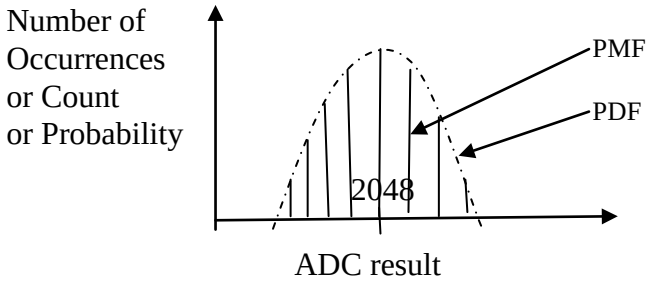
October 9, 2015, 10:00am-10:50am. Solution

(10) Question 1. To analyze noise we fix the input and measure the ADC sample multiple times. If there were no noise, all the ADC samples would be the same (but not necessarily all 2048). The difference between expected (2048) and actual is a measure of accuracy but not noise. Noise is the variability occurring with multiple measurements on a fixed input. Probability mass function shows number of occurrences for each possible ADC value. Because the system has noise, not all inputs will be the same. The shape need not be Gaussian. It is ok to label the x-axis as voltage. It is ok to label the y-axis as probability (unit-less, where the data sum to 1). A PMF has discrete values, whereas a probability density function (PDF) is a continuous curve.



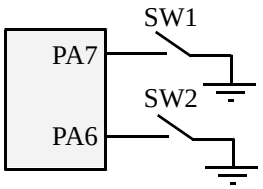
(10) Question 2. There are no critical sections, because the sequences are atomic (ISRs of the same priority will not interrupt each other). If the priorities were different then there would have been a critical section because the read-modify-write operation would have been nonatomic (the higher priority ISR could interrupt the lower priority ISR), and the solutions to this critical section would have been to 1) disable interrupts while toggling in the lower priority ISR; 2) use bit-specific or bit-banded address to remove the sharing; or 3) to move the debugging pin to another port.

(10) Question 3. Neither connection works. The problem does not ask for a solution, but to solve this interface we would use a driver (like BSS138) to boost the voltage levels as needed, making output low smaller and making output high larger. The rules for "does it work?" are

I_{OH} (output) $\geq I_{IH}$ (input) (remember I_{OH} is the maximum possible current, not the actual current)
I_{OL} (output) $\geq I_{IL}$ (input) (remember I_{OL} is the maximum possible current, not the actual current)
V_{OH} (output) $\geq V_{IH}$ (input)
V_{OL} (output) $\leq V_{IL}$ (input)

Part a) No, V_{OH} (A) < V_{IH} (B)
Part b) No, V_{OL} (B) > V_{IL} (A)

(40) Question 4.



Part a) This is just like PF4 and PF0 on the LaunchPad without external pullup. No points deducted for adding 10k pullup resistors to +3.3V. Negative logic means touching switch causes GPIO pin to go low.

Part b) Interrupt every 20ms to debounce (longer than bounce and shorter than the touch/release time. I.e., any time between 10 and 25 ms would be ok.)

```
uint32_t Last; // value of Port A at last interrupt
void Init(void){
  SYSCTL_RCGCGPIO_R |= 0x01;  // activate port A
  Counter = 0;
  NVIC_ST_RELOAD_R = 319999;  // reload value for 20ms
  NVIC_ST_CTRL_R = 7;         // activate and enable interrupts
  GPIO_PORTA_DIR_R &= ~0xC0;  // make PA7-PA6 inputs
```

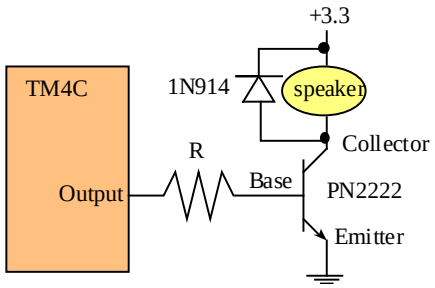
```
  GPIO_PORTA_DEN_R |= 0xC0;   // enable digital I/O on PA7-PA6
  GPIO_PORTA_PUR_R |= 0xC0;   // pullup
  Last = GPIO_PORTA_DATA_R;
  EnableInterrupts();         // I = 0
}
```

Part c) You must add a global variable to store previous values so you can detect a change in input. The most common error students made in this question was to create a solution if the switch were pressed and held down, the counter incremented/decremented multiple times (marked on the exam as "multiple".)

```
void SysTick_Handler(void){
  uint32_t now = GPIO_PORTA_DATA_R;
  if(((now&0x80)==0)&&((Last&0x80)==0x80)){ // SW1 touch
    Counter++;
  }
  if(((now&0x40)==0)&&((Last&0x40)==0x40)){ // SW2 touch
    Counter--;
  }
  Last = now; // edge detection needs a global
}
```

(10) Question 5. Same as aLec11 but the 1N914 diode is not needed when the inductance of the speaker is small.

I_{CE} = (3.3-0.3V)/60Ω = 50mA
I_B =I_{CE} /100 = 0.5mA
R = (3.3-0.6V)/0.5mA = 2.7V/0.5mA = 5.4kΩ,
make R less than this



(5) Question 6. The DNS takes an URL internet web site (like http://www.openweathermap.org) and returns the 32-bit IP address.

(5) Question 7. UDP is used for applications requiring fast communication but does not require reliable communication. TCP is used for applications requiring reliable communication.

(10) Question 8. Write the body of the function that implements fixed-point divide, *N3 = N2/N1*.

Start with the desired function N3 = N2/N1
Specify definitions of fixed point N1 = **I1**/16 N2 = **I2**/16 N3 = **I3**/16
Algebra **I3**/16 = (**I2**/16)/(**I1**/16)
Solve for **I3** **I3** = (**I2***16)/**I1**
Convert to C code minimizing dropout by multiplying by 16 before dividing by **I1**

```
uint32_t Divide(uint32_t I1, uint32_t I2){ uint32_t I3;
  I3 = (I2>>4)/I1;
  return I3;
}
```

or

```
uint32_t Divide(uint32_t I1, uint32_t I2){ uint32_t I3;
  I3 = ((I2<<4) + (I1>>1))/I1; // with rounding
  return I3;
}
```