

CMPE 460 Laboratory Exercise 5
MSP432 Timers, Interrupts, and Analog-to-Digital Converter

Mohammed Fareed
Trent Wesley
Performed: September 27, 2023
Submitted: October 18, 2023

Lab Section: 1
Instructor: Prof. Hussin Ketout
TA: Andrew Tevebaugh
Colin Vo

Lecture Section: 1
Professor: Prof. Hussin Ketout

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Your Signature: _____

Abstract

This laboratory exercise explored the implementation and utilization of interrupts, timers, ADC, a line scan module, and a DAC with an MSP432 microcontroller board and C code. Switches were configured to generate interrupts on the click of the button. These interrupts control the operation of two Timer32 modules. The Timer32 modules were used to periodically generate interrupts. Timer32-1 toggles an LED every 0.5 seconds. Timer32-2 both records time between button presses and controls the state of an LED. The time between presses gets displayed to a terminal with the UART. The analog to digital converter was used to read the voltage controlled by a photocell and a temperature sensor. The line scan module was interfaced with the MSP432 and verified with an oscilloscope and a MATLAB script. Additionally, the MSP432 was interfaced with an MCP 4901 DAC to generate a 1Hz sine wave. This exercise was successful since all parts operated as expected. The timers functioned reliably for controlling LEDs and timing operations, the ADC was able to successfully measure light intensity and temperature, the linescan camera produced an accurate output, and the DAC produced a clean 1Hz sine wave.

Design Methodology

Interrupts are a useful tool for efficiently handling events by pausing the CPU's current task to handle another. Interrupts provide the benefit of assigning priority to certain events so that they can be executed in a proper order. In addition, interrupts avoid the need to constantly poll, which can interfere with the flow of a program. Interrupts are used extensively in this exercise by timers, switches, UARTs, and GPIO.

The MSP432 contains two Timer32 modules for timing operations. Each module contains a counter which can be configured as a 16-bit or 32-bit down counter. In this exercise, C code was written so that pressing Switch1 on the MSP432 microcontroller board activates Timer32-1 which toggles LED1 every 0.5 seconds. To accomplish this, Switch1 was configured to generate interrupts on the falling edge of the button click. The interrupt from a button click starts the interrupt service routine for the switch's port. Within the ISR, the interrupt flag for Switch1 is checked to see if the interrupt came from it. If the interrupt was from Switch1, the state of Timer32-1 gets toggled and a boolean value tracking Timer32-1's state is updated. While Timer32-1 is active, it generates an interrupt every 0.5 seconds. Every interrupt triggers an ISR which checks the state of LED1 with a boolean variable and toggles both the variable and the LED state.

More code was written to track time between button presses and cycle an LED through a list of colors. Switch2 on the MSP432 microcontroller board was configured to generate interrupts when pressed. The ISR of Switch2 would first check its interrupt flag to make sure that the interrupt came from it. If Timer32-2 was not already running according to a boolean flag variable, a millisecond counter variable gets set to 0, LED2 gets set to the next color, and a color index variable gets updated. Timer32-2 generates an interrupt every millisecond and triggers an ISR which increments a millisecond counter. If Timer32-2 was

running on a Switch2 press, LED2 gets turned off and the UART sends a message of how much time has elapsed based on the millisecond counter.

Analog to digital converters are essential for converting real-world data into computer readable digital signals. The MSP432's ADC was used to measure the voltage which varied as the resistance of a photocell changed with light intensity. The circuit in Figure 1 was constructed to measure this varying voltage.

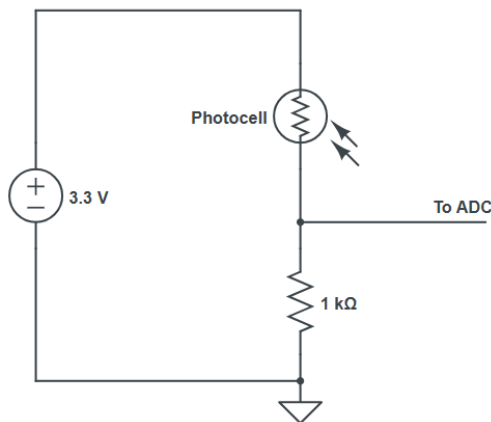


Figure 1: Photocell Circuit

This circuit shown in Figure 1 is a voltage divider circuit with the middle node going to the ADC. The voltage at the middle node follows the behavior shown in Equation 1.

$$V_N = 3.3V * \frac{1k\Omega}{1k\Omega + R_{Photocell}} \quad (1)$$

Equation 1 shows that if the resistance of the photocell increases, the voltage measured by the ADC decreases. Since the resistance of photocell decreases with increased light intensity, it is known that the measured voltage will increase as light intensity increases.

To measure the voltage with the MSP432, the ADC needs to be initialized with C code. First, the reference was configured to be a static 2.5V. The code waits for the reference voltage to be ready by continually checking the variable reference voltage ready status register until it is set to 1. Then, the ADC's enable conversion bit within the control 0 register is set to allow programming. The code waits for the ADC to be ready by continually checking the control 0 register's busy status bit. Once the ADC is not busy, the code sets up the ADC to operate in single-channel, 14-bit resolution mode, with specific clock and pulse-mode configurations. Channel 6 is selected for conversion, and interrupts are disabled. Additionally, pin P4.7 is configured for analog input.

Timer32-1 was configured to generate interrupts every 0.5 seconds. With each interrupt, an interrupt service routine reads the value in the ADC memory register and outputs the value in decimal and hexadecimal. The voltage is calculated by multiplying the value by 2.5 since

the reference range is 2.5V and dividing it by 2^{14} since a resolution of 14 bits is used.

The photocell was replaced with a TMP36 temperature sensor. Using the datasheet, it was found that the voltage output is a linear function of the temperature. The relation between the output voltage and the temperature in degrees celsius is shown in Equation 2.

$$T_{\circ C} = 100V_{\text{out}} - 50 \quad (2)$$

After finding the temperature in degrees Celsius with Equation 2, the temperature in Fahrenheit can easily be found with a simple Celsius to Fahrenheit conversion. The code was modified to output the temperature calculated in both Celsius and Fahrenheit with the UART.

The MSP432's ADC was also used to interface a line scan camera. The camera was connected to the MSP432 microcontroller board using the circuit shown in Figure 2.

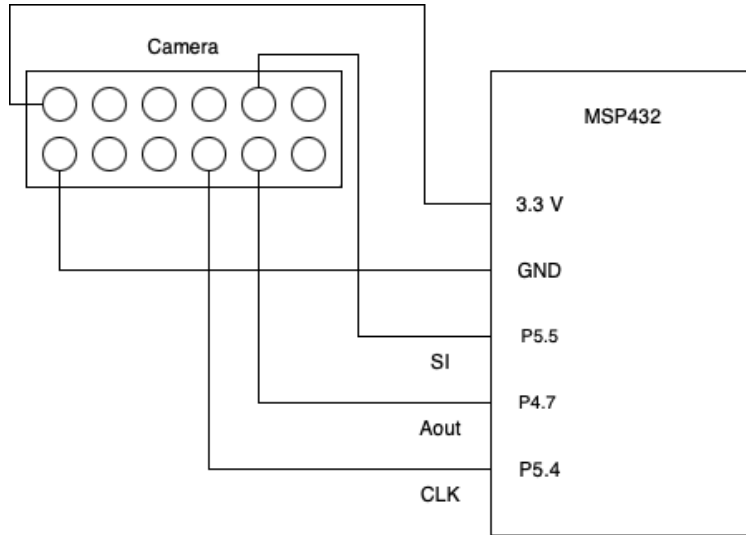


Figure 2: Camera Circuit Diagram

The diagram shows that the board was used to provide power to the camera and to receive data from it. The clock was generated using the SysTick timer with interrupts at 180 kHz. The SI signal was generated using Timer32-1 with interrupts and an integration time of 20 ms. The clock was generated at port 5 pin 4 and the SI signal at port 5 pin 5. The camera's data output was connected to port 5 pin 6, where the ADC was configured to read the data and send it over UART.

Code was written for the camera that initialized the UART, an LED that turns on while the camera is integrating, the clock and SI signals, and the ADC. The UART was initialized to send data at 9600 baud rate. The LED was initialized to be off and the ADC was initialized to read from channel 6. The ADC was also configured to use 14-bit resolution, 2.5 V reference, and no interrupts. The SysTick timer was configured to generate interrupts at double the provided period to handle both rising and falling edges. The Timer32 was configured to run in periodic mode, wrapping mode, 32-bit mode, and with interrupts. The Timer32's value

was adjusted to trigger camera usage at a frequency appropriate for MATLAB to process and plot.

A MATLAB script was also used to receive the camera data over UART and plot it. The script was configured to receive 128 bytes of data at a time, which is the amount of data the camera sends. The script was also configured to plot the data in three ways: raw, after being processed by a 5-point moving average filter, and after being converted to a binary image. The threshold used to convert the data to a binary image was adjusted through trial and error to 5000.

A sine wave was generated using the MCP4901 DAC by interfacing it with the MSP432 microcontroller board using SPI. The DAC was connected to the MSP432 microcontroller board using the circuit shown in Figure 3.

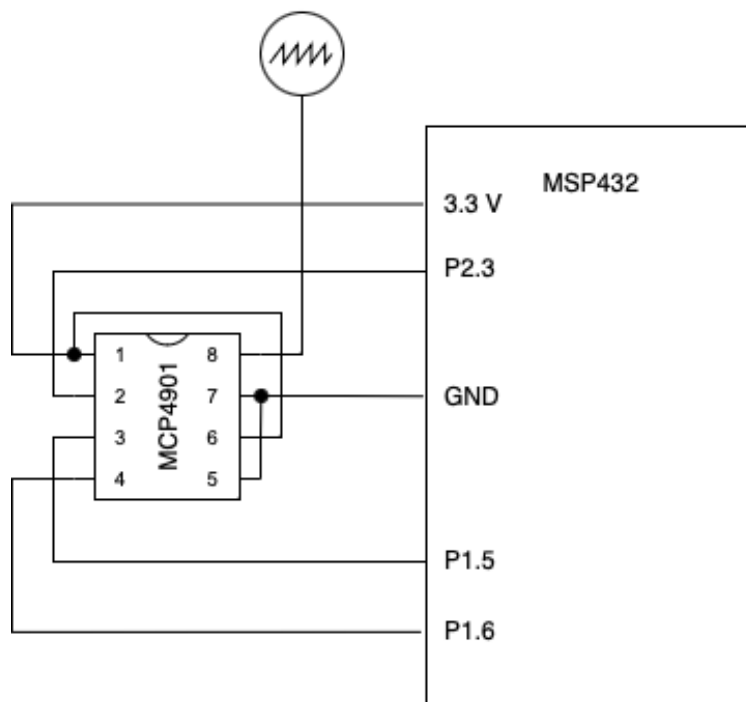


Figure 3: DAC Circuit Diagram

The diagram shows that the DAC was connected to the MSP432 microcontroller board using SPI, for which power is provided by the board. The DAC's output was connected to the oscilloscope to measure the generated sine wave. The DAC's CS signal was connected to port 2 pin 3, which is the slave select signal of the SPI, enabling the DAC only when the SPI is writing to it. The DAC's SCLK signal was connected to port 1 pin 5, which is the clock signal of the SPI. The DAC's SDI signal was connected to port 1 pin 6, which is the data signal of the SPI. The DAC's reference voltage used is the same as its power supply, which is 3.3 V, and its LDAC signal was connected to ground to enable the DAC's output to update immediately.

The SPI was initialized with the board as the master with a control word of the value 0xE9C1, with ports 1.5 and 1.6 as UCB0CLK and UCB0SIMO respectively. The DAC was then configured

to output a sine wave with a frequency of 1 Hz and a peak-to-peak voltage of 3.3 V. This was done by pre-computing 100 values of the sine wave and storing them in an array. The DAC was then configured to output the values in the array in order, with a delay in between each value to achieve the desired frequency. Equation 3 shows the equation used for the sine wave.

$$V_{\text{out}} = 120 * \sin(2\pi i/100) + 128 \quad (3)$$

The values were chosen as follows:

1. The DAC outputs values from 0 to 255, so the amplitude was chosen to cover the range minus 16 to avoid clipping.
2. The sine wave has a DC offset of 128 to center the wave in the range of the DAC output range.
3. The pre-computed values cover only one period, therefore the angular frequency was chosen to be $2\pi/100$ to span the entire array.

The delay between each write to the DAC was added to control the frequency of the signal and achieve the desired 1 Hz. The delay is loop-based with a constant number of cycles per delay. The number of cycles was modified until the desired frequency was achieved.

Results and Analysis

The timers were tested by pressing Switch1 on the MSP432 microcontroller board and verifying that LED1 toggles every 0.5 seconds. The timers were also tested by pressing Switch2 on the MSP432 microcontroller board and verifying that LED2 cycles through the colors red, green, and blue, cyan, magenta, yellow, white. Figure 4 shows the UART output of the time elapsed between button presses, calculated using Timer32-2.

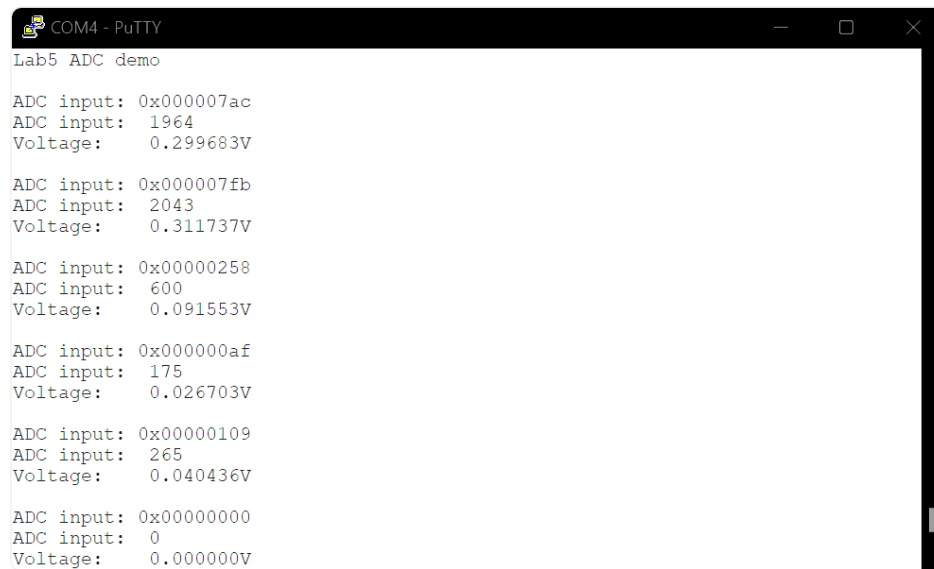


```
COM4 - PuTTY
Lab5 Timer demo
Seconds 3.106000
Seconds 0.909000
Seconds 0.332000
Seconds 0.373000
Seconds 0.363000
Seconds 0.319000
Seconds 0.257000
Seconds 0.216000
█
```

Figure 4: Part 1 Timer32-2 UART Output

The figure shows the time elapsed between button presses as the color of the LED changes back to red. The time between button presses was then verified by using the UART to print the time elapsed since the last button press, comparing the printed result with an external timer.

The ADC was tested by measuring the voltage of the photocell. The code for this part was tested by shining a bright light on the photocell and verifying that the measured voltage increased. The measured voltage was then verified to decrease when the photocell was covered. Figure 5 shows the UART output of the measured voltage.



```
COM4 - PuTTY
Lab5 ADC demo
ADC input: 0x000007ac
ADC input: 1964
Voltage: 0.299683V

ADC input: 0x000007fb
ADC input: 2043
Voltage: 0.311737V

ADC input: 0x00000258
ADC input: 600
Voltage: 0.091553V

ADC input: 0x000000af
ADC input: 175
Voltage: 0.026703V

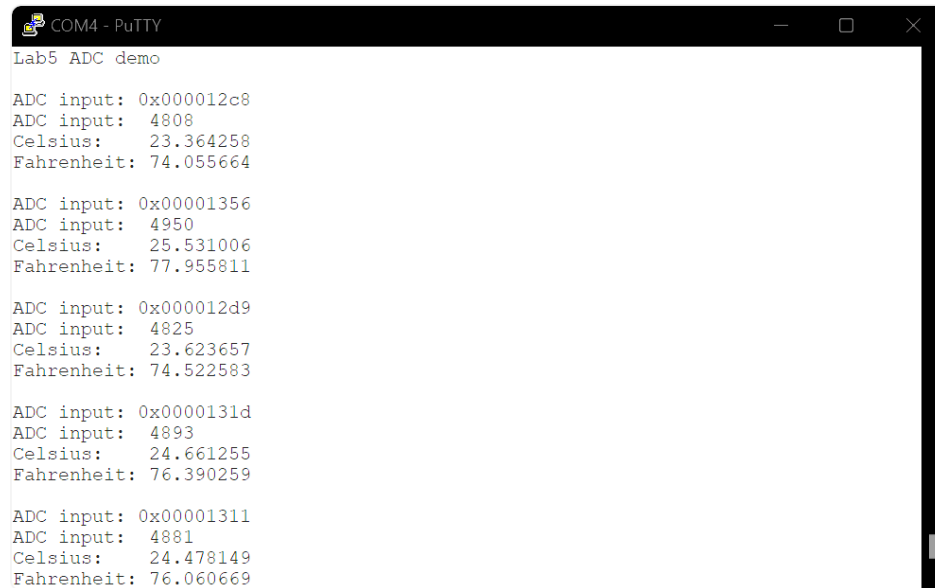
ADC input: 0x00000109
ADC input: 265
Voltage: 0.040436V

ADC input: 0x00000000
ADC input: 0
Voltage: 0.000000V
```

Figure 5: Part 2 ADC Voltage UART Output

The figure shows the measured voltage of the photocell as the light intensity changes. The measured voltage was verified by using the UART to print the voltage, comparing the printed result with a multimeter. The result shows that the measured voltage when light was shined on the photocell was around 0.3V, and the measured voltage when the photocell was covered was 0V.

The code was then modified to use a temperature sensor, calculating the temperature from the voltage in both Celsius and Fahrenheit. Figure 6 shows the UART output after replacing the photocell with the temperature sensor TMP36.



```
COM4 - PuTTY
Lab5 ADC demo

ADC input: 0x000012c8
ADC input: 4808
Celsius: 23.364258
Fahrenheit: 74.055664

ADC input: 0x00001356
ADC input: 4950
Celsius: 25.531006
Fahrenheit: 77.955811

ADC input: 0x000012d9
ADC input: 4825
Celsius: 23.623657
Fahrenheit: 74.522583

ADC input: 0x0000131d
ADC input: 4893
Celsius: 24.661255
Fahrenheit: 76.390259

ADC input: 0x00001311
ADC input: 4881
Celsius: 24.478149
Fahrenheit: 76.060669
```

Figure 6: Part 2 UART Output with TMP36

The figure shows the measured temperature of the sensor at room temperature, which was correctly around 24 degrees Celsius, or 75 degrees Fahrenheit.

The line scan camera was then interfaced using timers and an ADC to send the received data serially over UART. The code for this part was tested by connecting the line scan camera to the MSP432 microcontroller board and verifying that the received data was sent correctly over UART using PuTTY.

The camera was then tested by pointing it at a bright light then a dark area repeatedly. The received data was then plotted using MATLAB to verify that the received data behaved correctly. Figure 7 shows the received data plotted in MATLAB.

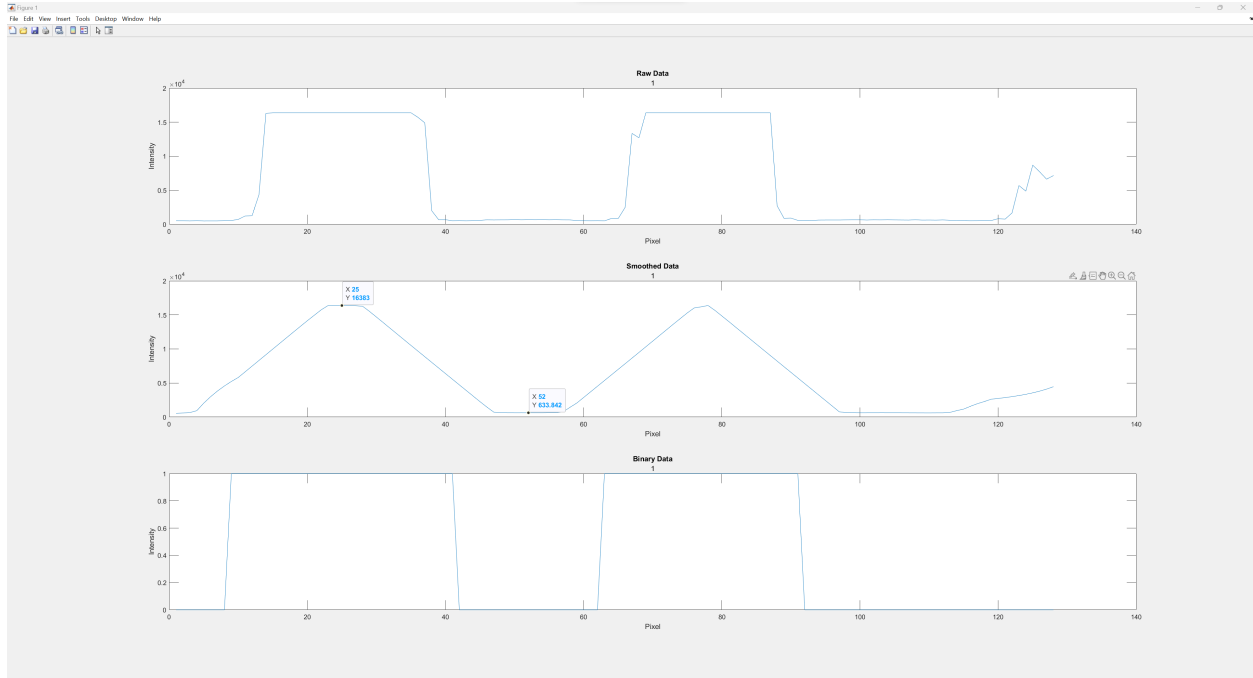


Figure 7: Part 3 MATLAB Camera Data Plots

The figure shows that the raw data correctly reflects the camera's output, with the dark area having a lower value than the bright area, creating a clock-like pattern. The figure also shows that the moving average filter correctly smooths the data, creating a triangular wave, gradually increasing and decreasing as the camera is moved from the bright area to the dark area. The figure also shows that the binary image correctly converts the data to a binary image, outputting a 1 for the bright area and a 0 for the dark area, which acts as an edge detector.

A sine wave was generated for part 4 of the exercise using MCP4901 DAC by interfacing it with the MSP432 microcontroller board using SPI using a reference voltage of 3.3 V. The code written for this part generated a 1 Hz sine wave spanning 240 out of the possible 256 of the DAC. Figure 8 shows the generated sine wave measured by an oscilloscope.

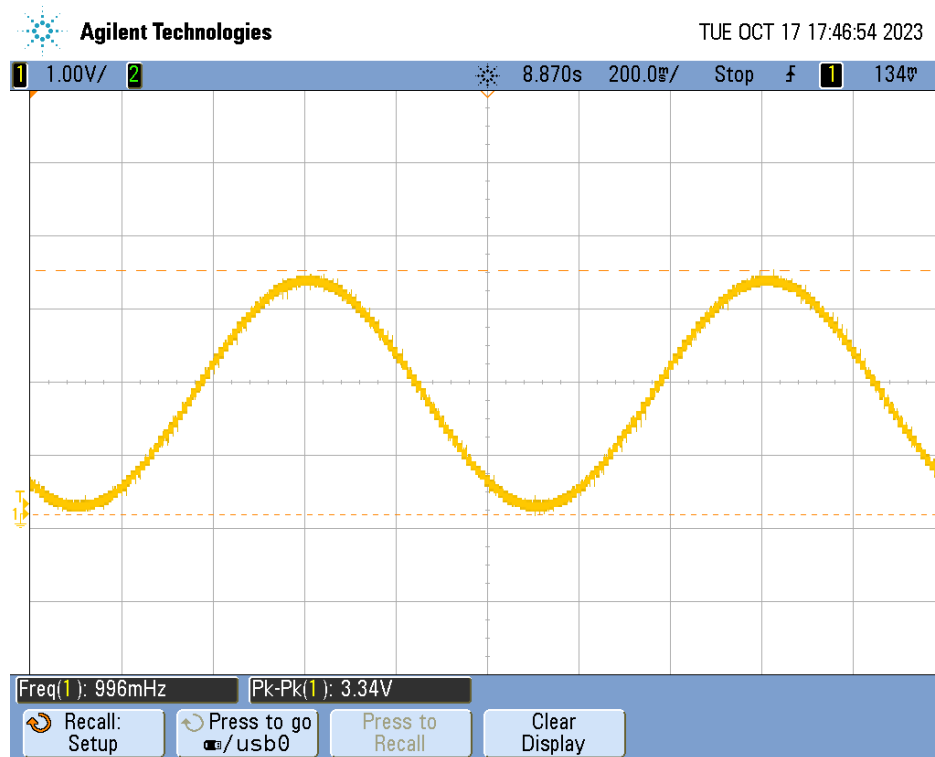


Figure 8: Part 4 DAC Oscilloscope Output

The figure shows the generated 1 Hz sine wave spanning the outputs of the DAC, minus 8 values from both peaks. The oscilloscope screen capture shows that the sine wave has a frequency of 0.996 Hz and a peak-to-peak voltage of 3.34 V. The desired frequency was achieved by using a loop-based delay of 625 cycles.

The smallest amount of time that could be measured on the MSP432 depends on the frequency the board is running at. Since an interrupt can be generated on every clock cycle, the smallest amount of time that can be measured is the period of the clock. The MSP432 microcontroller board used has a clock frequency of 48 MHz, so the smallest amount of time that can be measured is 20.83 ns. The longest amount of time that can be measured is arbitrary since an accumulator can be used to measure arbitrarily large amounts of time, triggering interrupts at any desired interval.

Questions

(a) *Why do the IRQ flags need to be cleared in the interrupt service routine functions?*

When the microcontroller detects that an IRQ flag is set, it will jump to the corresponding Interrupt Service Routine (ISR) to handle that event. Therefore, the IRQ flag needs to be cleared in the ISR so that the microcontroller does not jump to the ISR again when it detects that the IRQ flag is set.

(b) *When an interrupt occurs, how do you know which pin caused it?*

When an interrupt occurs, the microcontroller will jump to the corresponding Interrupt Service Routine (ISR) to handle that event. The ISR can then check the interrupt flag register **PxIFG** to see which pin caused the interrupt. For example, if bit 1 or **P1IFG** was set, then the interrupt was triggered by port 1 pin 1.

(c) *What is the startup sequence of SI and CLK to initiate data transfer from the Camera?*

Data transfer is initialized by pulsing the SI pin while the clock is low. The startup sequence of SI and CLK to initiate data transfer from the camera is as follows:

1. CLK is set to 0
2. SI is set to 0
3. SI is set to 1
4. SI is set to 0
5. CLK starts toggling

Conclusion

Exercise 5: MSP432 Timers, Interrupts, and Analog-to-Digital Converter

Student's Name: Mohammed Fared Trent Wesley Section: 1

| PreLab | | Point Value | Points Earned | Comments |
|--------|---------------|-------------|---------------|----------|
| PreLab | Prelab C Code | 10 | 10 | W 9/27 |

| Demo | | Point Value | Points Earned | Date |
|------|---|-------------|---------------|-----------|
| Demo | Functionality with SW1 | 5 | 5 | } W 9/27 |
| | Functionality with SW2 | 5 | 5 | |
| | ADC Functionality with CdS | 5 | 5 | W 10/4 |
| | ADC Functionality with Temp Sensor | 5 | 5 | CV 10/4 |
| | Linescan Camera MATLAB and Oscilloscope | 5 | 5 | CV 10/16 |
| | DAC SPI Interface Oscilloscope | 5 | 5 | ATT 10/17 |

To receive any grading credit students must earn points for both the demonstration and the report.