Jonathan W. Valvano May 16, 2011, 2-5pm Closed book part

(3) Question 1. The barrel-shifter allows for right/left shift without time penalty. So binary fixed-point will run faster than decimal fixed-point.

(3) Question 2. Yes, one ISR can interrupt another ISR on the LM3S8962. If an ISR is running at priority level n, then an interrupt of higher priority (lower priority level) can interrupt.

(3) Question 3. The answer depends on both n and the actual relationships. Let the first be $\mathbf{A}*\mathbf{n}^2$. If the second is $\mathbf{B}*\mathbf{n}*\log(\mathbf{n})$. Choose the first if $\mathbf{A}*\mathbf{n}^2$ is less than $\mathbf{B}*\mathbf{n}*\log(\mathbf{n})$, otherwise choose the second.

(3) Question 4. 10ms is 10 samples, so the transform will be $\mathbf{z^{-10} X(z)}$

(3) Question 5. Cycle-steal has lower latency because it does not have to finish the instruction. It can occur in the middle of an instruction.

(3) Question 6. There are only three wires in the bus (two signals and ground). Because the two signal lines can encode one bit, that bit can only flow one direction.

(3) Question 7. In the FAT table, there is linked list (or chain) defining all the free blocks. Because all free blocks can be used to create one big file, the FAT has no external fragmentation.

(3) Question 8. No, **bounded waiting** does not mean a there is a maximum time a thread must wait for a resource. It means once blocked on a resource, there will be a finite number of threads allowed to use the resource ahead of the blocked thread.

(3) Question 9. The largest source of **latency** occurring in background threads occurs running higher priority threads or running with interrupts disabled.

(3) Question 10. The logical address is divided into two fields: page and offset. For example, the logical address $12345 might be divided into page=$12 and offset=$345. The page field is used as an index to the page table. For example, the page table entry at index $12 might contain $89. The value in the page table determines the frame. The physical address is composed of the frame and offset. In this example, the physical address would be $89345.

(20) Question 11. A FIFO queue used to pass 32-bit data between foreground threads.

Put	Get
Wait(&DataRoomLeft)	Wait(&DataAvailable)
bWait(&Mutex)	bWait(&Mutex)
Enter data into Fifo	Remove data from Fifo
bSignal(&Mutex)	bSignal(&Mutex)
Signal(&DataAvailable)	Signal(&DataRoomLeft)

```
#define FIFOSIZE 16                  // can be any size
static long Mutex = 0;               // implements mutual exclusion
static long DataRoomLeft = FIFOSIZE; // size of queue
static long DataAvailable = 0;       // number currently in FIFO
typedef char dataType;
dataType static volatile *PutPt;  // put next
dataType static volatile *GetPt;  // get next
dataType static Fifo[FIFOSIZE];

void Fifo_Init(void){
  OS_Wait(&Mutex);          // this is critical
  PutPt = GetPt = &Fifo[0]; // Empty
  DataRoomLeft = FIFOSIZE;  // size of queue
  DataAvailable = 0;        // number currently in FIFO
  OS_Signal(&Mutex);        // end of critical section
}
```

```
void Fifo_Put(dataType data){
  OS_Wait(&DataRoomLeft); // wait for space
  OS_Wait(&Mutex);        // this is critical
  *(PutPt) = data;        // Put
  PutPt = PutPt+1;
  if(PutPt ==&Fifo[FIFOSIZE]){
    PutPt = &Fifo[0];      // wrap
  }
  OS_Signal(&Mutex);          // end of critical section
  OS_Signal(&DataAvailable);  // one more entry
}
void Fifo_Get(dataType *datapt){
  OS_Wait(&DataAvailable);  // wait for data
  OS_Wait(&Mutex);          // this is critical
  *datapt = *(GetPt++);     // return data
  if(GetPt==&Fifo[FIFOSIZE]){
    GetPt = &Fifo[0];       // wrap
  }
  OS_Signal(&Mutex);         // end of critical section
  OS_Signal(&DataRoomLeft);  // more space
}
```

Or we could implement

```
dataType Fifo_Get(void){ dataType data;
  OS_Wait(&DataAvailable);  // wait for data
  OS_Wait(&Mutex);          // this is critical
  data = *(GetPt++);        // get data
  if(GetPt==&Fifo[FIFOSIZE]){
    GetPt = &Fifo[0];       // wrap
  }
  OS_Signal(&Mutex);         // end of critical section
  OS_Signal(&DataRoomLeft);  // more space
  return data;
}
```

Open book part

(10) Question 12. This problem becomes simple because only one foreground thread in each processor will call `OS_bWait` and `OS_bSignal`. This means there are no critical sections.

```
void OS_bInit(shared long *semaPt, long value){
  *semaPt = value; // Initialize
}
void OS_bWait(shared long *semaPt){
  while(*semaPt == 0){};  // spin if zero
  *semaPt = 0;
}
void OS_bSignal(shared long *semaPt){
  *semaPt = 1;
}
```

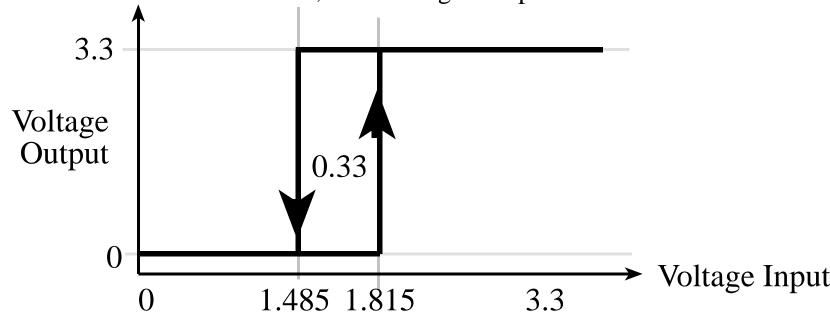
This semaphore could be used to stream data through a double buffer from the main computer to the graphics

(10) Question 13. Consider the following motor interface. MOSFETs are voltage controlled switches and need a large voltage to fully turn on.

Part a) V_{GS} is only 3.3V, so the I_D current is extremely limited. This circuit operates on the figure even lower than the V_{GS} = 4V curve.

Part b) This DC analysis so the inductor is a short. There is a total of 8.4 - -42.4 – 0.8V (50V) across the resistor. So the current is 50V/5Ω = 10 A.

(10) Question 14. This analog circuit is a voltage threshold detector with hysteresis. The voltage at the – terminal of the op amp is fixed at 3.3V/2 = 1.65V. If the voltage at the + terminal is above 1.65V, then V_{out} becomes 3.3V. If the voltage at the + terminal is below 1.65V, then V_{out} becomes 0 V. If V_{out} is already 0 V, then we find V_{in}, such that the + terminal of the op amp is 1.65V. This means V_{in} *100/(110) = 1.65V, which is V_{in} = 1.815 V. Therefore, if the output is 0V it will switch when V_{in} goes above 1.815V. If V_{out} is already 3.3 V, then we again find V_{in}, such that the + terminal of the op amp is 1.65V. This means V_{in}+(3.3-V_{in})*10/(110) = 1.65V, V_{in}-(V_{in}/11) = 1.35V, which is V_{in} = 1.485 V. Therefore, if the output is 3.3V it will switch when V_{in} goes below 1.485V. This creates a 0.33V hysteresis, meaning if the noise is less than 3.3V, this threshold detector will remove it, not causing extra pulses due to the noise.



(5) Question 15. This is actual measured data on the Tx and Rx pins of the microcontroller.
Part a) The extra zero in the Rx signal is the acknowledgement from the receiver, which is obviously not sent by the transmitter.

Part b) From the wave we see the frame takes about 110 µs to transmit. At 500,000 bits/sec, this is about 55 bits. There are 11 bits for the ID and 36 bits for everything that is not data. So these non-data bits take 94 µs. Each added byte takes 16 more µs to transmit. So the choices are

Data	Time µs
0	94
1	110
2	136

From the measurement, it is clearly not 94 or 136 ms, so I think there are exactly 8 bits of data in the frame. The bandwidth of this transmission is 8bits/110µs, which is 72727 bits/sec.

(15) Question 16.
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
  int Status;          // -1 not running, 0 to 15 running
};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
TCBPtr RunPt[16];     // thread currently running by each processor
TCBPtr OldRunPt;      // thread being suspended
TCBPtr NewRunPt;      // thread being launched next
```

Part b) Every 10 ms in each processor, but staggered by 10ms/16, so there will be no critical sections on the thread switching access to the TCBs.

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

```
void SysTick_Handler(void) { int me= Me();
```

```
  OldRunPt = RunPt[me];            // Pointer to running thread
  OldRunPt ->Status = 0;           // not running
  NewRunPt = OldRunPt;             // find next thread to run
  NewRunPt = NewRunPt ->Next;      // skip at least one
  while((NewRunPt->Status >= 0){   // do not run if already running
    NewRunPt = NewRunPt ->Next;    // find one not running
  }
  NewRunPt->Status = Me();         // running
  RunPt[Me] = NewRunPt;
  NVIC_INT_CTRL_R = 0x10000000;    // set pendsv bit to 1, force interrupt
} // end SysTick_Handler
```

Part d) Show the PendSV ISR running in each processor.

```
PendSV_Handler
    CPSID   I                   ; Prevent interruption during context switch
    PUSH    {R4-R11}            ; Save remaining regs r4-11
    LDR     R0, = OldRunPt      ; R0=pointer to RunPt, old thread
    LDR     R1, [R0]            ; OldRunPt->stackPointer = SP;
    STR     SP, [R1]            ; save SP of process being switched out

    LDR     R1, = NewRunPt      ; R1=pointer to NewRunPt, next thread to run
    LDR     R2, [R1]            ; R2=value of NewRunPt

    LDR     SP, [R2]            ; new thread SP; SP = NewRunPt->stackPointer;
    POP     {R4-R11}            ; restore regs r4-11

    CPSIE   I                   ; tasks run with I=0
    BX      LR                  ; Exception return will restore remaining context
```