İhsan Doğramacı Bilkent University

EEE 313 Term Project Report

3-bit ADC Design and Implementation

İnan Ulaş Metin
A. Emre Toktaş

January 2026

Contents

1 Introduction

In this project, we designed, configured, and implemented a 3-bit analog-to-digital converter (ADC) that digitizes a sinusoidal signal with 1 kHz frequency, 1 V peak amplitude, and a 3.5 V DC offset. The circuit was built using nMOS transistors (2N7000), pMOS transistors (BS250), an 8-channel analog multiplexer, an Arduino microcontroller, and standard passive components. The design was first simulated in LTSpice and then implemented in hardware.

The system consists of multiple subcircuits: a multiplexer with a resistor ladder to generate several analog reference voltage levels, a sample-and-hold circuit to store the input voltage during conversion, and a differential amplifier topology acting as the main comparator. The comparator compares the held input voltage with the selected reference voltage and produces a digital output.

2 LTSpice Design and Simulation

The full schematic of the design is shown in Fig. 1.

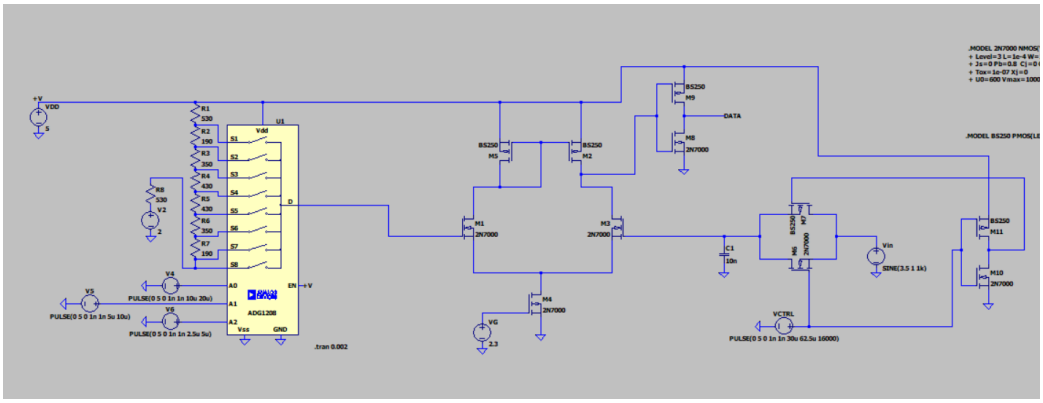

Figure 1: ADC circuit schematic in LTSpice.

2.1 MUX and Resistor Ladder

The MUX provides seven reference (comparison) voltage levels selected by the three-bit control signals. The comparison values are:

$$[2.53,\ 2.72,\ 3.07,\ 3.50,\ 3.93,\ 4.28,\ 4.47]\ \text{V}$$

Using a binary search approach, three comparisons are sufficient to determine the quantization level. Based on the comparisons, the Arduino board assigns the analog input value to one of the following representative voltage levels:

$$[2.50,\ 2.60,\ 2.88,\ 3.28,\ 3.72,\ 4.12,\ 4.40,\ 4.50]\ \text{V}$$

Both the 7 comparison and 8 assigning values are selected based on taking the values of our ofsetted sinusoidal at equal time intervals, so that a periodic sampling yields robust results. We used LTSPice to make sure of the resistor values to be used in the ladder, so that the intended comparison voltage levels are available at the ladder nodes. The design uses only the S1–S7 reference levels and not the 8th input, since 8 output values can be sufficiently assigned using only seven thresholds.

According to the design, the MUX control signals (A0, A1, A2) are to be generated by the Arduino. The algorithm starts by applying S4 (3.50 V). After a short settling delay, the comparator + inverter output is read. If the reading indicates that the sampled input is larger than S4, the Arduino selects the middle reference value of the upper half (e.g., S2), or vice versa. This procedure continues for three cycles until the final quantization level is determined.

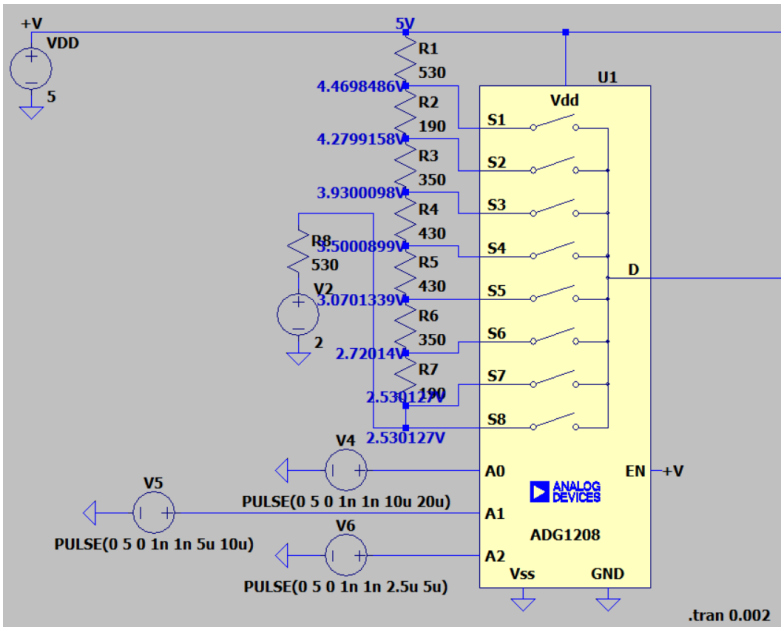

Figure 2: the MUX reference voltage levels and resistor values on the schematic.

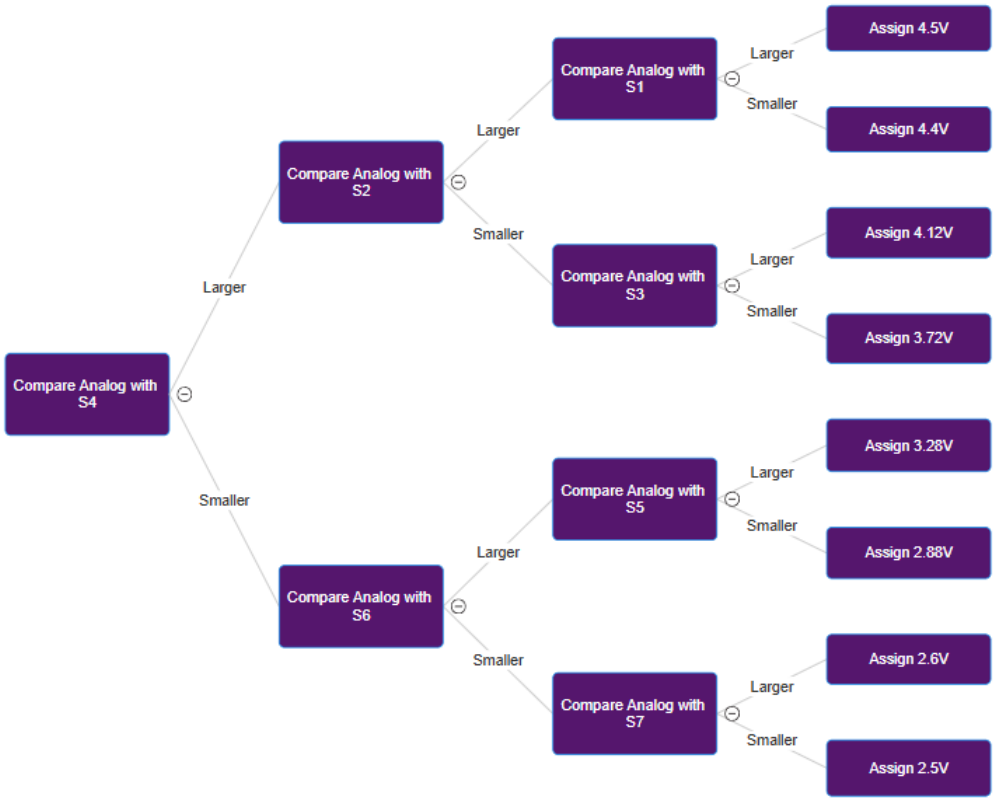

Figure 3: Decision tree for the binary search comparisons.

2.2 Sample and Hold

The sample-and-hold block samples the analog input at predefined time instants and holds the sampled value constant until the next sampling time. These sampling instants are controlled by a digital signal supplied by the Arduino.

To achieve 3-bit resolution for a 1 kHz sine wave and assigning 8 values per period, we configured V_{ctrl} with a period of 125 μs, with a 60 μs on-time. To make sure that this on-time is long enough to guarantee successful sample-and-hold, we used a small enough capacitor to make sure that it charges up fast enough, and we experimentally decided on 10nF.

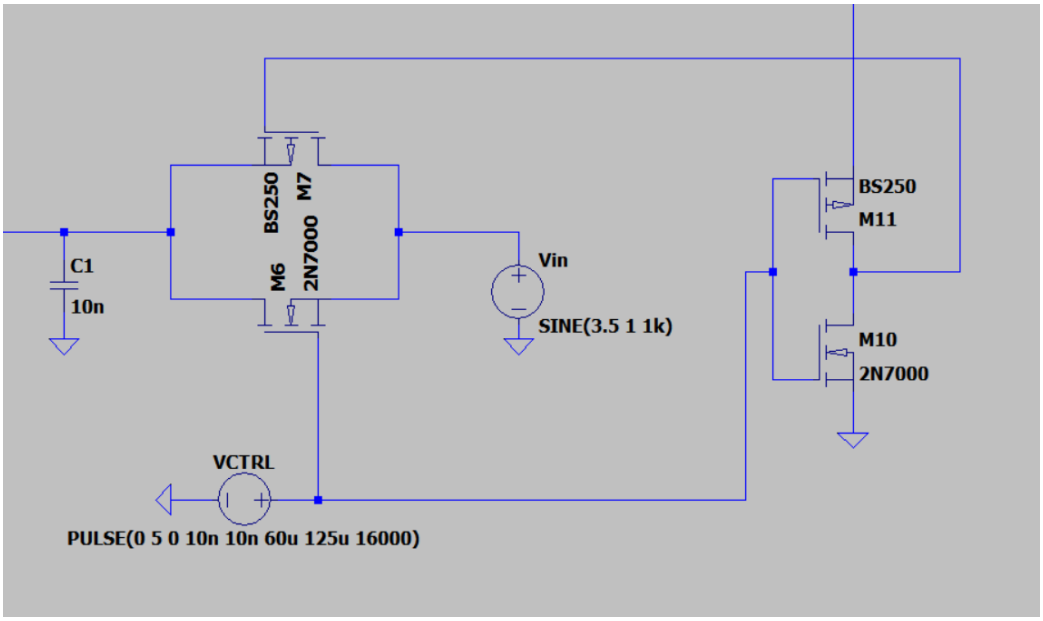

Figure 4: Sample-and-hold block.

We used a transmission gate topology to obtain low on-resistance and minimize signal distortion while keeping the block decoupled from the rest of the circuitry. The V_{ctrl} signal is directly connected to the nMOS gate and its inverted version is connected to the pMOS gate. A capacitor connected to the the comparator input stores the sampled voltage while the transmission gate is on. And after the gate turns off, the capacitor holds the sampled voltage for comparison, until the next sample-and hold cycle.

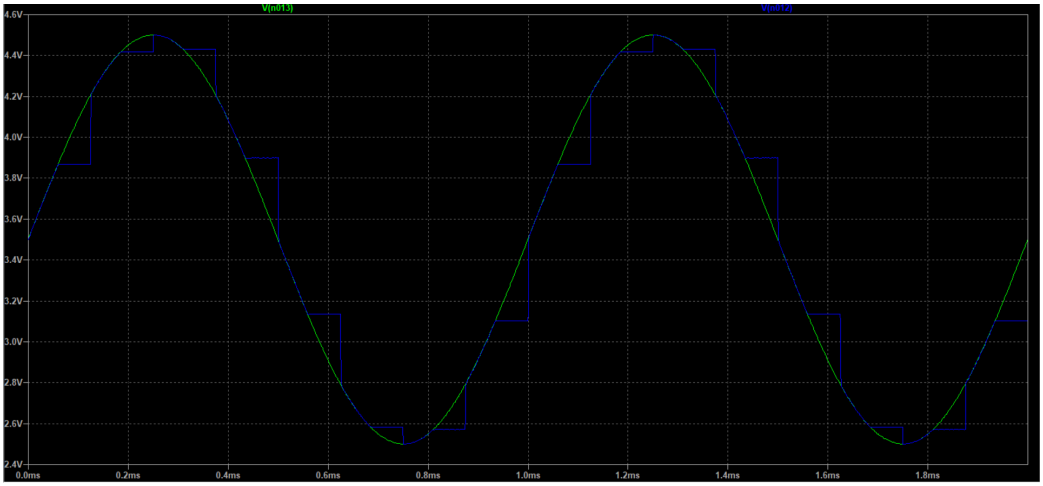

Figure 5: Sampled signal (blue) and the original sine wave (green).

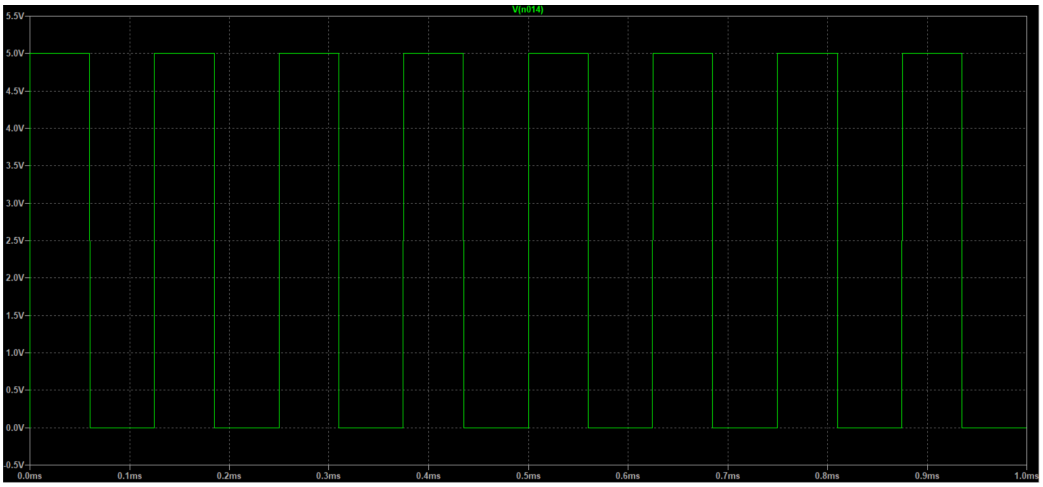

Figure 6: V_{ctrl} .

2.3 Comparator

The comparator was implemented using a MOSFET differential amplifier topology using 5 transistors. Its purpose is to sense small differences between the two inputs and channel the main current to one branch. As a result, the drain voltage on that branch is pulled close to 0 V, while the other branch

stays close to the supply voltage (5 V). To meet the project specifications, the comparator must provide a differential gain greater than 100 at 10 kHz (without the inverter stage).

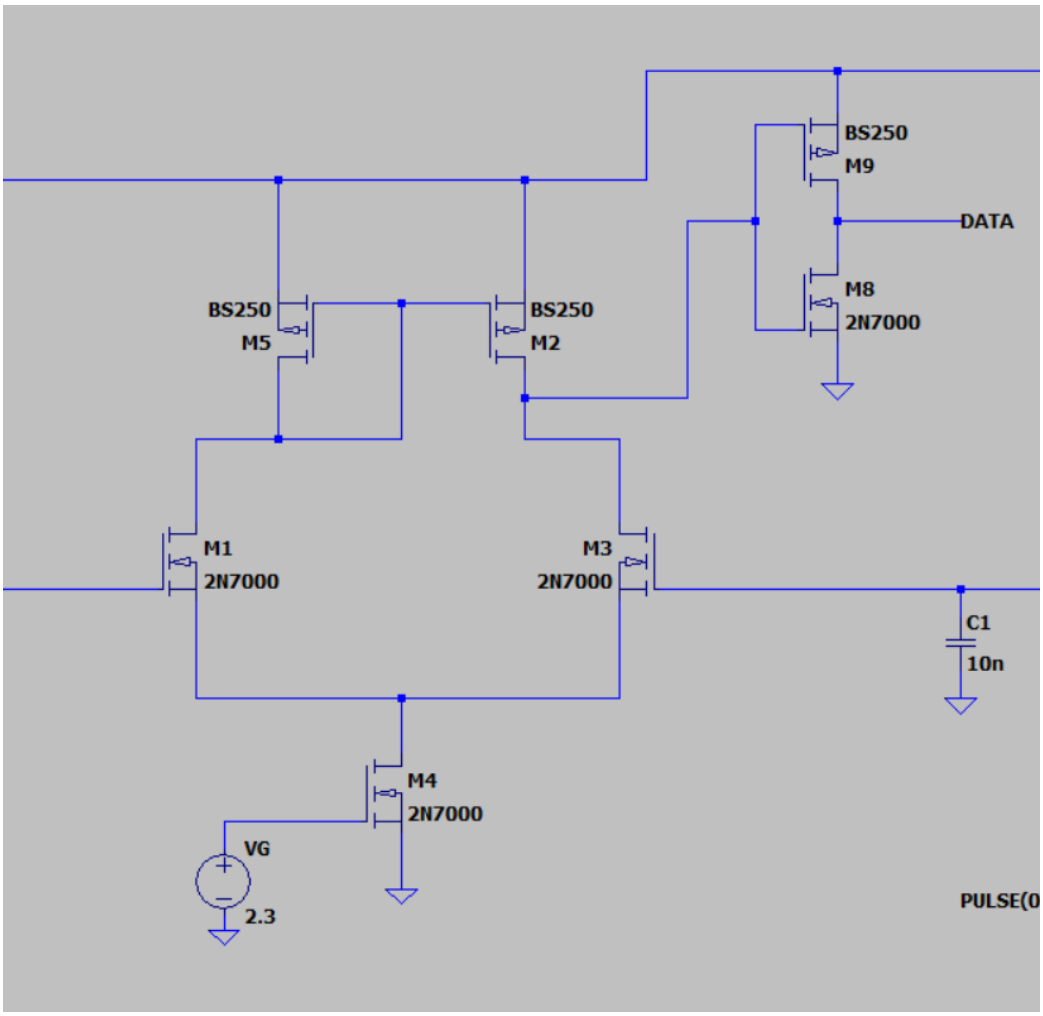

Figure 7: Comparator schematic with inverter.

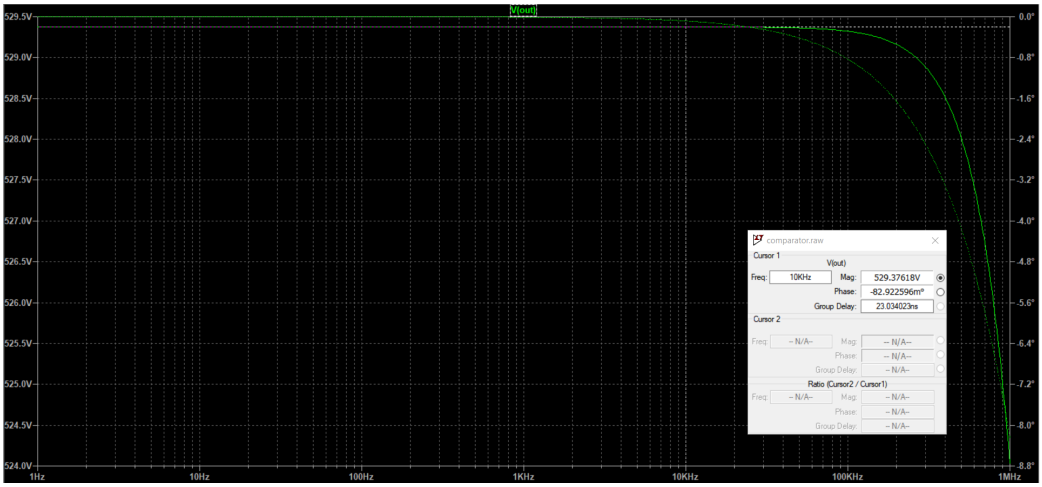

Figure 8: Linear gain vs. frequency.

As seen in Figure 8, the simulated gain is around 529, which is way above sufficient for the required operation. To directly associate a high output voltage with $V_{\text{in}} > V_{\text{ref}}$ and a low output voltage with $V_{\text{in}} < V_{\text{ref}}$, we added an inverter at the comparator output. This also sharpens the 0–5 V transition. The inverter output, D, is fed into the Arduino to perform the comparisons by reading the output D as 5 V logic levels.

3 Hardware Implementation and Results

The hardware circuit was implemented using 2N7000 nMOS transistors, BS250 pMOS transistors, a 4051 8-channel analog multiplexer, an Arduino Nano microcontroller board, resistors, capacitors, breadboards, and jumper cables. As seen in Figure 9, the breadboard on the left contains the MUX with resistor ladder, while the one on the right contains the sample and hold, comparator, and output inverter.

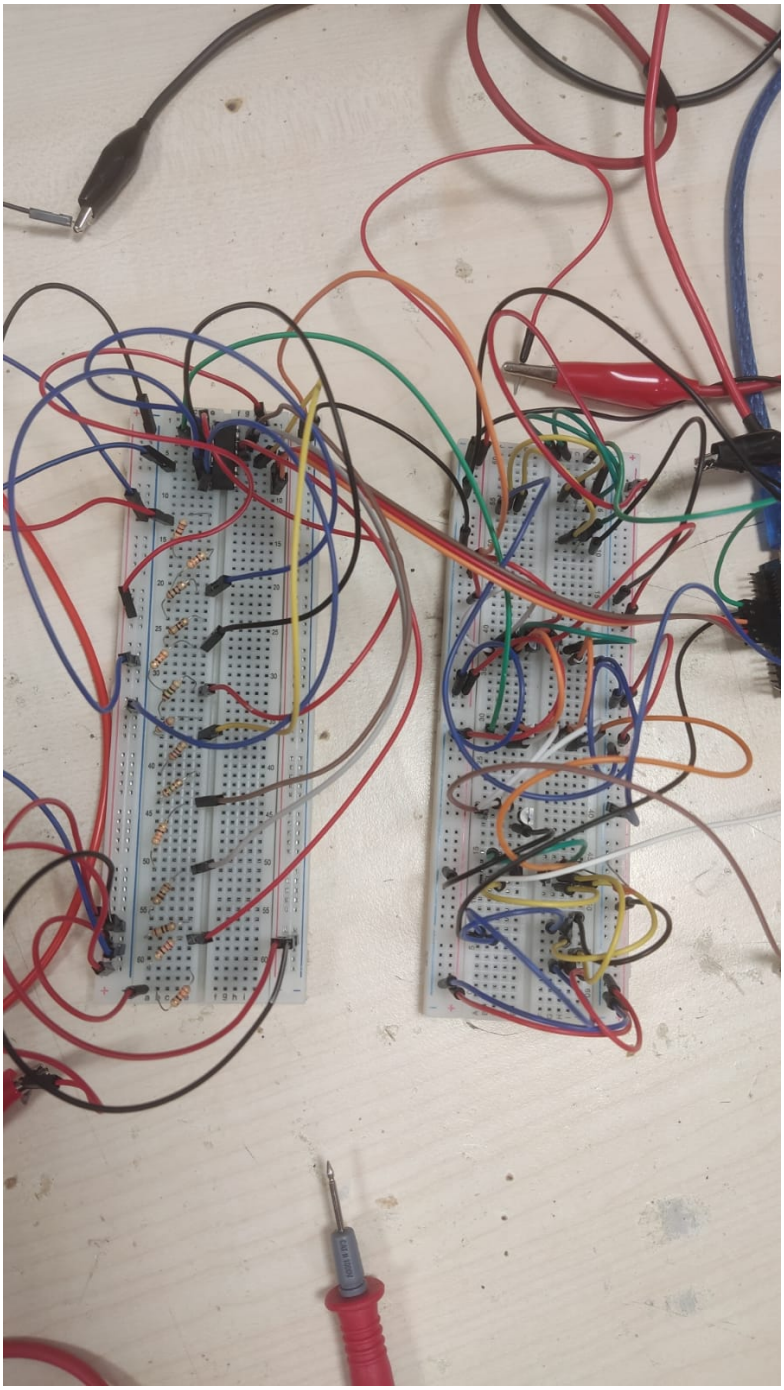

Figure 9: Final circuit hardware demonstration on breadboards.

3.1 MUX and Resistor Ladder

We connected out multiplexer to the breadboard together with the ladder with the determined resistor values. Together with the mux and resistor ladder on breadboard, the measured resistor ladder voltages fed into the analog multiplexer are shown below. We observed an approximately 0.05 V error in the reference voltages, which of course, did not prevent correct overall operation.

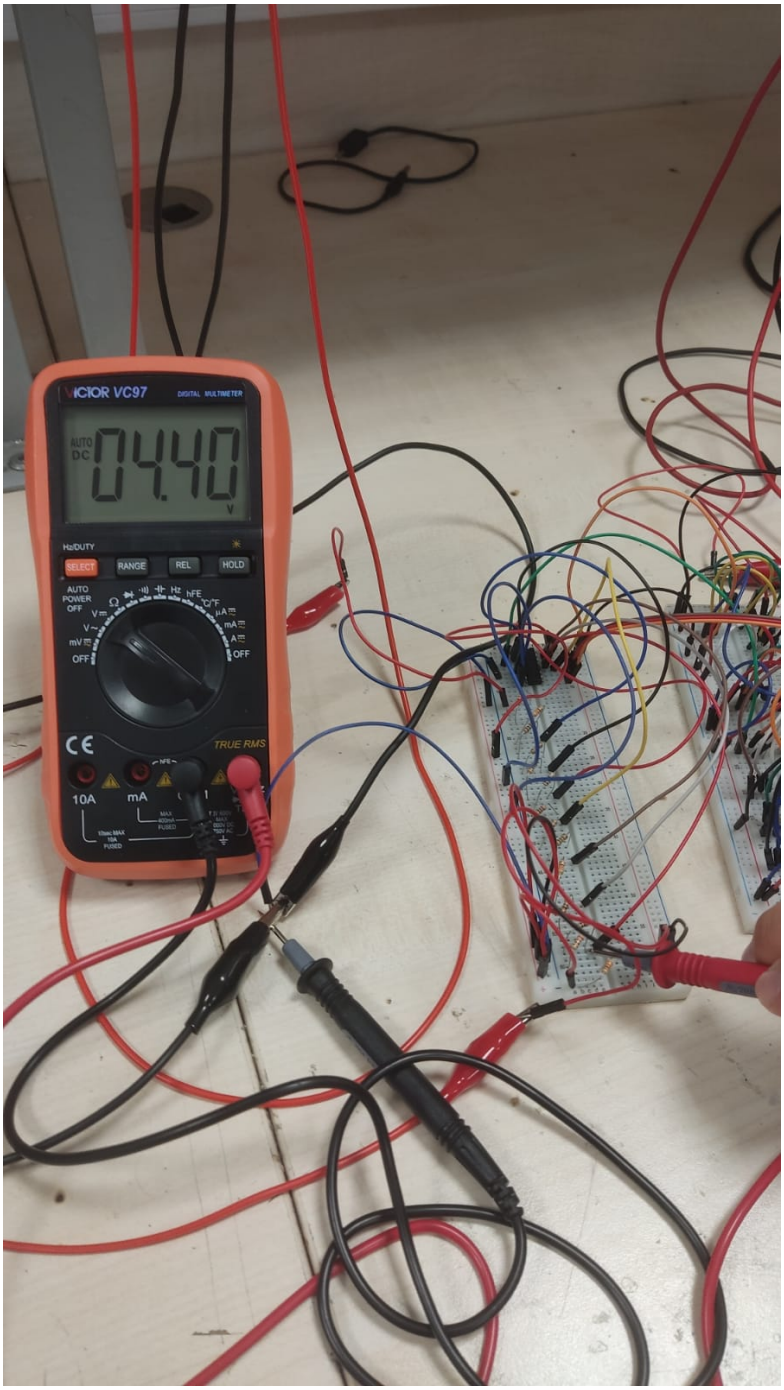

Figure 10: S1 voltage.

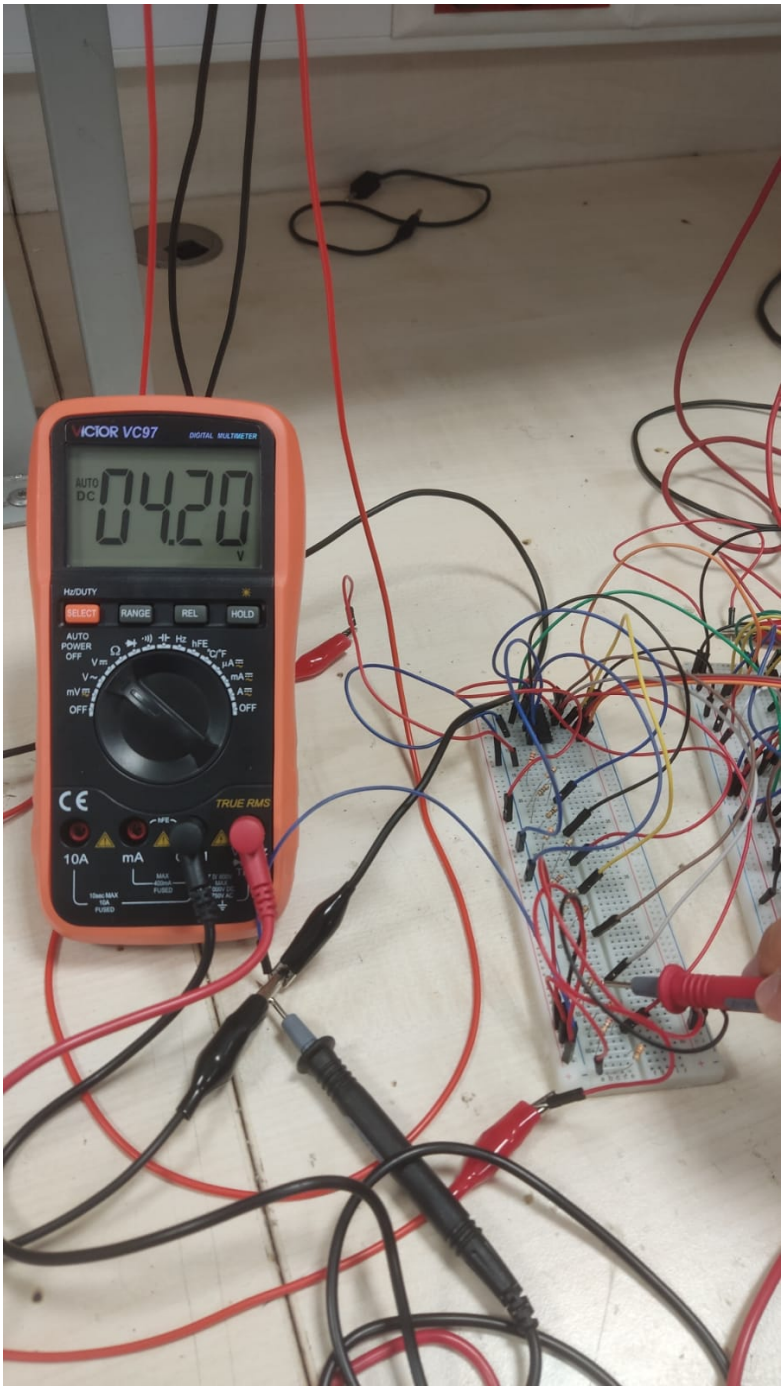

Figure 11: S2 voltage.

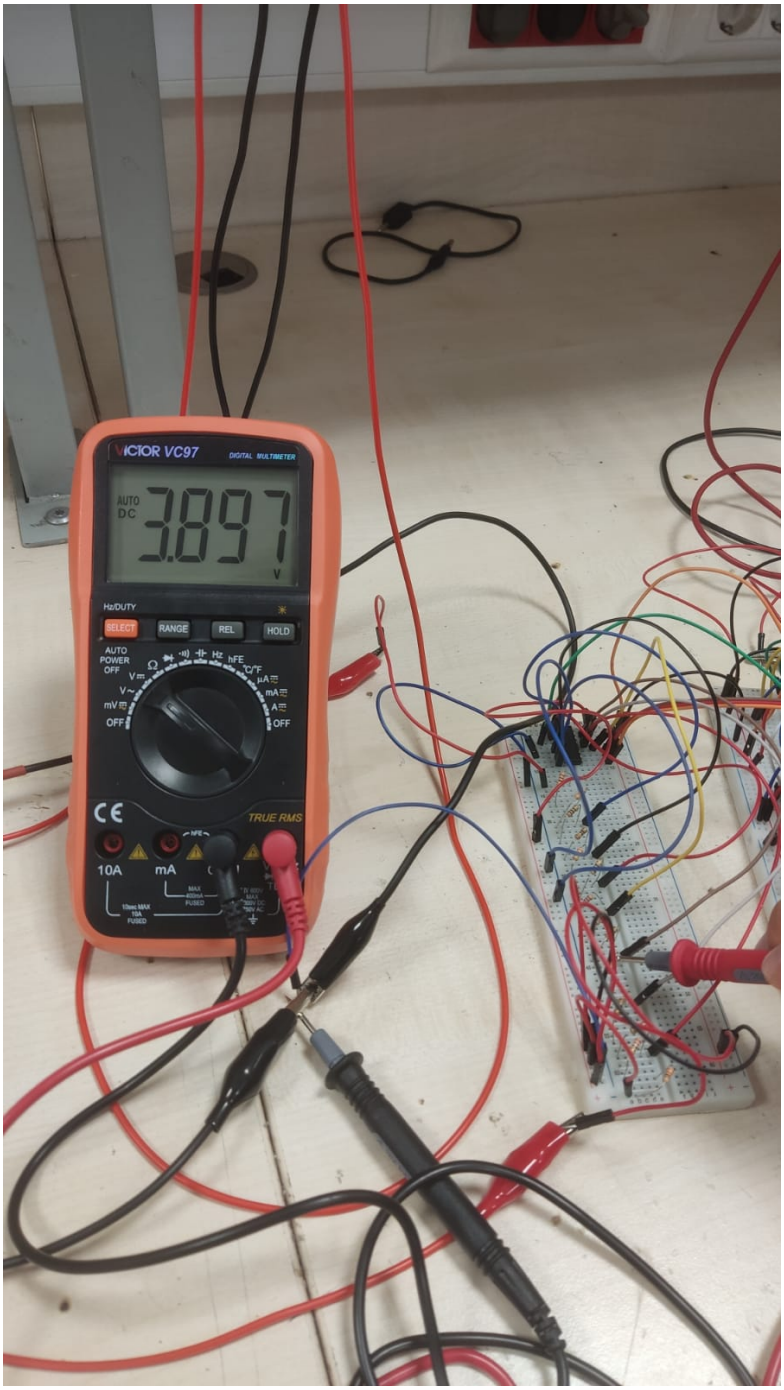

Figure 12: S3 voltage.

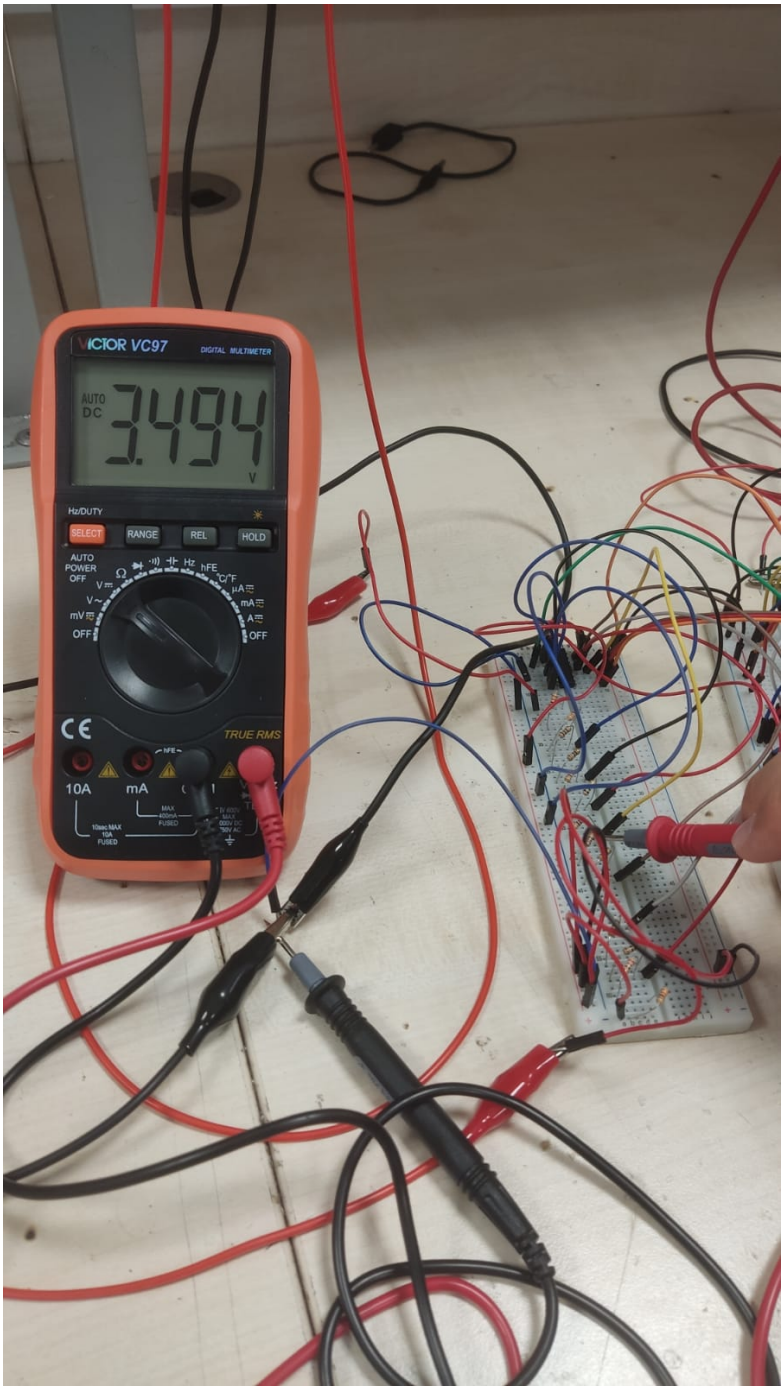

Figure 13: S4 voltage.

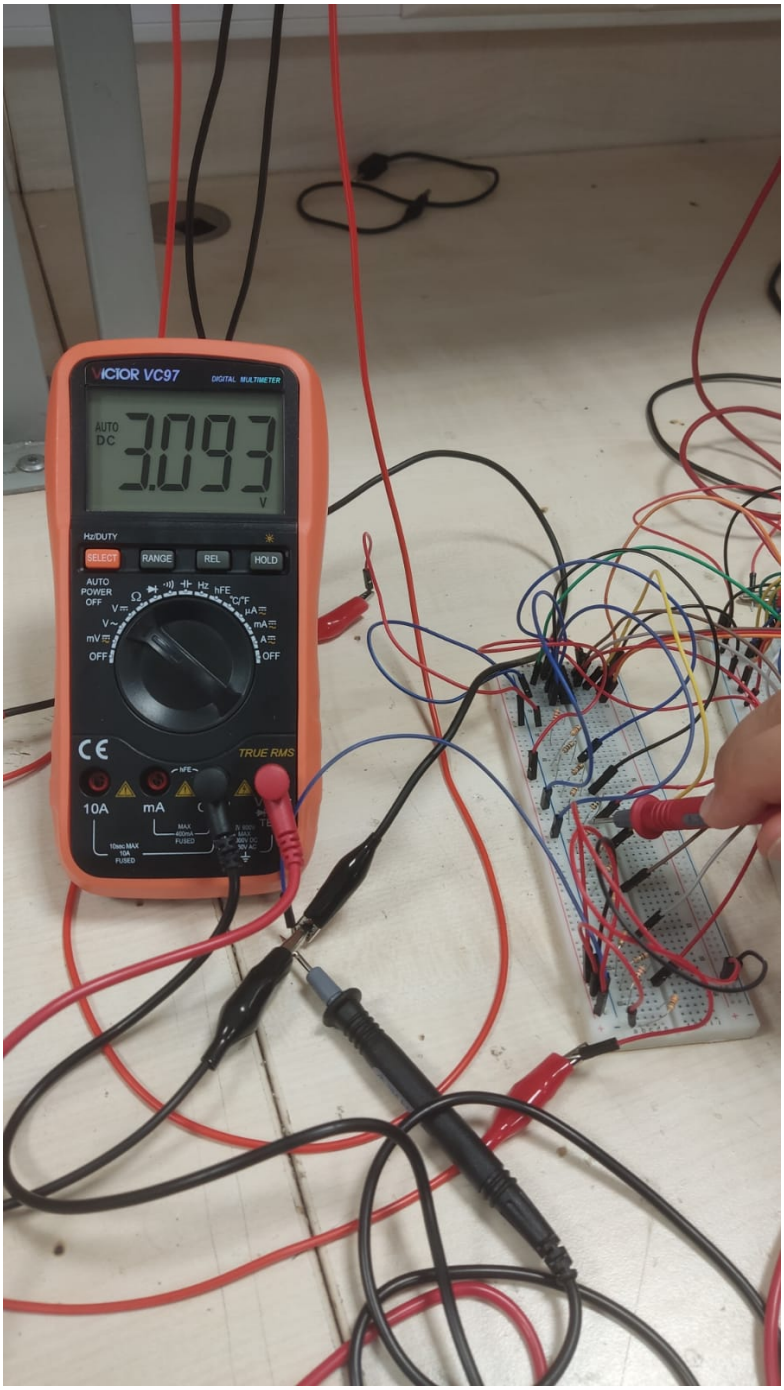

Figure 14: S5 voltage.

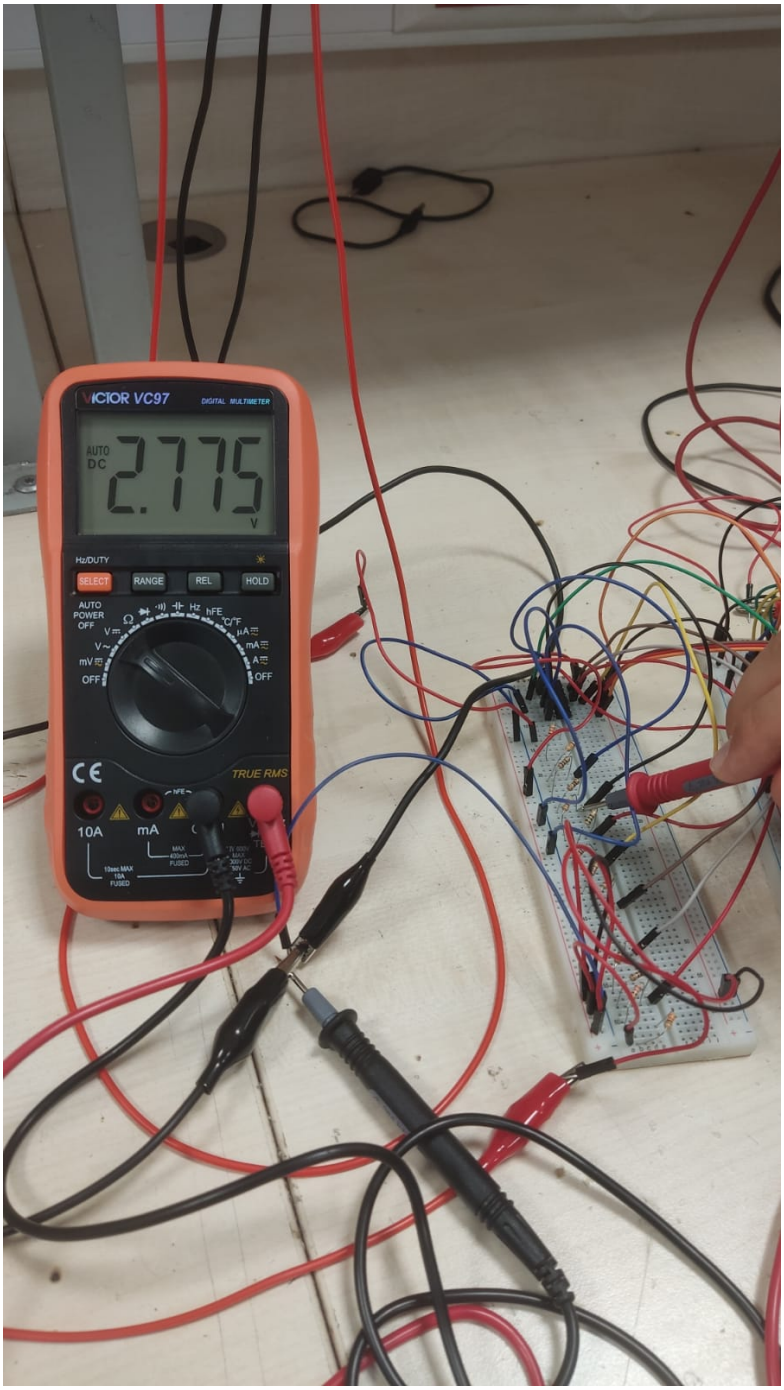

Figure 15: S6 voltage.

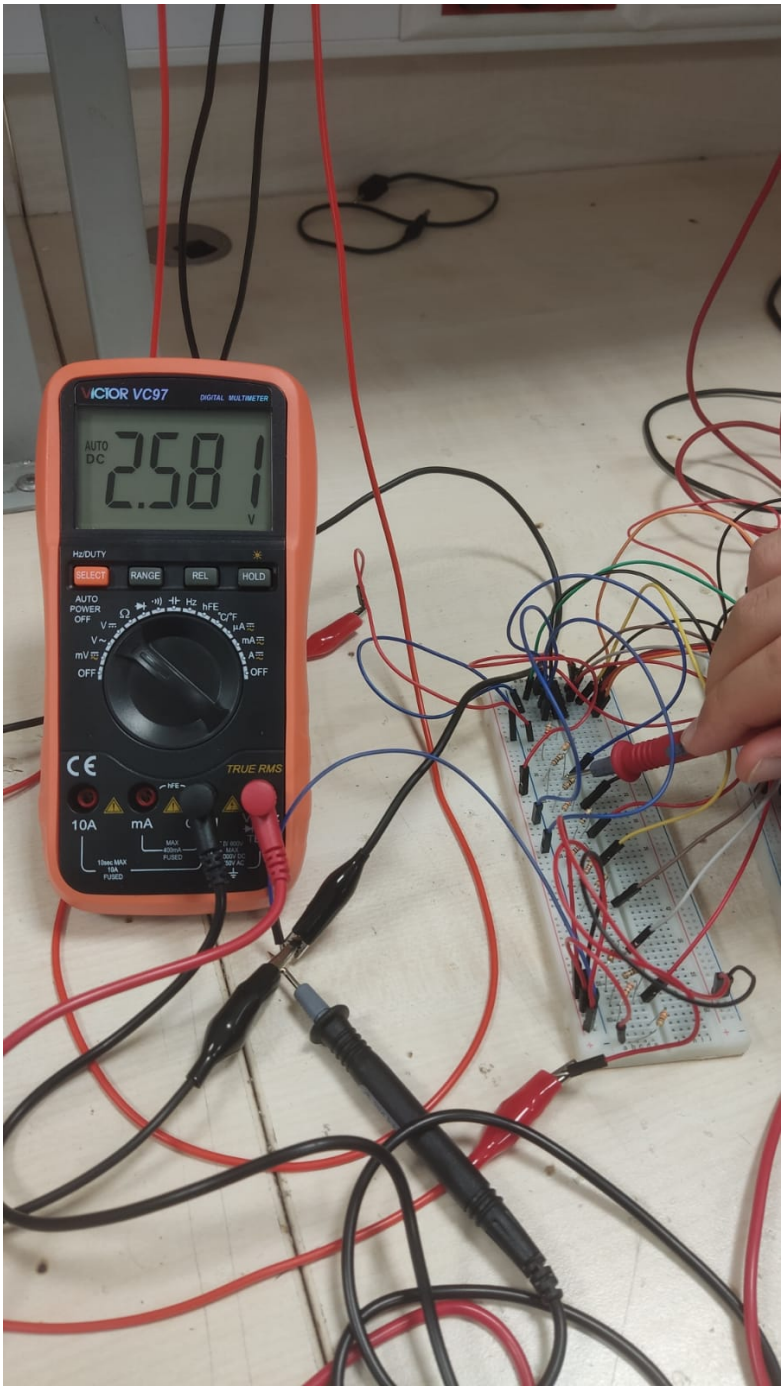

Figure 16: S7 voltage.

The MUX control signals A0, A1, and A2 are provided by the Arduino board. The waveforms of these three control bits are shown in Fig. 17.

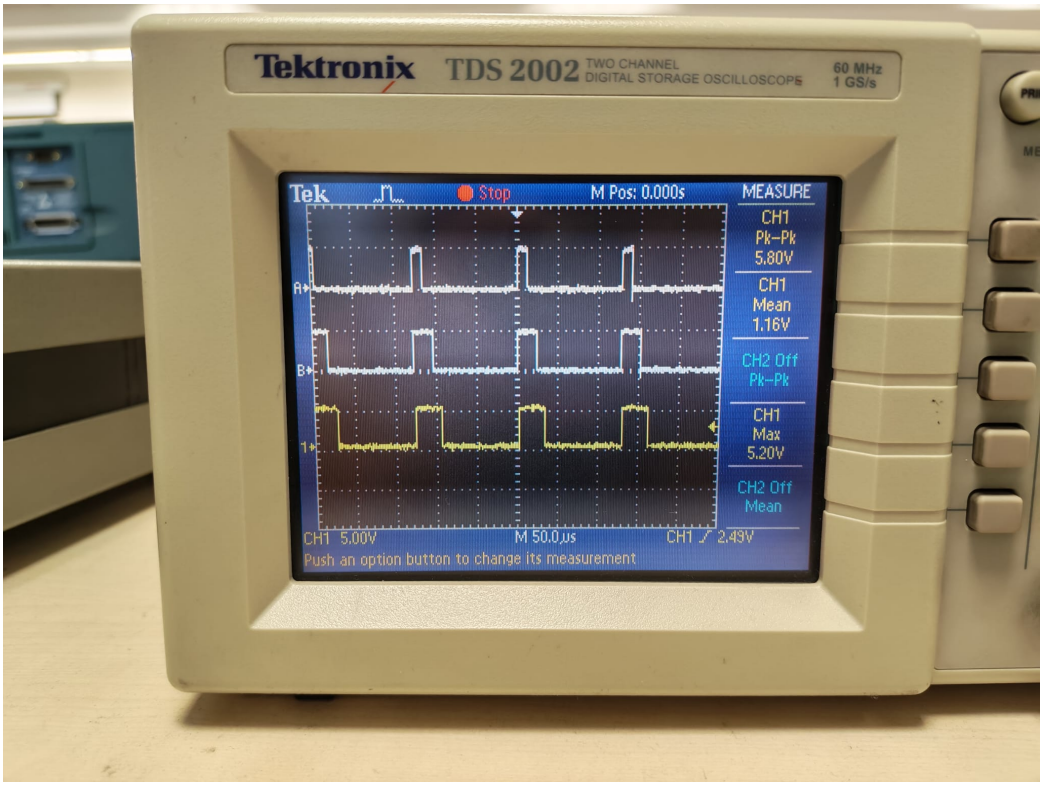

Figure 17: Control bits observed on the oscilloscope, ordered MSB to LSB (top to bottom).

3.2 Sample and Hold

The 2-tansistor transmission gate and the 10nF sample and hold capacitor, together with the 2-transistor inverter was successfully connected to the breadboard. The waveforms shown in Figure 18 show that the sample-and-hold circuit behaves as intended.

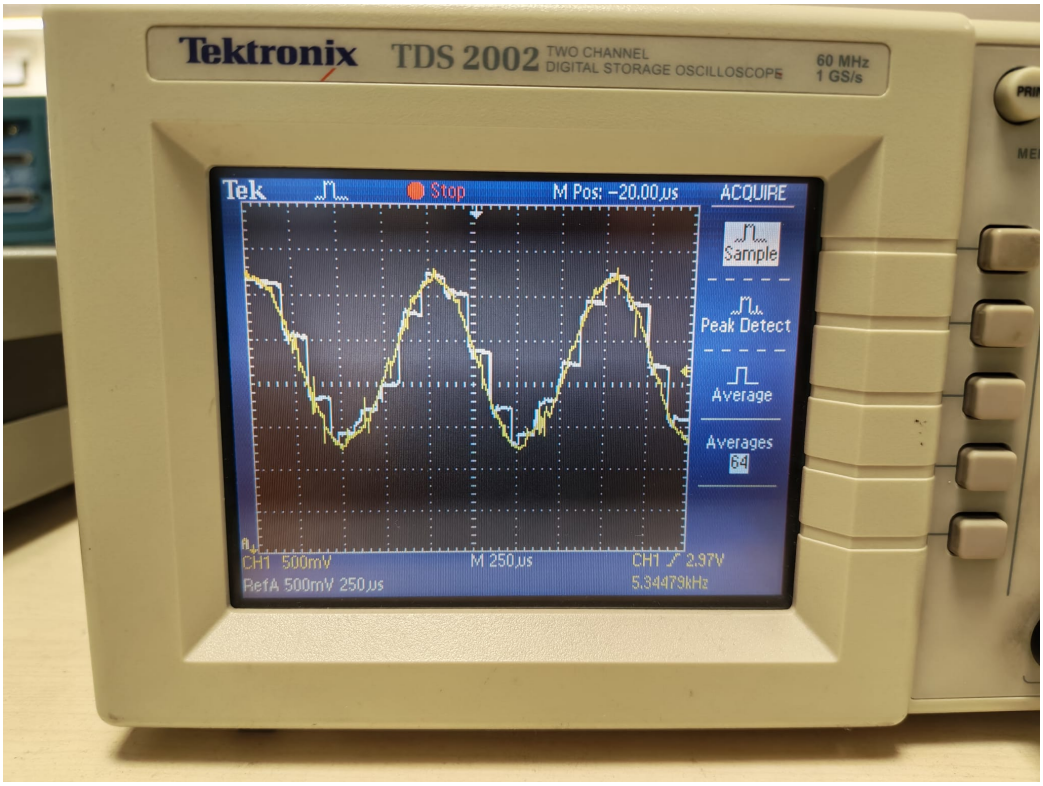

Figure 18: Sample-and-hold operation on hardware.

The V_{ctrl} signal that times the sample-and-hold and the 1 kHz input sine wave to be sampled (from the signal generator) are respectively shown in Fig. 19 and Fig. 20.

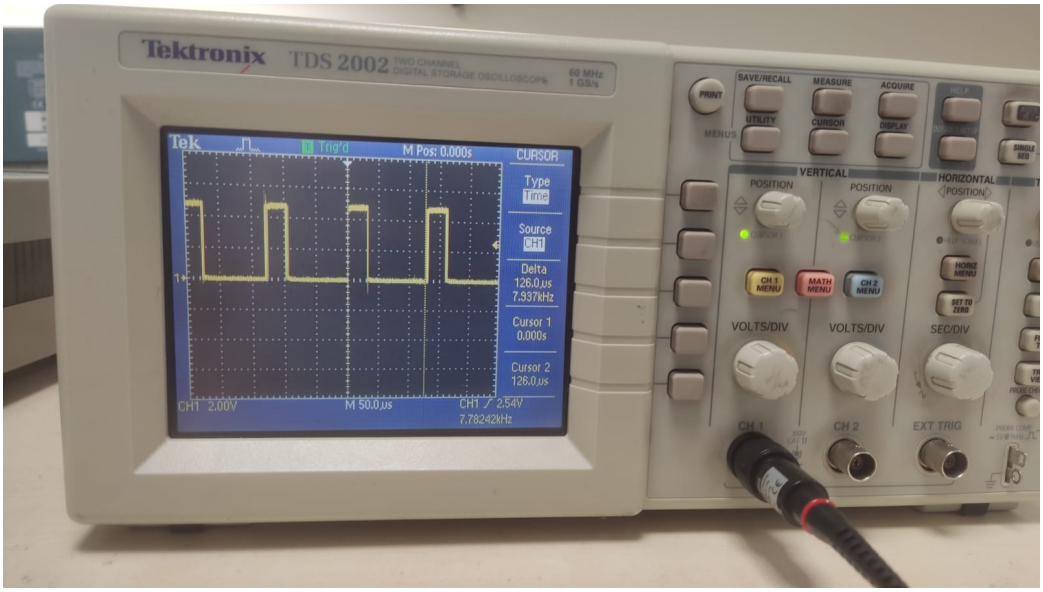

Figure 19: V_{ctrl} signal.

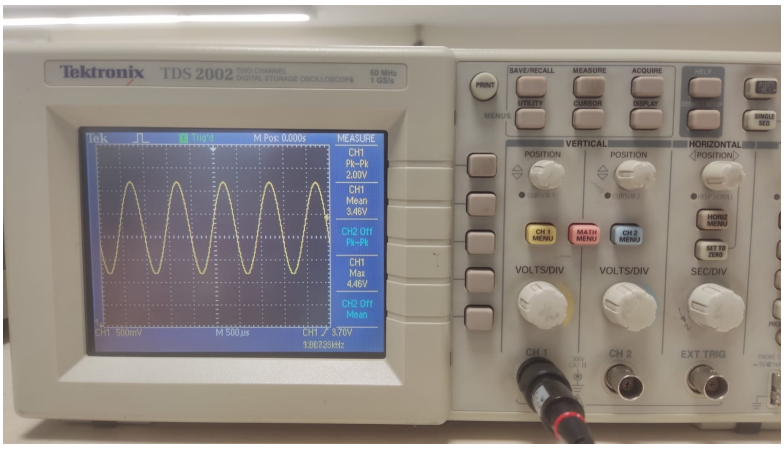

Figure 20: Input sine wave supplied by the signal generator.

3.3 Comparator

With 5V Vdd and a 2.3V Vg, we implemented the comparator circuit and the 2-transistor inverter at the output onto the breadboard. As discussed in the sample-and-hold section, observing signals at this speed on the oscilloscope

can be challenging. In addition, since the overall inverted comparator output D is a series of 1's and 0's, and it is only meaningful after it is processed by the Arduino board, the oscilloscope waveform of the signal D is not much helpful in terms of determining successful operation. However, we provide the oscilloscope reading of the signal in Figure 21, as it is still important in terms of observing the said 0s and 1s behavior, which means there is comparison being done. And of course, we confirmed total correct behavior using the overall analog-to-digital conversion and the final sampling result plotted, which is seen in Figure 22.

As seen in Figure 22, the overall system successfully samples the sinusoid by assigning 8 points per period with 8 total voltage levels, while preserving the correct sinusoidal shape as required.

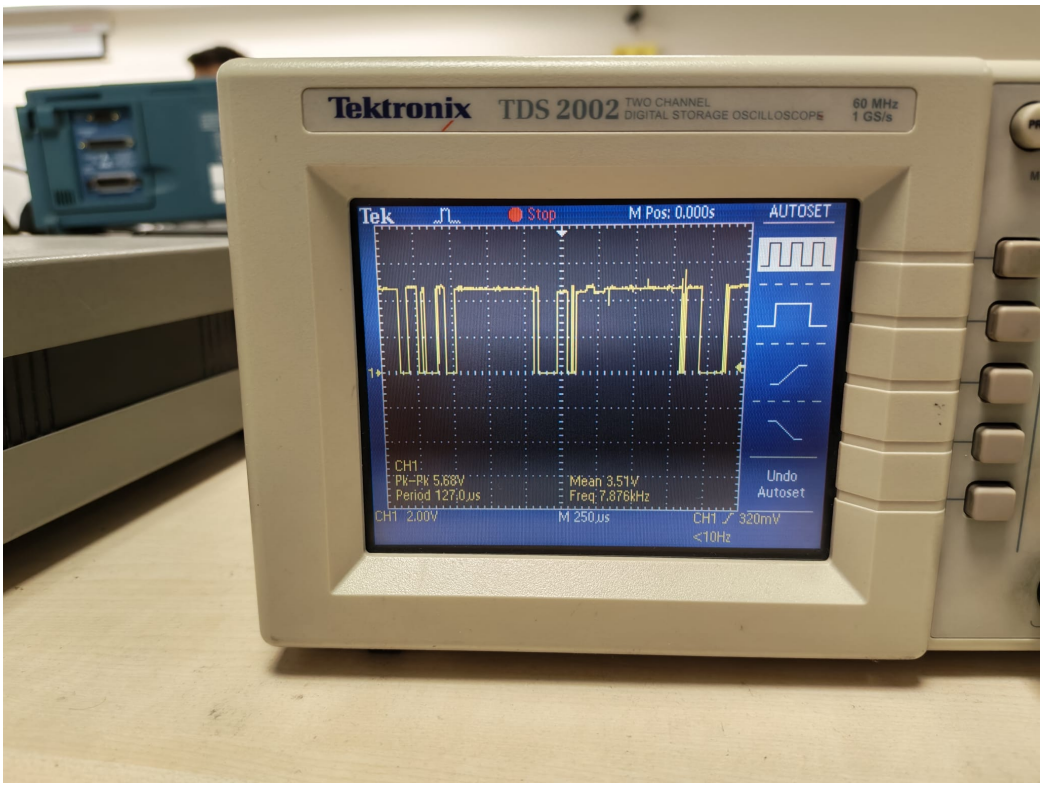

Figure 21: Raw output D of the comparator + inverter.

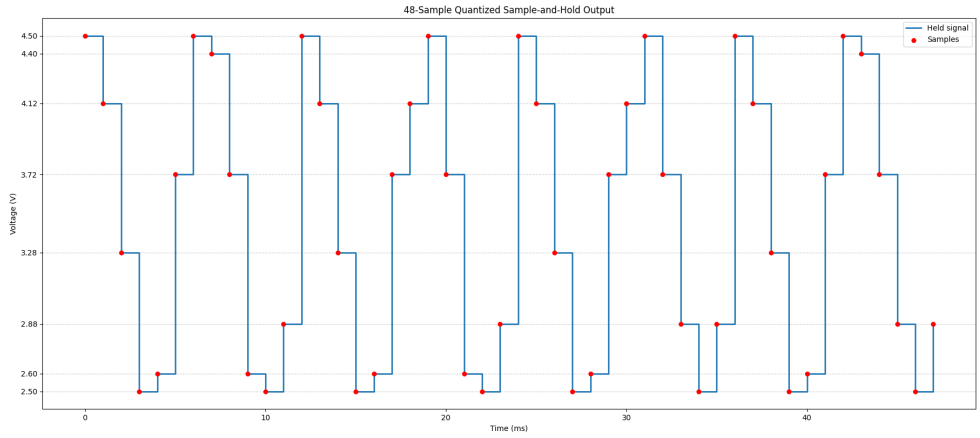

Figure 22: Plot of the digitized (quantized) waveform.

4 Arduino Operation and Plotting

4.1 C++ Code for The Arduino Operation

Here is the C++ code that was deployed to the Arduino micro-controller, to generate V_{ctrl}, drive the MUX select pins (A0–A2), perform comparisons, assign a voltage level, and store the sampling result in an array. After 48 cycles, the array of assigned voltage levels is printed to the serial monitor.

```
#include <Arduino.h>

// Pin Definitions
const uint8_t PIN_VCTRL  = 7;  // D7 = PD7 -> S/H switch gate
const uint8_t PIN_MUX_S0 = 4;  // D4 = PD4 -> CD4051 pin 11 (A, LSB)
const uint8_t PIN_MUX_S1 = 5;  // D5 = PD5 -> CD4051 pin 10 (B)
const uint8_t PIN_MUX_S2 = 6;  // D6 = PD6 -> CD4051 pin 9  (C, MSB)
const uint8_t PIN_D      = 8;  // D8 = PB0 -> comparator digital output

// Mux control codes (C B A) - Now: S2 S1 S0
const byte MUX_S1 = 0b000;  // 4.46V (S2=0, S1=0, S0=0)
const byte MUX_S2 = 0b001;  // 4.27V (S2=0, S1=0, S0=1)
```

```
const byte MUX_S3 = 0b010;  // 3.93V (S2=0, S1=1, S0=0)
const byte MUX_S4 = 0b011;  // 3.50V (S2=0, S1=1, S0=1)
const byte MUX_S5 = 0b100;  // 3.07V (S2=1, S1=0, S0=0)
const byte MUX_S6 = 0b101;  // 2.72V (S2=1, S1=0, S0=1)
const byte MUX_S7 = 0b110;  // 2.53V (S2=1, S1=1, S0=0)
const byte MUX_S8 = 0b111;  // 2.53V (S2=1, S1=1, S0=1) (not used)

// Output Voltage Levels
const float V4_5  = 4.50f;
const float V4_4  = 4.40f;
const float V4_12 = 4.12f;
const float V3_72 = 3.72f;
const float V3_28 = 3.28f;
const float V2_88 = 2.88f;
const float V2_6  = 2.60f;
const float V2_5  = 2.50f;

// Array to store 48 results
const int ARRAY_SIZE = 48;
float results[ARRAY_SIZE];
int n = 0;  // Counter for cycles

// ---------- FAST IO ----------
// VCTRL (D7 -> PD7)
#define VCTRL_HIGH()   (PORTD |=  _BV(PD7))
#define VCTRL_LOW()    (PORTD &= ~_BV(PD7))

// MUX pins: S0=D4(PD4), S1=D5(PD5), S2=D6(PD6)
static inline void setMuxFast(byte muxCode) {
  // muxCode bit0 -> S0 -> PD4
  // muxCode bit1 -> S1 -> PD5
  // muxCode bit2 -> S2 -> PD6
  uint8_t out =
      (((muxCode >> 0) & 0x01) << PD4) |
      (((muxCode >> 1) & 0x01) << PD5) |
      (((muxCode >> 2) & 0x01) << PD6);

  // Clear PD4..PD6, then write
```

```
  PORTD = (PORTD & ~(_BV(PD4) | _BV(PD5) | _BV(PD6))) | out;
}

// Comparator input D8 -> PB0
static inline bool readComparatorFast() {
  return (PINB & _BV(PB0)) != 0;
}

// ---------- Timer1 tick delay ----------
// Timer1 prescaler /8 => 16MHz/8 = 2MHz => 0.5 µs per tick
static inline void timer1_init_free_run() {
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  TCCR1B |= (1 << CS11); // /8
}

static inline void wait_ticks(uint16_t ticks) {
  uint16_t start = TCNT1;
  while ((uint16_t)(TCNT1 - start) < ticks) {
    // busy wait
  }
}

// µs -> ticks (0.5 µs per tick)
#define TICKS_30US   60
#define TICKS_5US    10
#define TICKS_3US     6
#define TICKS_2_5US   5

void setup() {
  // DDR setup for fast IO
  DDRD |= _BV(PD7);  // VCTRL output
  DDRD |= _BV(PD4) | _BV(PD5) | _BV(PD6); // MUX pins outputs
  DDRB &= ~_BV(PB0); // Comparator input

  VCTRL_LOW();
  setMuxFast(0);
```

```
  timer1_init_free_run();

  Serial.begin(115200);
  while (!Serial) { ; }

  Serial.println("Starting ADC measurement. Will collect 48 samples...");
}

void printResultsArray() {
  Serial.println("\n=== RESULTS ARRAY (48 samples) ===");
  Serial.println("Index,Voltage(V)");

  for (int i = 0; i < ARRAY_SIZE; i++) {
    Serial.print(i);
    Serial.print(",");
    Serial.println(results[i], 2);
  }
}

void loop() {
  if (n < ARRAY_SIZE) {
    // 1) VCTRL HIGH for 30µs (sample)
    VCTRL_HIGH();
    wait_ticks(TICKS_30US);

    // 2) VCTRL LOW (hold)
    VCTRL_LOW();

    // 3) settle
    wait_ticks(TICKS_2_5US);

    // Start with S4 (3.50V)
    setMuxFast(MUX_S4);
    wait_ticks(TICKS_5US);

    if (readComparatorFast()) { // Vin > S4
      wait_ticks(TICKS_3US);
```

```
    setMuxFast(MUX_S2);  // 4.27V
    wait_ticks(TICKS_5US);

    if (readComparatorFast()) { // Vin > S2
      wait_ticks(TICKS_3US);
      setMuxFast(MUX_S1);  // 4.46V
      wait_ticks(TICKS_5US);

      if (readComparatorFast()) {
        results[n] = V4_5;
      } else {
        results[n] = V4_4;
      }

    } else { // Vin < S2
      wait_ticks(TICKS_3US);
      setMuxFast(MUX_S3);  // 3.93V
      wait_ticks(TICKS_5US);

      if (readComparatorFast()) {
        results[n] = V4_12;
      } else {
        results[n] = V3_72;
      }
    }

  } else { // Vin < S4
    wait_ticks(TICKS_3US);
    setMuxFast(MUX_S6);  // 2.72V
    wait_ticks(TICKS_5US);

    if (readComparatorFast()) { // Vin > S6
      wait_ticks(TICKS_3US);
      setMuxFast(MUX_S5);  // 3.07V
      wait_ticks(TICKS_5US);

      if (readComparatorFast()) {
        results[n] = V3_28;
```

```
        } else {
          results[n] = V2_88;
        }

      } else { // Vin < S6
        wait_ticks(TICKS_3US);
        setMuxFast(MUX_S7);  // 2.53V
        wait_ticks(TICKS_5US);

        if (readComparatorFast()) {
          results[n] = V2_6;
        } else {
          results[n] = V2_5;
        }
      }
    }

    n++;

    if (n == ARRAY_SIZE) {
      printResultsArray();
    }

  } else {
    delay(1000);
  }
}
```

4.2 Python Code for Plotting

Here is the Python code used to format the array obtained from Arduino serial monitor and plot it.

```
import numpy as np
import matplotlib.pyplot as plt

raw = """
0,4.50
```

```
1,4.12
2,3.28
3,2.50
4,2.60
5,3.72
6,4.50
7,4.40
8,3.72
9,2.60
10,2.50
11,2.88
12,4.50
13,4.12
14,3.28
15,2.50
16,2.60
17,3.72
18,4.12
19,4.50
20,3.72
21,2.60
22,2.50
23,2.88
24,4.50
25,4.12
26,3.28
27,2.50
28,2.60
29,3.72
30,4.12
31,4.50
32,3.72
33,2.88
34,2.50
35,2.88
36,4.50
37,4.12
38,3.28
```

```
39,2.50
40,2.60
41,3.72
42,4.50
43,4.40
44,3.72
45,2.88
46,2.50
47,2.88
""".strip()

idx, v = [], []
for line in raw.splitlines():
    i, val = line.split(",")
    idx.append(int(i))
    v.append(float(val))

idx = np.array(idx)
v = np.array(v)

fs = 1000.0
Ts = 1 / fs
t = idx * Ts * 1000  # ms

levels = [2.50, 2.60, 2.88, 3.28, 3.72, 4.12, 4.40, 4.50]

plt.figure(figsize=(9, 4))
plt.step(t, v, where="post", linewidth=2, label="Held signal")
plt.scatter(t, v, s=30, zorder=3, label="Samples")
plt.yticks(levels)
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (V)")
plt.title("48-Sample Quantized Sample-and-Hold Output")
plt.legend()
plt.tight_layout()
plt.show()
```

5 Conclusion

In this project, a functional 3-bit ADC was designed, simulated, and implemented using discrete components and an Arduino-based control system. LTSpice simulations verified correct operation of the design of the sample-and-hold block and confirmed that the comparator gain requirement was satisfied. Then, measurements on the hardware implementation also demonstrated correct MUX reference generation, sampling behavior, and digital output generation. Finally, the ADC successfully quantized the input waveform into eight voltage levels at the required sampling rate.

References

[1] P. R. Gray, P. J. Hurst, S. H. Lewis, and R. G. Meyer, *Analysis and Design of Analog Integrated Circuits*, 5th ed., Wiley, 2009.