**CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA**
**COLLEGE OF ENGINEERING**

**ECE 3301L Fall 2025 Microcontroller Lab Felix Pinai**
**Session 2**

# LAB 11: I2C Bus Implementation

This lab will introduce the student to 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**

Normally the I2C bus uses a 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 flexibility to choose any GPIO pin for the implementation.

The attached file 'I2C_Soft.c' will provide the library function needed for the software I2C implementation. Now, let us choose two different pins for the signal SCL and SDA. See the attached schematics for the pin assignments of those two pins.

For example, assuming that we are physically using SCL at PORT C bit 0 and SDA at PORT C bit 1.

Edit the file 'I2C.h' by changing the port assignments of the two signals SCL_PIN and SDA_PIN.

```
#define SCL_PIN     PORTCbits.RC0
#define SCL_DIR     TRISCbits.RC0

#define SDA_PIN     PORTCbits.RC1
#define SDA_DIR     TRISCbits.RC1
```

**Check the provided schematics and determine the pin assignments for the signals SCL and SDA.** Edit the 'I2C.h' file to reflect the assignments of those two pins. In addition, connect the logic analyzer's channel 0 and 1 respectively to the signals SCL and SDA.

Copy the project from Lab #10 to make the new Lab #11. Add the modified 'I2C_h' into the Header folder and add the 'I2C_Soft.c' into the Sources folder.

Repeat the same process for the two provided 'I2C_Support.h' and 'I2C_Support.c'. Place the Header file into the header folder and the .c file into the Sources folder.

Add the following lines at the beginning of the main program:

```
#include "I2C.h"
#include "I2C_Support.h"
```

And the following lines below the 'pragma' lines:

```
char tempSecond = 0xff;
char second = 0x00;
char minute = 0x00;
char hour = 0x00;
char dow = 0x00;
char day = 0x00;
char month = 0x00;
char year = 0x00;
char setup_second, setup_minute, setup_hour, setup_day, setup_month, setup_year;
char alarm_second, alarm_minute, alarm_hour, alarm_date;
char setup_alarm_second, setup_alarm_minute, setup_alarm_hour;
```

**Part 1)**

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

**https://www.analog.com/media/en/technical-documentation/data-sheets/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 definition on how to read the actual temperature. Follow the following steps:

a) Locate the empty function **'char DS1621_Read_Temp()'** in the 'I2C_Support.c' file

b) Open the file 'I2C_Soft.c' and copy the entire content of the function '**char I2C_Write_Cmd_Read_One_Byte (char Device, char Cmd)'** and put it into the function **'char DS1621_Read_Temp()'**

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

d) Next, define another char variable called Cmd and set it to equal to the label READ_TEMP. That constant is defined at the beginning of the file to have the value 0xAA. It is the 'Read_Temperature' command for DS1621

e) That will conclude the definition of this routine.

Next, we will need to initialize the DS1621. We need to build another function for this task. Let us call that function **'void DS1621_Init()'.** Locate that empty function in the 'I2C_Support.c' file and do the following steps:

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)

Now we have created those two functions described above. We will call those functions are two places:

a) Add the following two lines to initialize the I2C interface and the DS1621 chip. Place them right before the line 'Init_Interrupt()':

```
I2C_Init(100000);
DS1621_Init();
```

b) In the 'while (1) loop' of the main() program from lab #10 and before the line 'if (Nec_OK == 1);' write the following two lines:

```
signed char tempC = DS1621_Read_Temp();
signed char tempF = (tempC * 9 / 5) + 32;
 printf (" Temperature = %d degreesC = %d degreesF\r\n", tempC, tempF);
Wait_One_Sec_Soft();
```

c) Compile the new program. If everything is set up properly, the temperature should be displayed on the TeraTerm every second.

d) Using the logic analyzer with its channel 0 connected to SCL line while channel 1 is on SDA line, capture the I2C transaction generated from the line 'DS1621_Read_Temp()' above. Add the I2C tool into the logic analyzer for the software to properly decode the I2C transaction. Capture that waveform for the report. Make sure on the analyzer the graph will show the transaction uses the address 0x48 and there is an 'ACK' to the selection of that address.

e) Copy the function 'DS1621_Read_Temp()' to another function called 'DS1621_Read_Temp_Bad()'. On the new function, change the value of 'Device' from 0x48 to something else like for example 0x49. Add the following line in the original program:

```
char tempC1 = DS1621_Read_Temp_Bad();
```

after the original line: char tempC = DS1621_Read_Temp();

Compile the program and download it to the board. Look at the analyzer and check the response. You should observe two transactions, one for device 0x48 and one for device 0x49. The response for the first transaction should have 'ACK' while with the second transaction the response should have 'NAK' instead. This does indicate that talking to the wrong address will result in a negative (NAK) response.

**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 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 data sheet. 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) Take the content of the sample routine '**char I2C_Write_Address_Read_One_Byte(char Device, char Address)** provided in the 'I2C_Soft.c' and place it into the empty function '**void DS3231_Read_Time()**' inside the 'I2C_Support.c'.

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

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

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

e) Delete the line ‘Data_Ret = I2C_Read(NAK);’

f) Add a new line ‘second = I2C_Read(ACK);’

g) The new line above allows the system to read the register ‘second’ from the 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

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

Once this routine is completely designed, modify Part 1) of while loop by replacing the newly added lines with the following:

```
DS3231_Read_Time();
if(tempSecond != second)
{
   tempSecond = second;
   char tempC = DS1621_Read_Temp();
   char tempF = (tempC * 9 / 5) + 32;
   printf ("%02x:%02x:%02x %02x/%02x/%02x",hour,minute,second,month,day,year);
   printf (" Temperature = %d degreesC = %d degreesF\r\n", tempC, tempF);
}
```

This new routine will display on the TeraTerm the time and date using the format “hh:mm:ss mm/dd/yy’ along with the actual temperature as performed in part 1)

Use the logic analyzer again to capture this time readout transaction. Save the capture for the report.

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

**Part 3)**

The next task is to write a routine to set up 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 DS3231 device.

c) Next, define another char variable 'Address' with the value of 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, make a single call to this function before the while (1) loop in part 2). The time that will be displayed will be the new start time set by the new function.

**Part 4)**

The remote-control section of lab #10 should work as normal. Check by pressing any button on the remote and make sure the LCD screen still shows the button being pressed and the buzzer still works along the RGB LEDs. Pick the button 'CH-'. Check what index position of that button on the remote. Add some new codes inside the while (1) routine and inside the 'if (Nec_OK == 1);' condition to check whether the actual button being pressed is the button that was chosen. If so, call the routine '**DS3231_Setup_Time()'** to reload the time into the Real-Time clock. **Do remove the same call that placed before the 'while (1)' loop in part 3).**