Lab 8 AD1Pmod and DA2Pmod in FreeRTOS

Christopher Antochy

[*tuh03116@temple.edu*](mailto:tuh03116@temple.edu)

**Summary**

This lab involves using queues and tasks with FreeRTOS on the Zybo board with a PModAD1 and PModDA2 attached to an EE board. A sample template is used for making the EE board generate a wave, sending it through the ADC to the board where it gets output on the DAC, and then making the EE board capture the signal on an oscilloscope which reflects the initial generated wave. Tasks in this lab will further push the code to utilize the FreeRTOS to control the PMod device readings and even insert an FIR filter. The tasks in this lab successfully operate to show how code changes affect data rate performance as well some noise that is present.

**Introduction**

In this lab, the hardware design for this lab requires the use of the PModAD1 and the PModDA2. These are the functioning peripherals required for the desired setup of taking an input analog signal, converting it to digital, having the board read and then sending it to the DAC to transfer signal back into its analog form. The hardware design made in Vivado is shown in Figure 1.

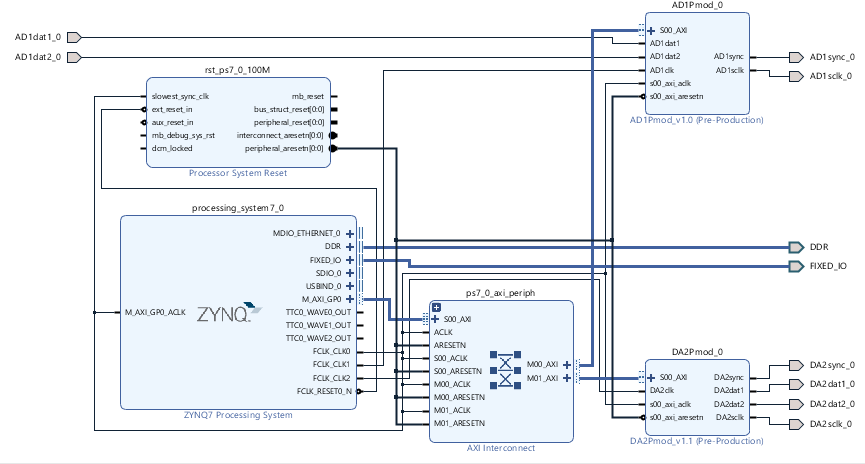


Figure 1 Hardware Design

After completion of the design, the Xilinx SDK is used to code into the Zybo board. The template code in Appendix 1 is a code designed to feed an analog signal to the ADC and convert it back through the DAC. The lab is then divided into following tasks.

Task 1: Describe in detail the single task, standalone operating system template application. What are the various control and status signals? Using the Waveforms oscilloscope, measure the sampled data throughput rate beginning at an initial ADC data

conversion (ADC SS high to active low) to the beginning of the next ADC data conversion (ADC SS high to active low). Show the measurements and report the result as samples/second fs. Measure the time for each of the components of this data acquisition sequence. That is, the measured time for ADC conversion (ADC SS active low to high), DAC conversion (DAC SS active low to high) and the remaining time as overhead of the single task application.

Task 2: Next reconfigure this single task, standalone operating system application as two tasks, AD1task and DA2task, within FreeRTOS. You are to use a queue function to transfer data for straight-through operation between the two tasks in sequence AD1task > DACtask. Repeat the measurements of the sampled data throughput rate as given in Task 1. Comment on the performance difference between Task 1 and Task 2.

Task 3: The second Laboratory task is to insert a third task FIRtask as the digital filter with FreeRTOS. The FIRtask will also uses a control and status signal (not shown) to coordinate data transfers. The task sequence then is AD1task > FIRtask > DACtask. The FIR filter is that which you implemented in Lab 5. Repeat the measurements of the sampled data throughput rate as given in Task 1. Comment on the performance difference between Task 2 and Task 3.

You are to verify the frequency response of your FIR filter by direct measurements using Waveforms and compare the results to the calculated frequency response from the magnitude of the transfer function | H(z) |. Note that here, unlike Lab 5, the actual sampling rate fs is measured and used in the calculation for the frequency response where the sampling period Ts = 1 / fs.

**Discussion**

Task 1: The template code in Appendix 1 is used for this section. It sends an ADC acquisition, waits for data to become available, reads the ADC data into a variable, waits for the data to reset, then sends ADC data to the DAC through a variable. The DAC sends out the data, checks if it’s been acquired, then waits for a reset. After, the ADC is looped back into and the cycle repeats. These control functions are important for securing the operation of each PMod without error during retrieval and sending of signal data.

The graphs below represent capture of the data rates of the ADC and DAC chip select.

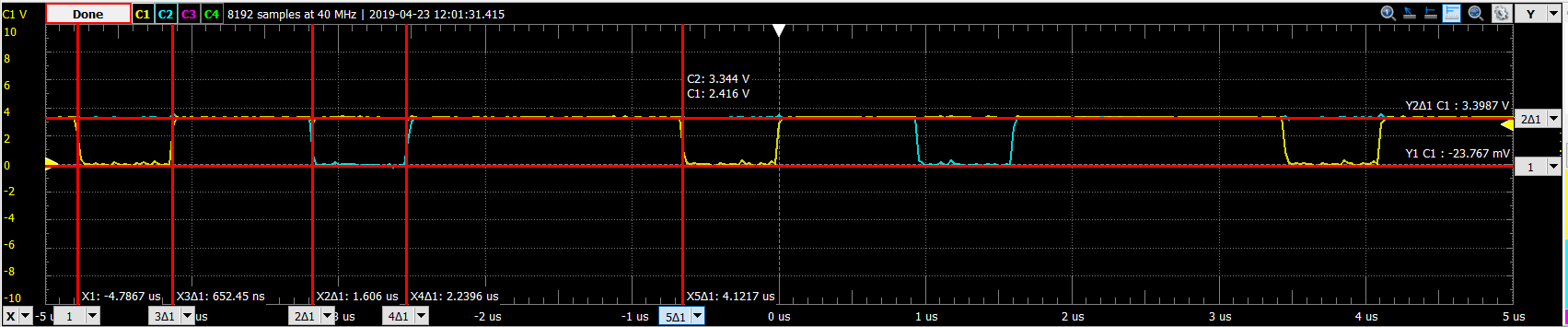


Figure 2 ADC & DAC data rate Task 1

The sample frequency for both the DAC and ADC is about 242kHz or 4.12 us sample time. The low period for the DAC is 0.634 us and the low period for the ADC is 0.652 us.

Appendix 4 shows the implementation in action on the hardware.

Task 2: The code for this section in found in Appendix 2. It uses the FreeRTOS task system and queue system to send the ADC data into the DAC data. The ADC task has higher priority and prepares the ADC and retrieves the digital value. Then it uses xQueueSend to send the data to the DAC. Then it loops around to the do the same but will block when it notices the queue is full. The DAC task will retrieve the data in the queue with xQueueRetrieve and send the data to the DAC. Then it loops back around but blocks when the queue to receive is empty. The queue allow blocking so that each task can perform it’s correct operation. The Figure 3 below shows the scope capture of the chip selects for the ADC and DAC.

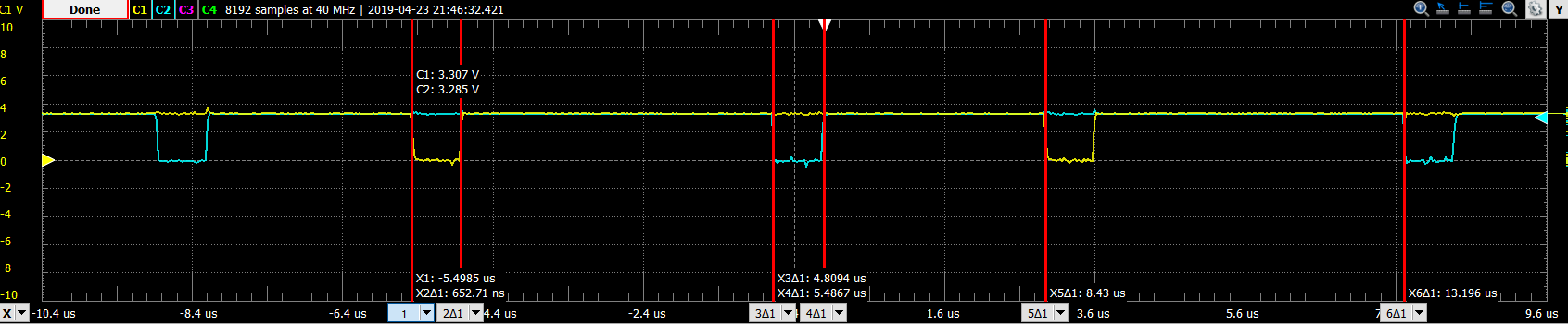


Figure ADC and DAC data rate Task 2

The ADC remains low around 0.653 us and DAC around 0.677 us. The sample time of the ADC and DAC are about 8.43us which is about twice the sample period of the Task 1. The implementation of the hardware and code can be seen in Appendix 4.

Task 3: The code for this section is found in Appendix 3. It uses an additional queue and task to implement an FIR filter such that a data sample should follow the following path. First, the ADC will retrieve the data from the wave generator and send it to queue1. Then it will read again but get blocked when it tries to send to queue1 again. The second task is the FIR filter which has the second highest priority. It first retrieves the content of queue 1, then the ADC repeats and blocks itself. The FIR continues to implement a pushdown table for the filter values, and then proceeds to send the filtered value to queue2. It will then loop to the top of it’s task where it looks to retrieve from queue1 again and ADC subsequently sends another to the queue1. FIR then filters this value but it is blocked on sending to queue2. The third task is the DAC which retrieves from queue2, the other tasks will then try to go again until FIR hits block on send to queue2 again and DAC continues to send the first value through the DAC channel and then this process repeats in perpetuity. The Figure 4 below shows the data rates of the chip selects of the ADC and DAC.

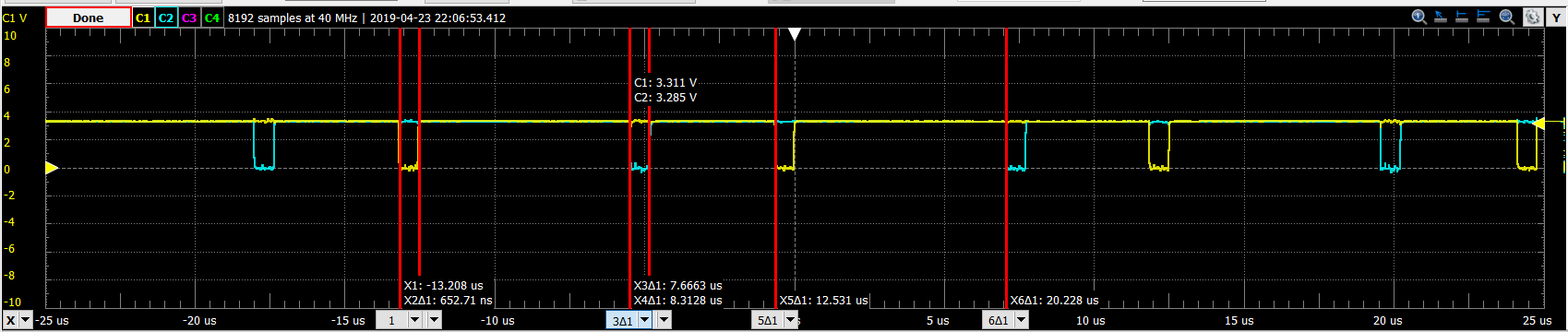


Figure DAC and ADC data rate for Task 3

The lows for the ADC and DAC still remain around 0.652 us but the periods themselves increased to 12.53 us which is 3 times the rate of Task 1 and half times an increase of Task 2. The Figure 5 below shows the capture of the ADC and DAC A0 pins

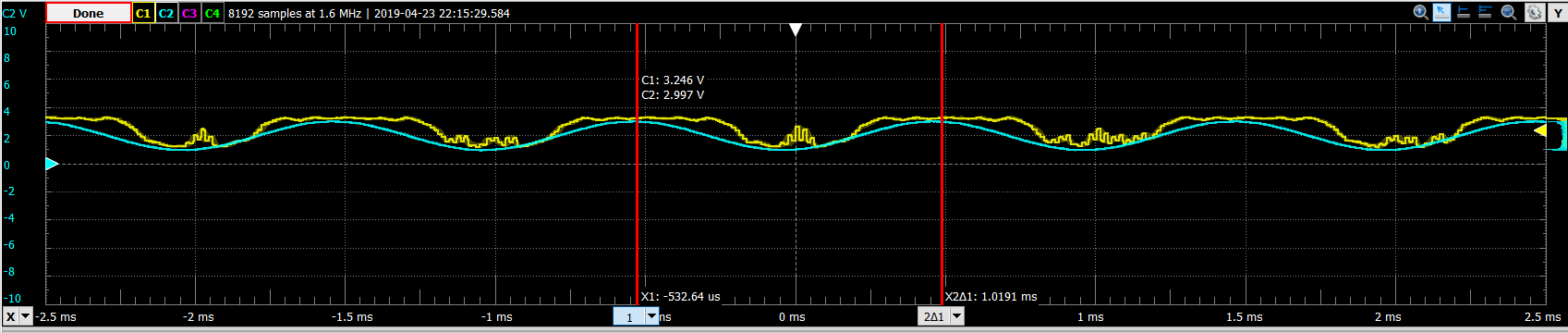


Figure ADC input and DAC output for Task 3

The code for the filter also divides itself by the gain of the frequency response function so the result is what should be a stabilized replica of the ADC input for the DAC. Unfortunately, by setup had a lot of noise and so the values look very fluctuated. However, the general shape of the sinusoid from the input is still present in the output. The implementation can be found in Appendix 4.

**Conclusion**

This lab is an introduction to implementing some DSP using the Zybo board and the FreeRTOS operating system. The PModAD1 and PModDA2 are used to read an input analog signal and output the corresponding signal. The tasks in this lab explore utilizing the FreeRTOS task management system and queue system to implement an original code piece that did this but also expands the implementation to include an FIR filter. The data rates of the ADC and DAC with the modifications show a general increase in sampling time to accommodate for the overhead found from the auxiliary code functions and the FreeRTOS themselves. It is also noticed that noise is present in raw output from the DAC itself.

**Appendix**

1. Template Code

//AD1PmodEx PmodAD1 ADC example ECE3622 c2019 Dennis Silage

**#include** "xparameters.h"

**#include** "xil\_io.h"

**#include** <stdio.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

**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);

**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);

}

}

1. FreeRTOS Task ADC/DAC Code

/\* FreeRTOS includes. \*/

**#include** "FreeRTOS.h"

**#include** "task.h"

**#include** "queue.h"

/\* Xilinx \*/

**#include** "xparameters.h"

**#include** "xil\_io.h"

**#include** <stdio.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

**static** **void** **AD1Task**( **void** \*pvParameters );

**static** **void** **DA2Task**( **void** \*pvParameters );

TaskHandle\_t xAD1Task;

TaskHandle\_t xDA2Task;

//TaskHandle\_t xFIRTask;

QueueHandle\_t xQueue1 = NULL;

//QueueHandle\_t xQueue2 = NULL;

**int** **main**(**void**)

{

**int** array[2];

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 );

xTaskCreate( DA2Task,

( **const** **char** \* ) "DA",

configMINIMAL\_STACK\_SIZE,

NULL,

tskIDLE\_PRIORITY+3,

&xDA2Task );

//xTaskCreate( FIRTask,

//configMINIMAL\_STACK\_SIZE,

//NULL,

//tskIDLE\_PRIORITY+2,

//&xFIRTask)

xQueue1 = xQueueCreate( 1, /\* There is only one space in the queue. \*/

**sizeof**( array ) ); /\* Each space in the queue is large enough to hold a uint32\_t. \*/

configASSERT( xQueue1 );

//xQueue2 = xQueueCreate( 1,

//sizeof( array ) );

/\* Check the queue was created. \*/

//configASSERT( xQueue2 );

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

vTaskStartScheduler();

**while** (1){};

}

/\*-----------------------------------------------------------\*/

**static** **void** **AD1Task**( **void** \*pvParameters )

{

**int** adcdav; //ADC data available

**int** adcdata[2]; //ADC channel data

**for**( ;; )

{

//ADC

Xil\_Out32(AD1acq,1); //ADC acquire

**while** (adcdav==0) //ADC data available?

adcdav=Xil\_In32(AD1dav);

Xil\_Out32(AD1acq,0); //ADC stop acquire

adcdata[0]=Xil\_In32(AD1dat1); //input ADC data

adcdata[1]=Xil\_In32(AD1dat2);

**while** (adcdav==1) //wait for reset

adcdav=Xil\_In32(AD1dav);

/\* Send the next value on the queue. The queue should always be

empty at this point so a block time of 0 is used. \*/

xQueueSend( xQueue1, /\* The queue being written to. \*/

adcdata, /\* The address of the data being sent. \*/

0UL ); /\* The block time. \*/

}

}

/\*-----------------------------------------------------------\*/

**static** **void** **DA2Task**( **void** \*pvParameters )

{

**int** dacdata[2]; //DAC channel data

**int** dacdav; //DAC data available

**for**( ;; )

{

/\* Block to wait for data arriving on the queue. \*/

xQueueReceive( xQueue1, /\* The queue being read. \*/

dacdata, /\* Data is read into this address. \*/

portMAX\_DELAY ); /\* Wait without a timeout for data. \*/

//DAC

Xil\_Out32(DA2dat1, dacdata[0]); //output DAC data

Xil\_Out32(DA2dat2, dacdata[1]);

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 FIRTask( void \*pvParameters )

{

for( ;; )

{

}

}

\*/

1. FreeRTOS Task AFC/FIR/DAC Code

/\* FreeRTOS includes. \*/

**#include** "FreeRTOS.h"

**#include** "task.h"

**#include** "queue.h"

/\* Xilinx \*/

**#include** "xparameters.h"

**#include** "xil\_io.h"

**#include** <stdio.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

**static** **void** **AD1Task**( **void** \*pvParameters );

**static** **void** **DA2Task**( **void** \*pvParameters );

**static** **void** **FIRTask**( **void** \*pvParameters );

TaskHandle\_t xAD1Task;

TaskHandle\_t xDA2Task;

TaskHandle\_t xFIRTask;

QueueHandle\_t xQueue1 = NULL;

QueueHandle\_t xQueue2 = NULL;

**int** **main**(**void**)

{

**int** array[2];

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 );

xTaskCreate( DA2Task,

( **const** **char** \* ) "DA",

configMINIMAL\_STACK\_SIZE,

NULL,

tskIDLE\_PRIORITY+3,

&xDA2Task );

xTaskCreate( FIRTask,

( **const** **char** \* ) "FR",

configMINIMAL\_STACK\_SIZE,

NULL,

tskIDLE\_PRIORITY+2,

&xFIRTask);

xQueue1 = xQueueCreate( 1, /\* There is only one space in the queue. \*/

**sizeof**( array ) ); /\* Each space in the queue is large enough to hold a uint32\_t. \*/

configASSERT( xQueue1 );

xQueue2 = xQueueCreate( 1,

**sizeof**( array ) );

/\* Check the queue was created. \*/

configASSERT( xQueue2 );

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

vTaskStartScheduler();

**while** (1){};

}

/\*-----------------------------------------------------------\*/

**static** **void** **AD1Task**( **void** \*pvParameters )

{

**int** adcdav; //ADC data available

**int** adcdata[2]; //ADC channel data

**for**( ;; )

{

//ADC

Xil\_Out32(AD1acq,1); //ADC acquire

**while** (adcdav==0) //ADC data available?

adcdav=Xil\_In32(AD1dav);

Xil\_Out32(AD1acq,0); //ADC stop acquire

adcdata[0]=Xil\_In32(AD1dat1); //input ADC data

adcdata[1]=Xil\_In32(AD1dat2);

**while** (adcdav==1) //wait for reset

adcdav=Xil\_In32(AD1dav);

/\* Send the next value on the queue. The queue should always be

empty at this point so a block time of 0 is used. \*/

xQueueSend( xQueue1, /\* The queue being written to. \*/

adcdata, /\* The address of the data being sent. \*/

0UL ); /\* The block time. \*/

}

}

/\*-----------------------------------------------------------\*/

**static** **void** **DA2Task**( **void** \*pvParameters )

{

**int** dacdata[2]; //DAC channel data

**int** dacdav; //DAC data available

**for**( ;; )

{

/\* Block to wait for data arriving on the queue. \*/

xQueueReceive( xQueue2, /\* The queue being read. \*/

dacdata, /\* Data is read into this address. \*/

portMAX\_DELAY ); /\* Wait without a timeout for data. \*/

//DAC

Xil\_Out32(DA2dat1, dacdata[0]); //output DAC data

Xil\_Out32(DA2dat2, dacdata[1]);

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** **FIRTask**( **void** \*pvParameters )

{

**int** push[5] = {0,0,0,0,0};

**int** val[2];

**int** y[2];

**for**( ;; )

{

xQueueReceive(xQueue1, val, portMAX\_DELAY);

push[4] = push[3];//rotating pushdown list

push[3] = push[2];

push[2] = push[1];

push[1] = push[0];

push[0] = val[0];

y[0]=(**int**)((push[0]+push[1]+(3\*push[2])+(2\*push[3])+(3\*push[4]))/10);

xQueueSend(xQueue2, y, 0UL);

}

}

1. Video

<https://www.youtube.com/watch?v=tJJQIE-nlBk>