Using an I2C LCD with the Arduino Wire Library

Scope and Audience

- This document gives a limited introduction into the use of I2C communications with the Arduino.
  - Specifically Single Master to single Slave, 7bit addressing, 100Kbps bus speed using the Arduino Wire Library with low to medium data signalling rates.
- It is targeted at the amateur electronics/maker hobbyist.
- Some rudimentary knowledge of electronics and software development would be an advantage when reading the following content.
- It is not intended to be a definitive guide, more to give an outline and help the hobbyist create projects using I2C devices (specifically controlling an I2C LCD with PCF8574 in this case).
  - And more importantly to dispel some myths and feel confident in its application and use.
- The examples cited are to support the use of the 'LiquidCrystal_I2C_PCF8574' library and should be read in conjunction with the 'Arduino I2C LCD Driver Library' Instructable.

Note : The following information is true at the time of writing this document 14/10/15

What is I2C?

- ‘I2C’, ‘I squared C’, ‘I^2C’ or IIC stands for Inter-Integrated Circuit bus.
- It is a standard created by Philips Semiconductors (now NXP Semiconductor) in the early 1980s ‘originally developed as a control bus for linking together microcontroller and peripheral ICs for Philips consumer products’[1] across short distances.
- In common parlance, sometimes known as ‘TWI’, Two Wire Interface by other manufacturers to circumvent trademarking issues.
  - There are however, some differences between I2C and TWI but are not in the scope of this document.
- Bus speeds are[2]; Standard Mode 100 kbit/s (Sm), Fast-mode 400 kbit/s (Fm), Fast-mode Plus 1 Mbit/s (Fm+), High-speed mode 3.4 Mbit/s (UFm)
  - In general it is possible for different speed devices to occupy the same bus via clock stretching. Though this is not in the scope of this document.
- Requires only two bus lines to operate;
  - Serial Data Line : (SDA)
  - Serial Clock Line : (SCL)

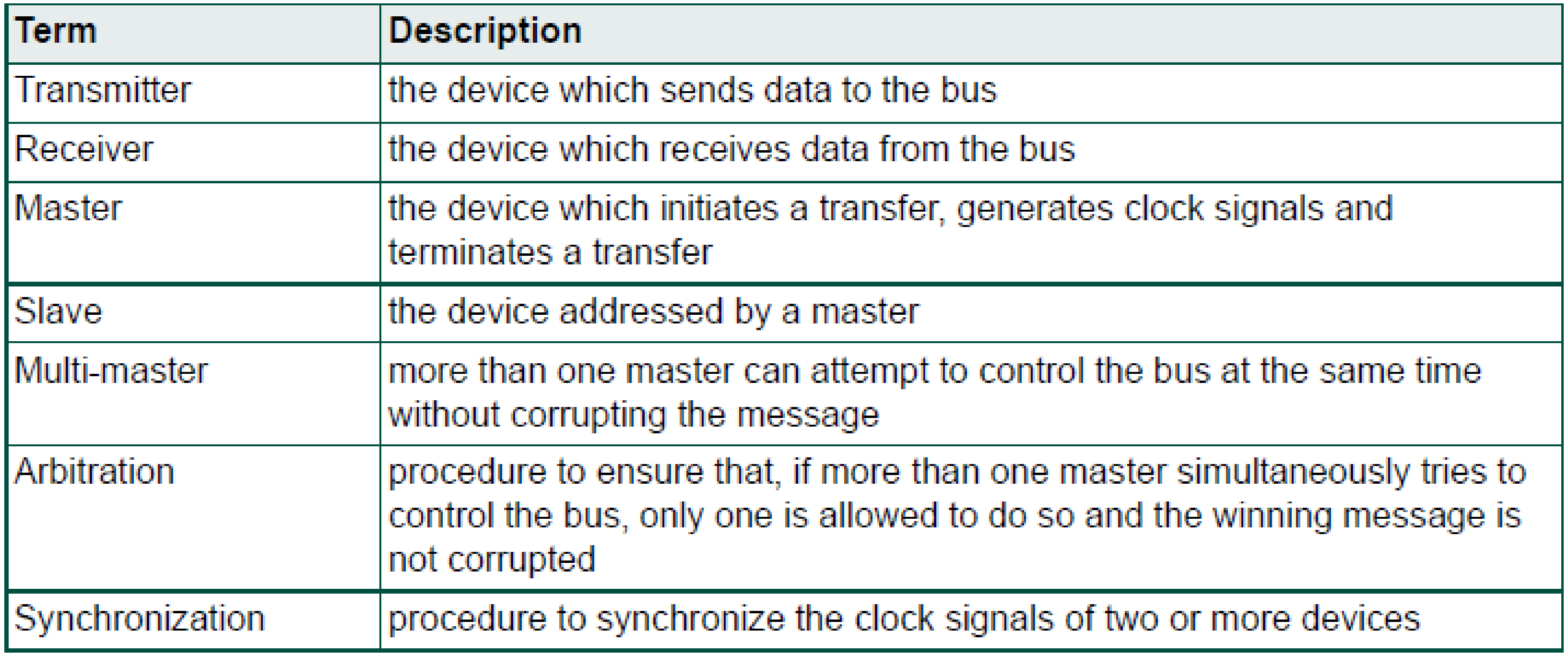
1. I2C Peripherals, Data Handbook IC12, 1996, Preface
2. UM10204, I^2C-bus specification and user manual, Rev. 6 — 4 April 2014

What is I2C?

- Bus comprises Masters and Slaves. Only one master can 'take control' of the bus at any one time.
- Each device connected to the bus is software addressable by a unique address (7 or 10 bits in length) and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers.
- It is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.
- Data transfer is bi-directional (not for UFm) 8 bit serial oriented.
- The number of ICs that can be connected to the same bus is limited by the maximum bus capacitance 200pF – 400pF, depending upon bus speed.
- Communications on the bus are serial, synchronously clocked via the SCL line.

Definition of I2C Terms

To dispel any ambiguities a copy of the I2C definition of terms is given below[3];



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. UM10204, I^2C-bus specification and user manual, Rev. 6 — 4 April 2014. § 3.1 Standard-mode, Fast-mode and Fast-mode Plus I2C-bus protocols

Arduino I2C Electrical Connections

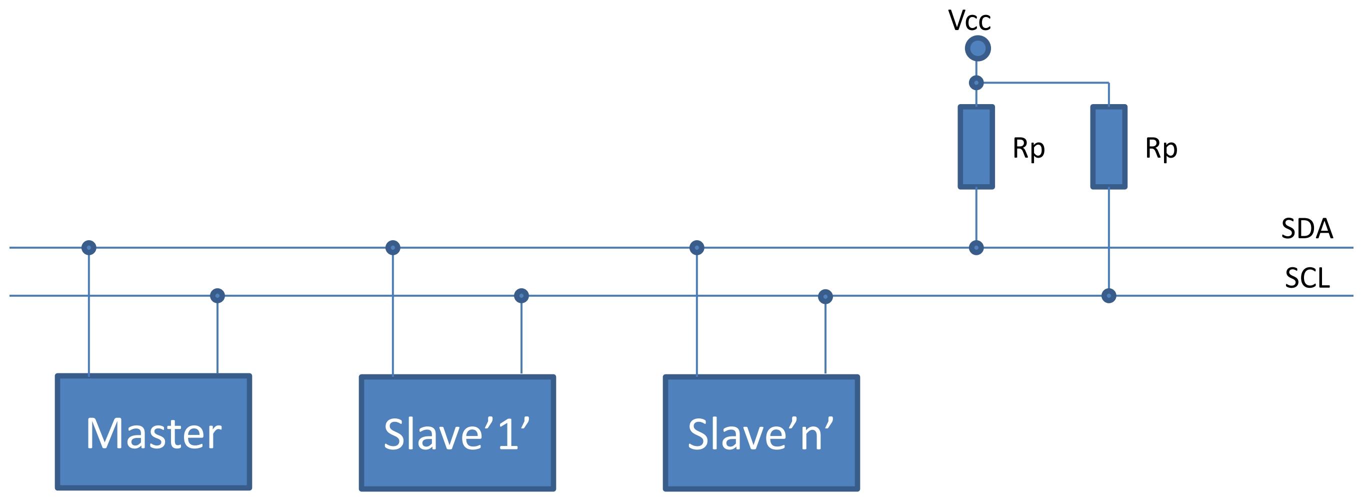
The following table identifies the I2C SDA/SCL pins for a range of Arduinos

Board	**I2C / TWI pins**
Uno, Ethernet	A4 (SDA), A5 (SCL). Note for R3 version of the Uno, these pins are also brought out to the header near AREF.
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

See : https://www.arduino.cc/en/Reference/Wire

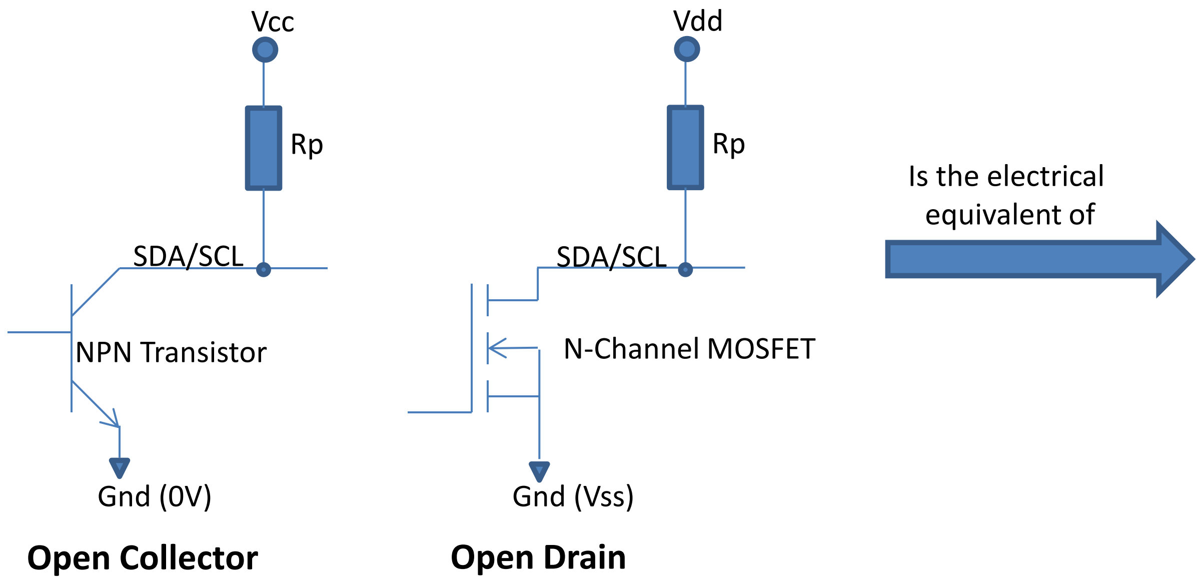
Electrical Inter-connections

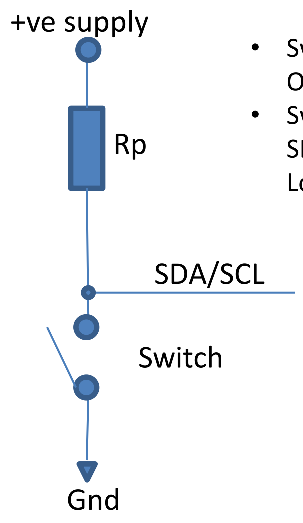
- Multiple devices can be ‘hung’ off the SDA and SCL lines much like ‘clothes on a washing on a line.’



Electrical Inter-connections

- The output stage of the SDA and SCL lines are know as Open Collector or Open Drain respectively which are pulled high usually via an external source resistance (Rp) connected to a positive supply voltage (Vcc/Vdd).





- Switch open SDA/SCL Output = High or '1'
- Switch closed SDA/SCL Output = Low or '0'

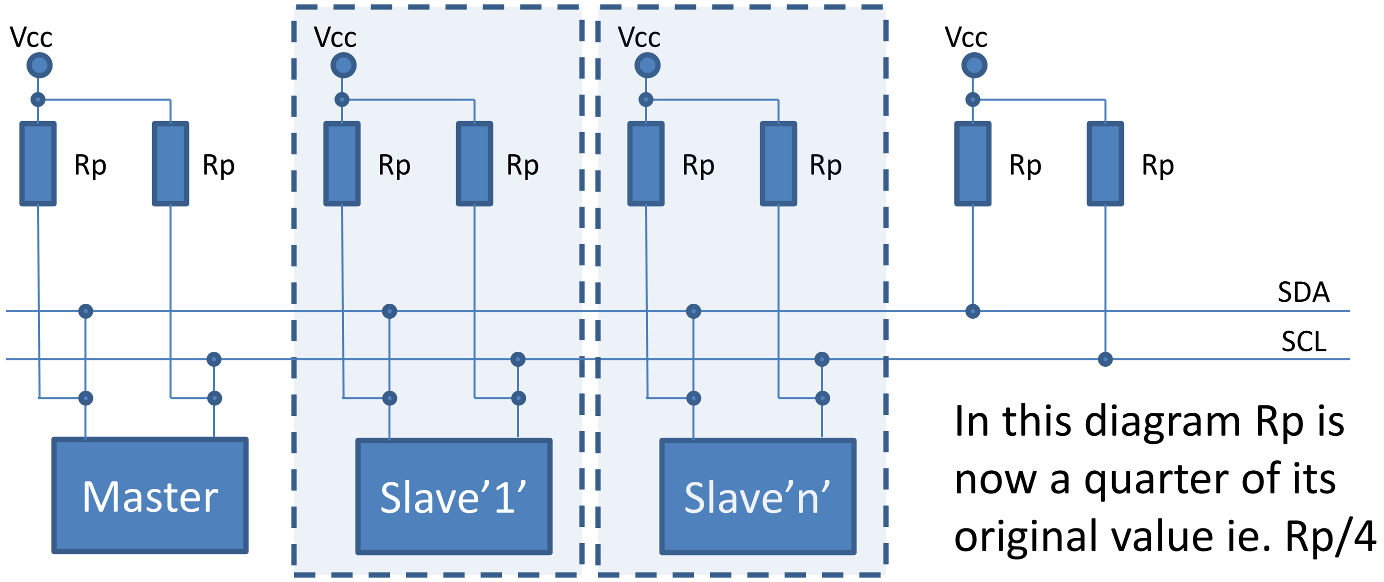
Electrical Inter-connections

- There are limitations as to the minimum (limited by supply voltage) and maximum (limited by bus capacitance) size of the pull up resistors ‘Rp’. As a pragmatic ‘rule of thumb’ try to keep this around 4K7Ω – 10KΩ in total[4].
- One important point to note is, if your project uses many pre-manufactured I2C breakout modular slave devices (I/O Expander, temp sensor, RTC, EEPROM etc.) sharing the same bus and each board comes fitted with its own pull up resistors you may find your I2C communications stops working or becomes intermittent.

4. UM10204, I^2C-bus specification and user manual, Rev. 6 — 4 April 2014. § 7.1 Pull up resistor sizing

Electrical Inter-connections

- This is because you are effectively adding more parallel load to the SDA/SCL bus lines. As below;



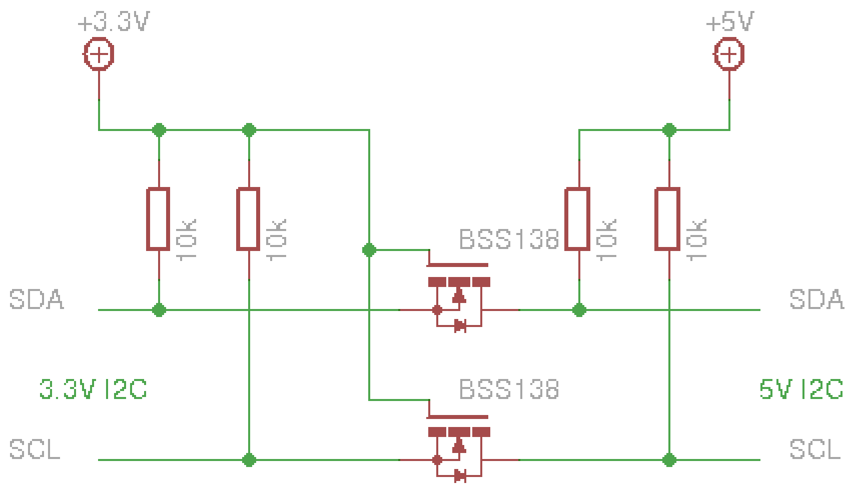
In this diagram Rp is now a quarter of its original value ie. Rp/4

- The solution is to remove some of the extra pull up resistors.

Note : The Arduino Uno R3 does not come with Rp fitted to either SDA or SCL. Though AVR Microcontrollers have internal pull ups which are automatically enabled by the Wire.begin() function call. Important point to consider if connecting different supply technologies together.

Interfacing between 5v – 3.3v Devices

- It is better to use a bi-directional level shifting circuit like that shown below[5] rather than connecting together different technology devices as some internet blogs would suggest.



This is primarily because an I2C compliant device running from 5v requires a value of >3.5v (0.7 * VDD)[6] in order to recognise a Logic 'High' level at it's I2C input. Consequently you are taking a gamble that the there is sufficient sensitivity in the 5v device to overcome the 200mv difference. As a result you may observe non-deterministic behaviour.

5. I2C bi-directional level shifter, http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter
6. UM10204, I^{2}C-bus specification and user manual, Rev. 6 — 4 April 2014. § 3.1.2 SDA and SCL logic levels

Interfacing between 5v – 3.3v Devices

- If you really must connect different technologies directly together there is a comprehensive explanation of the various design constraints/'pit falls' you would need to consider at the following Arduino web page.
  - See[7] : http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter
- In general you will need to ensure/consider the following;
  - For the device with the lowest supply, its output stage is capable of driving a sufficiently high output such that a logic level '1' is sensed at the input of the highest supply device.
  - For the device with the highest supply you will need to ensure the output stage does not drive its logic level '1' to a value that exceeds the maximum DC characteristics of the input stage of the lowest supply device.
- Further details are outside the scope of this document.

7. I2C bi-directional level shifter, http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter

I2C Protocol

- This section will focus on the following I2C protocol signalling entities;
  - **Start** : All I2C transactions begin with a start condition (S).
  - **Stop** : All I2C transactions end with a stop condition (P).
  - **Data** : Every data byte put on the SDA line must be eight bits long. Data is transferred with the MSB first.
    - Although the number of bytes that can be transmitted per transfer is unrestricted this document will only consider single byte transfer.
  - **Ack** : The acknowledge bit takes place after every byte transfer, allowing the receiver to signal the transmitter that the byte was successfully received and another byte may be sent
  - **Address** : The first byte after the start condition, it comprises 7 address bits and a single R/W bit sent MSB first on the SDA line.
- Note : The I2C specification provides for other protocol signalling entities, however, for the purpose of outlining its use with the PCF8574, low to medium signalling rates as typically observed when driving an LCD, to keep things simple, only the five items above will be considered.

I2C Protocol

- Start and Stop conditions

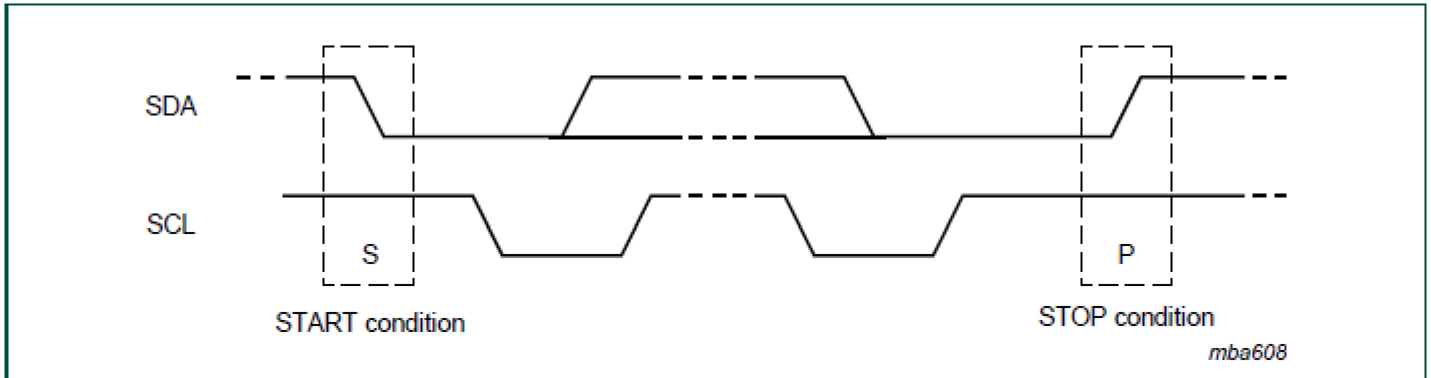


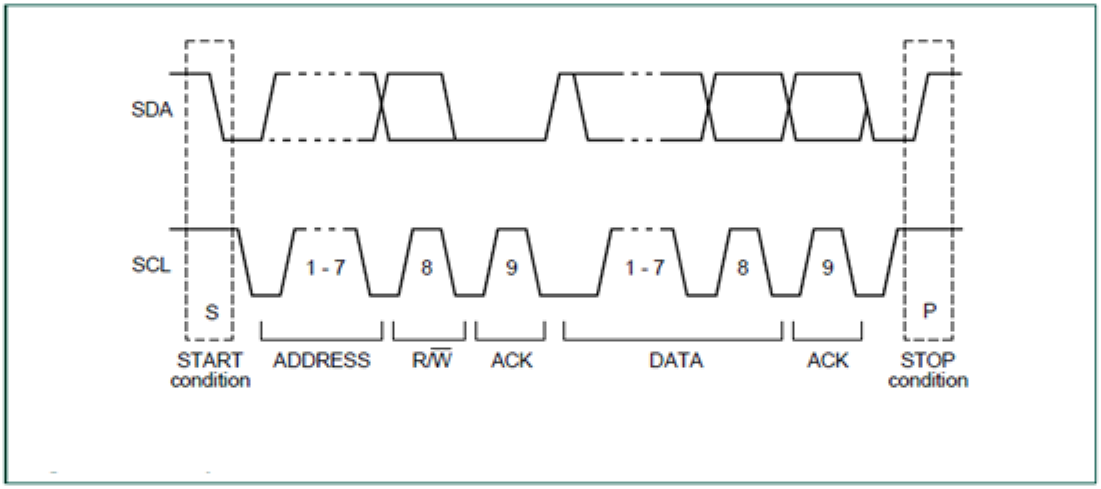
Fig 5. START and STOP conditions

- START and STOP conditions are always generated by the master.
- The bus is considered to be busy after the START condition.
- The bus is considered to be free again a certain elapsed time[8] after the STOP condition.

8. UM10204, I^{2}C-bus specification and user manual, Rev. 6 — 4 April 2014. § 6 Electrical specifications and timing for I/O stages and bus lines. pp 48

I2C Protocol

- Data and Ack



- As mentioned earlier every data byte put on the SDA line must be eight bits long.
- Each byte is followed by an a 9th bit, the acknowledge bit.
- The Acknowledge bit allows the receiver to signal the transmitter that the byte was successfully received and another byte may be sent.
- The master generates all clock pulses, including the acknowledge ninth clock pulse.

I2C Protocol

- Address

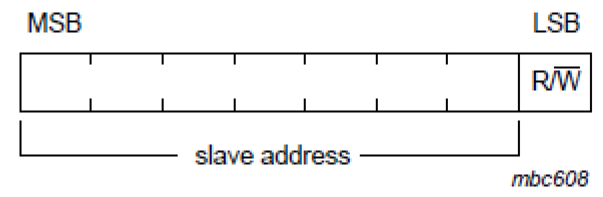


Fig 10. The first byte after the START procedure

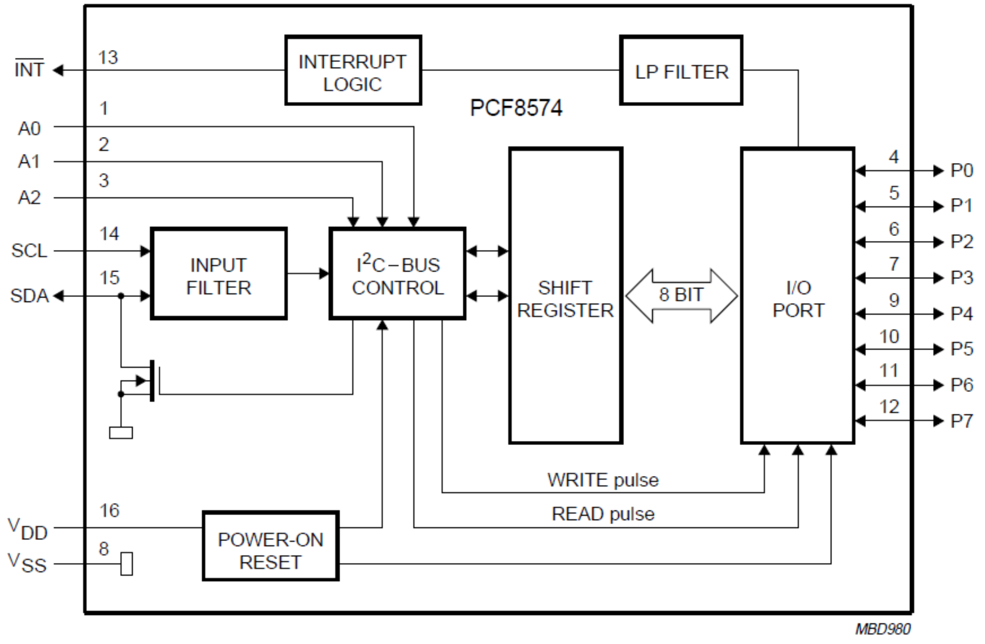
- For seven bit addressing mode, as the name implies, the address is seven bits long followed by an eighth bit which is a data direction bit (R/W)
  - ‘zero’ indicates a transmission (WRITE),
  - ‘one’ indicates a request for data (READ)
- This address is issued by the master

I2C Addressing

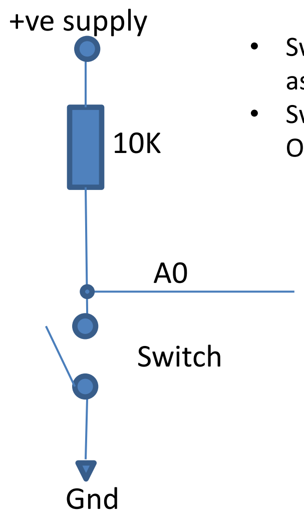
- As I2C addressing can be confusing at times, what continues should add sufficient background to help de-mystify it.
- There are two addressing modes used in I2C;
  - 7bit and 10bit
- As mentioned earlier, this document will cover only 7bit addressing which is by far the most common.
- When considering the use of multiple devices on an I2C bus you must ensure no two devices have the same address.
- For the PCF8574 IO Expander the I2C address is made up of two parts.
  - Fixed (hard coded into the device)
  - User Selectable (usually programmable at the IC)

I2C Addressing

The PCF8574 user selectable portion of the address is set by applying either a ‘High’ or ‘Low’ logic level to pins A0 … A2 respectively[9].



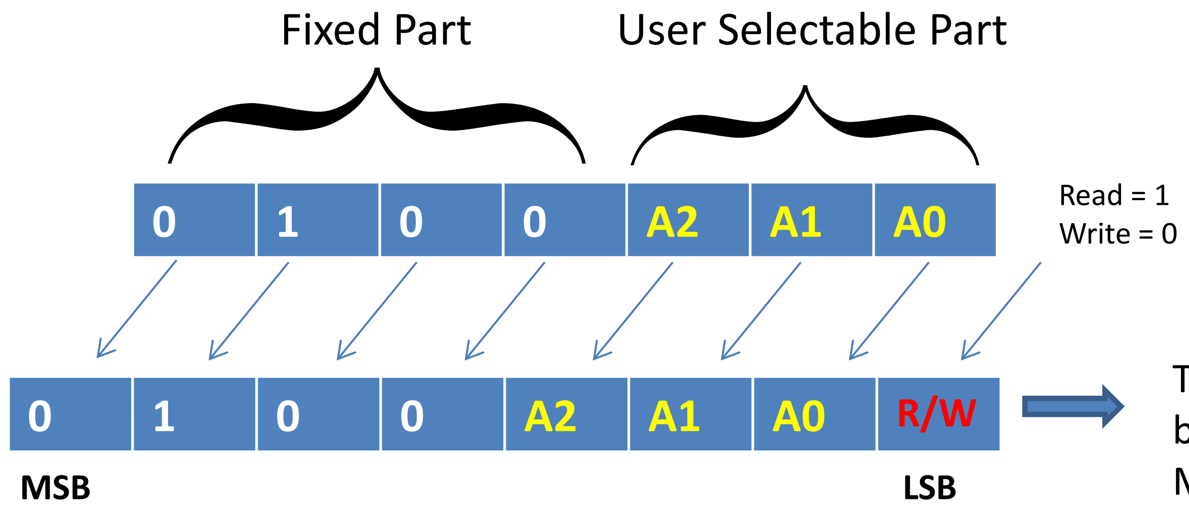
Note : In this context a ‘High’ means pulled up to the +ve supply (Vcc/Vdd), usually through a ‘pull up’ resistor and a ‘Low’ means pulled down to Ground (0V/Vss).



- Switch open, A0 reads as = ‘High’ or ‘1’
- Switch closed, A0 Output = ‘Low’ or ‘0’

9. PCF8574 Remote 8-bit I/O expander for I2C-bus. 2002 Nov 22

I2C Addressing



This is the 7bit address + R/W bit sent out on the SDA line MSB first

- Example Read from Slave address 0x27, B0100111

0	1	0	0	1	1	1	1

- Example Write to Slave address 0x27, B0100111

0	1	0	0	1	1	1	0

Note : The difference between the read and write addresses. Although documentation makes mention of address 0x27 the actual address placed on the SDA line is 0x4F and 0x4E for read and write respectively. In other words it has been left shifted by 1 bit position.

I2C Addressing

For the PCF8574 all the possible device addresses are as follows;

Bus Address Read	Bus Address Write	Address (Binary)	Address (Hex)	A2	A1	A0
01000001	01000000	0100000	0x20	0	0	0
01000011	01000010	0100001	0x21	0	0	1
01000101	01000100	0100010	0x22	0	1	0
01000111	01000110	0100011	0x23	0	1	1
01001001	01001000	0100100	0x24	1	0	0
01001011	01001010	0100101	0x25	1	0	1
01000111	01000110	0100011	0x26	0	1	1
01001111	01001110	0100111	0x27	1	1	1

Read/Write bit

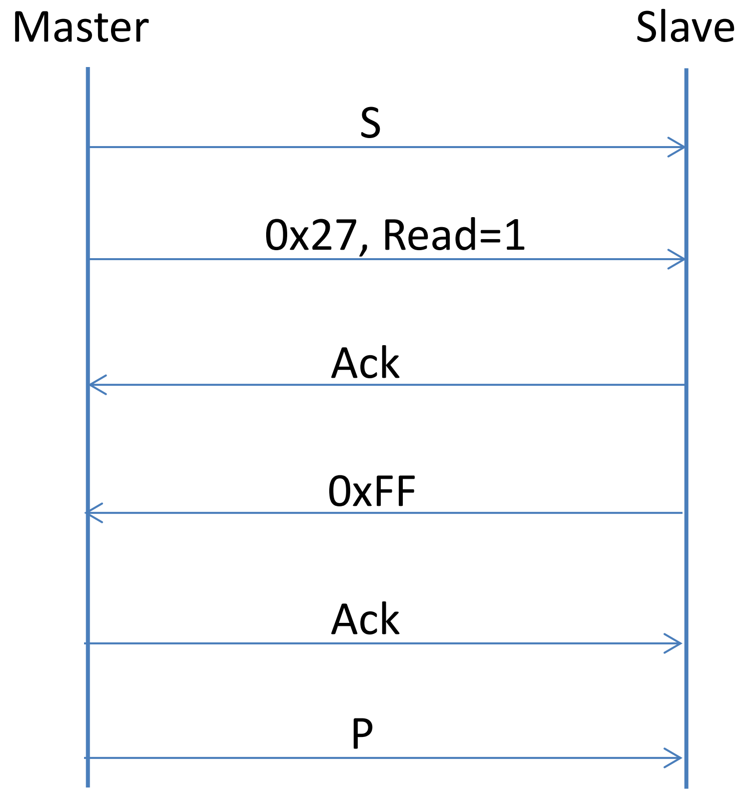
The Arduino Wire library takes care of the necessary bit shifting (ref TWI.cpp).

I2C Byte Reading

- When reading (Master) from an I2C Slave device you will generally need to consider the following;
  - Which Slave device you are reading from (the I2C address).
  - Which register in the slave you wish to read the contents of.
- So this generally will require two pieces of data to be sent on the I2C bus by the Master. The Slave will provide the actual data.
- In the case of the PCF8574 as it only contains one register you don’t need to send a register address.
  - Note : This varies from I2C device to device. You will need to consult the specific data sheet for details of another IC.

I2C Byte Reading

- Pictorially this can be represented by the following message sequence chart[10]



The example on the left assumes the following;

- Slave device : PCF8574
- Slave address 0x27
- Slave read register contains the value 0xFF

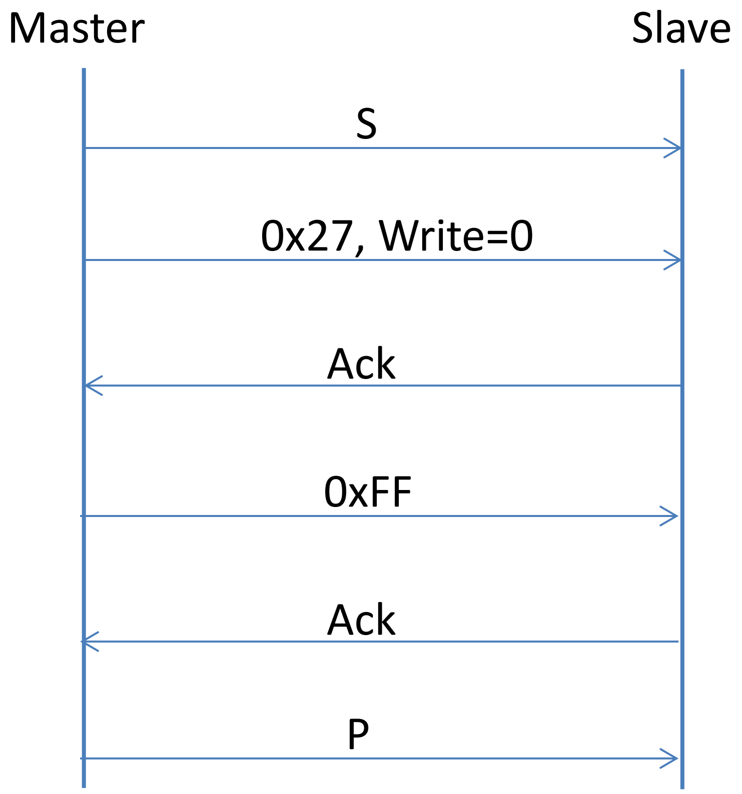
10. UM10204, I2C-bus specification and user manual, Rev. 6 — 4 April 2014. § 3.1.10 The slave address and R/W bit, Fig 12

I2C Byte Writing

- When writing (Master) to an I2C Slave device you will generally need to consider the following;
  - Which Slave you are writing to (the I2C address).
  - Which register in the Slave you wish to write to.
  - What value you wish to write to that register.
- So this will typically require three pieces of data to be sent on the I2C bus by the Master.
- Again in the case of the PCF8574 as it only contains one register you don’t need to send a register address.
  - Note : Just as in the above this varies from I2C device to device.

I2C Byte Writing

- Pictorially this can be represented by the following message sequence chart[11]



The example on the left assumes the following;

- Slave device : PCF8574
- Slave address 0x27
- Master wishes to write the value 0xFF to the Slave register

11. UM10204, I2C-bus specification and user manual, Rev. 6 — 4 April 2014. § 3.1.10 The slave address and R/W bit, Fig 11

Arduino Wire Library

- The Arduino IDE comes with an ‘off the shelf’ library to control I2C. This is known as the ‘Wire’ library;
  - https://www.arduino.cc/en/Reference/Wire
- A lot of the complexity of I2C comms has been removed and is hidden by the library.
- The Arduino ‘Wire’ library works at 100Kbps but can be adapted for higher speeds.
  - See : http://forum.arduino.cc/index.php?topic=16793.0
- It uses 7bit addressing
- It is included in your sketch the following line;
  - #include <Wire.h>

Example I2C Read

```
#include <Wire.h>

void setup() {
  Wire.begin();  // join i2c bus
}

void loop() {
  Wire.requestFrom(0x27, 1);      // request 1 byte from slave device 0x27
  unsigned char c = Wire.read();  // receive a byte
}
```

Example I2C Write

```
#include <Wire.h>

void setup() {
  Wire.begin();   // join i2c bus
}

void loop() {
  Wire.beginTransmission(0x27); // transmit to device 0x27
  Wire.write(0xFF);             // send a single byte with Value 0xFF
  Wire.endTransmission();       // stop transmitting
}
```

Trouble Shooting/Tips

- If your I2C serial display doesn’t show any text but is illuminated, try adjusting the contrast potentiometer.
- Ensure you have SDA to SDA and SLC to SCL lines connected. Also +Vcc and Gnd lines are attached.
- For SDA and SCL ensure the overall pull up resistance is around 4K7Ω - 10KΩ.
- Check when adding more than one modular I2C device, does it come with pull up resistors attached?
- If you are unsure what address has been set on your I2C device, try running the I2CScanner Arduino sketch to ‘sniff’ out any attached devices on your I2C bus.
  - http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter

References

1. http://www.nxp.com/documents/user_manual/UM10204.pdf
2. https://www.arduino.cc/en/Reference/Wire
3. https://en.wikipedia.org/wiki/I%C2%B2C
4. http://www.nxp.com/documents/data_sheet/PCF8574.pdf
5. http://playground.arduino.cc/Main/I2cScanner
6. http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter
7. http://playground.arduino.cc/Code/ATMELTWI#theSource

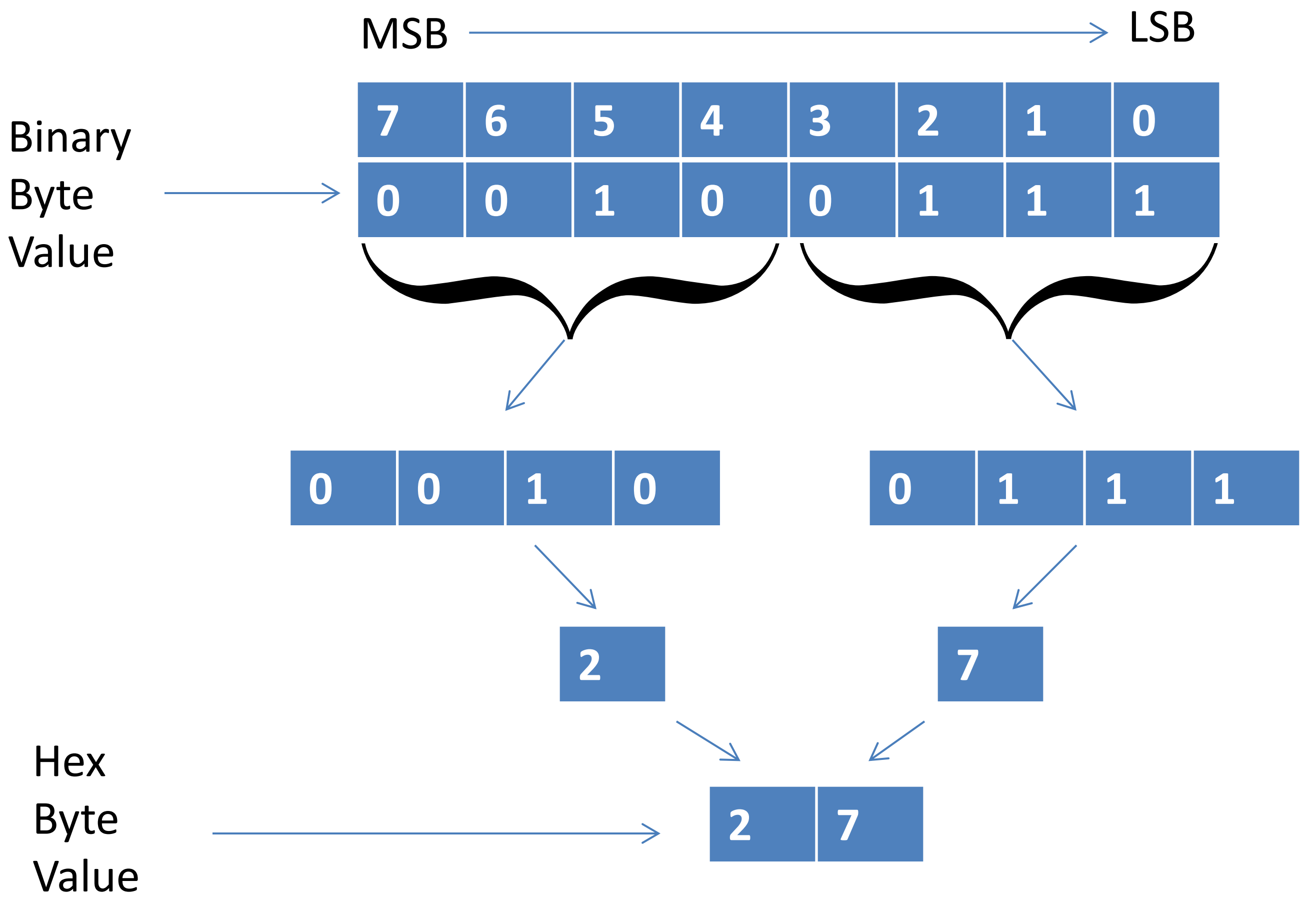
Appendix

- Decimal, Hexadecimal and Binary LUT.

Decimal	Hexadecimal (Hex)	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Appendix

- Converting a Binary byte into a Hex byte



1. Take binary value

2. Split in half. Two Nibbles (1 byte = 2 Nibbles)

3. Convert Binary to Hex. See table above

4. Put the new nibbles Back together

So… $B00100111_2 = 0x27_{16}$

Jargon Buster

- HEX : Hexadecimal, base 16 numbering system
- IDE : Integrated Development Environment
- LCD : Liquid Crystal Display
- LSB : Least significant bit
- LUT : Look Up Table
- MSB : Most significant bit
- SCL : Serial Clock Line
- SDA : Serial Data Line

System Configuration

- The ‘LiquidCrystal_I2C_PCF8574’ library was written using the following system configuration.
  - Arduino Uno R3 (Original)
  - Windows 7 64bit
  - Arduino Software IDE v1.6.5
- For excel spread sheet ‘CharGen1.xlsm’
  - Microsoft Excel 2010 was used.

Disclaimer