Version 1.1

eRIC_I2C:

eRIC has one I2C module.

The I2C mode features include:

- Compliance to the Philips Semiconductor I2C specification v2.1
- 7-bit and 10-bit device addressing modes
- General call
- START/RESTART/STOP
- Multi-master transmitter/receiver mode
- Slave receiver/transmitter mode
- Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- Slave receiver START detection for auto wake up from LPMx modes
- Slave operation in LPM2

I2C data is communicated using the serial data (SDA) pin and the serial clock (SCL) pin. Both SDA and SCL are bidirectional and must be connected to a positive supply voltage using a pullup resistor.

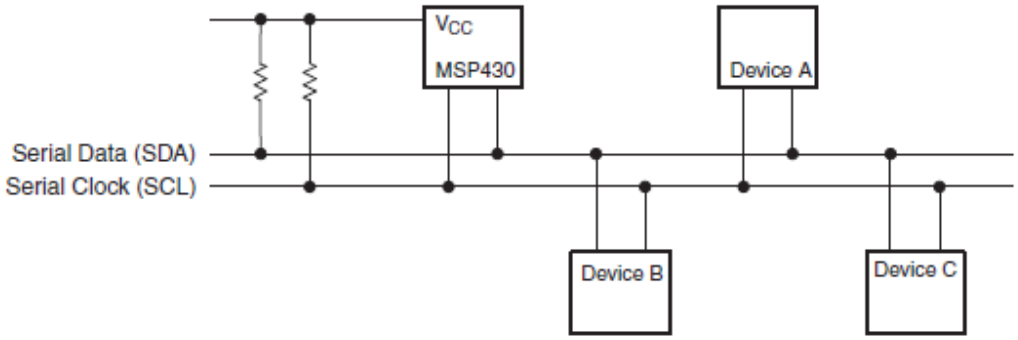

Figure 24-2. I²C Bus Connection Diagram

NOTE: SDA and SCL levels

The SDA and SCL pins must not be pulled up above the device V_{CC} level.

In this application, 10kohms pull up resistors are used.

Refer latest eRIC_eROS_Developers_Manual_x.x for complete list of I2C definitions. Refer SLAU259E document by Texas Instruments to use core I2C registers.

Initialising I2C with desired clock speed in either Master/Slave mode and sending and receiving data on SDA line with following registers or functions is all needed for it to work.

Initialising I2C:

eRIC_I2CB_Initialise(I2CClock, MasterorSlave, Address));
I2C is initialised by passing I2CClock, choosing master or Slave and address.
I2CClock: clock can be anything upto 100000 in standard mode and upto 400000 in fast mode.
MasterorSlave: Choose 1 if Master or 0 if slave
Address: It can be any one byte for 7 bit address mode or 2 bytes for 10 bit addressing mode.

7-Bit Addressing

In the 7-bit addressing format (see Figure 24-5), the first byte is the 7-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte.

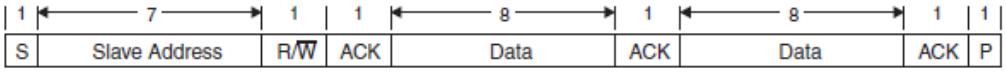

Figure 24-5. I²C Module 7-Bit Addressing Format

10-Bit Addressing

In the 10-bit addressing format (see Figure 24-6), the first byte is made up of 11110b plus the two MSBs of the 10-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte. The next byte is the remaining eight bits of the 10-bit slave address, followed by the ACK bit and the 8-bit data. See I2C Slave 10-bit Addressing Mode and I2C Master 10-bit Addressing Mode for details how to use the 10-bit addressing mode with the USCI module.

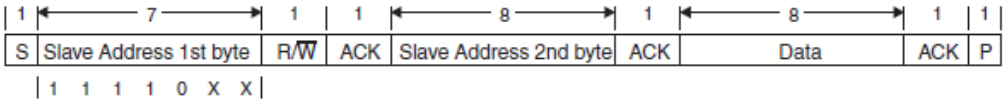

Figure 24-6. I²C Module 10-Bit Addressing Format

The address in Master mode is copied onto eRIC_I2CB_SlaveAddress and if it is in slave mode then it is copied on to eRIC_I2CB_OwnAddress. The address is right justified. In 7-bit addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit addressing mode, bit 9 is the MSB. By default 7 bit addressing mode is selected.

eRIC_I2CB_AsTransmitter();
I2C is set as transmitter to transmit data. This can also be used as Write bit for some I2C application like EEPROM.

eRIC_I2CB_AsReceiver();
I2C is set as receiver to receive data. This can also be used as Read bit for some I2C application like EEPROM.

eRIC_I2CB_SendByte(Data);
This will transmit one byte of data over SDA
eRIC_I2CB_ReceiveByte();
This will receive one byte of data over SDA
eRIC_I2CB_SoftwareResetEnable();
Some of the settings of I2C can only be when when software reset is enabled.

eRIC_I2CB_SoftwareResetDisable();
After changing settings, software reset should be disabled.

eRIC_I2CB_MultiMasterMode();
This is used when multi master devices are required.
Modify only when software reset is enabled.

eRIC_I2CB_SingleMasterMode();
This is used when single master is operated. By default this is used.
Modify only when software reset is enabled.

eRIC_I2CB_10BitSlaveAddress();
This is the register to hold 10 bit slave address in master mode. This can be already set in eRIC_I2CB_Initialise(I2CClock, MasterorSlave, Address));

eRIC_I2CB_7BitSlaveAddress();
This is the register to hold 7 bit slave address in master mode. This can be already set in eRIC_I2CB_Initialise(I2CClock, MasterorSlave, Address));

eRIC_I2CB_10BitOwnAddress();
This is the register to hold 10 bit own address in slave mode. This can be already set in eRIC_I2CB_Initialise(I2CClock, MasterorSlave, Address));
Modify only when software reset is enabled.

eRIC_I2CB_7BitOwnAddress();
This is the register to hold 7 bit own address in slave mode. This can be already set in eRIC_I2CB_Initialise(I2CClock, MasterorSlave, Address));
Modify only when software reset is enabled.

eRIC_I2CB_Start();
This is used to start I2C protocol. Transmit START condition in master mode. Ignored in slave mode. In master receiver mode, a repeated START condition is preceded by a NACK. This is automatically cleared after START condition and address information is transmitted. Ignored in slave mode.
eRIC_I2CB_Stop();
Transmit STOP condition in master mode. Ignored in slave mode. In master receiver mode, the STOP condition is preceded by a NACK. This is automatically cleared after STOP is generated.
eRIC_I2CB_IsStartActive();
Check if start condition is active.
eRIC_I2CB_IsStopConditionOn();
Check if stop condition is active.
eRIC_I2CB_IsSCL_Low();
This is read only. Checks if SCL is low or high.
eRIC_I2CB_IsBusBusy();
THis is read only. Checks if bus is busy or inactive.

eRIC_I2CB_TxInteruptDisable();
This is used to disable Tx interrupt.
eRIC_I2CB_TxInteruptEnable();
This is used to enable Tx interrupt.
eRIC_I2CB_TxIsEnabled();
This is used to check if Tx interrupt is enabled
eRIC_I2CB_TxBufferIsEmpty();
This will return 1 if Tx buffer is empty.
eRIC_I2CB_TxBufferIsBusy();
This will return 1 if Tx buffer is busy

eRIC_I2CB_RxInteruptDisable();
This is used to disable Rx interrupt.
eRIC_I2CB_RxInteruptEnable();
This is used to enable Rx interrupt.
eRIC_I2CB_RxIsEnabled();
This is used to check if Rx interrupt is enabled
eRIC_I2CB_RxBufferIsEmpty();
This will return 1 when Rx buffer has received a character and is not empty.
eRIC_I2CB_RxBufferIsBusy();
This will return 1 when Rx buffer is empty.

Pinx_FunctionI2CB_SCl();
eRIC Pin can be assigned as SCL. X can by any available remappable pin.
Pinx_FunctionI2CB_SDA();
eRIC Pin can be assigned as SDA. X can by any available remappable pin.

Code Example:

Connect two eric . One operated as Slave and one as Master.Connect pin22 of two erics and Pin21 fo eric eric together. Use 10Kohms pullup on Pin22 and Pin21.

```
1) #include<cc430f5137.h>
2) #include "eRIC.h"
3) #include <stdio.h>
4) #include <string.h>
5) #define Address 0x0050  //THis is address for slave or own
6) #define Receiver
7) int main(void)
8) {
9)        eRIC_WDT_Stop();                    //Stop watch dog timer, just in
10)Case
11)       eRIC_GlobalInterruptDisable();     //Global interrupts disabled
12)       eROS_Initialise(0);       //Initialise eROS. CPU freq is 1048576
13)Default
14)       Pin22_FunctionI2CB_SDA(); //I2C Data
15)       Pin21_FunctionI2CB_SCl(); //I2C clock
16)#ifdef Receiver
17)       eRIC_I2CB_Initialise(100000,0,Address);// Initialise I2C with
18)100000Clock, Slave, and own address as 0x50
```

```
19)       Pin16_SetAsOutput();
20)       Pin17_SetAsOutput();
21)       Pin18_SetAsOutput();
22)       Pin19_SetAsOutput();
23)       eRIC_I2CB_AsReceiver();
24)       eRIC_I2CB_Start();//Start I2C
25)       char Temp = 0;
26)       while(1)
27)       {
28)        Temp = 0;
29)        Temp = eRIC_I2CB_ReceiveByte();// Return received byte which is the
30)data at the address
31)        if(Temp & 0x01)
32)              Pin16_SetHigh();
33)        Else
34)              Pin16_SetLow();
35)        if(Temp & 0x02)
36)              Pin17_SetHigh();
37)        Else
38)              Pin17_SetLow();
39)        if(Temp & 0x04)
40)              Pin18_SetHigh();
41)        Else
42)              Pin18_SetLow();
43)        if(Temp & 0x08)
44)              Pin19_SetHigh();
45)        Else
46)              Pin19_SetLow();
47)       }
48)#else//Transmitter
49)       eRIC_I2CB_Initialise(100000,1,Address);// Initialise I2C with
50)100000Clock, master, and slave address as 0x50
51)       Pin16_SetAsInput();
52)       Pin16_PullDownEnable();
53)       Pin17_SetAsInput();
54)       Pin17_PullDownEnable();
55)       Pin18_SetAsInput();
56)       Pin18_PullDownEnable();
57)       Pin19_SetAsInput();
58)       Pin19_PullDownEnable();
59)       eRIC_I2CB_AsTransmitter();
60)    char Temp = 0;
61)       while(1)
62)       {
63)             Temp = 0;
64)             if(Pin16_Read())
65)             {
66)                   Temp |=0x01;
67)             }
68)             if(Pin17_Read())
69)             {
70)                   Temp |=0x02;
71)             }
72)             if(Pin18_Read())
```

```
73)             {
74)                     Temp |=0x04;
75)           }
76)             if(Pin19_Read())
77)             {
78)                     Temp |=0x08;
79)             }
80)             eRIC_I2CB_Start();
81)             eRIC_I2CB_SendByte(Temp);//Send data to write at the address
82)Location
83)             while(eRIC_I2CB_TxBufferIsBusy());
84)             eRIC_I2CB_Stop();//Stop i2C
85)             while(eRIC_I2CB_IsBusBusy()); //wait while I2C is busy
86)       }
87)#endif
88)}
89)//Added in V1.1
90)void eRIC_RfDataReceivedInterrupt()   //V1.1 Add code here to deal with
91)available received data..THis is triggered when interrupt is enabled and a
92)packet is received
93){
94)}
```

Line1 and line2 includes cc430F5137 and eRIC.h which is must for any program code. Main starts at line7.

In line 5, it is the address of slave in master mode or address of own in slave mode which is 0x55.

Line 6 can be removed if the code is used as Master mode(tx).

Watchdogtimer is stoped in line9. Global interrupts are disabled in line11, any interrupts even radio interrupts in eros will be disabled.

eROS is initialised in line 12. Cpu frequency is set as 1048576Hz.

Pin22 is mapped as SDA and Pin21 is mapped as SCL in line 14 and 15.

Pin16 receiver slave code starts.

I2C is initialised as Slave with 100k clock and address as 0x55 in line 17.

Pin16-19 are set as output In line91-22.

In line 23 and 24, I2c is set as receiver and started.

In Line 29 I2C continuously receives data from master.

From line 30- 47, Pin16-19 are set high or low depending on the received byte from Master. If 01 is received Pin16 is high, if 0x02 pin17 is high, if 0x04 pin18 is high ,if 0x08 pin19 is high.

In line 48 Master code is written.

In line 49 I2C is initialised as master with 100k clock and address 0x50.

From line 51-58, Pin16-19 are set as inputs and pull down enabled.

In line 59 I2C is set as transmitter.

From 61-79, it continuously checks Pin16-19 input state. Temp is initialised as 0 and If pin16 is pressed, 0x01 is added to Temp, ifpin17 is high 0x02 is added, if pin18 is high 0x04 is added and if pin19 is high 0x08 is added.

From 80-85, I2C is started and Temp data is sent and I2C is stopped after sending.

The code above is either Receiver or Transmitter and it can be compiled by removing receiver definition in line 6.

eRIC_RfDataReceivedInterrupt() is copied from eRIC.c which is removed and pasted in main at line 89-94. This has no effect in this example as there is no receiver enabled. But this code is needed for compiler to compile without error or un-remove this same code in eRIC.c.