CPE403 – Advanced Embedded Systems

# Design Assignment 2

Name: Jenifer Christina

Email: chrisj14@unlv.nevada.edu

Github Repository link: https://github.com/chrisj14/CCS-Assignment

Youtube Playlist link:https://youtu.be/cwDEEj2b8To

**Follow the submission guideline to be awarded points for this Assignment.**

1. Code for Tasks. for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only. Use separate page for each task.

Task 01: Interface the TMP006 IR Temperature Sensor (MKII Educational Boosterpack). Determine the object temperature. Perform all computations using IQ Math Structures and Functions. Display the results in the terminal.

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <string.h>

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

**#include** "inc/tm4c123gh6pm.h" //def. for the interrupt and register assignments on the Tiva C Series device on the launchPad board

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

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

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

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

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

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

**#include** "driverlib/pin\_map.h"

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

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

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

**#include** "driverlib/timer.h" //Defines and macros for Timer API of driverLib.

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

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

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

**#include** "utils/uartstdio.h"

// For Temperature value

uint32\_t ui32Period;

**char** buffer [4];

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Define source and destination buffers

**#define** MEM\_BUFFER\_SIZE 1024

**static** uint32\_t g\_ui32SrcBuf[MEM\_BUFFER\_SIZE];

**static** uint32\_t g\_ui32DstBuf[MEM\_BUFFER\_SIZE];

// Define errors counters

**static** uint32\_t g\_ui32DMAErrCount = 0;

**static** uint32\_t g\_ui32BadISR = 0;

// Define transfer counter

**static** uint32\_t g\_ui32MemXferCount = 0;

// The control table used by the uDMA controller. This table must be aligned to a 1024 byte boundary.

**#pragma** DATA\_ALIGN(pui8ControlTable, 1024)

uint8\_t pui8ControlTable[1024];

// Library error routine

**#ifdef** DEBUG

**void**

\_\_error\_\_(**char** \*pcFilename, uint32\_t ui32Line)

{

}

**#endif**

// uDMA transfer error handler

**void**

**uDMAErrorHandler**(**void**)

{

uint32\_t ui32Status;

// Check for uDMA error bit

ui32Status = **uDMAErrorStatusGet**();

// If there is a uDMA error, then clear the error and increment the error counter.

**if**(ui32Status)

{

**uDMAErrorStatusClear**();

g\_ui32DMAErrCount++;

}

}

// uDMA interrupt handler. Run when transfer is complete.

**void**

**uDMAIntHandler**(**void**)

{

uint32\_t ui32Mode;

// Check for the primary control structure to indicate complete.

ui32Mode = **uDMAChannelModeGet**(UDMA\_CHANNEL\_SW);

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

{

// Increment the count of completed transfers.

g\_ui32MemXferCount++;

// Configure it for another transfer.

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_SW, UDMA\_MODE\_AUTO,

g\_ui32SrcBuf, g\_ui32DstBuf, MEM\_BUFFER\_SIZE);

// Initiate another transfer.

**uDMAChannelEnable**(UDMA\_CHANNEL\_SW);

**uDMAChannelRequest**(UDMA\_CHANNEL\_SW);

uint32\_t ui32Status;

ui32Status = **UARTIntStatus**(UART0\_BASE, **true**); //get interrupt status

**UARTIntClear**(UART0\_BASE, ui32Status); //clear the asserted interrupts

**while**(**UARTCharsAvail**(UART0\_BASE)) //loop while there are chars

{

**char** cChar=**UARTCharGet**(UART0\_BASE);

**UARTCharPutNonBlocking**(UART0\_BASE, cChar); //echo character

**if** (cChar=='R') { //Turn on RED LED

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

}

**else** **if** (cChar=='r') { //Turn off RED LED

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

}

**else** **if** (cChar=='G') { //Turn on Green LED

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, GPIO\_PIN\_3);

}

**else** **if** (cChar=='g') { //Turn off Green LED

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0);

}

**else** **if** (cChar=='B') { //Turn on Blue LED

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2);

}

**else** **if** (cChar=='b') { //Turn off Blue LED

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0);

}

**else** **if** (cChar=='T') { //Show Temperature in Centigrade

**ADCIntClear**(ADC0\_BASE,2);

**ADCProcessorTrigger**(ADC0\_BASE, 2);

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

**UARTprintf**("\n C %3d\t \n",ui32TempValueC );

}

**else** **if** (cChar=='t') { //Show Temperature in Farenheit

**ADCIntClear**(ADC0\_BASE,2);

**ADCProcessorTrigger**(ADC0\_BASE, 2);

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

**UARTprintf**("\n F %3d\t \n",ui32TempValueF );

}

**else** **if** (cChar=='S') { //Show LED Status

**UARTprintf**("\n");

**if** (**GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1))

**UARTprintf**("Red LED is on \n");

**if** (**GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2))

**UARTprintf**("Blue LED is on \n");

**if** (**GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3))

**UARTprintf**("Green LED is on \n");

}

}

}

// If the channel is not stopped, then something is wrong.

**else**

{

g\_ui32BadISR++;

}

}

// Initialize the uDMA software channel to perform a memory to memory uDMA transfer.

**void**

**InitSWTransfer**(**void**)

{

uint32\_t ui32Idx;

// Fill the source memory buffer with a simple incrementing pattern.

**for**(ui32Idx = 0; ui32Idx < MEM\_BUFFER\_SIZE; ui32Idx++)

{

g\_ui32SrcBuf[ui32Idx] = ui32Idx;

}

// Enable interrupts from the uDMA software channel.

**IntEnable**(INT\_UDMA);

// Place the uDMA channel attributes in a known state. These should already be disabled by default.

**uDMAChannelAttributeDisable**(UDMA\_CHANNEL\_SW,

UDMA\_ATTR\_USEBURST | UDMA\_ATTR\_ALTSELECT |

(UDMA\_ATTR\_HIGH\_PRIORITY |

UDMA\_ATTR\_REQMASK));

// Configure the control parameters for the SW channel. The SW channel

// will be used to transfer between two memory buffers, 32 bits at a time,

// and the address increment is 32 bits for both source and destination.

// The arbitration size will be set to 8, which causes the uDMA controller

// to rearbitrate after 8 items are transferred. This keeps this channel from

// hogging the uDMA controller once the transfer is started, and allows other

// channels to get serviced if they are higher priority.

**uDMAChannelControlSet**(UDMA\_CHANNEL\_SW | UDMA\_PRI\_SELECT,

UDMA\_SIZE\_32 | UDMA\_SRC\_INC\_32 | UDMA\_DST\_INC\_32 |

UDMA\_ARB\_8);

// Set up the transfer parameters for the software channel. This will

// configure the transfer buffers and the transfer size. Auto mode must be

// used for software transfers.

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_SW | UDMA\_PRI\_SELECT,

UDMA\_MODE\_AUTO, g\_ui32SrcBuf, g\_ui32DstBuf,

MEM\_BUFFER\_SIZE);

// Now the software channel is primed to start a transfer. The channel

// must be enabled. For software based transfers, a request must be

// issued. After this, the uDMA memory transfer begins.

**uDMAChannelEnable**(UDMA\_CHANNEL\_SW);

**uDMAChannelRequest**(UDMA\_CHANNEL\_SW);

}

**int**

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

{

//Configure peripherals

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

//Setup for ADC

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

**ADCSequenceConfigure**(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 2);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3); //enable pin for LED PF2

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

**FPULazyStackingEnable**();

**SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN |

SYSCTL\_XTAL\_16MHZ);

**SysCtlPeripheralClockGating**(**true**);

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

**SysCtlPeripheralSleepEnable**(SYSCTL\_PERIPH\_UDMA);

**IntEnable**(INT\_UDMAERR);

**uDMAEnable**();

**uDMAControlBaseSet**(pui8ControlTable);

InitSWTransfer();

**UARTprintf**("Enter the cmd: \n"

"R: Red LED, \n"

"G: Green LED, \n"

"B: Blue LED, \n"

"T: Temperature, \n"

"S: status of the LEDs. \n");

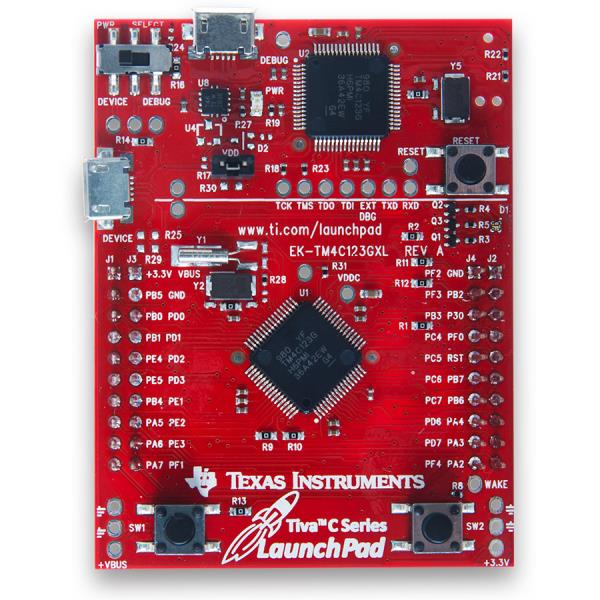
**while**(1)

{

}

}

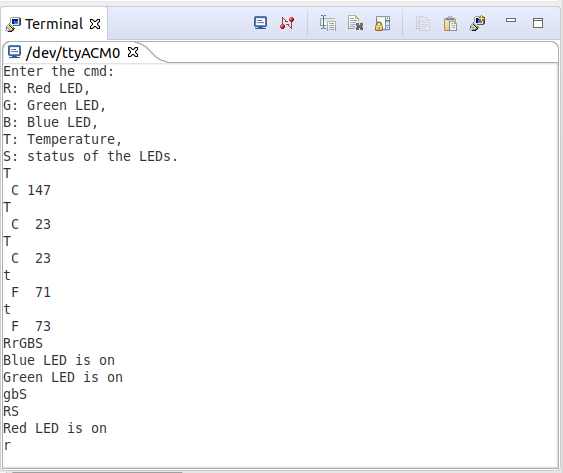
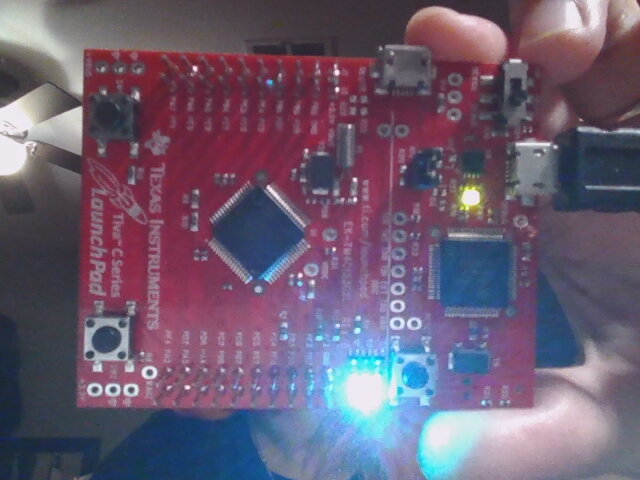
1. Block diagram and/or Schematics showing the components, pins used, and interface.



Connected to USB

GPIO\_PIN\_1,2,3 to show LED from uDMA Command

1. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.



1. Declaration

I understand the Student Academic Misconduct Policy - http://studentconduct.unlv.edu/misconduct/policy.html

“This assignment submission is my own, original work”.

Jenifer Christina