EE 2361 - Introduction to Microcontrollers

Laboratory #5

I2C - Interfacing with a Display

Background Reading Material

Background

Serial interfaces are very common in industry today. They allow for low pin count communication between integrated circuits (ICs, like the PIC24). As you've seen in Lab 2, there are custom serial protocols that can be implemented by manually "bit-banging" individual control line(s). You've also created sequential logic machines in EE2301 that handle some functions autonomously (like the multiplier, just give it a couple numbers and let it do the work.) These autonomous functions when implemented for you in the hardware of a microcontroller are called peripherals. Our PIC24FJ64GA002 microcontroller has peripheral support for three serial communication standards UART (often called just "serial" or RS-232 serial), SPI (Serial Peripheral Interface), and I2C (Inter-IC Control, sometimes called I^2C). Each of these standards are useful and have their own advantages.

	Advantages	Limitations
UART	Legacy Support Full Duplex - No master	Max 2 devices Requires precise timing
I2C	2 pins Addressable Fast (>1Mbps)	Knowledge of slave addr. Requires working slave to debug
SPI	Full Duplex Very Fast (~50Mbps)	2-pins + 1 per slave

Note: I2C is sometimes called SMBus, see this[1] app note by Maxim. There are only a couple minor differences between the specifications.

The decision on which standard to use for a project is usually fairly arbitrary. Sometimes the pin count and the data rate will dictate which system to use. Other times, it is simply a matter of which protocol is supported by the non-microcontroller based chips in the system.

This lab will concentrate on I2C as it is a commonly available modern protocol (and its needed for our Serial LCD display.) However, being able to interpret the datasheet and learn to independently use either of the other two supported protocols is an important part of being a good MCU programmer. Use what you learn in this lab and don't shy away from learning to use the other protocols in your project and future work. Doing so will cement your understanding of the necessities of how serial protocols operate.

[1] http://pdfserv.maximintegrated.com/en/an/AN476.pdf

Lab 5 Reading

An I2C Bit Stream

In I2C there is a master and at least one slave. Our PIC24 will be the master and the LCD display will the be slave. I2C specifies a two-wire interface where one wire is the serial clock (SCL) and the other is the serial data (SDA). The SCL wire is always controlled by the master. SDA is controlled by both the master and the slave. This ability to share a pin is created using what is called "open-drain" or "pull-down" outputs shown in the figure below.

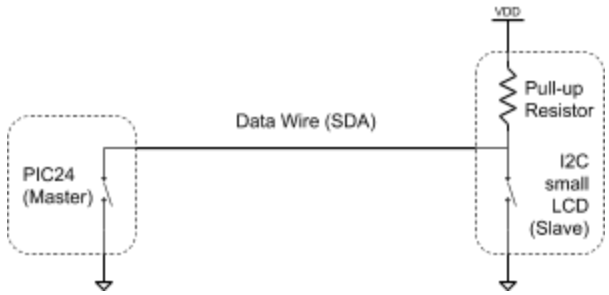
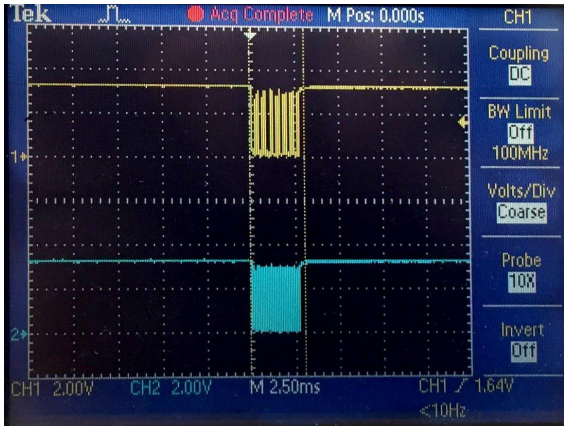


Diagram of the data wire interface for I2C

In an "open-drain" configuration, the Master and the Slave each control a switch to ground. This switch when it gets a HIGH/ON signal it will force the output (SDA) to be low (ie., it is inverting). When the switches are both LOW/OFF, the wire will be passively pulled HIGH by the pullup resistor. In this way multiple chips can share one wire for communication in both directions.

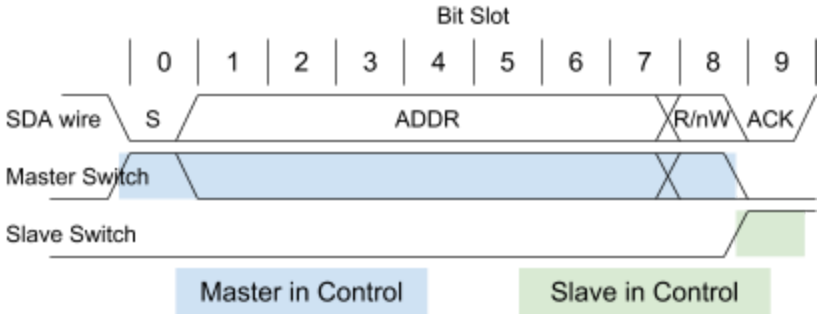
As you can see from the packet below, the SDA line (top) and SCL line (bottom) are normally high via the pull-up resistor. It is a fairly common mistake with I2C to forget to install pull-up resistors. However we're in luck, our LCD Display comes with internal pull-up resistors installed (see schematic).



Example of I2C communication packet

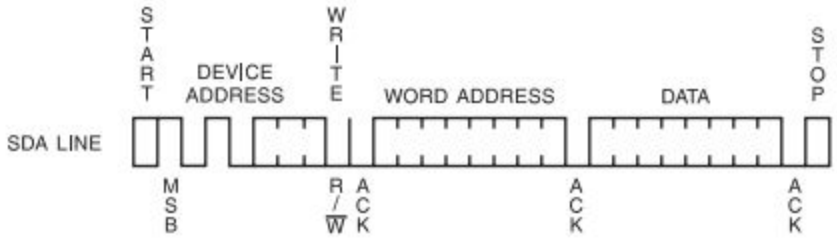
The I2C communication standard is broken into packets. The first part of an I2C communication packet is split into four sections, a start bit (S), a 7-bit address (ADDR), a read or not_write bit (R/nW), and an acknowledge bit (ACK).

Below you can see a timing diagram of the initial part of a data packet. The master is in control of the bits 0 through 8 and the slave is in control of the 9th bit. Not shown here is the clock wire that is entirely in the control of the master and tells both the master and the slave when each bit starts and stops.



Addressing part of an I2C communication packet

The two general types of communication that can follow the addressing part of the I2C packet above are a read or a write. Our LCD display generally only needs write packets, at least for now. Each individual I2C device will have a slightly different packet structure beyond this first part of the packet. The most basic structure of a write packet is shown below.



Example I2C Write[2]

This packet, consists of the following structure:

- The PIC sends a device address to get the LCD's attention, then waits
- The LCD pulls SDA low providing an ACKnowledge signal
- The PIC then sends a memory word address (ACK'd by the LCD)

[2] http://www.engscope.com/pic24-tutorial/10-2-i2c-basic-functions/

- The PIC then sends data to be stored in that address (also, ACK'd by the LCD)

Microchip I2C Peripheral Implementation

In order to execute the bit stream above Microchip has implemented dedicated hardware. This hardware is described in the PIC24_FRM_I2C.pdf datasheet. There are two dedicated hardware peripherals (I2C1 and I2C2) onboard our PIC24FJ chip. These peripherals are hard connected to pins SDA1/RP9/RB9 and SCL1/RP8/RB8; SDA2/RP2/RB2 and SCL2/RP3/RB3 respectively. We will be utilizing the I2C2 peripheral.

The most important registers in controlling the I2C2 peripheral for write are the following:

- I2C2BRG - I2C2 Baud Rate Generator Register
- I2C2CON - I2C2 Control Register
- I2C2TRN - I2C2 Transmit Register
- I2C2STAT - I2C2 Status Register

When read and written appropriately these registers can handle all of the master-side bit banging for a single "part" of a packet. All the microprocessor has to do is wait until the transaction is complete, then repeat the process for each "part" of the packet, slowly constructing a full packet.

The Baud Rate Generator is a timer-like circuit that controls the frequency of the clock pin (SCL). Before beginning the transmit process the peripheral needs to be set up. Please, review the PIC24FJ64GA002 Family Datasheet (section 16.3, page 153) as it will be needed in the procedure below.

The I2C2CON register controls all the actions that are needed to operate the I2C bus. Each bit represents a configuration option or an action to be taken on the bus.
The following are the core configuration bit(s):

- `I2C2CONbits.I2CEN`: Enable

The following are the core actions available:

- `I2C2CONbits.SEN` - Begin (S)tart sequence
- `I2C2CONbits.PEN` - Begin Sto(P) sequence

Additionally, writing to the I2C2TRN register will shift out 8 bits:

- `I2C2TRN` - Write 8 bits

Lastly, there is an interrupt flag for I2C:

- `IFS3bits.MI2C2IF` - Set when an I2C action is complete

The following pseudocode shows a typical sequence of actions to create a valid write packet:

```
SEN = 1;   // Send (S)tart Sequence
Wait;
I2C2TRN = <slave_chip_addr><R/nW>
Wait for ACK;
```

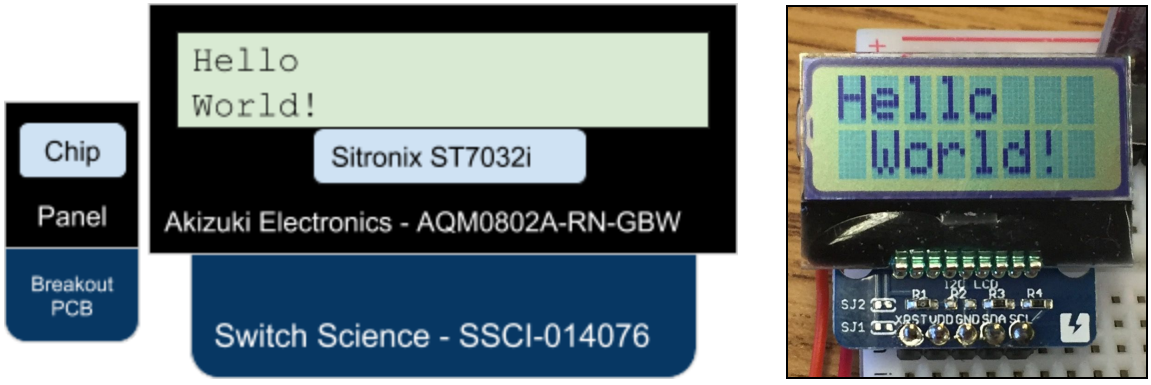
```
I2C2TRN = <slave__mem_addr>
Wait for ACK;
I2C2TRN = <data_for_slave>
Wait for ACK;
PEN = 1;  // Send Sto(P) Sequence
Wait;
```

As you can see above, the user must issue an action then wait for it to complete. Since data rates are low (100kHz vs. our 16MHz processor) and there can be undefined pauses to wait for the slave to acknowledge the transmission, it is generally undesirable to block MCU operation during I2C transmissions. The I2C2STAT register can be polled or interrupts used to determine if the transmission needs the attention of the MCU.

> A major advantages of I2C is that the master controls all timing. Slave devices will wait until the MCU is ready to proceed to the next action of the packet.

LCD Module Overview

The Switch Science Small - LCD module is actually constructed by three different manufactures. Switch Science makes the breakout PCB that plugs into your breadboard (2.54mm pins). The LCD panel is made by http://akizukidenshi.com/ (1mm pins). Finally, the chip on the LCD panel is made by Sitronix (BGA IC under the black sealant).



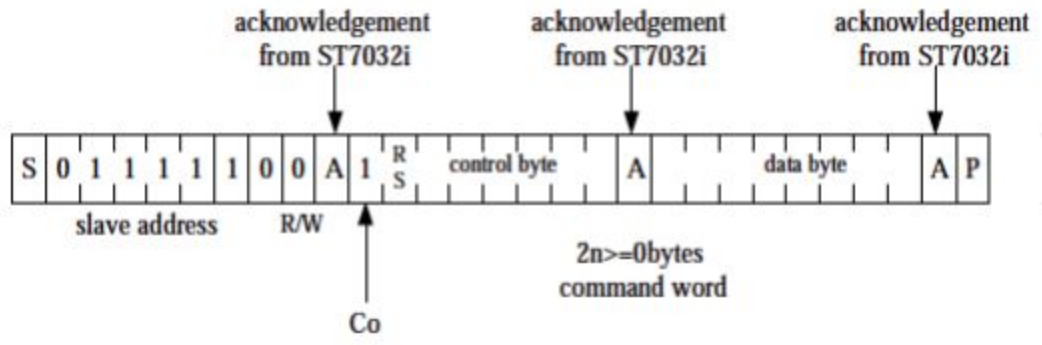
Hierarchy of Switch Science Small LCD display

In order to develop the code to operate this LCD we will utilize information from all three of these manufacturers.

Sitronix ST7032 - I2C write packet structure

The Sitronix ST7032 is the chip inside the packaged LCD display. The most complete documentation for this chip is contained in the ST7032 Datasheet, but much of the required information for basic operation of the LCD is replicated in the Akizuki LCD module datasheet.

The clearest description of the required write packet structure is available in both documents and replicated below for your reference:



ST7032 - I2C write packet description[3]

A write packet consists of the slave address as discussed previously, followed by a control byte (usually 0b00000000), then a data byte (sometimes called a command byte). Finally a stop bit terminates the transmission.

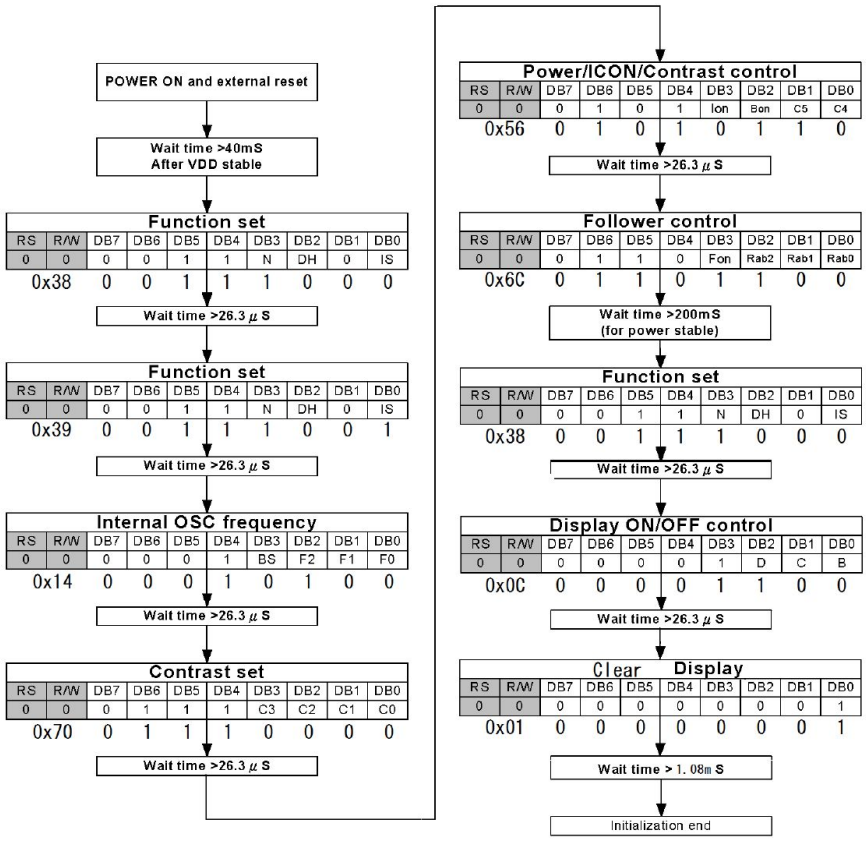
To execute this bit sequence with the PIC24 would consist of the following pseudocode:

```
void lcd_cmd(char Package) {
SEN = 1;       //Initiate Start condition
Wait for SEN == 0  // SEN will clear when Start Bit is complete
I2C2TRN = 0b01111100; // 8-bits consisting of the slave address and the R/nW bit
Wait for IFS3bits.MI2C2IF == 1
I2C2TRN = 0b00000000; // 8-bits consisting of control byte
Wait for IFS3bits.MI2C2IF == 1
I2C2TRN = Package; // 8-bits consisting of the data byte
Wait for IFS3bits.MI2C2IF == 1
I2C2CONbits.PEN = 1;
Wait for PEN==0 // PEN will clear when Stop bit is complete
}
```

Sitronix ST7032 / AQM0802 LCD Module - Startup Sequence

The LCD module needs several commands to be issued via the I2C bus in order to initialize correctly. These commands are simplified and included in a diagram in the Akizuki LCD module datasheet on page 2. This diagram is replicated below:

[3] Modified from ST7032 Datasheet, page 13



AQM0802 / ST7032 - Setup procedure for the Small LCD

As can be seen above, the procedure completes several commands including, setting up the oscillator frequency trim (internal LCD frequency, not I2C), enabling the display driver, setting up temperature controlled follower, and adjusting the contrast. We will follow all of these steps blindly, except the setting of contrast. More details of these commands and how they function see the ST7032 Datasheet (page 21).

Each of these commands consists of an instruction bit pattern, followed by several control bits. In the case of contrast control, the LCD display is controlled by 6 bits across 2 control commands. These commands are shown below:

Contrast set

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	1	1	C3	C2	C1	C0
0x70		0	1	1	1	0	0	0	0

Power/ICON/Contrast control									
RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	0	1	Ion	Bon	C5	C4
0x56		0	1	0	1	0	1	1	0

Contrast control commands for Small LCD display

In order to issue each command, a user needs to write a single I2C packet. The packet consists of the slave address+R/nW (8-bits), control byte (8-bits), and data byte (8-bits). The two grey bits in the figure above are the R/nW bit and the 2nd bit in the control byte. The white bits in the figure above are the data or command byte.

The first four bits of these command bytes are fixed (0111 and 0101 respectively), they tell the LCD display what command/instruction you are issuing. The contrast is set by the bits C0 through C5. Ion=0 and Bon=1 should always remain the same.

All of these bits need to be constructed into a valid I2C write packet as shown in the ST7032 - I2C write packet description figure. At the end of the day each command should look a lot like the pseudo-code shown above. Then the setup procedure becomes:

```
wait 50ms;
lcd_cmd(0b00111000); // function set, normal instruction mode
lcd_cmd(0b00111001); // function set, extended instruction mode
lcd_cmd(0b00010100); // interval osc
lcd_cmd(0b01110111); // contrast C3-C0
lcd_cmd(0b01011101); // C5-C4, Ion, Bon
lcd_cmd(0b01101100); // follower control
wait 200ms;
lcd_cmd(0b00111000); // function set, normal instruction mode
lcd_cmd(0b00001100); // Display On
lcd_cmd(0b00000001); // Clear Display
```

Pseudo-code for LCD Setup

AQM0802 LCD Module - Character Display Commands

The small LCD display draws pictures of character patterns stored in a ROM based on a frame buffer called “DDRAM”. The DDRAM frame buffer contains one 8-bit value for each location on the LCD Display (8x2 characters). The 8-bit value selects a character from roughly compatible with the ASCII characters standard (c-style “char”). For example, when you type `char foo = "A";` it gets stored in memory as 0b01000001 or 0x41.

The available character patterns are shown below:

CHARACTER PATTERNS

b7-b4 / b3-b0	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CG RAM															
0001																
0010																
0011																
0100																
0101																
0110																
0111																
1000																
1001																
1010																
1011																
1100																
1101																
1110																
1111																

Character Patterns for AQM0802 LCD Module

In order to write a character to the display we simply create an I2C write packet with the character we want to display. Looking through the commands available (page 19) we want the "Write data to RAM" command shown below:

Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0

Note that this is the one command that requires "RS" to be set to 1. The RS bit is actually the second bit in the control byte of the I2C packet (the normally 0b00000000 part). (see ST7032 - I2C write packet description) Putting this together we have the following pseudocode:

```
void lcd_printChar(char Package) {
SEN = 1;        //Initiate Start condition
Wait for SEN == 0  // SEN will clear when Start Bit is complete
I2C2TRN = 0b01111100; // 8-bits consisting of the slave address and the R/nW bit
Wait for IFS3bits.MI2C2IF == 1
I2C2TRN = 0b01000000; // 8-bits consisting of control byte /w RS=1
Wait for IFS3bits.MI2C2IF == 1
I2C2TRN = Package; // 8-bits consisting of the data byte
Wait for IFS3bits.MI2C2IF == 1
I2C2CONbits.PEN = 1;
Wait for PEN==0 // PEN will clear when Stop bit is complete
}
```

Consecutive Writes to Memory Locations

The above sections describe how to write individual commands (address byte, control byte, and data byte) used to setup the display. In order to push an entire string to the display, for example, "Hello World!", we will need the ability to write multiple memory locations quickly. Luckily I2C and the AQM0802/ST7032 support consecutive memory writes (without re-addressing).

Looking at the ST7032 Datasheet (page 13, replicated below) the Co bit in the control byte specifies if the current control byte / data byte pair are the final pair in the data stream.

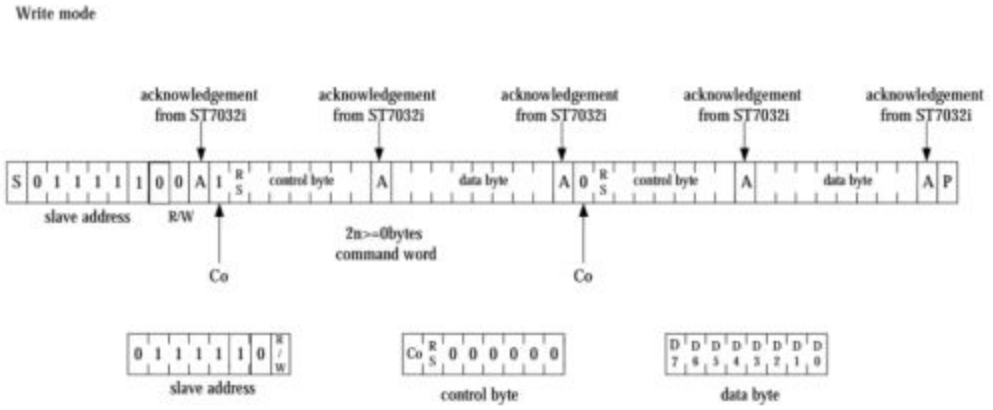


Figure 5. 2-line Interface protocol

Co	0	Last control byte to be sent. Only a stream of data bytes is allowed to follow. This stream may only be terminated by a STOP condition.
	1	Another control byte will follow the data byte unless a STOP condition is received.

ST7032 Datasheet - Modified Figure 5 (page 13)

If we write a control byte of 0b11000000 (ie., Co = 1 and Rs = 1) then it tells the ST7032 that this is the first of several writes. We follow that with a control/data byte pairs for representing the remainder of the string. The final byte must be sent with a control byte of 0b01000000. Finally, we finish with a STOP bit.

In pseudo code and just showing the I2C “actions”, this becomes:

```
START BIT
0b01111100 //ADDRESS BYTE (R/nW = 0)
0b11000000 // Control Byte: CO = 1 RS =1
0x46 // 'F'
0b11000000 // Control Byte: CO = 1 RS =1
0x6F // 'o'
...
0b01000000 // Control Byte: CO = 0 RS =1 "last byte"
0x6F // 'o'
STOP BIT
```

Memory Location Pointer

We now have the ability to send control commands and write data to the the LCD display. The last piece of the puzzle is the how the memory on the LCD display is organized and how to address particular locations.

The Small LCD panel is an 8 character wide by 2 character tall display.

Picture of LCD showing 12345678/012345678

The frame buffer on the ST7032 is arranged as follows:

	Column									
	0	1	2	3	4	5	6	7		
DDRAM Address	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0	Row
	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	1	

The location of a write to the frame buffer is determined by the address counter (AC). After reset the AC is set to 0x00 (upper left most character location). It can also be changed by the Set DDRAM address command (see the ST7032 Datasheet - page 19).

Set DDRAM address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0

Instruction format to change cursor location (see page 19 of ST7032 specification)

In order to set the physical location of a character to be displayed the AC register value must be determined from the intended x and y coordinates. An equation can map this relationship. For example, the address to write row 1, column 5; 0x40*(row) + (column) = 0x45. Notice however, that the MSB is always 1 in the instruction code for the “Set DDRAM address” command. This results in the 0x45=0100 0101 → 1100 0101 or 0xC5.

Putting this into psuedocode:

```
START BIT
0b01111100 //ADDRESS BYTE (R/nW = 0)
0b00000000 // Control Byte: CO = 0 RS =0
0b11000101 // 0xC5 (set AC to 0x45)
```

The default mode for the Small LCD is to automatically increment the write address by one. This has the effect of easily allowing a user to write up to 8 characters consecutively (a string) without having to change the address. Note that because the display driver is designed to support many different 2-line display lengths (upto 40 characters wide), continuing to write off the end of our display will not wrap onto the next line, but instead just be written into unused memory (DDRAM).

There are additional addressing modes that can create very nice text scrolling effects easily. Take a look at the “Entry Mode Set” and “Cursor or Display Shift” commands on pages 21 and 22 respectively.