Jonathan W. Valvano First Name: _____ Last Name:_____
May 16, 2011, 2-5pm
Closed book part

(3) Question 1. Explain how the existence of the barrel-shifter on the Arm architecture affects the design choice between decimal fixed-point and binary fixed-point.

(3) Question 2. Can one ISR interrupt another ISR on the LM3S8962? If so, explain under what conditions it can occur.

(3) Question 3. There are two possible implementations for a problem, both yielding adequate performance. Let **n** be the number of data points used in problem. The speed of the first implementation is related to $\mathbf{n}^2$. The speed of the second implementation is related to **n***log(**n**). Which implementation would you choose and why?

(3) Question 4. Let **x(n)** be the sampled data at $\mathbf{f_s}$ = 1000 Hz (every 1ms), and let **X(z)** be the **Z** transform if this data. If the sampled data is delayed by 10ms, what would be the **Z** transform of the delayed data?

(3) Question 5. Which has lower latency cycle-steal or burst DMA?

(3) Question 6. Why can data flow only in one direction at a time on the CAN bus?

(3) Question 7. Explain how a FAT file system manages free-space. Does FAT have external fragmentation?

(3) Question 8. Does **bounded waiting** mean a there is a maximum time a thread must wait for a resource? If yes, explain how it is guaranteed. If no, explain what bounded waiting really means.

(3) Question 9. What is the largest source of **latency** occurring in background threads?

(3) Question 10. Explain how a single **page table** is used to convert logical to physical address. Give an example.

(20) Question 11. Write C code for a FIFO queue used to pass 32-bit data between foreground threads. None of the FIFO functions will be called from the background. You must assume there may be multiple producers and multiple consumers. You can define and initialize semaphores simply by adding globals:

```
long semaphore=0;
```

You may call the following two semaphore functions without showing their implementation.

```
void OS_Wait(long *semaPt);
void OS_Signal(long *semaPt);
```

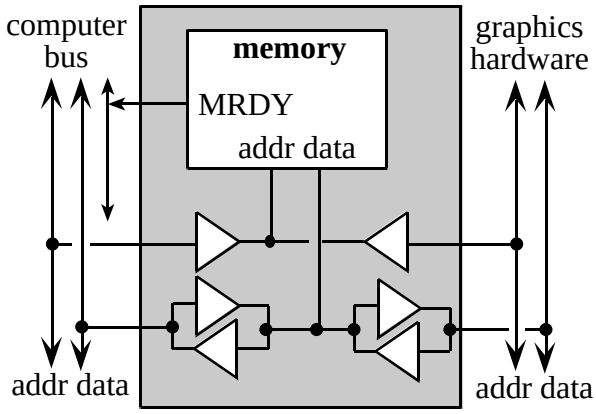
Add the **`static`** qualifier to private components, and no **`static`** for public components. A producer thread should block on full, and a consumer thread should block on empty

Jonathan W. Valvano First Name: _____ Last Name:_____

Open book part

Open book, open notes, calculator (no laptops, phones, devices with screens larger than a TI-89 calculator, devices with wireless communication). Please don't turn in any extra sheets.

(10) Question 12. Consider an architecture with two processors (computer and graphics) having shared memory. In this system if both processors wish to access the same memory module, the graphics processor goes first, and the computer is delayed using the MRDY hardware signal that delays the bus cycle.



To place a variable into shared memory, the qualifier **`shared`** is added. Write C code to implement a binary spinlock semaphore with the following prototypes. If the implementations are different for the two computers, show both versions. Otherwise I will assume both processors execute the same versions. You can assume only one processor calls **`OS_bInit`**, and **`OS_bInit`** is called before **`OS_bWait`** and **`OS_bSignal`** are called. You may also assume only one foreground thread in each processor will call **`OS_bWait`** and **`OS_bSignal`**. Full credit to the simplest correct answer.

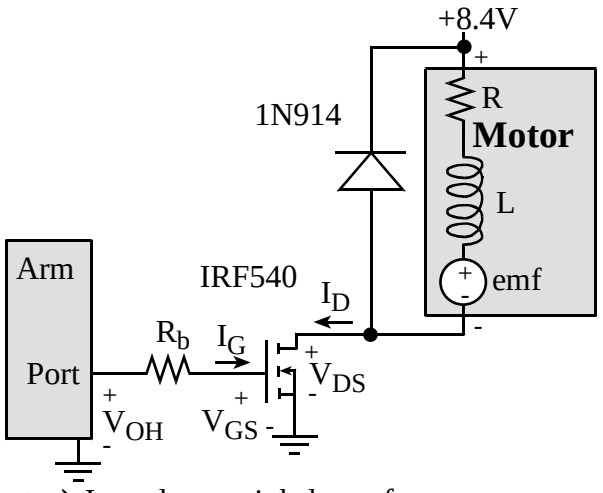
```
void OS_bInit(shared long *semaPt, long value); // Initialize
void OS_bWait(shared long *semaPt);
void OS_bSignal(shared long *semaPt);
```

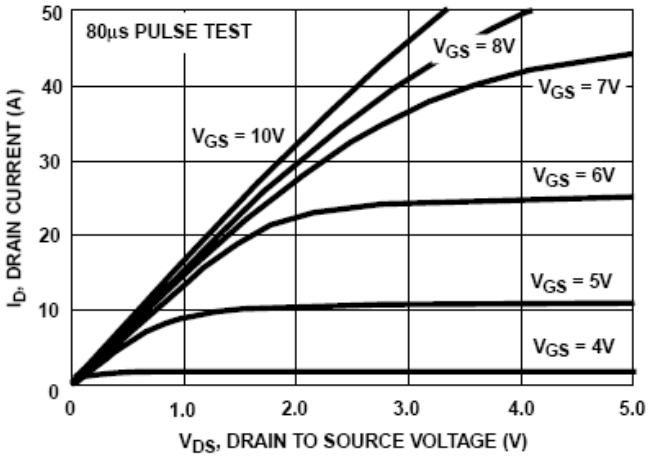
You cannot read MDRY in software. Both processors run ARM Thumb-2 instructions (like lab). Your system should be able to run on this rendezvous application

```
// thread on the Computer
shared long s1,s2;
  OS_bInit(&s1,0); // runs first
  OS_bInit(&s2,0);
  while(1){
    OS_bSignal(&s1);
    OS_bWait(&s2);
    //  stuff1
  }
```

```
// thread on the Graphics
  while(1){
    OS_bSignal(&s2);
    OS_bWait(&s1);
    //  stuff1
  }
```

(10) Question 13. Consider the following motor interface. A series-wound DC motor, like the ones in the robot kit, can be modeled by a series combination of a resistor (the ohmic contribution of the long thin wires used to wrap the electromagnets), an inductor (physically resulting from the cylindrical manner in which the electromagnets are wound), and an emf (a voltage caused by the load torque applied from the shaft back into the electromagnet). The resistance is what you get when you measure the resistance of the motor with an ohmmeter (a stopped motor, disconnected from power). In this case, R = 5 Ω and L = 100 μH. Hint: L/R is 2 μsec. A specification for the IRF540 is shown on the right.

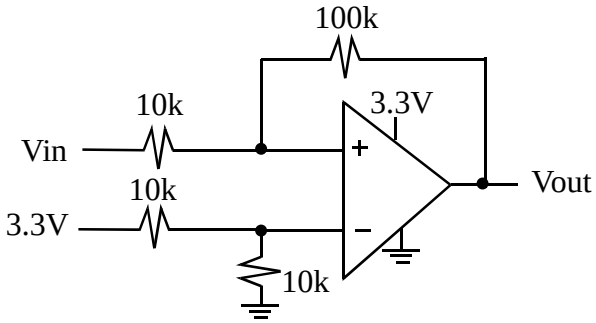




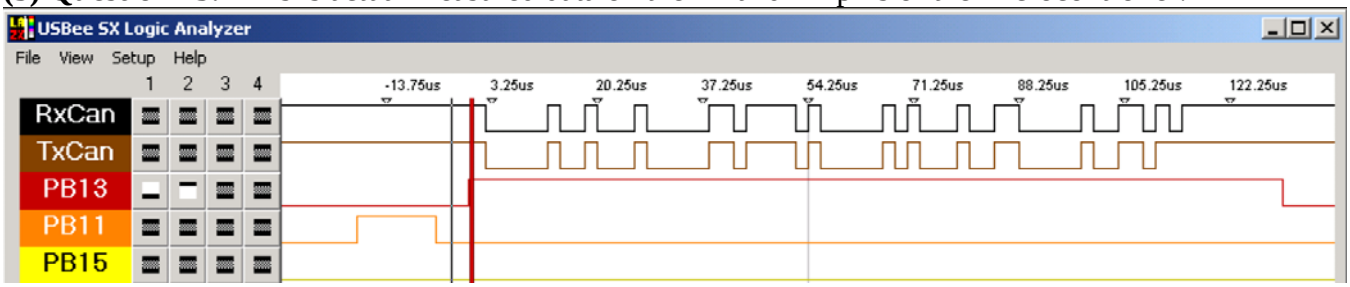
Part a) In order to sink lots of current, you use an IRF540 MOSFET, which is supposed to have a maximum drive current of 28 amps. However, the above circuit with the IRF540 can't drive more than 2A. Why?

Part b) The drain and source connections have not been changed, but the gate portion of the circuit has been redesigned so V_{DS} now equals 0.4 V. For this section assume DC conditions. When torque is applied to the motor, the emf becomes –42V. How much DC current will flow? Why?

(10) Question 14. The following analog circuit is built with a rail to rail op amp. Derive the input/output relationship for input voltages between 0 and 3.3V. Show a plot of this response. Hint: I used this exact circuit on my Lab 7 robot.



(5) Question 15. This is actual measured data on the Tx and Rx pins of the microcontroller.



Part a) Why is there an extra zero in the Rx signal, not present in the Tx signal?

Part b) Assume the CAN baud rate is 500,000 bits/sec and assume the ID is 11 bits. Estimate the bandwidth of this transmission in bits/sec assuming this transmission were repeated over and over without pausing or bus conflicts. I am asking you to determine the number of bytes in each frame by looking at the figure.

(15) Question 16. Consider an architecture with 16 processors and shared memory. Each processor executes Cortex M3 instructions similar your LM3S8963. Each processor has its own register set, its own NVIC, and its own Systick timer. However, the RAM and ROM are shared. The goal is to design a round robin thread scheduler that runs more than 16 threads. In this simple problem, there is no blocking, priority, killing, suspending or sleeping. Each thread runs for 10ms and is preempted. Only one processor at a time should be executing a particular thread. This problem will deal only with thread scheduling and not worry about any initialization code. There is a mechanism for the processor to know its number. In particular, the function **`Me()`** will return 0 to 15 specifying which of the processor executed the call. Every processor is running one thread, this means 16 threads are running in parallel. Each thread is either running once on one processor or it is not running. No processor will be idle. All threads run an equal share of the time.

Part a) Each thread has its own TCB and stack. The threads are in a circular linked list. This list is static and all threads are ready to run. Make changes to this TCB to accommodate the multiprocessor architecture.

```
struct TCB {
  long *stackPointer;  // pointer to top of stack
  unsigned long Id;    // thread number, zero if this TCB is free
  struct TCB *Next;    // TCBs are in a circular linked list


};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
TCBPtr RunPt;     // Pointer to tcb of thread currently running
```

Part b) All processors run off the same crystal. When would you schedule the Systick interrupts in each processor? In particular, how often is an interrupt requested and what is the relative timing of the interrupts in each processor?

Part c) Show the Systick ISR running in each processor. All 16 processors run this code.

Part d) Show the PendSV ISR running in each processor. All 16 processors run this code. Add pseudo-code to your comments to explain your assembly code.