Freescale Semiconductor
Application Note

Document Number: AN3938
Rev. 0, 10/2009

Using the MCF51EM Family for Infrared Communication

by: Carlos Casillas and Luis Puebla
RTAC Americas
Guadalajara, Mexico

1 Introduction

This application note shows the implementation of an infrared communication system composed of hardware and software. The hardware takes advantage of specific features of Freescale's MCF51EM256, while the software protocol is based on the command structure of protocol C12.18 for metering applications.

This application note is divided into several sections:

- Section 2, "Basic information," provides background information about infrared communication and hardware modules present in the MCF51EM256 that are used in this kind of communication. Also, the software projects included in this application note are listed.
- Section 3, "Hardware," shows characteristics of the MCF51EM256 and the advantages of using them to implement infrared communication.
- Section 4, "Software," includes software projects and shows flowcharts and customization options.

Contents

freescale semiconductor

- Section 5, “Hardware tests,” shows several tests done on the DEMOEM board.
- Finally, Section 6, “Considerations and references,” and Section 7, “Conclusion,” include some considerations and concluding information for the entire application note.

2 Basic information

Infrared is the basis for a type of optical communication that allows complete isolation between electric or electronic devices, because the communication channel is only light and has no electrical connections. Using it for communication in metering applications is valuable because doing so avoids the handling of high voltages, which can be dangerous for people or for other devices.

The features that take advantage of the specific hardware configuration inside the MCF51EM microcontroller unit (MCU) are these:

- Output driving

  Transmission pins can drive up to 50 mA (SCI TX1 and SCI TX2 pins), allowing an IR diode to be connected just by adding a series resistor.
- Signal conditioning

  Two reception pins (RX) can be internally connected to an analog comparator (ACMP). This is useful for conditioning the infrared analog signal to a digital signal.
- Modulation

  Two serial communication interface (SCI) transmission pins (TX) are capable of being internally modulated through the timer output.
- Dual flash array

  The MCF51EM has two flash memory arrays, allowing erasing and writing one of them while code is executing from the other.

Software for this application note includes three different projects:

- IR/RS-232 bridge

  Provides a communication bridge between a standard RS-232 PC serial port and an IR port. This project is used as a tool to verify the functionality of the infrared communication. It is implemented in DEMOEM.
- Protocol

  Offers an example of communication using basic commands to read and write data tables, based on the ANSI C12.18. It can be customized to be used for a specific hardware/application configuration.
- IR demo software

  This platform provides an LCD, user interface, and task manager, customized to test modules in the MCF51EM that are associated with IR (SCI, ACMP, TPM, GPIOs).

3 Hardware

The MCF51EM family are system-on-chip (SoC) devices that are based on the 32-bit ColdFire V1 core and have these hardware key features for IR communication:

- Drive strength for TX1 and TX2 pins can be enabled to drive twice the current of any other pins, up to a maximum of 50 mA, without an external driver.
- SCI1 and SCI2 TX can be modulated by internal Timers. This allows an AND operation of the SCI TX signal with TPMCH0, TMPCH1, MTIM2, or MTIM3 outputs.
- SCI1 and SCI2 Reception can be routed through the ACMP.
- ACMP output is available via an external pin to add an external resistor (R15), allowing the ACMP to provide positive feedback (hysteresis).

The hardware provided by the MCF51EM (on-chip components) and DEMOEM (on-board components) have these advantages (see Figure 1):

- The initialization requires only a few lines of code.
- The hardware does the task by itself (data transmission, modulation, signal conditioning using ACMP), so the processor can perform other tasks in parallel or go to low-power mode.
- Timers provide an accurate modulation of the transmission signal.
- The on-chip analog comparator at the receiver avoids the need of using an external comparator to condition the signal.

Figure 1 shows internal/external connections associated with IR communication that are implemented on the MCF51EM and DEMOEM.

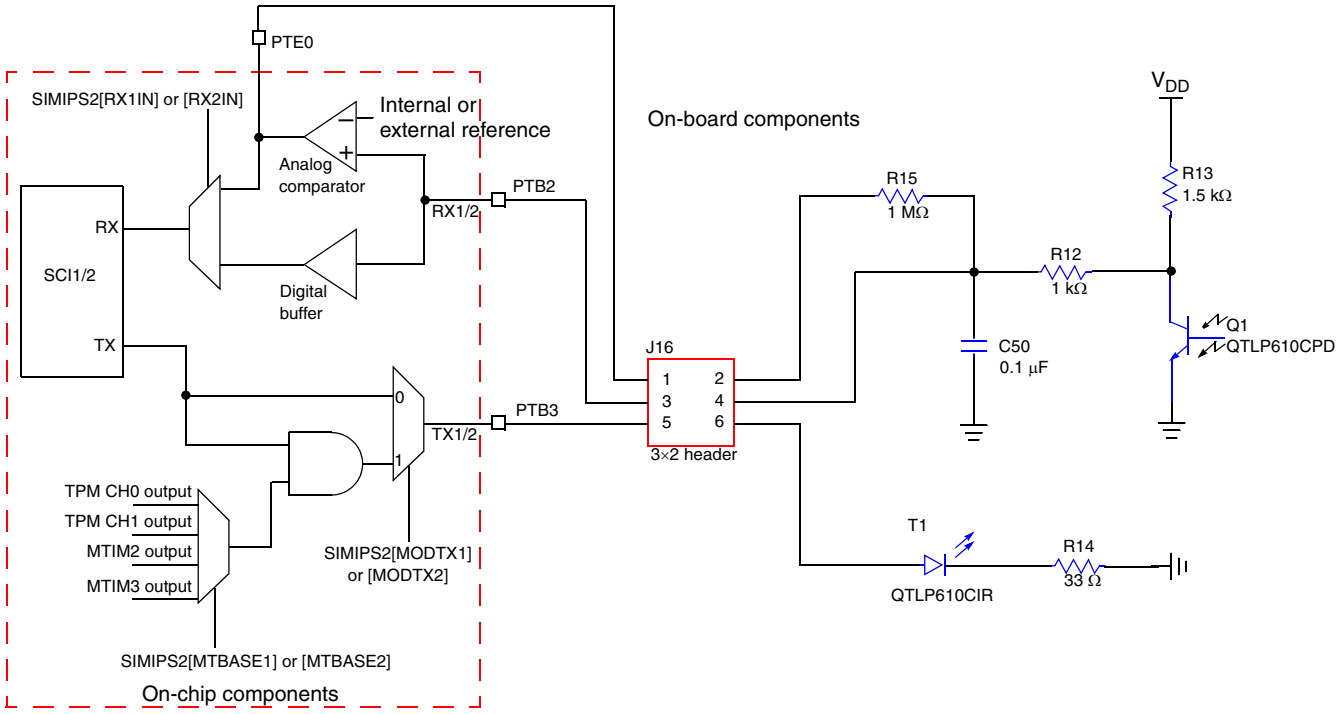


Figure 1. Hardware on MCF51EM256 and DEMOEM

Figure 2 shows the components on the DEMOEM board used for infrared communication:

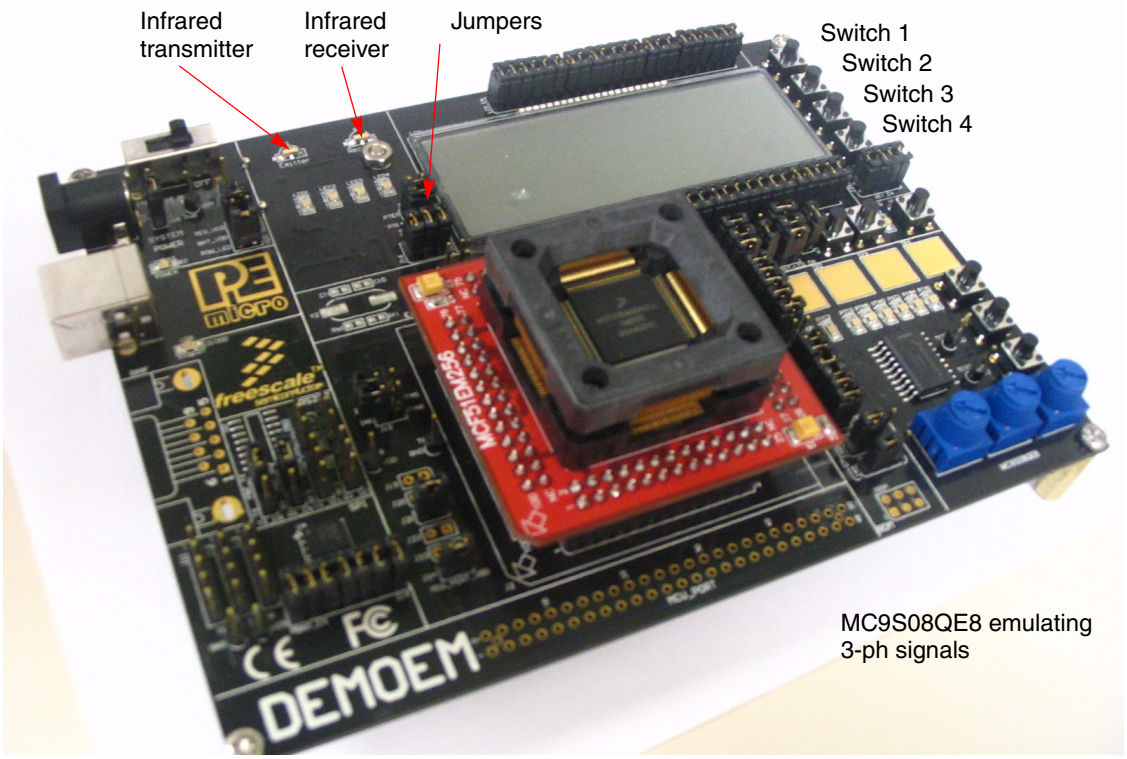


Figure 2. Hardware dedicated to infrared on DEMOEM

4 Software

The software is divided into three separate projects:

- Infrared bridge
- Protocol implementation
- Infrared demo software

Each of them is explained here.

4.1 BridgeIR project

This project provides a tool to communicate between a wired serial interface and an infrared communication port, using the DEMOEM board.

4.1.1 Software Architecture

The architecture of this project is based in the use of interruptions. The main() function initializes the hardware modules and enables the SCI interrupts. When a character is received by the wired port:

1. Data is taken from the reception register.
2. The reception interrupt of the IR port is disabled to avoid receiving data to be transmitted if the signal is reflected.

3. The transmission complete interrupt on the IR port is set to enable the reception interrupt on the IR port when data has been completely transmitted.
4. Finally, the data is transmitted by the IR port.

When a character is received on the IR port, the data is taken from the reception register and is transmitted by the wired port.

Flowcharts of the main() function and of the interrupts of this project are shown below.

NOTE

RTI indicates a return from interrupt.

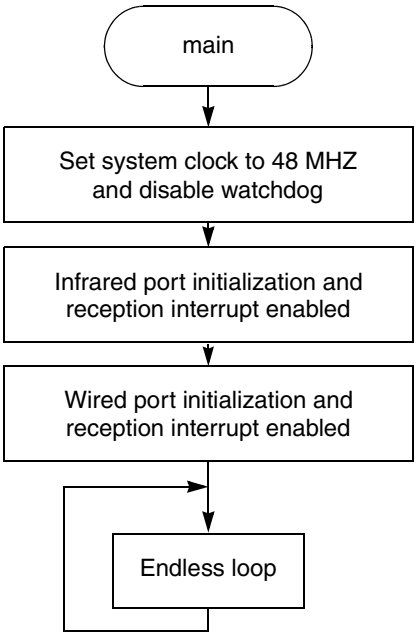


Figure 3. Flowchart of main() function of BridgeIR project

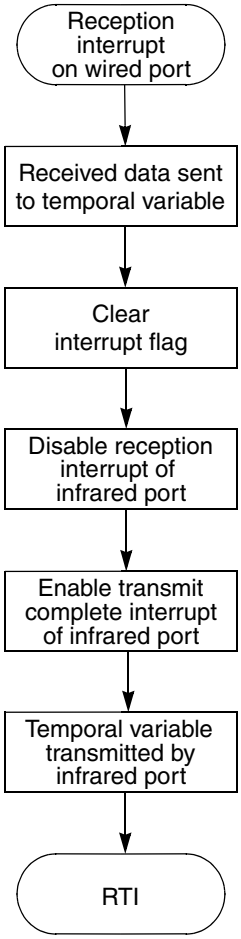


Figure 4. Flowchart of reception interrupt on wired port of BridgeIR project

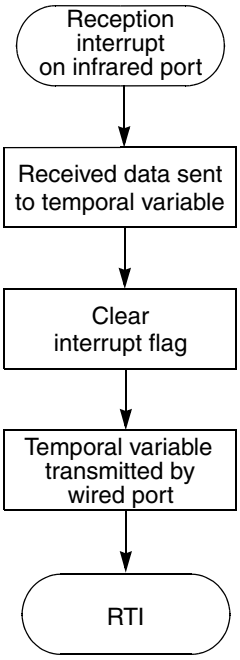


Figure 5. Flowchart of reception interrupt on infrared port of BridgeIR project

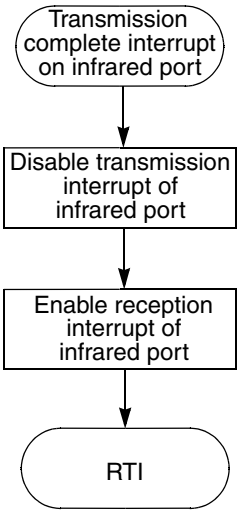


Figure 6. Flowchart of transmission complete interrupt of BridgeIR project

4.1.2 Hardware customization code

This section shows how to customize the code. Code lines in red are customizable by the user.

1. Customize SCI module for wired communication (WIRED_PORT), SCI module for infrared communication (IR_PORT), and baud rate used (BAUDRATE).
   a) Open the file main.c.
   b) Locate these lines (at the beginning of the file):

```
#include "m51em256demo.h"
#include "sci.h"
#include "infrared.h"

#define IR_PORT       1
#define WIRED_PORT    3
/* THESE PORTS CAN NOT BE THE SAME!!!!! */
```

c) Write the number of the SCI modules you want to use (only 1, 2, or 3). These cannot be the same (infrared port must be SCI1 in demo board).

d) Locate this line:

```
#define BAUDRATE      4800
/*1200, 2400, 4800, 9600 max*/
```

e) Write the value of the baud rate (bps) you want to use (commented values are standard baud rates). The max baud rate is limited to 9600 bps to assure signal integrity for the infrared communication.

2. Customize reference level used by the analog comparator.
   a) Open the file main.c.
   b) Locate this line:

```
#define REFERENCE_LEVEL 27
```

   c) Write the value of the reference you want to use. The minimum is 0 and the maximum is 31 (recommended values are 26 to 30), where the voltage reference level is VDD × REFERENCE_LEVEL ÷ 32.
3. Enabling or disabling the analog comparator and the 2× drive strength.
   a) Open the file main.c.
   b) Locate these lines:

```
#define COMPARATOR_USED        1
#define DRIVE_STRENGTH_USED    1
/*0 disables the function; 1 enables the function
```

   c) Write 0 to disable each function, or write 1 to enable it.

NOTE

SCI3 does not support input through the analog comparator on the MCF51EM256.

Here are explanations of the functions included in the infrared.c file, as well as of its input parameters.

4.1.3 Function: u8SCI_init

- Description: Initializes the SCI modules, configuring port, baud rate, data length, parity, transmission mode, and reception mode.

- ANSIC prototype: UINT8 u8SCI_init(UINT8 port, UINT32 baudrate, UINT8 length, UINT8 parity, UINT8 TXmode, UINT8 RXmode);

Table 1. Input parameters

Name	Type
port	unsigned char
baudrate	unsigned long
length	unsigned char
parity	unsigned char
TXmode	unsigned char
RXmode	unsigned char

Table 2. Return value (unsigned char)

Value	Description
0	Successful function execution
1	Invalid port
2	Baud rate error
3	Parity error
4	Transmission mode error
5	Reception mode error

4.1.4 Function: u8IR_TX_init

- Description: Configures pin options, drive strength, modulation source, and interrupt enable for transmission of an SCI module.
- ANSIC prototype: UINT8 u8IR_TX_init(UINT8 port, UINT8 tx_pin_option, UINT8 strength, UINT8 modulation, UINT8 tx_ie)

Table 3. Input parameters

Name	Type
port	unsigned char
tx_pin_option	unsigned char
strength	unsigned char
modulation	unsigned char
tx_ie	unsigned char

Table 4. Return value (unsigned char)

Value	Description
0	Successful function execution
1	Invalid port
2	Invalid modulation source
3	Incorrect selection

4.1.5 Function: u8IR_RX_init

- Description: Configures pin options, comparator reference level and interrupt enable for reception of an SCI module.
- ANSIC prototype: UINT8 u8IR_RX_init(UINT8 port, UINT8 rx_pin_option, UINT8 cmp_level, UINT8 rx_ie)

Table 5. Input parameters

Name	Type
port	unsigned char
rx_pin_option	unsigned char
cmp_level	unsigned char
rx_ie	unsigned char

Table 6. Return value (unsigned char)

Value	Description
0	Successful function execution
1	Invalid port
2	Invalid comparator reference level
3	Incorrect selection

4.2 Protocol project

This project includes a basic protocol with commands based on the ANSI C12.18, including values, packet structure, and length of commands.

The protocol is based on data interchange from and to information tables. These tables have properties that allow partial or complete read and write operations.

The project includes four tables of data, three located in RAM (which lose their value if the system is restarted or powered off) and one located in flash (to conserve its value even if the system is powered off or restarted).

As stated, the protocol is based on the ANSI C12.18 command structure. In this application note these are the services/commands that are implemented:

1. Full read

   Reads a complete data table.
2. Offset read

   Reads a portion of a table, specifying the offset to be added to the table origin and the amount of data to be read.
3. Full write

   Provides a fast command to write a complete data table.
4. Offset write

   Useful to write a portion of a table.

For this implementation, data is sent through an ASCII interface, allowing the use of a simple terminal to send and receive data in hexadecimal format. This ASCII interface can be removed easily to support binary communication.

Some considerations in the use of this protocol software are:

- Data length to be written using the FULL_WRITE command must be less than or equal to the buffer size, including command length.
- The OFFSET_WRITE command must be used to send more data than the buffer size will hold.
- Tables located in nonvolatile memory must be in the second flash array (from 0x20000 to 0x3FFFF), and must be aligned within a flash page. It is recommended to update as much information as possible on a single flash page, which is 1024 bytes. However, it is allowed to write a single byte of information.
- It is possible to execute code from one flash array while the other is being programmed, but it is not allowed to simultaneously program and execute code in a single flash array.

4.2.1 Packet structure

This section describes how packets are structured for implemented commands.

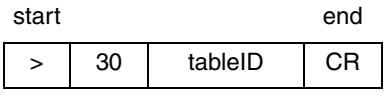
Packets sent through the ASCII interface begin with the character “>” (ASCII 0x3E), then have the command code, followed by the table identifier. The next parameters can change depending on a specific command, but all packets are terminated with a carriage return (CR) character (ASCII 0x0D).

NOTE

All values are in hexadecimal format.

4.2.1.1 Full Read Command

Request:



Element	Description
>	Indicates that a command starts.
30	One-byte command code for Full Read.
tableID	Two-byte identifier for each table.
CR	Carriage return indicates command termination.

Successful response:

OK	element count	data	checksum

Element	Description
OK	One-byte response that indicates that command was supported and executed. Its value is 00.
element count	Two-byte length data that indicates how many bytes will be sent in response to Read command
data	The read information. Its length is defined by “element count” parameter.
checksum	One byte of data — is the two’s-complement of the result of the addition of all data bytes sent.

Unsuccessful response:

NOK

“NOK” is a one-byte response that indicates that the command was not supported or contains an error. Its value is 01.

Table 7. Packet Example — Full Read Command

Request	Response
<command> = 30 <tableID> = 0001	If communication is unsuccessful <NOK> = 01
	If communication is successful <OK> = 00 <octet-count> = 0020 <data> = 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F <checksum> = 10

4.2.1.2 Offset Read Command

Request:

start end

>	3F	tableID	offset	element count	CR

Element	Description
>	Indicates that a command starts.
3F	One-byte command code for Offset Read.
tableID	Two-byte identifier for each table.
offset	Three-byte field that indicates the offset to be applied to the table to read sections of information.
element count	Two-byte field that indicates how many bytes will be read.
CR	Carriage return indicates command termination.

Successful response:

OK	element count	data	checksum

Element	Description
OK	One-byte response that indicates that command was supported and executed. Its value is 00.
element count	Two-byte length data that indicates how many bytes will be sent in response to Read command
data	The read information. Its length is defined by “element count” parameter.
checksum	One byte of data — is the two’s-complement of the result of the addition of all data bytes sent.

Unsuccessful response:

NOK

“NOK” is a one-byte response that indicates that the command was not supported or contains an error. Its value is 01.

Table 8. Packet Example — Offset Read Command

Request	Response
<command> = 3F <tableID> = 0002 <offset> = 000004 <octet-count> = 0010	If communication is unsuccessful <NOK> = 01
	If communication is successful <OK> = 00 <octet-count> = 0010 <data> = A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 <checksum> = 48

4.2.1.3 Full Write Command

Request:

start end

>	40	tableID	element count	data	checksum	CR

Element	Description
>	Indicates that a command starts.
40	One-byte command code for Full Write.
tableID	Two-byte identifier for each table.
element count	Two-byte field that indicates how many bytes will be written.
data	The information to be written. Its length is defined by “element count” parameter.
checksum	One byte of data — is the two’s-complement of the result of the addition of all data bytes sent.
CR	Carriage return indicates command termination.

Successful response:

OK

“OK” is a one-byte response that indicates that the command was supported and executed. Its value is 00.

Unsuccessful response:

NOK

“NOK” is a one-byte response that indicates that the command was not supported or contains an error. Its value is 01.

Table 9. Packet Example — Full Write Command

Request	Response
<command> = 40 <tableID> = 0001 <octet-count> = 0020 <data> = A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 <checksum> = CA	If communication is unsuccessful <NOK> = 01
	If communication is successful <OK> = 00

4.2.1.4 Offset Write Command

Request:

start end

>	4F	tableID	offset	element count	data	checksum	CR

Element	Description
>	Indicates that a command starts.
4F	One-byte command code for Offset Write.
tableID	Two-byte identifier for each table.
offset	Three-byte field that indicates the offset to be applied to the table to write sections of information.
element count	Two-byte field that indicates how many bytes will be written.
data	The information to be written. Its length is defined by “element count” parameter.
checksum	One byte of data — is the two’s-complement of the result of the addition of all data bytes sent.
CR	Carriage return indicates command termination.

Successful response:

OK

“OK” is a one-byte response that indicates that the command was supported and executed. Its value is 00.

Unsuccessful response:

NOK

“NOK” is a one-byte response that indicates that the command was not supported or contains an error. Its value is 01.

Table 10. Packet Example — Offset Write Command

Request	Response
<command> = 4F <tableID> = 0002 <offset> = 000000 <octet-count> = 0020 <data> = 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 <checksum> = 76	If communication is unsuccessful <NOK> = 01
	If communication is successful <OK> = 00

For more information about the protocol refer to the ANSI C12.18 and ANSI C12.19 specifications.

4.2.2 Software Architecture

Reception interrupt of the SCI stores received data in a buffer. If a command start is received RX_status variable indicates COMMAND_RECEIVING, if the command end is received RX_status indicates COMMAND_RECEIVED. Command received flag is read to know when a command has been received completely to allow the buffer conversion into hexadecimal format and analyze command that was received. Depending on the command, the program jumps to execute a specific function for each command to perform the required task.

The function returns an error if the command is wrong or contains incorrect information,

After the response is sent, the RX_status flag indicates that the protocol is waiting to receive a new command (COMMAND_WAITING).

Flow charts of this project are shown below.

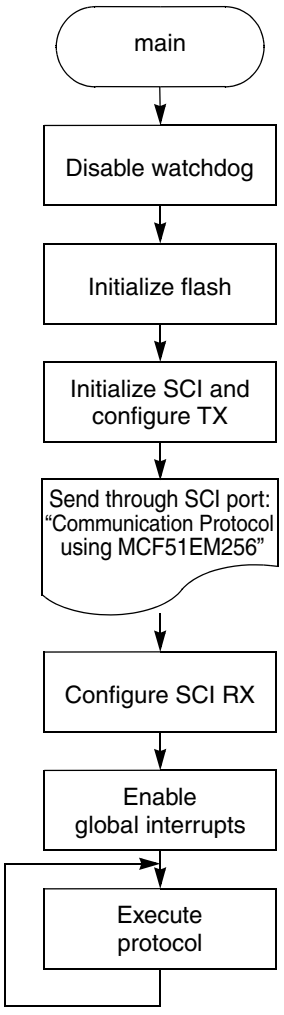


Figure 7. Flowchart of main() function of Protocol project

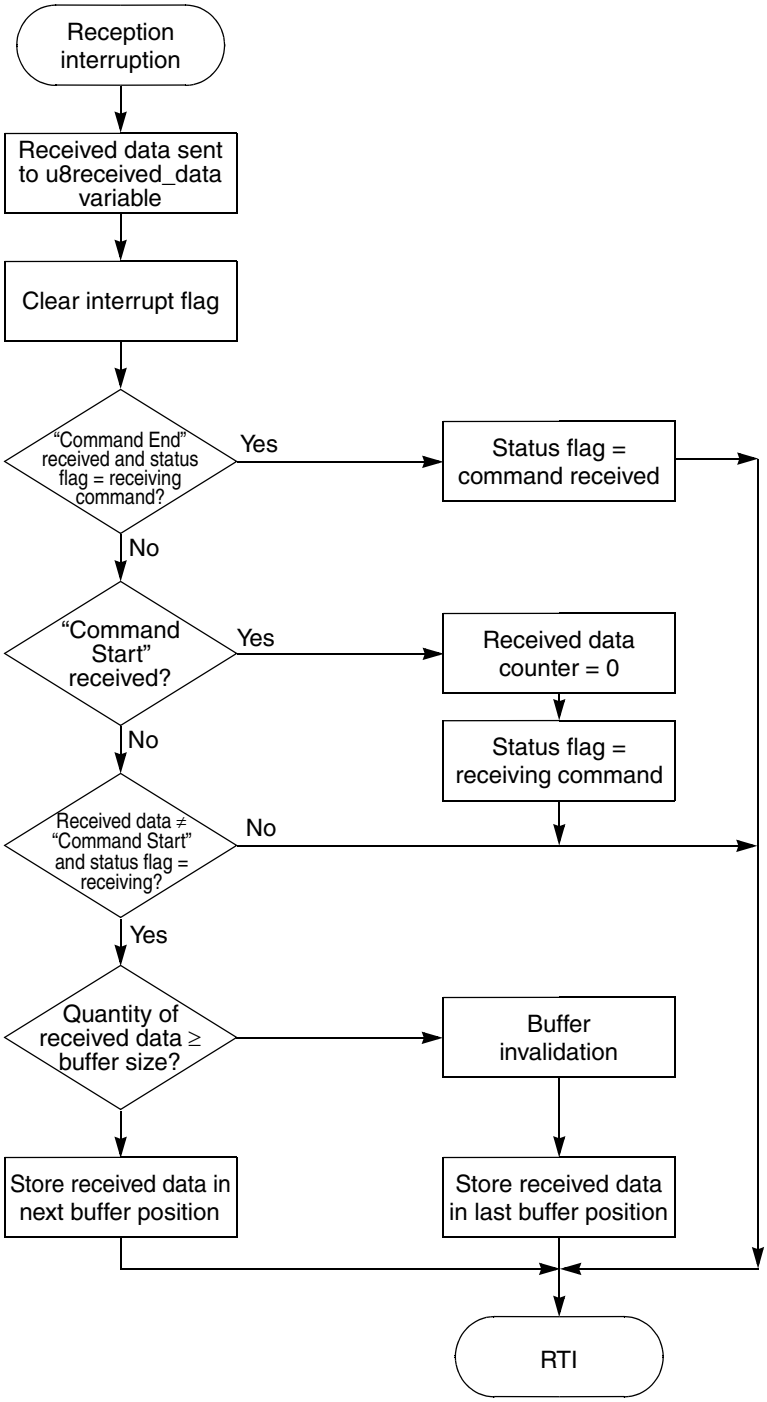


Figure 8. Flowchart of reception interrupt of Protocol project

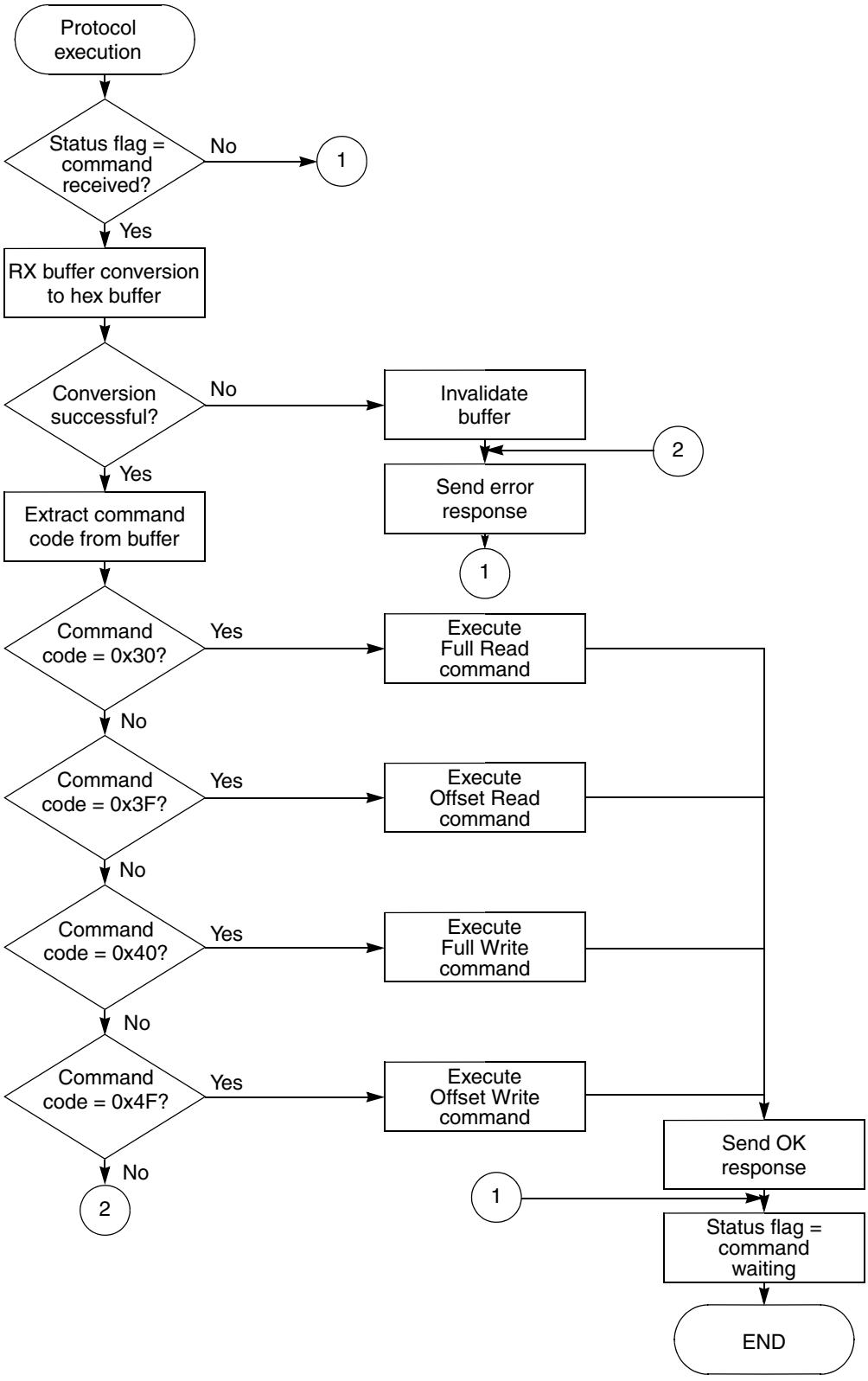


Figure 9. Flowchart of protocol function of Protocol project

4.2.3 Software Customization

Software customization is divided into two parts. The first configures the hardware required by the protocol, and its parameters can be customized:

- SCI module used for communication
- Reference level used by the analog comparator
- Baud rate
- Enable and disable inversion on transmission

The second part customizes the properties of the protocol, and its parameters can also be customized:

- Size of buffer used for reception
- Quantity of tables
- Type of tables (read-only, write-only, read and write)
- Location of the tables (RAM or flash)

4.2.3.1 Hardware customization code

This section shows how to customize the code. Red code lines are customizable. Blue lines are an example of how to add a new table.

1. Customize SCI module and baud rate used for communication.
   a) Open the file com_protocol.h.
   b) Locate this line (at the beginning of the file).

```
/******** DEFINES THE SCI PORT USED AND BAUDRATE *************/
#define USED_PORT  1     /*selects the port used for communication*/
#define BAUDRATE   4800  /*300, 600, 1200, 2400, 4800, 9600 max*/
```

   c) Write the number of the SCI module you want to use (only 1, 2, or 3).
   d) Locate this line:

```
/******** DEFINES THE SCI PORT USED AND BAUDRATE *************/
#define USED_PORT  1     /*selects the port used for communication*/
#define BAUDRATE   4800  /*300, 600, 1200, 2400, 4800, 9600 max*/
```

   e) Write the value of the baud rate (bps) you want to use (only the values shown in the comment are valid).
2. Customize reference level used by the analog comparator.
   a) Open the file com_protocol.h.
   b) Locate this line:

```
#define REFERENCE_LEVEL        27       /*Reference voltage used by the comparator*/

#define IR_ACTIVE              1        /* 1=TX INVERTED (reference level used); 0=TX
NORMAL (reference level ignored)*/
```

c) Write the value of the reference you want to use.
The minimum is 0 and the maximum is 31 (recommended values are 26 to 30).

3. Enable or disable inversion for transmitted data.
   a) Open the file com_protocol.h.
   b) Locate this line:

```
#define REFERENCE_LEVEL 27 /*Reference voltage used by the comparator*/

#define IR_ACTIVE        1 /* 1=TX INVERTED (reference level used); 0=TX NORMAL
(reference level ignored)*/
```

   c) Write 1 if you want to invert the SCI transmission (used in infrared) or write 0 if you do not want to invert the signal (normal mode).

4.2.3.2 Protocol customization code

4. Customize the size of the buffer used for reception.
   a) Open the file com_protocol.h.
   b) Locate this line:

```
/*************** CONSTANTS DEFINITIONS ******************/
#define RX_BUFFER_SIZE 2048
/*Defines the buffer size used for reception*/
#define HEX_BUFFER_SIZE RX_BUFFER_SIZE/2
/*Defines the buffer size used for converted values*/
```

   c) Write the size of the buffer you want to use. This value is the maximum number of ASCII characters that the system can receive per packet. The recommended maximum value is 2048, because the buffer of hexadecimal data is half of this value, and 1024 bytes is the size of a flash page. If you need to send more data, you must use the Offset Write command.

5. Customize the quantity, size, and type of tables, and generate the table description.
   a) Open the file com_protocol.h.
   b) Locate this line to define the number of tables:

```
#define TABLES_NUMBER     4        /*defines number of tables*/
```

   c) Write the number of tables you want to use. For example, to add a new table in existing code, the code line will be:

```
#define TABLES_NUMBER     5        /*defines number of tables*/
```

   d) Locate these lines to define the size of each table:

```
/* TABLEx_SIZE: size of the table in bytes */
#define TABLE0_SIZE  64   /*Defines the buffer size of Table 0 (electrical
measurements)*/
#define TABLE1_SIZE  32   /*Defines the buffer size of Table 1 (calibration [needs
password])*/
```

```
#define TABLE2_SIZE  32   /*Defines the buffer size of Table 2 (seetings)*/
#define TABLE3_SIZE  128  /*Defines the buffer size of Table 3  (data logger)*/
```

e) Write the size you want for each table. If a new table was inserted, its size must be defined in this part:

```
/* TABLEx_SIZE: size of the table in bytes */
#define TABLE0_SIZE  64   /*Defines the buffer size of Table 0 (electrical
measurements)*/
#define TABLE1_SIZE  32   /*Defines the buffer size of Table 1 (calibration [needs
password])*/
#define TABLE2_SIZE  32   /*Defines the buffer size of Table 2 (seetings)*/
#define TABLE3_SIZE  128  /*Defines the buffer size of Table 3  (data logger)*/
#define newTABLE_SIZE 16  /*Defines the buffer size of new table */
```

f) Locate these lines to define the type of tables:

```
/* TABLEx_TYPE: type of the table */
#define R_AND_W   0   /* read and write */
#define R_ONLY    1   /* read only      */
#define W_ONLY    2   /* write only     */

#define TABLE0_TYPE  R_AND_W
#define TABLE1_TYPE  R_AND_W
#define TABLE2_TYPE  R_AND_W
#define TABLE3_TYPE  R_AND_W
```

g) Write R_ONLY for read-only tables, W_ONLY for write-only tables, and R_AND_W for tables used for read and write. If a new table was included, its proprieties must be defined here:

```
#define TABLE0_TYPE  R_AND_W
#define TABLE1_TYPE  R_AND_W
#define TABLE2_TYPE  R_AND_W
#define TABLE3_TYPE  R_AND_W

#define newTABLE_TYPE  R_AND_W
```

h) Open the file com_protocol.c.

i) Locate these lines to generate the table description:

```
/************ STRUCT FOR TABLES PROPERTIES ************/
struct Table_Description Table_list[TABLES_NUMBER] =
{
   {&Table0[0],TABLE0_SIZE,TABLE0_TYPE,TABLE0_SECURITY,"0.- Measurements"},
   {&Table1[0],TABLE1_SIZE,TABLE1_TYPE,TABLE1_SECURITY,"1.- Calibration"},
   {&Table2[0],TABLE2_SIZE,TABLE2_TYPE,TABLE2_SECURITY,"2.- Settings"},
   {&Table3[0],TABLE3_SIZE,TABLE3_TYPE,TABLE3_SECURITY,"3.- Data logger"},
};

struct Table_Description *p_table_list;
```

j) The first parameter is the name of the array that contains the table, and is explained in the next step. Fill the rest of the parameters for each table with the parameters that were defined above. If a new table was inserted, its table descriptor will be added as shown here:

```
/************ STRUCT FOR TABLES PROPIETIES ************/
struct Table_Description Table_list[TABLES_NUMBER] =
{
   {&Table0[0],TABLE0_SIZE,TABLE0_TYPE,TABLE0_SECURITY,"0.- Measurements"},
   {&Table1[0],TABLE1_SIZE,TABLE1_TYPE,TABLE1_SECURITY,"1.- Calibration"},
   {&Table2[0],TABLE2_SIZE,TABLE2_TYPE,TABLE2_SECURITY,"2.- Settings"},
   {&Table3[0],TABLE3_SIZE,TABLE3_TYPE,TABLE3_SECURITY,"3.- Data logger"},

   {&newTable[0],newTABLE_SIZE,newTABLE_TYPE,newTABLE_SECURITY,"4.- New"},

};

struct Table_Description *p_table_list;
```

6. Generate the tables that will be located in RAM.

a) The previous lines explain how to define table properties and its descriptor, but the table was not yet created. Tables are an array of variables, and if they are located in RAM, open the file com_protocol.c and locate these lines:

```
/*************** DECLARATION OF THE VOLATILE TABLES ***************/
UINT8 Table0[TABLE0_SIZE];            /*Buffer used by Table 0*/
UINT8 Table2[TABLE2_SIZE];            /*Buffer used by Table 2*/
UINT8 Table3[TABLE3_SIZE];            /*Buffer used by Table 3*/

UINT8 temp_Table[HEX_BUFFER_SIZE];  /*Backup table used in flash write*/
```

b) To add a new table saved in RAM, it must be added as shown here:

```
/*************** DECLARATION OF THE VOLATILE TABLES ***************/
UINT8 Table0[TABLE0_SIZE];            /*Buffer used by Table 0*/
UINT8 Table2[TABLE2_SIZE];            /*Buffer used by Table 2*/
UINT8 Table3[TABLE3_SIZE];            /*Buffer used by Table 3*/

UINT8 newTable[newTABLE_SIZE];        /*Buffer used by new Table*/

UINT8 temp_Table[HEX_BUFFER_SIZE];  /*Backup table used in flash write*/
```

7. g) Generate tables located in flash.

To locate tables in flash memory, the Linker Command file must be edited as shown here:

a) Open the file Project.lcf.

b) In a new project, the original lines are:

```
# Memory ranges

MEMORY {
   code         (RX)   : ORIGIN = 0x00000410, LENGTH = 0x0003FBF0
   userram      (RWX)  : ORIGIN = 0x00800000, LENGTH = 0x00004000
}
```

c) To allow storing data in flash, the lines must be replaced with:

```
# Memory ranges

MEMORY {
   code         (RX)   : ORIGIN = 0x00000410, LENGTH = 0x0001FBF0
   my_tables    (RX)   : ORIGIN = 0x00020800, LENGTH = 0x00000400
   userram      (RWX)  : ORIGIN = 0x00800000, LENGTH = 0x00004000
}

FORCE_ACTIVE{_Table1}
```

In this case, the code will be located only in flash array 1 (128 KB), and a new type of data is created and saved in flash array 2 which begins at address 0x0020000. But the first two pages contain configuration registers, so the code will be used from the third page (address 0x0020800). The size of this data type is 1024 bytes (one memory sector), and is named my_tables. A new memory range must be added for each table to locate it in a different memory sector, allowing erasing and programming each of them independently. For example, a new table saved in flash is added at the last sector of flash array 2 (address 0x0003FC00), and its memory ranges must be declared as shown here:

```
# Memory ranges

MEMORY {
   code         (RX)   : ORIGIN = 0x00000410, LENGTH = 0x0001FBF0
   my_tables    (RX)   : ORIGIN = 0x00020800, LENGTH = 0x00000400
   my_table2    (RX)   : ORIGIN = 0x0003FC00, LENGTH = 0x00000400
   userram      (RWX)  : ORIGIN = 0x00800000, LENGTH = 0x00004000
}

FORCE_ACTIVE{_Table1}
FORCE_ACTIVE{_newTable}
```

d) Insert these lines in the Project.lcf file to create a new memory section to store flash tables.

```
#-------------------------------------------------------------------
#-------------------------------------------------------------------
.rom_symbols :
{
  __ROM_SYMBOLS =  . ;   #start address of the 4x1024 bytes symbols
  *(.romsymbols)
} > my_tables

#FSL:copy during runtime
___ROM_SYMBOLS   =   __ROM_SYMBOLS; #ADDR(.my_tables);

#-------------------------------------------------------------------
#-------------------------------------------------------------------
```

One section must be created for each table. For example, to add the section for a new table, add these lines:

```
#-------------------------------------------------------------------
```

```
#-------------------------------------------------------------------
.rom_symbols :
{
   __ROM_SYMBOLS =  . ;   #start address of the 4x1024 bytes symbols
   *(.romsymbols)
} > my_table


#FSL:copy during runtime
___ROM_SYMBOLS =                              __ROM_SYMBOLS; #ADDR(.my_table);

#-------------------------------------------------------------------
#-------------------------------------------------------------------

.rom_symbols2 :
{
   __ROM_SYMBOLS2 =  . ;  #start address of the 4x1024 bytes symbols
   *(.romsymbols2)
} > my_table2


#FSL:copy during runtime
___ROM_SYMBOLS2           =      __ROM_SYMBOLS2; #ADDR(.my_table2);

#-------------------------------------------------------------------
#-------------------------------------------------------------------
```

These lines locate the variables type my_tables in the memory section called romsymbols. Table 1 is saved here. The next steps explain how to do this.

e) Open the file com_protocol.c.

f) Locate these lines:

```
/************ DECLARATION OF THE NON-VOLATILE TABLES ************/
/*This is used to locate Nonvolatile Tables in Flash memory*/

#pragma define_section my_tables ".romsymbols"  /*start of pragma*/
/*following code will be located in memory section called .romsymbols */

#pragma section my_tables begin
UINT8 Table1[TABLE1_SIZE] =
{
   /* here must be the table data*/
};

#pragma section my_tables end/*end of pragma*/
```

The #pragma must be started and finished after invoking it. To add a new flash table, a new pragma must be inserted to include its array declaration in this part, as shown here:

```
#pragma define_section my_tables ".romsymbols"  /*start of pragma*/
/*following code will be located in memory section called .romsymbols */
```

```
#pragma section my_tables begin
UINT8 Table1[TABLE1_SIZE] =
{
   /* here must be the table data*/
};

#pragma section my_tables end/*end of pragma*/


#pragma define_section my_table2 ".romsymbols2"  /*start of pragma*/
/*following code will be located in memory section called .romsymbols2 */

#pragma section my_table2 begin

UINT8 newTable[newTABLE_SIZE] =
{
   /* here must be data of new table*/
};

#pragma section my_table2 end                    /*end of pragma*/
```

NOTE

Each flash array has reserved flash locations on pages 1 and 2. These registers must not be used to save tables. It is recommended to create sections to save nonvolatile tables from page 3 (address 0x0020800 on flash array 2).

4.3 EM256DemoIR project

This project consists of the execution of several different tasks, and the protocol is included as one of those tasks. It also uses the other tasks to implement different hardware tests, using modules present in the MCF51EM series, such as:

- LCD driver
- RTC timer
- SCI and infrared hardware
- TPM in PWM mode
- Analog comparator
- 2× drive strength functions for SCI TX1 and TX2

4.3.1 Including protocol as a task

To add the protocol to an application, the files to be added are:

For protocol definition and functions:

- com_protocol.h
- com_protocol.c

For infrared port initialization and settings:

- infrared.h
- infrared.c

For ASCII interface used by terminal

- ascii.h
- ascii.c

For flash initialization and erase and write functions:

- flash.h
- flash.c

For SCI functions, such as message transmission and others:

- sci.h
- sci.c

In addition, the line shown here must be added in your main initialization to configure the flash properties, allowing erase and write operations.

```
void main(void)
{
SOPT1_COPT = 0;/* Turn off Watchdog Timer*/
FlashInit(); /*Configures the Flash clock for erase and program operations*/
```

Also, depending on the application, initialization and configuration lines for SCI modules could need to be included. In the EM256DemoIR project, these lines are inserted:

```
if (u8SCI_init(USED_PORT, 300, DATA_LENGTH_EIGTH, PARITY_NONE, (IR_ACTIVE+1),
RX_NOT_INVERTED)) for(;;){};
if (u8IR_TX_init(USED_PORT, TX_PIN_OPTION_DEFAULT, DRIVE_STRENGTH_DISABLED,
MODULATION_DISABLED, TX_INTERRUPT_DISABLED)) for(;;){};
if (u8IR_RX_init(USED_PORT, RX_PIN_OPTION_DEFAULT, (REFERENCE_LEVEL*IR_ACTIVE),
RX_INTERRUPT_ENABLED)) for(;;){};
```

4.3.2 Software Architecture

This software is a task-scheduler, and each task does a different function. The task initialization prototypes are:

```
/*** Main Loop Tasks prototypes ***/
void vfnTSK_welcome(void);
void vfnTSK_show_change_br(void);
void vfnTSK_show_ir(void);
void vfnTSK_show_tx_irda(void);
void vfnTSK_protocol_welcome(void);
void vfnTSK_comm_protocol(void);
```

Relations between tasks are shown in Figure 10. Double-line circles are the tasks, and the single-line circles are functions called by each task.

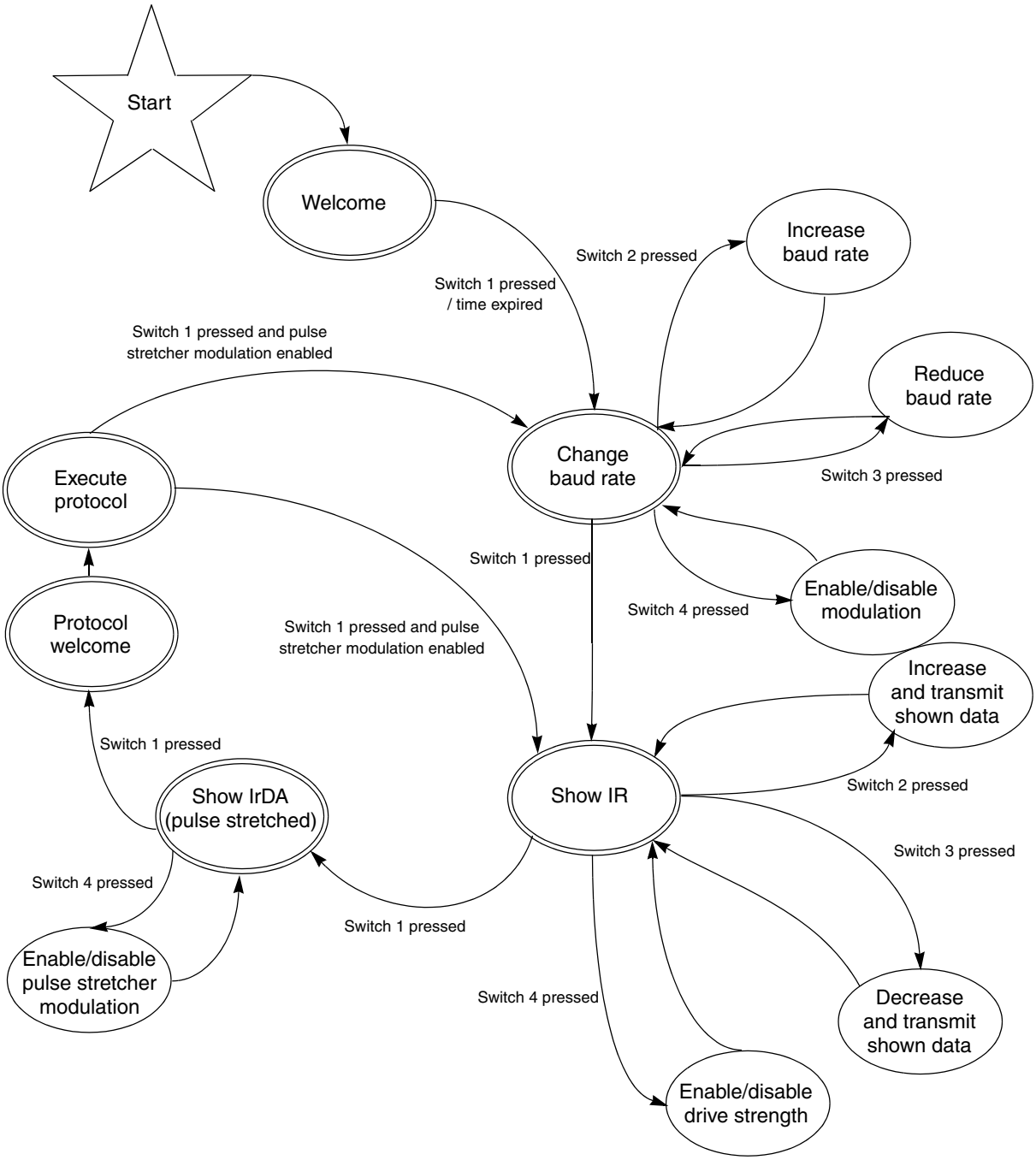


Figure 10. EM256DemoIR project state/tasks diagram

- vfnTSK_welcome

  This task shows a welcome message on the LCD screen. The screen shows “IR demoEM” for three seconds and then jumps to vfnTSK_show_change_br task. Press switch 1 to jump immediately.
- vfnTSK_show_change_br

This task sets the baud rate to be used by the application. Switch 2 increases the baud rate and switch 3 decreases it. This function also enables and disables the high-frequency modulation by pressing switch 4. Press switch 1 to jump to vfnTSK_show_ir task.

- vfnTSK_show_ir

  This task sends and receives characters via the IR port and values are shown on the LCD screen. Press switch 2 to increase the data to be sent, and switch 4 to reduce it. Press and hold switch 3 or switch 4 to send data every 100 milliseconds. It also enables and disables the 2× drive strength by pressing switch 4. Press switch 1 to jump to vfnTSK_show_tx_irda task.

- vfnTSK_show_tx_irda

  This task enables and disables the pulse stretcher modulation by pressing switch 4. If pulse stretcher modulation is enabled, the baud rate is forced to 9600 bps, and the task used to change baud rate is not available until the pulse stretcher modulation is disabled. Press switch 1 to jump to vfnTSK_protocol_welcome task.

- vfnTSK_protocol_welcome

  This task sends the text “Protocol” through the SCI port, displays it on the LCD screen, and then immediately jumps to vfnTSK_comm_protocol task.

- vfnTSK_comm_protocol

  This task executes the protocol explained previously. The LCD screen shows the message “Protocol” and LED4 blinks when the command is received and a response is sent. Press switch 1 to jump to the next task. Press switch 1 to jump to vfnTSK_show_change_br task.

The flow charts of the main and interrupt sections of this project are shown in Figure 11:

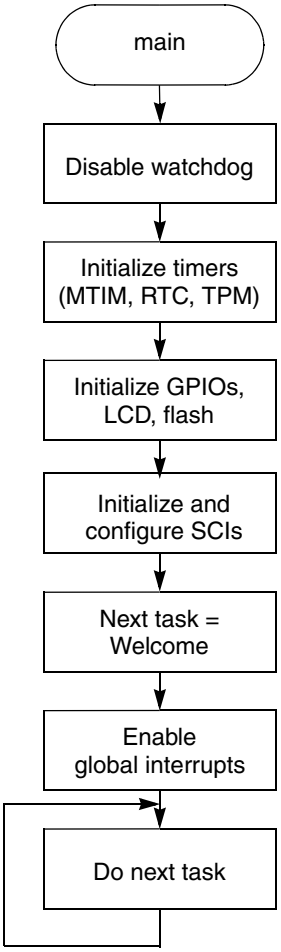


Figure 11. Flowchart of main() function of EM256DemoIR project

5 Hardware tests

The tests discussed in this section show results and measurements done on DEMOEM and highlight the features of the MCF51EM256 used for infrared communication, such as the use of 2× drive strength reception through the analog comparator, the possibility of generating a modulating signal using timers, plus the implementation of the provided communication protocol.

Figure 2 shows the switches and jumpers mentioned in the tests. A brief description of each test is given below:

- Test 1: Drive strength and analog comparator functions

  Transmission on infrared port with 2× drive strength, and infrared reception with SCI RX input connected through analog comparator.
- Test 2: High frequency modulation

  High frequency modulation (38 kHz) using 2× drive strength. Modulated signal is generated by timer TPM.
- Test 3: Pulse stretcher modulation

Timer TPM generates a PWM signal to modulate SCI transmission in pulse stretcher mode.

- Test 4: Protocol Implementation:

  Implementation of the communication protocol, showing screen captures of the request and response codes received and sent by the PC.

5.1 Test considerations

Tests one, two, and three are implemented using the EM256DemoIR project and these hardware considerations:

- V_{DD} = 3.3 V
- R_{13} = 1.5 KΩ[1] and R_{14} is 33 Ω

Test four is implemented using the BridgeIR project on one DEMOEM board, and the protocol project on the other DEMOEM board.

To do these tests, these jumper configurations on DEMOEM must be used:

- J16 populated for PTE0, PTB2, and PTB3
- J7 and J8 connecting pin 1 and pin 2
- J9 and J10 connecting pin 2 and pin 3

For more information about jumper settings or other hardware issues concerning the demo board, refer to document DEMOEMUM, *DEMOEM User Manual,* available at www.pemicro.com, and/or schematic.

5.2 Test 1: Drive strength and analog comparator functions

This test shows the behavior at different baud rates of a transmission signal using 2× drive strength with reception using an analog comparator, and the relation of this behavior to distance of communication.

Here are the conditions for this test:

- Serial port SCI1, PRACMP1 without feedback resistor
- Transmitted data = 0x41
- Drive strength enabled
- Distance is taken from board edges, with 6 mm acrylic between boards and transmitter aligned with receiver
- Light conditions: indoor fluorescent lamps and without optical filter
- Comparator reference is 28

Table 11 summarizes the results of the test at different baud rates and distances. Figure 12 is a schematic of the hardware used in the test.

1.DEMOEM board is originally populated with R13 = 15 kΩ resistor.

Table 11. Test 1 results

Distance (cm)	Baud Rate (bps)	Successful Communication?	Notes
0[1]	300	Yes	Figure 13
0[1]	600	Yes	
0[1]	1200	Yes	
0[1]	2400	Yes	Figure 14
0[1]	4800	Yes	
0[1]	9600	Yes	Figure 15
1.8	300	Yes	Figure 16
1.8	600	Yes	
1.8	1200	Yes	Figure 17
1.8	2400	Yes	
1.8	4800	Yes	
1.8[1]	9600	Yes	Figure 18

[1] The distance between IR transmitter and receiver plus the thickness of the acrylic is 1.5 cm.

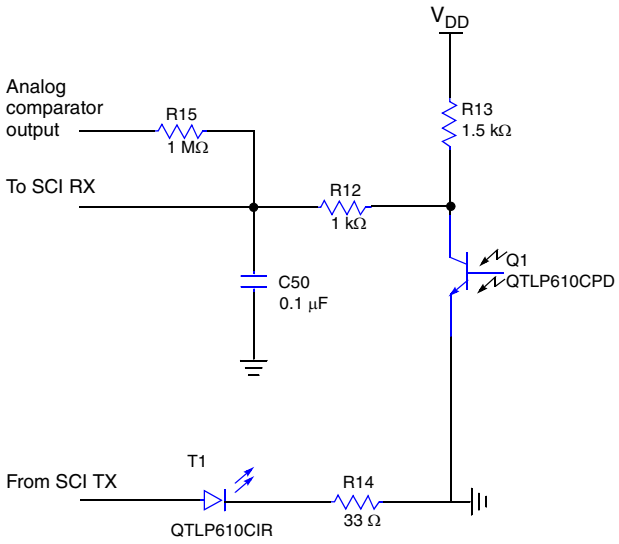


Figure 12. Connection of IR transmitter and receiver with feedback resistor

Figure 13 through figure 18 are screen captures of the oscilloscope showing measurements of the transmitted and received signals. Channel 1 (yellow) is the signal sent by the transmission board. Channel 2 (blue) is the signal received by the phototransistor and applied to the SCI RX pin. Channel 3 (pink) is the received signal at the output of the analog comparator.

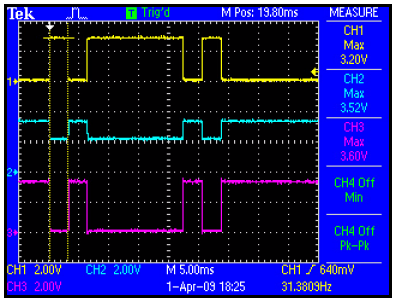


Figure 13. Distance = 0 cm, baud rate = 300 bps

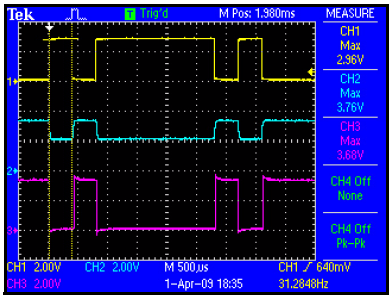


Figure 14. Distance = 0 cm, baud rate = 2400 bps

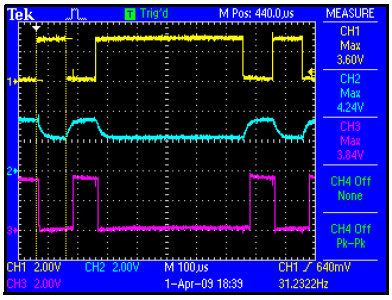


Figure 15. Distance = 0 cm, baud rate = 9600 bps

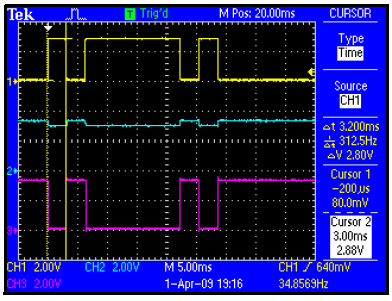


Figure 16. Distance = 1.8cm, baud rate = 300 bps

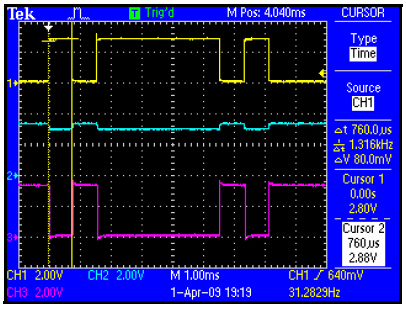


Figure 17. Distance = 1.8cm, baud rate = 1200 bps

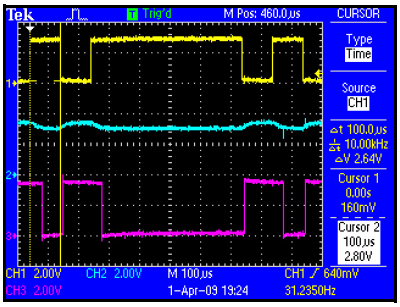


Figure 18. Distance = 1.8cm, baud rate = 9600 bps

5.3 Test 2: High frequency modulation

This test shows the implementation of the timer TPM to generate a 38 kHz signal applied to the transmission signal to get modulation.

Here are the conditions for this test:

- Modulation at 38 kHz using timer TPM channel 0
- Serial port SCI1
- Transmitted data = 0x41
- Drive strength enabled
- Light conditions: indoor fluorescent lamps

Figure 19 shows transmitted data modulated by a 38 kHz modulating signal. Figure 20 shows the period of the modulating signal.

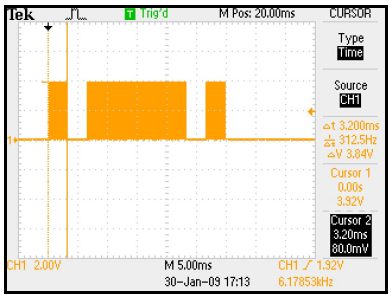


Figure 19. Transmission modulated at 38 Hz — baud rate is 300 bps

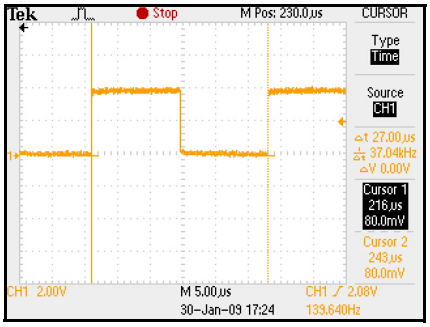


Figure 20. Modulating signal period

5.4 Test 3: Pulse stretcher modulation

This test shows the implementation of the timer TPM in PWM mode to generate a pulse signal applied to the transmission signal to get pulse stretcher modulation. This kind of modulation is used in IrDA. The conditions for this test are:

- Modulation to 3/16 of a bit, using SCI and timer in PWM mode
- A pulse indicates a logical 0
- Baud rate is 9600 bps and transmitted data is 0x41

In Figure 21 channel 1 of the scope is the signal without modulation, and channel 2 is the modulated signal.

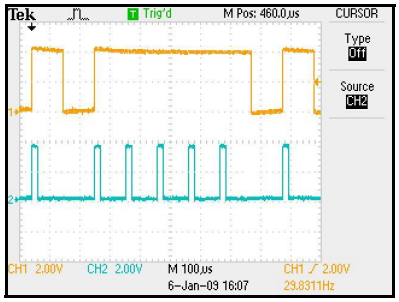


Figure 21. Pulse stretcher signal

5.5 Test 4: Protocol implementation

This test shows the implementation of the communication protocol using two demo boards, one used as an infrared bridge and the other executing the protocol.

Here are the conditions for this test:

- Computer is connected to DEMOEM#1 which has been programmed with BridgeIR software
- DEMOEM#2 has been programmed with protocol software
- Boards are aligned (TX→RX, RX→TX)
- Computer is running the Terminal Window application[1], because DEMOEM has an embedded serial port through USB port supported only by this application. If you want to use any other

1.TerminalWindow.exe is included in AN3938.zip, the associated software for this application note.

terminal application, you must populate the DB9 connector and U2 integrated circuit (transceiver), and change J7 and J8 jumpers.

Terminal Window application is configured with these settings:

- Port: USB COM
- Baud rate: 4800 (recommended value)
- Parity: none
- Bits: 8
- Duplex: Full
- Char delay (ms): 5
- EOL delay (ms): 15

To execute a command, write it in the base of the packet example structure explained in Section 4.2.1, "Packet structure."

Pre-written commands are included in AN3938.zip as text files, and are organized in four folders according to the command function. To execute a pre-written command, click on the button "Download File to ComPort" and select the text file you want to download.

Figure 22 and Figure 23 show an overview of the connections and orientation of the boards to execute the tests.

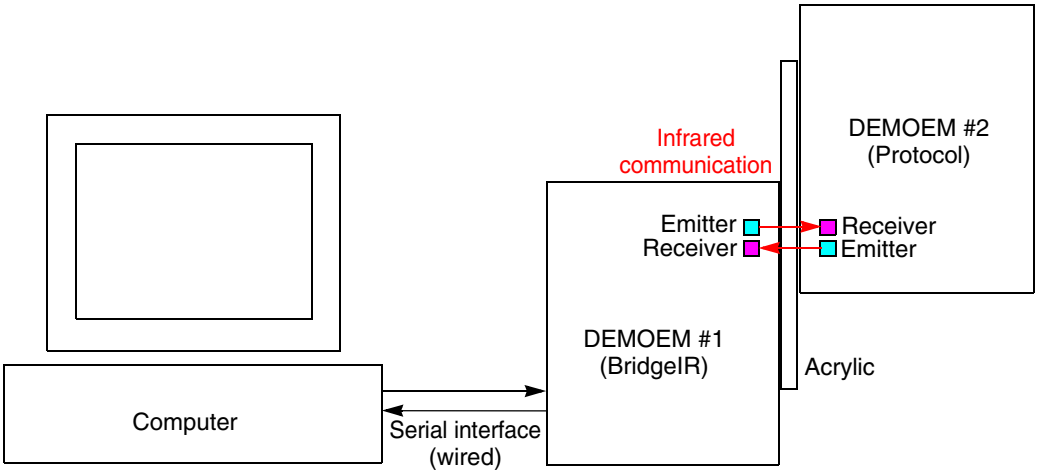


Figure 22. Block diagram of connections between boards

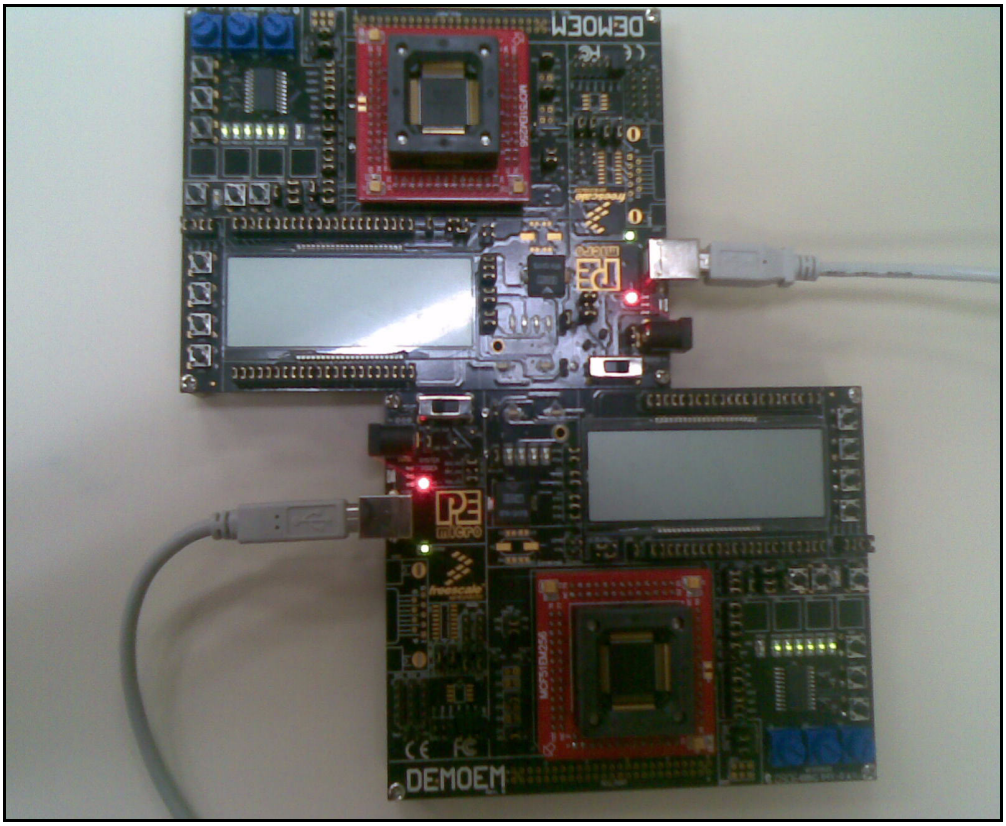


Figure 23. Aligned boards executing test

Figure 24 is a screen capture of the Terminal Window application showing the protocol executing commands.

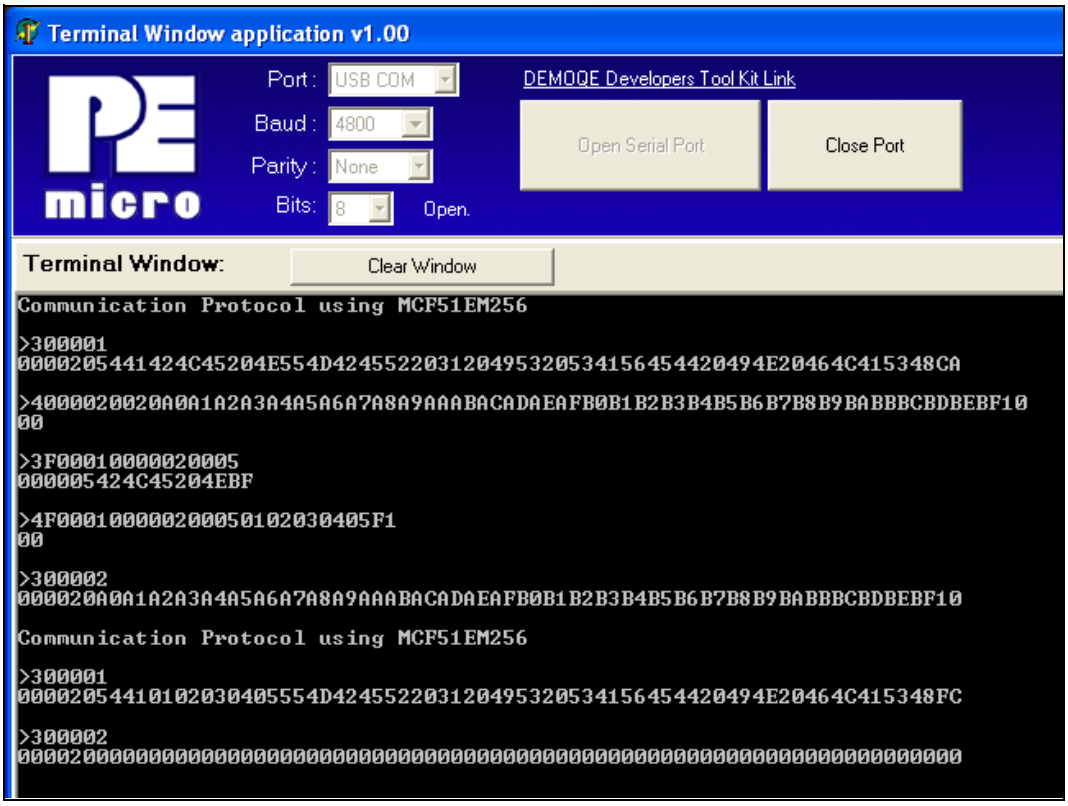


Figure 24. Terminal Window application: protocol executing commands

First command is Full Read on Table 1.

Next command is Full Write on Table 2.

Next command is an Offset Read command on Table 1, followed by an Offset Write command on Table 1.

Then a reset was applied to the board, and the protocol was restarted.

The next command, a Full Write of Table 1, shows the nonvolatile data stored in Table 1. But the data in Table 2 was lost, as shown in the last command, a Full Read of Table 2.

6 Considerations and references

- The user has to be careful when applying customization to the software to avoid conflicts or short circuits. Commented text can help to avoid this kind of issue.
- Recommended baud rates and reference levels help to achieve better results in communication, because higher values for the baud rate could corrupt information, and greater values of the reference level could cause the signal not to be recognized.
- Baud rate can be increased if wired communication is used. The protocol was tested up to 19200 bps, working without problems via wired communication.
- For more information about SCI, ACMP, and TPM, refer to Freescale document MCF51EM256RM, *MCF51EM Family ColdFire® Integrated Microcontroller Reference Manual,* available at www.freescale.com.

- The software was developed and tested with CodeWarrior for Coldfire v6.2.1.
- Download the tools and source files for AN3938SW.zip from www.freescale.com.

7 Conclusion

This application note showed the use of specific hardware features of the MCF51EM256 to establish infrared communication.

The transmission distances shown in Test one were obtained without using any optical protection of infrared elements; higher distances can be obtained using infrared optical protection that avoids external light interference.

The provided protocol is a portion of the C12.18 Protocol that allows a complete data interchange. It also provides bases to allow the user to complete the protocol or customize it to a specific application.

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: AN3938
Rev. 0
10/2009

freescale™
semiconductor