AD1Pmod and DA2Pmod in FreeRTOS v2

Robert Bara
Robert.Bara@temple.edu

Summary

Lab 7 builds up to the implementation of digital filtering by combining the PmodAD1 ADC, which was introduced in lab 6, and introducing the PmodDA2 DAC to convert digital data back to an analog waveform. The project is done in Vivado and passes data through queues in FreeRTOS, similar to Lab5.

Introduction

A sample standalone C project *AD1DA2PmodPT.c* is supplied to act as a template for this lab. The program should analyze for the various control and status signals, and using waveforms, an oscilloscope will measure the data for further analysis.

The C project should then be converted into a FreeRTOS program with two equal priority tasks *AD1task* and *DA2task*, which sends and receives data using a queue of size 10. Neither the tasks nor queues should have block time. Using waveforms, the same analysis should be performed using the oscilloscope.

To practice data manipulation, take the FreeRTOS program and generate a copy which now includes another equal priority task *SQRtask*, as well as another queue of size 10. Data should be collected from the *AD1task* and sent to a queue to be received by *SQRtask* which will then square the analog signal and then be sent to a queue to be received by *DA2task* to be outputted back as an analog wave for the oscilloscope analysis.

Discussion

Hardware Design

The hardware design is a straightforward design accomplished within Vivado. Start by adding the *AD1DA2JE*.xdc constraint file from Canvas, so that the Hardware design will sync up with the SDK software design. Create a block diagram and begin by adding the ZYNQ processor system block and running automation. Edit the processing system so that the PmodAD1 ADC will use FCLK_CLK1 at 30MHz, while PmodDA2 DAC will use FCLK_CLK2 at 50MHz. This is done by enabling the clocks through PL Fabric Clocks, just as was done within lab 6. Add the AD1Pmod and DA2Pmod blocks that were given by the custom library that was added from lab 6. Run automation and be sure to connect AD1Pmod and DA2Pmod to their respective clocks that is on the ZYNQ processor block. Make the AD1dat, AD1dat2, AD1sync, AD1sclk, DA2sync, DA2dat1, DA2dat2, and DA2sclk connections external. Verify the design. The completed block diagram should result as follows:

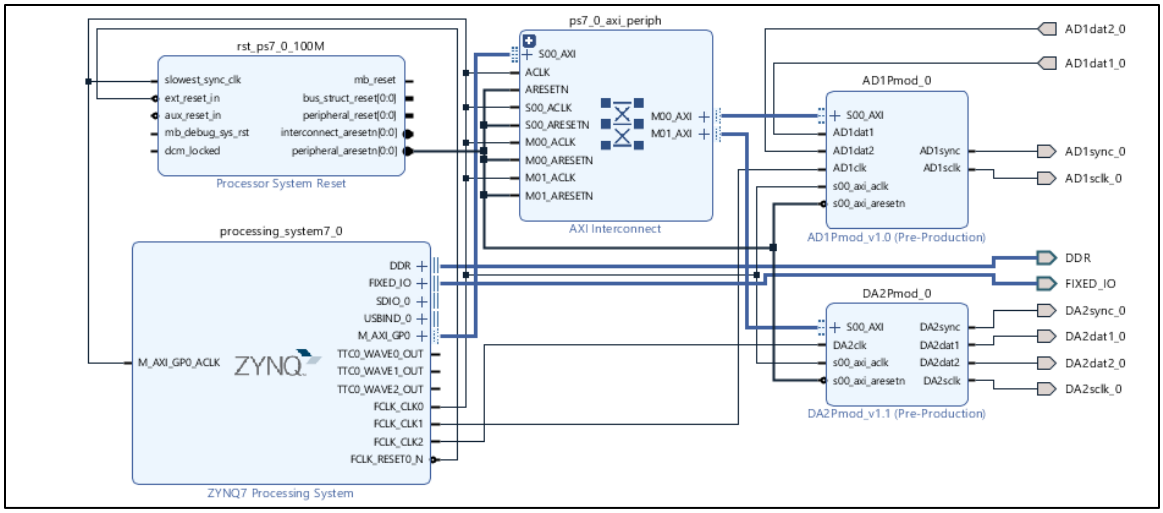


Figure 1. Completed Block Diagram

Create an HDL wrapper and generate a bitstream. Export the design to Hardware and run launch SDK.

Software Design

C Template Code

The template program written by Dr. Silage begins by defining the libraries and addresses obtained from the address editor in Vivado to map the external inputs and outputs of the PmodAD1 and PmodDA2:

```
//AD1DA2PmodPT Example  ECE3622  c2021 Dennis Silage

#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "sleep.h"

//AD1Pmod from Address Editor in Vivado, first IP
#define AD1acq  0x43C00000  //AD1 acquisition    - output
#define AD1dav  0x43C00004  //AD1 data available - input
#define AD1dat1 0x43C00008  //AD1 channel 1 data - input
#define AD1dat2 0x43C0000C  //AD1 channel 2 data - input

//DAC2Pmod from Address Editor in Vivado, second IP
#define DA2acq  0x43C10000      //DA2 acquisition    - output
#define DA2dav  0x43C10004  //DA2 data available - input
#define DA2dat1 0x43C10008      //DA2 channel 1 data - output
#define DA2dat2 0x43C1000C  //DA2 channel 2 data - output
```

Figure 2. Headers and Definitions

There is one subroutine, which is main, and can be broken up into two parts. The initializes starts by defining variables for each channel and using Xil_In/Out32, the AD1 and DA2 are initialized and checked for available data. If data is available, then channels will be initialized, and data will be collected:

```
int main(void)
{
    int adcdav;     //ADC data available
    int adcdata1;       //ADC channel 1 data
    int adcdata2;   //ADC channel 2 data

    int dacdata1;       //DAC channel 1 data
    int dacdata2;       //DAC channel 2 data
    int dacdav;     //DAC data available

    xil_printf("\n\rStarting AD1-DA2 Pmod demo test...\n");
    Xil_Out32(AD1acq, 0);       //ADC stop acquire
    adcdav = Xil_In32(AD1dav);  //ADC available?
    while (adcdav == 1)
        adcdav = Xil_In32(AD1dav);
    Xil_Out32(DA2acq, 0);       //DAC stop acquire
    dacdav = Xil_In32(DA2dav);  //DAC available?
    while (dacdav == 1)
        dacdav = Xil_In32(DA2dav);
```

Figure 3. Checking for Input Data

Data is checked within each channel 1 and channel 2 of the AD1 and DA2. When there I data the program inputs analog data from the AD1 and converts it to a digital signal. *Dacdata1* and *dacdata2* are then set equal to the channel 1 and channel2 digitalized data from the AD1. DA2 then outputs these back as analog signals that will be verified by the oscilloscope. Both ADC and DAC have appropriate conditions to check for resets and stop collecting data depending on the status of the inputs *adcdav* or *dacdav*.

```
    while (1)
    {
        //ADC
        Xil_Out32(AD1acq, 1);       //ADC acquire
        while (adcdav == 0)     //ADC data available?
            adcdav = Xil_In32(AD1dav);
        Xil_Out32(AD1acq, 0);       //ADC stop acquire
        adcdata1 = Xil_In32(AD1dat1);   //input ADC data
        adcdata2 = Xil_In32(AD1dat2);
        while (adcdav == 1)     //wait for reset
            adcdav = Xil_In32(AD1dav);

        dacdata1 = adcdata1;        //ADC -> DAC pass through
        dacdata2 = adcdata2;

        //DAC
        Xil_Out32(DA2dat1, dacdata1);   //output DAC data
        Xil_Out32(DA2dat2, dacdata2);
        Xil_Out32(DA2acq, 1);       //DAC acquire
        while (dacdav == 0)     //DAC data output?
            dacdav = Xil_In32(DA2dav);
        Xil_Out32(DA2acq, 0);       //stop DAC acquire
        while (dacdav == 1)     //wait for reset
            dacdav = Xil_In32(DA2dav);

    }
}
```

Figure 4. Passing data from the ADC to DAC

FreeRTOS Task 2

The second task from this lab essentially converts the C template program into a FreeRTOS program. The initialization is as follows. First libraries are defined as well as the tasks, task handles, and queue.

```
    Copyright (C) 2017 Amazon.com, Inc. or its affiliates.  All Rights Reserved.
//Robert Bara Lab 7 ECE3623 Spring 2021
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
#include "sleep.h"
//definitions
#define printf xil_printf
/*-----------------------------------------------------------*/
/* The tasks as described at the top of this file. */
static void AD1task( void *pvParameters );
static void DA2task( void *pvParameters );
/*-----------------------------------------------------------*/
/* Defining Task Handles */
static TaskHandle_t xAD1Task;
static TaskHandle_t xDA2Task;
/* The queue used by the tasks, as described at the top of this
file. */
static QueueHandle_t xQueueData = NULL;
```

Figure 5. Headers and Task/Queue Definitions

Using the address editor in Vivado, the AD1pmod and DAC2pmod external inputs and outputs are defined.

```
//AD1Pmod from Address Editor in Vivado, first IP
#define AD1acq  0x43C00000  //AD1 acquisition    - output
#define AD1dav  0x43C00004  //AD1 data available - input
#define AD1dat1 0x43C00008  //AD1 channel 1 data - input
#define AD1dat2 0x43C0000C  //AD1 channel 2 data - input

//DAC2Pmod from Address Editor in Vivado, second IP
#define DA2acq  0x43C10000      //DA2 acquisition    - output
#define DA2dav  0x43C10004  //DA2 data available - input
#define DA2dat1 0x43C10008      //DA2 channel 1 data - output
#define DA2dat2 0x43C1000C  //DA2 channel 2 data - output
```

Figure 6. Addresses and variable definitions for connections

The main function begins by outputting that the program is beginning to run and the two tasks for AD1 and DA2 are created with equal priority, while a queue for the data sent and received is created and checked with a size 10. The tasks begin to run and the rest of main runs as normal.

```
/* Creating the two tasks with equal priority for ADC and DAC */
xTaskCreate(    AD1task,                    /* The function that implements the task. */
                ( const char * ) "AD",      /* Text name for the task, provided to assist debugging only. */
                configMINIMAL_STACK_SIZE,   /* The stack allocated to the task. */
                NULL,                       /* The task parameter is not used, so set to NULL. */
                tskIDLE_PRIORITY + 1,       /* The task runs at the idle priority. */
                &xAD1Task );                /* Address of handler. */

xTaskCreate( DA2task,                       /* The function that implements the task. */
             ( const char * ) "DA",     /* Text name for the task, provided to assist debugging only. */
             configMINIMAL_STACK_SIZE,      /* The stack allocated to the task. */
             NULL,                          /* The task parameter is not used, so set to NULL. */
             tskIDLE_PRIORITY + 1,          /* The task runs at the idle priority. */
             &xDA2Task );                   /* Address of handler. */

    /* Create the queue used by the tasks. */
    xQueueData = xQueueCreate(  10,                     /* There are 5 spaces in the queue. */
                            sizeof( int ) );    /* Each space in the queue is large enough to hold a uint32_t. */

    /* Check the queue was created. */
    configASSERT( xQueueData );

/* Start the tasks and timer running. */
vTaskStartScheduler();

/* If all is well, the scheduler will now be running, and the following line
will never be reached.  If the following line does execute, then there was
insufficient FreeRTOS heap memory available for the idle and/or timer tasks
to be created.  See the memory management section on the FreeRTOS web site
for more details. */
for(;;){
```

Figure 7. Task and Queue Creation

AD1task

The AD1 task essentially takes every single line from the template program within main and copies them over to the AD1 task. This lab does not specify we needed to use both ADC inputs, so I only programmed channel 1, which made demoing my hardware easier. I implemented a queue to send the data from the ADC channel 1 and that brings us to the next task that will receive the data.

```
static void AD1task( void *pvParameters )
{   int adcdata1;   //ADC channel 1 data
    int adcdav;     //ADC data available
    int data;       //Data to be sent to the Queue

    Xil_Out32(AD1acq, 0);       //ADC stop acquire
    adcdav = Xil_In32(AD1dav);  //ADC available?
    while (adcdav == 1)
        adcdav = Xil_In32(AD1dav);

    while(1)
    {
        //ADC
        Xil_Out32(AD1acq, 1);       //ADC acquire
        while (adcdav == 0)     //ADC data available?
            adcdav = Xil_In32(AD1dav);
        Xil_Out32(AD1acq, 0);       //ADC stop acquire
        adcdata1 = Xil_In32(AD1dat1);   //input ADC data
        while (adcdav == 1)     //wait for reset
            adcdav = Xil_In32(AD1dav);

        data=adcdata1;
        /* Send the next value on the queue.  The queue should always be
           empty at this point so a block time of is used. */
        xQueueSend( xQueueData,           /* The queue being written to. */
                    &data,               /* The address of the data being sent. */
                    portMAX_DELAY );                    /* 0 sec block time. */
    }
}
```

Figure 8. AD1Task in FreeRTOS

DA2task

Coding DA2task was also done the same way as transforming the ADC code from C to FreeRTOS. Additionally, a queue receive line is created to collect the data from the AD1task that was sent to a queue, so this data will be outputted and converted back from a digital signal to an analog signal.

```
/*-----------------------------------------------------------*/
static void DA2task( void *pvParameters )
{   int Data;           //Data to be received from the Queue
    int dacdata1;       //DAC channel 1 data
    int dacdav;     //DAC data available

    Xil_Out32(DA2acq, 0);       //DAC stop acquire
    dacdav = Xil_In32(DA2dav);  //DAC available?
    while (dacdav == 1)
        dacdav = Xil_In32(DA2dav);

    while(1)
    {   xQueueReceive(xQueueData, &Data,portMAX_DELAY );

            dacdata1 = Data;        //ADC -> DAC pass through

            //DAC
            Xil_Out32(DA2dat1, dacdata1);   //output DAC data
            Xil_Out32(DA2acq, 1);       //DAC acquire
            while (dacdav == 0)     //DAC data output?
                dacdav = Xil_In32(DA2dav);
            Xil_Out32(DA2acq, 0);       //stop DAC acquire
            while (dacdav == 1)     //wait for reset
                dacdav = Xil_In32(DA2dav);

    }
}
```

Figure 9. DA2task in FreeRTOS

FreeRTOS Task 3 with SQRtask

This program is built off of the program used within the last task. The appendix can show the changes upon initialization which is that an SQRtask is defined, as well as an additional queue. The additional SQRtask and queue are created as follows:

```
/* Creating the three tasks with equal priority for ADC and DAC */
xTaskCreate(    AD1task,                    /* The function that implements the task. */
                ( const char * ) "AD",      /* Text name for the task, provided to assist debugging only. */
                configMINIMAL_STACK_SIZE,   /* The stack allocated to the task. */
                NULL,                       /* The task parameter is not used, so set to NULL. */
                tskIDLE_PRIORITY + 1,       /* The task runs at the idle priority. */
                &xAD1Task );                /* Address of handler. */

xTaskCreate( DA2task,                       /* The function that implements the task. */
             ( const char * ) "DA",         /* Text name for the task, provided to assist debugging only. */
             configMINIMAL_STACK_SIZE,      /* The stack allocated to the task. */
             NULL,                          /* The task parameter is not used, so set to NULL. */
             tskIDLE_PRIORITY + 1,          /* The task runs at the idle priority. */
             &xDA2Task );                   /* Address of handler. */

xTaskCreate( SQRtask,                           /* The function that implements the task. */
             ( const char * ) "SQR",            /* Text name for the task, provided to assist debugging only. */
             configMINIMAL_STACK_SIZE,          /* The stack allocated to the task. */
             NULL,                              /* The task parameter is not used, so set to NULL. */
             tskIDLE_PRIORITY + 1,              /* The task runs at the idle priority. */
             &xDA2Task );                       /* Address of handler. */

    /* Create the queues used by the tasks. */
    xQueueData = xQueueCreate(  10,             /* There are 10 spaces in the queue. */
                        sizeof( int ) );        /* Each space in the queue is large enough to hold a uint32_t. */
    //Queue for squaring the Analog signal
    xQueueSQR = xQueueCreate(   10,             /* There are 10 spaces in the queue. */
                                sizeof( int ) );    /* Each space in the queue is large enough to hold a uint32_t. */
    /* Check that the queues were created. */
    configASSERT( xQueueData );
    configASSERT( xQueueSQR );
```

Figure 10. Task and Queue Creation with SQRtask

The only other modifications to this program include the modifications to each task.

Modified AD1task

The only modifications within this task from the previous FreeRTOS program, is I created a temporary variable to collect the ADC data from channel 1 and it is now being sent to a queue to be received by the SQR task which will take care of the squaring. The rest of the AD1task remains the same, as seen in the appendix.

```
        data=adcdata1;
        //sending ADC data to the queue to be squared
        xQueueSend( xQueueData,           /* The queue being written to. */
                    &data,               /* The address of the data being sent. */
                    portMAX_DELAY);     /* Wait without a timeout for data. */
    }
}
```

Figure 11. Modification to AD1task

SQRtask

The SQR task refers to slide 14 in the *Zynq SPI Peripherals ADC and DAC* power point, which defines the conversion between computer units and voltage to be:

$$D[n-1:0] = G\,\frac{V_{IN} - V_{REF}}{V_{FS}}\,2^{n-1}$$

Figure 12. ADC equation from lecture

In this case, the input voltage is the ADC signal, with a reference to ground (Vref=0), since the ADC and DAC are 12bits, there will be a max of 4096 bits, and the gain is

determined by the DC voltage of 3.3V. Knowing this information the SQR task is programmed as follows: Receive the ADC voltage from the Queue, convert from computer units into volts, square the voltage, convert back into computer units by undoing the math, and finally send the data to a queue to be read by DA2task.

```
/*-----------------------------------------------------------*/
static void SQRtask( void *pvParameters )
{   int SQR_cpu, data;
    double data_volt, SQR_data;
    int G=3.3; //Gain in Volts
    int range=4095; //2^N where N=12, therefore range is 4096,
                    //but the range starts at 0 and ends at 4095

    while(1)
    {       xQueueReceive(xQueueData, &data,portMAX_DELAY );
            //convert the ADC data from Computer Units to Voltage
            data_volt=G*(double)data/range;
            //squaring the received data from the ADC
            SQR_data= data_volt*data_volt;
            //converting the Squared voltage back to computer units
            SQR_cpu=(int)(SQR_data*range/G);
            //sending the squared value to Queue to be received by DAC
            xQueueSend( xQueueSQR,      /* The queue being written to. */
                        &SQR_cpu,      /* The address of the data being sent. */
                        portMAX_DELAY);/* Wait without a timeout for data. */
    }
}
```

Figure 13. SQR Task

Modified DA2task

Finally, the DA2task is slightly modified from the previous FreeRTOS program, by receiving from the SQR queue now, and simply outputting the data through the DAC conversion. The rest of the task can be referred to within the appendix.

```
while(1)
{   xQueueReceive(xQueueSQR, &SQR_Data,portMAX_DELAY );
        //setting the DAC channel 1 to the SQR value received by the queue
        dacdata1 = SQR_Data;

        //DAC
        Xil_Out32(DA2dat1, dacdata1);   //output DAC data
        Xil_Out32(DA2acq, 1);       //DAC acquire
        while (dacdav == 0)     //DAC data output?
            dacdav = Xil_In32(DA2dav);
        Xil_Out32(DA2acq, 0);       //stop DAC acquire
        while (dacdav == 1)     //wait for reset
            dacdav = Xil_In32(DA2dav);

}
```

Figure 14. Modified DA2task

Verification

Waveforms

For consistency, I ran the same waveform in both channels for all three tasks to examine the original analog signal in channel 2, with the converted signal in channel 1.

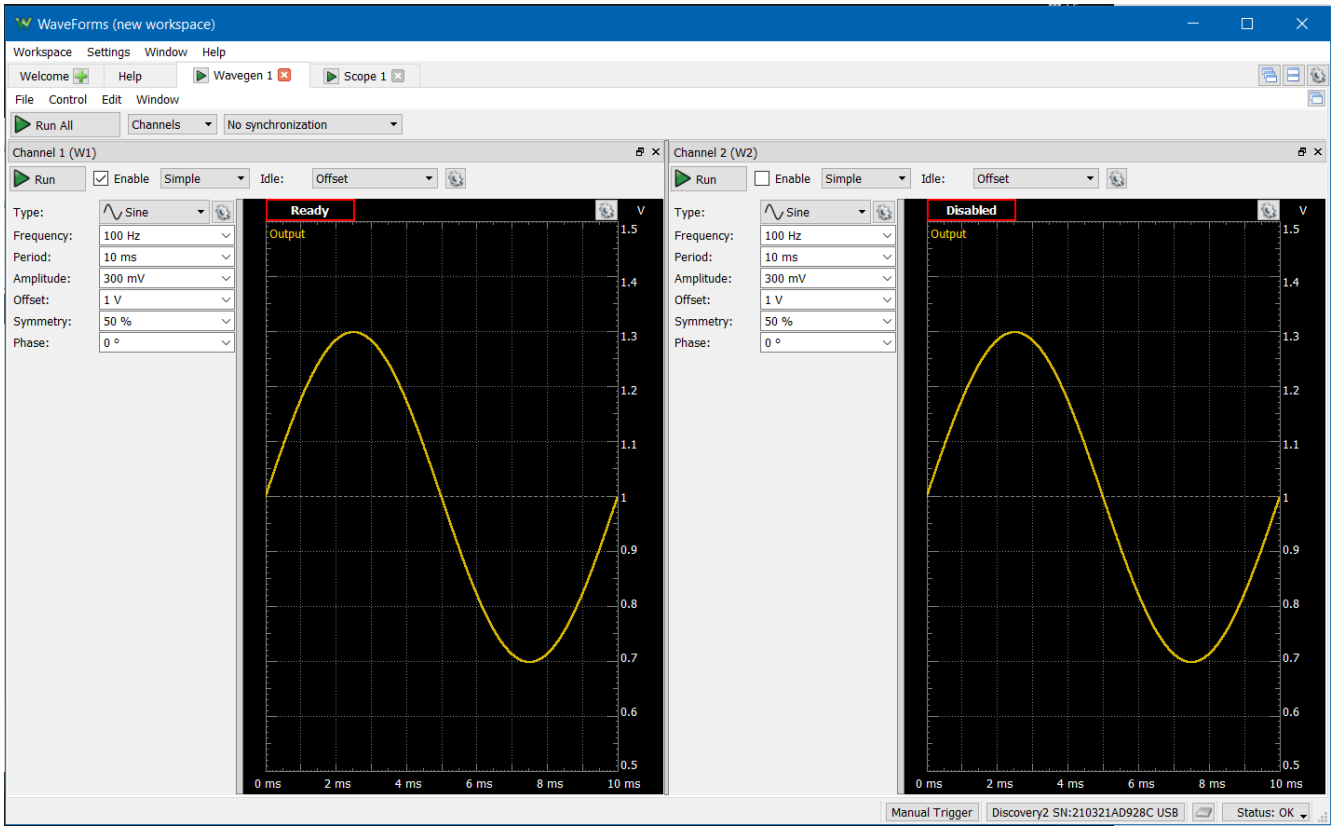


Figure 16. Input Waveforms

Verification using the C Template

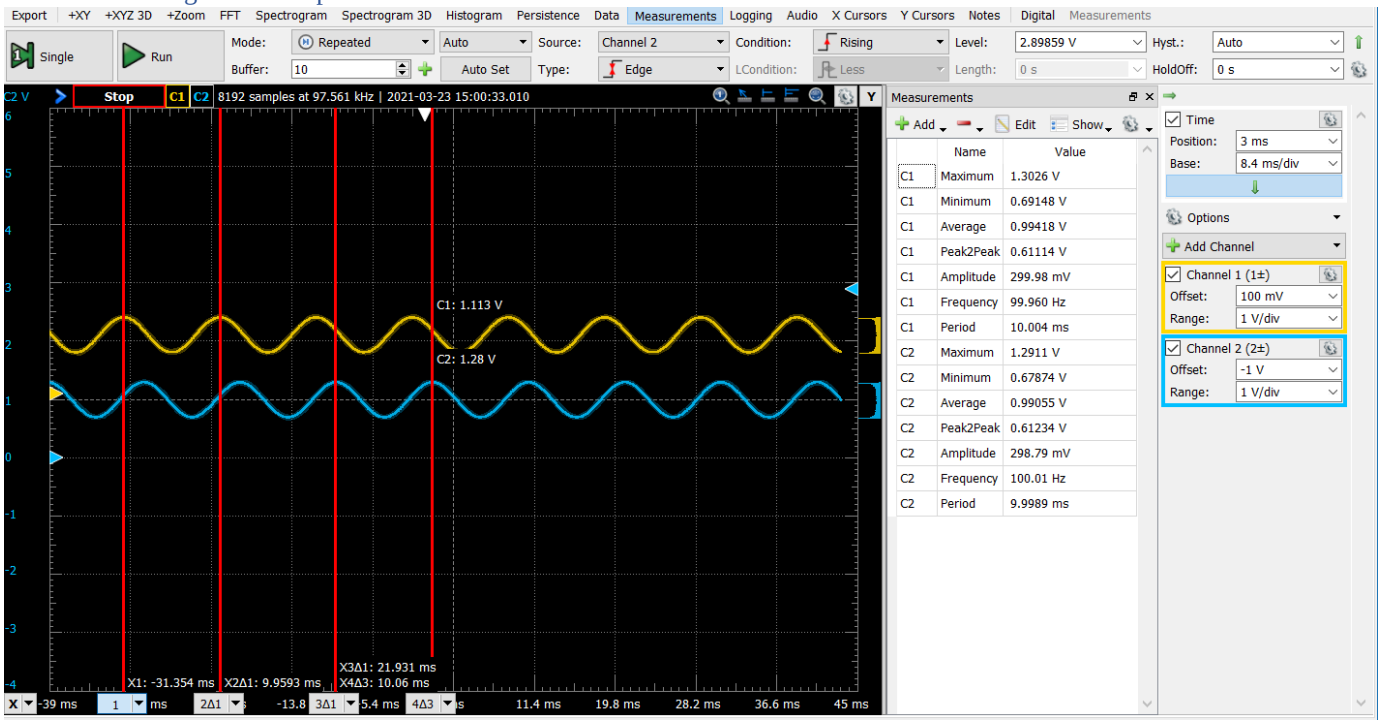
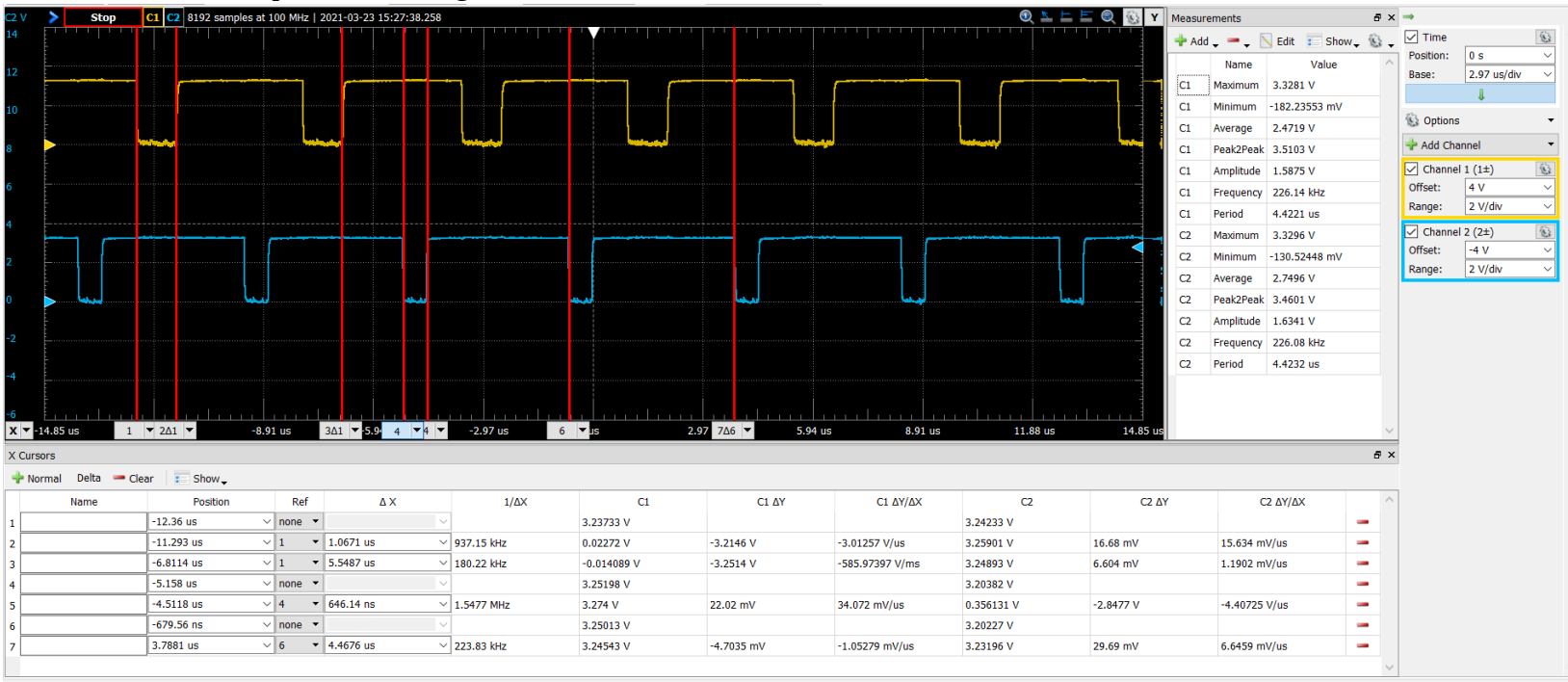


Figure 17. Confirming the ADC to DAC works by comparing the converted signal CH1 to analog signal CH2

Notice that the peaks are as expected and the period between each peak matches the oscilloscope's measurement of approximately at 10ms, generating a frequency of about 100Hz. Furthermore, despite their offsets, both waves contain an amplitude of approximately 300mV which is what I set the waveform to generate, so the program does act as a buffer between ADC to DAC.

Chip Select reading:



	Name	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX	C2	C2 ΔY	C2 ΔY/ΔX
1		-12.36 us	none			3.23733 V			3.24233 V		
2		-11.293 us	1	1.0671 us	937.15 kHz	0.02272 V	-3.2146 V	-3.01257 V/us	3.25901 V	16.68 mV	15.634 mV/us
3		-6.8114 us	1	5.5487 us	180.22 kHz	-0.014089 V	-3.2514 V	-585.97397 V/ms	3.24893 V	6.604 mV	1.1902 mV/us
4		-5.158 us	none			3.25198 V			3.20382 V		
5		-4.5118 us	4	646.14 ns	1.5477 MHz	3.274 V	22.02 mV	34.072 mV/us	0.356131 V	-2.8477 V	-4.40725 V/us
6		-679.56 ns	none			3.25013 V			3.20227 V		
7		3.7881 us	6	4.4676 us	223.83 kHz	3.24543 V	-4.7035 mV	-1.05279 mV/us	3.23196 V	29.69 mV	6.6459 mV/us

Figure 18. Chip Select Reading for Template Program

Examining the chip select, the sample frequency for both ADC and DAC is approximately 226.08kHz, generating a sample time of 4.4221us, and upon examining the periods of lows, the ADC holds a period of around 1.0671us, while the DAC holds a period of about 646.14ns.

Verification using FreeRTOS task2

Upon transforming the sample code into a FreeRTOS implementation using a queue, the period between each peak matches the oscilloscope's measurement of approximately at 9.9998ms or 10ms, generating a frequency of 100Hz. Both waves hold an amplitude of approximately 300mV which verifies that this program also acts as a buffer between the ADC and DAC.

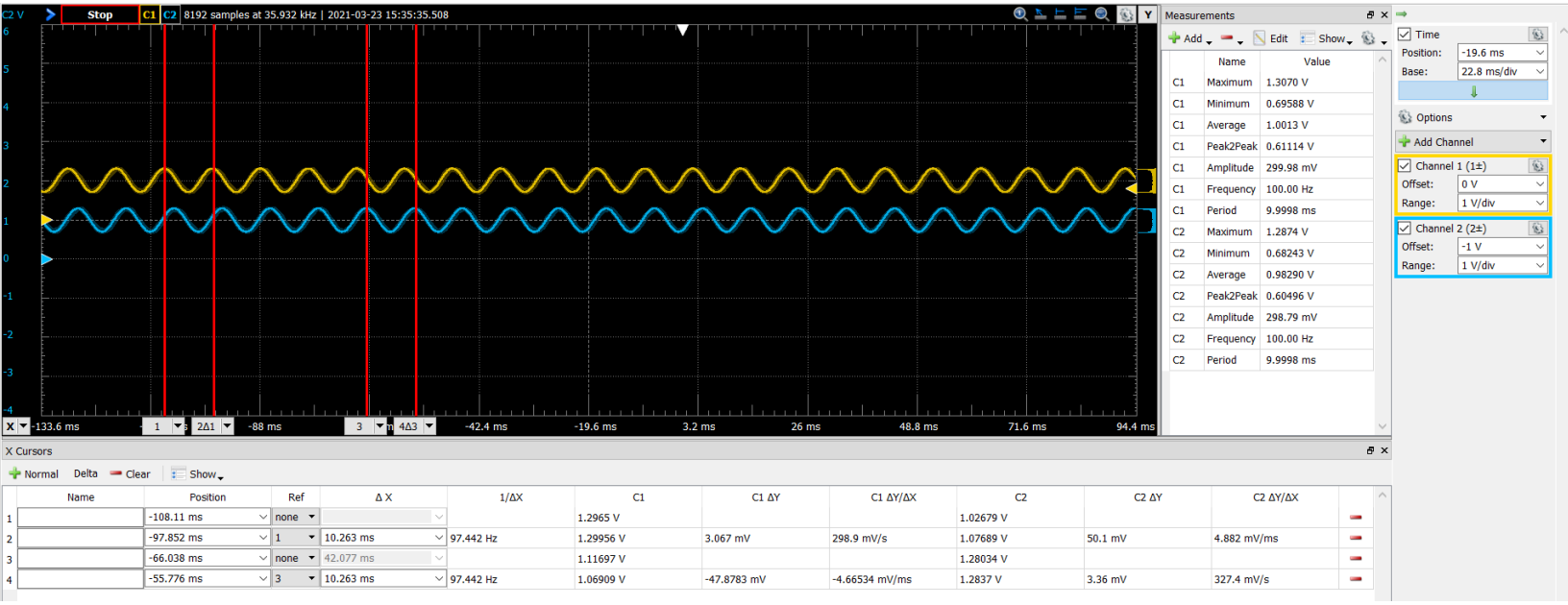


Figure 19. Confirming the ADC to DAC works by comparing the converted signal CH1 to analog signal CH2

Chip Select Reading:

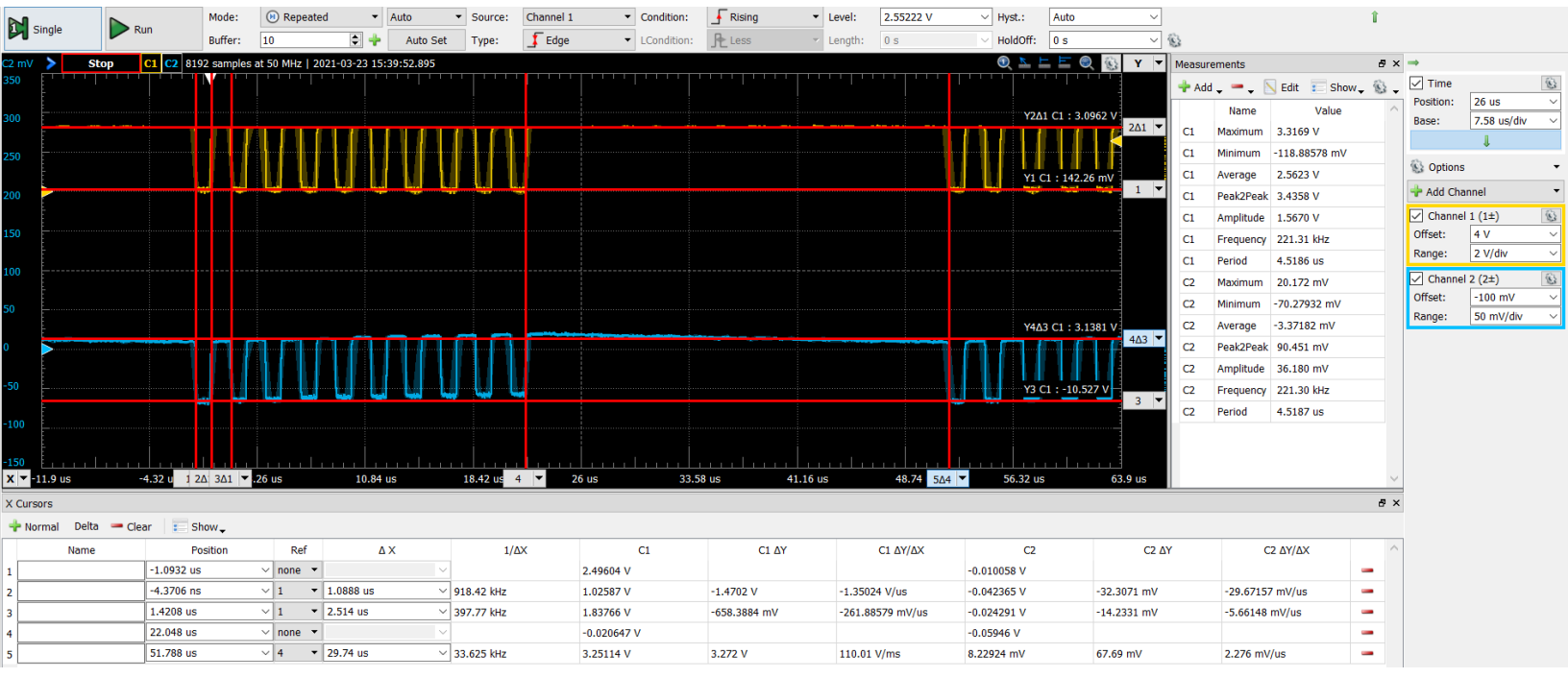


Figure 20. Chip Select Reading of FreeRTOS ADC to DAC task

Confirming these results, the period remains at approximately 10ms, generating 100Hz, and examining the max reading of 1.5621V from the converted signal, the square root of 1.5621V is approximately 1.2498V. Of course there is some slight percentage error since this is experimentally done

Verification using FreeRTOS task3

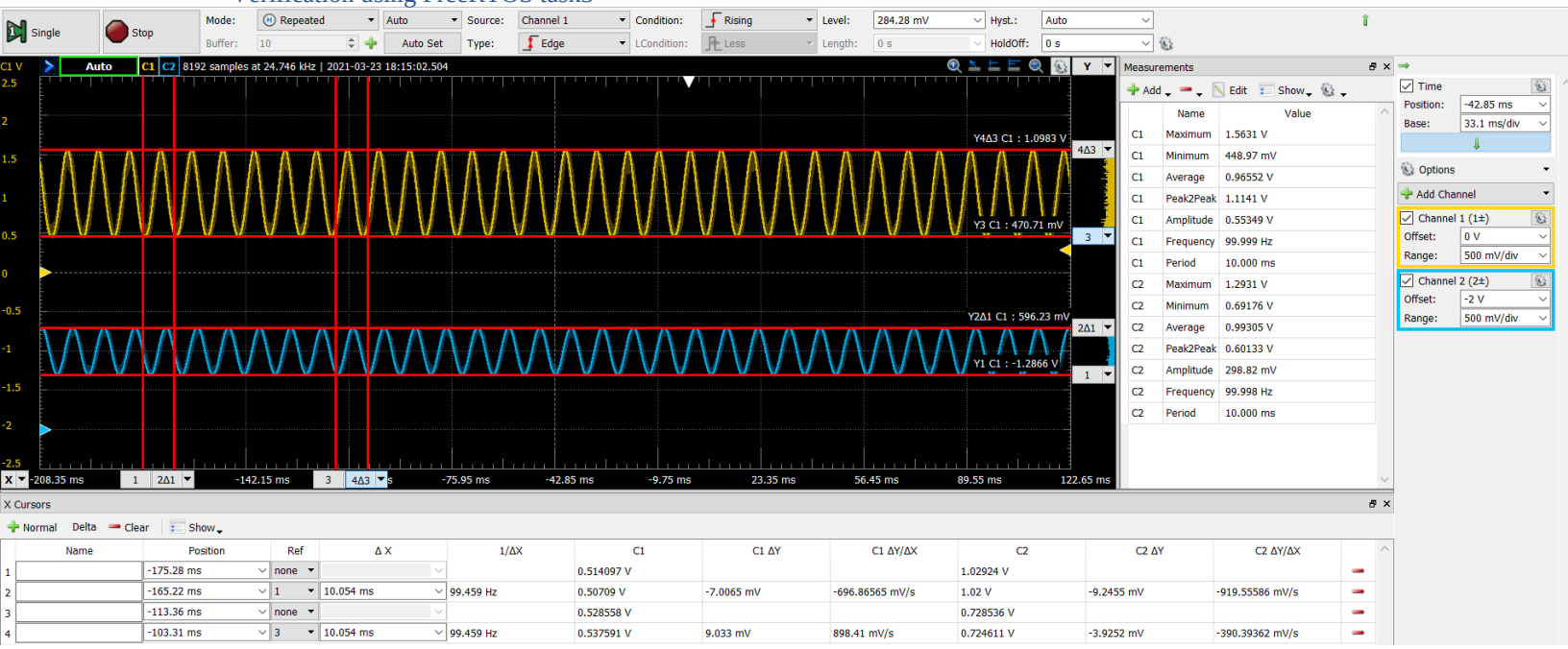
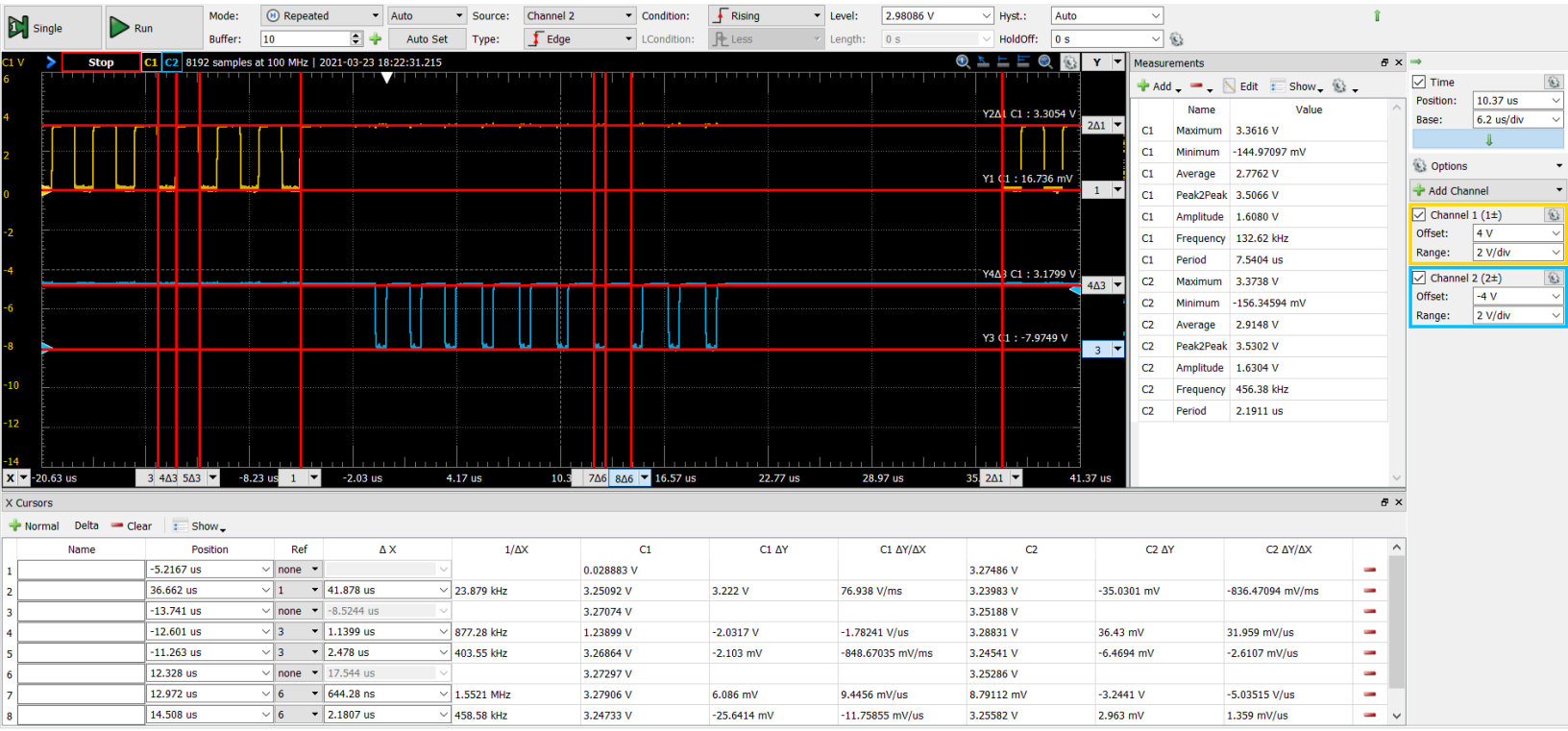


Figure 21. Confirming the ADC signal is Squared and sent to DAC by comparing the converted signal CH1 to analog signal CH2

Chip Select Reading:



	Name	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX	C2	C2 ΔY	C2 ΔY/ΔX
1		-5.2167 us	none			0.028883 V			3.27486 V		
2		36.662 us	1	41.878 us	23.879 kHz	3.25092 V	3.222 V	76.938 V/ms	3.23983 V	-35.0301 mV	-836.47094 mV/ms
3		-13.741 us	none	-8.5244 us		3.27074 V			3.25188 V		
4		-12.601 us	3	1.1399 us	877.28 kHz	1.23899 V	-2.0317 V	-1.78241 V/us	3.28831 V	36.43 mV	31.959 mV/us
5		-11.263 us	3	2.478 us	403.55 kHz	3.26864 V	-2.103 mV	-848.67035 mV/ms	3.24541 V	-6.4694 mV	-2.6107 mV/us
6		12.328 us	none	17.544 us		3.27297 V			3.25286 V		
7		12.972 us	6	644.28 ns	1.5521 MHz	3.27906 V	6.086 mV	9.4456 mV/us	8.79112 mV	-3.2441 V	-5.03515 V/us
8		14.508 us	6	2.1807 us	458.58 kHz	3.24733 V	-25.6414 mV	-11.75855 mV/us	3.25582 V	2.963 mV	1.359 mV/us

Figure 22. Chip Select Reading of FreeRTOS ADC to DAC task with Squaring

Upon reading the chip select, the y-cursors prove the voltage of 3.3v, while the ADC has a low period of 1.139us, and the DAC operates with a low period of 644.28ns.

Video Link Verification
https://youtu.be/wk56IoL1Hd0

Conclusion

This lab solidified everything we have been leading up to within the past few weeks such as task and queue management in FreeRTOS, and ADC to DAC conversion. Utilizing these methods, will expand upon future DSP projects such as the creation of a digital filter. Upon the completion of this lab, I was able to properly square the voltage and output the waves of through ADC to DAC conversion, and confirm this through Waveforms' oscilloscope.

Appendix

FreeRTOS task 2

```
/*
    Copyright (C) 2017 Amazon.com, Inc. or its affiliates.  All Rights Reserved.
    Copyright (C) 2012 - 2018 Xilinx, Inc. All Rights Reserved.

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software. If you wish to use our Amazon
    FreeRTOS name, please do so in a fair use way that does not cause confusion.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

    http://www.FreeRTOS.org
```

```
    http://aws.amazon.com/freertos


    1 tab == 4 spaces!
*/
//Robert Bara Lab 7 ECE3623 Spring 2021
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
#include "sleep.h"
//definitions
#define printf xil_printf
/*-----------------------------------------------------------*/
/* The tasks as described at the top of this file. */
static void AD1task( void *pvParameters );
static void DA2task( void *pvParameters );
/*-----------------------------------------------------------*/
/* Defining Task Handles */
static TaskHandle_t xAD1Task;
static TaskHandle_t xDA2Task;
/* The queue used by the tasks, as described at the top of this
file. */
```

```
static QueueHandle_t xQueueData = NULL;

//AD1Pmod from Address Editor in Vivado, first IP
#define AD1acq      0x43C00000   //AD1 acquisition    - output
#define AD1dav      0x43C00004   //AD1 data available - input
#define AD1dat1     0x43C00008   //AD1 channel 1 data - input
#define AD1dat2     0x43C0000C   //AD1 channel 2 data - input

//DAC2Pmod from Address Editor in Vivado, second IP
#define DA2acq  0x43C10000       //DA2 acquisition    - output
#define DA2dav      0x43C10004   //DA2 data available - input
#define DA2dat1     0x43C10008   //DA2 channel 1 data - output
#define DA2dat2 0x43C1000C       //DA2 channel 2 data - output

//Main Function
int main( void ){

        xil_printf("\n\rStarting AD1-DA2 Pmod demo test...\n");

    /* Creating the two tasks with equal priority for ADC and DAC */
    xTaskCreate(   AD1task,                         /* The function that
implements the task. */
                    ( const char * ) "AD",          /* Text name
for the task, provided to assist debugging only. */
                    configMINIMAL_STACK_SIZE,       /* The stack
allocated to the task. */
                    NULL,
    /* The task parameter is not used, so set to NULL. */
                    tskIDLE_PRIORITY + 1,           /* The task
runs at the idle priority. */
```

```
                                    &xAD1Task );                    /*
Address of handler. */


        xTaskCreate( DA2task,                                       /* The
function that implements the task. */
                                    ( const char * ) "DA",          /* Text name for the
task, provided to assist debugging only. */
                                    configMINIMAL_STACK_SIZE,       /* The stack
allocated to the task. */
                                    NULL,
        /* The task parameter is not used, so set to NULL. */
                                    tskIDLE_PRIORITY + 1,           /* The task
runs at the idle priority. */
                                    &xDA2Task );                    /*
Address of handler. */


                /* Create the queue used by the tasks. */
                xQueueData = xQueueCreate(          10,
        /* There are 5 spaces in the queue. */
                                                    sizeof( int ) );  /* Each space
in the queue is large enough to hold a uint32_t. */


                /* Check the queue was created. */
                configASSERT( xQueueData );


        /* Start the tasks and timer running. */
        vTaskStartScheduler();


        /* If all is well, the scheduler will now be running, and the following line
        will never be reached.  If the following line does execute, then there was
        insufficient FreeRTOS heap memory available for the idle and/or timer tasks
```

```
        to be created.  See the memory management section on the FreeRTOS web site
        for more details. */
        for(;;){
}
}

/*-----------------------------------------------------------*/
static void AD1task( void *pvParameters )
{       int adcdata1;   //ADC channel 1 data
        int adcdav;             //ADC data available
        int data;               //Data to be sent to the Queue

        Xil_Out32(AD1acq, 0);           //ADC stop acquire
        adcdav = Xil_In32(AD1dav); //ADC available?
        while (adcdav == 1)
                adcdav = Xil_In32(AD1dav);

        while(1)
        {
                //ADC
                Xil_Out32(AD1acq, 1);           //ADC acquire
                while (adcdav == 0)             //ADC data available?
                        adcdav = Xil_In32(AD1dav);
                Xil_Out32(AD1acq, 0);           //ADC stop acquire
                adcdata1 = Xil_In32(AD1dat1);       //input ADC data
                while (adcdav == 1)             //wait for reset
                        adcdav = Xil_In32(AD1dav);
```

```
                data=adcdata1;
                /* Send the next value on the queue.  The queue should always be
                   empty at this point so a block time of is used. */
                xQueueSend( xQueueData,                 /* The queue being written
to. */
                                        &data,                  /* The address of the
data being sent. */
                                        portMAX_DELAY );
        /* 0 sec block time. */
        }
}

/*-----------------------------------------------------------*/
static void DA2task( void *pvParameters )
{       int Data;                       //Data to be received from the Queue
        int dacdata1;           //DAC channel 1 data
        int dacdav;             //DAC data available

        Xil_Out32(DA2acq, 0);                   //DAC stop acquire
        dacdav = Xil_In32(DA2dav); //DAC available?
        while (dacdav == 1)
                dacdav = Xil_In32(DA2dav);

        while(1)
        {       xQueueReceive(xQueueData, &Data,portMAX_DELAY );

                        dacdata1 = Data;                //ADC -> DAC pass through

                        //DAC
```

```
                    Xil_Out32(DA2dat1, dacdata1);       //output DAC data
                    Xil_Out32(DA2acq, 1);               //DAC acquire
                    while (dacdav == 0)             //DAC data output?
                            dacdav = Xil_In32(DA2dav);
                    Xil_Out32(DA2acq, 0);               //stop DAC acquire
                    while (dacdav == 1)             //wait for reset
                            dacdav = Xil_In32(DA2dav);


            }
}
/*-----------------------------------------------------------*/
```

FreeRTOS task3 with *SQRtask*

```
/*
    Copyright (C) 2017 Amazon.com, Inc. or its affiliates.  All Rights Reserved.
    Copyright (C) 2012 - 2018 Xilinx, Inc. All Rights Reserved.

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software. If you wish to use our Amazon
    FreeRTOS name, please do so in a fair use way that does not cause confusion.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
```

*/

```
//Robert Bara Lab 7 ECE3623 Spring 2021
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
//#include "platform.h"
#include "sleep.h"
//definitions
```

```
#define printf xil_printf
/*-----------------------------------------------------------*/
/* The tasks as described at the top of this file. */
static void AD1task( void *pvParameters );
static void DA2task( void *pvParameters );
static void SQRtask( void *pvParameters );
/*-----------------------------------------------------------*/
/* Defining Task Handles */
static TaskHandle_t xAD1Task;
static TaskHandle_t xDA2Task;
static TaskHandle_t xSQRTask;
/* The queue used by the tasks, as described at the top of this
file. */
static QueueHandle_t xQueueData = NULL;
static QueueHandle_t xQueueSQR = NULL;

//AD1Pmod from Address Editor in Vivado, first IP
#define AD1acq        0x43C00000   //AD1 acquisition    - output
#define AD1dav        0x43C00004   //AD1 data available - input
#define AD1dat1       0x43C00008   //AD1 channel 1 data - input
#define AD1dat2       0x43C0000C   //AD1 channel 2 data - input

//DAC2Pmod from Address Editor in Vivado, second IP
#define DA2acq  0x43C10000         //DA2 acquisition    - output
#define DA2dav        0x43C10004   //DA2 data available - input
#define DA2dat1       0x43C10008   //DA2 channel 1 data - output
#define DA2dat2 0x43C1000C         //DA2 channel 2 data - output
```

```
//Main Function
int main( void ){

        xil_printf("\n\rStarting AD1-DA2 Pmod demo test...\n");

    /* Creating the three tasks with equal priority for ADC and DAC */
    xTaskCreate(   AD1task,                          /* The function that implements the task. */
                        ( const char * ) "AD",              /* Text name for the task, provided to assist debugging only. */
                        configMINIMAL_STACK_SIZE,   /* The stack allocated to the task. */
                        NULL,
    /* The task parameter is not used, so set to NULL. */
                        tskIDLE_PRIORITY + 1,       /* The task runs at the idle priority. */
                        &xAD1Task );                        /* Address of handler. */

    xTaskCreate( DA2task,                            /* The function that implements the task. */
                        ( const char * ) "DA",              /* Text name for the task, provided to assist debugging only. */
                        configMINIMAL_STACK_SIZE,   /* The stack allocated to the task. */
                        NULL,
    /* The task parameter is not used, so set to NULL. */
                        tskIDLE_PRIORITY + 1,       /* The task runs at the idle priority. */
```

```
                                    &xDA2Task );                                    /*
Address of handler. */


        xTaskCreate( SQRtask,                                                       /* The
function that implements the task. */

                                    ( const char * ) "SQR",                 /* Text name
for the task, provided to assist debugging only. */

                                    configMINIMAL_STACK_SIZE,               /* The
stack allocated to the task. */

                                    NULL,
        /* The task parameter is not used, so set to NULL. */

                                    tskIDLE_PRIORITY + 1,                   /* The
task runs at the idle priority. */

                                    &xDA2Task );
        /* Address of handler. */


                /* Create the queues used by the tasks. */

                xQueueData = xQueueCreate(          10,                             /*
There are 10 spaces in the queue. */

                                                        sizeof( int ) );  /* Each space
in the queue is large enough to hold a uint32_t. */

                //Queue for squaring the Analog signal

                xQueueSQR = xQueueCreate(           10,                             /*
There are 10 spaces in the queue. */

                                                                    sizeof( int ) );
        /* Each space in the queue is large enough to hold a uint32_t. */

                /* Check that the queues were created. */

                configASSERT( xQueueData );

                configASSERT( xQueueSQR );


        /* Start the tasks and timer running. */

        vTaskStartScheduler();
```

```
	/* If all is well, the scheduler will now be running, and the following line
	will never be reached.  If the following line does execute, then there was
	insufficient FreeRTOS heap memory available for the idle and/or timer tasks
	to be created.  See the memory management section on the FreeRTOS web site
	for more details. */
	for(;;){
}
}
/*-----------------------------------------------------------*/
static void AD1task( void *pvParameters )
{	int adcdata1;	//ADC channel 1 data
	int adcdav;		//ADC data available
	int data;		//Data to be sent to the Queue

	Xil_Out32(AD1acq, 0);		//ADC stop acquire
	adcdav = Xil_In32(AD1dav); //ADC available?
	while (adcdav == 1)
		adcdav = Xil_In32(AD1dav);

	while(1)
	{
		//ADC
		Xil_Out32(AD1acq, 1);		//ADC acquire
		while (adcdav == 0)		//ADC data available?
			adcdav = Xil_In32(AD1dav);
		Xil_Out32(AD1acq, 0);		//ADC stop acquire
		adcdata1 = Xil_In32(AD1dat1);	//input ADC data
```

```
		while (adcdav == 1)		//wait for reset
			adcdav = Xil_In32(AD1dav);

		data=adcdata1;
		//sending ADC data to the queue to be squared
		xQueueSend( xQueueData,			/* The queue being written to. */
					&data,			/* The address of the data being sent. */
					portMAX_DELAY);		/* Wait without a timeout for data. */
	}
}

/*-----------------------------------------------------------*/
static void DA2task( void *pvParameters )
{	int SQR_Data;			//Square'd Data to be received from the Queue
	int dacdata1;		//DAC channel 1 data
	int dacdav;		//DAC data available

	Xil_Out32(DA2acq, 0);			//DAC stop acquire
	dacdav = Xil_In32(DA2dav); //DAC available?
	while (dacdav == 1)
		dacdav = Xil_In32(DA2dav);

	while(1)
	{	xQueueReceive(xQueueSQR, &SQR_Data,portMAX_DELAY );
			//setting the DAC channel 1 to the SQR value received by the queue
			dacdata1 = SQR_Data;
```

```
                        //DAC
                        Xil_Out32(DA2dat1, dacdata1);       //output DAC data
                        Xil_Out32(DA2acq, 1);               //DAC acquire
                        while (dacdav == 0)             //DAC data output?
                                dacdav = Xil_In32(DA2dav);
                        Xil_Out32(DA2acq, 0);               //stop DAC acquire
                        while (dacdav == 1)             //wait for reset
                                dacdav = Xil_In32(DA2dav);


        }
}
/*-----------------------------------------------------------*/
static void SQRtask( void *pvParameters )
{       int SQR_cpu, data;
        double data_volt, SQR_data;
        int G=3.3; //Gain in Volts
        int range=4095; //2^N where N=12, therefore range is 4096,
                                //but the range starts at 0 and ends at 4095

        while(1)
        {               xQueueReceive(xQueueData, &data,portMAX_DELAY );
                        //convert the ADC data from Computer Units to Voltage
                        data_volt=G*(double)data/range;
                        //squaring the received data from the ADC
                        SQR_data= data_volt*data_volt;
                        //converting the Squared voltage back to computer units
                        SQR_cpu=(int)(SQR_data*range/G);
```

```
                    //sending the squared value to Queue to be received by DAC
                    xQueueSend( xQueueSQR,        /* The queue being written to. */
                                          &SQR_cpu,      /* The address of the data
being sent. */
                                          portMAX_DELAY);/* Wait without a
timeout for data. */
          }
}
```