Notes to accompany I2C Motor Controller.

1. Hardware

The controller assumes the role of an I2C SLAVE.

I have put a wired connector on the Explorer 16 expansion board. The expansion board should be plugged into the Explorer 16, it is impossible to insert the board incorrectly☺. With the expansion board copper fingers on the left the upper pin on the I2C connector (yellow wire) is RG3 or in I2C terms SDA, the middle pin on the I2C connector is GND, the lower pin on the I2C connector (red wire) is RG2 or in I2C terms SCL.

|  |  |
| --- | --- |
| **I2C Connector Pin** | **Wire** |
| SDA | Yellow |
| GND | Green |
| SCL | Red |

I have assumed the I2C pullup resistors are on the I2C MASTER.

I have not (currently) implemented a ChipSelect bit (wire) but this is outside the I2C bit.

1. Protocol

The 7 bit I2C SLAVE addressing is as follows:

SLAVE\_I2C\_ADDRESS = 0x60

I2CADD = SLAVE\_I2C\_ADDRESS >> 1

Remember the first bit (A0) of the address byte determines a Write or Read, this may be controlled by your I2C Master Library.

The “Read” packet should be in the following format:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (long) M1\_Encoder\_Counter | | | | (long) M2\_Encoder\_Counter | | | | (double)  M1\_Current\_Bearing | | | | (double)  M2\_Current\_Bearing | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

The “Write” packet should be in the following format:

|  |  |
| --- | --- |
| Command Number |  |
| M1 Bearing Requested |  |
|  |
|  |
|  |
| M2 Bearing Requested |  |
|  |
|  |
|  |
| M1\_Speed Requested |  |
|  |
| M2\_Speed\_Requested |  |
|  |
| M1\_Acceleration Requested |  |
|  |
| M2\_Acceleration Requested |  |
|  |
| M1\_Deceleration Requested |  |
|  |
| M2\_Deceleration Requested |  |
|  |

At the moment only Command Number 0x20 is implemented.

1. Theory of Operation

I have (I believe) set the Baud Rate to 400k but since the clocking is controlled by the I2C Master I do not think this matters.

I am using a module on the dsPIC33FJ256GP710 specifically for I2C. When a START is received it fires an interrupt and the attached code (mine) is executed. The documentation suggested that the module does the address checking, interrupting only if the data packet is addressed to this slave. As you can see the interrupt is a state machine, State 1 – awaiting address checks the R/W register of the module; if the MASTER has requested a READ then the state changes so that each interrupt will now put a new data byte in the output register waiting for the Master to clock it out, 16 bytes are available as described in the protocol paragraph (attempting to get more than 20 bytes will push the controller into Waiting for Address), a STOP being detected by the module will change the state to (once again) Waiting for Address; if the MASTER has requested a WRITE then the state changes so that each interrupt will now receive (clock is provided by MASTER) a byte until all the expected bytes are received, currently I am expecting 20 bytes as defined in the protocol paragraph.

The data read by the MASTER is continually updated by the firmware of the PIC, without a motor attached the M1\_Counter and M2\_Counter will not change, but are set as 123456789 and 987654321 respectively, the M1\_Current\_Bearing and M2\_Current\_Bearing will also not change but are set to 45.5 and 90.5 respectively. The firmware has been set to display a message on the LCD if a command packet is received. The two line message will show:

"I2C Packet Received"

“I2C=x”

where x is I2C\_inputbuffer[0]

but the messages may be reversed (the LCD code implements “/n” strangely).

<code>

void \_\_attribute\_\_ ((\_\_interrupt\_\_,\_\_no\_auto\_psv\_\_)) \_SI2C1Interrupt(void)

{

unsigned char i2c\_rx\_char = 0;

\_SI2C1IF = 0; // clear the interrupt flag

switch (my\_i2c\_state)

{

case STATE\_WAITING\_FOR\_ADDRESS:

i2c\_rx\_char = I2C1RCV; // dummy read to clear buffer full bit

i2c\_input\_pointer = 0;

i2c\_output\_pointer = 0;

if (I2C1STATbits.R\_W)

{ // Send data to master

I2C1TRN = I2C\_outputpacket.sg\_char[i2c\_output\_pointer++]; // get first data byte to transmit

I2C1CONbits.SCLREL = 1; // release the clock so that MASTER can drive it

my\_i2c\_state = STATE\_SENDING\_DATA\_TO\_MASTER; // change state to sending data to master

}

else

{ // Receive data from master

my\_i2c\_state = STATE\_RECEIVE\_DATA\_FROM\_MASTER; // change state to receiving data from master

}

break;

case STATE\_RECEIVE\_DATA\_FROM\_MASTER:

I2C\_inputbuffer[i2c\_input\_pointer++] = I2C1RCV; // save the received character in buffer and increment pointer

if (i2c\_input\_pointer >= I2C\_BUFFERSIZE) // check if the message is complete I2C\_BUFFERSIZE = 20

{

i2c\_packet\_received = true;

my\_i2c\_state = STATE\_WAITING\_FOR\_ADDRESS; // change state to waiting for address

}

break;

case STATE\_SENDING\_DATA\_TO\_MASTER:

I2CTRN = I2C\_outputpacket.sg\_char[i2c\_output\_pointer++];

I2C1CONbits.SCLREL = 1;

if (i2c\_output\_pointer >= I2C\_BUFFERSIZE)

{

my\_i2c\_state = STATE\_SENDING\_DATA\_LAST;

}

break;

case STATE\_SENDING\_DATA\_LAST: //

my\_i2c\_state = STATE\_WAITING\_FOR\_ADDRESS;

break;

default: //

my\_i2c\_state = STATE\_WAITING\_FOR\_ADDRESS;

}

}