EE 3171 HW5

Fall 2017

Due: 17 December 2017

No code for this assignment needs to be tested unless you feel it is necessary.

1. (3) Let the precision of a particular ADC be 14 bits. If the range of input voltages goes from 0–4 V, what is the resolution of this ADC?
   Recall the resolution equation: $\frac{4-0}{2^{14}} = 0.000244140625$.

2. (3) Consider that V_{in} to the ADC in Question 1 is 2.18 V. What is the result presented by the ADC (in hex)?
   Start with $\frac{2.18}{0.000244140625} = 8929.28$. No fractions allowed, so just drop the ".28" and convert to hex. Final answer: `0x22E1`.

3. (3) Again using the ADC from Question 1, if the converted result from the ADC is `0x110B`, what was the approximate input voltage?
   `0x110B`=4363. Multiply $4363 \times 0.000244140625 = 1.065185546875$ V.

4. (5) Consider that the same ADC is connected to a analog temperature sensor. When the temperature is 150°F, the sensor outputs 4 V. When the temperature is −150°F, the sensor outputs 0 V. If this sensor is in a refrigerator keeping your food at 39°F, what is the converted result presented by the ADC?
   First, let's calculate the equation relating temperature to voltage. Calculate the slope as $\frac{4-0}{150-(-150)} = 0.01333$. The y-intercept is 2 (unsurprising — that's halfway through the voltage range). So the equation that relates a temperature to a voltage is: $voltage = (0.01333)(temp) + 2$. Plug in the temperature for an input voltage of 2.52 V. Run through the same process as before: $\frac{2.52}{0.000244140625} = 10321.92$. Drop the fraction and convert to hex: `0x2851`.

5. (3) What page in the Tiva Reference Manual contains the most compact reference between the 12 analog input channels and the the GPIO pin that shares that input?
   Table 13-1 spans pages 802–803. There is also a table on page 651, but it is much less compact and contains much more information than just the ADC channels.

6. (8) Write all of the TivaWare functions to appropriately configure a GPIO port for `AIN7` to be used as an analog input.

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN0);
// Alternatively, use the following DRA code
GPIO_PORTD_DEN_R &= ~(0x01);
```

```
GPIO_PORTD_AFSEL_R |= 0x01;
GPIO_PORTD_AMSEL_R |= 0x01;
```

7. (9) Write all of the TivaWare code to configure the Tiva thusly:
   - Use ADC 0
   - Use sequence number 1, step number 0
   - Use a processor trigger
   - Set to the lowest priority
   - Set to do absolute voltage on channel `AIN7`
   - Configure this to be the final step
   - Enable interrupts on completion of the conversion

```
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 3);
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, (ADC_CTL_IE |
                                           ADC_CTL_END |
                                           ADC_CTL_CH7));
ADCSequenceEnable(ADC0_BASE, 1);
ADCIntEnable(ADC0_BASE, 1);
```

8. (5) Consider that we have configured the ADC as described in the previous question. Write the ISR that will read the results of the conversions as described in the previous question and will put each value into successive slots of a 30-element array. When you run out of empty slots, wrap around to the beginning again. (That is, after `ADCResults[29]`, the next entry goes in `ADCResults[0]`.) You will need a `static` or global variable to keep track of where to put the current sample.

```
// Here are global variables.
uint32_t ADCResults[30];
uint32_t* currSlot;
uint8_t i = 0;
currSlot = ADCResults;
bool interruptFired = false;

// inside the main()
while(1)
{
 ADCProcessorTrigger(ADC0_BASE, 1);
 while(!interruptFired) ; // empty loop
 interruptFired = false;
}

// Here is the ISR:
void ADC_ISR(void)
{
 // Clear the flag!
 ADCIntClear(ADC0_BASE, 1);

 // Now get the data.
 // Two ways to do this.  First, the way *I* like:
 ADCSequenceDataGet(ADC0_BASE, 1, currSlot);
 i++;
 if (30 == i)
 {
  i = 0;
  currSlot = ADCResults;
 }
 else
 {
  currSlot++;
 }

 // Alternatively:
 ADCSequenceDataGet(ADC0_BASE, 1, &ADCResults[i]);
 i++;
 if (30 == i)
 {
  i = 0;
 }

 // If you want to be terribly clever:
 ADCSequenceDataGet(ADC0_BASE, 1, &ADCResults[i]);
 i = ((i + 1) == 30) ? 0 : i++;

 // Only  of those is required.

 // This is not the interrupt flag. It's a way of making the
 // main program wait to trigger the next conversion until
 // the current one is finished.
 interruptFired = true;
}
```

9. (2) Describe what it means when a serial link is running at 38400-8-O-1.
Bits arrive at a rate of 38,400 bits per second. There will be 8 characters in between the framing bits. Odd parity will protect the data. There will only be 1 stop bit.

10. (5) Show the waveform that results when a device connects using the link in Question 9 and transmits the ASCII characters '&', 'P' and '0'.
See Figure 1.

11. (5) What are the values that would be placed into the BRDI and BRDF baud rate registers to configure the Tiva UART to a 56600 bps baud rate?
Recall the basic equation: BRDI + BRDF = UARTSysClk / (ClkDiv * Baud Rate). Filling in values: $16000000/(16 * 56600) = 17.667844522968198$. So BRDI = 17. BRDF = integer(0.667844522968198 * 64 + 0.5) = 43.

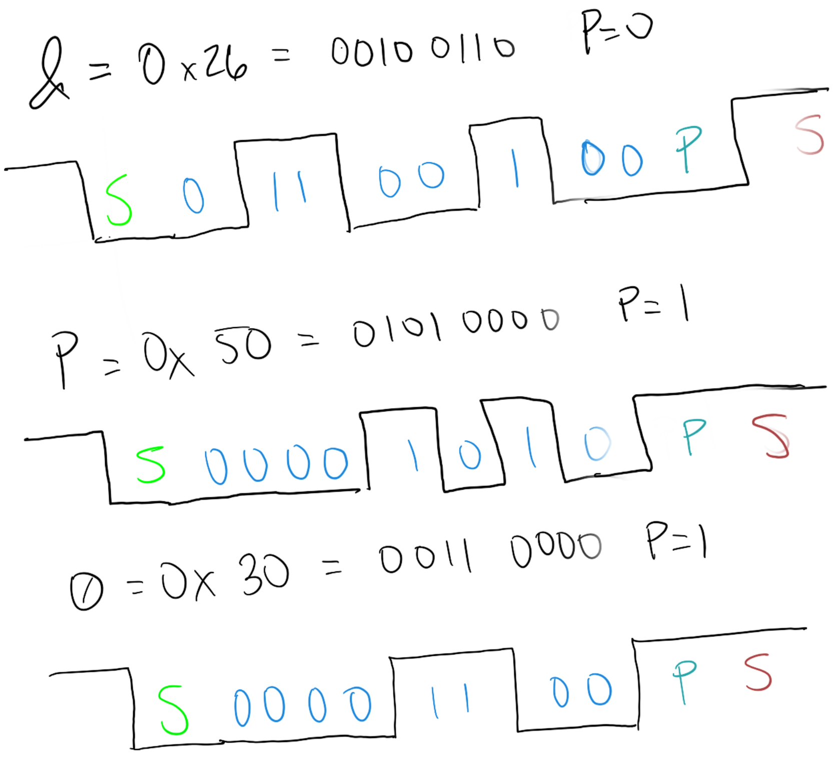

Figure 1: RS232 Waveform for Three Characters

12. (8) Write the TivaWare functions to configure the UART to use a 28,800-8-N-1 serial link, enable the UART and enable receiver interrupts. Don't worry about global interrupts or registering the interrupt handler.

```
UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 28800,
                    UART_CONFIG_WLEN_8 | UART_CONFIG_PAR_NONE |
                    UART_CONFIG_STOP_ONE);
UARTIntEnable(UART0_BASE, UART_IN_RX);
UARTEnable(UART0_BASE);
// Not necessary to register interrupt handler or global interrupt enable.
```