LAB_8 MCU-To-MCU I2C Basic Communication

1. Objective:

- To be familiar with PIC Microcontroller
- To be familiar with PIC16F877A Microcontroller internal and external structure and its configuration.
- To learn how to use I2C Communication works

2. Required Components

Table 1. Components

Qty.	Component Name
2	PIC16F877A or PIC18F2550 or anyother
2	BreadBoard
8	LED
1	Resistors Kit
1	Capacitors Kit
1	Jumper Wires Pack
2	LM7805 Voltage Regulator (5v)
2	Crystal Oscillator
1	PICkit2 or 3 Programmer
2	9v Battery or DC Power Supply

3. Introduction

3.1 I^2C

Up till now, we've introduced UART, SPI serial communication ports. We've also done a handful of practical LABs using both of them. Before introducing the I2C bus, let's just quickly review the previous serial ports.

UART

Universal Asynchronous Receiver/Transmitter or UART for short represents the hardware circuitry (module) being used for serial communication. UART is sold/shipped as a standalone integrated circuit (IC) or as an internal module within microcontrollers. There is a couple of IO pins dedicated to the UART serial communication module highlighted in the following figure. Namely, RX (data input – receiving end) & TX (data output – transmitting end).

There are actually two forms of UART Hardware as follows:

UART – Universal Asynchronous Receiver/Transmitter

USART – Universal Synchronous/Asynchronous Receiver/Transmitter

The Synchronous type of transmitters generates the data clock and sends it to the receiver which works accordingly in a synchronized manner. On the other hand, the Asynchronous type of transmitter generates the data clock internally. There is no incoming serial clock signal, so in order to achieve proper communication between the two ends, both of them must be using the same baud rate. The baud rate is the rate at which bits are being sent bps (bits per second).

SPI

SPI is an acronym for (Serial Peripheral Interface) pronounced as “S-P-I” or “Spy”. Which is a serial, synchronous, single-master, multi-slave, full-duplex interface bus typically used for serial communication between microcomputer systems and other devices, memories, and sensors. Usually used to interface Flash Memories, ADC, DAC, RTC, LCD, SD cards, and much more.

The SPI was originally developed by Motorola back in the 80s to provide full-duplex serial communication to be used in embedded systems applications. Specifically for very short distance communications (ob-board).

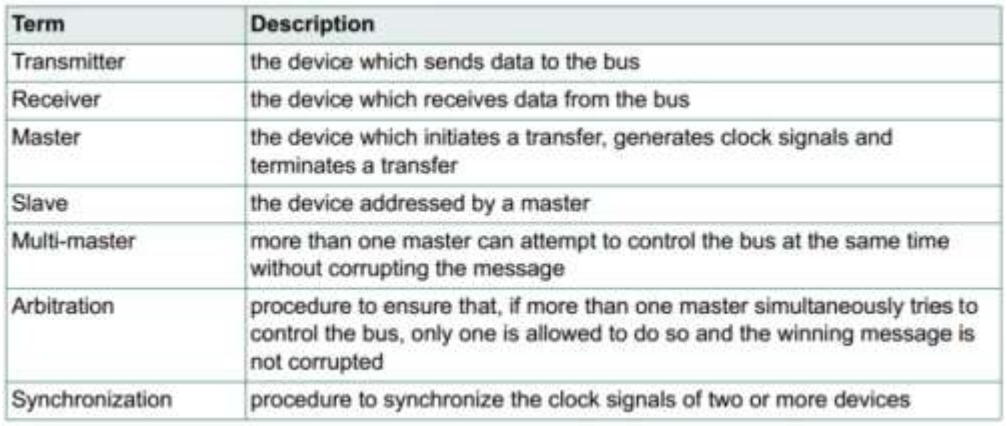
In typical SPI communication, there should be at least 2 devices attached to the SPI bus. One of them should be the master and the other will essentially be a slave. The master initiates communication by generating a serial clock signal to shift a data frame out, at the same time serial data is being shifted-in to the master. This process is the same whether it’s a read or write operation.

I2C

I2C (i-square-c) is an acronym for “Inter-Integrated-Circuit” which was originally created by Philips Semiconductors (now NXP) back in 1982. I2CTM is a registered trademark for its respective owner and maybe it was the reason they call it “Two Wire Interface (TWI)” in some microcontrollers like Atmel AVR. The I2C is a multi-master, multi-slave, synchronous, bidirectional, half-duplex serial communication bus. It’s widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

The I2C-bus is used in various control architectures such as System Management Bus (SMBus), Power Management Bus (PMBus), Intelligent Platform Management Interface (IPMI), Display Data Channel (DDC) and Advanced Telecom Computing Architecture (ATCA)

Table 2. Definition of I^{2}C bus terminology



Term	Description
Transmitter	the device which sends data to the bus
Receiver	the device which receives data from the bus
Master	the device which initiates a transfer, generates clock signals and terminates a transfer
Slave	the device addressed by a master
Multi-master	more than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	procedure to synchronize the clock signals of two or more devices

3.2. I^{2}C remark

Is This The Best I2C Driver?

It’s not the best I^{2}C driver that you can possibly create. However, it just works! and gives you a real hands-on experience with the I2C Bus interfaces and how to set up and run an I2C master, salve, and all this stuff. The drivers shown in this article are the bare-minimum C-Code you’ll need to in order to get something to work. But, you’ll definitely need to perform some more modifications to meet your need and make your system more reliable.

The I2C module's state must be checked before and after each and every single process. Some functions should actually return the status of the bus, ACK, NACK, etc. This is critical for many situations. Another thing to note is the bus collision, which happens sometimes, is actually checked for in the code above but it's not handled. These exceptions of a collision, overflow, etc should also be considered and well-handled.

The use of while(condition); statements for waiting to check a specific condition until it's met is NOT recommended at all. Almost in all situations, we call this "Blocking Code". If something goes wrong on the I2C bus, which is very probable, maybe all your microcontroller using the blocking code style will FREEZE forever. Bad clock stretching implementations can easily lead to such situations or any bus contention issue.

Briefly Describe The I2C Bus

- Inter-Integrated Circuit (I2C) is a Synchronous Serial Communication Port (Protocol)
- Multi-Master Multi-Slave Bus. Only one conversation may occur at a time.
- I2C is a Two-wire interface (TWI).
- Bi-directional open-drain hardware pins.
- Half-Duplex Communication. One way of data transfer at a time. Direction can be reversed but only after terminating the current stream and initiating a new start condition.
- The data bits are only valid when the clock line is HIGH. Data must be stable only then.
- I2C bus allows for hot-swapping. This means I2C devices could be easily hooked to and out of the bus without the need to restart devices on the bus.
- I2C Bus supports arbitration and clock synchronization.
- I2C supports clock stretching. This enables all slaves to hold the clock line low while servicing other interrupts until reading the incoming data, then it can release the SCL clock line and conversation continues.

I2C Bus Standards

Originally, the I2C-bus was limited to 100 kbit/s operation. Over time there have been several additions to the specification so that there are now five operating speed categories. Standard-mode, Fast-mode (Fm), Fast-modeand High-speed mode (Hs-mode) devices are downward-compatible. Which means any device may be operated at lower bus speed. Ultra Fast-mode devices are not compatible with previous versions since the bus is unidirectional.

Bidirectional bus:

- Standard-Mode (Sm), with a bit rate up to 100 kbit/s
- Fast-Mode (Fm), with a bit rate up to 400 kbit/s
- Fast-Mode Plus (Fm+), with a bit rate up to 1 Mbit/s
- High-speed Mode (Hs-mode), with a bit rate up to 3.4 Mbit/s.

Unidirectional bus:

- Ultra Fast-Mode (UFm), with a bit rate up to 5 Mbit/s

How Many Devices Can You Hook To The I2C Bus?

Theoretically speaking, the limit for 7-Bit address mode is the addressable count of 7-bit and similarly for 10-Bit address mode. However, in practice, we're limited by the total bus capacitance which deforms the clock and data signals.

I2C Bus Conditions (Elements)

- Start Condition (S)
- Stop Condition (P)
- Repeated Start (Restart) Condition (Sr)
- Acknowledge ACK (A)

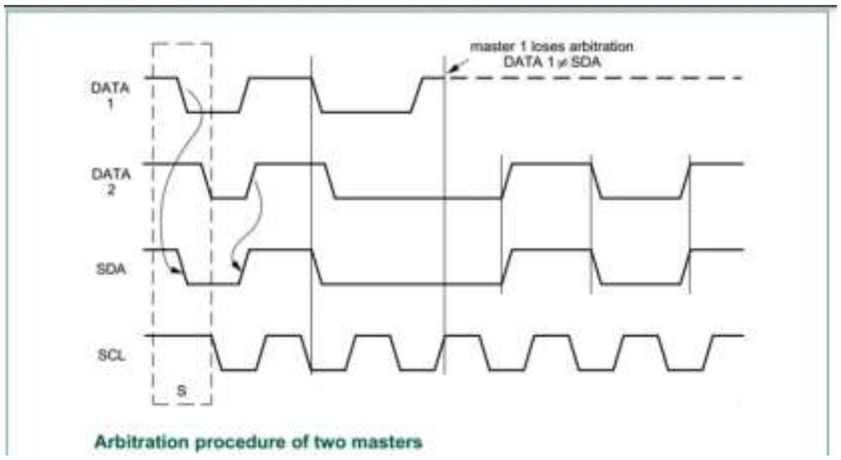
- Not Acknowledge NACK (~A)
- Address + R/W
- Data Byte

I2C Bus Arbitration

Arbitration, like synchronization, refers to a portion of the protocol required only if more than one master is used in the system. Slaves are not involved in the arbitration procedure. A master may start a transfer only if the bus is free. Two masters may generate a START condition within the minimum hold time (tHD;STA) of the START condition which results in a valid START condition on the bus. Arbitration is then required to determine which master will complete its transmission.

Arbitration proceeds bit by bit. During every bit, while SCL is HIGH, each master checks to see if the SDA level matches what it has sent. This process may take many bits. Two masters can actually complete an entire transaction without error, as long as the transmissions are identical. The first time a master tries to send a HIGH but detects that the SDA level is LOW, the master knows that it has lost the arbitration and turns off its SDA output driver. The other master goes on to complete its transaction.

No information is lost during the arbitration process. A master that loses the arbitration can generate clock pulses until the end of the byte in which it loses the arbitration and must restart its transaction when the bus is free. If a master also incorporates a slave function and it loses arbitration during the addressing stage, it is possible that the winning master is trying to address it. The losing master must, therefore, switch over immediately to its slave mode.


master 1 loses arbitration
DATA 1 ≠ SDA
DATA 1
DATA 2
SDA
SCL
S
Arbitration procedure of two masters

Consider The Following Example For 2 I2C Masters Arbitration

The moment there is a difference between the internal data level of the master generating DATA1 and the actual level on the SDA line, the DATA1 output is switched off. This does not affect the data transfer initiated by the winning master. Since control of the I2C-bus is decided solely on the address and data sent by competing masters, there is no central master, nor any order of priority on the bus. There is an undefined condition if the arbitration procedure is still in progress at the moment when one master sends a repeated START or a STOP condition while the other master is still sending data.

I2C Clock Synchronization

Two masters can begin transmitting on a free bus at the same time and there must be a method for deciding which takes control of the bus and complete its transmission. This is done by clock synchronization and arbitration. In single master systems, clock synchronization and arbitration are not needed.

Clock synchronization is performed using the wired-AND connection of I2C interfaces to the SCL line. This means that a HIGH to LOW transition on the SCL line causes the masters concerned to start counting off their LOW period and, once a master clock has gone LOW, it holds the SCL line in that state until the clock HIGH state is reached. However, if another clock is still within its LOW period, the LOW to HIGH transition of this clock may not change the state of the SCL line. The SCL line is therefore held LOW by the master with the longest LOW period. Masters with shorter LOW periods enter a HIGH wait-state during this time.I2C Clock Synchronization

When all masters concerned have counted off their LOW period, the clock line is released and goes HIGH. There is then no difference between the master clocks and the state of the SCL line, and all the masters start counting their HIGH periods. The first master to complete its HIGH period pulls the SCL line LOW again. In this way, a synchronized SCL clock is generated with its LOW period determined by the master with the longest clock LOW period, and its HIGH period determined by the one with the shortest clock HIGH period.

I2C Clock Stretching

Clock stretching pauses a transaction by holding the SCL line LOW. The transaction cannot continue until the line is released HIGH again. Clock stretching is optional and in fact, most slave devices do not include an SCL driver so they are unable to stretch the clock.

On the byte level, a device may be able to receive bytes of data at a fast rate, but needs more time to store a received byte or prepare another byte to be transmitted. Slaves can then hold the SCL line LOW after reception and acknowledgment of a byte to force the master into a wait state until the slave is ready for the next byte transfer in a type of handshake procedure.

On the bit level, a device such as a microcontroller with or without limited hardware for the I2C-bus can slow down the bus clock by extending each clock LOW period. The speed of any master is adapted to the internal operating rate of this device I2C Clock Synchronization

I2C Bus Debugging

Just make sure that you’ve got the right tools before starting to debug your I2C, You’ll need for this task a decent logic analyzer or even a DSO with decode feature enabled like my Siglent SDS1104. You can use the simulation tools available in Proteus if you don’t have any of the hardware tools available.

Then start your debugging session by stepping through your code especially the suspicious parts of it. If you’ve got a hardware debugger tool it would be great and highly recommended. If not, then you might use some LEDs, serial terminal, and a lot of artificial delay insertion.

4. Experiment

MCU-To-MCU I2C Basic Communication– LAB 1

4.1 Coding:

- **Open** the MPLAB IDE and create a new project.
- **Set** the configuration bits to match the generic setting
- The master I2C Transmitter initiates the I2C Communication and starts sending some bytes sequentially. Let’s send a running counter from 0 up to 255.
- Slave will setup the I2C slave receiver port and respond to data reception interrupt signals. Each time a byte of data is received via the I2C bus, the slave device will output it to PORTD in order to check it out for validation.
- Write the main loop (routine) of the system.

4.2 Simulation:

The master MCU (on the left) is sending bytes of data (0 to 255) one byte per second. And the receiver is forwarding the received data to PORTD so you can see the data bits on the LEDs. Moreover, the I2C Bus debugger shows you the timestamps and exact format of each byte of data being transferred over the I2C bus.

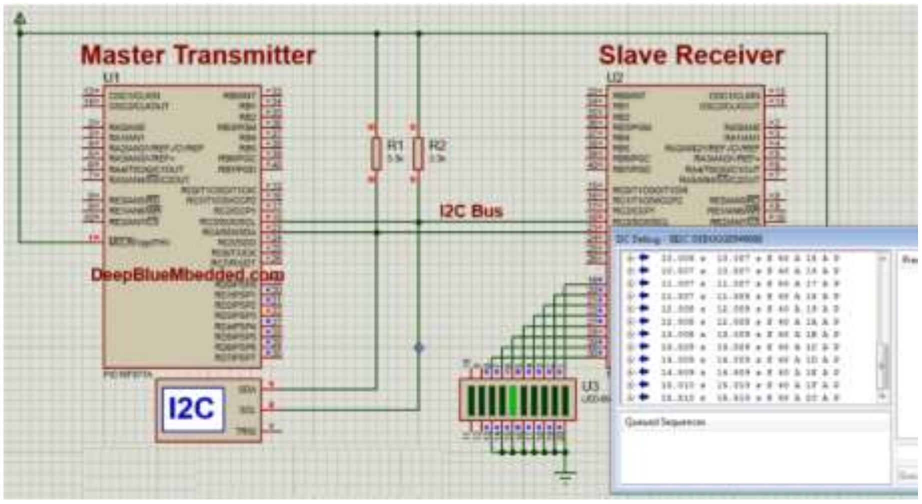


Figure 5. Circuit diagram

4.3. Prototyping

- Prototyping this project involves connecting a relatively larger power supply (12v).
- Follow the schematic diagrams

5. Homework

Write an C program for creating an I2C Communication network consisting of a master transmitter and a slave receiver I2C devices and making sure everything is running correctly.

Name: Student Code: Class: Lab:

1. Circuit
2. Algorithm flowchart
3. Code and explanation
4. Summary