PIC32MM I2C BIT BANG CODE EXAMPLE

Overview

This project allows adding I2C interface for PIC32 devices without I2C hardware module. The code supports master and slave modes. The master side code requires about 660 bytes of flash and doesn't use RAM. At 24 MIPS the master can work with the I2C clock up to 1 MHz. The slave code requires 720 bytes of flash and 8 bytes of RAM. At 24 MIPS the slave I2C clock can be up to 150 kHz.

I2C MASTER

Files

The files in table below must be added to the application project to implement master I2C interface.

File	Description
i2c_master.h	This header contains I/O definitions/selection and timing/clock speed settings used for the MASTER I2C interface. Also this file includes prototypes of MASTER I2C functions.
i2c_master.S	This assembler source file contains MASTER I2C functions implementations.

The library package includes demo projects.

The demo project located in "master_demo" folder shows how to use the I2C master bit bang functions to access EEPROM 24FC256. The files related to this demo are:

File	Description
i2c_master_eeprom_24fc256.h	This header contains prototypes of functions to access 24FC256 EEPROM.
i2c_master_eeprom_24fc256.c	This C source file contains functions implementations to access 24FC256 EEPROM.
master_demo_main.c	The main demonstration source file. It contains code to initialize I2C interface and write/read values to/from 24FC256 EEPROM.

Library Settings

The library settings are separate for slave and master sides. The following parameters must be configured for **I2C MASTER INTERFACE** in **i2c_master.h** header:

Parameter	**Description**
I2C_CLOCK_DELAY	This parameter is a quarter of I2C clock period in instruction cycles. It defines timing for I2C interface. I2C MAY NOT WORK IF THIS PARAMETER IS WRONG

	(CLOCK IS TOO FAST).
SCL_MASK	This parameter sets the bit position of I/O used for SCL signal.
SCL_TRIS_CLR	This parameter sets the TRIS CLEAR register of I/O used for SCL signal. VERIFY THAT IN YOUR APPLICATION THE SCL PIN IS CONFIGURED AS A DIGITAL INPUT IN ANSEL REGISTER.
SCL_ODC_SET	This parameter sets the ODC SET register of I/O used for SCL signal.
SCL_LAT_CLR	This parameter sets the LAT CLR register of I/O used for SCL signal.
SCL_LAT_SET	This parameter sets the LAT SET register of I/O used for SCL signal.
SCL_PORT	This parameter sets the PORT register of I/O used for SCL signal.
SDA_MASK	This parameter sets the bit position of I/O used for SDA signal.
SDA_TRIS_CLR	This parameter sets the TRIS CLEAR register of I/O used for SDA signal. VERIFY THAT IN YOUR APPLICATION THE SDA PIN IS CONFIGURED AS A DIGITAL INPUT IN ANSEL REGISTER.
SDA_TRIS_SET	This parameter sets the ODC SET register of I/O used for SDA signal.
SDA_ODC_SET	This parameter sets the LAT CLR register of I/O used for SDA signal.
SDA_LAT_CLR	This parameter sets the LAT SET register of I/O used for SDA signal.
SDA_LAT_SET	This parameter sets the PORT register of I/O used for SDA signal.
SDA_PORT	This parameter sets the bit position of I/O used for SDA signal.

The library doesn’t set SCL and SDA pins input type. It must be done in the application. SCL and SDA must be configured as digital inputs using ANSELx registers.

Functions and Macros

void I2CM_Init()

Description:

This function initializes SDA and SCL I/Os.

Parameters:

None.

Returned data:

None.

long I2CM_Start()

Description:

This function generates an I2C start signal.

Parameters:

None.

Returned data:

The function returns non-zero value if the bus collision is detected.

long I2CM_Stop()

Description:

This function generates I2C stop signal.

Parameters:

None.

Returned data:

The function returns non-zero value if the bus collision is detected.

long I2CM_Write(unsigned char data)

Description:

This function transmits 8-bit data to slave.

Parameters:

unsigned char data – data to be transmitted

Returned data:

This function returns acknowledgment from slave (0 means ACK and 1 means NACK).

unsigned char I2CM_Read(long ack)

Description:

This function reads 8-bit data from slave.

Parameters:

long ack – acknowledgment to be sent to slave

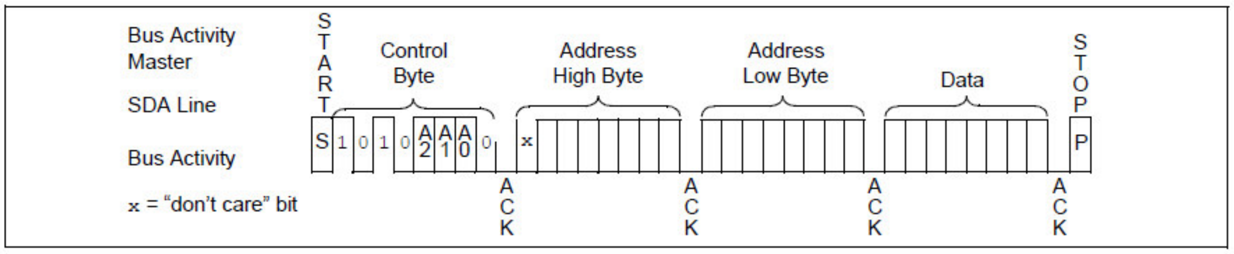
Returned data:

This function returns 8-bit data read.

Getting Started

Let's assume that PIC32 device must communicate with 24FC256 EEPROM.

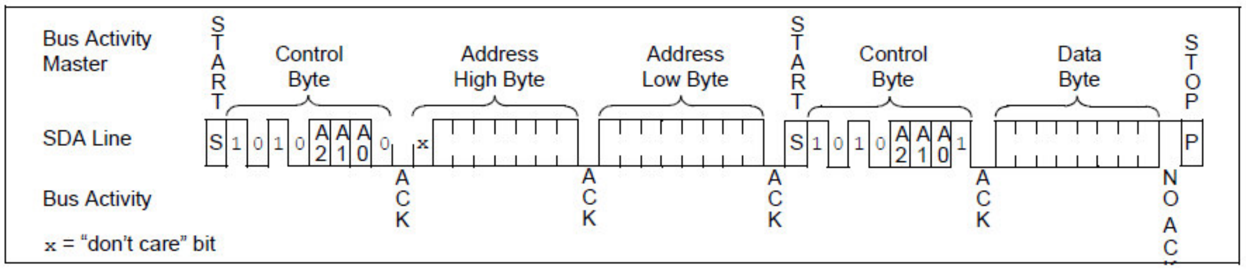
The following signals should be generated to write a byte:



It can be done by the following functions calls:

```
I2CM_Start();

I2CM_Write(0xA0);

I2CM_Write(address>>8);

I2CM_Write(address&0x00FF);

I2CM_Write(data);

I2CM_Stop();
```

To read a byte from EEPROM the following signals should be generated:



It can be done by the following functions calls:

```
I2CM_Start();

I2CM_Write(0xA0);

I2CM_Write(address>>8);

I2CM_Write(address&0x00FF);
```

```
I2CM_Start();

I2CM_Write(0xA1); // READ

data = I2CM_Read(1); // NO ACKNOWLEDGE

I2CM_Stop();
```

I2C SLAVE

Files

The files in table below must be added to the application project to implement slave I2C interface.

File	Description
i2c_slave.h	This header contains I/O definitions/selection used for the SLAVE I2C interface. Also this file includes prototypes of SLAVE I2C functions.
i2c_slave.c	This C source file contains SLAVE I2C functions implementations.

The library package includes demo projects.

The demo project located in "slave_demo" folder shows how to use the I2C slave bit bang functions to emulate EEPROM 24FC256. The files related to this demo are:

File	Description
i2c_slave_eeprom_24fc256.c	This C source file contains I2C **callback functions** implementations to emulate 24FC256 EEPROM.
slave_demo_main.c	The main demonstration source file. The code in this file initializes I2C slave interface and runs I2C task.

Library Settings

The following parameters must be configured for **I2C SLAVE INTERFACE** in **i2c_slave.h** header:

Parameter	**Description**
SCL_MASK	This parameter sets the bit position of I/O used for SCL signal.
SCL_TRIS_CLR	This parameter sets the TRIS CLEAR register of I/O used for SCL signal. VERIFY THAT IN YOUR APPLICATION THE SCL PIN IS CONFIGURED AS A DIGITAL INPUT IN ANSEL REGISTER.
SCL_TRIS_SET	This parameter sets the TRIS SET register of I/O used for SCL signal.
SCL_ODC_SET	This parameter sets the ODC SET register of I/O used for SCL signal.

SCL_LAT_CLR	This parameter sets the LAT CLR register of I/O used for SCL signal.
SCL_LAT_SET	This parameter sets the LAT SET register of I/O used for SCL signal.
SCL_PORT	This parameter sets the PORT register of I/O used for SCL signal.
SDA_MASK	This parameter sets the bit position of I/O used for SDA signal.
SDA_TRIS_CLR	This parameter sets the TRIS CLEAR register of I/O used for SDA signal. VERIFY THAT IN YOUR APPLICATION THE SDA PIN IS CONFIGURED AS A DIGITAL INPUT IN ANSEL REGISTER.
SDA_TRIS_SET	This parameter sets the ODC SET register of I/O used for SDA signal.
SDA_ODC_SET	This parameter sets the LAT CLR register of I/O used for SDA signal.
SDA_LAT_CLR	This parameter sets the LAT SET register of I/O used for SDA signal.
SDA_LAT_SET	This parameter sets the PORT register of I/O used for SDA signal.
SDA_PORT	This parameter sets the bit position of I/O used for SDA signal.
I2C_DISABLE_CLOCK_STRETCHING	Add/uncomment the definition of this parameter to disable clock stretching.

The library doesn't set SCL and SDA pins input type. It must be done in the application. SCL and SDA must be configured as digital inputs using ANSELx registers.

Functions and Macros

void I2CS_Init()

Description:

This function initializes SDA and SCL I/Os.

Parameters:

None.

Returned data:

None.

void I2CS_Task()

Description:

This function is an engine to process signals on SDA and SCL I/Os. If some I2C event will be detected this function will pass control to the corresponding CALLBACK FUNCTION implemented in the user's code. The following callback functions are called:

EVENT	CALLBACK FUNCTION
I2C START	I2CS_Start()
I2C STOP	I2CS_Stop()
BYTE RECEVED FROM MASTER	I2CS_Read(…)
REQUEST TO TRANSMIT A BYTE TO MASTER	I2CS_Write(…)

The I2CS_Task() function must be executed periodically. It can be done by:

1. Change Notification interrupts on both SCL and SDA I/Os (the interrupts must detect positive and negative edges/transitions)
2. calling it in main idle loop
3. timer interrupt

Parameters:

None.

Returned data:

None.

void I2CM_Start()

Description:

This is a CALL BACK function controlled by I2CS_Task() function. If it is implemented in the application it will be called each time when I2C start signal is detected.

Parameters:

None.

Returned data:

None.

void I2CM_Stop()

Description:

This is a CALL BACK function controlled by I2CS_Task() function. If it is implemented in the application it will be called each time when I2C stop signal is detected.

Parameters:

None.

Returned data:

None.

long I2CS_Read(unsigned char data)

Description:

This is a CALL BACK function controlled by I2CS_Task() function. If it is implemented in the application it will be called each time when 8-bit data are received from I2C master.

Parameters:

unsigned char data – data received from I2C master.

Returned data:

Return/Pass the Acknowledgment (bit #0) and Write Mode (bit #1) flags to the library. If the data received must be acknowledged then clear bit #0. For NACK return one in bit #0. If for the next transaction the I2C slave must transmit data to master then return/pass one in bit #1. If for the next transaction the I2C slave must still receive the data from master then clear bit #1.

unsigned char I2CS_Write(long prev_ack)

Description:

This is a CALL BACK function controlled by I2CS_Task() function. If it is implemented in the application it will be called each time when I2C master will request 8-bit data from slave.

Parameters:

long prev_ack – acknowledgment for the previous transaction. In most cases if the master answered with NACK (=1) before, the new data are not required and master will generate a stop event soon.

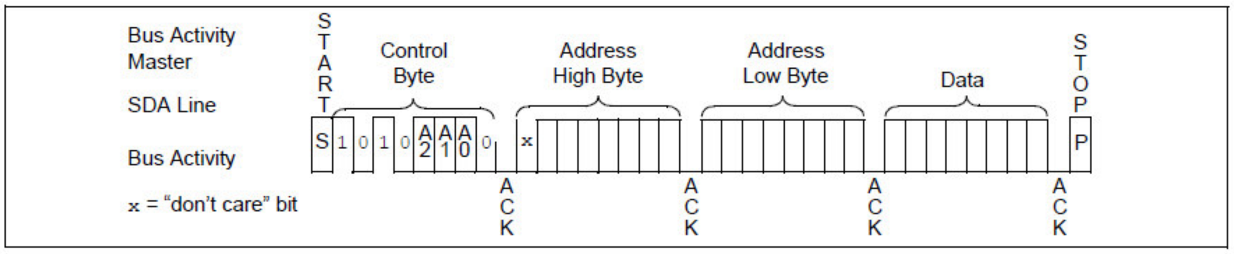
Returned data:

Return/Pass the 8-bit data to be transmitted to I2C master.

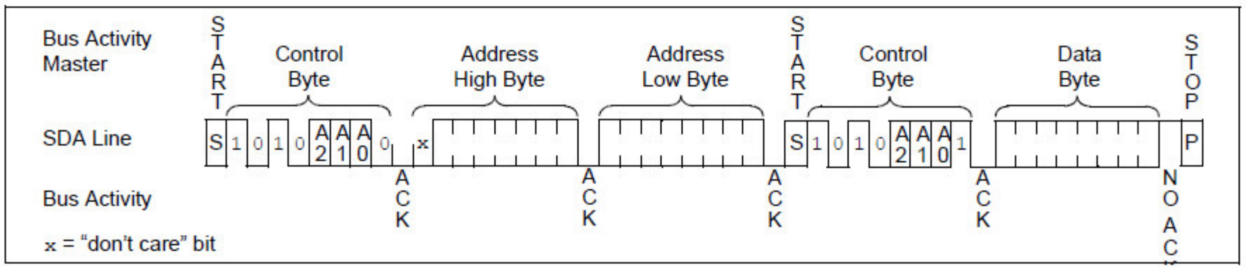
Getting Started

Let's assume that PIC32 device must emulate 24FC256 EEPROM.

The following signals will be generated by master to write a byte:



and to read a byte:



To emulate this protocol the I2C slave must:

1. Detect a start condition using I2CS_Start() callback function.
2. Receive the first byte after start and decode DEVICE address and read RW bit (bit #0) in the first byte.
3. If DEVICE address matches the required address and next byte will be read (RW bit = 0) then read 2 address bytes and store data byte received using I2CS_Read(…) callback function.
4. If RW bit = 1 the slave must transmit data to master using I2CS_Write(…) callback function.

It can be done if the following callback functions are implemented:

```
// 24FC256 COMMUNICATION PROTOCOL STATES
typedef enum {
    STATE_DEV_ADDRESS, // device address will be received
    STATE_ADDRESS_HIGH_BYTE, // high byte of memory address will be recieved
    STATE_ADDRESS_LOW_BYTE, // low byte of memory address will be recieved
    STATE_DATA_READ, // data byte will be read from master
    STATE_DATA_WRITE // data byte will be sent to master
} I2C_STATE;

// current state
I2C_STATE state = STATE_DEV_ADDRESS;

// 24FC256 device 7-bit address
#define EEPROM_DEV_ADDRESS     0x50  // from 24FC256 datasheet
#define EEPROM_SIZE            256   // 256 bytes

unsigned char eeprom_data[EEPROM_SIZE]; // memory storage
long          eeprom_address = 0; // current memory address

// This callback function is called every time when I2C start is detected
void I2CS_Start(){
    state = STATE_DEV_ADDRESS; // after start the device address byte will be
transmitted
}

// This callback function is called every time when data from I2C master are
received
long I2CS_Read(unsigned char data){

    switch(state){
        case STATE_DEV_ADDRESS:
            if((data >> 1) == EEPROM_DEV_ADDRESS){ // bits from #7 to #1 are
device address
                if(data&1){ // if bit #0 is set (=1) it indicates that the
next data go from slave to master
                    state = STATE_DATA_WRITE;
                    return 2; // ACK to master (bit #0 = 0), master reads
data on next transaction (bit #1 = 1)
                }else{ // if bit #0 is cleared (=0) it indicates that the
next data go from master to slave
                    state = STATE_ADDRESS_HIGH_BYTE;
                    return 0; // ACK to master
                }
            }
            return 1; // NACK to master if device address doesn't match
EEPROM_DEV_ADDRESS
        case STATE_ADDRESS_HIGH_BYTE:
            state = STATE_ADDRESS_LOW_BYTE;
            eeprom_address = (data<<8);
            return 0; // ACK to master
        case STATE_ADDRESS_LOW_BYTE:
            state = STATE_DATA_READ;
            eeprom_address |= data;
            return 0; // ACK to master
        case STATE_DATA_READ:
            state = STATE_DEV_ADDRESS;
            if(eeprom_address >= EEPROM_SIZE){
                return 1;  // NACK to master, the memory address is wrong
            }
            eeprom_data[eeprom_address] = data; // store the data received
            return 0; // ACK to master
        default:
                  state = STATE_DEV_ADDRESS;
            return 1; // NACK to master / unknown state
    }

    return 1; // NACK to master
}

// This callback function is called every time when data must be sent to I2C
master
unsigned char I2CS_Write(long prev_ack){
    if(eeprom_address >= EEPROM_SIZE)
    {
        return 0;
    }
    return eeprom_data[eeprom_address]; // send memory data to master
}

void main()
{
    I2CS_Init();

    while(1){
        I2CS_Task();
    }
}
```