Jonathan W. Valvano First Name: _____ Last Name:_____
March 2, 2012, 10:00 to 10:50am

Quiz 1 is a closed book exam. You may have one 8.5 by 11 inch sheet of hand-written crib notes, but no books or electronic devices. You may put answers on the backs of the pages.

(10) Question 0. Please staple your crib sheet to your exam. Your crib sheet will be graded on content and correctness.

(4) Question 1. There is a sampling capacitor at the input of most ADC converters. The ADC is started by temporarily connecting the input voltage to this capacitor using a transistor switch. This transistor switch is controlled by a trigger signal. This trigger establishes when the ADC is started. In a real-time data acquisition it is important to control timing of this trigger. The LM3S8962 has eight different ways the ADC can be configured to trigger. List **four** of the ways.

(6) Question 2. For the robot you will have four IR distance sensors; each has an analog signal related to the distance to the nearest object. In order to drive the robot straight down the track, your partner determined that all four sensors must be sampled 50 times a second. The sensors will be attached to ADC channels 0, 1, 2, and 3. The autopilot requires all four distance measurements at the same time. You are using sequencer 0 triggered with a period timer; an ADC interrupt should occur after the 4 conversions are complete. What initialization values would you put in ADCSSCTL0 and ADCSSMUX0? You do not need to write other code; rather just specify the 32-bit values for these two registers. (See data sheets on the following pages).

ADC0_SSCTL0_R = [ ] // ADCSSCTL0

ADC0_SSMUX0_R = [ ] //ADCSSMUX0

ADC Sample Sequence Control 0 (ADCSSCTL0)

Base 0x4003.8000
Offset 0x044
Type R/W, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TS7	IE7	END7	D7	TS6	IE6	END6	D6	TS5	IE5	END5	D5	TS4	IE4	END4	D4
Type R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS3	IE3	END3	D3	TS2	IE2	END2	D2	TS1	IE1	END1	D1	TS0	IE0	END0	D0
Type R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31	TS7	R/W	0	8th Sample Temp Sensor Select This bit is used during the eighth sample of the sample sequence and and specifies the input source of the sample. When set, the temperature sensor is read. When clear, the input pin specified by the **ADCSSMUX** register is read.
30	IE7	R/W	0	8th Sample Interrupt Enable This bit is used during the eighth sample of the sample sequence and specifies whether the raw interrupt signal (`INR0` bit) is asserted at the end of the sample's conversion. If the `MASK0` bit in the **ADCIM** register is set, the interrupt is promoted to a controller-level interrupt. When this bit is set, the raw interrupt is asserted. When this bit is clear, the raw interrupt is not asserted. It is legal to have multiple samples within a sequence generate interrupts.
29	END7	R/W	0	8th Sample is End of Sequence The `END7` bit indicates that this is the last sample of the sequence. It is possible to end the sequence on any sample position. Samples defined after the sample containing a set END are not requested for conversion even though the fields may be non-zero. It is required that software write the `END` bit somewhere within the sequence. (Sample Sequencer 3, which only has a single sample in the sequence, is hardwired to have the `END0` bit set.) Setting this bit indicates that this sample is the last in the sequence.
28	D7	R/W	0	8th Sample Diff Input Select The D7 bit indicates that the analog input is to be differentially sampled. The corresponding **ADCSSMUXx** nibble must be set to the pair number "i", where the paired inputs are "2i and 2i+1". The temperature sensor does not have a differential option. When set, the analog inputs are differentially sampled.

… bits 27 to 4 are similar to 31-28 and 3-0

3	TS0	R/W	0	1st Sample Temp Sensor Select Same definition as `TS7` but used during the first sample.
2	IE0	R/W	0	1st Sample Interrupt Enable Same definition as `IE7` but used during the first sample.
1	END0	R/W	0	1st Sample is End of Sequence Same definition as `END7` but used during the first sample.
0	D0	R/W	0	1st Sample Diff Input Select Same definition as `D7` but used during the first sample.

ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0)

Base 0x4003.8000
Offset 0x040
Type R/W, reset 0x0000.0000

Bit/Field	Name	Type	Reset	Description
31:30	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
29:28	MUX7	R/W	0x0	8th Sample Input Select The `MUX7` field is used during the eighth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. The value set here indicates the corresponding pin, for example, a value of 1 indicates the input is `ADC1`.

… bits MUX6 to MUX1 are similar to bits MUX7 and MUX0

3:2	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
1:0	MUX0	R/W	0x0	1st Sample Input Select The `MUX0` field is used during the first sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion.

(5) Question 3. Is it possible on the Cortex M3 to perform a memory read cycle fetching an op code at the very same instant as it performs a memory write cycle pushing data onto the stack? If yes, explain how. If not, explain why not.

(5) Question 4. There are two R13s. What is special about R13? Why are there two of them?

(12) Question 5. Select the best term from Chapter 4 that describes each definition.

Part a) This is a condition where once a thread blocks, there are a finite number of threads that will be allowed to proceed before this thread is allowed to proceed.

Part b) This technique could be used to prevent the user from executing I/O on a driver until after the user calls the appropriate initialization.

Part c) This is a scheduling algorithm that assigns priority linearly related to how often a thread needs to run. Threads needing to run more often have a higher priority.

Part d) This OS feature allows the user to run user-defined software at specific places within the OS. These programs are extra for the user's convenience and not required by the OS itself.

Part e) This OS feature allows you to use the OS in safety-critical applications.

Part f) This situation can occur in a priority thread scheduler where a high-priority thread is waiting on a resource owned by a low-priority thread.

(8) Question 6. Consider this filter written in C with the assembly code created by the compiler.

```
LowPassFilter:
  LDR   r2,[pc,#172]  ; @0x0000052C
  LDR   r1,[r2,#0x00]
  ADD   r0,r0,r1
  ASR   r0,r0,#1
  STR   r0,[r2,#0x00]
  BX    lr
```

```
long LowPassFilter(const long x){
static long y=0;
  y = (x+y)/2;
  return y;
}
```

Part a) Is this implementation of the function reentrant? Justify

Part b) How is the input parameters **x** passed?

Part c) What is in LR during the execution of the function?

(10) Question 7. Explain why a deadlock cannot occur when using a **monitor** for thread synchronization. In particular, describe how the monitor operates so deadlock will not occur.

(15) Question 8. Consider these foreground threads that I want to run with your Lab 2 OS, one at a time. These three threads are exactly as shown; no other code inside these threads exists. In each case you may assume the usual Lab 2 tasks (producer, consumer, etc.) are running.

```
void t1(void){
int i;




for(i=0;i<1000;i++){
}




}
```

```
void t2(void){
int i;

OS_AddThread(&t2);



for(i=0;i<1000;i++){
}



OS_Kill();

}
```

```
void t3(void){
int i;

OS_AddThread(&t3);
OS_AddThread(&t3);



for(i=0;i<1000;i++){
}


OS_Kill();
OS_Kill();

}
```

Assume for each case there are the other Lab 2 threads active that are not related to the one thread that I want to add. Assume your round robin preemptive thread scheduler runs every 2 ms.

Part a) If I just add thread **`t1`**, would your OS crash when **`t1`** finishes (and does not call **`OS_Kill`**). If it crashes, explain why. If it doesn't crash, explain why not.

Part b) If I just add thread **`t2`**, explain why or why not your OS will run out of TCBs.

Part c) If I just add thread **`t3`**, explain why or why not your OS will run out of TCBs.

(25) Question 9. You may assume you have the following Lab 2 OS functions available to you (you do not need to write them). Assume the user task runs to completion in less than 1 ms.

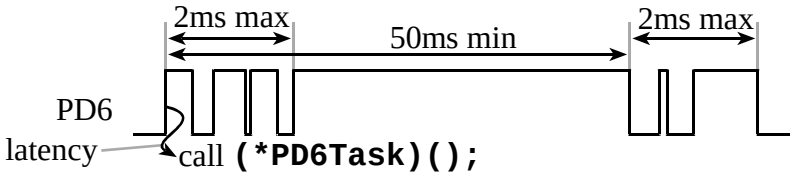
```
void OS_AddThread(void(*task)(void));
void OS_Kill(void);
void OS_Sleep(unsigned long time);  // in ms
void OS_Wait(long *semaPt);
void OS_Signal(long *semaPt);
```

There is a positive logic switch attached to Port D bit 6. The software driver for this is

```
#define PD6  (*((volatile unsigned long *)0x40007100))
void (*PD6Task)(void);  // user task running on PD6 rising edge
//******** OSAddPD6Task  ********
// add a background task to run on rise of PD6, priority 3
// Inputs: pointer to a void/void background user function
void OS_AddPD6Task(void(*task)(void)){
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOD; // activate port D
  PD6Task = task;             // user function
  GPIO_PORTD_DIR_R &= ~0x40;  // make PD6 in
  GPIO_PORTD_DEN_R |= 0x40;   // enable digital I/O on PD6
  GPIO_PORTD_IS_R &= ~0x40;   // PD6 is edge-sensitive
  GPIO_PORTD_IBE_R &= ~0x40;  // PD6 is not both edges
  GPIO_PORTD_IEV_R |= 0x40;   // PD6 rising edge event
  GPIO_PORTD_ICR_R = 0x40;    // clear flag6
  GPIO_PORTD_IM_R |= 0x40;    // enable interrupt on PD6
  GPIO_PORTD_PUR_R |= 0x40;   // PD6 does not have pullup
  NVIC_PRI0_R = (NVIC_PRI0_R&0x00FFFFFF)|(3<<29); // 3, bits 31-29
  NVIC_EN0_R |= NVIC_EN0_INT3;// enable interrupt 3 in NVIC
}

void GPIOPortD_Handler(void){
  (*PD6Task)();               // execute user task
  GPIO_PORTD_ICR_R = 0x40;    // acknowledge flag6
}
```

Unfortunately the switch has bounce. On both a touch and release, there can be from 0 to 2ms of bounce (extra edges). However, sometimes there is no bounce. You may assume the switch is touched for at least 50 ms (meaning the maximum typing rate is 10 strikes per second). This means the minimum time the input pin will be low is 50 ms, and the minimum time the input pin will be high will also be 50 ms. There is no maximum time the signal will be high or low. The user task should be run immediately on each touch (minimize latency). **There can be no backward jumps.**



Part a) Describe changes if any you wish to make to the initialization code (only show changes).

Part b) Rewrite the ISR to handle the bounce while still minimizing latency. In particular, the user task should be run from the ISR without delay after the touch. Of course, the user task should be run only once after a touch, and never run after a release. Show any other software needed.