71M6521D/71M6521F
71M6521B
Energy Meter IC Family

SOFTWARE USER'S GUIDE

8/6/2008

Revision 1.7

TERIDIAN Semiconductor Corporation

6440 Oak Canyon Rd., Suite 100

Irvine, CA 92618-5201

Ph: (714) 508-8800 ▪ Fax: (714) 508-8878

Meter.support@teridian.com

http://www.teridian.com/

71M652X

Energy Meter IC FAMILY

SOFTWARE USER'S GUIDE

Table of Contents

List of Figures

List of Tables

LIMITED USE LICENSE AGREEMENT

Acceptance: By using the Application Programming Interface and / or other software described in this document ("Licensed Software") and provided by TERIDIAN Semiconductor Corporation ("TSC"), the recipient of the software ("Licensee") accepts, and agrees to be bound by the terms and conditions hereof.

Acknowledgment: The Licensed Software has been developed for use specifically and exclusively in conjunction with TSC's meter products: 71M6521D, 71M6521F, and 71M6521B. Licensee acknowledges that the Licensed Software was not designed for use with, nor has it been checked for performance with, any other devices.

Title: Title to the Licensed Software and related documentation remains with TSC and its licensors. Nothing contained in this Agreement shall be construed as transferring any right, title, or interest in the Licensed Software to Licensee except as expressly set forth herein. TSC expressly disclaims liability for any patent infringement claims based upon use of the Licensed Software either solely or in conjunction with third party software or hardware.

Licensee shall not make nor to permit the making of copies of the Licensed Software (including its documentation) except as authorized by this License Agreement or otherwise authorized in writing by TSC. Licensee further agrees not to engage in, nor to permit the recompilation, disassembly, or other reverse engineering of the Licensed Software.

License Grant: TSC grants Licensee a limited, non-exclusive, non-sub licensable, non-assignable and non-transferable license to use the software solely in conjunction with the meter devices manufactured and sold by TSC.

Non-disclosure and confidentiality: For the purpose of this Agreement, "Confidential Information" shall mean the Licensed Software and related documentation and information received by Licensee from TSC. All Confidential Information shall be maintained in confidence by Licensee and shall not be disclosed to any third party and shall be protected with the same degree of care as the Licensee normally uses in the protection of its own confidential information, but in no case with any less degree than reasonable care. Licensee further agrees not to use any Confidential Information received from TSC except as contemplated by the license granted herein.

Disclaimer of Warranty: TSC makes no representations or warranties, express or implied, regarding the Licensed Software, including any implied warranty of title, no infringement, merchantability, or fitness for a particular purpose, regardless of whether TSC knows or has reason to know Licensee's particular needs. TSC does not warrant that the functions of the Licensed Software will be free from error or will meet Licensee's requirements. TSC shall have no responsibility or liability for errors or product malfunction resulting from Licensee's use and/or modification of the Licensed Software.

Limitation of Damages/Liability: IN NO EVENT WILL TSC NOR ITS VENDORS OR AGENTS BE LIABLE TO LICENSEE FOR INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH, OR ARISING OUT OF, THIS LICENSE AGREEMENT OR USE OF THE LICENSED SOFTWARE.

Export: Licensee shall adhere to the U.S. Export Administration Laws and Regulations ("EAR") and shall not export or re-export any technical data or products received from TSC or the direct product of such technical data to any proscribed country listed in the EAR unless properly authorized by the U.S. Government.

Termination: TSC shall have the right to terminate the license granted herein in the event Licensee fails to cure any material breach within thirty (30) days from receipt of notice from TSC. Upon termination, Licensee shall return or, at TSC's option certify destruction of, all copies of the Licensed Software in its possession.

Law: This Agreement shall be construed in accordance with the laws of the State of California. The Courts located in Orange County, CA shall have exclusive jurisdiction over any legal action between TSC and Licensee arising out of this License Agreement.

Integration: This License Agreement constitutes the entire agreement of the parties as to the subject matter hereof. No modification of the terms hereof shall be binding unless approved in writing by TSC.

1

1 INTRODUCTION

TERIDIAN Semiconductor Corporation's (TSC) 71M652X single chip Energy Meter Controllers are a family of Systems-on-Chip that supports all functionalities required to build a low-cost power meter. Demo Boards are available for each chip (71M6521DE/FE, 71M6521BE) to allow development of embedded application, in conjunction with an In-Circuit Emulator. Development of a 71M652X application can be started in either 80515 assembly language, or more favorably in C using the Demo Boards. TSC provides, along with the 71M652X Demo Boards, a development toolkit that includes a demonstration program ("Demo Code") written in ANSI C that controls all features present on the Demo Boards. This Demo Code includes functions to manage the low level 80515 core such as memory, clock, power modes, interrupts; and high level functions such as the LCD, Real Time Clock, Serial interfaces and I/Os. The use of Demo Code portions will help reduce development time dramatically, since they allow the developer to focus on developing the application without dealing with the low-level layer such as hardware control, timing, etc. This document describes the different software layers and how to use them.

The Demo Code should allow customers to evaluate various resources of the 652X ICs but should not be regarded as production code. The Demo Code and all its components, with the exception of the CE code, are only example code and the use of it is as is and without guarantees implied. Customers may use the Demo Code as starting point at any given released revision level but should keep themselves informed about subsequent revisions of the Demo Code. Demo Code revisions may not be directly compatible with previously released revisions and/or embedded software used by customers. Customers need to adapt the Demo Code or other example code supplied by TERIDIAN Application Engineering to their own code base, and in this context TERIDIAN Semiconductor can only provide indirect assistance and support.

This Software User's Guide provides information on the following separate subjects:

- General software architecture and minimum requirements (Design Guide)
- Memory model, programming, test tools (Design Reference)
- Demo code structure, flow-charts, data flow, functions (Demo Code Description)
- Installing and using the EEP, compiler, ICE (Tool Installation Guide)
- Understanding and using the 80515 micro controller (80515 Reference)

1.1 USING THIS DOCUMENT

The reader should have a basic familiarity with microprocessors, particularly the 80515 architecture, firmware, software development and power meter applications. Prior experience with, or knowledge of, the applicable ANSI and/or IEC standards will also be helpful.

This document presents the features included in the 71M652X Demo Boards in terms of software and some hardware. To get the most out of this document, the reader should also have available other 71M652X publications such as the 71M652X Demo Board User's Manual, respective data-sheets, errata list and application notes for additional details and recent developments.

1.2 RELATED DOCUMENTATION

Please refer to the following documents for further information:

- 71M6521 Demo Board User's Manual
- 71M6521DE/FE or 71M6521BE Data Sheet
- Signum Systems ADM-51 In-Circuit Emulator Manual
- Keil Compiler Manual (Version 7.5 or later)
- µVision2 (Version 2.20a or later) Manual

TERIDIAN's web site (http://www.teridian.com) should be frequently checked for updates, application notes and other helpful information.

Questions to TERIDIAN Applications Engineering can be directed via e-mail to the address:

- meter.support@teridian.com

1.3 COMPATIBILITY STATEMENT

Information presented in this manual applies to the following hardware and software revisions:

- 71M6521 Demo Code Revision 4.7a
- 71M6521 Demo Board D6521T12A1 (68-pin QFN) Revision 1.0 or later
- 71M6521 Demo Board D6521T4A8 (64-pin LQFP) Revision 8 or later
- Signum Systems Wemu51 Software 3.07.00 (2/14/2005) or later
- Signum Systems ADM51 firmware version 3 (2005/02/08) or later

The revision 4.7a of the Demo Board Code is the basis for all discussed sources, commands, register addresses and so forth. Known issues with this revision are disclosed within the code description, and workarounds or improvements are shown.

2

2 DESIGN GUIDE

This section provides designers with some basic guidance in developing power meter applications utilizing the TSC 71M652X devices. There are two types of applications that can be developed:

- Embedded application using the sources provided by TERIDIAN, or
- Embedded application using only customer generated functions.

2.1 HARDWARE REQUIREMENTS

The following are the minimum hardware requirements for developing custom programs:

- TERIDIAN 71M6521 Demo Board. This board interfaces with a PC via the RS232 serial interface (COM port).
- AC Adaptor (AC/DC output) or variable power supply.
- PC with 512MB RAM and 10GB hard drive, 1 COM port and 1 USB port, running either Windows 2000, or Windows ME or Windows XP.
- Signum Systems ADM-51 In-Circuit Emulator (for loading and debugging the embedded application) and its associated cables (not included in the demo kit). Signum references this device as ADM-51.

2.2 SOFTWARE REQUIREMENTS

The following are the minimum software requirements for embedded application programming:

- Keil Compiler version 7.5 or later.
- µVision2 version 3.05c (Note: this version comes with Keil Compiler version 7.5).
- Signum Systems software Wemu51 (comes with Signum Systems ADM-51 ICE hardware).

The following software tools/programs are included in the 71M652X development kit and should be present on the development PC:

- Demo Code with Command Line Interface (CLI) - Used to interface directly to metering functions and to the chip hardware.
- Source files
- Demo Code object file (hex file).

In order to generate and test software, the Keil compiler and the Signum in-circuit emulator (ICE) must be installed per the instructions in section 4. The include files and header files must also be present on the development PC. Typically, a design session consists of the following steps:

- Editing C source code using µVision2
- Compiling the source code using the Keil compiler
- Modifying the source code and recompiling until all compiler error messages are resolved
- Using the assembler and linker to generate executable code
- Downloading the executable code to the ICE
- Executing the code and watching its effects on the target

2.3 SOFTWARE ARCHITECTURE

The 71M652X software architecture is partitioned into three separate layers:

1. The lowest level is the device or hardware layer, i.e. the layer that directly communicates with the discrete functional blocks of the chip and the peripheral components ("hardware"), such as serial interfaces, AFE, LCD etc.
2. The second layer consists of buffers needed for some functions.
3. The third layer is the application layer. This layer is partially implemented by the Demo Code for evaluation purposes, but extensions and enhancements can be added by the application software developer to design suitable electronic power meter applications.

Figure 2-1: shows the partitions of each software component. As illustrated, there are many different designs an application can develop depending on its usage. Section 5 describes in more detail the functions within each component.

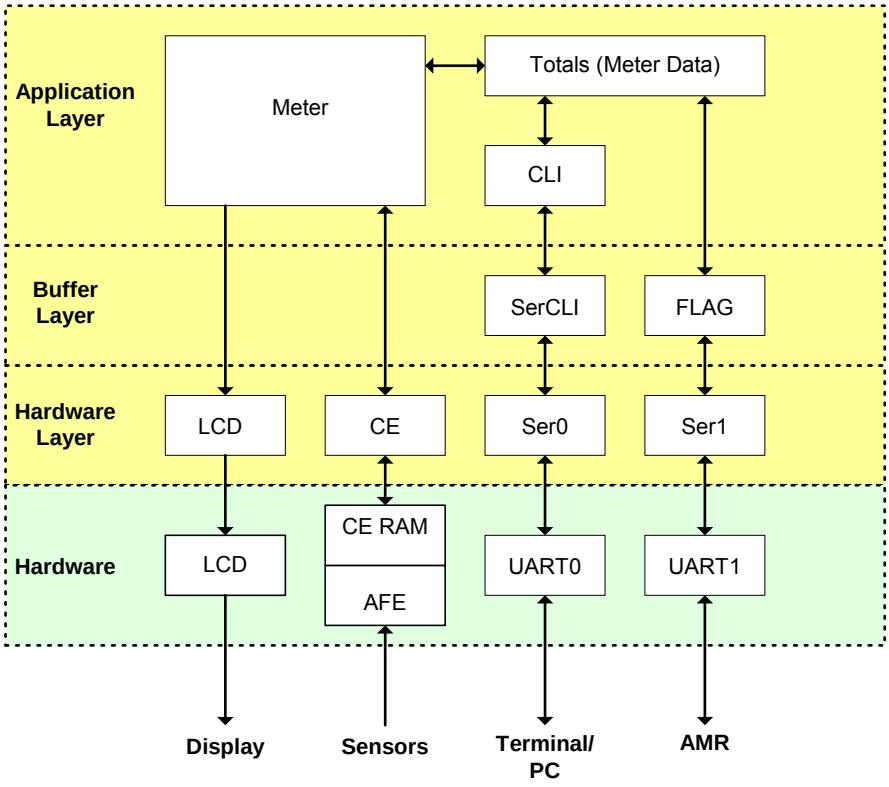


Figure 2-1: Software Structure

The Demo Code is highly modular. Each device in the chip and on the Demo Board has a corresponding set of driver software in the Hardware Layer. These driver software modules are very basic, enabling customers to easily locate and reuse the logic. For the serial devices and for the CE, the buffer handling has been abstracted and separated from the driver modules.

Where there are several similar devices (e.g. ser0, ser1, or tmr0, tmr1), the Demo Code simulates a virtual object base class using C preprocessor macros. For example, to initialize the first serial interace, ser0, the source file can include ser0.h, and then call ser_initialize(). To transmit a byte on ser0, the file can include ser0.h, and then call ser_xmit(). The convenience is that high-level code can be ported to another device by just (for example) including ser1.h, rather than ser0.h. Just by making variables static, entire high-level protocols can be written and maintained by copying the code debugged on one device, and having it include the other device's .h file.

The demo firmware uses this technique for the command line interface (ser0cli.c, ser1cli.c), the FLAG AMR interface (flag0.c, flag1.c) and for the software timer module (stm.c). The base-class emulation uses macros because on the 80515 MPU macros execute faster and are also more compact than the standard C++ (object-oriented) design with an implicit structure containing function pointers.

The Demo Code is also designed with an "options.h" file, which enables and disables entire features in the firmware.

The macro approach combined with the "options.h" file permitted the firmware team to adapt the same Demo Code to 8k, 16k, and 32k versions.

2.4 UTILITIES

Two utilities are offered that make it possible to perform certain operations on the object (HEX) files without having to use a compiler:

- D_MERGE.EXE allows combining the object file with a text script in order to change certain default settings of the program. For example, modified calibration coefficients resulting from an actual calibration can be inserted into the object file.
- CE_MERGE.EXE allows combining the object file with an updated image of the CE code.

Both utilities are executed from a DOS window (DOS command prompt). To invoke the DOS window, the "command prompt" option is selected after selecting Start – All Programs – Accessories.

The GUI subdirectory contains an unsupported MS Windows .NET implementation of a FLAG hand-held unit.

2.4.1 D_MERGE

Any changes to I/O RAM (Configuration RAM) can be made permanent by merging them into the object file. The first step for this is to create a maco file (macro.txt) containing the commands adjusting the I/O RAM, such as the following commands affecting calibration:

```
]8=+16381

]9=+16397

]E=+237
```

The d_merge program updates the 6521_demo.hex file with the values contained in the macro file. The d_merge program must be in the same directory as the source files, or a path to the executable must be declared. Executing the d_merge program with no arguments will display the syntax description. To merge the file macro.txt and the object file old_6521_demo.hex into the new object file new_6521_demo.hex, use the command:

```
d_merge old_6521_demo.hex macro.txt new_6521_demo.hex
```

2.4.2 CE_MERGE

The ce_merge program updates the 6521_demo.hex file with the CE program image contained in the CE.CE file and the data image CE.DAT. Both CE.CE and CE.DAT must be in Intel HEX format, i.e. both files are not in the source format but in the compiled format (Verilog HEX). These files will be made available from Teridian in the cases when updates to the CE images are necessary.
To merge the object file old_6521_demo.hex with CE.CE and CE.DAT into the new object file new_6521_demo.hex, use the command:

```
ce_merge old_6521_demo.hex ce.ce ce.dat 6521_demo.hex
```

3

3 DESIGN REFERENCE

As depicted in Figure 1 of section 2, the 71M652X provides a great deal of design flexibility for the application developer. Programming details are described below.

3.1 PROGRAM MEMORY

The embedded 80515 MPU within the 71M652X has separate program (32K, 16K, or 8K bytes) and data memory (2K bytes). The code for the Compute Engine program resides in the MPU program memory (flash).

The Flash program memory is addressed as a 64KB block, segmented in 512-byte pages. Selection of these individual blocks is accomplished using the function calls related to flash memory, which are described in more detail below.

3.2 DATA MEMORY

The 71M652X has 2K bytes of Data Memory for exclusive use of the embedded 80C515 MPU. In addition, there are 512 bytes for the Compute Engine. See Table 3-1: for a summary.

Address (hex)	Memory Technology	Memory Type	Typical Usage	Wait States (at 5MHz)	Memory Size (bytes)
0000-7FFF 0000-4FFF 0000-2FFF	Flash Memory	Non-volatile	Program and non-volatile data	0	32KB 16KB 8KB
0000-07FF	Static RAM	Volatile	MPU data XRAM,	0	2KB
1000-11FF	Static RAM	Volatile	CE data	6	512
2000-20FF	Static RAM	Volatile	Miscellaneous I/O RAM (configuration RAM)	0	256

Table 3-1: Memory Map

3.3 PROGRAMMING OF THE 71M652X CHIPS

There are two ways to download a hex file to the 71M652X Flash Memory:

- Using a Signum Systems ADM-51 ICE.
- Using the TERIDIAN Semiconductor Flash Programmer Module (FDBM) or the TERIDIAN Semiconductor Flash Download FDBM-TFP1 Stand-Alone Module

Note: For both programming and debugging code it is important that the hardware watchdog timer is disabled. See the Demo Board User's Manual for details.

Before downloading code to a 71M6521:

- **Stop the MPU**
- **Disable the CE by writing a 0 to XDATA at address 0x2000.**
- **Erase the flash memory.**

3.4 DEBUGGING OF THE 71M652X CHIPS

When debugging with the ADM51 in-circuit emulator, the CE continues to run, and this disables flash memory access because the code of the CE is located in flash memory.

When setting breakpoints, only two breakpoints can be used, because the first two breakpoints are "hardware" breakpoints, while the rest attempt to write to flash memory.

3.5 TEST TOOLS

A command line interface operated via the serial interface of the 71M652X MPU provides a test tool that can be used to exercise the functions provided by the low-level libraries. The command-line interface requires the following environment:

1) Demo Code (652X_demo.hex) must be resident in flash memory
2) The Demo Board is connected via a Debug Board to a PC running Hyperterminal or another type of terminal program.
3) The communication parameters are set at 300 bps, 7N2, XON/XOFF flow control, as described in section 3.5.1 .

3.5.1 Running the 652X_Demo.hex Program

This object file is the 71M652X embedded application developed by TERIDIAN to exercise all low-level function calls using a serial interface. Demo Boards ship pre-installed with this program. To run this program:

- Connect a serial cable between the serial port of the Debug Board RS232 and a COM port of a Windows PC.
- Open a Windows' Hyperterminal session at 9600 or 300 bps (depending on the setting of jumper connected to DIO_08 – see the 71M6521 Demo Board User's Manual), 8N1, one stop bit, with XON/XOFF flow control enabled. The setup dialog box is shown in Figure 3-1:
- Power on the Demo Board and hit <CR> a few times on the PC keyboard until '>' is displayed on the Hyperterminal screen.
- Type a command from the CLI Reference documented in the 71M6521 Demo Board User's Manual.
- All references to 'c' (lower case c) indicate any ASCII character, all other lowercase letters are one-byte numbers
- Numbers can be entered in decimal by preceding them with a plus-sign (e.g. hex 20 = +32)

The 71M6521 Demo Board User's Manual contains instructions on how to connect the serial cable.

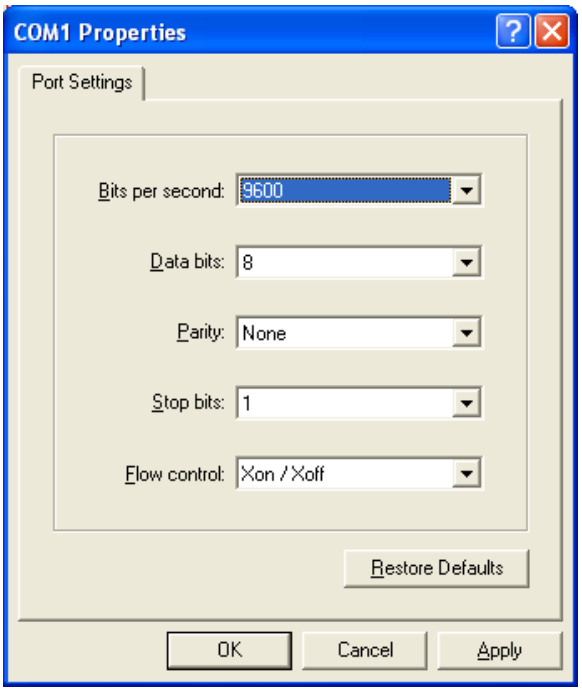


Figure 3-1: Port Speed and Handshake Setup

Note: HyperTerminal can be found by selecting Programs →Accessories → Communications from the Windows© start menu. The connection parameters are configured by selecting File → Properties and then by pressing the Configure button. Port speed and flow control are configured under the General tab, bit settings are configured by pressing the Configure button (Figure 3-1:), as shown below.

3.5.2 CLI Commands

The Demo Board User's Manual (DBUM) for the 71M6521 contains a complete list of the available commands.

Note: Only the 71M6521FE chip has enough memory to support a serial command line interface in addition to its metering functions. Communication with the 71M6521BE and 71M6521DE chips is implemented with a simpler interface, based on Intel hex records. This interface is also described in the Demo Board User's Manual.

3.5.3 Command (Macro) Files

Commands or series of commands may be stored in text (ASCII) files and sent to the 71M652X using the "Transfer – Send Text File" command of Hyperterminal or any other terminal program.

4

4 TOOL INSTALLATION GUIDE

This section provides detailed installation instructions for the Signum ADM-51 in-circuit emulator and for the Keil compiler.

4.1 INSTALLING THE PROGRAMS FOR THE ADM51 EMULATOR

The AMD51 ICE interfaces with the PC is via the USB serial interface.

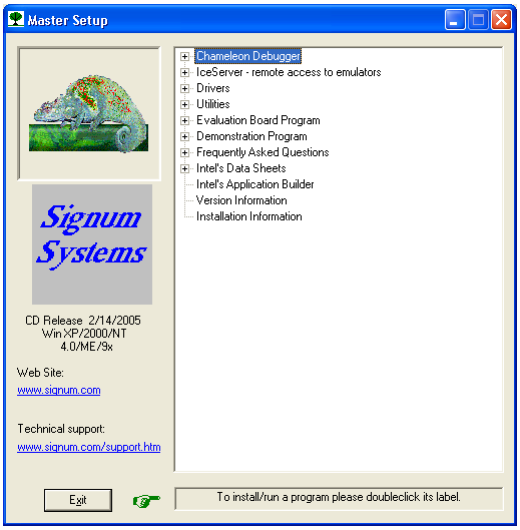
The installation process consists of the following steps:

1. Installing the Chameleon Debugger used with the Signum ICE
2. Installing the ADM51 USB driver
3. Installing updates
4. Creating a project

4.2 INSTALLING THE WEMU PROGRAM (CHAMELEON DEBUGGER)

Insert the CD from Signum Systems and connect the ICE ADM51 to the PC with the provided USB cable.

The following dialog box will appear (this dialog box also shows the release date of the program):



Click on "*Chameleon Debugger*" and then select "*ADM51 Emulator*".

Follow the instructions given by the installation program.

4.3 INSTALLING THE ADM51 USB DRIVER

The Wemu51 program communicates with the emulator ADM51 via the USB interface of the PC. The USB driver for the ADM51 has to be installed prior to using the emulator. After plugging in the USB cable into the PC and the ADM51 ICE the status light of the ADM51 emulator should come on.

A dialog box will appear, asking you to install the ADM51 driver.

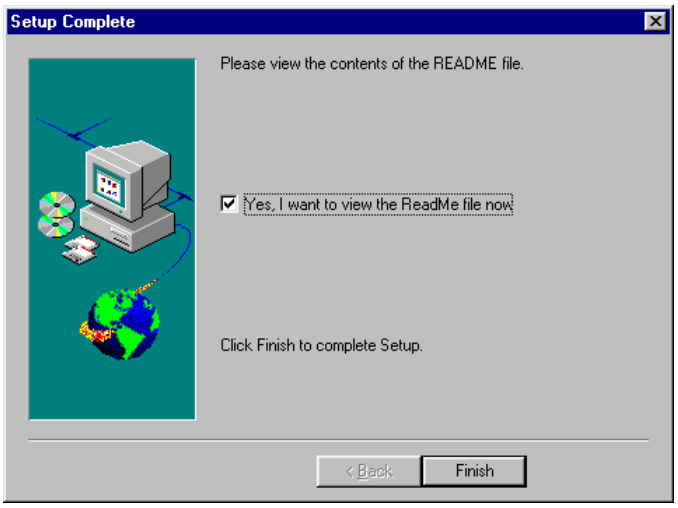
Click *Next*. Another dialog box will appear, asking how to search for the driver. Use the recommended method.

Click *Next*.

Another screen (not shown) will appear asking to locate the driver. Select *Specific Path* and browse to:

C:\Program Files\Signum Systems\Wemu51\Drivers\USB. Click Next.

Click *Finish*.



Click *Finish* again.

Note: USB 1.1 is sufficient for operation of the ADM51. If higher performance is desired and no USB 2.0 port is available on the host PC, a USB 2.0 card can be installed as an option.

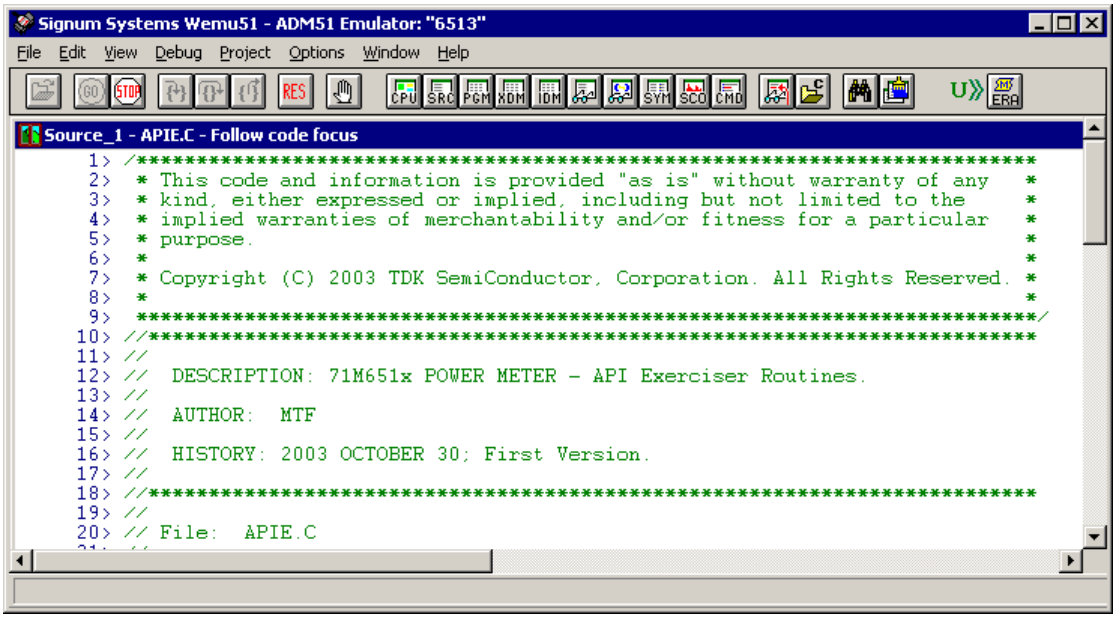
4.4 INSTALLING UPDATES TO THE EMULATOR PROGRAM AND HARDWARE

If the Wemu51 program is revision 3.07 or later, no special precautions have to be taken. Otherwise, the program should be updated using the Signum Systems web site (www.signum.com).

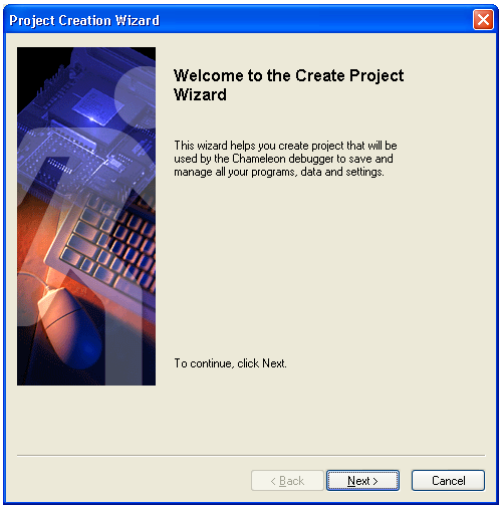
When running the Wemu51 program revision 3.07 or later, the firmware in the ADM51 will be checked automatically. ADM51 emulators with outdated firmware will not function properly. The Wemu51 will offer an automatic update for the ADM51, if necessary. For a successful upgrade it is vital to follow the instructions on screen precisely.

4.5 CREATING A PROJECT

Double click on the WEMU51 icon to start the Chameleon debugger.

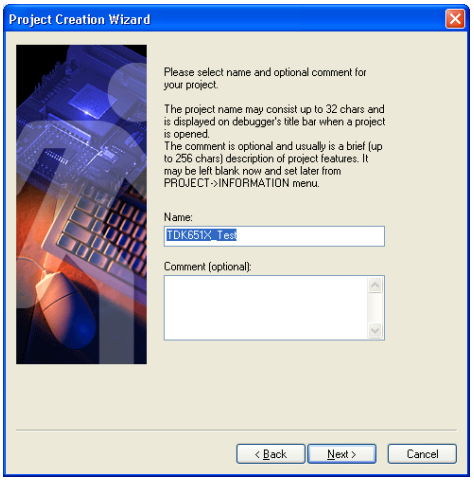


Click *Project/Create New Project*. The following screen will appear:

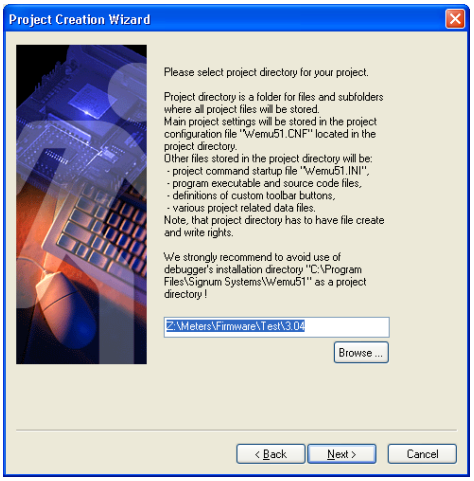


Follow the instructions of the Create Project Wizard by selecting *Next*.

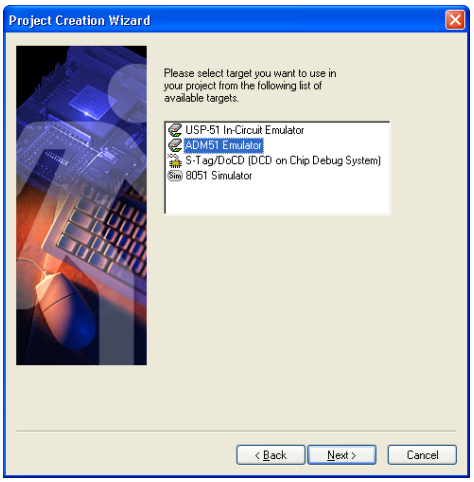
When prompted for the project name to be used, type a convenient project name. Click *Next*.



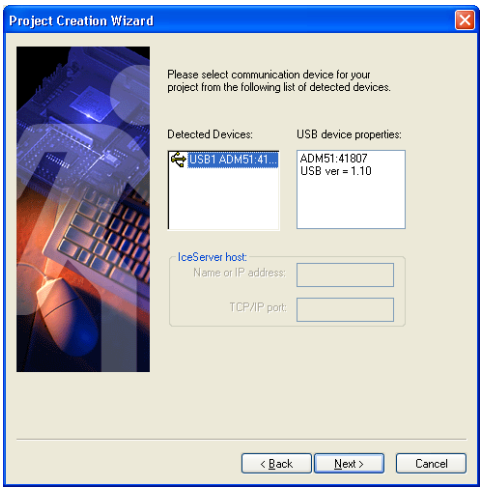
When prompted for the project directory to be used, select an existing folder on the PC. **Do NOT select any folder in the Wemu51 installation directory**! Click *Next*.



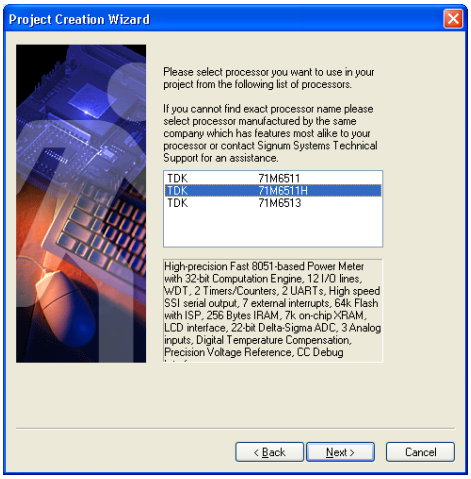
When prompted for the emulator to be used, select *ADM51 Emulator*. Click *Next*.



When prompted for the communication device to be used, select *USB ADM51*. Click *Next*.



When prompted for the processor to be used, select either *71M6521*. Click *Next*.



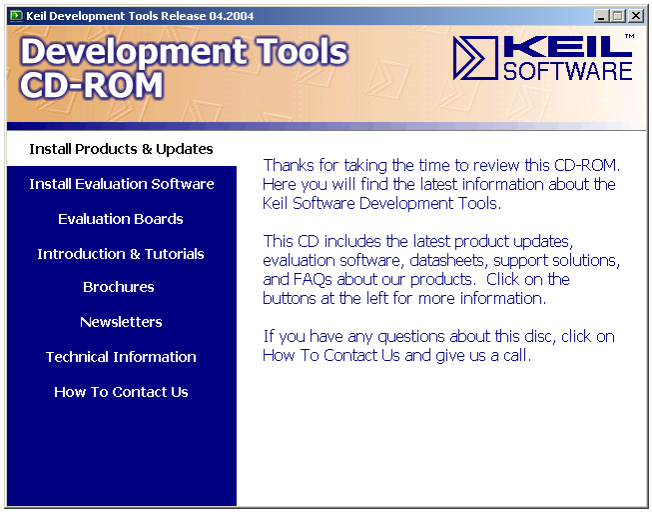
Click *Finish*.

4.6 INSTALLING THE KEIL COMPILER

After inserting the Keil CD-ROM into the CD drive of the PC, the on-screen instructions should be followed to install the Keil compiler.

Note: For PCs that can only use one type of drive at a time (CD-ROM drive, floppy drive, such as certain laptops), it is helpful to copy the contents of the floppy labeled "Add-On Disk" to the hard drive of the PC. That way, drives do not have to be swapped out during the installation.

The installer will display the following screen:



Select Install *Products & Updates*

Select C51 *Compiler and Tools*

Follow the on-screen instructions of the installation program. When prompted for the add-on disk, insert the disk in the floppy drive and click *Next* or browse to the location of the files (if they were previously copied to the hard drive of the PC) by clicking Browse.

4.7 CREATING A PROJECT FOR THE KEIL COMPILER

4.7.1 Directory Structure

The following directory structure is established when the files from the archive 652X_Demo.zip are unpacked while maintaining the structure of subdirectories:

<drive letter>:\…\meter project\

<drive letter>:\…\meter project\CE

<drive letter>:\…\meter project\CLI

<drive letter>:\…\meter project\CLI_652X

<drive letter>:\…\meter project\docs

<drive letter>:\…\meter project\flag

<drive letter>:\…\meter project\IO

<drive letter>:\…\meter project\Main_6521

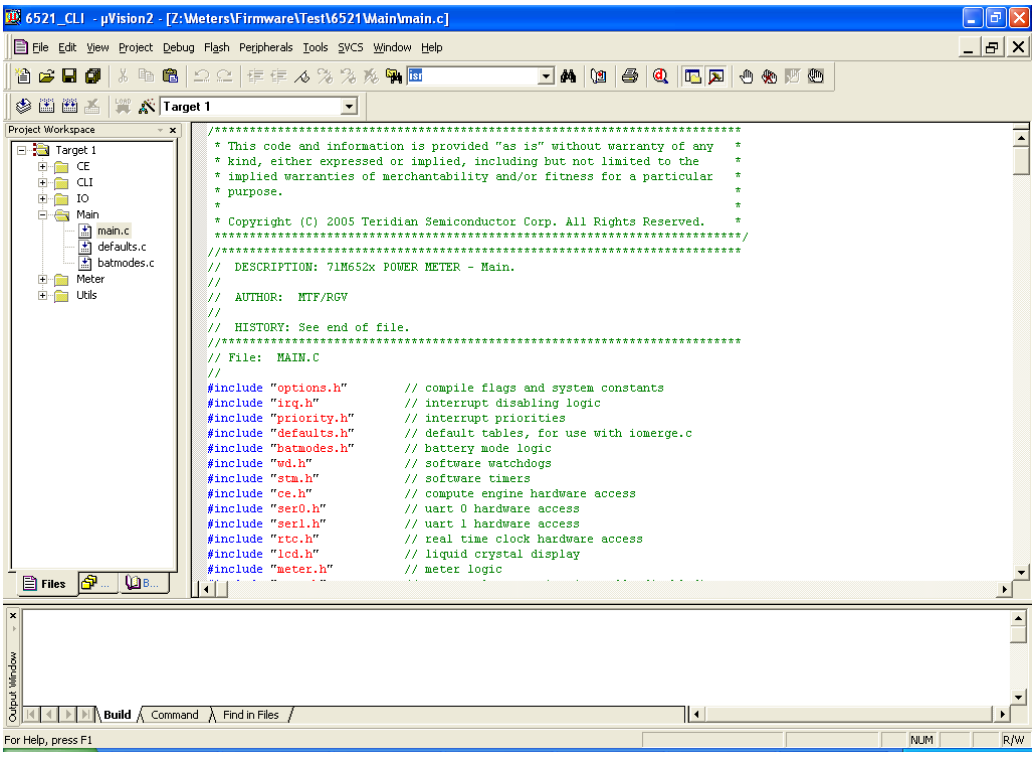
<drive letter>:\…\meter project\Main_6521_CLI

<drive letter>:\…\meter project\Meter

<drive letter>:\…\meter project\Util

The project control file 652X_demo.uv2 will be in the directory <drive letter>:\…\meter project. The Keil compiler can be configured easily by loading the file 652X_demo.uv2, using the *Project* Menu and selecting the *Open Project* command.

The window shown below should appear when the project control file is opened.



The Project Workspace screen on the left side of the window shows the main components of the source (CE, CLI, IO, Main, Meter, Utils) in folders. Folders can be opened by clicking on the plus sign next to them. Opening the folders will display the source files associated with them.

It should be noted that not all header files are physically present in the project directory. The files absacc.h, string.h, ctype.h, and setjmp.h are provided by the compiler manufacturer, and they are located in the Keil\C51\INC directory.

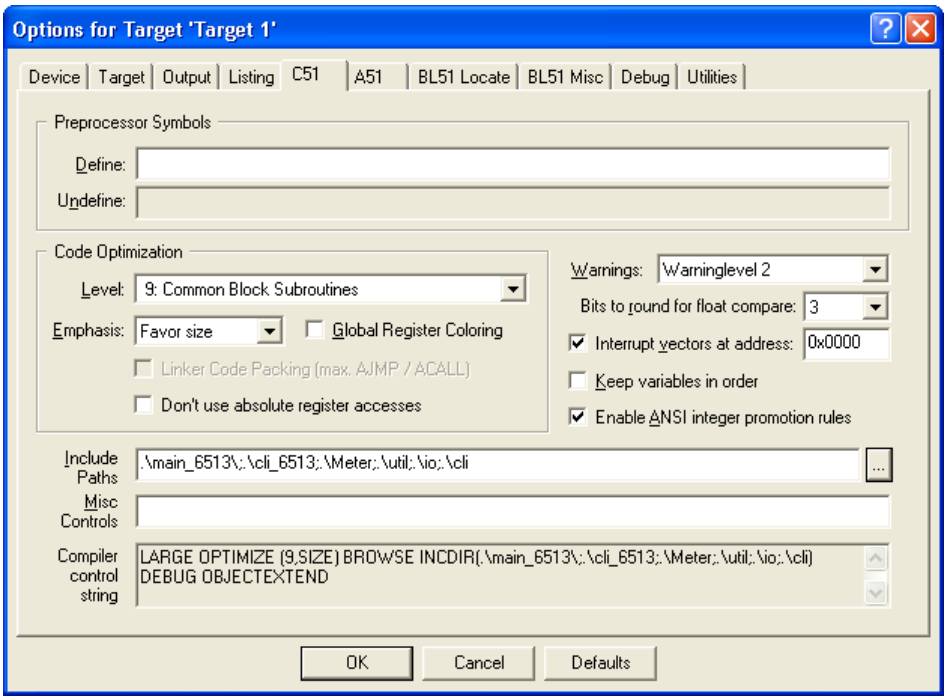
4.7.2 Adjusting the Keil Compiler Settings

Once, the Keil compiler is installed, the most convenient method to start the project is to double-click on the file 6521.UV2 (or 6521UV3). This will start the Keil compiler with the proper settings stored in the 6521.UV2 file.

Directory structures and drive names vary from PC to PC. The settings for the compiler can be adjusted using the following method:

1. Select "target1" in the leftmost window.
2. Select "project" from the top menu and then select "options for target 1".
3. Select the "C51" tab.
4. Click the button right next to the "Include Paths" window. Three paths will be listed, pointing to meter projects, meter projects\demo, and meter projects\demo\header files.
5. If necessary, delete these path entries (X button) and replace them with the corresponding path entries for your PC (□ button).

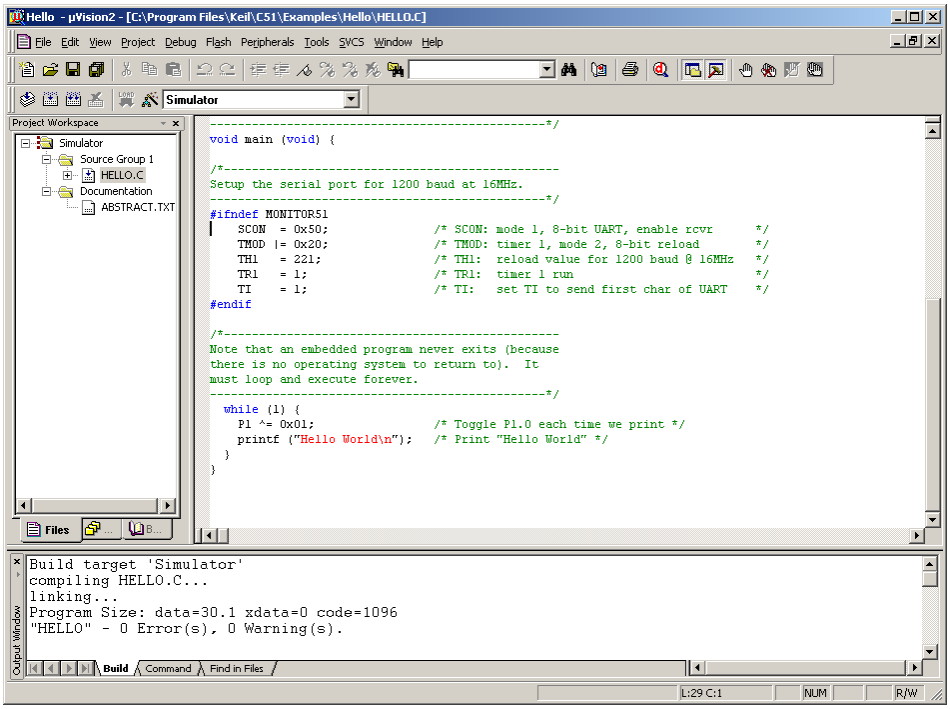
The dialog box should look like shown below. After making the necessary changes, the project file (652X_demo.UV2) should be stored.



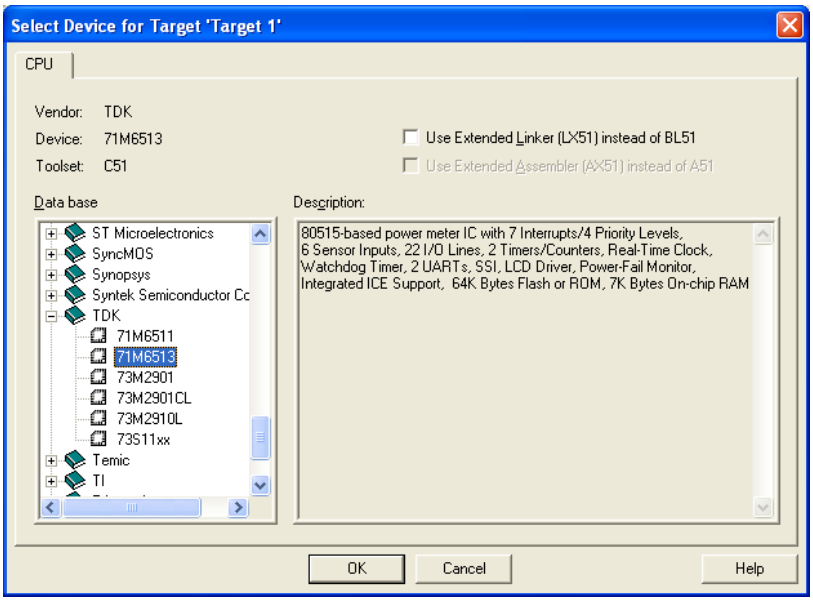
4.7.3 Manually Controlling the Keil Compiler Settings

If the method described in section "Adjusting the Keil Compiler Settings" is not used, the Keil compiler settings can also be controlled manually.

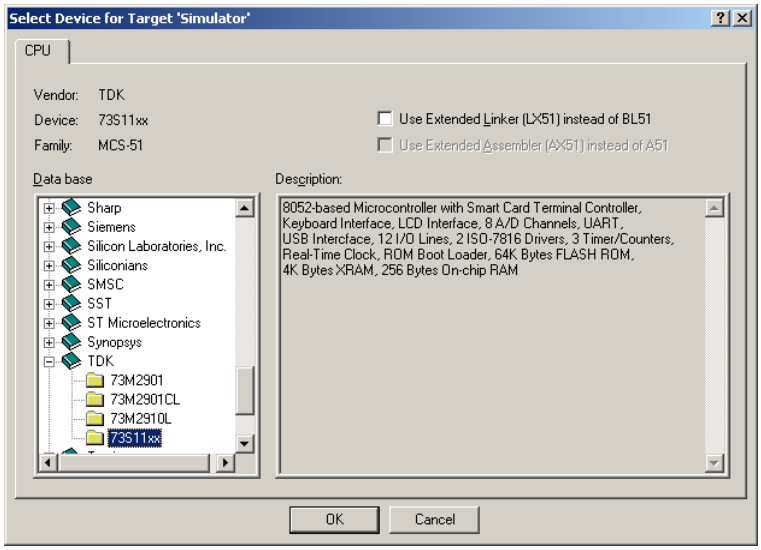
The target options should be selected in order to adapt the compiler controls properly to the target. The uVision compiler environment is started by selecting Programs → Keil → uVision2. uVision should start up and present the following window:



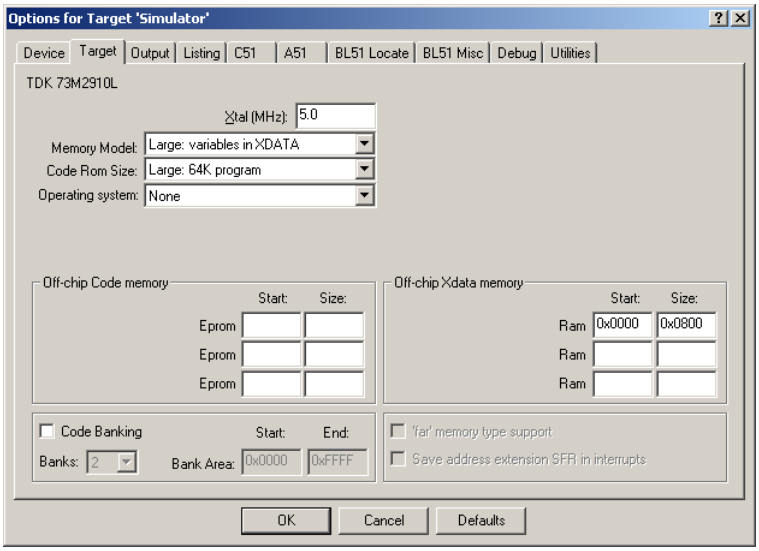
Under *Project → Options for Target1*, select the *Device* tab and check the selected device. Newer versions of the Keil Compiler offer selection of TERIDIAN (labeled "TDK") 71M6511 devices:



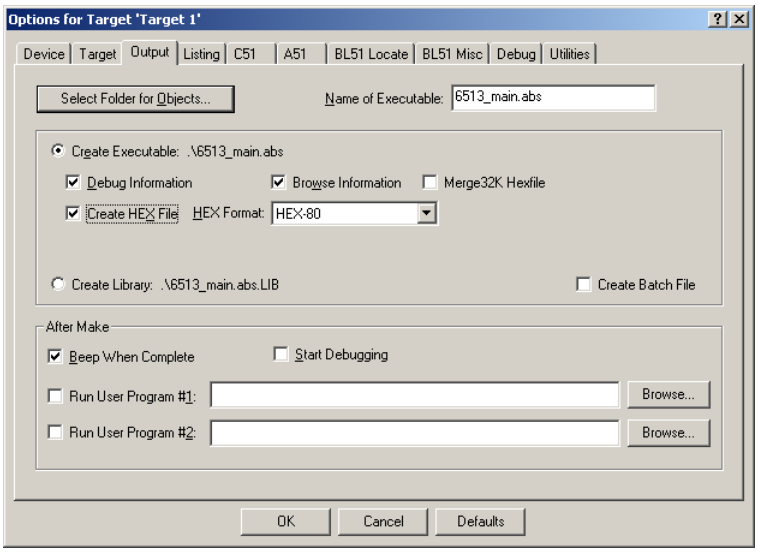
For older versions of the Keil compiler, select the TERIDIAN folder (labeled "TDK"), open it by clicking on the + sign and select *73M2910L* as the target device. Confirm by clicking *OK*.



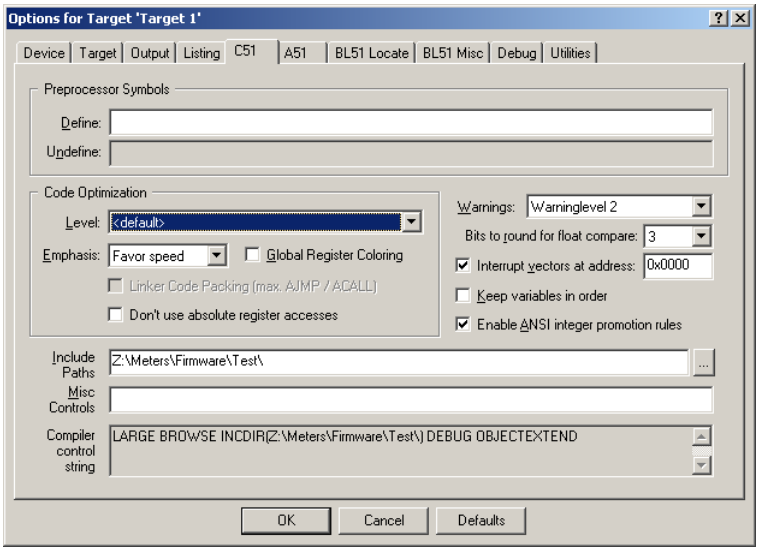
Under *Project → Options for Target1*, select the *Target* tab and enter the values in the fields as shown above. Confirm by clicking *OK*.



Under the Output tab, select a name for the executable (object) file with .abs extension' in the field labeled "Name of the executable" and check the fields by "Debug Information", "Browse Information" and "Create HEX File". This will guarantee that high-level source information will be embedded in the output file. Select *HEX-80* as the output format, as shown below:



Under the C51 tab, provide path names for the source files to be included, as shown below.



Click OK to set all the options selected for project and return to the main menu.

With the source and header files now existing in the newly created project, the files can be compiled using the Build Target option under the Project menu.

4.8 PROJECT MANAGEMENT TOOLS

With large software projects involving a multitude of source, object, list and other files in various revisions, it is very helpful to use a version control tool.

To manage file versions under Windows, Tortoise CVS, a free version control utility, might be useful. This utility can be found at http://www.tortoisecvs.org/ .

4.9 ALTERNATIVE COMPILERS

The Demo Code was written for the Keil compiler. However, alternative compilers may be used if the code is modified to ensure compatibility with the alternative compiler. One example of an alternative compiler is SDCC, a free compiler available from www. Sourceforge.net.

Note: The Keil extensions for the 8051 are not compatible with the 8051 extensions used by the SDCC.

The batch file BUILD6521.BAT is provided with the Demo Kit to support building object files using alternative compilers. This batch file uses the Keil compiler calls with the applicable compiler options and can therefore serve as examples on how to invoke alternative compilers. The linker control file LINK6521.TXT called by the batch files can show how to properly invoke linkers.

To compile with DOS-style tools, arrange for a DOS batch file to invoke the tools and set the properties of the batch file to leave the window open, so that errors can be seen. Then, to compile, double click on this batch file in Windows explorer.

4.10 ALTERNATIVE EDITORS

Many modern text editors have a feature called "tag jumping" that helps a programmer to read and understand unfamiliar code. TERIDIAN Semiconductor recommends using such an editor to read, understand and modify demonstration code. Tag jumping is a feature that is not supported by the Keil uVision editor.

This is how tag jumping works:

1. A "tag file generator" program is run on some directories full of .c or .h files. TERIDIAN Semiconductor recommends placing the tag file generator in a DOS batch file in the same directory as the project's make file. Wattmeter demonstration code includes such a batch file: "T.BAT". To run a batch file, double-click it in windows explorer. A DOS batch file is just an ASCII file (like a .C file) containing DOS commands. DOS commands are described at http://www.computerhope.com/msdos.htm .
2. The tag file should then be copied to convenient places for a text editor. TERIDIAN Semiconductor recommends copying the tag file into each source code directory. In that way, the default tag file location for most editors becomes just ".\tags" for all projects, and multiple projects do not conflict. Copying the tag file can be an automatic part of the DOS batch file that generates the tag file.
3. It is easiest if Windows explorer opens .C files automatically with the editor when they are clicked. To do this, change file associations. (See Windows help.)
4. Inside the editor, select a subroutine name or variable, then use the editor's "tag jump" feature. The editor immediately opens the file at the line where the subroutine or variable is defined. Or, if the same symbol is in several places, it offers a choice of files.

TERIDIAN Semiconductor recommends the "exuberant CTAGs utility" for generating tag files. The code can be found for free at: http://ctags.sourceforge.net/. The choice of a text editor is very personal. Many editors support Exuberant CTAGS. See the list of supporting tools at http://ctags.sourceforge.net/tools.html.

Some editors to be considered are:

- VIM, see http://www.vim.org/ a free VI editor. VIM is available in full-featured versions for Windows. VI is part of the POSIX standard, so using it is a portable skill. VIM wins awards for usability.
- UltraEdit http://www.ultraedit.com/ , an inexpensive (not free), professional Windows programming editor. This editor works like all other Windows applications, with extra features to support programming languages. NEDIT (The Nirvana Editor) is very similar, at http://www.nedit.org/. NEDIT runs on Unix with Motif, and also supports exuberant CTAGs.
- GNU Emacs, a free editor, also supports exuberant CTAGs. See: http://www.gnu.org/software/emacs/emacs.html

4.11 ALTERNATIVE LINKERS

Compiled and linked code can be significantly compacted by using the linker available with the Professional Compiler Kit PK51 from Keil (www.keil.com).

The LX51 Enhanced Linker supplied with the PK51 kit (http://www.keil.com/c51/lx51.asp) is capable of code compression by up to 8% by rearranging code segments for AJMP and ACALL usage.

All executables supplied with the Demo Boards were generated using the conventional compilers and linkers from Keil. That way, the supplied sources compile and link to the same code size as the pre-compiled object files.

If it is desired to add more options to the source code than the conventional linker can pack into a given code space, the LX51 Enhanced Linker should be considered.

5

5 DEMO CODE DESCRIPTION

5.1 80515 DATA TYPES AND COMPILER-SPECIFIC INFORMATION

5.1.1 Data Types

The 80515 MPU core is an 8-bit micro controller (MPU); thus operations that use 8-bit data types such as "char" or "unsigned char" work more efficiently than operations that use multi-byte types, such as "int" or "long". The Keil C51 compiler supports ANSI C data types as well as data types that are unique to the generic 8051 controller family. Table 5-2 lists available data types. Please refer to the Keil Cx51 Compiler User's Guide for more details.

Various types of address spaces are available for the 80515 MPU core of the 71M652X, and in order to utilize the various memory space types efficiently, the Demo Code uses variable type definitions (typedefs.) presented in this chapter.

To understand the data types, it helps to examine the internal data memory map of the 80515 MPU core, as shown in Table 5-1: .

<table>
<tr><th>Address</th><th>Direct addressing</th><th>Indirect addressing</th></tr>
<tr><td>0xFF</td><td rowspan="2">Special Function Registers (SFRs)</td><td rowspan="2">RAM</td></tr>
<tr><td>0x80</td></tr>
<tr><td>0x7F</td><td colspan="2" rowspan="2">Byte-addressable area</td></tr>
<tr><td>0x30</td></tr>
<tr><td>0x2F</td><td colspan="2" rowspan="2">Bit-addressable area</td></tr>
<tr><td>0x20</td></tr>
<tr><td>0x1F</td><td colspan="2" rowspan="2">Register banks R0…R7</td></tr>
<tr><td>0x00</td></tr>
</table>

Table 5-1: Internal Data Memory Map

General data type definitions:

typedef unsigned char uint8_t;

typedef unsigned short uint16_t;

typedef unsigned long uint32_t;

typedef signed char int8_t;

typedef signed short int16_t;

typedef signed long int32_t

Type definitions for internal data, lower 128 bytes, addressed directly:

typedef unsigned char data uint8d_t;

typedef unsigned short data uint16d_t;

typedef unsigned long data uint32d_t;

typedef signed char data int8d_t;

typedef signed short data int16d_t;

typedef signed long data int32d_t;

This is the fastest available memory (except registers), not battery-backed-up, but competes with stack, registers, booleans, and idata.

Note: For portability, see uint_fast8_t and its sisters, which are POSIX standard.

Type definitions for internal data, 16 bytes (0x20 to0x2F), addressed directly, and bit addressable:

typedef unsigned char bdata uint8b_t;

typedef unsigned short bdata uint16b_t;

typedef unsigned long bdata uint32b_t;

typedef signed char bdata int8b_t;

typedef signed short bdata int16b_t;

typedef signed long bdata int32b_t;

This is the fastest available memory, but it is not battery-backed-up. It competes with stack, registers, bools, data, and idata. The space is valuable for boolean globals and should not be wasted.

Booleans are not a normal part of stdint.h, but fairly portable. When using the Keil compiler, the Booleans are stored in the address range 0x20 to 0x2F. Keil functions return bools in the carry bit, which makes code that's fast and small.

typedef bit bool;

#define TRUE 1

#define FALSE 0

#define ON 1

#define OFF 0

Type definitions for internal data, 256 bytes, in the upper 128 bytes addressed indirectly:

typedef unsigned char idata uint8i_t;

typedef unsigned short idata uint16i_t;

typedef unsigned long idata uint32i_t;

typedef signed char idata int8i_t;

typedef signed short idata int16i_t;

typedef signed long idata int32i_t;

This is fairly fast, not battery-backed-up memory, slower than the data in the lower 128 bytes of internal memory. Competes with data for space.

Type definitions for external data, 256 bytes of 2K of CMOS RAM:

typedef unsigned char pdata uint8p_t;

typedef unsigned short pdata uint16p_t;

typedef unsigned long pdata uint32p_t;

typedef signed char pdata int8p_t;

typedef signed short pdata int16p_t;

typedef signed long pdata int32p_t;

The upper byte of the XDATA address is supplied by the SFR 0xBF (ADRMSB) on the 71M6521 meter ICs. On other 8051 processors, P2 is used for this purpose. This memory range is accessed indirectly, still fairly fast, not battery backed-up. This is a logical place for nonvolatile globals like power registers and configuration data.

Type definitions for external data, 2Kbytes of CMOS RAM, accessed indirectly via a 16-bit register:

This is the slowest but largest memory are, not battery backed-up. It can be used for everything possible. On Keil's large model, this is the default.

typedef unsigned char xdata uint8x_t;

typedef unsigned short xdata uint16x_t;

typedef unsigned long xdata uint32x_t;

typedef signed char xdata int8x_t;

typedef signed short xdata int16x_t;

typedef signed long xdata int32x_t;

Type definitions for external read-only data, located in code space:

typedef unsigned char code uint8r_t;

typedef unsigned short code uint16r_t;

typedef unsigned long code uint32r_t;

typedef signed char code int8r_t;

typedef signed short code int16r_t;

typedef signed long code int32r_t;

Access is indirect via a 16-bit register. This is the slowest but largest space, nonvolatile programmable flash memory. It should be used for constants and tables

Note: Throughout the Demo Code, an attempt has been made to put the most frequently used variables in the fastest memory space.

Data Type	Notation	Bits	Bytes	Comments
Bit	Bbool	1		Unique to 8051
Sbit		1		Unique to 8051
SFR		8	1	Unique to 8051
SFR16		16	2	Unique to 8051
signed/unsigned char	U08	8	1	ANSI C
enum	enum	8 or 16	1 or 2	ANSI C
unsigned short	U16	16	2	ANSI C
signed short	S16	16	2	ANSI C
signed/unsigned int	U16	16	2	ANSI C
signed int	S16	16	2	ANSI C
unsigned long	U32	32	4	ANSI C
Float	F32	32	4	ANSI C

Table 5-2: Internal Data Types

5.1.2 Compiler-Specific Information

The 8051 has 128 bytes of stack, and this motivates Keil C's unusual compiler design. By default, the Keil C compiler does not generate reentrant code. The linker manages local variables of each type of memory as a series of overlays, and uses a call-tree of the subroutines to arrange that the local variables of active subroutines do not overlap.

The overlay scheme can use memory very efficiently. This is useful because the 71M652X chips only have 2k of RAM, and 256 bytes of internal memory.

The compiler treats uncalled subroutines as possible interrupt routines, and starts new hierarchies, which can rapidly fragment each type of memory and interfere with its reuse.

To combat this, the following measures were taken when generating the Demo Code:

- The code is organized as a control loop, keeping most code in a single hierarchy of subroutines,
- The programmers eliminated unused subroutines by commenting them out when the linker complained about them. Also, the Demo Code explicitly defines interrupt code and routines called from interrupt code as "reentrant" so that the compiler keeps their variables on a stack.
- When data has a stable existence, the Demo Code keeps a single copy in a shared static structure.

With these measures applied, the Demo Code uses memory efficiently, and normally no memory issues are encountered. The Demo Code does not have deep call trees from the interrupts, so "small reentrant" definitions can be used, which keep the stack of reentrant variables in the fast (small) internal RAM.

The register sets are also in internal memory. The C compiler has special interrupt declaration syntax to use them. The "noaregs" pragma around reentrant routines stops the compiler from accessing registers via the shorter absolute memory references. This is because the Demo Code uses all four sets of registers for different high-speed interrupts.

Using "noaregs" lets any interrupt routine call any reentrant routine without overwriting a different interrupt's registers.

There is a known defect in version 7.50a of the Keil compiler:

Memory types must be explicitly defined in local variables. Using a predefined type is not explicit enough, i.e. "char xdata c;" is ok. "typedef char int8_t; ... int8_t data c;" is OK,

but "typedef char data int8d_t; ... int8d_t c;" is not OK.

5.2 DEMO CODE OPTIONS AND PROGRAM SIZE

Since the 71M6512 is available with three different memory sizes, different versions of the Demo Code are provided that take into account the available memory size (see Table 5-3). An attempt has been made to provide the most common features in each version of the Demo Code. Flexibility is provided by the source code for users when re-compiling the source code: If a certain feature is not required, it can be left out and replaced with a different feature of equal or smaller code size.

The object files contained in the Demo Kits have been generated with the following Keil compiler versions:

- C compiler: C51.exe, V8.02
- Assembler: A51.exe, V8.00
- Linker/Locator: BL51.exe, V6.00
- Librarian: LIB51.exe, V4.24
- Hex-converter: OH51.exe, V3.03
- Dialog DLL: DP51.dll, V2.47
- Target DLL: bin\mon51.dll, V2.40
- Dialog DLL: TP51.dll, V2.47

Version	Flash Code Size	Description
Basic Wattmeter	8KB	Demonstrates a meter with 8KB of code space. The software offers tamper protection, calibration and nonvolatile energy registers. It utilizes special CE code. This implementation has a 0.82KB margin of empty code space.
Intermediate Meter	16KB	Demonstrates a meter with 16KB of code space. The software is easy to reconfigure by recompiling, and offers full tamper protection, calibration and nonvolatile energy registers. This implementation has a 3.2KB margin of empty code space
Demonstration Meter	32KB	Demonstrates a meter with 32k of code space. The software is easy to reconfigure by recompiling. It shall has full tamper protection, calibration and nonvolatile energy registers. The software demonstrates a full feature set. This implementation has a 6.4KB margin of empty code space, not including the command line interpreter, but including a calibration interface

Table 5-3: Demo Code Versions

In addition to providing flexibility, an attempt has been made to leave a certain amount of unoccupied memory space when generating the Demo Code. This should provide some room for users who want to modify the Demo Code and experiment with small changes.

The tables presented below show the features available for the three versions of the Demo Code plus the code size required for each feature. Entries for code size are approximated and depend on code module combination.

Y means that the feature is implemented, N means that it is not. N/opt means that the feature may be implemented if enough memory space is available.

Feature	Code Size	8KB	16KB	32KB	Description
CT and shunt resistors	1KB to 2.5KB	Y	Y	Y	Configurations include one element, one phase, and neutral current, as well as two elements with two phases
Rogowski coils	2.5KB	N	N	N	Needs special CE code

Table 5-4: Current Sensing Options

Feature	Code Size	8KB	16KB	32KB	Description
Chopping of VREF	0.06KB	Y	Y	Y	Control of the chopping bit
Temperature compensation of VREF	0.1KB	Y	Y	Y	Digital compensation using the GAIN_ADJ input of the CE, based on linear and quadratic temperature coefficients
RTC compensa-tion using mains frequency	0.2KB	N/opt	N/opt	N/opt	Optional compensation of RTC by counting cycles on mains. Correction does not occur when frequency measurement is inhibited by low voltages.
RTC constant compensation	0.1KB	Y	N/opt	Y	Constant rate compensation only.
Full RTC compensation	0.2KB	N	N/opt	N/opt	2^{nd}-order compensation of RTC to 1ppb, using temperature. Correction does not occur when the ADC mux is off-line.
Temperature measurement	0.0K	Y	Y	Y	Provides difference from calibration temperature to 0.1 C when calibrated

Table 5-5: Compensation Features

Feature	Code Size	8KB	16KB	32KB	Description
Wh absolute value		Y	Y	Y	Standard option of kilowatt hours, "3. 999999" The annunciator 3 at the beginning is optional if no other registers are supported.
Pulse output for Wh absolute value	0.23KB	Y	Y	Y	Standard option of 1 kh/pulse on both DIO 6 and DIO 2. 8K version's pulse output is controlled by the CE without MPU intervention.
VAn register	1KB	N	N/opt	N/opt	Optional volt-amperes, " 999999", replaces Wh options
VAn pulse output	0.25KB	N	N/opt	N/opt	Optional volt-amperes, 1 kh/pulse, replaces Wh options
Wh equation 0	0.2KB	Y	Y	Y	

Feature	Code Size	8KB	16KB	32KB	Description
Wh equation 1	0.2KB	N	N/opt	N/opt	
Wh equation 2	0.2KB	N/opt	N/opt	N/opt	
Frequency register	0.1KB	N	Y	Y	Inhibited if freq > 70Hz or voltage is below the threshold
Wh net metering	0.4KB	N	N/opt	Y	Used only for automatic calibration. No display is provided.
Wh export register	0.25KB	N	N/opt	N/opt	Wh exported, display reads "3 999999"
Wh export pulse output	0.25KB	N	N/opt	N/opt	Wh exported, display reads "3 999999"
VARh register	0.1KB	N	N/opt	Y	For autocalibration, and power factor calculation, signed (net metering).
VARh pulse output	0.25KB	N	N/opt	Y	
VARh import register	0.4KB	N	N/opt	Y	
VARh import pulse output	0.25KB	N	N/opt	Y	
VARh export register	0.4KB	N	N/opt	Y	
VARh export pulse output	0.25KB	N	N/opt	Y	
Operating hours register	0.36KB	N	N/opt	N/opt	"5 99999.9" Nonvolatile count of tenths of hours of powered operation since first cold start.
RCT time register	0.18KB	N/opt	Y	Y	
RCT date register	0.21KB	N/opt	Y	Y	
Pulse source selection	0.4KB	N	N/opt	Y	This is the ability to route most calculated energy values to a pulse output. The 8K code provides only Wh pulses.
Dual IMAX registers	0.2KB	Y	Y	Y	IMAX2 adjusts current, Wh and VARh from channel B to same units as A. Creep thresholds are required, but need not be adjusted when IMAX2 changes.
RMS current register	0.2KB	N	N/opt	Y	Implemented for two phases "P", 1 & 2 "1.P 000.000"
RMS voltage register	0.2KB	N	N/opt	Y	Implemented for two phases "P", 1 & 2 "1.P 000.000"
Power factor register	0.3KB	N	N/opt	N/opt	Two phases are displayed with sign. Reset at reset and the start of each minute. Volatile.

Table 5-6: Power Registers and Pulse Output Features

Feature

Feature	Code Size	8KB	16KB	32KB	Description
Creep mode	0.37KB	Y	Y	Y	Adjustable at calibration. If abs(W) < Creep threshold, then creep mode.
Zero accumulator of CE	N/A	Y	Y	Y	The pulse accumulation register in the CE is cleared to prevent spurious pulses from low current noise.
Current threshold	N/A	Y	Y	Y	Adjustable at calibration. Set If max(abs(IA^2), abs(IB^2)) < Current threshold then creep mode. Current is calculated from RMS if possible, or, if below 0.1A, from VA / V, where VA is calculated as sqrt(Wh^2 + VARh^2) For all elements.
Voltage threshold	0.12KB	N	Y	Y	Adjustable at calibration. If max(abs(VA^2), abs(VB^2)) < Volt threshold inhibit frequency measurement, (frequency of zero) Inhibit use of zero crossing counts, (main edge count is zero), iInhibit voltage phase measurement (if any) This feature is needed only if frequency or mains edge count is present.

Table 5-7: Creep Functions

Feature	Code Size	8KB	16KB	32KB	Description
Brownout mode	0.1KB	Y	Y	Y	Used to enter sleep and LCD modes. Command line interface is available (32KB) when resetting into this mode. Command prompt in this mode to be "B>".
LCD mode	0.5KB	Y	Y	Y	Is entered automatically when a sag event occurs. Displays the Wh register, waits 7 sec using wakeup timer, then initiates sleep mode.
Wake button	0.5KB	Y	Y	Y	When in sleep mode, enters LCD mode.
Wake timer	0.5KB	Y	Y	Y	Used to exit the LCD mode, and enter sleep mode.

Table 5-8: Operating Modes

Note: The sleep mode does not require any support by MPU code. The mission mode is represented by the sum of the other code features.

Feature

Feature	Code Size	8KB	16KB	32KB	Description
FLAG interface protocol	2.5KB	N	N/opt	N/opt	Implements the FLAG protocol stack (see the FLAG specification). The FLAG protocol reads and writes registers in the meter and responds to all ports.
Reception of calibration parameters via the serial interface	2.0KB	Y	Y	Y	Simple serial calibration system that supports reading data and writing calibration values, including CE data, MPU calibration and RTC settings. Meter operation is not required when this feature is in use. Intel hex records are used.
Count of calibrations since first cold reset.	01.KB	Y	Y	Y	Counts calibrations. 0..254, 255 = "many". The count is protected by a checksum. The first cold reset is detected by an invalid EEPROM. This is a tamper-detection feature.
Auto-calibration	3.5KB	N	N/opt	Y	Internal automatic calibration, from command line interface if available, or DIO state at start. Calibration adjusts phase, as in the "fast calibration" described in the DBUM.
Command Line Interface (CLI)	14KB	N	N	Y	Text-based commands give access to CE data, RAM, IO registers. No help, profile or load features. Versions without CLI can be controlled with IOMERGE. The command line interface's space is to be counted as "unused" when calculating code space margin.
Optical FLAG	1.2KB	N	N/opt	N/opt	Implementation of the physical FLAG layer on UART 1, 300 BAUD, using pulse output
Wired FLAG	1.2KB	N	N/opt	N/opt	Implementation of the physical FLAG layer on UART 0, 9600 BAUD,
Save registers when sag occurs	0.75KB	Y	Y	Y	Saves power and error registers on sag detection.
Save to flash memory	0.9KB	N/opt	N/opt	N/opt	Compilation option to save calibration, error and power register data to internal flash. When a flash area is used-up, it is marked, and the next one is used. When all areas are used up, an error is recorded and write operations are inhibited.
Save to and restore from EEPROM	0.7KB	Y	Y	Y	Saves and restores calibration, error and power register data to and from EEPROM. When an EEPROM area is used-up, it is marked, and the next one is used. When all areas are used up, an error is recorded and write operations are inhibited.
Checksum	0.2KB	Y	Y	Y	Each revenue-affecting data area is protected by a simple checksum

Feature	Code Size	8KB	16KB	32KB	Description
Error recording and saving	0.4KB	N/opt	N/opt	N/opt	Errors are recorded in 16 bit words, one bit per error. All error collection is reset when the magnetic tamper DIO is asserted for 1 second. Error data is protected by a checksum. The time stamp (hour, day and month of assertion) and the bit number of the five most recent errors are saved.
Microwire EEPROM	0.2KB	N/opt	N/opt	N/opt	
I2C EEPROM	0.2KB	Y	Y	Y	

Table 5-9: Calibration and Various Services

5.3 PROGRAM FLOW

5.3.1 Startup and Initialization

The top-level functionality of the Demo Board is controlled by the high-level functions. As with every C program, the core of the function is in the main() program. The main() program is contained in the main.c source file. It performs the following steps (see Figure 5-1, Figure 2-1, and Figure 5-2):

1. Reset watchdog timer
2. Process the pushbutton (PB) when in BROWNOUT mode.
3. Initialization for hardware, pointers, metering variables, UART buffers and pointers, CE, restoration of calibration coefficients, initialization of LCD w/ "HELLO" message), enabling CE and pulse generators.
4. Execute the main_run() routine in an endless loop. In this loop, the background tasks, such as metering, processing of timers, etc. are performed. Afterwards, if a command is pending, the command line interface (CLI) is serviced.

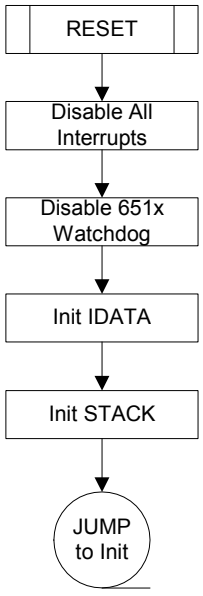


Figure 5-1: STARTUP.A51

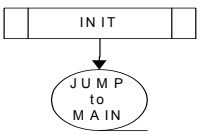


Figure 5-2: INIT.A51

Before the MPU gets to execute the main() program, it will execute the startup instructions contained in the STARTUP.A51 assembly program (Figure 5-1). Upon completion, STARTUP.A51 causes a jump to the label C_START, which is contained in the second startup assembly program named init.A51 (Keil/C51/LIB directory, see Figure 5-2). Init.A51 finally causes the jump to main(). The startup files are described in section 5.10 .

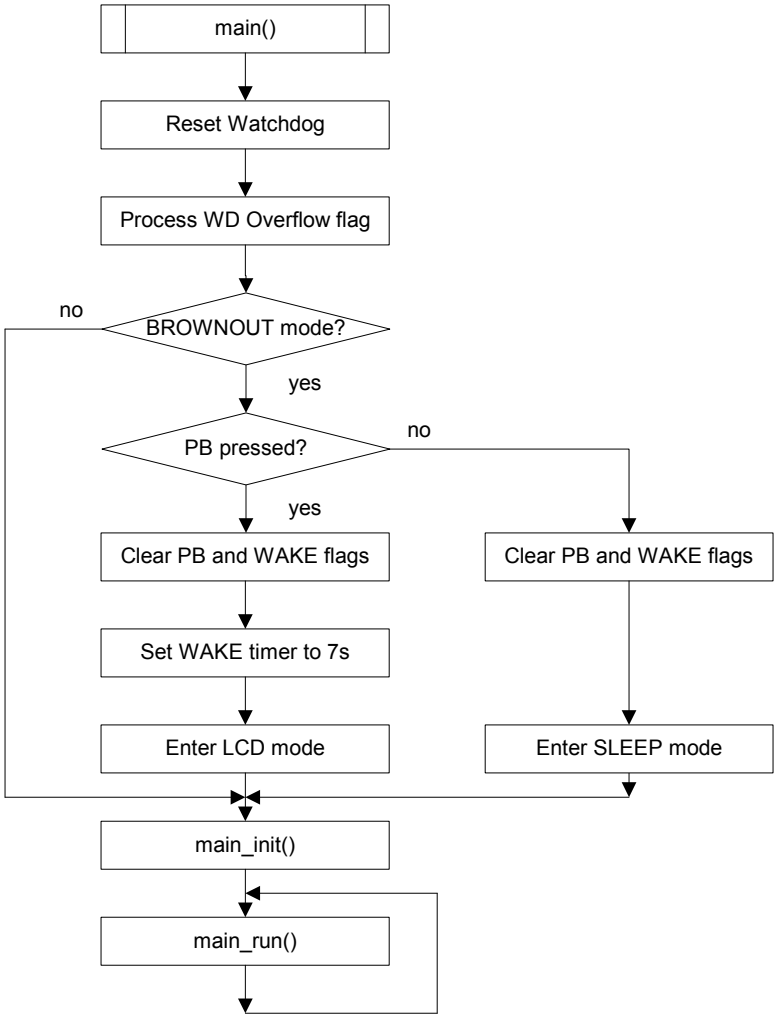


Figure 5-3: main() Program

The stack is located at 0x80, growing to higher values, while the reentrant stack is located at 0xFF, growing downwards.

Once operating, the main() program (Figure 5-4)expects regular interrupts from the CE. If no interrupts occur, the main() program will cease to trigger the watchdog timer, resulting in a reset condition, if the watchdog timer is enabled.

The main() program calls the main_init() (Figure 5-4) and the main_run() (Figure 5-5) routines. main_init() is used for hardware and software initialization, main_run() is the routine that is executed in an endless loop and that takes care of background and foreground processing.

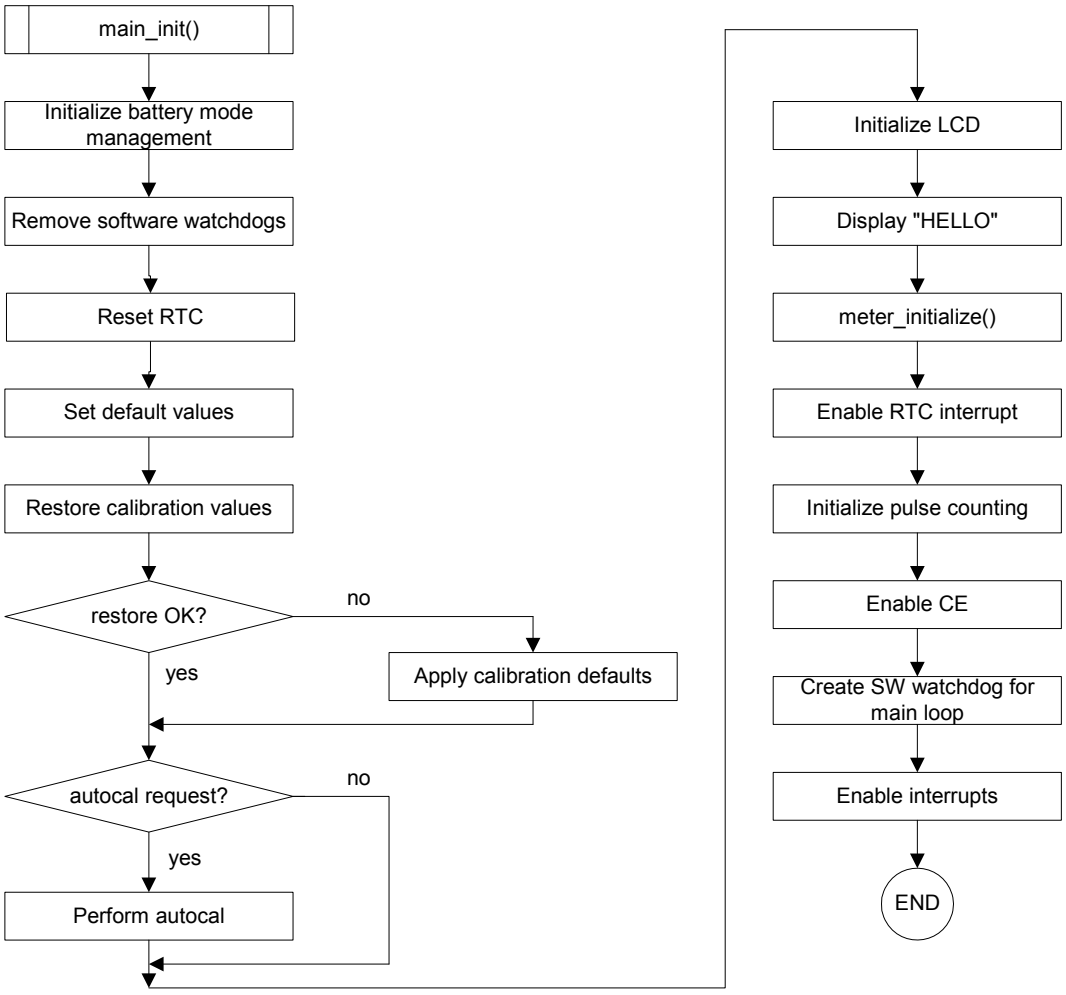


Figure 5-4: main_init() Function

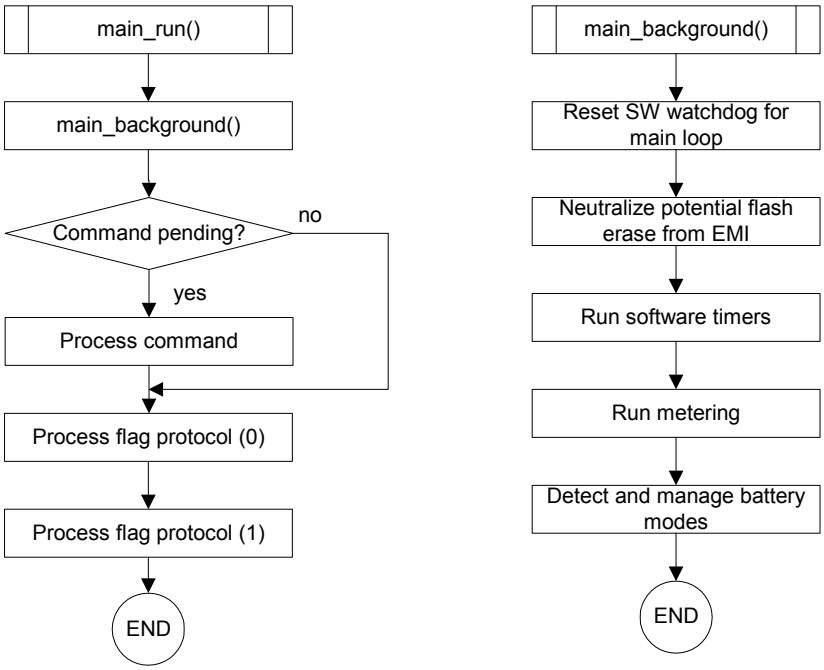


Figure 5-5: main_run() Function

5.4 BASIC CODE ARCHITECTURE

The TERIDIAN 71M652X firmware can be divided into two code parts. One is the Background task that is executed whenever there are no other higher priority exceptions such as the servicing of interrupts. The second part consists of the interrupt-driven code (Foreground) tasks, such as the CE_BUSY Interrupt, Timer Interrupt, and other Interrupt service routines. The background code takes care of the non time-critical functions starting with the system reset, and this code is executed every time when there are CPU resources available after taking care of all interrupt-driven tasks. The background of the 71M652X firmware is implemented as a very simple state machine. One state is serving the command inputs and the other is idle/Display control.

5.4.1 Initialization

When the power applied for the first time or RESETZ is asserted, the 71M652X device executes the code pointed to by the reset vector.

5.4.2 Foreground

There are total 12 interrupts available for the 80515, and the revision 4.7a Demo Code uses a total of 11 interrupts. Table 5-10 shows the interrupt service routines (ISRs), the corresponding vectors (Table 6-58 in section 6.3.5.4) and their priority, as assigned by the MPU using the IP0 and IP1 registers (see section 6.3.5.2).

Interrupt Source	Interrupt Service Routine	External or Internal Interrupt	In source file	Vector	Priority (3 = highest)
Pulse count	pcnt_w_isr()	EXT0	pcnt.c	0x03	0
Pulse count	pcnt_v_isr()	EXT1	pcnt.c	0x13	3
Flash-Write collision fwcol0	fwcol_isr()	EXT2	flash.c	0x4B	0
Flash-Write collision fwcol1	fwcol_isr()	EXT2	flash.c	0x4B	0
CE Busy	ce_busyz_int()	EXT3	ce.c	0x53	3
Power fail/power return	pll_isr()	EXT4	batmodes_20.c	0x5B	3
EEPROM	eeprom_isr()	EXT5	eeprom.c	0x63	0
XFER busy	ce_xfer_busyz_rtc_int()	EXT6 (shared w/ RTC)	ce.c	0x6B	2
RTC	rtc_isr()	EXT6 (shared w/ XFER)	rtc.c	0x6B	2
Timer0	tmr0_isr()		tmr0.c	0x0B	0
Timer1	tmr1_isr()		tmr1.c	0x1B	3
UART 0	es0_isr		serial.c	0x23	0
UART 1	es1_isr		serial.c	0x83	0

Table 5-10: Interrupt Service Routines

In general, a higher priority interrupt can preempt lower-priority interrupt code. The interrupt priority hardware is controlled by two registers, IP and IP1 (named IPL and IPH in the demo code). The MPU supports four priorities, and a fifth is possible with a small amount of software support.

The best practice is to set priorities once, near the start of initialization. Setting priorities dynamically while interrupts occur can have undefined results. Since some of the interrupts detect power failures that can occur at any time, changing interrupt priorities in the middle of the code is not recommended.

In the 6521 demo code, interrupt priorities are set higher for urgent tasks. Among equally-urgent tasks, priorities are set higher for faster interrupts. The following describes interrupt priorities for the version 4.3.3 of the Demo Code:

The priority is set once, in main_init() of main\Main.c. It is also cleared to 0s in the soft reset routine, but this is followed by logic that calls four RTIs to reset the interrupt acknowledge logic for all four hardware interrupt levels. The system priority value is assembled from constants in Main\options_gbl.h. The constants are defined in Util\priority2x.h.

The highest priority interrupt group are the PLL_OK interrupt (external interrupt 4, see Main\batmodes_20.c), and timer 1. PLL_OK is urgent because it indicates power supply failure, and the software must start battery modes. Timer 1 shares the same priority bits, and is currently unused (sample code is in Io\tmr1.c, &.h), though earlier versions used it to set the real-time-clock.

The high-priority interrupt group is used for CE_BUSY (external interrupt 3, see Meter\ce.c), pulse counting (external interrupts 0 and 1, Meter\pcnt.c) and Serial 1 (Io\ser1.c&.h). External interrupt 3 and 1 share priority bits, as does external interrupt 0 and serial 1. CE_BUSY is urgent because it occasionally reads the CE's status to detect sag. The pulse counting interrupts are less urgent, but they are small and run very quickly. Serial 1 is intended for AMR, so making its interrupts high priority should help its data transfer timing to be more reliable.

The low priority group contains Serial 0 and Timer 0. These can generally wait a millisecond, and if necessary, can afford to miss fast interrupts. Serial 0 is the command line interface (See the directory Cli), and Timer 0 is run at a 10 millisecond interval as the timebase for the software timers (Util\tmr.c, Io\tmr0.c&.h). Serial 0 shares its priority bits with

the interrupt of the EEPROM (external interrupt 5), currently unused (code is available in Io\eeprom.c). Timer 0 shares its interrupt priority bits with FWCOL, the flash write timing interrupt, also unused (flash code is in Util\flash.c).

The lowest priority is xfer_busy_isr() (Meter\ce.c) and the rtc_isr() interrupts (Io\rtc.c; both share external interrupt 6, Meter\io652x.c). These can usually wait up to half a second. The XFER_BUSY interrupt, in particular, takes up to 4 milliseconds to copy data from the CE, so though it is very important, it needs to be low priority in order to let other interrupts run.

The RTC can be calibrated by using the RTC-1-Second interrupt to toggle a DIO pin, and measuring the external square wave against a traceable time standard. In this case, a calibration mode must temporarily turn off the CE (it shares the interrupt) set external interrupt 6 to the highest priority and the code leading from the vector to the RTC's DIO-toggle should have an unchanging execution time.

Although the demo code does not do this, it is possible to run preemptive code at the same interrupt priority as the main loop. This creates a fifth priority below the lowest priority. To do this, set an interrupt to the lowest priority. This interrupt's service routine must push the address of the fifth-priority code on the stack, and run RTI. RTI clears the fourth-priority hardware, and then returns into the fifth-priority code, running it at the same interrupt level as the main loop. For example, this permits preemptive software timers that run at the same priority as the main loop.

All interrupt service routines (ISRs) must be declared "small reentrant". Also, all routines called by ISRs must be re-entrant as well. Priorities are set using the IP0 and IP1 SFRs, as follows:

- IP0 (SFR 0xA9) = 0x1A = 0001 1010
- IP1 (SFR 0xB9) = 0x2C = 0000 1100

This results in the priority assignment shown in Table 5-11.

Group	IP1 Bit	IP0 Bit	Priority	Affected Interrupts		
0	0	0	0	External interrupt 0 (DIO)	UART 1 interrupt	-
1	0	1	1	Timer 0 interrupt	-	Ext 2 (comparators)
2	1	0	2	External interrupt 1 (DIO)	-	Ext 3 (CE_BUSY)
3	1	1	3	Timer 1 interrupt	-	Ext 4 (comparators)
4	0	1	1	UART 0 interrupt	-	Ext 5 (EEPROM)
5	0	0	0	-	-	Ext 6 (XFER_BUSY, RTC_1S

Table 5-11: Interrupt Priority Assignment

Timer Interrupt

timer0 of the MPU is the main system timer, and it is used to generate a 10ms timer tick, which is adjusted for MPU clock speed. The timer tick (variable tick_count) is used to control the software timers. The software timers are updated by the stm_run() function in the main loop of the background task. Eight software timers can be simultaneously running.

If it is desired to change the system timer to timer1, the include file called out in stm.c has to be changed to tmr1.h.

timer1 is used for delay functions, e.g. for EEPROM or RTC access control. Timer 1 is enabled and starts functioning by calling the "Add_Delay_Func()" function as defined in the timer.c module.

Various macros are available to control the timers:

- tmr_start(A, B, C) has three parameters: A is the timer time, the number of ticks to reload on each interrupt. B is true if the timer should restart itself when it expires. C is a pointer to a reentrant function.
- tmr_stop() stops the timer.
- tmr_running() returns TRUE if the timer is running.

These routines are very similar to the software timer commands, in stm.h.

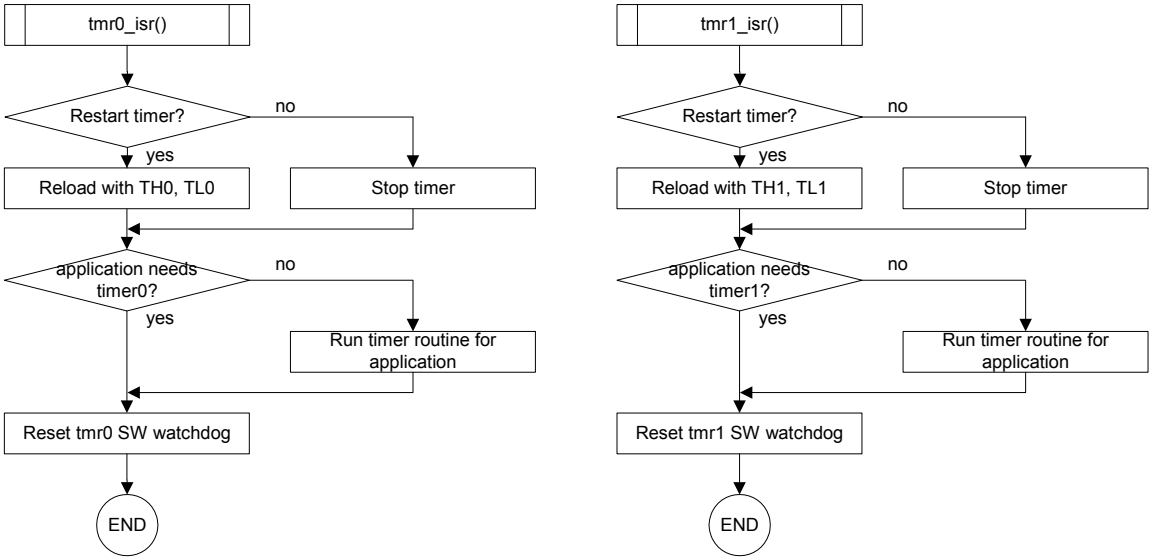


Figure 5-6: Timer ISRs

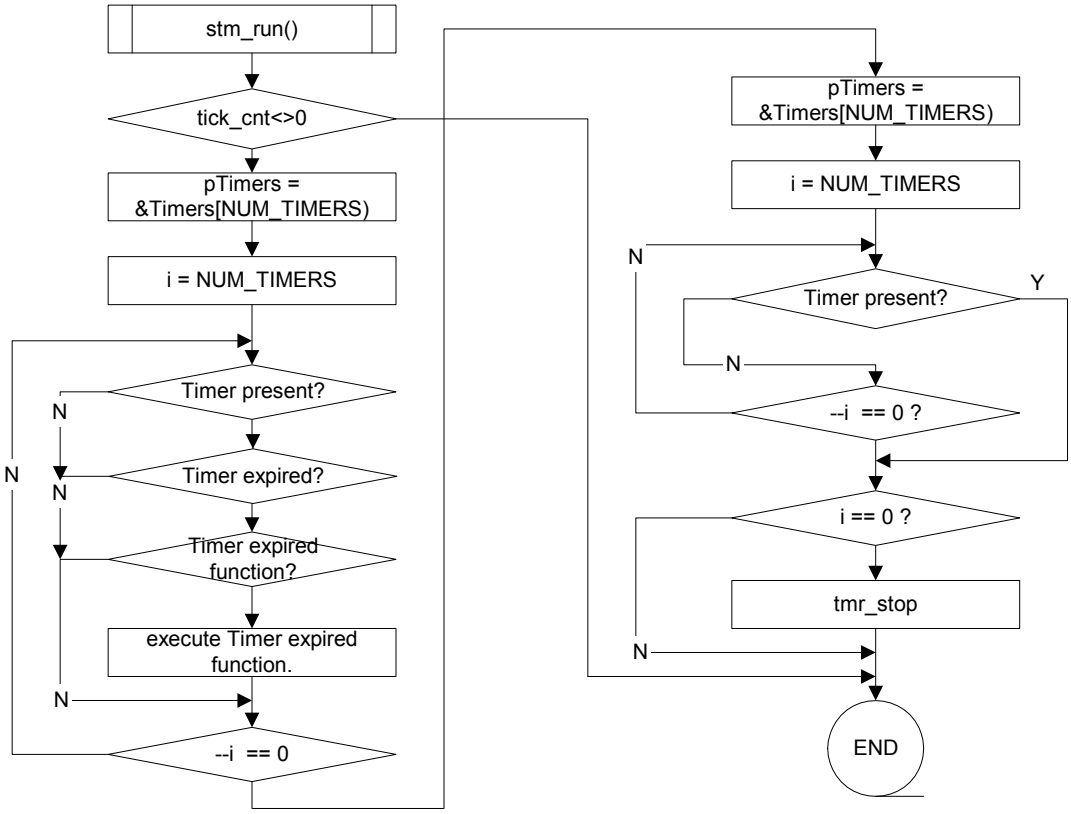


Figure 5-7: stm_run() - Process Software Timers (non-ISR)

CE_BUSY Interrupt

CE_BUSY interrupt is used for handling the outputs of the CE that are refreshed every 396µs, i.e. SAG detection.

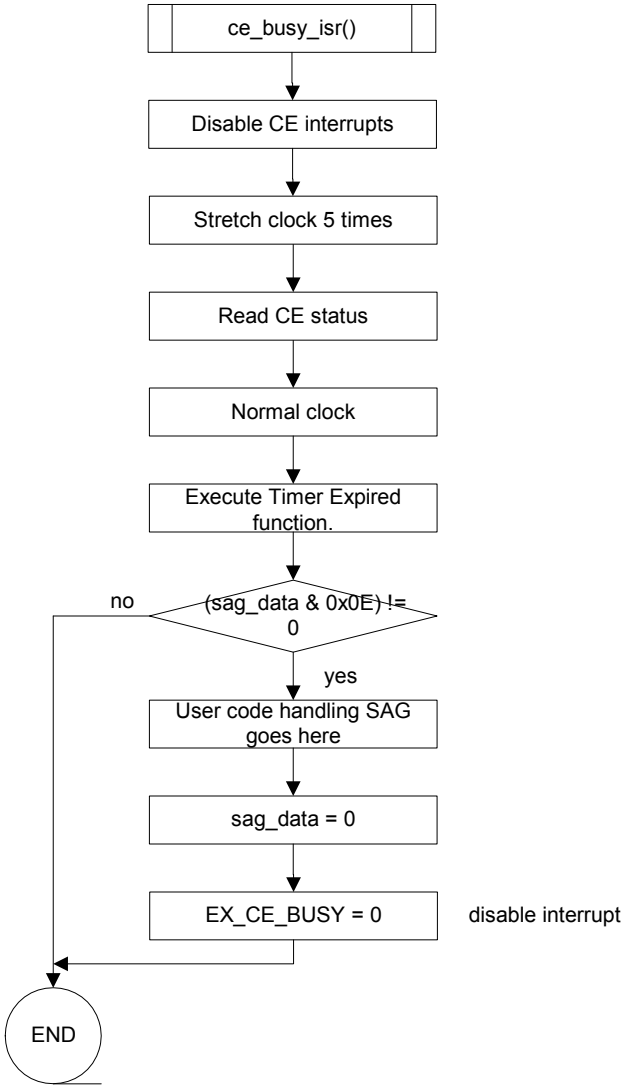


Figure 5-8: CE_BUSY ISR

XFER_BUSY and RTC Interrupt

The XFER Busy interrupt is requested by the CE at the conclusion of every accumulation cycle. The interrupt service routine copies the CE output data to the MPU internal data RAM for further processing by the MPU, which is performed by the background task. The handling of data for the generation of pulses is also managed in this ISR.

Processing of CE data waits until the second interrupt after one second has elapsed, since it takes roughly one second for the PLL in the CE to settle and (therefore) for the filtering to be reliable (variable ce_first_pass). Thus, the first samples from the CE are discarded.

The copy operations stated in the flow chart are implemented with the MEMCPY_MCE macro, which moves data between internal RAM and CE DRAM or vice versa. Due to the wait states that apply to accesses of CE DRAM, this operation cannot be done directly.

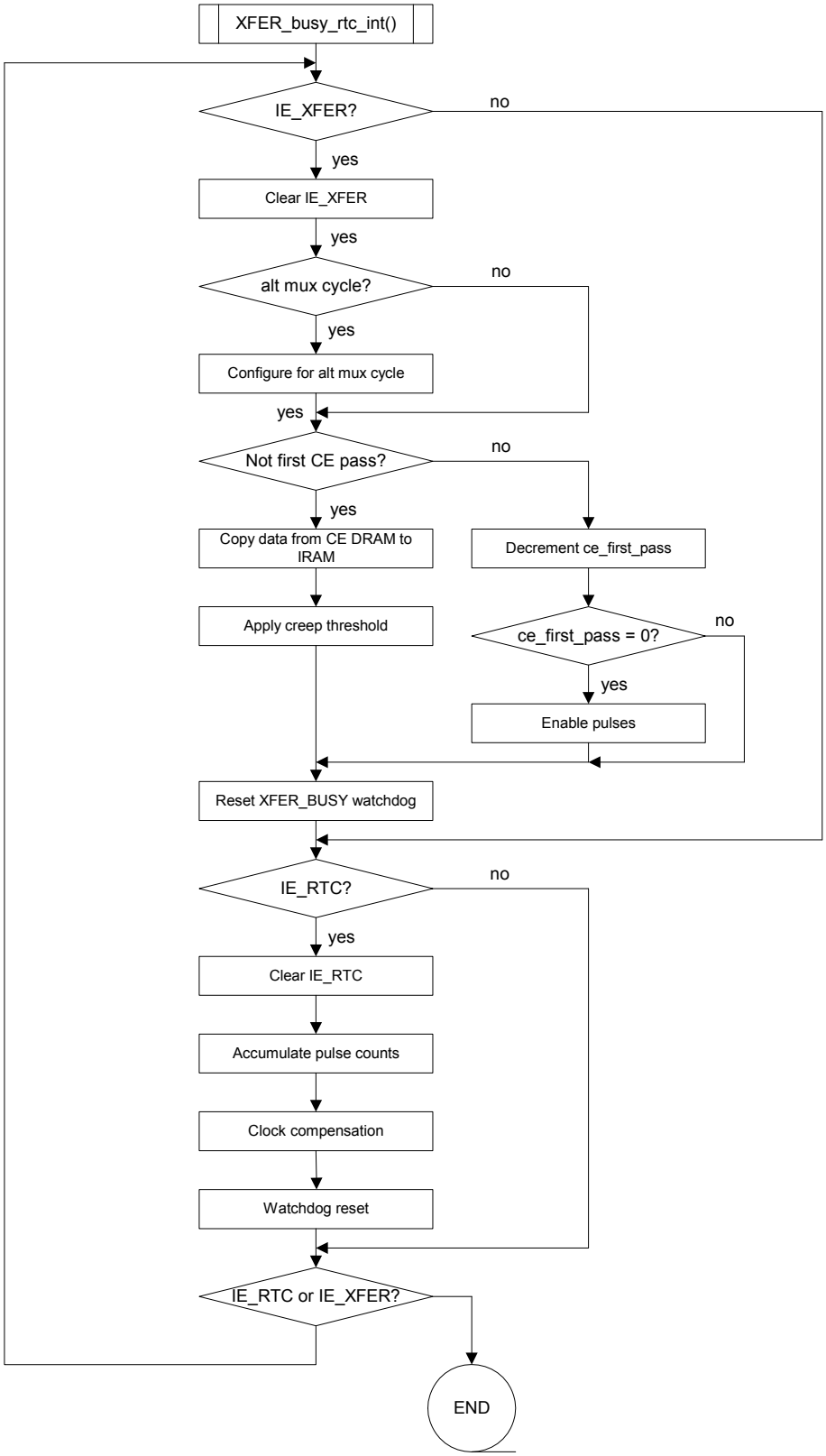


Figure 5-9: XFER_BUSY/RTC ISR

The interrupt service routine includes a loop. Without this loop, there is the chance of a rare, subtle timing error because interrupt EXT6 is edge-triggered and the two interrupt sources "or" into it. The timing error will occur if the

RTC interrupt happens, and then the XFER interrupt happens after the IE_XFER flag is already tested, but before the RTC interrupt is cleared. In this case, the signal to EXT6 will remain set, and never have an edge to cause another interrupt 6. Therefore, the XFER_BUSY interrupt will hang forever, thus preventing delivery of the data to the meter.

To prevent this error condition, at the end of the XFER_BUSY_RTC service routine, both interrupt flags are again checked, and when at least one of them is active, the processing starts again.

Both interrupts have a backup check - the main watchdog timer is never reset unless both interrupts run.

5.4.2.1 SERIAL Interrupt

es0_isr is the ISR servicing UART 0. In this ISR, the UART data is sent and received along using flow control, if enabled. Parity and other serial controls are managed in this ISR. The alternative serial port, UART 1 uses an ISR with identical code structure (es1_isr).

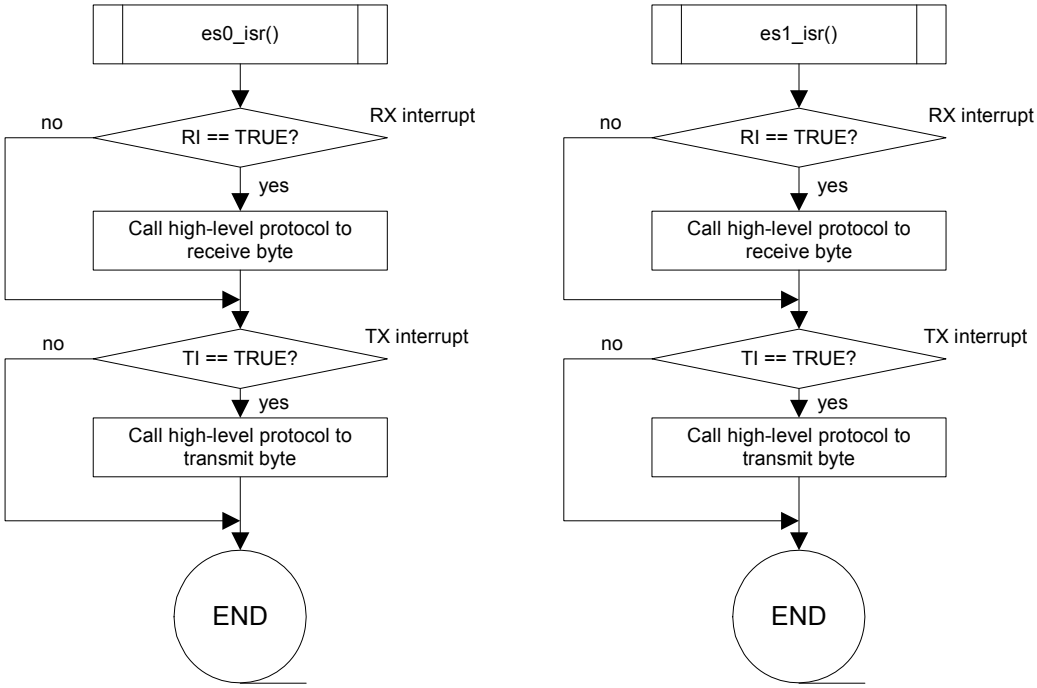


Figure 5-10: Serial 0 and 1 isr

5.4.3 Background Tasks

meter_run()

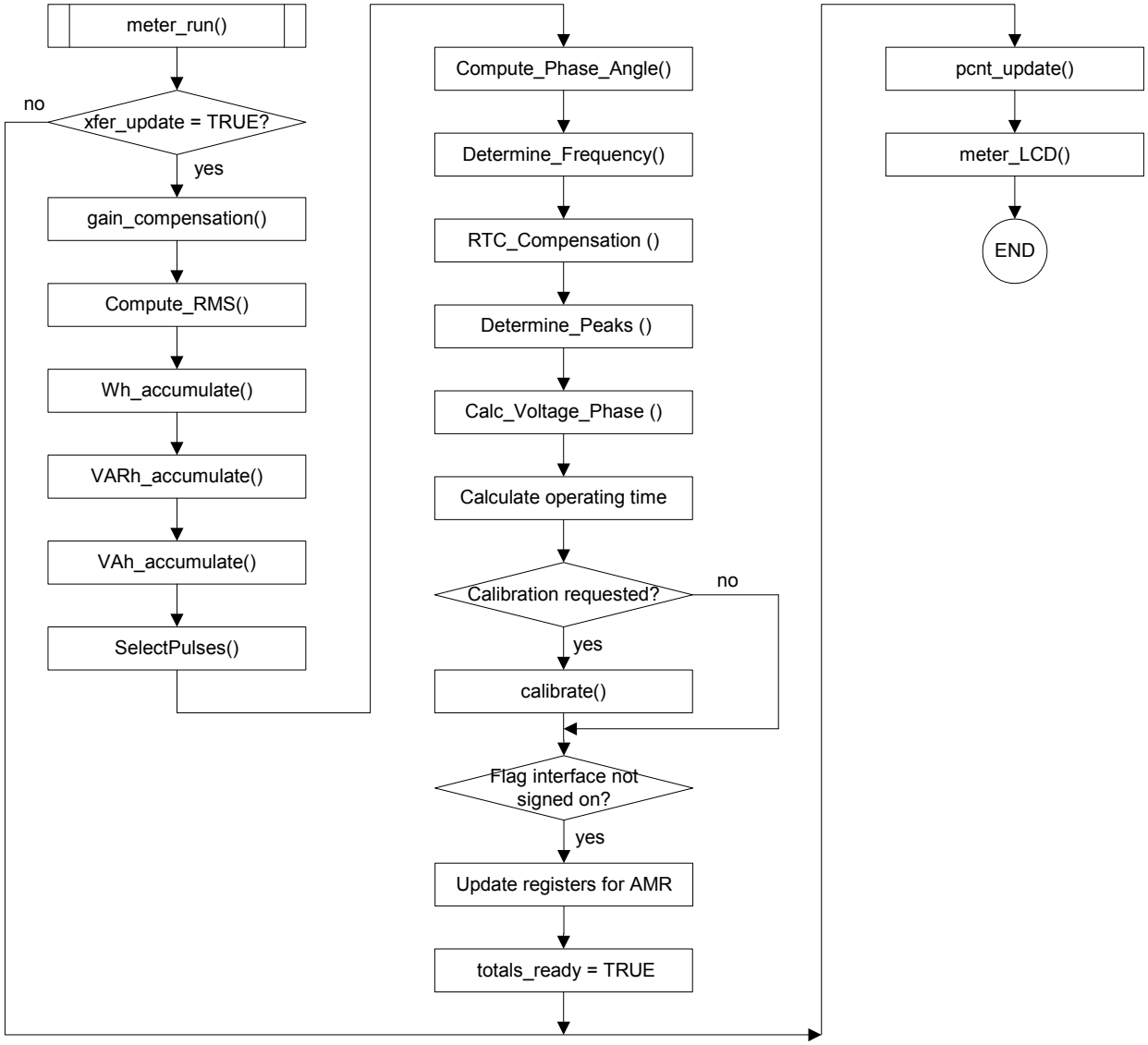


Figure 5-11: ce_update

meter_LCD

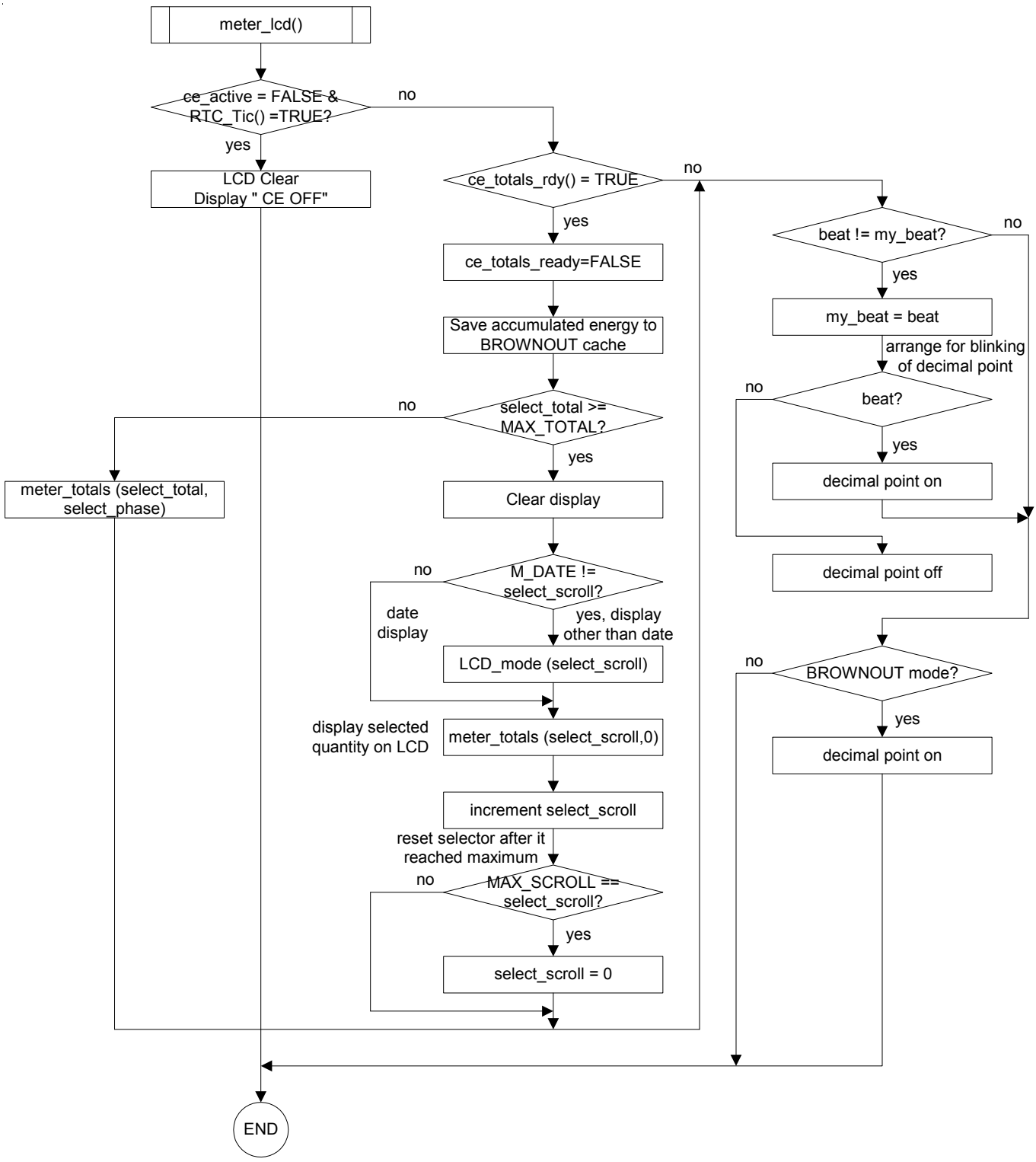


Figure 5-12: meter_LCD

Command Line Interpreter

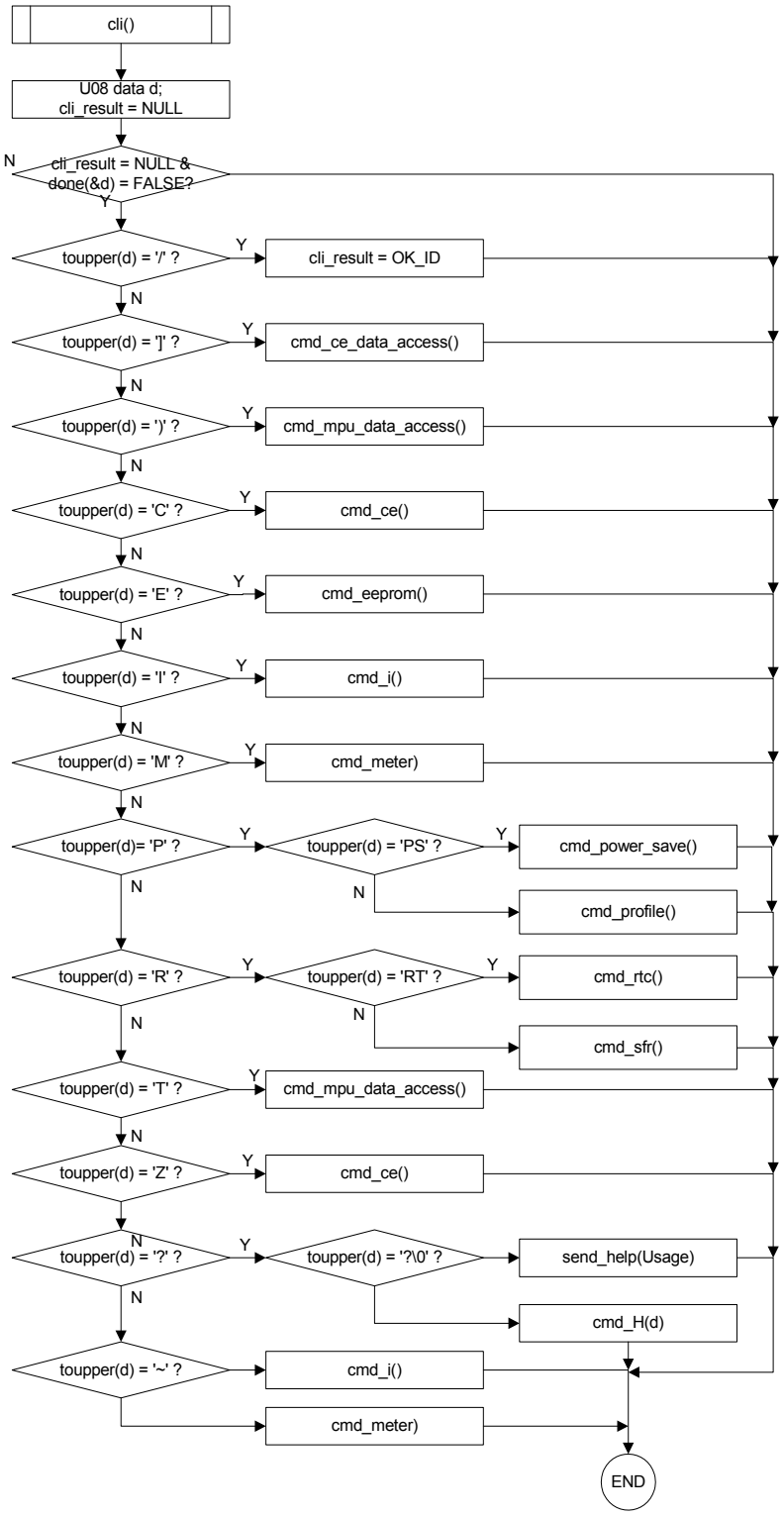


Figure 5-13: Command Line Interpreter

Auto-Calibration

The auto-calibration option (not compiled in the executable Demo Code) is a simplified calibration procedure based on voltage, real energy and reactive energy measurements.

Before the calibration starts, the desired accumulation time (SCAL) and the applied (ideal) voltage and current have to be entered by the user in the MPU memory locations VCAL and ICAL.

The procedure of this calibration method is the same as for the fast calibration procedure, as described in the DBUM: The tangens of the ratio of VARh and Wh determines the phase angle. The ratio between applied (ideal) and measured voltage determines the voltage gain. However, whereas the calibration spreadsheet uses extensive trigonometric functions, the auto-calibration procedure implemented in the Demo Code utilizes much simpler mathematical operations that are closer to the capabilities of the MPU.

As with the procedure presented in the DBUM, the target values should be applied to the meter and held constant during the auto-calibration process.

The routines shown in Figure 5-14 show how the auto-calibration is started. The cal_begin() routine starts a state-machine by setting the flag cal_flag to YES, after setting the calibration factors to default values, recording the calibration temperature, calculating the temperature compensation coefficients and setting the counter cs for calibration cycles.

The actual stabilization delay, measurement and adjustment phases are managed by separate routines that are activated by cal_flag being YES and controlled by the variable cs which counts down accumulation intervals.

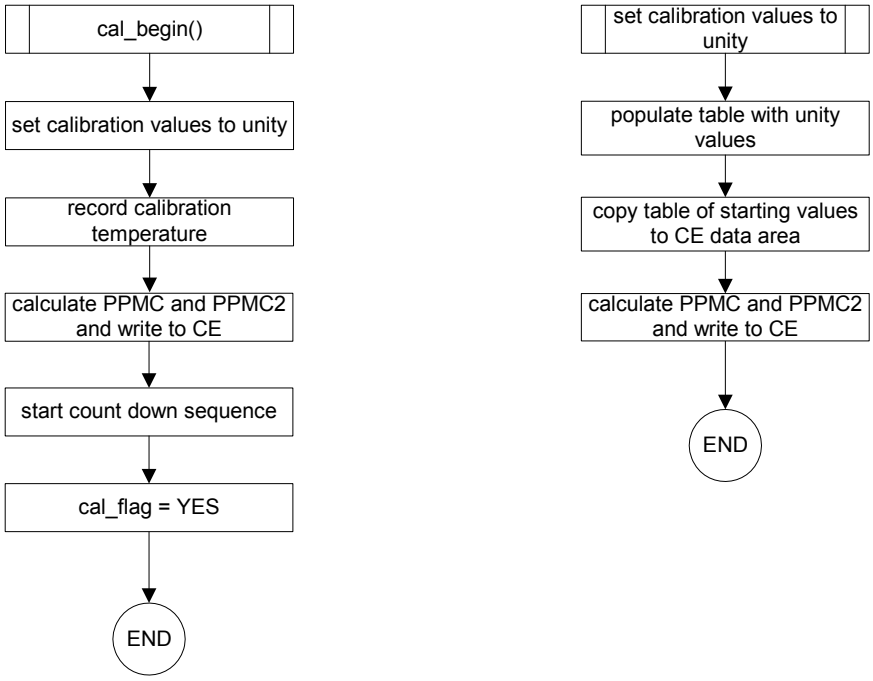


Figure 5-14: Auto-Calibration

The processing of the calibration steps is performed by the routine calibration(), which is called in ce_update() when new data becomes available, i.e. once per accumulation interval. The auto-calibration mechanism functions as a state-machine, sequenced by the variable "cs", which is used to count down accumulation intervals:

1) If cs > Scal: The state machine waits for the CE to settle after the unity gain and temperature compensation data are loaded in the routine cal_begin().

2) If cs = Scal: The variables for each cumulative voltage and current measurement are cleared.

3) If 0 <= cs <= Scal: For two accumulation intervals, prorated measurements of current and voltage are added to the variables. Using two accumulation intervals covers both chop polarities of temperature measurements.

4) If cs = 0: This signals the end of the calibration. Cumulative current and voltage measurements are then used to calculate and set the calibration coefficients for voltage and currents in CE DRAM.

See the source file calphased.c for details.

CE Default Calibration

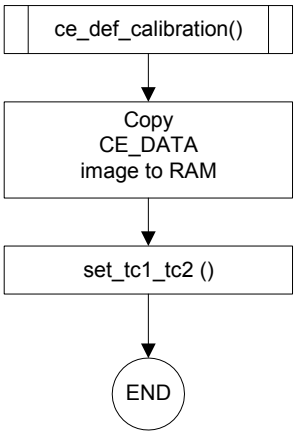


Figure 5-15: ce_default Calibration

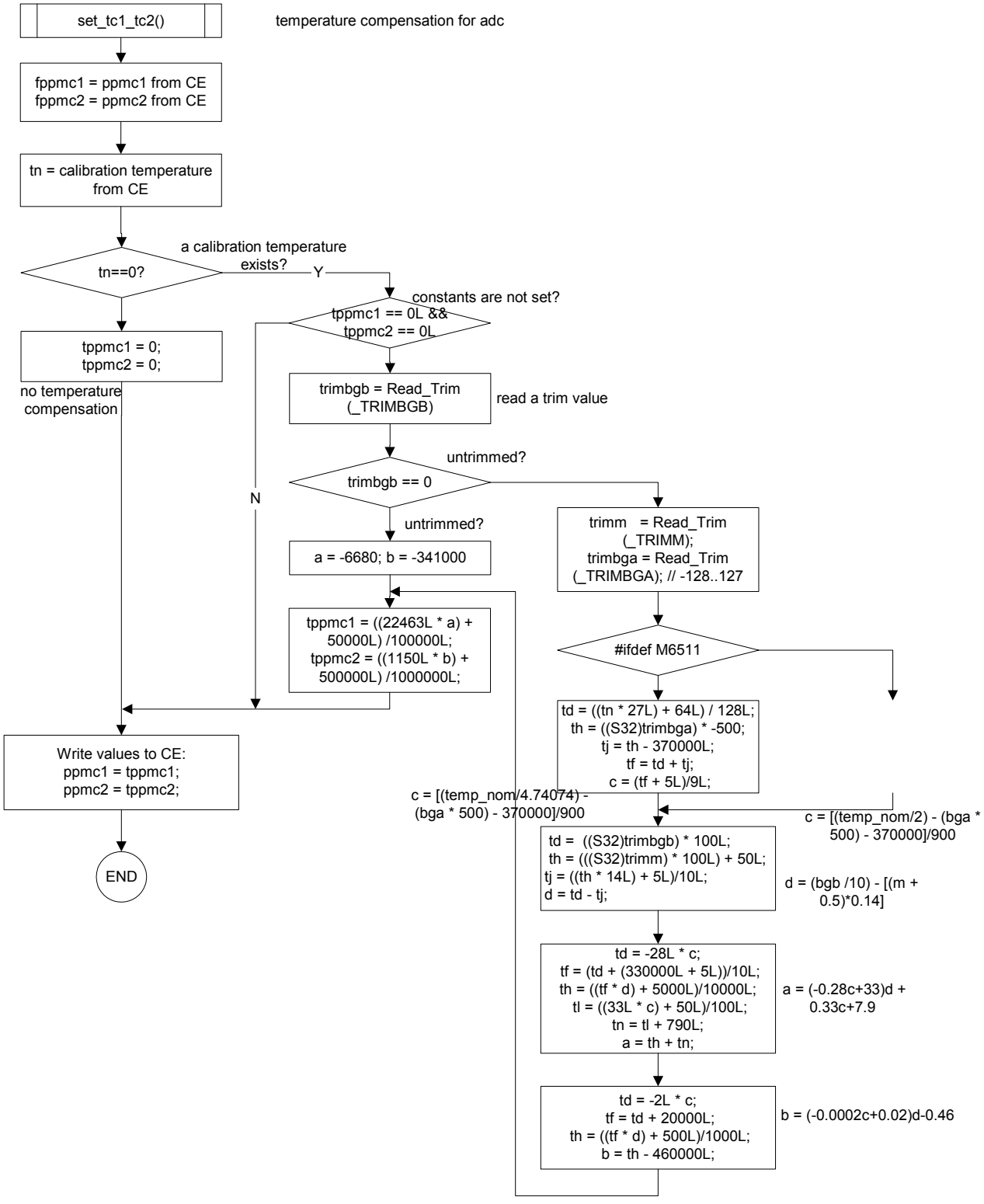


Figure 5-16: Calibration, continued

Command Pending

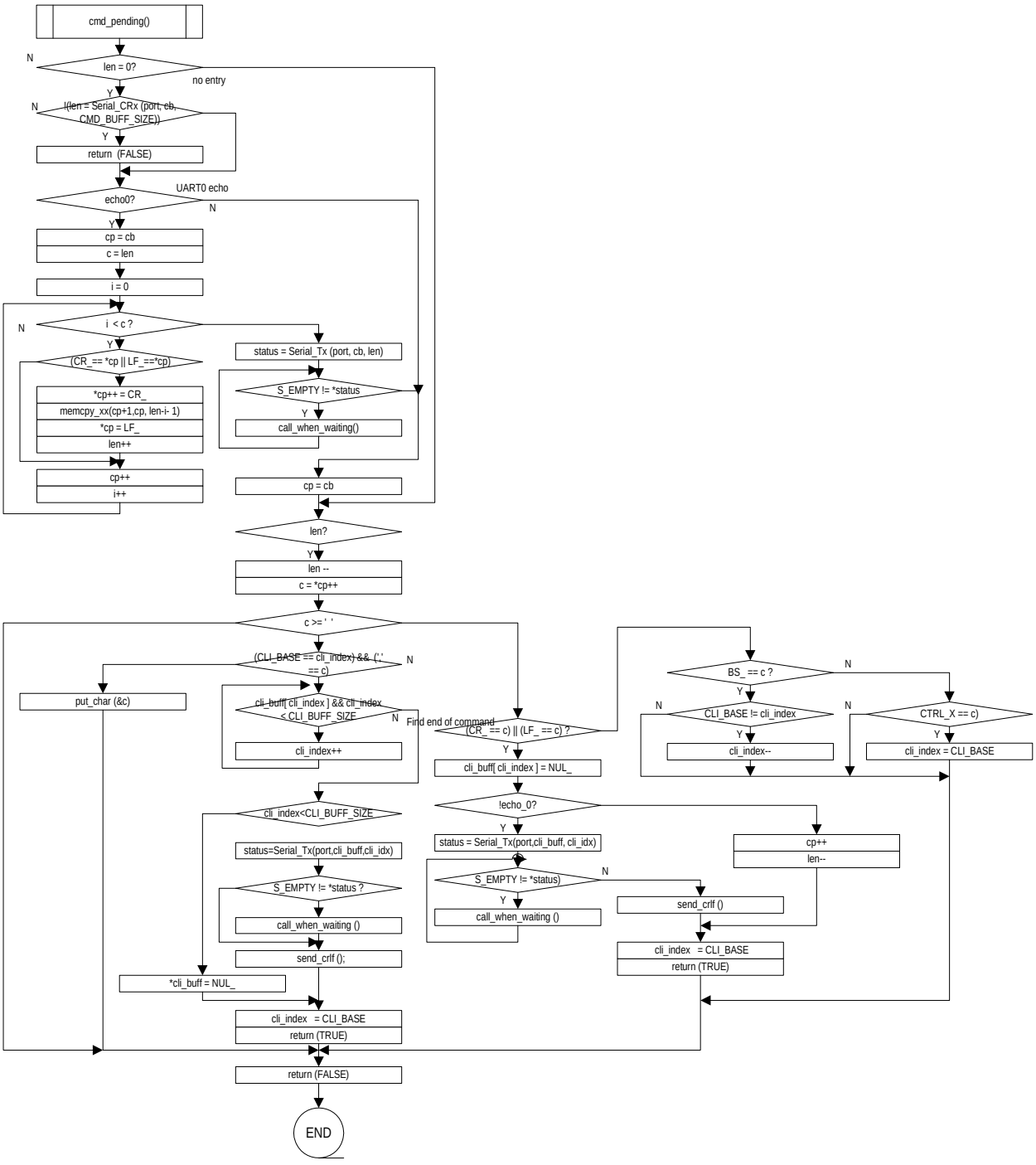


Figure 5-17: cmd_pending()

EEPROM Read/Write

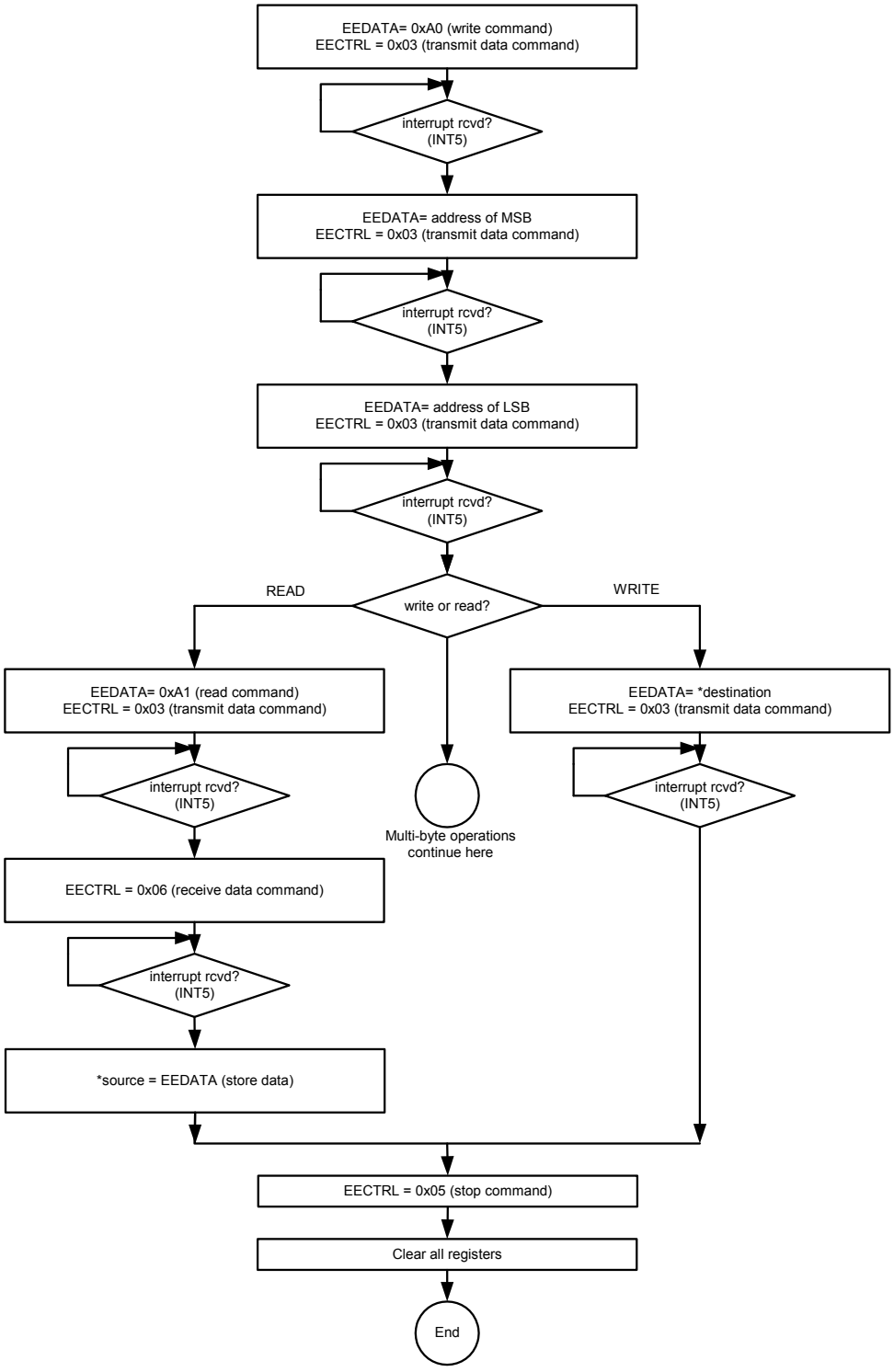


Figure 5-18: Single-Byte Read/Write

Registers and memory locations:

- EEDATA = SFR 0x9E
- EECTRL = SFR 0x9F
- *source = pointer to EEPROM address for read or write
- *destination = pointer to XRAM address
- count = byte count for multiple read/write

If the EEPROM interrupt service routine (INT5) returns the value 0x80 (illegal command), the loop should be exited, all registers should be refreshed and the operation should be restarted.

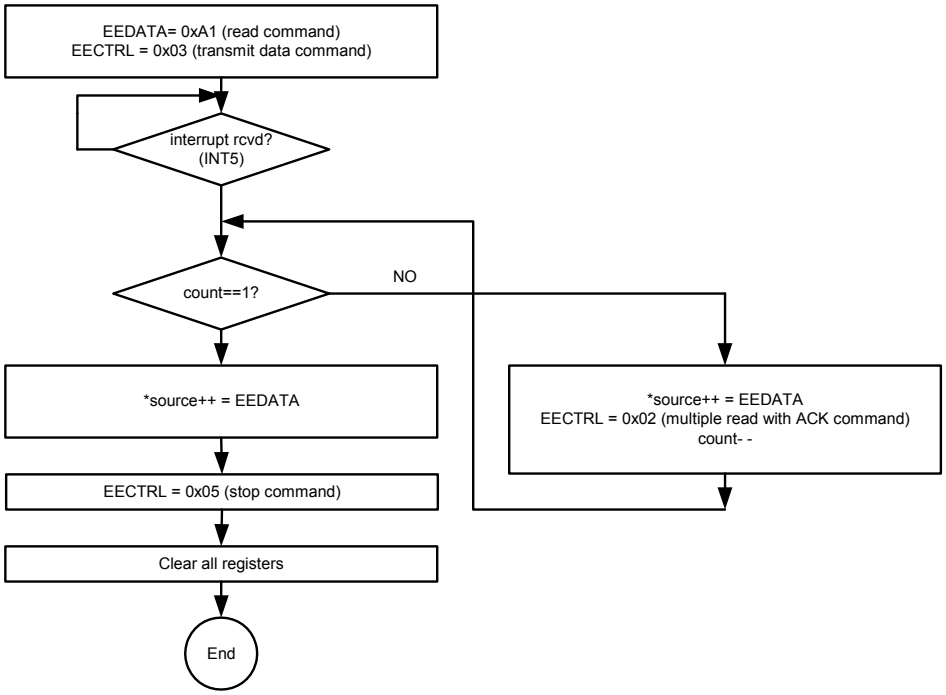


Figure 5-19: Multi-Byte Read

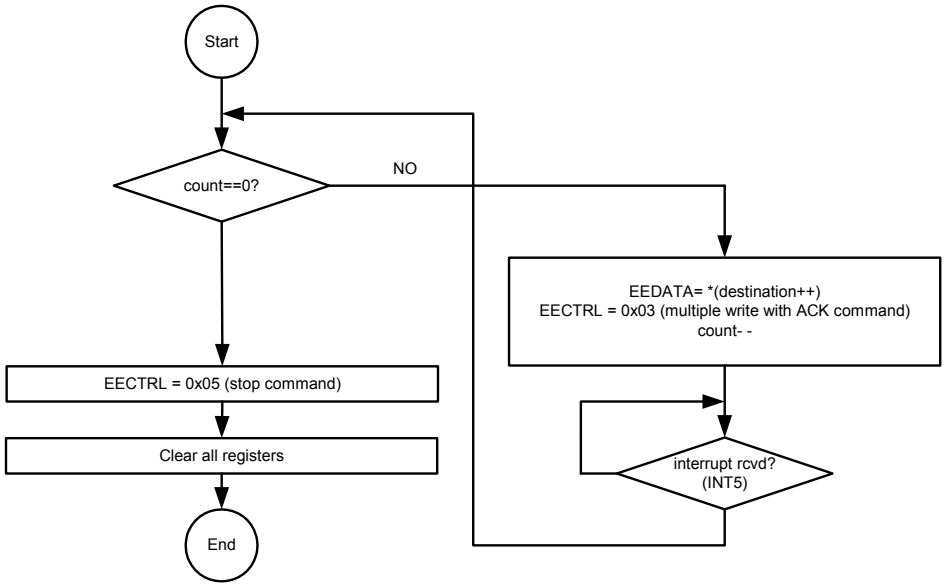


Figure 5-20: Multi-Byte Write

Notes:

- For larger EEPROMs, 1010xxR can be the first command (R=1 for read, R = 0 for write operation).
- The START command should be sent to the EEPROM before any read or write operation
- The algorithms cover single and multi-byte operations limited to a single page.
- EEPROMs are organized in pages. In general, ATMEL EEPROMs have 1Kbyte per page (256 x 32 bits). When reading, no special requirements with respect to page boundaries apply.
- Special precautions apply when a page boundary is crossed for write operations: When the end of a page is reached, the write.to the next page has to be preceded by a START command.
- EEPROMs typically respond to START commands with 5ms delay.

Battery Test

The battery test is based on sampling the voltage applied to the VBAT pin during an alternative multiplexer cycle. The function used for calculating the battery voltage from the count obtained from the ADC is int32_t mVBat (int32_t v).

In this function, the ADC sample count is shifted right 9 bits (to account for the left-shift operation automatically done by the ADC). The measured value is not very accurate, since the chip-to-chip variations in offset and LSB resolution are not calibrated (these may have 5% variations).

The routine battest_start() may be invoked from the command line interface. battest_start() sets the variable bat_sample_cnt to 2. This signals to the XFER_BUSY interrupt (in ce.c) to take two measurement (to account for the variations caused by the amplifier chopping). The RTC date is recorded in the structure last_day. That way, an automated battery test is run only once per day (when the date changes right after midnight).

The routine battest_run (void) is called from the part of meter_run() that only operates when the CE is active. This is because the battery test can only run when the CE is active. The routine battest_run (void) compares the current date with last_day. If it detects a difference, indicating that the date has just changed), it calls battest_start ().

Power Factor Measurement

The power-factor option (not compiled in the executable Demo Code) provides both instantaneous and accumulated (over fractions of an hour) display of power factor by phase. All power factor calculations are performed using floating point variables.

The power factor (PF = cosφ) calculation is based on the equations:

P = S * cosφ = S * PF

==> PF = P/S,

with P = real energy, S = apparent energy, PF = power factor

or VAh divided by Wh.

5.4.4 Watchdog Timer

The Demo Code revision 4.03 uses only the hardware watchdog timer provided by the 80515. This fixed-duration timer is controlled with SFR register WDI (0xE8).

The software watchdog timer is described in section 6.3.4, but should not be used. The hardware watchdog timer is more reliable since it cannot be accidentally disabled.

The hardware watchdog timer requires a refresh by the MPU firmware, i.e. bit 7 of WDI set, at least every 1.5 seconds. If this refresh does not occur, the hardware watchdog timer overflows, and the 80515 is reset as if RESETZ were pulled low. When overflow occurs, the bit *WD_OVF* is set in the configuration RAM. Using the *WD_OVF* bit, the MPU can determine whether a reset or a hardware watchdog timer overflow occurred. The *WD_OVF* bit is cleared when RESETZ is pulled low.

Note: The bits of the WDI register (SFR 0xE8) should not be individually set or reset. Instead, byte operations should be used.

The following macro code should be used for resetting (clearing) the watchdog, IE_RTC or IE_XFER bits:

```
#define WD_RST_      0xFF     // WatchDog bit.
#define IE_RTC_      0x02     // RTC ticked.
#define IE_XFER_     0x01     // XFER data available.

#define RESET_WD()       IFLAGS = WD_RST_;
#define CLR_IE_XFER()    IFLAGS = ~IE_XFER_ & 0x7F; // 0x7E
#define CLR_IE_RTC()     IFLAGS = ~IE_RTC_ & 0x7F; // 0x7D
```

5.4.5 Real-Time Clock (RTC)

The RTC is accessible through the I/O RAM (Configuration RAM) registers RTC_SEC through RTC_YR (addresses 0x2015 through 0x201B), as described in the data sheets.

Since the RTC runs on a much slower clock than the MPU, only one write operation can be performed per RTC clock cycle. This means that write operations to set the RTC must be separated by at least 396us. The sample code uses a software timer to perform this delay, so any code modification must make sure that hardware timer 1 is still useable for the RTC functions.

5.5 MANAGING MISSION AND BATTERY MODES

After a reset or power up, the processor must first decide what mode it is in and then take the appropriate action. It is useful to concentrate all activities related to power modes and reset into one centralized module. The Demo Code revision 4.7a does the switching of modes in the main() routine, based on decisions made in batmodes_20.c. Figure 5-21 shows the actions taken by the Demo Code and chip hardware after entering the main() routine. The code uses the following inputs and flags to determine which mode to enter:

- Battery mode enable jumper (see the DBUM for a detailed description of this input)
- PLL_OK flag
- RESET input
- PB input

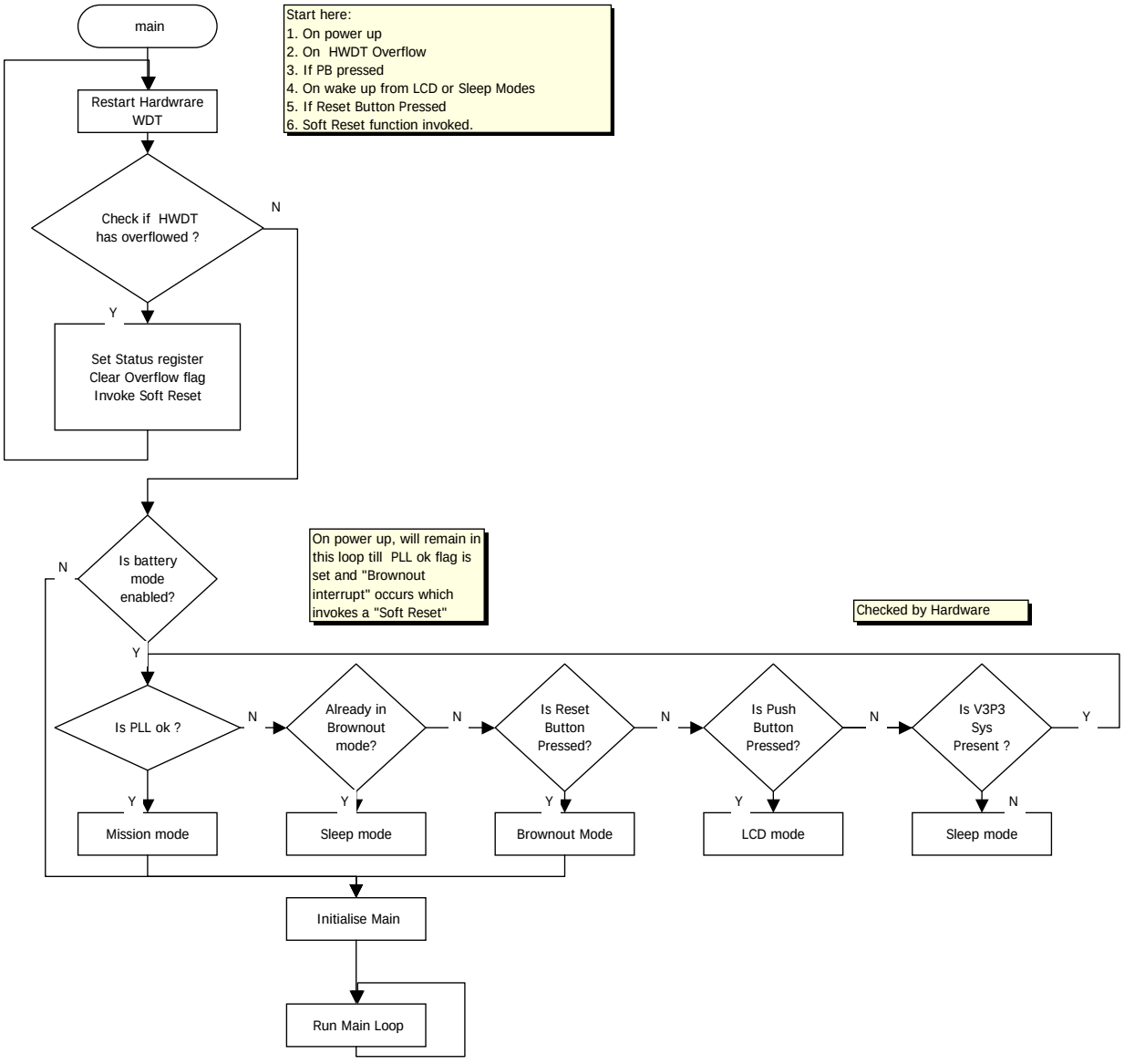


Figure 5-21: Power-Up Sequence

Precautions when adding a battery: When a battery or other DC supply is added to a Demo Board that is powered down, the 71M6521 Demo Code will cause the chip to enter Brownout mode and stay in Brownout mode. It is possible that the VBAT pins of the chip draws up to 1mA in this state, since the I/O pins are not initialized when Brownout mode is entered from a state where the chip is powered down (if Brownout mode is entered from Mission mode, the I/O pins are properly initialized, and the chip will enter Sleep mode automatically causing much lower supply current into the VBAT).

In general, to work in an operational meter (not a demo meter), the firmware has to be written to handle the case of connecting a battery to a powered-down board (since in a factory setting, batteries will most likely be added to meter boards that are powered down). The firmware must immediately enter sleep mode in this situation.

5.6 DATA FLOW

The ADC collects data from the electrical inputs on a cycle that repeats at 2520Hz. On each ADC cycle, the compute engine (CE) code digitally filters and adjusts the data using gain parameters (*CAL_Ix*, *CAL_Vx*) and phase adjustment parameters (*PHADJ_x*).

Normally, a calibration operation during manufacturing defines these adjustments and stores them in flash or EEPROM to be placed into CE memory by the MPU when the meter powers up. The Demo Code includes a basic linear self-calibration function that can typically reach 0.05% accuracy. (meter.c: meter_run(), calphased.c: cal_begin(), calibration()).

Better calibration schemes are possible. The calibration save and restore operations (cal_save() and cal_restore()) save and restore all adjustment variables, such as the constants for the real-time clock, not just the ones for electrical measurements.

On each ADC cycle, 2520 times per second, the CE performs the following tasks:

1. It calculates intermediate results for that set of samples.
2. It runs a debounced check for sagging mains, with a configurable debounce function.
3. It has three equally-spaced opportunities to pulse each pulse output.

On each ADC cycle, an MPU interrupt, "ce_busy" (see ce.c, ce_busyz_isr()) is generated. Normally, the interrupt service routine checks the CE's status word for the sag detection bits, and begins sag logic processing if a sag of the line voltage is detected.

In the event of a sag detection (announcing a momentary brownout condition or even a blackout), the cumulative quantities in memory are written to the EEPROM.

By the end of each accumulation interval, each second on the Demo Code, the CE performs the following tasks:

1. It calculates deviation from nominal calibration temperature (*TEMP_X*).
2. It calculates the frequency on a particular phase (*FREQ_X*).
4. It calculates watt hours (Wh) for each conductor, and the meter (*WxSUM_X*).
5. It calculates var hours (VARh) for each phase and the meter (*VARxSUM_X*).
6. It calculates summed squares of currents for each phase (*IxSQSUM_X*).
7. It calculates summed squares of voltages for each phase (*VxSQSUM_X*).
8. It counts zero crossings on the same phase as the frequency (*MAINEDGE_X*).

The CE code (see ce652x.c for a "C" image) digitally filters out the line frequency component of the signals, eliminating any long-term inaccuracy caused by heterodyning between the line frequency and the sampling or calculation rates. This also permits a meter to be used at 50 or 60Hz, or with inaccurate line frequencies.

Each metering equation has a CE code written for that calculation, so that the 6521 can calculate according to the most common metering methods.

Once per accumulation interval, the MPU requests the CE code to make an alternative measurement (alternate multiplexer cycle).

At the end of each accumulation interval, an MPU interrupt, the "xfer_interrupt" occurs (see ce.c, xfer_busy_isr()) occurs. This is the signal for the MPU to copy the above data to stable storage for further use.

At this time, the MPU performs creep detection (meter.c Apply_Creep()). If the measured voltage, current and/or power is below the minimum, no results for volts, current or watts are reported. If the voltage is below the threshold, no frequency or edge counts are reported. If the current is below the minimum, no current, Wh, VARh or VAh are reported.

The MPU's creep thresholds are configurable (VThrshld, IThrshld).

The MPU calculates human-readable values, and accumulates cumulative quantities (see meter.c, meter_run.ce.c, ce_update()). The MPU scales these values to the voltage and current sensors used on the PCB (see *VMAX* and *IMAX*).

Wh and VARh quantities are signed, permitting the MPU to perform net metering by assigning negative values to "export" and positive values to "import" (see meter.c. Wh.c, VAh.c and VARh.c.

Meters require more precision than standard C floating point provides. The Demo Code has reusable calculations for meter math (mmath.c). These automatically convert CE counts into a major running count of Wh, and a minor remainder of CE counts.

The MPU also places a scaled value into the CE RAM for each pulse output (meter.c, meter_run(), pulse_src.c, selectpulses()). This adjusts the pulse output frequency in such a way as to reflect that accumulation's contribution to the total pulse interval. Pulse intervals are cumulative, and cumulatively accurate, even though the frequency is updated only periodically.

Placing the pulse value selection logic into the MPU software means that any quantity from any phase or combination of phases can control either pulse output (see PulseSrcFunc[] for a list of transfer functions).

The MPU also performs temperature adjustments of the real-time clock (rtc_10.c, RTC_Trim(), RTC_Adjust_Trim()). The Demo Code can adjust the clock speed to a resolution of 1 part per billion, roughly one second per thirty years. The adjustments include offset (Y_CAL), temperature-linear (Y_CALC) and temperature-squared (Y_CALC2) parameters.

Once a human-readable quantity is available, it can be translated into a set of segments (meter.c, lcd.c) to display on the liquid crystal display, or read from a register in memory by means of the command-line interface (cli.c), or possibly some other serial protocol such as Flag (see flag.c) or NEMA.

5.7 CE/MPU INTERFACE

The interface between the CE and the MPU is described completely in the 71M6521 Data Sheet.

5.8 BOOT LOADER

It is possible to implement code that functions as a boot loader. This feature is useful for field updates and various test scenarios.

See the TERIDIAN Application Note number 031 for details.

5.9 SOURCE FILES

The functionality of the Demo Code is implemented in the following files and directories:

1. **CLI: Command Line Interface – General Commands**

access.c	SFR, I/O RAM, MPU and CE memory access routines
access_x.c	extended memory access routines
c_serial.c	parser for command line interface
cli.c	command line interface routines
cmd_ce.c	sub-parser for CE commands
cmd_misc.c	sub-parser for RTC, EEPROM, trim and PS commands
help.c	display of help text
io.c	number conversion functions and auxiliary routines for CLI
load.c	upload and download
profile.c	data collection for support of profile command
ser0cli.c	
ser1cli.c	
sercli.c	buffer serial I/O for the CLI

When compiled without the on-line help option (help.c), CLI.C takes about 14Kbytes of program space. Adding the on-line help will use another 5Kbytes. When designing a real meter, CLI.C can easily be removed without major changes to the software.

2.	**FLAG**	**Basic FLAG AMR Protocol**
	flag0.c	implements a basic FLAG AMR protocol for SER0
	flag1.c	implements a basic FLAG AMR protocol for SER1
	flag.c	code shared shared by flag0.c and flag1.c
3.	**IO:**	**Input/Output**
	cal_ldr.c	load routines for calibration factors
	eep24C08.c	routines supporting the 24C08 EEPROM
	eeprom.c	interrupt-driven serial EEPROM routines
	eepromp.c	high-speed polling EEPROM routines
	eepromp3.c	polling interface for µWire EEPROM
	iiceep.c	I2C bus interface using the chip's I2C hardware
	iolite.c	IO subroutines for use by the calibration loader (cal_ldr.c)
	lcd.c	initialization, configuration, read and write routines for LCDs
	lcd_VIM808.c	routines for driving Varitronix VIM-808 LCS
	rtc.c	RTC read, write, reset, and trim routines
	ser.c	baud rate table shared by ser0.c and ser1.c
	ser0.c	initialization, configuration, interrupt, read and write routines for SER0
	ser1.c	initialization, configuration, interrupt, read and write routines for SER1
	serial.c	legacy code that implements a fully buffered interrupt-driven serial driver
	tmr0.c	initialization, configuration, interrupt, read and write routines for TMR0
	tmr1.c	initialization, configuration, interrupt, read and write routines for TMR1
	uwrdio.c	3-wire interface using direct control of DIO4 and DIO5. It can be adapted to nonstandard clock polarities and edges, 4-wire SPI EEPROMs, and TSC chips other than the 71M6521 (see comments in the source file)
	uwreep.c	a 3-wire interface using the high-speed 3-wire interface hardware of the 71M6521
4.	**Main:**	**Main top-level tasks, 6521-specific**
	batmodes_20.c	battery mode logic
	defaults.c	contains the table of start-up default values
	main.c	main() with startup sequence and main task switch
	main.c	initialization and main loop
5.	**Meter:**	**Metering Functions**
	calphased.c	auto-calibration
	ce.c	initialization, configuration, interrupt, read and write routines for the compute engine
	ce652X.c	data exchange between CE data RAM and XRAM
	error.c	error recording and logging
	freq.c	routines to calculate and display frequency
	io652X.c	control of analog front end, multiplexer, RTM, I/O pins
	meter.c	contains overall meter logic to calculate and display meter data
	misc.c	unused legacy code for managing interrupts and priorities
	pcnt.c	code for counting output pulses
	peak_alerts.c	detects out-of-range line values
	phase_angle.c	calculates and displays voltage-to-current phase angles
	psoft.c	generates two additional pulse outputs using DIO pins
	pulse_src.c	directs line measurements to any pulse output
	pwrfct.c	routines for calculating the power factor
	rms.c	calculates and displays Vrms and Irms
	vah.c	calculates VAh
	varh.c	calculates VARh
	vphase.c	calculates voltage-to-voltage phase angles for multiphase meters
	wh.c	calculates Wh
6.	**UnitTest:**	**Test and Verification (not Shipped with standard Demo Code)**
	eepromtest.c	basic test of the EEPROM driver

flag0test.c	a test for a FLAG system
modetest.c	a simple test of the 6520's battery modes
ser0test.c	tests the serial driver for SER0
ser1test.c	tests the serial driver for SER1
stmtest.c	tests the software timers
tmr0test.c	tests the driver for TMR0
tmr1test.c	tests the driver for TMR1

7. **Util:** **Utilities**

dead.c	defines unused flash space for the boot loader
dio.h	defines high-level access to DIO pins
flash.c	flash memory read, write, erase, compare and checksum calculation
irq.c	securely disables and enables interrupts
library.c	routines for memory copy, compare, CRC calculation, string length
math.c	contains routines for multiple-precision math
oscope.h	a utility to trigger oscilloscope loops using DIO7
priority.h	header file defining priorities for IP0 and IP1
sfrs.c	access to SFRs
startup.a51	startup assembly code
startup_boot.a51	
startup_boot_secure.a51	
startup_secure.a51	
stm.c	software timer routines
timers.c	unused software timer legacy code
wd.c	routines that support the hardware watchdog

5.10 AUXILIARY FILES

A variety of startup files is provided with the Demo Kits. The function of these files is as follows:

1. STARTUP.A51:
   This file provides memory and stack initialization. It is part of the Keil compiler package.
2. STARTUP_SECURE.A51:
   This file is almost identical to STARTUP.A51. The only difference is that this variation sets the *SECURE* bit. This bit enables security provisions that prevent external reading of flash memory and CE program memory. The code segment below sets the security bit located at SFR register address 0xB2:

```
STARTUP1:
    CLR   0xA8^7      ; Disable interrupts
    MOV   0B2h,#40h   ; Set security bit.
    MOV   0E8h,#0FFh  ; Refresh nonmaskable watchdog
```

3. INIT.A51:
   A secondary startup file. It is part of the Keil compiler package. This code is executed, if the application program contains initialized variables at file level.
4. STARTUP_BOOT.A51:
   This startup file is to be used when the code is to be compiled as a bootloader.

5.11 INCLUDE/HEADER FILES

In line with common industry practice, each C file in the Demo Code source code has a corresponding header file that ends in .H and that provides the interface to the C file's code. A number of include files are special cases, and provide global data or hardware definitions.

- Main_6521B\options.h selects the features used by the code that is less than 8K
- Main_6521D\options.h selects the features used by the code that is less than 16K
- Main_6521_CLI\options.h selects the features used by the code that is less than 32K
- main\option_gbl.h defines global configuration values used in all meter versions.
- meter\meter.h defines the meter's configuration and power registers.
- meter\ce652x.h defines the CE memory used to communicate with the MPU.
- meter\io652x.h defines the memory-mapped registers of the 652x chips.
- util\reg652x.h defines the special function registers of the 652x chips.
- util\stdint.h defines `a standard integer package for TSC meter chips using 8051s.

5.11.1 OPTIONS.H

The file OPTIONS.H is especially important because it controls entire features in a firmware build. When an option is 1, it means that the feature is to be compiled and linked into the build. The idea is that by adding or subtracting features, a customer can quickly tune the Demo Code to approximate the desired meter configuration. If the comments in OPTIONS.H are not clear, feel free to use grep, or another code-searching tool to locate where the flags occur in the code. While TERIDIAN has made a good-faith effort to test representative combinations of compile flags, there are too many combinations to test exhaustively.

When OPTIONS.H is changed, there are three usual results. Either the build complains that it needs some subroutines, or it complains that it has too many subroutines, or it is good. When it needs subroutines, enable the option flags for the needed subroutines. When it has too many subroutines, try to disable the option flags for the unneeded subroutines.

If the resulting build is too big to fit the available program memory, then more features must be disabled.

Usually, the option flags are tested either right after options.h is included in a file, or around the subroutines.

5.11.2 Register Definitions

Register definitions can be found in the following files:

- REG80515.H - Register definition for the 80515 MPU core
- REG652X.H - Register definition of 652X SFRs and I/Os
- IO652X.H and IO6512X.C - I/O RAM register definitions
- CE652X.H and CE652X.C - CE data and structure declarations

5.11.3 Other Include/Header Files

Other Include/Header files are:

- CLI.H - Result code and Common ASCII code definition used for CLI
- HELP.H - HELP message prototype declarations
- IO.H – I/O subroutines for CLI
- SER0CLI.H, SER1CLI.H – hardware access layer for UART0/UART1
- SERCLI.H – include definitions for UART 0/1 debug routines
- FLAG0.H, FLAG1.H, FLAG.H – shared logic for all FLAG interfaces
- EEPROM.H – EEPROM
- II2.H – I2C Interface
- LCD.H – LCD
- RTC.H – Real-Time clock
- SER0.H, SER1.H, SER.H – serial interface
- SERIAL.H – serial interface API prototypes and definitions
- TMR0.H, TMR1.H – timer routines
- UWR.H – microwire (µwire), or three-wire interface
- BATMODES.H – battery modes (BROWNLOUT, LCD, SLEEP)
- DEFAULTS.H – default values
- OPTIONS_GBL.H – global compile-time options
- OPTIONS.H – general compile-timeoptions, defining meter functionality
- CALIBRATION.H – calibration
- CE.H – compute engine interface includes
- FREQ.H – frequency and main-edge count
- METER.H – meter structures, enumerates and definitions
- PCNT.H – pulse counting
- PEAK_ALERTS.H – voltage/current peak alerts
- PHASE_ANGLE.H – phase angle calculation
- PSOFT.H – pulse generation by MPU software (external pulse generation)
- PULSE_SRC.H – pulse source definitions and support
- RMS.H – RMS calculation
- VAH.H – VAh accumulation
- VARH.H – VARh accumulation
- WH.H – Wh accumulation
- DIO.H – DIO structures, enumerations and definitions
- FLASH.H – flash copy and CRC routines
- IRQ.H – interrupt kernel
- LIBRARY.H – library routines
- MATH.H – meter math library
- PRIORITY.H – interrupt masks and priority definitions
- SERIAL.H – serial interface structures, enumerates and definitions
- SFRS.H – low-level API for SFRs and memory
- STDINT.H – standard integer definitions
- STM.H – software timer definitions
- WD.H – watchdog bit definitions

5.12 CE IMAGE FILES

The CE code uses pre-designed, pre-validated algorithms and calculations, which are accurate to the noise floor of the integrated circuit, saving substantial engineering and development time.

The source code for the CE is proprietary. Only the code and data images (binary images) are available to the user. The code image must be merged with the MPU code residing in flash memory.

Teridian provides two files for each ce code. One file is the code for the compute0engine. The other is a set of data to copy into the CE's RAM area, to initialize the CE program variables.

Teridian has two standard CE codes for the 6521:

- For one-element two-wire single phase meters, use ce21a04_ce.c, and ce21a04_dat.c (the CE code and data files). When the hardware field *EQU* is 0x00, this CE code provides two metering elements using VA and IA and VA and IB. The equations are WhA = (VA * IA) and WhB = (VA * IB).

  There are two metering elements so that the neutral current can be measured for tamper detection. The MPU software must decide which element is more accurate at any given time. This gives a perfectly flexible method for detecting and mitigating tampering.

  For one-element three-wire split phase meters, also use ce21a04_ce.c, and ce21a04_dat.c (the CE code and data files). When the hardware field EQU is 0x01, this ce code provides two metering elements. One has an equation of Wh = VA (IA - IB)/2. The other has WhB = VA * IB.
  There are two metering elements so that the second element can be used to reduce the calibration steps during meter manufacturing. The second element provides the data needed to calibrate the meter with both current sensors operating at the same time. This permits more accurate, realistic calibrations.

- For two-element three-wire delta meters, or dual non-tamper-detecting single-phase meters, use CE21A03_CE.C and CE21A03_DAT.C. When the hardware field *EQU* is 02, this CE code provides two metering elements. One has an equation of WhA = VA * IA. The other is WhB = VB * IB. This measures two legs of a delta configuration, and therefore, by Blondel's theorem, when these are added together, the total delivered power is metered. Since the two channels are independent, they can also be used as two single-phase meters, even if the meters are operating on different phases.

Teridian has a number of other CE codes for other sensors and needs, including code for tamper-resistant meters.

Images of the CE data and program code are provided with the Demo Kits. They are to be linked into the object code. CE images are provided by the following files:

1. CE21B_CE.C:
   This file provides the image of the 6521 CE program in C notation.
2. CE21C_DAT.C:
   This file provides the image of the 6521 CE default data in C notation.

5.13 COMMON MPU ADDRESSES

In the Demo Code, certain MPU XRAM parameters have been given fixed addresses in order to permit easy external access. These variables can be read via the command line interface (if available), with the **)n$** command and written with the **)n=xx** command where n is the word address. Note that accumulation variables are 64 bits long and are accessed with **)n$$** (read) and **)n=hh=ll** (write) in the case of accumulation variables.

Name	Purpose	Function or LSB Value	CLI	Format	L in bits	XDATA
IThrshldA	Starting current, element A	$LSB = 2^{16}\sqrt{I0SQSUM}$ 0 in this position disables creep logic for both element A and B.	)0	unsigned	32	0x0000
Config	Configure meter operation on the fly.	bit 0:** reserved 0: VA = Vrms * Irms; 1: $VA = \sqrt{Wh^2 + VARh^2}$ bit1:* 1 = Clears accumulators bit2:*1 = Calibration mode bit3:** reserved: 1 = enable tamper detection	)1	N/A	8	0x0004
VPThrshld	error if exceeded.*	$LSB = 2^{16}\sqrt{V0SQSUM}$	)2	unsigned	32	0x0005
IPThrshld	error if exceeded.*	$LSB = 2^{16}\sqrt{I0SQSUM}$	)3	unsigned	32	0x0009
Y_Cal_Deg0	RTC adjust	100ppb	)4	signed	16	0x000D
Y_Cal_Deg1	RTC adjust, linear by temp.*	10ppb*ΔT, in 0.1?C	)5	signed	16	0x000F
Y_Cal_Deg2	RTC adjust, squared by temp.*	1ppb*ΔT^2, in 0.1?C	)6	signed	16	0x0011
PulseWSource PulseVSource	Wh Pulse source, VARh pulse source selection*	See table for PulseWSource and PulseVSource	)7)8	unsigned	8	0x0013 0x0014
Vmax	Scaling Maximum Voltage for PCB, equivalent to 176mV at the VA/VB pins	0.1V	)9	unsigned	16	0x0015
ImaxA	Scaling maximum current for PCB, element A, equivalent to 176mV at the IA pin	0.1A	)A	unsigned	16	0x0017
ppmc1	ADC linear adjust with temperature	PPM per degree centigrade	)B	signed	16	0x0019
ppmc2	ADC quadratic adjust with temperature	PPM per degree centigrade squared	)C	signed	16	0x001B
Pulse 3 source	Source for software pulse output 3**	See table for PulseWSource and PulseVSource	)D	unsigned	8	0x001D
Pulse 4 source	Source for software pulse output 4**	See table for PulseWSource and PulseVSource	)E	unsigned	8	0x001E
Scal	Duration for auto-calibration** in seconds	Count of accumulation intervals to be used for auto-calibration.	)F	unsigned	16	0x001F
Vcal	Voltage value to be used for auto-calibration**	Nominal RMS voltage applied to all elements during auto-calibration (LSB = 0.1V).	)10	unsigned	16	0x0021

Name	Purpose	Function or LSB Value	CLI	Format	L in bits	XDATA
Ical	Current value to be used for autocalibration**	Nominal RMS current applied to all elements during auto-calibration (LSB = 0.1V). Power factor must be 1.	)11	unsigned	16	0x0023
VThrshld	Voltage at which to measure frequency, zero crossing, etc.	$LSB = 2^{16}\sqrt{V0SQSUM}$ This feature is approximated using the CE's sag detection.)	)12	unsigned	16	0x0025
PulseWidth	Maximum time pulse is on.	t = (2*PulseWidth + 1)*397µs, 0xFF disables this feature. Takes effect only at start-up.	)13	signed	16	0x0029
temp_nom	Nominal temperature, the temperature at which calibration occurs.	Units of TEMP_RAW, from CE. The value read from the CE must be entered at this address.	)14	unsigned	32	0x002B
ImaxB	Scaling maximum current for PCB element B, equivalent to 176mV at the IA pin	0.1A	)15	unsigned	16	0x002F
IThrshldB	Starting current, element B	$2^{16}\sqrt{I1SQSUM}$	)16	unsigned	32	0x0031
VBatMin*	Minimum battery voltage.	Same as VBAT, below	)17	unsigned	32	0x0035
CalCount	Count of calibrations	Counts the number of times calibration is saved, to a maximum of 255	)18	unsigned	8	0x0039
RTC copy	Nonvolatile copy of the most recent time the RTC was read.	Sec, Min, Hr, Day, Date, Month, Year	)19 1A 1B 1C 1D 1E 1F	unsigned	8 8 8 8 8 8 8	0x163
deltaT	Difference between raw temperature and temp_nom	Same units as TEMP_RAW	)20	signed	32	0x003B
Frequency*	Frequency	Units from CE.	)21	unsigned	32	0x003F
VBAT*	Last measured battery voltage*	$VBAT = \frac{n_{ADC}}{2^9}$ ADC counts, logically shifted right by 9 bits. Note: battery voltage is measured once per day, except when it is being displayed or requested with the BT command.	)22	unsigned	32	0x0043

Name	Purpose	Function or LSB Value	CLI	Format	L in bits	XDATA
Vrms_A	Vrms, element A	$2^{16}\sqrt{V0SQSUM}$	)23	unsigned	32	0x004B
Irms_A	Irms, element A	$2^{16}\sqrt{I0SQSUM}$	)24	unsigned	32	0x004F
Vrms_B	Vrms, element B**,†	$2^{16}\sqrt{V1SQSUM}$	)25	unsigned	32	0x0053
Irms_B	Irms, element B	$2^{16}\sqrt{I1SQSUM}$	)27	unsigned	32	0x0057
STATUS	Status of meter	See table for STATUS register	)2A	unsigned	32	0x0063
CAI	Count of accumulation intervals since reset, or last clear.	count	)28	signed	32	0x0067
Whi**	Imported Wh, all elements.	LSB of W0SUM	)2C	signed	64	0x006B
Whi_A**	Imported Wh, element A	"	)2E	signed	64	0x0073
Whi_B**	Imported Wh, element B	"	)30	signed	64	0x007B
VARhi*	Imported VARh, all elements.	LSB of W0SUM	)34	signed	64	0x008B
VARhi_A*	Imported VARh, element A	"	)36	signed	64	0x0093
VARhi_B*	Imported VARh, element B	"	)38	signed	64	0x009B
VAh**	VAh, all elements.	LSB of W0SUM	)3C	signed	64	0x00AB
VAh_A**	VAh, element A	"	)3E	signed	64	0x00B3
VAh_B**	VAh, element B	"	)40	signed	64	0x00BB
Whe**	Exported Wh, all elements.	LSB of W0SUM	)44	signed	64	0x00CB
Whe_A**	Exported Wh, element A	"	)46	signed	64	0x00D3
Whe_B**	Exported Wh, element B	"	)48	signed	64	0x00DB
VARhe**	Exported VARh, all elements.	LSB of W0SUM	)4C	signed	64	0x00EB
VARhe_A**	Exported VARh, element A	"	)4E	signed	64	0x00F3
VARhe_B**	Exported VARh, element B	"	)50	signed	64	0x00FB

Name	Purpose	Function or LSB Value	CLI	Format	L in bits	XDATA
Whn	Net metered Wh, all elements A,	LSB of W0SUM	)54	signed	64	0x010B
Whn_A*	Net metered Wh, element A, for autocalibration	LSB of W0SUM	)56	signed	64	0x0113
Whn_B*	Net metered Wh, element B, for autocalibration	"	)58	signed	64	0x011B
VARhn*	Net metered VARh, all elements	LSB of w0sum	)5C	signed	64	0x012B
VARhn_A*	Net metered VARh, element A, for auto-calibration	LSB of w0sum	)5E	signed	64	0x0133
VARhn_B*	Net metered VARh, element B, for auto-calibration	"	)60	signed	64	0x013B
MainEdgeCnt	Count of voltage zero crossings	count	)64	unsigned	32	0x014B
Wh	Default sum of Wh, nonvolatile	LSB of w0sum	)65	signed	64	0x014F
Wh_A	Wh, element A, nonvolatile	"	)67	signed	64	0x0157
Wh_B	Wh, element B, nonvolatile	"	)69	signed	64	0x015F
StatusNV	Nonvolatile status	See Status	)6D	n/a	32	0x016F

*M6521F (32K) only; compilation option in M6521D (16K)
**Compilation option (available Demo Code), variable present but not in use.
† Requires features not in standard demo PCB

Table 5-12: MPU Memory Location

5.14 FIRMWARE APPLICATION INFORMATION

5.14.1 Sag Detection

A sag is defined as an undervoltage condition that persists for more than one period. A shorter undervoltage condition is called a dip (see Figure Figure 5-22). The occurrence of sags can announce an impending loss of power. Since accumulated energy values etc. in the meter will have to be saved to non-volatile memory in the case of loss of power, a sag can be used to initiate data saving operations. Some applications may instead save or count the sag event for the purpose of recording power quality data.

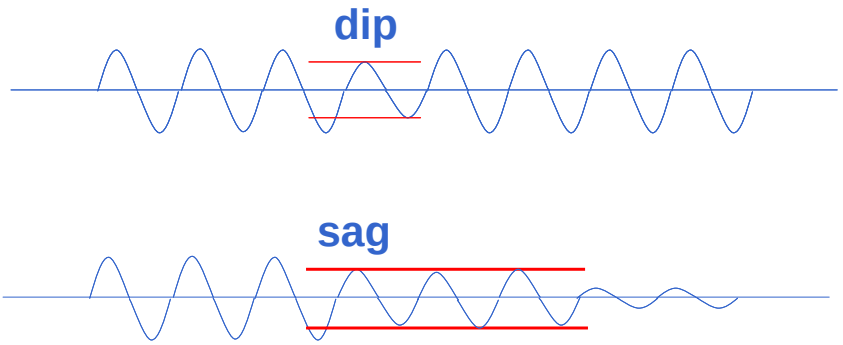


Figure 5-22: Sag and Dip Conditions

Sag detection is performed by the CE, based on the CE DRAM registers SAG_THR and SAG_CNT. SAG_THR defines the threshold which the input voltage has to be continuously below, and SAG_CNT defines the number of samples required to trigger the sag bit (see Figure 5-23).

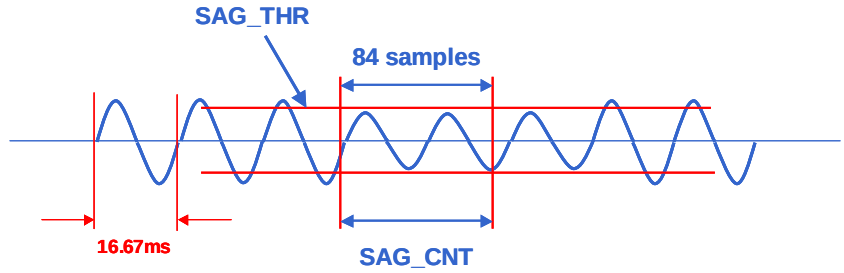


Figure 5-23: Sag Event

When the CE detects a sag that meets the sag conditions specified in SAG_THR and SAG_CNT on one of the input voltage channels, it will reflect this in the corresponding bit (SAG for single-phase, or SAG_A, SAG_B, SAG_C for poly-phase) of the CE STATUS Word. See the CE Interface section in the 652X Data Sheet for details.

It is up to the MPU firmware to decide what is to be done in case a sag is detected. The Demo Code does not have any provisions for actions due to sags detected by the CE.

5.14.2 Temperature Measurement

The temperature output of the on-chip temperature sensor (*TEMP_RAW*) is provided by the CE in CE DRAM location 0x7B. The relative chip temperature *deltaT* (MPU location 0x20) is derived by subtracting the raw temperature from the nominal temperature (*TEMP_NOM*) and multiplying it with a constant factor. Thus, once the raw temperature obtained at a known environmental temperature is stored in *TEMP_NOM*, deltaT will always reflect the deviation from nominal temperature. The scaling is in tenths of Centigrades, i.e. a reading of 75 means that the measured temperature is 7.5°C higher than the reference temperature.

5.14.3 Temperature Compensation for Measurements

The internal voltage reference of the 652X ICs is calibrated during device manufacture. Trim data is stored in on-chip fuses. The temperature coefficients TC1 and TC2 are given as constants that represent typical component behavior.

The bandgap temperature is provided to the embedded MPU, which then may digitally compensate the power outputs. This permits a system-wide temperature correction over the entire system rather than local to the chip. The incorporated thermal coefficients may include the current sensors, the voltage sensors, and other influences. Since the band gap is chopper stabilized via the *CHOP_EN* bits, the most significant long-term drift mechanism in the voltage reference is removed.

The CE applies the gain supplied by the MPU in *GAIN_ADJ*. This external type of compensation enables the MPU to control the CE gain based on any variable, and when *EXT_TEMP* = 15, GAIN_ADJ is an input to the CE.

5.14.4 Temperature Compensation for the RTC

The flexibility provided by the MPU allows for compensation of the RTC using the substrate temperature. To achieve this, the crystal has to be characterized over temperature and the three coefficients *Y_CAL*, *Y_CALC*, and *Y_CAL_C2* have to be calculated. Provided the IC substrate temperatures tracks the crystal temperature the coefficients can be used in the MPU firmware to trigger occasional corrections of the RTC seconds count, using the *RTC_DEC_SEC* or *RTC_INC_SEC* registers in I/O RAM.

Example: Let us assume a crystal characterized by the measurements shown in Table 5-13.

Deviation from Nominal Temperature [°C]	Measured Frequency [Hz]	Deviation from Nominal Frequency [PPM]
+50	32767.98	-0.61
+25	32768.28	8.545
0	32768.38	11.597
-25	32768.08	2.441
-50	32767.58	-12.817

Table 5-13: Frequency over Temperature

The values show that even at nominal temperature (the temperature at which the chip was calibrated for energy), the deviation from the ideal crystal frequency is 11.6 PPM, resulting in about one second inaccuracy per day, i.e. more than some standards allow. As Figure 5-24 shows, even a constant compensation would not bring much improvement, since the temperature characteristics of the crystal are a mix of constant, linear, and quadratic effects.

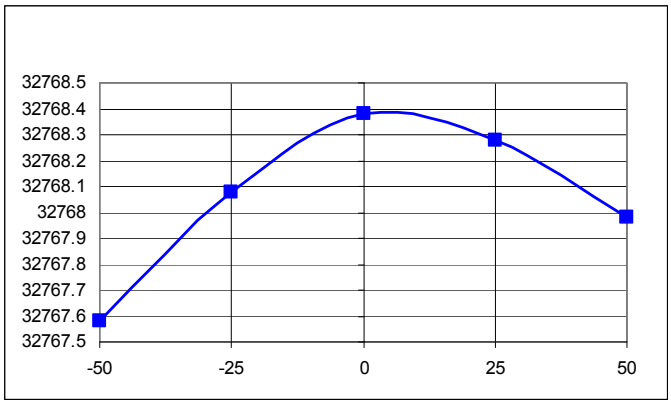


Figure 5-24: Crystal Frequency over Temperature

One method to correct the temperature characteristics of the crystal is to obtain coefficients from the curve in Figure 31 by curve-fitting the PPM deviations A fairly close curve fit is achieved with the coefficients a = 10.89, b = 0.122, and c = –0.00714 (see Figure 32).

$$f = f_{nom} * (1 + a/10^6 + T * b/10^6 + T2* c/10^6)$$

When applying the inverted coefficients, a curve (see Figure 5-25) will result that effectively neutralizes the original crystal characteristics. The frequencies were calculated using the fit coefficients as follows:

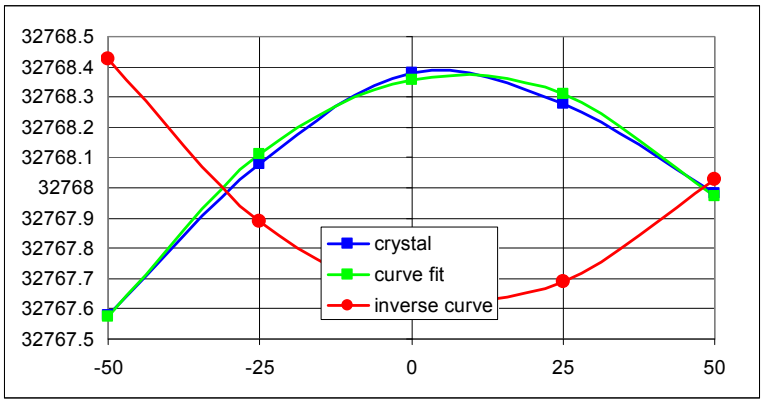


Figure 5-25: Crystal Compensation

The MPU Demo Code supplied with the TERIDIAN Demo Kits has a direct interface for these coefficients and it directly controls the *RTC_DEC_SEC* or *RTC_INC_SEC* registers. This interface is implemented by the MPU variables *Y_CAL*, *Y_CALC*, and *Y_CALC2* (MPU addresses 0x04, 0x05, 0x06). For the Demo Code, the coefficients have to be entered in the form:

$$CORRECTION(ppm) = \frac{Y\_CAL}{10} + T \cdot \frac{Y\_CALC}{100} + T^2 \cdot \frac{Y\_CALC2}{1000}$$

Note that the coefficients are scaled by 10, 100, and 1000 to provide more resolution. For our example case, the coefficients would then become (after rounding):

Y_CAL = 109, *Y_CALC* = 12, *Y_CALC2* = 7

Alternatively, the mains frequency may be used to stabilize or check the function of the RTC. For this purpose, the CE provides a count of the zero crossings detected for the selected line voltage in the *MAIN_EDGE_X* address. This count is equivalent to twice the line frequency, and can be used to synchronize and/or correct the RTC.

5.14.5 Validating the Battery

For applications that utilize the RTC it is very important to validate the battery. A brief loss of battery power when the 652X IC is powered down may result in corrupted RTC data.

The battery monitor function can be used to obtain the battery charge status.

After battery power is lost, the RTC will read the year 2001, the month January, and the day 1 (2001/01/01). The time information will be 01:01:01. If the MPU firmware program detects this date upon power-up or reset, it is safe to conclude that the RTC is corrupted, most likely due to a missing or low-voltage battery.

If invalid time/date information is detected, it sets them to 01:01:01, 1/1/2001.

5.15 ALPHABETICAL FUNCTION REFERENCE

Function/Routine Name	Description	Input	Output	File Name
abs_x()	x = abs(x0); Take absolute value of n-byte 'x0'.	uint8_tx *x, uint8_tx *x0, n	none	math.c
add()	*x += y; where 'x' & 'y' are 'n' bytes wide.	uint8_tx *x, uint8_tx *y, n	uint8_t	math.c
add_1()	*x += y; where 'x' is 'n' bytes wide, 'y' is single byte.	uint8_tx *x, y, n	uint8_t	math.c
add8_4()	(uint64_t) x += (uint32_t) y;	uint8_tx *x, uint8_tx *y	none	math.c
add8_8 ()	(uint64_t) x += (uint64_t) y	uint8_tx *x, uint8_tx *y	none	math.c
add8_8()	adds two unsigned 8-byte numbers	uint8_tx *x, uint8_tx *y	none	ce.c
Apply_Creep_Threshold()	Prevents creep.	void	void	meter.c
batmode_is_brownout ()	Returns true if battery mode is brownout. False is mission mode	void	bool	batmode_20.c
batmode_lcd ()	Enters LCD-only mode from brownout mode. Exit from LCD-only mode resembles a reset.	void	void	batmode_20.c
batmode_sleep ()	Enters sleep mode from brownout mode. Exit from sleep mode resembles a reset.	void	void	batmode_20.c
batmode_wait_minutes ()	Sets the wake timer in minutes.	uint16_t minutes	none	batmode_20.c
batmode_wait_seconds ()	Sets the wake timer in seconds.	uint16_t seconds	none	batmode_20.c
cal_begin()	starts auto-calibration process	none	bool	caphased.c
cal_restore()	Restores calibration from EEPROM	none	bool	calibration.c
cal_save()	saves calibration data to EEPROM	none	none	calibration.c
Calc_Voltage_Phase ()	Calculates phase angles between voltages of different phases.	void	void	vphase.c

Function/Routine Name	Description	Input	Output	File Name
calibrate()	processes measurements during auto-calibration	none	none	calphased.c
ce_active()	returns CE status	none	bool	ce.c
ce_enable()	Enables or disables the CE	bool enable	none	ce.c
ce_init()	Initializes the CE	none	bool	ce.c
ce_reset()	resets the CE	none	none	ce.c
cli ()	command Line Interpreter	none	none	cli.c
cli_init()	Initializes the SLI's interface to any serial port.	enum SERIAL_PORT port, enum SERIAL_SPD speed, bool xon_xoff	bool	sercli.c3
cli0_init()	Initializes the SLI's interface to SER0	enum SERIAL_SPD speed, bool xon_xoff	bool	ser0cli.c3
cli1_init()	Initializes the SLI's interface to SER1	enum SERIAL_SPD speed, bool xon_xoff	bool	ser1cli.c3
cmax()	returns maximum of unsigned char 'a' and 'b'.	uint8_t a, uint8_t b	uint8_t	math.c
cmd_ce ()	processes CE commands	none	none	cmd_ce.c
cmd_ce_data_access ()	Processes context for CE DATA	none	none	access.c
cmd_download()	downloads/uploads code/data between various sources and serial port	none	none	load.c
cmd_eeprom()	processes EEPROM commands	none	none	cmd_misc.c
cmd_error()	assigns generic command mode error result code	none	none	cli.c
cmd_lcd()	processes "D" commands	none	none	display.c
cmd_load()	implements user dialog for data/code download/upload	none	none	load.c
cmd_meter()	processes "M" commands	none	none	meter.c
cmd_mpu_data_access()	processes context for MPU DATA	none	none	access.c
cmd_power_save()	processes power save command	none	none	cmd_misc.c
cmd_rtc()	processes RTC commands	none	none	cmd_misc.c
cmd_trim()	processes trim commands	none	none	cmd_misc.c
cmin()	returns minimum of unsigned char 'a' and 'b'.	uint8_t a, uint8_t b	uint8_t	math.c
complement_x()	x = x0 ^ 1s; takes ones-complement of n-byte 'x0'.	uint8_tx *x, uint8_tx *x0, n	none	math.c
Compute_Phase_Angle()	Computes the V/I phase angle.	void	void	phase_angle.c
Compute_RMS()	Computes Vrms and Irms.	void	void	rms.c

Function/Routine Name	Description	Input	Output	File Name
LRC_Calc_NVR()	calculates longitudinal parity on NVRAM	uint8_tx *ptr, uint16_t len, U01 set	bool	library.c
ctoh()	converts ascii hex character to hexadecimal digit	uint8_t c	uint8_t	load.c
date_lcd ()	Displays the current date.	void	void	rtc.c
Delta_Time ()	Figure the elapsed time between two times.	struct RTC_t start, struct RTC_t end	int32_t seconds	rtc.c
Determine_Frequency()	Sets the frequency. Uses sag status and voltage thresholds to return 0 if the voltages are off.	void	void	freq.c
Determine_Peaks()	Sets status bits if voltages, currents or temperature are outside limits. Sag tests are in xfer_busy_int()	void	void	peak_alerts.c
done()	exits control	uint8_td *c	*c	cli.c
EEProm_Config()	connects/disconnects DIO4/5 for I2C interface to serial EEPROM	bool access, uint16_t page_size, uint8_t tWr	none	eeprom.c, eepromp3.c
es0_isr ()	serial port 0 service routine	none	none	ser0.c
es1_isr()	serial port 1 service routine	none	none	se1.c
flag0_in()	Input interrupt for FLAG AMR module on SER0	none	none	flag0.c3
flag0_initialize()	Initialize the FLAG AMR module on SER0	none	none	flag0.c3
flag0_out()	Output interrupt for FLAG AMR module on SER0	none	none	flag0.c3
flag0_run()	Run main loop logic for FLAG AMR module on SER0	none	none	flag0.c3
flag1_in()	Input interrupt for FLAG AMR module on SER1	none	none	flag1.c3
flag1_initialize()	Initialize the FLAG AMR module on SER1	none	none	flag1.c3

Function/Routine Name	Description	Input	Output	File Name
flag1_out()	Output interrupt for FLAG AMR module on SER1	none	none	flag1.c
flag1_run()	Run main loop logic for FLAG AMR module on SER1	none	none	flag1.c
frequency_lcd ()	Displays the frequency on the LCD.	void	void	freq.c
get_ce_constants()	Copies CE configuration constants to a data structure so they can be viewed in the emulator.	void	void	ce.c
get_char()	gets next character from CLI buffer	none	uint8_t	io.c
get_char_d()	gets next character from CLI buffer	uint8_t idata *d	uint8_t	io.c
get_digit()	gets next decimal (or hex) digit from CLI buffer	uint8_t idata *d	uint8_t	io.c
get_long()	converts ascii decimal (or hex) long to binary number	none	int32_t	io.c
get_long_decimal()	converts ascii decimal long to binary number.	uint8_t c	int32_t	io.c
get_long_hex()	converts ASCII hexadecimal number to binary number	none	U32	io.c
get_num()	converts ascii decimal (or hex) number to binary number	none	S08	io.c
get_num_decimal()	converts ascii decimal number to binary number	none	S08	io.c
get_num_hex()	converts ascii hexdecimal byte to binary number	none	uint8_t	io.c
get_short()	converts ascii decimal (or hex) short to binary number	none	int16_t	io.c
get_short_decimal()	converts ascii decimal short to binary number	none	int16_t	io.c
get_short_hex()	converts ascii hexdecimal short to binary number	none	uint16_t	io.c
htoc()	converts hexadecimal digit to ascii hex character	uint8_t c	uint8_t	load.c
IICGetBit()	gets a bit, used to reset some parts	none	bit	iiceep.c
IICInit()	initializes DIO4/5 as EEPROM interface	none	none	iiceep.c

Function/Routine Name	Description	Input	Output	File Name
IICStart()	IIC bus's start condition	none	none	iiceep.c
IICStop()	IIC bus's stop condition	none	none	iiceep.c
init_meter()	Initializes meter to default values	none	none	defaults.c
IRQ_DEFINES	Defines variables used by macros to enable and disable interrupts.	n/a	n/a	irq.h
irq_disable()	Disables interrupts.	void	void	irq.c
IRQ_DISABLE()	The fastest way to disable interrupts. Requires IRQ_DEFINES to be earlier in the code, or that the needed symbols be defined.	n/a	n/a	irq.h
irq_enable()	Enables interrupts	void	void	irq.c
IRQ_ENABLE()	The fastest way to enable interrupts. Requires IRQ_DEFINES to be earlier in the code, or that the needed symbols be defined.	n/a	n/a	irq.h
irq_init()	Initializes interrupt control.	void	void	irq.c
labs()	returns the absolute value	int32_t x	S332	math.c
latan2()	returns the arcTangent	int32_t sy, int32_t sx	U32	math.c
LCD_CE_Off()	displays "CE OFF" on LCD	none	none	lcd.c
LCD_Command()	turns LCD on or off, clears display	uint8_t LcdCmd	none	lcd.c
LCD_Config()	configures LCD parameters	uint8_t num, enum eLCD_mode bias, enum LCD_CLK clock	none	lcd.c
LCD_Data_Read()	reads from selected icon of LCD	uint8_t Icon	uint16_t	lcd.c
LCD_Data_Write()	writes to selected icon of LCD	uint8_t icon, uint16_t Mask	none	lcd.c
LCD_Hello()	displays "HELLO" on LCD	none	none	lcd.c
LCD_Init()	clears LCD, enables LCD segment drivers	none	none	lcd.c
LCD_Mode	Display a mode number.	uint8_t mode	none	lcd.c

Function/Routine Name	Description	Input	Output	File Name
LCD_Number()	Displays a number on the LCD.	int32_t number uint8_t num_digits_before_decimal_point, uint8_t num_digits_after_decimal_point	none	lcd.c
lmax()	returns maximum of unsigned long 'a' and 'b'.	U32 a, U32 b	U32	math.c
lmin()	returns minimum of unsigned long 'a' and 'b'.	U32 a, U32 b	U32	math.c
log2()	returns binary logarithm	uint16_t k	uint8_t	math.c
main_background()	executes background processing	none	none	main.c
main_edge_cnt_lcd()	Displays either the instantaneous edge count, or the cumulative edge count.	uint8_t select	void	freq.c
main_soft_reset()	initiates soft reset	none	none	main.c
max()	returns maximum of unsigned int 'a' and 'b'.	uint16_t a, uint16_t b	uint16_t	options_glib.h
memcpy_cei ()	Copies from IDATA to the CE memory.	int32x_t *pDst, int32i_t *pSrc, uint8_t len	void	ce.c
memcpy_cer ()	Copies from flash to the CE memory.	int32x_t *pDst, int32r_t *pSrc, uint8_t len	void	ce.c
memcpy_cex ()	Copies from XDATA to the CE memory.	int32x_t *pDst, int32x_t *pSrc, uint8_t len	void	ce.c
MEMCPY_MCE ()	Copies from the CE memory to IDATA.	int32i_t *pDst, int32x_t *pSrc, uint8_t len	void	ce.c
memcpy_xce ()	Copies from the CE memory to XDATA.	int32x_t *pDst, int32x_t *pSrc, uint8_t len	void	ce.c
memget_ce ()	Reads a word of the CE memory	int32i_t *pDst	int32_t	ce.c
memset_ce ()	Sets a word of the CE memory	int32i_t *pDst, int32_t src	void	ce.c
meter_lcd ()	Display the current quantity on the LCD.	void	void	meter.c
meter_run ()	Performs meter data processing.	void	void	meter.c
memcmp_rx()	compares xdata to flash code	uint8_tr *rsrc, uint8_tx *xsrc, uint16_t len	S08	library.c

Function/Routine Name	Description	Input	Output	File Name
memcmp_xx()	compares xdata to xdata	uint8_tx *xsrc1, uint8_tx *xsrc2, uint16_t len	S08	library.c
memcpy_ix()	copies xdata to idata	uint8_ti *dst, uint8_tx *src, uint8_t len	none	library.c
memcpy_px()	Copies data to serial EEPROM	U32 Dst, uint8_tx *pSrc, uint16_t len	enum	eeprom.c, eepromp.c, eepromp3.c
memcpy_rce()	reads from or writes to flash	int32_tr *dst, int32_tx *src, uint8_t len	none	flash.c
memcpy_rx()	Copies xdata to code (flash)	uint8_tr *dst, uint8_tx *src, uint16_t len	bool	flash.c
memcpy_xi()	Copies idata to xdata	uint8_tx *dst, uint8_ti *src, uint8_t len	none	library.c
memcpy_xp()	copies data from serial EEPROM	uint8_tx *pDst, U32 Src, uint16_t len	enum	eeprom.c, eepromp.c, eepromp3.c
memcpy_xr()	copies xdata from code (flash)	uint8_tx *dst, uint8_tr *src, uint16_t len	none	library.c
memcpy_xx()	copies xdata to xdata	uint8_tx *dst, uint8_tx *src, uint16_t len	none	library.c
memset_x()	sets xdata to specified value	uint8_tx *dst, uint8_t s, uint16_t len	none	library.c
meter_initialize()	initializes most I/O functions thaty read line power	none	none	meter.c
meter_totals ()	Display a selected quantity on the LCD.	uint8_t select,uint8_t phase	void	meter.c
microseconds2tmr_reg()	Converts to timer's count.	number	uint16_t	tmr0.h, tmr1.h
milliseconds()	Converts milliseconds to clock ticks, usually for a software timer.	any number	uint16_t	stm.h
milliseconds2tmr_reg()	Converts to timer's count.	number	uint16_t	tmr0.h, tmr1.h
min()	returns minimum of unsigned int 'a' and 'b'.	uint16_t a, uint16_t b	uint16_t	options_glib.h
MPU_Clk_Select()	selects MPU clock speed	enum MPU_SPD speed	bool	serial.c
MPU_Clk_Select()	Describes the clock speed of the MPU to a serial interface.	enum SERIAL_PORT port, enum eMPU_DIV speed	bool	sercli.c3
MPU_Clk_Select0()	Describes the clock speed of the MPU to the serial interface.	enum eMPU_DIV speed	bool	ser0cli.c3
MPU_Clk_Select1()	Describes the clock speed of the MPU to the serial interface.	enum eMPU_DIV speed	bool	ser1cli.c3

Function/Routine Name	Description	Input	Output	File Name
normalize()	puts a register into normal form, in which the major count of Wh is greater than or equal to zero, and less than 1,000,000,000. The minor count of ce counts (e.g. from w0sum) is made to be less than 1 Wh and positive, by transfering larger amounts into the major count.	uint8x_t *register_ptr	none	math.c
operating_lcd ()	Displays the number of hours of operation.	void	void	rtc.c
OperatingHours()	Calculates hours of operation from the last valid mark.	none	int32_t hours	rtc.c
OSCOPE_INIT	Defines DIO_7, the VAR pulse output as a DIO.	n/a	n/a	oscope.h
OSCOPE_ONE	Set DIO_7, the same pin as the VARh pulse output, to high.	n/a	n/a	oscope.h
OSCOPE_TOGGLE	Inverts DIO_7, the same pin as the VARh pulse output.	n/a	n/a	oscope.h
OSCOPE_ZERO	Set DIO_7, the same pin as the VARh pulse output, to low.	n/a	n/a	oscope.h
pcnt_accumulate()	Accumulates counts from the previous second.	void	void	pcnt.c
pcnt_init ()	Initialize logic to count output pulses.	void	void	pcnt.c
pcnt_lcd()	Display pulse count on LCD	uint8_t select	void	pcnt.c
pcnt_start()	Starts plse-counting for a fixed number of seconds.	int16_t seconds	void	pcnt.c
pcnt_update()	Synchronizes pulse counts with noninterrupting code.	void	void	pcnt.c
phase_angle_lcd ()	Displays a V/I phase angle.	uint8_t phase	void	phase_angle.c
psoft_init ()	Initializes software pulse outputs.	void	void	psoft.c
psoft_out()	Generates two additional pulse outputs. Call from ce_busy_isr	void	void	psoft.c

Function/Routine Name	Description	Input	Output	File Name
psoft_update ()	The inputs are watt hours, as generated by the CE, and set the extra pulse generators to blink at the same rate as CE pulse outputs, with the same units. This should be called each time a new accumulation interval has data.	int32_t pulse3_in, int32_t pulse4_in	void	psoft.c
put_char()	puts character into CLI buffer	uint8_t idata *c	none	io.c
Read_Trim()	reads the trim value for selected trim word	enum eTRIM select	S08	ce.c
rms_i_lcd()	Displays current.	uint8_t phase	void	rms.c
rms_v_lcd()	Displays voltage.	uint8_t phase	void	rms.c
RTC_Adjust_Trim()	Safely sets the compensation variables.	bool clr_cnt, int32_t value	none	rtc.c
RTC_Compensation()	Calculates and adjusts the temperature compensation for the RTC.	none	none	rtc.c
rtc_isr ()	Interrupt code to adjust clock each second.	void	void	rtc.c
RTClk_Read()	reads current values of RTC	none	none	rtc.c
RTClk_Reset()	resets the RTC	none	none	rtc.c
RTC_Trim()	Calculates the temperature compensation using Y_Cals	none	int32_t ppb	rtc.c
RTClk_Write()	writes/sets to RTC	none	none	rtc.c
seconds()	Converts seconds to clock ticks, usually for a software timer.	any number	uint16_t	stm.h
SelectPulses()	Selects pulse sources for 2 CE pulse outputs, and optionally, for two additional software pulse outputs. The controls are in MPU variables initialized from the default table.	void	void	pulse_src.c
send_a_result()	sends passed result code to UART	uint8_t c	none	cli.c
send_byte()	sends a [0, 255] byte to DTE.	S08 n	none	io.c
send_char()	sends single character	uint8_t c	none	io.c
send_crlf()	sends <CR><LF> out to UART.	none	none	io.c
send_digit()	sends single ASCII hex or decimal digit out to SERIAL0	uint8_t c	none	io.c

Function/Routine Name	Description	Input	Output	File Name
send_help()	sends text in code at specified location to serial port	uint8_tr * code *s	none	cli.c
send_hex()	sends byte out SERIAL0 in HEX	uint8_t n	none	io.c
send_long()	sends a [0, 9,999,999,999] value to DTE.	int32_t n	none	io.c
send_long_hex()	sends a [0, FFFFFFFF] value to DTE	U32 n	none	io.c
send_num()	sends a [0, 9,999,999,999] value to DTE	int32_t n, uint8_t size	none	io.c
send_result()	looks up result code, primes pump for result codes	none	none	cli.c
send_rtc()	displays RTC data	none	none	cmd_misc.c
send_short()	sends a [0, 99,999] value to DTE.	int16_t n	none	io.c
send_short_hex()	sends a [0, FFFF] value to DTE	uint16_t n	none	io.c
ser_disable_rcv_rdy()	Disable the receive interrupt.	void	void	ser0.h, ser1.h
ser_disable_xmit_rdy()	Disable the transmit interrupt.	void	void	ser0.h, ser1.h
ser_enable_rcv_rdy()	Enable the receive interrupt.	void	void	ser0.h, ser1.h
ser_enable_xmit_rdy()	Enable the transmit interrupt.	void	void	ser0.h, ser1.h
Ser_initialize()	configures the serial port specified in the include file ser0.h or ser1.h	enum baud	none	ser0.h, ser1.h
ser_rcv ()	Get a byte from the serial port.	void	uint8_t	ser0.h, ser1.h
ser_rcv_err()	Returns true if the last received byte had an error.	void	bool	ser0.h, ser1.h
ser_rcv_rdy()	Returns true if the serial port has gotten another byte.	void	bool	ser0.h, ser1.h
ser_xmit ()	Send a byte to the serial port.	uint8_t	void	ser0.h, ser1.h
ser_xmit_err()	Returns true if the last sent byte had an error.	void	bool	ser0.h, ser1.h
ser_xmit_free ()	Unimplemented routine to permit other uses of transmit electronics.	void	void	ser0.h, ser1.h
ser_xmit_off ()	Unimplemented routine to disable transmit electronics.	void	void	ser0.h, ser1.h
ser_xmit_on ()	Unimplemented routine to enable transmit electronics.	void	void	ser0.h, ser1.h

Function/Routine Name	Description	Input	Output	File Name
ser_xmit_rdy()	Returns true if the serial port can send another byte.	void	bool	ser0.h, ser1.h
Serial_CRx()	Receive a string up to a maximum length.	enum SERIAL_PORT port, uint8x_t *buffer, uint16_t len	uint16_t length-received	sercli.c3
Serial_CTx()	Transmit a string up to a maximum length.	enum SERIAL_PORT port, uint8x_t *buffer, uint16_t len	uint16_t length-sent	sercli.c3
Serial_CRx()	gets additional bytes from the receive buffer	enum SERIAL_PORT port, uint8_tx *buffer, uint16_t len	uint16_t	secli.c
Serial_CTx ()	puts additional bytes into the transmit buffer	enum SERIAL_PORT port, uint8_tx *buffer, uint16_t len	uint16_t	sercli.c
Serial_Rx()	Receive a string of any length.	enum SERIAL_PORT port, uint8x_t *buffer, uint16_t len	none	sercli.c3
Serial_Rx ()	sets up receive buffer and starts receiving	enum SERIAL_PORT port, uint8_tx *buffer, uint16_t len	enum SERIAL_RC data	sercli.c
Serial_RxFlowOff()	Force an XOFF to be sent on the selected port.	enum SERIAL_PORT port	none	sercli.c3
Serial_RxFlowOn()	Force an XON to be sent on the selected port.	enum SERIAL_PORT port	none	sercli.c3
Serial_RxLen()	returns the number of bytes received	enum SERIAL_PORT port	uint16_t	sercli.c
Serial_Tx()	Transmit a string of any length.	enum SERIAL_PORT port, uint8x_t *buffer, uint16_t len	none	sercli.c
Serial_Tx()	sets up transmission buffer and starts transmission	enum SERIAL_PORT port, uint8_tx *buffer, uint16_t len	enum SERIAL_RC data	sercli.c
Serial_TxLen()	returns the number of bytes left to transmit	enum SERIAL_PORT port	uint16_t	sercli.c
Serial0_CRx()	Receive a string up to a maximum length.	uint8x_t *buffer, uint16_t len	uint16_t length-received	ser0cli.c
Serial0_CTx()	Transmit a string up to a maximum length.	uint8x_t *buffer, uint16_t len	uint16_t length-sent	ser0cli.c
Serial0_Rx()	Receive a string of any length.	uint8x_t *buffer, uint16_t len	none	ser0cli.c
Serial0_RxFlowOff()	Force an XOFF to be sent on this port.	none	none	ser0cli.c
Serial0_RxFlowOn()	Force an XON to be sent on this port.	none	none	ser0cli.c
Serial0_Tx()	Transmit a string of any length.	uint8x_t *buffer, uint16_t len	none	ser0cli.c
Serial1_CRx()	Receive a string up to a maximum length.	uint8x_t *buffer, uint16_t len	uint16_t length-received	ser1cli.c

Function/Routine Name	Description	Input	Output	File Name
Serial1_CTx()	Transmit a string up to a maximum length.	uint8x_t *buffer, uint16_t len	uint16_t length-sent	ser1cli.c
Serial1_Rx()	Receive a string of any length.	uint8x_t *buffer, uint16_t len	none	ser1cli.c
Serial1_RxFlowOff()	Force an XOFF to be sent on this port.	none	none	ser1cli.c
Serial1_RxFlowOn()	Force an XON to be sent on this port.	none	none	ser1cli.c
Serial1_Tx()	Transmit a string of any length.	uint8x_t *buffer, uint16_t len	none	ser1cli.c
SFR_Read()	reads from SFR	uint8_t s, S08d *pc	enum SFR_RC	sfrs.c
SFR_Write()	writes to SFR	uint8_t s, uint8_t c_set, uint8_t c_clr	enum SFR_RC	sfrs.c
start_tx_ram()	sends RAM string out PC UART	uint8_tx *c	none	io.c
start_tx_rslt()	sends ROM string out PC UART	uint8_tr *c	none	io.c
stm_run()	This counts down the software timers when called from the main loop.	void	void	stm.c
stm_start()	Starts a software timer. If restart is zero, the timer stops, otherwise it continues indefinitely. When a timer expires, its function is run. Timers count down and are deallocated if they cease to run.	uint16_t tick_count, uint8_t restart, void (code *fn_ptr) (void)	volatile uint16x_t *cnt_ptr	stm.c
stm_stop()	Uses a count pointer from start to identify which software timer to stop.	volatile uint16x_t *cnt_ptr	void	stm.c

Function/Routine Name	Description	Input	Output	File Name
stm_wait()	Waits for the passed number of clock ticks.	uint16_t	void	stm.c
strlen_r ()	returns length of string in flash code	uint8_tr *src	uint16_t	library.c
strlen_x()	returns length of string in xdata	uint8_tx *src	uint16_t	library.c
sub8_4()	(uint64_t) x -= (uint32_t) y	uint8_tx *x, uint8_tx *y	none	math.c
sub8_8()	(uint64_t) x -= (uint64_t) y	uint8_tx *x, uint8_tx *y	none	math.c
temperature_lcd()	Displays the current delta from the calibration temperature in degrees C on the LCD.	void	void	meter.c
time_lcd ()	Displays the current time.	void	void	rtc.c
tmr_disable ()	Halt a timer.	none	none	tmr0.h, tmr1.h
tmr_enable ()	Lets a timer run (timer start does this by default)	none	none	tmr0.h, tmr1.h
tmr_running ()	Returns true if the timer is running.	none	bool	tmr0.h, tmr1.h
tmr_start ()	Starts a hardware timer.	uint16_t time (in timer units), uint8_t restart_flag (zero means interrupt once), void (code *pfn) (void) (code to execute)	none	tmr0.h, tmr1.h, tmr0.c, tmr1.c
tmr_stop ()	Stops a hardware timer.	none	none	tmr0.h, tmr1.h
tmr0_isr ()	Timer interrupt for TMR0	none	none	tmr0.c
tmr1_isr ()	Timer interrupt for TMR1	none	none	tmr1.c
update_register ()	Move data from AMR's copy of power registers into power registers.	void	void	meter.c
uwr_busy_wait ()	Wait for programming com-plete indication.	none	none	uwrdio.c, uwreep.c2
uwr_init ()	Initialize a 3-wire (similar to uWire™) interface	none	none	uwrdio.c, uwreep.c2
uwr_read ()	Get a counted string of bytes.	uint8x_t *pbOut, uint16_t cnt	none	uwrdio.c, uwreep.c2
uwr_select ()	Select a chip by passing its address; 0 = none; This must be ported to new PCBs.	uint8_t address	none	uwrdio.c, uwreep.c2
uwr_write ()	Transmit a counted string of bytes.	uint8x_t *pbOut, uint16_t cnt	bool true = success.	uwrdio.c, uwreep.c2

Function/Routine Name	Description	Input	Output	File Name
VAh_Accumulate()	Calculates VAh	void	void	vah.c
VARh_Accumulate()	Calculates VARh	void	void	varh.c
voltage_phase_lcd()	Display voltage phases on LCD.	uint8_t select	void	vphase.c
wd_create()	Creates a software watchdog.	uint8_t wd	void	wd.c
wd_destroy()	Destroys a software watchdog.	uint8_t wd	void	wd.c
wd_reset()	Resets a software watch-dog. If all software watch-dogs have been reset, the hardware watchdog is reset.	uint8_t wd	void	wd.c
wh_accumulate()	Calculate watt hours.	void	void	wh.c
wh_brownout_to_lcd()	Displays a precalculated 6-digit number.	uint32_t number	void	wh.c
wh_lcd()	Displays a watt-hour value on the LCD in milliwatt-hours.	uint8_t *val	void	wh.c
wh_sum_export ()	Adds (0 - w1) to s, only if w1 is negative, yielding a total of exported power in w.	uint8x_t *s, int32i_t *wl	void	wh.c
wh_sum_import ()	Adds w1 to s, only if w1 is positive, yielding a total of imported power in w.	uint8x_t *s, int32i_t *wl	void	wh.c
wh_sum_net()	Adds w1 to s, yielding a net sum of watthours in s.	uint8x_t *s, int32i_t *wl	void	wh.c
wh_to_long ()	Convert a 64-bit internal watts count to a 6-digit value (i.e. this is the routine that precalculates values for wh_brownout_to_lcd()).	uint8_t *val	uint32_t	wh.c

5.16 ERRATA

The up-to-date list of known issues with the current revision of the Demo Code can be found in the readme.txt file contained in the 6521_demo ZIP file shipped with the Demo Kits. The readme.txt file also contains a list of corrected issues, that might assist customers who utilized older versions of Teridian demo code.

The factory should be contacted for updates to the Demo Code.

5.17 PORTING 71M6511/6513 CODE TO THE 71M6521

5.17.1 Memory Use

The biggest issue when moving code from the 6511/6513 to the 71M6521 is the reduced program memory. While the 71M6511 and 6513 have 64K, the 6521 has 32K, 16K and 8K versions. The standard 6521 CE code has 414 bytes, and takes up space in flash. In order to make the code fit, and reduce the risk of running out of memory, TERIDIAN's firmware engineers coded to a space budget, and adopted a coding standard that permits entire features to be added and removed easily. This method proved to be very useful when there were changes of scope in one or another version of the demonstration firmware.

5.17.2 CE Code Location

Another difference between 71M6511/6513 and the 71M6521 is that the CE code now resides in the flash. It is not copied to the CE program RAM as in the 71M651X chips. Instead, the register CE_LCTN, bits 0…4 at XDATA 0x20A8 is set to the most significant 5 bits of the program flash address where the CE program resides. It is best to place the CE program within the first 8Kbytes, so that the program design adapts easily to the three size variants of the chip.

Since the CE resides in flash memory, there are safeguards that prevent the CE program memory from being erased or reprogrammed while the CE is running. When programming flash memory, the CE must first be disabled, then the code must wait until the CE is halted. Only then, programming of the flash memory can occur.

Should there be an attempt to modify flash memory while the CE is running, the FWCOL0 and FWCOL1 interrupt bits are set. If the interrupt is enabled, recovery action can occur. TSC's demo firmware has not found a use for the FWCOL interrupts.

5.17.3 Battery Modes

One of the most significant innovations for the 71M6521 is the battery-power feature. This feature provides three operational modes, that apply when the supply voltage is removed and the chip is powered by the battery. The operation modes and their transitions are shown in Figure 5-26.

In the brownout mode, operation continues at 32kHz, and RAM and DIO pins remain powered. However, the clock slows down and is so slow that the timers and serial port give dramatically different timings. Only the RTC, and its 1-second interrupt run at an unchanged speed.

In addition to the flags given in Figure 5-26, the following considerations apply to state transitions:

- Mission to brownout mode: The MPU keeps running, but the clock slows down.
- Brownout to mission mode: The MPU keeps running, but the clock speeds up.
- LCD or sleep mode to brownout mode: The MPU will start code execution at address 0x000.

The sleep and LCD modes shut down all of the 71M6521's internal and XDATA RAM, as well as the pin drivers for DIOs, and many of the memory cells that store the hardware configuration.

In particular, the meter should be designed so that the DIO pins and serial port outputs do not need to be powered in battery modes.

The data sheet for the 71M6521 shows which bits are reset, and which are maintained in the battery modes.

The transitions between the modes are managed by changes in supply voltage, transitions of the push button pin signal, and a wake-up timer.

The push-button operation is very simple: Pressing the button wakes the part from LCD or sleep mode into brownout mode. Afterward, a bit is set: IE_PB, bit 4 of IFLAGS, SFR E8.

One of the characteristics of the 71M6521 is that it is not able to enter LCD or sleep mode if IE_PB or IE_WAKE (the wake timer's bit, see below) are set. The Demo Code clears these bits at the earliest convenient instant, transferring their state to bits in the demo firmware's status variable. This technique preserves data about how the chip last woke, but also permits the chip to transition to the LCD and sleep modes easily.

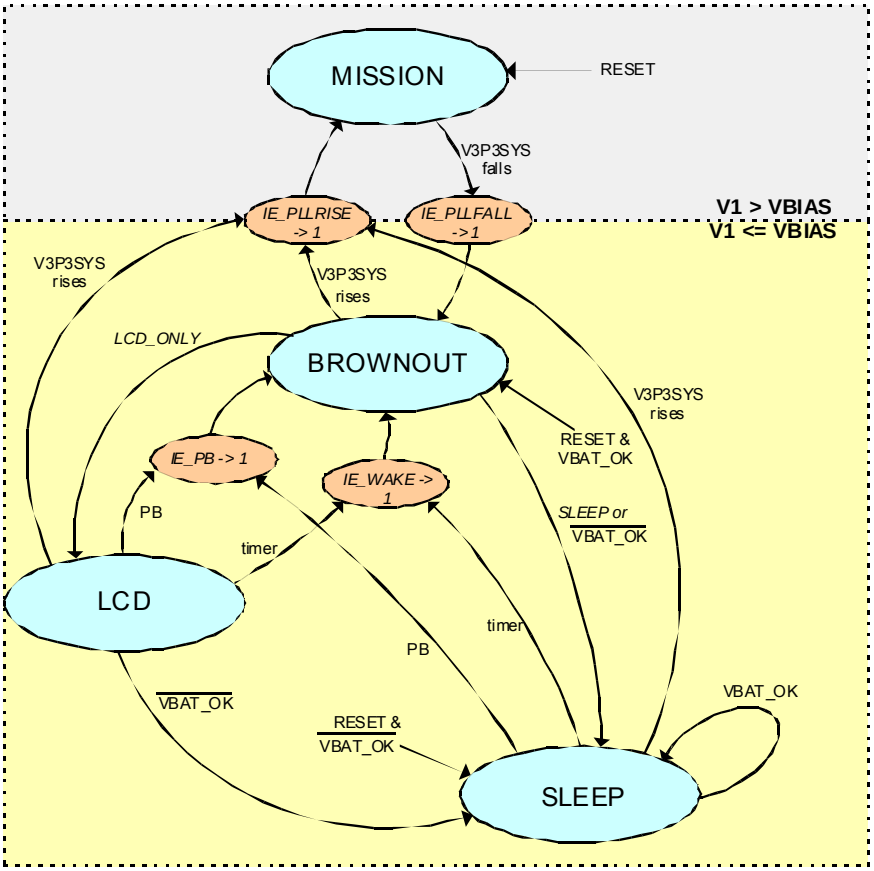


Figure 5-26: Operation Modes State Diagram

The wake-up timer is a little trickier to use than the pushbutton. To use it, one must first write the timer data, and then, after a brief delay (about 32µs), enter LCD or sleep mode. The timer does not measure elapsed time. Instead, it counts the RTC's transitions. For example, if one programs a two minute delay at 00:00:30, the timer will actually wake the chip at 00:02:00. Properly used, this is a feature, of course, but it can be surprising.

The wake-up timer, LCD and sleep modes are controlled by the bits in the WAKE register, XDATA address 0x20A9.

The lack of nonvolatile memory during the battery modes can be disconcerting at first. There are usually a few bytes worth of available nonvolatile space in the unused LCD segment control bits in XDATA addresses 0x2036...0x2056.

The transition from mission mode to brownout and from brownout to mission mode is invisible to the code without special care. First, there is a bit PLL_OK in the I/O RAM at address 0x2003 that reflects whether the phase locked loop (PLL) is running or not. In order to save power, the PLL does not run when the part runs from any battery mode. PLL_OK also drives logic that sets the bits IE_PLLRISE and IE_PLLFALL in IFLAGS, SFR E8. These are logically-ored and routed to external interrupt 4, which is an edge-triggered interrupt. This interrupt could be called "the brownout mode interrupt" because it signals any transition between brownout and mission mode. It is very important that both IE_PLLFALL and IE_PLLRISE be cleared at the end of the interrupt, in the same instruction, otherwise the edge needed for the next interrupt might fail to occur.

The brownout mode interrupt has to manage the transition to and from brownout mode. The Demo Code's brownout interrupt handles this by displaying a watt-hour value, and then performing a soft reset.

The chip starts a normal power-up in brownout mode, and then transitions to mission mode about 4.1 milliseconds after power is applied. The code is often quite far along the brownout mode path at this point, and must gracefully transition to mission mode. The soft reset in the brownout interrupt handles this requirement very well.

The Demo Code still includes a "belt and suspenders" test for a change to or from brownout mode in the main loop. This also performs a soft reset.

The demo software is designed so that the serial port always runs at 300 bd, in both mission mode and brownout mode. The 6521 has special clock-interpolation logic in the baud rate generation so that 300 baud works in brownout mode. The designers chose this 300 bd especially because it is compatible with some AMR applications, such as FLAG, and it was achievable in the chip. The Demo Code tests for brownout mode, and sets the 300 bd values in the serial ports' bd rate generator in this case.

In brownout mode, the code runs 150 times slower than in mission mode, and it can easily fail to reset the watchdog. The Demo Code arranges to reset the watchdog from both the main loop and the RTC's 1-second interrupt, which has unchanged timing. It uses a "software watchdog" scheme to try to keep this respectable. The idea is that as soon as all the needed places have called the watchdog routine, the hardware watchdog is reset.

Code for brownout mode should minimize calculations, because brownout mode is 150 times slower than mission mode. To minimize the calculations, in the Demo Code, every accumulation interval in mission mode caches a pre-calculated Wh value for use in a transition to brownout mode. When the interrupt for brownout mode executes, this value is converted to the digits of the LCD registers. The LCD registers are nonvolatile in sleep and LCD modes, so they are not lost in any battery-mode transitions. Later, when the Demo Code awakes (probably because the push-button was pressed) in brownout mode, it runs through the brief initialization needed by the C environment, and in main(), it tests for brownout mode. In the brownout mode's code it runs a very simple, fast state machine that uses the wake button flag to decide whether to enter sleep mode or LCD mode, and just depends on the LCD registers to remain unchanged.

An algorithm similar to this could be adapted to display several values, setting the new value in the LCD as the last step after all other calculations were done in brownout mode. After displaying the values in the LCD, the code could enter LCD mode to save power and still display the value.

When the chip wakes from sleep or LCD mode, the PC is cleared to zero, and the I/O bits that are not needed for the RTC or LCD are reset. The experience of the firmware designers is that it is most convenient to treat transitions from LCD and sleep modes like resets. This permits a relatively simple start-up initialization to handle the state-transitions, as well as power-up in mission mode. That scheme proved so convenient that the Demo Code also used the same scheme to transition to and from brownout mode.

It's not clear at first how to distinguish hard resets from battery-mode transitions. The code can use the nonvolatile LCD control registers. The trick is that after a reset, the LCD registers are cleared. In particular, LCD_NUM, bits 0..4 of XDATA address 0x2020 are cleared after a hard reset.

5.17.4 Three-Wire EEPROM Hardware

The 71M6521 includes a new three-wire serial EEPROM interface, which is designed to be compatible with MicroWire™ EEPROMs.

The new 3-wire interface hardware is very fast, transferring a byte in only 16µs. This high speed has made it relatively uneconomical to use the interrupt provided on this interface. At 16µs per byte, the interrupt overhead would be most of the delay in the EEPROM control firmware. Therefore, the Demo Code uses a polling driver that reads the ready bit.

Some 3-wire serial EEPROMs (e.g. the Microchip 93C76C) signal completion of a write operation by driving the data line from low to high. The 71M6521 handles this with two controls: First, there is a "HIZ" bit in EECTRL that forces the output of the 71M6521 to a high-impedance state after the last bit is sent. Also, the bit WFR (wait for ready) in EECTRL makes the 71M6521's BUSY status bit stay true until the data line becomes high. However, there is a period during which the data line is not driven. If the data line is not pulled-down, a trailing 1 on the last data bit will leave the line capacitance holding the line well above the transition voltage, causing BUSY to become prematurely false. But if the pull-down is too powerful, the EEPROM may not be able to drive it (e.g. the 93C76C has only 400µA of drive on the high state of the data line). An alternative method (uwreep.c in the Demo Code) that is clumsy but reliable and inexpensive (it saves the resistor), is to complete the last data write without any special modes, then send another 8 bits of zero, with the wait-for-ready and high-Z bits set for that transfer. Note that sending one bit of zero works in simulations, but not in the lab, for reasons that are not yet clear.

The demo source code also includes a programmed-IO (bit-banging) driver (uwrdio.c). This driver lets 71M6511 and 71M6513 chips use 3-wire EEPROMs, so meter product-lines can share an inventory of identical EEPROMs. Also, the 71M6521's 3-wire interface hardware is not flexible enough to drive some items designed for SPI. It only supports one clock polarity, one clock edge and the data line is half duplex. The programmed-IO driver is designed to be easily modified for these environments.

5.17.5 Temperature Compensation

When operating with "internal" temperature compensation, the 71M6511 and 71M6513 ICs use the CE as the compensation mechanism. Compensation is then based on the temperature deviation from nominal and the PPMC and PPMC2 factors that are either derived from the on-chip fuses (71M6511H/6513H) or standard values (71M6511/6513) that apply to the average chip.

In the 71M6521, the CE is no longer in charge of temperature compensation. In the 71M6521, the temperature calculations are performed once per second in the MPU firmware (see Gain_Compensation() in meter.c). The gain calculations set a global gain parameter ("gain_adj") used by the CE code. As a side-effect, the parameters PPMC and PPMC2, the coefficients that control the meter's linear and quadratic gain by temperature, are now in MPU memory space, rather than CE memory space. This causes very little loss of accuracy because the temperature changes only slowly.

5.18 TEST MODULES

Various Test Modules are available from TERIDIAN. These Test Modules are small Keil projects that can be used to test various functions of the 71M6521 IC. The available Test Modules are described in this section.

5.18.1 6513 CE Example

Even though written for the 71M6513, this Test Module can be used for the 71M6521. It builds a simple test code that starts and runs the compute engine, collects meter data in RAM, and generates pulses for one accumulation interval.

The Keil project file is `6513_ce_example.uv2`.

5.18.2 Serial Port Tests

These Test Modules build simple tests of the serial ports. The tests start by sending the ASCII character "E" in a loop, e.g. For testing with an oscilloscope. As soon as a character is received, the test code begins echoing typed characters, using polling IO. Sending the period character (".") switches the I/O to interrupting I/O.

Note that ser0test.c and ser1test.c use identical text, except for the include file. This is a very convenient technique for moving serial I/O to a different port when requirements change.

The Keil project files are `ser0test.uv2` and `ser1test.uv2`.

5.18.3 Timer Tests

These Test Modules build simple routines for testing of the interrupting timers, run both once, and periodically. The routines include an extended 30-second test that can be used with a stop-watch timer to measure accuracy.

Note that tmr0test.c and tmr1test.c use identical text except for the include file. This is a very convenient technique for moving a timer IO to a different port when requirements change.

The Keil project files are `tmr0test.uv2` and `tmr1test.uv2`.

5.18.4 EEPROM Tests

This routine demonstrates the use and test of the eeprom interface.

The Keil project file is `eepromtest.uv2`.

5.18.5 Generating DIO Pulses on Reset

This Test Module is written in 8051 assembler and is executed after processor reset. It pulses DIO7 on a meter chip. This function is useful as a scope loop to discover if the chip resets when expected.

The Keil project file is `RESET_PULSES_DIO7.UV2`.

5.18.6 Testing the Security Bit

This Test Module is written in 8051 assembler and is executed after processor reset. It sets the security bit and then displays the security bit on DIO_7. It is useful to test the behavior of the security bit under various system conditions.

The Keil project file is RESET_READ_SE.UV2.

5.18.7 Software Timer Test

This project, consisting of several files, demonstrates the use and test of the software timer using a hardware timer that is multiplexed into many slower timers.

The Keil project file is stmtest.uv2.

5.18.8 Interrupt Test

This Test Module is written in 8051 assembler and can be used for testing the function of the INT0, INT1, TMR0, and TMR1 control using DIO_Rx. When the code is run it configures all possible DIO pins as DIO input pins. When testing, a breakpoint should be set on the vector for one of INT0, INT1, TMR0, TMR1. Also, one of the I/O RAM registers DIO_R0...DIO_R11 should be set to the resource code for that vector. When the DIO pin under test is probed with GND or V3P3, the programmed interrupt is generated.

The code sets up DIO 4, 5, 6 and 7 for one each of four interrupts.

The Keil project file is inttest.uv2.

6

6 80515 MPU REFERENCE

An 80515 core is implemented on the TERIDIAN 71M652X chips. This section is intended for software engineers who plan to use the 80515.

6.1 80515 OVERVIEW

The 80515 is a fast single-chip 8-bit micro controller (MPU) core. It is a fully functional 8-bit embedded controller that executes all ASM51 instructions and has the same instruction set as the 80C31. The 80515 provides software and hardware interrupts, an interface for serial communications, a timer system and a watchdog timer.

6.1.1 80515 Performance

The architecture eliminates redundant bus states and implements parallel execution of fetch and execution phases. Normally a cycle is aligned with a memory fetch, therefore, most of the 1-byte instructions are performed in a single cycle. The 80515 uses 1 clock per cycle leading to an 8x performance improvement (in terms of MIPS) over the Intel 8051 device running at the same clock frequency.

Note: The original 8051 had a 12-clock architecture. A machine cycle needed 12 clocks and most instructions were either one or two machine cycles. Thus, except for the MUL and DIV instructions, the 8051 used either 12 or 24 clocks for each instruction. Furthermore, each cycle in the 8051 used two memory fetches. In many cases the second fetch was a dummy, and extra clocks were wasted.

Table 6-1 shows the speed advantage of the 80515 over the standard 8051. A speed advantage of 12 means that the 80515 performs the same instruction twelve times faster that the 8051.

Speed advantage	Number of instructions	Number of opcodes
24	1	1
12	27	83
9.6	2	2
8	16	38
6	44	89
4.8	1	2
4	18	31
3	2	9
Average: 8.0	Sum: 111	Sum: 255

Table 6-1: Speed Advantage Summary

The average speed advantage is 8x, however, the actual speed improvement observed in a system will depend on the instruction mix.

6.1.2 80515 Features

Below is a summary of the 80515 features:

- Control Unit
  - 8-bit instruction decoder
  - Reduced instruction cycle time up to 12 times
- Arithmetic-Logic Unit
  - 8 bit arithmetic and logical operations
  - Boolean manipulations
  - 8 x 8 bit multiplication and 8 / 8 bit division
- 32-bit Input/Output ports
  - Four 8-bit I/O ports
  - Alternate port functions such as external interrupts and serial interfaces are separated, providing extra port pins when compared with the standard 8051
- Two 16-bit Timer/Counters
- Two Serial Peripheral Interfaces in full duplex mode, capable of parity generation
  - Synchronous mode, fixed baud rate, Serial 0 only
  - 8-bit UART mode, variable baud rate
  - 9-bit UART mode, fixed baud rate, Serial 0 only
  - 9-bit UART mode, variable baud rate
  - 7E1, 7O1, 7N2, 7E2, 7O2, 8N1, 8E1, 8O1, 8N2, 9N1 data formats supported
  - Two Internal baud rate generators independent from timers
- Interrupt Controller
  - Four priority levels with 11 interrupt sources
- 15 bit Programmable Watchdog Timer
- Internal Data Memory interface
  - Can address up to 256B of internal data memory space
- External Memory interface
  - Can address up to 64kB of external program memory (32KB, 16KB or 8KB provided on-chip)
  - Can address up to 2kB of external data memory plus 512 bytes CE DRAM
  - Separate address/data bus to allow easy connection to memories
  - Variable length code fetch and MOVC to access fast/slow program memory
  - Variable length MOVX to access fast/slow RAM or peripherals
  - Dual data pointer for fast data block transfer
- Special Function Registers interface
  - Services up to 74 external special function registers

6.2 80515 ARCHITECTURAL OVERVIEW

6.2.1 Memory organization

The 80515 MPU core incorporates the Harvard architecture with separate code and data spaces.

Memory organization in the 80515 is similar to that of the industry standard 8051. There are three memory areas: program memory (External Flash), external data memory (External RAM), and internal data memory (Internal RAM).

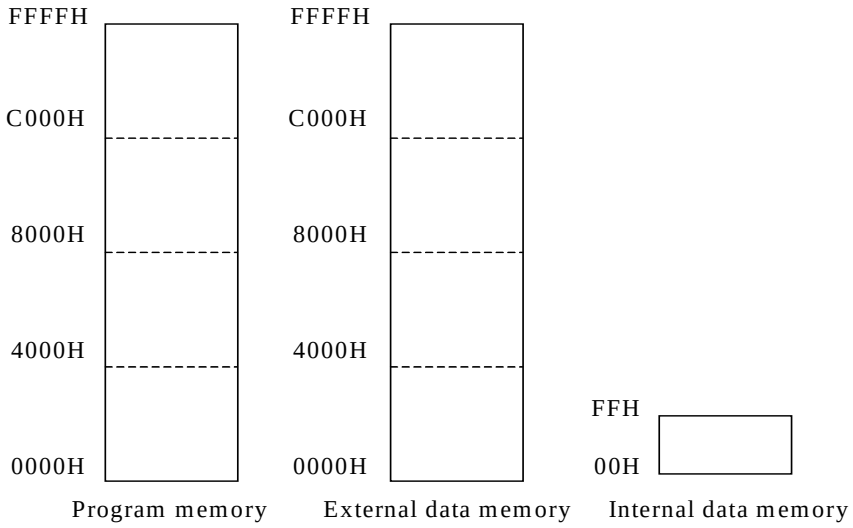


Figure 6-1: Memory Map

Program Memory

The 80515 can address up to 64kB of program memory space from 0000H to FFFFH. Program memory is read when the MPU fetches instructions or performs a MOVC.

After reset, the MPU starts program execution from location 0000H. The lower part of the program memory includes a reset and interrupt vectors. The interrupt vectors are spaced at 8-byte intervals, starting from 0003H.

External Data Memory

The 80515 can address up to 64kB of external data memory in the space from 0000H to FFFFH. The 80515 writes into external data memory when the MPU executes a MOVX @Ri,A or MOVX @DPTR,A instruction. The external data memory is read when the MPU executes a MOVX A,@Ri or MOVX A,@DPTR instruction.

There is an improved variable length access for the MOVX instructions to access fast or slow external RAM and external peripherals. The three low ordered bits of the CKCON register define the stretch memory cycles. Setting all the CKCON stretch bits to one allows access to very slow external RAM or external peripherals.

Table 6-2 shows how the signals of the External Memory Interface change when stretch values are set from 0 to 7. The widths of the signals are counted in MPU clock cycles. The post-reset state of the CKCON register, which is in bold in the table, performs the MOVX instructions with a stretch value equal to 1.

CKCON register			Stretch Value	Read signals width		Write signal width	
CKCON.2	CKCON.1	CKCON.0		memaddr	memrd	memaddr	memwr
0	0	0	0	1	1	2	1
0	0	1	1	2	2	3	1
0	1	0	2	3	3	4	2
0	1	1	3	4	4	5	3
1	0	0	4	5	5	6	4
1	0	1	5	6	6	7	5
1	1	0	6	7	7	8	6
1	1	1	7	8	8	9	7

Table 6-2: Stretch Memory Cycle Width

There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type (MOVX@Ri), the contents of R0 or R1, in the current register bank, provide the eight lower-ordered bits of address. The eight high-ordered bits of address are specified with the USR2 SFR. This method allows the user paged access (256 pages of 256 bytes each) to the full 64KB of external data RAM. In the second type of MOVX instruction (MOVX@DPTR), the data pointer generates a sixteen-bit address. This form is faster and more efficient when accessing very large data arrays (up to 64 Kbytes), since no additional instructions are needed to set up the eight high ordered bits of address.

It is possible to mix the two MOVX types. This provides the user with four separate data pointers, two with direct access and two with paged access to the entire 64KB of external memory range.

Dual Data Pointer

The Dual Data Pointer accelerates the block moves of data. The standard DPTR is a 16-bit register that is used to address external memory or peripherals. In the 80515 core the standard data pointer is called DPTR, the second data pointer is called DPTR1. The data pointer select bit chooses the active pointer. The data pointer select bit is located at the LSB of the DPS register (DPS.0). DPTR is selected when DPS.0 = 0 and DPTR1 is selected when DPS.0 = 1.

The user switches between pointers by toggling the LSB of the DPS register. All DPTR-related instructions use the currently selected DPTR for any activity.

The second data pointer may or may not be supported by certain compilers.

Internal Data Memory

The Internal data memory interface services up to 256 bytes of off-core data memory. The internal data memory address is always 1 byte wide. The memory space is 256 bytes (00H to FFH), and can be accessed by either direct or indirect addressing. The Special Function Registers occupy the upper 128 bytes. This SFR area is available only by direct addressing. Indirect addressing accesses the upper 128 bytes of Internal RAM.

The lower 128 bytes contain working registers and bit-addressable memory. The lower 32 bytes form four banks of eight registers (R0-R7). Two bits on the program memory status word (PSW) select which bank is in use. The next 16 bytes form a block of bit-addressable memory space at bit addressees 00H-7FH. All of the bytes in the lower 128 bytes are accessible through direct or indirect addressing.

Table 6-3 shows the internal data memory map.

Address	Direct addressing	Indirect addressing
0xFF	Special Function Registers (SFRs)	RAM
0x80		
0x7F	Byte-addressable area	
0x30		
0x2F	Bit-addressable area	
0x20		
0x1F	Register banks R0…R7	
0x00		

Table 6-3: Internal Data Memory Map

Special Function Registers Location

A map of the Special Function Registers is shown in Table 6-4. Only a few addresses are occupied, the others are not implemented. SFRs specific to the 652X are shown in **bold** print (see 71M652X data sheet for descriptions of these registers). Any read access to unimplemented addresses will return undefined data, while any write access will have no effect. The registers at 0x80, 0x88, 0x90, etc., are bit-addressable, all others are byte-addressable.

Hex/Bin Address	X000	X001	X010	X011	X100	X101	X110	X111	Bin/Hex Address
	Bit-addressable	Byte-addressable							
F8	**INTBITS**								FF
F0	B								F7
E8	**WDI**								EF
E0	A								E7
D8	WDCON								DF
D0	PSW								D7
C8	T2CON								CF
C0	IRCON								C7
B8	IEN	IP1	S0RELH	S1RELH				USR2	BF
B0			**FLSHCTL**					**PGADR**	B7
A8	IEN0	IP0	S0RELL						AF
A0	**DIO11 (P2)**	**DIO12 (P2)**	**DIO8 (P0)**						A7
98	S0CON	S0BUF	IEN2	S1CON	S1BUF	S1RELL	**EEDATA**	**EECTRL**	9F
90	**DIO9 (P1)**	**DIO10 (P1)**	DPS		**ERASE**				97
88	TCON	TMOD	TL0	TL1	TH0	TH1	CKCON		8F
80	**DIO7 (P0)**	SP	DPL	DPH	DPL1	DPH1	WDTREL	PCON	87

Table 6-4: Special Function Registers Locations

Generic Special Function Register Overview

All generic SFRs are explained in detail in section 6.3.2.

Register	Symbol	Description
Program status word	PSW	The PSW contains program status information
Accumulator	ACC	The accumulator register. Mmemonics for instructions involving the accumulator refer to the accumulator as A.
B register	B	This register is used for multiply and divide operations. It may also be used as a scratchpad (temporary) register.
Stack pointer	SP	The stack pointer is 8 bits wide and is incremented before data is stored with PUSH and CALL operations. SP is initialized to 0x07 after reset.
Data pointer	DPL, DPH	Since the DP consists of two bytes (DPH and DPL), it can hold a 16-bit address. It may be manipulated as a 16-bit register or as two separate 8-bit registers.
Secondary data pointer	DPL1, DPH1	This register is a second 16-bit data pointer. Note: Check with the documentation on the compiler used for generating MPU code whether this pointer is utilized or not.
Data pointer select register	DPS	This register selects which data pointer is to be used for the current operation.
Port registers	P0, P1, P2	These registers hold bit patters that are written to or read from the DIO ports
Serial data buffer	S0BUF, S1BUF	These registers hold data received from the serial interfaces 0 and 1. Data to be transmitted via the serial interfaces is written to S0BUF or S1BUF.
Serial port reload registers	S0RELL, S0RELH, S1RELL, S1RELH	These register pairs can be used to control the baud rate for the serial ports 0 and 1.
Timer registers	TL0, TL1, TH0, TH1	These register pairs (TH0/TL0 and TH1/TL1) are the 16-bit counting registers for timers 0, 1, and 2.
Interrupt control registers	IP0, IP1, IEN, IEN0, TMOD, TCON, T2CON, SCON, PCON, IRCON	These registers contain control and status bits pertaining to the interrupt system, the timers/counters, and the serial port.
Clock control register	CKCON	The clock control/configuration register. It is used to implement stretch memory cycles for memory access.
Watchdog timer reload register	WDTREL	This register holds the reload count for the software watchdog timer
Baud rate generator selector	WDCON	This register determines whether UART0 is controlled by timer 1 or by the internal baud rate generator.

Generic Special Function Registers Location and Reset Values

Table 6-5 shows the location of the SFRs and the value they assume at reset or power-up.

Register	Location	Reset value	Description
P0	80H	FFH	Port 0
SP	81H	07H	Stack Pointer
DPL	82H	00H	Data Pointer Low 0
DPH	83H	00H	Data Pointer High 0
DPL1	84H	00H	Data Pointer Low 1
DPH1	85H	00H	Data Pointer High 1
WDTREL	86H	00H	Watchdog Timer Reload register
PCON	87H	00H	Power Control
TCON	88H	00H	Timer/Counter Control
TMOD	89H	00H	Timer Mode Control
TL0	8AH	00H	Timer 0, low byte
TL1	8BH	00H	Timer 1, high byte
TH0	8CH	00H	Timer 0, low byte
TH1	8DH	00H	Timer 1, high byte
CKCON	8EH	01H	Clock Control (Stretch=1)
P1	90H	FFH	Port 1
DPS	92H	00H	Data Pointer select Register
S0CON	98H	00H	Serial Port 0, Control Register
S0BUF	99H	00H	Serial Port 0, Data Buffer
IEN2	9AH	00H	Interrupt Enable Register 2
S1CON	9BH	00H	Serial Port 1, Control Register
S1BUF	9CH	00H	Serial Port 1, Data Buffer
S1RELL	9DH	00H	Serial Port 1, Reload Register, low byte
P2	A0H	00H	Port 2
IEN0	A8H	00H	Interrupt Enable Register 0
IP0	A9H	00H	Interrupt Priority Register 0
S0RELL	AAH	D9H	Serial Port 0, Reload Register, low byte
IEN1	B8H	00H	Interrupt Enable Register 1
IP1	B9H	00H	Interrupt Priority Register 1
S0RELH	BAH	03H	Serial Port 0, Reload Register, high byte
S1RELH	BBH	03H	Serial Port 1, Reload Register, high byte
USR2	BFH	00H	User 2 Port, high address byte for MOVX@Ri
IRCON	C0H	00H	Interrupt Request Control Register
T2CON	C8H	00H	Timer 2 control register (only bits I2FR and I3FR are used)
PSW	D0H	00H	Program Status Word
WDCON	D8H	00H	Baud Rate Control Register (only WDCON.7 bit used)
A	E0H	00H	Accumulator
B	F0H	00H	B Register

Table 6-5: Special Function Registers Reset Values

Special Function Registers Specific to the 652X

Register	Alternative Name	SFR Address	R/W	Description
DIO0	DIO_0	0x80	R/W	Register for port 0 read and write operations (pins DIO0…DIO7)
DIO8	DIO_DIR0	0xA2	R/W	Data direction register for port 0. Setting a bit to 1 means that the corresponding pin is an output.
DIO9	DIO_1	0x90	R/W	Register for port 1 read and write operations (pins DIO8…DIO15)
DIO10	DIO_DIR1	0x91	R/W	Data direction register for port 1. Setting a bit to 1 means that the corresponding pin is an output.
DIO11	DIO_2	0xA0	R/W	Register for port 2 read and write operations (pins DIO16…DIO21)
DIO12	DIO_DIR2	0xA1	R/W	Data direction register for port 2. Setting a bit to 1 means that the corresponding pin is an output.
ERASE	FLSH_ERASE	0x94	W	This register is used to initiate either the Flash Mass Erase cycle or the Flash Page Erase cycle. Specific patterns are expected for FLSH_ERASE in order to initiate the appropriate Erase cycle (default = 0x00). 0x55 – Initiate Flash Page Erase cycle. Must be proceeded by a write to FLSH_PGADR @ SFR 0xB7. 0xAA – Initiate Flash Mass Erase cycle. Must be proceeded by a write to FLSH_MEEN @ sfr 0xB2 and the debug (CC) port must be enabled. Any other pattern written to FLSH_ERASE will have no effect.
PGADDR	FLSH_PGADR	0xB7	R/W	Flash Page Erase Address register containing the flash memory page address (page 0 thru 127) that will be erased during the Page Erase cycle. (default = 0x00). Must be re-written for each new Page Erase cycle.
EEDATA		0x9E	R/W	I2C EEPROM interface data register
EECTRL		0x9F	R/W	I2C EEPROM interface control register. If the MPU wishes to write a byte of data to EEPROM, it places the data in EEDATA and then writes the 'Transmit' code to EECTRL. The write to EECTRL initiates the transmit.
FLSHCRL		0xB2		This multi-purpose register contains the following bits:
			R/W	Bit 0 (FLSH_PWE): Program Write Enable: 0 – MOVX commands refer to XRAM Space, normal operation (default). 1 – MOVX @DPTR,A moves A to Program Space (Flash) @ DPTR. This bit is automatically reset after each byte written to flash. Writes to this bit are inhibited when interrupts are enabled.
			W	Bit 1 (FLSH_MEEN): Mass Erase Enable: 0 – Mass Erase disabled (default). 1 – Mass Erase enabled. Must be re-written for each new Mass Erase cycle.
			R/W	Bit 6 (SECURE): Enables security provisions that prevent external reading of flash memory and CE program RAM. This bit is reset on chip reset and may only be set. Attempts to write zero are ignored.
			R	Bit 7 (PREBOOT): Indicates that the preboot sequence is active.

WDI		0xE8	R/W R/W W	Only byte operations on the whole WDI register should be used when writing. This multi-purpose register contains the following bits: Bit 0 (IE_XFER): XFER Interrupt Flag: This flag monitors the XFER_BUSY interrupt. It is set by hardware and must be cleared by the interrupt handler Bit 1 (IE_RTC): RTC Interrupt Flag: This flag monitors the RTC_1SEC interrupt. It is set by hardware and must be cleared by the interrupt handler Bit 7 (WD_RST): WD Timer Reset: The WDT is reset when a 1 is written to this bit.
INTBITS	INT0…INT6	0xF8	R	Interrupt inputs. The MPU may read these bits to see the input to external interrupts INT0, INT1, up to INT6. These bits do not have any memory and are primarily intended for debug use

Table 6-6: SFRs Specific to the 652X

6.2.2 The 80515 Instruction Set

All 80515 instructions are binary code compatible and perform the same functions as they do with the industry standard 8051. The following tables give a summary of the instruction set cycles of the 80515 MPU core.

Table 6-7 and Table 6-8 contain notes for mnemonics used in instruction set tables.

Table 6-9 through Table 6-17 show the instruction hexadecimal codes, the number of bytes, and the number of machine cycles required for each instruction to execute.

Rn	Working register R0-R7
direct	256 internal RAM locations, any Special Function Registers
@Ri	Indirect internal or external RAM location addressed by register R0 or R1
#data	8-bit constant included in instruction
#data 16	16-bit constant included as bytes 2 and 3 of instruction
bit	256 software flags, any bit-addressable I/O pin, control or status bit
A	Accumulator

Table 6-7: Notes on Data Addressing Modes

addr16	Destination address for LCALL and LJMP may be anywhere within the 64-kB of program memory address space.
addr11	Destination address for ACALL and AJMP will be within the same 2-kB page of program memory as the first byte of the following instruction.
rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to the first byte of the following instruction

Table 6-8: Notes on Program Addressing Modes

Instructions Ordered by Function

Mnemonic	Description	Code	Bytes	Cycles
ADD A,Rn	Add register to accumulator	28-2F	1	1
ADD A,direct	Add direct byte to accumulator	25	2	2
ADD A,@Ri	Add indirect RAM to accumulator	26-27	1	2
ADD A,#data	Add immediate data to accumulator	24	2	2
ADDC A,Rn	Add register to accumulator with carry flag	38-3F	1	1
ADDC A,direct	Add direct byte to A with carry flag	35	2	2
ADDC A,@Ri	Add indirect RAM to A with carry flag	36-37	1	2
ADDC A,#data	Add immediate data to A with carry flag	34	2	2
SUBB A,Rn	Subtract register from A with borrow	98-9F	1	1
SUBB A,direct	Subtract direct byte from A with borrow	95	2	2
SUBB A,@Ri	Subtract indirect RAM from A with borrow	96-97	1	2
SUBB A,#data	Subtract immediate data from A with borrow	94	2	2
INC A	Increment accumulator	04	1	1
INC Rn	Increment register	08-0F	1	2
INC direct	Increment direct byte	05	2	3
INC @Ri	Increment indirect RAM	06-07	1	3
INC DPTR	Increment data pointer	A3	1	1
DEC A	Decrement accumulator	14	1	1
DEC Rn	Decrement register	18-1F	1	2
DEC direct	Decrement direct byte	15	2	3
DEC @Ri	Decrement indirect RAM	16-17	1	3
MUL AB	Multiply A and B	A4	1	5
DIV	Divide A by B	84	1	5
DA A	Decimal adjust accumulator	D4	1	1

Table 6-9: Arithmetic Operations

Mnemonic	Description	Code	Bytes	Cycles
ANL A,Rn	AND register to accumulator	58-5F	1	1
ANL A,direct	AND direct byte to accumulator	55	2	2
ANL A,@Ri	AND indirect RAM to accumulator	56-57	1	2
ANL A,#data	AND immediate data to accumulator	54	2	2
ANL direct,A	AND accumulator to direct byte	52	2	3
ANL direct,#data	AND immediate data to direct byte	53	3	4
ORL A,Rn	OR register to accumulator	48-4F	1	1
ORL A,direct	OR direct byte to accumulator	45	2	2
ORL A,@Ri	OR indirect RAM to accumulator	46-47	1	2
ORL A,#data	OR immediate data to accumulator	44	2	2
ORL direct,A	OR accumulator to direct byte	42	2	3
ORL direct,#data	OR immediate data to direct byte	43	3	4
XRL A,Rn	Exclusive OR register to accumulator	68-6F	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	65	2	2
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	66-67	1	2
XRL A,#data	Exclusive OR immediate data to accumulator	64	2	2
XRL direct,A	Exclusive OR accumulator to direct byte	62	2	3
XRL direct,#data	Exclusive OR immediate data to direct byte	63	3	4
CLR A	Clear accumulator	E4	1	1
CPL A	Complement accumulator	F4	1	1
RL A	Rotate accumulator left	23	1	1
RLC A	Rotate accumulator left through carry	33	1	1
RR A	Rotate accumulator right	03	1	1
RRC A	Rotate accumulator right through carry	13	1	1
SWAP A	Swap nibbles within the accumulator	C4	1	1

Table 6-10: Logic Operations

Mnemonic	Description	Code	Bytes	Cycles
MOV A,Rn	Move register to accumulator	E8-EF	1	1
MOV A,direct	Move direct byte to accumulator	E5	2	2
MOV A,@Ri	Move indirect RAM to accumulator	E6-E7	1	2
MOV A,#data	Move immediate data to accumulator	74	2	2
MOV Rn,A	Move accumulator to register	F8-FF	1	2
MOV Rn,direct	Move direct byte to register	A8-AF	2	4
MOV Rn,#data	Move immediate data to register	78-7F	2	2
MOV direct,A	Move accumulator to direct byte	F5	2	3
MOV direct,Rn	Move register to direct byte	88-8F	2	3
MOV direct1,direct2	Move direct byte to direct byte	85	3	4
MOV direct,@Ri	Move indirect RAM to direct byte	86-87	2	4
MOV direct,#data	Move immediate data to direct byte	75	3	3
MOV @Ri,A	Move accumulator to indirect RAM	F6-F7	1	3
MOV @Ri,direct	Move direct byte to indirect RAM	A6-A7	2	5
MOV @Ri,#data	Move immediate data to indirect RAM	76-77	2	3
MOV DPTR,#data16	Load data pointer with a 16-bit constant	90	3	3
MOVC A,@A+DPTR	Move code byte relative to DPTR to accumulator	93	1	3
MOVC A,@A+PC	Move code byte relative to PC to accumulator	83	1	3
MOVX A,@Ri	Move external RAM (8-bit addr.) to A	E2-E3	1	3-10
MOVX A,@DPTR	Move external RAM (16-bit addr.) to A	E0	1	3-10
MOVX @Ri,A	Move A to external RAM (8-bit addr.)	F2-F3	1	4-11
MOVX @DPTR,A	Move A to external RAM (16-bit addr.)	F0	1	4-11
PUSH direct	Push direct byte onto stack	C0	2	4
POP direct	Pop direct byte from stack	D0	2	3
XCH A,Rn	Exchange register with accumulator	C8-CF	1	2
XCH A,direct	Exchange direct byte with accumulator	C5	2	3
XCH A,@Ri	Exchange indirect RAM with accumulator	C6-C7	1	3
XCHD A,@Ri	Exchange low-order nibble indirect RAM with A	D6-D7	1	3

Table 6-11: Data Transfer Operations

Mnemonic	Description	Code	Bytes	Cycles
ACALL addr11	Absolute subroutine call	xxx11	2	6
LCALL addr16	Long subroutine call	12	3	6
RET	Return from subroutine	22	1	4
RETI	Return from interrupt	32	1	4
AJMP addr11	Absolute jump	xxx01	2	3
LJMP addr16	Long jump	02	3	4
SJMP rel	Short jump (relative addr.)	80	2	3
JMP @A+DPTR	Jump indirect relative to the DPTR	73	1	2
JZ rel	Jump if accumulator is zero	60	2	3
JNZ rel	Jump if accumulator is not zero	70	2	3
JC rel	Jump if carry flag is set	40	2	3
JNC	Jump if carry flag is not set	50	2	3
JB bit,rel	Jump if direct bit is set	20	3	4
JNB bit,rel	Jump if direct bit is not set	30	3	4
JBC bit,direct rel	Jump if direct bit is set and clear bit	10	3	4
CJNE A,direct rel	Compare direct byte to A and jump if not equal	B5	3	4
CJNE A,#data rel	Compare immediate to A and jump if not equal	B4	3	4
CJNE Rn,#data rel	Compare immed. to reg. and jump if not equal	B8-BF	3	4
CJNE @Ri,#data rel	Compare immed. to ind. and jump if not equal	B6-B7	3	4
DJNZ Rn,rel	Decrement register and jump if not zero	D8-DF	2	3
DJNZ direct,rel	Decrement direct byte and jump if not zero	D5	3	4
NOP	No operation	00	1	1

Table 6-12: Program Branches

Mnemonic	Description	Code	Bytes	Cycles
CLR C	Clear carry flag	C3	1	1
CLR bit	Clear direct bit	C2	2	3
SETB C	Set carry flag	D3	1	1
SETB bit	Set direct bit	D2	2	3
CPL C	Complement carry flag	B3	1	1
CPL bit	Complement direct bit	B2	2	3
ANL C,bit	AND direct bit to carry flag	82	2	2
ANL C,/bit	AND complement of direct bit to carry	B0	2	2
ORL C,bit	OR direct bit to carry flag	72	2	2
ORL C,/bit	OR complement of direct bit to carry	A0	2	2
MOV C,bit	Move direct bit to carry flag	A2	2	2
MOV bit,C	Move carry flag to direct bit	92	2	3

Table 6-13: Boolean Manipulations

Instructions Ordered by Opcode (Hexadecimal)

Opcode	Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic
0x00	NOP	0x20	JB bit.rel	0x40	JC rel
0x01	AJMP addr11	0x21	AJMP addr11	0x41	AJMP addr11
0x02	LJMP addr16	0x22	RET	0x42	ORL direct,A
0x03	RR A	0x23	RL A	0x43	ORL direct,#data
0x04	INC A	0x24	ADD A,#data	0x44	ORL A,#data
0x05	INC direct	0x25	ADD A,direct	0x45	ORL A,direct
0x06	INC @R0	0x26	ADD A,@R0	0x46	ORL A,@R0
0x07	INC @R1	0x27	ADD A,@R1	0x47	ORL A,@R1
0x08	INC R0	0x28	ADD A,R0	0x48	ORL A,R0
0x09	INC R1	0x29	ADD A,R1	0x49	ORL A,R1
0x0A	INC R2	0x2A	ADD A,R2	0x4A	ORL A,R2
0x0B	INC R3	0x2B	ADD A,R3	0x4B	ORL A,R3
0x0C	INC R4	0x2C	ADD A,R4	0x4C	ORL A,R4
0x0D	INC R5	0x2D	ADD A,R5	0x4D	ORL A,R5
0x0E	INC R6	0x2E	ADD A,R6	0x4E	ORL A,R6
0x0F	INC R7	0x2F	ADD A,R7	0x4F	ORL A,R7
0x10	JBC bit,rel	0x30	JNB bit.rel	0x50	JNC rel
0x11	ACALL addr11	0x31	ACALL addr11	0x51	ACALL addr11
0x12	LCALL addr16	0x32	RETI	0x52	ANL direct,A
0x13	RRC A	0x33	RLC A	0x53	ANL direct,#data
0x14	DEC A	0x34	ADDC A,#data	0x54	ANL A,#data
0x15	DEC direct	0x35	ADDC A,direct	0x55	ANL A,direct
0x16	DEC @R0	0x36	ADDC A,@R0	0x56	ANL A,@R0
0x17	DEC @R1	0x37	ADDC A,@R1	0x57	ANL A,@R1
0x18	DEC R0	0x38	ADDC A,R0	0x58	ANL A,R0
0x19	DEC R1	0x39	ADDC A,R1	0x59	ANL A,R1
0x1A	DEC R2	0x3A	ADDC A,R2	0x5A	ANL A,R2
0x1B	DEC R3	0x3B	ADDC A,R3	0x5B	ANL A,R3
0x1C	DEC R4	0x3C	ADDC A,R4	0x5C	ANL A,R4
0x1D	DEC R5	0x3D	ADDC A,R5	0x5D	ANL A,R5
0x1E	DEC R6	0x3E	ADDC A,R6	0x5E	ANL A,R6
0x1F	DEC R7	0x3F	ADDC A,R7	0x5F	ANL A,R7

Table 6-14: Instruction Set in Hexadecimal Order

Opcode	Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic
0x60	JZ rel	0x80	SJMP rel	0xA0	ORL C,bit
0x61	AJMP addr11	0x81	AJMP addr11	0xA1	AJMP addr11
0x62	XRL direct,A	0x82	ANL C,bit	0xA2	MOV C,bit
0x63	XRL direct,#data	0x83	MOVC A,@A+PC	0xA3	INC DPTR
0x64	XRL A,#data	0x84	DIV AB	0xA4	MUL AB
0x65	XRL A,direct	0x85	MOV direct,direct	0xA5	Reserved
0x66	XRL A,@R0	0x86	MOV direct,@R0	0xA6	MOV @R0,direct
0x67	XRL A,@R1	0x87	MOV direct,@R1	0xA7	MOV @R1,direct
0x68	XRL A,R0	0x88	MOV direct,R0	0xA8	MOV R0,direct
0x69	XRL A,R1	0x89	MOV direct,R1	0xA9	MOV R1,direct
0x6A	XRL A,R2	0x8A	MOV direct,R2	0xAA	MOV R2,direct
0x6B	XRL A,R3	0x8B	MOV direct,R3	0xAB	MOV R3,direct
0x6C	XRL A,R4	0x8C	MOV direct,R4	0xAC	MOV R4,direct
0x6D	XRL A,R5	0x8D	MOV direct,R5	0xAD	MOV R5,direct
0x6E	XRL A,R6	0x8E	MOV direct,R6	0xAE	MOV R6,direct
0x6F	XRL A,R7	0x8F	MOV direct,R7	0xAF	MOV R7,direct
0x70	JNZ rel	0x90	MOV DPTR,#data16	0xB0	ANL C,bit
0x71	ACALL addr11	0x91	ACALL addr11	0xB1	ACALL addr11
0x72	ORL C,direct	0x92	MOV bit,C	0xB2	CPL bit
0x73	JMP @A+DPTR	0x93	MOVC A,@A+DPTR	0xB3	CPL C
0x74	MOV A,#data	0x94	SUBB A,#data	0xB4	CJNE A,#data,rel
0x75	MOV direct,#data	0x95	SUBB A,direct	0xB5	CJNE A,direct,rel
0x76	MOV @R0,#data	0x96	SUBB A,@R0	0xB6	CJNE @R0,#data,rel
0x77	MOV @R1,#data	0x97	SUBB A,@R1	0xB7	CJNE @R1,#data,rel
0x78	MOV R0.#data	0x98	SUBB A,R0	0xB8	CJNE R0,#data,rel
0x79	MOV R1.#data	0x99	SUBB A,R1	0xB9	CJNE R1,#data,rel
0x7A	MOV R2.#data	0x9A	SUBB A,R2	0xBA	CJNE R2,#data,rel
0x7B	MOV R3.#data	0x9B	SUBB A,R3	0xBB	CJNE R3,#data,rel
0x7C	MOV R4.#data	0x9C	SUBB A,R4	0xBC	CJNE R4,#data,rel
0x7D	MOV R5.#data	0x9D	SUBB A,R5	0xBD	CJNE R5,#data,rel
0x7E	MOV R6.#data	0x9E	SUBB A,R6	0xBE	CJNE R6,#data,rel
0x7F	MOV R7.#data	0x9F	SUBB A,R7	0xBF	CJNE R7,#data,rel

Table 6-15: Instruction Set in Hexadecimal Order

Opcode	Mnemonic	Opcode	Mnemonic
0xC0	PUSH direct	0xD0	POP direct
0xC1	AJMP addr11	0xD1	ACALL addr11
0xC2	CLR bit	0xD2	SETB bit
0xC3	CLR C	0xD3	SETB C
0xC4	SWAP A	0xD4	DA A
0xC5	XCH A,direct	0xD5	DJNZ direct,rel
0xC6	XCH A,@R0	0xD6	XCHD A,@R0
0xC7	XCH A,@R1	0xD7	XCHD A,@R1
0xC8	XCH A,R0	0xD8	DJNZ R0,rel
0xC9	XCH A,R1	0xD9	DJNZ R1,rel
0xCA	XCH A,R2	0xDA	DJNZ R2,rel
0xCB	XCH A,R3	0xDB	DJNZ R3,rel
0xCC	XCH A,R4	0xDC	DJNZ R4,rel
0xCD	XCH A,R5	0xDD	DJNZ R5,rel
0xCE	XCH A,R6	0xDE	DJNZ R6,rel
0xCF	XCH A,R7	0xDF	DJNZ R7,rel
0xE0	MOVX A,@DPTR	0xF0	MOVX @DPTR,A
0xE1	AJMP addr11	0xF1	ACALL addr11
0xE2	MOVX A,@R0	0xF2	MOVX @R0,A
0xE3	MOVX A,@R1	0xF3	MOVX @R1,A
0xE4	CLR A	0xF4	CPL A
0xE5	MOV A,direct	0xF5	MOV direct,A
0xE6	MOV A,@R0	0xF6	MOV @R0,A
0xE7	MOV A,@R1	0xF7	MOV @R1,A
0xE8	MOV A,R0	0xF8	MOV R0,A
0xE9	MOV A,R1	0xF9	MOV R1,A
0xEA	MOV A,R2	0xFA	MOV R2,A
0xEB	MOV A,R3	0xFB	MOV R3,A
0xEC	MOV A,R4	0xFC	MOV R4,A
0xED	MOV A,R5	0xFD	MOV R5,A
0xEE	MOV A,R6	0xFE	MOV R6,A
0xEF	MOV A,R7	0xFF	MOV R7,A

Table 6-16: Instruction Set in Hexadecimal Order

Instructions that Affect Flags

Instruction	Affected Flag			Instruction	Affected Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C, bit	X		
MUL	0	X		ANL C, /bit	X		
DIV	0	X		ORL C, bit	X		
DA	X			ORL C, /bit	X		
RRC	X			MOV C, bit	X		
RLC	X			CJNE	X		
SETB C	1						

Table 6-17: Instructions Affecting Flags

Note: Operations affecting the PSW or bits in the PSW will also affect flag settings

6.3 80515 HARDWARE DESCRIPTION

The 80515 core implemented in the 71M652X chips consists of:

1. Control processor unit (CPU), also referred to as MPU throughout this document
2. Arithmetic-logic unit
3. Clock control unit
4. Memory control unit
5. RAM and SFR control unit
6. Ports registers unit
7. Timer 0, 1 unit
8. Serial 0, 1 interfaces
9. Watchdog timer
10. Interrupt service routine unit

6.3.1 Block Diagram

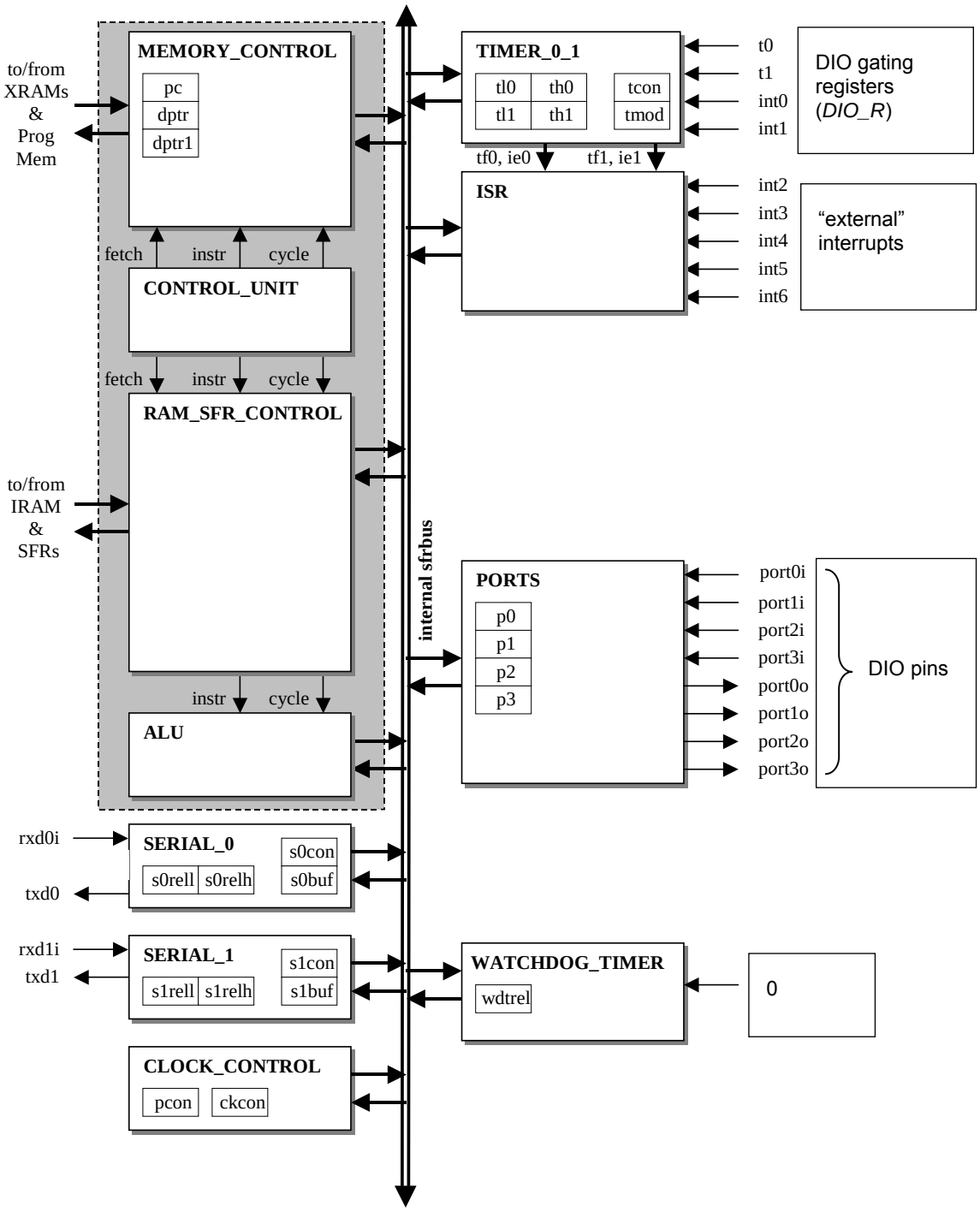


Figure 6-2: 80515 µC Block Diagram

6.3.2 80515 MPU

The 80515 MPU is composed of four components:

1. Control unit
2. Arithmetic-logic unit
3. Memory control unit
4. RAM and SFR control unit

The 80515 MPU allows instruction fetch from program memory and instruction execution using RAM or SFR. The following chapter describes the main MPU registers.

Accumulator

ACC is the accumulator register. Most instructions use the accumulator to hold the operand. The mnemonics for accumulator-specific instructions refer to accumulator as "A", not ACC.

The B Register

The B register is used during multiply and divide instructions. It can also be used as a scratch-pad register to hold temporary data.

Program Status Word (PSW)

MSB							LSB
CV	AC	F0	RS1	RS	OV	-	P

Table 6-18: PSW Register Flags

Bit	Symbol	Function
PSW.7	CV	Carry flag
Psw.6	AC	Auxiliary Carry flag for BCD operations
PSW.5	F0	General purpose Flag 0 available for user
PSW.4	RS1	Register bank select control bits. The contents of rs1 and rs0 select the working register bank as follows: (0, 0): Bank 0 (0x00-0x07) (0,1): Bank 1 (0x08-0x0F) (1, 0): Bank 2 (0x10-0x17) (1, 1): Bank 3 (0x18-0x1F)
PSW.3	RS0	
PSW.2	OV	Overflow flag
PSW.1	-	User defined flag
PSW.0	P	Parity flag, affected by hardware to indicate odd / even number of "one" bits in the Accumulator, i.e. even parity.

Table 6-19: PSW Bit Functions

The state of bits RS1 and RS0 select the working registers bank as follows:

RS1/RS0	Bank selected	Location
00	Bank 0	(00H – 07H)
01	Bank 1	(08H – 0FH)
10	Bank 2	(10H – 17H)
11	Bank 3	(18H – 1FH)

Table 6-20: Register Bank Location

Stack Pointer

The stack pointer is a 1-byte register initialized to 07H after reset. This register is incremented before PUSH and CALL instructions, causing the stack to begin at location 08H.

Data Pointer

The data pointer (DPTR) is 2 bytes wide. The lower part is DPL, and the highest is DPH. It can be loaded as a 2-byte register (MOV DPTR,#data16) or as two registers (e.g. MOV DPL,#data8). It is generally used to access external code or data space (e.g. MOVC A,@A+DPTR or MOVX A,@DPTR respectively).

Program Counter

The program counter (PC) is 2 bytes wide initialized to 0000H after reset. This register is incremented during the fetching operation code or when operating on data from program memory.

Ports

Port registers 'P0', 'P1', and 'P2' are Special Function Registers. The contents of the SFR can be observed on corresponding pins on the chip. Writing a '1' to any of the ports causes the corresponding pin to be at high level (VCC), and writing a '0' causes the corresponding pin to be held at low level (GND).

All DIO ports on the chip are bi-directional. Each of them consists of a Latch (SFR 'P0' to 'P2'), an output driver, and an input buffer, therefore the MPU can output or read data through any of these ports if they are not used for alternate purposes.

Timers 0 and 1

The 80515 has two 16-bit timer/counter registers: Timer 0 and Timer 1. These registers can be configured for counter or timer operations.

In timer mode, the register is incremented every machine cycle meaning that it counts up after every 12 periods of the MPU clock signal.

In counter mode, the register is incremented when the falling edge is observed at the corresponding input pin t0 or t1. Since it takes 2 machine cycles to recognize a 1-to-0 event, the maximum input count rate is 1/2 of the oscillator frequency. There are no restrictions on the duty cycle, however to ensure proper recognition of 0 or 1 state, an input should be stable for at least 1 machine cycle.

Four operating modes can be selected for Timer 0 and Timer 1. Two Special Function Registers (TMOD and TCON) are used to select the appropriate mode.

Timer/Counter Mode Control Register (TMOD)

MSB							LSB
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				**Timer 0**			

Table 6-21: The TMOD Register

Bit	Symbol	Function
TMOD.7 TMOD.3	Gate	If set, enables external gate control (pin int0 or int1 for Counter 0 or 1, respectively). When int0 or int1 is high, and trx bit is set (see TCON register), a counter is incremented every falling edge on t0 or t1 input pin
TMOD.6 TMOD.2	C/T	Selects Timer or Counter operation. When set to 1, a Counter operation is performed. When cleared to 0, the corresponding register will function as a Timer.
TMOD.5 TMOD.1	M1	Selects the mode for Timer/Counter 0 or Timer/Counter 1, as shown in TMOD description.
TMOD.4 TMOD.0	M0	Selects the mode for Timer/Counter 0 or Timer/Counter 1, as shown in TMOD description.

Table 6-22: The TMOD Register Bits Description

M1	M0	Mode	Function
0	0	Mode 0	13-bit Counter/Timer with 5 lower bits in the TL0 or TL1 register and the remaining 8 bits in the TH0 or TH1 register (for Timer 0 and Timer 1, respectively). The 3 high order bits of TL0 and TL1 are held at zero.
0	1	Mode 1	16-bit Counter/Timer.
1	0	Mode 2	8-bit auto-reload Counter/Timer. The reload value is kept in TH0 or TH1, while TL0 or TL1 is incremented every machine cycle. When tl(x) overflows, a value from th(x) is copied to tl(x).
1	1	Mode 3	If Timer 1 m1 and m0 bits are set to '1', Timer 1 stops. If Timer 0 m1 and m0 bits are set to '1', Timer 0 acts as two independent 8 bit Timer/Counters.

Table 6-23: Timers/Counters Mode Description

Note: In Mode 3, TL0 is affected by TR0 and gate control bits, and sets TF0 flag on overflow, while TH0 is affected by TR1 bit, and the TF1 flag is set on overflow.

Timer/Counter Control Register (TCON)

MSB							LSB
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Table 6-24: The TCON Register

Bit	Symbol	Function
TCON.7	TF1	The Timer 1 overflow flag is set by hardware when Timer 1 overflows. This flag can be cleared by software and is automatically cleared when an interrupt is processed.
TCON.6	TR1	Timer 1 Run control bit. If cleared, Timer 1 stops.
TCON.5	TF0	Timer 0 overflow flag set by hardware when Timer 0 overflows. This flag can be cleared by software and is automatically cleared when an interrupt is processed.
TCON.4	TR0	Timer 0 Run control bit. If cleared, Timer 0 stops.
TCON.3	IE1	Interrupt 1 edge flag is set by hardware when the falling edge on external pin int1 is observed. Cleared when an interrupt is processed.
TCON.2	IT1	Interrupt 1 type control bit. Selects either the falling edge or low level on input pin to cause an interrupt.
TCON.1	IE0	Interrupt 0 edge flag is set by hardware when the falling edge on external pin int0 is observed. Cleared when an interrupt is processed.
TCON.0	IT0	Interrupt 0 type control bit. Selects either the falling edge or low level on input pin to cause interrupt.

Table 6-25: The TCON Register Bit Functions

6.3.2.1 Allowed Combinations of Operation Modes

Table 6-25 specifies the combinations of operation modes allowed for timer 0 and timer 1.

	Timer 1		
	Mode 0	Mode 1	Mode 2
Timer 0 - mode 0	YES	YES	YES
Timer 0 - mode 1	YES	YES	YES
Timer 0 - mode 2	Not allowed	Not allowed	YES

Table 6-26: Timer Modes

6.3.3 Serial Interface 0 and 1

The serial buffer consists of two separate registers, a transmit buffer and a receive buffer.

Writing data to the Special Function Register S0BUF or S1BUF sets this data in the serial output buffer and starts transmission. Reading from the S0BUF or S1BUF reads data from the serial receive buffer. The serial port can simultaneously transmit and receive data. It can also buffer 1 byte at receive, preventing the receive data from being lost if the MPU reads the first byte before transmission of the second byte is completed.

Serial Interface 0 Modes

The Serial Interface 0 can operate in 4 modes:

Mode 0

Pin rxd0 serves as an input and an output. Txd0 outputs the shift clock. 8 bits are transmitted starting with the LSB. The baud rate is fixed at 1/12 of the MPU frequency. Reception is initialized in Mode 0 by setting the flags in S0CON as follows: RI0=0 and REN0=1. In other modes, when REN0 = 1, a start bit initiates receiving serial data.

Mode 1

Pin rxd0 serves as an input, and txd0 serves as a serial output. No external shift clock is used. 10 bits are transmitted: a start bit (always 0), 8 data bits (LSB first), and a stop bit (always 1). On receive, a start bit synchronizes the transmission. 8 data bits are available by reading S0BUF, and the stop bit sets the flag RB80 in the Special Function Register S0CON. In mode 1 either the internal baud rate generator or timer 1 can be use to specify the baud rate.

Mode 2

This mode is similar to Mode 1, with two differences. The baud rate is fixed at 1/32 or 1/64 of the oscillator frequency and 11 bits are transmitted or received: a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). The 9th bit can be used to control the parity of the serial interface: at transmission, bit TB80 in S0CON is output as the 9th bit, and at receive, the 9th bit affects RB80 in the Special Function Register S0CON.

Mode 3

The only difference between Mode 2 and Mode 3 is that in Mode 3, either the internal baud rate generator or timer 1 can be use to specify the baud rate.

Note: The common FLAG protocol requires the data format to be 7E1. This can be implemented using one of the 8-bit modes, where the MSB (bit 0) is the parity bit. In this mode, the MPU calculates parity

Serial Interface 0 Control Register (S0CON).

The function of the serial port 0 depends on the setting of the Serial Port Control Register S0CON.

MSB							LSB
SM0	SM1	SM20	REN0	TB80	RB80	TI0	RI0

Table 6-27: The S0CON Register

Bit	Symbol	Function
S0CON.7	SM0	Sets baud rate
S0CON.6	SM1	Sets baud rate
S0CON.5	SM20	reserved
S0CON.4	REN0	If set, enables serial reception. Cleared by software to disable reception.
S0CON.3	TB80	The 9th transmitted data bit in Modes 2 and 3. Set or cleared by the MPU, depending on the function it performs (parity check, multiprocessor communication etc.)
S0CON.2	RB80	In Modes 2 and 3 it is the 9th data bit received. In Mode 1, if SM20 is 0, RB80 is the stop bit. In Mode 0 this bit is not used. Must be cleared by software
S0CON.1	TI0	Transmit interrupt flag, set by hardware after completion of a serial transfer. Must be cleared by software.
S0CON.0	RI0	Receive interrupt flag, set by hardware after completion of a serial reception. Must be cleared by software

Table 6-28: The S0CON Bit Functions

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	Fclk/12
0	1	1	8-bit UART	Variable
1	0	2	9-bit UART	Fclk/32 or /64
1	1	3	9-bit UART	Variable

Table 6-29: Serial Port 0 Modes

Note: The speed in Mode 2 depends on the SMOD bit in the Special Function Register PCON when SMOD = 1, Fclk/32.

Serial Interface 1 Modes

The Serial Interface 1 can operate in 2 modes:

SM	Mode	Description	Baud Rate
0	A	9-bit UART	variable
1	B	8-bit UART	variable

Table 6-30: Serial 1 Modes

Mode A

This mode is similar to Mode 2 and 3 of serial interface 0. 11 bits are transmitted or received: a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). The 9th bit can be used to control the parity of the serial interface: at transmission, bit tb81 in S1CON is output as the 9th bit, and at receive, the 9th bit affects rb81 in the Special Function Register S1CON. The only difference between Mode 3 and A is that in Mode A, only the internal baud rate generator can be use to specify baud rate.

Mode B

This mode is similar to Mode 1 of serial interface 0. Pin rxd1 serves as an input, and txd1 serves as a serial output. No external shift clock is used. 10 bits are transmitted: a start bit (always 0), 8 data bits (LSB first), and a stop bit (always 1). On receive, a start bit synchronizes the transmission. 8 data bits are available by reading S1BUF, and the stop bit sets the flag rb81 in the Special Function Register S1CON. In mode B, the internal baud rate generator specifies the baud rate.

Serial Interface 1 Control Register (S1CON).

The function of the serial port depends on the setting of the Serial Port Control Register S1CON.

MSB							LSB
SM	-	SM21	REN1	TB81	RB81	TI1	RI1

Table 6-31: The S1CON Register

Bit	Symbol	Function
S1CON.7	SM	Sets baud rate
S1CON.5	SM21	Enables the multiprocessor communication feature (see description above).
S1CON.4	REN1	If set, enables serial reception. Cleared by software to disable reception.
S1CON.3	TB81	The 9th transmitted data bit in Mode A. Set or cleared by the MPU, depending on the function it performs (parity check, multiprocessor communication etc.)
S1CON.2	RB81	In Modes 2 and 3, it is the 9th data bit received. In Mode B, if sm21 is 0, rb81 is the stop bit. In Mode 0 this bit is not used. Must be cleared by software
S1CON.1	TI1	Transmit interrupt flag, set by hardware after completion of a serial transfer. Must be cleared by software.
S1CON.0	RI1	Receive interrupt flag, set by hardware after completion of a serial reception. Must be cleared by software

Table 6-32: The S1CON Bit Functions

6.3.3.1 Baud Rate generator

Serial 0 modes 1 and 3 only (Fclk = MPU clock rate):

Timer1 baud rate generator (WDCON.7 = 0)

$$baudrate = \frac{F_{clk} \cdot 2^{s\,\mathrm{mod}}}{384 \cdot (256 - th1)}$$

Internal baud rate generator (WDCON.7 = 1)

$$baudrate = \frac{F_{clk} \cdot 2^{s\,\mathrm{mod}}}{64 \cdot (2^{10} - s0rel)}$$

Note: s0rel is a 10 bit value formed by concatenating S0RELH and S0RELL as follows:

s0rel = {S0RELH.[1:0], S0RELL.[7:0]}

Serial 1 all modes: (Fclk = MPU clock rate):

Internal baud rate generator only

$$baudrate = \frac{F_{clk}}{32 \cdot (2^{10} - s1rel)}$$

Note: s1rel is a 10 bit value formed by concatenating S1RELH and S1RELL as follows:

s1rel = {S1RELH.[1:0], S1RELL.[7:0]}

6.3.4 Software Watchdog Timer

The watchdog timer is a 16-bit counter that is incremented once every 24 or 384 clock cycles. After an external reset, the watchdog timer is disabled and all registers are set to zero.

Software Watchdog Timer structure

The watchdog consists of a 16-bit counter (wdt), a reload register (WDTREL), prescalers (by 2 and by 16), and control logic.

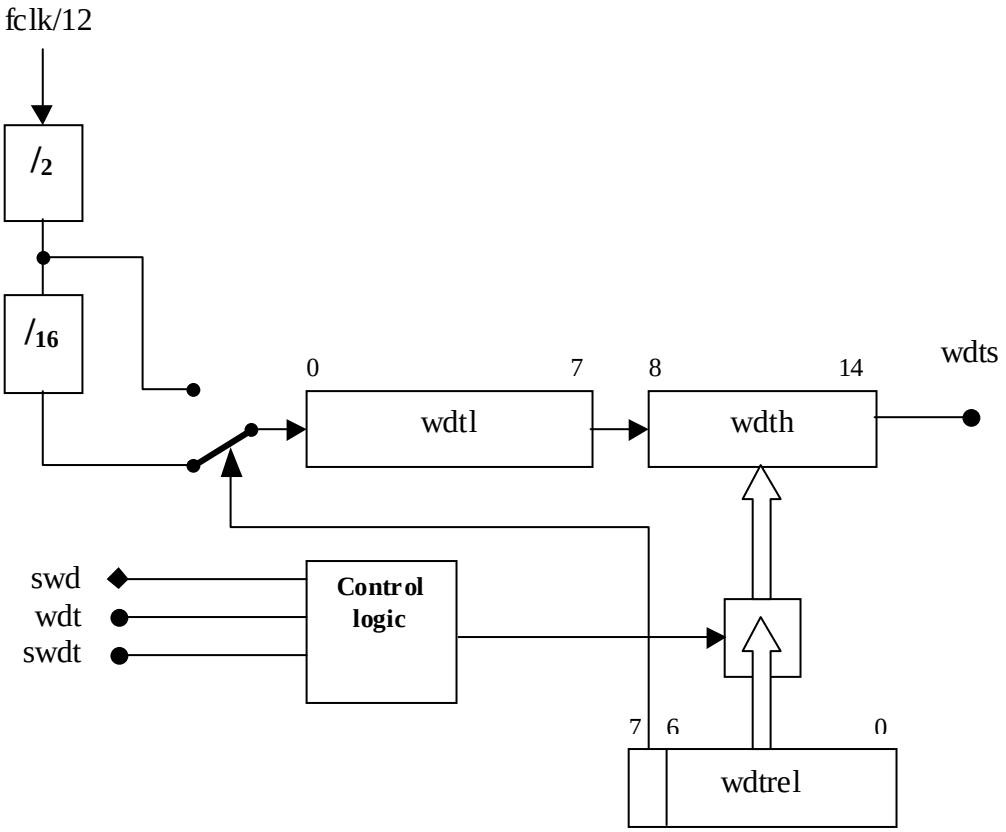


Figure 6-3: Watchdog Block Diagram

6.3.4.1 WD Timer Start Procedure

During an active internal reset signal, the programmer can start the watchdog later. It will occur when the SWD signal becomes active. Once the watchdog is started, it cannot be stopped unless the internal reset signal becomes active.

When the WDT registers enters the state 0x7CFF , an asynchronous WDTS signal will become active. The signal WDTS sets bit 6 in the IP0 register and requests a reset state. WDTS is cleared either by the reset signal or changing the state of the WDT timer.

Refreshing the WD Timer

The watchdog timer must be refreshed regularly to prevent the reset request signal from becoming active. This requirement imposes an obligation on the programmer to issue two instructions. The first instruction sets WDT and the second instruction sets SWDT. The maximum delay allowed between setting WDT and SWDT is 12 clock cycles. If this period has expired and SWDT has not been set, WDT is automatically reset, otherwise the watchdog timer is reloaded with the content of the WDTREL register and WDT is automatically reset.

Special Function Registers for the WD Timer

Interrupt Enable 0 Register (IEN0):

MSB							LSB
EALI	WDT	ET2	ES0	ET1	EX1	ET0	EX0

Table 6-33: The IEN0 Register

Bit	Symbol	Function
IEN0.6	WDT	Watchdog timer refresh flag. Set to initiate a refresh of the watchdog timer. Must be set directly before SWDT is set to prevent an unintentional refresh of the watchdog timer. WDT is reset by hardware 12 clock cycles after it has been set.

Table 6-34: The IEN0 Bit Functions

Note: The remaining bits in the IEN0 register are not used for watchdog control

Interrupt Enable 1 Register (IEN1):

MSB							LSB
EXEN2	SWDT	EX6	EX5	EX4	EX3	EX2	

Table 6-35: The IEN1 Register

Bit	Symbol	Function
IEN1.6	SWDT	Watchdog timer start/refresh flag. Set to activate/refresh the watchdog timer. When directly set after setting WDT, a watchdog timer refresh is performed. Bit SWDT is reset by the hardware 12 clock cycles after it has been set.

Table 6-36: The IEN1 Bit Functions

Note: The remaining bits in the IEN1 register are not used for watchdog control

Interrupt Priority 0 Register (IP0):

MSB							LSB
OWDS	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0

Table 6-37: The IP0 Register

Bit	Symbol	Function

IP0.6	WDTS	Watchdog timer status flag. Set by hardware when the watchdog timer was started. Can be read by software.

Table 6-38: The IP0 Bit Functions

Note: The remaining bits in the IP0 register are not used for watchdog control

Watchdog Timer Reload Register (WDTREL):

MSB							LSB
7	6	5	4	3	2	1	0

Table 6-39: The WDTREL Register

Bit	Symbol	Function
WDTREL.7	7	Prescaler select bit. When set, the watchdog is clocked through an additional divide-by-16 prescaler
WDTREL.6 to WDTREL.0	6-0	Seven bit reload value for the high-byte of the watchdog timer. This value is loaded to the WDT when a refresh is triggered by a consecutive setting of bits WDT and SWDT.

Table 6-40: The WDTREL Bit Functions

The WDTREL register can be loaded and read at any time.

6.3.5 The Interrupt Service Routine Unit

The 80515 provides 11 interrupt sources with four priority levels. Each source has its own request flag(s) located in a special function register (TCON, IRCON, SCON). Each interrupt requested by the corresponding flag can be individually enabled or disabled by the enable bits in SFRs IEN0, IEN1, and IEN2.

6.3.5.1 Interrupt Overview

When an interrupt occurs, the MPU will vector to the predetermined address as shown in Table 6-58. Once interrupt service has begun, it can be interrupted only by a higher priority interrupt. The interrupt service is terminated by a return from instruction, "RETI". When an RETI is performed, the processor will return to the instruction that would have been next when the interrupt occurred.

When the interrupt condition occurs, the processor will also indicate this by setting a flag bit. This bit is set regardless of whether the interrupt is enabled or disabled. Each interrupt flag is sampled once per machine cycle, then samples are polled by the hardware. If the sample indicates a pending interrupt when the interrupt is enabled, then the interrupt request flag is set. On the next instruction cycle, the interrupt will be acknowledged by hardware forcing an LCALL to the appropriate vector address, if the following conditions are met:

- No interrupt of equal or higher priority is already in progress.
- An instruction is currently being executed and is not completed.
- The instruction in progress is not RETI or any write access to the registers IEN0, IEN1, IEN2, IP0 or IP1.

Interrupt response will require a varying amount of time depending on the state of the MPU when the interrupt occurs. If the MPU is performing an interrupt service with equal or greater priority, the new interrupt will not be invoked. In other cases, the response time depends on the current instruction. The fastest possible response to an interrupt is 7 machine cycles. This includes one machine cycle for detecting the interrupt and six cycles to perform the LCALL.

6.3.5.2 Special Function Registers for Interrupts

Interrupt Enable 0 Register (IE0)

MSB LSB

EAL	WDT		ES0	ET1	EX1	ET0	EX0

Table 6-41: The IEN0 Register

Bit	Symbol	Function
IEN0.7	EAL	EAL=0 – disable all interrupts
IEN0.6	WDT	Not used for interrupt control
IEN0.5	-	
IEN0.4	ES0	ES0=0 – disable UART 0 interrupt
IEN0.3	ET1	ET1=0 – disable timer 1 overflow interrupt
IEN0.2	EX1	EX1=0 – disable external interrupt 1
IEN0.1	ET0	ET0=0 – disable timer 0 overflow interrupt
IEN0.0	EX0	EX0=0 – disable external interrupt 0

Table 6-42: The IEN0 Bit Functions

Interrupt Enable 1 Register (IEN1)

MSB LSB

	SWDT	EX6	EX5	EX4	EX3	EX2	

Table 6-43: The IEN1 Register

Bit	Symbol	Function
IEN1.7	-	
IEN1.6	SWDT	Not used for interrupt control
IEN1.5	EX6	EX6=0 – disable external interrupt 6
IEN1.4	EX5	EX5=0 – disable external interrupt 5
IEN1.3	EX4	EX4=0 – disable external interrupt 4
IEN1.2	EX3	EX3=0 – disable external interrupt 3
IEN1.1	EX2	EX2=0 – disable external interrupt 2
IEN1.0	-	

Table 6-44: The IEN1 Bit Functions

Interrupt Enable 2 Register (IEN2)

MSB LSB

-	-	-	-	-	-	-	ES1

Table 6-45: The IEN2 Register

Bit	Symbol	Function
IEN2.0	ES1	ES1=0 – disable UART 1 interrupt

Table 6-46: The IEN2 Bit Functions

Timer/Counter Control Register (TCON)

MSB							LSB
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Table 6-47: The TCON Register

Bit	Symbol	Function
TCON.7	TF1	Timer 1 overflow flag
TCON.6	TR1	Not used for interrupt control
TCON.5	TF0	Timer 0 overflow flag
TCON.4	TR0	Not used for interrupt control
TCON.3	IE1	External interrupt 1 flag
TCON.2	IT1	External interrupt 1 type control bit
TCON.1	IE0	External interrupt 0 flag
TCON.0	IT0	External interrupt 0 type control bit

Table 6-48: The TCON Bit Functions

Interrupt Request Register (IRCON)

MSB							LSB
		IEX6	IEX5	IEX4	IEX3	IEX2	

Table 6-49: The IRCON Register

Bit	Symbol	Function
IRCON.7	-	
IRCON.6	-	
IRCON.5	IEX6	External interrupt 6 edge flag
IRCON.4	IEX5	External interrupt 5 edge flag
IRCON.3	IEX4	External interrupt 4 edge flag
IRCON.2	IEX3	External interrupt 3 edge flag
IRCON.1	IEX2	External interrupt 2 edge flag
IRCON.0	-	

Table 6-50: The IRCON Bit Functions

Note: Only TF0 and TF1 (timer 0 and timer 1 overflow flag) will be automatically cleared by hardware when the service routine is called (Signals t0ack and t1ack – port ISR – active high when the service routine is called).

6.3.5.3 External Interrupts

The 71M6521 MPU allows seven external interrupts. These are connected as shown in Table 46. The direction of interrupts 2 and 3 is programmable in the MPU. Interrupts 2 and 3 should be programmed for falling sensitivity, using the I2FR and I3FR bits of the T2CON register (see Table 6-50).

Interrupt Request register (T2CON)

MSB							LSB
	I3FR	I2FR					

Table 6-51: The T2CON Register

Bit	Symbol	Function
T2CON.7		
T2CON.6	I3FR	This bit controls the polarity of external interrupt 3
T2CON.5	I2FR	This bit controls the polarity of external interrupt 2
T2CON.4		
T2CON.3		
T2CON.2		
T2CON.1		
T2CON.0		

Table 6-52: The T2CON Bit Functions

6.3.5.4 Interrupt Priority Level Structure

All interrupt sources are combined in groups, as shown in Table 6-52. The priority of each group is controlled by the bits of SFR registers IP1 and IP0.

Group	IP Bits	Affected Interrupts		
0	Ip1.0, IP0.0	External interrupt 0	UART 1 interrupt	-
1	Ip1.1, IP0.1	Timer 0 interrupt	-	External interrupt 2
2	Ip1.2, IP0.2	External interrupt 1	-	External interrupt 3
3	Ip1.3, IP0.3	Timer 1 interrupt	-	External interrupt 4
4	Ip1.4, IP0.4	UART 0 interrupt	-	External interrupt 5
5	IP1.5, IP0.5	-	-	External interrupt 6

Table 6-53: Priority Level Groups

Each group of interrupt sources can be programmed individually to one of four priority levels by setting or clearing one bit in the special function register IP0 and one in IP1. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced first.

The functionality and edge polarity of the external interrupts are described in Table 6-51.

External Interrupt	Connection	Polarity	Flag Reset
0	Digital I/O High Priority	see *DIO_Rx*	automatic
1	Digital I/O Low Priority	see *DIO_Rx*	automatic
2	Comparator	falling	automatic
3	CE_BUSY	falling	automatic

4	Comparator	rising	automatic
5	EEPROM busy	falling	automatic
6	XFER_BUSY OR RTC_1SEC	falling	manual

Table 6-54: External MPU Interrupts

Enable Bit	Description
EX0	Enable external interrupt 0
EX1	Enable external interrupt 1
EX2	Enable external interrupt 2
EX3	Enable external interrupt 3
EX4	Enable external interrupt 4
EX5	Enable external interrupt 5
EX6	Enable external interrupt 6
EX_XFER	Enable XFER_BUSY interrupt
EX_RTC	Enable RTC_1SEC interrupt

Flag Bit	Description
IE0	External interrupt 0 flag
IE1	External interrupt 1 flag
IEX2	External interrupt 2 flag
IEX3	External interrupt 3 flag
IEX4	External interrupt 4 flag
IEX5	External interrupt 5 flag
IEX6	External interrupt 6 flag
IE_XFER	XFER_BUSY interrupt flag
IE_RTC	RTC_1SEC interrupt flag

Table 6-55: Control Bits for External Interrupts

SFR (special function register) enable bits must be set to permit any of these interrupts to occur. Likewise, each interrupt has its own flag bit which is set by the interrupt hardware and is reset automatically by the MPU interrupt handler (0 through 5). XFER_BUSY and RTC_1SEC, which are OR-ed together, have their own enable and flag bits in addition to the interrupt 6 enable and flag bits (see Table 6-52), and these interrupts must be cleared by the MPU software.

Interrupt Priority 0 Register (IP0)

MSB							LSB
OWDS	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0

Table 6-56: The IP0 Register

Note: OWDS, WDTS are not used for interrupt controls

Interrupt Priority 1 Register (IP1)

MSB							LSB
-	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0

Table 6-57: The IP1 Register

IP1.x	IP0.x	Priority Level
0	0	Level0 (lowest)
0	1	Level1
1	0	Level2
1	1	Level3 (highest)

Table 6-58: Priority Levels

External interrupt 0	Polling sequence
UART 1 interrupt	
Timer 0 interrupt	
External interrupt 2	

External interrupt 1	
External interrupt 3	
Timer 1 interrupt	
External interrupt 4	
UART 0 interrupt	
External interrupt 5	
External interrupt 6	

Table 6-59: Polling Sequence

6.3.5.5 Interrupt Sources and Vectors

The vectors associated with each interrupt source are displayed in Table 6-59.

Interrupt Request Flags	Interrupt Vector Address
IE0 – External interrupt 0	0003H
TF0 – Timer 0 interrupt	000BH
IE1 – External interrupt 1	0013H
TF1 – Timer 1 interrupt	001BH
RI0/TI0 – UART 0 interrupt	0023H
RI1/TIi1 – UART 1 interrupt	0083H
IEX2 – External interrupt 2	004BH
IEX3 – External interrupt 3	0053H
IEX4 – External interrupt 4	005BH
IEX5 – External interrupt 5	0063H
IEX6 – External interrupt 6	006BH

Table 6-60: Interrupt Vectors

External Interrupt Edge Detect

The external interrupts 4, 5 and 6 are activated by a positive transition. The external source must hold the request pin low (high for int2 and int3, if it is programmed to be negative transition-active) for at least one MPU clock period. Afterwards, it must be held high (low) for at least one MPU clock period to ensure the transition is recognized and the corresponding interrupt request flag is set.

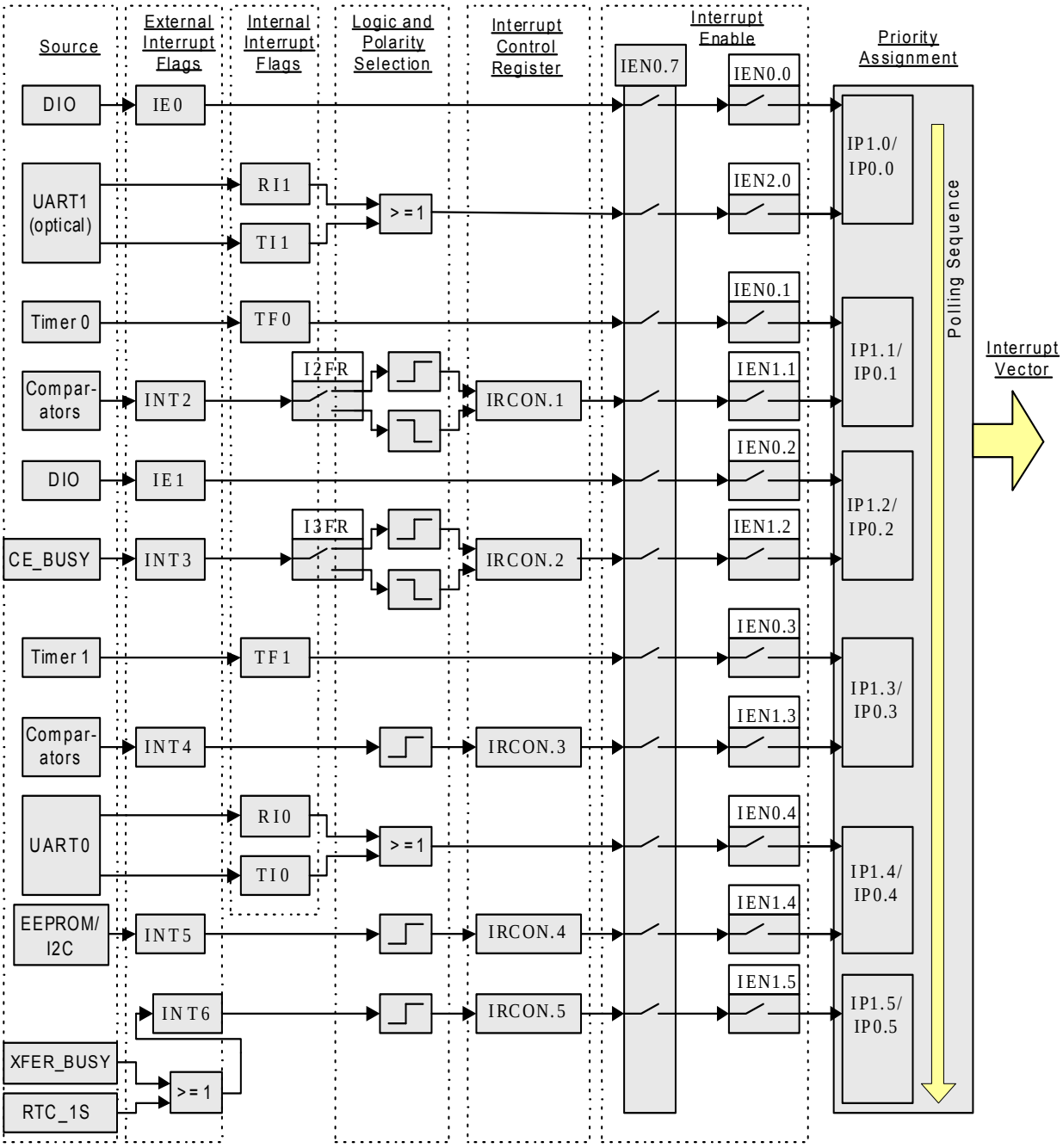


Figure 6-4: Interrupt Sources Diagram

7

7 APPENDIX

7.1 ACRONYMS

AC	Alternating Current – current with changing polarity
AMR	Automated Meter Reading, usually performed via an optical port or modem
ANSI	American National Standardization Institution, part of ISO
ANSI C	C Programming Language, standardized by ANSI in 1983. Keil C, used throughout this User's Guide is not strictly ANSI compliant.
API	Application Programming Interface
C	The C Programming Language, as defined by Kernighan and Ritchie
CE	Computation Engine
<CR>	Carriage Return or Enter Key on PC Keyboard
COM	Communication Port
CPU	Control Processor Unit (MPU)
DC	Direct Current
EEP	Engineering Evaluation Platform (Demo Board)
EEPROM	Electrically Erasable PROM
FLAG	An international protocol for reading of meters using an optical port, initially developed by Ferranti and Landis&Gyr
GB	Gigabyte(s)
ICE	In-Circuit Emulator
IDE	Integrated Development Environment – usually a combination of editor, compiler, assembler, linker, debugger, ICE
IEC	International Electrotechnical Commission (Geneva, Switzerland)
INT	Interrupt
ISO	International Standards Organization
ISR	Interrupt Service Routine
KB	Kilobyte(s) – 1,024 bytes
LCD	Liquid Crystal Display
<LF>	Line-feed character

LSB	Least Significant Bit
MB	Megabyte(s) – 1,024 kilobytes
MPU	Microprocessor/microcontroller Unit
MSB	Most Siginificant Bit
NV	Non-Volatile
PC	Personal Computer, Program Counter
PSU	Power Supply Unit
PSW	Program Status Word
RAM	Random Access Memory
SFR	Special Function Register (of the 8051 MPU)
TOU	Time-of-Use (variable metering tariffs usually based on time of day)
TSC	TERIDIAN Semiconductor Corporation
USB	Universal Serial Bus
VA	Volt-Amperes (apparent power unit)
VAh	Volt-Ampere-Hour (apparent energy unit)
VAR	Reactive Power
VARh	Reactive energy unit
W	Watt (power unit)
WD	Watchdog
WDT	Watchdog timer
WEMU51	The emulator control program by Signum Systems
Wh	Watt-Hour (energy unit)

7.2 REVISION HISTORY

Revision	Date	Description
1.0		Initial release
1.1		
1.2		
1.3	July 11, 2006	Added description of TCON2 register in MPU section.
1.4	August 11, 2006	Improved formatting and numbering, deleted reference to CE development tools.
1.5	October 12, 2006	Added explanation on interrupt priorities. Fixed interrupt priorities table.
1.6	May 15, 2007	Deleted list of CLI commands and description of hex records (all this information is contained in the 6521 DBUM). Added explanation on handling battery modes. Fixed formulae for baud rate generator and diagram "meter_LCD". Updated SW revision to 4.3.4 in compatibility statement.
1.7	August 6, 2008	Updated SW revision to 4.7a in compatibility statement. Eliminated references to ROM code. Corrected baud rate for CLI. Updated variable and routine names to match usage in revision 4.7a. Completely revised chapter 5.6 (Data Flow). Updated Teridian street address.

Software User Guide: This User Guide contains proprietary product definition information of TERIDIAN Semiconductor Corporation (TSC) and is made available for informational purposes only. TERIDIAN assumes no obligation regarding future manufacture, unless agreed to in writing.

If and when manufactured and sold, this product is sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement and limitation of liability. TERIDIAN Semiconductor Corporation (TSC) reserves the right to make changes in specifications at any time without notice. Accordingly, the reader is cautioned to verify that a data sheet is current before placing orders. TSC assumes no liability for applications assistance.

TERIDIAN Semiconductor Corp., 6440 Oak Canyon Rd., Suite 100, Irvine, CA 92618-5201
TEL (714) 508-8800, FAX (714) 508-8877, http://www.teridian.com

 8/6/2008