**Project Report 5**

Compilation Overview of the Project till now:

1. Initially, we constructed an 8\*1 Digital multiplexer using the 74151 IC and provided it with a sine wave input in simulation.
2. Our next step involved linking this multiplexer to a demultiplexer to compare their outputs using an oscilloscope.
3. Surprisingly, the observed output displayed random high and low stems, deviating from our intended result.
4. Further investigation led us to distinguish between two distinct MUX types: Analog and Digital.
5. Subsequently, we replaced the Digital MUX (74151 IC) with an Analog MUX (4051 IC).
6. To validate the consistency between the Mux and Demux outputs, we connected the Analog Mux to the Demux and monitored the signals via an oscilloscope.
7. This time, the output aligned with our expectations, displaying a sample and hold pattern.
8. Regarding the observed delay increase at higher frequencies:

* Inherent characteristics of the multiplexer, such as gate and propagation delays, might become more prominent with increased frequency. This can result in an overall delay escalation, evident in the introduced propagation delay at 400kHz.
* It's also plausible that the discrepancy in observed results could partially stem from potential issues with simulation accuracy in Proteus.

**Hardware Implementation:**

The 74151N, an 8-channel multiplexer with 3 select lines, was used and controlled via the ESP32 utilizing its timer registers through code written in the Arduino IDE.

Upon comprehensive testing, the multiplexer functioned in accordance with its datasheet specifications, accurately responding to **digital** High and Low inputs and producing the expected outputs at the specified timings.

A graph with green lines

Description automatically generated

A graph with green and white lines

Description automatically generated

A graph with green and white lines

Description automatically generated

//Code for the timer register

const int ledPins[] = {16, 5, 18, 19};

const int numLeds = sizeof(ledPins) / sizeof(ledPins[0]);

hw\_timer\_t \*timer = NULL;

volatile int patternIndex = 0;

void IRAM\_ATTR onTimer() {

  for (int j = 0; j < numLeds; j++) {

    digitalWrite(ledPins[j], (patternIndex >> (numLeds - 1 - j)) & 1);

  }

  patternIndex = (patternIndex + 1) % (1 << numLeds);

}

void setup() {

  for (int i = 0; i < numLeds; i++) {

    pinMode(ledPins[i], OUTPUT);

  }

  // Setup and attach the timer

  timer = timerBegin(0, 80, true); // Timer 0, prescaler 80, count up

  timerAttachInterrupt(timer, &onTimer, true);

  timerAlarmWrite(timer, 500000, true); // Set the timer interval to 500,000 microseconds (0.5 seconds)

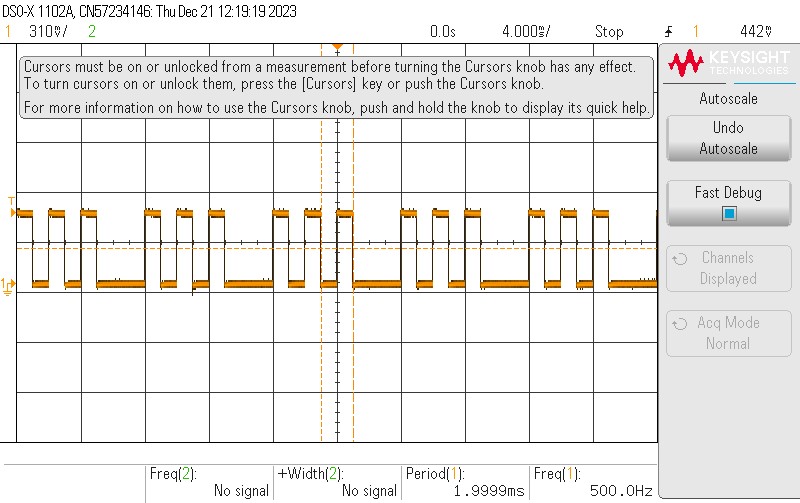
  timerAlarmEnable(timer);

}

void loop() {

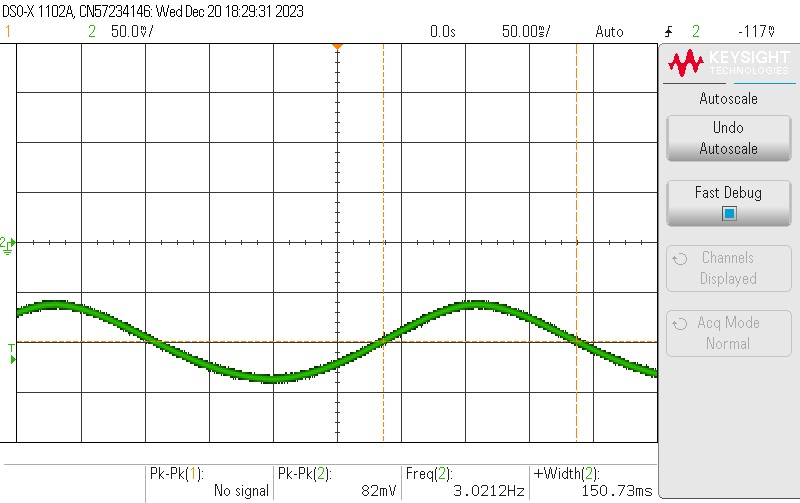
  // Your main loop code, if needed

}



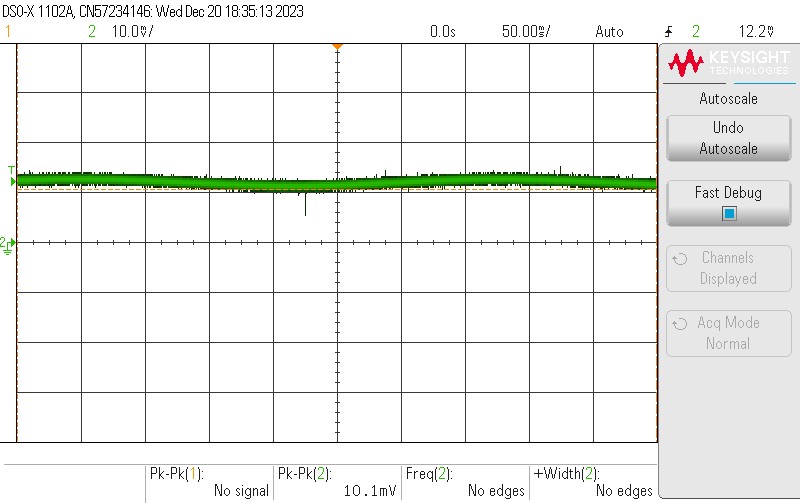
When a digital input with a sampling frequency of 500Hz is supplied, setting I1, I2, I3, I5, and I7 as LOW while keeping the rest HIGH consistently yields the expected output.

This outcome remains consistent across various input combinations.

**Analog signals to the multiplexer:**

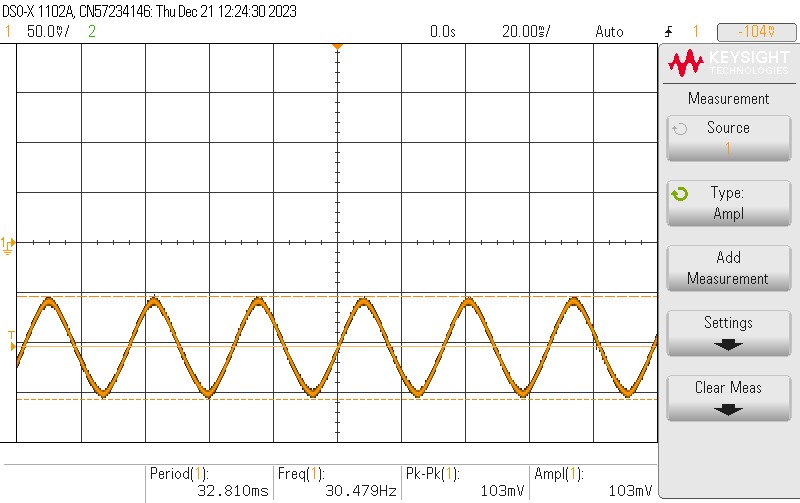
On providing a sinusoidal input of frequency 3Hz and voltage of 4V with rest of the inputs as 0.

With Vcc 5.5 V from a dc source.



The following result is obtained.

This is **not** what we are supposed to get as the output.

This case continued for many observations:

Sine wave generated from the function generator of 30Hz frequency.

A graph with a line graph

Description automatically generated with medium confidenceSine wave observed on the input pin of 74151N IC.

Output at Y:

A graph with a line going up

Description automatically generated

**Inference**

The multiplexer used isn't designed for sampling analog signals, it lacks the necessary functionality for this task.

The snapshots are obtained from the DSO using a USB connection.

**Next Step**

The CD4051 multiplexer enables the sampling of analog signals, making it well-suited for this purpose based on its datasheet.

• Utilize the CD4051 IC to sample analog inputs through 8 channels initially, then expand to 16 channels, and finally progress to 32 channels.

• Connect the sampled output to an ESP32 for ADC conversion, allowing visualization via the serial monitor or software.

• Visualize the channel-to-channel time propagation, sampling, and any attenuated losses within the circuit.