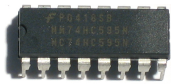# Project 17

LED Dot Matrix -

Basic Animation

Project 17 – LED Dot Matrix – Basic Animation

In this project we are going to use the 74HC595 chips to control a Dot Matrix array of 64 LED's (8x8) and produce some basic animations.

This Project requires the Mini Dot Matrix Display.

What you will need

|	
2 x 74HC595 Shift Registers	
8 x 240Ω Resistor	
Mini Dot Matrix	

Connect it up

The two 595 chips are left the same as in Project 16. Leave the first 8 resistors and remove the 2nd 8. Put your Dot Matrix unit at the end of the breadboard. Make sure it is the right way around. To do this turn it upside down and make sure the words are the right way up then flip it over (from right to left) in your hand. Push it in carefully.

If you take it that bottom left pin of the matrix is Pin 1, bottom right is Pin 8, top right is Pin 9 and top left is Pin 16 then connect jumper wires between the 16 outputs as follows :-

Output	Pin
1	9
2	14
3	8
4	12
5	1
6	7
7	2
8	5
9	13
10	3
11	4
12	10
13	6
14	11
15	15
16	16

Examine the diagrams carefully.
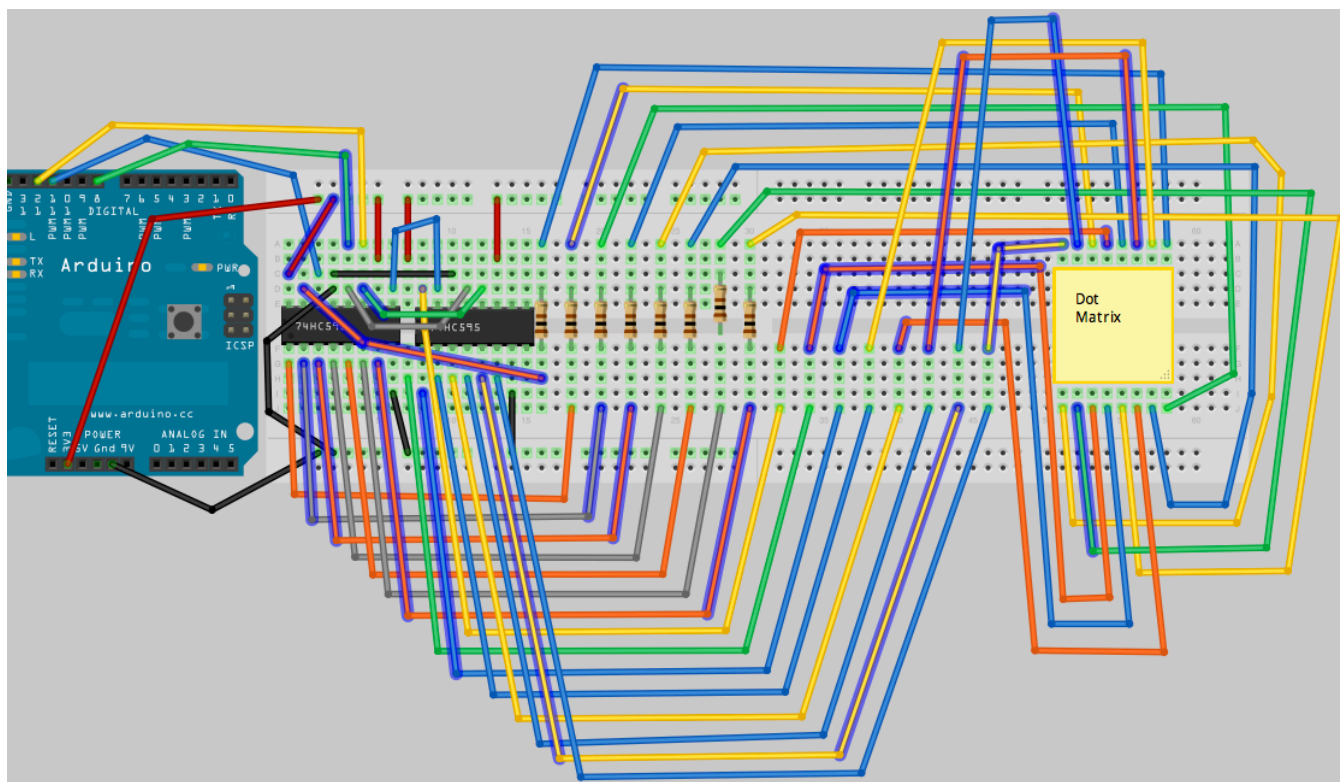
Enter the Code

```
// Project 17

#include <TimerOne.h>

//Pin connected to Pin 12 of 74HC595 (Latch)
int latchPin = 8;
//Pin connected to Pin 11 of 74HC595 (Clock)
int clockPin = 12;
//Pin connected to Pin 14 of 74HC595 (Data)
int dataPin = 11;

uint8_t led[8];
long counter1 = 0;
long counter2 = 0;

void setup() {
  //set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  led[0] = B11111111;
  led[1] = B10000001;
  led[2] = B10111101;
  led[3] = B10100101;
  led[4] = B10100101;
  led[5] = B10111101;
  led[6] = B10000001;
  led[7] = B11111111;
  Timer1.initialize(10000);
 Timer1.attachInterrupt(screenUpdate);
}

void loop() {
  counter1++;
    if (counter1 >=100000) {counter2++;}
if (counter2 >= 10000) {
  counter1 = 0;
  counter2 = 0;
  for (int i=0; i<8; i++) {
    led[i]= ~led[i];
  }
  }
}

void screenUpdate() {
uint8_t row = B00000001;
 for (byte k = 0; k < 9; k++) {
      // Open up the latch ready to receive data
      digitalWrite(latchPin, LOW);
      shiftIt(~row );
      shiftIt(led[k] ); // LED array

   // Close the latch, sending the data in the registers out to the
matrix
digitalWrite(latchPin, HIGH);     row = row << 1;
 }
}
```

```
void shiftIt(byte dataOut) {
  // Shift out 8 bits LSB first,
  // on rising edge of clock

  boolean pinState;

  //clear shift register read for sending data
  digitalWrite(dataPin, LOW);

  // for each bit in dataOut send out a bit
  for (int i=0; i<8; i++)  {
      //set clockPin to LOW prior to sending bit
    digitalWrite(clockPin, LOW);

    // if the value of DataOut and (logical AND) a bitmask
    // are true, set pinState to 1 (HIGH)
    if ( dataOut & (1<<i) ) {
      pinState = HIGH;
    }
    else {
      pinState = LOW;
    }

    //sets dataPin to HIGH or LOW depending on pinState
    digitalWrite(dataPin, pinState);
    //send bit out on rising edge of clock
    digitalWrite(clockPin, HIGH);
    digitalWrite(dataPin, LOW);
  }

  //stop shifting
  digitalWrite(clockPin, LOW);
}
```

When this code is run, you will see a very basic animation of a heart that flicks back and forth between a positive and a negative image.

Before this code will work you will need to download the TimerOne library from the Arduino website. It can be downloaded from http://www.arduino.cc/playground/uploads/Code/TimerOne.zip

Once downloaded, unzip the package and place the folder, called TimerOne, into the hardware/libraries directory.

Project 17 - Hardware Overview

For this Project we will take a look at the Hardware before we look at how the code works.

The 74HC595 and how to use them has been explained in the previous projects. The only addition to the circuit this time was an 8x LED Dot Matrix unit.

Dot Matrix units typically come in either an 5x7 or 8x8 matrix of LED's. The LED's are wired in the matrix such that either the anode or cathode of each LED is common in each row. E.g. In a Common Anode LED DOt Matrix unit, each row of LED's would have all of their anodes in that row wired together. The Cathodes of the LED's would all be wired together in each column.   The reason for this will become apparent soon.

A typical single colour 8x8 Dot Matrix unit will have 16 pins, 8 for each row and 8 for each column. You can also obtain bi-colour units (e.g. Red and Green) as well as Full Colour RGB (Red, Green and Blue) Units, such as is used in large video walls. Bi or Tri (RGB) colour units have 2 or 3 LED's in each pixel of the array. These are very small and next to each other.

By turning on different combinations of Red, Green or Blue in each pixel and by varying their brightnesses, any colour can be obtained.

The reason the rows and columns are all wired together is to minimise the number of pins required. If this was not the case, a single colour 8x8 Dot Matrix unit would have to have 65 pins. One for each LED and a common Anode or Cathode connector. By wiring the rows and columns together only 16 pin outs are required.

However, this now poses a problem. If we want a particular LED to light in a certain position. If for example we had a Common Anode unit and wanted to light the LED at X, Y position 5, 3 (5th column, 3rd row), then we would apply a current to the 3rd Row and Ground the 5th column pin. The LED in the 5th column and 3rd row would now light.

Now let us imagine that we want to also light the LED at column 3, row 6. So we apply a current to the 6th row and ground the 3rd column pin. The LED at column 3, row 6 now illuminates. But wait… The LED's at column 3, row 6 and column 5, row 6 have also lit up.

This is because we are applying power to row 3 and 6 and grounding columns 3 and 5. We can't turn off the unwanted LED's without turning off the ones we want on also. It would appear that there is no way we can light just the two required LED's with the rows and columns wired together as they are. The only way this would work would be to have a separate pinout for each LED meaning the number of pins would jump from 16 to 65. A 65 pin Dot Matrix unit would be very hard to wire up and also to control as you'd need a microcontroller with at least 64 digital outputs.

Is there a way to get around this problem? Yes there is, and it is called 'multiplexing' (or muxing).

Multiplexing

Multiplexing is the technique of switching one row of the display on at a time. By selecting the appropriate columns that we want an LED to be lit in that row and then turning the power to that row (or the other way round for common cathode displays) on, the chosen LED's in that row will illuminate. That row is then turned off and the next row is turned on, again with the appropriate columns chosen and the LED's in the 2nd row will now illuminate. Repeat with each row till we get to the bottom and then start again at the top.

If this is done fast enough (more than 100Hz or 100 times per second) then the phenomenon of 'persistance of vision' (where an afterimage remains on the retina for approx 1/25th of a second) will mean that the display will appear to be steady, even though each row is turned on and off in sequence.

By using this technique we get around the problem of being able to display individual LED's without other LED's in the same column or row also being lit.

For example, we want to display the following image on our display:-
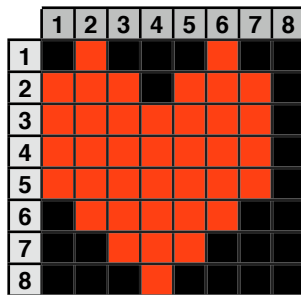
Then each row would be lit in turn like so…
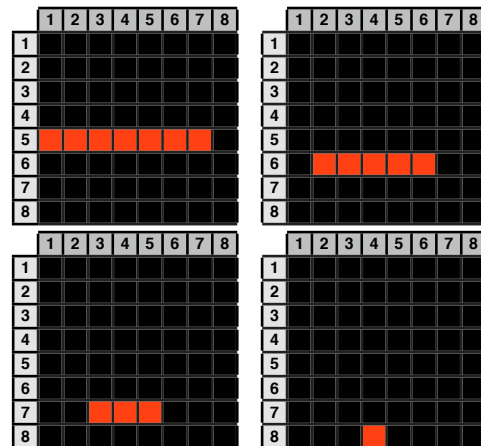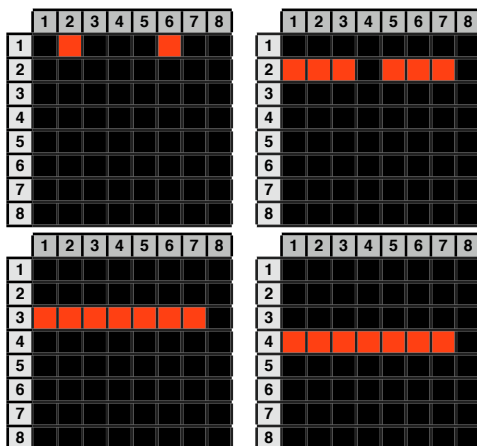
By scanning down the rows and illuminating the respective LED's in each column of that row and doing this very fast (more than 100Hz) the human eye will perceive the image as steady and the image of the heart will be recognisable in the LED pattern.

We have used this multiplexing technique in our Project's code and that is how we are able to display the heart animation without also displaying extraneous LED's.

Project 17 - Code Overview

The code for this project uses a feature of the Atmega chip called a Hardware Timer. This is essentially a timer on the chip that can be used to trigger an event. In our case we are setting our ISR (Interrupt Service Routine) to fire every 10000 microseconds, which is every 100th of a second.

We make use of a library that has already been written for us to enable easy use of interrupts and this is the TimerOne library. TimerOne makes creating an ISR very easy. We simply tell the function what the interval is, in this case 10000 microseconds, and the name of the function we wish to activate every time the interrupt is fired, in our case this is the 'screenUpdate' function.

TimerOne is an external library and we therefore need to include it in our code. This is easily done using the include command.

```
#include <TimerOne.h>
```

After this the pins used to interface with the shift registers are declared.

```
//Pin connected to Pin 12 of 74HC595 (Latch)
int latchPin = 8;
//Pin connected to Pin 11 of 74HC595 (Clock)
int clockPin = 12;
//Pin connected to Pin 14 of 74HC595 (Data)
int dataPin = 11;
```

We now create an array of type uint8_t that has 8 elements and two counters of type long. An uint8_t is simply the same as a byte. It is an unsigned integer of 8 bits.

```
uint8_t led[8];
long counter1 = 0;
long counter2 = 0;
```

The led[8] array will be used to store the image we are going to display on the Dot Matrix display. The counters will be used to create a delay.

In the setup routine we set the latch, clock and data pins as outputs.

```
void setup() {
  //set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
```

Once the pins have been set to outputs, the led array is loaded with the 8-bit binary images that will be displayed in each row of the 8x8 Dot Matrix Display.

```
led[0] = B11111111;
led[1] = B10000001;
led[2] = B10111101;
led[3] = B10100101;
led[4] = B10100101;
led[5] = B10111101;
led[6] = B10000001;
led[7] = B11111111;
```

By looking at the array above you can make out the image that will be displayed, which is a box within a box. You can, of course, adjust the 1's and 0's yourself to make any 8x8 sprite you wish.

After this the Timer1 function is used. First, the function needs to be initialised with the frequency it will be activated at. In this case we set its period to 10000 microseconds, or 1/100th of a second. Once the interrupt has been initialised we need to attach to the interrupt a function that will be executed every time the time period is reached. This is the 'screenUpdate' function which will fire every 1/100th of a second.

```
Timer1.initialize(10000);
Timer1.attachInterrupt(screenUpdate);
```

In the main loop we set a counter that counts to 1,000,000,000 to create a delay. This is done by two loops, one that loops 100,000 times and the other one that loops 10,000 times. I have done it this way, instead of using delay() as the current version of the Timer1 library seems to interfere with the delay function. You may try delay() instead and see if it works (updates to the libraries are occurring from time to time). counter1 and counter2 are reset to zero once they reach their targets.

```
void loop() {
  counter1++;
    if (counter1 >=100000) {counter2++;}
if (counter2 >= 10000) {
  counter1 = 0;
  counter2 = 0;
```

Once the end of the delay is reached, a for loop cycles through each of the 8 elements of the led array and inverts the contents using the ~ or NOT bitwise operator.

```
  for (int i=0; i<8; i++) {
    led[i]= ~led[i];
  }
  }
}
```

This simply turns the binary image into a negative of itself by turning all 1's to 0's and all 0's to 1's.

We now have the screenUpdate function. This is the function that the interrupt is activating every 100th of a second. This whole routine is very important as it is responsible for ensuring our LED's in the DOt Matrix array are lit correctly and displays the image we wish to convey. It is a very simple but very effective function.

```
void screenUpdate() {
uint8_t row = B00000001;
 for (byte k = 0; k < 9; k++) {
      // Open up the latch ready to receive
data
       digitalWrite(latchPin, LOW);
      shiftIt(~row );
      shiftIt(led[k] ); // LED array

   // Close the latch, sending the data in
the registers out to the matrix
digitalWrite(latchPin, HIGH);     row = row
<< 1;
 }
}
```

An 8 bit integer called 'row' is declared and initialised with the value B00000001.

```
uint8_t row = B00000001;
```

We now simply cycle through the led array and send that data out to the Shift Registers preceded by the row (which is processed with the bitwise NOT ~ to make sure the row we want to display is turned off, or grounded).

```
 for (byte k = 0; k < 9; k++) {
      // Open up the latch ready to receive
data
       digitalWrite(latchPin, LOW);
      shiftIt(~row );
      shiftIt(led[k] ); // LED array
```

Once we have shifted out that current rows 8 bits the value in row is bitshifted left 1 place so that the next row is displayed.

```
row = row << 1;
```

Remember from the hardware overview that the multiplexing routine is only displaying one row at a time, turning it off and then displaying the next row. This is done at 100Hz which is too fast for the human eye to see the flicker.

Finally, we have a ShiftOut function, the same as in the previous Shift Register based projects, that sends the data out to the 74HC595 chips.

```
void shiftIt(byte dataOut)
```

So, the basic concept here is that we have an interruot routine that executes every 100th of a second. In that routine we simply take a look at the contents of a screen buffer array (in this case led[]) and display it on the dot matrix unit one row at a time, but do this so fast that to the human eye it all seems to be lit at once.

The main loop of the program is simply changing the contents of the screen buffer array and letting the ISR do the rest.

The animation in this project is very simple, but by manipulating the 1's and 0's in the buffer we can make anything we like appear on the Dot Matrix unit from shapes to scrolling text.