Jonathan W. Valvano March 1, 2018, 3:30pm-4:45pm.

(5) Question 1. First, look for shared variables. The only shared variable is **Status.flag**. Next, look for nonatomic sequences that access the shared variable, one of which must be a write. All accesses to **Status.flag** are atomic (one simple write). No they are not critical. There is not even a race conditions, because all accesses set the flag to 1.

(5) Question 2. When one output is connected to multiple inputs, we must sum the input current requirements. It will not work because I_{OL} of the TM4C123 is 2mA, and I_{IL} of the system is 3*1mA. It will work however with 8mA mode.

(10) Question 3. Capacitor, I = C dV/dt. Inductor V = L dI/dt.

(10) Question 4. You wish to create a 32-bit binary fixed-point number system that can hold values from -1 to +1. Your system must include the value -1. Your system defines values up to +1, but does not include +1.
(3) Part a) The variable must be signed 32 bit. Therefore, use **int32_t** or **long** or **int**. You could use **int64_t**, but it would not be efficient. You cannot use any of the floating point types, like **float** or **double**.

(4) Part b) Value = integer*resolution. Signed 32-bit integers vary from -2^{31} to $(2^{31}-1)$. If we choose resolution equal to $\mathbf{2^{-31}}$, then the value ranges from $-2^{31}*2^{-31}$ to $(2^{31}-1)*2^{-31}$. The smallest value will be $-2^{31}*2^{-31}$, which is -1. The largest value is $(2^{31}-1)*2^{-31}$, which is very close to +1

(3) Part c) $-½ = (-2^{30}*2^{-31})$. -2^{30} is equal to 0xC0000000.

(10) Question 5. The function **`CreateConnection`** calls the function **`sl_Htonl`** The rationale for this tricky code is the simple link driver code was written in a layered fashion to allow deployment on multiple architectures (some of which are little endian like the Cortex M, while others are big endian). The internet protocol expects the endianness of the IP address to be big, and this function guarantees the 32-bit IP address is big endian.
Part a) The function **`sl_Htonl`** takes an IP address in any format and returns the IP address in big endian, regardless of the endianness of the existing processor.

Part b) Given the fact that the input is an IP address, the individual elements of the array **`p[]`**, contain the four components of the address. For example, if the IP address is 192.168.1.0, P[0] will contain 192, P[1] will contain 168, P[2] will contain 1, and P[3] will contain 0.

Part c) If the architecture were to already be big endian, then p[0] would be zero, and the output value is simply the input value.

(5) Question 6. The **`greet`** field was used to pass data from client TM4C123 to server. This field contained the information that was logged on the web site (e.g., voltage, ADC value, temperature).

(5) Question 7. The key here is to consider what information will be/must be changed by running the ISR. The ISR must modify PC, LR so these registers must be automatically saved. The ISR almost allows modifies the PSR (condition code bits), so it is efficient to save PSR. The ISR usually modify some registers. In fact AAPCS allows the function to modify R0,R1,R2,R3,R12 without saving them. Therefore to be AAPCS compliant the interrupt must save and restore R0,R1,R2,R3,R12. One the other hand, the interrupt context switch does not modify the I bit. The I bit remains 0 during the execution of the ISR so a higher priority interrupt can be processed immediately. Since the I bit is not changed during the context switch, there is no need to save and restore it automatically. The ISR could modify the I bit. However, must ISRs do not modify the I bit. If there were a critical section, I=1 at beginning and I is restored after the critical section. The servicing of an interrupt **DOES NOT** push the I-bit on the stack.

(10) Question 8. Consider an output device that uses a FIFO to pass data
(5) Part a) Since the FIFO is usually empty the output device usually runs faster than the software, meaning it is CPU bound. To increase bandwidth for the system, one should make the software run faster so it creates more data per second. Since increasing software increases bandwidth, it is CPU bound.

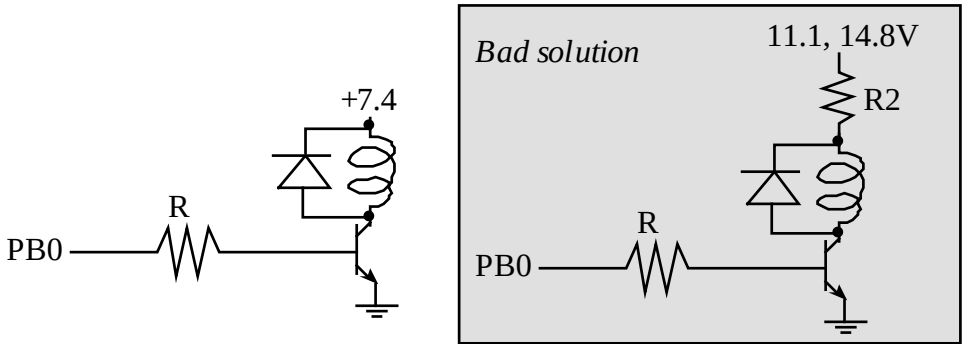
(5) Part b) Occasionally, the FIFO becomes full, so yes you could increase efficiency by increasing the size of the FIFO. On the other hand since it is usually empty, the increase in efficiency by increasing the size of the FIFO would be modest.

(10) Question 9. Interface an 8-Ω speaker to the microcontroller. Key parameters to consider
Current: The current is up to 1A. This means you must have a driver and should not connect it directly to the microcontroller. The I_{CE} of the 2N2222 is only 500mA, so 2N2222 cannot be used. Use a TIP120 N-channel darlington to provide needed 1A current.

Inductive load: This means the speaker is not really an 8-Ω resistor. The conversion from electrical mechanical energy means the equivalent resistance is not fixed at 8 Ω. The resistance can be much more than 8 Ω if mechanical energy is being converted to electrical energy (current less than 1A). The resistance can be much less than 8 Ω if electrical energy is being converted to mechanical energy (current more than 1A). This means you cannot use a series resistor to adjust the applied voltage to the load (like the circuit on the right). You must select a battery voltage as needed by the load. You could add a regulator, like lab 6 and 7 to adjust voltage if needed to be exactly some value. For this problem, connect top of speaker to +7.4V supply to use all the available power

Back emf: V = L dI/dt across the inductive load will cause large voltages if the inductance is large or if the turn off time of the transistor is small. Use a 1N914 snubber diode to prevent back EMF when transistor turned off,

Design steps:
If I_{ce} is 1A, I_b needs to be at least 0.5mA. (current gain is up to 2000)
Select the base resistor to drive transistor into saturation, $R \leq (V_{OH}-V_{be})/ I_b$.
$R \leq (2.4-0.8V)/ 0.5mA = 1.6V/ 0.5mA = 3.2\ k\Omega$. Choose R smaller than 3.2 kΩ.



(25) Question 10. A positive logic switch is interfaced to PA7. There is significant bounce on the switch.
(12) Part a) Write the initialization function that configures the SysTick interrupts.

```
void (*UserTouch)(void);
void (*UserRelease)(void);
uint32_t LastPA7; // PA7 value from previous interrupt
void Switch_Init(void(*T)(void), void(*R)(void)){
  UserTouch = T;
  UserRelease = R;
  LastPA7 = GPIO_PORTA_DATA_R&0x80;
  NVIC_ST_RELOAD_R = 159999; // 10 ms (between 5 to 20ms)
  NVIC_ST_CTRL_R = 0x07; //enable, arm
  NVIC_SYS_PRI3_R = NVIC_SYS_PRI3_R&0x00FFFFFF; // priority 0
  EnableInterrupts();
}
```

(13) Part b) Write the SysTick interrupt service routine

```
void SysTick_Handler(void){
  uint32_t thisPA7;
  thisPA7 = GPIO_PORTA_DATA_R&0x80;
  if(thisPA7 != LastPA7){ // change
    if(thisPA7){          // rising
      (*UserTouch)();     // user function on touch
    }else{
      (*UserRelease)();   // user function on release
    }
  }
  LastPA7 = thisPA7;      // set up for next time
}
```

This is ok

```
void SysTick_Handler(void){
  uint32_t static last=0;
  thisPA7 = GPIO_PORTA_DATA_R&0x80;
  if(thisPA7 && (last==0){ // first time seeing high
    (*UserTouch)();     // user function on touch
    last = 1; // saw a high
  }else if((thisPA7==0x00 && (last==1){ // first time seeing low
    (*UserRelease)();   // user function on release
    last = 0; // saw a low
  }
}
```

This bad software executes the functions multiple times

```
void SysTick_Handler(void){
  if(GPIO_PORTA_DATA_R&0x80){
      (*UserTouch)();     // user function on touch
    }else{
      (*UserRelease)();   // user function on release
    }
  }
}
```

This bad software has a very hard to find race condition because it reads the port multiple times. Each time you read Port A, it might give a different value, because it bounces.

```
void SysTick_Handler(void){
  if(GPIO_PORTA_DATA_R&0x80 != LastPA7){ // change
    if(GPIO_PORTA_DATA_R&0x80){          // rising
      (*UserTouch)();     // user function on touch
    }else{
      (*UserRelease)();   // user function on release
    }
  }
  LastPA7 = GPIO_PORTA_DATA_R&0x80;      // set up for next time
}
```

Any for-loop, do-while, or while-loop would be extremely bad.