**Date Submitted: 10/02/20**

**Task 01:**

**Youtube Link:**

No link because this did not work in the terminal

**Modified Code:**

**This CODE DID NOT WORK IN THE TERMINAL**

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "inc/hw\_ints.h"

#include "driverlib/interrupt.h"

#include "string.h"

int main(void) {

SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

//enable UART Peripherals

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//Cofigure pins for reciever and transmiter

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF); //enable GPIO port for LED

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

//Initialize UART parameters: 115200 baud rate, 8-1

UARTConfigSetExpClk(UART0\_BASE, SysCtlClockGet(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

IntMasterEnable();

IntEnable(INT\_UART0); //enable the UART interrupts

UARTIntEnable(UART0\_BASE, UART\_INT\_RX|UART\_INT\_RT);//only enable rx and tx

//create prompt

UARTCharPut(UART0\_BASE, 'E');

UARTCharPut(UART0\_BASE, 'n');

UARTCharPut(UART0\_BASE, 't');

UARTCharPut(UART0\_BASE, 'e');

UARTCharPut(UART0\_BASE, 'r');

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, 'T');

UARTCharPut(UART0\_BASE, 'e');

UARTCharPut(UART0\_BASE, 'x');

UARTCharPut(UART0\_BASE, 't');

UARTCharPut(UART0\_BASE, ':');

UARTCharPut(UART0\_BASE, ' ');

while (1)

{

// if (UARTCharsAvail(UART0\_BASE)) UARTCharPut(UART0\_BASE, UARTCharGet(UART0\_BASE));

}

}

void UARTIntHandler(void)

{

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

{

UARTCharPutNonBlocking(UART0\_BASE, UARTCharGetNonBlocking(UART0\_BASE)); //echo character

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2); //blink LED

SysCtlDelay(SysCtlClockGet() / (1000 \* 3));

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0); //turn off LED

}

}

**------------------------------------------------------------------------------------**

**Task 02:**

Youtube Link:

https://www.youtube.com/watch?v=iHGdtzH-cT0

**Modified Code:**

**#include <stdint.h>**

**#include <stdbool.h>**

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

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

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

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

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

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

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

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

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

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

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

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**typedef struct{**

**uint32\_t adcBuffer[4];**

**uint32\_t AVG;**

**uint32\_t C;**

**uint32\_t F;**

**}Temp\_t;**

**typedef enum{**

**WAIT,**

**RED, BLUE, GREEN,**

**red, blue, green,**

**TEMP,**

**temp,**

**STAT**

**}State\_enum;**

**typedef struct{**

**unsigned char input;**

**}UART\_t;**

**void check\_status(uint32\_t status, const char\* string);**

**void send\_status(const char\* string, bool status);**

**void calculate\_avg\_temperature(Temp\_t \*temp\_t);**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#define GREEN\_LED GPIO\_PIN\_3**

**#define BLUE\_LED GPIO\_PIN\_2**

**#define RED\_LED GPIO\_PIN\_1**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**int main(void)**

**{**

**//Config System Clock**

**SysCtlClockSet(SYSCTL\_SYSDIV\_5 | SYSCTL\_XTAL\_16MHZ | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN );**

**Temp\_t temp\_t;**

**volatile unsigned char status\_of\_leds;**

**volatile State\_enum state, nextState = WAIT;**

**volatile UART\_t uart\_t;**

**volatile uint32\_t period;**

**//Config UART**

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

**SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);**

**GPIOPinConfigure(GPIO\_PA0\_U0RX);**

**GPIOPinConfigure(GPIO\_PA1\_U0TX);**

**GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);**

**UARTStdioConfig(0, 115200, SysCtlClockGet());**

**//**

**//Config ADC**

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

**ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);**

**ADCSequenceStepConfigure(ADC0\_BASE, 1, 0, ADC\_CTL\_TS);**

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

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

**ADCSequenceStepConfigure(ADC0\_BASE, 1, 3, ADC\_CTL\_TS | ADC\_CTL\_END | ADC\_CTL\_IE);**

**ADCSequenceEnable(ADC0\_BASE, 1);**

**//Config LEDs**

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

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

**GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);**

**//Next State for UI**

**while(1)**

**{**

**state = nextState;**

**switch(state){**

**case WAIT:**

**while(!UARTCharsAvail(UART0\_BASE)){};**

**uart\_t.input = UARTCharGet(UART0\_BASE);**

**if(uart\_t.input == 'R') nextState = RED;**

**else if(uart\_t.input == 'B') nextState = BLUE;**

**else if(uart\_t.input == 'G') nextState = GREEN;**

**else if(uart\_t.input == 'r') nextState = red;**

**else if(uart\_t.input == 'b') nextState = blue;**

**else if(uart\_t.input == 'g') nextState = green;**

**else if(uart\_t.input == 'T') nextState = TEMP;**

**else if(uart\_t.input == 't') nextState = temp;**

**else if(uart\_t.input == 'S') nextState = STAT;**

**else nextState = WAIT;**

**break;**

**case RED:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, RED\_LED, RED\_LED);**

**printm("Turning on Red LED");**

**nextState = WAIT;**

**break;**

**case red:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, RED\_LED, 0);**

**printm("Turning off Red LED");**

**nextState = WAIT;**

**break;**

**case BLUE:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, BLUE\_LED, BLUE\_LED);**

**printm("Turning on Blue LED");**

**nextState = WAIT;**

**break;**

**case blue:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, BLUE\_LED, 0);**

**printm("Turning off Blue LED");**

**nextState = WAIT;**

**break;**

**case GREEN:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, GREEN\_LED, GREEN\_LED);**

**printm("Turning On Green LED");**

**nextState = WAIT;**

**break;**

**case green:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, GREEN\_LED, 0);**

**printm("Turning off Green LED");**

**nextState = WAIT;**

**break;**

**case TEMP:**

**calculate\_avg\_temperature(&temp\_t);**

**UARTprintf("C %3d\t", temp\_t.C);**

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

**nextState = WAIT;**

**break;**

**case temp:**

**calculate\_avg\_temperature(&temp\_t);**

**UARTprintf("F: %3d\t",temp\_t.F);**

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

**nextState = WAIT;**

**break;**

**case STAT:**

**status\_of\_leds = GPIOPinRead(GPIO\_PORTF\_BASE, RED\_LED | BLUE\_LED | GREEN\_LED);**

**check\_status((status\_of\_leds&RED\_LED) >> 1, "Red: ");**

**check\_status((status\_of\_leds&BLUE\_LED) >> 2, "Blue: ");**

**check\_status((status\_of\_leds&GREEN\_LED) >> 3, "Green: ");**

**nextState = WAIT;**

**break;**

**default:**

**nextState = WAIT;**

**continue;**

**}**

**}**

**}**

**void check\_status(uint32\_t status, const char\* string){**

**if(status)**

**send\_status(string, 1);**

**else**

**send\_status(string, 0);**

**}**

**void send\_status (const char\* string, bool status){**

**UARTprintf(string);**

**if(status)**

**UARTprintf("ON");**

**else**

**UARTprintf("OFF");**

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

**}**

**//Calculate the average**

**void calculate\_avg\_temperature(Temp\_t \*temp\_t){**

**//Trigger ADC and wait for interrupt**

**ADCProcessorTrigger(ADC0\_BASE, 1);**

**while(!ADCIntStatus(ADC0\_BASE, 1, false)){};**

**ADCIntClear(ADC0\_BASE, 1);**

**ADCSequenceDataGet(ADC0\_BASE, 1, temp\_t->adcBuffer);**

**//Calculate Average**

**temp\_t->AVG = (temp\_t->adcBuffer[0] + temp\_t->adcBuffer[1] + temp\_t->adcBuffer[2] + temp\_t->adcBuffer[3])/4;**

**temp\_t->C = (1475 - ((2475 \* temp\_t->AVG)) / 4096)/10;**

**temp\_t->F = ((temp\_t->C \* 9) + 160) / 5;**

**}**

**---------------------------------------------------------------------------------**

**Task 03:**

Youtube Link:

https://www.youtube.com/watch?v=p2XHK3QvZcg

**Modified Code:**

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

**#include <stdbool.h>**

**#include <stdint.h>**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**#define ADC\_SAMPLE\_BUF\_SIZE 1**

**#if defined(ewarm)**

**#pragma data\_alignment=1024**

**uint8\_t pui8ControlTable[1024];**

**#elif defined(ccs)**

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

**uint8\_t pui8ControlTable[1024];**

**#else**

**uint8\_t pui8ControlTable[1024] \_\_attribute\_\_ ((aligned(1024)));**

**#endif**

**//Structs**

**enum BUFFER\_STATUS**

**{**

**EMPTY,**

**FULL**

**};**

**typedef struct{**

**uint16\_t adcBuffer[ADC\_SAMPLE\_BUF\_SIZE];**

**uint32\_t AVG;**

**uint32\_t C;**

**uint32\_t F;**

**}Temp\_t;**

**typedef enum{**

**WAIT,**

**RED, BLUE, GREEN,**

**red, blue, green,**

**TEMP,**

**temp,**

**STAT**

**}State\_enum;**

**typedef struct{**

**unsigned char input;**

**}UART\_t;**

**///////////////////////////////////////////**

**//Variables**

**volatile static enum BUFFER\_STATUS BufferStatus;**

**volatile Temp\_t temp\_t;**

**volatile uint32\_t Count;**

**///////////////////////////////////////////**

**//Prototype**

**void CalculateTemperatureAvg(void);**

**void ConfigureUART(void);**

**void ConfigureADC(void);**

**void ConfigureUDMA(void);**

**void check\_status(uint32\_t status, const char\* string);**

**void send\_status(const char\* string, bool status);**

**///////////////////////////////////////////**

**#define GREEN\_LED GPIO\_PIN\_3**

**#define BLUE\_LED GPIO\_PIN\_2**

**#define RED\_LED GPIO\_PIN\_1**

**///////////////////////////////////////////**

**void printm(char \*str)**

**{**

**//While it's not NULL**

**while(\*str != '\0')**

**{**

**UARTCharPut(UART0\_BASE,\*str);**

**++str;**

**}**

**}**

**int main(void){**

**SysCtlClockSet(SYSCTL\_SYSDIV\_2\_5 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_INT | SYSCTL\_XTAL\_16MHZ);**

**volatile unsigned char status\_of\_leds;**

**volatile State\_enum state, nextState = WAIT;**

**volatile UART\_t uart\_t;**

**BufferStatus = EMPTY;**

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

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

**//Config LEDs**

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

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

**GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);**

**ConfigureUART();**

**ConfigureUDMA();**

**ConfigureADC();**

**IntMasterEnable();**

**while(1)**

**{**

**state = nextState;**

**switch(state){**

**case WAIT:**

**while(!UARTCharsAvail(UART0\_BASE)){};**

**uart\_t.input = UARTCharGet(UART0\_BASE);**

**if(uart\_t.input == 'R') nextState = RED;**

**else if(uart\_t.input == 'B') nextState = BLUE;**

**else if(uart\_t.input == 'G') nextState = GREEN;**

**else if(uart\_t.input == 'r') nextState = red;**

**else if(uart\_t.input == 'b') nextState = blue;**

**else if(uart\_t.input == 'g') nextState = green;**

**else if(uart\_t.input == 'T') nextState = TEMP;**

**else if(uart\_t.input == 't') nextState = temp;**

**else if(uart\_t.input == 'S') nextState = STAT;**

**else nextState = WAIT;**

**break;**

**case RED:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, RED\_LED, RED\_LED);**

**printm("Turning on Red LED");**

**nextState = WAIT;**

**break;**

**case red:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, RED\_LED, 0);**

**printm("Turning off Red LED");**

**nextState = WAIT;**

**break;**

**case BLUE:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, BLUE\_LED, BLUE\_LED);**

**printm("Turning on Blue LED");**

**nextState = WAIT;**

**break;**

**case blue:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, BLUE\_LED, 0);**

**printm("Turning off Blue LED");**

**nextState = WAIT;**

**break;**

**case GREEN:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, GREEN\_LED, GREEN\_LED);**

**printm("Turning On Green LED");**

**nextState = WAIT;**

**break;**

**case green:**

**GPIOPinWrite(GPIO\_PORTF\_BASE, GREEN\_LED, 0);**

**printm("Turning off Green LED");**

**nextState = WAIT;**

**break;**

**case TEMP:**

**CalculateTemperatureAvg();**

**UARTprintf("C %3d\t", temp\_t.C);**

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

**nextState = WAIT;**

**break;**

**case temp:**

**CalculateTemperatureAvg();**

**UARTprintf("F: %3d\t",temp\_t.F);**

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

**nextState = WAIT;**

**break;**

**case STAT:**

**status\_of\_leds = GPIOPinRead(GPIO\_PORTF\_BASE, RED\_LED | BLUE\_LED | GREEN\_LED);**

**check\_status((status\_of\_leds&RED\_LED) >> 1, "Red: ");**

**check\_status((status\_of\_leds&BLUE\_LED) >> 2, "Blue: ");**

**check\_status((status\_of\_leds&GREEN\_LED) >> 3, "Green: ");**

**nextState = WAIT;**

**break;**

**default:**

**nextState = WAIT;**

**break;**

**}**

**}**

**}**

**void check\_status(uint32\_t status, const char\* string){**

**if(status)**

**send\_status(string, 1);**

**else**

**send\_status(string, 0);**

**}**

**void send\_status (const char\* string, bool status){**

**UARTprintf(string);**

**if(status)**

**UARTprintf("ON");**

**else**

**UARTprintf("OFF");**

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

**}**

**void CalculateTemperatureAvg(void){**

**ADCProcessorTrigger(ADC0\_BASE, 0);**

**while(BufferStatus == EMPTY){};**

**temp\_t.AVG = 0;**

**for(Count = 0; Count < ADC\_SAMPLE\_BUF\_SIZE; Count++)**

**{**

**temp\_t.AVG += temp\_t.adcBuffer[Count];**

**temp\_t.adcBuffer[Count] = 0;**

**}**

**BufferStatus = EMPTY;**

**uDMAChannelTransferSet(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT,**

**UDMA\_MODE\_BASIC,**

**(void \*)(ADC0\_BASE + ADC\_O\_SSFIFO0),**

**&temp\_t.adcBuffer, ADC\_SAMPLE\_BUF\_SIZE);**

**uDMAChannelEnable(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT);**

**temp\_t.AVG = ((temp\_t.AVG +**

**(ADC\_SAMPLE\_BUF\_SIZE / 2)) /**

**ADC\_SAMPLE\_BUF\_SIZE);**

**temp\_t.C = (1475 - ((2475 \* temp\_t.AVG)) / 4096)/10;**

**temp\_t.F = ((temp\_t.C \* 9) + 160) / 5;**

**}**

**void ConfigureUART(void){**

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

**SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);**

**GPIOPinConfigure(GPIO\_PA0\_U0RX);**

**GPIOPinConfigure(GPIO\_PA1\_U0TX);**

**GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);**

**UARTStdioConfig(0, 115200, SysCtlClockGet());**

**}**

**void ConfigureADC(void){**

**IntDisable(INT\_ADC0SS0);**

**ADCIntDisable(ADC0\_BASE, 0);**

**ADCSequenceDisable(ADC0\_BASE, 0);**

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

**ADCSequenceStepConfigure(ADC0\_BASE, 0, 0, ADC\_CTL\_TS | ADC\_CTL\_END |**

**ADC\_CTL\_IE);**

**ADCSequenceEnable(ADC0\_BASE, 0);**

**ADCIntClear(ADC0\_BASE, 0);**

**ADCSequenceDMAEnable(ADC0\_BASE, 0);**

**ADCIntEnable(ADC0\_BASE, 0);**

**IntEnable(INT\_ADC0SS0);**

**}**

**void ConfigureUDMA(void){**

**uDMAEnable();**

**uDMAControlBaseSet(pui8ControlTable);**

**uDMAChannelAttributeDisable(UDMA\_CHANNEL\_ADC0,**

**UDMA\_ATTR\_ALTSELECT | UDMA\_ATTR\_HIGH\_PRIORITY |**

**UDMA\_ATTR\_REQMASK);**

**uDMAChannelControlSet(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_SIZE\_16 |**

**UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_16 | UDMA\_ARB\_64);**

**uDMAChannelTransferSet(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT,**

**UDMA\_MODE\_BASIC,**

**(void \*)(ADC0\_BASE + ADC\_O\_SSFIFO0),**

**&temp\_t.adcBuffer, ADC\_SAMPLE\_BUF\_SIZE);**

**uDMAChannelAttributeEnable(UDMA\_CHANNEL\_ADC0, UDMA\_ATTR\_USEBURST);**

**uDMAChannelEnable(UDMA\_CHANNEL\_ADC0);**

**}**

**void ADCSeq0Handler(void){**

**ADCIntClear(ADC0\_BASE, 0);**

**if ((uDMAChannelModeGet(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT) == UDMA\_MODE\_STOP))**

**{**

**BufferStatus = FULL;**

**}**

**}**

**------------------------------------------------------------------------------------**