The 74HC595 shift register

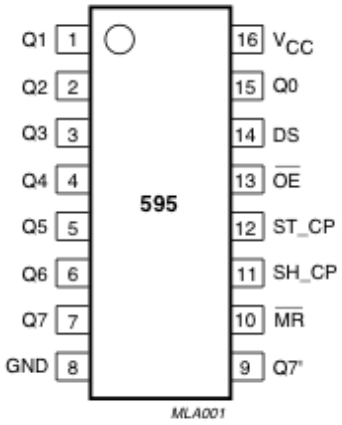
The 74HC595 shift register is an "8-bit serial in, serial/parallel out" device. If used as a serial-in parallel-out, then it requires only 3 pins on the microprocessor to control its 8 output pins, thus saving microprocessor pins.

The *serial-in* operation works through the so called "synchronous serial communication" which means that data is sent to the chip in series, one bit at a time, synchronized by a clock pin.

The *parallel-out* operation works by enabling of a latch pin: Once a byte is completely shifted out form the controller to the chip, a latch pin is enabled in order to transfer the whole byte at once to the register output pins.

The *serial-out* operation works through an extra pin provided on the register which can pass the serial data received from the controller out again as is. We will use this feature to transmit 2 bytes (16 bits) from the controller. The first 8 bits will flow through the first register to the second where it will appear on the latter output pins. For more on this, the reader may consult https://www.arduino.cc/en/Tutorial/ShiftOut.

A schematic of the shift register is shown below.

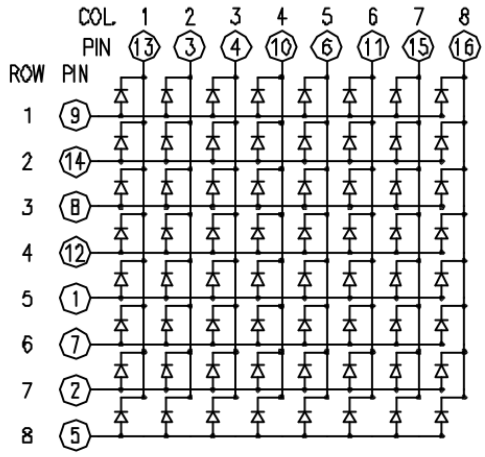


The pinouts are shown in the following table; see also Phillip's datasheet.

PINS 1-7, 15	Q0 : Q7	Output Pins
PIN 8	GND	Ground, Vss
PIN 9	Q7’	Serial Out
PIN 10	$\overline{MR}$	Master Reclear, active low
PIN 11	SH_CP	Shift register clock pin
PIN 12	ST_CP	Storage register clock pin (latch pin)
PIN 13	$\overline{OE}$	Output enable, active low
PIN 14	DS	Serial data input
PIN 16	Vcc	Positive supply voltage

How 74H595 shift registers may drive an 8X8 dot matrix

The pinout of an 8X8 dot matrix display is shown below.



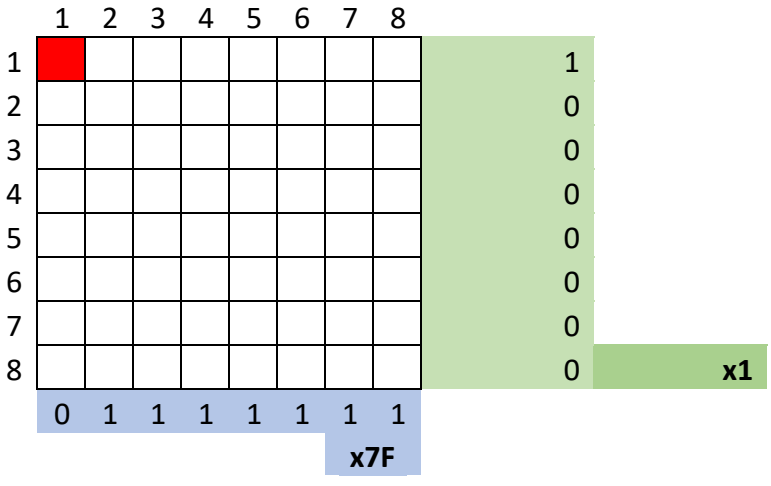
The matrix can be driven by two 595 registers. One register, to be called the **R register**, is directly connected to the Arduino and the other, to be called the **C register**, is cascaded to the R register using the serial-out operation. We choose to configure the R an C registers to drive the rows and columns of the dot matrix, respectively. Accordingly, the following connections are made.

R register		
Row #	Matrix Pin	595 Pins
1	9	7 (Q0)
2	14	6 (Q1)
3	8	5 (Q2)
4	12	4 (Q3)
5	1	3 (Q4)
6	7	2 (Q5)
7	2	1 (Q6)
8	5	15 (Q7)

C register		
Column #	Matrix Pin	595 Pins
1	13	7 (Q0)
2	3	6 (Q1)
3	4	5 (Q2)
4	10	4 (Q3)
5	6	3 (Q4)
6	11	2 (Q5)
7	15	1 (Q6)
8	16	15 (Q7)

So, for example, all LED's in row 1 of the dot matrix are controlled by pin 9 on the physical matrix. So we connect pin 9 on the matrix to output pin 7 on the R register and label it "Q0".
Similarly, the LED's in row 4 of the dot matrix are controlled by pin 12 on the physical matrix. So we connect pin 12 on the matrix to output pin 3 on the R register and label it "Q3", …etc.
Column-wise, the leads in column 4 of the dot matrix are set by pin 10 on the physical matrix. So we connect pin 10 on the matrix to output pin 4 on the C register and label it "Q3", …etc.

This configuration makes it easy to determine how to lit up any pixel in the dot matrix. In general, a specific LED is lit up by setting its corresponding row and column to 1 (HIGH) and 0 (LOW), respectively. For example, to turn on the pixel in row 1 and column 1 (R1C1), we use the settings shown below

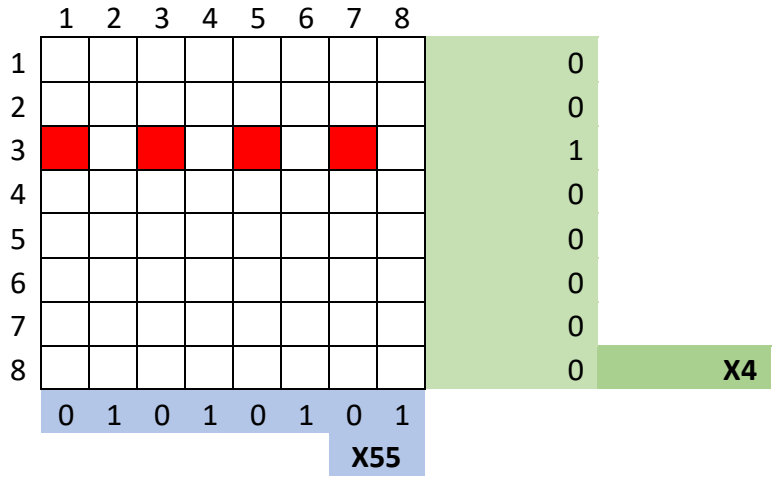


This means that the row pattern 0b00000001(=0X1) needs to be shifted out to the R register and the column pattern 0b11111110(=0X7F) needs to be shifted out to the C register. Arduino `shiftOut()` function is used to accomplish this. We will discuss this function in more detail later.

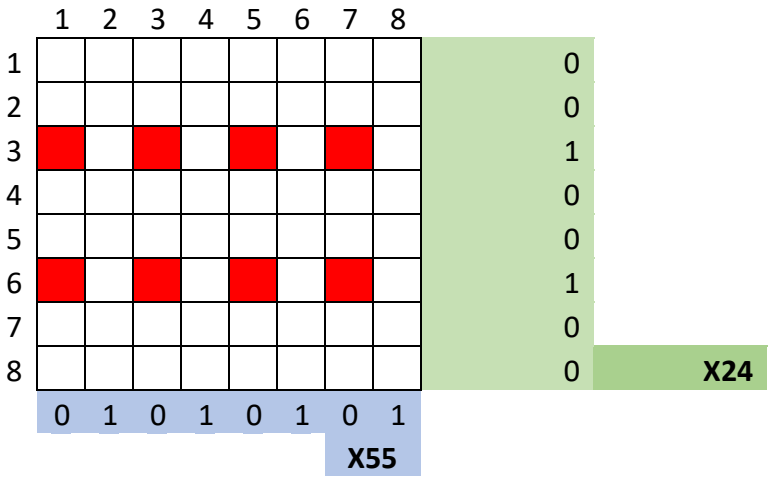
```
shiftOut(dataPin, clockPin, MSBFIRST, 0x7F);
shiftOut(dataPin, clockPin, LSBFIRST, 0x01);
digitalWrite(latchPin, HIGH);
digitalWrite(latchPin, LOW);
```

Observe that the column values (here 0X7F) are shifted out first then the row values (here 0X1). As explained in the previous section, the column values are first shifted out to the chip directly connected to the Arduino (the R register) and then cascaded to the other chip (The C register) as the row byte is shifted out to the R register.

The following figure gives an example of controlling several columns in the same row.

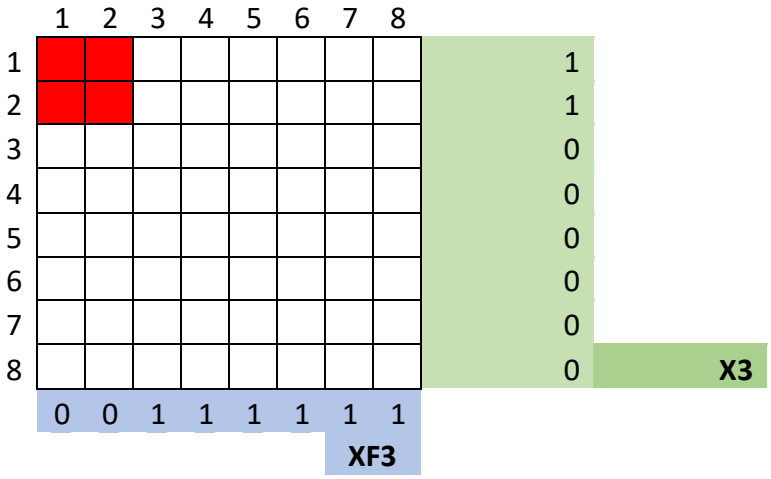


It is also possible to control pixels in more than one row simultaneously as the following figure shows



However, some unintended effects can result because of enabling more than one row at a time. For example suppose we want to fire up pixels R1C1 and R2C2 simultaneously. One would be tempted to enable rows 1 and 2 together with columns 1 and 2. The result is shown in the following figure. The unintended pixels R2C2 and R2C1 get lit up in the process.

To safeguard against such unintended effects we recommend enabling *only one* row at a time.



Before discussing this, let's first observe that, due to the mismatch between column numbering and bitwise values of columns, the latter is shifted out most significant bit first (`MSBFIRST`).

The unintended effects in the last example can be handled by the following piece of code.

```
byte Cols[2]={0X01,0X02};
void loop()
{
    for (i=0;i<=1; i++){
      shiftOut(dataPin, clockPin, MSBFIRST, Cols[i]);
      shiftOut(dataPin, clockPin, LSBFIRST, 1<<i);
      digitalWrite(latchPin, HIGH);
      digitalWrite(latchPin, LOW);
    }
}
```

How Arduino controls the 74HC595 shift register: Basic mechanism

In this section we are going to discuss how the microprocessor controls the 595 chip and pay a closer look at the inner working of the `shiftOut()` function. We will actually mimic what this function does.

Arduino Uno (Mega) controls the 74HC595 using only 3 pins. All programs in this presentation use the following configuration:

74HC595 Pin	Connected to Arduino	Program variable name
Data pin (DS)	Digital pin 11	dataPin
Shift clock pin (SH_CP)	Digital pin 12	clockPin
Storage clock pin (ST_CP)	Digital pin 8	latchPin

Therefore, the initial setup is as follows.

```
int latchPin = 8;
int clockPin = 12;
int dataPin = 11;

void setup() {
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}
```

A byte of data is shifted out and stored on the 595 register one bit at a time. The following steps are used to accomplish this.

1. latchPin is set to LOW while shifting out data

2. A bit is sent to dataPin
3. clockPin is set to HIGH to start shifting out the value of dataPin to the 595 DS pin.
4. clockPin is set to LOW to end shifting out.
5. Steps 2:4 are repeated for all bits in the byte.
6. latchPin is set to HIGH to enable transferring the byte to the 595 output pins Q0:Q7.

The following code illustrates these steps by sending byte 0b0001111 to the 595 output pins.

```
void loop() {
  // sending byte 0b0001111 to the 595 output pins
    digitalWrite(latchPin, 0);          // Step 1.

    for (int i=0;i<=7;i++){             //loop to implement Steps 2:5
         if (i<=3){
           digitalWrite(dataPin,1);     // Step 2.
           digitalWrite(clockPin,1);    // Step 3.
           digitalWrite(clockPin,0);    // Step 4.
      }
         else{
           digitalWrite(dataPin,0);     // Step 2.
           digitalWrite(clockPin,1);    // Step 3.
           digitalWrite(clockPin,0);    // Step 4.
      }
    }
    digitalWrite(latchPin,1);           // Step 6.
  }
```

If you attach LED's between pins Q0:Q7 and the ground (through 330 Ω resistors) you will see LED's Q0:Q3 lit up.

The code can be simplified (and generalized) using an array.

```
void loop() {
    byte  data[]=0b0001111;              // byte to be shifted out.
    digitalWrite(latchPin, 0);           // Step 1.

    for (int i=0;i<=7;i++){             //loop to implement Steps 2:5
       digitalWrite(dataPin,data[i]);   // Step 2.

           digitalWrite(clockPin,1);    // Step 3.
           digitalWrite(clockPin,0);    // Step 4.
      }
    digitalWrite(latchPin,1);           // Step 6.
  }
```

The above code, although perfectly functional, is greatly simplified by using the Arduino `shiftOut()` function. Its syntax is as follows.

```
shiftOut(dataPin, clockPin, bitOrder, value)
```

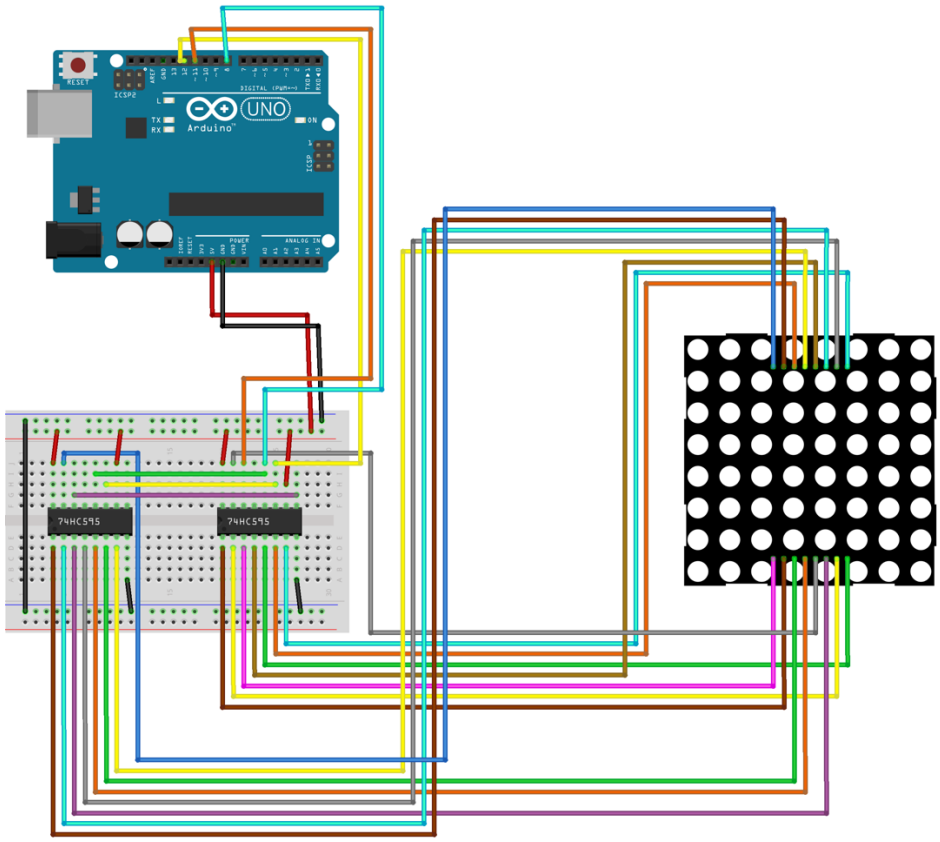
where:
`dataPin`: the pin on which to output each bit. Allowed data types: `int`.
`clockPin`: the pin to toggle once the dataPin has been set to the correct value. Allowed data types: `int`.
`bitOrder`: which order to shift out the bits; either MSBFIRST or LSBFIRST. (Most Significant Bit First, or, Least Significant Bit First).
`value`: the data to shift out. Allowed data types: `byte`.

Then the previous code simplifies to

```
void loop() {
    byte  data=0b0001111;              // byte to be shifted out.
    digitalWrite(latchPin, 0);         // disable output while shifting
    shiftout(dataPin,clockPin, LSFIRST, data);
    digitalWrite(latchPin,1);           // latch byte to 595 output pins
  }
```

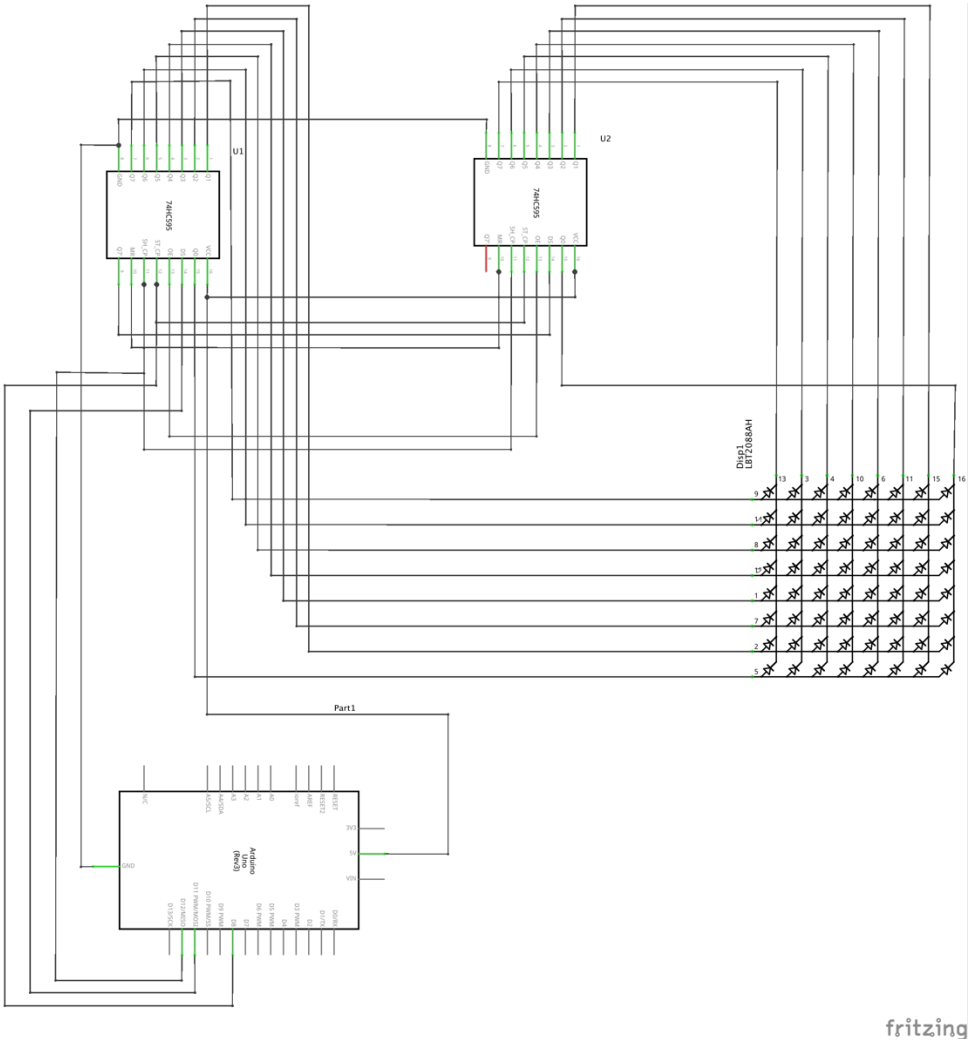
The full picture

It is time now to explain the whole setup. The dot matrix is now driven by two 74HC595 registers as explained in the previous section and the system is controlled by Arduino. I used Arduino MEGA, but UNO will work just as well. In fact, the schematic and circuit diagram depict Arduino UNO as shown below.



Led Matrix control with Arduino Uno (Mega) and two 595 chips: Left chip controls columns and right chip controls rows

fritzing



For illustration, we show how to display a letter “C” on the dot matrix. Any other 8X8 pattern can be displayed in the same manner. All we need to do is create the column pattern for the desired design.

First we need a design for the letter “C”. Excel was used for this purpose. The ultimate design is shown below.

1	1	1	0	0	1	1	1
1	1	0	1	1	0	1	1
1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	1	1	0	1	1
1	1		0	0	1	1	1

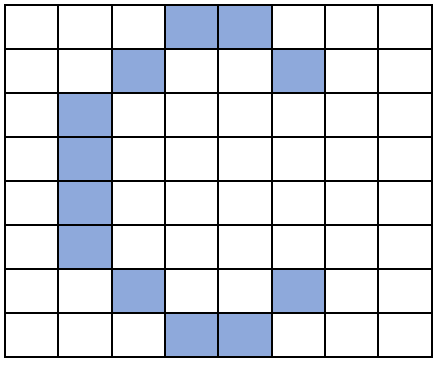
11100111	E7
11011011	DB
10111111	BF
10111111	BF
10111111	BF
10111111	BF
11011011	DB
11100111	E7

The following steps are taken.

1. Draw an 8X8 table

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

2. Arrive at a convenient design for the letter “C” by changing the relevant cells background color.



3. Assign the value 0 to the colored cells and 1 to all other cells.

1	1	1	0	0	1	1	1
1	1	0	1	1	0	1	1
1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	1	1	0	1	1
1	1	1	0	0	1	1	1

4. Copy the patterns of 1's and 0's to a new column

1	1	1	0	0	1	1	1	11100111
1	1	0	1	1	0	1	1	11011011
1	0	1	1	1	1	1	1	10111111
1	0	1	1	1	1	1	1	10111111
1	0	1	1	1	1	1	1	10111111
1	0	1	1	1	1	1	1	10111111
1	1	0	1	1	0	1	1	11011011
1	1	1	0	0	1	1	1	11100111

5. Change the binary values to hexadecimals (the Excel function "=BIN2HEX(cell_value)" may be used here).

11100111	E7
11011011	DB
10111111	BF
10111111	BF
10111111	BF
10111111	BF
11011011	DB
11100111	E7

6. The resulting hex values represent the column patterns that need to be shifted out to the C register.

Now compile and run the following code to display the letter "C".

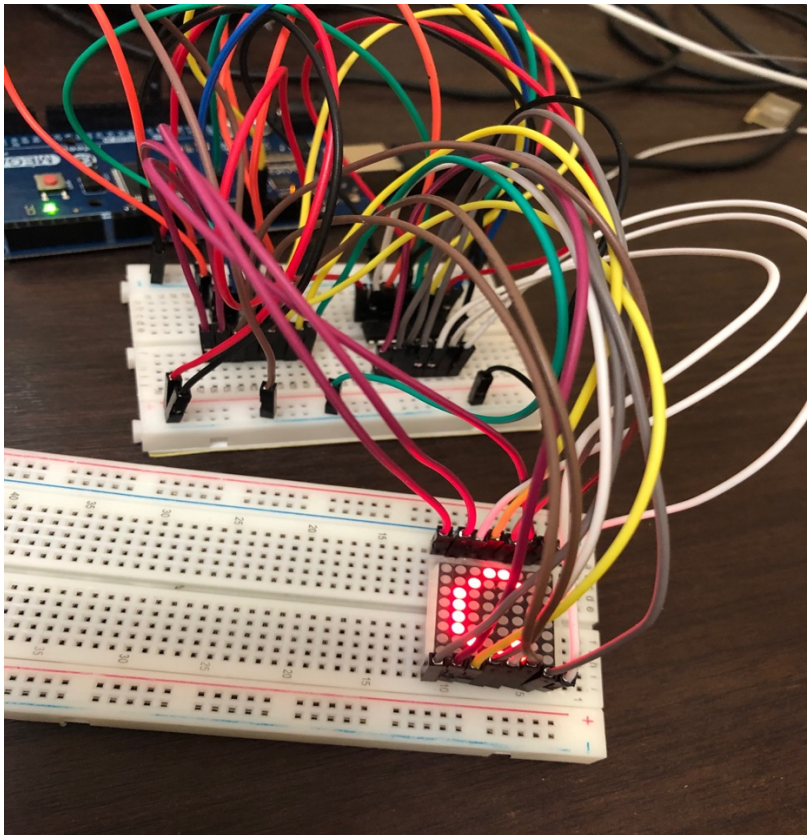
```
/*
  Controling Led matrix with two 74HC595 Shift Registers
  This code displays a static letter "C"
 */
//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
//Pin connected to DS of 74HC595
int dataPin = 11;

// C register; cascaded; column patterns for the letter C
byte Cols[]={0xE7, 0xDB,0xBF,0xBF,0xBF,0xBF,0xDB,0xE7};

void setup() {
  //define pins as output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop()
  {
        for(int i=0;i<=7;i++){
          shiftOut(dataPin,clockPin,MSBFIRST,Cols[i]); // pattern for row i
          shiftOut(dataPin,clockPin,LSBFIRST, 1<<i);   //row i
          digitalWrite(latchPin,HIGH);
          digitalWrite(latchPin, LOW);
        }
  }
```

The main action in this program is the loop

```
for(int i=0;i<=7;i++){
          shiftOut(dataPin,clockPin,MSBFIRST,Cols[i]); // pattern for row i
          shiftOut(dataPin,clockPin,LSBFIRST, 1<<i);   //row i
          digitalWrite(latchPin,HIGH);
          digitalWrite(latchPin, LOW);
}
```

It basically controls, for each row, the columns to be enabled `(Cols[i])`. Notice that row number i corresponds to binary code 2^i. The same coding can be achieved by shifting the number 1 i bits to the left `(1<<i)`. The looping effect, `loop()`, creates the allusion of seeing a full letter C. Setting the latchPin to HIGH causes the shifted out bytes to appear on the output pins of the 595 shift registers. It is then set to LOW in preparation for the next two bytes shift out.

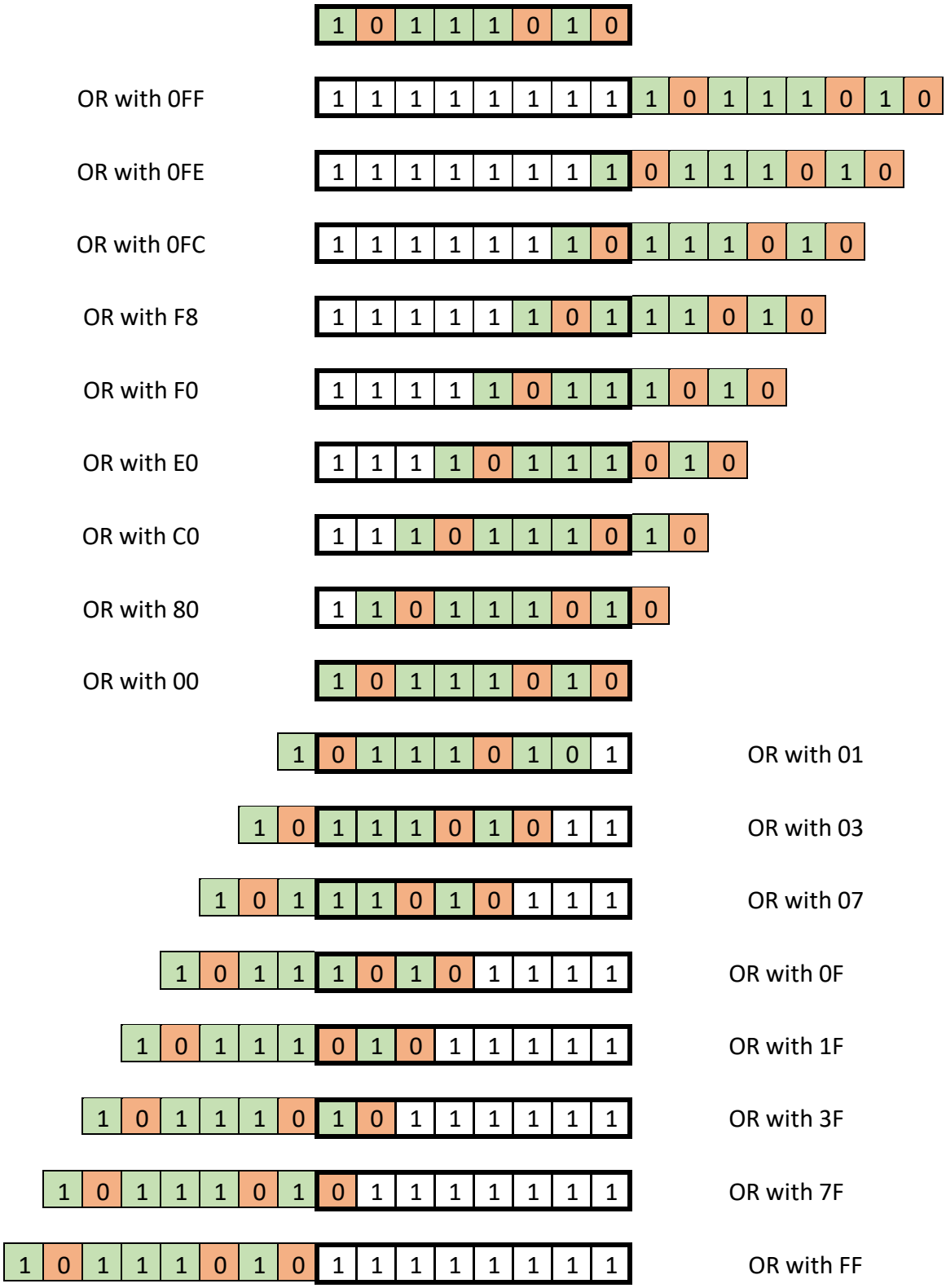
The result of running this program is shown below.



Next we turn to the discussion of how to create an animated display. Again, we illustrate using the letter "C".

The animation starts by flushing out, to the right, the whole column pattern for the letter C. This means that every entry should be shifted out 8 bits to the right `(Cols[i]>>8).` Then as we start shifting back to the left, the number of shift bits start to decrease. At the jth step we shift `(Cols[i]>>8-j)` bits. However, the remaining bits on the C-register still need to be disabled. This is accomplished by taking bitwise OR with `0XFF` properly shifted to the left `(Cols[i]>>8-j)|0XFF<<j`. As the patterns reach their original static positions (design positions) we need to continue shifting to the left. Therefore, we use `(Cols[i]<<j-8)`. The remaining bits on the C-register still need to be disabled. This is a more delicate situation as it requires taking bitwise OR with `0X00` at the beginning, `0X01` in the next step, `0X03` in the next step and so on. Thus, in step number `k,` we need to OR with $2^k - 1$ or `(1<<k)-1`. In our program implementation, we use the statement `(Cols[i]<<j-8)|((1<<j-8)-1)`.

The following example illustrates this discussion. It starts with a column pattern and applies the shifts discussed above. It takes 17 (2*16+1) steps to complete the full scroll of the pattern.



The full program is listed below

```
/*
  Animated letter C
 Displays a letter C which shifts from right to left
 by Mohamed Elguebeily

 */
//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
//Pin connected to DS of 74HC595
int dataPin = 11;
//Column pattern of letter C:
byte Cols[]={0xE7, 0xDB,0xBF,0xBF,0xBF,0xBF,0xDB,0xE7};

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

void loop()
  {
    int sh_col;
    for (int j=0;j<=16;j++){
      //time cycle to create the allusion of a printed pattern
      for (int t=0;t<200;t++){
        for(int i=0;i<=7;i++){
          if (j<=8){
            sh_col=Cols[i]>>(8-j)|0xFF<<j; //disable remaining bits
          }
          else{
            sh_col=Cols[i]<<(j-8)|(1<<j-8)-1; //disable remaining bits
          }
          shiftOut(dataPin,clockPin,MSBFIRST,sh_col); //pattern for row i
          shiftOut(dataPin,clockPin,LSBFIRST,1<<i); //row i
          digitalWrite(latchPin,HIGH);
          digitalWrite(latchPin, LOW);
        }
      }
    }
  }
```

It remains to comment on the time loop. It repeats the “pattern drawing” loop t times. This creates the allusion of seeing a pattern displayed on the dot matrix.