Hardware / Software I2C Bus Implementation

This lab will introduce the use of the I2C bus and in particular to interface to a Real-Time Clock (RTC) and Digital Temperature Sensor.

The following link will introduce the student to the I2C bus:

http://www.ti.com/lit/an/slva704/slva704.pdf

I will go over in class to the concept of this bus.

Part 1)

The DS1621 is a digital thermometer with the I2C bus interface. Here is the link to its datasheet:

https://pdfserv.maximintegrated.com/en/ds/DS1621.pdf

Study this device and pay attention to page 9 regarding the serial communications to this device. On the next page (page 10), the command 'Read Temperature' will provide the reading of the actual temperature. Write a routine called **'char DS1621_Read_Temp()'** to return the reading of the temperature. Follow the following steps:

a) Use the attached 'Lab11_sample.c' as the reference for the rest of this lab.

b) Go to the second attached file 'i2c.c' and copy the function '**char I2C_Write_Cmd_Read_One_Byte (char Device, char Cmd)'**. Name the new function **'char DS1621_Read_Temp()'**

c) The new void routine does not need any argument. Remove the argument list

d) At the beginning of the routine, define a char variable 'Device' being 0x48. That is the I2C address for this DS1621 device

e) Next, define another char variable called Cmd and set it to equal to the label READ_TEMP (0xAA). That is the 'Read_Temperature' command

f) That will conclude the definition of this routine.

In addition, we will need to initialize the DS1621. We need to build another function for this task. Let us call that function **'void DS1621_Init()'**

a) At the beginning of the routine, define a char variable 'Device' being 0x48. That is the I2C address for this DS1621 device

b) Insert the following 2 lines
    a. I2C_Write_Cmd_Write_Data (Device, ACCESS_CFG, CONT_CONV);
    b. I2C_Write_Cmd_Only(Device, START_CONV)

When both new functions are defined, do the following steps:

a) Call (add) the function **'void DS1621_Init()'** in the Do_init() routine to initialize the DS1621 chip

b) Write a while loop that will wait every second (Use Wait_One_Second() routine) and call the routine '**char DS1621_Read_Temp()'** to get the temperature (in degree Celsius). Calculate the equivalent Fahrenheit temperature and printout both temperatures on TeraTerm.

Part 2)

Next, we are going to use the I2C to interface to a Real-Time-Clock (RTC) device called DS3231. This device is used to keep the time. Below is the link to the datasheet of this device:

https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

The RTC has a set of registers that store the actual time in terms of second, minute, hour, day, day of the week, month and year. Refer to page 11 of the datasheet. The registers with the addresses from 00 to 06 are the ones that contain the time values. We will need to write a routine called **'void DS3231_Read_Time()**' to fetch all the mentioned registers and to display them on the TeraTerm. In addition, to avoid displaying duplicated data, it is required to check the 'Second' register to determine whether it has been updated with a new second. If so, then the new data can be displayed on the screen. When a new time is displayed, also called the routine 'int DS1621_Read_Temperature()' from Part 1) to get the temperature and show it on the screen.

Here is how to implement this routine:

a) Use the sample routine '**char I2C_Write_Address_Read_One_Byte(char Device, char Address)** provide in the 'i2c.inc' and copy it and rename it as '**void DS3231_Read_Time()**'.

b) The new void routine does not need any argument. Remove the argument list.

c) Since this is now a void routine, there is no need for the 'return' statement. Remove that line at the end

d) At the beginning of the routine, define a char variable 'Device' being 0x68. That is the I2C address for this DS1621 device. That is the I2C address for this DS3231 device

e) Next, add another char variable 'Address' with the value 0x00. This is the value for the register 0x00 pointing to the register 'second'

f) Delete the line 'Data_Ret = I2C_Read(NAK);'

g) Add a new line 'second = I2C_Read(ACK);'

h) The new line above allows the system to read the register 'second' from DS3231. Add similar lines below that line for the variables 'minute', 'hour', 'dow' (day of week), 'day', 'month', and 'year'. Do that in the order provided. For the last variable 'year' substitute 'ACK' by 'NAK' to terminate the read sequence

i) Make sure to end the function with 'I2C_Stop();'

Once this routine is completely designed, modify Part 1) of while loop by removing the 'Wait_One_Second()' routine and instead call this new routine 'DS3231_Read_Time()' to get the new time and in particular the reading of the 'second' register. Compare the new reading of the 'second' with an old reading. If the new second is the same as the old, then do nothing. If different, then print out the actual time. Use the following printf format

printf ("%02x:%02x:%02x %02x/%02x/%02x",hour,minute,second,month,day,year);

This will display the time and date using the format "hh:mm:ss mm/dd/yy'.
In addition, print out the actual temperature as performed in part 1)

followed by the readout of the temperature.

Note: Due to the potential effect that the RTC might not properly initialized, the time and date might not be correct. That will be fixed on the next part.

Part 3)

The next task is to write a routine to setup an initial time for the RTC. Call that routine '**void DS3231_Setup_Time()'.** The purpose of this routine is to pre-program all the registers from 00 to 06 of the RTC with fixed numbers.

Here is how to implement this function:

a) Copy the function '**void I2C_Write_Address_Write_One_Byte(char Device, char Address, char Data)'** and name it as '**void DS3231_Setup_Time()'.** There is no parameter needed for this function

b) At the beginning of the routine, define a char variable 'Device' being 0x68. That is the I2C address for this DS1621 device.

c) Next, add another char variable 'Address' with the value 0x00. This is the value for the register 0x00 pointing to the register 'second'

d) Set the values that you want to use to the variables 'second', 'minute', 'hour', 'dow', 'day', 'month' and 'year'. Set the value as in bcd value. For example, if the minute is to be set as 45, don't convert that number into hex number but use the value 0x45 instead.

e) Replace the line 'I2C_Write(Data);' with I2C_Write(second);'

f) Then add below that line the same I2C_Write command but with the next variable 'minute'

g) Repeat the action above for the remaining variables.

h) Make sure to end the function with I2C_Stop();

Once this function is implemented, modify Part 2) above to use the push-button connected to INT0 to force a call to this routine. When pressed down, the button at INT0 will generate an interrupt. When the interrupt is detected, call the routine '**void DS3231_Setup_Time()'** to preset the time and then clear the interrupt.

Part 4)

The above part dealt with the concept of the I2C bus using the hardware technique whereas all the data bit transmissions are performed by internal hardware inside the microcontroller. This method allows faster data transmissions but the two SCL and SDA pins must be dedicated. In addition, the PIC 18 microcontroller that we are using merges the two I2C and SPI functions into the same pins thus allowing the use of only one function and not both simultaneously. If we want to use both SPI and I2C, we will need to switch from the use of hardware I2C to software I2C. This new technique will use the concept of software bit banging to perform all the data bit transmission. For sure this technique will not have the same speed performance as the hardware I2c but it does allow the microcontroller to communicate with any I2C device. Since bit banging can be applied to any pin, software I2C also allows the flexibility to choose any GPIO pin for the implementation.

The attached file ‘soft_i2c.inc’ will provide the needed library function for the software I2C implementation. Now, for this lab, let us choose two different pins for the signal SCL and SDA. For example, let us physically move SCL to PORT D bit 3 and SDA to PORT D bit 4.

Take the I2C implementation from the above part and locate the line showing ‘#include ‘i2c.c’. Replace that line with the following lines:

```
#define     SCL_PIN     PORTDbits.RD3
#define     SCL_DIR     TRISDbits.RD3
#define     SDA_PIN     PORTDbits.RD4
#define     SDA_DIR     TRISDbits.RD4

#include "softi2c.c”
```

When done, recompile the program and check through TeraTerm that the output on the screen. Make sure to move the two pins SCL and SDA to the new locations.