Freescale Semiconductor
Application Note

Document Number: AN4115
Rev. 0, 07/2010

IrDA Driver and SD Card File System on MM/JE Flexis Families

by: Carlos Casillas and Jose Ruiz
RTAC Americas
Guadalajara, Mexico

1 Introduction

This application note discusses a physical layer driver used to establish IrDA communication on Freescale devices in the MM/JE Flexis families, using the Carrier Modulator Timer (CMT) as transmitter and a Timer/Pulse-Width Modulator (TPM) as receptor.

It also explains a driver to handle a Secure Digital (SD) card through a Serial Peripheral Interface (SPI) module, plus a file system allowing the microcontroller to open, edit, save, and close files stored in the SD card.

Both drivers are available for download in a zip file associated with this application note.

The two drivers are integrated in a demonstration application that reads a text file stored in an SD card and sends the content modulated by IrDA using the CMT. It then receives this data with a TPM, and sends it to a PC through an RS-232 serial port using a Serial Communication Interface (SCI) module and a terminal application to show the text. Also, the text written in the

Contents

freescale semiconductor

terminal application on the PC is sent to the microcontroller, received by the SCI, and added to the file stored in the SD card. Figure 1 shows a block diagram of the application.

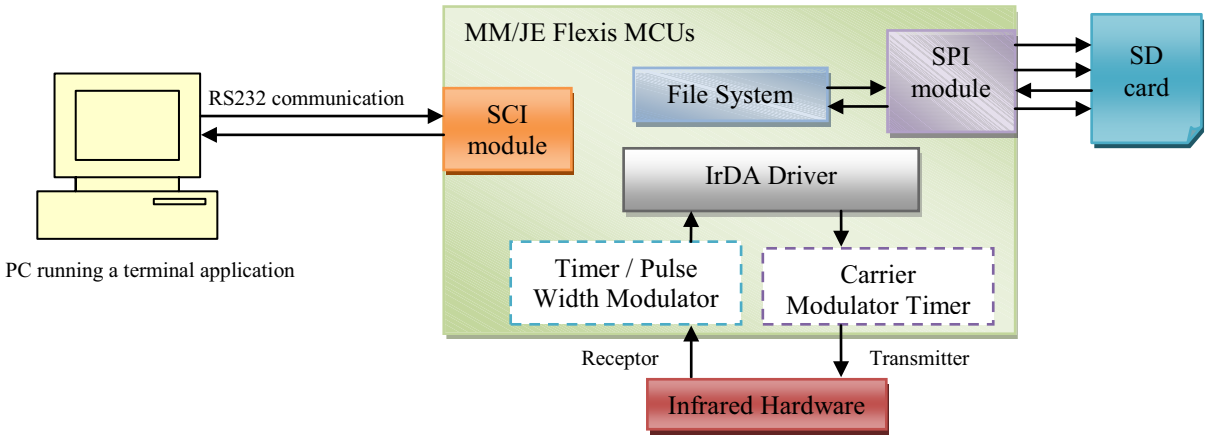

Figure 1. General block diagram

2 Using CMT to transmit IrDA modulated signals

This section shows the block diagram of the CMT, and explains the code required to configure the module for this application.

The modulation at 3/16 of bit duration implemented in this application is based on the IrDA physical layer specification.

NOTE

This application note just provides the driver for the physical layer of the IrDA specification, and does not include the complete IrDA protocol. If you need more information about the IrDA protocol and the physical layer specification, please visit the IrDA webpage at www.irda.org.

2.1 CMT module initialization

The module block diagram is shown in Figure 2:

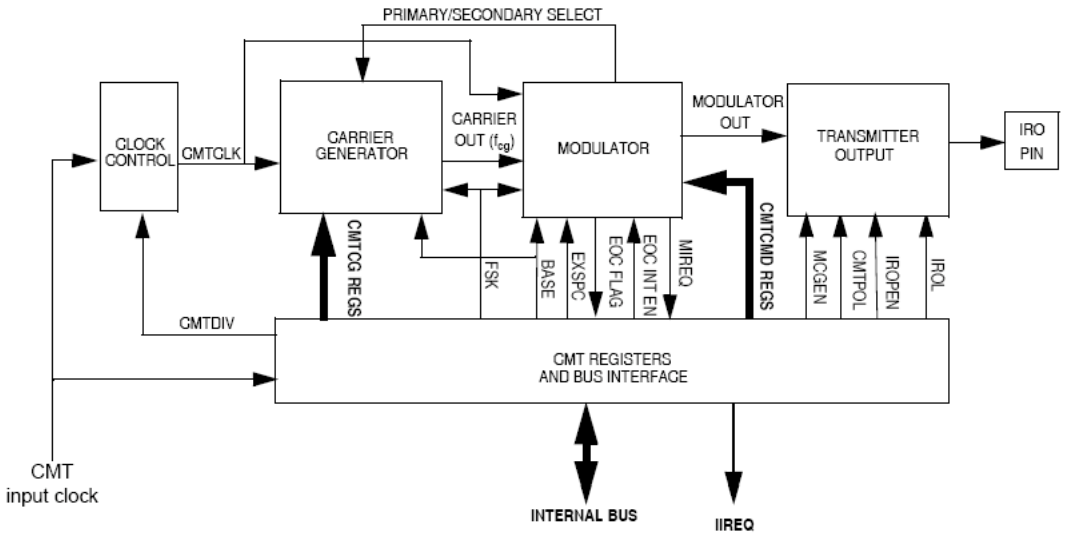

Figure 2. Carrier Modulator Timer block diagram

Here are the configuration options included in the function *void vfnCMT_init (void)* in IrDA.c file.

1. The first step is configuring the clock divisor, which can divide the input clock by 1, 2, 4, or 8. It is set in a 2-bit field named CMTDIV in the register CMTMSC.

   This divisor is defined in the file IrDA.h, depending on the used baud rate.

NOTE

Coldfire V1 devices also have another clock divisor (bitfield CMT_CLK_SEL in register SOPT2), which allows the clock to be divided by 3, but it is not being used in this application (clock is divided by 1 because its reset status is zero).

2. Next, configurations must be applied in the CMT Modulator Status and Control Register (CMTMSC) to enable operation in time mode:
   a) The extended space mode is disabled by clearing the bit EXSPC.
   b) The base band mode is disabled by clearing the bit BASE.
   c) The frequency-shift keying (FSK) mode is disabled by clearing the bit FSK.
   d) The end of cycle interrupt is disabled by clearing the bit EOCIE. It will be enabled just before a transmission and disabled when the transmission finishes.
3. Next, these configurations are part of the IRO pin logic:
   a) The polarity of the IRO pin is set to active-high by setting the bit CMTPOL of the register CMTOC.
   b) The IRO pin is enabled by setting the bit IROPEN of the register CMTOC.
   c) The double current drain on IRO pin is enabled by setting the bit CMT_PAD on the System Options 3 Register (SOPT3). It helps to increase the amount of current that the IRO pin drives, allowing more luminescence to an IR diode connected directly to the IRO pin by just adding a series resistor.

4. Because the CMT will operate in time mode, the registers that contain the number of input clocks required to generate the carrier high and low time periods should be adjusted. For this application, the pulse time should be 3/16 of the bit duration, and the space time should be the remaining 13/16 of bit duration. These values are defined in the file IrDA.h, depending on the bus clock and baud rate.
5. Finally, the registers that control the mark and space periods of the modulator are adjusted. The registers CMTCMD1 and CMTCMD2 are set before a transmission, and the registers CMTCMD3 and CMTCMD4 are adjusted to zero, because the driver uses only the mark period, thus allowing continuous pulse generation. Also, the counters are disabled by clearing the bit MCGEN of the register CMTMSC, and will be enabled just before a transmission.

2.2 Transmission function explanation

The function *void vfnIrDA_transmission (UINT8 u8data)* in the IrDA.c file sends one byte through the CMT. Each code line is explained below:

1. Check the status of the IrDA communication (variable *u8irda_tx_status*). This function will wait if the status is “sending” to avoid data overlap, or if the status is “receiving” to avoid light interference.
2. Variable *u8irda_tx_counter* is set to zero because it will count which bit of the byte is the next to be transmitted.
3. The variable *u8irda_tx_status* changes to IRDA_SENDING status.
4. The modulator gate is enabled, allowing the pulses to be sent to the IRO pin.
5. The modulator period is set to half of bit duration, and the modulator and carrier generator are enabled. It is because the start bit is a zero (a pulse with duration of 3/16 of bit period) generated by the carrier output (blue signal on Figure 3) and the modulator gate is open (green signal on Figure 3) for 1/2 of the duration of a bit period.
6. The period registers are then immediately loaded with the value of a complete bit duration time, because the CMT will load these values at the beginning of the next period. It is used to open or close the modulation gate in the middle of the bit period, avoiding false pulses. This task is performed in the CMT interrupt. Figure 3 shows the behavior of this task.
7. Finally, the end-of-cycle interrupt is enabled, and the function waits while the status is IRDA_SENDING. This status is changed in the interrupt after the transmission of the last bit is complete. Then, the variable u16stop (loaded at the beginning of the function with the number of counts of pulse time plus space time) is decremented until it reaches zero, generating the stop bit (logical one, which is a space).

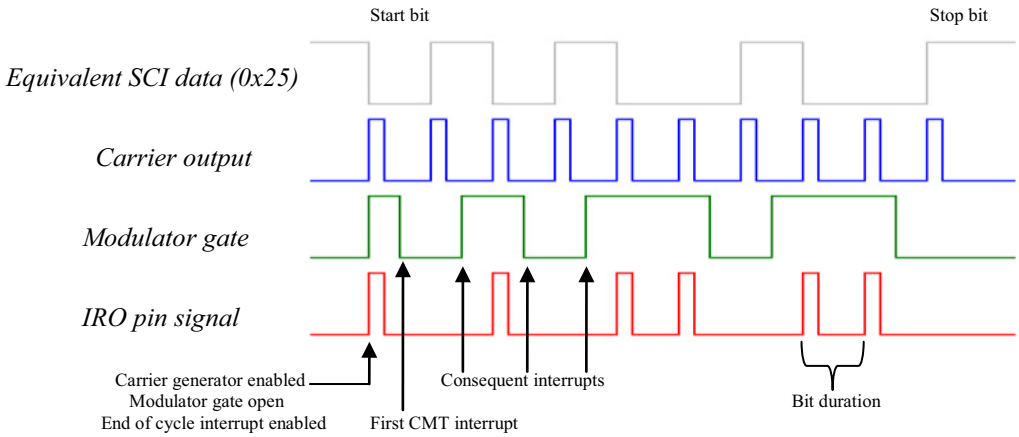

Figure 3. Modulated signal explanation

The function *void IrDA_Message (UINT8 *string)* in the IrDA.c file sends a string of bytes through the CMT, calling the function *void vfnIrDA_transmission (UINT8 u8data)* at each byte transmission.

2.3 CMT interrupt

There is only one interrupt associated with the CMT. It is the end-of-cycle interrupt, and it is set by the end of cycle flag. The actions performed by the interrupt are:

1. Clear the CMT interrupt flag, then read the CMTMSC and CMTCMD2 registers.
2. If variable *u8irda_tx_counter* is less than eight, the modulator gate will be enabled or disabled depending on the data to be transmitted, and then the counter variable is incremented.
3. If variable *u8irda_tx_counter* is equal to eight it means that the last bit was transmitted, so the modulator gate, the end of cycle interrupt, the modulator, and the carrier generator are disabled. Finally, the variable *u8irda_tx_status* is set with the status IRDA_TX_IDLE.

3 Using TPM to receive IrDA modulated signals

This section explains the code required to configure the TPM for this application.

3.1 Using Analog Comparator for signal conditioning

The ACMP module can be configured to connect the output of the analog comparator to the TPM1 input capture channel 0 by setting the bit ACIC in the SOPT2 register. The block diagram of these connections is shown in Figure 4.

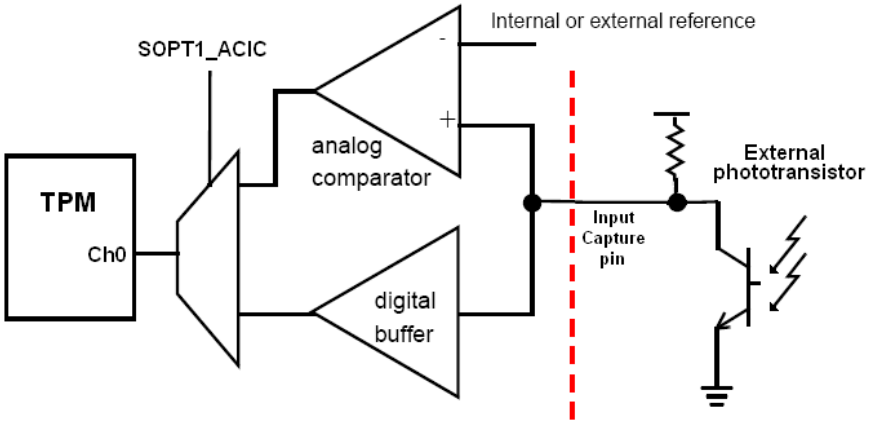

Figure 4. Analog Comparator internal connections

These are the configuration options included in the function *UINT8 u8ACMP_rx_init (UINT8 acmp_level)* in the IrDA.c file.

1. Disables the option of routing the analog comparator output to pin ACMPO.
2. Connects the comparator positive input to internal resistor ladder reference voltage.
3. Connects the comparator negative input to external pin.
4. Selects V_{DD} as reference voltage for resistor ladder.
5. Enables analog comparator programmable mode.
6. Selects the voltage level of the resistor ladder. This is a 5-bit value and each unit adds V_{DD} /32 volts.
7. Finally, the analog comparator is enabled.

3.2 TPM module initialization

Here are the configuration options included in the function *void vfnTPM1_rx_init (void)* in the IrDA.c file.

1. Enables the option of routing the analog comparator output to TPM1 channel 0.
2. Disable the TPM1 overflow interrupt.
3. Loads the value of the register TPM1MOD with the definitions declared as TMP_PERIOD. This definition is used to match the number of counts used in the TPM to the number of counts of the CMT, according to the baud rate.
4. Then TPM1 is configured in input capture mode at rising edge, and the channel 0 interrupt is enabled.
5. Next, the clock divisor is selected, according to the definition DIVISOR which takes different values depending on the baud rate.
6. Finally, the clock source is selected, setting the bus clock as clock source.

3.3 TPM interrupts

The TPM has two interrupts:

- The end-of-cycle interrupt that occurs if the bit time ends because a pulse has not been detected.
- The Channel 0 interrupt that occurs when a rising edge is detected. This interrupt is the first that is detected because a transmitted byte has a start bit that is a pulse.

Figure 5 shows an example of a received modulated byte and the occurrence of each interrupt. Green arrows indicate Channel 0 interrupts; purple arrows indicate end-of-cycle interrupts.

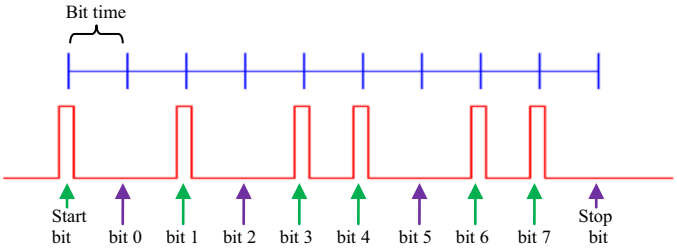

Figure 5. Occurrence of TPM interrupts

Each interrupt is explained below.

3.3.1 End-of-cycle interrupt

Here are the tasks performed by the interrupt *void interrupt VectorNumber_Vtpm1ovf TPM1_rx_space (void)* in the IrDA.c file.

1. Clears interrupt flag.
2. If *u8irda_rx_counter* (variable that counts the bit number received) is equal to nine, it will be cleared to zero, the reception status will be set as idle, and the timer overflow interrupt enable mask will be disabled.
3. If *u8irda_rx_counter* is any other value, a logical one rotated the number of positions equal to the current bit number will be summed to the reception variable.
4. If *u8irda_rx_counter* is not zero, it will be incremented by one.

3.3.2 Channel 0 interrupt

Here are the tasks performed by the interrupt *void interrupt VectorNumber_Vtpm1ch0 TPM1_rx_pulse (void)* in the IrDA.c file.

1. Clears interrupt flag and disables timer count and interrupt.
2. If *u8irda_rx_counter* is zero (first pulse reception), the reception status will be set as receiving, and the reception variable will be cleared (set to zero).
3. Variable *u8irda_rx_counter* is incremented by one.
4. The counter registers of the TPM are cleared and the bus clock is again selected as the clock source for the TPM.
5. Finally, the timer overflow interrupt enable is enabled again.

4 SD card file system

This section explains the configuration of the SPI module to establish communication with an SD card. Also includes descriptions of each function included in the file system. For more information about accessing an SD card through an SPI module, please refer to sections 2.3 and 3.4 of Freescale document DRM104, *SD Card Reader Using the M9S08JM60 Series,* Rev. 0, July 2008, available on the Freescale website.

4.1 SPI module configuration

The SPI module must be configured in this way to establish communication with an SD card:

1. Correspondent 8-bit SPI module must be initialized.
2. The SPI slave select pin must be defined in the file SPI.h.
3. The SPI clock must be initialized at 375 kHz for compatibility with Multimedia Card (MMC) memories.
4. In this driver the interrupts are not being used, because the system waits for transmission or reception termination.
5. The SPI module is configured to operate as master.
6. The clock polarity is selected as active-high, and the clock phase is selected as the first edge occurs in the middle of the first cycle of a data transfer.
7. The SPI module is enabled.

4.2 File system explanation

This section explains each function implemented on the file system.

4.2.1 Function void FAT_Read_Master_Block(void)

Description:

Fill file system general parameters.

4.2.2 Function void FAT_LS (void)

Description:

Send to terminal (serial) the content of the root directory (filename and extension).

4.2.3 Function void FAT_FileWrite(UINT8*, UINT32)

Description:

This function writes content to the current file. To enable writing and modifying, it is necessary to open the file first using the FAT_FileOpen function.

Input parameters:

- UINT8* — Pointer to file data
- UINT32 — Data buffer size

4.2.4 Function UINT16 Fat_FileRead(UINT8*)

Description:

This function reads the file content and stores it into a user buffer. It reads a maximum 512B from the memory each time (depending on the file size), so if the file is bigger than 512B the user needs to call the function as many times as necessary until the function returns 0.

Input parameters:

UINT8* — Pointer to store read data

Output parameters:

UINT16 — Remaining bytes to read from memory

4.2.5 Function void FAT_FileClose (void)

Description:

This function writes the fat and rot directory entries. This function is only for modify and write options for the FAT_FileOpen function, and must be called after completing the write procedure for each file.

5 Running the application

This section includes information about the required material, jumper settings, hardware assembly, code download, configurable parameters, and how to execute the application.

5.1 Required material

To implement this application, the following material is required:

- Tower Functional and Dummy elevators (TWR-ELEV)
- Tower Serial Module (TWR-SER)
- Tower Memory Module (TWR-MEM)
- TWR-S08MM128, TWR-S08JE128, TWR-MCF51MM256, or TWR-MCF51JE256 board (Whichever one is used, it will be referred to as the MCU board in the following sections of this application note.)
- One A/Mini-B USB cable
- One male/female RS-232 cable
- MMC or SD memory card that supports an SPI interface
- PC with CodeWarrior for Microcontrollers v6.3 with MM or JE service packs installed

5.2 Tower module jumper settings

For correct functionality of the application, the following jumper settings must be applied on each board.

5.2.1 Jumper settings for MCU board module

The required jumper configuration on the MCU board module to run the application is:

- J4 — open
- J9 — populated
- J11 — populated
- J12 — open
- J24 — open
- J25 — 5 & 6
- J26 — 2 & 3

5.2.2 Jumper settings for serial module

The required jumper configuration on the serial module:

- J16 — 1 & 2

5.2.3 Jumper settings for memory module

The required jumper configuration on the memory module:

- J1 — 1 & 2
- J2 — open
- J3 — 1 & 2
- J12 — 3 & 4
- J13 — open

5.3 Hardware assembly

These steps should be followed to assemble the hardware:

1. Make sure that all the boards have the jumper configurations already mentioned.
2. Connect serial module, memory module, and MCU module boards to the functional elevator. Make sure that they are connected from the side marked as PRIMARY.
3. Connect the dummy elevator to all boards. Make sure that the sides of the boards are marked as SECONDARY.
4. Connect the USB cable to the USB port of the MCU board (this is the open source BDM).
5. Connect the RS-232 cable to the serial module.
6. Insert an SD or MMC card into the SD slot of the memory module.
7. Connect USB and RS-232 cables to the PC.

An example of the hardware after it has been assembled is shown in Figure 6.

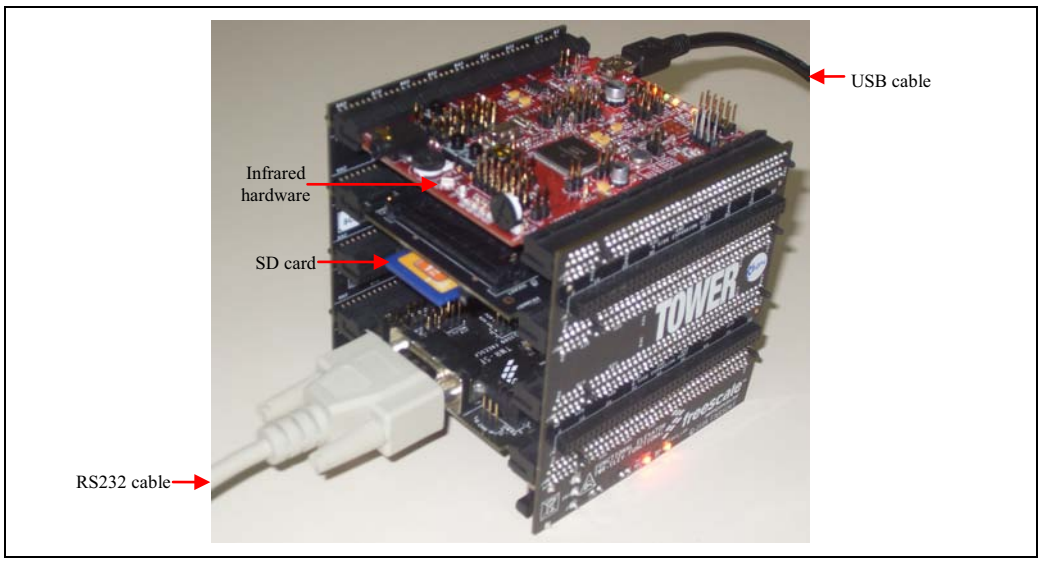

Figure 6. Assembled hardware using MCF51JE256 device

5.4 Code download

To download the code to the devices:

1. Open CodeWarrior for Microcontrollers v6.3.
2. Depending on the device that is being used, open one of these projects:
   - IrDA_and_SD_S08MM.mcp
   - IrDA_and_SD_S08JE.mcp
   - IrDA_and_SD_MCF51MM.mcp
   - or IrDA_and_SD_MCF51JE.mcp
3. Make sure that the selected target is “HCS08 FSL Open Source BDM” or “CFV1 FSL Open Source BDM”, and click on the Debug button.

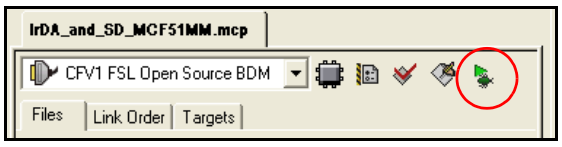

Figure 7. Code Warrior Project Debug button

4. If one of the following warnings appears (depending on the device), select OK.

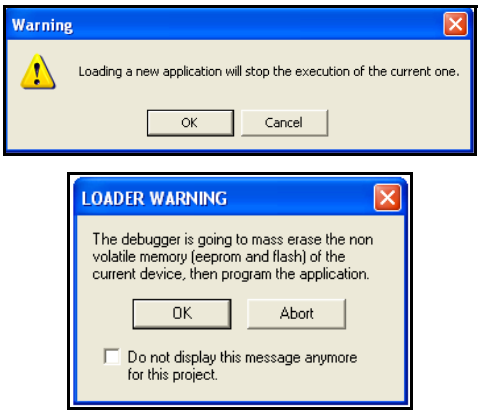

Figure 8. Warning windows for S08 and V1 devices

5. When the device has been programmed, click on the Start/Continue button and close the window.

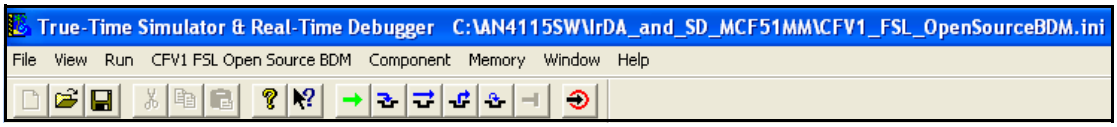

Figure 9. True-Time Simulator & Real-Time Debugger window

5.5 Configurable parameters

The following parameters, defined in the file SDbyIrDA.h, can be modified depending on the application requirements:

```
/********** DEFINES SCI USED PORT, BAUDRATE AND COMPARATOR REFERENCE ************/
#define USED_PORT       1       /* Selects the port used for communication */
#define BAUDRATE        38400   /* Use 9600, 19200 or 38400 in this application */
#define REFERENCE_LEVEL 20      /* Reference voltage used by the comparator */
```

The parameter USED_PORT defines which SCI module will be used (1 or 2).

The parameter BAUDRATE indicates the baud rate that will be used. In this application the defined baud rate will apply for the SCI ports and for IrDA communication, so the only supported baud rates are 9600, 19200, and 38400 bps. More baud rates could be implemented, but the user should check the clock divisors and time values for specific bus clocks.

The parameter REFERENCE_LEVEL indicates the reference voltage used by the analog comparator. It must be in the range of 0–31, because each unit is 1/32 of V_{DD}.

5.6 Executing the application

To execute the application:

1. Open a terminal application on the PC and configure it to match with SCI module initialization performed by the functions *u8SCI_init* and *u8IR_TX_init* in the file main.c, plus the baud rate defined in file SDbyIrDA.h.
2. Press the RESET button on the MCU board. The communication will start.
3. Press SW2 and SW3 to write and read text from the file stored in the SD card.

An example of the application running is shown in Figure 10 and Figure 11.

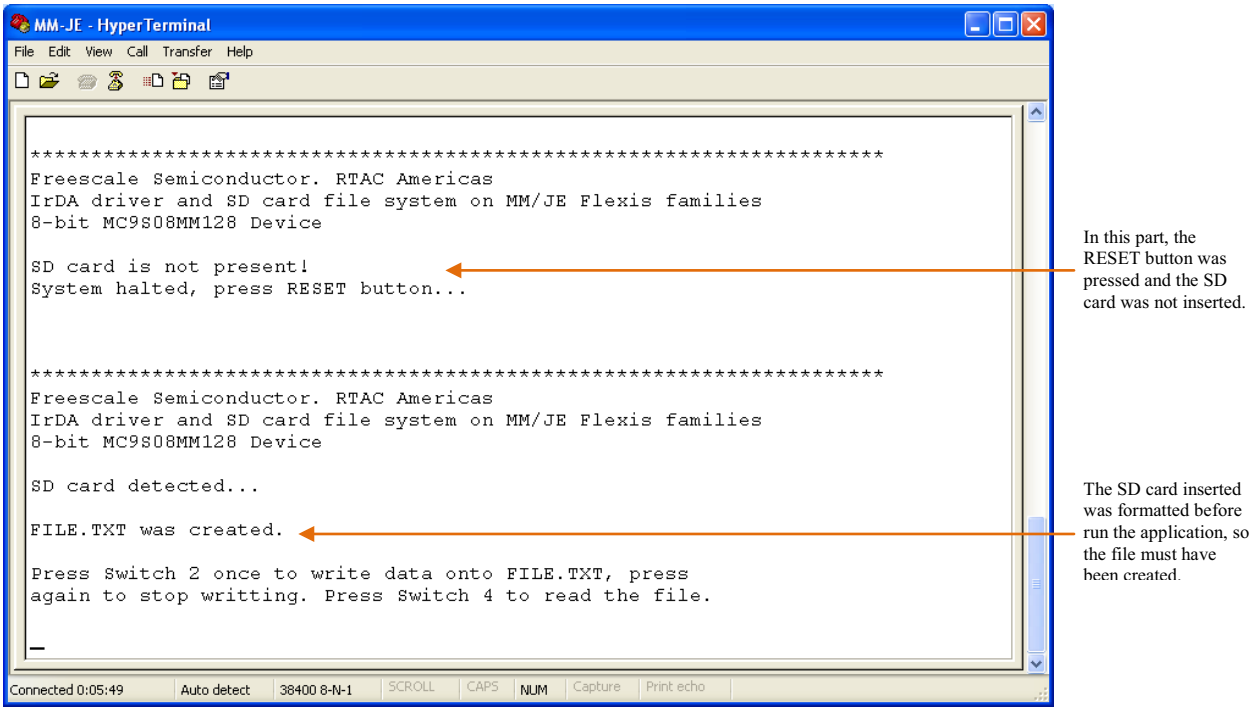

Figure 10. Application running — part 1

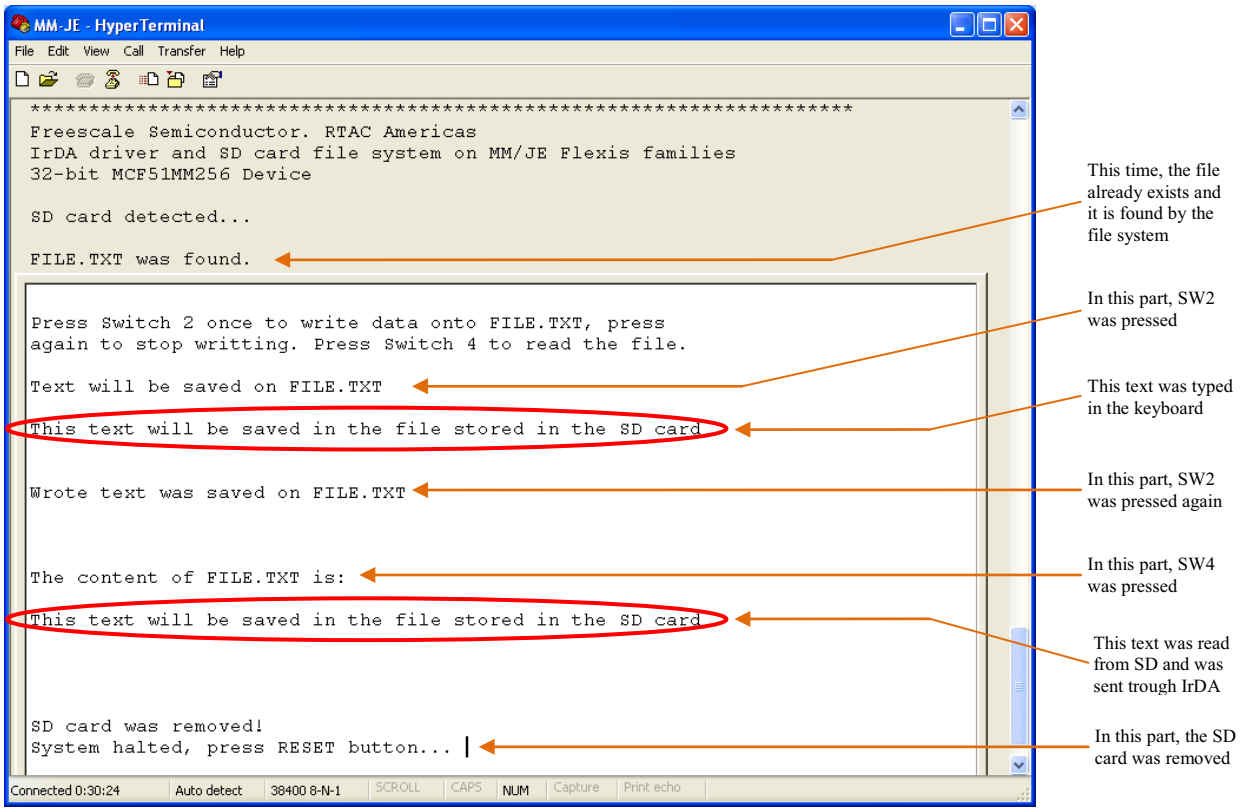

Figure 11. Application running — part 2

6 Conclusions and references

This application note showed the capabilities of the CMT and TPM modules of Freescale's MM/JE Flexis families to implement IrDA communication, and also the ability to handle an SD card and a file system to read and write data through the SPI interface.

Software for this application note is developed on Code Warrior v.6.3, and can be found on the Freescale website as AN4115SW.

If you want to learn more about the MC9S08MM, MC9S08JE, MCF51MM, or MCF51JE devices, please visit the Freescale website, and explore the product summary, fact sheet, data sheet, and other documentation.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4115
Rev. 0
07/2010

freescale™
semiconductor