VIDEO#39: Controlling a BIG LED Matrix?! How Shift Registers work!

Understanding LED Matrices and Shift Registers

Introduction to LED Matrices

An LED matrix is a display device composed of multiple LEDs arranged in a grid pattern. These matrices are commonly used for displaying characters, symbols, and animations. In this study, we explore the functionality and control mechanism of a 32x12 LED matrix, which consists of 384 LEDs in total.

Components of the LED Matrix

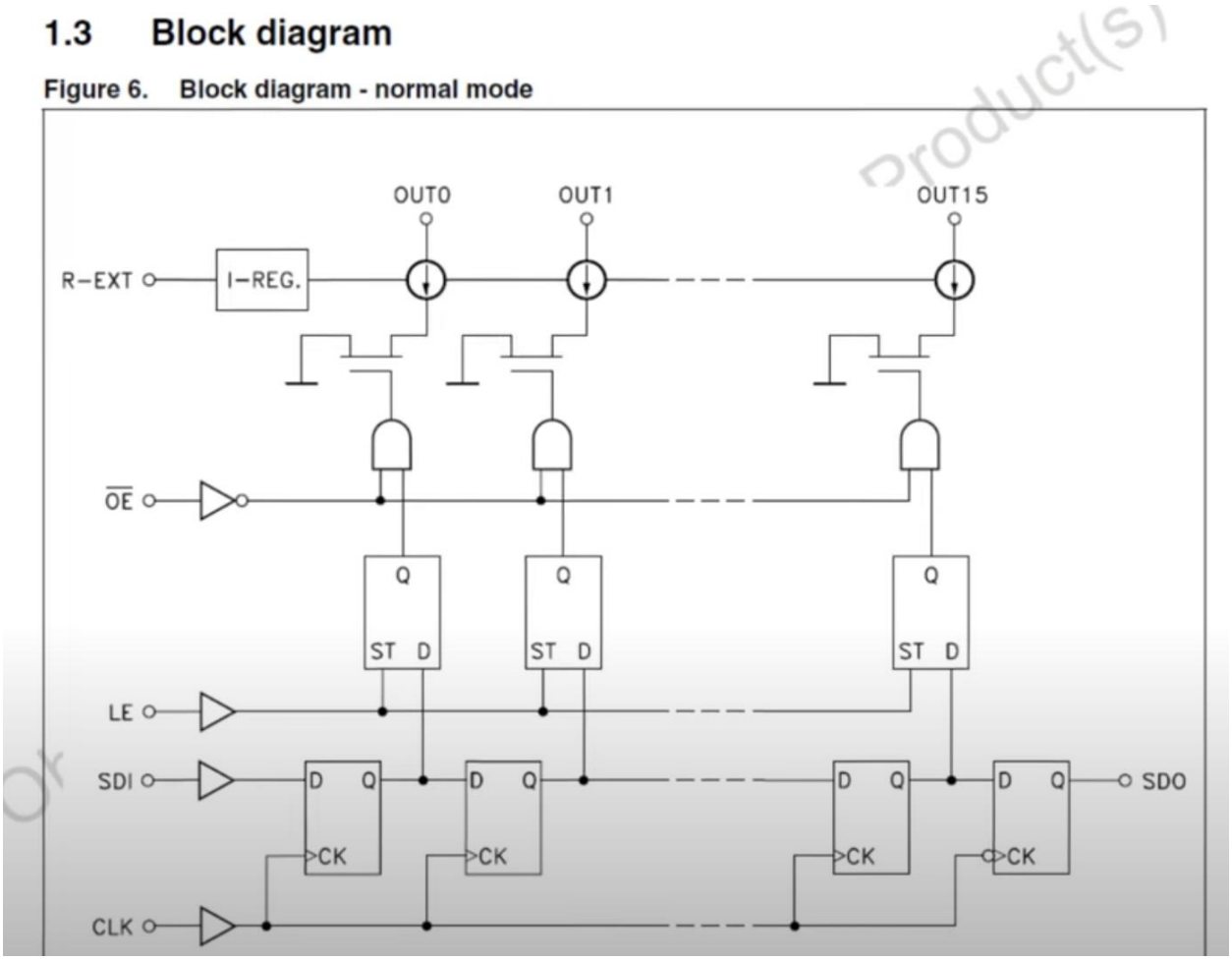
1. **LEDs:** The fundamental light-emitting components of the matrix.
2. **Printed Circuit Board (PCB):** Houses the LEDs, headers, and necessary electronics.
3. **Male and Female Headers:** Used for interfacing with microcontrollers.
4. **ICs (Integrated Circuits):** Includes shift registers and LED drivers.
5. **Capacitors and Resistors:** Used for signal stability and current regulation.

Understanding the Circuit Wiring

- The anodes of the LEDs are grouped in sets of three lines and connected to power supply traces.
- The cathodes of the LEDs are linked to **STP16C596** ICs, which function as **constant current LED sink drivers**.
- Each driver IC can control 16 LEDs, but due to parallel connections across lines, the entire matrix can be managed using **multiplexing**.

1.3 Block diagram

Figure 6. Block diagram - normal mode



Multiplexing Principle

- The LED matrix uses multiplexing to control 384 LEDs using a limited number of control pins.
- The cathodes of every **fourth** line are connected in parallel, reducing the number of required driver ICs.
- By switching power supply connections rapidly, different rows can be illuminated at different times, creating the illusion of a static image to the human eye.

Shift Registers and Their Working

- The **STP16C596 ICs** are 16-bit **serial-in parallel-out (SIPO) shift registers** that help control the LEDs efficiently.
- These ICs allow microcontrollers to use only **two pins** to control **multiple output pins**, crucial for limited GPIO scenarios.

- Shift registers operate using **D-type flip-flops**, which save the state of input data on a clock pulse.
- **Serial Data Input (SDI):** Inputs data to the shift register.
- **Clock Pin (CLK):** Moves the data through the shift register.
- **Latch Pin (LATCH):** Transfers the stored data to the output.

Schmitt Trigger Inverters

- **SN74LS15 Hex Schmitt Trigger Inverters** are used in the circuit to clean up signal noise.
- They ensure proper logic levels for data transmission.

Microcontroller Interfacing

- An **Arduino Nano** is used for controlling the LED matrix.
- The clock and latch signals are triggered in software using timer interrupts.
- A **Boolean matrix** is used to represent displayed text, such as "HI" or "COOL".

Coding for LED Matrix Control

1. **Setting up Timer Interrupts**
    - Timer 1 is configured to generate interrupts at **100 µs** and **200 µs**.
    - These interrupts control data transmission timing.
2. **Displaying Text**
    - The boolean matrix maps desired text onto the LED matrix.
    - Shift registers are used to send data bit by bit.
3. **Implementing Multiplexing**
    - **P-channel MOSFETs** are used to switch power lines.
    - Four digital output pins from the Arduino cycle through different row segments.
    - The final result is a dynamic display of scrolling text.

Challenges and Improvements

- **Timing issues** may result in flickering or improper display.
- **Optimization of code** can enhance efficiency and reduce latency.

- **More efficient multiplexing algorithms** can be implemented for smoother visuals.

This study demonstrates the working principles of LED matrices and shift registers, highlighting their efficiency in controlling large LED arrays with minimal GPIO usage. While implementing such a system requires careful timing and multiplexing, understanding the fundamental concepts of **shift registers, LED drivers, and microcontroller interfacing** can significantly improve design outcomes.

THE CODE IS:

```
// bool doonce = 0;

const char sd1 = 2;

const char sd2 = 3;

const char sd3 = 4;

const char clk = 5;

const char le = 6;

const char line1 = 7;

const char line2 = 8;

const char line3 = 9;

const char line4 = 10;

int counter = 0;

int counterms = 0;

int countermultiplex = 0;

bool matrix [12][32] =

{ {1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
0},

  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0},

  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0},

  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0},
```

```
  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
  {1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1}
};
bool matrix2[12][32];

void setup() {
  pinMode(sd1, OUTPUT);
  pinMode(sd2, OUTPUT);
  pinMode(sd3, OUTPUT);
  pinMode(clk, OUTPUT);
  pinMode(le, OUTPUT);
  pinMode(line1, OUTPUT);
  pinMode(line2, OUTPUT);
  pinMode(line3, OUTPUT);
  pinMode(line4, OUTPUT);
  digitalWrite(le, LOW);
  digitalWrite(line1, HIGH);
  digitalWrite(line2, HIGH);
```

```
  digitalWrite(line3, HIGH);
  digitalWrite(line4, HIGH);
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1B |= (1 << WGM12) | (1 << CS10);
  TIMSK1 |= (1 << OCIE1A) | (1 << OCIE1B);
  OCR1A = 2600;
  OCR1B = 1300;
  TCCR2A = 0;
  TCCR2A |= (1 << WGM21);
  TCCR2B = 0;
  TCCR2B |= (1 << CS20) | (1 << CS21) | (1 << CS22);
  TIMSK2 |= (1 << OCIE2A);
  OCR2A = 157;
}

ISR(TIMER1_COMPA_vect) {
  digitalWrite(clk, HIGH);
  digitalWrite(sd1, LOW);
  digitalWrite(sd2, LOW);
  digitalWrite(sd3, LOW);
  if (counter == 32) {
    digitalWrite(le, LOW);
    countermultiplex++;
    counter = 0;
  }
}

ISR(TIMER1_COMPB_vect) {
```

```
digitalWrite(clk, LOW);
if (countermultiplex == 4) {
  countermultiplex = 0;
}
if (countermultiplex == 0) {
  digitalWrite(line4, HIGH);
  digitalWrite(sd3, matrix[9][31 - counter]);
  digitalWrite(sd2, matrix[1][31 - counter]);
  digitalWrite(sd1, matrix[2][31 - counter]);
  digitalWrite(line1, LOW);
}
if (countermultiplex == 1) {
  digitalWrite(line1, HIGH);
  digitalWrite(sd3, matrix[3][31 - counter]);
  digitalWrite(sd2, matrix[4][31 - counter]);
  digitalWrite(sd1, matrix[5][31 - counter]);
  digitalWrite(line2, LOW);
}
if (countermultiplex == 2) {
  digitalWrite(line2, HIGH);
  digitalWrite(sd3, matrix[6][31 - counter]);
  digitalWrite(sd2, matrix[7][31 - counter]);
  digitalWrite(sd1, matrix[11][31 - counter]);
  digitalWrite(line3, LOW);
}
if (countermultiplex == 3) {
  digitalWrite(line3, HIGH);
  digitalWrite(sd3, matrix[0][31 - counter]);
  digitalWrite(sd2, matrix[10][31 - counter]);
```

```
    digitalWrite(sd1, matrix[8][31 - counter]);
    digitalWrite(line4, LOW);
  }
  counter++;
  if (counter == 32) {
    digitalWrite(le, HIGH);
  }
}

ISR(TIMER2_COMPA_vect) {
  counterms++;
  if (counterms == 50) {
    for (int n = 0; n < 12; n++) {
      for (int m = 0; m < 32; m++) {
        if (m == 0) {
          if (matrix[n][m] == 0) {
            matrix2[n][m] = 0;
            matrix2[n][m + 1] = 0;
          }
          if (matrix[n][m] == 1) {
            matrix2[n][m] = 0;
            matrix2[n][m + 1] = 1;
          }
        }
        else {
          if (matrix[n][m] == 0) {
            matrix2[n][m + 1] = 0;
          }
          if (matrix[n][m] == 1) {
```

```
            matrix2[n][m + 1] = 1;
          }
        }
      }
    }
    for (int n = 0; n < 12; n++) {
      for (int m = 0; m < 32; m++) {
        matrix[n][m] = matrix2[n][m];
      }
    }
    counterms = 0;
  }
}

void loop() {
}

//
```