Version v0.9b

**The Challenge**

Implementing ‘legacy mode’ methods using the new I2C module such as the 18F25K42 microcontroller called ‘K mode’ [after the K42}.

Simply to send the following via I2C.

I2CStart ‘Start of i2c packet

I2CSend ( 0x78 ) ‘slave address

I2CSend ( 0x51 ) ‘send data

I2CSend ( 0x52 )

I2CSend ( 0x53 )

I2CSend ( 0x54 )

I2CSend ( 0x55 )

I2CSend ( 0x56 )

I2CSend ( 0x57 )

I2CSend ( 0xFF )

I2CStop ‘End of i2c packet

The ‘K mode’ I2C module implementation does not have reference code for the method of implementation shown above.

The big difference between ‘legacy mode’ and ‘k mode’ is not able to use the assumed ‘k mode’ methods of i2c1\_write1ByteRegister, i2c1\_writeNBytes or i2c1\_write2ByteRegister. This ‘legacy mode’ methods assume start(), send(address), send(data)…. send(data\_of\_unknownlength), stop().

We need a reference implementation to support ‘legacy mode’.

See section **Current implementation Issues** for the latest state of the solution.

Evan R. Venn

Contents

[Background 3](#_Toc495915410)

[A review of Master Mode in ‘legacy mode’ 4](#_Toc495915411)

[Implementation of ‘Legacy Mode’ 5](#_Toc495915412)

[Usage of ‘legacy mode’ 6](#_Toc495915413)

[An Implementation of ‘legacy mode’ 7](#_Toc495915414)

[Setting the Ports in ‘Legacy Mode’ 11](#_Toc495915415)

[A review of Master Mode in ‘K mode’ that interoperates with ‘Legacy mode’ 12](#_Toc495915416)

[Implementation of ‘K Mode’ mode’ that interoperates with ‘Legacy mode’ 13](#_Toc495915417)

[An Implementation of ‘K mode’ that interoperates with ‘Legacy mode’ 16](#_Toc495915418)

[Setting the Ports in ‘K Mode’ 21](#_Toc495915419)

[Current implementation Issues 22](#_Toc495915420)

[Operation ASM and HEX 24](#_Toc495915421)

# Background

There is a need to determine the methods to be used to support ‘legacy mode’ I2C mode where users have created libraries to communicate with I2C devices such as I2C Memory, I2C clocks, I2C GLCD displays etc. for the ‘K mode’ microcontrollers.

This document has three sections. A review of the ‘legacy mode’, a review of the implementation of ‘K mode’ and a current status of solutioning on a real 18F25K42.

This document initially focuses on Master Mode/Sending Data. Master Mode/Receiving Data will be address in a later revision of the document.

# A review of Master Mode in ‘legacy mode’

‘Legacy mode’ Master mode is enabled by setting and clearing the appropriate SSPM bits in SSPCON and by setting the SSPEN bit. In Master mode, the SCL and SDA lines are manipulated by the MSSP hardware. Master mode of operation is supported by interrupt generation on the detection of the Start and Stop conditions. The Stop (P) and Start (S) bits are cleared from a Reset or when the MSSP module is disabled. Control of the I2C bus may be taken when the P bit is set or the bus is Idle, with both the S and P bits clear. In Firmware Controlled Master mode, user code conducts all I2C bus operations based on **Start** and **Stop** bit conditions. Once Master mode is enabled, and the ports et, the user has six options.

1. Set the port up
2. Assert a Start condition on SDA and SCL.
3. Assert a Repeated Start condition on SDA and SCL.
4. Write to the SSPBUF register, initiating transmission of data/address.
5. Configure the I2C port to receive data.
6. Generate an Acknowledge condition at the end of a received byte of data.
7. Generate a Stop condition on SDA and SCL.

A baud rate used for the I2C ‘legacy mode’ operation and this baud rate is used to set the SCL clock frequency for either 100 kHz, 400 kHz or 1 MHz I2C operation.

## Implementation of ‘Legacy Mode’

Implementing the ‘legacy mode’ was relatively simple. An eight step process – this not show error handling.

1. Set the port up
2. Call a method called **I2CStart()**. This is sets SEN
3. If required, assert a Repeated Start condition on SDA and SCL by calling a method called **I2CReStart()**. This is sets RSEN
4. Call a method called **I2CSend( outbyte)**, this write the output data to the to the SSPBUF register, which will initiating transmission of data/address.
5. Call **I2CReceive( outbyte, ACK | NACK )** to receive data to the SSPBUF register.
6. Within the **I2CReceive** method, generate an Acknowledge condition at the end of a received byte of data, setting ACKDT to 0|1 and set ACKEN.
7. Call **I2CStop()** to generate a Stop condition on SDA and SCL by setting PEN.

Overview of ‘Legacy Mode’ methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Usage** | **Parameters** | **Command** |
| I2CStart | Assert a Start condition on SDA and SCL. | None | SEN |
| I2CReStart | Assert a Restart condition on SDA and SCL. | None | RSEN |
| I2CSend | Send data to slave device. The data sent can be a byte only and may be an address or data | Byte\_data\_out.  Can be an Address or Data | SSPBUF |
| I2CReceive | Receive data from slave | Byte\_data\_in, ACK | NACK  ACK or NACK depends on slaves device handler for end of sending data. | SSPBUF |
| I2CStop | Assert a Stop condition on SDA and SCL. | None | PEN |

## Usage of ‘legacy mode’

Baud Rate

The calculation of Baud Rate is relatively simple.

I2C\_BAUD\_TEMP = int((ChipMhz \* 1000000)/(4000 \* DESIRED\_I2C\_BAUD\_RATE)) – 1

The assignment of the Baud Rate is also simple, using the constant I2C\_BAUD\_TEMP

SSPADD = I2C\_BAUD\_TEMP And 127

Methods

The Start(), ReStart() and Stop() are essentially to control the lines. I2CSend has one parameter, the I2C protocol determines the first byte to be an address after a Start() or ReStart), then data is either sent or received.

It is important to note the methods do not need to know the size of the data to be sent or the I2C slave register to set or read. This device specific library handles the specifics of the I2C device.

## An Implementation of ‘legacy mode’

'Set of constants and methods to support the EMC1001 on the MicroChip Xpress Board.

'Default address

#define EMC1001\_ADDRESS 0x70 ' slave device address

' EMC1001 registers

#define EMC1001\_TEMP\_HI 0 ' temperature value high byte

#define EMC1001\_TEMP\_STATUS 1 ' Status

#define EMC1001\_TEMP\_LO 2 ' low byte containing 1/4 deg fraction

#define EMC1001\_TEMP\_CONFIG 3 ' configuration

#define EMC1001\_CONV\_RATE 4 ' conversation rate

#define EMC1001\_TEMP\_LIMIT\_HI\_H 5 ' temp high limit high byte

#define EMC1001\_TEMP\_LIMIT\_HI\_L 6 ' temp high limit low byte

#define EMC1001\_TEMP\_LIMIT\_LO\_H 7 ' temp low limit high byte

#define EMC1001\_TEMP\_LIMIT\_LO\_L 8 ' temp low limit low byte

#define EMC1001\_TEMP\_ONE\_SHOT 0x0F ' temp high limit high byte

#define EMC1001\_TEMP\_THERM\_LIMIT 0x20 ' THERM limit

#define EMC1001\_TEMP\_THERM\_HYSTERIS 0x21 ' THERM hysteris

#define EMC1001\_TEMP\_SMBus\_Timeout 0x22 ' SMBus timeout

#define EMC1001\_TEMP\_PROD\_ID 0xFD ' prod ID

#define EMC1001\_TEMP\_MANUFACTURE\_ID 0xFE ' manufacturing ID

#define EMC1001\_TEMP\_REV\_NUMBER 0xFF ' revision number

'Return the read data as the second parameter.

'The first being... the register address

method **EMC1001\_Read**( in \_emc\_reg, out \_emc\_Data ) {

I2CStart

I2CSend ( EMC1001\_ADDRESS )

I2CSend ( \_emc\_reg )

I2CReStart

I2CSend( EMC1001\_ADDRESS + 1 ) ;set the read flag

I2CReceive( \_emc\_Data, NACK ) ;read one byte and conclude

I2CStop

}

'Write the byte data as the second parameter.

'The first being... the register address

method **EMC1001\_Write**( in \_emc\_reg, in \_emc\_Data ) {

I2CStart

I2CSend ( EMC1001\_ADDRESS )

I2CSend ( emc\_reg )

I2CSend( \_emc\_Data ) ; Send the data

I2CStop

}

‘PUBLIC LIBRARIES

Method I2CStart {

'Master mode

Set SEN On

I2CWaitMSSP ‘error handler

}

method I2CReStart {

'Master mode

Set RSEN On

I2CWaitMSSP ‘error handler

}

method I2CSend ( in IC2Byte ) {

SSPCON1.WCOL = 0

'Load data to send

SSPBUF = I2CByte

I2CWaitMSSP

if ACKSTAT = 1 then ‘ determine ackstat status and

I2CAckPollState = true

else

HI2CAckPollState = false

end if

If SSPCON1.WCOL = On Then

If HI2CCurrentMode <= 10 Then Goto RetryHI2CSend

End If

}

Method HI2CReceive (Out I2CByte, Optional In HI2CGetAck = 1 ) {

'Enable receive for legacy I2C

'Master mode

If HI2CCurrentMode > 10 Then

if HI2CGetAck.0 = 1 then

'Acknowledge

ACKDT = 0

else

'Not Acknowledge

ACKDT = 1

end if

RCEN = 1

End If

'Clear Collisions

SET SSPCON1.WCOL OFF

SET SSPCON1.SSPOV Off

'Wait for receive

Wait Until SSPSTAT.BF = 1 AND SSPIF = 1

I2CByte = SSPBUF

SSPIF = 0 'Support for SSPIF

ACKEN = 1; Send ACK DATA now

' Clear flag - this is required

SSPSTAT.BF = 0

HI2CWaitMSSP ‘ error handler

'Master mode

SSPCON2.RCEN = 0

}

Method I2CStop {

wait while R\_NOT\_W = 1 'wait for completion of activities

Set SSPCON2.PEN On

HI2CWaitMSSP ‘error handler

#endif

}

## Setting the Ports in ‘Legacy Mode’

Setting the ports requires the SCL and SDA pins to be open-drain, and the these pins **MUST** programmed as **INPUTS** by setting the appropriate TRISC bits. And, to ensure proper operation of the module, pull-up resistors must be provided externally to the SCL and SDA pins.

# A review of Master Mode in ‘K mode’ that interoperates with ‘Legacy mode’

‘K mode’ Master mode is enabled by setting and clearing the appropriate bits I2C1CON0.

In Master mode, the SCL and SDA lines are manipulated by the I2C hardware.

Master mode of operation is supported by interrupt generation on the detection of the Start and Stop conditions.

1. Set the ports up
2. Assert a Start condition on SDA and SCL.
3. Assert a Repeated Start condition on SDA and SCL.
4. Write data to the output register, initiating transmission of data/address.
5. Configure the I2C port to receive data.
6. Generate an Acknowledge condition at the end of a received byte of data.
7. Generate a Stop condition on SDA and SCL.

A register is used for I2C baud rate control in ‘K mode’ operation. The baud rate is used to set the SCL clock.

# Implementation of ‘K Mode’ mode’ that interoperates with ‘Legacy mode’

Implementing the ‘K mode’ that interoperates with ‘Legacy mode’requires the user to define a state engine. An eight step process – this not show error handling.

1. Set the ports up and set StateEngine = 0, see below of registers.
2. Call a method called **I2CStart()**. This is sets the StateEngine = 0
3. If required, assert a Repeated Start condition on SDA and SCL by calling a method called **I2CReStart()**. And, set the StateEngine to 3
4. Call a method called **I2CSend( outbyte)**, this method handles the state engine as follows: State handler as follows.
5. StateEngine = 1: Clear output buffer, set the buffer output count to 0, clear interrupts and load the Address to the Address buffer, wait while I2C1PIR.SCIF = 0, set I2C1CON0.RSEN=1 and set I2C1CON0.S = 1. Finally, set state I2C StateEngine = 2
6. StateEngine = 2. This writes the output data to the to the ouptut buffer register, which will initiating transmission of data/address by waiting while I2C1STAT1.TXBE= 0, set I2C1CNT = 1 and load the output buffer I2C1TXB = I2Coutbyte
7. StateEngine = 3: Clear output buffer, set the buffer output count to 0, clear interrupts and load the Address to the Address buffer, wait while I2C1PIR.SCIF = 0, set I2C1CON0.RSEN=1 and set I2C1CON0.S = 1. Finally, set state I2C StateEngine = 2
8. ~~Call~~ **~~I2CReceive( outbyte, ACK | NACK )~~** ~~….NO IDEA. I have not got this far!~~
9. Call **I2CStop()** to generate a Stop condition on SDA and SCL by waiting for I2C1PIR.PCIF = 1

Overview of ‘K Mode’ methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Usage** | **Parameters** | **Command** |
| I2CStart | Assert a Start condition on SDA and SCL. | None | Sets state engine = 2 |
| I2CReStart | Assert a Restart condition on SDA and SCL. | None | Sets state engine = 2 |
| I2CSend | Send data to slave device. The data sent can be a byte only and may be an address or data | Byte\_data\_out.  Can be an Address or Data | Handles state engine |
| ~~I2CReceive~~ | ~~Receive data from slave~~ | ~~Byte\_data\_in, ACK | NACK~~  ~~ACK or NACK depends on slaves device handler for end of sending data.~~ |  |
| I2CStop | Assert a Stop condition on SDA and SCL. | None | Set StateEngine = 0 |

**Registers**

Port Registers need to be set to enable this mode of operation, this assumes PPS has been set, ports are set as **OUTPUTS** and the logic levels are set., as shown below in pseudo code:

RC4PPS = 0x22

RC3PPS = 0x21; ;RC3->I2C1:SCL1;

I2C1SCLPPS = 0x13; ;RC3->I2C1:SCL1;

I2C1SDAPPS = 0x14; ;RC4->I2C1:SDA1;

#define SI2C\_DATA PORTC.4

#define SI2C\_CLOCK PORTC.3

Dir SI2C\_DATA out

Dir SI2C\_CLOCK out

RC3I2C.TH0=1 ‘bsf RC3I2C,TH0,BANKED

RC4I2C.TH0=1 ‘bsf RC4I2C,TH0,BANKED

I2C Registers need to be set to enable this mode of operation as shown below in pseudo code:

I2C1CON1 = 0x80;

I2C1CON2 = 0x21

I2C1CLK = 0x03

I2C1CON0 = 0x04;

I2C1PIR = 0 ;Clear all the error flags

I2C1ERR = 0

I2C1CON0.EN=1

I2C1CON2.ACNT = 0

I2C1CON2.ABD=0

I2C1CON0.MDR=1

## An Implementation of ‘K mode’ that interoperates with ‘Legacy mode’

I2CStart

I2CSend ( 0x78 )

I2CSend ( 0x51 )

I2CSend ( 0x52 )

I2CSend ( 0x53 )

I2CSend ( 0x54 )

I2CSend ( 0x55 )

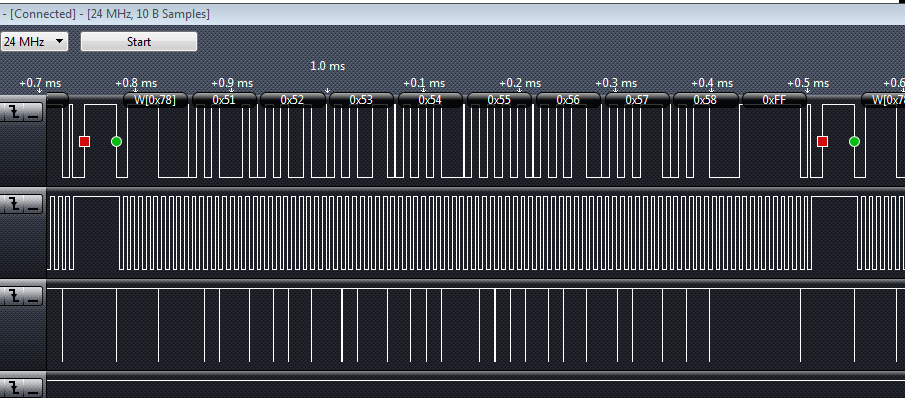
I2CSend ( 0x56 )

I2CSend ( 0x57 )

I2CSend ( 0xFF )

I2CStop

Generates as transmission as shown below. This is a correct package transmission. However, the methods require validation, see Current implementation Issues.



**ASM**

I2CSTART

movlw 1

movwf HI2C10\_STARTOCCURRED,BANKED

return

SI2CSTOP

;HI2C10\_StartOccurred = 0

clrf HI2C10\_STARTOCCURRED,BANKED

;HI2CWaitMSSPTimeout = 0

clrf HI2CWAITMSSPTIMEOUT,BANKED

;do while HI2CWaitMSSPTimeout < 255

SysDoLoop\_S2

movlw 255

banksel HI2CWAITMSSPTIMEOUT

subwf HI2CWAITMSSPTIMEOUT,W,BANKED

btfsc STATUS, C,ACCESS

bra SysDoLoop\_E2

;HI2CWaitMSSPTimeout++

incf HI2CWAITMSSPTIMEOUT,F,BANKED

;if I2C1PIR.PCIF = 1 then

banksel I2C1PIR

btfss I2C1PIR,PCIF,BANKED

bra ELSE1\_1

;wait 20 us

movlw 6

movwf DELAYTEMP,ACCESS

DelayUS1

decfsz DELAYTEMP,F,ACCESS

bra DelayUS1

nop

;exit sub

banksel 0

return

;else

bra ENDIF1

ELSE1\_1

;wait 1 us

nop

;end if

ENDIF1

;loop

bra SysDoLoop\_S2

SysDoLoop\_E2

Return

I2CSEND

;Select Case HI2C10\_StartOccurred

;case 2 'send data

SysSelect1Case1

movlw 2

subwf HI2C10\_STARTOCCURRED,W,BANKED

btfss STATUS, Z,ACCESS

bra SysSelect1Case2

;wait while I2C1STAT1.TXBE = 0

SysWaitLoop1

banksel I2C1STAT1

btfss I2C1STAT1,TXBE,BANKED

bra SysWaitLoop1

;I2C1CNT = 1

movlw 1

movwf I2C1CNT,BANKED

;I2C1TXB = I2Cbyte

movffl I2CBYTE,I2C1TXB

;exit sub

banksel 0

return

;case 1 'A start

bra SysSelectEnd1

SysSelect1Case2

decf HI2C10\_STARTOCCURRED,W,BANKED

btfss STATUS, Z,ACCESS

bra SysSelect1Case3

;I2C1STAT1.CLRBF = 1

banksel I2C1STAT1

bsf I2C1STAT1,CLRBF,BANKED

;I2C1CNT = 0

clrf I2C1CNT,BANKED

;I2C1PIR.SCIF = 0

bcf I2C1PIR,SCIF,BANKED

;I2C1ADB1 = I2Cbyte

movffl I2CBYTE,I2C1ADB1

;do while I2C1PIR.SCIF = 0

SysDoLoop\_S3

btfsc I2C1PIR,SCIF,BANKED

bra SysDoLoop\_E3

;I2C1CON0.S = 1

bsf I2C1CON0,S,BANKED

;loop

bra SysDoLoop\_S3

SysDoLoop\_E3

;HI2C10\_StartOccurred = 2 'Set state engine to send data

movlw 2

banksel HI2C10\_STARTOCCURRED

movwf HI2C10\_STARTOCCURRED,BANKED

;case 3 'A restart. TBD

bra SysSelectEnd1

SysSelect1Case3

movlw 3

subwf HI2C10\_STARTOCCURRED,W,BANKED

btfss STATUS, Z,ACCESS

bra SysSelectEnd1

;end select

SysSelectEnd1

return

## Setting the Ports in ‘K Mode’

The I2C module provides a synchronous interface between the microcontroller and other I2C-compatible devices using the two-wire I2C serial bus. Devices communicate in a master/slave environment. The I2C bus specifies two signal connections which are bidirectional open-drain lines, each requiring pull-up resistors to the supply voltage.

Specific pins are configured for I2C and SMB™ 3.0/2.0 logic levels. The SCLx/SDAx signals may be assigned to any of the RB1/RB2/RC3/RC4 pins. PPS assignments to the other pins (e.g., RA5) will operate, but, input logic levels will be standard TTL/ST as selected by the INLVL register, instead of the I2C specific or SMBUS input buffer thresholds.

I/O pins assigned to these signals must be configured as such through the ANSELx and ODCONx registers, and PPS, respectively. The PPS enables these pins to function as peripheral inputs (I2CxSCLPPS and I2CxSDAPPS) and outputs (RxyPPS). Input threshold, slew rate and internal pull-up settings are configured in the RxyI2C Control registers.

Essentially, setting the ports requires the SCL and SDA pins to be open-drain, and the these pins **MUST** programmed as **OUTPUTS** by setting the appropriate TRISC bits which is very different from the ‘legacy mode’.

## Current implementation Issues

The following implementation issues exist.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Item | Added in Doc Level | Issue | Query | Closure Note | Status Open/Closed |
| 1 | 0.9a | What is the correct Start of I2c transaction? | Is this correct?  I2C1STAT1.CLRBF = 1 ‘clear buffer  I2C1CNT = 0 ‘set to count to 0  I2C1PIR.SCIF = 0 ‘ clear int  I2C1ADB1 = I2Cbyte ‘set the slave address  do while I2C1PIR.SCIF = 0 ‘wait…  I2C1CON0.S = 1 ‘set Start  loop | SCIF doesn’t clear itself as it states in the datasheet (HS = Hardware set and only this)  You should wait until I2CCON0.S is cleared because it’s cleared by HW  But the sequence should be correct. | Open |
| 2 | 0.9a | When sending data is it correct to set the CNT to 1 when I2C1CON2.ACNT=0 ? | Is this correct?  wait while I2C1STAT1.TXBE = 0  I2C1CNT = 1  I2C1TXB = I2Cbyt | Before Start, software loads one byte in I2CxTXB. You can use also I2CxTXIF as it describes at page 570.  If you want to only write 1 byte, then CNT=1. | Open |
| 3 | 0.9a | What is the correct Stop? | Is this correct?  Wait until I2C1PIR.PCIF = 1 | The HW will assert a stop automatically when I2CCNT=0; You will then have to test if the PCIF is set then wait for 2xTSCL to be sure.  So it should be like below | Open |
| 4 | 0.9a | Within Stop method. 1) Why is a delay required? If I DO NOT have a 20us delay the last byte is not transmitted  2) Is this delay frequency dependent? | Like…  if I2C1PIR.PCIF = 1 then  **wait 70 us**  exit method  end if | The delay might be because of the information at page 569 stating that it requires a TSCL delay.  For your frequency I calculated that it needs about 80us, so it is ok to have a delay.  Yes it is dependent as it states on 56. | Open |
| 5 | 0.9a | What is the correct implementation of Restart? | Is this correct?  I2C1STAT1.CLRBF = 1 ‘clear buffer  I2C1CNT = 0 ‘set to count to 0  I2C1PIR.SCIF = 0 ‘ clear int  I2C1ADB1 = I2Cbyte ‘set the slave address  do while I2C1PIR.SCIF = 0 ‘wait…  I2C1CON0.RSEN = 1 ‘set ReStart  Loop |  | Open |
| 6 | 0.9b | When using the initialisation shown in this document for ‘K mode’ the first byte transmitted (it will be an address because of the state engine) does not set the I2C1CON1.ACKSTAT correctly. | I2C1CON1.ACKSTAT is always 0 for the first byte transmitted, subsequent bytes are handled correctly. Why? |  | Open |

# Operation ASM and HEX

These files are an operational demonstration of the ‘legacy mode’ implemented for ‘k mode’.

