Physics 4050H: Project 1

Jeremy Favro (0805980), Melody Berhane ()
Department of Physics & Astronomy
Trent University, Peterborough, ON, Canada

October 1, 2025

Todo list

Abstract

We successfully automated PHYS-2250H's lab 4 on operational amplifiers through microcontroller operated wave generation and input element switching, however due to time constraints did not automate data collection. Results are similar to those that could be obtained by "manually" completing the lab with most inaccuracy owing to additional resistance in the switching element(s) .

Any other areas?

1 Introduction

The goal of this project was to take a lab from PHYS-2250H (Electronics) and improve upon/automate the content of that lab. We opted to automate lab 4 which involved constructing different configurations of operational amplifiers and verifying that the theoretical equations derived in lecture corresponded to the real-world behaviour of the device. Lab 4 specifically looked at varying the input resistance of an inverting amplifier and looked at a single configuration of differentiating amplifier. Our initial concept for automation was to go the most direct route and use a microcontroller to control a switch which would allow selecting different components as in the case of both the inverting and differentiating amplifiers the only component which is varied between measurements in the element lying between the signal source and inverting input of the amplifier. We additionally sought to eliminate as many components external to those which sit directly on a breadboard and so created a function generator to replace the one used in the original lab.

2 High Level Overview

The automation consists of two key segments, one which generates a fixed frequency and peak-to-peak voltage input and one which controls which element is connected to the inverting input of the amplifier. There is technically a third segment of the device which provides a stable 5 volts supply for the various integrated circuits but due to its simplicity it is left off the following diagrams.

Whole circuit block diagram here

2.1 Wave Generator

The input signal generator uses a single ATMmega328P [2] (through hole, 28-PDIP) microcontroller connected through SPI to an MCP4822[1] digital-to-analog converter.

2.2 Amplifier Control & Measurement

3 Detailed Methods

3.1 Wave Generator

The ATmega328P (hereon referred to as a 328P) runs the code seen in 6.3. The core operating principal of this uses something known as a phase accumulator. A phase accumulator is convenient as it allows easy control of the frequency of the output wave. At the beginning of the program we calculate a period over which the phase accumulator operates and populate a lookup table with precalculated values of our desired output wave, sin in this case. Then, for every iteration of the loop, we first check if sufficient time has passed that we can update the output of the DAC. If not, we continue the loop. If sufficient time has passed, we first write out the value in our lookup table which corresponds to the current "phase value". We then increment the phase modulo the size of our table, then increment the next update time by our precalculated period. Writing a value to the DAC involves, for the MCP4822, sending a 16-bit integer via SPI. The contents of this integer, are, ordered from most to least significant bits, 4 command bits whose meanings are shown in 6.3 for our specific case and explained more generally in the datasheet[1], then the 12-bit value to be written to the output of the DAC. Dependent on command bits, this value may be a fraction of either 4096 (twice the internal voltage reference of the MCP4822) or 2048. As our application requires a 1V peak-to-peak wave, we opted for

the latter. The 12-bit value is then represented at the output of the DAC as the given fraction of the internal voltage reference. As it becomes difficult to write with sufficient speed to produce a high-fidelity wave directly from the DAC when high-frequencies are desired we only operate with 16 possible states for the output of the DAC. Because we desire a smooth wave for the input of the op-amp we need to feed the stepped output that this low sample count results in through a suitable reconstruction filter. Here this will be a filter which rejects the sudden changes of the stepped wave and makes them more gradual, e.g. a low-pass filter. The output of the DAC is already fed through a 1 kΩ current limiting resistor so we opted to include a 1 µF capacitor to ground in parallel with this output resistor, giving a time constant of 1 ms which, though not exactly the same as our sampling frequency, gives a sufficiently smoothed wave and was chosen based on on-hand components. For the final step of this first stage of the whole device we feed the signal through an emitter-follower amplifier to pass the loading created by the second stage from the DAC to an op-amp which fixes the output voltage at the voltage generated by the DAC.

Block diagram

3.2 Amplifier Control

The amplifier control circuit is much simpler than the frequency generator and is comprised of a single 328P which is dedicated to switching four of its digital output pins between high and low in sequence. These pins are connected to the control pins of an SN74HC4066N[?] whose input pins are chained together and connected to the output of stage 1 (the emitter-follower output). Each output pin of the switch is then connected to a 220 Ω, 1 kΩ, 3.6 kΩ, 9.1 kΩ resistor, each of which are in turn connected to the inverting input of our final op-amp. We then have the inverting input connected to the output through a 10 kΩ resistor (and the non-inverting input connected to ground) to create a variable inverting amplifier. The code which controls the switching via the 328P is available here6.2.

4 Conclusion

4.1 Results

4.2 Areas for Improvement

5 References

References

[1] Microchip Technology. *8/10/12-Bit Dual Voltage Output Digital-to-Analog Converter with Internal VREF and SPI Interface*, 49 2015. Rev. B.

[2] Microchip Technology. *ATmega48A/PA/88A/PA/168A/PA/328/P*, 9 2020. Rev. B.

6 Appendix

6.1 Full Schematic

6.2 Switch Control Code

```
const uint32_t SW_PERIOD_US = 2e3;  // 2s
int cur_pin = 2;

void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  Serial.begin(9600);
}

void loop() {
    digitalWrite(cur_pin, LOW);

    cur_pin++;
    if (cur_pin > 5) {
      cur_pin = 2;
    }
    switch (cur_pin) {
      case 2:
        Serial.print("220Ω");
        Serial.print("\n");
        break;
      case 3:
        Serial.print("1kΩ");
        Serial.print("\n");
        break;
      case 4:
        Serial.print("3.6kΩ");
        Serial.print("\n");
        break;
      case 5:
        Serial.print("9.1kΩ");
        Serial.print("\n");
        break;
      default:
        break;
    }
    digitalWrite(cur_pin, HIGH);
    delay(SW_PERIOD_US);
}
```

6.3 Frequency Generator Code

```
#include <SPI.h>
#define SAMPLES 16
#define FREQUENCY 1000
#define CS_PIN 10
const uint32_t PERIOD = 1e6 / (FREQUENCY * SAMPLES);
uint32_t next = 0;

uint16_t phase = 0;

uint16_t SIN_LUT[SAMPLES] = {};

void setup()
{
  // Populate lookup table @ ~1V_pp
  for (int k = 0; k < SAMPLES; k++)
  {
    SIN_LUT[k] = ((2048 / 2) * (sin(2 * PI * (float)k / SAMPLES) + 1));
  }

  SPI.begin();

  // Chip select pin active low, this will deselect the DAC
  digitalWrite(CS_PIN, HIGH);
}

void loop()
{
  uint32_t now = micros();

  if (now > next)
  {
    writeDAC(SIN_LUT[phase]);

    phase = (phase + 1) % SAMPLES;
    next += PERIOD;
  }
}

void writeDAC(uint16_t value)
{
  uint16_t command = 0x3000; // 0011...,
  // 0 -> Write Channel A
  // 0 -> DC
  // 1 -> V_o=V_r * value/4096
  // 1 -> Output shutdown, operating on HI
  // See datasheet for more

  // OR command w/ masked value, gives
  // 0011 CMD
  command |= (value & 0x0FFF);

  digitalWrite(CS_PIN, LOW);

  SPI.transfer16(command);

  digitalWrite(CS_PIN, HIGH);
}
```