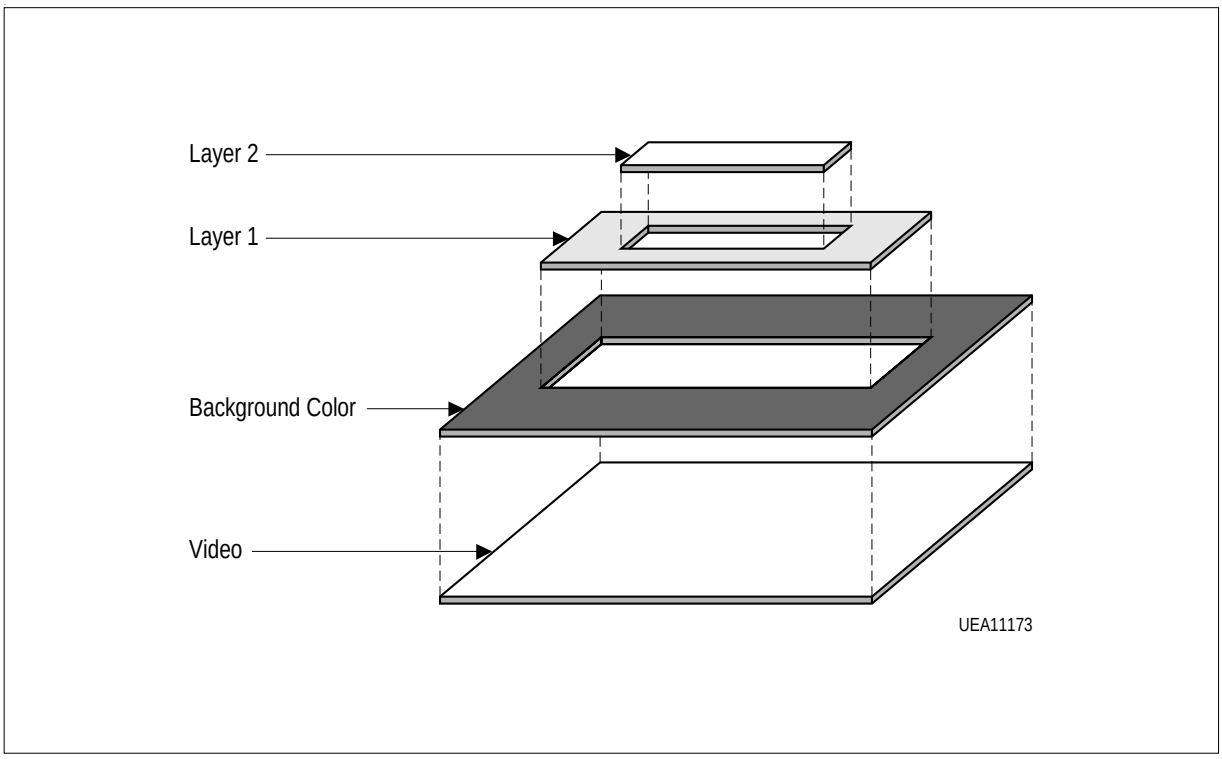


Figure 10-4 Priority of Layers in Embedded Layer Mode

Depending on the transparency bits of both layers, the following signals are switched to the RGB, COR and BLANK (normal polarity assumed) outputs of M2. As a result, one of the two layers or the screen background will be visible on the screen. If layer 2 is not available for a pixel, signals COR, BLANK and RGB output depends only on layer 1.

The transparency hierarchy is again controlled, for each pixel, by two bits (TR1 … 0) which are defined in the pixel format. On the RGB output of M2 there is a mix of layer 1 and layer 2 RGB stream with the screen background.

Depending on the transparency bits of one of the layers subsequent signals are switched to RGB, COR and BLANK (normal polarity assumed).

The output signals of M2 (COR, BLANK, RGB) depend only on transparency bits of layer 1 **or** on layer 2 and never on both layers because there is no pixel position where both layers are available in parallel.

10.3.3 Transparency in Screen Background Area

Depending on the height and width definition of layer 1 there is a remaining portion of visible screen area without any pixel definition of layer 1 or layer 2. For this area the transparency bits for layer 1 and layer 2 are not available. For this ‘Screen Background Area’ the colour attributes and transparency attributes are globally defined for the whole screen. Also refer to instruction SAR described in **Chapter 10.7.1**.

Table 10-2 Behavior of M2's Outputs in Embedded Layer Mode

Layer 1		Layer 2		Screen Back-ground	BLANK Pin	COR Pin	RGB Pins	RGB Tube
TR1	TR0	TR1	TR0	SBTL				
0	0	n.a.	n.a.	0	0	0	Layer 1	Layer 1
0	1	n.a.	n.a.	0	Meshed	0	Layer 1	Layer 1 / Video
1	0	n.a.	n.a.	0	0	0	Back-ground	Back-ground
1	1	n.a.	n.a.	0	0	0	Back-ground	Back-ground
n.a.	n.a.	0	0	0	0	0	Layer 2	Layer 2
n.a.	n.a.	0	1	0	Meshed	0	Layer 2	Layer 2 / Video
n.a.	n.a.	1	0	0	0	0	Back-ground	Back-ground
n.a.	n.a.	1	1	0	0	0	Back-ground	Back-ground
0	0	n.a.	n.a.	1	0	0	Layer 1	Layer 1
0	1	n.a.	n.a.	1	Meshed	0	Layer 1	Layer 1 / Video
1	0	n.a.	n.a.	1	1	0	Back-ground	Video
1	1	n.a.	n.a.	1	1	1	Back-ground	Contrast red. Video
n.a.	n.a.	0	0	1	0	0	Layer 2	Layer 2
n.a.	n.a.	0	1	1	Meshed	0	Layer 2	Layer 2 / Video
n.a.	n.a.	1	0	1	1	0	Back-ground	Video
n.a.	n.a.	1	1	1	1	1	Back-ground	Contrast red. Video

Table 10-3 Behavior of M2's Outputs in Background Area

Screen Background		BLANK Pin	COR Pin	RGB Pins	RGB Tube
STR1	STR0				
0	0	0	0	RGB values defined in SAR	RGB values defined in SAR
0	1	Meshed	0	RGB values defined in SAR	RGB values defined in SAR/Video
1	0	1	0	RGB values defined in SAR	Video
1	1	1	1	RGB values defined in SAR	Contrast red. Video

10.4 Input and Output Formats

The transfer of one memory area to another is executed by the graphic accelerator (GA). Transfer means reading data from a source memory area in a given input format, modifying the data and writing it in a defined output format to the destination memory area. Different input and output formats are supported:

Table 10-4 Overview on Formats

Input Formats	Output Formats
1-bit bitmap	2-bit format CLUT2 vector[1)]
2-bit bitmap	8-bit format CLUT2 vector
4-bit bitmap	16-bit format (4:4:4:2) RGB
8-bit bitmap	16-bit format (TTX) CLUT2 vector
8-bit data (for direct data transfer)	16-bit format (5:6:5) RGB
CLUT1 input	8-bit data (for direct data transfer)
16-bit format (4:4:4:2)	–

Note: [1)] The 2-bit format is defined as a format for the frame buffer but not supported by the GA.

10.4.1 Input Formats

The following figures describe how bitmaps for different input bit map formats are stored in the source memory area . The 16-bit format is described in **Figure 10-12**.

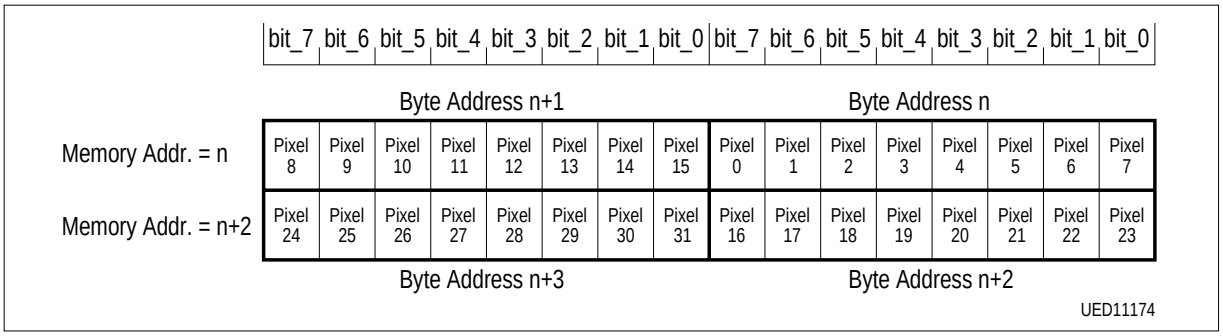


Figure 10-5 Format of 1-bitplane Bitmap

Note: The 1-bitplane format is used to address vectors 1 … 0 of CLUT1.

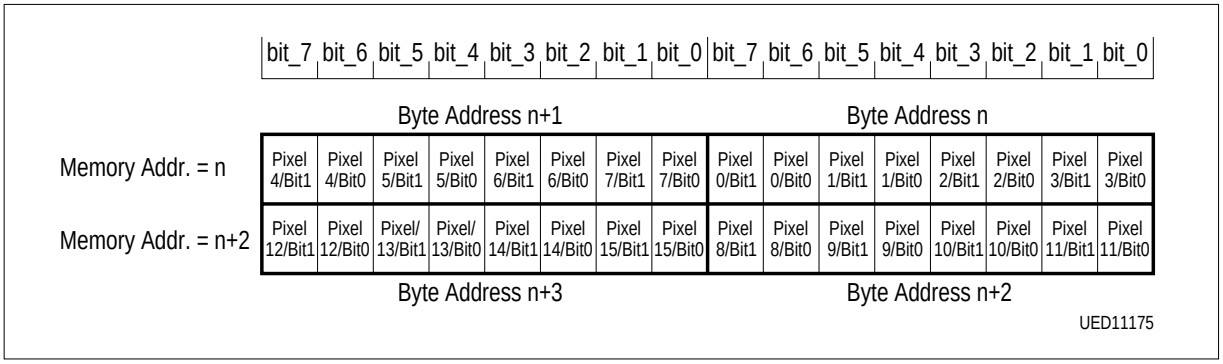


Figure 10-6 Format of 2-bitplane Bitmap

Note: The 2-bitplane format is used to address vectors 3 … 0 of CLUT1.

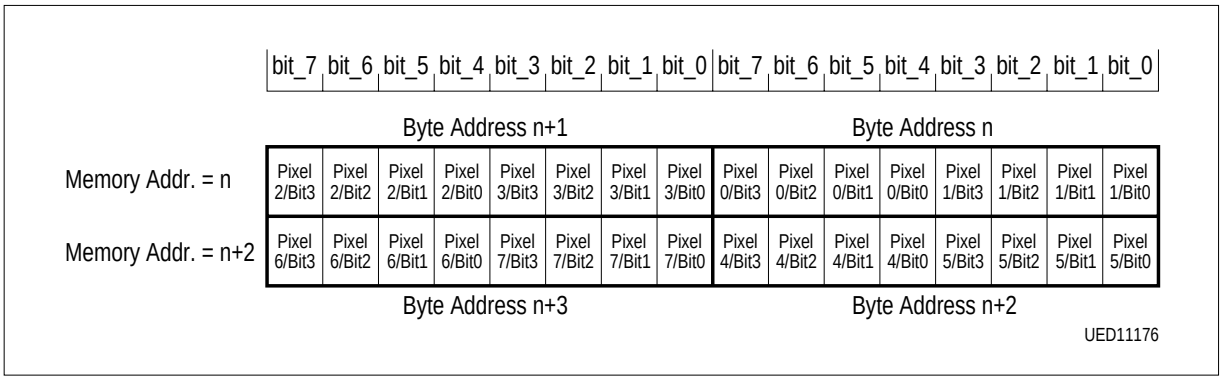


Figure 10-7 Format of 4-bitplane Bitmap

Note: The 4-bitplane format is used to address vectors 15 … 0 of CLUT1.

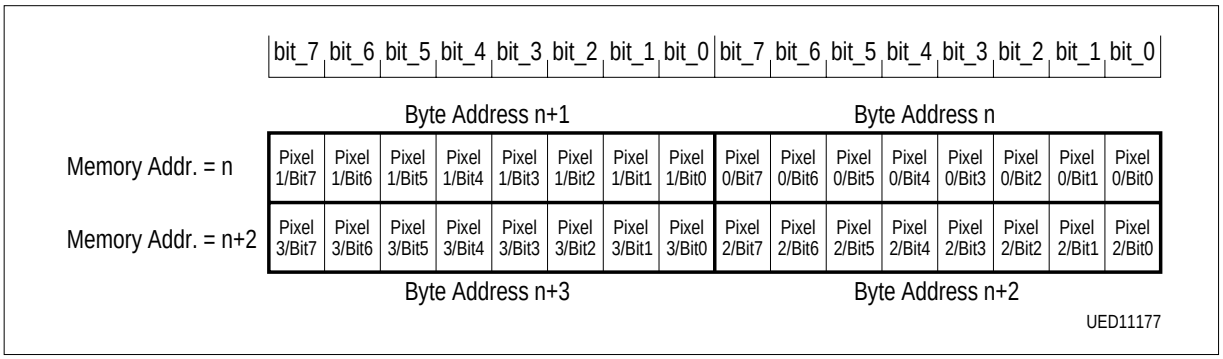


Figure 10-8 Format of 8-bitplane Bitmap

Note: The 8-bitplane format is used to address vectors 255 … 0 of CLUT1.

10.4.2 Output Formats

The output of the GA is the input of the SRU. The SRU contains a 256 × 14-bit colour look up table (CLUT2). This CLUT2 contains 256 different RGB values with 4 bits for each colour (4:4:4) and 2-bit transparency information.

2-bit formats, 8-bit formats and the TTX format are using this CLUT2. For 16-bit formats (TTX or 4:4:4:2) it depends on bit ‘M’ (mode), if the 16-bit information is used as colour look up vectors or the 16-bit information is bypassing CLUT2. The 5:6:5 format always bypasses CLUT2.

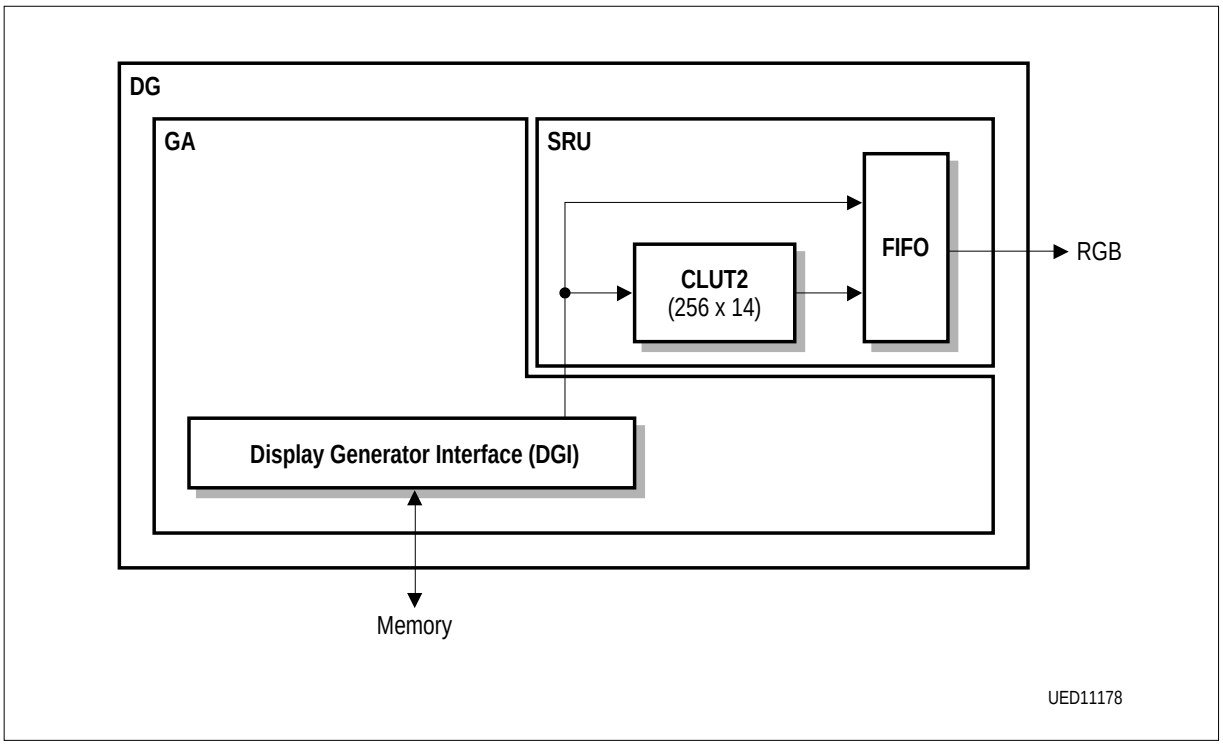


Figure 10-9 Overview on SRU

The different formats of pixels stored in a frame buffer which are used by the SRU are described in the following paragraphs:

Frame Buffer in 2-bit Pixel Format

Using this format, the frame buffer contains colour vectors. These 2-bitplane colour vectors will be converted into 4:4:4:2 format (R:G:B: transparency value) by addressing vector 252 … 255 of CLUT2.

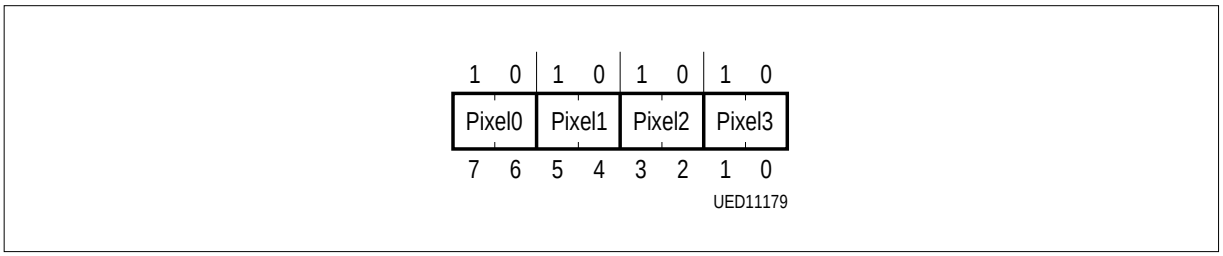


Figure 10-10 2-bit Pixel Format for Use in Frame Buffer

Frame Buffer in 8-bit Pixel Format

Using this format, the frame buffer contains colour vectors. These 8-bitplane colour vectors will be converted into 4:4:4:2 format (R:G:B: transparency value) by CLUT2.

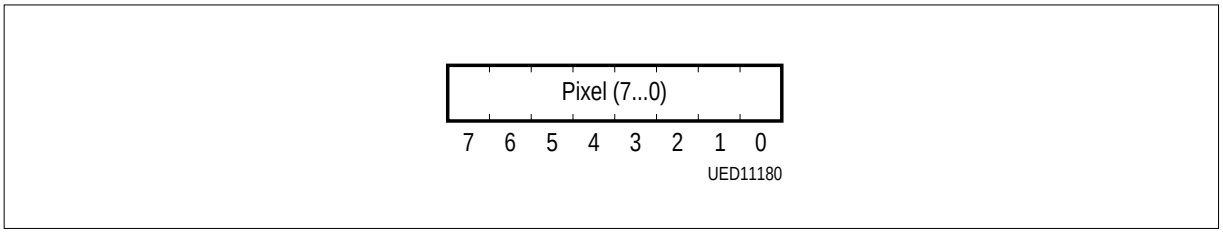


Figure 10-11 8-bit Pixel Format for Use in Frame Buffer

Frame Buffer in 16-bit Pixel Format (4:4:4:2 or TTX)

This 16-bit format supports an RGB mode (4:4:4:2) and a TTX mode. Which of these modes is in use can be decided pixel by pixel with bit ‘M’:

- TTX format (M = ‘0’)
- 12-bitplanes RGB (4:4:4:2) format (M = ‘1’) with 2 transparency bits TR(1 … 0).

*Note: The meaning of transparency bits is described in more detail in **Chapter 10.3.1** and **Chapter 10.3.2**.*

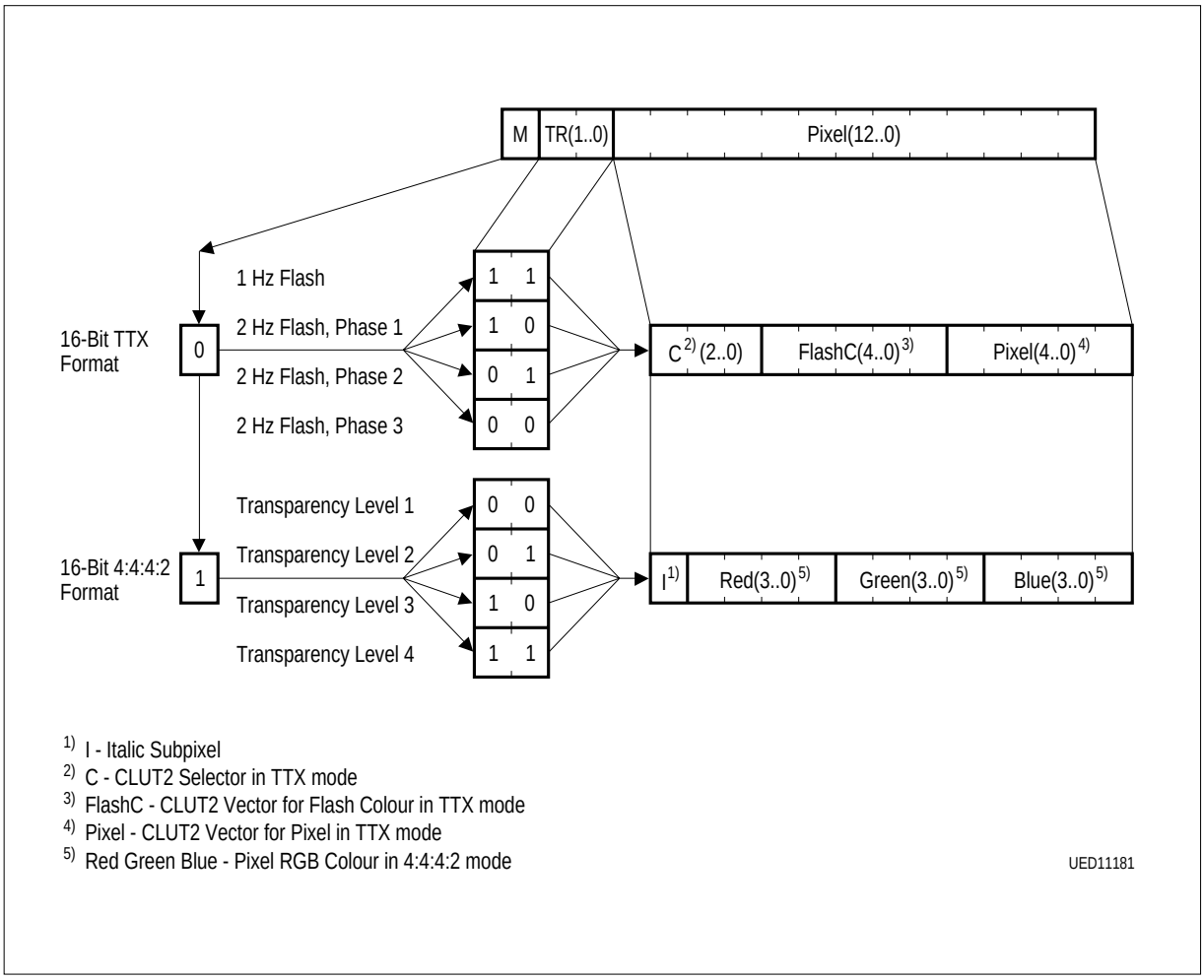


Figure 10-12 16-bit Pixel Format (4:4:4:2/TTX) for Use in Frame Buffer

Pixels in 16-bit format which are stored in TTX format contain two 5-bit colour look up vectors and two flash mode indicator bits. The flash mode indicator bits are used to choose the flash rate and the flash phase. The meaning of flash is: The colour alternates between two 5-bit colour vectors (FlashC and Pixel) which are chosen within the format definition.

- **Non Flash**: Flash can be disabled if both 5-bit colour vectors point to the same CLUT2 location.
- **Inverted Flash**: Inverted flash is supported by exchanging the ‘FlashC’ vector with the ‘Pixel’ vector.

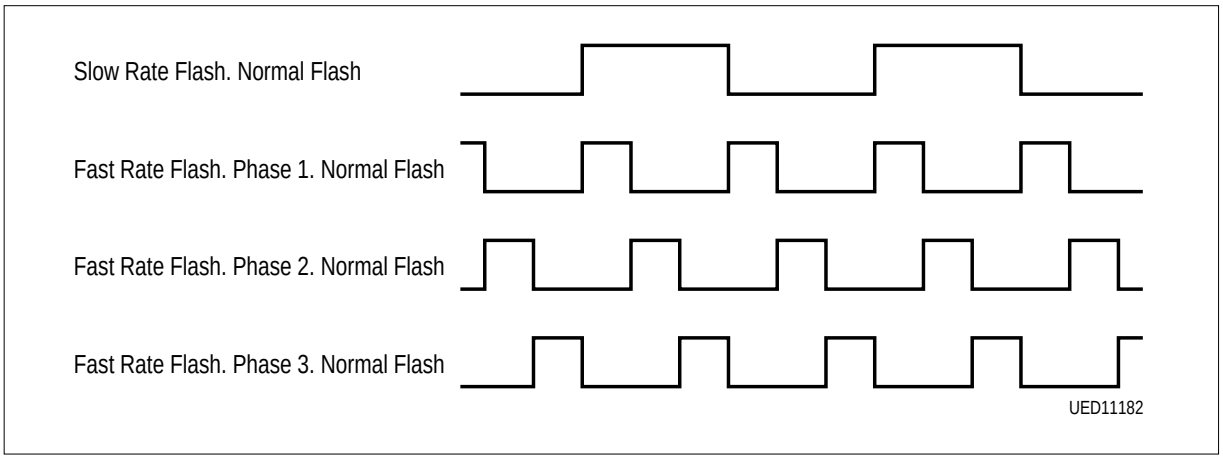


Figure 10-13 Internally Generated Flash Signals in Different Flash Phases

5-bit colour look up vectors are converted into a 12-bit RGB value and a 2-bit transparency level by CLUT2 during display. To do this, the input side of CLUT2 the 5-bit look up value is used for Bit0 to Bit4, the C^2-values are used as Bit7 to Bit5. The 12-bit RGB value is fed to the D/A converter and the 2-bit transparency information to the BLANK/COR pins.

16-bit format pixels which are stored in the 12-bit RGB format are not passing the CLUT2. The 12-bit RGB value and the 2-bit transparency information is directly fed into the D/A converter and the BLANK/COR pins.

Frame Buffer in 16-bit Pixel Format (5:6:5)

In this mode the frame buffer contains RGB values with 5 bits for red colour components, 6 bits for green colour components and 5 bits for blue colour components:

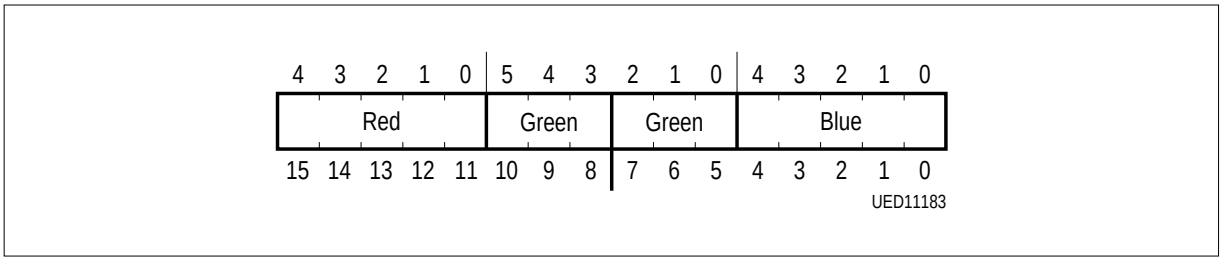


Figure 10-14 16-bit Pixel Format (5:6:5) for Use in Frame Buffer

Transparency between layers is not supported if this mode is in use. The 5:6:5 format is directly transferred to the D/A converter. CLUT2 is out of use in this mode.

10.5 Initialization of Memory Transfers

Transferring an input format of the source area to an output format in the destination area is supported by different transfer modes. For this, a pixel modification unit and CLUT1 of the GA is used. CLUT1 has a size of 256 × 16 bit.

Next to the transfer mode the transfer areas (source and destination) must be defined by GA instructions. Source is either an address in the external memory or a constant value coming from CLUT1. Destination is always an address in the external memory. Constant values coming from CLUT1 are used to draw lines or fill parallelograms.

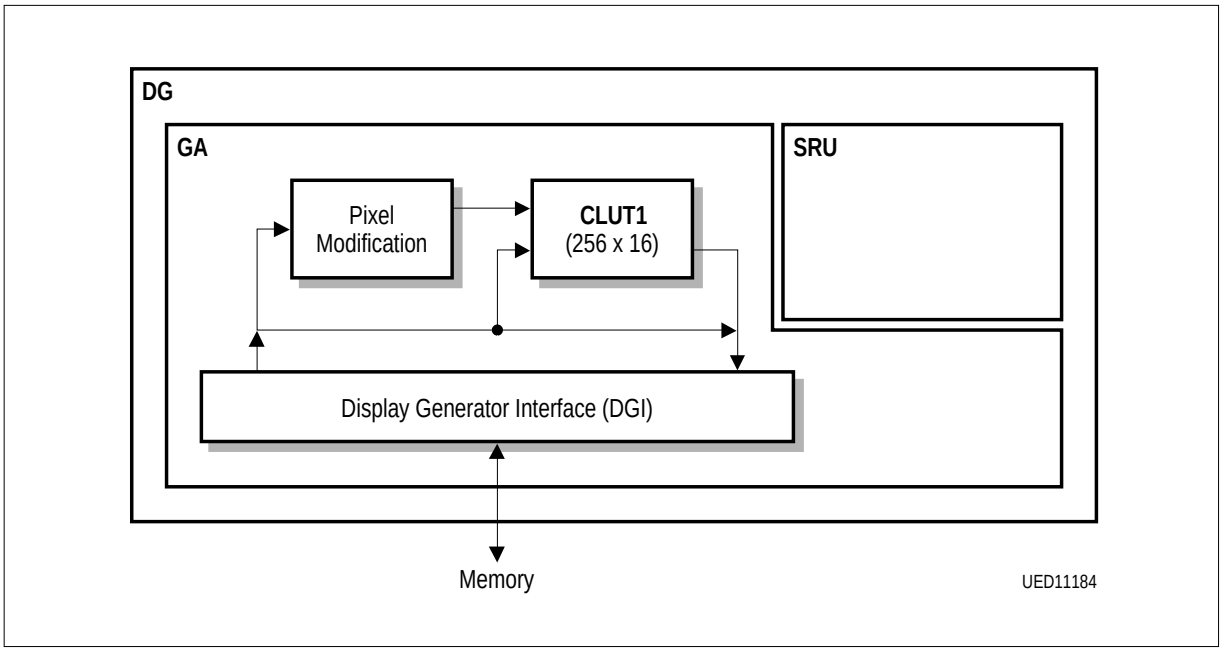


Figure 10-15 Overview of GA

10.5.1 Transfer Modes

Different transfer modes are available. The table below shows all possible combinations of input and output formats.

‘Transparency available’ means that if transparency mode is chosen (see also GAI-instruction ‘TAR’ (Bit: TRM)) and Bit TR1 of the output word is set to ‘0’ (refer also to **Figure 10-12**) this pixel is written to the destination, otherwise this pixel is not written to the destination.

‘Modification’ describes in which way the output information is generated by using the input information, CLUT1 and the settings of the ‘TAR’-instruction. Using 16-bit TTX output two bits are replaced by another 2 bits (Flash) which are also defined by graphic accelerator instruction ‘TAR’. The table below shows all possible combinations of input and output formats:

Table 10-5 Supported Transfer Modes

TMOD (4 … 0)	Input Format	Output Format	Transparency Available	Modification
00000	1-bit bitmap	16-bit (TTX)	No	OUT(15) = ‘0’ OUT(14 … 13) = FLA(1 … 0) OUT(12 … 0) = CLUT1(‘0000000’ & IN(0))(12 … 0)
00001	1-bit bitmap	16-bit (4:4:4:2)	Yes	OUT(15) = ‘1’ OUT(14 … 13) = CLUT1(‘0000000’ & IN(0))(14 … 13)) OUT(12) = IT OUT(11 … 0) = CLUT1(‘0000000’ & IN(0))(11 … 0))
00010	1-bit bitmap	8-bit	No	OUT(7 … 0) = CLUT1(‘0000000’ & IN(0))(7 … 0)
00011	1-bit bitmap	16-bit (5:6:5)	No	OUT(15 … 0) = CLUT1(‘0000000’ & IN(1 … 0))(15 … 0))
00100	2-bit bitmap	16-bit (TTX)	No	OUT(15) = ‘0’ OUT(14 … 13) = FLA(1 … 0) OUT(12 … 0) = CLUT1(‘000000’ & IN(1 … 0))(12 … 0)
00101	2-bit bitmap	16-bit (4:4:4:2)	Yes	OUT(15) = ‘1’ OUT(14 … 13) = CLUT1(‘0000000’ & IN(1 … 0))(14 … 13)) OUT(12) = IT OUT(11 … 0) = CLUT1(‘0000000’ & IN(1 … 0))(11 … 0))
00110	2-bit bitmap	8-bit	No	OUT(7 … 0) = CLUT1(‘000000’ & IN(1 … 0))(7 … 0)
00111	2-bit bitmap	16-bit (5:6:5)	No	OUT(15 … 0) = CLUT1(‘000000’ & IN(1 … 0))(15 … 0))
01000	4-bit bitmap	16-bit (TTX)	No	OUT(15) = ‘0’ OUT(14 … 13) = FLA(1 … 0) OUT(12 … 0) = CLUT1(‘0000’ & IN(3 … 0))(12 … 0)

Table 10-5 Supported Transfer Modes (cont'd)

TMOD (4 … 0)	Input Format	Output Format	Transparency Available	Modification
01001	4-bit bitmap	16-bit (4:4:4:2)	Yes	OUT(15) = '1' OUT(14 … 13) = CLUT1('0000000' & IN(3 … 0))(14 … 13)) OUT(12) = IT OUT(11 … 0) = CLUT1('0000000' & IN(3 … 0))(11 … 0))
01010	4-bit bitmap	8-bit	No	OUT(7 … 0) = CLUT1('0000' & IN(3 … 0))(7 … 0)
01011	4-bit bitmap	16-bit (5:6:5)	No	OUT(15 … 0) = CLUT1('0000' & IN(3 … 0))(15 … 0))
01100	8-bit bitmap	16-bit (TTX)	No	OUT(15) = '0' OUT(14 … 13) = FLA(1 … 0) OUT(12 … 0) = CLUT1(IN(7 … 0))(12 … 0)
01101	8-bit bitmap	16-bit (4:4:4:2)	Yes	OUT(15) = '1' OUT(14 … 13) = CLUT1('0000000' & IN(7 … 0))(14 … 13)) OUT(12) = IT OUT(11 … 0) = CLUT1('0000000' & IN(7 … 0))(11 … 0))
01110	8-bit bitmap	8-bit	No	OUT(7 … 0) = CLUT1(IN(7 … 0))(7 … 0)
01111	8-bit bitmap	16-bit (5:6:5)	No	OUT(15 … 0) = CLUT1(IN(7 … 0))(15 … 0))
10000	CLUT1 input	8-bit	No	OUT(7 … 0) = CLUT1(0)(7 … 0)
10001	CLUT1 input	16-bit (TTX)	No	OUT(15) = '0' OUT(14 … 13) = FLA(1 … 0) OUT(12 … 0) = CLUT1(0)(12 … 0)
10010	CLUT1 input	16-bit (4:4:4:2)	Yes	OUT(15) = '1' OUT(14 … 0) = CLUT1(0)(14 … 0))
10011	16-bit (TTX or 4:4:4:2)	16-bit (TTX or 4:4:4:2)	Only In 4:4:4:2 Mode	OUT(15 … 0) = IN(15 … 0)

Table 10-5 Supported Transfer Modes (cont'd)

TMOD (4 … 0)	Input Format	Output Format	Transparency Available	Modification
10100	8-bit data	8-bit data	No	OUT(7 … 0) = IN(7 … 0)
Others	Reserved			

Note: There is no transfer mode defined which uses the 2-bit format as an output format, because in this case layer 2 is restricted to a width of 64 pixels. If the 2-bit format is required, the direct byte by byte transfer can be used.

10.5.2 Transfer Areas

Next to the transfer mode the transfer area has to be defined by graphic accelerator instructions (GAI). For source and destination, 'linear' and 'rectangle' areas can be defined.

'**Linear**' means that data is accessed byte by byte without any irregularity in the addressing of the memory.

'**Rectangle**' means that after the access of n bytes defined by a parameter 'width' an 'offset' to the last address is automatically added. This feature can be used if an array of data has to be copied from one memory location into another (bigger) array at any other memory location. The linear addressing is a subset of the rectangle addressing scheme if 'offset' is set to '0'.

Source Area

There are different settings necessary to define the source area. These settings are done by using the graphic accelerator instruction set. The table below describes the necessary settings and corresponding GAIs with the affected bit position inside the GAI:

Used GAI	Bit Position Inside GAI	Description
SDR	S_ADDR	Start address of source area
TSR	WIDTH_IN	Width of source area
TOR	S_OFFSET	Offset to describe rectangular source areas

As described before, there are seven different formats on the input side of the transfer:

- 1-bit bitmap
- 2-bit bitmap
- 4-bit bitmap
- 8-bit bitmap
- 8-bit data (used for direct data transfer)
- CLUT1 input (used for drawing of filled parallelograms, rectangles, lines)
- 16-bit (4:4:4:2) RGB

CLUT1 input does not need any more detailed area description. The input value comes directly from the address ‘0’ of CLUT1 and not from the RAM. This mode can be used for drawing lines, filling rectangles or parallelograms. For the other input modes a more detailed description is given below:

From the point of view of the register settings, which are used to define the source area, the different input formats can be divided in three groups which are handled in different ways.

Group 1:	Group 2:	Group3:
1-bit bitmap	8-bit data	16-bit pixel format (4:4:4:2)
2-bit bitmap	8-bit bitmap	
4-bit bitmap		

Formats of group 1 are formats which define each pixel with less than a byte. Group 2 formats are formats which define each pixel by 8 bits, and group 3 formats are formats which define each pixel by 16 bits.

Group 1:
In 1-bit bitmap, 2-bit bitmap and 4-bit bitmap input mode it is expected, that the bitmaps are stored linearly in the memory as described in **Chapter 10.4.1**. Therefore the settings of WIDTH_IN as well as S_OFFSET are ignored. Only the 24-bit source address pointer S_ADDR is used. The amount of pixels which are read from the source and written to the destination is only defined by the destination settings. The user has to take care that the destination settings fit with the bitmap inherent alignments.

Group 2:
In this mode WIDTH_IN and S_OFFSET are also taken into account. The amount of memory which is described by WIDTH_IN and S_OFFSET is described by numbers of bytes.

Group 3:
In this mode WIDTH_IN and S_OFFSET are also taken into account. The amount of memory which is described by WIDTH_IN and S_OFFSET is described by numbers of words.

Note: The number of bytes to be read from the source area is defined by the destination area (see below) and the transfer mode. This is why no explicit definition of height is needed for the source.

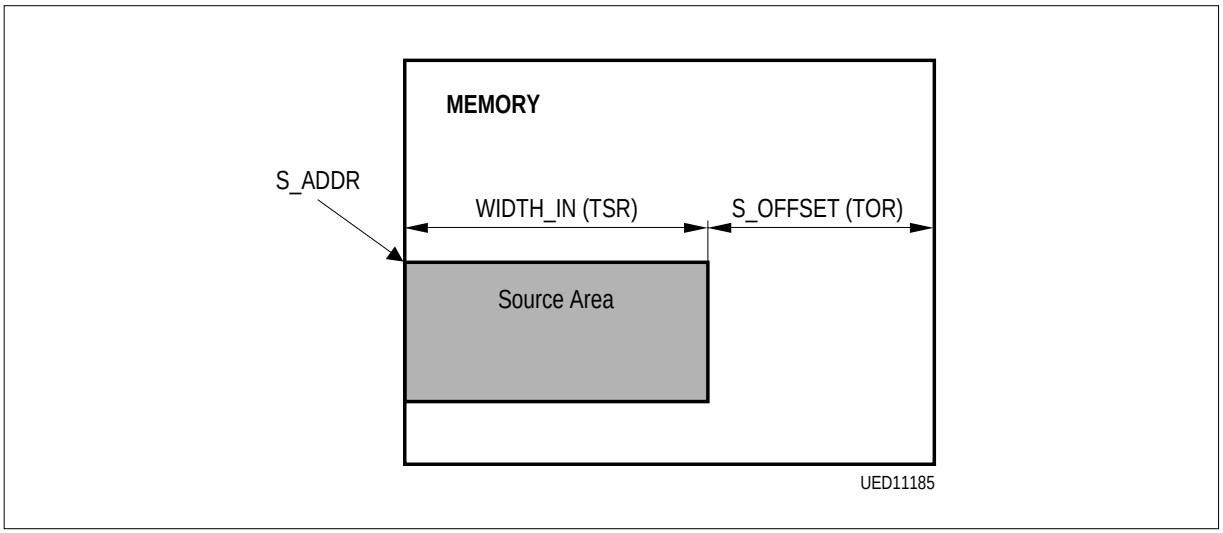


Figure 10-16 Use of Register Settings to Specify Source Area

Destination Area

There are additional settings necessary to define the destination area. The table below describes the settings and the corresponding GAIs with the affected bit position inside the GAI:

Used GAI	Bit Position Inside GAI	Description
DDR	D_ADDR	Start address of destination area
TDR	HEIGHT_OUT, WIDTH_OUT	Height and width of destination area
TOR	D_OFFSET	Offset to describe rectangular destination areas

Next to the destination area itself a clipping area can be defined. The clipping area needs to be defined within the destination area. During a memory transfer these ‘clipped’ memory areas are excluded from the transfer. The table below describes the settings and the corresponding GAIs with the affected bit positions inside the GAI:

Used GAI	Bit Position Inside GAI	Description
CUR	C_ADDR	Start address of clipping area
CBR	HEIGHT_CLIP, WIDTH_CLIP	Height and width of clipping area
CBR, CUR	C_OFFSET	Offset to describe rectangular clipping areas

As described before, there are five different formats on the output side of the transfer: (Also please refer to **Chapter 10.4.2**)

- 8-bit format CLUT2 vector
- 16-bit format (4:4:4:2) RGB
- 16-bit format (TTX) CLUT2 vector
- 16-bit format (5:6:5) RGB
- Direct data transfer (byte by byte)

From the point of view of the register settings which are used to define the alignment of the destination area, these formats can be divided in two groups. Each group is handled in a different way.

Group 1:	Group 2:
16-bit format (4:4:4:2) RGB	8-bit format CLUT2 vector
16-bit format (TTX) CLUT2 vector	8-bit data (byte by byte transfer)
16-bit format (5:6:5) RGB	

Formats of group 1 are formats which define each pixel by 16 bits. Group 2 formats are formats which define each pixel by 8 bits.

*Note: Bit fields HEIGHT_CLIP, HEIGHT_OUT and WIDTH_CLIP, WIDTH_OUT describe the height and width of the destination in count of pixels and not in bytes. So for output formats of group 1 the memory area which is described by a HEIGHT_OUT value and a WIDTH_OUT value needs the double amount of memory, than output formats which are described by the same HEIGHT_OUT value and WIDTH_OUT value for output formats of group 2. Also, the OFFSET values are pixels and **not** bytes or words.*
C_ADDR and D_ADDR are real byte addresses in memory.

Note: For a rectangle destination area the sum of WIDTH and D_OFFSET of the destination area must be equal to the width of the frame buffer (WIDTH_L1(2)). Otherwise the shape of the copied frame will be a parallelogram and not a rectangle.

Note: For a rectangle clipping area inside the destination area the sum of the clipping offset and the clipping width must be the same as the sum of the destination area width and the destination offset. Otherwise the clipping area will be a parallelogram and not a rectangle.

Note: Source area and destination area should not overlap. Otherwise it may appear that a pixel is overwritten as a destination pixel, and afterwards used as a source pixel.

Meaning of Double Width and Double Height during Transfer

The destination area can be stretched horizontally and vertically by using GA instruction TAR.

If double width (TDW) is set to ‘1’, the graphic accelerator writes each pixel twice horizontally to the destination area.

TQH	TDH	Meaning
0	**0**	Normal transfer
0	**1**	If double height (TDH) is set to ‘1’ and quadruple height (TQH) is set to ‘0’ in the destination output side each pixel in vertical direction is repeated twice.
1	**X**	If quadruple height (TQH) is set to ‘1’ on the destination output side each pixel in vertical direction is repeated four times.

If double height (TDH) is set to ‘1’ in the destination output side each pixel in vertical direction is repeated twice.

For example: If double width is set to ‘1’ and quadruple height is set to ‘1’ each pixel of the source area needs 8 pixels of the destination area.

Note: Parameters like WIDTH_OUT, WIDTH_CLIP, HEIGHT_OUT and HEIGHT_CLIP are still pixel related. WIDTH_OUT, HEIGHT_OUT and D_OFFSET have to be adapted by the software to get a complete character. Clipping is not affected by TDH and TDW.

The following table is an example of a 1-bit bitmap (30×50) which should be transferred either in normal size or in double size to an 8 bit output format in a frame buffer with a size of 100×200 pixels.

Parameters	TDW = 0; TDH = 0; TQH = 0	TDW = 1; TDH = 1; TQH = 0
WIDTH_L1	100	
HEIGHT_L1	200	
WIDTH_IN	30	
WIDTH_OUT	30	60
D_OFFSET	70	40
HEIGHT_OUT	50	100

Note: In case of double width transfer WIDTH_OUT has to be an even number. In case of double/quad height transfer HEIGHT_OUT has to be an even/divisable by 4 number.

Figure 10-17 gives a graphical overview of how to specify the different areas.

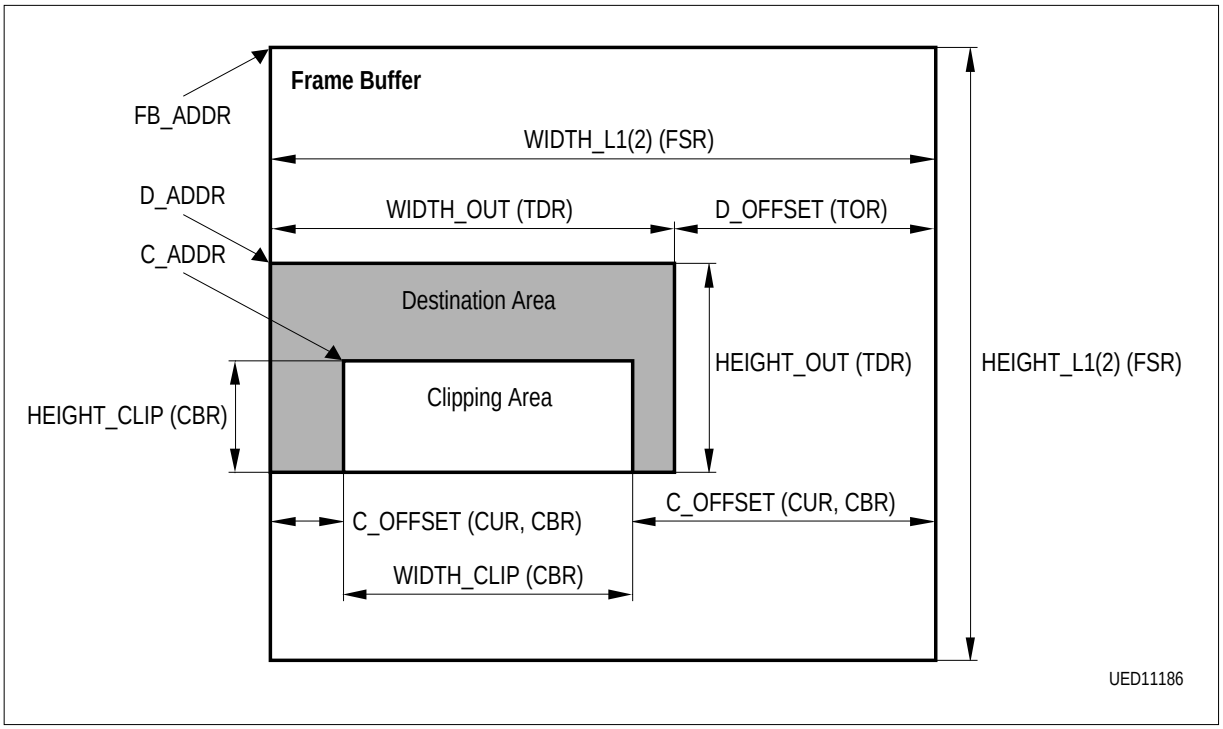


Figure 10-17 Use of Register Settings to Specify Destination and Clipping Area

10.5.3 Italic Mode

For transfer modes with 16-bit (4:4:4:2) format as an output, an automatic italic transfer option is available. If italic mode is chosen only the destination area (not the source area) is affected. The following two figures explain the different representation of pixels in the frame buffer in non-italic and italic mode.

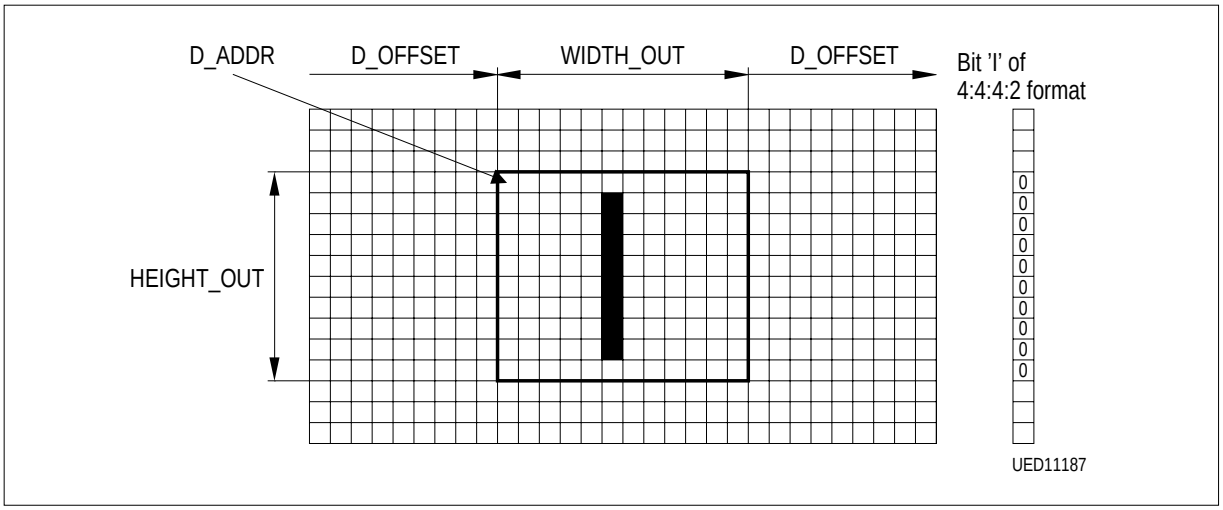


Figure 10-18 Result for a Non-italic Transferred Memory Area in Frame Buffer

If this transfer is executed with the same register settings but in italic mode instead of non italic mode, the subsequent destination area is used inside the frame buffer:

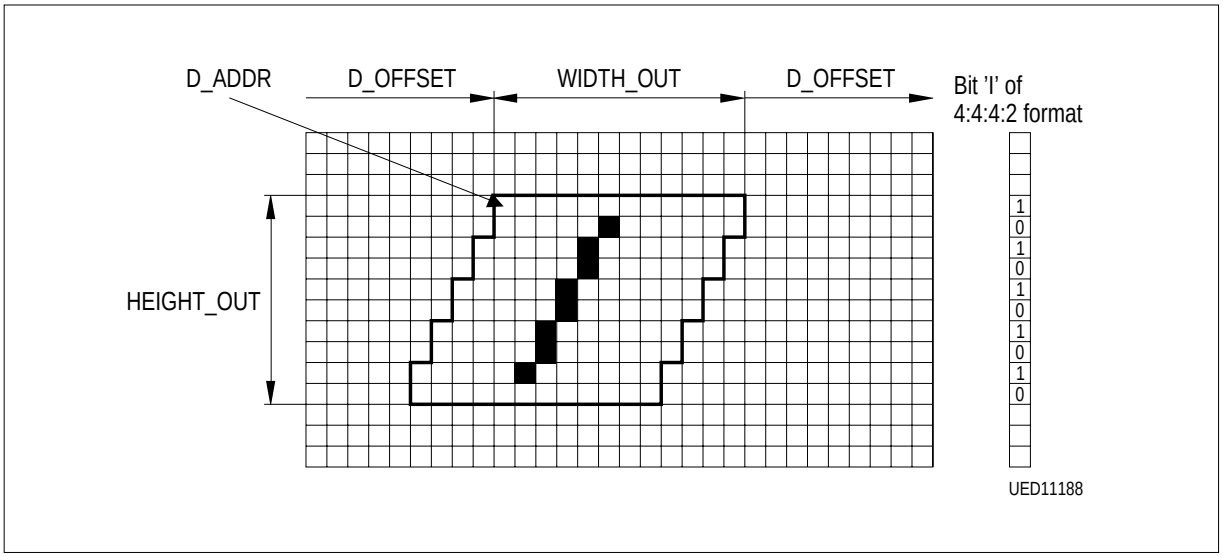


Figure 10-19 Result for a Italic Transferred Memory Area in Frame Buffer

Next to the destination pixel offset from line to line, the italic bit (‘I’) alternates from line to line. This italic bit is used to control the D/A converter to realize a horizontal line alternating half pixel shift on RGB output.

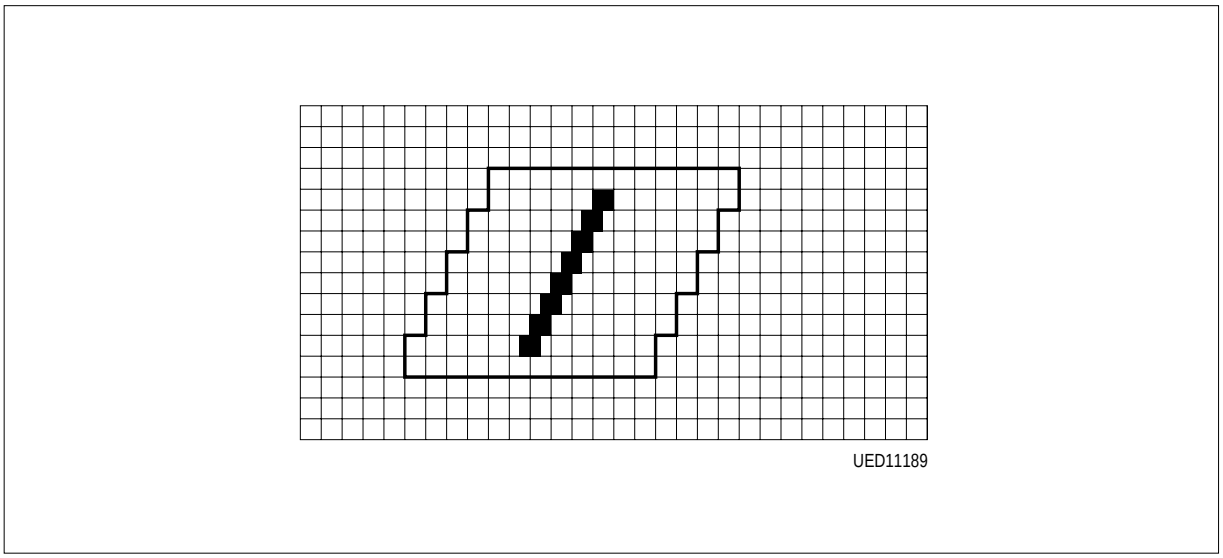


Figure 10-20 Result for an Italic Transferred Memory Area at D/A Converter Output

Note: Italic can not be used together with double width and double/quad height transfers.

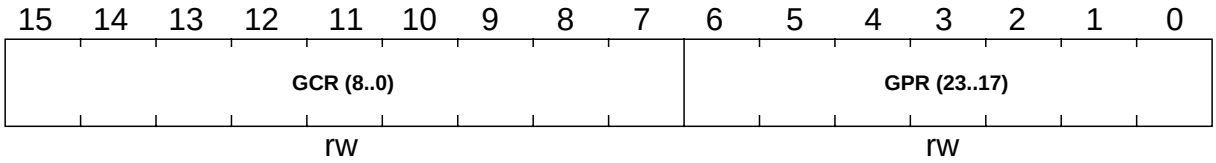
10.6 Register Description

10.6.1 Special Function Registers

The display generator is controlled by 3 special function registers and a list of accelerator instructions. Two of the registers define the position and the length of the instruction list, and one register is used for general control of the DG. After the list of instructions (GAIs) is written to the memory, the start address and the length of this list must be written to special function registers GPRGCRH and GPRGCRL. Then the controller commands the GA to execute these instructions (see SFR DGCON). After the GA has finished executing all GAIs, the GA gives the controller an interrupt (GAFIR).

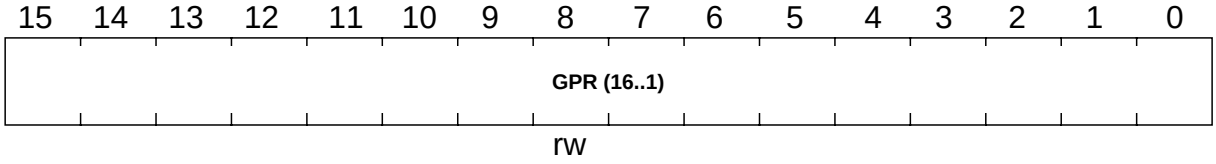
Note: GPRGCRH, GPRGCRL and DGCON are the only special function registers to control the display generator. GAIs described in the next paragraph are stored in the RAM and not defined after power on.

GPRGCRH **Reset Value: 0000_H**



Bit	Function
GCR (8 … 0)	**Defines the amount of GAIs in the instruction list.** GCR × 4 is the length of the GAI area in count of bytes. *Note: After the last instruction is executed an interrupt is given to the controller*
GPR (23 … 17)	**Define the MSBs of the 23-bit address pointer to the start of the GAI area.**

GPRGCRL **Reset Value: 0000_H**



Bit	Function
GPR (16 … 1)	**Define the LSBs of the 23-bit address pointer to the start of the GAI area.**

DGCON is for general control of the DG.

DGCON **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	BSY GA	ST GA	EO DG	EA DG
												r	rw	rw	rw

Bit	Function
EADG	**Enables access from DG to SDRAM.** 0: All requests from DG (SRU and GA) to the memory are disabled. 1: The DG has normal access to the memory. *Note: The running memory access is finalized before this bit becomes active.*
EODG	**Enables output of DG.** 0: All outputs of the DG are disabled (RGB outputs are switched to black level, COR = 0 and BLANK = 1) (‘Normal’ pin polarity assumed. Please refer also to register SCR). 1: The outputs have the function according to the specifications described in the following paragraphs. *Note: The SRU registers FBR and DBR have to be programmed with a valid memory address before enabling display output.*
STGA	**Starts processing of the instruction list by the GA** 0: STGA has to be reset by SW before the GA can be started again. 1: The GA starts the execution of the instruction list at address GPR. It ends after the specified number of instructions (see bit field GCR) is executed. After that an interrupt (GAFIR) is given to the controller.
GABSY	0: GA is idle, waiting for GAI sequence. 1: GA is busy, GAI sequence is processed.

The register PXDEL controls an individual delay of 0 … 2 clock cycles of the SRU outputs. Each output (R, G, B, Italic, blank, cor) is contolled by 2 bits in the PXDEL register.

PXDEL **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res	res	res	res	Tcor		Tblank		Titalic		Tblu		Tgreen		Tred	

Bits	
1 … 0	delay for red
3 … 2	delay for green
5 … 4	delay for blue
7 … 6	delay for italic
9 … 8	delay for blank
11 … 10	delay for cor
15 … 12	reserved

Bits set to “00” - no delay
“01” - delay of 1 pixel clock cycle
“10” - delay of 2 pixel clock cycles
“11” - reserved

The reset value of PXDEL is set to 0000_H, which means no dely of the SRU outputs after reset.

10.7 Description of Graphic Accelerator Instructions

GAIs are 32-bit instructions which are used as an interface from µC to DG. They are written sequentially to the SDRAM by the controller in form of an instruction list. **Figure 10-21** shows the organization of GAIs in the memory.

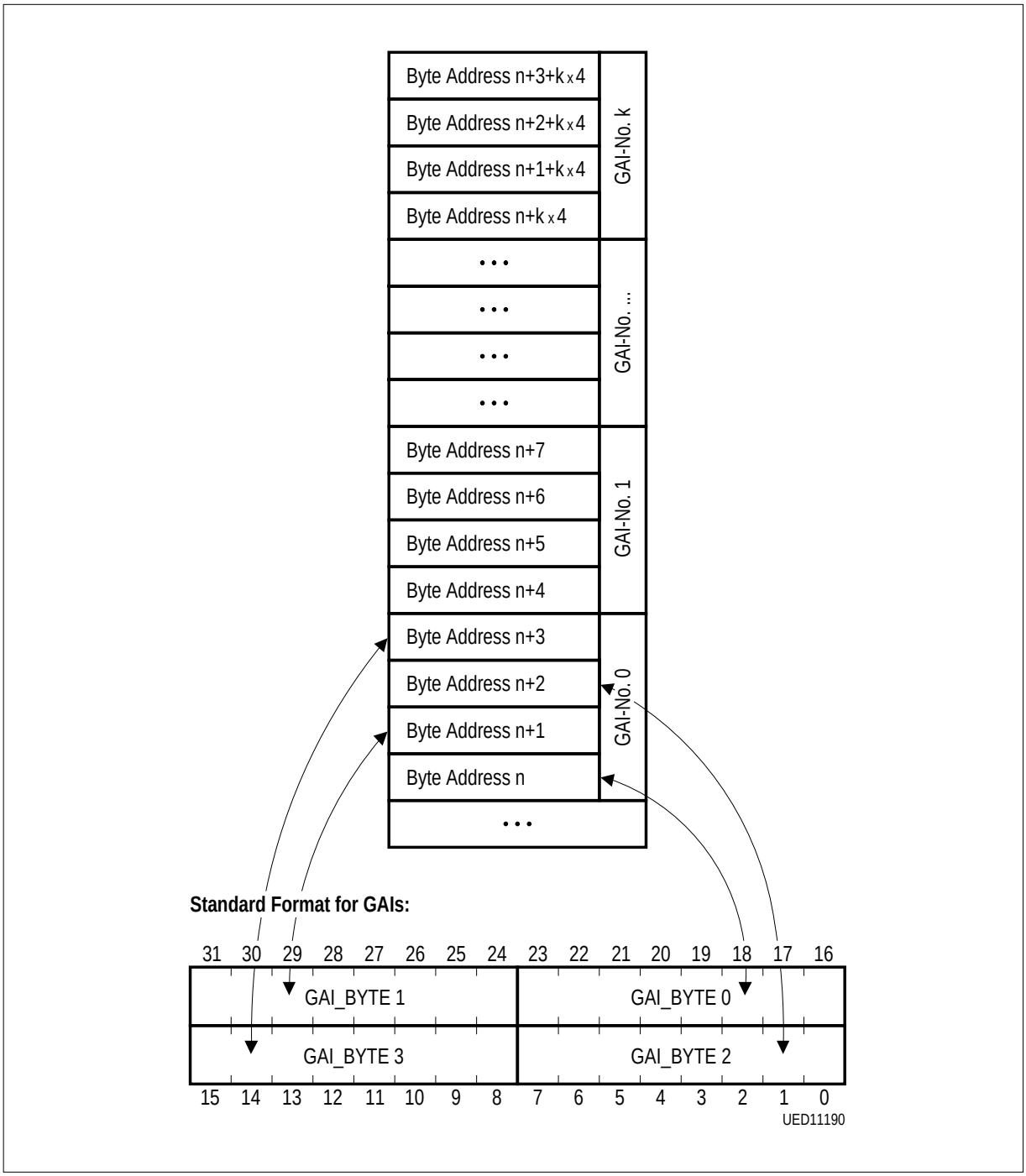


Figure 10-21 Organization of GAIs in the External SDRAM

In the following GAI description, ‘–’ means that these bits are reserved for future use and have to be set to ‘0’. The meaning of an instruction is not given by the physical location (address) of the instruction but by its opcode which is represented by bits 31 … 28. Bits 27 … 0 are equivalent to an operand.

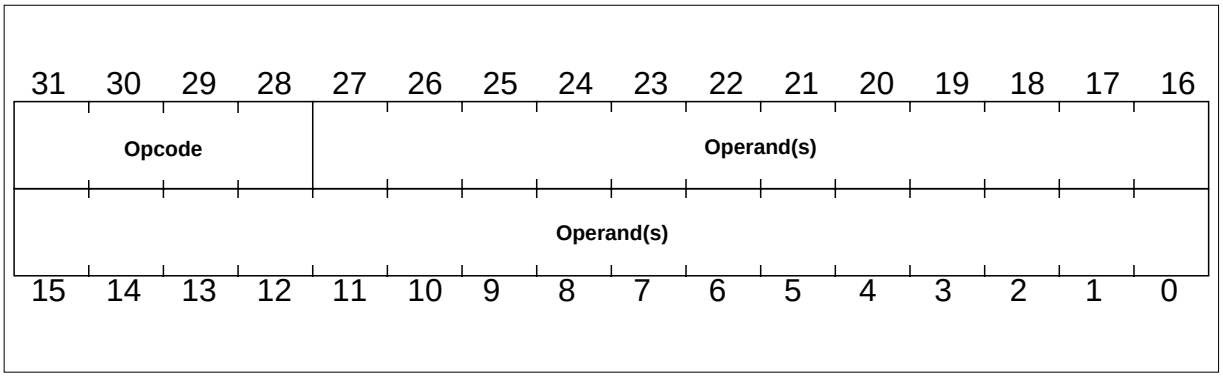


Figure 10-22 GAI Instruction Format

There are two types of GAIs. One type is used to setup **global parameters** for SRU and GA, like SRU setup, frame buffer setup or CLUT setup. The second type of GAI is used to setup the **transfer parameters** for DMA functions.

Global Parameters

The following global instructions are only executed by the GA during the vertical sync area. If such a GA-instruction is read by the GA outside the V-sync area, it waits until the next V-sync appears.

Note: The V-sync area is defined for that purpose as the first 4 lines of a field.

- **SRU setup**
  SAR (opcode = 1111) - set screen attributes

- **Frame buffer 1 (layer 1) and frame buffer 2 (layer 2) setup**
  FBR (opcode = 1000) - set address of the beginning of frame buffer 1
  FSR (opcode = 1001) - set size of frame buffer 1
  DBR (opcode = 1010) - set address of the beginning of frame buffer 2
  DSR (opcode = 1011) - set size of frame buffer 2
  DCR (opcode = 1100) - set frame buffer 2 reference coordinates

- **CLUT Setup**
  CLR (opcode = 1101) - set contents of the CLUT1 or CLUT2

Note: If CLUT2 should be loaded the GA waits until the next V-sync appears.

- **No operation**
  NOP (opcode = 1110) - no operation

Transfer Parameters

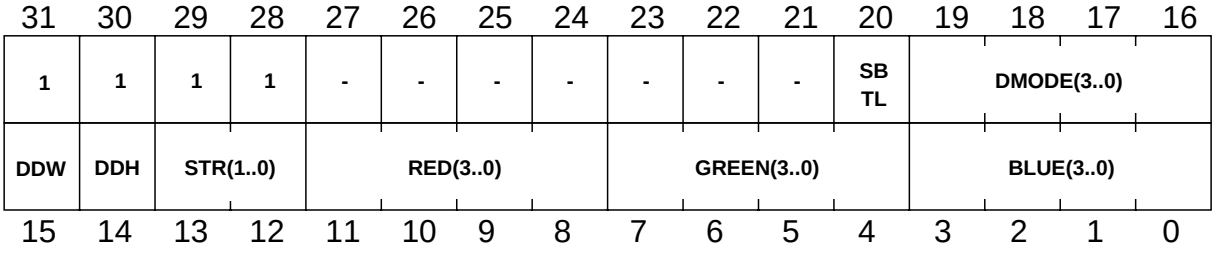
These instructions are immediately executed by the GA.

- **CUR** (opcode = 0000) - set clipping coordinates
- **CBR** (opcode = 0001) - set clipping coordinates
- **SDR** (opcode = 0010) - set source descriptor for data transfer
- **DDR** (opcode = 0011) - set destination descriptor for data transfer
- **TSR** (opcode = 0100) - set definitions for source memory area
- **TDR** (opcode = 0101) - set definitions for destination memory area
- **TOR** (opcode = 0110) - set offset definitions for transferred area
- **TAR** (opcode = 0111) - set attributes for transfer

10.7.1 Screen Attributes (SAR)

This instruction controls different screen attributes.

SAR



Bit	Function
SBTL	**Screen Background Transparency under Layer Area** Defines whether the screen background area is transparent under a layer 1 (2) area or not. 0: The screen background within a layer area is not transparent. Under transparent layer pixels the background colour can be seen. 1: The screen background within a layer area is transparent. Under transparent layer pixels the video can be seen. *Note: Transparency definitions for the screen background outside a layer area are made with bits STR. Also please refer to **Chapter 10.3.3**.*
DMODE (3 … 0)	**Display Mode Bits** Defines the possible combinations of both layers with the available pixel formats. Please see table ‘Display Modes’ below.

Bit	Function
DDW	**Double Width Display** 0: Normal width 1: Double width. The contents of the screen are stretched in horizontal direction. The SRU repeats the same pixel information twice in horizontal direction. *Note: DDW = '1' the frame buffer width (WIDTH_L1(L2) has to be divided by two to get the same area displayed on the screen.*
DDH	**Double Height Display** 0: Normal height 1: Double height. The contents of the screen are stretched in vertical direction. The SRU repeats the same pixel information twice in vertical direction. *Note: DDH = '1' the frame buffer height (HEIGHT_L1(L2) has to be divided by two to get the same area displayed on the screen.*
STR (1 … 0)	**Screen Background Transparency Level** Define the transparency of the screen background area outside the layer area. Please refer to **Chapter 10.3.3**.
RED (3 … 0)	**Screen Background Red Colour** 4-bit red component
GREEN (3 … 0)	**Screen Background Green Colour** 4-bit green component
BLUE (3 … 0)	**Screen Background Blue Colour** 4-bit blue component

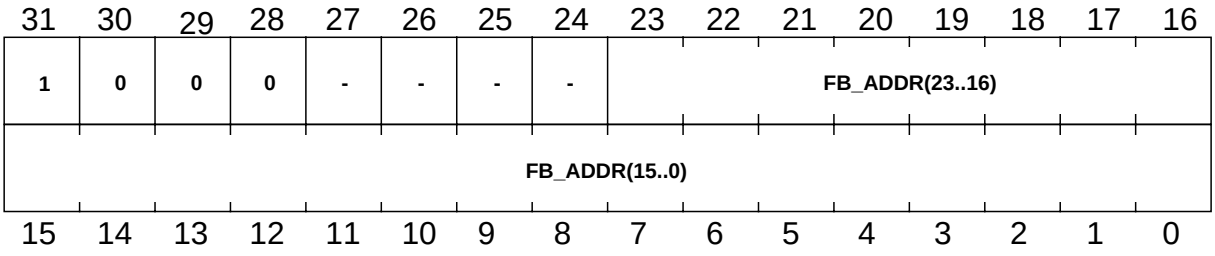
Table 10-6 Display Modes

DMODE (3 … 0)	Layer Formats		Layer Mode
	Layer 1	Layer 2	
0000	16-bit (4:4:4:2 or TTX)	–	Layer 2 switched off
0001	8-bit	–	Layer 2 switched off
0010	16-bit (5:6:5)	–	Layer 2 switched off
0011	–	–	Layer 1 & 2 switched off
0100	reserved	–	–
0101	reserved		
0110	16-bit (4:4:4:2 or TTX)	2-bit	overlapped
0111	reserved		
1000	16-bit (4:4:4:2 or TTX)	16-bit (4:4:4:2 or TTX)	embedded
1001	8-bit	8-bit	embedded
1010	16-bit (4:4:4:2 or TTX)	8-bit	embedded
1011	8-bit	16-bit (4:4:4:2 or TTX)	embedded
1100	16-bit (5:6:5)	16-bit (5:6:5)	embedded
1101	reserved		
1110			
1111			

10.7.2 Startaddress of Layer 1 (FBR)

The start address of frame buffer 1 in the memory must be set by the controller.

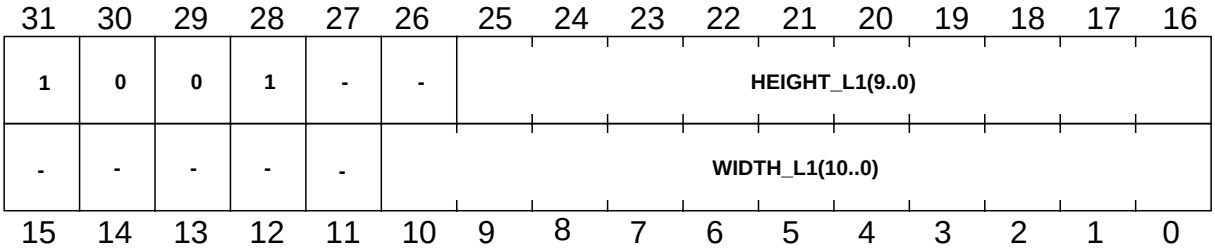
FBR



Bit	Function
FB_ADDR (23 … 0)	**Startaddress of frame buffer 1** Bit 23 … 0 of a byte address.

10.7.3 Size of Layer 1 (FSR)

FSR

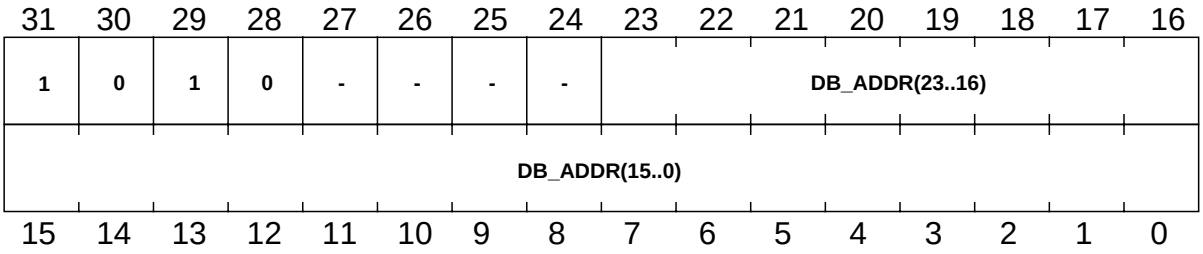


Bit	Function
HEIGHT_L1 (9 … 0)	**Height of Frame Buffer 1** The height of the frame buffer can vary between 0 (HEIGHT_L1 = ‘0’$_d$) and 1023 pixels (HEIGHT_L1 = ‘1023’$_d$).
WIDTH_L1 (10 … 0)	**Width of Frame Buffer 1** The width of the frame buffer can vary between 0 (WIDTH_L1 = ‘0’$_d$) and 2046 pixels (WIDTH_L1 = ‘2046’$_d$).

10.7.4 Startaddress of Layer 2 (DBR)

The start address of frame buffer 2 in the memory must be set by the controller.

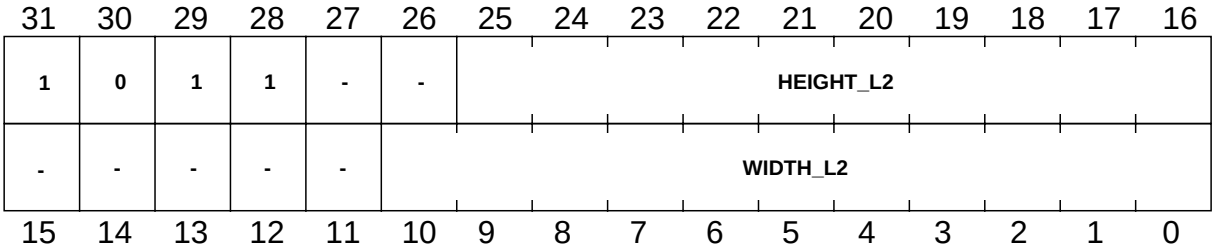
DBR



Bit	Function
DB_ADDR (23 … 0)	**Startaddress of Frame Buffer 2** Bit 23 … 0 of a byte address.

10.7.5 Size of Layer 2 (DSR)

DSR

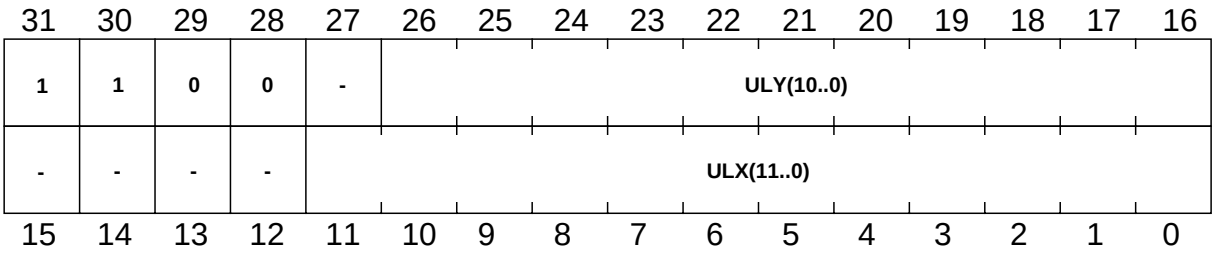


Bit	Function
HEIGHT_L2 (9 … 0)	**Height of Layer 2** The height of layer 2 can vary between 0 (HEIGHT_L2 = ‘0’$_d$) and 1023 pixels (HEIGHT_L2 = ‘1023’$_d$).
WIDTH_L2 (10 … 0)	**Width of Layer 2** The width of the layer 2 can vary between 0 (WIDTH_L2 = ‘0’$_d$) and 2046 pixels (WIDTH_L2 = ‘2046’$_d$). *Note: Width_L2 is ignored, if layer 2 is displayed in 2-bit CLUT2 vector format. In this case width_L2 is set to 64.*

10.7.6 Display Coordinates of Layer 2 (DCR)

This instruction is used to place layer 2 in layer 1. By these coordinates the left top corner of layer 2 is placed in relation to the top left corner of layer 1. In this sense the coordinate ULY = 0/ULX = 0 is identical to the top left corner of layer 1. Negative coordinates are also supported so it is also possible to move a layer 2 window from the top or left side of a layer 1 window.

DCR

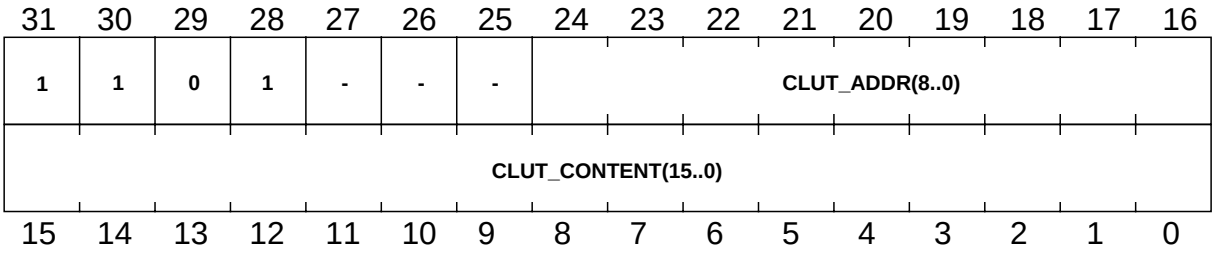


Bit	Function
ULY (10 … 0)	**Upper left corner Y-coordinate** in one's complement representation: '11111111111' = '– 1023_D' … '10000000000' = '– 0_D' '00000000000' = '+ 0_D' … '01111111111' = '+ 1023_D'
ULX (11 … 0)	**Upper left corner X-coordinate** in one's complement representation: '111111111111' = '– 2046_D' … '100000000000' = '– 0_D' '000000000000' = '+ 0_D' … '011111111111' = '+ 2046_D'

10.7.7 Contents of CLUT (CLR)

CLR instruction allows the contents of the CLUT1 and CLUT2 to be set.

CLR



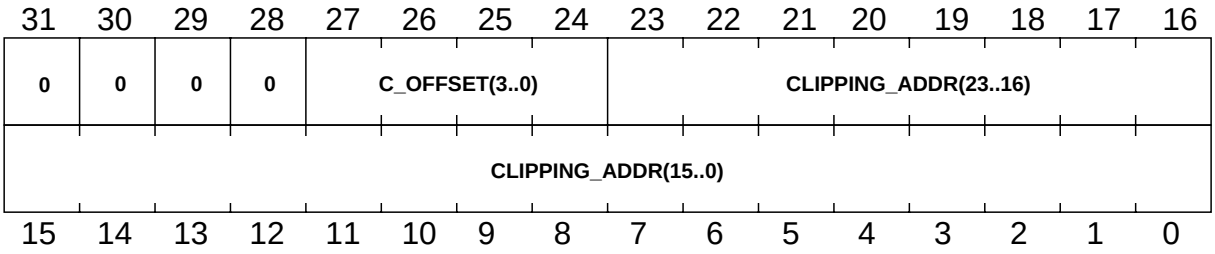
Bit	Function
CLUT_ ADDR (8 … 0)	**CLUT address** Bits 7 … 0 are CLUT vectors of either CLUT1 or CLUT2. Bit 8 selects the CLUT: 0: CLUT1 (256 × 16 bit) 1: CLUT2 (256 × 14 bit)
CLUT_ CONTENT (15 … 0)	**Contents to be written** to one of the look up tables addressed by the CLUT vector defined above. For CLUT1 all the 16 bits are used, for CLUT2 only bits 13 … 0 are used.

10.7.8 Clipping Coordinates (CUR and CBR)

These GAIs are used to set clipping coordinates for memory transfers. The clipping coordinates describe a rectangle area in the destination memory area. During a memory transfer these ‘clipped’ memory areas are excluded from the transfer. The CUR instruction is used to set the clipping start address. The CBR instruction is used to set the width and height of the clipping area in amounts of pixels. A width of ‘0’ means that no clipping is processed, a width of ‘1’ means the clipping area has the width of 1 pixel, …. A height of ‘0’ means, that no clipping is processed, a height of ‘1’ means the clipping area has the height of 1 line, … . Next to the start address, height and width a clipping offset must be given for clipping. For a rectangle clipping area the sum of this clipping offset and the clipping width must be the same as the sum of the destination area width and the destination offset. Otherwise the clipping area will be a parallelogram, not a rectangle.

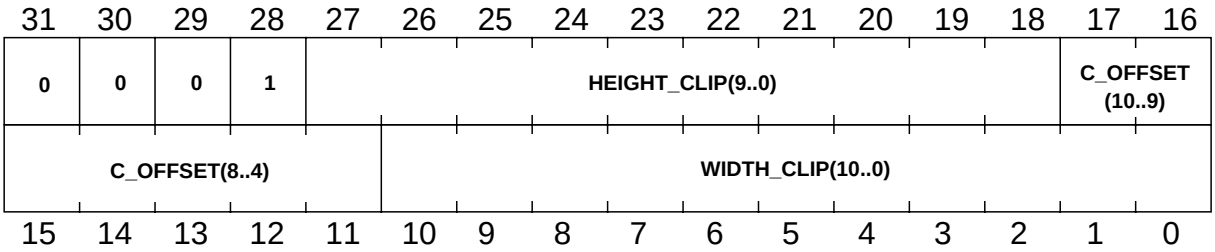
Within instruction TAR it is decided whether the clipping area is inverted or non inverted. Thus the outside region of the defined rectangle can be used for clipping as well as the inside region. Please also refer to **Chapter 10.5.2**.

CUR



Bit	Function
C_OFFSET (3 … 0)	**Clipping Offset (Bit3 … 0)** The MSBs of C_OFFSET are defined by instruction CBR
CLIPPING _ADDR (23 … 0)	**Beginning of the clipping area** Bit 23 … 0 of a byte address.

CBR

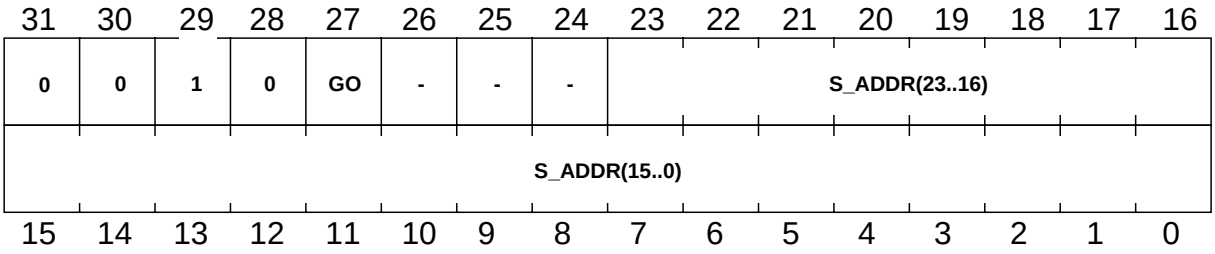


Bit	Function
HEIGHT_ CLIP (9 … 0)	**Height of the clipping area** The height of the clipping area can vary between 0 (HEIGHT_CLIP = '0_d') and 1023 pixels (HEIGHT_CLIP = '1023_d').
C_OFFSET (10 … 4)	**Clipping Offset (Bit 10 … 4)** The LSBs of C_OFFSET are defined by instruction CUR.
WIDTH_ CLIP (10 … 0)	**Width of the clipping area** The width of the clipping area can vary between 0 (WIDTH_CLIP = '0_d') and 2046 pixels (WIDTH_CLIP = '2046_d').

10.7.9 Source Descriptor for Data Transfer (SDR)

This instruction defines the beginning of the memory area to be read and transferred.

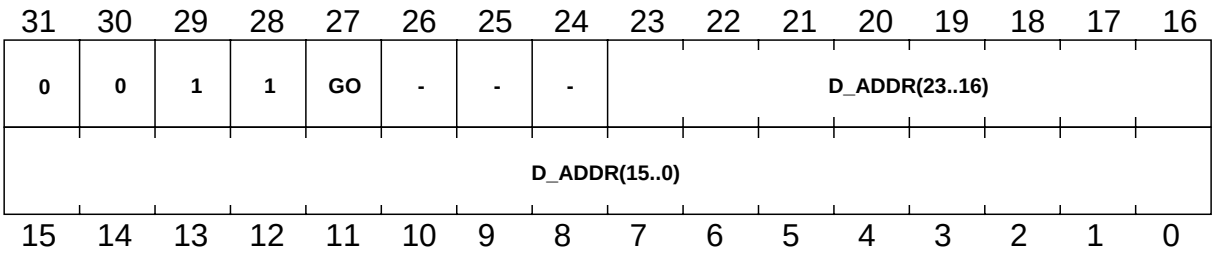
SDR



Bit	Function
GO	Must be set to ‘1’ if GA is to start memory transfer after executing this GA-instruction. Otherwise this bit must be set to ‘0’.
S_ADDR (23 … 0)	**Start address of memory area to be transferred.** Bit 23 … 0 of a byte address.

This instruction defines the beginning of the destination memory area.

DDR



Bit	Function
GO	Must be set to ‘1’ if GA is to start memory transfer after executing this GA-instruction. Otherwise this bit must be set to ‘0’.
D_ADDR (23 … 0)	**Beginning of the destination memory area** Bit 23 … 0 of a byte address.

10.7.10 Source Size of Transferred Memory Area (TSR)

This register contains different information depending on transfer mode. The size of transferred memory is described by width and height of the source of the transfer area. Please also refer to **Chapter 10.5.2**.

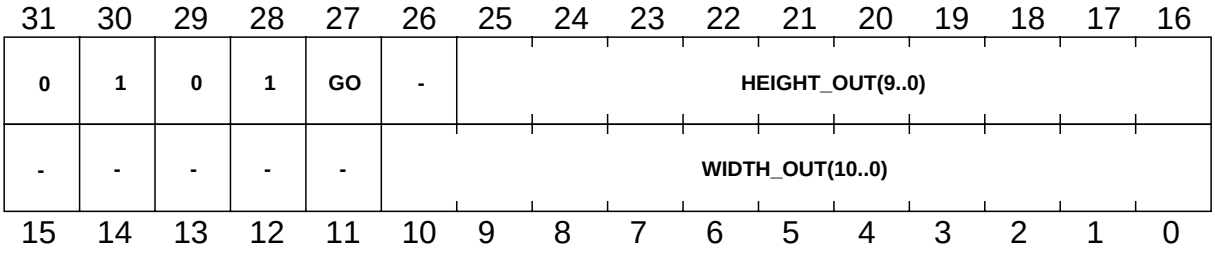
TSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	1	0	0	GO	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	WIDTH_IN(10..0)										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit	Function
GO	Must be set to ‘1’ if GA is to start memory transfer after executing this GA-instruction. Otherwise this bit must be set to ‘0’.
WIDTH_IN (10 … 0)	**Width of the transferred area in count of pixels.** WIDTH_IN = ‘0’: No transfer will be executed.

10.7.11 Destination Size of Transferred Memory Area (TDR)

TDR

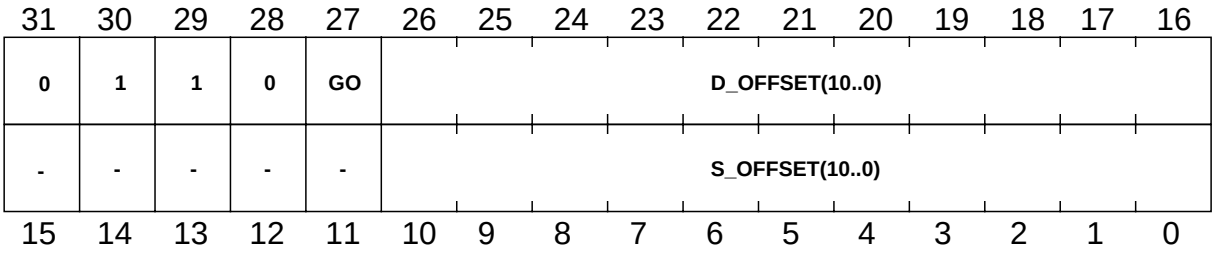


Bit	Function
GO	Must be set to ‘1’ if GA is to start memory transfer after executing this GA-instruction. Otherwise this bit must be set to ‘0’.
HEIGHT_ OUT (9 … 0)	**Height of the transferred area in count of pixels** HEIGHT_OUT = ‘0’: No transfer will be executed.
WIDTH_ OUT (10 … 0)	**Width of the transferred area in count of pixels** WIDTH_OUT = ‘0’: No transfer will be executed.

10.7.12 Offset of Transferred Memory Area (TOR)

This instruction defines the offset value for rectangle memory transfers.

TOR



Bit	Function
GO	Must be set to ‘1’ if GA is to start memory transfer after executing this GA-instruction. Otherwise this bit must be set to ‘0’.
S_OFFSET (10 … 0)	**Source offset value for non linear transfer** For more information about S_OFFSET refer to **Chapter 10.5.2**.
D_OFFSET (10 … 0)	**Destination offset value for non linear transfer** For more information about D_OFFSET refer to **Chapter 10.5.2**.

10.7.13 Attributes of Transfer (TAR)

This instruction defines the transfer mode.

TAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	1	1	1	GO	-	-	-	-	-	-	-	-	-	-	-
-	TQH	UN	CL(1..0)		TRM	TMODE(4..0)					IT	TDW	TDH	FLA(1..0)	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 10-7

Bit	Function
GO	Must be set to ‘1’ if GA is to start memory transfer after executing this GA-instruction. Otherwise this bit must be set to ‘0’.
TQH	**Quadruple Height during Transfer** ‘0’: Normal height is selected. ‘1’ The memory transfer is stretched in vertical direction on the output side.
UN	**Underline on/off** 0: Underline is switched off. 1: The last line (independent from ‘TDH’) in the destination area is filled with a constant CLUT1 input (vector 0 of CLUT1) instead of the source bitmap input.
CL (1 … 0)	**Clipping on/off** 00: Clipping is switched off. 01: Reserved. 10: Clipping is switched on. Pixels within the clipping area will be affected. 11: Clipping is switched on. Pixels outside the clipping area will be affected.
TRM	**Transparency Mode for Transfer** ‘0’: The complete area defined by instruction SDR and TDR is written to the destination. ‘1’: Only these (4:4:4:2) pixels are written to the destination area, for which Bit TR1 = ‘0’. *Note: This bit is only relevant for 16-bit (4:4:4:2) RGB output formats Please also refer to **Chapter 10.5.1**.*

Table 10-7 (cont'd)

Bit	Function
TMODE (4 … 0)	**Transfer Mode** This mode is used to decide which transfer mode should be used. With this bit it is decided which input and which output format is used for transformation. For detailed information of the register settings please refer to **Chapter 10.5.1**
IT	**Italic** This bit is used in transfer modes if 16-bit (4:4:4:2) RGB output mode is selected. '0': Italic is switched off. '1': Italic mode is enabled.
TDW	**Double Width during Transfer** '0': Normal width is selected. '1': The memory transfer is stretched in horizontal direction on the output side.
TDH	**Double Height during Transfer** '0': Normal height is selected. '1' The memory transfer is stretched in vertical direction on the output side. TQH must be set to '0'.
FLA(1 … 0)	**Flash Definition** This bit is used in transfer modes if output mode is in 16-bit (4:4:4:2) format for TTX. This bit is used to replace one of the 16 output bits. Please refer to **Chapter 10.5.1**. Bits Flash (1 … 0) describe the flash phase for memory transfer modes: '00': slow rate (1 Hz) '01': Fast rate flash (2 Hz) Phase 1 '10': Fast rate flash (2 Hz) Phase 2 '11': Fast rate flash (2 Hz) Phase 3

D/A Converter

11 D/A Converter

M2 uses a 3 × 6-bit voltage D/A converter to generate analog RGB output signals with a nominal amplitude of 0.7 V (also available: 0.5 V, 1.0 V and 1.2 V) peak to peak. Two different modes are available in order to allow the reduction of power consumption for applications which require a lower RGB bandwidth.

11.1 Register Description

DACCON **Reset Value: 0005_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	BWC	RGBGAIN (1..0)	
													rw	rw	

Bit	Function
RGBGAIN (1 … 0)	**Gain Adjustment of RGB Converter.** The user can change the output gain of the DAC. 00: 0.5 V 01: 0.7 V 10: 1.0 V 11: 1.2 V
BWC	**Bandwidth Control** 0: The effective bandwidth of the DAC is set to 50 MHz 1: The effective bandwidth of the DAC is set to 32 MHz. This reduces the current consumption of analog supply.

Slicer and Acquisition

12 Slicer and Acquisition

12.1 General Function

M2 provides a full digital slicer including digital H- and V-sync separation and digital sync processing. The acquisition interface is capable to process on all known data services starting from line 6 to line 23 for TV (Teletext, VPS, CC, G+, WSS) as well as any full channel services. Four different framing codes (two of them programmable from field to field) are available for each field. Digital signal processing algorithms are applied to compensate various disturbance mechanisms. These are:

- Noise measurement and compensation.
- Attenuation measurement and compensation.
- Group delay measurement and compensation.

Note: Thus, M2 is optimized for precise data clock recovery and error free reception of data widely unaffected from noise and the currently valid channel characteristics.

Two slicers with separated A/D converters and separated CVBS inputs are implemented. The first one is a full service slicer. The second one is a slicer which supports only the capturing of WSS data. Both CVBS inputs contain an on-chip clamping circuit. The integrated A/D converters are 7 bit video converters running at the internal frequency of 33.33 MHz.

The sliced data is synchronized to the frequency of the clock-run-in of the actual data service and to the framing code of the data stream, framing code checked and written to a programmable VBI buffer in the external memory. After line 23 is received an interrupt can be given to the microcontroller. The microcontroller starts to process the data of this buffer. That means, the data is error checked by software and stored in the memory in the appropriate data base format.

To improve the signal quality the slicer control logic generates horizontal and vertical windows in which the reception of the framing code is allowed. The framing code can be programmed for each line individually, so that in each line a different service can be received. For VPS and WSS the framing code is hardwired. In a special mode the framing code can be bypassed, so that all incoming data will be stored in the VBI buffer. The framing code check can then be made by software. All following acquisition tasks are performed by the internal controller, so in principal the data of every data service can be acquired.

12.2 Slicer Architecture

The slicer is composed of five main blocks:

- The full service slicer (Slicer 1)
- The WSS only slicer (Slicer 2)
- The H/V synchronization for full service slicer (Sync 1)

- The H/V synchronization for WSS only slicer (Sync 2)
- The acquisition interface

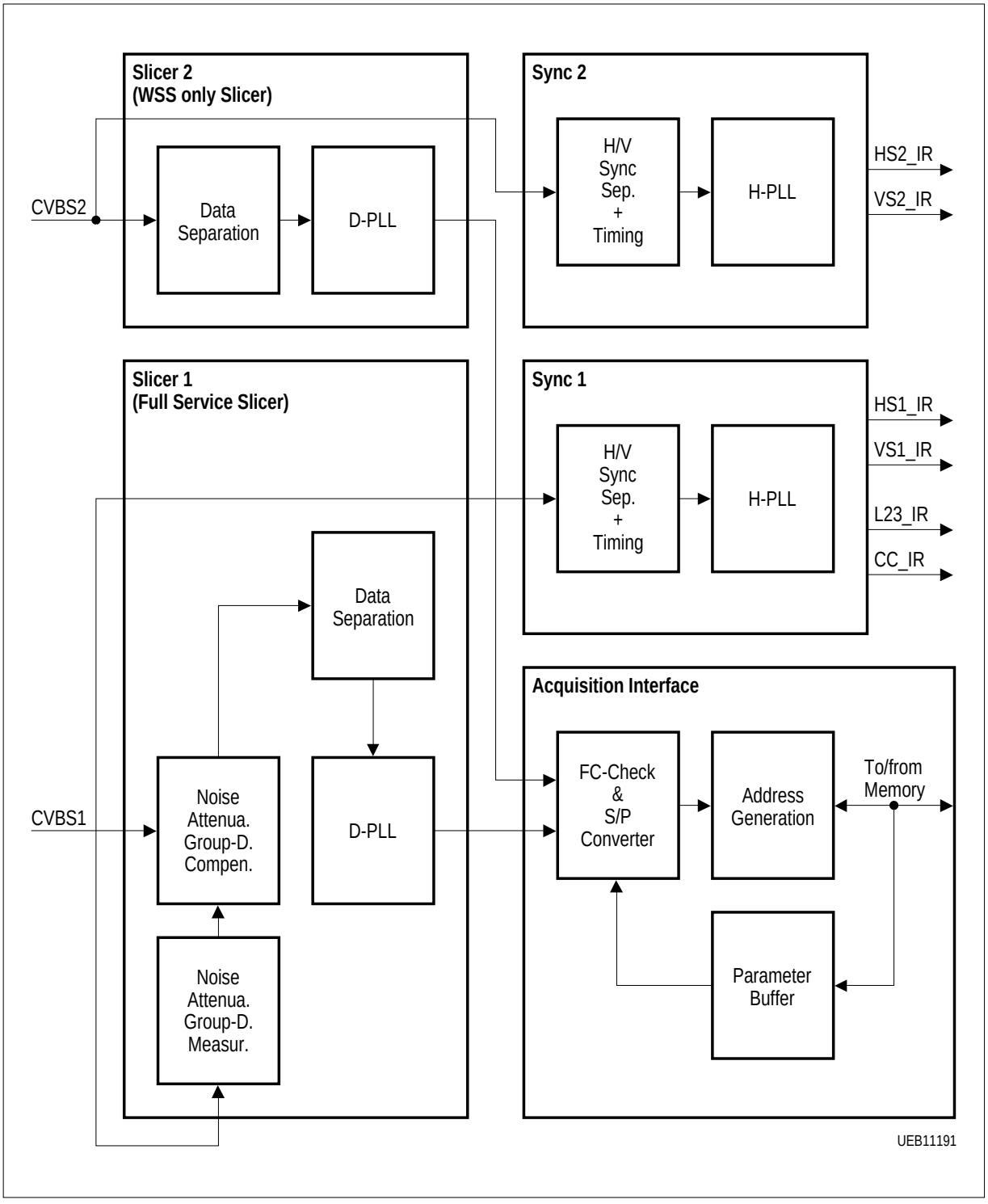


Figure 12-1 Block Diagram of Digital Slicer and Acquisition Interface

12.2.1 Distortion Processing

For the full service slicer the digital bit stream is applied to a circuitry which corrects transmission distortion. In order to apply the correct counter-measures, a signal evaluation is done in parallel. This measurement device can detect the following distortions.

Noise

The noise measurement unit incorporates two different algorithms. Both algorithm are using the value between two equalizing pulses which corresponds to the black level. As the black level is known to the system a window is placed between two equalizing pulses of line four. The first algorithm compares successive samples inside a window placed in line 4. The difference between this samples is measured and a flag is set as soon as this difference over several TV lines is greater than a specified value. This algorithm is able to detect higher frequency noise (e.g. with noise). The second algorithm measures the difference between the black value and the actual sampled value inside this window. As soon as this difference over several TV lines is greater than a specified value a second flag is set. This algorithm is sensitive against low frequency noise as it is known from co-channel distortion. Both flags can be used to optimize the correcting circuit characteristic in order to achieve best reception performance.

Frequency Attenuation

During signal transmission the CVBS can be attenuated severely. This attenuation normally is frequency depending. That means that the higher the frequency the stronger the attenuation. As the clock-run-in (from now on CRI) for teletext represents the highest possible frequency (3.5 MHz) it can be used to measure the attenuation. As only strong negative attenuation causes problems during data slicing a flag is needed to notify highly negative attenuation. If this flag is set a special peaking filter is switched on in the data-path.

Group Delay

Quite often the data stream is corrupted because of group delay distortion introduced by the transmission channel. The teletext framing code ($E4_H$) is used as a reference for measurement. The delay of the edges inside this code can be used to measure the group delay distortion. The measurement is done every teletext line and filtered over several lines. It can be detected whether the signal has positive, negative or no group delay distortions. Two flags are set accordingly. By means of these two flags an allpass contained in the correcting circuit is configured to compensate the positive or negative group delays. All of the above filters ca be individually disabled, forced or set to an automatic mode via control registers.

Note: Slicer 2 does not have any compensation circuits.

12.2.2 Data Separation

Parallel to the signal analyses and distortion compensation a filter is used to calculate the slicing level. The slicing level is the mean-value of the CRI. As the teletext is coded using the NRZ format, the slicing level can not be calculated outside the CRI and is therefore frozen after CRI. Using this slicing level the data is separated from the digital CVBS signal. The result is a stream of zeros and ones. In order to find the logical zeros and ones which have been transmitted, the data clock needs to be recovered as well. Therefore a digital data PLL (D-PLL) is synchronized to the data clock during CRI using the transitions in the sliced data stream. For TV-mode this D-PLL is also frozen after CRI, during VCR-mode it is tuned throughout the line using a slow time constant.

Timing informations for freezing the slicing level, stopping the D-PLL and other actions are generated by the timing circuit. It generates all control signals which are dependent on the data start.

In order to improve the reception performance the actual measured slicing level for each line is stored in the VBI-buffer. Using this slicing level the user is able to average the value over several fields for each data line by means of software filtering. If the averaged value becomes stable this value can be used for slicing instead of the internally calculated slicing level (for further information see RAM-register description).

For data separation the WSS slicer (slicer 2) uses the same algorithms as the full service slicer.

12.3 H/V-Synchronization

Slicer and acquisition interface need many signals synchronized to the incoming CVBS (e.g. line number, field or line start). Therefore a sync slicing level is calculated and the sync signal is sliced from the filtered digital CVBS signal. Using digital integration vertical and horizontal sync pulses are separated. The horizontal pulses are fed into a digital H-PLL which has flywheel functionality. The H-PLL includes a counter which is used to generate all the necessary horizontal control signals. The vertical sync is used to synchronize the line counter from which the vertical control signals are derived.

The synchronization block includes a watchdog which keeps control of the actual lock condition of the H-PLL.The watchdog can produce an interrupt (CC_IR) if synchronization has been lost. It could therefore be an indication for a channel change or missing input signal.

Note: This H/V synchronization for the slicer 2 uses the same algorithms as described above.

12.4 Acquisition Interface

The acquisition interface manages the data transfer between both slicers and memory. First of all a byte synchronization is performed (FC-check). Following this, the data is

paralleled and shifted into memory as 16 bit words. In the other direction parameters are loaded from memory to the slicer. After every vertical sync, parameters needed for the field are downloaded and after every horizontal sync line parameters are downloaded. The parameters are used for slicer configuration.

The data acquisition supports several features. The FC-checker is able to handle four different framing codes for one field. Two of these framing codes are programmable and could therefore be changed from field to field. The acquisition can be switched from normal mode (line 6 to 23) to full channel mode (line 6 to end of field).

12.4.1 FC-Check

There are four FC's which are compared to the incoming signal. The first one is 8-bit wide and is loaded down with the field parameters. The second one is 16-bit wide and fixed to the FC of VPS. The third one is also 16-bit wide, but can be loaded with the field parameters. If the third one is used, the user can specify not only the FC but also a don't-care mask. The fourth FC is reserved for WSS. The actual FC can be changed line by line.

FC1

This FC should be used for all services with 8-bit framing codes (e.g. for TTX). The actual framing code is loaded down each field. The check can be done without any error tolerance or with a one bit error tolerance.

Note: If FC1 = $E4_H$ this pattern is used as a reference for group delay measurement.

FCVPS

This FC is fixed to that of VPS. Only an error free signal will enable the reception of the VPS data line.

Note: If VPS should be sliced in field 1 and TTX in field 2 the appropriate line parameters for line 16 have to be dynamically changed from field to field.

FC3

This 16-bit framing code is loaded with the field parameters as well as a don't care mask. The incoming signal is compared to both, framing code and don't care mask. Further reception is enabled if all bits, which are not don't care, match the incoming data stream.

FCWSS

This FC is fixed to that of WSS. A special algorithm makes sure that the WSS-FC is detected even if the CVBS signal is coming from a video tape.

No FC-check

If FC-check is disabled, the data recording is triggered by the data start recognition. In this case the software needs to do the byte synchronization.

FC-Check Select

There is a two bit line parameter called FCSEL. With this parameter the user will be able to select which FC-Check is used for the actual line. If NORM is set to WSS the WSS FCcheck is used independently of FCSEL.

12.4.2 Interrupts

Some events which occur inside the slicer, the sync separation or the acquisition interface can be used to trigger an interrupt. They are summarized in register ACQISN. The hardware sets the associated interrupt flag which must be manually reset by SW before the next interrupt can be accepted. All ACQ interrupts are bundled into one interrupt which is fed to the ACQ-interrupt node (ACQIC) of the controller.

12.4.3 VBI Buffer and Memory Organization

Slicer and acquisition interface need parameters for configuration and they produce status information for the CPU.

Some of these parameters and status bits are constant for a field. Those parameters are called field parameters. They are downloaded after the vertical sync of slicer 1. If the synchronization of slicer 1 is missing the vertical sync from slicer 2 is used to initialize the parameter download.

Other parameters and status bits may change from line to line (e.g. data service dependent values). Those parameters are called line parameters. They are downloaded after each horizontal sync.

The start address of the VBI (VBI = vertical blanking interval) buffer can be configured with special function register ‘STRVBI’. 32 16-bit words have to be reserved for every sliced data line. If 18 (in full channel mode 350) lines of data have been sent to memory no further acquisition takes place until the next vertical pulse appears and the H-PLL is still locked. That means if at least 1176 Bytes (22424 Bytes in full channel mode) are reserved for the VBI buffer no VBI overflow is possible. The acquisition can be started and stopped by the controller using bit ‘ACQON’ of register STRVBI. The acquisition is stopped as soon as this bit changed to ‘0’. If the bit is changed back to ‘1’ switching on of the acquisition is synchronized to the next V-pulse. The start address (Bit 13 … 0 of register STRVBI) of the VBI buffer should only be changed if the acquisition is switched off.

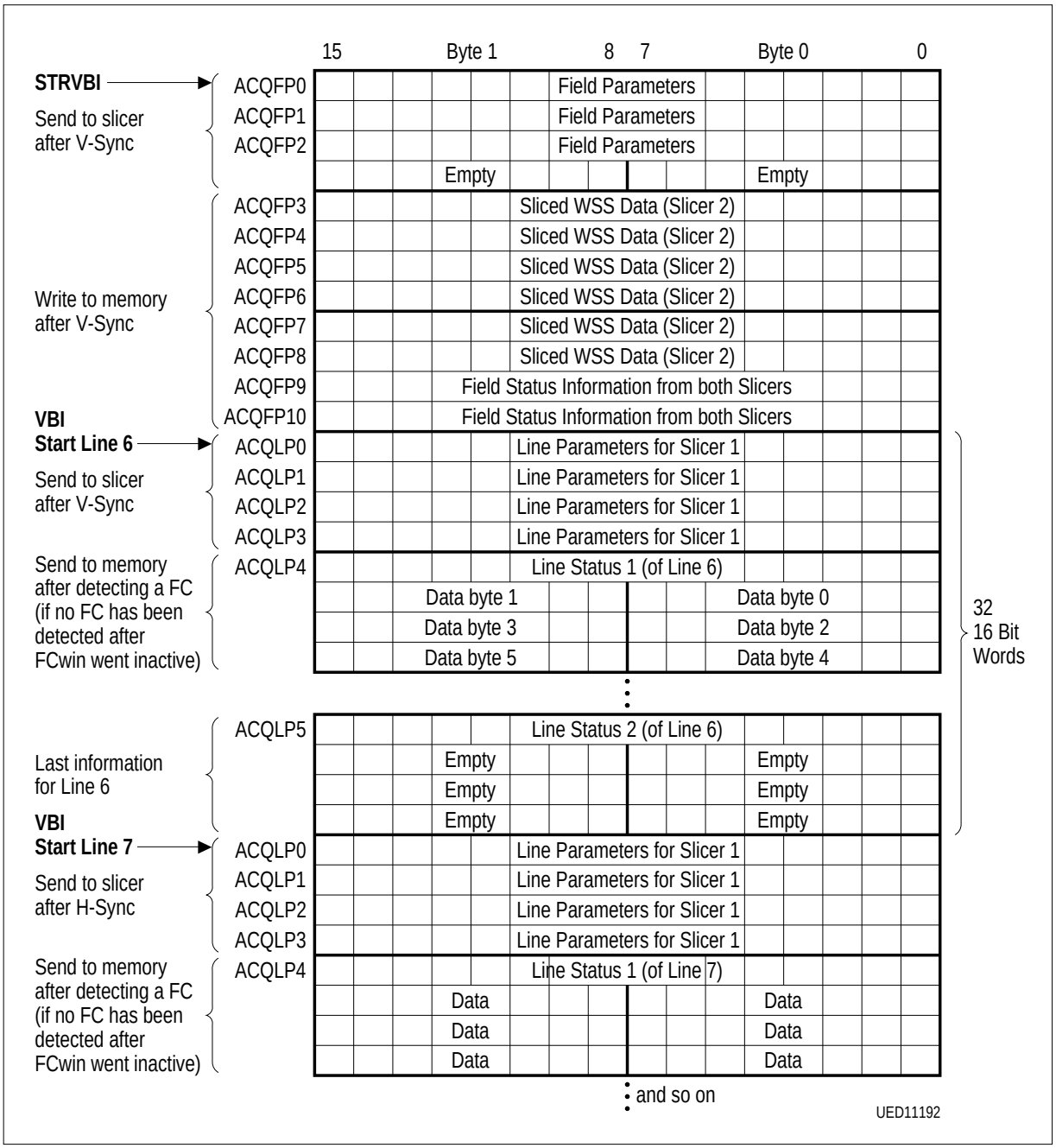


Figure 12-2 VBI Buffer: General Structure

12.5 Register Description

The acquisition interface has only two SFR Registers. The line and field parameters are stored in the RAM (RAM Registers). They have to be initialized by software before starting the acquisition.

Special Function Registers:

STRVBI **Reset Value: 0400_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACQ ON	0	VBIADR(23..10)													

Bit	Function
ACQON	**Enable Acquisition** 0: The ACQ interface does not access memory (immediately inactive) 1: The ACQ interface is active and writes data to memory (switching on is synchronous to V)
VBIADR (23 … 10)	**Define the 14 MSB's of the start address of the VBI buffer. The VBI buffer location can be aligned with any 1 KByte memory segment.**

ACQISN **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	CC_IE	CC_IR	L23_ IE	L23_ IR	HS2_ IE	HS2_ IR	VS2_ IE	VS2_ IR	HS1_ IE	HS1_ IR	VS1_ IE	VS1_ IR
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
VS1_IR	**VS interrupt**. The vertical sync impulse can be used to have field synchronization for the software. (VS of slicer 1 is used). 0: No request pending. 1: This source has raised an interrupt request.
VS1_IE	**Interrupt Enable Bit** 0: Disables the interrupt. 1: Enables the interrupt.
HS1_IR	**HS interrupt**. The horizontal sync impulse can be used to implement a software line counter. (HS of slicer 1 is used). 0: No request pending. 1: This source has raised an interrupt request.

Bit	Function
HS1_IE	**Interrupt Enable Bit** 0: Disables the interrupt. 1: Enables the interrupt.
VS2_IR	**VS interrupt**. The vertical sync impulse can be used to have field synchronization for the software. (VS of slicer 2 is used). 0: No request pending. 1: This source has raised an interrupt request.
VS2_IE	**Interrupt Enable Bit** 0: Disables the interrupt. 1: Enables the interrupt.
HS2_IR	**HS interrupt**. The horizontal sync impulse can be used to implement a software line counter. (HS of slicer 2 is used). 0: No request pending. 1: This source has raised an interrupt request.
HS2_IE	**Interrupt Enable Bit** 0: Disables the interrupt. 1: Enables the interrupt.
L23_IR	**Line 23 Interrupt**.Tells the controller that line 23 of the VBI is sliced (Slicer 1 is used). 0: No request pending. 1: This source has raised an interrupt request.
L23_IE	**Interrupt Enable Bit** 0: Disables the interrupt. 1: Enables the interrupt.
CC_IR	**Channel Change Indicator** The H-PLL has lost the synchronization. (Slicer 1 is used). 0: No request pending. 1: This source has raised an interrupt request. *Note: Also refer to status bits STAB1 or VDOK1*
CC_IE	**Interrupt Enable Bit** 0: Disables the interrupt. 1: Enables the interrupt.

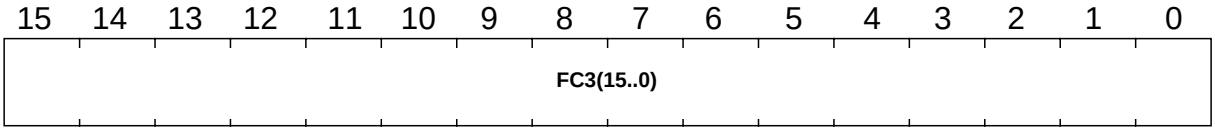
Note: The interrupt request flags of the ACQ interrupt subnode have to be cleared by software within the interrupt service routine.

12.5.1 RAM Registers

Field Parameters

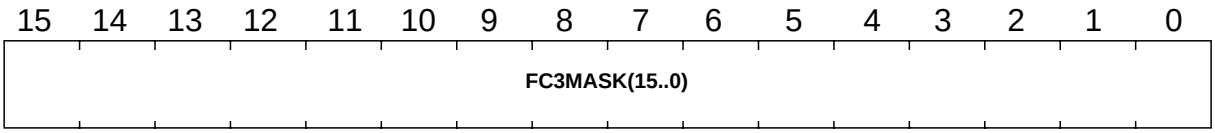
All field parameters are updated once in a field. This means that the status information written from the acquisition interface to the memory at that time only represents a snapshot of the status. Hardware ensures that field parameters are updated even if only one of the two CVBS signals has a valid sync timing. So it is assured that even if CVBS1 is not available data of CVBS2 still can be sliced.

ACQFP0 **Reset Value: $XXXX_H$**



Bit	Function
FC3 (15 … 0)	**Framing code 3** Bit 15: First received bit of FC. Bit 0: Last received bit of FC.

ACQFP1 **Reset Value: $XXXX_H$**



Bit	Function
FC3MASK (15 … 0)	**Mask for Framing code 3** Bit 15: Mask for first received bit of FC. Bit 0: Mask for last received bit of FC.

ACQFP2 **Reset Value: $XXXX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FC1(7..0)								AGD ON	AFR ON	ANO OM	0	0	0	0	FULL

Bit	Function
FC1 (7 … 0)	**Framing code 1** Bit 7: First received bit of FC Bit 0: Last received bit of FC
AGDON	**Automatic group delay compensation** 0: Automatic compensation Off 1: Automatic compensation On (Automatic: Measurement Depending Compensation)
AFRON	**Automatic frequency depending attenuation compensation** 0: Automatic compensation Off 1: Automatic compensation On (Automatic: measurement depending compensation)
ANOON	**Automatic noise compensation** 0: Automatic compensation Off 1: Automatic compensation On (Automatic: measurement depending compensation)
FULL	0: Full channel mode off 1: Full channel mode on *Note: Don't forget to reserve enough memory for the VBI buffer and to initialized the appropriate line parameters.*

ACQFP3

Reset Value: XXXX$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WSS2OK	WSS2_ACK	0	0	0	0	0	0	0	0	0	0	WSS2_DATA(83..80)			

Bit	Function
WSS2OK	0: No new WSS data from slicer 2 is available 1: New WSS data from slicer 2 is available (written to memory by ACQ-interface)
WSS2_ACK	0: WSS data from slicer 2 are the same as in last slicer 1 field 1: New WSS data from slicer 2 received
WSS2_ DATA (83 … 80)	4 bits of sliced data of slicer 2 (WWS2_DATA(83) = first received bit) (written to memory by ACQ-interface) *Note: See also ACQFP4 t0 ACQFP8*

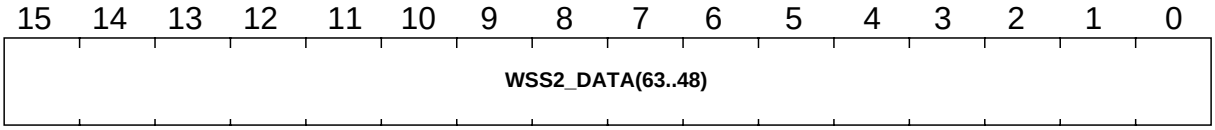
ACQFP4

Reset Value: XXXX$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WSS2_DATA(79..64)															

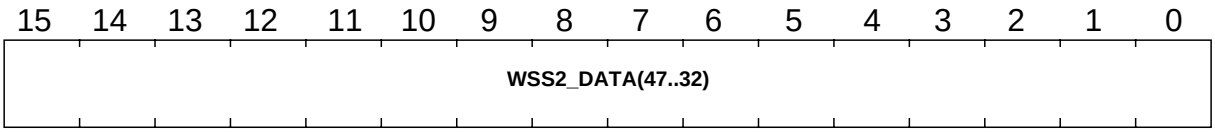
Bit	Function
WSS2_ DATA (79 … 64)	16 bits of sliced data of slicer 2 (written to memory by ACQ-interface). *Note: See also ACQFP3 and ACQFP5 to ACQFP8*

ACQFP5 **Reset Value: XXXX$_H$**



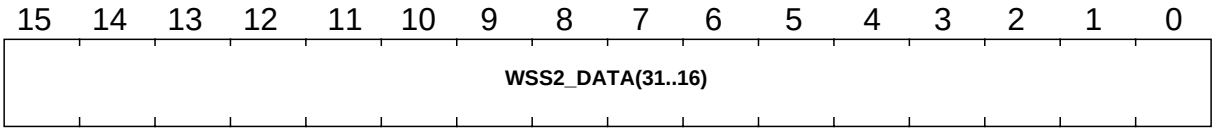
Bit	Function
WSS2_ DATA (63 … 48)	16 bits of sliced data of slicer 2 (written to memory by ACQ-interface). *Note: See also ACQFP3, ACQFP4, ACQFP6 to ACQFP8*

ACQFP6 **Reset Value: XXXX$_H$**



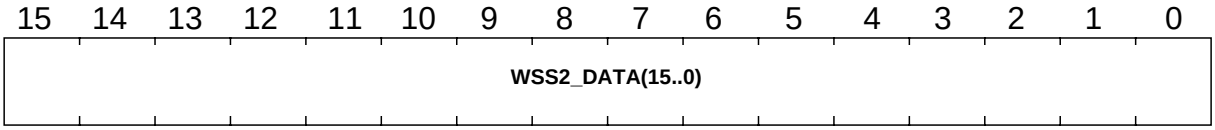
Bit	Function
WSS2_ DATA (47 … 32)	16 bits of sliced data of slicer 2 (written to memory by ACQ-interface). *Note: See also ACQFP3 to ACQFP5 and ACQFP7 to ACQFP8*

ACQFP7 **Reset Value: XXXX$_H$**



Bit	Function
WSS2_ DATA (31 … 16)	16 bits of sliced data of slicer 2 (written to memory by ACQ-interface). *Note: See also ACQFP3 to ACQFP6 and ACQFP8*

ACQFP8 **Reset Value: XXXX$_H$**



Bit	Function
WSS2_ DATA (15 … 0)	16 bits of sliced data of slicer 2 (WWS2_DATA(0) = last received bit) (written to memory by ACQ-interface). *Note: See also ACQFP3 to ACQFP7*

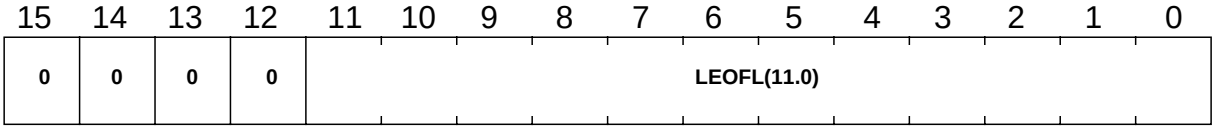
ACQFP9 **Reset Value: XXXX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	STAB 1	VOK 1	FIELD 1	FREA TTF	NOIS E(1)	NOIS E(0)	GRD ON	GRD SIGN	STAB 2	VOK 2	FIELD 2

Bit	Function
STAB1 (status bit)	0: H-PLL of slicer 1 not locked 1: H-PLL of slicer 1 locked (Written to memory by ACQ-interface)
VOK1 (status bit)	**Vertical sync watchdog of slicer 1** 0: V-sync of slicer 1 not stable 1: V-sync of slicer 1 stable (Written to memory by ACQ-interface)
FIELD1 (status bit)	0: Actual field of slicer 1 is field 1 1: Actual field of slicer 1 is field 2 (Written to memory by ACQ-interface)
FREATTF (status bit)	**Frequency depending attenuation measurement** (Field indicator) High frequency CVBS1components (around 3.5 MHz) are strongly damped (6 to 9 dB) compared to lower frequency CVBS1 components 0: no frequency depending attenuation has been detected during the last field 1: for at least one text line during the last field frequency depending attenuation has been detected. (Written to memory by ACQ-interface)

Bit	Function
NOISE (1 … 0) (status bit)	**Noise and co-channel detector of slicer 1** 00: No noise and no co-channel-distortion has been detected. 01: No noise but **co-channel-distortion** has been detected. 10: **Noise** but no co-channel-distortion has been detected. 11: **Strong noise** has been detected. (Written to memory by ACQ-interface)
GRDON (status bit)	**Group delay detector of slicer 1** 0: No group delay distortion detected 1: Group delay distortion detected (Written to memory by ACQ-interface)
GRDSIGN (status bit)	0: If group delay distortion has been detected it was positive 1: If group delay distortion has been detected it was negative (Written to memory by ACQ-interface, CVBS input of slicer 1 is used)
STAB2 (status bit)	0: H-PLL of slicer 2 not locked 1: H-PLL of slicer 2 locked (Written to memory by ACQ-interface)
VOK2 (status bit)	**Vertical sync watchdog of slicer 2** 0: V-sync of slicer 2 not stable 1: V-sync of slicer 2 stable (Written to memory by ACQ-interface)
FIELD2 (status bit)	0: Actual field of slicer 2 is field 1 1: Actual field of slicer 2 is field 2 (Written to memory by ACQ-interface)

ACQFP10 **Reset Value: $XXXX_H$**



Bit	Function
LEOFLI (11 … 0)	This value is the output of the filter of the H-PLL of slicer 1 and represents the actual horizontal period of CVBS1 in 33.33 MHz clock cycles. This information can be used to measure the actual line frequency of the CVBS signal.

Line Parameters

Note: Line parameters only work on slicer 1 and have no influence on slicer 2.

ACQLP0 **Reset Value: XXXX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DINCR(15..0)															

Bit	Function
DINCR (15 … 0)	Specifies the frequency of the D-PLL of slicer 1. This parameter is used to configure the D-PLL output frequency according to the service used. **DINCR = $f_{data} \times 2^{18}$ / 33.33 MHz** **f_{data} [MHz]** — **DINCR** 6.9375 — 54559 5.7273 — 45041 5.0 — 39321 1.006993 — 7920

ACQLP1 **Reset Value: XXXX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	ACCU ON	PLLT ON	LOWP ON	PFIL ON	GDPO N	GDNO N	FRE ON	NOI ON

Bit	Function
ACCUON	**Accumulator on** Improves slicing level calculation under noisy conditions. If noise has been detected during automatic mode or if the bit NOION has been set the internal slicing level calculation can be improved by setting this bit. 0: Standard slicing level calculation 1: Improved slicing level calculation (improvement depends also on parameter ALENGTH)
PLLLON	**PLL tune on** If noise has been detected during automatic mode or if the bit NOION has been set the data clock recovery PLL can be tuned throughout the line by setting this bit. 0: PLL is frozen after clock run in 1: PLL is tuned throughout the line
LOWPON	**Low Pass On** If noise has been detected during automatic mode or if the bit NOION has been set a special low pass can be switched into the signal pass by setting this bit (useful if mainly high frequency noise above 3.5 MHz is present). 0: Low pass is not used 1: Low pass is used
PFILLON	**Pre Filter On** If noise has been detected during automatic mode or if the bit NOION has been set a second filter can be switched into the signal pass by setting this bit (also useful if mainly high frequency noise above 3.5 MHz is present). 0: Low pass is not used. 1: Low pass is used.
GDPON	0: Group delay compensation depends on AGDON 1: Positive group delay compensation is always on
GDNON	0: Group delay compensation depends on AGDON 1: Negative group delay compensation is always on
FREON	0: Frequency depending attenuation compensation depends on AFRON 1: Frequency depending attenuation compensation is always on
NOION	0: Noise compensation depends on ANOON 1: Noise compensation is always on

ACQLP2 **Reset Value: $XXXX_H$**

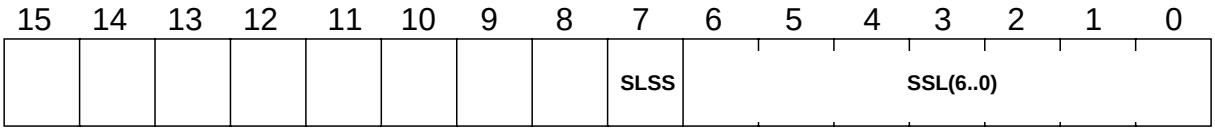
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FC1 ER	MLENGTH(2..0)			ALENGTH (1..0)		CLKDIV(2..0)			NORM(2..0)			FCSEL(1..0)		VCR	0

Bit	Function
FC1ER	Error tolerance of FC1 check 0: No error allowed 1: One error allowed
MLENGTH (2 … 0)	For noise suppression reasons a median filter has been introduced after the actual data separation because of oversampling successive samples could be averaged. Therefore an odd number of sliced successive samples is taken and if the majority are ‘1’ a ‘1’ is sliced otherwise a ‘0’. MLENGTH specifies how many samples are taken. **MLENGTH** — **Number of samples** 000 — 1 001 — 3 010 — 5 011 — 7 100 — 9 101 — 11 110 — 13 111 — 15
ALENGTH (1 … 0)	If noise has been detected or if NOISEON = 1, the output of the slicing level filter is further averaged by means of an accumulation (arithmetic averaging). ALENGTH specifies the number of slicing level filter output values used for averaging. The accumulation clock depends on CLKDIV. **ALENGTH** — **Number of Slicing Level Output Values used for Averaging** 00 — 2 01 — 4 10 — 8 11 — 16

Bit	Function
CLKDIV	The slicing level filter needs to find the DC value of the CVBS during CRI. In order to do this it should suppress at least the CRI frequency. As different services use different data frequencies the CRI frequency will be different as well. Therefore the filter characteristic needs to be shifted. This can be done by using different clocks for the filter. The filter itself shows sufficient suppression for frequencies between $0.0757 \times SL_{CLK}$ and $0.13 \times SL_{CLK}$ (SL_{CLK} is the actual filter clock and corresponds to slicer 1) **CLKDIV** — **SL_{CLK}** 000 — $1 \times f_s$ 001 — $1/2 \times f_s$ 010 — $1/3 \times f_s$ 011 — $1/4 \times f_s$ 100 — $1/5 \times f_s$ 101 — $1/6 \times f_s$ 110 — $1/7 \times f_s$ 111 — $1/8 \times f_s$ *Note: f_s = 33.33 MHz*
NORM	Most timing signals are closely related to the actual data service used. Therefore 3 bits specify the service received in the actual line. **NORM** — **Service** 000 — TXT 001 — NABTS 010 — VPS 011 — WSS 100 — CC 101 — G+ 110 — reserved 111 — no data service

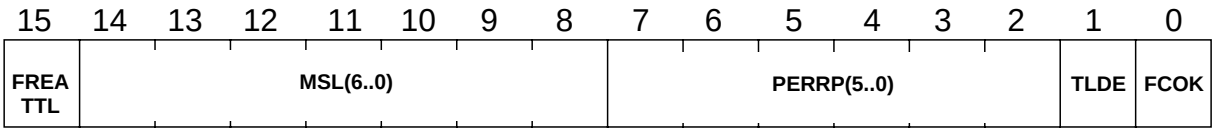
Bit	Function
FCSEL	There are three different framing codes which can be used for each field. The framing code used for the actual line is selected with FCSEL (corresponds to slicer 1). **FCSEL** **FC** 00 FC1 01 FC2 10 FC3 11 No FC-check (all data are dumped to the VBI buffer)
VCR	**This bit is used to change the behavior of the D-PLL and H-PLL.** 0: D-PLL tuning is stopped after CRI; H-PLL -> slow time constant 1: D-PLL is tuned throughout the line; H-PLL -> fast time constant

ACQLP3 **Reset Value: $XXXX_H$**



Bit	Function
SLSS	**Slicing Level Source Selector** The slicer allows the use of an internal calculated slicing level or an external set slicing level. 0: Internal calculated slicing level is used. 1: External set slicing level is used.
SSL(6 … 0)	**Set Slicing Level** If the bit SLSS is set the slicer is using the value of SSL as slicing level instead of the internal calculated slicing level. The slicing level output in parameter MSL is never the less the internal calculated value.

ACQLP4 **Reset Value: XXXX$_H$**



Bit	Function
FREATTL	**Frequency Depending Attenuation Measurement** (Line indicator) High frequency-CVBS1-components (around 3.5 MHz) are strongly damped (6 to 9 dB) compared to lower frequency-CVBS1-components 0: no frequency depending attenuation has been detected for the following line 1: strong frequency depending attenuation has been detected for the following line (Written to memory by ACQ-interface)
MSL(6 … 0)	**Measured Slicing Level** The value represents the slicing level which has been measured for the current data line. The value can be used to calculate a better slicing level especially for noisy signals by means of software averaging algorithms. The improved slicing level can be set for the following fields by writing to parameter SSL.
PERRP (5 … 0)	**Phase Error Watch Dog Preliminary** (detection of test line CCIR331a or b) The value shows how often in a line the internal PLL found strong phase deviations between PLL and sliced data. The value can be used to detect test line CCIR331a or b. This value is only preliminary as an exact result is only available at the end of each line. For the exact value see PERR at ACQLP5. PERRP < 32? No test line. PERRP > 31? Test line CCIR331a or b detected
TLDE	**Test Line Detected** (CCIR17 or CCIR18 or CCIR330) 0: No test line of the above mentioned test lines has been detected 1: The following data has most likely be sliced from a test line and should therefor be ignored.
FCOK	**Framing Code Received** 0: No framing code has been detected (no new data has been written to memory) 1: The selected framing code has been detected (new data has been written to memory

ACQLP5 **Reset Value: $XXXX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	PERR(5..0)					

Bit	Function
PERR (5 … 0)	**Phase Error Watch Dog** (detection of test line CCIR331a or b) This is the exact phase error watch dog output for the current line. The value shows how often in a line the internal PLL found strong phase deviations between PLL and sliced data. The value can be used to detect test line CCIR331a or b. PERR < 32? No test line. PERR > 31? Test line CCIR331a or b detected

12.5.2 Recommended Parameter Settings

	TTX	VPS	WSS	CC	G+
AGDON	1	0	0	0	0
AFRON	1	0	0	0	0
ANOON	1	1	1	1	1
GDPON	0	0	0	0	0
GDNON	0	0	0	0	0
FREON	0	0	0	0	0
PFILON	0	1	1	1	1
LOWPON	0	1	1	1	1
PLLTON	0	1	1	1	1
ACCUON	0	1	1	1	1
NOION	0	0	0	0	0
FULL	0	0	0	0	0
DINCR	54559	45041	39321	7864	7920
FC1ER	0	0	0	0	0
MLENGTH	1	2	7	7	7
ALENGTH	2	2	2	2	2
CLKDIV	0	0	2	5	5
NORM	0	2	3	4	5
FCSEL	0	1	2	2	2
VCR	0	0	0	0	0
FC1	228	don’t care	don’t care	don’t care	don’t care
FC3	don’t care	don’t care	don’t care	3	1261
FC3MASK	don’t care	don’t care	don’t care	65472	63488

Register Overview

13 Register Overview

This section summarizes all SFR and ESFR registers, which are implemented in M2 and explains the description format which is used in the previous chapters to describe the functionality of the SFRs. Display generators and slicers are mainly programmed via RAM registers which are not mentioned in this chapter, due to their variable position in the RAM. RAM registers are principally undefined after reset.

For easy reference the registers are ordered according to two different keys:

- Ordered by their functional context
- Ordered by register address, to find the location of a specific register.

13.1 Register Description Format

In the respective chapters the function and the layout of the SFRs is described in a specific format which provides a number of details about the described special function register. The example below shows how to interpret these details.

A register looks like this:

REG_NAME **Reset Value: ****$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	write only	read only	std bit	hw bit	bitfield(2:0)		
									w	r	rw	rw		rw	

Bit	Function
bit(field) name	Explanation of bit(field)name *Description of the functions controlled by this bit(field)*.

REG_NAME Name of this register.

*** * * *** Register contents after reset.
0/1: defined value, '**X**': undefined after power up.

rw Bits that are set/cleared by hardware are marked with a shaded access box.

r Read only register.

rw Register can be read and written.

- Reserved register bit. Reading such bits delivers an undefined value. If such a bit is written then only '0' is allowed.

13.2 CPU General Purpose Registers (GPRs)

The GPRs form the register bank with which the CPU works. This register bank may be located anywhere within the internal RAM via the Context Pointer (CP). Due to the addressing mechanism, GPR banks can only reside within the internal RAM. All GPRs are bit-addressable.

Name	Physical Address	8-Bit Address	Description	Reset Value
R0	(CP) + 0	$F0_H$	CPU General Purpose (Word) Register R0	$XXXX_H$
R1	(CP) + 2	$F1_H$	CPU General Purpose (Word) Register R1	$XXXX_H$
R2	(CP) + 4	$F2_H$	CPU General Purpose (Word) Register R2	$XXXX_H$
R3	(CP) + 6	$F3_H$	CPU General Purpose (Word) Register R3	$XXXX_H$
R4	(CP) + 8	$F4_H$	CPU General Purpose (Word) Register R4	$XXXX_H$
R5	(CP) + 10	$F5_H$	CPU General Purpose (Word) Register R5	$XXXX_H$
R6	(CP) + 12	$F6_H$	CPU General Purpose (Word) Register R6	$XXXX_H$
R7	(CP) + 14	$F7_H$	CPU General Purpose (Word) Register R7	$XXXX_H$
R8	(CP) + 16	$F8_H$	CPU General Purpose (Word) Register R8	$XXXX_H$
R9	(CP) + 18	$F9_H$	CPU General Purpose (Word) Register R9	$XXXX_H$
R10	(CP) + 20	FA_H	CPU General Purpose (Word) Register R10	$XXXX_H$
R11	(CP) + 22	FB_H	CPU General Purpose (Word) Register R11	$XXXX_H$
R12	(CP) + 24	FC_H	CPU General Purpose (Word) Register R12	$XXXX_H$
R13	(CP) + 26	FD_H	CPU General Purpose (Word) Register R13	$XXXX_H$
R14	(CP) + 28	FE_H	CPU General Purpose (Word) Register R14	$XXXX_H$
R15	(CP) + 30	FF_H	CPU General Purpose (Word) Register R15	$XXXX_H$

The first 8 GPRs (R7 **...** R0) may also be accessed via the byte. Other than with SFRs, writing to a GPR byte does not affect the other byte of the respective GPR.
The respective half of the byte-accessible registers receive special names:

Name	Physical Address	8-Bit Address	Description	Reset Value
RL0	(CP) + 0	$F0_H$	CPU General Purpose (Byte) Register RL0	XX_H
RH0	(CP) + 1	$F1_H$	CPU General Purpose (Byte) Register RH0	XX_H
RL1	(CP) + 2	$F2_H$	CPU General Purpose (Byte) Register RL1	XX_H
RH1	(CP) + 3	$F3_H$	CPU General Purpose (Byte) Register RH1	XX_H
RL2	(CP) + 4	$F4_H$	CPU General Purpose (Byte) Register RL2	XX_H
RH2	(CP) + 5	$F5_H$	CPU General Purpose (Byte) Register RH2	XX_H
RL3	(CP) + 6	$F6_H$	CPU General Purpose (Byte) Register RL3	XX_H
RH3	(CP) + 7	$F7_H$	CPU General Purpose (Byte) Register RH3	XX_H
RL4	(CP) + 8	$F8_H$	CPU General Purpose (Byte) Register RL4	XX_H
RH4	(CP) + 9	$F9_H$	CPU General Purpose (Byte) Register RH4	XX_H
RL5	(CP) + 10	FA_H	CPU General Purpose (Byte) Register RL5	XX_H
RH5	(CP) + 11	FB_H	CPU General Purpose (Byte) Register RH5	XX_H
RL6	(CP) + 12	FC_H	CPU General Purpose (Byte) Register RL6	XX_H
RH6	(CP) + 13	FD_H	CPU General Purpose (Byte) Register RH6	XX_H
RL7	(CP) + 14	FE_H	CPU General Purpose (Byte) Register RL7	XX_H
RH7	(CP) + 15	FF_H	CPU General Purpose (Byte) Register RH7	XX_H

13.3 Registers Ordered by Context

The following table lists all SFRs which are implemented in the M2 grouped by their context. Their actual address can be seen in the next chapter.

Table 13-1

Name	Description	Physical Address	8-Bit Address	Reset Value
SSC Registers				
SSCCON	Control Register	$FFB2_H$	$D9_H$	0000_H
SSCBR	Baud Rate Timer Reload Register	$F0B4_H$	$5A_H$	0000_H
SSCTB	Transmit Buffer Register	$F0B0_H$	58_H	0000_H
SSCRB	Receive Buffer Register	$F0B2_H$	59_H	0000_H
ASC Registers				
S0CON	Control Register	$FFB0_H$	$D8_H$	0000_H

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
S0ABSTAT	Autobaud Status Register	$F0B8_H$	$5C_H$	0000_H
S0ABCON	Autobaud Control Register	$F1B8_H$	DC_H	0000_H
S0BG	Baud Rate Timer Reload Register	$FEB4_H$	$5A_H$	0000_H
S0FDV	Fractional Divider Register	$FEB6_H$	$5B_H$	0000_H
S0PMW	IrDA Pulse Mode and Width Register	$FEAA_H$	55_H	0000_H
S0TBUF	Transmit Buffer Register	$FEB0_H$	58_H	0000_H
S0RBUF	Receive Buffer Register	$FEB2_H$	59_H	0000_H
I^2C Registers				
ICCFG[2)]	I^2C Configuration Register	$E810_H$	-/-	0000_H
ICCON[2)]	I^2C Control Register	$E812_H$	-/-	0000_H
ICST[2)]	I^2C Status Register	$E814_H$	-/-	0000_H
ICADR[2)]	I^2C Address Register	$E816_H$	-/-	0000_H
ICRTBL[2)]	I^2C Receive/Transmit Buffer (Low Word)	$E818_H$	-/-	0000_H
ICRTBH[2)]	I^2C Receive/Transmit Buffer (High Word)	$E81A_H$	-/-	0000_H
IICPISEL[2)]	I^2C Port Input Selection Register	$E804_H$	-/-	0000_H
Watchdog Timer Registers				
WDTCON	Control Register	$FFAE_H$	$D7_H$	$000X_H$
WDT	Timer Register	$FEAE_H$	57_H	0000_H
Realtime Clock Registers				
RTCCON	Control Register	$F1CC_H$	$E6_H$	0003_H
T14REL	Prescaler Timer Reload	$F0D0_H$	68_H	0000_H
T14	Prescaler Timer T14	$F0D2_H$	69_H	0000_H
RTCL	Count Register Low Word	$F0D4_H$	$6A_H$	0000_H
RTCH	Count Register High Word	$F0D6_H$	$6B_H$	0000_H
RTCRELL	Reload Register Low Word	$F0CC_H$	66_H	0000_H
RTCRELH	Reload Register High Word	$F0CE_H$	67_H	0000_H
RTCISNC	RTC Interrupt Subnode Control (1st Level)	$F1C8_H$	$E4_H$	0000_H
ISNC	RTC Interrupt Subnode Control (2nd Level)	$F1DE_H$	EF_H	0000_H

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
General Purpose Timer Registers (GPT1/2)				
T2CON	GPT1 Timer 2 Control Register	$FF40_H$	$A0_H$	0000_H
T3CON	GPT1 Timer 3 Control Register	$FF42_H$	$A1_H$	0000_H
T4CON	GPT1 Timer 4 Control Register	$FF44_H$	$A2_H$	0000_H
T5CON	GPT2 Timer 5 Control Register	$FF46_H$	$A3_H$	0000_H
T6CON	GPT2 Timer 6 Control Register	$FF48_H$	$A4_H$	0000_H
T2	GPT1 Timer 2 Register	$FE40_H$	20_H	0000_H
T3	GPT1 Timer 3 Register	$FE42_H$	21_H	0000_H
T4	GPT1 Timer 4 Register	$FE44_H$	22_H	0000_H
T5	GPT2 Timer 5 Register	$FE46_H$	23_H	0000_H
T6	GPT6 Timer 2 Register	$FE48_H$	24_H	0000_H
CAPREL	GPT1 Capture Reload Register	$FE4A_H$	25_H	0000_H
ADC Registers				
ADDAT1	ADC Data Register for Channel 1 and 2	$FEA0_H$	50_H	0000_H
ADDAT2	ADC Data Register for Channel 3 and 4	$FEA2_H$	51_H	0000_H
ADCCON	ADC Control Register	$FEA4_H$	52_H	0000_H
Interrupt Control Registers				
T2IC	Timer 2 Interrupt Control Register	$FF60_H$	$B0_H$	0000_H
T3IC	Timer 3 Interrupt Control Register	$FF62_H$	$B1_H$	0000_H
T4IC	Timer 4 Interrupt Control Register	$FF64_H$	$B2_H$	0000_H
T5IC	Timer 5 Interrupt Control Register	$FF66_H$	$B3_H$	0000_H
T6IC	Timer 6 Interrupt Control Register	$FF68_H$	$B4_H$	0000_H
CRIC	CAPREL Interrupt Control Register	$FF6A_H$	$B5_H$	0000_H
EX0IC	External Interrupt Control Register 0	$FF88_H$	$C4_H$	0000_H
EX1IC	External Interrupt Control Register 1	$FF8A_H$	$C5_H$	0000_H
EX2IC	External Interrupt Control Register 2	$FF8C_H$	$C6_H$	0000_H
EX3IC	External Interrupt Control Register 3	$FF8E_H$	$C7_H$	0000_H
EX4IC	External Interrupt Control Register 4	$FF90_H$	$C8_H$	0000_H
EX5IC	External Interrupt Control Register 5	$FF92_H$	$C9_H$	0000_H
EX6IC	External Interrupt Control Register 6	$FF94_H$	CA_H	0000_H

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
EX7IC	External Interrupt Control Register 7	$FF96_H$	CB_H	0000_H
ADC1IC	A/D Conversion Interrupt Control (Channel 1 + 2)	$FF98_H$	CC_H	0000_H
ADC2IC	A/D Conversion Interrupt Control (Channel 3 + 4)	$FF9A_H$	CD_H	0000_H
ADWIC	A/D Wake Up Interrupt	$F178_H$	BC_H	0000_H
SSCEIC	SSC Error Interrupt Control	$FF76_H$	BB_H	0000_H
SSCRIC	SSC Receive Interrupt Control	$FF74_H$	BA_H	0000_H
SSCTIC	SSC Transmit Interrupt Control	$FF72_H$	$B9_H$	0000_H
S0EIC	ASC Error Interrupt Control	$FF70_H$	$B8_H$	0000_H
S0RIC	ASC Receive Interrupt Control	$FF6E_H$	$B7_H$	0000_H
S0TIC	ASC Transmit Interrupt Control	$FF6C_H$	$B6_H$	0000_H
S0TBIC	ASC Transmit Buffer Interrupt Control	$F19C_H$	CE_H	0000_H
ABSTOIC	ASC Autobaud Stop Interrupt Control	$F17A_H$	BD_H	0000_H
ABSTAIC	ASC Autobaud Start Interrupt Control	$FF9E_H$	CF_H	0000_H
I^2CTIC	I^2C Transfer Interrupt Control	$F194_H$	CA_H	0000_H
I^2CPIC	I^2C Protocol Interrupt Control	$F18C_H$	$C6_H$	0000_H
I^2CTEIC	I^2C Transmission End Interrupt Control	$F184_H$	$C2_H$	0000_H
ACQIC	Acquisition-Interrupt Control	$F176_H$	BB_H	0000_H
VSDISIC	Display-Vertical-Sync Interrupt Control	$F174_H$	BA_H	0000_H
HSDISIC	Display-Horizontal-Sync Interrupt Control	$F172_H$	$B9_H$	0000_H
GAFIC	Graphic Accelerator Interrupt Control	$FF9C_H$	CE_H	0000_H
RTCIC	Realtime Clock Interrupt Control	$F19E_H$	CF_H	0000_H
PECCLIC	PEC Link Interrupt Control	$F180_H$	$C0_H$	0000_H
PEC Interrupt Control Registers				
PECC0	PEC Channel 0 Control Register	$FEC0_H$	60_H	0000_H
PECC1	PEC Channel 1 Control Register	$FEC2_H$	61_H	0000_H
PECC2	PEC Channel 2 Control Register	$FEC4_H$	62_H	0000_H
PECC3	PEC Channel 3 Control Register	$FEC6_H$	63_H	0000_H
PECC4	PEC Channel 4 Control Register	$FEC8_H$	64_H	0000_H

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
PECC5	PEC Channel 5 Control Register	$FECA_H$	65_H	0000_H
PECC6	PEC Channel 6 Control Register	$FECC_H$	66_H	0000_H
PECC7	PEC Channel 7 Control Register	$FECE_H$	67_H	0000_H
PECSN0	PEC Segment Number Channel 0 Register	$FED0_H$	68_H	0000_H
PECSN1	PEC Segment Number Channel 1 Register	$FED2_H$	69_H	0000_H
PECSN2	PEC Segment Number Channel 2 Register	$FED4_H$	$6A_H$	0000_H
PECSN3	PEC Segment Number Channel 3 Register	$FED6_H$	$6B_H$	0000_H
PECSN4	PEC Segment Number Channel 4 Register	$FED8_H$	$6C_H$	0000_H
PECSN5	PEC Segment Number Channel 5 Register	$FEDA_H$	$6D_H$	0000_H
PECSN6	PEC Segment Number Channel 6 Register	$FEDC_H$	$6E_H$	0000_H
PECSN7	PEC Segment Number Channel 7 Register	$FEDE_H$	$6F_H$	0000_H
CLISNC	PEC Channel Link Interrupt Subnode Register	$FFA8_H$	$D4_H$	0000_H
Port Registers				
RP0H	Reset Configuration at Port 4 (read only)	$F108_H$	85_H	XX_H
P2	Port 2 Register	$FFC0_H$	$E0_H$	0000_H
P3	Port 3 Register	$FFC4_H$	$E2_H$	0000_H
P4	Port 4 Register	$FFC8_H$	$E4_H$	0000_H
P5	Port 5 Register	$FFA2_H$	$D1_H$	0000_H
P6	Port 6 Register	$FFCC_H$	$E6_H$	0000_H
DP2	Direction Control Register Port 2	$FFC2_H$	$E1_H$	0000_H
DP3	Direction Control Register Port 3	$FFC6_H$	$E3_H$	0000_H
DP6	Direction Control Register Port 6	$FFCE_H$	$E7_H$	0000_H
ODP3	Open Drain Control Register Port 3	$F1C6_H$	$E3_H$	0000_H
ODP6	Open Drain Control Register Port 6	$F1CE_H$	$E7_H$	0000_H
P5BEN	Analog/Digital Input Enable Register Port 5	$F1C2_H$	$E1_H$	0000_H
ALTSEL0P6	Alternate Function Enable Register Port 6	$F12C_H$	96_H	0000_H
Specific Control Registers				
STRVBI	VBI Buffer Start Register	$F1A0_H$	$D0_H$	0000_H

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
PXDEL	Pixel Delay Register	$F198_H$	CC_H	0000_H
ACQISN	Acquisition Interrupt Subnode Register	$F1A2_H$	$D1_H$	0000_H
GPRGCRL	GAI Instruction Start Register (Low Word)	$F1A4_H$	$D2_H$	0000_H
GPRGCRH	GAI Instruction Start Register (High Word)	$F1A6_H$	$D3_H$	0000_H
DGCON	Display Generator Control	$FEFA_H$	$7D_H$	0000_H
SCR	Sync Control Register	$F1A8_H$	$D4_H$	0000_H
VLR	Vertical Line Register	$F1AA_H$	$D5_H$	0271_H
BVCR	Begin of Vertical Clamping	$FEF6_H$	$7B_H$	0001_H
EVCR	End of Vertical Clamping	$FEF8_H$	$7C_H$	0005_H
HPR	Horizontal Period Register	$F1AC_H$	$D6_H$	0855_H
SDV	Vertical Sync Delay Register	$F1AE_H$	$D7_H$	0020_H
SDH	Horizontal Sync Delay Register	$F1B0_H$	$D8_H$	0020_H
HCR	Horizontal Clamping Register	$F1B2_H$	$D9_H$	1400_H
PFR	Pixel Frequency Register	$F1B6_H$	DB_H	$00A4_H$
DACCON	RGB - DAC Control Register	$F1B4_H$	DA_H	0005_H
External Bus Interface Control Registers				
REDIR	Memory Map Redirection Register	$F1BA_H$	DD_H	0000_H
REDIR1	Memory Map Redirection Register 1	$F1CA_H$	$E5_H$	$00FF_H$
EBICON	Control Register for EBI	$F1BC_H$	DE_H	0000_H
EBIDIR	Direct Access Register for EBI	$F1BE_H$	DF_H	0000_H
OCDS Registers				
COMDATA	Communication Mode Data Register	$F068_H$	34_H	0000_H
RWDATA	Read/Write Mode Data Register	$F06A_H$	35_H	0000_H
IOSR	Communication Mode Status Register	$F06C_H$	36_H	0000_H
DCMPLL	Hardware Trigger Range Comparison Lower Bound	$F0DC_H$	$6E_H$	0000_H [1)]
DCMPLH	Extension of Register DCMPLL	$F0DE_H$	$6F_H$	0000_H [1)]
DCMPGL	Hardware Trigger Range Comparison Upper Bound	$F0E0_H$	70_H	0000_H [1)]
DCMPGH	Extension of Register DCMPGL	$F0E2_H$	71_H	0000_H [1)]

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
DCMP0L	Hardware Trigger Equal Comparison Register 0	$F0E4_H$	72_H	0000_H [1]
DCMP0H	Extension of Register DCMP0L	$F0E6_H$	73_H	0000_H [1]
DCMP1L	Hardware Trigger Equal Comparison Register 1	$F0E8_H$	74_H	0000_H [1]
DCMP1H	Extension of Register DCMP1L	$F0EA_H$	75_H	0000_H [1]
DCMP2L	Hardware Trigger Equal Comparison Register 2	$F0EC_H$	76_H	0000_H [1]
DCMP2H	Extension of Register DCMP2L	$F0EE_H$	77_H	0000_H [1]
DTREVT	Hardware Trigger Event Control	$F0F0_H$	78_H	0000_H [1]
DSWEVT	Software Trigger Event Control	$F0F4_H$	$7A_H$	0000_H [1]
DEXEVT	External Trigger Event Control	$F0F8_H$	$7C_H$	0000_H [1]
DBGSR	Debug Status Register	$F0FC_H$	$7E_H$	0000_H [1]
System & CPU Registers				
TFR	Trap Flag Register	$FFAC_H$	$D6_H$	0000_H
ADDRSEL1	Address Select Register 1	$FE18_H$	$0C_H$	0000_H
ADDRSEL2	Address Select Register 2	$FE1A_H$	$0D_H$	0000_H
ADDRSEL3	Address Select Register 3	$FE1C_H$	$0E_H$	0000_H
ADDRSEL4	Address Select Register 4	$FE1E_H$	$0F_H$	0000_H
BUSCON0	Bus Configuration Register 0	$FF0C_H$	86_H	0000_H
BUSCON1	Bus Configuration Register 1	$FF14_H$	$8A_H$	0000_H
BUSCON2	Bus Configuration Register 2	$FF16_H$	$8B_H$	0000_H
BUSCON3	Bus Configuration Register 3	$FF18_H$	$8C_H$	0000_H
BUSCON4	Bus Configuration Register 4	$FF1A_H$	$8D_H$	0000_H
SYSCON	CPU System Configuration Register	$FF12_H$	89_H	0400_H
SYSCON1	CPU System Configuration Register 1	$F1DC_H$	EE_H	0000_H
SYSCON2	CPU System Configuration Register 2	$F1D0_H$	$E8_H$	0000_H
XADRS1	External Address Select Register 1	$F014_H$	$0A_H$	0000_H
XADRS2	External Address Select Register 2	$F016_H$	$0B_H$	0000_H
XADRS3	External Address Select Register 3	$F018_H$	$0C_H$	0000_H

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
XADRS4	External Address Select Register 4	$F01A_H$	$0D_H$	0000_H
XADRS5	External Address Select Register 5	$F01C_H$	$0E_H$	0000_H
XADRS6	External Address Select Register 6	$F01E_H$	$0F_H$	0000_H
XBCON1	XBUS Control Register 1	$F114_H$	$8A_H$	0000_H
XBCON2	XBUS Control Register 2	$F116_H$	$8B_H$	0000_H
XBCON3	XBUS Control Register 3	$F118_H$	$8C_H$	0000_H
XBCON4	XBUS Control Register 4	$F11A_H$	$8D_H$	0000_H
XBCON5	XBUS Control Register 5	$F11C_H$	$8E_H$	0000_H
XBCON6	XBUS Control Register 6	$F11E_H$	$8F_H$	0000_H
DPP0	CPU Data Page Pointer 0 (10bits)	$FE00_H$	00_H	0000_H
DPP1	CPU Data Page Pointer 1 (10bits)	$FE02_H$	01_H	0001_H
DPP2	CPU Data Page Pointer 2 (10bits)	$FE04_H$	02_H	0002_H
DPP3	CPU Data Page Pointer 3 (10bits)	$FE06_H$	03_H	0003_H
MDH	CPU Multiply Divide Control Register High Word	$FE0C_H$	06_H	0000_H
MDL	CPU Multiply Divide Control Register Low Word	$FE0E_H$	07_H	0000_H
MDC	CPU Multiply Divide Control Register	$FF0E_H$	87_H	0000_H
PSW	CPU Program Status Word	$FF10_H$	88_H	0000_H
STKUN	CPU Stack Underflow Pointer Register	$FE16_H$	$0B_H$	0000_H
STKOV	CPU Stack Overflow Pointer Register	$FE14_H$	$0A_H$	0000_H
SP	CPU System Stack Pointer	$FE12_H$	09_H	$FC00_H$
CP	CPU Context Pointer	$FE10_H$	08_H	$FC00_H$
CSP	CPU Code Segment Control Register	$FE08_H$	04_H	0000_H
SCUSLC	Security Level Command Register	$F0C0_H$	60_H	0000_H
SCUSLS	Security Level Status Register	$F0C2_H$	61_H	0000_H
ZEROS	Constant Value 0's Register (read only)	$FF1C_H$	$8E_H$	0000_H
ONES	Constant Value 1's Register (read only)	$FF1E_H$	$8F_H$	$FFFF_H$
XPERCON	X-Peripheral Control Register	$F024_H$	12_H	0000_H
EXISEL	Alternative External Interrupt Selection	$F1DA_H$	ED_H	0000_H

Table 13-1 (cont'd)

Name	Description	Physical Address	8-Bit Address	Reset Value
EXICON	External Interrupt Control	$F1C0_H$	$E0_H$	0000_H
OSCCON	Oscillator Pad Control Register	$F1C4_H$	$E2_H$	0001_H
IDMANUF	Manufacture ID	$F07E_H$	$3F_H$	$XXXX_H$
IDCHIP	Chip ID	$F07C_H$	$3E_H$	$XXXX_H$
TM_LO	Hardware Testmode Register Low	$FEFC_H$	$7E_H$	0000_H
TM_HI	Hardware Testmode Register High	$FEFE_H$	$7F_H$	0000_H
FOCON	SCU Register (no Function within M2)	$FFAA_H$	$D5_H$	0000_H
SYSCON3	SCU Register (no Function within M2)	$F1D4_H$	EA_H	0000_H

1) OCDS related registers that are not reset during a controller reset.

2) No 8-bit addresses provided for XBUS registers.

13.4 Registers Ordered by Address

The following tables summarize the register symbols and their "short addresses". The physical address can be calculated by multiplying the short address by 2 and adding that value to $FE00_H$ for the SFR register area and adding $F000_H$ for the extended SFR area. Bit-addressable registers are highlighted in gray.

13.4.1 Registers in SFR Area

Address	+ 00_H	+ 01_H	+ 02_H	+ 03_H	+ 04_H	+ 05_H	+ 06_H	+ 07_H
00_H	DPP0	DPP1	DPP2	DPP3	CSP	reserved	MDH	MDL
08_H	CP	SP	STKOV	STKUN	ADDRSEL1	ADDRSEL2	ADDRSEL3	ADDRSEL4
10_H	-	-	-	-	-	-	-	-
18_H	-	-	-	-	-	-	-	-
20_H	T2	T3	T4	T5	T6	CAPREL	reserved	-
28_H	-	-	-	-	-	-	-	-
30_H	-	-	-	-	-	-	-	-
38_H	-	-	-	-	-	-	-	-
40_H	-	-	-	-	-	-	-	-
48_H	-	-	-	-	-	-	-	-
50_H	ADDAT1	ADDAT2	ADCCON	-	-	S0PMW	-	WDT
58_H	SOTBUF	SORBUF	S0BG	S0FDV	S0ABSTAT	-	-	-
60_H	PECC0	PECC1	PECC2	PECC3	PECC4	PECC5	PECC6	PECC7
68_H	PECSN0	PECSN1	PECSN2	PECSN3	PECSN4	PECSN5	PECSN6	PECSN7
70_H	-	-	-	-	-	-	-	-
78_H	-	-	-	BVCR	EVCR	DGCON	TM_LO	TM_HI
80_H	**-**	**-**	**-**	**-**	**-**	**-**	**BUSCON0**	**MDC**
88_H	**PSW**	**SYSCON**	**BUSCON1**	**BUSCON2**	**BUSCON3**	**BUSCON4**	**ZEROS**	**ONES**
90_H	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
98_H	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
$A0_H$	**T2CON**	**T3CON**	**T4CON**	**T5CON**	**T6CON**	**-**	**-**	**-**
$A8_H$	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
$B0_H$	**T2IC**	**T3IC**	**T4IC**	**T5IC**	**T6IC**	**CRIC**	**S0TIC**	**S0RIC**
$B8_H$	**S0EIC**	**SSCTIC**	**SSCRIC**	**SSCEIC**	**-**	**ABSTOIC**	**-**	**-**
$C0_H$	**-**	**-**	**-**	**-**	**CC0IC**	**CC1IC**	**CC2IC**	**CC3IC**
$C8_H$	**CC4IC**	**CC5IC**	**CC6IC**	**CC7IC**	**ADC1IC**	**ADC2IC**	**GAFIC**	**ABSTAIC**
$D0_H$	**-**	**P5**	**-**	**-**	**CLISNC**	**FOCON**	**TFR**	**WDTCON**
$D8_H$	**S0CON**	**SSCCON**	**-**	**-**	**S0ABCON**	**reserved**	**-**	**-**
$E0_H$	**P2**	**DP2**	**P3**	**DP3**	**P4**	**-**	**P6**	**DP6**
$E8_H$	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
$F0_H$	**R0**	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
$F8_H$	**R8**	**R9**	**R10**	**R11**	**R12**	**R13**	**R14**	**R15**

13.4.2 Registers in ESFR Area

Address	+ 00_H	+ 01_H	+ 02_H	+ 03_H	+ 04_H	+ 05_H	+ 06_H	+ 07_H
00_H	-	-	-	-	-	-	CPUID	-
08_H	-	-	XADRS1	XADRS2	XADRS3	XADRS4	XADRS5	XADRS6
10_H	-	-	XPERCON	-	-	-	-	-
18_H	-	-	-	-	-	-	reserved	reserved
20_H	reserved	reserved	reserved	-	-	-	-	-
28_H	-	-	-	-	-	-	-	-
30_H	-	-	-	-	COMDATA	RWDATA	IOSR	-
38_H	IDRT	IDSCU	-	IDMEM2	IDPROG	IDMEM	IDCHIP	IDMANUF
40_H	-	-	-	-	-	-	-	-
48_H	-	-	-	-	-	-	-	-
50_H	-	-	-	-	-	-	-	-
58_H	SSCTB	SSCRB	SSCBR	reserved	reserved	-	-	-
60_H	SCUSLC	SCUSLS	-	-	reserved	-	RTCRELL	RTCRELH
68_H	T14REL	T14	RTCL	RTCH	reserved	reserved	DCMPLL	DCMPLH
70_H	DCMPGL	DCMPGH	DCMP0L	DCMP0H	DCMP1L	DCMP1H	DCMP2L	DCMP2H
78_H	DTREVT	-	DSWEVT	-	DEXEVT	-	DBGSR	-
80_H	-	-	-	-	**RP0H**	-	-	-
88_H	-	-	**XBCON1**	**XBCON2**	**XBCON3**	**XBCON4**	**XBCON5**	**XBCON6**
90_H	-	-	-	-	-	-	ALTSEL0P6	-
98_H	-	-	-	-	-	-	-	-
$A0_H$	-	-	-	-	-	-	-	-
$A8_H$	-	-	-	-	-	-	-	-
$B0_H$	-	-	-	-	-	-	-	-
$B8_H$	-	**HSDISIC**	**VSDISIC**	**ACQIC**	**ADWIC**	-	-	-
$C0_H$	**PECCLIC**	-	**I^2CTEIC**	-	-	-	**I^2CPIC**	-
$C8_H$	-	-	**I^2CTIC**	-	**PXDEL**	-	**S0TBIC**	**RTCIC**
$D0_H$	**STRVBI**	**ACQISN**	**GPRGCRL**	**GPRGCRH**	**SCR**	**VLR**	**HPR**	**SDV**
$D8_H$	**SDH**	**HCR**	**DACCON**	**PFR**	**reserved**	**REDIR**	**EBICON**	**EBIDIR**
$E0_H$	**EXICON**	**P5BEN**	**OSCCON**	**ODP3**	**RTCISNC**	**REDIR1**	**RTCCON**	**ODP6**
$E8_H$	**SYSCON2**	-	**SYSCON3**	-	-	**EXISEL**	**SYSCON1**	**ISNC**
$F0_H$	**R0**	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
$F8_H$	**R8**	**R9**	**R10**	**R11**	**R12**	**R13**	**R14**	**R15**

Elelctrical Characteristics

14 Electrical Characteristics

14.1 Absolute Maximum Ratings

Note: The maximum ratings may not be exceeded under any circumstances, not even momentarily and individually, as permanent damage to the IC will result.

Table 14-1 Ambient Temperature
T_A = 0 °C … + 70 °C

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Supply voltage 3.3 V	$V_{DD33\ 1\text{-}7}$	–	4.0	V	–
Supply voltage 2.5 V	$V_{DD25\ 1\text{-}2}$	–	3.0	V	–
Analog supply voltage	$V_{DDA\ 1\text{-}4}$	–	3.0	V	–
Storage temperature	T_{stg}	– 20	125	°C	–
Electrostatic discharge	–	2000	–	V	100 pF, 1.5 kΩ HBM

14.2 Operating Range

Table 14-2 Operating Range

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Ambient temperature	T_A	0	70	°C	–
Supply voltage 3.3 V	$V_{DD33\ 1\text{-}7}$	3.1	3.6	V	–
Supply voltage 2.5 V	$V_{DD25\ 1\text{-}2}$	2.25	2.75	V	–
Analog supply voltage	$V_{DDA\ 1\text{-}4}$	2.25	2.75	V	–
Total Power Consumption	P_{total}	–	1.5	W	–

Note: In the operating range, the functions given in the circuit description are fulfilled.

14.3 DC Characteristics

Table 14-3 DC Characteristics

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Supply Currents					
Digital supply current for 3.3 V domain	$I_{3.3\ V}$	–	150	mA	all ports as inputs, f_{pixel} = 50 MHz, 100 MHz bus configuration
Digital supply current for 2.5 V domain	$I_{2.5\ V}$	–	150	mA	f_{pixel} = 50 MHz, 100 MHz bus configuration
Analog Power Supply Current	I_{ANA}	–	100	mA	–
Idle mode supply current (with A/D wake up, RTC and External Interrupts in active state)	I_{IDLE}	–	12	mA	Analog and digital supply
Sleep mode supply current (RTC running)	I_{SLEEP}	–	1.2	mA	Analog and digital supply
Power down mode supply current (RTC disabled)	I_{PWDN}	–	0.5	mA	Analog and digital supply
I/O Voltages (valid for any pin unless otherwise stated)					
Input low voltage	V_{IL}	– 0.4	0.8	V	–
Input high voltage	V_{IH}	2.0	3.6	V	–
Output low voltage	V_{IL}	–	0.45	V	–
Output high voltage	V_{IH}	2.5	–	V	–
Leakage current	I_{IL}	–	0.2	µA	@ 0.5 V < V_{in} < (V_{IHNOM} – 0.5 V)
Output Current	I_O	–	8	mA	–
Crystal Oscillator: XTAL1(input), XTAL2(output)					
Amplifier Transconductance	–	–	4.2	mS	–
Oscillation Frequency	C_{FB}	6.0 – 50 ppm	6.0 + 50 ppm	MHz	–

Table 14-3 DC Characteristics (cont'd)

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Duty Cycle	–	45	55	%	–
High time	t_H	50	–	ns	–
Pin capacitance (XTAL1)	C_I	–	3.5	pF	–
CVBS-Input: CVBS1A (ADC_DIFF = 0; differential CVBS Input)					
Pin capacitance	C_P	–	–	pF	–
Input impedance	Z_P	–	–	1/MΩ	–
Ext. coupling capacitance	$C_{CPL1,2}$	10	100	nF	–
Source impedance	–	–	< 500	Ω	–
Overall CVBS amplitude	V_{CVBS}	0.75	1.3	V	–
CVBS sync amplitude	V_{SYNC}	0.18	0.6	V	–
TXT data amplitude	V_{DATA}	0.3	0.7	V	–
De-coupling Capacitors to V_{DDA} at Pins CVBSi	C_{Dec_CPL}	–	–	nF	–
CVBS-Input: CVBS1A (ADC_DIFF = 1; non-differential CVBS Input)					
Pin capacitance	C_P	–	–	pF	–
Input impedance	Z_P	–	–	1/MΩ	–
Ext. coupling capacitance	C_{CPL1}	10	100	nF	–
Source impedance		–	< 500	Ω	–
Overall CVBS amplitude	V_{CVBS}	0.75	1.3	V	–
CVBS sync amplitude	V_{SYNC}	0.18	0.6	V	–
TXT data amplitude	V_{DATA}	0.3	0.7	V	–
De-coupling Capacitors to V_{DDA} at Pins CVBSi	C_{Dec_CPL}	–	–	nF	–
CVBS-Input: CVBS2					
Pin capacitance	C_P	–	–	pF	–
Input impedance	Z_P	–	–	1/MΩ	–
Ext. coupling capacitance	C_{CPL}	10	100	nF	–
Source impedance	–	–	< 500	Ω	–
Overall CVBS amplitude	V_{CVBS}	0.75	1.3	V	–
CVBS sync amplitude	V_{SYNC}	0.18	0.6	V	–

Table 14-3 DC Characteristics (cont'd)

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
TXT data amplitude	V_{DATA}	0.3	0.7	V	–
De-coupling Capacitors to V_{DDA} at Pins CVBSi	C_{Dec_CPL}	–	–	nF	–
RGB-Outputs					
Load capacitance	C_P	–	20	pF	–
Output voltage swing	V_{outpp}	0.5	1.2	V	available: 0.5 V; 0.7 V; 1.0 V; 1.2 V
RGB offset	U_{offset}	0.27	0.33	V	–
Rise/Fall Times	t_{RF}	8.0	12.5	ns	8.0 ns (50 MHz output BW) 12.5 ns (32 MHz output BW)
Load resistance	R_L	10	–	kΩ	–
Diff. non-linearity	–	– 0.5	0.5	LSB	–
Int. non-linearity	–	– 0.5	0.5	LSB	–
Output current tracking	–	–	3	%	–
Skew to COR, Blank	t_{skew}	– 5	5	ns	–
Jitter to Horizontal Sync Reference	t_{Jit}	–	4	ns	–
Address Bits: A0 to A9, $\overline{RD}$, $\overline{CSROM}$					
Output Rise Time	t_r	–	16	ns	(10% - 90%)
Output Fall Time	t_f	–	16	ns	(10% - 90%)
Load Capacitance	C_L	–	35	pF	–
Address Bits: A10 to A15					
Output Rise Time	t_r	–	6	ns	(10% - 90%)
Output Fall Time	t_f	–	6	ns	(10% - 90%)
Load Capacitance	C_L	–	35	pF	–
P4.(0 .. 5)					
Output Rise Time	t_r	–	16	ns	(10% - 90%)
Output Fall Time	t_f	–	16	ns	(10% - 90%)
Load Capacitance	C_L	–	35	pF	–

Table 14-3 DC Characteristics (cont'd)

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Data Bits: D0 to D15					
Output Rise Time	t_r	–	6	ns	(10% - 90%)
Output Fall Time	t_f	–	6	ns	(10% - 90%)
Load Capacitance	C_L	–	35	pF	–
Pin capacitance	C_I	–	5	pF	–
$\overline{WR}$, $\overline{CSSDRAM}$, CLKEN, LDQM, UDQM					
Output Rise Time	t_r	–	6	ns	(10% - 90%)
Output Fall Time	t_f	–	6	ns	(10% - 90%)
Load Capacitance	C_L	–	35	pF	–
Pin capacitance	C_I	–	5	pF	–
MEMCLK					
Output Rise Time	t_r	–	2	ns	(10% - 90%)
Output Fall Time	t_f	–	2	ns	(10% - 90%)
Load Capacitance	C_L	–	20	pF	–
BLANK/CORBLA					
Output Rise Time	t_r	8	12.5	ns	(10% - 90%)
Output Fall Time	t_f	8	12.5	ns	(10% - 90%)
Load Capacitance	C_L	–	20	pF	–
BLANK/CORBLA (Control bit CORBL = 0; BLANK only)					
Output voltage no data insertion (Video)	$V_{i\text{-}n}$	0	0.5	V	–
Output voltage for data insertion	$V_{i\text{-}y}$	0.9	–	V	–
BLANK/CORBLA (Control bit CORBL = 1; BLANK and COR)					
Output voltage no data insertion no contrast reduction	$V_{ic\text{-}n}$	0	0.5	V	–
Output voltage for contrast reduction and no data insertion	$V_{c\text{-}y}$	0.9	1.2	V	–

Table 14-3 DC Characteristics (cont'd)

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Output voltage for data insertion	$V_{i\text{-}y}$	1.8	–	V	–
HSYNC					
Input Rise Time	t_r	–	100	ns	(10% - 90%)
Input Fall Time	t_f	–	100	ns	(10% - 90%)
Input Hysteresis	V_{HYST}	300	600	mV	–
Input Pulse Width	T_{IPWH}	100	–	ns	–
Output Pulse Width	T_{OPWH}	1		us	–
Output Rise Time	t_r	–	100	ns	(10% - 90%)
Output Fall Time	t_f	–	100	ns	(10% - 90%)
Load Capacitance	C_L	–	50	pF	–
Pin capacitance	C_I	–	5	pF	–
VSYNC					
Input Rise Time	t_r	–	200	ns	(10% - 90%)
Input Fall Time	t_f	–	200	ns	(10% - 90%)
Input Hysteresis	V_{HYST}	300	600	mV	–
Input Pulse Width	T_{IPWV}	2/fh	–	–	–
Output Pulse Width	T_{IPWV}	$1/f_H$		–	Depends on Register HPR
Output Rise Time	t_r	–	100	ns	(10% - 90%)
Output Fall Time	t_f	–	100	ns	(10% - 90%)
Load Capacitance	C_L	–	50	pF	–
Pin capacitance	C_I	–	5	pF	–
VCS Timing (Master mode)					
Pulse width of H-Sync	t_{HPVCS}	4.59		µs	–
Distance between Equalizing Impulses	t_{DEP}	31.98		µs	–
Pulse Width of Equalizing Impulses	t_{EP}	2.31		µs	–
Pulse Width of Field Sync Impulses	t_{FSP}	27.39		µs	–

Table 14-3 DC Characteristics (cont'd)

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Horizontal Period	t_{HPR}	–		µs	Depends on Register HPR
P2.x, P3.x, P5.x, P6.x					
Output Rise Time	t_r	–	25	ns	(10% - 90%)
Output Fall Time	t_f	–	25	ns	(10% - 90%)
Load Capacitance	C_L	–	100	pF	–
Pin capacitance	C_I	–	10	pF	–
Input Impedance (Analog Ports)	Z_P	–	–	1/MΩ	–
Input Sample Frequency (General Purpose Ports)	F_S	–	33	MHz	–
Output Current (P3.10, P3.14)	I_o	–	8	mA	–
Hysteresis Voltage (I^2C Inputs): P6.5, P6.6, P6.7, P3.0, P3.1)	U_{HSYT}	–	100	mV	–
A/D Converter Characteristics (Port 5.0 to P5.3)					
Input Voltage Range	V_{ain}	0	2.5	V	–
ADC Resolution	RES	8		BIT	binary
Output during Underflow	–	0		–	–
Output during Overflow	–	255		–	–
Bandwidth	–	10.5		kHz	–
Sampling Time	t_S	2		µs	–
Sampling Frequency	f_{SAM}	21	–	kHz	–
Input Source Resistance	R_S	–	100	kΩ	–
Pin capacitance (Analog Ports)	C_P	–	40	pF	–
Reset $\overline{\text{RSTIN}}$					
Pin capacitance	C_I	–	5	pF	–
Reset In Pull Up Resistor	R_{pullup}	47	100	kΩ	–
Input High Voltage	U_{IH}	2	–	V	–

14.4 Timings

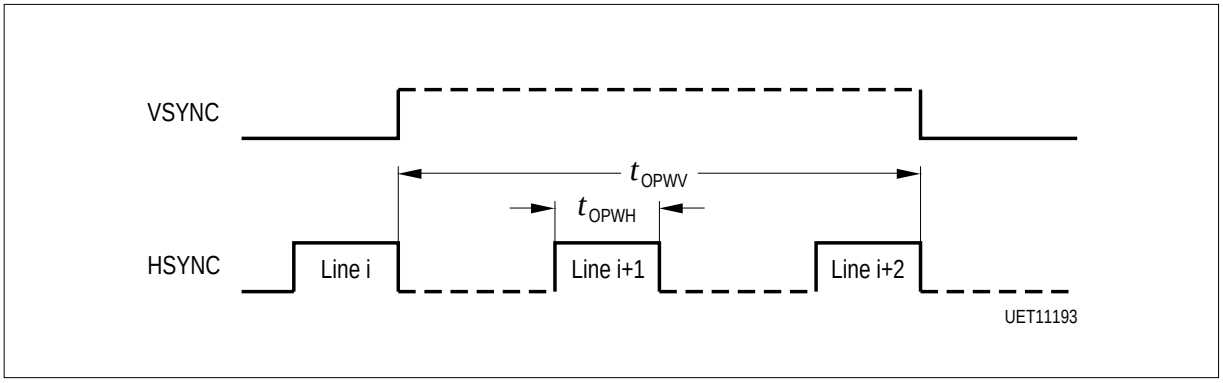


Figure 14-1 H/V - Sync-Timing (Sync-master mode)

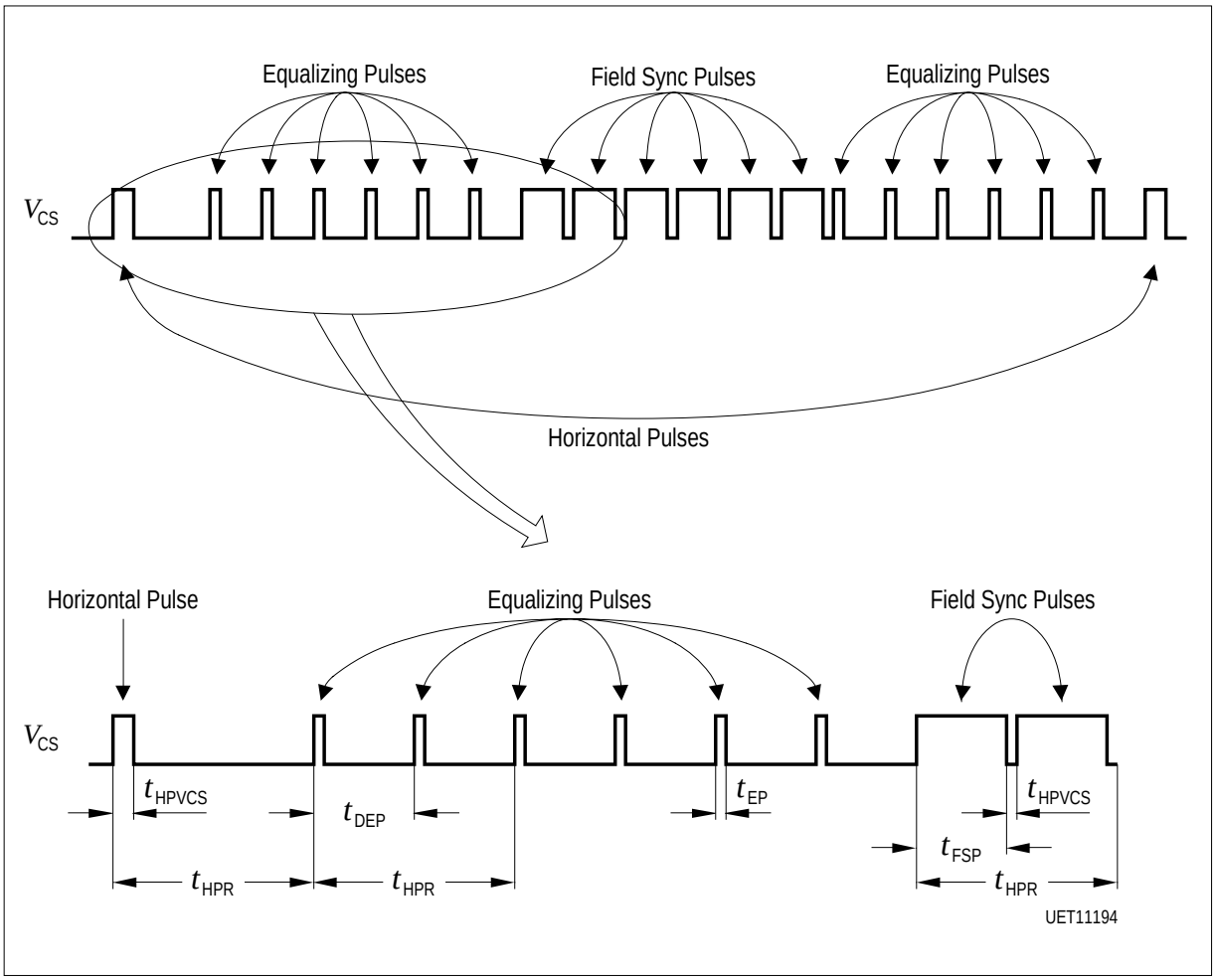
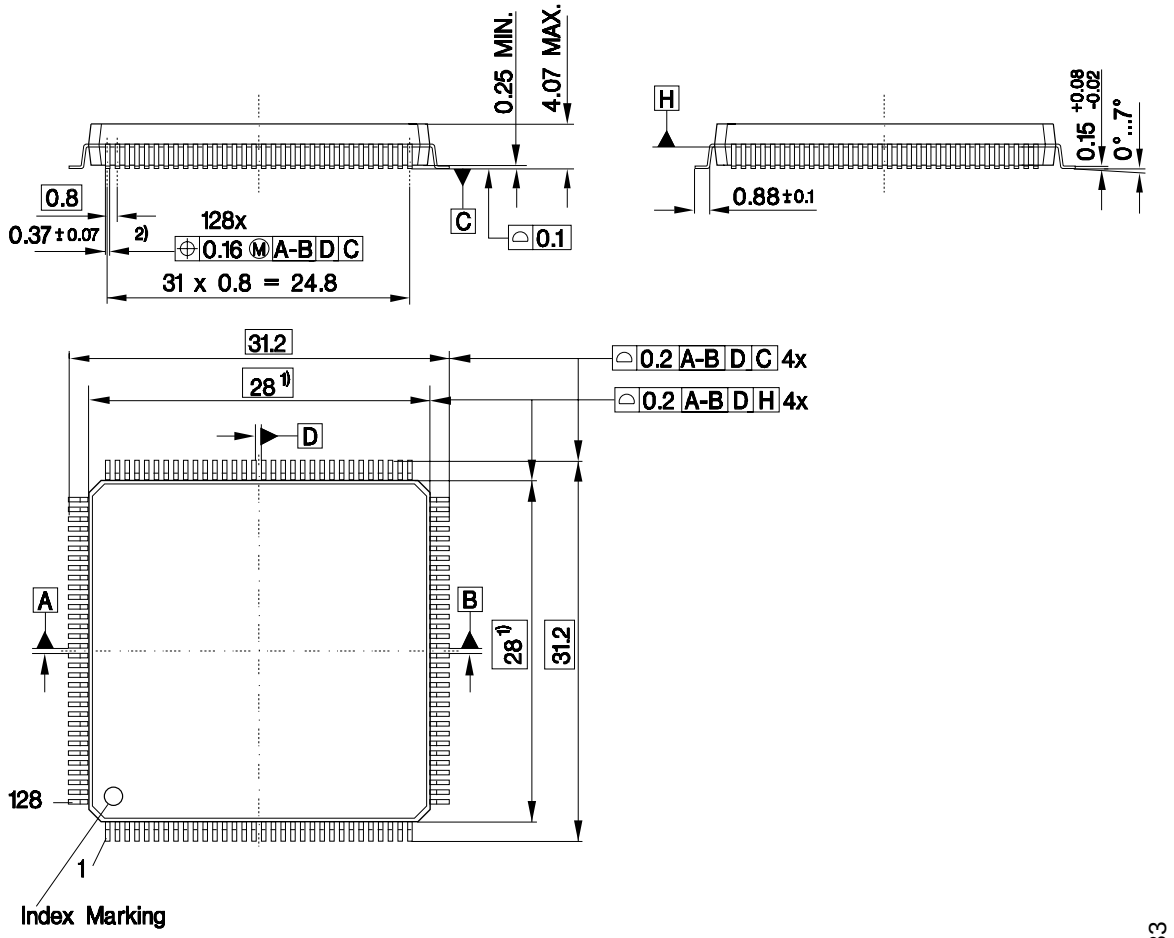


Figure 14-2 VCS -Timing (Sync-master mode)

14.5 Package Outlines

P-MQFP-128-2
(Plastic Metric Quad Flat Package)



2) Does not include dambar protrusion of 0.08 max. per side

1) Does not include plastic or metal protrusion of 0.25 max. per side

Sorts of Packing

Package outlines for tubes, trays etc. are contained in our Data Book “Package Information”.

SMD = Surface Mounted Device

Dimensions in mm