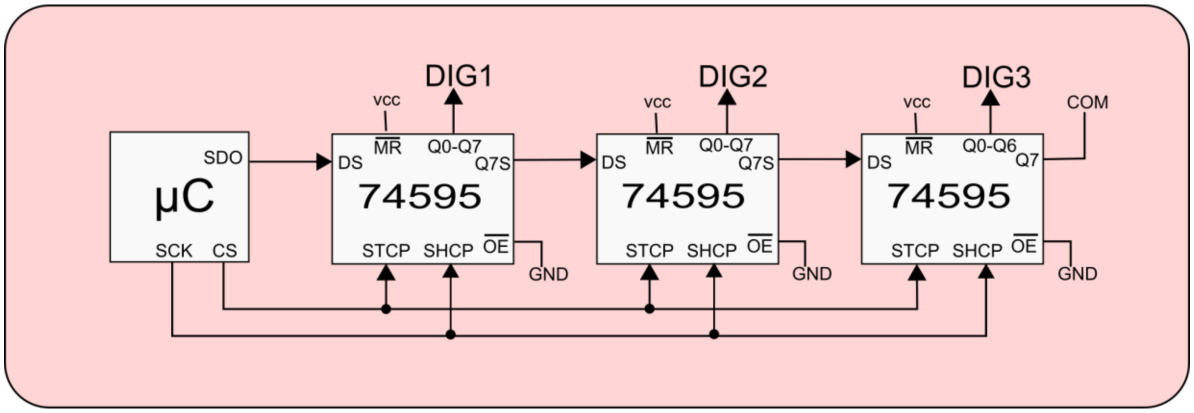
Lab 10 - Serial Peripheral Interface/ LCD Display

In this lab you are going to learn about how to use LCD displays and practice reading some datasheets.

Part 1 – Connecting the Chips

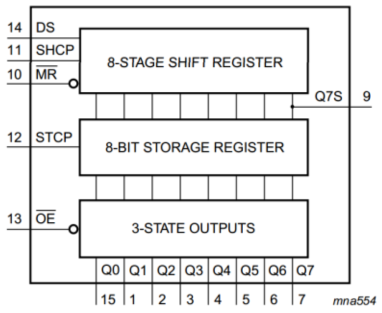
Often when wiring up a circuit we will just have a block diagram. Block diagrams are used to clarify the big picture when looking at a circuit. Below is a block diagram for the LCD display that we will be hooking up to the microcontroller.



To decode this diagram must first identify all the blocks:

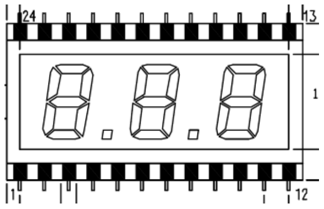
Block Diagram	Description
uC	uC is short for microcontroller, in our case the PIC32 Fubarino board. To find the pinout of this device you are using refer to the pinout diagrams from lab 2.
74595	There are three of these devices shown each is its own chip. To find the pin outs of this chip refer to the diagram below.
DIG1, DIG2, DIG3	These are the three digits of the LCD display. These all come from a single pinout for the LCD shown below.

The pinout for the 74595 is shown to the left. This pinout shows only the logical I/O for the device. Don't forget to hook up power and ground for each chip. Pin 8 is GND and pin 16 is +V (5V Vout). You can power these chips with 3.3V or 5V but the 3.3V regulator on the Fubarino gets warm if we use 3.3V due to the high current draw of these chips, so be sure to power them with 5V.



The pinout for the LCD is shown in the following chart:

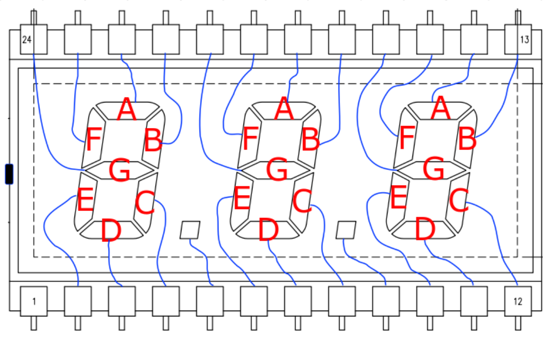
PIN	1	2	3	4	5	6	7	8	9	10	11	12
SEGMENT	COM	E1	D1	C1	DP1	E2	D2	C2	DP2	E3	D3	C3
PIN	13	14	15	16	17	18	19	20	21	22	23	24
SEGMENT	B3	A3	F3	G3	B2	A2	F2	G2	B1	A1	F1	G1



At first glance the pinout can seem confusing. But if you think for a moment what is being presented you can see that the first row is a list of pins, and the second row is a list of segments. This is again repeated on the third and fourth row of the device. The location of the segments may seem random but if we look closer at the construction of the LCD we can see that there placement makes sense from the design stand point of the LCD designer.

If present the pinouts next to the display and overlay the segment names (in red) along with a possible electrode placements (in blue) then we can see that the pinout flows from the shortest path from the pin to the segment that is associated with it:

PIN	13	14	15	16	17	18	19	20	21	22	23	24
SEGMENT	B3	A3	F3	G3	B2	A2	F2	G2	B1	A1	F1	G1



PIN	1	2	3	4	5	6	7	8	9	10	11	12
SEGMENT	COM	E1	D1	C1	DP1	E2	D2	C2	DP2	E3	D3	C3

So, the name A1, means segment A of digit 1, and B1 means segment B of digit 1 and so on… Pin 1 or COM is the common electrode that is shared among all pins and is also known as the backplane.

So now if we look back at the block diagram we can see that we need to connect the microcontroller to the shift registers (3 jumpers) and CS and SCK are connected in parallel so additional jumpers are needed (2 signals x 2 more chips = 4 jumpers). The shift registers need power and ground (3 x 2 = 6 jumpers) and that there are two pins on each shift register that need to be pulled high or low (3 x 2 = 6 jumpers). Then eight pins of each shift register need to be connected to the eight pins of the LCD display (3 x 8 = 24 jumpers). In total 43 jumper wires are needed to wire this circuit. All the components require two long bread boards to complete the wiring.

Test Software

To test the above circuit some minimal amount of code is needed to setup and shift data out to the LCD. Below is a class that is provided. We can use this class from our parser presented in previous labs. To do this we can create a new tab named "LCD.h" and copy this code into that tab.

```
uint8_t LCDS301C31TR_map [16] = { 0x3f };
class LCDS301C31TR
{
    void lcd_out(uint32_t value)
    {
      digitalWrite(LCDS301C31TR_cs,LOW);
      SPI.transfer((uint8_t)( (value >>  0) & 0xff ));
      SPI.transfer((uint8_t)( (value >>  8) & 0xff ));
      SPI.transfer((uint8_t)( (value >>  16) & 0xff ));
      digitalWrite(LCDS301C31TR_cs,HIGH);
      }
    uint8_t LCDS301C31TR_cs;
  public:
    uint32_t lcd_value;
    LCDS301C31TR(uint8_t cs)
    {
      pinMode(cs, OUTPUT);
      LCDS301C31TR_cs = cs;
      SPI.begin();
    }
    void lcd_present()
    {
      static uint8_t inverted = 0;
      if( inverted )
      {
        lcd_out( ~lcd_value );
        inverted = 0;
      }
      else
      {
        lcd_out( lcd_value );
        inverted = 1;
      }
    }

    uint8_t lcd_control( char* command )
    {
      if( strncmp( command, "lcd.", 4 ) == 0 ){
        char *sub_command;
        sub_command = &command[4];

        if( strcmp( sub_command, "inc" ) == 0 ) {
          if( lcd_value == 0 )
            lcd_value = 1;
          else
            lcd_value <<= 1;
          lcd_out( lcd_value );
        }
        else
        {
          return 0; //return false if lcd command not found
        }
        return 1; //return true if lcd command and found
      }
      else
        return 0; //return false if not lcd command
    }

    void lcd_display_hex(uint16_t value) {
      lcd_value = (
              ((uint32_t)LCDS301C31TR_map[(value >> 8) & 0x0f]) << 16 |
              ((uint32_t)LCDS301C31TR_map[(value >> 4) & 0x0f]) <<  8 |
              ((uint32_t)LCDS301C31TR_map[(value >> 0) & 0x0f])
            );
      lcd_present();
    }
    void lcd_display_base10(uint16_t value) {
      uint16_t v100 = value / 100;
      uint16_t v10 = (value - (v100 * 100)) / 10;
      uint16_t v1 =  (value - (v100 * 100) - (v10 * 10));

      lcd_value = (
              ((uint32_t)LCDS301C31TR_map[v100]) << 16 |
              ((uint32_t)LCDS301C31TR_map[v10]) <<  8 |
              ((uint32_t)LCDS301C31TR_map[v1])
            );
      lcd_present();
    }

};
```

To use this code, at the top of your parser add the following code to include the SPI library, the new LCD library and create an instance of the LCDS301C31TR class. The LCDS301C31TR instance created is an object with the name LCD. When the object it created we pass in the value of 9 to identify that we are going to use pin 9 of our board as the chip select pin.

```
#include <SPI.h> //use the SPI library
#include "LCD.h" //needed because it is .h
LCDS301C31TR LCD(9); //create instance of class from LCD.h
                     //with pin 9 as chip select
```

<u>Add a call to LCD.lcd_present() at the start of the loop function before the parser so that it is called every cycle. A delay (10) after this call may also be needed to lower flicker on the display.</u>

Finally add the following to the commands in your parser

```
    else if (LCD.lcd_control(command)); //just one line adds the commands
                                        //contained in lcd_control
```

LCDS301C31TR Class API

The LCDS301C31TR class as presented above provide these following public attributes and methods. Remember to access any of these public methods you just need to do so by using the declared object called LCD followed by a decimal point then the attribute or method name.

`uint32_t lcd_value;`

Thirty-two-bit attribute that holds the current 24-bit value (raw decoded bits) that is being presented to the LCD. The values stored in this variable can be visualized from MSB to LSB with the chart below.

lcd_value																															
UNUSED								Digit 1								Digit 2								Digit 3							
								DP1	g	f	e	d	c	b	a	DP2	g	f	e	d	c	b	a	COM	g	f	e	d	c	b	a

To access this value, you could do the following:

```
   Serial.println(LCD.lcd_value, BIN);
```

`LCDS301C31TR(uint8_t cs);`

The constructor for the class. This I classed with the object is declared at the top of the program.

`void lcd_present();`
The function inverts the lcd_value and sends it out the SPI port. This is the function that provides the AC values for the LCD display to prevent the segments from blurring over time.
Usage:

```
   LCD.lcd_present();
```

`uint8_t lcd_control( char* command );`
The function provides an extension to the parser by first looking for commands that start with “lcd.” then parsing out post decimal point data. As implemented in this part of the lab there is a single lcd command that is available: “lcd.inc”. This command will shift a single bit through each segment of the LCD to allow testing of the wiring.

`void lcd_display_hex(uint16_t value);`
Display a value on the LCD display in hexadecimal.
Usage:

```
   LCD.lcd_display_hex(0x123);
```

`void lcd_display_base10(uint16_t value);`
Display a value on the LCD display in base 10.
Usage:

```
   LCD.lcd_display_hex(123);
```

Troubleshooting

If you cannot get the lcd.inc command to work confirm at a minimum these things:

1. The 74595 IC’s are powered (pins 8 and 16)
2. SCO, SCK and CS are connected from the microcontroller to the first shift register
3. Comment out LCD.lcd_present() and disconnect LCD COM (pin 1) from the shift register and connect it to ground. If the COM is ground and shift register are wired correctly then issuing the lcd.inc command will light up one segment at a time until (24 or of 32 calls). After the 32 call the sequence repeats.

Check off

Run program and verify it and the LCD are working. Test that the "lcd.inc" command moves the active pixel on the LCD. Call the instructor demonstrate your LCD in action.
Be prepared to show where the "lcd.inc" command came from and explain how it works.

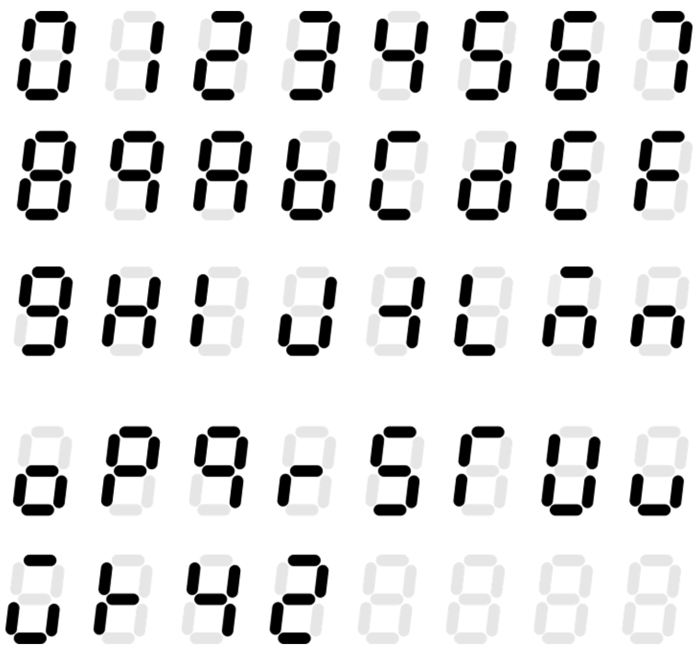
Part 2 – Character Bitmaps

Create a bitmap for each character in the provided table.

Looking at this lecture notes for this lab, for pictures and the order of bits determine Hex codes for all numbers and letters. Put them in the provided worksheet.

Value	p	g	f	e	d	c	b	a	Hex
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									
A									
B									
C									
D									
E									
F									
G									
H									

Value	p	g	f	e	d	c	b	a	Hex
I									
J									
K									
L									
M									
N									
O									
P									
Q									
R									
S									
T									
U									
V									
W									
X									
Y									
Z									



Check off

Call the instructor over see your values.

Part 3 – Using the Codes

Make Array

In the lcd.h file there is a nearly empty array use the codes you just generated to fill in the array for 0-F so that the lcd_display_hex function has data to use.

Add calls to lcd_display_hex in "count up" and "count down" so that the numbers are displayed on the LCD also.

Check off

Call the instructor over see your program and LCD in action.

Part 4 – Display your Initials

Add a command called “lcd.me” to the ***<u>lcd_control</u>*** code

Assign the hex values for your initials directly to the variable lcd_value so that they show on the LCD

```
else if( strcmp( sub_command, "me" ) == 0 ) {
  lcd_value = LCDS301C31TR_map['J' - 'A' + 10] << 16 |
              LCDS301C31TR_map['S' - 'A' + 10] << 8 |
              LCDS301C31TR_map['C' - 'A' + 10] << 0;
}
else
```

Check off

Call the instructor over see your program in action.

Part 5 – Clear the display

Add a command called “lcd.clear”

Assign the hex values to blank out the LCD by directly writing to the variable lcd_value so that LCD will blank out when issued

```
else if( strcmp( sub_command, "clear" ) == 0 ) {
  lcd_value = 0;
}
```

Check off

Call the instructor over see your program in action.

Part 6 – Create a LCD chase pattern

Add a command called “lcd.chase”
Cause the LCD to display a chase pattern of your choice

```
        else if( strcmp( sub_command, "chase" ) == 0 ) {
            // loop (while or for)
          // in loop call lcd_present();
        }
```

Check off

Call the instructor over see your program in action.

Part 7 – Display ADC on LCD

Place in your code a call to lcd_display_hex that will continuously update the LCD with the output from an ADC connected to a pot.

Check off

Call the instructor over see your program in action.

Part 8 – Volt Meter

Make a copy of the lcd_display_hex function and modify it to output base10 instead of hex.
Replace the direct ADC value with one using base 10 converted to volts.

```
LCD.lcd_display_base10(((uint32_t)analogRead(A7)*330)/1023); //convert adc to
volts*100
LCD.lcd_value=LCD.lcd_value | 0x800000; //add the decimal point
```

Check off

Call the instructor over see your program in action.