Dalton Altstaetter & Ken Lee
Due 2/13/15
445M

Lab Report 1

Objectives:

The objectives of Lab 1 are to familiarize ourselves with basic interfacing of the TM4C using the provided memory mapped driver libraries, getting familiar with the Keil environment, and to review the software modules that will be used and built upon this semester(ADC, UART, FIFO, Timers, SysTick, PLL). We also interfaced with an external LCD, ST7735, for debugging and displaying data/information. We split the LCD into two separate displays, which can be used by separate threads. We built a simple interpreter and added commands for debugging the ADC, LCD, and Timer drivers. We also got re-acquainted with the Digital Logic Analyzer when testing our system and gathering profiling information to test the accuracy of our timer interrupts. In addition, we learned basic commands on how to better maintain and integrate our code for future labs (using Git).

Hardware Design: none

Software Design:

```
void ST7735_Message (int device, int line, char *string, long value){
		if(device==0){
			if(line>7){
				ST7735_SetCursor(0,0);
				ST7735_OutString((uint8_t*)"            ");
				ST7735_SetCursor(0,0);
				ST7735_OutString((uint8_t*)"line out of bounds");
			}else{
				ST7735_SetCursor(0,line);
				ST7735_OutString((uint8_t*)"            ");
				ST7735_SetCursor(0,line);
				ST7735_OutString((uint8_t*)string);
				ST7735_OutChar(' ');
				ST7735_OutUDec(value);
			}
		}else if(device==1){
			if(line<8){
				ST7735_SetCursor(0,8);
				ST7735_OutString((uint8_t*)"            ");
				ST7735_SetCursor(0,8);
				ST7735_OutString((uint8_t*)"line out of bounds");
			}else{
				ST7735_SetCursor(0, line);
```

```
                        ST7735_OutString((uint8_t*)"           ");
                        ST7735_SetCursor(0, line);
                        ST7735_OutString((uint8_t*)string);
                        ST7735_OutChar(' ');
                        ST7735_OutUDec(value);
                    }
                }else{
                    ST7735_SetCursor(0,0);
                    ST7735_OutString((uint8_t*)"            ");
                    ST7735_SetCursor(0,0);
                    ST7735_OutString((uint8_t*)"Invalid Device");
                }
}

// SS3 interrupts: enabled and promoted to controller
volatile uint32_t NumberOfSamples=0;
volatile uint16_t* Buffer;
volatile uint32_t Status=1;
void ADC_Collect(uint8_t channelNum, uint32_t fs, uint16_t buffer[],uint32_t
                numberOfSamples){
  volatile uint32_t delay;
  // **** GPIO pin initialization ****
  switch(channelNum){         // 1) activate clock
    case 0:
    case 1:
    case 2:
    case 3:
    case 8:
    case 9:                  //    these are on GPIO_PORTE
      SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R4; break;
    case 4:
    case 5:
    case 6:
    case 7:                  //    these are on GPIO_PORTD
      SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R3; break;
    case 10:
    case 11:                 //    these are on GPIO_PORTB
      SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1; break;
    default: return;         //    0 to 11 are valid channels on the LM4F120
  }
  delay = SYSCTL_RCGCGPIO_R;      // 2) allow time for clock to stabilize
  delay = SYSCTL_RCGCGPIO_R;
  switch(channelNum){
    case 0:                  //     Ain0 is on PE3
      GPIO_PORTE_DIR_R &= ~0x08;  // 3.0) make PE3 input
      GPIO_PORTE_AFSEL_R |= 0x08; // 4.0) enable alternate function on PE3
      GPIO_PORTE_DEN_R &= ~0x08;  // 5.0) disable digital I/O on PE3
      GPIO_PORTE_AMSEL_R |= 0x08; // 6.0) enable analog functionality on PE3
      break;
    case 1:                  //      Ain1 is on PE2
      GPIO_PORTE_DIR_R &= ~0x04;  // 3.1) make PE2 input
      GPIO_PORTE_AFSEL_R |= 0x04; // 4.1) enable alternate function on PE2
      GPIO_PORTE_DEN_R &= ~0x04;  // 5.1) disable digital I/O on PE2
      GPIO_PORTE_AMSEL_R |= 0x04; // 6.1) enable analog functionality on PE2
      break;
    case 2:                  //      Ain2 is on PE1
      GPIO_PORTE_DIR_R &= ~0x02;  // 3.2) make PE1 input
      GPIO_PORTE_AFSEL_R |= 0x02; // 4.2) enable alternate function on PE1
      GPIO_PORTE_DEN_R &= ~0x02;  // 5.2) disable digital I/O on PE1
      GPIO_PORTE_AMSEL_R |= 0x02; // 6.2) enable analog functionality on PE1
      break;
    case 3:                  //      Ain3 is on PE0
      GPIO_PORTE_DIR_R &= ~0x01;  // 3.3) make PE0 input
      GPIO_PORTE_AFSEL_R |= 0x01; // 4.3) enable alternate function on PE0
      GPIO_PORTE_DEN_R &= ~0x01;  // 5.3) disable digital I/O on PE0
      GPIO_PORTE_AMSEL_R |= 0x01; // 6.3) enable analog functionality on PE0
      break;
    case 4:                  //      Ain4 is on PD3
      GPIO_PORTD_DIR_R &= ~0x08;  // 3.4) make PD3 input
      GPIO_PORTD_AFSEL_R |= 0x08; // 4.4) enable alternate function on PD3
      GPIO_PORTD_DEN_R &= ~0x08;  // 5.4) disable digital I/O on PD3
      GPIO_PORTD_AMSEL_R |= 0x08; // 6.4) enable analog functionality on PD3
      break;
    case 5:                  //      Ain5 is on PD2
      GPIO_PORTD_DIR_R &= ~0x04;  // 3.5) make PD2 input
      GPIO_PORTD_AFSEL_R |= 0x04; // 4.5) enable alternate function on PD2
      GPIO_PORTD_DEN_R &= ~0x04;  // 5.5) disable digital I/O on PD2
      GPIO_PORTD_AMSEL_R |= 0x04; // 6.5) enable analog functionality on PD2
      break;
    case 6:                  //      Ain6 is on PD1
      GPIO_PORTD_DIR_R &= ~0x02;  // 3.6) make PD1 input
      GPIO_PORTD_AFSEL_R |= 0x02; // 4.6) enable alternate function on PD1
      GPIO_PORTD_DEN_R &= ~0x02;  // 5.6) disable digital I/O on PD1
      GPIO_PORTD_AMSEL_R |= 0x02; // 6.6) enable analog functionality on PD1
      break;
    case 7:                  //      Ain7 is on PD0
      GPIO_PORTD_DIR_R &= ~0x01;  // 3.7) make PD0 input
      GPIO_PORTD_AFSEL_R |= 0x01; // 4.7) enable alternate function on PD0
      GPIO_PORTD_DEN_R &= ~0x01;  // 5.7) disable digital I/O on PD0
      GPIO_PORTD_AMSEL_R |= 0x01; // 6.7) enable analog functionality on PD0
      break;
    case 8:                  //      Ain8 is on PE5
```

```
   GPIO_PORTE_DIR_R &= ~0x20;  // 3.8) make PE5 input
   GPIO_PORTE_AFSEL_R |= 0x20; // 4.8) enable alternate function on PE5
   GPIO_PORTE_DEN_R &= ~0x20;  // 5.8) disable digital I/O on PE5
   GPIO_PORTE_AMSEL_R |= 0x20; // 6.8) enable analog functionality on PE5
   break;
  case 9:                  //      Ain9 is on PE4
   GPIO_PORTE_DIR_R &= ~0x10;  // 3.9) make PE4 input
   GPIO_PORTE_AFSEL_R |= 0x10; // 4.9) enable alternate function on PE4
   GPIO_PORTE_DEN_R &= ~0x10;  // 5.9) disable digital I/O on PE4
   GPIO_PORTE_AMSEL_R |= 0x10; // 6.9) enable analog functionality on PE4
   break;
  case 10:                 //      Ain10 is on PB4
   GPIO_PORTB_DIR_R &= ~0x10;  // 3.10) make PB4 input
   GPIO_PORTB_AFSEL_R |= 0x10; // 4.10) enable alternate function on PB4
   GPIO_PORTB_DEN_R &= ~0x10;  // 5.10) disable digital I/O on PB4
   GPIO_PORTB_AMSEL_R |= 0x10; // 6.10) enable analog functionality on PB4
   break;
  case 11:                 //      Ain11 is on PB5
   GPIO_PORTB_DIR_R &= ~0x20;  // 3.11) make PB5 input
   GPIO_PORTB_AFSEL_R |= 0x20; // 4.11) enable alternate function on PB5
   GPIO_PORTB_DEN_R &= ~0x20;  // 5.11) disable digital I/O on PB5
   GPIO_PORTB_AMSEL_R |= 0x20; // 6.11) enable analog functionality on PB5
   break;
}
DisableInterrupts();
SYSCTL_RCGCADC_R |= 0x01;     // activate ADC0
SYSCTL_RCGCTIMER_R |= 0x01;   // activate timer0
delay = SYSCTL_RCGCTIMER_R;   // allow time to finish activating
TIMER0_CTL_R = 0x00000000;    // disable timer0A during setup
TIMER0_CTL_R |= 0x00000020;   // enable timer0A trigger to ADC
TIMER0_CFG_R = 0;            // configure for 32-bit timer mode
TIMER0_TAMR_R = 0x00000002;   // configure for periodic mode, default down-
                 count settings
TIMER0_TAPR_R = 0;           // prescale value for trigger
TIMER0_TAILR_R = 80000000/fs-1;    // start value for trigger assuming 80 MHz


TIMER0_IMR_R = 0x00000000;    // disable all interrupts
TIMER0_CTL_R |= 0x00000001;   // enable timer0A 32-b, periodic, no interrupts
ADC0_PC_R = 0x01;        // configure for 125K samples/sec
ADC0_SSPRI_R = 0x3210;    // sequencer 0 is highest, sequencer 3 is lowest
ADC0_ACTSS_R &= ~0x08;    // disable sample sequencer 3
ADC0_EMUX_R = (ADC0_EMUX_R&0xFFFF0FFF)+0x5000; // timer trigger event
ADC0_SSMUX3_R = channelNum;
ADC0_SSCTL3_R = 0x06;         // set flag and end
ADC0_IM_R |= 0x08;           // enable SS3 interrupts
```

```
  ADC0_ACTSS_R |= 0x08;         // enable sample sequencer 3
  NVIC_PRI4_R = (NVIC_PRI4_R&0xFFFF00FF)|0x00004000; //priority 2
  NVIC_EN0_R = 1<<17;           // enable interrupt 17 in NVIC
            NumberOfSamples = numberOfSamples;
            Buffer = buffer;
  EnableInterrupts();
}
volatile uint32_t ADCvalue;
void ADC0Seq3_Handler(void){
            static int i=0;
            long sr;
  ADC0_ISC_R = 0x08;         // acknowledge ADC sequence 3 completion
            sr = StartCritical();
  Buffer[i] = ADC0_SSFIFO3_R;  // 12-bit result
            if(i==NumberOfSamples){
                  Status=1;                                   //ADC
            conversion complete
                  ADC0_IM_R &= ~0x08;        // disable SS3 interrupts when
            buffer is full
            }else{
                  Status=0;                                   //ADC
            still filling buffer
                  i++;
            }
            EndCritical(sr);
}

int ADC_Status(void){
            return Status;
}

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: programmable using variable 'channelNum' [0:11]
// SS3 interrupts: enabled but not promoted to controller
void ADC_Open(uint32_t channelNum){
            volatile uint32_t delay;
  switch(channelNum){           // 1) activate clock
    case 0:
```

```
 case 1:
 case 2:
 case 3:
 case 8:
 case 9:                  //    these are on GPIO_PORTE
   SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R4; break;
 case 4:
 case 5:
 case 6:
 case 7:                  //    these are on GPIO_PORTD
   SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R3; break;
 case 10:
 case 11:                  //    these are on GPIO_PORTB
   SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1; break;
 default: return;            //    0 to 11 are valid channels on the LM4F120
}
delay = SYSCTL_RCGCGPIO_R;      // 2) allow time for clock to stabilize
delay = SYSCTL_RCGCGPIO_R;
switch(channelNum){
 case 0:                  //      Ain0 is on PE3
   GPIO_PORTE_DIR_R &= ~0x08;  // 3.0) make PE3 input
   GPIO_PORTE_AFSEL_R |= 0x08; // 4.0) enable alternate function on PE3
   GPIO_PORTE_DEN_R &= ~0x08;  // 5.0) disable digital I/O on PE3
   GPIO_PORTE_AMSEL_R |= 0x08; // 6.0) enable analog functionality on PE3
   break;
 case 1:                  //      Ain1 is on PE2
   GPIO_PORTE_DIR_R &= ~0x04;  // 3.1) make PE2 input
   GPIO_PORTE_AFSEL_R |= 0x04; // 4.1) enable alternate function on PE2
   GPIO_PORTE_DEN_R &= ~0x04;  // 5.1) disable digital I/O on PE2
   GPIO_PORTE_AMSEL_R |= 0x04; // 6.1) enable analog functionality on PE2
   break;
 case 2:                  //      Ain2 is on PE1
   GPIO_PORTE_DIR_R &= ~0x02;  // 3.2) make PE1 input
   GPIO_PORTE_AFSEL_R |= 0x02; // 4.2) enable alternate function on PE1
   GPIO_PORTE_DEN_R &= ~0x02;  // 5.2) disable digital I/O on PE1
   GPIO_PORTE_AMSEL_R |= 0x02; // 6.2) enable analog functionality on PE1
   break;
 case 3:                  //      Ain3 is on PE0
   GPIO_PORTE_DIR_R &= ~0x01;  // 3.3) make PE0 input
   GPIO_PORTE_AFSEL_R |= 0x01; // 4.3) enable alternate function on PE0
   GPIO_PORTE_DEN_R &= ~0x01;  // 5.3) disable digital I/O on PE0
   GPIO_PORTE_AMSEL_R |= 0x01; // 6.3) enable analog functionality on PE0
   break;
 case 4:                  //      Ain4 is on PD3
   GPIO_PORTD_DIR_R &= ~0x08;  // 3.4) make PD3 input
   GPIO_PORTD_AFSEL_R |= 0x08; // 4.4) enable alternate function on PD3
```

```
    GPIO_PORTD_DEN_R &= ~0x08;  // 5.4) disable digital I/O on PD3
    GPIO_PORTD_AMSEL_R |= 0x08; // 6.4) enable analog functionality on PD3
    break;
  case 5:                  //      Ain5 is on PD2
    GPIO_PORTD_DIR_R &= ~0x04;  // 3.5) make PD2 input
    GPIO_PORTD_AFSEL_R |= 0x04; // 4.5) enable alternate function on PD2
    GPIO_PORTD_DEN_R &= ~0x04;  // 5.5) disable digital I/O on PD2
    GPIO_PORTD_AMSEL_R |= 0x04; // 6.5) enable analog functionality on PD2
    break;
  case 6:                  //      Ain6 is on PD1
    GPIO_PORTD_DIR_R &= ~0x02;  // 3.6) make PD1 input
    GPIO_PORTD_AFSEL_R |= 0x02; // 4.6) enable alternate function on PD1
    GPIO_PORTD_DEN_R &= ~0x02;  // 5.6) disable digital I/O on PD1
    GPIO_PORTD_AMSEL_R |= 0x02; // 6.6) enable analog functionality on PD1
    break;
  case 7:                  //      Ain7 is on PD0
    GPIO_PORTD_DIR_R &= ~0x01;  // 3.7) make PD0 input
    GPIO_PORTD_AFSEL_R |= 0x01; // 4.7) enable alternate function on PD0
    GPIO_PORTD_DEN_R &= ~0x01;  // 5.7) disable digital I/O on PD0
    GPIO_PORTD_AMSEL_R |= 0x01; // 6.7) enable analog functionality on PD0
    break;
  case 8:                  //      Ain8 is on PE5
    GPIO_PORTE_DIR_R &= ~0x20;  // 3.8) make PE5 input
    GPIO_PORTE_AFSEL_R |= 0x20; // 4.8) enable alternate function on PE5
    GPIO_PORTE_DEN_R &= ~0x20;  // 5.8) disable digital I/O on PE5
    GPIO_PORTE_AMSEL_R |= 0x20; // 6.8) enable analog functionality on PE5
    break;
  case 9:                  //      Ain9 is on PE4
    GPIO_PORTE_DIR_R &= ~0x10;  // 3.9) make PE4 input
    GPIO_PORTE_AFSEL_R |= 0x10; // 4.9) enable alternate function on PE4
    GPIO_PORTE_DEN_R &= ~0x10;  // 5.9) disable digital I/O on PE4
    GPIO_PORTE_AMSEL_R |= 0x10; // 6.9) enable analog functionality on PE4
    break;
  case 10:                  //      Ain10 is on PB4
    GPIO_PORTB_DIR_R &= ~0x10;  // 3.10) make PB4 input
    GPIO_PORTB_AFSEL_R |= 0x10; // 4.10) enable alternate function on PB4
    GPIO_PORTB_DEN_R &= ~0x10;  // 5.10) disable digital I/O on PB4
    GPIO_PORTB_AMSEL_R |= 0x10; // 6.10) enable analog functionality on PB4
    break;
  case 11:                  //      Ain11 is on PB5
    GPIO_PORTB_DIR_R &= ~0x20;  // 3.11) make PB5 input
    GPIO_PORTB_AFSEL_R |= 0x20; // 4.11) enable alternate function on PB5
    GPIO_PORTB_DEN_R &= ~0x20;  // 5.11) disable digital I/O on PB5
    GPIO_PORTB_AMSEL_R |= 0x20; // 6.11) enable analog functionality on PB5
    break;
}
```

```
  SYSCTL_RCGC0_R |= 0x00010000;   // 7) activate ADC0 (legacy code)
//  SYSCTL_RCGCADC_R |= 0x00000001; // 7) activate ADC0 (actually doesn't
                        work)
  delay = SYSCTL_RCGC0_R;         // 8) allow time for clock to stabilize
  delay = SYSCTL_RCGC0_R;
//  SYSCTL_RCGC0_R &= ~0x00000300;  // 9) configure for 125K (legacy code)
  ADC0_PC_R &= ~0xF;              // 9) clear max sample rate field
  ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
  ADC0_SSPRI_R = 0x3210;          // 10) Sequencer 3 is lowest priority
  ADC0_ACTSS_R &= ~0x0008;        // 11) disable sample sequencer 3
  ADC0_EMUX_R &= ~0xF000;         // 12) seq3 is software trigger
  ADC0_SSMUX3_R &= ~0x000F;       // 13) clear SS3 field
  ADC0_SSMUX3_R += channelNum;    //     set channel
  ADC0_SSCTL3_R = 0x0006;         // 14) no TS0 D0, yes IE0 END0
  ADC0_IM_R &= ~0x0008;           // 15) disable SS3 interrupts
  ADC0_ACTSS_R |= 0x0008;         // 16) enable sample sequencer 3
}



//------------ADC_In------------
// Busy-wait Analog to digital conversion
// Input: none
// Output: 12-bit result of ADC conversion
uint16_t ADC_In(void){
                        uint32_t result;
  ADC0_PSSI_R = 0x0008;            // 1) initiate SS3
  while((ADC0_RIS_R&0x08)==0){};   // 2) wait for conversion done
    // if you have an A0-A3 revision number, you need to add an 8 usec wait here
  result = ADC0_SSFIFO3_R&0xFFF;   // 3) read result
  return result;
}

uint16_t TestBuffer[64];
int main(void){
                        char input_str[30];
                        int input_num,i,device,line;
                        int freq, numSamples;
                        PLL_Init();
                        UART_Init();            // initialize UART
                        Output_Init();                                          // initialize LCD
                        GPIO_PortF_Init();                              // initialize PortF
                        OutCRLF();
                        OutCRLF();
                        OS_AddPeriodicThread(&PF1_Toggle, 1, 100, 4); //Toggle PF1 at 10
                        Hz
```

```
OS_LaunchThread(&PF1_Toggle, 1);

//Print Interpreter Menu
printf("Debugging Interpreter Lab 1\n\r");
printf("Commands:\n\r");
printf("LCD\n\r");
printf("ADC_Open - must call before ADC_In\n\r");
printf("ADC_In\n\r");
printf("ADC_Collect\n\r");
printf("ADC_Status\n\r");
printf("OS-RTP - OS_ReadTimerPeriod\n\r");
printf("OS-RTV - OS_ReadTimerValue\n\r");
printf("OS-CPT - OS_ClearPeriodicTime\n\r");
printf("OS-ST - OS_StopThread\n\r");

while(1){
        printf("\n\rEnter a command:\n\r");
        for(i=0;input_str[i]!=0;i++){input_str[i]=0;}
        //Flush the input_str
        UART_InString(input_str,30);
        if(!strcmp(input_str,"LCD")){
                printf("\n\rMessage to Print: ");
                for(i=0;input_str[i]!=0;i++){input_str[i]=0;}
        //Flush the input_str
                UART_InString(input_str,30);
                printf("\n\rNumber to Print: ");
                input_num=UART_InUDec();
                printf("\n\rDevice to Print to: ");
                device = UART_InUDec();
                printf("\n\rLine to Print to: ");
                line = UART_InUDec();
                ST7735_Message(device,line,input_str,input_num);
        }

        else if(!strcmp(input_str,"ADC_Open")){
                printf("\n\rChannel to Open: ");
                input_num=UART_InUDec();
                ADC_Open(input_num);
        }

        else if(!strcmp(input_str,"ADC_In")){
                input_num=ADC_In();
                printf("\n\rSample from ADC: %d",input_num);
        }

        else if(!strcmp(input_str,"ADC_Collect")){
```

```
		printf("\n\rChannel to Open: ");
		input_num=UART_InUDec();
		printf("\n\rSampling Frequency: ");
		freq=UART_InUDec();
		printf("\n\rNumber of Samples: ");
		numSamples=UART_InUDec();
		ADC_Collect(input_num,freq,TestBuffer,numSamples);
	}

	else if(!strcmp(input_str,"ADC_Status")){
		int i;
		input_num = ADC_Status();
		if(input_num==0){
			printf("\n\rStatus: Busy");
		}else{
			printf("\n\rStatus: Done");
			printf("\n\rSamples Collected:");
			for(i=0;i<numSamples;i++){
				printf("\n\r%d",TestBuffer[i]);
			}
		}
	}

	else if(!strcmp(input_str,"OS-RTP")){
		printf("\n\rTimer to Read:");
		input_num = UART_InUDec();
		printf("\n\rCurrent Timer Period: ");
		printf("%d",OS_ReadTimerPeriod(input_num));
	}

	else if(!strcmp(input_str,"OS-RTV")){
		printf("\n\rTimer to Read:");
		input_num = UART_InUDec();
		printf("\n\rCurrent Timer Value: ");
		printf("%d",OS_ReadTimerValue(input_num));
	}

	else if(!strcmp(input_str,"OS-CPT")){
		printf("\n\rTimer to Clear:");
		input_num = UART_InUDec();
		OS_ClearPeriodicTime(input_num);
	}

	else if(!strcmp(input_str,"OS-ST")){
		printf("\n\rTimer to Stop:");
		input_num = UART_InUDec();
```

```
				OS_StopThread(&PF1_Toggle,input_num);
			}

			else{
				printf("\n\rInvalid Command. Try Again\n\r");
			}
		}
}

// initializes a new thread with given period and priority
int OS_AddPeriodicThread(void(*task)(void), int timer, unsigned long period,
		unsigned long priority)
{// period and priority are used when initializing the timer interrupts
		int status;
		int sr;
		sr = StartCritical();
		// initialize a timer specific to this thread
		// each timer should be unique to a thread so that it can interrupt
		when
		// it counts to 0 and sets the flag, this requires counting timers

		status = 0;
		status = OS_TimerInit(task,timer, period, priority);
		if(status == -1)
		{
			//printf("Error Initializing timer number(0-11): %d\n",
		timer);
		}
		EndCritical(sr);
		return 0;
}

// Resets the 32-bit counter to zero
void OS_ClearPeriodicTime(int timer)
{
		switch(timer)
		{
			case 0: // TIMERA0
				TIMER0_CTL_R &= ~TIMER_CTL_TAEN; // disable
		TimerA0
				TIMER0_TAV_R = 0; // set Timer to 0
				TIMER0_CTL_R |= TIMER_CTL_TAEN; // enable
		TimerA0
				break;

			case 1: // TIMERB0
			TIMER0_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB0
			TIMER0_TBV_R = 0; // set Timer to 0
			TIMER0_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB0
			break;

		case 2: // TIMERA1
			TIMER1_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA1
			TIMER1_TAV_R = 0; // set Timer to 0
			TIMER1_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA1
			break;

		case 3: // TIMERB1
			TIMER1_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB1
			TIMER1_TBV_R = 0; // set Timer to 0
			TIMER1_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB1
			break;

		case 4: // TIMERA2
			TIMER2_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA2
			TIMER2_TAV_R = 0; // set Timer to 0
			TIMER2_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA2
			break;

		case 5: // TIMERB2
			TIMER2_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB2
			TIMER2_TBV_R = 0; // set Timer to 0
			TIMER2_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB2
			break;

		case 6: // TIMERA3
			TIMER3_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA3
			TIMER3_TAV_R = 0;// set Timer to 0
			TIMER3_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA3
			break;
```

```
}

            case 7: // TIMERB3
                    TIMER3_CTL_R &= ~TIMER_CTL_TBEN; // disable
        TimerB3
                    TIMER3_TBV_R = 0; // set Timer to 0
                    TIMER3_CTL_R |= TIMER_CTL_TBEN; // enable
        TimerB3
                    break;

            case 8: // TIMERA4
                    TIMER4_CTL_R &= ~TIMER_CTL_TAEN; // disable
        TimerA4
                    TIMER4_TAV_R = 0; // set Timer to 0
                    TIMER4_CTL_R |= TIMER_CTL_TAEN; // enable
        TimerA4
                    break;

            case 9: // TIMERB4
                    TIMER4_CTL_R &= ~TIMER_CTL_TBEN; // disable
        TimerB4
                    TIMER4_TBV_R = 0; // set Timer to 0
                    TIMER4_CTL_R |= TIMER_CTL_TBEN; // enable
        TimerB4
                    break;

            case 10: // TIMERA5
                    TIMER5_CTL_R &= ~TIMER_CTL_TAEN; // disable
        TimerA5
                    TIMER5_TAV_R = 0; // set Timer to 0
                    TIMER5_CTL_R |= TIMER_CTL_TAEN; // enable
        TimerA5
                    break;

            case 11: // TIMERB5
                    TIMER5_CTL_R &= ~TIMER_CTL_TBEN; // disable
        TimerB5
                    TIMER5_TBV_R = 0; // set Timer to 0
                    TIMER5_CTL_R |= TIMER_CTL_TBEN; // enable
        TimerB5
                    break;

            default:
                    break;
        }
}
```

```
// Returns the number of bus cycles in a full period
unsigned long OS_ReadTimerPeriod(int timer)
{
        unsigned long busCyclePerPeriod = 0;
        switch(timer)
        {
            case 0:
                busCyclePerPeriod = TIMER0_TAILR_R;
                break;

            case 1:
                busCyclePerPeriod = TIMER0_TBILR_R;
                break;

            case 2:
                busCyclePerPeriod = TIMER1_TAILR_R;
                break;

            case 3:
                busCyclePerPeriod = TIMER1_TBILR_R;
                break;

            case 4:
                busCyclePerPeriod = TIMER2_TAILR_R;
                break;

            case 5:
                busCyclePerPeriod = TIMER2_TBILR_R;
                break;

            case 6:
                busCyclePerPeriod = TIMER3_TAILR_R;
                break;

            case 7:
                busCyclePerPeriod = TIMER3_TBILR_R;
                break;

            case 8:
                busCyclePerPeriod = TIMER4_TAILR_R;
                break;

            case 9:
                busCyclePerPeriod = TIMER4_TBILR_R;
                break;
```

```
				case 10:
					busCyclePerPeriod = TIMER5_TAILR_R;
					break;

				case 11:
					busCyclePerPeriod = TIMER5_TBILR_R;
					break;

				default:
					break;
			}
			return busCyclePerPeriod;
}

unsigned long OS_ReadTimerValue(int timer)
{
			unsigned long count = 0;
			switch(timer)
			{
				case 0: // TimerA0
					count =  TIMER0_TAR_R; // Read value, x, stored in
			Timer0, 0 < x < TIMER0_TAILR
					break;

				case 1: // TimerB0
					count = TIMER0_TBR_R; // Read value, x, stored in
			Timer0, 0 < x < TIMER0_TBILR
					break;

				case 2: // TimerA1
					count = TIMER1_TAR_R; // Read value, x, stored in
			Timer1, 0 < x < TIMER1_TAILR
					break;

				case 3: // TimerB1
					count = TIMER1_TBR_R; // Read value, x, stored in
			Timer1, 0 < x < TIMER1_TBILR
					break;

				case 4: // TimerA2
					count = TIMER2_TAR_R; // Read value, x, stored in
			Timer2, 0 < x < TIMER2_TAILR
					break;

				case 5: // TimerB2
				count = TIMER2_TBR_R; // Read value, x, stored in
		Timer2, 0 < x < TIMER2_TBILR
				break;

			case 6: // TimerA3
				count = TIMER3_TAR_R; // Read value, x, stored in
		Timer3, 0 < x < TIMER3_TAILR
				break;

			case 7: // TimerB3
				count = TIMER3_TBR_R; // Read value, x, stored in
		Timer3, 0 < x < TIMER3_TBILR
				break;

			case 8: // TimerA4
				count = TIMER4_TAR_R; // Read value, x, stored in
		Timer4, 0 < x < TIMER4_TAILR
				break;

			case 9: // TimerB4
				count = TIMER4_TBR_R; // Read value, x, stored in
		Timer4, 0 < x < TIMER4_TBILR
				break;

			case 10: // TimerA5
				count = TIMER5_TAR_R; // Read value, x, stored in
		Timer5, 0 < x < TIMER5_TAILR
				break;

			case 11: // timerB5
				count = TIMER5_TBR_R; // Read value, x, stored in
		Timer5, 0 < x < TIMER5_TBILR
				break;

			default:
				break;
		}
		return count;
}


// launches all programs
//void OS_LaunchAll(void(**taskPtrPtr)(void))
//{
////		int i;
////		int timerN;
```

```
////            unsigned int TAEN_TBEN;
////            for(i = 0; i <= timerCount; i++)
////            {
////                    timerN = usedTimers[i]; // its the timerID 0 to 11
////                    TAEN_TBEN = ((timerN%2) ?
                TIMER_CTL_TBEN:TIMER_CTL_TAEN); // if timerN even => TAEN,
                timerN odd => TBEN
////                    *(timerCtrlBuf[timerN]) |= TAEN_TBEN;
////            }
//
//              // this assumes that the timers initialized in the same sequence as the
                tasks
//              // i.e. usedTimer[i] was initialized for the functionPtr *(taskPtrPtr[i])
//              int i;
//              for(i = 0; i <= timerCount; i++)
//              {
//                      OS_LaunchThread(taskPtrPtr[i],usedTimers[i]);
//              }
//}

// enables interrupts in the NVIC vector table
void OS_NVIC_EnableTimerInt(int timer)
{
                switch(timer)
                {
                        case 0: // Timer0A
                                NVIC_EN0_R = NVIC_EN0_INT19;
                                break;

                        case 1: // Timer0B
                                NVIC_EN0_R = NVIC_EN0_INT20;
                                break;

                        case 2: // Timer1A
                                NVIC_EN0_R = NVIC_EN0_INT21;
                                break;

                        case 3: // Timer1B
                                NVIC_EN0_R = NVIC_EN0_INT22;
                                break;

                        case 4: // Timer2A
                                NVIC_EN0_R = NVIC_EN0_INT23;
                                break;

                        case 5: // Timer2B
                                NVIC_EN0_R = NVIC_EN0_INT24;
                                break;

                        case 6: // Timer3A
                                NVIC_EN1_R = NVIC_EN1_INT35;
                                break;

                        case 7: // Timer3B
                                NVIC_EN1_R = NVIC_EN1_INT36;
                                break;

                        case 8: // Timer4A
                                NVIC_EN2_R = NVIC_EN2_INT70;
                                break;

                        case 9: // Timer4B
                                NVIC_EN2_R = NVIC_EN2_INT71;
                                break;

                        case 10: // Timer5A
                                NVIC_EN2_R = NVIC_EN2_INT92;
                                break;
                        case 11: // Timer5B
                                NVIC_EN2_R = NVIC_EN2_INT93;
                                break;
                }
}


void OS_NVIC_DisableTimerInt(int timer)
{
                switch(timer)
                {
                        case 0: // Timer0A
                                NVIC_DIS0_R = NVIC_DIS0_INT19;
                                break;

                        case 1: // Timer0B
                                NVIC_DIS0_R = NVIC_DIS0_INT20;
                                break;

                        case 2: // Timer1A
                                NVIC_DIS0_R = NVIC_DIS0_INT21;
                                break;

                        case 3: // Timer1B
```

```
                NVIC_DIS0_R = NVIC_DIS0_INT22;
                break;

            case 4: // Timer2A
                NVIC_DIS0_R = NVIC_DIS0_INT23;
                break;

            case 5: // Timer2B
                NVIC_DIS0_R = NVIC_DIS0_INT24;
                break;

            case 6: // Timer3A
                NVIC_DIS1_R = NVIC_DIS1_INT35;
                break;

            case 7: // Timer3B
                NVIC_DIS1_R = NVIC_DIS1_INT36;
                break;

            case 8: // Timer4A
                NVIC_DIS2_R = NVIC_DIS2_INT70;
                break;

            case 9: // Timer4B
                NVIC_DIS2_R = NVIC_DIS2_INT71;
                break;

            case 10: // Timer5A
                NVIC_DIS2_R = NVIC_DIS2_INT92;
                break;
            case 11: // Timer5B
                NVIC_DIS2_R = NVIC_DIS2_INT93;
                break;
        }
}


void OS_LaunchThread(void(*taskPtr)(void), int timer)
{
        int timerN;
        unsigned int TAEN_TBEN;

        timerN = timer; // its the timerID (0 to 11)
        OS_NVIC_EnableTimerInt(timer);
        TAEN_TBEN = ((timerN%2) ? TIMER_CTL_TBEN : TIMER_CTL_TAEN);
        // if timerN even => TAEN, timerN odd => TBEN
```

```
		*(timerCtrlBuf[timerN]) |= TAEN_TBEN; // start timer for this thread,
		X in {0,1,2,3,4,5}, x in {A,B}
		// ^^^ is *(TIMERX_CTRL_PTR_R) |= TIMER_CTL_TxEN and/or
		TIMERX_CTL_R |= TIMER_CTL_TxEN
		//(*taskPtr)(); // begin task for this thread Do this in the ISR
}

void OS_StopThread(void(*taskPtr)(void), int timer)
{
		int timerN;
		unsigned int TAEN_TBEN;

		timerN = timer; // its the timerID (0 to 11)
		OS_NVIC_DisableTimerInt(timer);
		TAEN_TBEN = ((timerN%2) ? TIMER_CTL_TBEN : TIMER_CTL_TAEN);
		// if timerN even => TAEN, timerN odd => TBEN
		*(timerCtrlBuf[timerN]) &= ~TAEN_TBEN; // start timer for this
		thread, X in {0,1,2,3,4,5}, x in {A,B}
		// ^^^ is *(TIMERX_CTRL_PTR_R) |= TIMER_CTL_TxEN and/or
		TIMERX_CTL_R |= TIMER_CTL_TxEN
		//(*taskPtr)(); // begin task for this thread Do this in the ISR
}

// this configures the timers for 32-bit mode, periodic mode
//
int OS_TimerInit(void(*task)(void), int timer, unsigned long desiredFrequency,
		unsigned long priority)
{
		int delay;
		unsigned long cyclesPerPeriod;

		// will fail if frequency is a decimal number close to 0 relative to the
		bus speed
		cyclesPerPeriod = CLOCKSPEED_80MHZ/desiredFrequency;

		switch(timer)
		{
			case 0:		//TimerA0
				SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R0;   //
		activate timer0
				delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
		activating
				TIMER0_CTL_R &= ~TIMER_CTL_TAEN; // disable
		TimerA0
				TIMER0_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
		configure for 32-bit mode
```

```
                    TIMER0_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
                    TIMER0_TAILR_R = cyclesPerPeriod-1;
                    TIMER0_TAPR_R = 0; // set prescale = 0
                    TIMER0_ICR_R = TIMER_ICR_TATOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
                    TIMER0_IMR_R |= TIMER_IMR_TATOIM; // arm the
timeout interrupt
                    NVIC_PRI4_R = (NVIC_PRI4_R &
~NVIC_PRI4_INT19_M)|(priority << NVIC_PRI4_INT19_S); //clears
PRI bits then shifts the mask into the appropriate place
                    //TIMER0_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA0, do this in OS_Launch(.)
                    HandlerTaskArray[0] = task; // fill function pointer
array w/address of task
                    break;

          case 1:          //TimerB0
                    SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R0;   //
activate timer0
                    delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
           TIMER0_CTL_R &= ~TIMER_CTL_TBEN; // disable TimerB0
                    TIMER0_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
                    TIMER0_TBMR_R = TIMER_TBMR_TBMR_PERIOD;
                    TIMER0_TBILR_R = cyclesPerPeriod-1;
                    TIMER0_TBPR_R = 0; // set prescale = 0
                    TIMER0_ICR_R = TIMER_ICR_TBTOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
                    TIMER0_IMR_R |= TIMER_IMR_TBTOIM; // arm the
timeout interrupt
                    NVIC_PRI5_R = (NVIC_PRI5_R &
~NVIC_PRI5_INT20_M)|(priority << NVIC_PRI5_INT20_S); //clears
PRI bits then shifts the mask into the appropriate place
                    //TIMER0_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB0, do this in OS_Launch(.)
                    HandlerTaskArray[1] = task; // fill function pointer
array w/address of task
                    break;

          case 2:          //TimerA1
                    SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R1;   //
activate timer1
                    delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
```

```
			TIMER1_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA1
			TIMER1_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
			TIMER1_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
			TIMER1_TAILR_R = cyclesPerPeriod-1;
			TIMER1_TAPR_R = 0; // set prescale = 0
			TIMER1_ICR_R = TIMER_ICR_TATOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
			TIMER1_IMR_R |= TIMER_IMR_TATOIM; // arm the
timeout interrupt
			NVIC_PRI5_R = (NVIC_PRI5_R &
~NVIC_PRI5_INT21_M)|(priority << NVIC_PRI5_INT21_S); //clears
PRI bits then shifts the mask into the appropriate place
			//TIMER1_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA1, do this in OS_Launch(.)
			HandlerTaskArray[2] = task; // fill function pointer
array w/address of task
			break;

		case 3:		//TimerB1
			SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R1;   //
activate timer1
			delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
			TIMER1_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB1
			TIMER1_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
			TIMER1_TBMR_R = TIMER_TBMR_TBMR_PERIOD;
			TIMER1_TBILR_R = cyclesPerPeriod-1;
			TIMER1_TBPR_R = 0; // set prescale = 0
			TIMER1_ICR_R = TIMER_ICR_TBTOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
			TIMER1_IMR_R |= TIMER_IMR_TBTOIM; // arm the
timeout interrupt
			NVIC_PRI5_R = (NVIC_PRI5_R &
~NVIC_PRI5_INT22_M)|(priority << NVIC_PRI5_INT22_S);
			//TIMER1_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB1, do this in OS_Launch(.)
			HandlerTaskArray[3] = task; // fill function pointer
array w/address of tasks
			break;

		case 4:		//TimerA2
```

```
                SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R2;   //
activate timer2
                delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
                TIMER2_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA2
                TIMER2_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
                TIMER2_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
                TIMER2_TAILR_R = cyclesPerPeriod-1;
                TIMER2_TAPR_R = 0; // set prescale = 0
                TIMER2_ICR_R = TIMER_ICR_TATOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
                TIMER2_IMR_R |= TIMER_IMR_TATOIM; // arm the
timeout interrupt
                NVIC_PRI5_R = (NVIC_PRI5_R &
~NVIC_PRI5_INT23_M)|(priority << NVIC_PRI5_INT23_S); //clears
PRI bits then shifts the mask into the appropriate place
                //TIMER2_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA2, do this in OS_Launch(.)
                HandlerTaskArray[4] = task; // fill function pointer
array w/address of tasks
                break;

        case 5:         //TimerB2
                SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R2;   //
activate timer2
                delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
                TIMER2_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB2
                TIMER2_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
                TIMER2_TBMR_R = TIMER_TBMR_TBMR_PERIOD;
                TIMER2_TBILR_R = cyclesPerPeriod-1;
                TIMER2_TBPR_R = 0; // set prescale = 0
                TIMER2_ICR_R = TIMER_ICR_TBTOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
                TIMER2_IMR_R |= TIMER_IMR_TBTOIM; // arm the
timeout interrupt
                NVIC_PRI6_R = (NVIC_PRI6_R &
~NVIC_PRI6_INT24_M)|(priority << NVIC_PRI6_INT24_S); //clears
PRI bits then shifts the mask into the appropriate place
                //TIMER2_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB2, do this in OS_Launch(.)
```

```
                    HandlerTaskArray[5] = task; // fill function pointer
array w/address of tasks
                    break;

          case 6:            //TimerA3
                    SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R3;   //
activate timer3
                    delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
                    TIMER3_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA3
                    TIMER3_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
                    TIMER3_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
                    TIMER3_TAILR_R = cyclesPerPeriod-1;
                    TIMER3_TAPR_R = 0; // set prescale = 0
                    TIMER3_ICR_R = TIMER_ICR_TATOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
                    TIMER3_IMR_R |= TIMER_IMR_TATOIM; // arm the
timeout interrupt
                    NVIC_PRI8_R = (NVIC_PRI8_R &
~NVIC_PRI8_INT35_M)|(priority << NVIC_PRI8_INT35_S); //clears
PRI bits then shifts the mask into the appropriate place
                    //TIMER3_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA3, do this in OS_Launch(.)
                    HandlerTaskArray[6] = task; // fill function pointer
array w/address of tasks
                    break;

          case 7:            //TimerB3
                    SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R3;   //
activate timer3
                    delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
                    TIMER3_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB3
                    TIMER3_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
                    TIMER3_TBMR_R = TIMER_TBMR_TBMR_PERIOD;
                    TIMER3_TBILR_R = cyclesPerPeriod-1;
                    TIMER3_TBPR_R = 0; // set prescale = 0
                    TIMER3_ICR_R = TIMER_ICR_TBTOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
                    TIMER3_IMR_R |= TIMER_IMR_TBTOIM; // arm the
timeout interrupt
```

```
            NVIC_PRI9_R = (NVIC_PRI9_R &
~NVIC_PRI9_INT36_M)|(priority << NVIC_PRI9_INT36_S); //clears
PRI bits then shifts the mask into the appropriate place
            //TIMER3_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB3, do this in OS_Launch(.)
            HandlerTaskArray[7] = task; // fill function pointer
array w/address of tasks
            break;

      case 8:         //TimerA4
            SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R4;   //
activate timer4
            delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
            TIMER4_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA4
            TIMER4_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
            TIMER4_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
            TIMER4_TAILR_R = cyclesPerPeriod-1;
            TIMER4_TAPR_R = 0; // set prescale = 0
            TIMER4_ICR_R = TIMER_ICR_TATOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
            TIMER4_IMR_R |= TIMER_IMR_TATOIM; // arm the
timeout interrupt
            NVIC_PRI17_R = (NVIC_PRI17_R &
~NVIC_PRI17_INTC_M)|(priority << NVIC_PRI17_INTC_S); //clears
PRI bits then shifts the mask into the appropriate place
            //TIMER4_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA4, do this in OS_Launch(.)
            HandlerTaskArray[8] = task; // fill function pointer
array w/address of tasks
            break;

      case 9:         //TimerB4
            SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R4;   //
activate timer4
            delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
            TIMER4_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB4
            TIMER4_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
            TIMER4_TBMR_R = TIMER_TBMR_TBMR_PERIOD;
            TIMER4_TBILR_R = cyclesPerPeriod-1;
            TIMER4_TBPR_R = 0; // set prescale = 0
```

```
			TIMER4_ICR_R = TIMER_ICR_TBTOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
			TIMER4_IMR_R |= TIMER_IMR_TBTOIM; // arm the
timeout interrupt
			NVIC_PRI17_R = (NVIC_PRI17_R &
~NVIC_PRI17_INTD_M)|(priority << NVIC_PRI17_INTD_S); //clears
PRI bits then shifts the mask into the appropriate place
			//TIMER4_CTL_R |= TIMER_CTL_TBEN; // enable
TimerB4, do this in OS_Launch(.)
			HandlerTaskArray[9] = task; // fill function pointer
array w/address of tasks
			break;

		case 10:		//TimerA5
			SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R5;   //
activate timer5
			delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
			TIMER5_CTL_R &= ~TIMER_CTL_TAEN; // disable
TimerA5
			TIMER5_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
configure for 32-bit mode
			TIMER5_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
			TIMER5_TAILR_R = cyclesPerPeriod-1;
			TIMER5_TAPR_R = 0; // set prescale = 0
			TIMER5_ICR_R = TIMER_ICR_TATOCINT; // clear
timeout flag, friendly since writing a 0 does nothing
			TIMER5_IMR_R |= TIMER_IMR_TATOIM; // arm the
timeout interrupt
			NVIC_PRI23_R = (NVIC_PRI23_R &
~NVIC_PRI23_INTA_M)|(priority << NVIC_PRI23_INTA_S); //clears
PRI bits then shifts the mask into the appropriate place
			//TIMER5_CTL_R |= TIMER_CTL_TAEN; // enable
TimerA5, do this in OS_Launch(.)
			HandlerTaskArray[10] = task; // fill function pointer
array w/address of tasks
			break;

		case 11:		//TimerB5
			SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R5;   //
activate timer5
			delay = SYSCTL_RCGCTIMER_R;   // allow time to finish
activating
			TIMER5_CTL_R &= ~TIMER_CTL_TBEN; // disable
TimerB5
```

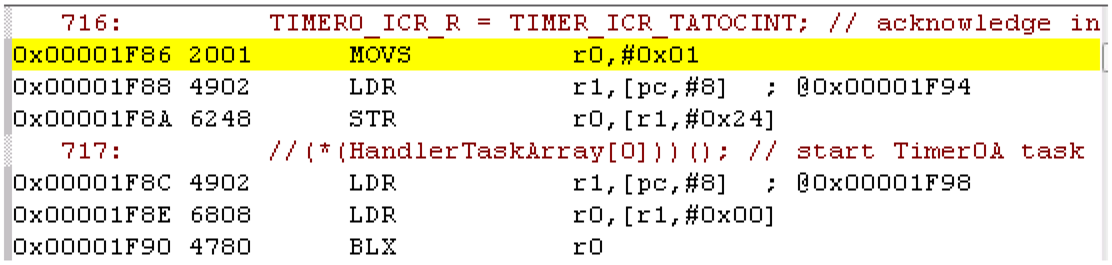
```
                        TIMER5_CFG_R  = TIMER_CFG_32_BIT_TIMER; //
                configure for 32-bit mode
                        TIMER5_TBMR_R = TIMER_TBMR_TBMR_PERIOD;
                        TIMER5_TBILR_R = cyclesPerPeriod-1;
                        TIMER5_TBPR_R = 0; // set prescale = 0
                        TIMER5_ICR_R = TIMER_ICR_TBTOCINT; // clear
                timeout flag, friendly since writing a 0 does nothing
                        TIMER5_IMR_R |= TIMER_IMR_TBTOIM; // arm the
                timeout interrupt
                        NVIC_PRI23_R = (NVIC_PRI23_R &
                ~NVIC_PRI23_INTB_M)|(priority << NVIC_PRI23_INTB_S); //clears
                PRI bits then shifts the mask into the appropriate place
                        //TIMER5_CTL_R |= TIMER_CTL_TBEN; // enable
                TimerB5, do this in OS_Launch(.)
                        HandlerTaskArray[11] = task; // fill function pointer
                array w/address of tasks
                        break;

                    default:
                        return -1;
                }

                timerCount++; // used for launching the correct number of threads
                that were successfully initialized
                usedTimers[timerCount] = timer; // store the timerID of which timer
                to launch
                return 0;
}



void GPIO_PortF_Init(void)
{
                unsigned long delay;
                SYSCTL_RCGCGPIO_R |= SYSCTL_RCGC2_GPIOF;
                delay = SYSCTL_RCGCGPIO_R;

                GPIO_PORTF_DIR_R |= 0x0F; // PF0-3 output
                GPIO_PORTF_DEN_R |= 0x0F; // enable Digital IO on PF0-3
                GPIO_PORTF_AFSEL_R &= ~0x0F; // PF0-3 alt funct disable
                GPIO_PORTF_AMSEL_R &= ~0x0F; // disable analog functionality on
                PF0-3

                GPIO_PORTF_DATA_R = 0x06;
}
```

```
void PF0_Toggle(void)
{
			GPIO_PORTF_DATA_R ^= 0x01;
}

void PF1_Toggle(void)
{
				GPIO_PORTF_DATA_R ^= 0x02;
}
void PF2_Toggle(void)
{
			GPIO_PORTF_DATA_R ^= 0x04;
}

void PF3_Toggle(void)
{
				GPIO_PORTF_DATA_R ^= 0x08;
}

#ifdef TESTING
int main(void)
{

   int timer0;
   int priority0;
   int period0;

   int timer1;
   int priority1;
   int period1;

   int timer2;
   int priority2;
   int period2;

   int timer3;
   int priority3;
   int period3;

   timer0 = 0;
   priority0 = 0;
   period0 = 10;

   timer1 = 2;
   priority1 = 0;
   period1 = 1000;

                                timer2 = 4;
   priority2 = 0;
   period2 = 1000;

   timer3 = 6;
   priority3 = 0;
   period3 = 65;

                                GPIO_PortF_Init();
                                PLL_Init();
                                //PF2_Toggle();
   OS_AddPeriodicThread(&PF1_Toggle, timer0, period0, priority0);
                                //PF2_Toggle();
 //   OS_AddPeriodicThread(&PF2_Toggle, timer1, period1, priority1);
                                //OS_AddPeriodicThread(&PF2_Toggle, timer2, period2,
                        priority2);
//    OS_AddPeriodicThread(&PF3_Toggle, timer3, period3, priority3);

                                      OS_LaunchThread(&PF1_Toggle, timer0);
//    OS_LaunchThread(&PF2_Toggle, timer1);
 //   OS_LaunchThread(&PF2_Toggle, timer2);
//    OS_LaunchThread(&PF3_Toggle, timer3);
                                while(1)
                                {;
                                }

   return 0;
}
#endif
void Timer0A_Handler(void)
{
                        TIMER0_ICR_R = TIMER_ICR_TATOCINT; // acknowledge interrupt
                        flag
                        (*(HandlerTaskArray[0]))(); // start Timer0A task
}

void Timer0B_Handler(void)
{
                        TIMER0_ICR_R = TIMER_ICR_TBTOCINT; // acknowledge interrupt
                        flag
                        (*(HandlerTaskArray[1]))(); // start Timer0B task

}
```

```
void Timer1A_Handler(void)
{
                TIMER1_ICR_R = TIMER_ICR_TATOCINT; // acknowledge interrupt
                flag
                (*(HandlerTaskArray[2]))(); // start Timer1A task

}

void Timer1B_Handler(void)
{
                TIMER1_ICR_R = TIMER_ICR_TBTOCINT; // acknowledge interrupt
                flag
                (*(HandlerTaskArray[3]))(); // start Timer1B task

}

void Timer2A_Handler(void)
{
                TIMER2_ICR_R = TIMER_ICR_TATOCINT; // acknowledge interrupt
                flag
                (*(HandlerTaskArray[4]))(); // start Timer2A task

}

void Timer2B_Handler(void)
{
                TIMER2_ICR_R = TIMER_ICR_TBTOCINT; // acknowledge interrupt
                flag
                (*(HandlerTaskArray[5]))(); // start Timer2B task

}

void Timer3A_Handler(void)
{
                TIMER3_ICR_R = TIMER_ICR_TATOCINT; // acknowledge interrupt
                flag
                (*(HandlerTaskArray[6]))(); // start Timer3A task

}

void Timer3B_Handler(void)
{
                TIMER3_ICR_R = TIMER_ICR_TBTOCINT; // acknowledge interrupt
                flag
                (*(HandlerTaskArray[7]))(); // start Timer3B task
}

void Timer4A_Handler(void)
{
		TIMER4_ICR_R = TIMER_ICR_TATOCINT; // acknowledge interrupt
		flag
		(*(HandlerTaskArray[8]))(); // start Timer4A task

}

void Timer4B_Handler(void)
{
		TIMER4_ICR_R = TIMER_ICR_TBTOCINT; // acknowledge interrupt
		flag
		(*(HandlerTaskArray[9]))(); // start Timer4B task

}

void Timer5A_Handler(void)
{
		TIMER5_ICR_R = TIMER_ICR_TATOCINT; // acknowledge interrupt
		flag
		(*(HandlerTaskArray[10]))(); // // start Timer5A task

}

void Timer5B_Handler(void)
{
		TIMER5_ICR_R = TIMER_ICR_TBTOCINT; // acknowledge interrupt
		flag
		(*(HandlerTaskArray[11]))(); // start Timer5B task

}
```

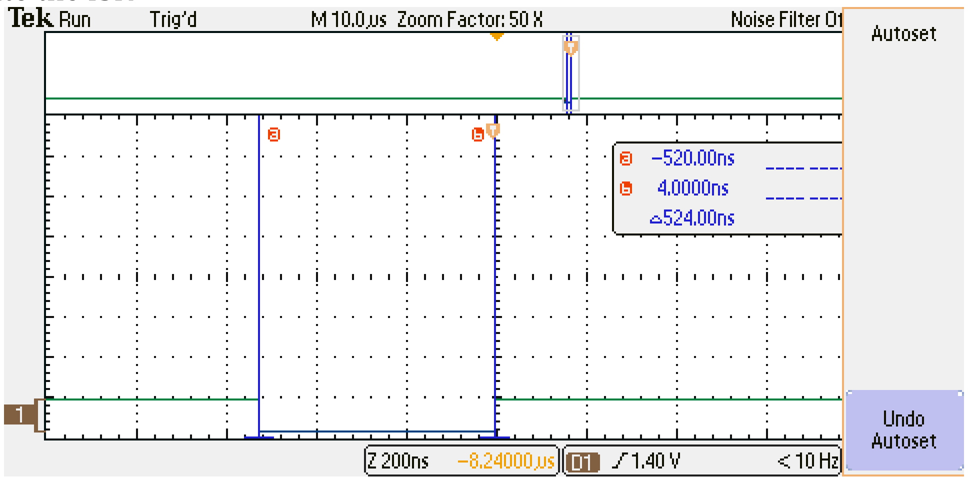
Measurement Data:

1) Estimated time to run periodic interrupt. The ISR was 6 assembly instructions and based on the assumption an instruction occurs every cycle it would be estimated at 6*ClockPeriod, where Clock Period = 12.5ns. Estimated time is 75ns.

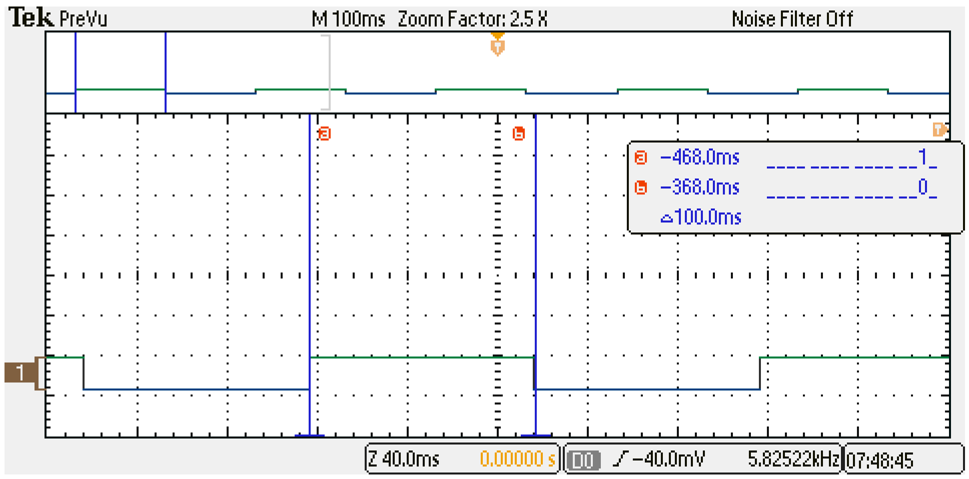

```
   716:          TIMER0_ICR_R = TIMER_ICR_TATOCINT; // acknowledge in
0x00001F86 2001      MOVS          r0,#0x01
0x00001F88 4902      LDR           r1,[pc,#8]   ; @0x00001F94
0x00001F8A 6248      STR           r0,[r1,#0x24]
   717:          //(*(HandlerTaskArray[0]))(); // start Timer0A task
0x00001F8C 4902      LDR           r1,[pc,#8]   ; @0x00001F98
0x00001F8E 6808      LDR           r0,[r1,#0x00]
0x00001F90 4780      BLX           r0
```

2) Measured time to run the ISR was 524 ns as can be seen by the Logic Analyzer snapshot. This deviation from the estimate can be attributed to the Reads and Writes in the ISR that access memory and take many more than a single cycle. Specifically, we do a write to acknowledge the interrupt. Also system operations occur even though the function is a 'do-nothing' task and it still requires pushing and popping the system state (Registers, LR, PC) before and after the function call. This measurement neglects the system overhead required to call the ISR since we profiled the ISR inside the Handler by setting and clearing a bit as the first and last instructions. This also introduces more time on the clearing of the bit since that requires an additional write to memory that is not part of the function of the ISR.

Time to execute the ISR



Time between ISRs



Analysis and Discussion:

1) What are the range, resolution, and precision of the ADC?

   The range = a*resolution*precision + b, a,b are constants which scale the range from [-1, 1) to [-a+b, a+b). The range essentially maps from ADC values to the desired output value range (usually volts e.g. –Vdd to +Vdd). The range of our ADC is 0 to 4096 or 0 to 3.3 V

   Precision is the total number of distinguishable values, precision = 2^n, where n = # of ADC bits used. The precision for the 12-bit TM4C123 ADC, the precision is 4096 alternatives.

   Resolution is the smallest change that can be represented. For an n bit ADC that is 2^-n. The resolution of our ADC is 1/4096 or 0.8 mV.

2) List the ways the ADC conversion can be started. Explain why you chose the way you did.

The first approach to triggering an ADC sampling sequence is to use a periodic timer such as SysTick or one of the Timer modules to initiate an ADC conversion interrupt when the time rolls over. Another approach is to use software to trigger an ADC conversion by simply reading from the ADC FIFO built into hardware. This approach involves busy-wait synchronization as software triggers the conversion and waits for its completion. A third approach to triggering ADC conversion is edge-triggered interrupts from switches or external digital signals. This implementation is the same as for periodic timers. Our ADC modules implemented periodic timer ADC interrupts and software-triggered busy-wait synchronization. Periodic timer ADC interrupts should be used when an accurate sampling rate is necessary such as in digital acquisition systems. The software-triggered ADC is used for acquiring a single sample for debugging purposes.

3) The measured time to run the periodic interrupt can be measured directly by setting a bit high at the start of the ISR and clearing that bit at the end of the ISR. It could also be measured indirectly by measuring the time lost when running a simple main program that toggles an output pin. How did you measure it? Compare and contrast you method to these two.

We measured the time for the periodic interrupt by placing the debugging instruments in the ISR itself. This method is easier to implement and can be used when multiple interrupts are enabled. Toggling in the main program and measuring time lost includes the overhead of the time for a context switch but cannot be used if multiple interrupts are enabled.

4) Divide the time to execute one instance of the ISR by the total instructions in the ISR to get the average time to execute an instruction. Compare this to the 12.5 ns system clock period (80MHz).

   524 ns/6 instructions = 87.33 ns/instruction
   => (87.33 ns/instruction)/ (12.5 ns/cycle) ~ 7 cycles/instruction

5) What are the range, resolution and precision of the SysTick Timer? i.e. answer the question relative to the NVIC_ST_CURRENT_R register in the Cortex M4 core peripherals.

   Precision is 2^24 distinguishable values. Resolution is 12.5 ns (1 clock cycle).
   Range = (2^24)*(12.5 ns) ~ 210 ms