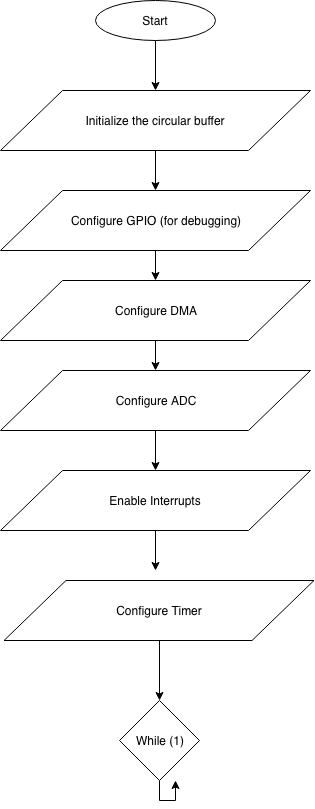
Mini Project Report.

Name: Rohan Sagar K.S

Email: rohansagar@mail.sfsu.edu

Student ID: 917929160

Block Diagram:



**Blocks Explanation:**

Circular Buffer declaration:

We declare a data structure for buffers that consists of the actual data, the count and also the pointer to the next structure. We use this data structure to store the actual data, the count of how many times it is filled and what buffer to store the data after filling the current buffer in next.

Configure GPIO:

We setup the GPIO’s as output so that we can debug our code using logic analyzer.

Configure DMA:

In this function we setup the DMA to run in Ping Pong mode, set the control structure and enable the ADC peripheral to work with DMA. We also set the channel parameters such as transfer size and arbitration size using uDMAChannelControlSet(). Finally, we setup the first transfer from the primary control structure (Ping) to the buffer\_1 and the first pong transfer to the buffer\_2.

Configure ADC:

The ADC is configured to use SS3 so that there is only one sample per trigger. We also set the trigger to Timer trigger so that the ADC is triggered by the timer. This is done by passing ADC\_TRIGGER\_TIMER argument to the ADCSequenceConfigure function.

Enable Interrupts:

We enable interrupts so that the DMA can be triggered

Configure Timer:

We enable the timer in the Periodic mode then set the timer period to be 100 hertz. This can be done by using TimerConfigure and TimerLoadSet functions.

**Window length:**

The window length mentioned is 100 samples and since the sampling frequency is 100hz.

Window length in time = (sampling interval )\* number of samples

=

=

Therefore, Window length in time = 1 second

**Window increment :**

**=** 50 samples

Therefore, using the same formulae

Window increment in time = 0.5 second

**Configurations:**

Timer configuration:

Timer frequency = 100 hz

We set this up by getting the clock frequency using SysCtlClockGet() and then dividing it by 100 (to get 100 hz frequency) and then passing it as an argument to TimerLoadSet() function.

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32Period -1); // the -1 because the interrupt fires at 0

We setup the timer so that it can trigger the ADC by using.TimerTriggerControl() Function.

**TimerControlTrigger**(TIMER0\_BASE, TIMER\_A, 1);

ADC configuration:

What sample sequencer is used = SS3

We do this by using the function ADCSequenceConfigure().

**ADCSequenceConfigure**(ADC0\_BASE, 3, ADC\_TRIGGER\_TIMER, 0); // configure the ADC0 to use SS3 and be triggered by using timer

We enable the ADC to be triggered by the timer by using ADC\_TIMER\_TRIGGER macro that is passes as parameter to the ADCSequenceConfigure function mentioned above.

uDMA Configuration:

Mode: Ping Pong

We setup parameters like the channel, the arbitration size and the number of bits using uDMAControlSet.

**uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_PRI\_SELECT, UDMA\_SIZE\_32 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_32 |UDMA\_ARB\_1); //Configuring the control parameters of the primary control structure

**uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_ALT\_SELECT,UDMA\_SIZE\_32 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_32 |UDMA\_ARB\_1);// Configure the control parameters for the alternate control structure for

ADC interrupt handler is where setting up the circular buffers are used.

**ADC interrupt handler:**

This is where we setup the next buffer in the circular buffer to which the data has to be stored. We check if the buffer is fill and then while the DMA is transferring data to the other buffer we setup the next buffer.

We also implement the processing delay here.

**Code:**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/adc.h"

**#include** "inc/hw\_adc.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/udma.h"

**#include** "inc/tm4c123gh6pm.h"

**#include** "driverlib/timer.h"

**#include** "driverlib/gpio.h"

uint32\_t ui32Period; // the period variable for timer configuration

**#define** ADC\_BUF\_SIZE 50 // setting the window increment size by declaring buffer size to 50.

// this is the data structure for the buffer

**struct** \_buffer;

**typedef** **struct** \_buffer buffer;

**typedef** buffer\* buffer\_pointer; // pointer to buffer

**struct** \_buffer{

uint32\_t data\_array [ADC\_BUF\_SIZE];

uint32\_t count;

buffer\_pointer next;

};

**static** buffer\_pointer current\_buffer\_pointer;

// Variables needed for the ADC conversion

uint32\_t raw\_adc[1];

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Aligning the control structures.

**#pragma** DATA\_ALIGN(ControlTable, 1024)

uint8\_t ControlTable[1024];

**void** **Configure\_GPIO**(**void**){

// This function configures the on board LEDs as output so that they can be used as outputs.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

**while**(!**SysCtlPeripheralReady**(SYSCTL\_PERIPH\_GPIOF)) // wait for the peripheral to be enabled

{

}

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3|GPIO\_PIN\_2|GPIO\_PIN\_1);

}

// Configuring the ADC

**void** **Configure\_ADC**(**void**)

{

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ); // Configure the system clock to be 40MHz

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //Enable the ADC

**SysCtlDelay**(2); //small delay to ensure the above steps are sucessfully completed

**ADCSequenceDisable**(ADC0\_BASE, 1); //disable ADC0 to set the configuration parameters

**ADCSequenceConfigure**(ADC0\_BASE, 3, ADC\_TRIGGER\_TIMER, 0); // configure the ADC0 to use SS3 and be triggered by using timer

**ADCSequenceStepConfigure**(ADC0\_BASE,3,0,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END); // Configuring the step of SS3

**IntPrioritySet**(INT\_ADC0SS3, 0x00); // configure ADC0 SS3 interrupt priority as 0

**IntEnable**(INT\_ADC0SS3); // enabling the interrupt for the adc sequencer This is used for adc triggering

**ADCIntEnableEx**(ADC0\_BASE, ADC\_INT\_SS3); // arm interrupt of ADC0 SS3

**ADCSequenceEnable**(ADC0\_BASE, 3); // enabling the sequence

}

//Configure DMA

**void** **Configure\_DMA**(**void**)

{

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UDMA); // enabling the module

**uDMAEnable**();

**uDMAControlBaseSet**(ControlTable); // setting the control table

**uDMAChannelAttributeDisable**(UDMA\_CHANNEL\_ADC3, UDMA\_ATTR\_ALTSELECT | UDMA\_ATTR\_HIGH\_PRIORITY | UDMA\_ATTR\_REQMASK);

**uDMAChannelAttributeEnable**(UDMA\_CHANNEL\_ADC3, UDMA\_ATTR\_USEBURST); //Set USEBURST: configure the uDMA controller to respond to burst requests only

**uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_PRI\_SELECT, UDMA\_SIZE\_32 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_32 |UDMA\_ARB\_1); //Configuring the control parameters of the primary control structure

**uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_ALT\_SELECT,UDMA\_SIZE\_32 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_32 |UDMA\_ARB\_1);// Configure the control parameters for the alternate control structure for

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_PRI\_SELECT,UDMA\_MODE\_PINGPONG,(**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO3),current\_buffer\_pointer -> data\_array, ADC\_BUF\_SIZE); // setting up initial buffers to transfer primary control structure

current\_buffer\_pointer -> count ++; // increment the count variable

current\_buffer\_pointer = current\_buffer\_pointer->next; // circular linked list pointer setup

// Set up the transfer parameters for the ADC0 SS3 alternate control structure. The mode is set to ping-pong, the transfer source is the ADC0 SS3 FIFO result register

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_ALT\_SELECT,UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO3),current\_buffer\_pointer->data\_array, ADC\_BUF\_SIZE);// setting up initial buffers to transfer secondary control structure

current\_buffer\_pointer -> count ++;// increment the count variable

current\_buffer\_pointer = current\_buffer\_pointer->next; // circular linked list pointer setup

**uDMAChannelEnable**(UDMA\_CHANNEL\_ADC3);// Enable the uDMA channel

}

// The interrupt handler for ADC0 SS3. This interrupt will occur when a DMA transfer is complete using the ADC0 SS3 uDMA channel.

**void** **adc0\_ss3\_handler**(**void**)

{

uint32\_t ui32Mode;

ui32Mode = **uDMAChannelModeGet**(UDMA\_CHANNEL\_ADC3 | UDMA\_PRI\_SELECT);

// when the primary buffer is full

**if**(ui32Mode == UDMA\_MODE\_STOP)

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, GPIO\_PIN\_1);

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_PRI\_SELECT, UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO3),current\_buffer\_pointer->data\_array, ADC\_BUF\_SIZE);

**uDMAChannelEnable**(UDMA\_CHANNEL\_ADC3); //Re-enable the uDMA channel

current\_buffer\_pointer -> count ++; //increment the buffer count

current\_buffer\_pointer = current\_buffer\_pointer->next; // rotate the circular buffer.

}

ui32Mode = **uDMAChannelModeGet**(UDMA\_CHANNEL\_ADC3 | UDMA\_ALT\_SELECT);

// when the alternate structure is full

**if**(ui32Mode == UDMA\_MODE\_STOP)

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0);

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC3 | UDMA\_ALT\_SELECT,UDMA\_MODE\_PINGPONG,(**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO3), current\_buffer\_pointer->data\_array, ADC\_BUF\_SIZE);

**uDMAChannelEnable**(UDMA\_CHANNEL\_ADC3);//Re-enable the uDMA channel

current\_buffer\_pointer -> count ++; //increment the buffer count

current\_buffer\_pointer = current\_buffer\_pointer->next; // rotate the circular buffer

}

**SysCtlDelay**(**SysCtlClockGet**()/10); // feasible processing delay

//SysCtlDelay(SysClockGet());// infeasible processor delay

}

//In this project we use timer to trigger the ADC in such a way that the sampling frequency is 100 hz

**void** **ConfigureTimer**(**void**){

// this function configures the timers

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0); // Enable Timer 0

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);// Configure Timer 0 to Periodic Mode

ui32Period = (**SysCtlClockGet**()/100 ) ; // this is for 100 hertz

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32Period -1); // the -1 because the interrupt fires at 0

**TimerControlTrigger**(TIMER0\_BASE, TIMER\_A, 1);

**IntMasterEnable**(); // enable interrupts globally

**TimerEnable**(TIMER0\_BASE, TIMER\_A); // now that the configuration is all done enable the timer.

}

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

{

**static** buffer buffer\_1, buffer\_2, buffer\_3; //instantiate buffers to store data

// circular buffer setup.

buffer\_1.next = &buffer\_2;

buffer\_2.next = &buffer\_3;

buffer\_3.next = &buffer\_1;

buffer\_1.count = 0;

buffer\_2.count = 0;

buffer\_3.count = 0;

current\_buffer\_pointer = &(buffer\_1); // setting the first buffer to buffer\_1

Configure\_GPIO(); // configure the GPIOs

Configure\_DMA(); // Configure DMA

Configure\_ADC(); // Configure ADC

**IntMasterEnable**(); // globally enable interrupt

ConfigureTimer(); // Configure Timer

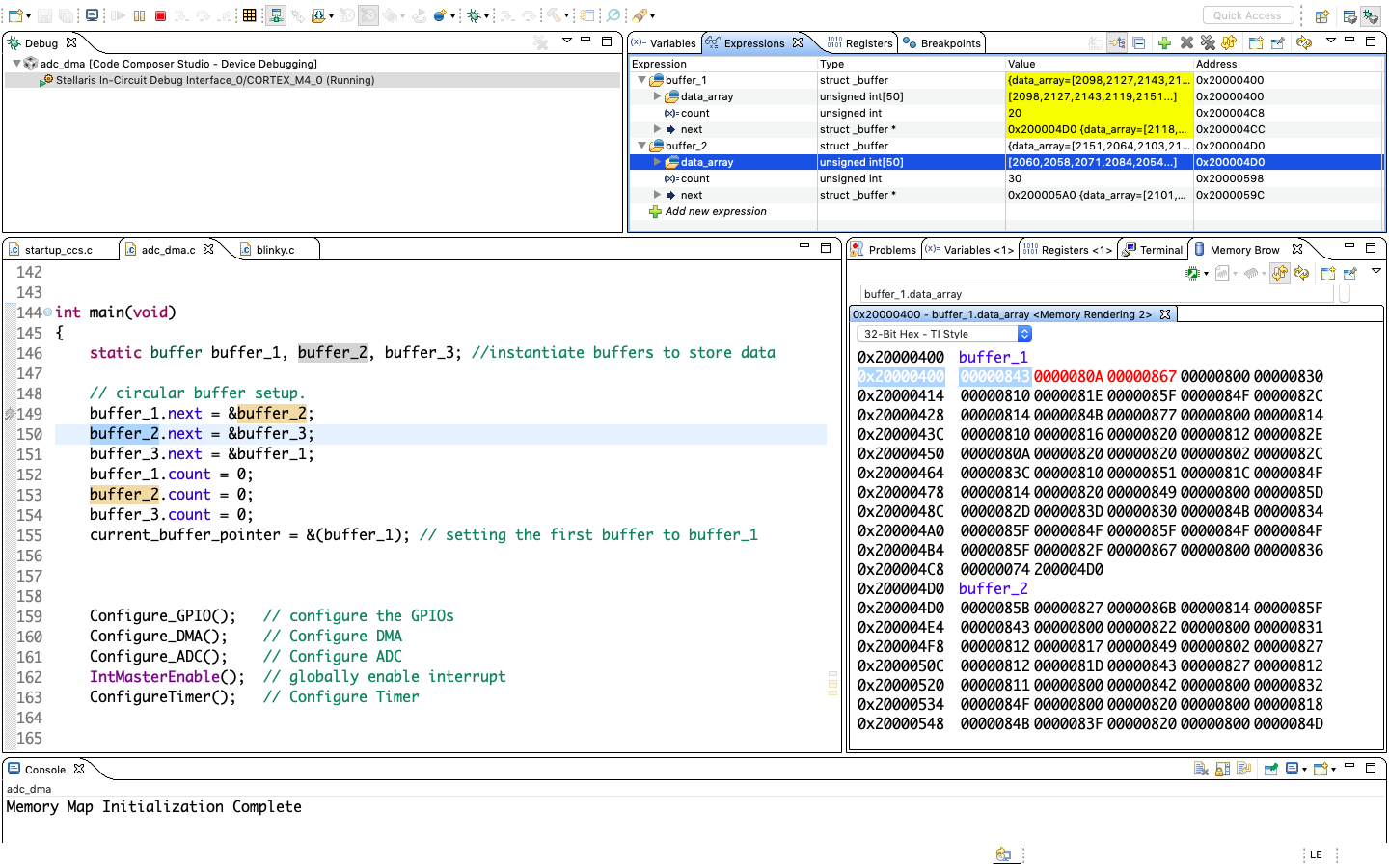
**while**(1)

{

}

}

**Debugging Results for feasible delay:**

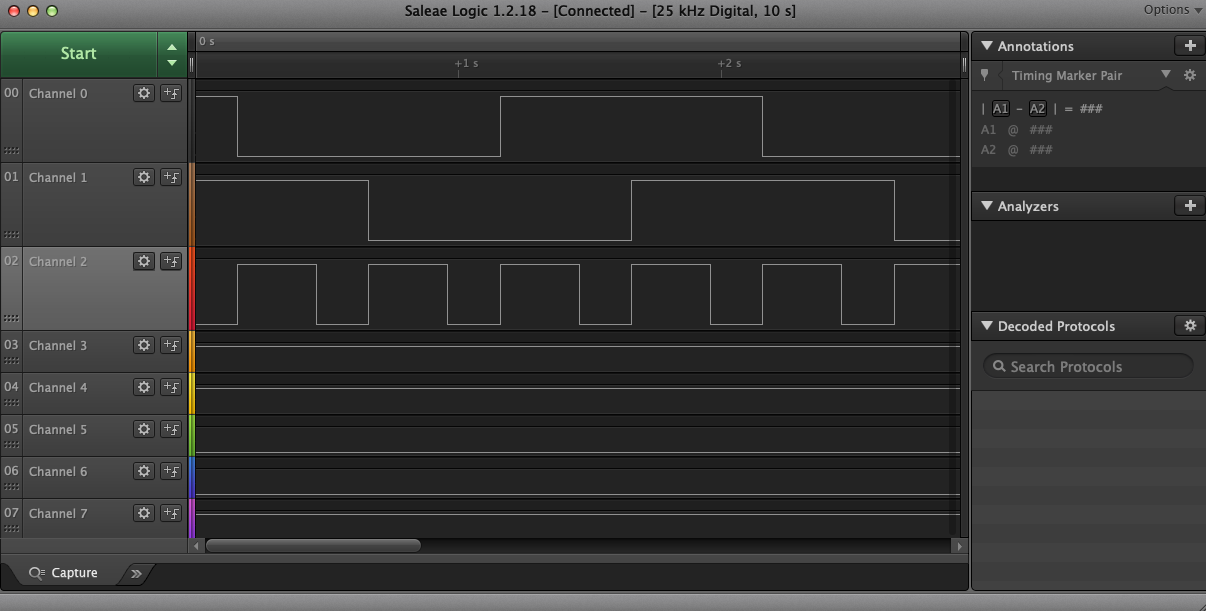
****

Logic Analyzer Output for feasible delay:

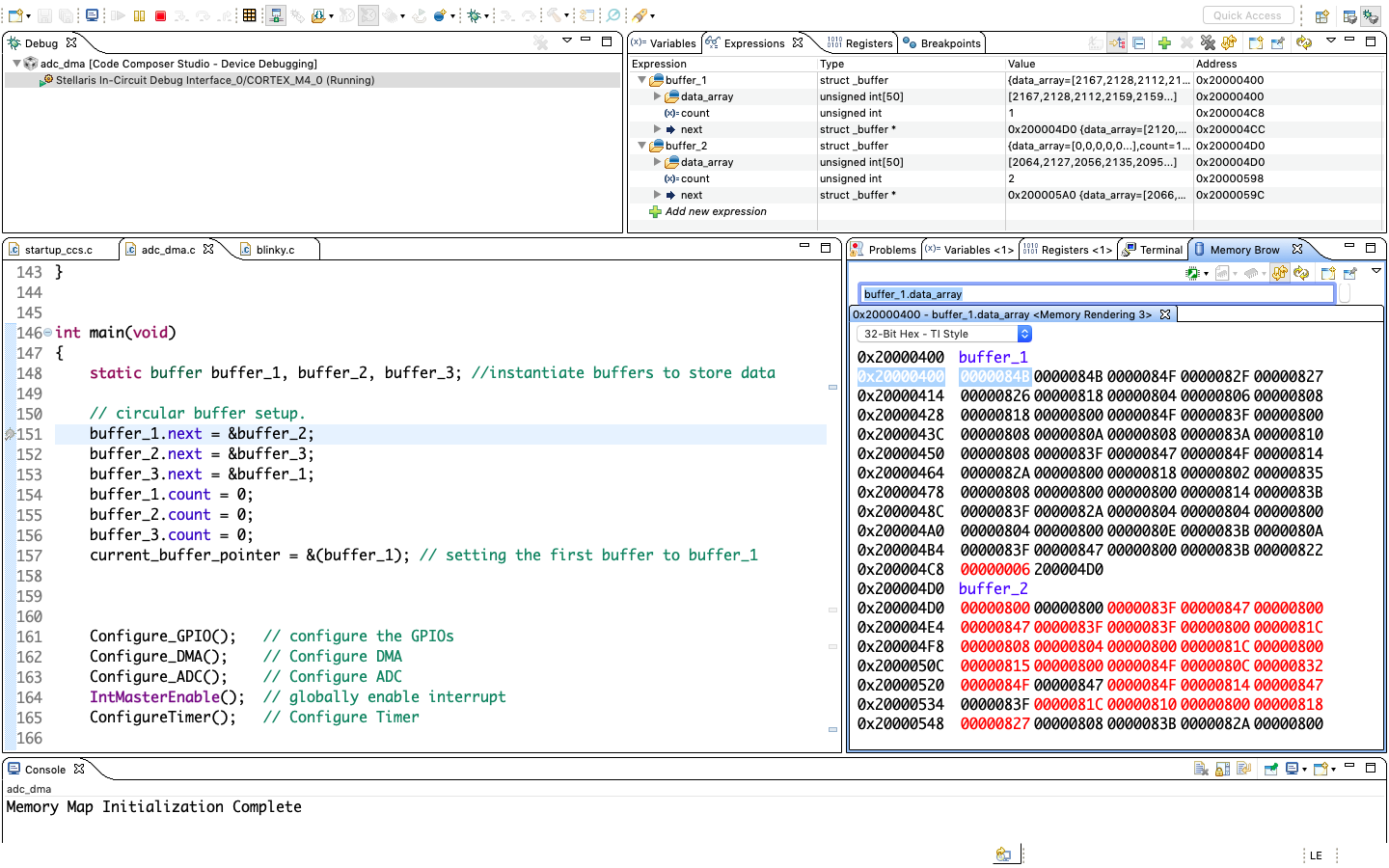
Channel 1 : Connected to GPIO F2 and is toggled whenever the ping buffer is full.

Channel 2: Connected to GPIO F3 and is toggled whenever the pong buffer is full.

Channel 3: Toggled before and after the processing delay.



**Debugging Results for infeasible Delay:**

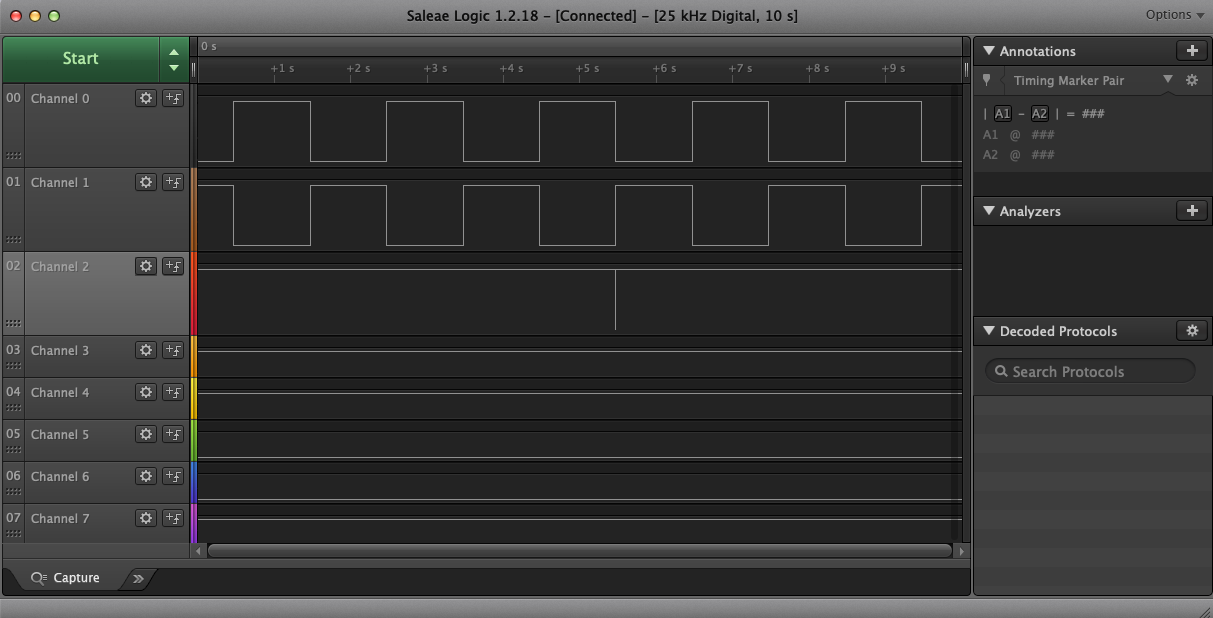
****

**Logic analyzer output for infeasible delay:**

Channel 1 : Connected to GPIO F2 and is toggled whenever the ping buffer is full.

Channel 2: Connected to GPIO F3 and is toggled whenever the pong buffer is full.

Channel 3: Toggled before and after the processing delay.

****

Discussion:

Through this experiment I learnt how to implement DMA using ping pong Mode. In addition, I learnt how to configure various parameters of DMA and how the processing interval compared to the window length effects the program execution (processing delay should be lesser than the window increment). One of the problem that I encountered was relate to the timer. Initially I triggered the ADC from within the interrupt handler of the timer but later realized that there is another way of triggering the ADC from timer using the timer trigger.

Another problem that I encountered was with the timing of the ping and pong buffers which was a mistake made in setting arbitration size.