Application Note

AN2262/D
Rev. 1, 6/2002

Wireless HC08 Modem

MOTOROLA intelligence everywhere™ | digital dna™

By Pavel Lajsner
Motorola Czech System Application Laboratory
Roznov pod Radhostem, Czech Republic

Introduction

This document details the hardware and software required for a low-cost wireless communication application based on:

- The Motorola 8-bit microcontroller (MCU), MC68HC908GP32
- Accompanied by the Motorola RF (radio frequency) chip set:
  - MC33493 (transmitter)[1]
  - MC33591/3 (receiver)

The main purpose of this document is to demonstrate the ability of the 8-bit MCU to be connected by means of a wireless medium. Or, in other words, how you can connect several HC08s forming a simple wireless network.

The RF communication is analyzed first, then the configuration of the chips, the RF protocol, and finally the software techniques are discussed. All embedded software was written using the Hi-Ware ANSI-C/cC++ Compiler for HC08 from Metrowerks®[2] Software (http://www.metrowerks.com).

Design Overview

'The wireless HC08 modem' is targeted as one possible communication medium for a complete 'home interconnectivity' system.

1. The MC33491 (code name Tango II) is no longer available. The MC33493 (code name Tango III) is it's full replacement. For a more detailed description, refer to the footnote on **page 4** and to the **MC33491 to MC33493 Migration Notes on page 36**.
2. Metrowerks is a registered trademark of Metrowerks, Inc., a wholly owned subsidiary of Motorola, Inc.

The example system (shown in **Figure 1**) is based on gateway which communicates with the Internet on one side and with numerous embedded devices (e.g., at home) on the other side.

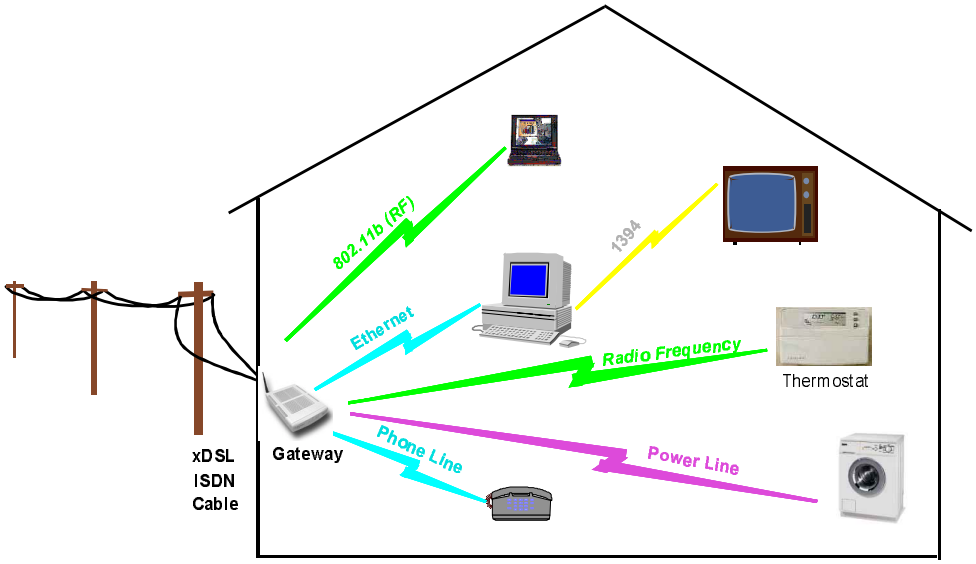


Figure 1. Home Interconnectivity System Overview

The main idea on the background is that it's not necessary to connect all such small devices directly over IP (Internet protocol). Although well-established and known, IP has several disadvantages which don't allow its effective implementation for low-bandwidth media and/or for devices with very limited resources (mainly RAM/ROM storage). Also, the overhead during transfer of small amounts of data is pretty heavy and not suitable for low-bandwidth links.

It is wiser then to implement the gateway between 'the IP world' (e.g., the Internet) and the world of many embedded devices located in one niche (e.g., in one household). Besides many more advantages (like centralized security and central access rights management), simpler protocols and 'cheaper RF media' can be effectively used on the 'home' side. The same gateway can also serve as a central point for Internet access for a home LAN (local area network) but this is out of the scope of this document.

The low-cost RF technology for unlicensed ISM (industrial, scientific, and medical) bands enables trouble-free installation and use of home devices that require only moderate data (both speed and amount). Simple on/off control, general setup, or basic diagnostics of white goods are typical examples of such communication. Baud rates of 9600–19200 bps with 100 ms + latency are usual and quite acceptable for the most of applications of this nature.

Hardware Platform

This section provides technical data for the RF-08 board.

Microprocessor

The heart of this application is a member of the generic 8-bit M68HC08 Motorola family — the FLASH-based MC68HC908GP32. For the RF communication the Motorola RF chip set is used (MC33491* (transmitter), MC33591/3 (receiver)).

Several typical peripherals are added as the following:

- Three light emitting diodes (LED) — red, yellow, green
- Three accompanying push buttons
- A buzzer
- An optical sensor
- A knob
- An RS232/RS485 serial interface

All these peripherals serve as typical examples of input/output devices that can be found in most applications, e.g., in white goods, home appliances, etc.

NOTE: *Two generic function LEDs and two LEDs indicating activity on the serial interface have been added.*

Radio Frequency (RF)

For wireless communication, the Motorola RF chip set is used.

- MC33493 (transmitter)[1]
- MC33591/3 (receiver)

These low-cost, single-channel chips for ISM bands were originally designed for key fob applications used in the car industry. The high integration, low component count, no RF adjustment, and low cost allow its use in such cost-sensitive applications like home connectivity.

The main features of the MC33493/MC33591/3 family are:

- Selectable 315/434/868/915 MHz operation
- Single-channel, OOK (on-off keying), or FSK (frequency shift keying) operation
- Fully-integrated voltage controller oscillator (VCO)
- Data rate: 1 to 11 kBaud
- A transmitter: MC33493[1]
  - 14-pin thin shrink small outline package (TSSOP)
  - Low current consumption: 15 mA typical in transmit mode
- A receiver: MC33591/3
  - 24-pin low-profile quad flat pack (LQFP)
  - Low current consumption: 5 mA typical in run mode
  - An image-cancelling mixer
  - Manchester coded data clock recovery
  - Fully-configurable using the serial peripheral interface (SPI)

Circuit Description

A schematic of the RF-08 demo board is provided in **Figure 2**. The MC68HC908GP32 MCU itself needs only a few external elements for its operation. A wide range of peripheral functions including a 32-Kbyte FLASH memory are integrated on-chip. For the RF-08, the 44-pin quad flat pack (QFP) version was chosen instead of the 40-pin dual in-line package (DIP).

1. The MC33491 (code name Tango II) is no longer available. The MC33493 (code name Tango III) is it's full replacement. The MC33493 input pins are not specified above V_{CC} voltage. Thus, the MC33493 cannot be used within described RF-08 schematics that employ R1 and R3 serial resistors to protect the MC33493 3-V interface. For interfacing between an 5 V MCU and 3 V MC33493 a proper level shifter must be designed or a different configuration used. For a more detailed description, refer to the **MC33491 to MC33493 Migration Notes on page 36**.

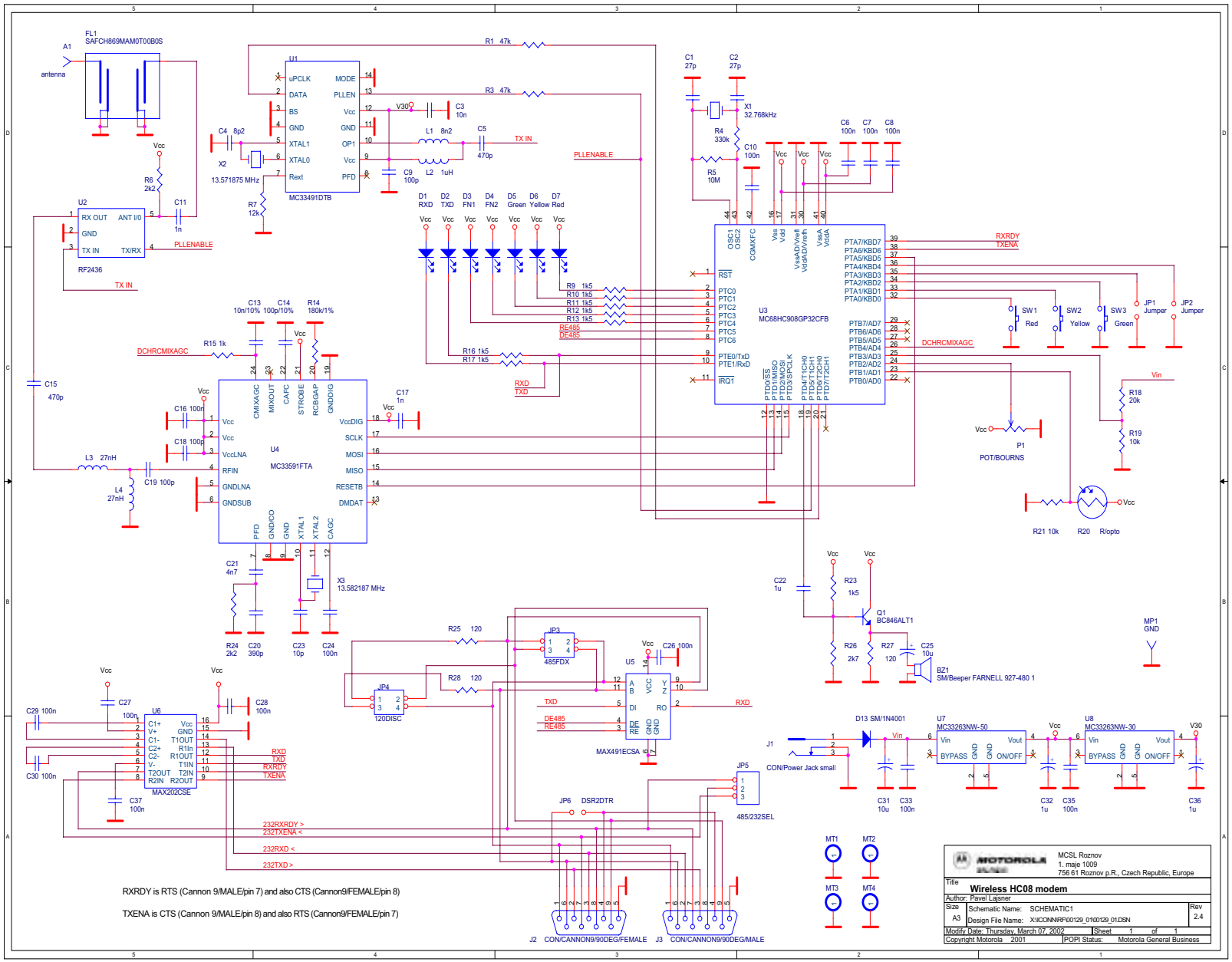


Figure 2. Application Schematic

Power Supply

The D13 diode protects the complete circuit from reverse voltage.

The MCU operating voltages $V_{DD,}$ V_{DDA}, and V_{REFH}, are supported by capacitors C6, C7, and C8 close to the MCU on their respective ground pins. They are all driven from the regulated 5-volt V_{CC} net.

The MC33491 chip requires a 3-volt power supply, provision is made for a 3-volt power supply named V_{30}.

MCU Core Circuit

For clock generation, the external elements X1, C1, C2, C10, R4, and R5 are used. These elements form a Pierce oscillator together with the active elements integrated in the MCU. This oscillator produces a clock frequency of 32.768 kHz using ultra low-cost watch quartz. The internal bus clock of the MCU (3.36864 MHz) is derived by means of the phase-locked loop (PLL) from the main clock frequency.

Input/Output Functions

For demonstration purposes, the board has:

- Three push buttons
- Two configuration jumpers
- Five + two dedicated LEDs
- An optical sensor (photo resistor)
- A knob
- A buzzer
- A power supply measurement circuit (resistor divider)

The push buttons are connected to the three port pins PTA0, PTA1, and PTA2 and the jumpers to PTA3 and PTA4. By pushing the buttons (or closing the jumpers), a low level is produced on the appropriate input. Since the port pins have internal pull-up resistors, no external resistors are required. The occurrence of the high-low edge at the respective pin of port A is an input event which may result in the generation of a keyboard interrupt by the MC68HC908GP32 (this feature is not used in ‘transparent’ software described in **Software**).

For optical signalling, five LEDs are attached to port C (pins PTC0, PTC1, PTC2, PTC3, and PTC4), three of them are identical to those used in the original NET.08 demo (they are marked **Red**, **Yellow,** and **Green**), the next two diodes are for generic use (marked **FN1** and **FN2**). These port pins have a high drive capability of up to 10 mA. Therefore, it is not necessary to use a driver.

The last two LEDs are connected to port E whose pins are shared with two serial communication interface (SCI) lines (PTE0/TXD and PTE1/RXD).
In such a configuration the user simply gets an indication of activity on the serial line.

A buzzer with amplifier circuitry (C22, C25, R23, R26, R27, and Q1) (also found on the original NET.08 board) is connected to the PTD4/T1CH0 pin. This dual function pin (beside its normal I/O function) is shared with a 16-bit timer module input/output. Here, in pulse-wide modulation (PWM) mode, it is used for tone generation.

Some of remaining port pins are used for controlling the MC33491 and MC33591/3 RF chips.

SPI module pins (PTD0/$\overline{SS}$, PTD1/MISO, PTD2/MOSI, and PTD3/SPCLK) plus two generic I/O pins (PTA5 and PTB4) interface with the MC33591/3. The SPI implementation is described in detail in **SPI Communication**

Correspondingly, two pins which are employed by the 16-bit timer module (PTD5/T1CH1 and PTD6/T2CH0) are used to drive the MC33491 transmitter. See **16-Bit Timer (TIM) Operation in Manchester Mode** which also describes an effective usage of the 16-bit timer module (TIM) for data stream generation. Resistors R1 and R3 lower the 5-volt voltage to the levels required by MC33491. One of the signals also serves as an antenna switch control. (See also **MC33491 to MC33493 Migration Notes on page 36**.)

User RS232 Ports

A provision is made on the PCB so that the user can select either the RS232 or the RS485 serial interface. For simplicity, the selection is not made by jumper or another similar method, but by assembling the relevant interface chips.

The user RS232 serial port consists of:

- A level shifter, U6
- Required capacitors C27, C28, C29, C30, and C37
- A JP5 jumper with the 2-3 position closed
- The JP6 (DSR2DTR) jumper could be closed providing simple status information for the device that is connected on the remote side of serial line. By connecting the DSR and DTR lines, a remote device may detect whether or not the RF-08 board is connected. Refer to **Table 1** and **Table 2** for a detailed explanation of all serial signals.
- The paired signals RTS and CTS provide information about whether the remote device is temporarily unable to receive data. If a serial communication uses these signals, it's called "hardware handshake" or "hardware flow control".

Table 1. RS232 Signal Mapping on J3 (Male) DB-9 Connector

Signal Name	Signal Direction	Function	Mapping on J3 (Male)
TXD	Output	Transmit data	3
RXD	Input	Receive data	2
RTS	Output	Request to send	7
CTS	Input	Clear to send	8
DTR	Output	Data terminal ready	4
DSR	Input	Data set ready	6
GND	—	Ground	5

Table 2. RS232 Signal Mapping on J2 (Female) DB-9 Connector

Signal Name	Signal Direction	Function	Mapping on J2 (Female)
TXD	Output	Transmit data	2
RXD	Input	Receive data	3
RTS	Output	Request to send	8
CTS	Input	Clear to send	7
DTR	Output	Data terminal ready	6
DSR	Input	Data set ready	4
GND	—	Ground	5

User RS485 Ports

The user RS485 current loop serial port looks simple but is relatively more complicated to connect and configure. It consists of:

- A level shifter, U5
- Required capacitor, C26
- Optional resistors, R25 and R28

Two-wire or four-wire topology can be selected. In two-wire topology, both RS485 transmitter and receiver share one pair of wires and cannot transmit and receive at the same time. Such operation is call half-duplex communication. Level shifter U5 has separate receiver enable and driver output enable pins which are connected to MCU pins PTC5 and PTC6. The RS485 transmitter must be enabled prior to transmission and disabled after the transmission is finished. Jumper JP3 connects the RS485 receiving lines to the transmitting lines creating the two-wire topology. Connect JP3's 1-2 and 3-4.

On the other side, full-duplex operation (four-wire topology) enables simultaneous communication in both directions. Thus, the transmitter and receiver are enabled at all times. Jumper JP3 must be left open.

For proper operation of the current loop, each current loop has to be terminated (i.e., effectively closing the current path back to the originating driver). Jumper JP4 allows you to connect 120-Ω resistors between RS485 receiving and transmitting lines respectively.

Connect 1-2 to add 120Ω between the RS485 transmitting lines or connect 3-4 to add 120Ω between the RS485 receiving lines.

For a detailed explanation of typical RS485 configurations refer to Maxim®(1) Integrated Products MAX485 data sheet (http://www.maxim-ic.com).

RS232 versus RS485 Configurations

For RS232 configuration:

- Populate: U6, C27, C28, C29, C30, and C37
- Don't populate: U5, C28, R25, and R28
- Leave JP3 and JP4 open
- Connect JP5 2-3
- JP6 shorted if DSR to DTR connection required

For RS485 configuration:

- RS485 half-duplex (two wires) configuration:
  - Populate: U5, C26, R25, and R28
  - Don't populate: U6, C27, C28, C29, C30, and C37
  - Connect JP3 1-2, 3-4
  - Connect JP4 1-2 if 120-Ω load required, otherwise all open
  - Connect JP5 1-2
  - Leave JP6 open
- RS485 full-duplex (four wires) configuration:
  - Populate: U5, C26, R25, and R28
  - Don't populate: U6, C27, C28, C29, C30, and C37
  - Leave JP3 open
  - Connect JP4 1-2 if 120-Ω TX load required (not usually)
  - Connect JP4 3-4 if 120-Ω RX load required (usually)
  - Connect JP5 1-2
  - Leave JP6 open

1. Maxim is a registered trademark of Maxim Integrated Products

Bill of Materials

Table 3 summarizes all components assembled for the RF-08 application:

Table 3. Bill of Materials (Sheet 1 of 3)

Quantity	Reference	Part
1	A1	Whip 868-MHz antenna See **Antenna Design**
1	BZ1	SM/Beeper (transducer) Farnell 927-480 1
2	C1, C2	27 pF
1	C3	10 nF
1	C4	8.2 pF
2	C5, C15	470 pF
14	C6, C7, C8, C10, C16, C24, C26(1), C27(2), C28(2), C29(2), C30(2), C33, C35, C37(2)	100 nF
3	C9, C18,C19	100 pF
2	C11, C17	1 nF
1	C13	10 nF/10%
1	C14	100 pF/10%
1	C20	390 pF
1	C21	4.7 nF
3	C22, C32, C36	1 µF
1	C23	10 pF
2	C25, C31	10 µF
1	D1	RXD LED
1	D2	TXD LED
1	D3	FN1 LED
1	D4	FN2 LED
1	D5	Green LED
1	D6	Yellow LED
1	D7	Red LED
1	SW3	Green
1	SW2	Yellow
1	SW1	Red
1	D13	SMD 1N4001

Table 3. Bill of Materials (Sheet 2 of 3)

Quantity	Reference	Part
1	FL1	SAFCH869MAM0T00B0S (Murata)
1	J1	CON/Power Jack small Farnell 224-674
1	J2	CON/Cannon 9 — female Farnell 892-452
1	J3	CON/Cannon 9 — male Farnell 892-439
1	L1	8.2 nH
1	L2	1 µH
2	L4, L3	27 nH
1	MP1	GND
1	P1	50 kΩ trimmer Farnell 347-346
1	Q1	BC846ALT1
2	R3, R1	47 kΩ
1	R4	330 kΩ
1	R5	10 MΩ
2	R24, R6	2.2 kΩ
1	R7	12 kΩ
8	R9, R10, R11, R12, R13, R16, R17, R23	1.5 kΩ
1	R14	180 kΩ/1%
1	R15	1 kΩ
1	R18	20 kΩ
2	R19, R21	10 kΩ
1	R20	R/opto Farnell 316-8335
3	R25(1), R27, R28(1)	120 Ω
1	R26	2.7 kΩ
1	U1	MC33491DTB (Motorola)
1	U2	RF2436 (RF Micro Devices)

Table 3. Bill of Materials (Sheet 3 of 3)

Quantity	Reference	Part
1	U3	MC68HC908GP32CFB (Motorola)
1	U4	MC33593FTA (Motorola)
1	U5(1)	MAX491ECSA (Maxim)
1	U6(2)	MAX202CSE (Maxim)
1	U7	MC33263NW-50 (ON Semiconductor)
1	U8	MC33263NW-30 (ON Semiconductor)
1	X1	32.768 kHz
1	X2	13.571875 MHz
1	X3	13.582187 MHz

1. Components are **NOT** assembled in RS232 mode. See **RS232 versus RS485 Configurations**.
2. Components are **NOT** assembled in RS485 mode. See **RS232 versus RS485 Configurations**.

NOTE: *All SMD capacitors and resistors are of standard 0805 size, unless otherwise specified.*

For power connection to J1 use Farnell 224-911 cable end.

The MC33491 is no longer available. See ***MC33491 to MC33493 Migration Notes on page 36****.*

Antenna Design

The whip antenna is the simplest design. This is a quarter wavelength wire that stands above a ground plane. The λ/4 length is purely theoretical. Depending on the nature of the wire and the geometry of the ground plane, consider the length as (k * λ/4) with k comprised between 0.93 and 0.98. For the 868-MHz band, the antenna is made of wire 84 mm long.

Some commercially available antennae are, for example, PU-BNC-868 or FLEXI-BNC-868 types from RF Solutions, Ltd. (http://www.rfsolutions.co.uk)

Application Schematic

See **Figure 2**.

PCB Layout

The RF-08 PCB is a two-layer board with solder masks on both sides. The silk screen is used only on the top (component) side since no components are assembled on the bottom side.

The board thickness as well as the copper thickness are not critical and standard sizes are used. See **Figure 3**, **Figure 4**, and **Figure 5**.

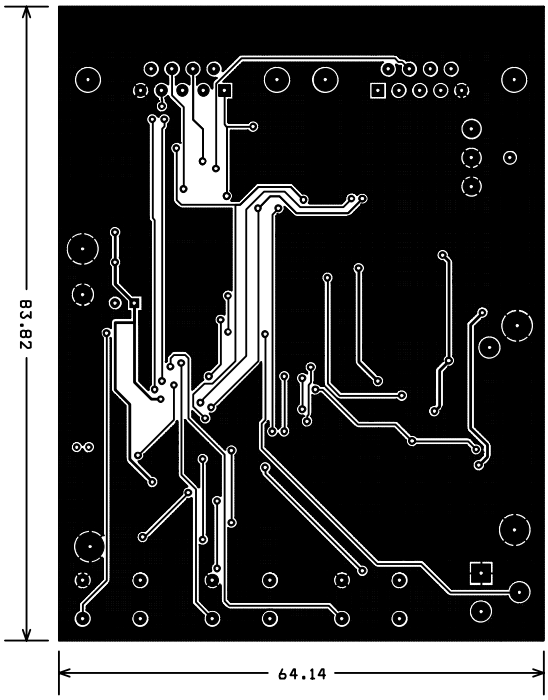


Figure 3. Bottom Copper Side

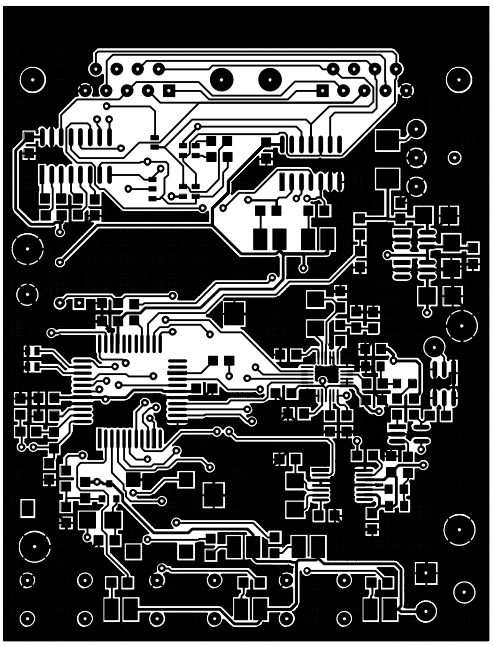


Figure 4. Top Copper Side

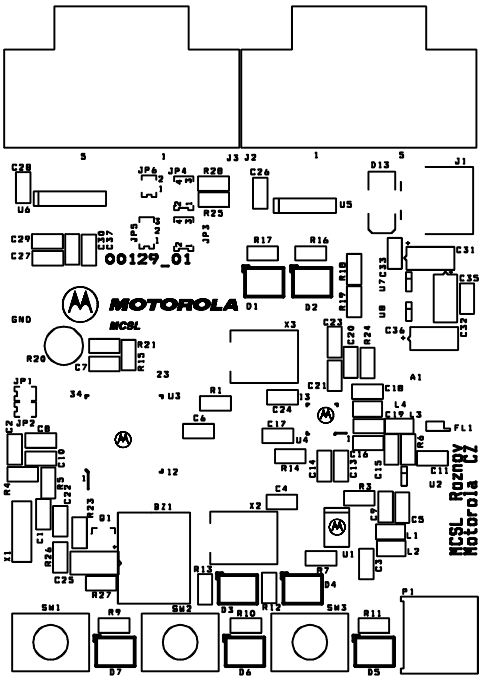


Figure 5. PCB Silk Screen

RF Chip Set MC33491/MC33493/MC33591/3 and RF Protocol

This section provides information on the RF chip set and protocol.

RF Circuitry

In this application 868-MHz RF chips are used. MC33491 (code name Tango II) is used as an ASK transmitter, MC33591/3 (code name Romeo II) is a complementary ASK receiver. The two are paired for seamless operation in the European (868 MHz) or US (915 MHz) unlicensed bands. The chips are ultra low-cost, highly integrated components originally targeted for remote keyless car entry applications.

MC33491/MC33493 Transmitter

The MC33491/MC33493 transmitter uses the configuration shown in **Figure 6**.

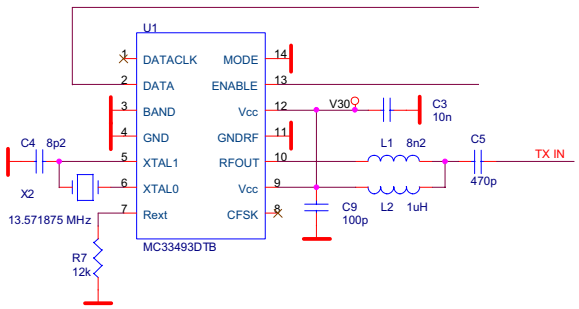


Figure 6. MC33491/MC33493 Application Schematic

Besides two decoupling capacitors (C3 and C9) only minimum components are required:

- Quartz (X2) and the required capacitor (C4)
- An impedance matching inductor (L1)
- An RF path DC decoupling capacitor (C5)
- A choke (L2) to bias output stage
- A resistor (R7) controls the output power

MC33491/MC33493 Control Mode

The MC33491/MC33493 has several operation modes. A simple control mode is useful when only on-off keying (OOK) modulation is required and the Manchester coding capability of the MC33491/MC33493 is not used. For detailed information on all modes, refer to the MC33493 Data Sheet (Motorola order number, MC33493/D).

The microcontroller drives only two pins (PLLEN and DATA). The MODE pin is wired to GND. Setting PLLEN to 1 enables the PLL; the signal sent to DATA is then directly transmitted in OOK format. See **Figure 7**.

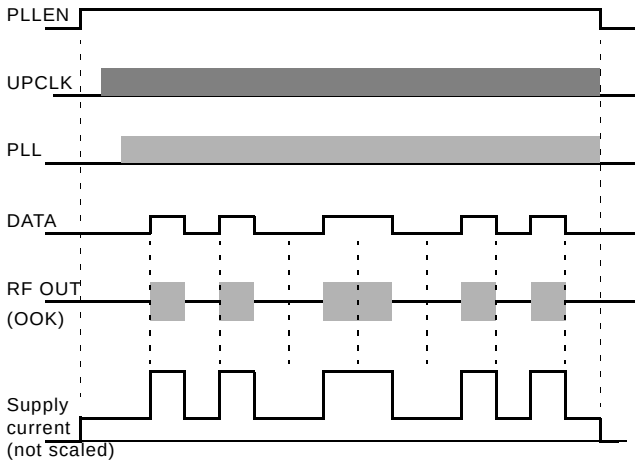


Figure 7. Simple Control Mode

This configuration is used in the RF-08 application since the interface is really minimal and the capability of the CPU allows a direct Manchester encoded data stream.

MC33591/3 Receiver

The MC33591/3 receiver uses the configuration shown in **Figure 8**. As the receiver is more complex, detailed information and application notes follow.

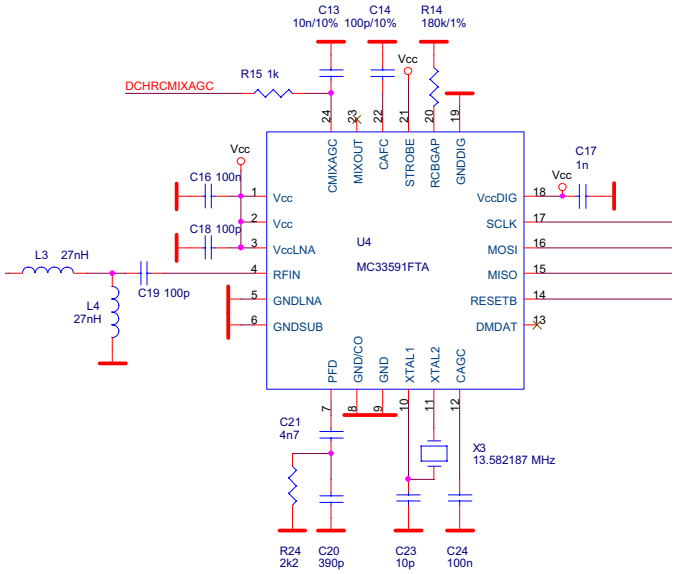


Figure 8. MC33591/3 Application Schematic

Component functions are:

- C16, C17, and C18 are decoupling capacitors
- R24, C20, and C21 are PLL filters
- L3, L4, and C19 form receive RF path impedance matching
- C14, C24, and R24 are miscellaneous external components required for correct receiver operation

The circuit around pin #24 (CMIXAGC), C13, and R15 will be explained in the following subsections.

AGC Capacitor Discharge

A new phenomena is introduced when MC33493 and MC33591/3 chips are assembled into one application. MC33591/3 (as an ASK (OOK) system) the demodulation is performed by an envelope detector. This block (and also some others along the signal path) are sensitive to the signal strength and they also employ a certain time constant. In the other words, if the signal path is saturated by a strong (i.e., local) signal, it takes a certain amount of time to establish a full sensitivity to weak (distant) signals.

To help this situation and avoid ‘deafness’ of MC33591/3 (e.g., immediately after data is sent out of the local module) a discharge circuit is designed. The C13 capacitor becomes charged depending on the signal strength. If the input signal ceases, the voltage on C13 gradually decreases.

An additional resistor, R15, is connected to an I/O pin of the microprocessor. Normally this I/O pin stays in input mode (high impedance, Hi-Z). After data is sent out and re-establishing the full receiver sensibility is required, the I/O pin is switched into output mode and the output level is set to ‘0’ (low). This effectively discharges the C13 capacitor. Then, the I/O pin function is reverted back to input mode.

SPI Interface

For configuration and received data transfer, a standard SPI is used. The receiver’s SPI lines (MOSI, MISO, and SPCLK) are connected to their respective pins of the CPU SPI module. The RESETB line, which determines between configuration and reception modes, is connected to a regular I/O pin of the CPU. For more details, refer to **SPI Communication** and to the MC33591/3 Data Sheet (Motorola order number, MC33591/D).

RF Protocol

The RF-08 application uses a very simple protocol on the RF medium. The use of the MC33591/3 receiver implies several fixed attributes of the RF protocol:

- 9600 baud maximum baud rate, ASK (OOK) modulation
- Manchester encoding
- An AGC settling field must appear prior to any reception
- An ID (identification) field must appear right before the data field
- An EOM (end-of-message) field must appear right after the data field

Since the ‘strobing’ feature is not used on the receiving side (STROBE pin is tied high, forcing the MC33591/3 to receive at all times), no provision for waking up needs to be done. Refer to the MC33591/3 Data Sheet (Motorola order number, MC33591/D) for details)

Assuming all these conditions have been met, the frames conforming to the protocol shown in **Figure 9** are transferred.

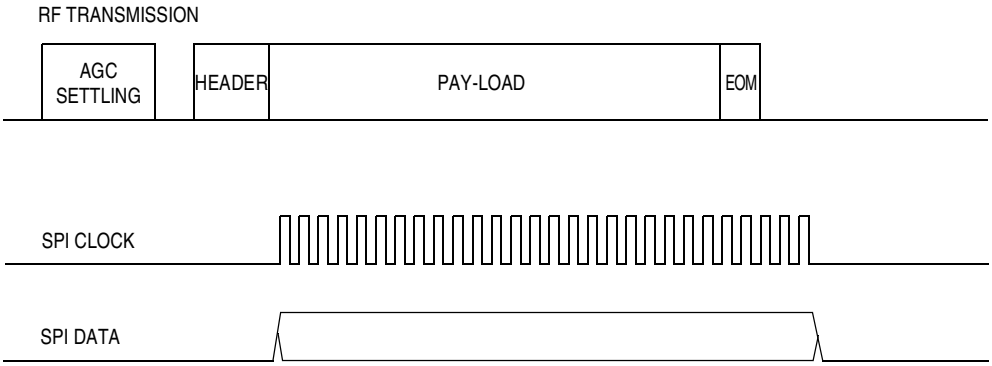


Figure 9. RF Protocol

This RF protocol defines only the physical appearance of RF frames in order to be correctly transferred using the MC33493 and MC33591/3 chips. The pay-load is then really application specific and is described in **RF-08 Protocol**.

Software

This section provides the 'transparent' software description for the RF-08 board.

Application Software Basics

The software described here provides a very basic function. It is called 'transparent mode'. The data received from the SCI (serial) line is formatted into frames and the frames are sent out via RF. See **Figure 10**.

In the opposite direction, if such a frame is received via RF its consistency is checked and the data is re-transmitted to the serial line (if correct). Such operation allows the use of simple packet (frame) oriented protocols through an RF communication channel. All frames are secured with a so-called cyclic redundancy code field which contains information that is used on the recipient's side to check whether or not the received data is consistent. If the frame is corrupted during the RF transfer it is discarded. Only good frames are transferred over the complete system.

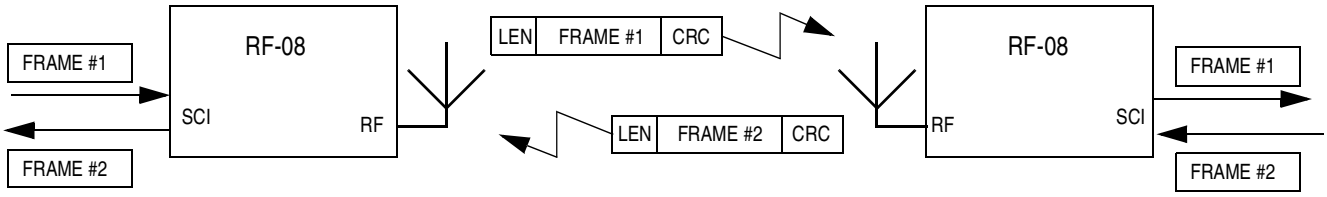


Figure 10. Frame Handling

RF Transmission

For operation of the MC33493 transmitter chip it is necessary to control two lines in the minimal configuration:

- One named PLLEN serves as the on/off control of the complete transmitter.
- The second one is called TXDATA and it actually controls the output of the transmitter (i.e., OOK (on-off keying) modulation). The Manchester encoded data stream has to appear on this line to conform to the specification of the MC33591/3 receiver. For Manchester encoding operation see **Figure 11**.

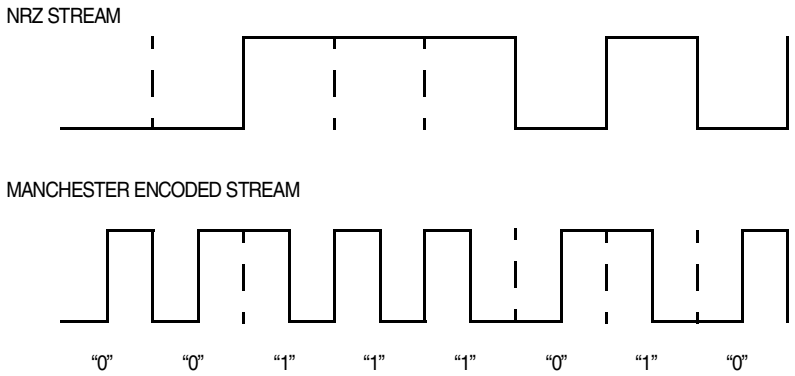


Figure 11. Manchester Encoding Explanation

The basic principle of Manchester encoding is that each bit is divided into two halves. The level in the first period is the same as in the original NRZ (non-return to zero) stream while the level in the second period is inverted. In other words, the line level is inverted in the middle of each bit interval.

Besides some disadvantages (e.g., doubled bandwidth) it brings several advantages which are beneficial especially for low-cost systems.

- First, the edge that appears in the middle of each bit can be used as a source for recovery of the received data clock as ‘it is always there’. It also helps in the decoding of the bit boundary.
- Second, the DC-level of a Manchester encoded stream remains 0 (or constant), thus independent of the data pattern. Such an attribute is often helpful in systems where the DC level is not transferred through the channel and doesn’t need to be recovered at the receiving side. This is also used in the MC33591/3 receiver.

16-Bit Timer (TIM) Operation in Manchester Mode

The timer interface module (TIM) is the peripheral found in almost all members of Motorola M68HC08 Family. TIM can operate in several modes.

Figure 12 shows the structure of the TIM. The central component of each TIM is the 16-bit TIM counter that can operate as a free-running counter or a modulo-up counter. The TIM counter provides the timing reference for the input capture and output compare functions. The TIM counter modulo registers, TMODH and TMODL, control the modulo value of the TIM counter. Software can read the TIM counter value at any time without affecting the counting sequence. The two TIM channels are independently programmable as input capture or output compare channels.

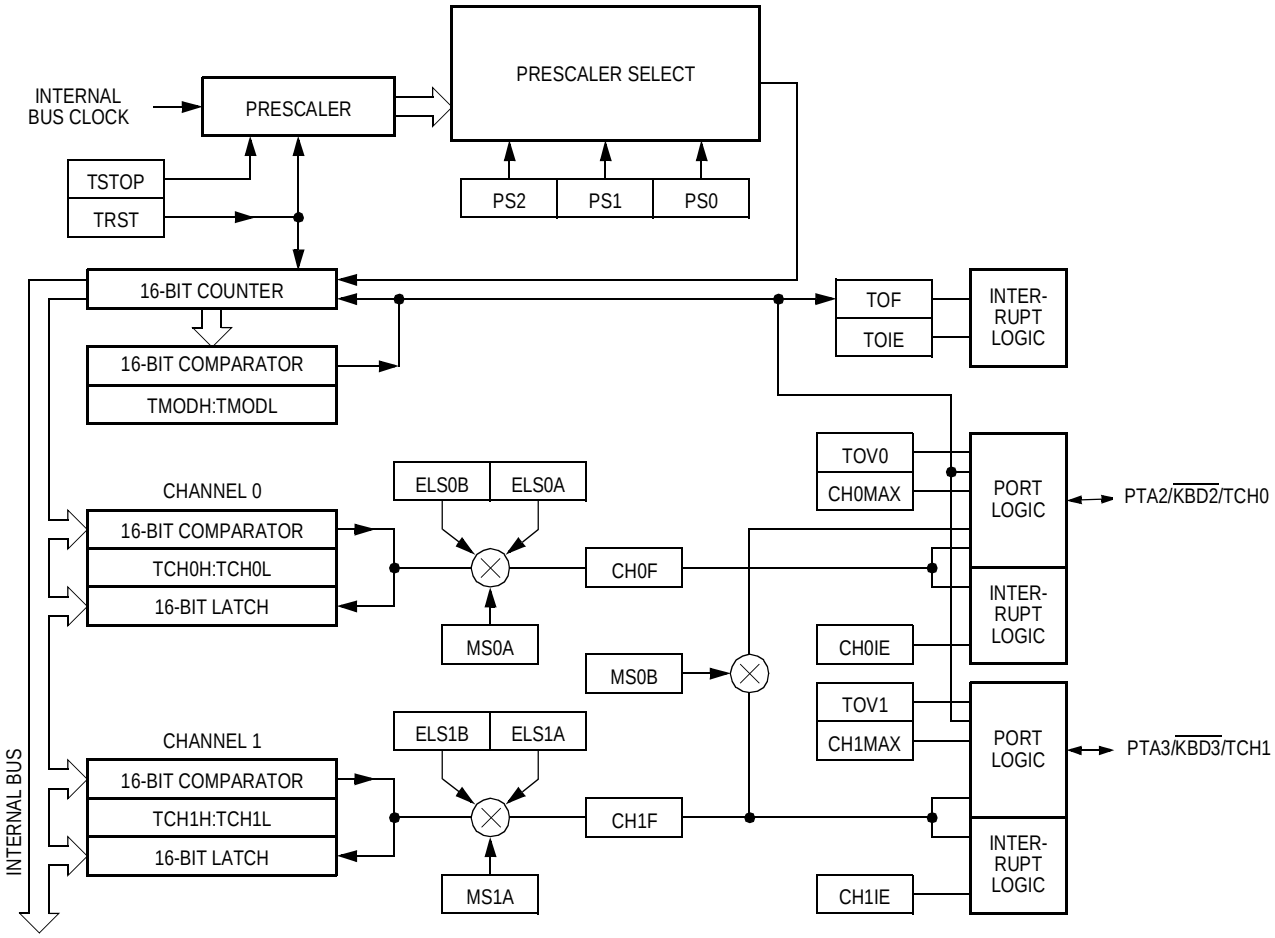


Figure 12. TIM Block Diagram

Symbolic Renaming of TIM Registers

During coding, the intention was to make the source code independent of which channel and which timer module is used for generating the Manchester encoded data stream. Also some indicators and other peripherals might be different on different hardware platforms. To keep the source code the same, symbolic, function related names of TIM registers and peripherals (e.g., LEDs) are used. Board specific definitions are made in the **board.h** file.

All timer symbolic names relating to RF functions start with `RFTimer`. Since the second channel of the same timer is used during reception, there are three groups of register symbolic names:

- Control and status registers that are common to both channels start with `RFTimerCTRL`
- Control and status registers that generate the TX data stream start with `RFTimerTXD`
- Control and status registers that are used to generate timeout during reception start with `RFTimerTMOUT`

The second 16-bit timer (TIM #2) is used as `RFTimer` in the software for RF-08 board. Channel 0 generates the TX data stream, channel 1 serves as timeout timer during reception. See **RF Reception** for details.

NOTE: *All board-specific peripherals like LEDs, buttons, the buzzer, etc. are also defined here including all necessary commands (functions) that are required to modify their state. For example, the green LED's bit is defined by the `#define LED_GREEN PTA2` definition and also the `LedGreenOn()`and `LedGreenOff()` functions define the code to set the green LED on and off using the C preprocessor directive `#define`. If the peripheral is not available on a particular platform, such a definition is left blank and thus omitted during the compile process.*

Output Compare

For Manchester encoding, the modulo-up counter mode of TIM with the output compare function has been chosen.

In the output compare mode, the TIM can generate a periodic pulse with programmable polarity, duration, and frequency. When the counter reaches the value in the register of an output compare channel, the TIM can (by hardware means) set, clear, or toggle the channel pin. Output compares can generate TIM CPU interrupt requests.

This configuration is used in the Manchester stream generation. The process is completely interrupt driven and all data edges are generated by TIM hardware, thus they appear at the exact time irrespective of any latencies caused for example by other interrupt sources.

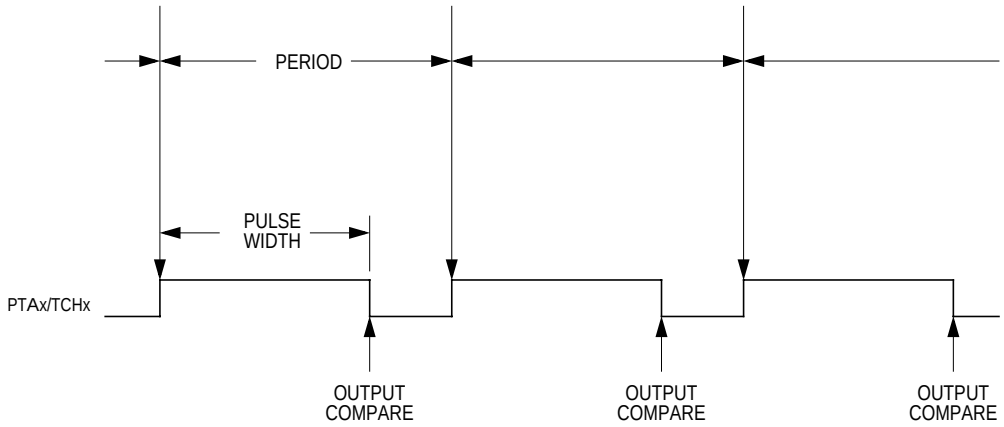


Figure 13. PWM Mode (Modulo-Up Counter)

The modulo register T2MOD (in the software for the RF-08 board, symbolic name `RFTimerMOD`) is set so the TIM module in PWM mode generates the data at the required speed. At 9600 baud, the time period is equal to 104 µs.

The constants that are written to the modulo and output compare registers are calculated at compile time by a series of macros:

In **wem.h**

```
#define BUS_CLOCK_HZ 3686400
```

In **rf2.h**

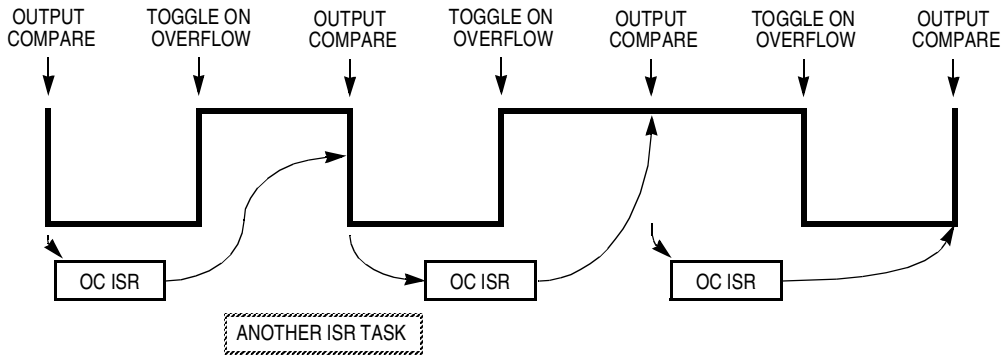
```
#define RF_SPEED 9600L

#define RF_FULLBIT (BUS_CLOCK_HZ)/RF_SPEED
#define RF_HALFBIT (BUS_CLOCK_HZ)/(2*RF_SPEED)
```

The ‘Toggle on overflow’ feature is set, so the output level of the TXDATA pin is toggled by TIM hardware at every timer overflow.

The output compare register T2CH0 (in the software for the RF-08 board, symbolic name `RFTimerCHTXD`) is set to the half of this overflow period and the ‘Set/reset on output compare’ feature prepares the value to be set/reset at the next output compare event depending on which bit should be sent next. This method effectively generates precise Manchester encoded data.

Figure 14 shows how the Manchester encoding is handled. When the timer reaches the value stored in the output compare register, the level (value) set by edge/level select bits in the status and control registers T2SC0_ELS0B and T2SC0_ELS0A is transferred to the TXDATA pin. The status and control registers T2SC0_ELS0B and T2SC0_ELS0A have the symbolic names `RFTimerTXD_ELSB` and `RFTimerTXD_ELSA` in the RF-08 application. In addition, the output compare interrupt is generated and the output compare interrupt service routine (OC ISR) is called. Within this routine a value for the next bit of transmission is stored to the edge/level select bits.



OC ISR = Output Compare Interrupt Service Routine

Figure 14. Manchester Data Encoding Through Output Compare Interrupt

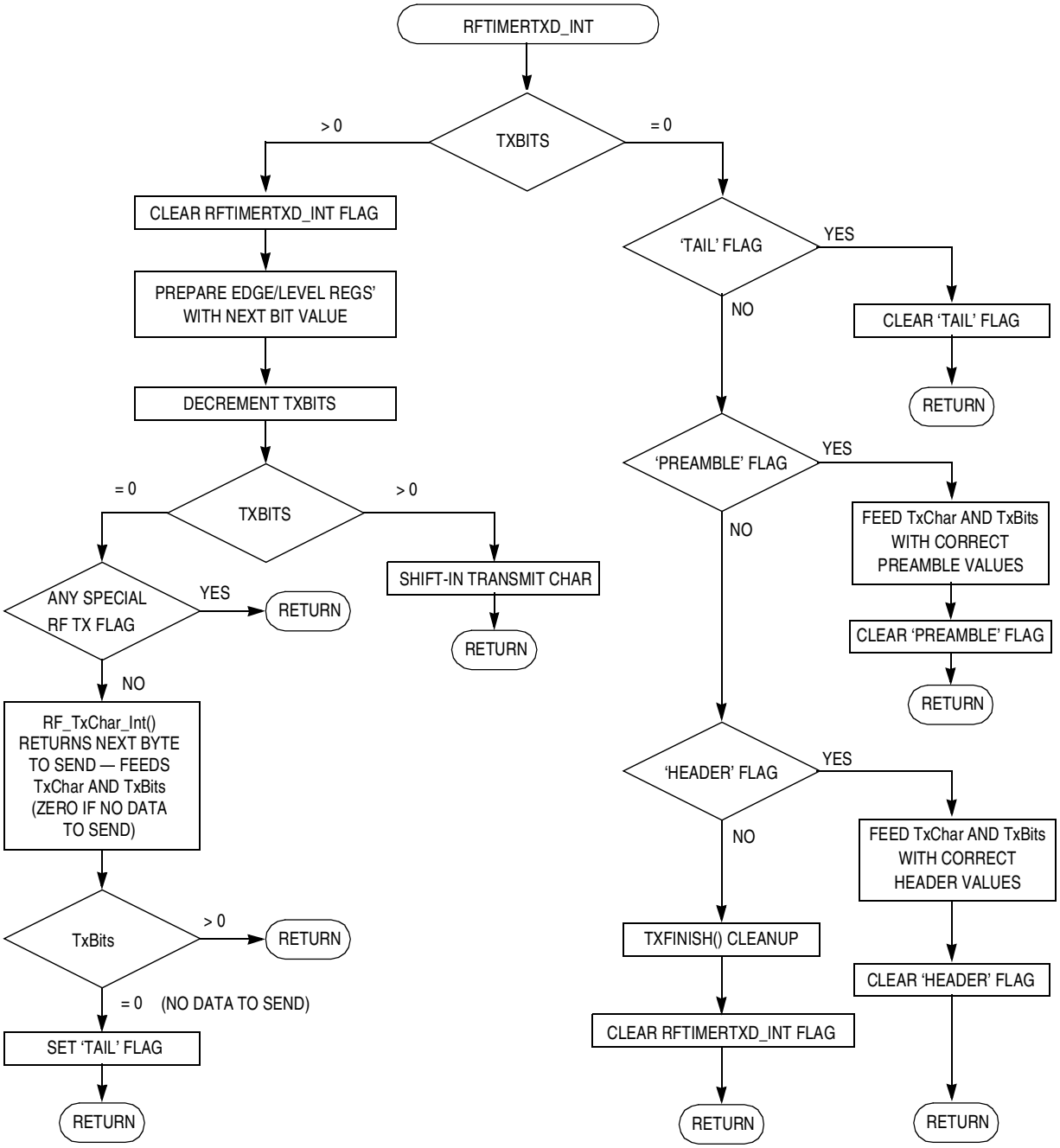
This method is insensitive to any latencies which may occur (e.g., when some other source of interrupt is being serviced). The only limitation is that the actual OC interrupt service routine must complete before the next output compare event occurs. This is demonstrated in **Figure 14**, where the second OC ISR is slightly shifted (delayed) after the output compare event, for example, until another interrupt service routine (which is not interrupted by the OC ISR) finishes its job.

NOTE: *The same code can be used to generate NRZ data stream. Clearing the 'Toggle on overflow' feature, no toggle in the middle of the bit occurs. For example, this method could be helpful in generating a 'software-SCI' data stream.*

OC Interrupt Service Routine

Since the transmit routine is a core part of the application, a more detailed explanation of its function follows. **Figure 15** shows the where program control flows. It must be understood that this routine is periodically called at a bit rate, right after an output compare event. The flow branches depending on whether data (pay-load) or special (RF specific) symbols are to be sent. Special symbols (if signalled by respective flags) always take precedence, upon RF transmit initialization 'preamble' and 'header' flags are set, so transmission starts with preamble followed by header and data.

Tail is defined as Manchester violation during two successive bits (logic 0 or logic 1 during two bits time). Within this application, the data transmission is simply stopped (creating logic 0 effectively). When all data are sent, the tail flag is set. During the next interrupt (after the last data bit is really sent out), the 'tail' flag is cleared, transmit cleanup is done, transmission is finished, and this condition is signalled to the main loop.



NOTES:

During transmit initialization the flags are set as follows:

'preamble' flag is set, 'header' flag is set, 'tail' flag is cleared, txBits is zeroed

If RFTimerTxd_Int flag is not cleared during this interrupt service, this routine is immediately called again. This is NOT a bug, it's intended to simplify the routine implementation.

Figure 15. Main Transmit Interrupt Service Routine Flowchart

Figure 16 shows the interaction between all special TX flags:

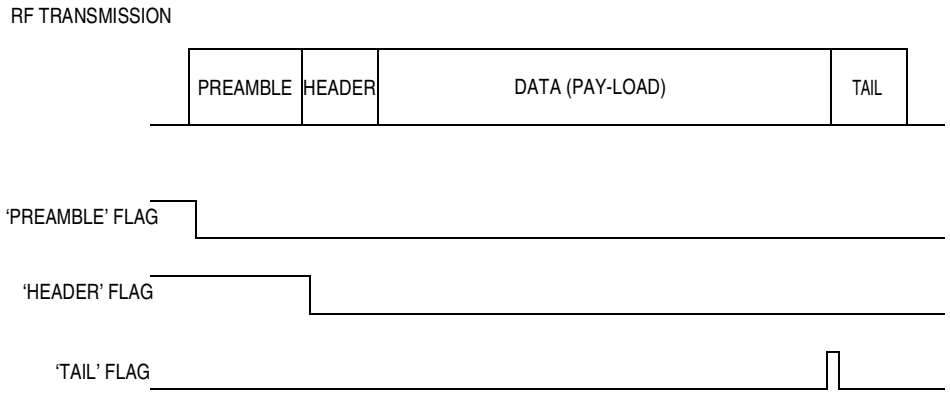


Figure 16. Special TX Flag Interactions

RF-08 Protocol

The RF-08 protocol used in this application is truly simple. Every frame sent over RF consists of three fields:

1. A length field (8 bit value), the length of a complete frame including the length and CRC fields.
2. Data (variable length), limited by memory restrictions of the CPU transmit and receive buffers, also limited by 255 bytes (maximum value of length field)
3. A CRC field (16 bits)

The RF-08 protocol is shown on **Figure 17**.

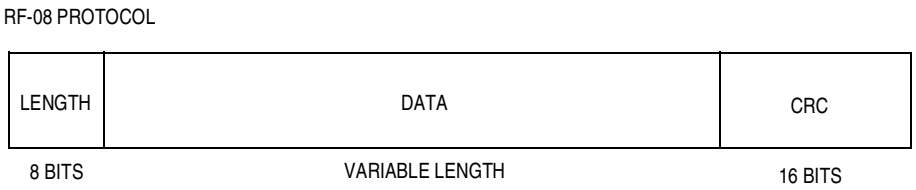


Figure 17. RF-08 Protocol

CRC Security

The cyclic redundancy codes method (also known as polynomial codes) is used to verify the integrity of every frame sent over the RF medium. An additional field is appended to every data block at the time of transmission and is checked at the time of reception for correctness. One of the well known 16-bit CRC polynomials called CRC-16, is used in the RF-08 application:

$x^{16} + x^{15} + x^2 + 1$

Further mathematical details can be found in an article entitled “A Tutorial on CRC Computations” published in the *IEEE Micro Magazine* dated August, 1988.

A table lookup algorithm has been chosen as a good compromise between speed and memory consumption. A table of 256 constants that are 16 bits long is stored in permanent storage (ROM like) memory. The actual CRC calculation routine is simple:

```
int calc_crc(char *buf, unsigned char n)
{
int crc;

  crc = 0;
  while (n--)
    crc = ((crc >> 8) & 0xff) ^ crc_table[(crc ^ *buf++) & 0xff];
  return crc;
}
```

The same routine is used both to compute the CRC field before transmission and also to check the CRC after reception of the data block. If the received data block is correct, the result of the CRC computation of the data block (including the received CRC field) is 0.

RF Reception

The data reception and decoding task is usually the most resource consuming task in similar systems. RF-08 application makes the difference mainly because of the heavy support from the MC33591/3 chip. **MC33591/3 Receiver** for the details about the hardware.

The MC33591/3 chip uniquely helps the microprocessor by providing all data preprocessing, block start detection, and data clock recovery. The MC33591/3 receiver, besides some other functions, generates “clean” data with the accompanied data clock after the block boundary mark appears. Such a data stream is easily glued to SPI (serial peripheral interface) enabled devices like the MC68HC908GP32 or its derivatives. Since **SPI Interface** describes the hardware, the description of the software services follows.

SPI Communication

The SPI is another configurable peripheral module present in the M68HC08 Family. The SPI communication is of a master-slave type.

Here in the application the roles of master and slave change depending on whether the MC33591/3 chip needs to be configured (MCU is the master, MC33591/3 is the slave) or if MC33591/3 is in normal (reception) mode, then MC33591/3 is the master leaving the MCU to be in the slave role. This is relative to the SPI operation.

The differentiation between these two modes is driven by the MCU. The input RESETB pin of MC33591/3 tells it whether the receiver is being configured (and its SPI is in slave mode) or if it should work in normal mode (its SPI is in master mode). RESETB is connected to the general-purpose I/O pin of the MCU in output mode.

The switching of SPI operation on MCU side is done by software.

NOTE: *The SPI direction is normally driven by a dedicated hardware pin (in MC68HC908GP32 it is the SPI $\overline{SS}$ pin). This pin is an input that in multiple-slave systems selects the unique slave. Since this feature is not supported, the $\overline{SS}$ pin is tied low on the board and the SPI is configured so that it ignores this pin.*

SPI Data Format

The SPI peripheral may operate in several modes and data/clock polarities. Data coming from MC33591/3 are valid at the falling clock edge. When no data are output, SCLK and MOSI force a low level. Using Motorola acronyms, this means that the clock phase and polarity control bits of the microcontroller SPI have to be CPOL = 0 and CPHA = 1.

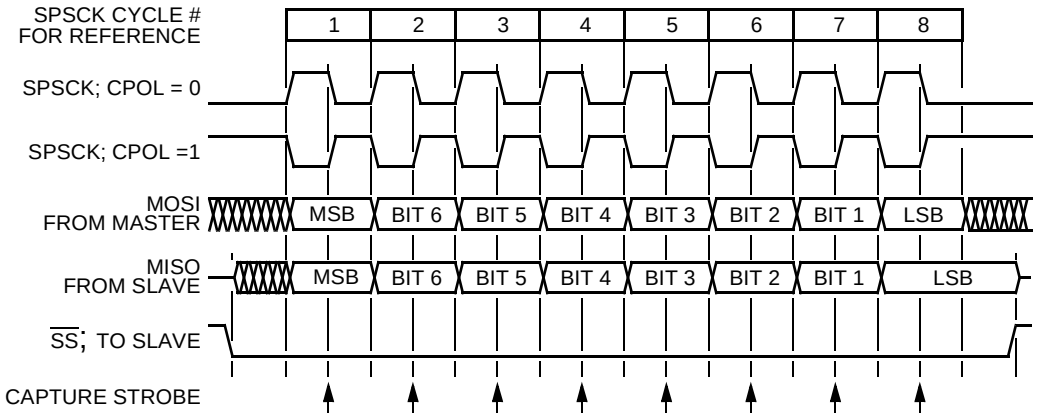


Figure 18. SPI Polarity Chart

SPI Framing

The process of receiving data is also driven by an interrupt. Here the SPI receiver interrupt is generated after a complete byte is received through the SPI. There is no information available about the start of the frame. Another method has to be used to detect when the frame starts.

When no reception is in progress, there is also no activity on the SPI. Thus, the first received byte is the start of the frame and the correct end of the frame must be detected instead. This is effectively accomplished by simple timeout method. On every SPI receive interrupt the timeout is reset. If within this

timeout no other byte is received, the timeout interrupt is activated indicating the end of the frame. This is also shown in **Figure 19**.

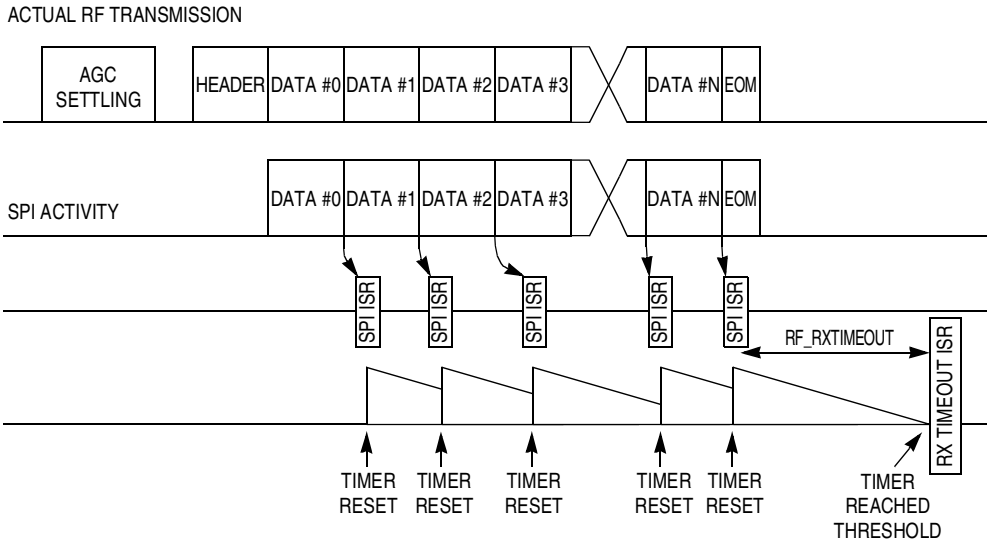


Figure 19. Detecting the End of the Frame

The timeout length is selected at twice the length of one byte. It may theoretically happen that one SPI read (and timeout reset) will occur immediately after data is received, the following SPI read could be delayed due to the other tasks. The maximum delay must be less than one byte in duration (otherwise actual unread byte is rewritten and lost by overrun). The maximum period between two such reads is less than two bytes (16 bits) length. The constant is again calculated at compile time in **rf2.h**

```
#define RF_RXTIMEOUT 16L*RF_FULLBIT
```

When the end of frame is detected, the following conditions must be met in order to declare that the received frame is correct:

- The received data is 4 bytes long or longer (the shortest frame consists of 1 byte length, 1 byte data, and 2 bytes CRC) — see **RF-08 Protocol**.
- The data length information (first position) must be 4 bytes or longer but not longer than the `RFRXBUFFSIZE` constant. A shorter value is meaningless, while a longer value means that such a frame can't fit the recipient's receive buffer.
- The CRC calculated over the complete buffer is 0.

If all conditions pass, this condition is indicated to the main loop as well as when the check conditions fail.

SCI Operation

Another part of the application is the serial communication interface (SCI). A set of universal routines have been used within this application. The main features configurable at compile time are:

- Two independent (receive and transmit) ring buffers — static (fixed length) or dynamic buffer allocation can be selected
- Selectable RS232 (full-duplex) or RS485 (half-duplex or full-duplex) mode of operation
- Hardware flow control (RTS/CTS) mode in RS232 mode
- Software flow control (XON/XOFF) mode in RS232 mode
- Idle line or timeout detection of the ‘End-Of-SCI-Frame’ condition

All functions are declared in **sci.h** and defined in **sci.c**. If timeout detection is required, the timing features are defined through one function in **tbm.h** and **tbm.c**.

SCI Timeout Detection

The RF-08 application has been mainly targeted to transfer non-continuous blocks of data which are not of fixed length. If so, some method needs to be used to specify the moment when SCI reception is considered to be finished (interrupted) and retransmission to the RF medium shall start.

There were basically two methods to choose from:

- Idle line detection
- Timeout detection

The idle line detection uses the SCI hardware to detect the idle line condition. The condition is met when 10 or 11 consecutive logical 1’s shift in from the RxD pin. Such a condition can set the appropriate SCI flag and generate an interrupt.

Since the used source of SCI data (typically a PC with Windows®(1) OS) doesn’t ensure that an idle condition won’t appear in between bytes that are sent through standard Windows I/O routines, the timeout detection method is used in this application instead.

This works quite simply: a timebase module (TBM) generates a ‘slow’ periodical interrupt every millisecond. In each SCI receive interrupt service routine the timeout variable is cleared and during every TBM interrupt this variable is incremented by one. If a value of two is reached, the timeout condition is signalled to the main loop.

1. Windows is a registered trademark of Microsoft Corporation in the U.S. and/or other countries.

NOTE: *The actual implementation is slightly different and is based only on two variables that are one bit long (flags). The first is* `sciRxTmout`*, and the second is* `sciRxTmoutHlpr`*. At the beginning of SCI reception both flags,* `sciRxTmout` *and* `sciRxTmoutHlpr`*, are cleared.* `sciRxTmoutHlpr` *is cleared during each SCI receive interrupt (action #3 in* ***Figure 20****).*

During every TBM interrupt. `sciRxTmoutHlpr` *is first copied into* `sciRxTmout` *(action #1) then after* `sciRxTmoutHlpr` *is set (action #2). If no data are received from SCI (*`sciRxTmoutHlpr` *is not cleared) the* `sciRxTmout` *flag becomes set during the second TBM interrupt indicating the end of SCI data.*

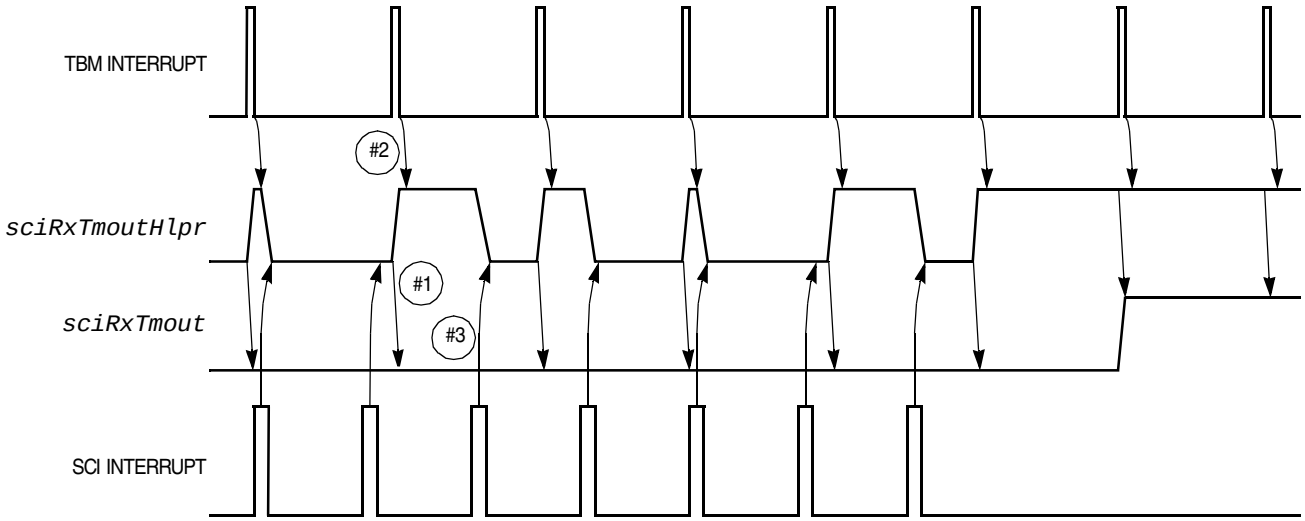


Figure 20. SCI Timeout Explanation

Main Loop

The main loop of the application simply polls for the data coming either from the SCI or RF. Data is then scheduled to be transmitted to the ‘opposite’ media. All other tasks are interrupt driven as previously described (SCI transmit, SCI receive, RF transmit, and RF receive). See **Figure 21**.

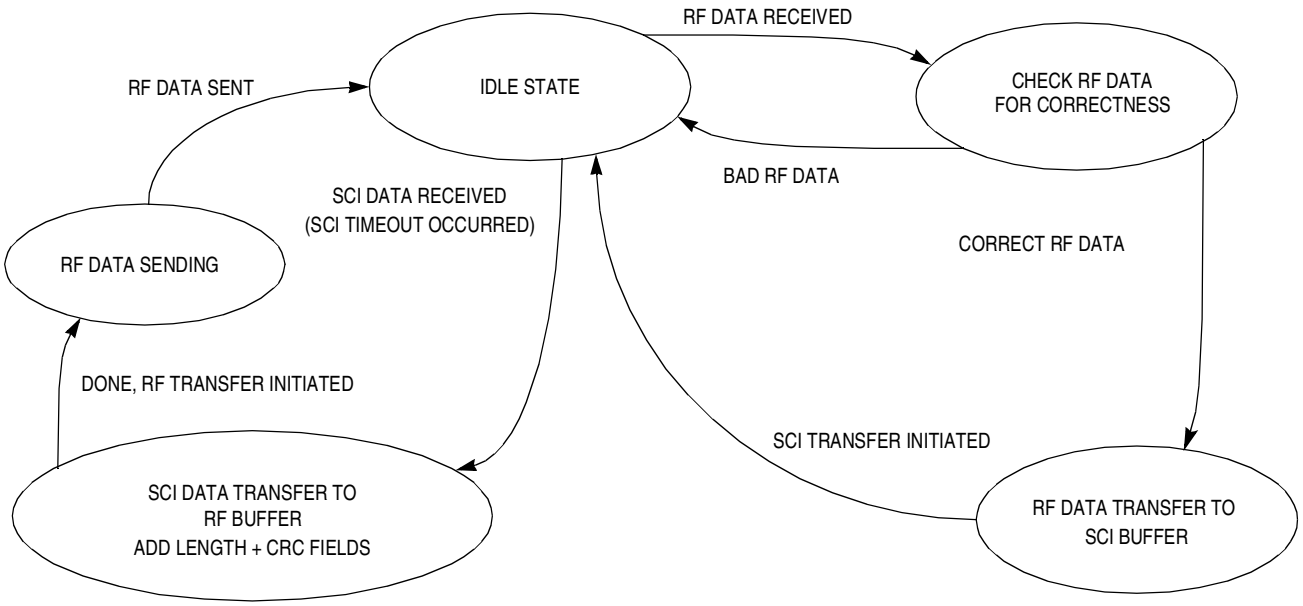


Figure 21. Main Loop State Diagram

If a user wishes to implement his own application for an end RF application, some sort of application protocol needs to be implemented and then the main loop may look like **Figure 22**.

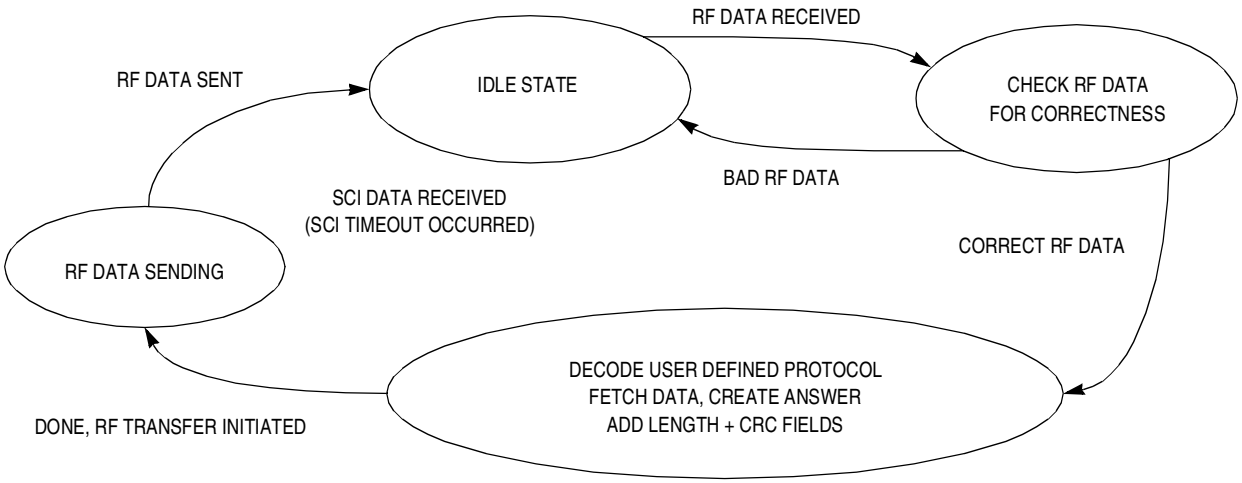


Figure 22. Example Main Loop State Diagram

Wireless HC08 Modem Testing & Demonstration

This section describes the method used to test the communication through a wireless medium. Any terminal (serial) communication software can be used. For example HyperTerminal®(1) that is a part of standard Windows installation (found under Start>Programs>Accessories>Communications>HyperTerminal menu).

ASCII Terminal Settings

There are several parameters in your terminal software that you will need to set:

- Proper serial port (COM1, COM2, COM3...)
- Communication speed (bit rate) of the demonstration is 57600 baud
- 8 data bits per character
- Parity: none
- 1 stop bit
- No flow control

Connecting the RF-08 Boards to the PC

The board supply current can be delivered by the external AC/DC converter or 9-V battery.

Perform the following steps to connect the RF-08 board cables:

1. Connect the serial extension cable to the selected serial port of the host computer or end device.
2. Connect the other end of the serial extension cable to J2 on the RF-08 board assuming that RS232 version is assembled. This provides the connection which allows the host computer/end device to communicate to the RF-08 board.
3. Connect the power supply plug into an AC power source or 9-V battery.
4. Follow the steps 1 to 3 for the second RF-08 board.

Demo Configuration

There are several ways to handle the RF connections. The most typical are:

- Where the end device sits on one side of the communication channel
- Where the client control terminal or host computer (for example, personal computer) is located

1. HyperTerminal is a registered trademark of Hilgraeve Inc.

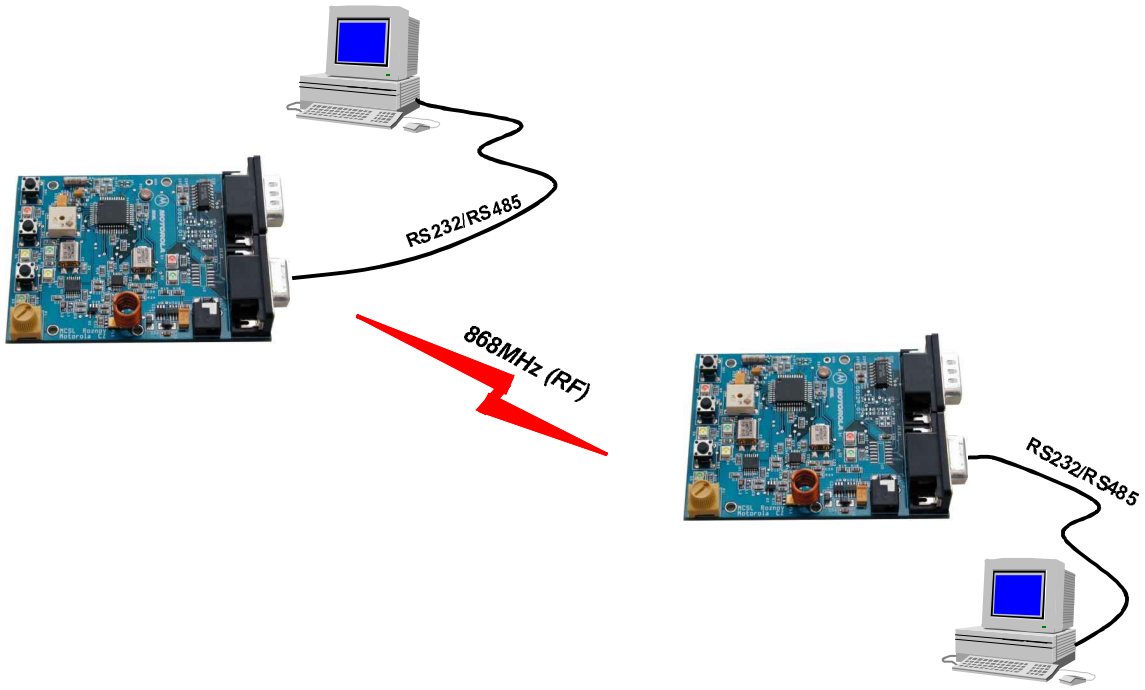


Figure 23. Demo System Overview

For demonstration purposes, the personal computers with terminal programs running are used on both sides of communication channel; first one as the end device while the second one acts as the control terminal. For this configuration, one PC with either two serial (COM) ports or two PCs has to be used.

While typing on one PC, the text also appears on the second PC screen and vice versa. This effectively shows the communication happening. If even more PCs with RF-08 boards are placed within the operating range, the text appears on all receiving parties.

References

For additional information, refer to these referenced documents:

1. MC68HC908GP32 Data Sheet (Motorola order number, MC68HC908GP32/D)
2. MC33493 Data Sheet (Motorola order number, MC33493/D)
3. MC33591/3 Data Sheet (Motorola order number, MC33591/D)
4. An article appearing in *IEEE Micro Magazine* dated August, 1988 entitled “A Tutorial on CRC Computations“
5. Maxim Integrated Products MAX202 and MAX485 Data Sheets
6. RF Micro Devices Incorporated RF2436 Data Sheet
7. AN2195 Motorola Application Note *Tango3 and Romeo2 Layout Recommendations* (Motorola order number, AN2195/D)

NOTE: *To obtain the most up-to-date information, refer to:*

Motorola, Inc. — http://www.motorola.com/semiconductors/
Maxim Integrated Products — http://www.maxim-ic.com
RF Micro Devices Incorporated — http://www.rfmd.com

MC33491 to MC33493 Migration Notes

The MC33491 (code name Tango II) is no longer available (as of Q2 2002). The MC33493 (code name Tango III) is it's full replacement and within full 3 V-only environment there's no change between the basic configurations of these two.

The MC33493 input pins are not specified above V_{CC} voltage. Thus, the MC33493 cannot be used within described RF-08 schematics that employ R1 and R3 serial resistors to protect MC33493 3-V interface. Because of reliability issues, such an interface is not recommended anyway.

For interfacing between a 5-V MCU and a 3-V MC33493, a proper level shifter must be designed or a different configuration used. An example of such a configuration is provided in **Figure 24**. The main difference between this example and the RF-08 schematic is that a 5 V to 3 V 'barrier' is inserted between the 5-V MC33591/3 (receiver) and the 3-V MCU. Again, correct voltage level shifting is highly recommended (e.g., using MAX3370/3371 from Maxim Integrated Products).

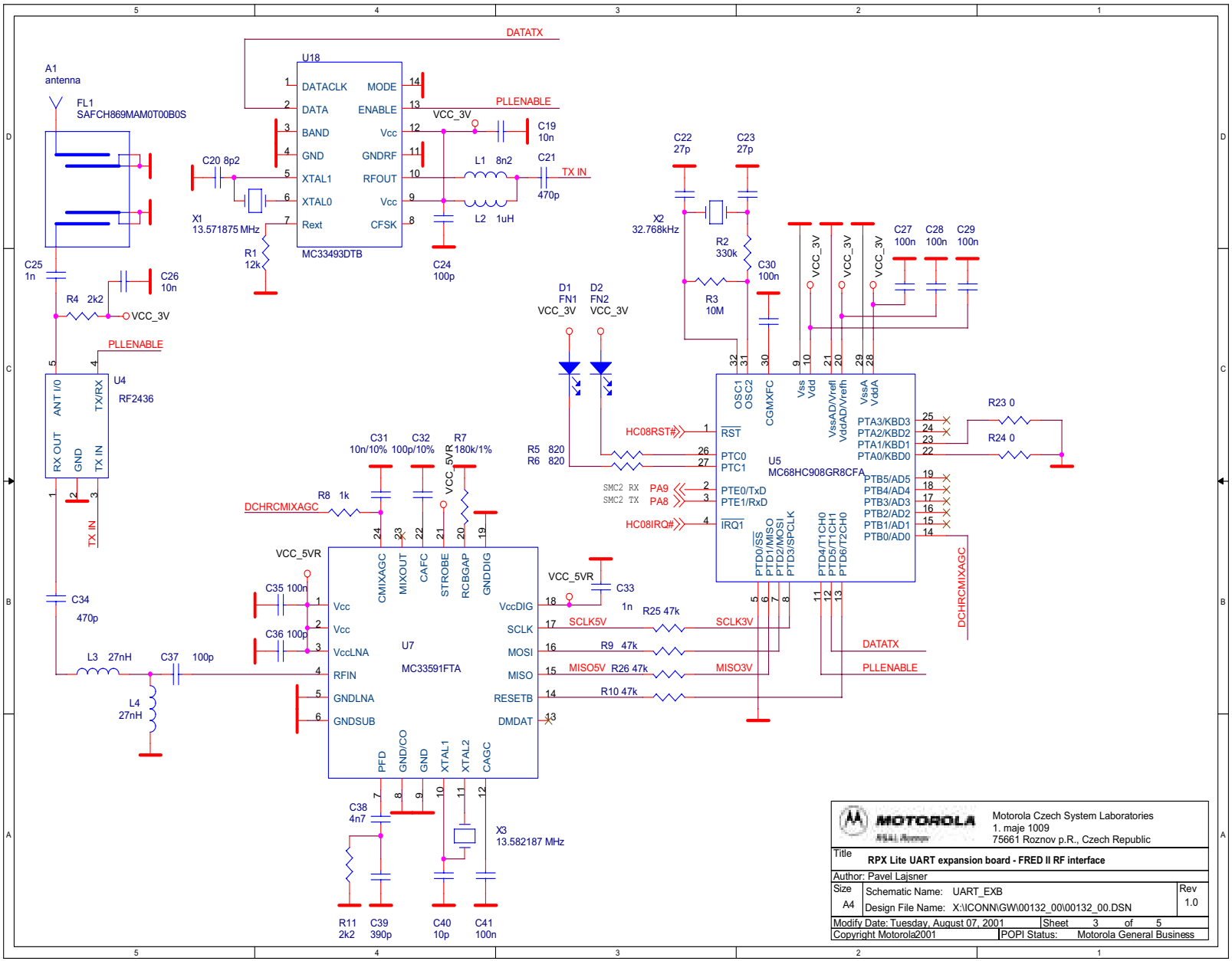


Figure 24. Application Schematic Using MC33493 (Tango III)

Source Code

board.h

```
/*****************************************************************************
* HEADER_START                                                               *
*                                                                            *
*      Name:           BOARD.H                                               *
*      Project:        Interconnectivity SRDT                                *
*      Description:    Wireless RS232 board header file                      *
*      Processor:      HC08                                                  *
*      Revision:       1.0                                                   *
*      Date:           Feb 26 2002                                           *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG              *
*      Author:         Michal Hanak                                          *
*      Company:        Motorola SPS                                          *
*      Security:       General Business                                      *
*                                                                            *
* ========================================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.              *
*                                                                            *
* ========================================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY                      *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE        *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR         *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR                     *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,           *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT               *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;               *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)                   *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,        *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)              *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED        *
* OF THE POSSIBILITY OF SUCH DAMAGE.                                         *
* ========================================================================== *
*
* HEADER_END
*/

#ifndef _BOARD_H
#define _BOARD_H

#define WEM232_I   1
#define WEM232_II  2
#define RFGW       3


#define BOARD WEM232_II
```

```
#if BOARD == RFGW

  // port pins direction
#define I_DDRA 0x00 /*  // all inputs                    */
#define I_DDRB 0x00 /*  // all inputs                    */
#define I_DDRC 0xff /*  // all outputs                   */
#define I_DDRD 0xf0 /*  // d0-d3 inputs; d4-d7 outputs   */
#define I_DDRE 0x00 /*  // e0-e1 inputs                  */


#define LED_FN1     PTC1
#define LED_FN2     PTC0

#define LED_ON      0
#define LED_OFF     1

#define LedFn1On()  LED_FN1 = LED_ON
#define LedFn1Off() LED_FN1 = LED_OFF
#define LedFn2On()  LED_FN2 = LED_ON
#define LedFn2Off() LED_FN2 = LED_OFF

#define LedGreenOn()
#define LedGreenOff()
#define LedRedOn()
#define LedRedOff()
#define LedYellowOn()
#define LedYellowOff()

#define LedYellowToggle()

// TANGO2 port related macros
#define RF_PLLEN  PTD4  // output to TANGO2
#define RF_DAT    PTD5  // output to TANGO2

// ROMEO2 port related macros
#define RF_RESETB   PTD6  // output to ROMEO2

#define RF_DCHRGCMIXAGC PTB0  // output to ROMEO2
#define RF_DCHRDDR  DDRB  // port DCHRG ROMEO2
#define RF_DCHRGMASK  0x01  // DDR MASK DCHRG ROMEO2

#define RFDchrgInputs()  RF_DCHRDDR &= ~RF_DCHRGMASK
#define RFDchrgOutputs()  RF_DCHRDDR |= RF_DCHRGMASK

#define StrobeEnable()
#define RFModeDisable()
#define RFDchrgCAgcOff()
#define RFDchrgCMixAgcOff()  RF_DCHRGCMIXAGC = 0
```

```
#define IfBtnJp1BuzzerOn()
#define IfBtnJp1BuzzerOff()

#define RFTimerCTRL  T1SC
#define RFTimerCTRL_TOF T1SC_TOF
#define RFTimerCTRL_TOIE T1SC_TOIE

#define RFTimerTMOUT T1SC0
#define RFTimerTMOUT_CHF T1SC0_CH0F
#define RFTimerTMOUT_CHIE T1SC0_CH0IE

#define RFTimerTMOUT_TOV T1SC0_TOV0
#define RFTimerTMOUT_MSA T1SC0_MS0A

#define RFTimerTMOUT_ELSB T1SC0_ELS0B
#define RFTimerTMOUT_ELSA T1SC0_ELS0A

#define RFTimerTXD T1SC1
#define RFTimerTXD_CHF T1SC1_CH1F
#define RFTimerTXD_CHIE T1SC1_CH1IE

#define RFTimerTXD_TOV T1SC1_TOV1
#define RFTimerTXD_MSA T1SC1_MS1A

#define RFTimerTXD_ELSB T1SC1_ELS1B
#define RFTimerTXD_ELSA T1SC1_ELS1A

#define RFTimerMOD T1MOD
#define RFTimerCHTMOUT T1CH0
#define RFTimerCHTXD T1CH1

#define RFTimerCTRL_TSTOP T1SC_TSTOP
#define RFTimerCTRL_TRST T1SC_TRST


#elif BOARD == WEM232_II

  // port pins direction
#define I_DDRA 0xe0    /*  // a0-a4 inputs; a5-a7 outputs */
#define I_DDRB 0x38    /*  // b3-b5 outputs (b0-b2 ADC)   */
#define I_DDRC 0xff    /*  // c0-c7 outputs               */
#define I_DDRD 0xf0    /*  // d0-d3 inputs; d4-d7 outputs */
#define I_DDRE 0x00    /*  // e0-e1 inputs                */


#define LED_FN1     PTC4
#define LED_FN2     PTC3

#define LED_GREEN   PTC2
#define LED_YELLOW  PTC1
#define LED_RED     PTC0
```

```
#define LED_ON      0
#define LED_OFF     1

#define BUZZER      PTD4
#define BUZZER_ON   0
#define BUZZER_OFF  1

#define BTN_JP2     PTA4
#define BTN_JP1     PTA3
#define BTN_GREEN   PTA2
#define BTN_YELLOW  PTA1
#define BTN_RED     PTA0

#define BTN_DOWN    0
#define BTN_UP      1


#define LedFn1On()     LED_FN1 = LED_ON
#define LedFn1Off()    LED_FN1 = LED_OFF
#define LedFn2On()     LED_FN2 = LED_ON
#define LedFn2Off()    LED_FN2 = LED_OFF

#define LedGreenOn()   LED_GREEN = LED_ON
#define LedGreenOff()  LED_GREEN = LED_OFF
#define LedRedOn()     LED_RED = LED_ON
#define LedRedOff()    LED_RED = LED_OFF
#define LedYellowOn()  LED_YELLOW = LED_ON
#define LedYellowOff() LED_YELLOW = LED_OFF

#define LedYellowToggle() LED_YELLOW = ~LED_YELLOW             /**/

// TANGO2 port related macros
#define RF_PLLEN  PTD5  // output to TANGO2
#define RF_DAT    PTD6  // output to TANGO2

// ROMEO2 port related macros
#define RF_RESETB   PTA5  // output to ROMEO2
#define RF_DCHRGCMIXAGC PTB5  // output to ROMEO2

#define RF_DCHRDDR  DDRB  // port DCHRG ROMEO2
#define RF_DCHRGMASK  0x20  // DDR MASK DCHRG ROMEO2

#define RFDchrgInputs()  RF_DCHRDDR &= ~RF_DCHRGMASK
#define RFDchrgOutputs()  RF_DCHRDDR |= RF_DCHRGMASK

#define StrobeEnable()
#define RFModeDisable()
#define RFDchrgCAgcOff()
#define RFDchrgCMixAgcOff()  RF_DCHRGCMIXAGC = 0
```

```
#define IfBtnJp1BuzzerOn()  if (BTN_JP1) BUZZER = BUZZER_ON
#define IfBtnJp1BuzzerOff()  if (BTN_JP1) BUZZER = BUZZER_OFF


#define RFTimerCTRL  T2SC
#define RFTimerCTRL_TOF T2SC_TOF
#define RFTimerCTRL_TOIE T2SC_TOIE

#define RFTimerTXD T2SC0
#define RFTimerTXD_CHF T2SC0_CH0F
#define RFTimerTXD_CHIE T2SC0_CH0IE

#define RFTimerTXD_TOV T2SC0_TOV0
#define RFTimerTXD_MSA T2SC0_MS0A

#define RFTimerTXD_ELSB T2SC0_ELS0B
#define RFTimerTXD_ELSA T2SC0_ELS0A

#define RFTimerTMOUT T2SC1
#define RFTimerTMOUT_CHF T2SC1_CH1F
#define RFTimerTMOUT_CHIE T2SC1_CH1IE

#define RFTimerTMOUT_TOV T2SC1_TOV1
#define RFTimerTMOUT_MSA T2SC1_MS1A

#define RFTimerTMOUT_ELSB T2SC1_ELS1B
#define RFTimerTMOUT_ELSA T2SC1_ELS1A

#define RFTimerMOD T2MOD
#define RFTimerCHTXD T2CH0
#define RFTimerCHTMOUT T2CH1

#define RFTimerCTRL_TSTOP T2SC_TSTOP
#define RFTimerCTRL_TRST T2SC_TRST

#elif  BOARD == WEM232_I

  // port pins direction
#define I_DDRA 0xe0   /* a0-a4 inputs; a5-a7 outputs */
#define I_DDRB 0x38   /* b3-b5 outputs (b0-b2 ADC)   */
#define I_DDRC 0xff   /* c0-c7 outputs               */
#define I_DDRD 0xf0   /* d0-d3 inputs; d4-d7 outputs */
#define I_DDRE 0x00   /* e0-e1 inputs                */

#define LED_FN1     PTC4
#define LED_FN2     PTC3

#define LED_GREEN   PTC2
#define LED_YELLOW  PTC1
#define LED_RED     PTC0
```

```
#define LED_ON      0
#define LED_OFF     1

#define BUZZER      PTD4
#define BUZZER_ON   0
#define BUZZER_OFF  1

#define BTN_JP2     PTA4
#define BTN_JP1     PTA3
#define BTN_GREEN   PTA2
#define BTN_YELLOW  PTA1
#define BTN_RED     PTA0

#define BTN_DOWN    0
#define BTN_UP      1

#define LedFn1On()     LED_FN1 = LED_ON
#define LedFn1Off()    LED_FN1 = LED_OFF
#define LedFn2On()     LED_FN2 = LED_ON
#define LedFn2Off()    LED_FN2 = LED_OFF

#define LedGreenOn()   LED_GREEN = LED_ON
#define LedGreenOff()  LED_GREEN = LED_OFF
#define LedRedOn()     LED_RED = LED_ON
#define LedRedOff()    LED_RED = LED_OFF
#define LedYellowOn()  LED_YELLOW = LED_ON
#define LedYellowOff() LED_YELLOW = LED_OFF

#define LedYellowToggle() LED_YELLOW = ~LED_YELLOW             /**/

// TANGO2 port related macros
#define RF_PLLEN  PTD5  // output to TANGO2
#define RF_MODE   PTD7  // output to TANGO2
#define RF_DAT    PTD6  // output to TANGO2

// ROMEO2 port related macros
#define RF_RESETB   PTA5  // output to ROMEO2
#define RF_STROBE   PTB3  // output to ROMEO2
#define RF_DCHRGCMIXAGC PTB5  // output to ROMEO2
#define RF_DCHRGCAGC PTB4     // output to ROMEO2

#define RF_DCHRDDR  DDRB  // port DCHRG ROMEO2
#define RF_DCHRGMASK  0x30  // DDR MASK DCHRG ROMEO2

#define RFDchrgInputs()  RF_DCHRDDR &= ~RF_DCHRGMASK
#define RFDchrgOutputs()  RF_DCHRDDR |= RF_DCHRGMASK

#define StrobeEnable()  RF_STROBE = 1
#define RFModeDisable() RF_MODE = 0
#define RFDchrgCAgcOff()  RF_DCHRGCAGC = 0
#define RFDchrgCMixAgcOff()  RF_DCHRGCMIXAGC = 0
```

```
#define IfBtnJp1BuzzerOn()  if (BTN_JP1) BUZZER = BUZZER_ON
#define IfBtnJp1BuzzerOff()  if (BTN_JP1) BUZZER = BUZZER_OFF

#define RFTimerCTRL  T2SC
#define RFTimerCTRL_TOF T2SC_TOF
#define RFTimerCTRL_TOIE T2SC_TOIE

#define RFTimerTXD T2SC0
#define RFTimerTXD_CHF T2SC0_CH0F
#define RFTimerTXD_CHIE T2SC0_CH0IE

#define RFTimerTXD_TOV T2SC0_TOV0
#define RFTimerTXD_MSA T2SC0_MS0A

#define RFTimerTXD_ELSB T2SC0_ELS0B
#define RFTimerTXD_ELSA T2SC0_ELS0A

#define RFTimerTMOUT T2SC1
#define RFTimerTMOUT_CHF T2SC1_CH1F
#define RFTimerTMOUT_CHIE T2SC1_CH1IE

#define RFTimerTMOUT_TOV T2SC1_TOV1
#define RFTimerTMOUT_MSA T2SC1_MS1A

#define RFTimerTMOUT_ELSB T2SC1_ELS1B
#define RFTimerTMOUT_ELSA T2SC1_ELS1A

#define RFTimerMOD T2MOD
#define RFTimerCHTXD T2CH0
#define RFTimerCHTMOUT T2CH1

#define RFTimerCTRL_TSTOP T2SC_TSTOP
#define RFTimerCTRL_TRST T2SC_TRST

#endif

#endif
```

hc08gp32.h

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           hc08gp32.h                                *
*      Project:        Interconnectivity SRDT                    *
*      Description:    HC08GP32 header file                      *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Pavel Lajsner                             *
*      Based on:       original KX6/8 header file by William Mackay  *
*                      & Ken Berringer                           *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT       *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;       *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)           *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)      *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

/*************************************************************************/
/*************** HC08 device mapping for 68HC908GP32             ************/
/*************************************************************************/
/*

Register Definitions

    This header file defines all of the registers using the register name exactly
    as listed in the data book. Each register name is entered in ALL CAPS. The
    registers are all unsigned volatile bytes to ensure that the compiler will
    not remove sequential write commands. This still does not guarentee that
    the compiler will execute each assignment exactly as coded, as some compilers
```

```
    may still try to optimize volatile bytes. The compiler should be tested
    with this header file to ensure register assignments are not optimized.

Bitfields

    Each bit is defined using the following format:

        REGISTERNAME_BITNAME

    While this may result in redundant names in some cases, it will
    always prevent duplicate bit names. If a bit has a descriptive name
    such as FLG the descriptive name is used.

    If the bit does not have a descriptive name the generic terms BIT0
    through BIT7 are used.

    In some cases the data book may define bits by adding a digit to the
    register name. In these cases the generic BIT0 names are used
    instead. For eample, DDRA_BIT0 is used instead of the redundant
    DDRA_DDRA0. Short Aliases are defined to provide compatibility for
    these particular cases.


Exceptions

Short Aliases

    Short Aliases are defined for particular cases where the customary usage
    is to use short bit names. For example, it is customary to use the short
    term PTA0 instead of PTA_BIT0. This header permits both usages.

 */

#ifndef __hc08gp32_h
#define __hc08gp32_h      /* if this H file is not included, include */


#define BIT0 0x01
#define BIT1 0x02
#define BIT2 0x04
#define BIT3 0x08
#define BIT4 0x10
#define BIT5 0x20
#define BIT6 0x40
#define BIT7 0x80
```

```
/*****************************************************************************/
/* Register Mapping Structures and Macros                                    */
/*****************************************************************************/

#define DBLREG(a)  (*((volatile unsigned int *)(a)))
#define REGISTER(a)  (*((volatile unsigned char *)(a)))
#define BIT(a,b)  (((vbitfield *)(a))->bit##b)

/* assumes right to left bit order, highware default */

typedef volatile struct{
  volatile unsigned int bit0   : 1;
  volatile unsigned int bit1   : 1;
  volatile unsigned int bit2   : 1;
  volatile unsigned int bit3   : 1;
  volatile unsigned int bit4   : 1;
  volatile unsigned int bit5   : 1;
  volatile unsigned int bit6   : 1;
  volatile unsigned int bit7   : 1;
} vbitfield;

/*****************************************************************************/
/* Input Output Ports                                                        */
/*****************************************************************************/

/* Port A Data register */
#define  PTA          REGISTER(0x00)
#define  PTA_BIT0     BIT(0x00,0)
#define  PTA_BIT1     BIT(0x00,1)
#define  PTA_BIT2     BIT(0x00,2)
#define  PTA_BIT3     BIT(0x00,3)
#define  PTA_BIT4     BIT(0x00,4)
#define  PTA_BIT5     BIT(0x00,5)
#define  PTA_BIT6     BIT(0x00,6)
#define  PTA_BIT7     BIT(0x00,7)


/* Port B Data register */
#define  PTB          REGISTER(0x01)
#define  PTB_BIT0     BIT(0x01,0)
#define  PTB_BIT1     BIT(0x01,1)
#define  PTB_BIT2     BIT(0x01,2)
#define  PTB_BIT3     BIT(0x01,3)
#define  PTB_BIT4     BIT(0x01,4)
#define  PTB_BIT5     BIT(0x01,5)
#define  PTB_BIT6     BIT(0x01,6)
#define  PTB_BIT7     BIT(0x01,7)
```

```
/* Port C Data register */
#define  PTC          REGISTER(0x02)
#define  PTC_BIT0     BIT(0x02,0)
#define  PTC_BIT1     BIT(0x02,1)
#define  PTC_BIT2     BIT(0x02,2)
#define  PTC_BIT3     BIT(0x02,3)
#define  PTC_BIT4     BIT(0x02,4)
#define  PTC_BIT5     BIT(0x02,5)
#define  PTC_BIT6     BIT(0x02,6)


/* Port D Data register */
#define  PTD          REGISTER(0x03)
#define  PTD_BIT0     BIT(0x03,0)
#define  PTD_BIT1     BIT(0x03,1)
#define  PTD_BIT2     BIT(0x03,2)
#define  PTD_BIT3     BIT(0x03,3)
#define  PTD_BIT4     BIT(0x03,4)
#define  PTD_BIT5     BIT(0x03,5)
#define  PTD_BIT6     BIT(0x03,6)
#define  PTD_BIT7     BIT(0x03,7)


/* Port E Data register */
#define  PTE          REGISTER(0x08)
#define  PTE_BIT0     BIT(0x08,0)
#define  PTE_BIT1     BIT(0x08,1)


/* Port A Data Direction Register */
#define  DDRA           REGISTER(0x04)
#define  DDRA_BIT0      BIT(0x04,0)
#define  DDRA_BIT1      BIT(0x04,1)
#define  DDRA_BIT2      BIT(0x04,2)
#define  DDRA_BIT3      BIT(0x04,3)
#define  DDRA_BIT4      BIT(0x04,4)
#define  DDRA_BIT5      BIT(0x04,5)
#define  DDRA_BIT6      BIT(0x04,6)
#define  DDRA_BIT7      BIT(0x04,7)


/* Port B Data Direction Register */
#define  DDRB          REGISTER(0x05)
#define  DDRB_BIT0     BIT(0x05,0)
#define  DDRB_BIT1     BIT(0x05,1)
#define  DDRB_BIT2     BIT(0x05,2)
#define  DDRB_BIT3     BIT(0x05,3)
#define  DDRB_BIT4     BIT(0x05,4)
#define  DDRB_BIT5     BIT(0x05,5)
#define  DDRB_BIT6     BIT(0x05,6)
#define  DDRB_BIT7     BIT(0x05,7)
```

```
/* Port C Data Direction Register */
#define  DDRC          REGISTER(0x06)
#define  DDRC_BIT0     BIT(0x06,0)
#define  DDRC_BIT1     BIT(0x06,1)
#define  DDRC_BIT2     BIT(0x06,2)
#define  DDRC_BIT3     BIT(0x06,3)
#define  DDRC_BIT4     BIT(0x06,4)
#define  DDRC_BIT5     BIT(0x06,5)
#define  DDRC_BIT6     BIT(0x06,6)
#define  DDRC_BIT7     BIT(0x06,7)


/* Port D Data Direction Register */
#define  DDRD          REGISTER(0x07)
#define  DDRD_BIT0     BIT(0x07,0)
#define  DDRD_BIT1     BIT(0x07,1)
#define  DDRD_BIT2     BIT(0x07,2)
#define  DDRD_BIT3     BIT(0x07,3)
#define  DDRD_BIT4     BIT(0x07,4)
#define  DDRD_BIT5     BIT(0x07,5)
#define  DDRD_BIT6     BIT(0x07,6)
#define  DDRD_BIT7     BIT(0x07,7)


/* Port E Data Direction Register */
#define  DDRE          REGISTER(0x0c)
#define  DDRE_BIT0     BIT(0x0c,0)
#define  DDRE_BIT1     BIT(0x0c,1)


/* Port A Pull Up Enable Register */
#define  PTAPUE          REGISTER(0x0d)
#define  PTAPUE_BIT0     BIT(0x0d,0)
#define  PTAPUE_BIT1     BIT(0x0d,1)
#define  PTAPUE_BIT2     BIT(0x0d,2)
#define  PTAPUE_BIT3     BIT(0x0d,3)
#define  PTAPUE_BIT4     BIT(0x0d,4)
#define  PTAPUE_BIT5     BIT(0x0d,5)
#define  PTAPUE_BIT6     BIT(0x0d,6)
#define  PTAPUE_BIT7     BIT(0x0d,7)


/* Port C Pull Up Enable Register */
#define  PTCPUE          REGISTER(0x0e)
#define  PTCPUE_BIT0     BIT(0x0e,0)
#define  PTCPUE_BIT1     BIT(0x0e,1)
#define  PTCPUE_BIT2     BIT(0x0e,2)
#define  PTCPUE_BIT3     BIT(0x0e,3)
#define  PTCPUE_BIT4     BIT(0x0e,4)
#define  PTCPUE_BIT5     BIT(0x0e,5)
#define  PTCPUE_BIT6     BIT(0x0e,6)
```

```
/* Port D Pull Up Enable Register */
#define  PTDPUE          REGISTER(0x0f)
#define  PTDPUE_BIT0     BIT(0x0f,0)
#define  PTDPUE_BIT1     BIT(0x0f,1)
#define  PTDPUE_BIT2     BIT(0x0f,2)
#define  PTDPUE_BIT3     BIT(0x0f,3)
#define  PTDPUE_BIT4     BIT(0x0f,4)
#define  PTDPUE_BIT5     BIT(0x0f,5)
#define  PTDPUE_BIT6     BIT(0x0f,6)
#define  PTDPUE_BIT7     BIT(0x0f,7)
/*************************************************************************/
/* Serial Peripheral Interface Registers                                 */
/*************************************************************************/

/* SPI Control Register */
#define  SPCR            REGISTER(0x10)
#define  SPCR_SPTIE      BIT(0x10,0)
#define  SPCR_SPE        BIT(0x10,1)
#define  SPCR_SPWOM      BIT(0x10,2)
#define  SPCR_CPHA       BIT(0x10,3)
#define  SPCR_CPOL       BIT(0x10,4)
#define  SPCR_SPMSTR     BIT(0x10,5)
#define  SPCR_DMAS       BIT(0x10,6)
#define  SPCR_SPRIE      BIT(0x10,7)

/* SPI Status and Control Register */
#define  SPSCR           REGISTER(0x11)
#define  SPSCR_SPR0      BIT(0x11,0)
#define  SPSCR_SPR1      BIT(0x11,1)
#define  SPSCR_MODFEN    BIT(0x11,2)
#define  SPSCR_SPTE      BIT(0x11,3)
#define  SPSCR_MODF      BIT(0x11,4)
#define  SPSCR_OVRF      BIT(0x11,5)
#define  SPSCR_ERRIE     BIT(0x11,6)
#define  SPSCR_SPRF      BIT(0x11,7)

/* SPI Data Register */
#define  SPDR            REGISTER(0x12)

/*************************************************************************/
/* Serial Communications Interface Registers                             */
/*************************************************************************/

/* SCI Control Register 1 */
#define  SCC1            REGISTER(0x13)
#define  SCC1_PTY        BIT(0x13,0)
#define  SCC1_PEN        BIT(0x13,1)
#define  SCC1_ILTY       BIT(0x13,2)
#define  SCC1_WAKE       BIT(0x13,3)
#define  SCC1_M          BIT(0x13,4)
#define  SCC1_TXINV      BIT(0x13,5)
```

```
#define  SCC1_ENSCI     BIT(0x13,6)
#define  SCC1_LOOPS     BIT(0x13,7)

/* SCI Control Register 2 */
#define  SCC2           REGISTER(0x14)
#define  SCC2_SBK       BIT(0x14,0)
#define  SCC2_RWU       BIT(0x14,1)
#define  SCC2_RE        BIT(0x14,2)
#define  SCC2_TE        BIT(0x14,3)
#define  SCC2_ILIE      BIT(0x14,4)
#define  SCC2_SCRIE     BIT(0x14,5)
#define  SCC2_TCIE      BIT(0x14,6)
#define  SCC2_SCTIE     BIT(0x14,7)

/* SCI Control Register 3 */
#define  SCC3           REGISTER(0x15)
#define  SCC3_PEIE      BIT(0x15,0)
#define  SCC3_FEIE      BIT(0x15,1)
#define  SCC3_NEIE      BIT(0x15,2)
#define  SCC3_ORIE      BIT(0x15,3)
#define  SCC3_DMATE     BIT(0x15,4)
#define  SCC3_DMARE     BIT(0x15,5)
#define  SCC3_T8        BIT(0x15,6)
#define  SCC3_R8        BIT(0x15,7)

/* SCI Status Register 1 */
#define  SCS1           REGISTER(0x16)
#define  SCS1_PE        BIT(0x16,0)
#define  SCS1_FE        BIT(0x16,1)
#define  SCS1_NF        BIT(0x16,2)
#define  SCS1_OR        BIT(0x16,3)
#define  SCS1_IDLE      BIT(0x16,4)
#define  SCS1_SCRF      BIT(0x16,5)
#define  SCS1_TC        BIT(0x16,6)
#define  SCS1_SCTE      BIT(0x16,7)

/* SCI Status Register 2 */
#define  SCS2           REGISTER(0x17)
#define  SCS2_RPF       BIT(0x17,0)
#define  SCS2_BKF       BIT(0x17,1)

/* SCI Data Register */
/* bit manipulation not recomended */
#define  SCDR           REGISTER(0x18)

/* SCI Baud Rate Register */
#define  SCBR           REGISTER(0x19)
#define  SCBR_SCR0      BIT(0x19,0)
#define  SCBR_SCR1      BIT(0x19,1)
#define  SCBR_SCR2      BIT(0x19,2)
#define  SCBR_SCP0      BIT(0x19,4)
#define  SCBR_SCP1      BIT(0x19,5)
```

```
/*****************************************************************************/
/* Keyboard Registers                                                        */
/*****************************************************************************/

/* Keyboard Status and Control Register */
#define  INTKBSCR          REGISTER(0x1a)
#define  INTKBSCR_MODEK    BIT(0x1a,0)
#define  INTKBSCR_IMASKK   BIT(0x1a,1)
#define  INTKBSCR_ACKK     BIT(0x1a,2)
#define  INTKBSCR_KEYF     BIT(0x1a,3)


/* Keyboard Interrupt Enable Register */
#define  INTKBIER           REGISTER(0x1b)
#define  INTKBIER_KBIE0     BIT(0x1b,0)
#define  INTKBIER_KBIE1     BIT(0x1b,1)
#define  INTKBIER_KBIE2     BIT(0x1b,2)
#define  INTKBIRE_KBIE3     BIT(0x1b,3)
#define  INTKBIER_KBIE4     BIT(0x1b,4)
#define  INTKBIER_KBIE5     BIT(0x1b,5)
#define  INTKBIER_KBIE6     BIT(0x1b,6)
#define  INTKBIER_KBIE7     BIT(0x1b,7)


/*****************************************************************************/
/* Time Base Control Register                                                */
/*****************************************************************************/

#define  TBCR           REGISTER(0x1c)
#define  TBCR_TBON      BIT(0x1c,1)
#define  TBCR_TBIE      BIT(0x1c,2)
#define  TBCR_TACK      BIT(0x1c,3)
#define  TBCR_TBR0      BIT(0x1c,4)
#define  TBCR_TBR1      BIT(0x1c,5)
#define  TBCR_TBR2      BIT(0x1c,6)
#define  TBCR_TBIF      BIT(0x1c,7)


/*****************************************************************************/
/* IRQ Status and Control Register                                           */
/*****************************************************************************/

#define  ISCR           REGISTER(0x1d)
#define  ISCR_MODE1     BIT(0x1d,0)
#define  ISCR_IMASK1    BIT(0x1d,1)
#define  ISCR_ACK1      BIT(0x1d,2)
#define  ISCR_IRQF1     BIT(0x1d,3)
```

```
/*****************************************************************************/
/* Configuration Write-Once Registers                                        */
/*                                                                           */
/* note: bit fields or bit manipulation is not permitted on write once reg   */
/*                                                                           */
/*****************************************************************************/

#define  CONFIG2    REGISTER(0x1e)
#define  CONFIG1    REGISTER(0x1F)

/*****************************************************************************/
/* Timer Registers #1                                                        */
/*****************************************************************************/

/* Timer Status and Control Register */
#define  T1SC            REGISTER(0x20)
#define  T1SC_PS0        BIT(0x20,0)
#define  T1SC_PS1        BIT(0x20,1)
#define  T1SC_PS2        BIT(0x20,2)
#define  T1SC_TRST       BIT(0x20,4)
#define  T1SC_TSTOP      BIT(0x20,5)
#define  T1SC_TOIE       BIT(0x20,6)
#define  T1SC_TOF        BIT(0x20,7)

/* Timer Counter Register */
#define  T1CNT  DBLREG(0x21)

#define  T1CNTH  REGISTER(0x21)
#define  T1CNTL  REGISTER(0x22)

/* Timer Modulo Register  */

#define  T1MOD  DBLREG(0x23)

#define  T1MODH  REGISTER(0x23)
#define  T1MODL  REGISTER(0x24)

/* Timer Status and Control Register Channel 0 */
#define  T1SC0   REGISTER(0x25)
#define  T1SC0_CH0MAX  BIT(0x25,0)
#define  T1SC0_TOV0    BIT(0x25,1)
#define  T1SC0_ELS0A   BIT(0x25,2)
#define  T1SC0_ELS0B   BIT(0x25,3)
#define  T1SC0_MS0A    BIT(0x25,4)
#define  T1SC0_MS0B    BIT(0x25,5)
#define  T1SC0_CH0IE   BIT(0x25,6)
#define  T1SC0_CH0F    BIT(0x25,7)

/* Timer Channel 0 Register */
#define  T1CH0   DBLREG(0x26)
```

```
#define  T1CH0H  REGISTER(0x26)
#define  T1CH0L  REGISTER(0x27)

/* Timer Status and Control Register Channel 1 */
#define  T1SC1   REGISTER(0x28)
#define  T1SC1_CH1MAX  BIT(0x28,0)
#define  T1SC1_TOV1    BIT(0x28,1)
#define  T1SC1_ELS1A   BIT(0x28,2)
#define  T1SC1_ELS1B   BIT(0x28,3)
#define  T1SC1_MS1A    BIT(0x28,4)
#define  T1SC1_CH1IE   BIT(0x28,6)
#define  T1SC1_CH1F    BIT(0x28,7)

/* Timer Channel 1 Register */
#define  T1CH1   DBLREG(0x29)

#define  T1CH1H  REGISTER(0x29)
#define  T1CH1L  REGISTER(0x2a)

/*************************************************************************/
/* Timer Registers #2                                                    */
/*************************************************************************/

/* Timer Status and Control Register */
#define  T2SC            REGISTER(0x2b)
#define  T2SC_PS0        BIT(0x2b,0)
#define  T2SC_PS1        BIT(0x2b,1)
#define  T2SC_PS2        BIT(0x2b,2)
#define  T2SC_TRST       BIT(0x2b,4)
#define  T2SC_TSTOP      BIT(0x2b,5)
#define  T2SC_TOIE       BIT(0x2b,6)
#define  T2SC_TOF        BIT(0x2b,7)

/* Timer Counter Register */
#define  T2CNT  DBLREG(0x2c)

#define  T2CNTH  REGISTER(0x2c)
#define  T2CNTL  REGISTER(0x2d)

/* Timer Modulo Register  */

#define  T2MOD  DBLREG(0x2e)

#define  T2MODH  REGISTER(0x2e)
#define  T2MODL  REGISTER(0x2f)

/* Timer Status and Control Register Channel 0 */
#define  T2SC0   REGISTER(0x30)
#define  T2SC0_CH0MAX  BIT(0x30,0)
#define  T2SC0_TOV0    BIT(0x30,1)
#define  T2SC0_ELS0A   BIT(0x30,2)
```

```
#define  T2SC0_ELS0B   BIT(0x30,3)
#define  T2SC0_MS0A    BIT(0x30,4)
#define  T2SC0_MS0B    BIT(0x30,5)
#define  T2SC0_CH0IE   BIT(0x30,6)
#define  T2SC0_CH0F    BIT(0x30,7)

/* Timer Channel 0 Register */
#define  T2CH0   DBLREG(0x31)

#define  T2CH0H  REGISTER(0x31)
#define  T2CH0L  REGISTER(0x32)

/* Timer Status and Control Register Channel 1 */
#define  T2SC1   REGISTER(0x33)
#define  T2SC1_CH1MAX  BIT(0x33,0)
#define  T2SC1_TOV1    BIT(0x33,1)
#define  T2SC1_ELS1A   BIT(0x33,2)
#define  T2SC1_ELS1B   BIT(0x33,3)
#define  T2SC1_MS1A    BIT(0x33,4)
#define  T2SC1_CH1IE   BIT(0x33,6)
#define  T2SC1_CH1F    BIT(0x33,7)
/* Timer Channel 1 Register */
#define  T2CH1   DBLREG(0x34)

#define  T2CH1H  REGISTER(0x34)
#define  T2CH1L  REGISTER(0x35)


/*****************************************************************************/
/* Phase Locked Loop Module Registers                                        */
/*****************************************************************************/

/* PLL Control Register */
#define  PCTL          REGISTER(0x36)
#define  PCTL_VPR0     BIT(0x36,0)
#define  PCTL_VPR1     BIT(0x36,1)
#define  PCTL_PRE0     BIT(0x36,2)
#define  PCTL_PRE1     BIT(0x36,3)
#define  PCTL_BCS      BIT(0x36,4)
#define  PCTL_PLLON    BIT(0x36,5)
#define  PCTL_PLLF     BIT(0x36,6)
#define  PCTL_PLLIE    BIT(0x36,7)

/* PLL Bandwidth Register */
#define  PBWC          REGISTER(0x37)
#define  PBWC_ACQ      BIT(0x37,5)
#define  PBWC_LOCK     BIT(0x37,6)
#define  PBWC_AUTO     BIT(0x37,7)
```

```
/* PLL Multiplier High Register */
#define  PMSH          REGISTER(0x38)

/* PLL Multiplier Low Register */
#define  PMSL          REGISTER(0x39)

/* PLL VCO Select Range Register */
#define  PMRS          REGISTER(0x3a)

/* PLL Reference Divider Select Register */
#define  PMDS          REGISTER(0x3b)
/*****************************************************************************/
/* Analogue To Digital Converter Registers                                   */
/*****************************************************************************/

/* A/D Status and Control Register */
#define  ADSCR          REGISTER(0x3c)
#define  ADSCR_ADCH0    BIT(0x3c,0)
#define  ADSCR_ADCH1    BIT(0x3c,1)
#define  ADSCR_ADCH2    BIT(0x3c,2)
#define  ADSCR_ADCH3    BIT(0x3c,3)
#define  ADSCR_ADCH4    BIT(0x3c,4)
#define  ADSCR_ADCO     BIT(0x3c,5)
#define  ADSCR_AIEN     BIT(0x3c,6)
#define  ADSCR_COCO     BIT(0x3c,7)

/* A/D-Data Register */
#define  ADR            REGISTER(0x3d)

/* A/D Input Clock Register */
#define  ADCLK          REGISTER(0x3e)
#define  ADCLK_ADICLK   BIT(0x3e,4)
#define  ADCLK_ADIV0    BIT(0x3e,5)
#define  ADCLK_ADIV1    BIT(0x3e,6)
#define  ADCLK_ADIV2    BIT(0x3e,7)



/*****************************************************************************/
/* System Integration Module Registers                                       */
/*****************************************************************************/

/* Break Status Register */
#define  SBSR          REGISTER(0xFE00)
#define  SBSR_SBSW     BIT(0xFE00,1)

/* SIM Reset Status Register */
#define  SRSR         REGISTER(0xFE01)
#define  SRSR_LVI     BIT(0xFE01,1)
#define  SRSR_MODRST  BIT(0xFE01,2)
#define  SRSR_ILAD    BIT(0xFE01,3)
```

```
#define  SRSR_ILOP    BIT(0xFE01,4)
#define  SRSR_COP     BIT(0xFE01,5)
#define  SRSR_PIN     BIT(0xFE01,6)
#define  SRSR_POR     BIT(0xFE01,7)

/* SIM Upper Byte Address Register */
#define  SUBAR         REGISTER(0xFE02)

/* Break Flag Control Register */
#define  BFCR         REGISTER(0xFE03)
#define  BFCR_BCFE    BIT(0xFE03,7)
/* Break Address Registers */
#define  BRKH         REGISTER(0xFE09)
#define  BRKL         REGISTER(0xFE0a)

/* Break Status & Control Register */
#define  BRKSCR       REGISTER(0xFE0b)
#define  BRKSCR_BRKA  BIT(0xFE0b,6)
#define  BRKSCR_BRKE  BIT(0xFE0b,7)


/*************************************************************************/
/* Interrupt Registers                                                   */
/*************************************************************************/

/* Interrupt Status Register 1 */
#define  INT1         REGISTER(0xFE04)
#define  INT1_IF1     BIT(0xFE04,2)
#define  INT1_IF2     BIT(0xFE04,3)
#define  INT1_IF3     BIT(0xFE04,4)
#define  INT1_IF4     BIT(0xFE04,5)
#define  INT1_IF5     BIT(0xFE04,6)
#define  INT1_IF6     BIT(0xFE04,7)

/* Interrupt Status Register 2 */
#define  INT2         REGISTER(0xFE05)
#define  INT2_IF7     BIT(0xFE05,0)
#define  INT2_IF8     BIT(0xFE05,1)
#define  INT2_IF9     BIT(0xFE05,2)
#define  INT2_IF10    BIT(0xFE05,3)
#define  INT2_IF11    BIT(0xFE05,4)
#define  INT2_IF12    BIT(0xFE05,5)
#define  INT2_IF13    BIT(0xFE05,6)
#define  INT2_IF14    BIT(0xFE05,7)

/* Interrupt Status Register 3 */
#define  INT3         REGISTER(0xFE06)
#define  INT3_IF15    BIT(0xFE06,0)
#define  INT3_IF16    BIT(0xFE06,1)
```

```
/*****************************************************************************/
/* Flash Registers                                                           */
/*****************************************************************************/

/* Flash Control Register 1 */
#define  FLCR         REGISTER(0xFE08)
#define  FLCR_PGM     BIT(0xFE08,0)
#define  FLCR_ERASE   BIT(0xFE08,1)
#define  FLCR_MASS    BIT(0xFE08,2)
#define  FLCR_HVEN    BIT(0xFE08,3)

#define  M_FLCR_PGM     BIT0
#define  M_FLCR_ERASE   BIT1
#define  M_FLCR_MARGIN  BIT2
#define  M_FLCR_HVEN    BIT3

/* Flash Block Protect Register 1 */
#define  FLBPR        REGISTER(0xFF7E)
#define  FLBPR_BPR0   BIT(0xFF7E,0)
#define  FLBPR_BPR1   BIT(0xFF7E,1)
#define  FLBPR_BPR2   BIT(0xFF7E,2)
#define  FLBPR_BPR3   BIT(0xFF7E,3)
#define  FLBPR_BPR4   BIT(0xFF7E,4)
#define  FLBPR_BPR5   BIT(0xFF7E,5)
#define  FLBPR_BPR6   BIT(0xFF7E,6)
#define  FLBPR_BPR7   BIT(0xFF7E,7)


/*****************************************************************************/
/* COP Control Registers                                                     */
/*****************************************************************************/

#define  COPCTL REGISTER(0xFFFF)


/*****************************************************************************/
/* Short Aliases                                                             */
/*****************************************************************************/

/* Port A short aliases */
#define PTA0    PTA_BIT0
#define PTA1    PTA_BIT1
#define PTA2    PTA_BIT2
#define PTA3    PTA_BIT3
#define PTA4    PTA_BIT4
#define PTA5    PTA_BIT5
#define PTA6    PTA_BIT6
#define PTA7    PTA_BIT7
```

```
/* Port B short aliases */
#define PTB0    PTB_BIT0
#define PTB1    PTB_BIT1
#define PTB2    PTB_BIT2
#define PTB3    PTB_BIT3
#define PTB4    PTB_BIT4
#define PTB5    PTB_BIT5
#define PTB6    PTB_BIT6
#define PTB7    PTB_BIT7

/* Port C short aliases */
#define PTC0    PTC_BIT0
#define PTC1    PTC_BIT1
#define PTC2    PTC_BIT2
#define PTC3    PTC_BIT3
#define PTC4    PTC_BIT4
#define PTC5    PTC_BIT5
#define PTC6    PTC_BIT6

/* Port D short aliases */
#define PTD0    PTD_BIT0
#define PTD1    PTD_BIT1
#define PTD2    PTD_BIT2
#define PTD3    PTD_BIT3
#define PTD4    PTD_BIT4
#define PTD5    PTD_BIT5
#define PTD6    PTD_BIT6
#define PTD7    PTD_BIT7

/* Port E short aliases */
#define PTE0    PTE_BIT0
#define PTE1    PTE_BIT1

/*************************************************************************/
/* Interrupt Vectors                                                     */
/*************************************************************************/

#define IV_TBM        17    /* Time Base Module                */
#define IV_ADC        16    /* Analogue To Digital Converter   */
#define IV_KBRD       15    /* Keyboard                        */
#define IV_SCI_TX     14    /* Serial Communication Transmit   */
#define IV_SCI_RX     13    /* Serial Communication Receive    */
#define IV_SCI_ERROR  12    /* Serial Communication Error      */
#define IV_SPI_TX     11    /* SPI Transmit                    */
#define IV_SPI_TRX    10    /* SPI Receive                     */
#define IV_T2OVF      9     /* Timer B Channel 1               */
#define IV_T2CH1      8     /* Timer B Channel 0               */
#define IV_T2CHO      7     /* Modulo B Timer                  */
#define IV_T1OVF      6     /* Timer A Channel 1               */
```

```
#define IV_T1CH1      5     /* Timer A Channel 0              */
#define IV_T1CHO      4     /* Modulo A Timer                 */
#define IV_PLL        3     /* Phase Locked Loop              */
#define IV_IRQ1       2     /* Interrupt Request1             */
#define IV_SWI        1     /* Software Interrupt             */
#define IV_RESET      0     /* Reset                          */

#endif
```

rf2.h

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           RF2.H                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    Tango3/Romeo2 control                     *
*      Processor:      HC08                                      *
*      Revision:       1.1                                       *
*      Date:           Jun 5 2002                                *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak, Pavel Lajsner               *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

#ifndef _RF_H
#define _RF_H

#include "board.h"

// Romeo header
#define RF_HEADER_BITS  0x8
#define RF_HEADER   0xc8

// preamble initiating transmission
#define RF_PREAMBLE 0xff
#define RF_PREAMBLE_BITS 8

#if !defined(BUS_CLOCK_HZ)
```

```
#error "RF: need BUS_CLOCK_HZ symbol defined"
#endif
// single bit length
#ifndef RF_SPEED
#define RF_SPEED 9600L
#endif

#ifndef RF_FULLBIT
#define RF_FULLBIT (BUS_CLOCK_HZ)/RF_SPEED
#endif

#define RF_HALFBIT (BUS_CLOCK_HZ)/(2*RF_SPEED)

#define RF_TAILLEN 10L*RF_FULLBIT
#define RF_RXTIMEOUT 16L*RF_FULLBIT

////////////////////////////////////////////////////////
// macros

#define RF_R2CFG_HE   0x01
#define RF_R2CFG_DME  0x02
#define RF_R2CFG_SR0  0x04
#define RF_R2CFG_SR1  0x08
#define RF_R2CFG_SOE  0x10
#define RF_R2CFG_MOD  0x20
#define RF_R2CFG_CF   0x40

#define RF_R2CFG_9600 0xC0
#define RF_R2CFG_4800 0x80
#define RF_R2CFG_2400 0x40
#define RF_R2CFG_1200 0x00

#if RF_SPEED==9600L
#define RF_R2CFG_SPEED RF_R2CFG_9600
#elif RF_SPEED==4800L
#define RF_R2CFG_SPEED RF_R2CFG_4800
#elif RF_SPEED==2400L
#define RF_R2CFG_SPEED RF_R2CFG_2400
#elif RF_SPEED==1200L
#define RF_R2CFG_SPEED RF_R2CFG_1200
#else
//#error "RF: RF_R2CFG_SPEED not defined"
#define RF_R2CFG_SPEED RF_R2CFG_9600
#endif

// several local f() defs
#define RF_TimerReset() RFTimerCTRL_TRST=1
#define RF_TimerStop() RFTimerCTRL_TSTOP=1;
#define RF_TimerStart() RFTimerCTRL_TSTOP=0;

#define RF_TxServiceOn RFTimerTxdEnable
```

```
#define RF_TxServiceOff RFTimerTxdDisable

#define RF_ManchesterOn()  RFTimerTXD_TOV=1;
#define RF_ManchesterOff() RFTimerTXD_TOV=0;
void RF_RxTimeoutOn(void);
void RF_RxTimeoutOff(void);

void DischargeAGC(void);

////////////////////////////////////////////////////////
// global variables

#pragma DATA_SEG SHORT MY_ZEROPAGE

extern volatile BYTE rfTxIdx;
extern volatile BYTE rfTxLen;
extern volatile BYTE rfRxIdx;
extern volatile BYTE rfRxLen;

#pragma DATA_SEG DEFAULT

#define RFTXBUFFSIZE 49
#define RFRXBUFFSIZE RFTXBUFFSIZE+1

extern BYTE RF_TxBuff[RFTXBUFFSIZE];
extern BYTE RF_RxBuff[RFRXBUFFSIZE];

////////////////////////////////////////////////////////
// RF functions
void RF_Init(BYTE* romeoCfg);

// initiate transmission
void RF_TxStart(void);

// initiate reception
void RF_RxStart(void);
void RF_RxStop(void);

// user interrupt-time call-backs prototypes
void RF_TxChar_Int(void);
void RF_TxFinish_Int(void);
void RF_TxFinish(void);
void RF_RxChar_Int(BYTE ch);
void RF_TxChar(BYTE tx);

BYTE RF_RxBuff_OK(void);
void RF_SendBuff(BYTE len);

#endif
```

sci.h

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           SCI.H                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    SCI control                               *
*      Processor:      HC08                                      *
*      Revision:       1.1                                       *
*      Date:           Apr 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak                              *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

#ifndef _SCI_H
#define _SCI_H

#define XON  17
#define XOFF 19

#define DisTxInt() SCC2_SCTIE=0
#define EnaTxInt() SCC2_SCTIE=1
#define DisTcInt() SCC2_TCIE=0
#define EnaTcInt() SCC2_TCIE=1
#define DisRxInt() SCC2_SCRIE=0
#define EnaRxInt() SCC2_SCRIE=1
#define EnaRxIdle() SCC2_ILIE=1;
```

```
#ifdef SCI_FIXED_BUFFS

// we can save 6 bytes when using fixed buffers
#define txBuff sciTxBuff
#define rxBuff sciRxBuff
#define txSize sciTxSize
#define rxSize sciRxSize

#endif

// those bit-mappings should be set by user
#if !defined(sciRxOverflow)
#error "SCI: sciRxOverflow not defined"
#endif
#if !defined(sciRxIdle) && defined(SCI_IDLE_DETECTION)
#error "SCI: sciRxIdle not defined"
#endif
#if !defined(sciRxIdleS) && defined(SCI_IDLE_DETECTION)
#error "SCI: sciRxIdleS not defined"
#endif
#if !defined(sciRxTmout) && defined(SCI_TIMEOUT_DETECTION)
#error "SCI: sciRxTmout not defined"
#endif
#if !defined(sciRxTmoutHlpr) && defined(SCI_TIMEOUT_DETECTION)
#error "SCI: sciRxTmoutHlpr not defined"
#endif

#if defined(SCI_XOXOFF_CONTROL)
  #if !defined(XON)
  #error "SCI: XON char not defined"
  #endif
  #if !defined(XOFF)
  #error "SCI: XOFF char not defined"
  #endif
#endif

#if defined(SCI_CTS_CONTROL)
  #if !defined(SCI_CTS)
  #error "SCI: SCI_CTS port-bit not defined"
  #endif
  #if !defined(SCI_CTS_OFF) || !defined(SCI_CTS_ON)
  #error "SCI: SCI_CTS ON-OFF states not defined"
  #endif
#endif // SCI_CTS_CONTROL

#if defined(SCI_RTS_CONTROL)
  #if !defined(SCI_RTS)
  #error "SCI: SCI_RTS port-bit not defined"
  #endif
  #if !defined(SCI_RTS_OFF) || !defined(SCI_RTS_ON)
```

```
  #error "SCI: SCI_RTS ON-OFF states not defined"
  #endif
  // RTS is not yet implemented !
  #error "SCI: SCI_RTS_CONTROL is not yet implemented"
#endif // SCI_RTS_CONTROL
#if defined(SCI_TXE_CONTROL)
  #if !defined(SCI_TXE)
  #error "SCI: SCI_TXE port-bit not defined"
  #endif
  #if !defined(SCI_TXE_OFF) || !defined(SCI_TXE_ON)
  #error "SCI: SCI_TXE ON-OFF states not defined"
  #endif
    // separate RX enable pin handling ?
    #ifdef SCI_TXE_CONTROL_RX
    #if !defined(SCI_RXE)
    #error "SCI: SCI_RXE port-bit not defined (and SCI_TXE_CONTROL_RX defined)"
    #endif
    #if !defined(SCI_RXE_OFF) || !defined(SCI_RXE_ON)
    #error "SCI: SCI_RXE ON-OFF states not defined (and SCI_TXE_CONTROL_RX defined)"
    #endif
  #endif // SCI_TXE_CONTROL_RX
#endif // SCI_TXE_CONTROL

// SCI clock should be specified separately from clock freq
// since it might depend on SCI clock source setting etc.
#ifdef SCI_CLOCK_HZ

#if SCI_CLOCK_HZ==3686400     // SCICLOCK = BUS clock 3686400, XTAL/VCO 14745600,
CONFIG2 = 0x01 (SCIBDSRC (bit 0) = 1)
#define SCI_SCBR_57600  0x00  // 57600
#define SCI_SCBR_28800  0x01  // 28800
#define SCI_SCBR_19200  0x10  // 19200
#define SCI_SCBR_14400  0x02  // 14400
#define SCI_SCBR_9600   0x11  // 9600
#define SCI_SCBR_7200   0x03  // 7200
#define SCI_SCBR_4800   0x12  // 4800
#define SCI_SCBR_3600   0x04  // 3600
#define SCI_SCBR_2400   0x13  // 2400
#define SCI_SCBR_1800   0x05  // 1800
#define SCI_SCBR_1200   0x14  // 1200

#elif SCI_CLOCK_HZ==4915200   // SCI clock 4915200
#define SCI_SCBR_76800  0x00  // 76800
#define SCI_SCBR_38400  0x01  // 38400
#define SCI_SCBR_19200  0x02  // 19200
#define SCI_SCBR_9600   0x03  // 9600
#define SCI_SCBR_4800   0x04  // 4800
#define SCI_SCBR_2400   0x05  // 2400
#define SCI_SCBR_1200   0x06  // 1200
```

```
#elif SCI_CLOCK_HZ==5000000   // SCI clock 5000000
#define SCI_SCBR_76800  0x00  // 78125
#define SCI_SCBR_38400  0x01  // 39063
#define SCI_SCBR_19200  0x02  // 19531
#define SCI_SCBR_9600   0x03  // 9766
#define SCI_SCBR_4800   0x04  // 4883
#define SCI_SCBR_2400   0x05  // 2441
#define SCI_SCBR_1200   0x06  // 1221
#elif SCI_CLOCK_HZ==7372800   // SCI clock 7372800
#define SCI_SCBR_115200 0x00  // 115200
#define SCI_SCBR_57600  0x01  // 57600
#define SCI_SCBR_38400  0x10  // 38400
#define SCI_SCBR_28800  0x02  // 28800
#define SCI_SCBR_19200  0x11  // 19200
#define SCI_SCBR_14400  0x03  // 14400
#define SCI_SCBR_9600   0x12  // 9600
#define SCI_SCBR_7200   0x04  // 7200
#define SCI_SCBR_4800   0x13  // 4800
#define SCI_SCBR_3600   0x05  // 3600
#define SCI_SCBR_2400   0x14  // 2400
#define SCI_SCBR_1800   0x06  // 1800
#define SCI_SCBR_1200   0x15  // 1200

#elif SCI_CLOCK_HZ==8000000   // SCI clock 8000000
#define SCI_SCBR_9600   0x30  // 9615
#define SCI_SCBR_4800   0x31  // 4807
#define SCI_SCBR_2400   0x32  // 2403
#define SCI_SCBR_1200   0x33  // 1201

#elif SCI_CLOCK_HZ==9830400   // SCI clock 9830400
#define SCI_SCBR_153600 0x00  // 153600
#define SCI_SCBR_76800  0x01  // 76800
#define SCI_SCBR_38400  0x02  // 38400
#define SCI_SCBR_19200  0x03  // 19200
#define SCI_SCBR_9600   0x04  // 9600
#define SCI_SCBR_4800   0x05  // 4800
#define SCI_SCBR_2400   0x06  // 2400
#define SCI_SCBR_1200   0x07  // 1200

#elif SCI_CLOCK_HZ==14745600  // SCI clock 14745600
#define SCI_SCBR_230400 0x00  // 230400
#define SCI_SCBR_115200 0x01  // 115200
#define SCI_SCBR_76800  0x10  // 76800
#define SCI_SCBR_57600  0x02  // 57600
#define SCI_SCBR_38400  0x11  // 38400
#define SCI_SCBR_28800  0x03  // 28800
#define SCI_SCBR_19200  0x12  // 19200
#define SCI_SCBR_14400  0x04  // 14400
#define SCI_SCBR_9600   0x13  // 9600
#define SCI_SCBR_7200   0x05  // 7200
#define SCI_SCBR_4800   0x14  // 4800
```

```
#define SCI_SCBR_3600   0x06  // 3600
#define SCI_SCBR_2400   0x15  // 2400
#define SCI_SCBR_1800   0x07  // 1800
#define SCI_SCBR_1200   0x16  // 1200

#elif SCI_CLOCK_HZ==16000000  // SCI clock 16000000
#define SCI_SCBR_19200  0x30  // 19230
#define SCI_SCBR_9600   0x31  // 9615
#define SCI_SCBR_4800   0x32  // 4807
#define SCI_SCBR_2400   0x33  // 2403
#define SCI_SCBR_1200   0x34  // 1201

#elif SCI_CLOCK_HZ==19660800  // SCI clock 19660800
#define SCI_SCBR_307200 0x00  // 307200
#define SCI_SCBR_153600 0x01  // 153600
#define SCI_SCBR_76800  0x02  // 76800
#define SCI_SCBR_38400  0x03  // 38400
#define SCI_SCBR_19200  0x04  // 19200
#define SCI_SCBR_9600   0x05  // 9600
#define SCI_SCBR_4800   0x06  // 4800
#define SCI_SCBR_2400   0x07  // 2400
#define SCI_SCBR_1200   0x26  // 1200

#elif SCI_CLOCK_HZ==20000000  // SCI clock 20000000
#define SCI_SCBR_307200 0x00  // 312500
#define SCI_SCBR_153600 0x01  // 156250
#define SCI_SCBR_76800  0x02  // 78125
#define SCI_SCBR_38400  0x03  // 39063
#define SCI_SCBR_19200  0x04  // 19531
#define SCI_SCBR_9600   0x05  // 9766
#define SCI_SCBR_4800   0x06  // 4883
#define SCI_SCBR_2400   0x07  // 2441
#define SCI_SCBR_1200   0x26  // 1221

#elif SCI_CLOCK_HZ==29491200  // SCI clock 29491200
#define SCI_SCBR_460800 0x00  // 460800
#define SCI_SCBR_230400 0x01  // 230400
#define SCI_SCBR_153600 0x10  // 153600
#define SCI_SCBR_115200 0x02  // 115200
#define SCI_SCBR_76800  0x11  // 76800
#define SCI_SCBR_57600  0x03  // 57600
#define SCI_SCBR_38400  0x12  // 38400
#define SCI_SCBR_28800  0x04  // 28800
#define SCI_SCBR_19200  0x13  // 19200
#define SCI_SCBR_14400  0x05  // 14400
#define SCI_SCBR_9600   0x14  // 9600
#define SCI_SCBR_7200   0x06  // 7200
#define SCI_SCBR_4800   0x15  // 4800
#define SCI_SCBR_3600   0x07  // 3600
#define SCI_SCBR_2400   0x16  // 2400
#define SCI_SCBR_1200   0x17  // 1200
```

```
#elif SCI_CLOCK_HZ==32000000  // SCI clock 32000000
#define SCI_SCBR_38400  0x30  // 38462
#define SCI_SCBR_19200  0x31  // 19230
#define SCI_SCBR_9600   0x32  // 9615
#define SCI_SCBR_4800   0x33  // 4807
#define SCI_SCBR_2400   0x34  // 2403
#define SCI_SCBR_1200   0x35  // 1201

#else  /* SCI_CLOCK_HZ==xxx */

#error "SCI: baudrate not defined"

#endif /* SCI_CLOCK_HZ==xxx */

#else  /* #ifdef SCI_CLOCK_HZ */

#error "SCI: SCI_CLOCK_HZ not defined"

#endif /* #ifdef SCI_CLOCK_HZ */

////////////////////////////////////////////////
// global variables

#pragma DATA_SEG SHORT MY_ZEROPAGE

extern BYTE sciRxLen;
#ifdef SCI_XONXOFF_CONTROL
extern BYTE sciTxPriorityByte;  /* SCI transmitter: XONXOFF control priority byte */
#endif

#pragma DATA_SEG DEFAULT

////////////////////////////////////////////////
// shared functions

#ifdef SCI_FIXED_BUFFS

// no parameters if we use fixed buffers
void SCI_InitTx(void);
void SCI_InitRx(void);

#else

// parameters required if we use runtime-defined buffers
void SCI_InitTx(BYTE* buff, BYTE size);
void SCI_InitRx(BYTE* buff, BYTE size);

#endif
```

```
// sending/receiving data
BOOL SCI_TxBuff(BYTE* buff, BYTE len);
BOOL SCI_TxInPlaceSend(BYTE* buff, BYTE len);
BYTE SCI_RxPoll(BYTE* buff, BYTE len);

void SciRx_Off(void);
void SciRx_On(void);


// safe read of SCI received characters count
// it is pure read of global variable now since it is single byte operation
// and no sync is needed (this may change if app is ported)
#define SCI_RxLen() sciRxLen

#endif
```

std.h

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           STD.C                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    Standard header                           *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak, Pavel Lajsner               *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

#ifndef _STD_H
#define _STD_H

////////////////////////////////////////////////
// this header should define all symbols
// needed by standard modules (SCI, RF, ...)

// global types

#define  BYTE unsigned char
#define  WORD unsigned short
#define  DWORD  unsigned long
#define  BOOL unsigned char
```

```
// macros

#define EnaInts() asm cli
#define DisInts() asm sei

#define FALSE 0
#define TRUE  1


// global symbols defined to specific values required by this
// project are in main project header file

#include "wem.h"

#endif
```

tbm.h

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           TBM.C                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    TBM control                               *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak                              *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

#ifndef _TBM_H
#define _TBM_H

void TBM_Enable(void);
void TBM_Disable(void);


#endif
```

wem.h

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           WEM.H                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    Wireless RS232 demo main header file      *
*                      This file is included via std.h into all  *
*                      standard files and sets their control macros *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak                              *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT       *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;       *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)           *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)      *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

#ifndef _WEM_H
#define _WEM_H

#include "hc08gp32.h"


// our clock speeds (needed by other modules)
#define BUS_CLOCK_HZ 3686400
#define SCI_CLOCK_HZ 3686400

#if BUS_CLOCK_HZ==7372800
```

```
#define PMSHDEF 0x03
#define PMSLDEF 0x84
#define PCTL_VPR1DEF 1
#define PCTL_VPR0DEF 0   /* E=2 */
#define PMRSDEF 0xC0
// our SCI baudrate (see sci.h for known values)
#define SCI_BAUDRATE SCI_SCBR_38400

#elif BUS_CLOCK_HZ==3686400

#define PMSHDEF 0x01
#define PMSLDEF 0xC0
#define PCTL_VPR1DEF 0
#define PCTL_VPR0DEF 1   /* E=1 */
#define PMRSDEF 0xC0
// our SCI baudrate (see sci.h for known values)
#define SCI_BAUDRATE SCI_SCBR_57600

#endif

// SCI/RS232 port related usage switches

//#define SCI_IDLE_DETECTION
#define SCI_TIMEOUT_DETECTION // timeout detection on RX line
//#define SCI_XONXOFF_CONTROL 1

// use Error Correction Codes
//#define RF_ECC

// use COP
#define COP_ENABLE 1

#ifdef COP_ENABLE
#define CONFIG1DEF 0x80;    /* COP enabled */
#else
#define CONFIG1DEF 0x01;    /* COP disabled */
#endif


////////////////////////////////////////////////////
// system flags types


typedef struct
{
  unsigned int rfTxActive : 1;    /* RF transmitter: 1 = just transmitting */
  unsigned int rfRxActive : 1;    /* RF receiver: 1 = in listen state */
#ifdef RF_ECC
  unsigned int eccNibble : 1;  /* TEST */
#endif
```

```
} SYSFLAGS1;

typedef struct
{
  unsigned int sciRxOverflow : 1; /* SCI receiver: buffer overflow */
  unsigned int sciRxIdle : 1;   /* SCI receiver: idle now */
  unsigned int sciRxIdleS : 1;    /* SCI receiver: idle sticky bit */
  unsigned int sciRxTmout : 1;    /* SCI receiver: TBM controlled receiver timout */
  unsigned int sciRxTmoutHlpr : 1;  /* SCI receiver: TBM controlled receiver timout
(helper bit)*/
#ifdef SCI_XONXOFF_CONTROL
  unsigned int sciTxPriority : 1;  /* SCI transmitter: XONXOFF control priority bit */
#endif
} SYSFLAGS2;

////////////////////////////////////////////////////
// global variables

#pragma DATA_SEG SHORT MY_ZEROPAGE

extern SYSFLAGS1 sys1;
extern SYSFLAGS2 sys2;
extern BYTE i,j;
extern BYTE *p;

#pragma DATA_SEG DEFAULT


////////////////////////////////////////////////////
// SCI stuff

// we use fixed buffers (allocated in wrs.c)
#define SCI_FIXED_BUFFS
extern BYTE sciTxBuff[];
extern BYTE sciRxBuff[];
#define sciTxSize 47
#define sciRxSize 46

// map SCI bits into our global flags
#define sciRxOverflow sys2.sciRxOverflow  // SCI receiver buffer overflow
#define sciRxIdle sys2.sciRxIdle      // SCI receiver is idle
#define sciRxIdleS sys2.sciRxIdleS      // SCI receiver was idle (sticky bit)
#define sciRxTmout sys2.sciRxTmout      // SCI receiver timeout (TBM)
#define sciRxTmoutHlpr sys2.sciRxTmoutHlpr  // SCI receiver timeout helper bit (TBM)
#ifdef SCI_XONXOFF_CONTROL
#define sciTxPriority sys2.sciTxPriority /* SCI transmitter: XONXOFF control priority
bit */
#endif
```

```
//////////////////////////////////////////////////
// RF stuff


// map RF bits into our global flags
#define rfTxActive sys1.rfTxActive    // RF transmitter activity
#define rfRxActive sys1.rfRxActive      // RF receiver activity

#ifdef RF_ECC
#define eccNibble sys1.eccNibble
#endif



extern int calc_crc(char *buf, unsigned char n);

#endif
```

rf2.c

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           RF2.C                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    Tango2/Romeo2 control                     *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak, Pavel Lajsner               *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

/* Include Header Files */


#include “board.h”
#include “std.h”
#include “rf2.h”

#pragma DATA_SEG SHORT MY_ZEROPAGE

// local variables

// rx and tx tmp variables may overlap
static union
{
```

```
  struct
  {
    BYTE  txChar;   // currently transmitted char
    BYTE  txBits;   // transmit bit counter
  } tx;

  struct
  {
    BYTE rxChar;
    BYTE rxTimeout;
  } rx;
} rtx;


// status bits dedicated to internal rx and tx operation

static union
{
  struct
  {
    unsigned int txPre : 1;   /* preamble to be transmitted */
    unsigned int txHead : 1;    /* header to be transmitted */
    unsigned int txTail : 1;    /* tail to be transmitted */
  } bit;

  BYTE all;
} rfFlgs;

volatile BYTE rfTxIdx;
volatile BYTE rfTxLen;
volatile BYTE rfRxIdx;
volatile BYTE rfRxLen;

#pragma DATA_SEG DEFAULT

BYTE RF_TxBuff[RFTXBUFFSIZE];
BYTE RF_RxBuff[RFRXBUFFSIZE];

// this mask should cover txTail | txPre | txHead
#define SPECIALTX_MASK  0x07

// shortcuts are used in the code below
#define txChar rtx.tx.txChar
#define txBits rtx.tx.txBits
#define rxChar rtx.rx.rxChar

#define txPre  rfFlgs.bit.txPre
#define txHead rfFlgs.bit.txHead
#define txTail rfFlgs.bit.txTail
#define thisBit rfFlgs.bit.thisBit
```

```
//////////////////////////////////////////////////////
// RF initialization

// basic init, it is assumed that timers and
// other resources are in after-reset state

void RF_Init(BYTE* romeoCfg)
{
  RF_RxTimeoutOff();    // disable RX timeout for case we're in receiving some bytes

  // transmitter idle
  rfTxActive = 0;

  // receiver also idle
  rfRxActive = 0;

  StrobeEnable();

  RF_RESETB = 1;  // needs to be 1 before entering reset

  // shut down the transmitter
  RF_DAT = 0;
  RF_PLLEN = 0;
  RFModeDisable();

  // configure TIM2 CH0 (RF_ManchesterOff)
  RFTimerTXD = 0x10;  // no irq, output compare, pin ctl, no tgl, no chmax

  RF_RESETB = 0;  // enter configuration state

  // setup SPI
  SPCR = 0x28;  // no ints, master, cpol=0, cpha=1
  SPSCR = 3;    // baud rate divisor 128
  SPCR_SPE = 1; // enable and reset SPI

  while(!SPSCR_SPTE) {/* DO NOTHING & WAIT! */};
  for(i=0; i<3; i++)
  {
    SPDR = romeoCfg[i];
    // wait for finishing the transmission
    while(!SPSCR_SPRF) {/* DO NOTHING & WAIT! */};
    // clear SPRF
    SPSCR; SPDR;
  }

  DischargeAGC();

  SPCR = 0x8;   // put SPI into reset (as slave)
  SPCR_SPE = 0; // disable SPI
  RF_RESETB = 1;  // enable ROMEO as SPI master
}
```

```
//////////////////////////////////////////////////////
// RF transmitter part
#pragma INLINE
void RFTimerOvEnable(void)
{
  RFTimerCTRL;      // ctrl reg. bulk read, needed for correct function of timer
  RFTimerCTRL_TOF = 0;
  RFTimerCTRL_TOIE = 1; // Overflow Interrupt enable
}

#pragma INLINE
void RFTimerOvDisable(void)
{
  RFTimerCTRL_TOIE = 0;   // Overflow Interrupt disable TIM2
}

#pragma INLINE
void RFTimerTmoutEnable(void)
{
  RFTimerTMOUT;
  RFTimerTMOUT_CHF = 0;         /* clearing RFTimer CH0 flag */

  RFTimerTMOUT_CHIE = 1;      /* RFTimer Channel 0 Interrupt enable */
}

#pragma INLINE
void RFTimerTmoutDisable(void)
{
  RFTimerTMOUT_CHIE = 0;      /* RFTimer Channel 0 Interrupt disable */
}

#pragma INLINE
void RFTimerTxdEnable(void)
{
  RFTimerTXD;
  RFTimerTXD_CHF = 0;         /* clearing RFTimer CH1 flag */

  RFTimerTXD_CHIE = 1;      /* RFTimer Channel 1 Interrupt enable */
}

#pragma INLINE
void RFTimerTxdDisable(void)
{
  RFTimerTXD_CHIE = 0;      /* RFTimer Channel 1 Interrupt disable */
}

#pragma INLINE
void RF_RxTimeoutOn(void)
{
  RFTimerMOD = -1;
  RFTimerTmoutEnable();
```

```
  RF_TimerStart();
}

#pragma INLINE
void RF_RxTimeoutOff(void)
{
  RFTimerTmoutDisable();
}

////////////////////////////////////////////////////////
// initiate the transmission
void RF_TxStart(void)
{
  LedFn1On();

  IfBtnJp1BuzzerOn();

  // shut down receiver (we don't want the loopback)
  // since we share the rx and tx registers
  RF_RxStop();

  // disable RF-related interrupts
  RF_TxServiceOff();
  RF_ManchesterOff();

  RF_TimerStop();   // stop the timer
  RF_TimerReset();  // and reset the timer

  rfTxActive = 1;   // signal transmitter busy

  rfFlgs.all = 0;   // clear all tmp flags
  txBits = 0;     // starting with special signal

  RF_PLLEN = 1;   // PLL ENABLE
  RF_DAT = 0;     // DATA low at the begining

  txPre = 1;    // start with preamble
  txHead = 1;   // header processing will start data transmission also

#ifdef RF_ECC
  eccNibble = 0;  // initial state
#endif

  // so toggle has the right data to toggle
  RFTimerTXD_MSA = 1;     // output compare (or PWM) (0:1)

  RFTimerTXD_ELSB = 1;     // output compare, set on compare (1:1)
  RFTimerTXD_ELSA = 1;     // output compare, set on compare (1:1)
```

```
  RFTimerCHTXD = RF_HALFBIT;  // normal timing for complete message
  RFTimerMOD = RF_FULLBIT;  // normal timing for complete message

  // start services
  RF_TxServiceOn();
  RF_ManchesterOn();
  // start the timer
  RF_TimerStart();
}

void RF_SendBuff(BYTE len)
{
int _crc;

  RF_TxBuff[0] = len+2;

  _crc = calc_crc(RF_TxBuff, len);

  RF_TxBuff[len++] = (BYTE) (_crc & 0x00ff);        // lo CRC byte first
  RF_TxBuff[len++] = (BYTE) ((_crc & 0xff00) >> 8); // hi CRC byte then

  rfTxLen = len;
  rfTxIdx = 0;

  RF_TxStart();
}


#pragma INLINE
void TXFinish(void)
{
  // shut down the transmitter
  RF_DAT = 0;
  RF_PLLEN = 0;

  RFModeDisable();

  // turn off Manchester coding
  RF_ManchesterOff();
  RF_TxServiceOff();

  // signal transmitter ready again
  rfTxActive = 0;

  // callback the user, he can start another transmission
  RF_TxFinish_Int();

  LedFn1Off();
}
```

```
//////////////////////////////////////////////////////////////
// overflow interrupt toggles the data bit to achieve Manchester coding
#pragma TRAP_PROC
void RFTimerOv_Int(void)
{
  RFTimerCTRL;      // ctrl reg. bulk read, needed for correct function of timer
  RFTimerCTRL_TOF = 0;   // just for the case
}
//////////////////////////////////////////////////////////////////////////
// output compare interrupt, all tango transmission is done here
#pragma TRAP_PROC
void RFTimerTxd_Int(void)
{
#define nextBit BIT(&txChar,7)

  // note: we don't serve the interrupt always, in some cases we prefer to
  // re-call this function immediatlly.

  // any bits to send ?
  if(txBits)
  {
    // serve the interrupt
    RFTimerTXD; RFTimerTXD_CHF = 0;         /* clearing RFTimer CH1 flag */

    // prepare next bit
    RFTimerTXD_ELSA = nextBit;     // output compare, set/clear on compare (1:x)

    // any bit for next time ?
    if(--txBits)
      // yes, prepare it
      txChar <<= 1;
   // no bit for next time, process next data only if no special TX request is pending
    // if there is one, it gets processed next time because of txBits==0
    else if(!(rfFlgs.all & SPECIALTX_MASK))
    {
      RF_TxChar_Int();

      // any next bits ?
      if(!txBits) // no more data for next round
      {
        // send tail
        txBits = 1; // send one more bit for Romeo2
        txTail = 1;
      }
    }

  }
  // !txBits => time to special ctl TX
  else
  {
    // tail = long non-manchester one
```

```
    if(txTail)
    {
      // if no head is pending, this is End-Of-Packet tail
      // tail done
      txTail = 0;
    }
        // preamble
    else if(txPre)
    {
      // our preamble will be our next char
      txChar = RF_PREAMBLE;
      txBits = RF_PREAMBLE_BITS;
      // preamble is done
      txPre = 0;
      // don't serve interrupt
      return;
    }
    // header
    else if(txHead)
    {
      // our character is header
      txChar = RF_HEADER;
      txBits = RF_HEADER_BITS;
      // head done
      txHead = 0;
      // don't serve interrupt
      return;
    }
    // nothing => finish
    else
      TXFinish();

    // serve the interrupt for branches which have not called 'return'
    RFTimerTXD; RFTimerTXD_CHF = 0;         /* clearing RFTimer CH1 flag */
  }
}

//////////////////////////////////////////////////////////////////////////
// output compare interrupt toggles the data bit to achieve Manchester coding
#pragma TRAP_PROC
void RFTimerTmout_Int(void)
{
  RFTimerTMOUT;
  RFTimerTMOUT_CHF = 0;         /* clearing RFTimer CH0 flag */

  RF_RxStop();
  LedFn2Off();

  rfRxLen = rfRxIdx;      // copy the length
  RF_RxTimeoutOff();      // and disable this timer
}
```

```
////////////////////////////////////////////////////////
// RF receiver part
void DischargeAGC(void)
{
BYTE i=0xff;

  // ground AGC capacitors
  RFDchrgOutputs();
  RFDchrgCMixAgcOff();
  RFDchrgCAgcOff();

  // delay ... ??
  while (i--)
  { asm nop; asm nop; }

  // enable AGC capacitors again
  RFDchrgInputs();
}

////////////////////////////////////////////////////////
// enable receiver
void RF_RxStart(void)
{
  // get ready to listen
  DischargeAGC();
  // reconfigure SPI (slave, no ints, cpol=0, cpha=1, disabled)
  SPCR = 0x08;
  // clear any pending SPI RX interrupt
  SPSCR; SPDR;
  // signal receiver is listenning
  rfRxActive = 1;
  // enable SPI RX int
  SPCR_SPRIE = 1;
  // last: enable SPI
  SPCR_SPE = 1;

  // initialize buffer oriented variables
  rfRxLen = 0;
  rfRxIdx = 0;

#ifdef RF_ECC
  eccNibble = 0;
#endif

  // intialize RX timeout int. but still do not enable!
  RFTimerTMOUT_MSA = 1;
  RFTimerTMOUT_ELSB = 0;
  RFTimerTMOUT_ELSA = 0;
  RFTimerCHTMOUT = RF_RXTIMEOUT;
}
```

```
void RF_RxStop(void)
{
  // shut down SPI (slave, no ints, cpol=0, cpha=1, disabled)
  SPCR = 0x8;
  // and signal receiver idle state
  rfRxActive = 0;
}

//////////////////////////////////////////
// SPI RX interrupt, we got data from ROMEO
#pragma TRAP_PROC
void SpiRx_Int(void)
{
  // serve the interrupt
  SPSCR;

  // call back to user code
  RF_RxChar_Int(SPDR);
  LedFn2On();

  IfBtnJp1BuzzerOff();

}

////////////////////////////////////////////////////////////
// user "interrupt acknowledges"

// continue transmission with given char
// called from RF_TxChar_Int() to continue transmission
#pragma INLINE
void RF_TxChar(BYTE tx)
{
  txChar = tx;  // store the byte
  txBits = 8;   // this also flags that valid byte was stored
}

// finish transmission, no more data
// called from RF_TxChar_Int() to finish transmission
#pragma INLINE
void RF_TxFinish(void)
{
  // do nothing is enough (leaves txBits == 0)
}

// get received byte
// called from RF_RxChar_Int()
#pragma INLINE
BYTE RF_RxChar(void)
{
  return rxChar;
}
```

sci.c

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           SCI.C                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    SCI control                               *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak                              *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

/* Include Header Files */

#include "std.h"
#include "sci.h"

////////////////////////////////////////////////////////
// global vars

#pragma DATA_SEG SHORT MY_ZEROPAGE

BYTE sciRxLen;

#ifdef SCI_XONXOFF_CONTROL
BYTE sciTxPriorityByte;   /* SCI transmitter: XONXOFF control priority byte */
#endif

#pragma DATA_SEG DEFAULT
```

```
////////////////////////////////////////////////////////////
// local vars

#pragma DATA_SEG SHORT MY_ZEROPAGE

#ifndef SCI_FIXED_BUFFS

static BYTE* txBuff;
static BYTE  txSize;
static BYTE* rxBuff;
static BYTE  rxSize;

#endif

static char  txWP;
static char  txRP;
static char  rxWP;
static char  rxRP;
static BYTE  _scs1;
static BYTE  _rxch;

#pragma DATA_SEG DEFAULT

// local macros


////////////////////////////////////////////////////////////////////////
// transmitter

// init SCI transmitter BYTE queue

#ifdef SCI_FIXED_BUFFS
void SCI_InitTx(void)
#else
void SCI_InitTx(BYTE* buff, BYTE size)
#endif
{
#ifndef SCI_FIXED_BUFFS
  txBuff = buff;
  txSize = size;
#endif
  txWP = 0;
  txRP = 0;

#ifdef SCI_TXE_CONTROL
  // turn off the line driver
  SCI_TXE = SCI_TXE_OFF;
#ifdef SCI_TXE_CONTROL_RX
  // and turn on the receiver
  SCI_RXE = SCI_TXE_ON;
#endif
```

```
#endif
#ifdef SCI_XONXOFF_CONTROL
  sciTxPriority = 0;
#endif
}

// copy data to tx buffer, do not call in interrupts (using global i)

BOOL SCI_TxBuff(BYTE* buff, BYTE len)
{
  // free space in queue
  DisTxInt();
  i = txRP - txWP;
  if(txRP <= txWP)
    i += txSize;
  // continue with previous chars
  EnaTxInt();

  // space to copy data ? (not that there must always be one BYTE left free)
  if(len >= i)
    return 0;

  for(; len; len--)
  {
    txBuff[txWP++] = *buff++;
    if(txWP == txSize)
      txWP = 0;
  }

  // start the tx if previously stopped
  EnaTxInt();
  return 1;
}

// send data prepared already in buffer, can be called from interrupts
// no checking if valid data are passed !!

BOOL SCI_TxInPlaceSend(BYTE* buff, BYTE len)
{
  // free space in queue
  DisTxInt();
    txRP = (int)buff - (int)txBuff;
    txWP = txRP + len;
  EnaTxInt();
  return 1;
}

#pragma TRAP_PROC
void SciTx_Int(void)
{
  // serve interrupt, two possible sources here: SCTE, TC
  _scs1 = SCS1;
```

```
#ifdef SCI_XONXOFF_CONTROL
  // ready to receive
  if(sciTxPriority == 1)
  {
    sciTxPriority = 0;
    SCDR = sciTxPriorityByte;
  }
  else
#endif
  {

    // any next char ?
    if(txWP == txRP)
    {
      // no, disable SCTE interrupt
      DisTxInt();

  #ifdef SCI_TXE_CONTROL
          // if this is TC interrupt...
          if(_scs1 & 0x40)
          {
            // turn off the line driver
            SCI_TXE = SCI_TXE_OFF;
  #ifdef SCI_TXE_CONTROL_RX
            // and turn on the receiver
            SCI_RXE = SCI_TXE_ON;
  #endif
              // no more TC interrupt needed
              DisTcInt();
          }
  #endif
    }
    else
    {
      // yes, send another char
  #ifdef SCI_TXE_CONTROL
      // turn on the line driver
      SCI_TXE = SCI_TXE_ON;
  #ifdef SCI_TXE_CONTROL_RX
      // and turn off the receiver (avoid loopback)
      SCI_RXE = SCI_TXE_OFF;
  #endif
  #endif

          // fetch the char
      SCDR = txBuff[txRP++];
      if(txRP == txSize)
        txRP = 0;

  #ifdef SCI_TXE_CONTROL
      // enable TC interrupt to detect end-of-transmission
```

```
      EnaTcInt();
  #endif
    }
  }
}

////////////////////////////////////////////////////////////////////////
// receiver

// assign RX buffer and it size to SCI receiver

#ifdef SCI_FIXED_BUFFS
void SCI_InitRx(void)
#else
void SCI_InitRx(BYTE* buff, BYTE size)
#endif
{
#ifndef SCI_FIXED_BUFFS
  rxBuff = buff;
  rxSize = size;
#endif

  rxWP = 0;
  rxRP = 0;
  sciRxLen = 0;
  sciRxOverflow = 0;
  EnaRxInt();
#ifdef SCI_IDLE_DETECTION
  EnaRxIdle();
#endif

#ifdef SCI_CTS_CONTROL
  // ready to receive
  SCI_CTS = SCI_CTS_ON;
#endif

#ifdef SCI_XONXOFF_CONTROL
  // ready to receive
  sciTxPriority = 0;
  sciTxPriorityByte = XON;
#endif

}

// poll len bytes from queue (no sanity checks!)
// don't call with len==0 (CTS is always set here)

BYTE SCI_RxPoll(BYTE* buff, BYTE len)
{
  for(i=0; i<len; i++)
  {
    *buff++ = rxBuff[rxRP++];
```

```
    // fast modulo
    if(rxRP == rxSize)
      rxRP = 0;
  }

  // space for other bytes
  sciRxLen -= len;

  return len;   // and return # of bytes delivered
}

void SciRx_On(void)
{
#ifdef SCI_CTS_CONTROL
  // signal it
  SCI_CTS = SCI_CTS_ON;
#endif
#ifdef SCI_XONXOFF_CONTROL
  // ready to receive
  sciTxPriority = 1;
  sciTxPriorityByte = XON;
  EnaTxInt();
#endif
  EnaRxInt();
}

void SciRx_Off(void)
{
#ifdef SCI_CTS_CONTROL
  // signal it
  SCI_CTS = SCI_CTS_OFF;
#endif
#ifdef SCI_XONXOFF_CONTROL
  // ready to receive
  sciTxPriority = 1;
  sciTxPriorityByte = XOFF;
  EnaTxInt();
#endif
//  DisRxInt();
}

#pragma TRAP_PROC
void SciRx_Int(void)
{
  // serve the interrupt
  _scs1 = SCS1;
  _rxch = SCDR;

  // note: if no idle detection is needed, the
  // only reason for interrupt is SCRF (no need for if)
#ifdef SCI_IDLE_DETECTION
  // SCRF interrupt ?
```

```
  if(_scs1 & 0x20)
#endif
  {
    if(sciRxLen < rxSize)
    {
      sciRxLen++;
      // store recvd char and advance pointer
      rxBuff[rxWP++] = _rxch;
      // fast modulo
      if(rxWP == rxSize)
        rxWP = 0;
#ifdef SCI_CTS_CONTROL
      // signal that buffer is nearly full
      if(sciRxLen >= rxSize-1)
        SCI_CTS = SCI_CTS_OFF;
#endif
#ifdef SCI_XONXOFF_CONTROL
      if (sciRxLen >= 10/*rxSize-1*/)
      {
        sciTxPriority = 1;
        sciTxPriorityByte = XOFF;
        EnaTxInt();
      }
#endif
    }
    //  buffer overflow (set sticky bit)
    else sciRxOverflow = 1;

#ifdef SCI_IDLE_DETECTION
    // we are not idle now
    sciRxIdle = 0;
#endif

#ifdef SCI_TIMEOUT_DETECTION
    // not timeout now also
    sciRxTmout = 0;
    sciRxTmoutHlpr = 0;
#endif SCI_TIMEOUT_DETECTION
  }

#ifdef SCI_IDLE_DETECTION
  // IDLE interrupt ?
  if(_scs1 & 0x10)
  {
    // receiver is idle now
    sciRxIdle = 1;
    // set the sticky bit also
    sciRxIdleS = 1;
  }
#endif
}
```

tbm.c

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           TBM.C                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    TBM control                               *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak                              *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

#include "board.h"
#include "std.h"
#include "wem.h"
#include "tbm.h"
#include "sci.h"
#include "rf2.h"

#pragma DATA_SEG SHORT MY_ZEROPAGE

static BYTE div;

#pragma DATA_SEG DEFAULT
```

```
//////////////////////////////////////////////////////
// Timer Base Module interrupt service routine (1 x base)

#pragma INLINE
void Period_1x(void)
{
#ifdef SCI_TIMEOUT_DETECTION
  // SCI timeout
  sciRxTmout = sciRxTmoutHlpr;
  sciRxTmoutHlpr = 1;
#endif
}

#pragma INLINE
void Period_2x(void)
{
}

#pragma INLINE
void Period_4x(void)
{
}

#pragma INLINE
void Period_8x(void)
{
}

#pragma INLINE
void Period_16x(void)
{
}

#pragma INLINE
void Period_32x(void)
{
}

#pragma INLINE
void Period_64x(void)
{
  LedYellowToggle();
}

#pragma INLINE
void TBM_Enable(void)
{
  TBCR_TACK = 1;  // TBM int. flag cleared
  TBCR_TBON = 1;  // TBM enabled
}
```

```
#pragma INLINE
void TBM_Disable(void)
{
  TBCR_TBON = 0;  // TBM disabled
}

#pragma TRAP_PROC
void TBM_Int(void)
{
  TBCR_TACK = 1;  // TBM int. flag cleared

  // our internal divider
  div++;

  Period_1x();

  if(!(div & 0x01))
    Period_2x();

  if(!(div & 0x03))
    Period_4x();

  if(!(div & 0x07))
    Period_8x();

  if(!(div & 0x0F))
    Period_16x();

  if(!(div & 0x1F))
    Period_32x();

  if(!(div & 0x3F))
    Period_64x();
}
```

wem.c

```
/*****************************************************************
* HEADER_START                                                   *
*                                                                *
*      Name:           WEM.C                                     *
*      Project:        Interconnectivity SRDT                    *
*      Description:    Wireless RS232 demo main file             *
*      Processor:      HC08                                      *
*      Revision:       1.0                                       *
*      Date:           Feb 26 2002                               *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG  *
*      Author:         Michal Hanak                              *
*      Company:        Motorola SPS                              *
*      Security:       General Business                          *
*                                                                *
* ============================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.  *
*                                                                *
* ============================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY          *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR         *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                             *
* ============================================================== *
*
* HEADER_END
*/

#include "board.h"
#include "std.h"
#include "sci.h"
#include "rf2.h"

//////////////////////////////////////////////////////
// globals

#pragma DATA_SEG SHORT MY_ZEROPAGE

SYSFLAGS1 sys1;
SYSFLAGS2 sys2;
BYTE i,j;
BYTE* p;
```

```
#pragma DATA_SEG DEFAULT
// fixed SCI buffers
BYTE sciRxBuff[sciRxSize];
BYTE sciTxBuff[sciTxSize];

//////////////////////////////////////////////////////
// locals

#pragma CONST_SEG ROM_CONST
static const BYTE romeoCfg[3] = { RF_R2CFG_CF | RF_R2CFG_SOE | RF_R2CFG_SR0 |
RF_R2CFG_DME,
                                  RF_HEADER,
                                  RF_R2CFG_SPEED };

static const BYTE romeoCfgOff[3] = { RF_R2CFG_CF | RF_R2CFG_SOE | RF_R2CFG_SR0 /*|
RF_R2CFG_DME*/,
                                  RF_HEADER,
                                  RF_R2CFG_SPEED };

//////////////////////////////////////////////////////
// constant variables

// Error Corrections lookup tables
#ifdef RF_ECC

// systematic code taken from example on
// http://www.tisl.ukans.edu/~paden/Reference/ECC/linear/index.html

static const BYTE eccEncoder[16] =
{
  0x00, //  0 0 0 0 0 0 0 0 0 0 0 0
  0x51, //  1 1 0 1 0 0 0 1 0 0 0 1
  0x72, //  2 1 1 1 0 0 1 0 0 0 1 0
  0x23, //  3 0 1 0 0 0 1 1 0 0 1 1
  0x34, //  4 0 1 1 0 1 0 0 0 1 0 0
  0x65, //  5 1 1 0 0 1 0 1 0 1 0 1
  0x46, //  6 1 0 0 0 1 1 0 0 1 1 0
  0x17, //  7 0 0 1 0 1 1 1 0 1 1 1
  0x68, //  8 1 1 0 1 0 0 0 1 0 0 0
  0x39, //  9 0 1 1 1 0 0 1 1 0 0 1
  0x1A, //  A 0 0 1 1 0 1 0 1 0 1 0
  0x4B, //  B 1 0 0 1 0 1 1 1 0 1 1
  0x5C, //  C 1 0 1 1 1 0 0 1 1 0 0
  0x0D, //  D 0 0 0 1 1 0 1 1 1 0 1
  0x2E, //  E 0 1 0 1 1 1 0 1 1 1 0
  0x7F, //  F 1 1 1 1 1 1 1 1 1 1 1
};

static const BYTE eccDecoder[128] =
{
```

```
  0x0, 0x0, 0x0, 0x3, 0x0, 0xD, 0x6, 0x7,
  0x0, 0xD, 0xA, 0xB, 0xD, 0xD, 0xE, 0xD,
  0x0, 0x1, 0xA, 0x7, 0x4, 0x7, 0x7, 0x7,
  0xA, 0x9, 0xA, 0xA, 0xC, 0xD, 0xA, 0x7,
  0x0, 0x3, 0x3, 0x3, 0x4, 0x5, 0xE, 0x3,
  0x8, 0x9, 0xE, 0x3, 0xE, 0xD, 0xE, 0xE,
  0x4, 0x9, 0x2, 0x3, 0x4, 0x4, 0x4, 0x7,
  0x9, 0x9, 0xA, 0x9, 0x4, 0x9, 0xE, 0xF,
  0x0, 0x1, 0x6, 0xB, 0x6, 0x5, 0x6, 0x6,
  0x8, 0xB, 0xB, 0xB, 0xC, 0xD, 0x6, 0xB,
  0x1, 0x1, 0x2, 0x1, 0xC, 0x1, 0x6, 0x7,
  0xC, 0x1, 0xA, 0xB, 0xC, 0xC, 0xC, 0xF,
  0x8, 0x5, 0x2, 0x3, 0x5, 0x5, 0x6, 0x5,
  0x8, 0x8, 0x8, 0xB, 0x8, 0x5, 0xE, 0xF,
  0x2, 0x1, 0x2, 0x2, 0x4, 0x5, 0x2, 0xF,
  0x8, 0x9, 0x2, 0xF, 0xC, 0xF, 0xF, 0xF,
};
#endif

#pragma CONST_SEG DEFAULT

#pragma DATA_SEG SHORT MY_ZEROPAGE

static BYTE myAddr;
static BYTE destAddr;

static BYTE len;

// iterrupt vars
static BYTE _cs;
static BYTE _i;
static BYTE* _p;

#define RTBUFF_SIZE (sizeof(sciTxBuff)-4)

typedef struct
{
  unsigned int len : 6;
  unsigned int ack : 1;
  unsigned int nack : 1;

} RFPKTFLGS;

typedef struct
{
  BYTE from;
    BYTE to;
    RFPKTFLGS sys;

} RFPKTHDR;
```

```
typedef struct
{
  RFPKTHDR hdr;
    BYTE buff[RTBUFF_SIZE];
    BYTE _floating_cs;

} RFPKT;

// RF transmit packet
RFPKT rtpkt;

static BYTE rtWP;

#pragma DATA_SEG DEFAULT

////////////////////////////////////////////////////////

#pragma TRAP_PROC
void NullHand(void)
{
  return;
}

void SystemInit(void)
{
  PCTL_BCS = 0;     /* & unselect VCO as a source */
  PCTL_PLLON = 0;   /* PLL OFF, 32k as a source now */

  CONFIG2 = 0x01;   /* SCIBDSRC = bus */
  CONFIG1 = CONFIG1DEF;   /* LVI resets disabled, COP ena/disabled */

  PCTL_VPR1 = PCTL_VPR1DEF;   /* E multiplier*/
  PCTL_VPR0 = PCTL_VPR0DEF;   /* E multiplier*/

  PMSH = PMSHDEF;
  PMSL = PMSLDEF;     /* PLL Multi */
  PMRS = PMRSDEF;   /* PLL VCO Lock range*/

  PCTL_PLLON = 1;   /* PLL ON now */
  PBWC_AUTO = 1;    /* AUTO on */

  while(!(PBWC_LOCK)); /* wait until it locks */
  PCTL_BCS = 1;     /* & select VCO as a source */

  // port pins direction
  DDRA = I_DDRA;
  DDRB = I_DDRB;
  DDRC = I_DDRC;
  DDRD = I_DDRD;
  DDRE = I_DDRE;
```

```
  // reset all outputs
  PTA = 0;
  PTB = 0;
  PTC = 0;
  PTD = 0;
  PTE = 0;

  // keyboard
  INTKBSCR_IMASKK = 1; // mask kbd int
  PTAPUE = 0xff;    // all A pins are pulled up
  INTKBSCR_ACKK = 1;  // clear kbd int bit (just in case)

  // TBM
  TBCR_TBR2 = 1;    // set TBM period to 1 ms (1024 Hz)
  TBCR_TBR1 = 0;
  TBCR_TBR0 = 1;

  TBCR_TBON = 1;    // turn TBM on
  TBCR_TBIE = 1;    // enable TBM int

  // SCI setup (see wem.h)
  SCBR = SCI_BAUDRATE;

  SCC1_ENSCI = 1;   // enable SCI
  SCC2_TE = 1;    // enable transmitter
  SCC2_RE = 1;    // enable receiver

#ifdef SCI_CTS_CONTROL
  SCI_CTS = SCI_CTS_OFF;  // no reception until Init
#endif
#ifdef SCI_XONXOFF_CONTROL
  sciTxPriority = 1;
  sciTxPriorityByte = XOFF;
  EnaTxInt();
#endif
#ifdef SCI_TXE_CONTROL
  SCI_TXE = SCI_TXE_OFF;  // disable 485 transmitter
#endif
#ifdef SCI_TXE_CONTROL_RX
  SCI_RXE = SCI_RXE_OFF;  // disable 485 reciver
#endif

    asm CLI;
}

void memset(void* ptr, BYTE set, BYTE len)
{
  while(len--) *((BYTE*)ptr)++ = set;
}

void memcpy(void* dest, void* src, BYTE len)
```

```
{
  while(len--) *((BYTE*)dest)++ = *((BYTE*)src)++;
}

void main(void)
{
  SystemInit();
  RF_Init(romeoCfg);

  // SCI init: no arguments since we use fixed buffs
  SCI_InitTx();
  SCI_InitRx();

  RF_RxStart();   // start RF sniffing

  for (;;)  // never ending main loop
  {

#ifdef COP_ENABLE
    COPCTL = 0xff;      // bump COP (if enabled during compile)
#endif

    if (rfRxLen && !rfRxActive)      // we've got some bytes in rfrx buff. + nothing
else is being rcvd
    {
     if (len = RF_RxBuff_OK())        // check rfrx buffer for sanity & remember the
length in one step
      {
        LedGreenOn();
        LedRedOff();

        SCI_TxBuff(RF_RxBuff+1, len);  // schedule SCI transmission (shall be empty
cause SCI is faster)
        RF_RxStart();
      }
      else /*if (!rfTxActive)*/
      {
        LedGreenOff();
        LedRedOn();

        RF_RxStart();                   // wrong rfrx buff, restore receive (if not
transmitting)
      }
    } // if (rfRxLen && !rfRxActive)

    if (SCI_RxLen() && (sciRxTmout || (SCI_RxLen()>= rxSize-5)) && (rfRxLen == 0)) //
SCI data ready, RF not rxing
    {
      sciRxTmout = 0;
      SciRx_Off();
```

```
      RF_Init(romeoCfgOff);
      TBM_Disable();

      RF_SendBuff(SCI_RxPoll(RF_TxBuff+1, SCI_RxLen())+1);

      while (rfTxActive)
      {/* do nothing & wait for data to be transmitted
          don't let be disturbed by ANY sci int or another int */};

      TBM_Enable();
      RF_Init(romeoCfg);

      SciRx_On();

      LedGreenOff();
      LedRedOff();

      RF_RxStart();                  // start RF RX after TX ready!
    }
  }
}

/*-------------------------------------------------------------------------**
**                                                                         **
** The CRC is based on polynom:                                            **
**                                                                         **
**          16    15   2                                                  **
**         X   + X  + X  + 1                                               **
**                                                                         **
**            See also 4. "A Tutorial on CRC Computations"                 **
**            article in IEEE Micro magazine, August 1988                  **
**-------------------------------------------------------------------------*/

int calc_crc(char *buf, unsigned char n)
{

#pragma CONST_SEG ROM_CONST
  static const int  crc_table[] = {
    0x0000, 0xc0c1, 0xc181, 0x0140, 0xc301, 0x03c0, 0x0280, 0xc241,
    0xc601, 0x06c0, 0x0780, 0xc741, 0x0500, 0xc5c1, 0xc481, 0x0440,
    0xcc01, 0x0cc0, 0x0d80, 0xcd41, 0x0f00, 0xcfc1, 0xce81, 0x0e40,
    0x0a00, 0xcac1, 0xcb81, 0x0b40, 0xc901, 0x09c0, 0x0880, 0xc841,
    0xd801, 0x18c0, 0x1980, 0xd941, 0x1b00, 0xdbc1, 0xda81, 0x1a40,
    0x1e00, 0xdec1, 0xdf81, 0x1f40, 0xdd01, 0x1dc0, 0x1c80, 0xdc41,
    0x1400, 0xd4c1, 0xd581, 0x1540, 0xd701, 0x17c0, 0x1680, 0xd641,
    0xd201, 0x12c0, 0x1380, 0xd341, 0x1100, 0xd1c1, 0xd081, 0x1040,
    0xf001, 0x30c0, 0x3180, 0xf141, 0x3300, 0xf3c1, 0xf281, 0x3240,
    0x3600, 0xf6c1, 0xf781, 0x3740, 0xf501, 0x35c0, 0x3480, 0xf441,
    0x3c00, 0xfcc1, 0xfd81, 0x3d40, 0xff01, 0x3fc0, 0x3e80, 0xfe41,
    0xfa01, 0x3ac0, 0x3b80, 0xfb41, 0x3900, 0xf9c1, 0xf881, 0x3840,
    0x2800, 0xe8c1, 0xe981, 0x2940, 0xeb01, 0x2bc0, 0x2a80, 0xea41,
```

```
    0xee01, 0x2ec0, 0x2f80, 0xef41, 0x2d00, 0xedc1, 0xec81, 0x2c40,
    0xe401, 0x24c0, 0x2580, 0xe541, 0x2700, 0xe7c1, 0xe681, 0x2640,
    0x2200, 0xe2c1, 0xe381, 0x2340, 0xe101, 0x21c0, 0x2080, 0xe041,
    0xa001, 0x60c0, 0x6180, 0xa141, 0x6300, 0xa3c1, 0xa281, 0x6240,
    0x6600, 0xa6c1, 0xa781, 0x6740, 0xa501, 0x65c0, 0x6480, 0xa441,
    0x6c00, 0xacc1, 0xad81, 0x6d40, 0xaf01, 0x6fc0, 0x6e80, 0xae41,
    0xaa01, 0x6ac0, 0x6b80, 0xab41, 0x6900, 0xa9c1, 0xa881, 0x6840,
    0x7800, 0xb8c1, 0xb981, 0x7940, 0xbb01, 0x7bc0, 0x7a80, 0xba41,
    0xbe01, 0x7ec0, 0x7f80, 0xbf41, 0x7d00, 0xbdc1, 0xbc81, 0x7c40,
    0xb401, 0x74c0, 0x7580, 0xb541, 0x7700, 0xb7c1, 0xb681, 0x7640,
    0x7200, 0xb2c1, 0xb381, 0x7340, 0xb101, 0x71c0, 0x7080, 0xb041,
    0x5000, 0x90c1, 0x9181, 0x5140, 0x9301, 0x53c0, 0x5280, 0x9241,
    0x9601, 0x56c0, 0x5780, 0x9741, 0x5500, 0x95c1, 0x9481, 0x5440,
    0x9c01, 0x5cc0, 0x5d80, 0x9d41, 0x5f00, 0x9fc1, 0x9e81, 0x5e40,
    0x5a00, 0x9ac1, 0x9b81, 0x5b40, 0x9901, 0x59c0, 0x5880, 0x9841,
    0x8801, 0x48c0, 0x4980, 0x8941, 0x4b00, 0x8bc1, 0x8a81, 0x4a40,
    0x4e00, 0x8ec1, 0x8f81, 0x4f40, 0x8d01, 0x4dc0, 0x4c80, 0x8c41,
    0x4400, 0x84c1, 0x8581, 0x4540, 0x8701, 0x47c0, 0x4680, 0x8641,
    0x8201, 0x42c0, 0x4380, 0x8341, 0x4100, 0x81c1, 0x8081, 0x4040
  };
#pragma CONST_SEG DEFAULT

  int crc;

  crc = 0;
  while (n--)
    crc = ((crc >> 8) & 0xff) ^ crc_table[(crc ^ *buf++) & 0xff];
  return crc;
}

/*---------------------------------------------------------------------------*/
BYTE RF_RxBuff_OK(void)
{
  if ((rfRxLen < 4)                              // not long enough (min. 4 bytes (LEN
DATA CRC CRC))
  ||  (RF_RxBuff[0] < 4)                           // too short len info
  ||  (RF_RxBuff[0] > RFRXBUFFSIZE)                // too long len info
  ||  (calc_crc(RF_RxBuff, RF_RxBuff[0]) != 0))  // wrong CRC
    return FALSE;
  else
   return RF_RxBuff[0]-3;             // return actual data length (minus LEN, minus
2x CRC)
}

// RF interrupt handlers
void RF_TxChar_Int(void)
{
  if (rfTxIdx < rfTxLen)
#ifdef RF_ECC
  {
    if (eccNibble == 0)
```

```
    {
      eccNibble = 1;
      RF_TxChar(eccEncoder[(RF_TxBuff[rfTxIdx]>>4) & 0x0F]);  // high nibble
    }
    else
    {
      eccNibble = 0;
      RF_TxChar(eccEncoder[RF_TxBuff[rfTxIdx++] & 0x0F]);     // low nibble
    };
  }
#else
  {
    RF_TxChar(RF_TxBuff[rfTxIdx++]);
  }
#endif
  else
    RF_TxFinish();
}

void RF_TxFinish_Int(void)
{
  return ;
}

void RF_RxChar_Int(BYTE ch)
{
  if (rfRxIdx >= RFRXBUFFSIZE)    // outside buffer
  {
   rfRxLen = rfRxIdx;         // copy length even if it is very probably wrong (long)
  }
  else
  {
#ifdef RF_ECC
    if (eccNibble == 0)
    {
      eccNibble = 1;
     RF_RxBuff[rfRxIdx] = eccDecoder[ch & 0x7f] << 4;// feed the buffer - high nibble
    }
    else
    {
      eccNibble = 0;
      RF_RxBuff[rfRxIdx++] |= eccDecoder[ch & 0x7f]; // feed the buffer - low nibble
    };
#else
    RF_RxBuff[rfRxIdx++] = ch; // feed the buffer
#endif
  }

  RF_TimerReset();            // restart RX timeout tmr
  RF_RxTimeoutOn();           // & enable if not yet (first byte usually)
}
```

wem232_ii.prm

```
NAMES
       start08.o
       ansi.lib
END

VECTOR 0   _Startup
VECTOR 1    NullHand
VECTOR 2    NullHand    /* Irq1_Int */
VECTOR 3    NullHand    /* PLL_Int */
VECTOR 4    NullHand    /* Tim1Ch0_Int */
VECTOR 5    NullHand    /* Tim1Ch1_Int */
VECTOR 6    NullHand    /* Tim1Ov_Int  */
VECTOR 7    RFTimerTxd_Int
VECTOR 8    RFTimerTmout_Int
VECTOR 9    RFTimerOv_Int
VECTOR 10   SpiRx_Int
VECTOR 11   NullHand    /* SpiTx_Int */
VECTOR 12   NullHand    /* SciErr_Int */
VECTOR 13   SciRx_Int
VECTOR 14   SciTx_Int
VECTOR 15   NullHand    /* Keyboard_Int */
VECTOR 16   NullHand    /* ADC_Int */
VECTOR 17   TBM_Int

SECTIONS
    AREA_FLASH = READ_ONLY  0x8000 TO 0xFDFF;
    AREA_ZPRAM = READ_WRITE 0x0040 TO 0x00BF;
    AREA_HIRAM = READ_WRITE 0x0100 TO 0x023F;
    AREA_STACK = NO_INIT    0x00C0 TO 0x00FF;

PLACEMENT
    MY_ZEROPAGE INTO AREA_ZPRAM;
    DEFAULT_RAM INTO AREA_HIRAM;
    SSTACK INTO AREA_STACK;

    DEFAULT_ROM, ROM_VAR INTO AREA_FLASH;
END
```

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors