CPE 325: Embedded Systems Laboratory

Lab10

Analog-to-Digital Converter and Digital-to-Analog Converter

Submitted by: Esther Shore

Date of Experiment: July 19, 2023

Report Deadline: July 25, 2023

Demonstration Deadline: August 2, 2023

**Introduction**

This lab involves configuring and utilizing the 12-bit analog-to-digital converter (ADC12) and digital-to-analog converter (DAC) on the MSP430. The first program involves interfacing the ADC with the external analog inputs of the ADXL335 3-dimensional accelerometer. The second program involves using the DAC to generate waveforms on an oscilloscope.

**Theory Topics**

1. Accelerometers

Accelerometers are sensors used to measure forces of acceleration. This is useful for detecting movement. It can be used to monitor orientation or tilt as well. The MSP430 is able to interface with an external accelerometer sensor. These sensors tend to output analog values and can be converted using an ADC.

1. ADC and DAC

An ADC is an analog to digital converter. It is capable of converting an analog signal to digital data to be used in a circuit like on the MSP430. It is typically used to enable reading sensor data from external peripherals that provide analog output signals such as accelerometers. The number of bits represents the resolution, or accuracy of the conversion, which also applies to the DAC.

A DAC is a digital to analog converter. These are used to convert digital data into an analog signal. For instance, data points can be sent through a DAC to produce a continuous signal output, viewable through an oscilloscope probed at the specific pin.

**Program 1 (Accelerometer Interfacing + Bubble Level)**

***Program Description:***

This program involves interfacing with the external ADXL335 accelerometer peripheral to receive and display acceleration data in the Serial App. This is accomplished by implementing an analog to digital converter in order to convert the raw analog inputs from the sensor to digital outputs in units that are better understood. This C program samples x, y, and z axes acceleration 5 times per second in terms of g, the gravitational acceleration of Earth and sends the samples over UART to display graphically in the Serial App.

The timer A count was configures to sample at 5Hz as follows:

TACCR0 = 6554; // 6554 / 32768 Hz = 0.2s = 5Hz

The acceleration for each axis is displayed as a separate line and the configuration is shown below.

The next problem is an extension of this program so that the accelerometer could be used as a sort of bubble level. The home position is assumed to be the straight orientation where the z-axis points upwards. The angular deviation from that home position along the x-axis in degrees is then determined and sent as another stream of data in the Serial App. When the deviation is positive and greater than 15 degrees, LED1 turns ON and LED2 is set to OFF, and when the deviation is negative and less than -15 degrees, LED2 turns ON and LED1 is set to OFF. The default state when the deviation is between -15 degrees and 15 degrees sets both LEDs OFF.

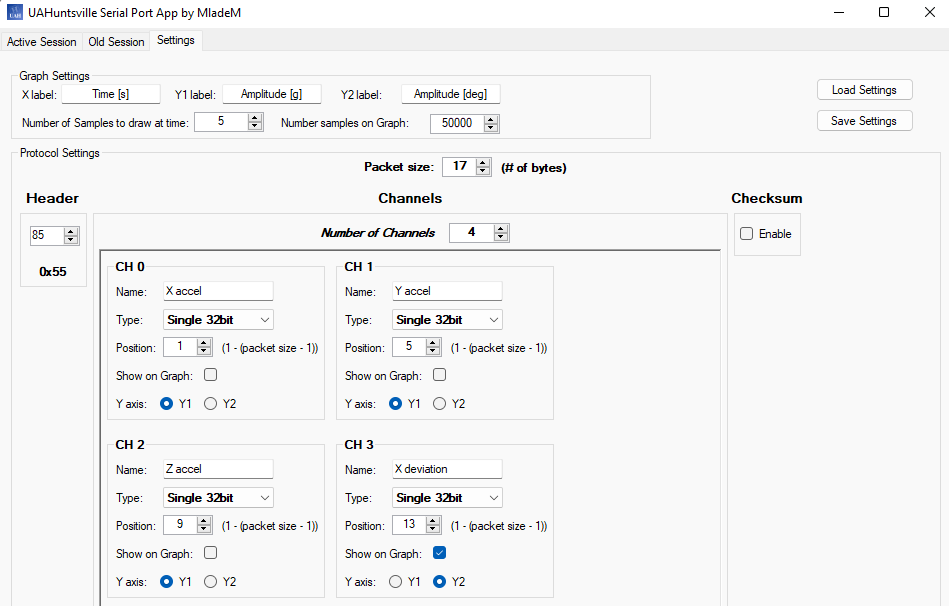
For each acceleration value, the analog value is converted to a digital output reading based on the specified reference voltage of 3V. Then, the acceleration in terms of g is calculated using the typical values for zero output and sensitivity for the sensor from the datasheet.

acceleration (g) = output reading (V) - zero output (V) / sensitivity (V/g)

ex. Xaccel = (ADCXval\*3.0/4095 - 1.5) / 0.3;

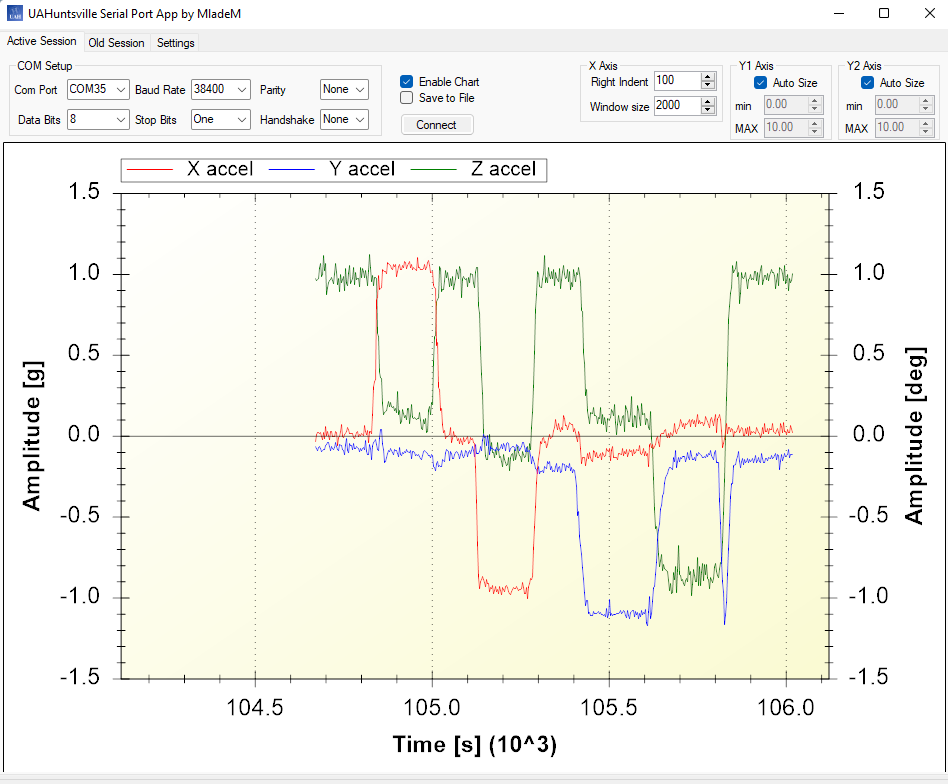
Equation for X-deviation Angle:

Serial App Configuration

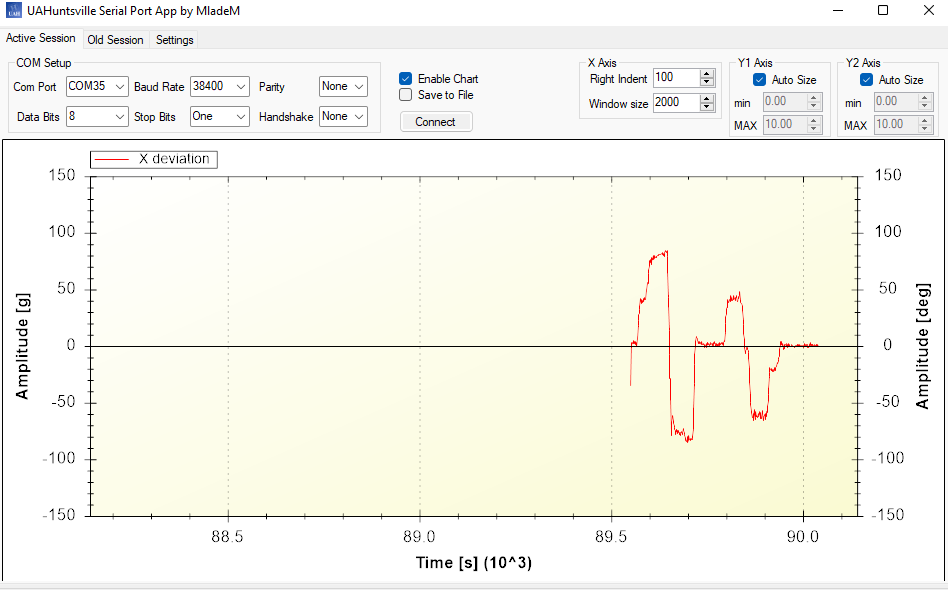


***Program Output:***

Accelerometer Data in Serial App



X-Deviation Angle in Serial App



**Program 2 (Wave Outputs)**

***Program Description:***

This C program produces different waveforms using the DAC. When no switches are held, a sine wave is generated. When Switch 1 is pressed, a triangular wave of the same frequency but half the amplitude is produced. And when Switch 3 is held, the frequency of the current waveform is halved.

The sine waveform is produced using a 512-element lookup table (LUT). Using the provided MATLAB script, changed x increment to 2\*pi/512 in order to generate a lookup table with 512 entries.

The chosen peak-to-peak amplitude for the sinusoidal waves was decided to be 2.5V based on the selected 2.5V reference voltage. Thus, the peak-to-peak amplitude of the triangular waves would be half that or 1.25V. The initial frequency of either waveform was set to 30Hz by calculating the timer count to each data point. The triangle wave was generated in the microcontroller on the fly without using an array to store it in memory.

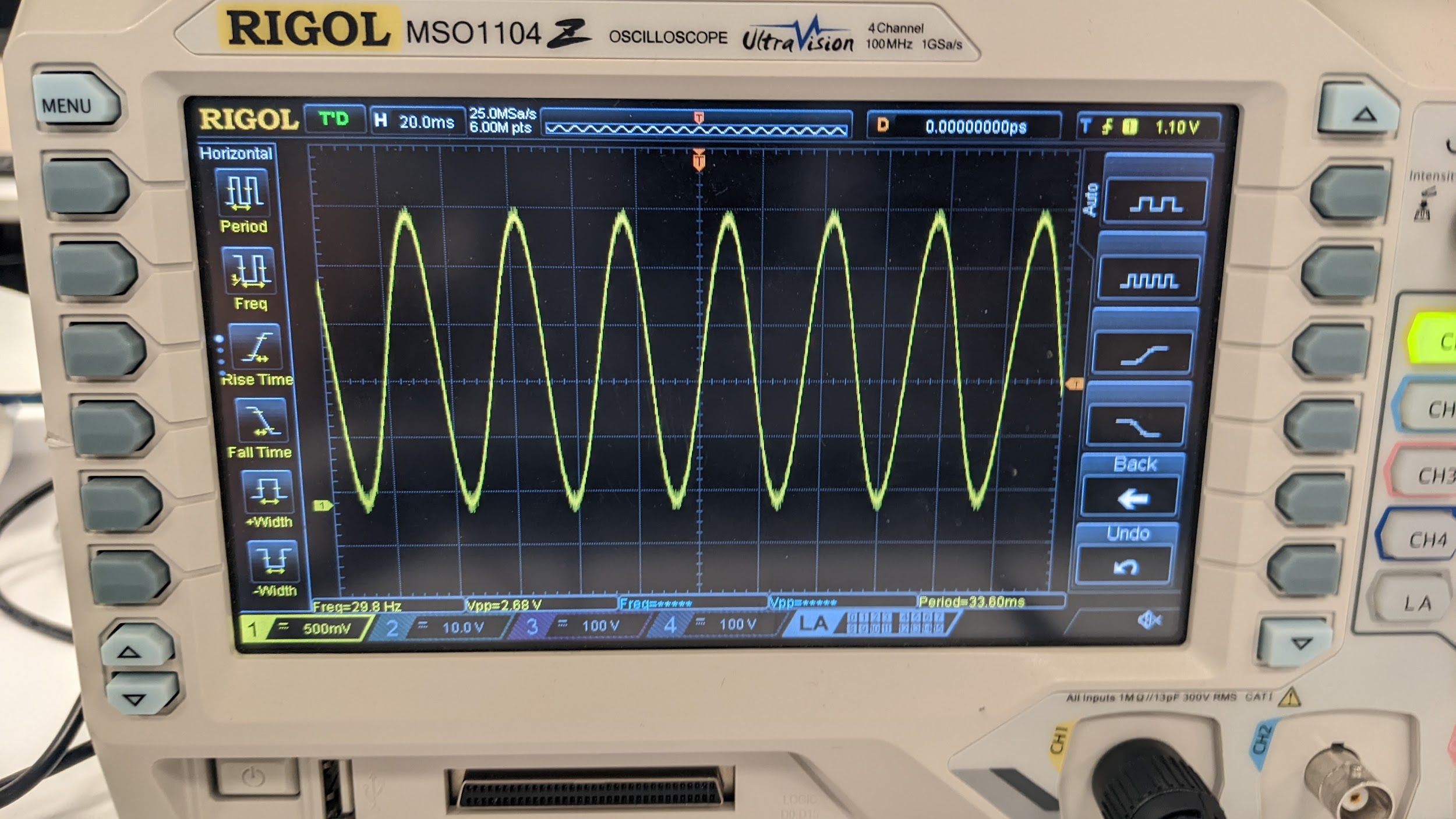
Counts for Target Frequencies Using the SMCLK:

30Hz: CCR0 = 1048576/30/512 = 68

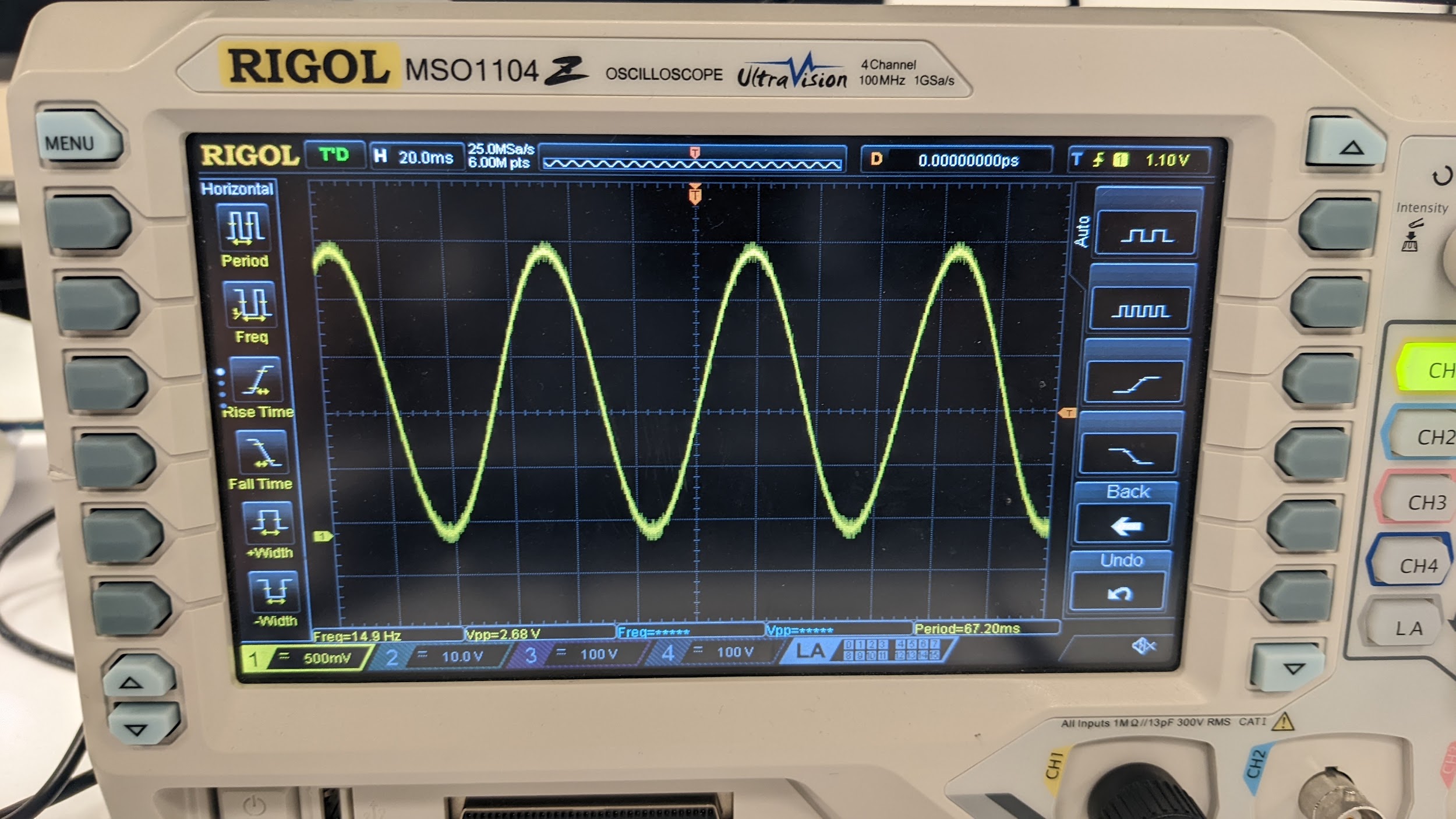
15Hz: CCR0 = 1048576/15/512 = 136

***Program Output:***

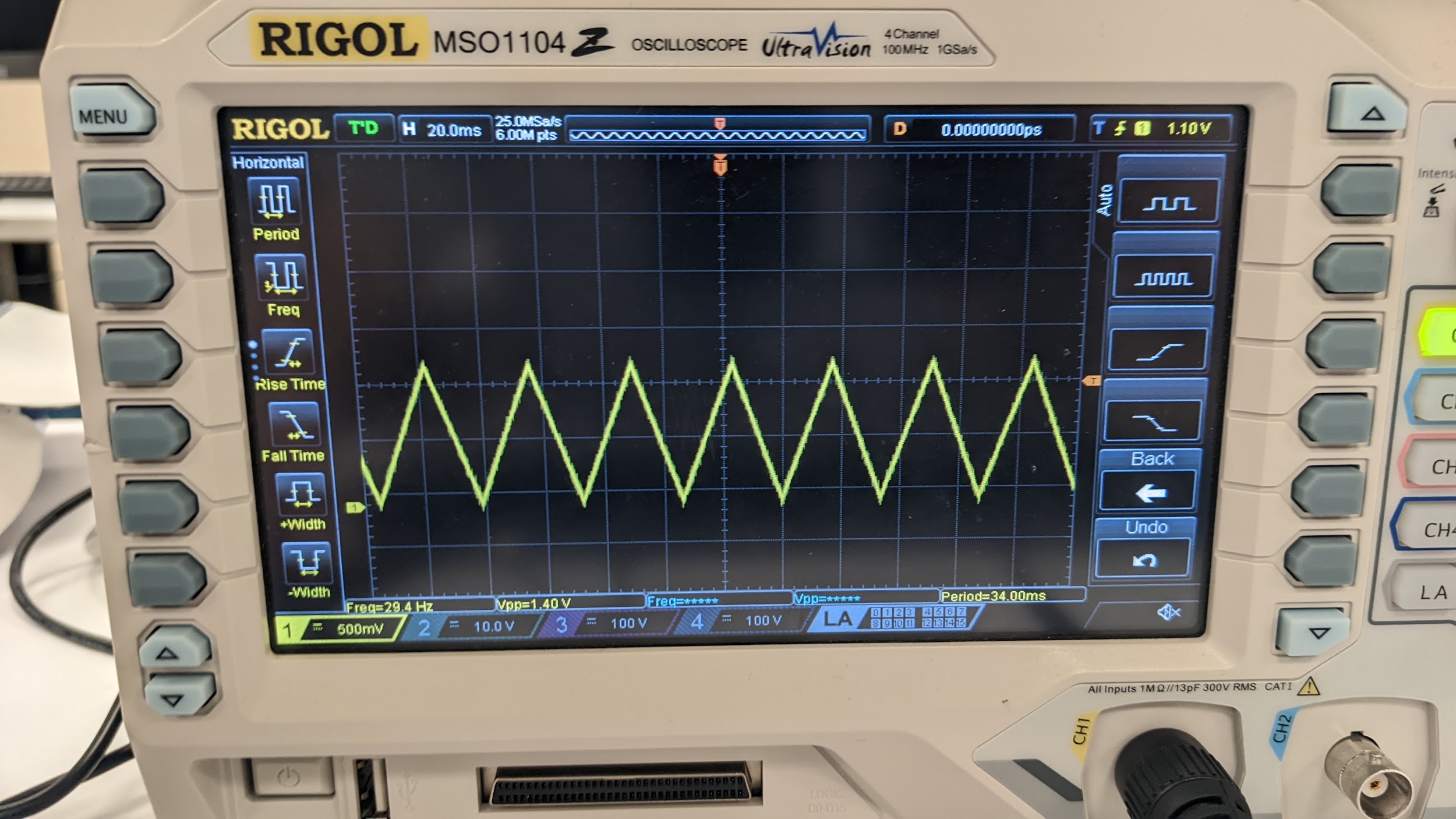
Sinusoidal wave, 30Hz



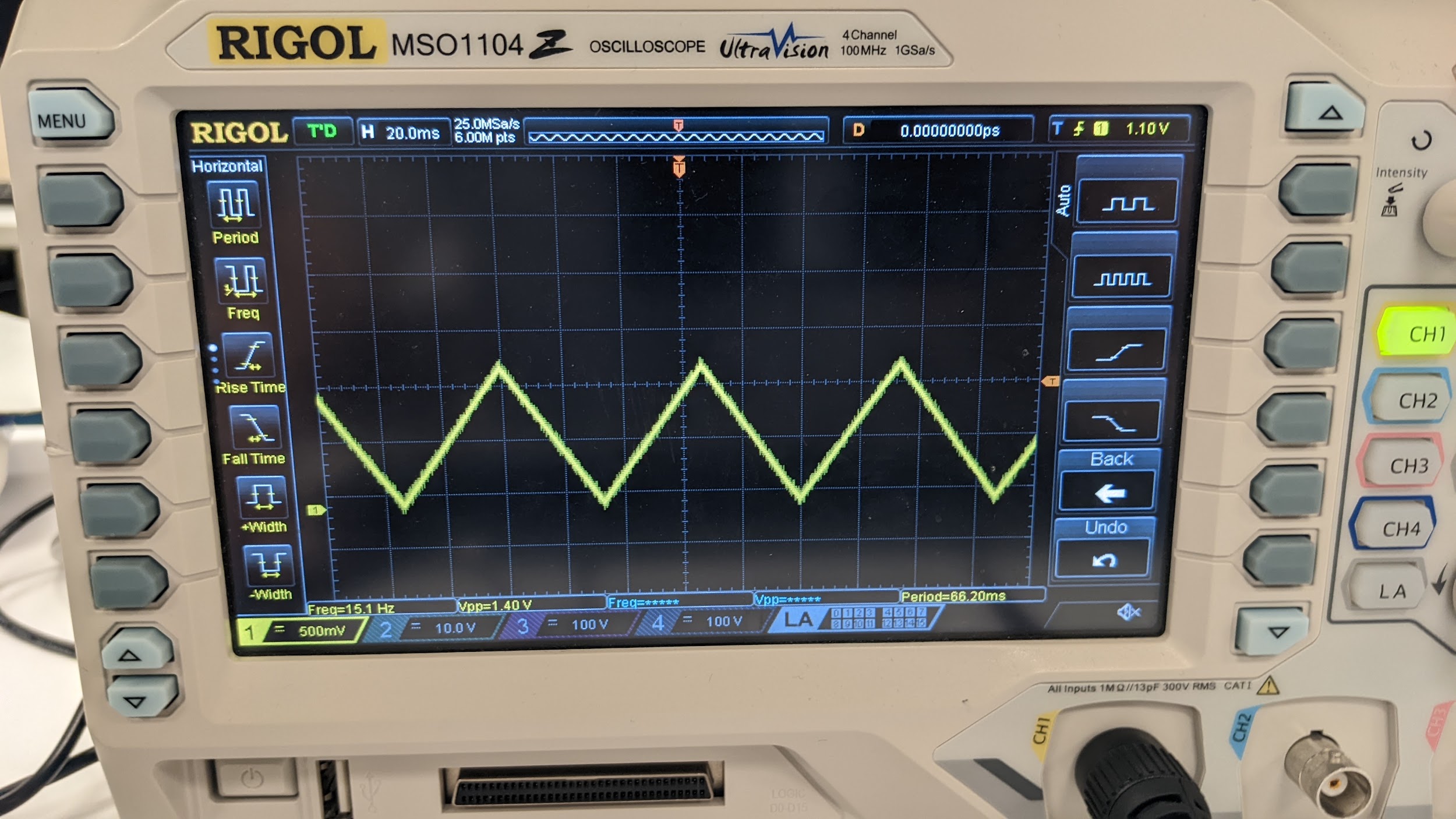
Sinusoidal wave, 15Hz



Triangular wave, 30Hz, half amplitude of sinusoid



Triangular wave, 15Hz, half amplitude of sinusoid



**Conclusion**

This lab was a useful introduction to connecting an external sensor and properly receiving the data with the ADC as well as utilizing the DAC to generate waveforms and view them on the oscilloscope. There were a number of nuances to this lab such as the exact equations used and calculating to get the specified frequencies, which posed a challenge.

***Appendix:***

**Table 1:** Program 1 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab10\_P1.c  \* Function: Interfacing accelerometer (MPS430FG4618)  \* Input: Raw analog data from accelerometer sensor  \* Output: Displays acceleration data in UAH serial app  \* Author: Esther Shore  \*------------------------------------------------------------------------------\*/  #include <msp430xG46x.h>  #include <math.h>  volatile long int ADCXval, ADCYval, ADCZval;  volatile float Xaccel, Yaccel, Zaccel, Xdeviation;  void TimerA\_setup(void) {  TACCR0 = 6554; // 6554 / 32768 Hz = 0.2s => 5Hz  TACTL = TASSEL\_1 + MC\_1; // ACLK, up mode  TACCTL0 = CCIE; // Enabled interrupt  }  void ADC\_setup(void) {  int i = 0;  P6DIR &= ~BIT3+ ~BIT7 + ~BIT5; // Config P6.3, P6.7, P6.5 as input  P6SEL |= BIT3+ BIT7 + BIT5; // Config P6.3, P6.7, P6.5 as analog  ADC12CTL0 = ADC12ON + SHT0\_6 + MSC; // configure ADC converter  ADC12CTL1 = SHP + CONSEQ\_1; // Use sample timer, single sequence  ADC12MCTL0 = INCH\_3; // ADC A3 pin - X-accel  ADC12MCTL1 = INCH\_7; // ADC A7 pin - Y-accel  ADC12MCTL2 = INCH\_5 + EOS; // ADC A5 pin - Z-accel  // EOS - End of Sequence for Conversions  ADC12IE |= 0x02; // Enable ADC12IFG.1  for (i = 0; i < 0x3600; i++); // Delay for reference start-up  ADC12CTL0 |= ENC; // Enable conversions  }  void UART\_putCharacter(char c) {  while(!(IFG2 & UCA0TXIFG)); // Wait for previous character to be sent  UCA0TXBUF = c; // Send byte to the buffer for transmitting  }  void UART\_setup(void) {  P2SEL |= BIT4 + BIT5; // Set up Rx and Tx bits  UCA0CTL0 = 0; // Set up default RS-232 protocol  UCA0CTL1 |= BIT0 + UCSSEL\_2; // Disable device, set clock  UCA0BR0 = 27; // 1048576 Hz / 38400  UCA0BR1 = 0;  UCA0MCTL = 0x94;  UCA0CTL1 &= ~BIT0; // Start UART device  }  void LED\_setup(void) {  P2DIR |= BIT2 | BIT1; // Set LED1 and LED2 as output  P2OUT = 0x00; // Clear LED1, LED2 status  }  void sendData(void) {  int i;  Xaccel = (ADCXval\*3.0/4095 - 1.5) / 0.3; // Calculate accelerations  Yaccel = (ADCYval\*3.0/4095 - 1.5) / 0.3;  Zaccel = (ADCZval\*3.0/4095 - 1.5) / 0.3;  // acceleration (g) = output reading (V) - zero output (V) / sensitivity (V/g)  Xdeviation = atan(Xaccel / sqrt(pow(Yaccel, 2.0) + pow(Zaccel, 2.0))) \* 180.0 / M\_PI;  // Use character pointers to send one byte at a time  char \*xpointer=(char \*)&Xaccel;  char \*ypointer=(char \*)&Yaccel;  char \*zpointer=(char \*)&Zaccel;  char \*devpointer=(char \*)&Xdeviation;  UART\_putCharacter(0x55); // Send header  for(i = 0; i < 4; i++) { // Send x accel - one byte at a time  UART\_putCharacter(xpointer[i]);  }  for(i = 0; i < 4; i++) { // Send y accel - one byte at a time  UART\_putCharacter(ypointer[i]);  }  for(i = 0; i < 4; i++) { // Send z accel - one byte at a time  UART\_putCharacter(zpointer[i]);  }  for(i = 0; i < 4; i++) { // Send x deviation - one byte at a time  UART\_putCharacter(devpointer[i]);  }  }  void main(void) {  WDTCTL = WDTPW +WDTHOLD; // Stop WDT  TimerA\_setup(); // Setup timer to send ADC data  ADC\_setup(); // Setup ADC  UART\_setup(); // Setup UART for RS-232  LED\_setup();  \_EINT();  while (1){  ADC12CTL0 |= ADC12SC; // Start conversions  if (Xdeviation > 15.0) { // if deviation angle > 15 degrees  P2OUT |= BIT2; // LED1 ON, LED2 OFF  P2OUT &= ~BIT1;  } else if (Xdeviation < -15.0) { // if deviation angle < -15 degrees  P2OUT |= BIT1; // LED1 OFF, LED2 ON  P2OUT &= ~BIT2;  } else {  P2OUT &= ~BIT2; // LED1 OFF, LED2 OFF  P2OUT &= ~BIT1;  }  \_\_bis\_SR\_register(LPM0\_bits + GIE); // Enter LPM0  }  }  #pragma vector = ADC12\_VECTOR  \_\_interrupt void ADC12ISR(void) {  ADCXval = ADC12MEM0; // Move results, IFG is cleared  ADCYval = ADC12MEM1;  ADCZval = ADC12MEM2;  \_\_bic\_SR\_register\_on\_exit(LPM0\_bits); // Exit LPM0  }  #pragma vector = TIMERA0\_VECTOR  \_\_interrupt void timerA\_isr() {  sendData(); // Send data to serial app  \_\_bic\_SR\_register\_on\_exit(LPM0\_bits); // Exit LPM0  } |
| --- |

**Table 2:** Program 2 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab10\_P2.c  \* Function: Sinusoidal wave using 512LUT with DAC (MPS430FG4618)  \* Input: Switch 1 or 2 press and hold  \* Output: Sinusoidal or triangular waveform at P6.6  \* Author: Esther Shore  \*------------------------------------------------------------------------------\*/  #include <msp430.h>  #include <sine\_lut\_512.h> /\*512 samples are stored in this table \*/  #include <math.h>  unsigned char SW1pressed = 0; // SW1 status (0 not pressed, 1 pressed)  unsigned char SW2pressed = 0; // SW2 status (0 not pressed, 1 pressed)  unsigned char sinusoid = 1;  void TimerA\_setup(void) {  TACTL = TASSEL\_2 + MC\_1; // SMCLK, up mode  TACCR0 = 68; // Sets Timer Freq (1048576\*0.0333333sec/512)  TACCTL0 = CCIE; // CCR0 interrupt enabled  }  void DAC\_setup(void) {  ADC12CTL0 = REF2\_5V + REFON; // Turn on 2.5V internal ref volage  unsigned int i = 0;  for (i = 50000; i > 0; i--); // Delay to allow Ref to settle  DAC12\_0CTL = DAC12IR + DAC12AMP\_5 + DAC12ENC; //Sets DAC12  }  void Switch\_setup(void) {  P1IE |= BIT0 | BIT1; // P1IE.BIT0 and BIT1 interrupt enabled  P1IES |= BIT0 | BIT1; // P1IES.BIT0 and BIT1 hi/low edge  P1IFG &= ~(BIT0 | BIT1); // P1IFG.BIT0 and BIT1 are cleared  }  void main(void) {  WDTCTL = WDTPW + WDTHOLD; // Stop WDT  TimerA\_setup(); // Set timer to uniformly distribute the samples  DAC\_setup(); // Setup DAC  Switch\_setup();  unsigned int i = 0;  unsigned int j = 0;  int value = 0; // half amplitude  int sign = -1;  while (1) {  \_\_bis\_SR\_register(LPM0\_bits + GIE); // Enter LPM0, interrupts enabled  if (sinusoid == 1) {  DAC12\_0DAT = LUT512[i]; // use 512LUT  i=(i+1)%512; // keep iterating over 512 points  } else if (sinusoid == 0) {  DAC12\_0DAT = value;  if (j % 256 == 0) { // switch direction  sign = -sign;  }  value += sign \* 8; // increment 8 because 2048(half amplitude)/256  j = (j+1)%256; // direction changes every 256 points  }  }  }  #pragma vector = TIMERA0\_VECTOR  \_\_interrupt void TA0\_ISR(void) {  \_\_bic\_SR\_register\_on\_exit(LPM0\_bits); // Exit LPMx, interrupts enabled  }  #pragma vector = PORT1\_VECTOR  \_\_interrupt void Port1\_ISR(void) {  if (P1IFG&BIT0) {  if (SW1pressed == 0) {  sinusoid = 0;  TACCR0 = 68;  SW1pressed = 1;  P1IFG &= ~BIT0; // P1IFG.BIT0 is cleared  P1IES &= ~BIT0; // P1IES.BIT0 low/high edge  } else if (SW1pressed == 1) {  sinusoid = 1;  TACCR0 = 68;  SW1pressed = 0;  P1IFG &= ~BIT0; // P1IFG.BIT0 is cleared  P1IES |= BIT0; // P1IES.BIT0 hi/low edge  }  }  else if (P1IFG&BIT1) {  if (SW2pressed == 0) {  SW2pressed = 1;  TACCR0 = 136; // 15Hz  P1IFG &= ~BIT1; // P1IFG.BIT1 is cleared  P1IES &= ~BIT1; // P1IES.BIT1 low/high edge  } else if (SW2pressed == 1) {  SW2pressed = 0;  TACCR0 = 68; // 30Hz  P1IFG &= ~BIT1; // P1IFG.BIT1 is cleared  P1IES |= BIT1; // P1IES.BIT1 hi/low edge  }  }  } |
| --- |

**Table 3:** sine\_lut\_512\_generator.m

| x=(0:2\*pi/512:2\*pi);  y=2.0\*(sin(x)+1);  dac12=y\*4095/4;  dac12r = round(dac12);  dlmwrite('sine\_lut\_512.h',dac12r, ','); |
| --- |

**Table 4:** sine\_lut\_512.h

| int LUT512[] = { 2048,2073,2098,2123,2148,2173,2198,2223,2248,2273,2298,2323,2348,2373,2398,2422,2447,2472,2496,2521,2545,2569,2594,2618,2642,2666,2690,2714,2737,2761,2784,2808,2831,2854,2877,2900,2923,2946,2968,2990,3013,3035,3057,3078,3100,3122,3143,3164,3185,3206,3226,3247,3267,3287,3307,3327,3346,3366,3385,3404,3423,3441,3459,3477,3495,3513,3530,3548,3565,3581,3598,3614,3630,3646,3662,3677,3692,3707,3722,3736,3750,3764,3777,3791,3804,3816,3829,3841,3853,3865,3876,3888,3898,3909,3919,3929,3939,3949,3958,3967,3975,3984,3992,3999,4007,4014,4021,4027,4034,4040,4045,4051,4056,4060,4065,4069,4073,4076,4080,4083,4085,4087,4089,4091,4093,4094,4094,4095,4095,4095,4094,4094,4093,4091,4089,4087,4085,4083,4080,4076,4073,4069,4065,4060,4056,4051,4045,4040,4034,4027,4021,4014,4007,3999,3992,3984,3975,3967,3958,3949,3939,3929,3919,3909,3898,3888,3876,3865,3853,3841,3829,3816,3804,3791,3777,3764,3750,3736,3722,3707,3692,3677,3662,3646,3630,3614,3598,3581,3565,3548,3530,3513,3495,3477,3459,3441,3423,3404,3385,3366,3346,3327,3307,3287,3267,3247,3226,3206,3185,3164,3143,3122,3100,3078,3057,3035,3013,2990,2968,2946,2923,2900,2877,2854,2831,2808,2784,2761,2737,2714,2690,2666,2642,2618,2594,2569,2545,2521,2496,2472,2447,2422,2398,2373,2348,2323,2298,2273,2248,2223,2198,2173,2148,2123,2098,2073,2048,2022,1997,1972,1947,1922,1897,1872,1847,1822,1797,1772,1747,1722,1697,1673,1648,1623,1599,1574,1550,1526,1501,1477,1453,1429,1405,1381,1358,1334,1311,1287,1264,1241,1218,1195,1172,1149,1127,1105,1082,1060,1038,1017,995,973,952,931,910,889,869,848,828,808,788,768,749,729,710,691,672,654,636,618,600,582,565,547,530,514,497,481,465,449,433,418,403,388,373,359,345,331,318,304,291,279,266,254,242,230,219,207,197,186,176,166,156,146,137,128,120,111,103,96,88,81,74,68,61,55,50,44,39,35,30,26,22,19,15,12,10,8,6,4,2,1,1,0,0,0,1,1,2,4,6,8,10,12,15,19,22,26,30,35,39,44,50,55,61,68,74,81,88,96,103,111,120,128,137,146,156,166,176,186,197,207,219,230,242,254,266,279,291,304,318,331,345,359,373,388,403,418,433,449,465,481,497,514,530,547,565,582,600,618,636,654,672,691,710,729,749,768,788,808,828,848,869,889,910,931,952,973,995,1017,1038,1060,1082,1105,1127,1149,1172,1195,1218,1241,1264,1287,1311,1334,1358,1381,1405,1429,1453,1477,1501,1526,1550,1574,1599,1623,1648,1673,1697,1722,1747,1772,1797,1822,1847,1872,1897,1922,1947,1972,1997,2022 }; |
| --- |