CG1112 Engineering Principles and Practices II for Computer Engineering

Week 3 Studio 2 – Soldering

1. Introduction

The objective of this studio is to learn to solder, and for that you will build a serial-to-parallel converter circuit on stripboard.

Typically when developing a new piece of hardware you would breadboard it first to test the circuit design, then prototype it on a board like the stripboard or protoboard, before going on to designing a custom built PCB. Due to lack of time we have skipped the breadboarding part, but you are all familiar with breadboards in EPP-1.

For this studio you will be getting:

Whiteboard marker	x1
330 ohm resistors, 0.25W	x8
Red LED 5mm	x8
74HC595 serial to parallel converter	x1
16-pin DIP socket	x1
40-way M/F ribbon cable	x1
Pin Header	x1
Stripboard	x1
Stripboard cutter tool	x1
Wire cutter/stripper	x1
Arduino UNO	x1
Soldering iron	x1
Soldering stand	x1
Desoldering sucker pump	x1
Some solder	
Wires for soldering	

2. Team Setup

Please seat in your final project teams.

3. Activities – Building a Serial to Parallel Converter

In the following activities you will learn how to use the 74HC595 shift register to display the binary equivalent of a number.

Binary Numbers

But whoaa… what are binary numbers? Put simply, a computer system can only represent ALL data (including text and graphics) as integers, and these integers must in turn be represented as 0's and 1's. This is because 0's and 1's are conveniently represented by 0 volts and 5 (or 3.3) volts respectively. Since the distance between 0 volts and 5 volts (or 3.3 volts) is fairly large, computer circuits are relatively immune to electrical noise. Each 0 and 1 is called a "bit", and a collection of 8 bits is called a "byte".

The tricky thing now is that representing a number like 16 as a 0 and a 1 takes some imagination. Actually it's not that hard. In decimal we have 0 to 9, and we count to 16 like this:

00
01
02
03
04
05
06
07
08
09

Darn, we're out of symbols. But no fear, we have another column! We make use of that column and carry on, while cycling the first column to back to 0:

10
11
12
13
14
15
16

We carry on since this is so fun:

17
18
19

Now we're out of symbols again in the first column. We increment the second column (gasp! We can do that?) again, while cycling the first column back to 0:

20
21

22
(You get the idea).

In binary, we do the exact same thing, except that we have only two symbols (0 and 1) for each column. So we follow the same "algorithm", cycling the previous column back to 0 while increasing the next column.

00: 00000
01: 00001
02: 00010
03: 00011
04: 00100
05: 00101
06: 00110
07: 00111
08: 01000
09: 01001
10: 01010
11: 01011
12: 01100
13: 01101
14: 01110
15: 01111
16: 10000

Congratulations! You just counted to 16! In binary!

Now let's continue with our Studio:

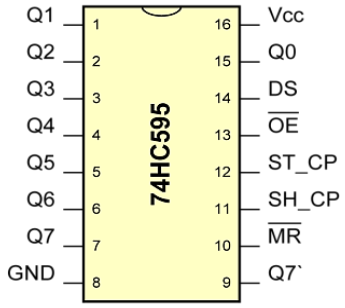
The 74HC595 Serial to Parallel Converter

The 74HC595 is a "serial to parallel converter", meaning that there is one single input for you to feed in 8 bits, and 8 outputs to simultaneously (in parallel) produce the 8 bits you just fed in.

Why would you do that? Put simply, transmitting 8-bits over 8 wires across long distances – like across the bench in your DSA lab – is expensive (no, really) and cumbersome. For that reason computer engineers love converting an 8-bit number into a single stream of bits. On the other side, we would use something like the 74HC595 to convert it back to 8 parallel bits.

Now where would we find such strange and arcane technology? In USB – Universal Serial Bus – for one.

The diagram below shows the pin-out of the 74HC595 chip:



74HC595 8-Bit Shift Register Pinouts

Pin	Symbol	Description
1	Q1	Parallel data output (bit-1)
2	Q2	Parallel data output (bit-2)
3	Q3	Parallel data output (bit-3)
4	Q4	Parallel data output (bit-4)
5	Q5	Parallel data output (bit-5)
6	Q6	Parallel data output (bit-6)
7	Q7	Parallel data output (bit-7)
8	GND	Ground (0 V)
9	Q7`	Serial Data Output
10	$\overline{MR}$	Master Reset (Active Low)
11	SH_CP	Shift Register Clock Input
12	ST_CP	Storage Register Clock Input
13	$\overline{OE}$	Output Enable (Active Low)
14	DS	Serial Data Input
15	Q0	Parallel data output (bit-8)
16	Vcc	Positive Supply Voltage

Pin 15 and pins 1 to 7 form the 8-bit “parallel” output Q0 to Q7, pin 14 is DS, which takes in the serial (aka “one-by-one”) stream of bits to be output onto Q0 to Q7, VCC and GND are your standard power supply connections, MR’ (shown as MR with a bar in the table above) is the Master Reset, letting you reset the chip to take in a new stream of bits. You do this by pulling MR’ to 0 volts (which is what the bar, or the ‘ means). Pin 13 is the OE’ pin switches the chip on if you pull it low to 0 volts (otherwise Q0 to Q7 are placed into a “high impedence” state).

Pins 11 and 12 are the clock (SH_CP) and latch (ST_CP) pins respectively. When SH_CP goes from low to high, the bit at DS is shifted (read) into the 74HC595. You thus place each bit one by one on the DS pin, toggle SH_CP from low to high to copy that bit, and repeat for the remaining 7 bits.

When all 8 bits have been “shifted” in, you toggle ST_CP from low to high to copy the bits to Q0 to Q7.

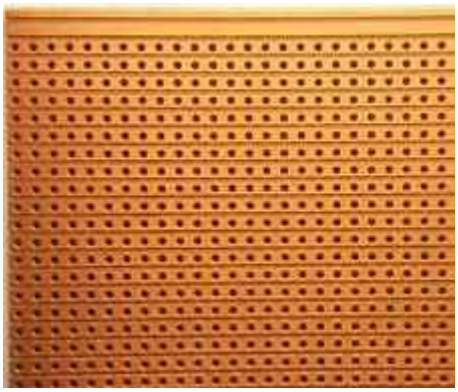
Simple, no?

Let’s start the activity!

Activity 1 Laying Out Your Stripboard

You will now use a stripboard (sometimes called a veroboard) to solder your serial to parallel conversion circuit (given later) together.

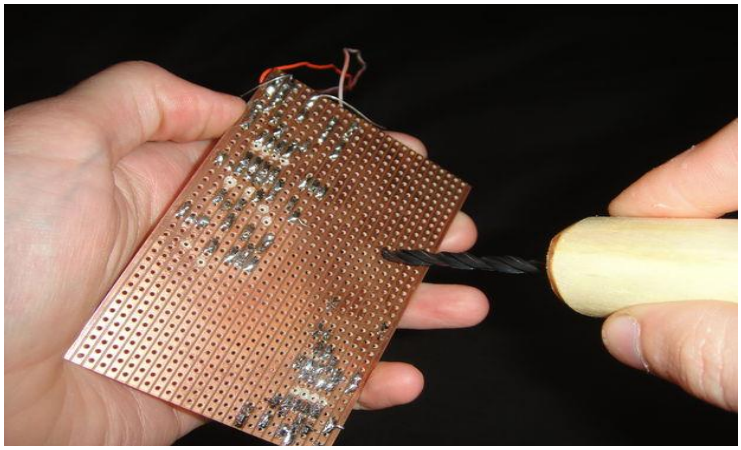
Stripboard is plastic on one side, and has copper tracks on the other side, like so:



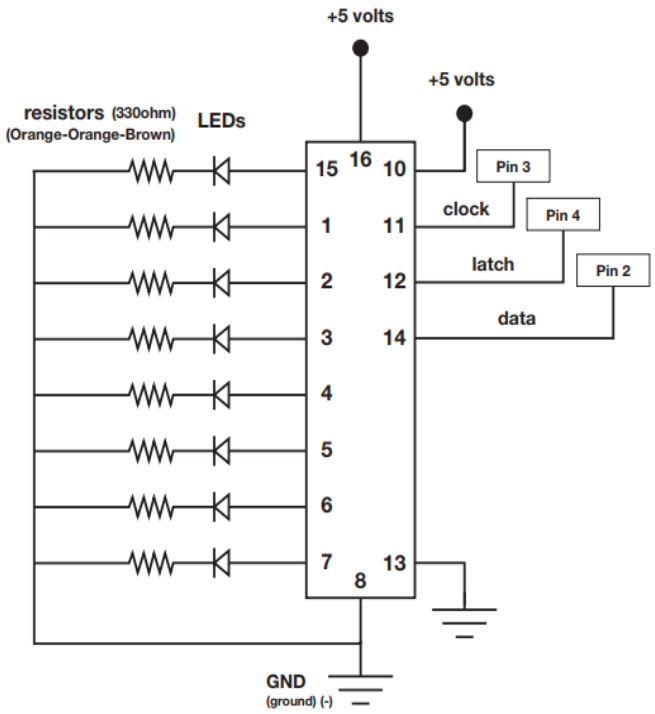
It looks pretty cool, but the catch is that you need to break the tracks at strategic points so that you don't short circuit your chip or the LED or anything else, or so you don't connect up components you didn't intend to connect.

You can break the tracks by using a veroboard cutter tool that looks like this:

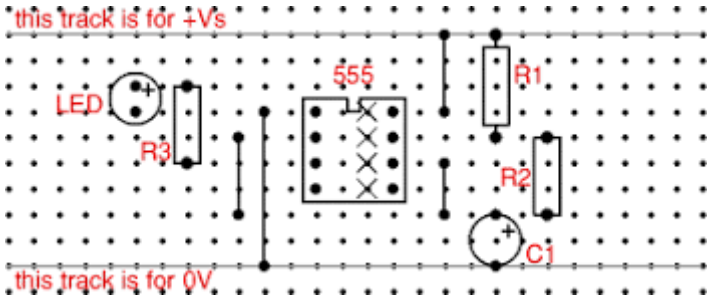
Using it is really simple. Select a hole where you want to break the track, then drill into the hole using this tool until the track is broken:



The circuit you are building is shown below:



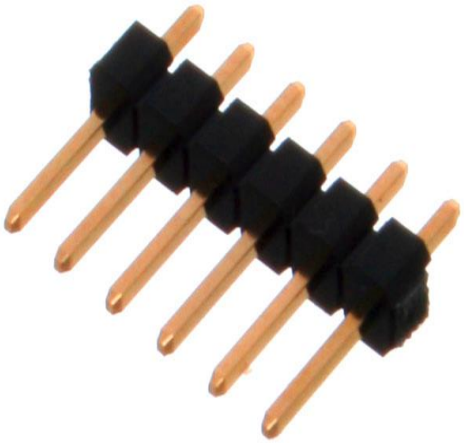
Use the stripboard planning sheet at the end of this studio write-up to plan your circuit first, by drawing your components onto the sheet, and marking with an “X” the spots where you intend to break the tracks. An example planning sheet is shown below:



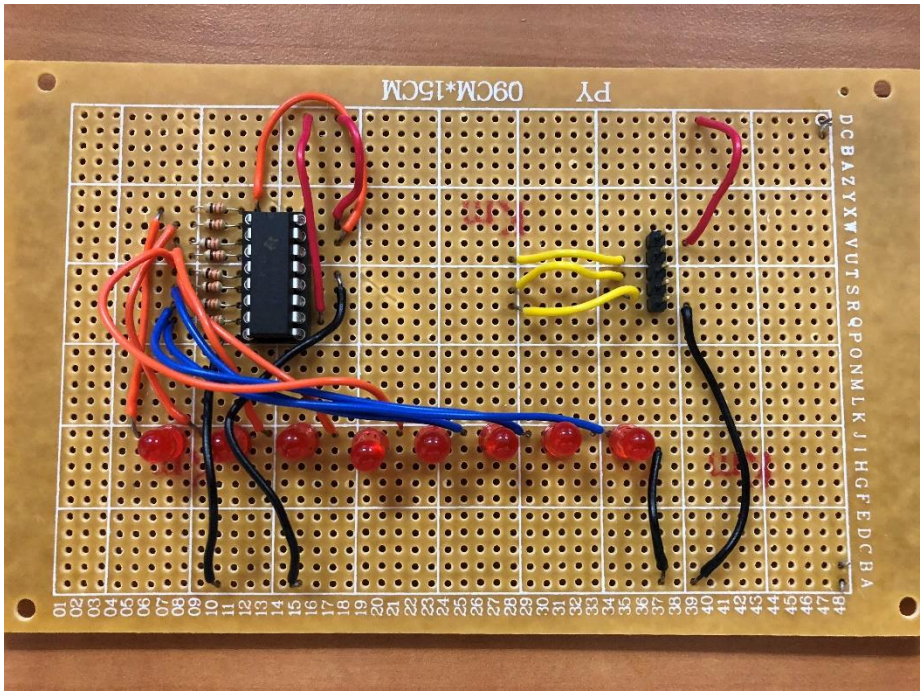
You can use wires to bridge across tracks. You can also use solder to bridge between adjacent tracks.

You don't need to use the entire stripboard (though you can).

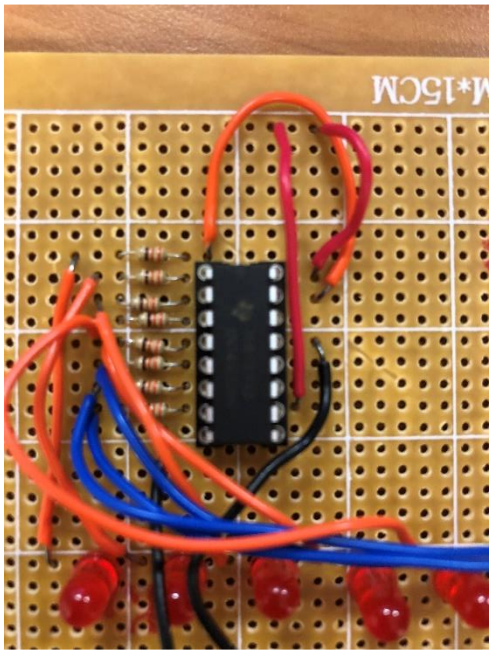
To connect your circuit to your Arduino, we will use a single-row pin header. You will use 5 pins. You can use a cutter to cut out a strip of 5 pins from the single-row header you are given. Keep the rest for your project.



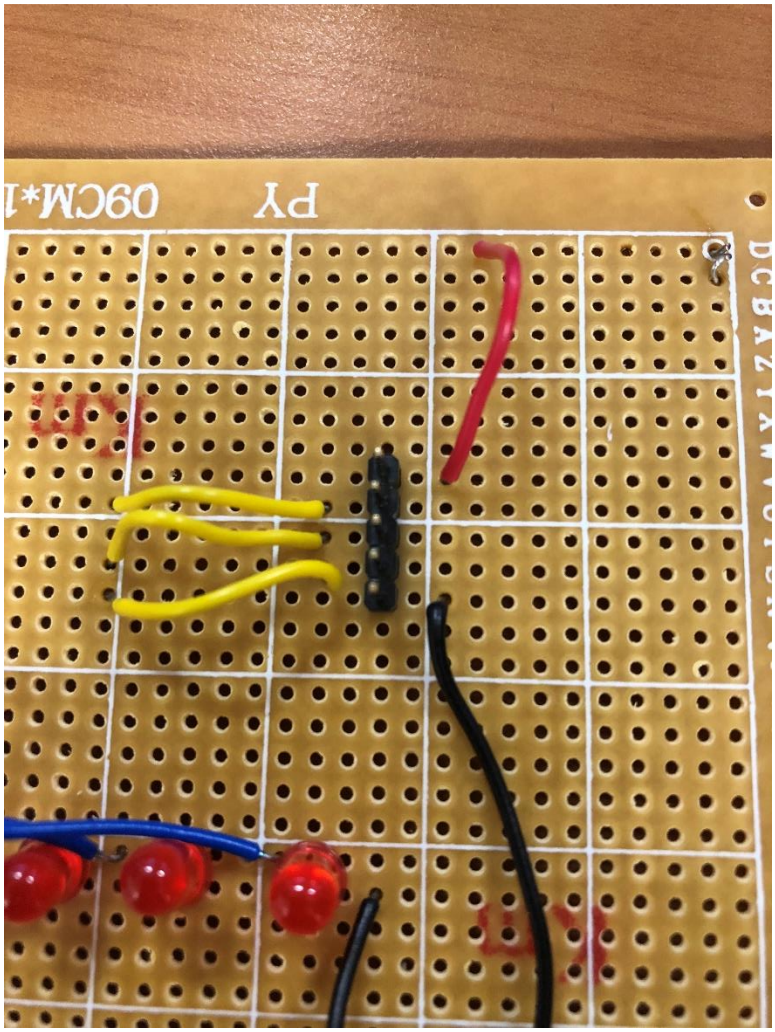
Planning the stripboard is not trivial, so to help you along, here are some pictures from our own reference design.



General view of component layout



Close-up of 74HC595 and 330 ohm 0.25W resistors. Note you should mount the 74HC595 in a 16-pin DIP socket. The notch on the IC is top-most in this picture. The topmost left pin is pin 1.



Close-up of single row pin header. The topmost pin connects VCC on the Arduino to your circuit, the next pin connects pin 2 of the Arduino to pin 14 of the 74HC595, the following pin connects pin 4 of the Arduino to pin 12 of the 74HC595, and the next pin connects pin 3 of the Arduino to pin 11 of the 74HC595. The last pin connects VCC on the Arduino to your circuit.

As with breadboards, you can use dedicate the topmost copper track for VCC and the bottom-most for GND. We have done that in our own design.

NOTE: These pictures are only suggestions on how to lay out your components. You must break the copper tracks on the other side appropriately to prevent short circuits (e.g. between the pins on the two sides of the IC) and to prevent unwanted connections between components (e.g. between LEDs).

You will also need to plan, using the circuit diagram on Page 6, how to wire your circuit with the wires given. **You might also find it easier to plug in the components on the actual stripboard to see space constraints and help in your planning, and using a whiteboard marker to mark on the copper where you intend to break it.**

PLAN YOUR DESIGN ON THE SHEET AND SHOW IT TO YOUR TA BEFORE PROCEEDING. You probably ought to use a pencil.

Activity 2 – Soldering Your Circuit

We will begin soldering the circuit. There will be a total of (8 x 2 + 8 x 2 + 16 x 1 + 3 x 1 + 2 x N) = 51 + 2N junctions to solder, where N is the number of wires you are using in your design.

Take turns within the team to solder so all of you get some experience. You will on average solder around 10-15 joints each.

Step 0

As one team mate solders, another can start the Arduino IDE and type in the following program.

```
//Pin connected to ST_CP of 74HC595
int latchPin = 4;
//Pin connected to SH_CP of 74HC595
int clockPin = 3;
////Pin connected to DS of 74HC595
int dataPin = 2;

void setup()
{
  //set pins to output so you can control the shift register
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop()
{
  // count from 0 to 255 and display the number
  // on the LEDs
  for (int numberToDisplay = 0; numberToDisplay < 256;
          numberToDisplay++) {
    // take the latchPin low so
    // the LEDs don't change while you're sending in bits:
    digitalWrite(latchPin, LOW);
    // shift out the bits:
    shiftOut(dataPin, clockPin, MSBFIRST, numberToDisplay);

    //take the latch pin high so the LEDs will light up:
    digitalWrite(latchPin, HIGH);
    // pause before next value:
    delay(500);
  }
}
```

Step 1

Using your plan from Activity 1, break the tracks at the spots which you marked with an "X" in your planning sheet (or marked with whiteboard marker on the copper track).

PUT UP YOUR HAND AND SHOW THE BOARD WITH ALL BREAKS FULLY MADE.

Step 2

Now you will begin soldering your components. Start with the shortest components first, which will probably be the resistors. Insert the resistors, and solder them in. You will find it easier if you used a small piece of sticky tape to hold the resistor while soldering, to stop them from falling out.

After soldering a resistor, snip off the legs as close as possible to the board before soldering the next resistor.

Step 3

Next solder in the IC socket. Again you will find it easier if you taped down the socket before soldering. Re-use your tape each time.

Step 4

Insert the LEDs and solder them. NOTE THE POLARITY OF YOUR LEDs! The negative (shorter leg) side should connect to ground, while the positive (longer leg) side should connect to the outputs of the 74HC595 via the 330 ohm resistors.

After each LED is soldered, snip off the legs before moving on to the next one.

Step 5

Cut off 5 pins from the strip of pin headers, insert them into the right place on the board according to your planning sheet. Solder them in.

(Note: Stopping them from falling out is tricky. The trick here is to press on ONE pin of the header against the board to hold it in place, then solder another pin. DO NOT SOLDER THE PIN YOU ARE PRESSING ON OR YOU WILL BURN YOUR FINGERS!)

Step 6

Solder in all your wiring.

Step 7

Check all your soldering and wiring for correctness, then plug the 74HC595 into the socket, MAKING SURE THAT THE NOTCH ON THE IC ALIGNS WITH THE NOTCH ON THE SOCKET.

Activity 3

Step 1

Take the 40-way M/F ribbon cable given to you (the colourful one) and tear off a strip of 5 wires. Use these to connect your board to the Arduino. Once again if you follow the reference design, it should be connected as follows, taking the topmost pin down to the bottom-most pin.

Top-most pin (VCC for circuit)	VCC on Arduino
Second pin (DS on 74HC595)	Pin 2 on Arduino
Third pin (ST_CP on 74HC595)	Pin 4 on Arduino
Fourth pin (SH_CP on 74HC595)	Pin 3 on Arduino
Bottom-most pin (GND for circuit)	GND on Arduino

Step 2

Compile and upload the program to the Arduino. **Put up your hand and show what you've done to your GA.**