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)

{

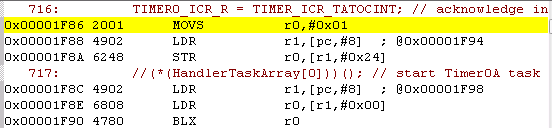
TIMER5\_ICR\_R = TIMER\_ICR\_TBTOCINT; // acknowledge interrupt flag

(\*(HandlerTaskArray[11]))(); // start Timer5B task

}

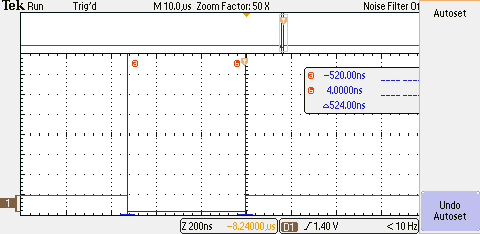
**Measurement Data:**

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.

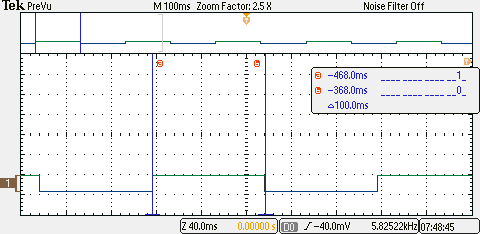


2) Measured time to run the ISR was 524 ns as can be seen by the Logic Analyzer snapshot. This deviation from the estimate can be attributed to the Reads and Writes in the ISR that access memory and take many more than a single cycle. Specifically, we do a write to acknowledge the interrupt. Also system 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.

1. 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.

1. 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.

1. 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

1. 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