Lab 3 Report
EEC 172 - A02
Spring 2016

Alex Elkman
Hardik Patel

LAB 3: Data Acquisition and Analysis

Objective:

The purpose of this lab was to use our knowledge of ADC (analog-to-digital converter) and DAC (digital-to-analog converter) along with SPI to create programs that perform data acquisition from different shaped waveforms. In the second part, we were tasked with using DMA for continuous polling with a ping-pong buffer from the ADC.

Design and Test Procedures:

Part I: Interfacing to the CC3200 ADC and the MAX549A SPI Dual 8-bit DAC

In this section we were tasked with implementing the ADC1 demo project with some small modifications to test a conversion of a DC signal from the 10KOhm potentiometer, and also various AC signals from the wave function generator, through the CC3200 ADC and then out of an SPI port MOSI into the MAX549A Dual 8-Bit DAC, and then into the Oscilloscope where we would verify that it was the same signal that was going in from the potentiometer.

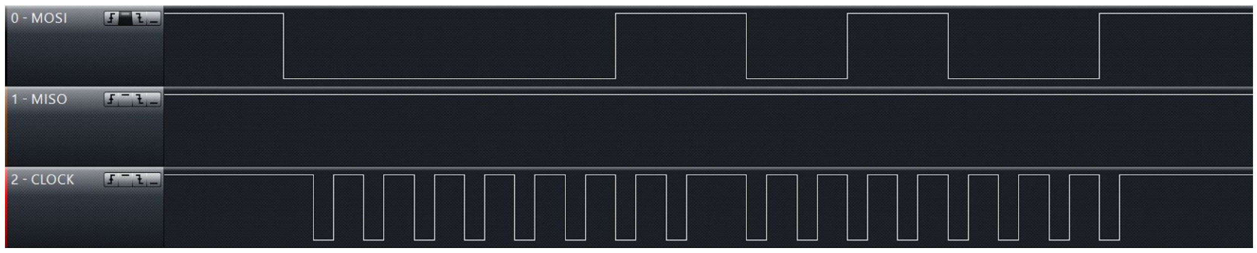

Figure 1: MOSI signal from the CC3200 with the Command and Data bits (first, and last 8-bits respectively)

Problems: We found some problems to be understanding the various command bits set for the DAC chip, we were confused about which one to use, but in the end found that using the continuously updating input pinmode command was the correct one to use. This way, it will continuously poll the input pin and give updated values, which were necessary for this section of the lab.

Part II: Data Acquisition / Signal Processing Application

In this section we were tasked with using DMA ping-pong buffers to acquire a continuously polling voltage signal in a manner very similar to that of part 1. The only real difference is we are no longer concerned with the DAC signal or the MOSI port, but instead interpreting the signal from the ADC on the CC3200 to determine the type of AC wave (square, triangle, sine) and it’s frequency. The algorithm we used for determining the frequency took advantage of the continuous polling and took some data from the ping and pong buffers, and looked through it for peak voltage signals which determined the start of a period. We then continued polling and checked for the next rising peak voltage signal. Then we used our knowledge of the ADC’s frequency of 62.5KHz to determine the frequency of the signal. This turned out to be quite accurate, and worked at up to 10x the required maximum frequency required for the lab. At this point, we used some of the data from that frequency algorithm to determine which form of wave the signal was. We accomplished this by using a variable to tell essentially the slope of the signal, and if it has a higher slope downward like a triangle wave, we found a good cutoff for how many of the polled voltages were above a certain threshold for the duration of the buffer. We found that we had the most counts for the square wave since it remains high until it hits a negedge, the slope is effectively 0.

Problems: We encountered lots of issues in finding a good algorithm for determining frequency, we ended up having to force our algorithm to wait a whole period before checking the signal, since we wanted to count at the peak rising edge and for some signals like the square wave, all the signals are the same voltage until a negedge occurs, which could mess up the algorithm if it doesn’t wait a whole clock period before starting the frequency test. After we got this working, it worked like a charm.

Conclusion: This lab helped us to learn more about embedded system signals, including developing unique algorithms for finding frequencies without the use of a timer, and also analyzing different AC waveforms and interpreting them through the CC3200 ADCs. It was a beneficial experience that has helped us broaden our knowledge on the subject matter.

PART 1:

```
// Standard includes
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>

// Driverlib includes
#include "utils.h"
#include "hw_memmap.h"
#include "hw_common_reg.h"
#include "hw_types.h"
#include "hw_adc.h"
#include "hw_ints.h"
#include "hw_gprcm.h"
#include "rom.h"
#include "rom_map.h"
#include "interrupt.h"
#include "prcm.h"
#include "uart.h"
#include "pin_mux_config.h"
#include "pin.h"
#include "adc.h"
#include "spi.h"
#include "uart_if.h"

#define USER_INPUT
#define UART_PRINT          Report
#define FOREVER             1
#define APP_NAME            "ADC Reference"
#define NO_OF_SAMPLES                64
#define SPI_IF_BIT_RATE  100000
unsigned long pulAdcSamples[4096];

//*****************************************************************************
//                      GLOBAL VARIABLES
//*****************************************************************************
#if defined(ccs)
extern void (* const g_pfnVectors[])(void);
#endif
#if defined(ewarm)
extern uVectorEntry __vector_table;
#endif

/****************************************************************************/
/*                      LOCAL FUNCTION PROTOTYPES                           */
/****************************************************************************/
static void BoardInit(void);
static void DisplayBanner(char * AppName);
```

```
//*****************************************************************************
//
//! Application startup display on UART
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
static void
DisplayBanner(char * AppName)
{
    Report("\n\n\n\r");
    Report("\t\t *************************************************\n\r");
    Report("\t\t       CC3200 %s Application       \n\r", AppName);
    Report("\t\t *************************************************\n\r");
    Report("\n\n\n\r");
}

//*****************************************************************************
//
//! Board Initialization & Configuration
//!
//! \param  None
//!
//! \return None
//
//*****************************************************************************
static void
BoardInit(void)
{
/* In case of TI-RTOS vector table is initialize by OS itself */
#ifndef USE_TIRTOS
    //
    // Set vector table base
    //
#if defined(ccs)
    MAP_IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);
#endif
#if defined(ewarm)
    MAP_IntVTableBaseSet((unsigned long)&__vector_table);
#endif
#endif
    //
    // Enable Processor
    //
    MAP_IntMasterEnable();
    MAP_IntEnable(FAULT_SYSTICK);

    PRCMCC3200MCUInit();
}

//*****************************************************************************
```

```
//
//! main - calls Crypt function after populating either from pre- defined vector
//! or from User
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
void
main()
{
       float temp;
    unsigned int  uiIndex=0;
    unsigned long ulSample;
    unsigned long temp2;
    unsigned long temp3;
    //
    // Initialize Board configurations
    //
    BoardInit();

    //adc
    // Configuring UART for Receiving input and displaying output
    // 1. PinMux setting
    // 2. Initialize UART
    // 3. Displaying Banner
    //
    PinMuxConfig();
    InitTerm();
    DisplayBanner(APP_NAME);

        //
        // Initialize Array index for multiple execution
        //
        uiIndex=0;


        //
        // Configure ADC timer which is used to timestamp the ADC data samples
        //
 //       MAP_ADCTimerConfig(ADC_BASE,2^17);
        MAP_PRCMPeripheralClkEnable(PRCM_GSPI,PRCM_RUN_MODE_CLK);
       MAP_PRCMPeripheralReset(PRCM_GSPI);
       MAP_SPIConfigSetExpClk(GSPI_BASE,MAP_PRCMPeripheralClockGet(PRCM_GSPI),
                             SPI_IF_BIT_RATE,SPI_MODE_MASTER,SPI_SUB_MODE_0,
                             (SPI_SW_CTRL_CS |
                             SPI_4PIN_MODE |
                             SPI_TURBO_OFF |
                             SPI_CS_ACTIVELOW |
                             SPI_WL_16));
       MAP_SPIEnable(GSPI_BASE);
       MAP_SPICSEnable(GSPI_BASE);
        //
        // Enable ADC timer which is used to timestamp the ADC data samples
        //
//        MAP_ADCTimerEnable(ADC_BASE);
```

```
        //
        // Enable ADC module
        //
        MAP_ADCEnable(ADC_BASE);

        //
        // Enable ADC channel
        //







        //
        // Print out ADC samples
        //
while(1){
         MAP_ADCChannelEnable(ADC_BASE, ADC_CH_3);
    while(uiIndex < NO_OF_SAMPLES + 4)
    {
        if(MAP_ADCFIFOLvlGet(ADC_BASE, ADC_CH_3))
        {
            ulSample = MAP_ADCFIFORead(ADC_BASE, ADC_CH_3);
            pulAdcSamples[uiIndex++] = ulSample;
        }


    }

    MAP_ADCChannelDisable(ADC_BASE, ADC_CH_3);

    uiIndex = 0;

        while(uiIndex < NO_OF_SAMPLES)
        {
              temp = (((float)((pulAdcSamples[4+uiIndex] >> 2 ) & 0xFFFF))*1.4)/4096+temp;
            UART_PRINT("%f\t",(((float)((pulAdcSamples[4+uiIndex] >> 2 ) &
0xFFFF))*1.4)/4096);
            uiIndex++;

            if ((uiIndex % 4)==0)
              UART_PRINT("\n\r");
        }
        uiIndex = 0;
        temp = temp / NO_OF_SAMPLES;
        UART_PRINT("\n\r%f", temp);
      //  SPIDataPut(GSPI_BASE, temp);
        temp2=(unsigned long)3;
        //     SPIDataPutNonBlocking(GSPI_BASE, pulAdcSamples);
        //     SPIDataGet(GSPI_BASE ,temp3);
        int intVal = (temp/3.3)*255; //roughly 50 for this application
        UART_PRINT("\n\rinVal: %d\n\r", intVal);
       MAP_SPICSEnable(GSPI_BASE);
       intVal = intVal + 3840;   //3840 is the DAC command for loading and updating the DAC
registers
```

```
        SPIDataPut(GSPI_BASE, (unsigned long)intVal);   //sends command byte 3 to load both
DAC input
        SPIDataGet(GSPI_BASE, temp3);
       // SPIDataPut(GSPI_BASE, intVal);
        //SPIDataGet(GSPI_BASE ,temp3);
        MAP_SPICSDisable(GSPI_BASE);
       }
}
```

PART 2:

```
//*****************************************************************************

//Standard includes
#include <stdlib.h>
#include <string.h>

// Driverlib includes
#include "hw_common_reg.h"
#include "hw_memmap.h"
#include "hw_apps_rcm.h"
#include "hw_types.h"
#include "hw_ints.h"
#include "hw_uart.h"
#include "hw_adc.h"
#include "uart.h"
#include "prcm.h"
#include "rom.h"
#include "rom_map.h"
#include "systick.h"
#include "utils.h"
#include "udma.h"
#include "interrupt.h"
#include "adc.h"
#include "math.h"
#include "gpio.h"
#include "gpio_if.h"



//#include "systick.h"

// Common interface includes
#include "udma_if.h"
#include "uart_if.h"
#include "pin_mux_config.h"

#define APPLICATION_VERSION     "1.1.1"
#define APP_NAME                "ADC uDMA Example"
#define UART_PRINT              Report

//*****************************************************************************
```

```
//
// The size of the data buffer.
//
//*****************************************************************************
#define BUFFER_SIZE         1024

//*****************************************************************************
//                  GLOBAL VARIABLES -- Start
//*****************************************************************************

// The destination buffers used for uDMA transfers.
volatile unsigned long g_ulPing[BUFFER_SIZE];
volatile unsigned long g_ulPong[BUFFER_SIZE];

volatile unsigned long g_ulPingCount;
volatile unsigned long g_ulPongCount;
//volatile unsigned long timerBuffer[BUFFER_SIZE];
volatile unsigned char g_ucPingflag;
volatile unsigned char g_ucPongflag;


// vector table entry
#if defined(ewarm)
    extern uVectorEntry __vector_table;
#endif

#if defined(ccs)
    extern void (* const g_pfnVectors[])(void);
#endif
//*****************************************************************************
//                  GLOBAL VARIABLES -- End
//*****************************************************************************



//*****************************************************************************
//!
//! The interrupt handler for ADC.  This interrupt will occur when a DMA
//! transfer is complete.
//!
//! \param None
//!
//! \return None
//*****************************************************************************
void ADCIntHandler(void)
{
       unsigned long ulMode;
       unsigned short Status;
       Status = ADCIntStatus(ADC_BASE, ADC_CH_3);
       ADCIntClear(ADC_BASE, ADC_CH_3, Status | ADC_DMA_DONE);
```

```
        //timer[1] = MAP_SysTickValueGet() - timer[0];
        //timer[0] = MAP_SysTickValueGet();

        ulMode = MAP_uDMAChannelModeGet(UDMA_CH17_ADC_CH3 | UDMA_PRI_SELECT);
        if (ulMode == UDMA_MODE_STOP) {
                g_ucPingflag=1;
                g_ulPingCount++;

                UDMASetupTransfer(UDMA_CH17_ADC_CH3 | UDMA_PRI_SELECT, UDMA_MODE_PINGPONG,
                                BUFFER_SIZE, UDMA_SIZE_32, UDMA_ARB_1,
                                (void *)(ADC_BASE + ADC_O_channel6FIFODATA), UDMA_SRC_INC_NONE,
                                (void *)&(g_ulPing[0]), UDMA_DST_INC_32);
        }

        ulMode = MAP_uDMAChannelModeGet(UDMA_CH17_ADC_CH3 | UDMA_ALT_SELECT);
        if      (ulMode == UDMA_MODE_STOP) {
                g_ucPongflag=1;
                g_ulPongCount++;

                UDMASetupTransfer(UDMA_CH17_ADC_CH3 | UDMA_ALT_SELECT, UDMA_MODE_PINGPONG,
                                        BUFFER_SIZE, UDMA_SIZE_32, UDMA_ARB_1,
                                        (void *)(ADC_BASE + ADC_O_channel6FIFODATA),
UDMA_SRC_INC_NONE,
                                        (void *)&(g_ulPong[0]), UDMA_DST_INC_32);

        }

}
//*****************************************************************************

void InitAdcDma( void )
{
        unsigned short Status;

        UDMAInit();

        MAP_uDMAChannelAssign(UDMA_CH17_ADC_CH3);

        UDMASetupTransfer(UDMA_CH17_ADC_CH3 | UDMA_PRI_SELECT, UDMA_MODE_PINGPONG,
                        BUFFER_SIZE, UDMA_SIZE_32, UDMA_ARB_1,
                        (void *)(ADC_BASE + ADC_O_channel6FIFODATA), UDMA_SRC_INC_NONE,
                        (void *)&(g_ulPing[0]), UDMA_DST_INC_32);

        UDMASetupTransfer(UDMA_CH17_ADC_CH3 | UDMA_ALT_SELECT, UDMA_MODE_PINGPONG,
                        BUFFER_SIZE, UDMA_SIZE_32, UDMA_ARB_1,
                        (void *)(ADC_BASE + ADC_O_channel6FIFODATA), UDMA_SRC_INC_NONE,
                        (void *)&(g_ulPong[0]), UDMA_DST_INC_32);

        g_ulPingCount=0;
        g_ulPongCount=0;
        g_ucPingflag=0;
        g_ucPongflag=0;
        ADCDMAEnable(ADC_BASE, ADC_CH_3);
        ADCIntRegister(ADC_BASE, ADC_CH_3, ADCIntHandler);
        Status = ADCIntStatus(ADC_BASE, ADC_CH_3);
```

```
       ADCIntClear(ADC_BASE, ADC_CH_3, Status | ADC_DMA_DONE);
       ADCIntEnable(ADC_BASE, ADC_CH_3, ADC_DMA_DONE);
       ADCEnable(ADC_BASE);
       ADCChannelEnable(ADC_BASE, ADC_CH_3);
}

//*****************************************************************************
//
//! Application startup display on UART
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
void
DisplayBanner()
{
    UART_PRINT("\n\n\n\r");
    UART_PRINT("\t\t   *******************************************\n\r");
    UART_PRINT("\t\t        CC3200 %s Application        \n\r", APP_NAME);
    UART_PRINT("\t\t   *******************************************\n\r");
    UART_PRINT("\n\n\n\r");

}


float roundToHundredths(float x){
      x *=10000;
       return floor(x) / 10000;
}

//*****************************************************************************
//
//! Board Initialization & Configuration
//!
//! \param  None
//!
//! \return None
//
//*****************************************************************************
static void
BoardInit(void)
{
/* In case of TI-RTOS vector table is initialize by OS itself */
#ifndef USE_TIRTOS
  //
  // Set vector table base
  //
#if defined(ccs)
    MAP_IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);
#endif
#if defined(ewarm)
    MAP_IntVTableBaseSet((unsigned long)&__vector_table);
#endif
#endif
  //
  // Enable Processor
  //
  MAP_IntMasterEnable();
  MAP_IntEnable(FAULT_SYSTICK);

  PRCMCC3200MCUInit();
}


//*****************************************************************************
//
//! Main function for uDMA Application
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
void main() {
       unsigned int uiIndex;
       unsigned char Inchar;

    //
    // Initailizing the board
    //
    BoardInit();
    //
    // Muxing for Enabling UART_TX and UART_RX and ADC Channel 3 (pin 60)
    //
    PinMuxConfig();
    GPIO_IF_LedConfigure(LED1|LED2|LED3);
    GPIO_IF_LedOff(MCU_ALL_LED_IND);
    //
    // Initialising the Terminal.
    //
    InitTerm();

    //
    // Display Welcome Message
    //
    DisplayBanner();
    UART_PRINT("Make sure analog signal in range 0 - 1.4V\n\r");
    UART_PRINT("Simple test program for Ping-Pong DMA;\r\n");
    UART_PRINT("Displays just a small portion of buffers\r\n");
    UART_PRINT("Press any key to start\r\n");
    UART_PRINT("Press 'q' key to stop\r\n");

    while(MAP_UARTCharsAvail(UARTA0_BASE) == false)  { ; }
    Inchar = MAP_UARTCharGetNonBlocking(UARTA0_BASE);
    int sin =1;
    InitAdcDma();
    int i;
    //MAP_SysTickPeriodSet(16777216);
    //MAP_SysTickEnable();
    Inchar = ' ';
    do {
```

```
	if (g_ucPingflag) {
		g_ucPingflag=0;
		UART_PRINT("Ping: \r\n");
		uiIndex=0;
		while (uiIndex < 24) {
			UART_PRINT("%f\t",(((float)((g_ulPing[uiIndex] >> 2 ) &
0x0FFF))*1.467)/4096);
			uiIndex++;
			if ((uiIndex % 4)==0)
				UART_PRINT("\n\r");
		}
	//This section of code determines the frequency assuming the voltage is 1.0V
	/*	if(square){
			int tempOld = 0;
			int flag = 0;
				int count = 0;
				int temp = 0;
				for(i=0; i<1024; i++){
					if(flag == 1)
						count++;
					else if (flag == 2)
						break;
					temp = (int)((((((float)((g_ulPing[i] >> 2 ) &
0x0FFF))*1.467)/4096)) + 0.5);
					if(temp == tempOld)
						flag++;
					if((((((float)((g_ulPing[i] >> 2 ) &
0x0FFF))*1.467)/4096)) > .6)
						tempOld = 1;
					if((((((float)((g_ulPing[i] >> 2 ) &
0x0FFF))*1.467)/4096)) < .6)
						tempOld = 0;

				}

				UART_PRINT("Frequency is: %d\n\r", 62500/count);
			}*/

		int highFlag = 0;
		int lowFlag = 0;
		if(sin){
			int A=0, B=0, C=0;
			for(i=0; i<1024; i++){

				if(((((((float)((g_ulPing[i] >> 2 ) & 0x0FFF))*1.467)/4096)) >
.99) && B==0)
							highFlag = 1;
				else if(((((((float)((g_ulPing[i] >> 2 ) & 0x0FFF))*1.467)/4096))
< .99) && A==0)
							lowFlag = 1;

					if(highFlag ==1 && lowFlag ==1){
						if(((((((float)((g_ulPing[i] >> 2 ) &
0x0FFF))*1.467)/4096)) > .99) && B==0)
							A++;
						else if(((((((float)((g_ulPing[i] >> 2 ) &
0x0FFF))*1.467)/4096)) < .99) && A!=0)
```

```
                                                B++;
                                        else if((((((float)((g_ulPing[i] >> 2 ) &
0x0FFF))*1.467)/4096)) > .99) && B!=0){
                                                break;
                                        }
                                }
                        }
                        UART_PRINT("A is: %d \n\r ", A);
                        if(A<3){
                                C = B+A;
                            GPIO_IF_LedOff(MCU_ALL_LED_IND);
                                UART_PRINT("Frequency is: %d  and wave is: ", 62500/C, A);
                                UART_PRINT("TRIANGLE\n\r" );
                                GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);

                        }
                        else if(A>=3 && A<11){
                                C = B+A;
                            GPIO_IF_LedOff(MCU_ALL_LED_IND);
                                UART_PRINT("Frequency is: %d  and wave is: ", 62500/C, A);
                                UART_PRINT("SINE \n\r" );
                                GPIO_IF_LedOn(MCU_GREEN_LED_GPIO);
                        }
                        else{
                                        C = B+A;
                                    GPIO_IF_LedOff(MCU_ALL_LED_IND);
                                        UART_PRINT("Frequency is: %d  and wave is: ", 62500/C, A);
                                UART_PRINT("SQUARE \n\r" );
                                GPIO_IF_LedOn(MCU_RED_LED_GPIO);
                        }

                }



        }

        if (g_ucPongflag) {
                g_ucPongflag=0;
                UART_PRINT("Pong: \r\n");
                uiIndex=0;
                while (uiIndex < 24) {
                        UART_PRINT("%f\t",(((float)((g_ulPong[uiIndex] >> 2 ) &
0x0FFF))*1.467)/4096);
                        uiIndex++;
                        if ((uiIndex % 4)==0)
                                UART_PRINT("\n\r");
                }
                //UART_PRINT("Spot 500: %f\t\n\r",(((float)((g_ulPong[500] >> 2 ) &
0x0FFF))*1.467)/4096);
        }
         if (MAP_UARTCharsAvail(UARTA0_BASE))  {
                Inchar = MAP_UARTCharGetNonBlocking(UARTA0_BASE);
         }
    } while (Inchar != 'q');
     UART_PRINT("Program stopped\r\n");
        ADCDMADisable(ADC_BASE, ADC_CH_3);
```

```
ADCChannelDisable(ADC_BASE, ADC_CH_3);}
```