Bert Zhao
EE 474

Abstract: This lab report provides an overview of lab 3 of EE/CSE 474. There is a breakdown of the two sections to this lab, along with relevant results and outputs. The implication of the tasks of this lab in regards to the final project is also discussed. All code for this lab in included in the appendix.

Introduction: There are a few different areas of focus for this lab. In the first section of the lab, we are introduced to the analog to digital converter (ADC) and the phase-lock-loop (PLL). We use the PLL is manipulate the clock speed and we use the ADC to measure the on-board temperature sensor as a response to the system clock frequency. In the second section of this lab, we are introduced to communication between the TIVA and the PC. To communicate between the PC and the TIVA board, we use a serial connection between the two systems. In addition to the standard USB cable connection, we also setup the wireless bluetooth connection.

Procedure: In section A of the lab, the goal is to use the on-board switches to manipulate the system clock frequency via the PLL, and then to measure the on-board temperature sensor with the ADC. We display the temperature through the on-board LEDs, with each color representing a different temperature range. We first configure the necessary PLL registers to generate 4MHz and 80MHz system clock frequencies. The important register fields of note are the DIV400, SYSDIV2, and SYSDIV2LSB within the RCC2 register. The following table from the datasheet provides some examples on how to set these fields to obtain a desired clock frequency.

SYSDIV2	SYSDIV2LSB	Divisor	Frequency (BYPASS2=0)[a]
0x00	reserved	/2	reserved
0x01	0	/3	reserved
	1	/4	reserved
0x02	0	/5	80 MHz
	1	/6	66.67 MHz
0x03	0	/7	reserved
	1	/8	50 MHz
0x04	0	/9	44.44 MHz
	1	/10	40 MHz
...	...	...	...
0x3F	0	/127	3.15 MHz
	1	/128	3.125 MHz

Figure 1: Clock configuration table

We should then initialize the timer so that we can use a timer trigger for each sequence read by the ADC of the temperature sensor. The GPIO port F is configured similar to the previous labs, enabling the pins for the switches as input and the pins for the LEDs as outputs. In initializing the ADC, we use sample sequencer 3 since we only need one input. The ADC operates through an interrupt which handles the temperature calculation. The program will then process the switch inputs to determine the proper PLL configuration, and the proper timer configuration to maintain the sample rate at once per second. The temperature is sent as an output via the LEDs according the table below.

Color	PF3 PF2 PF1 value	Temperature in Celsius
Red	001	0-17
Blue	010	17-19
Violet	011	19- 21
Green	100	21-23
Yellow	101	23-25
Light Blue	110	25-27
White	111	27-40

Figure 2: LED temperature table

In the second section of this lab, we use UART explore communication between the TIVA board and a PC. The TIVA board has built in UART functionality, and on the PC we use PuTTY to set up a connection with the TIVA. In configuring the UART module on the TIVA we need to calculate the appropriate baud rate divisors so that the UART can communicate with the PC. The necessary formulas for the baud divisor are listed below.

```
BRD = BRDI + BRDF = UARTSysClk / (ClkDiv * Baud Rate)
UARTFBRD[DIVFRAC] = integer(BRDF * 64 + 0.5)
```

Figure 3: Baud divisor formulas

The integer component of the divisor needs to be stored in the UARTIBRD register and the fractional component of the divisor needs to be stored in the UARTFBRD register. When using the standard USB port connection, the baud rate is 9600 and when using bluetooth, the baud rate is 115200. When incorporating the bluetooth communication to the temperature reading system, the UART system clock with be either 4MHz or 80MHz, so the baud divisor will need to be reconfigured when switching between the different system clock frequencies.

Results: The final outputs of the system consist of the on-board LEDs and the PuTTY console on the PC. A sample of the output through PuTTY is included at the end of this section. Some issues during this lab included some inconsistencies in the temperature readings between different boards. The temperature readings on my TIVA board were different from the readings on other TIVA boards, despite running the same program. This issue is not a major problem as the main concept regarding the temperature as a response to system clock frequency can still be observed. Another issue in this lab was with the bluetooth communication when there was no limit to the frequency of data transmissions. When testing the bluetooth module, the transmission between the TIVA and PC would sometimes stop abruptly when there is a constant stream of data being sent, such as when the read character/string functions are in the infinite loop in main. Since the lab specifications required only one reading per second, limiting these data transmissions solved this issue.

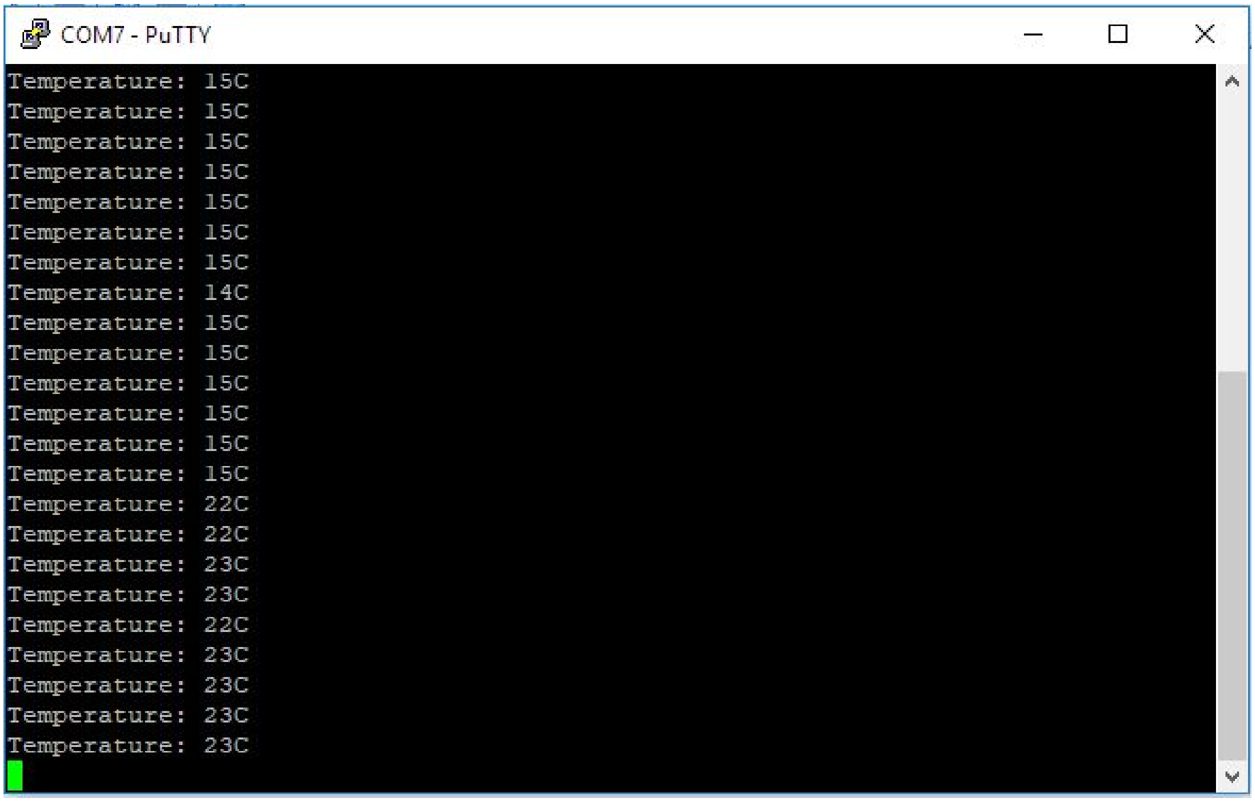
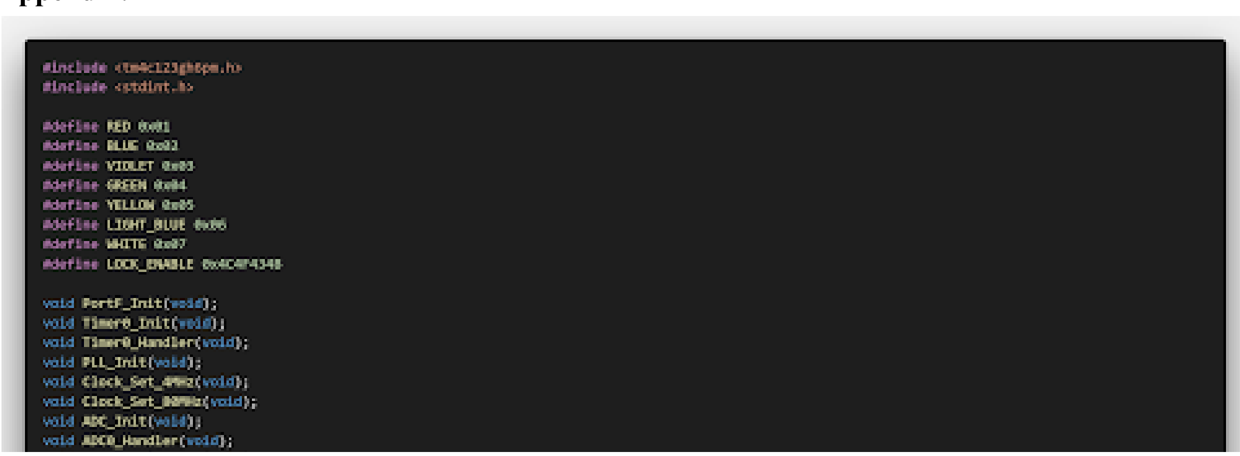


Figure 4: PuTTY sample output

Conclusion: This lab introduced the ADC, PLL, and UART communication. These are three important tools in embedded programming and they can be used in our final project. In fact, the bluetooth is a required function of the final protect so it is important to have gained a solid understanding of bluetooth from this lab.

Appendix:



```c
#include <tm4c123gh6pm.h>
#include <stdint.h>

#define RED 0x01
#define BLUE 0x02
#define VIOLET 0x03
#define GREEN 0x04
#define YELLOW 0x05
#define LIGHT_BLUE 0x06
#define WHITE 0x07
#define LOCK_ENABLE 0x4C4F434B

void PortF_Init(void);
void Timer0_Init(void);
void Timer0_Handler(void);
void PLL_Init(void);
void Clock_Set_4MHz(void);
void Clock_Set_80MHz(void);
void ADC_Init(void);
void ADC0_Handler(void);
```

Bert Zhao
EE 474

```c
void UART1_Init_4MHz(void);
void UART1_Init_80MHz(void);
void UART1_printChar(char c);
void printString(char * string);
char Int_To_Char(int i);
void Temp_LED_Ctrl(void);

volatile int temp;

int main(void) {
  PLL_Init();
  UART1_Init_80MHz();
  PortF_Init();
  Timer0_Init();
  ADC_Init();
  while (1) {
    volatile unsigned long sw = (GPIO_PORTF_DATA_R & 0x11);
    if (sw == 0x01) {
      Clock_Set_4MHz();
      UART1_Init_4MHz();
    } else if (sw == 0x10) {
      Clock_Set_80MHz();
      UART1_Init_80MHz();
    }
  }
  return 0;
}

void PortF_Init(void) {
  volatile unsigned long delay;
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF; // enable port F
  delay = SYSCTL_RCGC2_R; // wait for clock setup
  GPIO_PORTF_LOCK_R = LOCK_ENABLE; // unlock PF0
  GPIO_PORTF_CR_R |= 0xFF; // 0b1111.1111 enable write to PUR
  GPIO_PORTF_AFSEL_R = 0x00; // disable alternate functions of pins
  GPIO_PORTF_DIR_R = 0x0E; // 0b01110 PF4 and PF0 inputs, PF1, PF2, PF3 outputs
  GPIO_PORTF_PUR_R = 0x11; // enable PUR on PF4 and PF0
  GPIO_PORTF_DEN_R = 0x1F; // enable pins PF4 to PF0
  GPIO_PORTF_DATA_R = 0; // all LEDs off
}

void Timer0_Handler(void) {
  TIMER0_ICR_R |= (1<<0);
}

void Timer0_Init(void) {
  SYSCTL_RCGCTIMER_R |= 0x01; // enable timer 0 clock
  TIMER0_CTL_R &= ~(1<<0); // disable timer maybe 0x01
  TIMER0_CFG_R |= 0x00000000; // set 32-bit config
  TIMER0_TAMR_R |= (0x2<<0); // set TAMR to periodic timer mode
  TIMER0_CTL_R |= (1<<5); // enable timer trigger for ADC
  TIMER0_TAMR_R &= ~(1<<4); // timer countdown
  TIMER0_TAILR_R = 0x04C4B400; // start counter at 80,000,000
  NVIC_EN0_R |= (1<<19); // enable timer handler
  TIMER0_IMR_R |= (1<<0); // enable timer interrupt mask might need to move
  TIMER0_CTL_R |= (1<<0); // enable timer
}

void PLL_Init(void) {
  SYSCTL_RCC2_R |= (1<<31); // use RCC2
  SYSCTL_RCC2_R |= (1<<11); // bypass PLL
  SYSCTL_RCC_R = (SYSCTL_RCC_R & ~0x7C0) + 0x540; // select crystal and oscillator
  SYSCTL_RCC2_R &= ~0x70; // config main oscillator
  SYSCTL_RCC2_R &= ~(1<<13); // activate PLL with PWRDN
  SYSCTL_RCC2_R |= (1<<30); // set DIV400
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R & ~0x1FC00000) + (0x2<<22); // selecting divisor for 80MHz
  while ((SYSCTL_RIS_R & 0x40) == 0) {}; // wait for PLL lock
  SYSCTL_RCC2_R &= ~(1<<11); // clear bypass
}

void ADC_Init(void) {
  SYSCTL_RCGCADC_R |= (1<<0); // enable ADC0
  volatile unsigned long delay = SYSCTL_RCGCADC_R; // wait for clock
  ADC0_ACTSS_R &= ~(1<<3); // disable sequencer 3
  ADC0_EMUX_R = (0x5<<12); // tiggered by timer
  ADC0_SSCTL3_R |= (1<<3); // set input to temperature sensor once per sequence
  ADC0_SSCTL3_R |= (1<<1); // set END0 to avoid unpredictable behaviors
  ADC0_SSCTL3_R |= (1<<2); // enable interrupt
  ADC0_IM_R |= (1<<3); // set sequencer 3 interrupt mask  (0x7<<1)
  ADC0_ISC_R |= (1<<3); // clear previous flags
  NVIC_PRI4_R |= (0x2<<13); // set interrupt priority
  NVIC_EN0_R |= (1<<17); // enable ADC0 interrupt handler
  ADC0_ACTSS_R |= (1<<3); // enable sequencer 3

}
```

```
void ADC0_Handler(void) {
  temp = (int) (147.5 - (247.5 * ADC0_SSFIFO3_R) / 4096.0);
  Temp_LED_Ctrl();
  printString("Temperature: ");
  UART1_printChar(int_To_Char(temp/10));
  UART1_printChar(int_To_Char(temp%10));
  printString("C\n\r");
  ADC0_ISC_R |= (1<<3);
}

void Clock_Set_4MHz(void) {
  TIMER0_CTL_R &= ~(1<<0); // disable timer
  TIMER0_TAILR_R = 0x003D0900; // start counter at 4,000,000
  SYSCTL_RCC2_R |= (1<<31); // use RCC2
  SYSCTL_RCC2_R |= (1<<11); // bypass PLL
  SYSCTL_RCC_R = (SYSCTL_RCC_R & ~0x7C0) + 0x540; // select crystal and oscillator
  SYSCTL_RCC2_R &= ~0x70; // config main oscillator
  SYSCTL_RCC2_R &= ~(1<<13); // activate PLL with PWRDN
  SYSCTL_RCC2_R |= (1<<30);
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R & ~0x1FC00000) + (0x31<<22); // selecting divisor for 4MHz
  while ((SYSCTL_RIS_R & 0x40) == 0) {}; // wait for PLL lock
  SYSCTL_RCC2_R &= ~(1<<11);
  TIMER0_CTL_R |= (1<<0); // enable timer
}

void Clock_Set_80MHz(void) {
  TIMER0_CTL_R &= ~(1<<0); // disable timer
  TIMER0_TAILR_R = 0x04C4B400; // start counter at 80,000,000
  SYSCTL_RCC2_R |= (1<<31); // use RCC2
  SYSCTL_RCC2_R |= (1<<11); // bypass PLL
  SYSCTL_RCC_R = (SYSCTL_RCC_R & ~0x7C0) + 0x540; // select crystal and oscillator
  SYSCTL_RCC2_R &= ~0x70; // config main oscillator
  SYSCTL_RCC2_R &= ~(1<<13); // activate PLL with PWRDN
  SYSCTL_RCC2_R |= (1<<30); // set DIV400
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R & ~0x1FC00000) + (0x4<<22); // selecting divisor for 80MHz
  while ((SYSCTL_RIS_R & 0x40) == 0) {}; // wait for PLL lock
  SYSCTL_RCC2_R &= ~(1<<11); // clear bypass
  TIMER0_CTL_R |= (1<<0); // enable timer
}

void UART1_Init_4MHz(void) {
  SYSCTL_RCGCUART_R |= (1<<1); // provide clock to UART1
  SYSCTL_RCGC2_R |= (1<<1); // provide clock to GPIOB
  GPIO_PORTB_AFSEL_R = (1<<1) | (1<<0); // enable alternate functions on pins PB0 and PB1
  GPIO_PORTB_PCTL_R = (1<<0) | (1<<4); // configure PMCn fields for pins
  GPIO_PORTB_DEN_R = (1<<0) | (1<<1); // configure inputs and outputs
  UART1_CTL_R &= ~(1<<0); // disable UART1
  UART1_IBRD_R = 2; // integer portion
  UART1_FBRD_R = 11; // decimal portion
  UART1_LCRH_R = (0x3<<5) | (1<<4); // configure signal parameters 8-bits, no parity, 1-bit stop
  UART1_CC_R = 0x0; // send system clock to UART
  UART1_CTL_R = (1<<0) | (1<<8) | (1<<9); // enable UART0, Tx, and Rx
}

void UART1_Init_80MHz(void) {
  SYSCTL_RCGCUART_R |= (1<<1); // provide clock to UART1
  SYSCTL_RCGC2_R |= (1<<1); // provide clock to GPIOB
  GPIO_PORTB_AFSEL_R = (1<<1) | (1<<0); // enable alternate functions on pins PB0 and PB1
  GPIO_PORTB_PCTL_R = (1<<0) | (1<<4); // configure PMCn fields for pins
  GPIO_PORTB_DEN_R = (1<<0) | (1<<1); // configure inputs and outputs
  UART1_CTL_R &= ~(1<<0); // disable UART1
  UART1_IBRD_R = 43; // integer portion
  UART1_FBRD_R = 26; // decimal portion
  UART1_LCRH_R = (0x3<<5) | (1<<4); // configure signal parameters 8-bits, no parity, 1-bit stop
  UART1_CC_R = 0x0; // send system clock to UART
  UART1_CTL_R = (1<<0) | (1<<8) | (1<<9); // enable UART0, Tx, and Rx
}

void UART1_printChar(char c) {
  while ((UART1_FR_R & (1<<5)) != 0) {}; // wait for previous transmission to complete
  UART1_DR_R = c;
}

void printString(char * string) {
  while (*string) {
    UART1_printChar(*(string++));
  }
}

char int_To_Char(int i) {
    return (char) (i + 48);
}
```

```c
void Temp_LED_Ctrl(void) {
  if ((temp >= 0) & (temp <= 17)) {
    GPIO_PORTF_DATA_R = (RED<<1);
  } else if ((temp > 17) & (temp <= 19)) {
    GPIO_PORTF_DATA_R = (BLUE<<1);
  } else if ((temp > 19) & (temp <= 21)) {
    GPIO_PORTF_DATA_R = (VIOLET<<1);
  } else if ((temp > 21) & (temp <= 23)) {
    GPIO_PORTF_DATA_R = (GREEN<<1);
  } else if ((temp > 23) & (temp <= 25)) {
    GPIO_PORTF_DATA_R = (YELLOW<<1);
  } else if ((temp > 25) & (temp <= 27)) {
    GPIO_PORTF_DATA_R = (LIGHT_BLUE<<1);
  } else if ((temp > 27) & (temp <= 40)) {
    GPIO_PORTF_DATA_R = (WHITE<<1);
  }
}
```

Code 1: Section 2 complete program