Introduction to robotics

5th lab

Remember, when possible, choose the wire color accordingly:

- **BLACK** for **GND (dark colors if not available)**
- **RED** for **POWER (3.3V / 5V / VIN) (bright colors if not available)**
- **Bright Colored** for read and write signal (use **red** when none available and **black** only as a last option)
- We know it is not always possible to respect this due to lack of wires, but the first rule is **DO NOT USE BLACK FOR POWER OR RED FOR GND!**

Now, let's pick it up where we left off…
Pull out your Arduino and breadboard and connect them like in the schematic. This is to "power up" the breadboard so we can easily have access to **5V** and **GND**.

Attention! Remember how the breadboard works. Use correct wire colors.

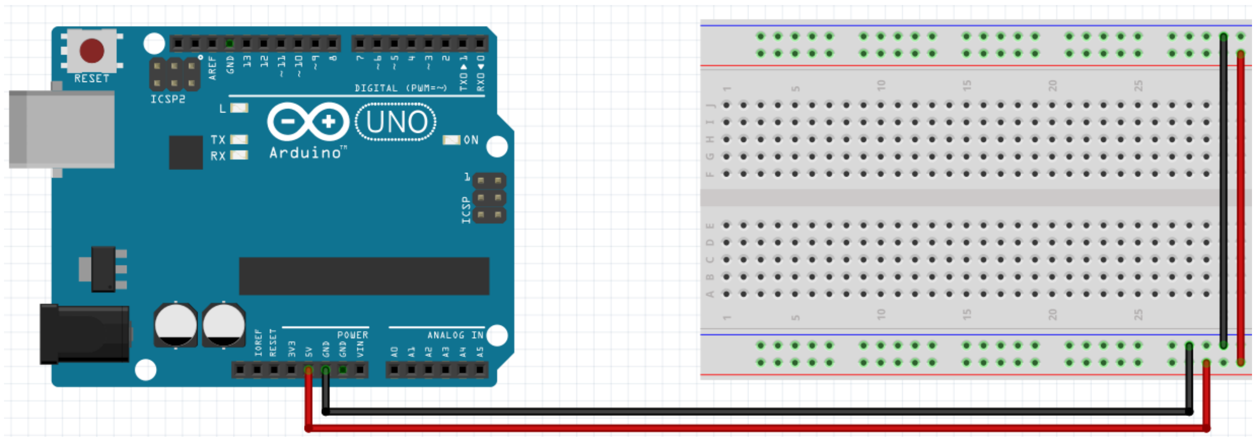


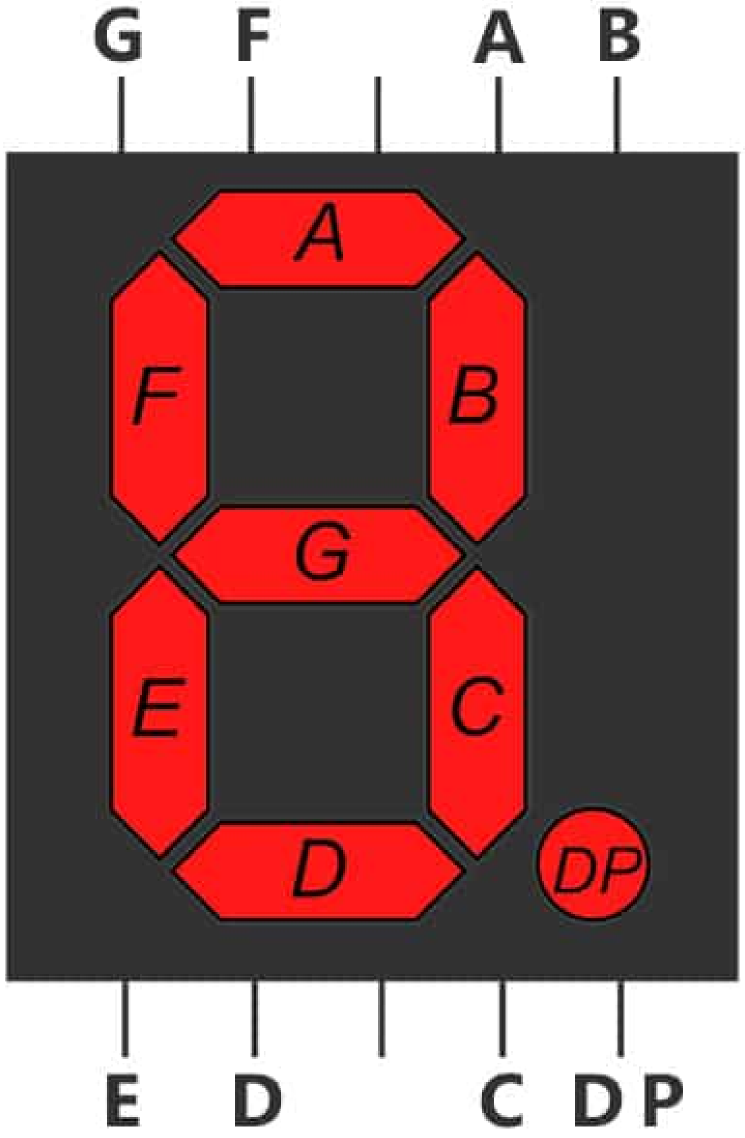
Fig. 1.1 - Default setup

0. Homework check

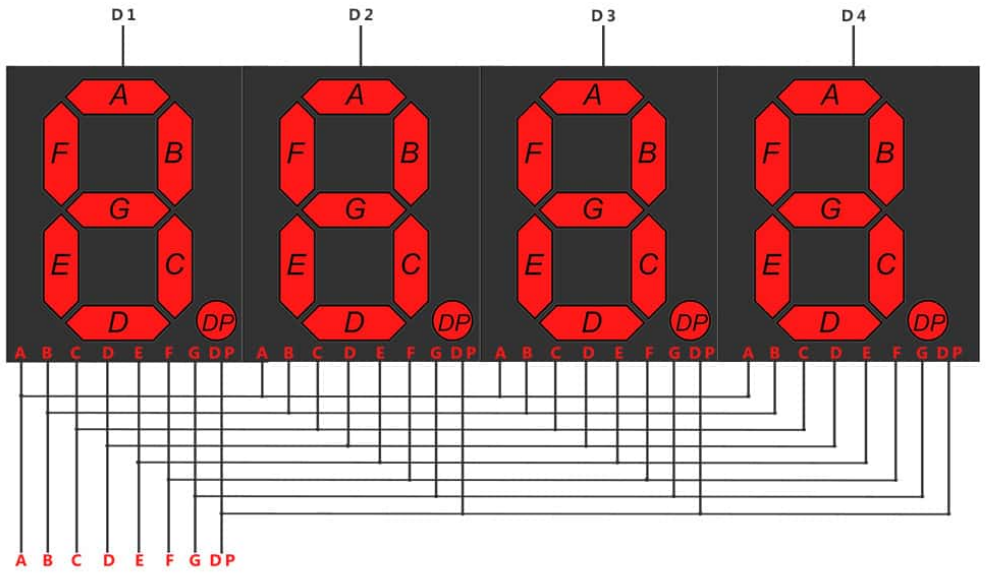
1. 4 digit 7-segment display

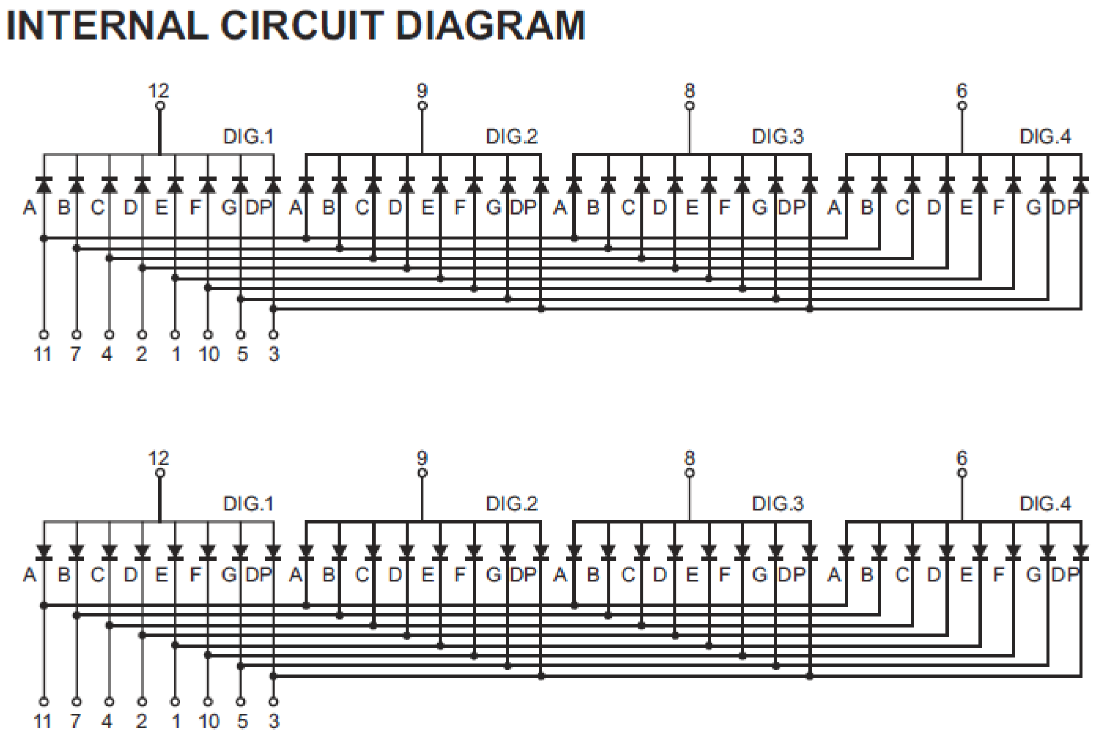
1.1 Introduction

A 7-Segment LED Display consists of 8 discrete segments arranged in the shape of the number "8", with an additional LED segment for a decimal point. These segments are controlled by individual input pins, allowing the display to show specific characters or numbers.
There are two wirings when it comes to 7-Segment LED displays: common anode and common cathode. Typically, all of the cathodes (negative terminals) or all of the anodes (positive terminals) of the segment LEDs are connected and brought out to a common pin; this is referred to as a "common cathode" or "common anode" device.

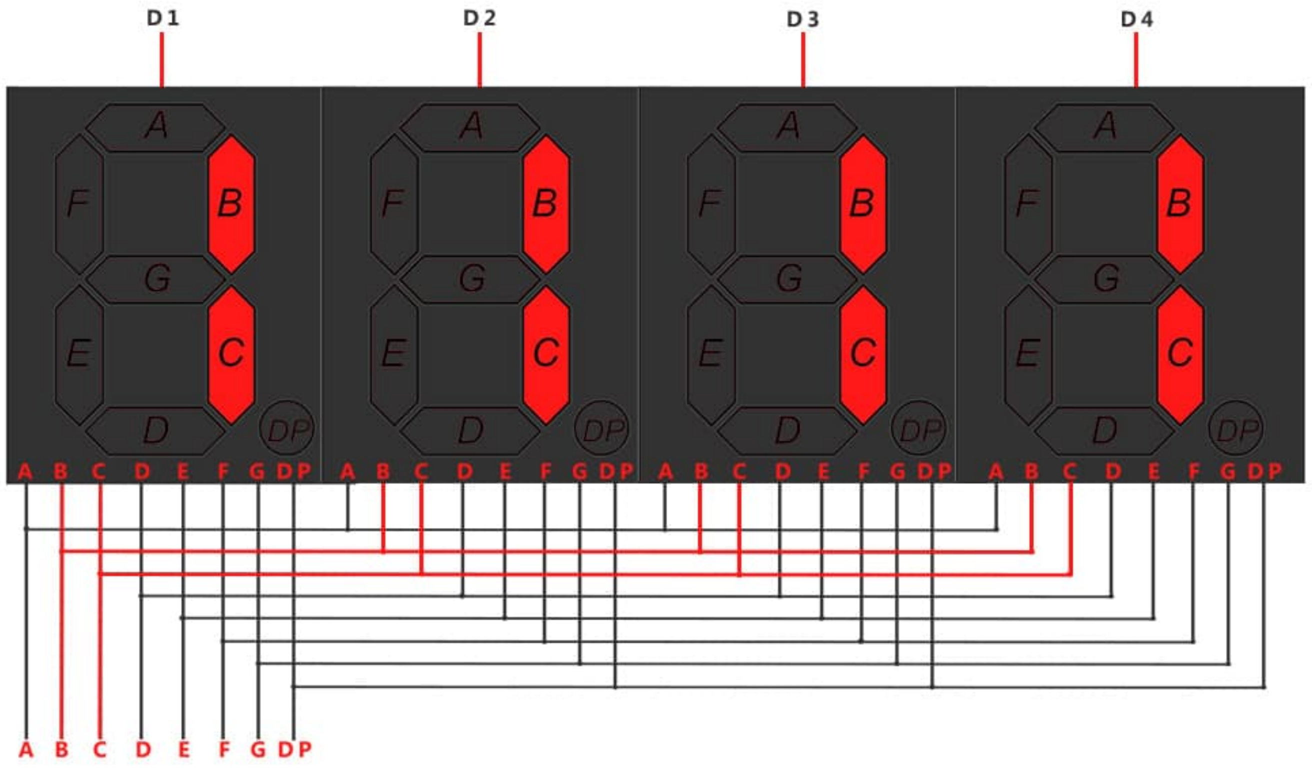


To create a 4-digit display, four individual 7-segment displays are usually connected together, with each display representing a single digit. To reduce the number of pins required for a 4-digit LED display, we multiplex the pins of each LED display block with the others. As a result, each LED block's segment pin is connected to all the other LED block's corresponding segments.





Four of these pins (D1, D2, D3, and D4) are used to control the individual digits and determine which signals pass through the LED blocks. This replaces the fixed common value in the 7 segment display common anode or cathode.
Assuming we have a common cathode, if we want to display the number 1111, we have to apply voltage to the D1, D2, D3 and D4 because all displays will show a digit. We also need to apply voltage to inputs B and C as shown below:

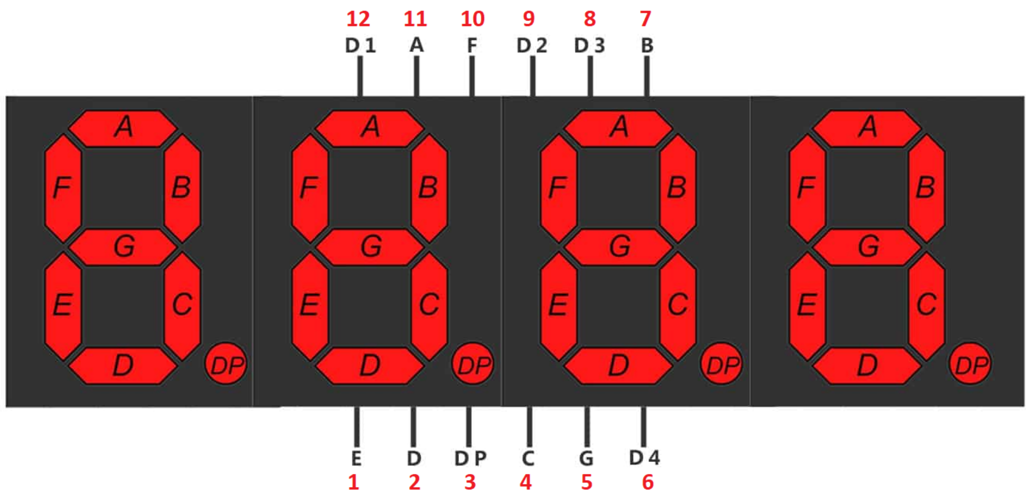


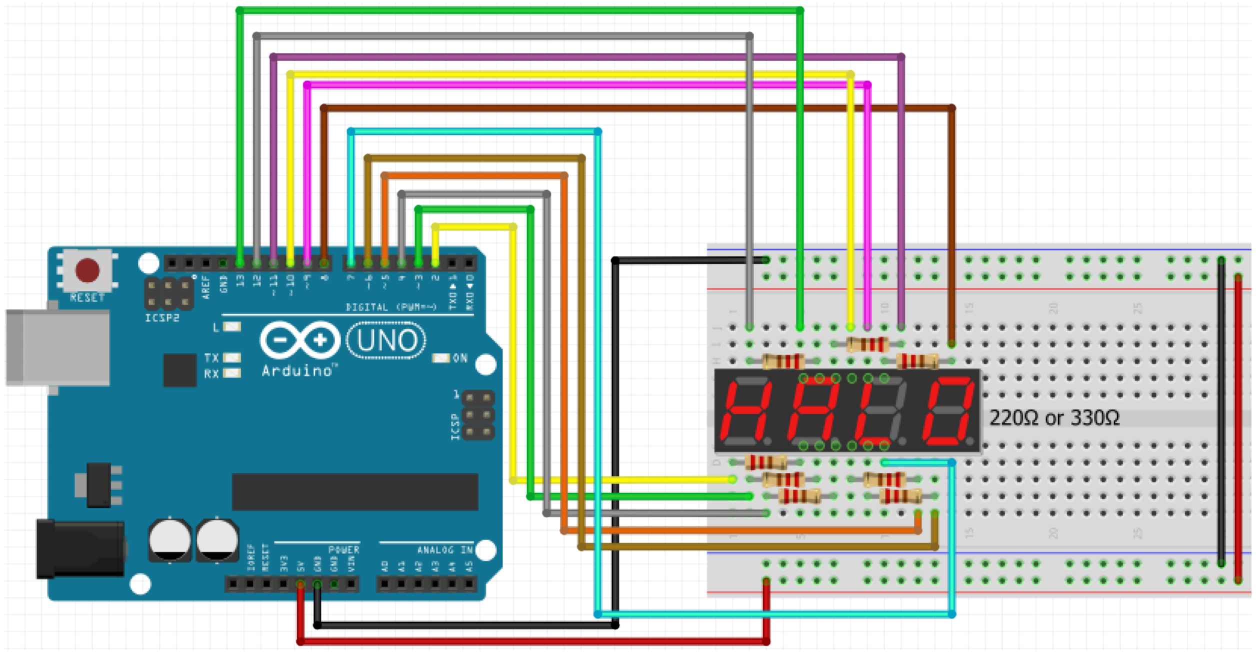
Test the display with a multimeter. Is it common anode or common cathode? How do you check?

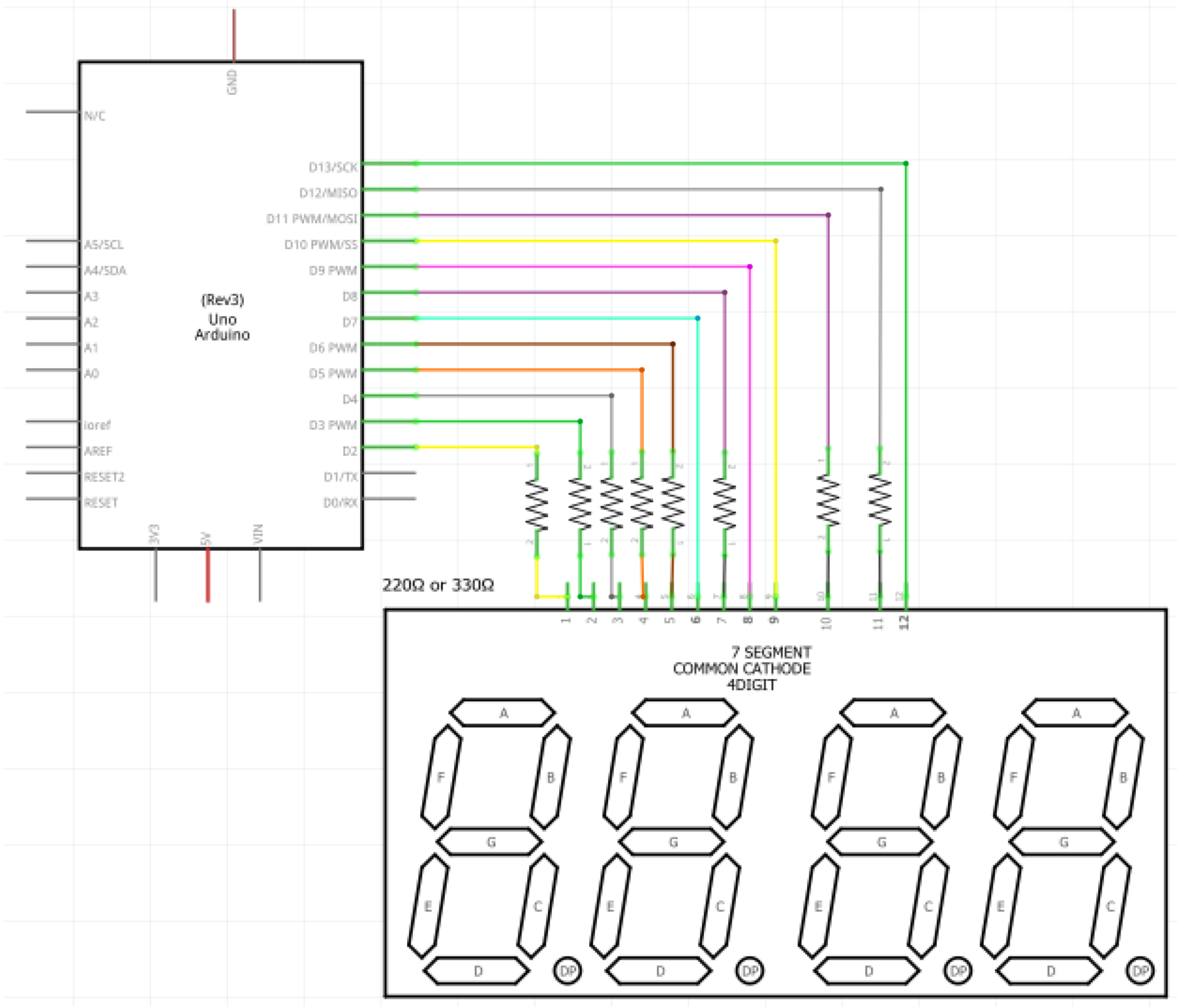
1.2 4-Digit 7-Segment LED display Connections **(discussion only)**

A typical 4-digit 7-segment LED display has 12 pins, with six pins on each side, as shown in the figure below.Four of these pins (D1, D2, D3, and D4) are used to control the individual digits and determine which signals pass through the LED blocks. The remaining pins correspond to the individual segments (A, B, C, D, E, F, G, and DP) .

Wiring diagram. D1 - D4 control the shown digits

Arduino PIN	**Display PIN**	**Schematic**
2	**1 (E)**	 12 D1 11 A 10 F 9 D2 8 D3 7 B A B C D E F G DP E 1 D 2 DP 3 C 4 G 5 D4 6
3	**2 (D)**	
4	**3 (DP)**	
5	**4 (C)**	
6	**5 (G)**	
7	**6 (D1)**	
8	**7 (B)**	
9	**8 (D2)**	
10	**9 (D3)**	
11	**10 (F)**	
12	**11 (A)**	
13	**12 (D4)**	

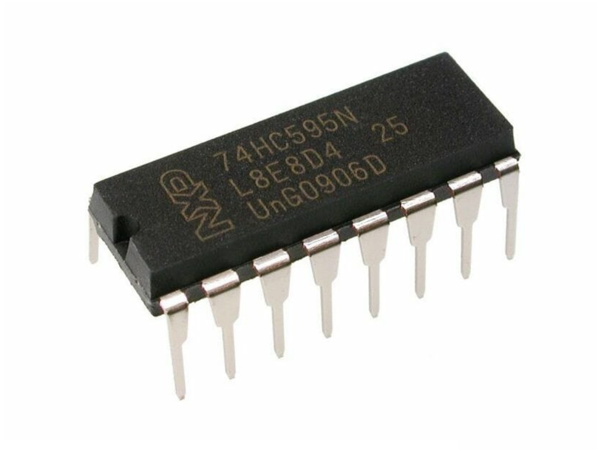

RESET
ICSP2
DIGITAL (PWM=~)
UNO
Arduino
TX
RX
ON
ICSP
POWER
ANALOG IN
220Ω or 330Ω


GND
N/C
D13/SCK
D12/MISO
D11 PWM/MOSI
D10 PWM/SS
D9 PWM
D8
D7
D6 PWM
D5 PWM
D4
D3 PWM
D2
D1/TX
D0/RX
A5/SCL
A4/SDA
A3
A2
A1
A0
ioref
AREF
RESET2
RESET
(Rev3)
Uno
Arduino
3V3
5V
VIN
220Ω or 330Ω
7 SEGMENT
COMMON CATHODE
4DIGIT

2. Shift Register: 74HC595

Consult the course and the datasheet for more details
https://www.diodes.com/assets/Datasheets/74HC595.pdf

- It's a sequential logic circuit
- Used for storage or transfer of binary data
- Can convert from serial to parallel data
- Used as memory or buffer stages within other chips
- For our use case, they allow us to use less pins in an Arduino



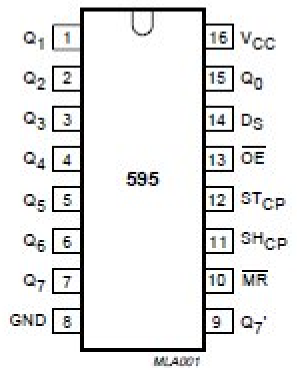


Fig.1 Pin configuration.

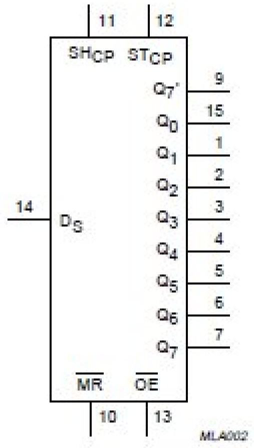
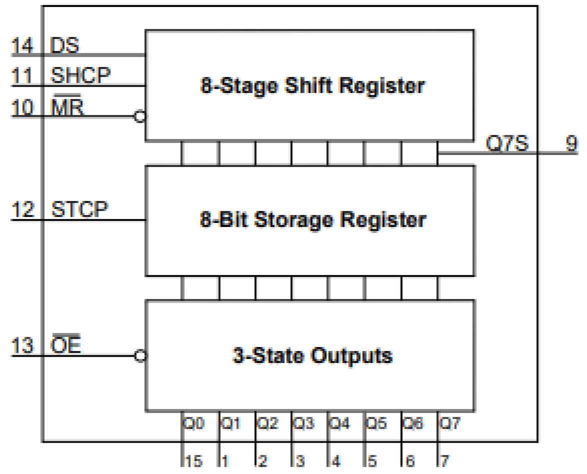
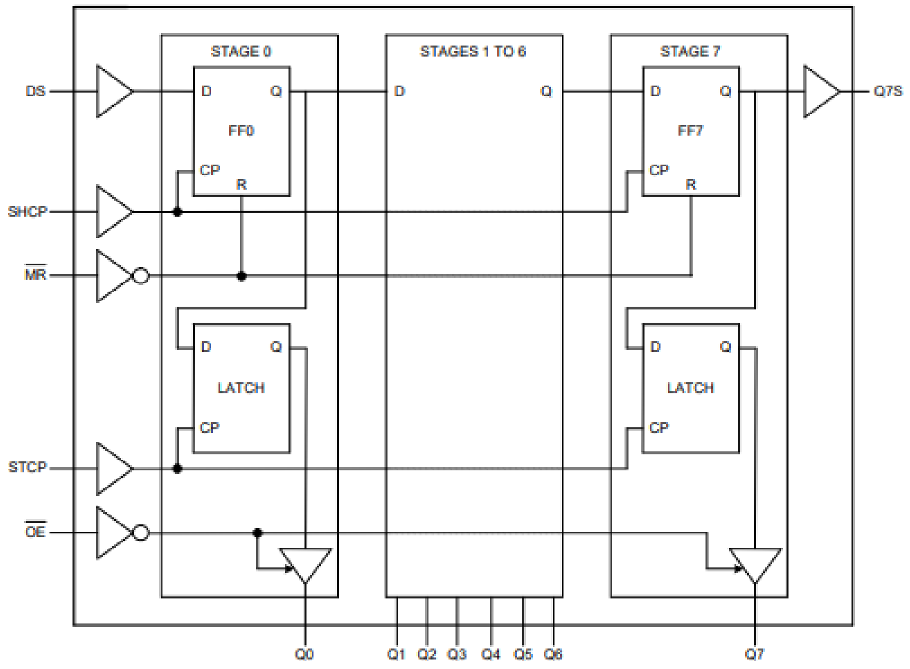


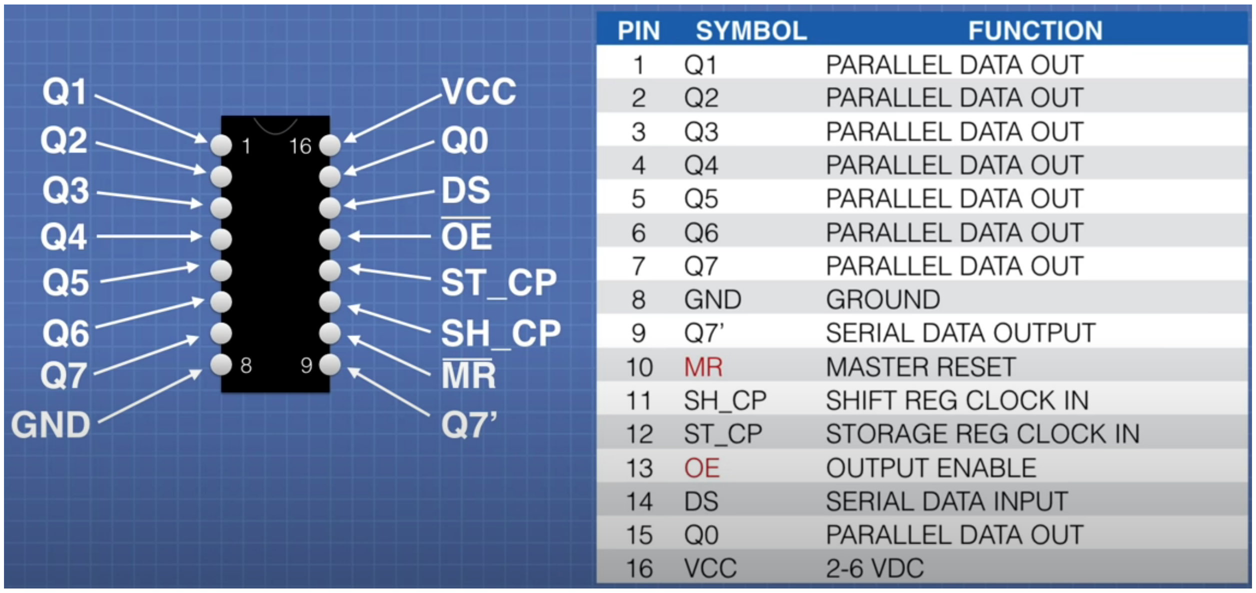
Fig.2 Logic symbol.

Functional Diagram



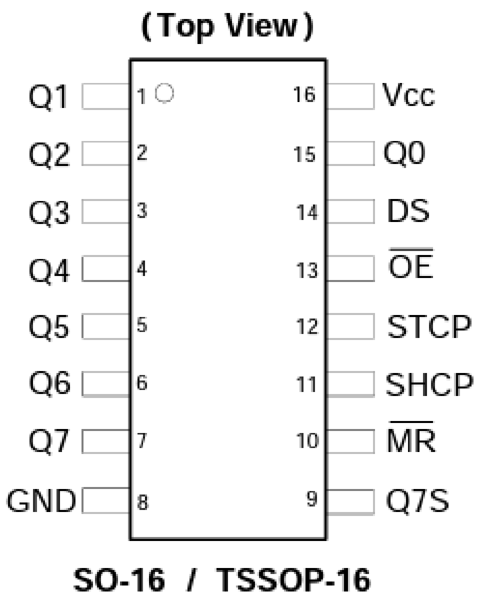
Logic Diagram

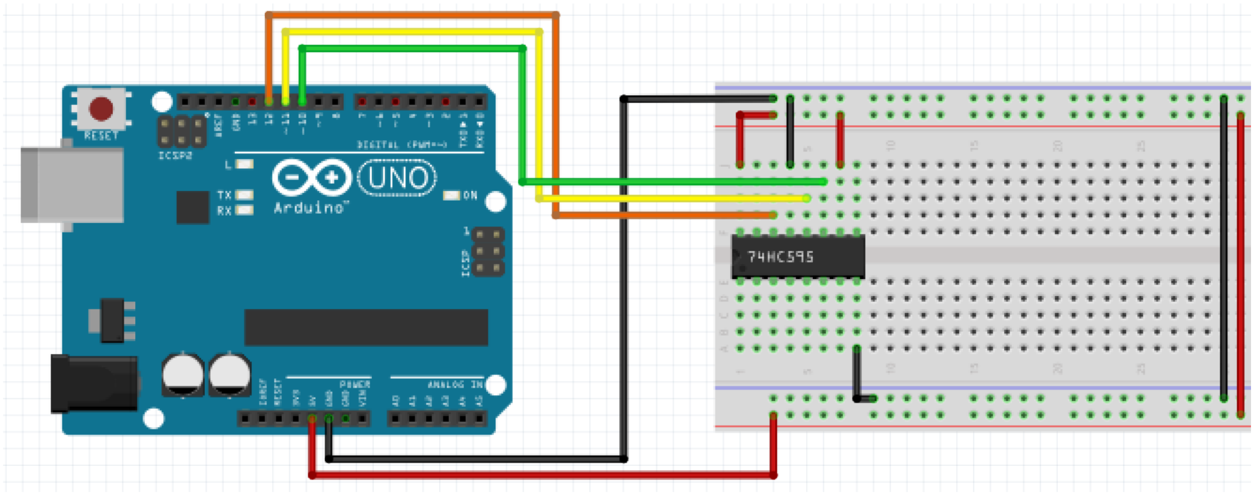


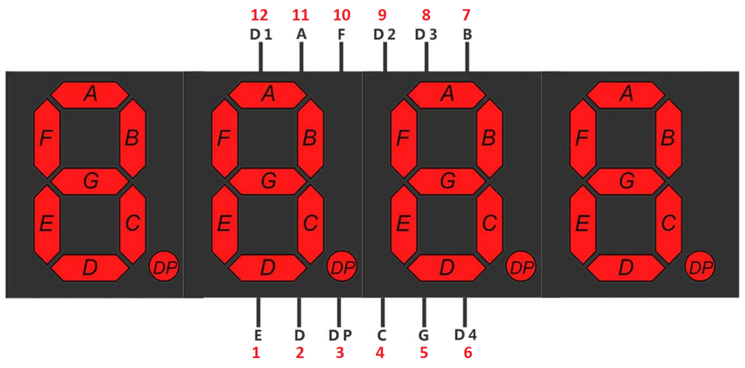


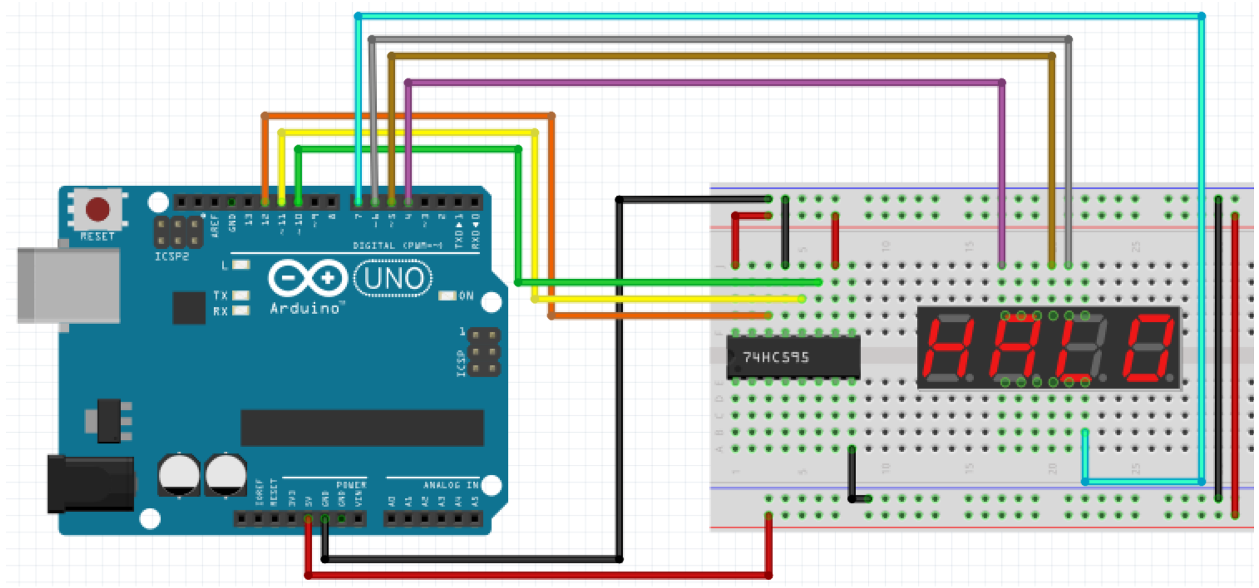
PIN	SYMBOL	FUNCTION
1	Q1	PARALLEL DATA OUT
2	Q2	PARALLEL DATA OUT
3	Q3	PARALLEL DATA OUT
4	Q4	PARALLEL DATA OUT
5	Q5	PARALLEL DATA OUT
6	Q6	PARALLEL DATA OUT
7	Q7	PARALLEL DATA OUT
8	GND	GROUND
9	Q7'	SERIAL DATA OUTPUT
10	MR	MASTER RESET
11	SH_CP	SHIFT REG CLOCK IN
12	ST_CP	STORAGE REG CLOCK IN
13	OE	OUTPUT ENABLE
14	DS	SERIAL DATA INPUT
15	Q0	PARALLEL DATA OUT
16	VCC	2-6 VDC

2.1 Connections

Shift Register PIN	Arduino PIN	Schematic (shift register)
DS	12	 (Top View) Q1 1 Q2 2 Q3 3 Q4 4 Q5 5 Q6 6 Q7 7 GND 8 16 Vcc 15 Q0 14 DS 13 OE 12 STCP 11 SHCP 10 MR 9 Q7S SO-16 / TSSOP-16
STCP	11	
SHCP	10	
GND	GND	
VCC	5V	
MR	5V	
OE	GND	

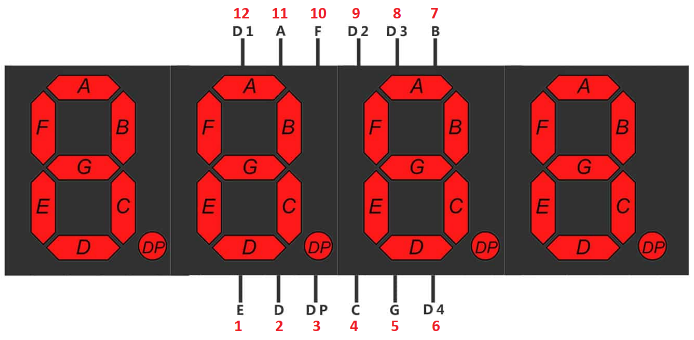
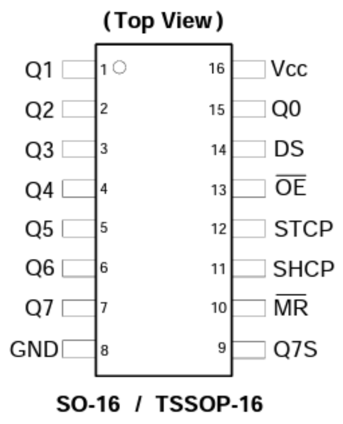


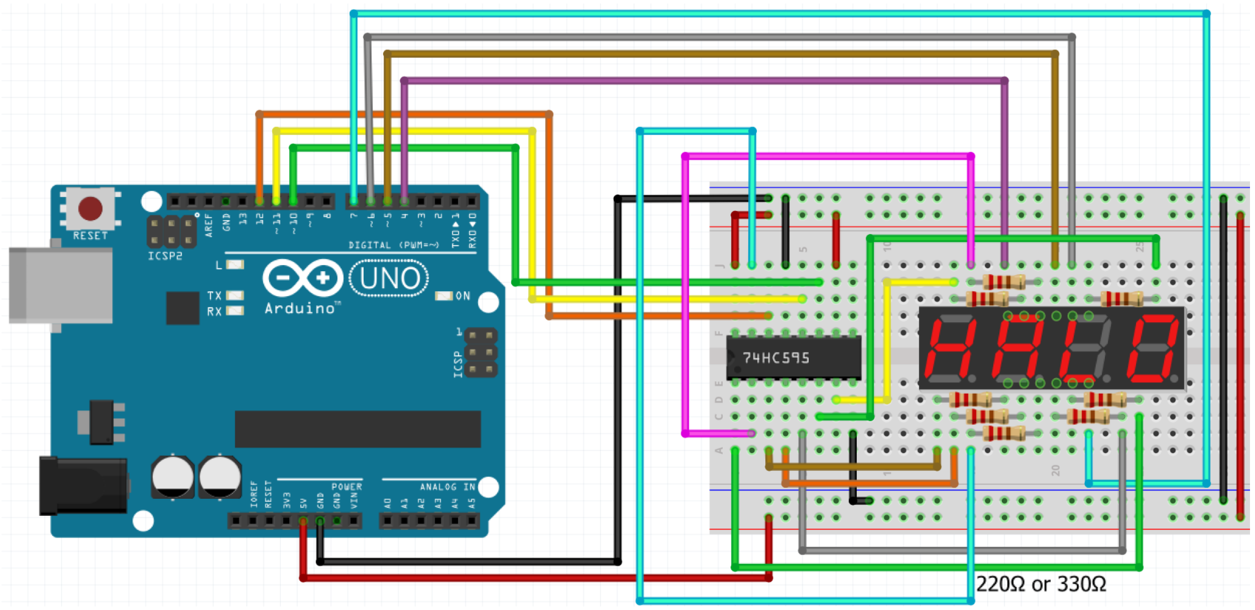
Display PIN	Arduino	Schematic
D1	4	 12 D1 11 A 10 F 9 D2 8 D3 7 B E 1 D 2 DP 3 C 4 G 5 D4 6 A B C D E F G DP
D2	5	
D3	6	
D4	7	

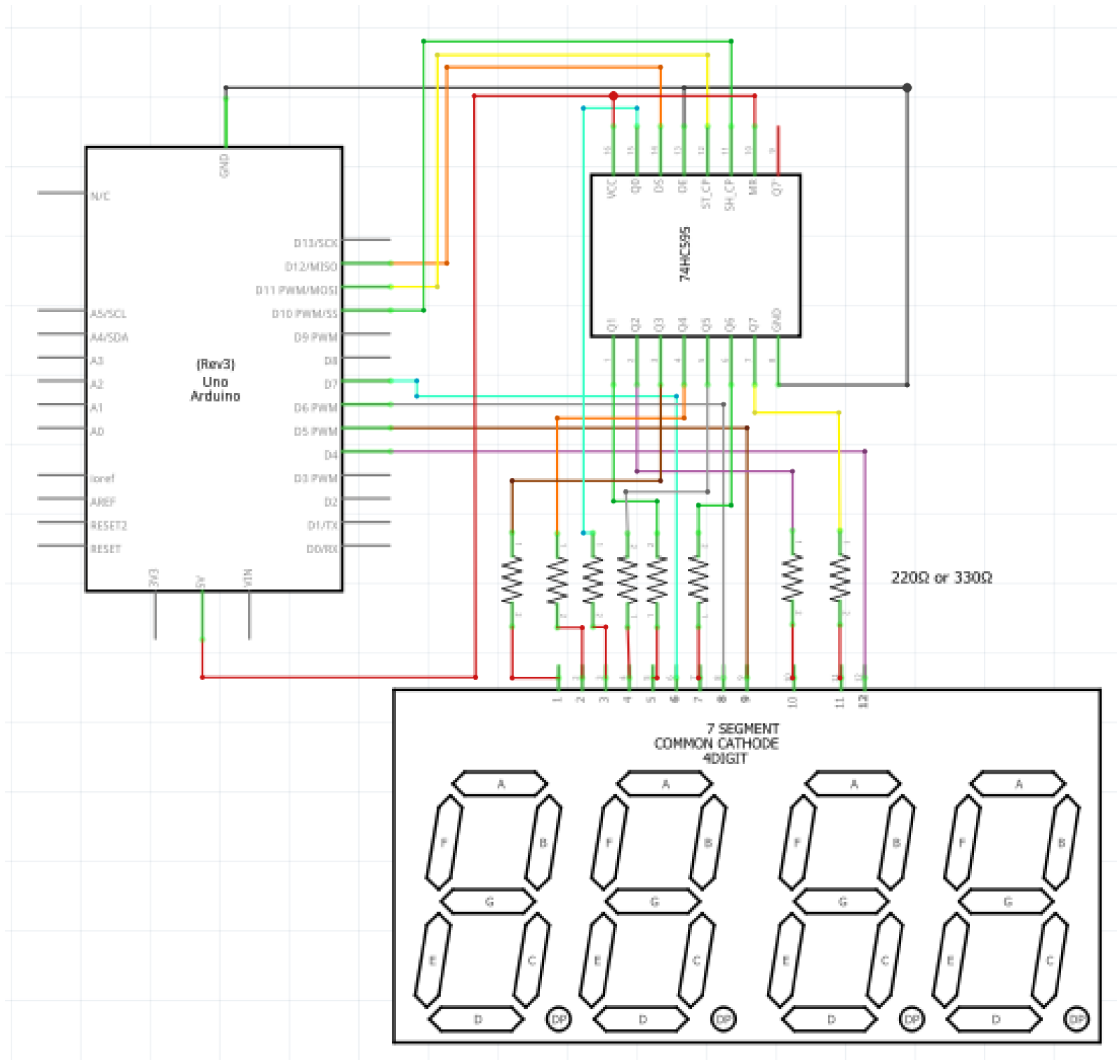


The D1 to D4 pins act as the collective grounding points (for common cathode displays) or power inputs (for common anode displays) for all segments corresponding to a particular digit in a multi-digit display. Unlike a single-digit 7-segment display where this connection is fixed to ground or power, in a 4-digit display, these pins are dynamically controlled by connecting them to digital output pins on a microcontroller. By toggling these pins between HIGH and LOW, we can selectively activate or deactivate each digit, enabling individual control of each segment across all four digits.

Connect the segments to the shift register using 220 or 330 ohm resistors!

Shift Register PIN	Display PIN (220Ω / 330Ω)	Schematic	Schematic (1 digit)
Q7	A	 12 D1 11 A 10 F 9 D2 8 D3 7 B A B C D E F G DP E 1 D 2 DP 3 C 4 G 5 D4 6	 (Top View) Q1 1 16 Vcc Q2 2 15 Q0 Q3 3 14 DS Q4 4 13 OE Q5 5 12 STCP Q6 6 11 SHCP Q7 7 10 MR GND 8 9 Q7S SO-16 / TSSOP-16
Q6	B		
Q5	C		
Q4	D		
Q3	E		
Q2	F		
Q1	G		
Q0	DP		




GND
N/C
D13/SCK
D12/MISO
D11 PWM/MOSI
D10 PWM/SS
D9 PWM
D8
D7
D6 PWM
D5 PWM
D4
D3 PWM
D2
D1/TX
D0/RX
A5/SCL
A4/SDA
A3
A2
A1
A0
ioref
AREF
RESET2
RESET
3V3
5V
VIN
(Rev3)
Uno
Arduino
VCC Q0 DS OE ST_CP SH_CP MR Q7'
74HC595
Q1 Q2 Q3 Q4 Q5 Q6 Q7 GND
220Ω or 330Ω
7 SEGMENT
COMMON CATHODE
4DIGIT
A B C D E F G DP

Let’s code!

2.2 Cycle through all the segments, turning them all on and off. Write bits manually.

Use this function to test the connections. After seeing that it all works, de-comment some of the digit lines in setup() in order to turn off any of the 4 digits.

```
// DS = [D]ata [S]torage - data
// STCP = [ST]orage [C]lock [P]in latch
// SHCP = [SH]ift register [C]lock [P]in clock

const byte latchPin = 11; // STCP to 12 on Shift Register
const byte clockPin = 10; // SHCP to 11 on Shift Register
const byte dataPin = 12; // DS to 14 on Shift Register

// Pin assignments for controlling the common cathode/anode pins of the 7-segment digits
const byte segD1 = 4;
const byte segD2 = 5;
const byte segD3 = 6;
const byte segD4 = 7;

// Size of the register in bits
const byte regSize = 8;

// Array to keep track of the digit control pins
byte displayDigits[] = {
  segD1, segD2, segD3, segD4
};
const byte displayCount = 4; // Number of digits in the display

// Array representing the state of each bit in the shift register
byte registers[regSize];

void setup() {
  // Initialize the digital pins connected to the shift register as outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize the digit control pins as outputs and turn them off
  for (byte i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  // Decomment any of the following lines to turn off the specific digit.
  // digitalWrite(displayDigits[0], HIGH);
  // digitalWrite(displayDigits[1], HIGH);
  // digitalWrite(displayDigits[2], HIGH);
  // digitalWrite(displayDigits[3], HIGH);
  // Initialize serial communication
  Serial.begin(9600);
}

void loop() {
  // Turn on all segments in forward order and print the state
  for (int i = 0; i < regSize; i++) {
    registers[i] = HIGH;
    writeReg(registers);
    printRegisters(); // Print the state for debugging
    delay(200); // Small delay to see the effect on the display
  }
  Serial.println(); // New line for readability

  // Turn off all segments in reverse order and print the state
  for (int i = regSize - 1; i >= 0; i--) {
    registers[i] = LOW;
    writeReg(registers);
    printRegisters(); // Print the state for debugging
    delay(200); // Small delay to see the effect on the display
  }
  Serial.println(); // New line for readability
}

// Function to write the contents of the 'encoding' array to the shift register
void writeReg(byte encoding[]) {
  digitalWrite(latchPin, LOW); // Begin sending data to the shift register
  for (int i = 0; i < regSize; i++) { // Loop over each bit
    digitalWrite(clockPin, LOW); // Prepare to send the bit
    digitalWrite(dataPin, encoding[i]); // Send the bit
    digitalWrite(clockPin, HIGH); // Clock in the bit
  }
  digitalWrite(latchPin, HIGH); // Latch the data, updating the output of the shift register
}
```

```
void printRegisters() {
  for (int i = 0; i < regSize; i++) {
    Serial.print(registers[i]);
  }
  Serial.println();
}
```

2.3 Manual Encoding and Bit Writing using encoding matrix

This technique employs a detailed process where the binary representation for display symbols is managed via an encoding matrix. Each bit is individually and manually set to high or low states and written to the shift register with precise control over the clock signal.

Advantages:

- Fundamental Understanding: Provides an in-depth insight into shift register operations and binary serialization.
- Direct Bit Control: Offers an explicit demonstration of how data bits are processed and controlled in sequence.

Disadvantages:

- Complex Implementation: More prone to human error due to the meticulous nature of managing individual bits.
- Code Length: Results in lengthier code which may impact readability and comprehension, particularly for those new to programming.
- Execution Speed: The method may exhibit slower operational speeds because of the granularity of manual bit handling.

```
// Define pin connections to the shift register
const int latchPin = 11; // Pin for latching data into the register
const int clockPin = 10; // Pin for the shift register clock
const int dataPin = 12;  // Pin for the serial data input to the shift register

// Define control pins for the individual digits of the display
const int segD1 = 4; // Control pin for the first digit
const int segD2 = 5; // Control pin for the second digit
const int segD3 = 6; // Control pin for the third digit
const int segD4 = 7; // Control pin for the fourth digit

// Constants for the register and number of encodings
const byte regSize = 8; // Number of bits in the register
```

```
// Array of digit control pins
int displayDigits[] = {
  segD1, segD2, segD3, segD4
};
const int displayCount = 4; // Number of digits in the display
const int encodingsNumber = 16; // Number of different character encodings

// Encoding array representing the segments of the display for numbers 0-9 and letters A-F
byte encodingArray[encodingsNumber][regSize] = {
// A  B  C  D  E  F  G  DP
  {1, 1, 1, 1, 1, 1, 0, 0},  // 0
  {0, 1, 1, 0, 0, 0, 0, 0},  // 1
  {1, 1, 0, 1, 1, 0, 1, 0},  // 2
  {1, 1, 1, 1, 0, 0, 1, 0},  // 3
  {0, 1, 1, 0, 0, 1, 1, 0},  // 4
  {1, 0, 1, 1, 0, 1, 1, 0},  // 5
  {1, 0, 1, 1, 1, 1, 1, 0},  // 6
  {1, 1, 1, 0, 0, 0, 0, 0},  // 7
  {1, 1, 1, 1, 1, 1, 1, 0},  // 8
  {1, 1, 1, 1, 0, 1, 1, 0},  // 9
  {1, 1, 1, 0, 1, 1, 1, 0},  // A
  {0, 0, 1, 1, 1, 1, 1, 0},  // b
  {1, 0, 0, 1, 1, 1, 0, 0},  // C
  {0, 1, 1, 1, 1, 0, 1, 0},  // d
  {1, 0, 0, 1, 1, 1, 1, 0},  // E
  {1, 0, 0, 0, 1, 1, 1, 0},  // F

};

void setup() {
  // Initialize the shift register pins and the display control pins
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize digit control pins and set them to off
  for (int i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  Serial.begin(9600); // Start serial communication (not used in this program)
}

void loop() {
 // Sequentially display all encodings on the 7-segment display
 for (int i = 0; i < encodingsNumber; i++) {
  writeReg(encodingArray[i]); // Write the encoding to the shift register
  delay(400); // Wait 400 milliseconds before displaying the next character
 }
}

void writeReg(byte encoding[]) {
 // Send each bit of the encoding to the shift register
 digitalWrite(latchPin, LOW); // Prepare to send data
 for (int i = 0; i < regSize; i++) {
  digitalWrite(clockPin, LOW); // Set the clock pin low before sending data
  digitalWrite(dataPin, encoding[i]); // Send the data bit
  digitalWrite(clockPin, HIGH); // Clock the data bit into the register
 }
 digitalWrite(latchPin, HIGH); // Latch the data into the register to update the display
}
```

2.4 Lab ex 1: Byte Encoding with Manual Bit Referencing

Requirement: modify your initial code to implement byte encoding with manual bit referencing. Instead of the encoding matrix, you'll use byte arrays to represent each digit and symbol. Access and manipulate individual bits within these bytes to control the segments of your 7-segment display.
Tip: check out **bitRead** function.

This method streamlines the encoding process by using an array of bytes, each representing a unique symbol encoding. Bits are still written manually, but reference to their positions within the byte is used to simplify the operation.

Advantages:

- Structured Bit Manipulation: Introduces a more systematic approach to handling bits while maintaining manual control over the writing process.
- Improved Code Organization: Offers a balance between direct bit control and code efficiency.

Disadvantages:

- Manual Clock Control: Retains some complexity as the clock signal must still be managed manually.
- Moderate Code Verbosity: While less verbose than the matrix method, there remains an element of explicit bit handling which can add to code length.

2.4.1 Initial code

```
// Define pin connections to the shift register
const int latchPin = 11; // Pin for latching data into the register (STCP)
const int clockPin = 10; // Pin for the shift register clock (SHCP)
const int dataPin = 12;  // Pin for the serial data input to the shift register (DS)

// Define control pins for the individual digits of the display
const int segD1 = 4; // Control pin for the first digit
const int segD2 = 5; // Control pin for the second digit
const int segD3 = 6; // Control pin for the third digit
const int segD4 = 7; // Control pin for the fourth digit

// Constants for the register size in bits
const byte regSize = 8; // Define the register size to be 8 bits (1 byte)

// Array of digit control pins
int displayDigits[] = {
  segD1, segD2, segD3, segD4
};
const int displayCount = 4; // Number of digits in the display
const int encodingsNumber = 16; // Number of different character encodings

// Array to store the binary encoding for each segment of a 7-segment display for the digits 0-9 and
letters A-F
int byteEncodings[encodingsNumber] = {
 // Binary representations for each character
 //A B C D E F G DP
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B10111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11110110, // 9
  B11101110, // A
  B00111110, // b
  B10011100, // C
  B01111010, // d
  B10011110, // E
  B10001110  // F
};

void setup() {
  // Initialize the pin modes for the shift register pins and the display control pins
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize and turn off all display digit control pins
  for (int i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  Serial.begin(9600); // Start serial communication (unused in this snippet)
}

void loop() {
  // Loop through all character encodings and display them sequentially
  for (int i = 0; i < encodingsNumber; i++) {
    writeReg(byteEncodings[i]); // Write each encoding to the shift register
    delay(400); // Delay between each character display for visibility
  }
}

void writeReg(byte encoding) {
   // TODO: Implement the logic to write the byte array to the shift register
}
```

2.4.2 Solution

```
// Define pin connections to the shift register
const int latchPin = 11; // Pin for latching data into the register (STCP)
const int clockPin = 10; // Pin for the shift register clock (SHCP)
const int dataPin = 12;  // Pin for the serial data input to the shift register (DS)

// Define control pins for the individual digits of the display
const int segD1 = 4; // Control pin for the first digit
const int segD2 = 5; // Control pin for the second digit
const int segD3 = 6; // Control pin for the third digit
const int segD4 = 7; // Control pin for the fourth digit

// Constants for the register size in bits
const byte regSize = 8; // Define the register size to be 8 bits (1 byte)

// Array of digit control pins
int displayDigits[] = {
  segD1, segD2, segD3, segD4
};
const int displayCount = 4; // Number of digits in the display
const int encodingsNumber = 16; // Number of different character encodings
// Array to store the binary encoding for each segment of a 7-segment display for the digits 0-9 and
letters A-F
int byteEncodings[encodingsNumber] = {
 // Binary representations for each character
 //A B C D E F G DP
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B10111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11110110, // 9
  B11101110, // A
  B00111110, // b
  B10011100, // C
  B01111010, // d
  B10011110, // E
  B10001110  // F
};

void setup() {
  // Initialize the pin modes for the shift register pins and the display control pins
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize and turn off all display digit control pins
  for (int i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  Serial.begin(9600); // Start serial communication (unused in this snippet)
}

void loop() {
  // Loop through all character encodings and display them sequentially
  for (int i = 0; i < encodingsNumber; i++) {
    writeReg(byteEncodings[i]); // Write each encoding to the shift register
    delay(400); // Delay between each character display for visibility
  }
}
void writeReg(byte encoding) {
   // DONE: Implement the logic to write the byte array to the shift register
  // Function to write a byte to the shift register
  digitalWrite(latchPin, LOW); // Begin the transmission by pulling the latch low
  byte msb; // Variable to hold the most significant bit for transmission
  // Loop through each bit in the encoding and send it to the shift register
  for (int i = 0; i < regSize; i++) {
    // Read each bit from the encoding, starting with the most significant bit
    msb = bitRead(encoding, regSize - i - 1);
    // Send the bit to the shift register
    digitalWrite(clockPin, LOW);
    digitalWrite(dataPin, msb);
    digitalWrite(clockPin, HIGH); // Clock the bit into the shift register
  }
  digitalWrite(latchPin, HIGH); // Latch the data into the output register to update the display
}
```

2.5 Byte Encoding with shiftOut Function

Advancing towards abstraction, this method maintains the use of byte encodings for symbols while employing the shiftOut() function to handle the bit writing process. This function automates the control of data and clock pins for transferring bits to the shift register.

Advantages:

- Code Simplicity: Enhances code clarity by abstracting the details of bit writing, making it more accessible, especially for those new to electronics and programming.
- Operational Efficiency: Increases the efficiency and reliability of the operation due to the optimized nature of the shiftOut() function.
- Ease of Implementation: Simplifies the learning curve by reducing the need for intricate control over the bit writing process.

Disadvantages:

- Abstracted Details: May obscure the finer details of shift register mechanics, which could be important for advanced understanding.
- Limited Fine Control: Reduces the ability to manually adjust timing and sequence, which might be necessary for certain applications.

```
// Pin definitions for interfacing with the shift register
const int latchPin = 11; // Connects to STCP of the shift register
const int clockPin = 10; // Connects to SHCP of the shift register
const int dataPin = 12;  // Connects to DS of the shift register

// Pin definitions for controlling the individual digits of the 7-segment display
const int segD1 = 4;
const int segD2 = 5;
const int segD3 = 6;
const int segD4 = 7;

// Constant for the size of the shift register in bits (8 bits for a standard register)
const byte regSize = 8;

// Array to hold the pins that control the segments of each digit of the display
int displayDigits[] = {
  segD1, segD2, segD3, segD4
};
const int displayCount = 4; // Total number of digits in the display
const int encodingsNumber = 16; // Total number of encodings for the hexadecimal characters (0-F)
```

```
// Array holding binary encodings for numbers and letters on a 7-segment display
byte byteEncodings[encodingsNumber] = {
  // Encoding for segments A through G and the decimal point (DP)
  //A B C D E F G DP
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B10111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11110110, // 9
  B11101110, // A
  B00111110, // b
  B10011100, // C
  B01111010, // d
  B10011110, // E
  B10001110  // F
};

void setup() {
  // Setup function is run once at the start of the program
  // Set the shift register pins as outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize the digit control pins and set them to an off state
  for (int i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  Serial.begin(9600); // Begin serial communication, not used in this code snippet
}

void loop() {
  // Main loop runs indefinitely
  for (int i = 0; i < encodingsNumber; i++) {
    writeReg(byteEncodings[i]); // Write each encoding to the shift register to display it
    delay(400); // Wait for 400ms between updates to the display
  }
}

void writeReg(int encoding) {
  // Function to output a byte to the shift register
  digitalWrite(latchPin, LOW); // Pull latch low to start data transfer
  // Shift out the bits of the 'encoding' to the shift register
  shiftOut(dataPin, clockPin, MSBFIRST, encoding); // MSBFIRST means the most significant bit is
shifted out first
  // Pull latch high to transfer data from shift register to storage register
  digitalWrite(latchPin, HIGH); // This action updates the output of the shift register
}
```

2.6 Lab exercise 2: Implementing Single Digit Display Function

Your next objective is to create a function capable of displaying a single digit on the 7-segment display at any given time. Name this function activateDisplay and design it to selectively enable one of the four digits, laying the groundwork for later multiplexing operations.

2.6.1 Initial code

```
// Pin definitions for interfacing with the shift register
const int latchPin = 11; // Connects to STCP of the shift register
const int clockPin = 10; // Connects to SHCP of the shift register
const int dataPin = 12;  // Connects to DS of the shift register

// Pin definitions for controlling the individual digits of the 7-segment display
const int segD1 = 4;
const int segD2 = 5;
const int segD3 = 6;
const int segD4 = 7;

// Constant for the size of the shift register in bits (8 bits for a standard register)
const byte regSize = 8;

// Array to hold the pins that control the segments of each digit of the display
int displayDigits[] = {
  segD1, segD2, segD3, segD4
};
const int displayCount = 4; // Total number of digits in the display
const int encodingsNumber = 16; // Total number of encodings for the hexadecimal characters (0-F)

// Array holding binary encodings for numbers and letters on a 7-segment display
byte byteEncodings[encodingsNumber] = {
  // Encoding for segments A through G and the decimal point (DP)
  //A B C D E F G DP
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B10111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11110110, // 9
  B11101110, // A
  B00111110, // b
  B10011100, // C
  B01111010, // d
  B10011110, // E
  B10001110  // F
};

void setup() {
  // Setup function is run once at the start of the program
  // Set the shift register pins as outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize the digit control pins and set them to an off state
  for (int i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  Serial.begin(9600); // Begin serial communication, not used in this code snippet
}

void loop() {
  // Main loop runs indefinitely
  // TODO: Use the newly written activateDisplay(int displayNumber) function to change the active
digit
  for (int i = 0; i < encodingsNumber; i++) {
    writeReg(byteEncodings[i]); // Write each encoding to the shift register to display it
    delay(400); // Wait for 400ms between updates to the display
  }
}

void writeReg(int encoding) {
  // Function to output a byte to the shift register
  digitalWrite(latchPin, LOW); // Pull latch low to start data transfer
  // Shift out the bits of the 'encoding' to the shift register
  shiftOut(dataPin, clockPin, MSBFIRST, encoding); // MSBFIRST means the most significant bit is shifted
out first
  // Pull latch high to transfer data from shift register to storage register
  digitalWrite(latchPin, HIGH); // This action updates the output of the shift register
}



void activateDisplay(int displayNumber) {
  // TODO: Implement a function named 'activateDisplay' that will:
  // 1. Deactivate all digit control pins to prevent ghosting on the display.
  // 2. Activate only the digit corresponding to 'displayNumber' parameter.
  //    This will be used to control which digit is illuminated on a 7-segment display.
}
```

2.6.2 Solution

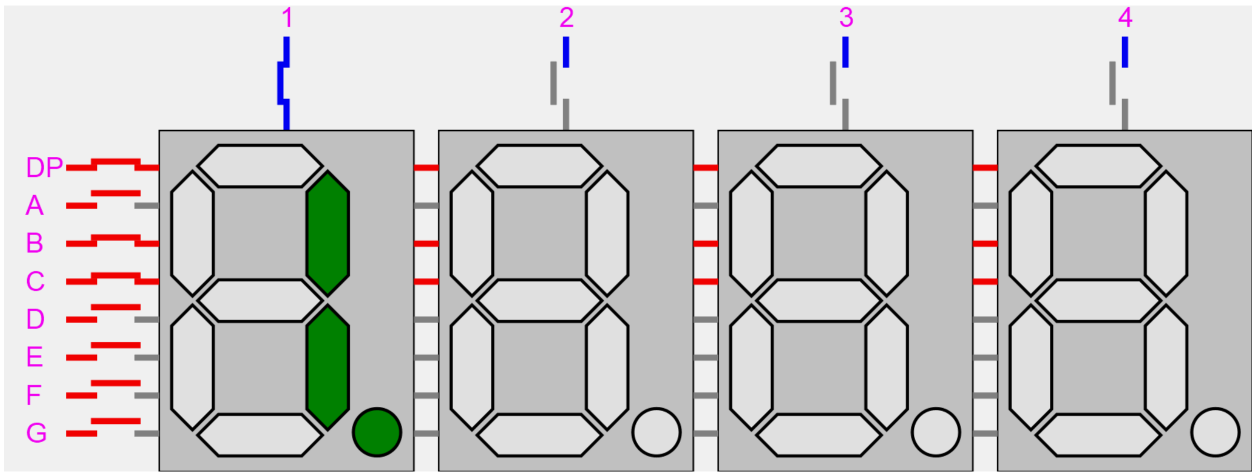
```
// Pin definitions for interfacing with the shift register
const int latchPin = 11; // Connects to STCP of the shift register
const int clockPin = 10; // Connects to SHCP of the shift register
const int dataPin = 12;  // Connects to DS of the shift register

// Pin definitions for controlling the individual digits of the 7-segment display
const int segD1 = 4;
const int segD2 = 5;
const int segD3 = 6;
const int segD4 = 7;

// Constant for the size of the shift register in bits (8 bits for a standard register)
const byte regSize = 8;

// Array to hold the pins that control the segments of each digit of the display
int displayDigits[] = {
  segD1, segD2, segD3, segD4
};
const int displayCount = 4; // Total number of digits in the display
const int encodingsNumber = 16; // Total number of encodings for the hexadecimal characters (0-F)

// Array holding binary encodings for numbers and letters on a 7-segment display
byte byteEncodings[encodingsNumber] = {
  // Encoding for segments A through G and the decimal point (DP)
  //A B C D E F G DP
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B10111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11110110, // 9
  B11101110, // A
  B00111110, // b
  B10011100, // C
  B01111010, // d
  B10011110, // E
  B10001110  // F
};

void setup() {
  // Setup function is run once at the start of the program
  // Set the shift register pins as outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize the digit control pins and set them to an off state
  for (int i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  Serial.begin(9600); // Begin serial communication, not used in this code snippet
}

void loop() {
// DONE: Use the newly written activateDisplay(int displayNumber) function to change the active digit
  activateDisplay(1);
  for (int i = 0; i < encodingsNumber; i++) {
    writeReg(byteEncodings[i]); // Write each encoding to the shift register to display it
    delay(400); // Wait for 400ms between updates to the display
  }
}

void writeReg(int encoding) {
  // Function to output a byte to the shift register
  digitalWrite(latchPin, LOW); // Pull latch low to start data transfer
  // Shift out the bits of the 'encoding' to the shift register
  shiftOut(dataPin, clockPin, MSBFIRST, encoding); // MSBFIRST means the most significant bit is shifted
out first
  // Pull latch high to transfer data from shift register to storage register
  digitalWrite(latchPin, HIGH); // This action updates the output of the shift register
}


void activateDisplay(int displayNumber) {
  // DONE: Implement a function named 'activateDisplay' that will:
  // 1. Deactivate all digit control pins to prevent ghosting on the display.
  for (int i = 0; i < displayCount; i++) {
    digitalWrite(displayDigits[i], HIGH);
  }
  // 2. Activate only the digit corresponding to 'displayNumber' parameter.
  //    This will be used to control which digit is illuminated on a 7-segment display.
  digitalWrite(displayDigits[displayNumber], LOW);
}
```

3. Multiplexing

Multiplexing is a technique used in electronics to drive multiple display elements with fewer inputs or control lines. In the context of a 4-digit 7-segment display, multiplexing involves controlling each segment of all digits by turning them on and off very quickly in sequence. Since human vision has a persistence effect, if this switching is done fast enough, the individual segments appear to be continuously illuminated. This creates the illusion of a steady display across all digits.

By rapidly activating each digit in turn, while simultaneously setting the appropriate segments for that digit, we can display a different number on each digit. The key is to cycle through all the digits many times per second. Typically, for the display to appear steady and without flicker to the human eye, each digit should be activated at a rate faster than the flicker fusion threshold, which is around 60 Hz or more.

This method is not only efficient—requiring fewer pins from a microcontroller—but it also allows for dynamic control of each digit, enabling the display of different numbers or patterns.



3.1 Lab exercise 3: Multiplexing to Display a Timer Using millis()

Display a Number with Multiplexing
For this task, you will refine the control of your 4-digit 7-segment display by implementing multiplexing. This technique allows each digit to be turned on and off rapidly in sequence, which makes use of persistence of vision to display numbers across all four digits simultaneously.

Update your code to include a function writeNumber that takes an integer and displays it on the 7-segment display using multiplexing. This function should break down the integer into its individual digits and use activateDisplay to illuminate each digit in quick succession.

3.1.1 Initial code

```
// Define connections to the shift register
const int latchPin = 11; // Connects to STCP (latch pin) on the shift register
const int clockPin = 10; // Connects to SHCP (clock pin) on the shift register
const int dataPin = 12; // Connects to DS (data pin) on the shift register

// Define connections to the digit control pins for a 4-digit display
const int segD1 = 4;
const int segD2 = 5;
const int segD3 = 6;
const int segD4 = 7;

// Store the digits in an array for easy access
int displayDigits[] = {segD1, segD2, segD3, segD4};
const int displayCount = 4; // Number of digits in the display

// Define the number of unique encodings (0-9, A-F for hexadecimal)
const int encodingsNumber = 16;
// Define byte encodings for the hexadecimal characters 0-F
byte byteEncodings[encodingsNumber] = {
//A B C D E F G DP
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B10111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11110110, // 9
  B11101110, // A
  B00111110, // b
  B10011100, // C
  B01111010, // d
  B10011110, // E
  B10001110  // F
};
// Variables for controlling the display update timing
unsigned long lastIncrement = 0;
unsigned long delayCount = 50; // Delay between updates (milliseconds)
unsigned long number = 0; // The number being displayed


void setup() {
  // Initialize the pins connected to the shift register as outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Initialize digit control pins and set them to LOW (off)
  for (int i = 0; i < displayCount; i++) {
    pinMode(displayDigits[i], OUTPUT);
    digitalWrite(displayDigits[i], LOW);
  }
  // Begin serial communication for debugging purposes
  Serial.begin(9600);
}

void loop() {
 // TODO: Check if the current time is greater than 'lastIncrement' plus 'delayCount'
  // If so, increment the number and reset 'lastIncrement'
  // TODO: Display the incremented number on the 7-segment display using multiplexing
  // TODO: Ensure that 'number' wraps around after reaching 9999 to start over from 0
  // TODO: Display the incremented number on the 7-segment display using multiplexing
}

void writeReg(int digit) {
    // Prepare to shift data by setting the latch pin low
    digitalWrite(latchPin, LOW);
    // Shift out the byte representing the current digit to the shift register
    shiftOut(dataPin, clockPin, MSBFIRST, digit);
    // Latch the data onto the output pins by setting the latch pin high
    digitalWrite(latchPin, HIGH);
}

void activateDisplay(int displayNumber) {
  // Turn off all digit control pins to avoid ghosting
  for (int i = 0; i < displayCount; i++) {
    digitalWrite(displayDigits[i], HIGH);
  }
  // Turn on the current digit control pin
  digitalWrite(displayDigits[displayNumber], LOW);
}

void writeNumber(int number) {
  // TODO: Initialize necessary variables for tracking the current number and digit position
  // TODO: Loop through each digit of the current number
  // TODO: Extract the last digit of the current number
  // TODO: Activate the current digit on the display
  // TODO: Output the byte encoding for the last digit to the display
  // TODO: Implement a delay for multiplexing visibility
  // TODO: Move to the next digit
  // TODO: Update 'currentNumber' by removing the last digit
  // TODO: Clear the display to prevent ghosting between digit activations
}
```

3.1.2 Solution

```
// Define connections to the shift register
const int latchPin = 11; // Connects to STCP (latch pin) on the shift register
const int clockPin = 10; // Connects to SHCP (clock pin) on the shift register
const int dataPin = 12; // Connects to DS (data pin) on the shift register

// Define connections to the digit control pins for a 4-digit display
const int segD1 = 4;
const int segD2 = 5;
const int segD3 = 6;
const int segD4 = 7;

// Store the digits in an array for easy access
int displayDigits[] = {segD1, segD2, segD3, segD4};
const int displayCount = 4; // Number of digits in the display

// Define the number of unique encodings (0-9, A-F for hexadecimal)
const int encodingsNumber = 16;
// Define byte encodings for the hexadecimal characters 0-F
byte byteEncodings[encodingsNumber] = {
//A B C D E F G DP
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B10111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11110110, // 9
  B11101110, // A
  B00111110, // b
  B10011100, // C
  B01111010, // d
  B10011110, // E
  B10001110  // F
};
// Variables for controlling the display update timing
```

```
unsigned long lastIncrement = 0;
unsigned long delayCount = 50; // Delay between updates (milliseconds)
unsigned long number = 0; // The number being displayed

void setup() {
 // Initialize the pins connected to the shift register as outputs
 pinMode(latchPin, OUTPUT);
 pinMode(clockPin, OUTPUT);
 pinMode(dataPin, OUTPUT);

 // Initialize digit control pins and set them to LOW (off)
 for (int i = 0; i < displayCount; i++) {
  pinMode(displayDigits[i], OUTPUT);
  digitalWrite(displayDigits[i], LOW);
 }
 // Begin serial communication for debugging purposes
 Serial.begin(9600);
}

void loop() {
 // DONE: Check if the current time is greater than 'lastIncrement' plus 'delayCount'
 if (millis() - lastIncrement > delayCount) {
  // DONE: Increment the number and reset 'lastIncrement'
  number++;
  // DONE: Ensure that 'number' wraps around after reaching 9999 to start over from 0
  number %= 10000; // Wrap around after 9999
  lastIncrement = millis();
 }

 // DONE: Display the incremented number on the 7-segment display using multiplexing
 writeNumber(number);
}

void writeReg(int digit) {
  // Prepare to shift data by setting the latch pin low
  digitalWrite(latchPin, LOW);
  // Shift out the byte representing the current digit to the shift register
  shiftOut(dataPin, clockPin, MSBFIRST, digit);
  // Latch the data onto the output pins by setting the latch pin high
  digitalWrite(latchPin, HIGH);
}
```

```
void activateDisplay(int displayNumber) {
  // Turn off all digit control pins to avoid ghosting
  for (int i = 0; i < displayCount; i++) {
    digitalWrite(displayDigits[i], HIGH);
  }
  // Turn on the current digit control pin
  digitalWrite(displayDigits[displayNumber], LOW);
}

void writeNumber(int number) {
  // DONE: Initialize necessary variables for tracking the current number and digit position
  int currentNumber = number;
  int displayDigit = 3; // Start with the least significant digit
  int lastDigit = 0;

  // DONE: Loop through each digit of the current number
  while (currentNumber != 0) {
    // DONE: Extract the last digit of the current number
    lastDigit = currentNumber % 10;
    // DONE: Activate the current digit on the display
    activateDisplay(displayDigit);
    // DONE: Output the byte encoding for the last digit to the display
    writeReg(byteEncodings[lastDigit]);
    // DONE: Implement a delay if needed for multiplexing visibility
    delay(0); // A delay can be increased to visualize multiplexing
    // DONE: Move to the next digit
    displayDigit--;
    // DONE: Update 'currentNumber' by removing the last digit
    currentNumber /= 10;

    // DONE: Clear the display to prevent ghosting between digit activations
    writeReg(B00000000); // Clear the register to avoid ghosting
  }
}
```

4. Extra exercises

1. **Multiplexing extension:** Extend the 3.1 Multiplexing function so that it can write numbers such as “0123” and text.
2. **Countdown Timer:** Reverse the clock! Program a countdown timer where you can set the starting number and watch it tick down to zero.
3. **Custom Character Creation:** Experiment with creating and displaying your custom characters or symbols on the 7-segment display.
4. **Brightness Control:** Use PWM to adjust the brightness of the display for better visibility under different light conditions.
5. **Decimal Point Display:** Modify your display code to include decimal points, enabling the representation of floating-point numbers.
6. **Reaction Timer Game:** Test your reflexes with a game that challenges you to stop the timer as close to zero as possible.
7. **Scrolling Text:** Get creative by programming your display to scroll text across the 4 digits.
8. **Stopwatch Function:** Build a stopwatch complete with start, stop, and reset capabilities, all triggered by button presses.

5. Resources

- Persistence of vision: https://en.wikipedia.org/wiki/Persistence_of_vision
- Multiplexing: https://en.wikipedia.org/wiki/Multiplexing
- Shift register on arduino.cc (with daisy chain): https://www.arduino.cc/en/Tutorial/Foundations/ShiftOut
- Shiftout function: https://www.arduino.cc/reference/en/language/functions/advanced-io/shiftout/
- Good explanation of shift register on youtube (highly recommended): https://www.youtube.com/watch?v=Ys2fu4NINrA
- Datasheet on 74HC595: https://www.diodes.com/assets/Datasheets/74HC595.pdf
- https://softwareparticles.com/learn-how-a-4-digit-7-segment-led-display-works-and-how-to-control-it-using-an-arduino/
- http://www.circuitbasics.com/arduino-7-segment-display-tutorial/
- https://softwareparticles.com/learn-how-shift-registers-work-and-how-to-use-them-using-arduino/