ECE2049 E22: Homework 6 Solutions

Problem 1

Part a

$V_{min} = 0.00475 * (-60°C) + 1.29V = 1.005V$

$V_{max} = 0.00475 * (90°C) + 1.29V = 1.7175V$

Part b

Since the full range of voltages returned by the sensor will be 1.005V to 1.1715V, V_{REF+} should be selected as 2.5V. This will measure the highest resolution possible while ensuring that it is possible to measure the full temperature range.

Part c

The sensor equation is linear with a slope of 0.00475V / ° C.

With a 12-bit ADC and a reference voltage of 2.5V, the resolution can be computed as:

$\frac{2.5V}{2^{12}\ bits} = 0.000610V/bit$

Using this information, we can find the change in temperature per bit by setting up a ratio:

$\frac{0.000610V}{1\ bit} * \frac{1°C}{0.00475V} \simeq 0.12842°C/bit$

Part d

First we need to find the voltage output by the sensor at this temperature, V_{Temp}:

$V_{Temp} = V(24.8\ C) = 0.00475 * (24.8°C) + 1.29V = 1.408V$

Next, we can compute the ADC code for this voltage based on our ADC12 configuration. There are two ways to do this. The simplest way involves using the resolution:

ADC Output = $\lfloor V_{IN}/Resolution \rfloor = \lfloor (1.408V)/(0.000610V/bit) \rfloor = \lfloor 2306.54 \rfloor = 2306$

It is technically more precise to use the full equation for the ADC:

$$\text{ADC Output} = \left\lfloor \frac{V_{Sensor} - V_{REF-}}{V_{REF+} - V_{REF-}} * (2^k - 1) \right\rfloor \tag{1}$$

Since $V_{REF-} = 0V$, we can write:

$$\text{ADC Output} = \left\lfloor \frac{V_{Sensor}}{V_{REF+}} * (2^k - 1) \right\rfloor = \left\lfloor V_{Sensor} * \frac{2^k - 1}{V_{REF+}} \right\rfloor \tag{2}$$

Thus, for our problem: $V_{Temp} = \left\lfloor 1.408V * \frac{2^{12}-1}{2.5V} \right\rfloor = \lfloor 2306.3 \rfloor = 2306$

The difference between the two methods is subtle: the second version uses $(2^k - 1)$ in place of 2^k: this ensures that a code with the maximum value of 4095 represents the maximum voltage value for V_{REF+}.

Part e

This problem is the inverse of part (d). First, we find the V_{Temp} corresponding to this ADC code:

$V_{Temp} = (ADC\ Code) * (Resolution) = (2415\ bits) * (0.000610V/bit) = 1.47V$

Then we can compute the temperature based on the equation for the sensor:

$1.47V = 0.00475 * (Temp°C) + 1.29V$
$Temp = 38.73°C$

Part f

```c
// We can perform this conversion in a similar manner to
// part (e): first we find the voltage that corresponds to the ADC
// code, then we can find the temperature
#define VOLTS_PER_BIT (.000610f) // (2.5/4095)

float convert_temp(unsigned int adc_code) {
  float volts = ((float)adc_code) * VOLTS_PER_BIT;
  float deg_c = (volts - 1.29) / 0.00475;

  return deg_c;
}
```

Problem 2

Part a

The maximum value that can fit in a 32-bit unsigned integer is $2^{32} - 1 = 4294967295$. We also can figure out that there are 31,5360,000 seconds in a year. With the following calculation we can find the number of years held in a maximum value 32-bit unsigned integer:

$4,294,967,295 sec * \frac{1 year}{31{,}540{,}000 sec} = 136.18\ years$

Which approximates to **136 years**.

Part b

`count = 5217504` in seconds

We can compute the number of days, hours, minutes, and seconds as follows:
$3665044\ sec * \frac{1day}{86400sec} = \lfloor \mathbf{42}.41040907 days \rfloor = \mathbf{42\ days}$

$3665044\ sec\ \mathbf{mod}\ \frac{86400sec}{1day} = 36244 seconds\ left$

$\lfloor 36244\ sec * \frac{1hour}{3600sec} \rfloor = \mathbf{10\ hours}$

$36244\ sec\ \mathbf{mod} \frac{3600sec}{1hour} \rfloor = 244\ seconds\ left$

$\lfloor 244\ sec * \frac{1min}{60sec} \rfloor = \mathbf{4\ min}$

$244\ sec\ \mathbf{mod} \frac{60sec}{1hour} \rfloor = \mathbf{4\ seconds}$

The count of 42 days indicates the date is February 12th: 42 full days (31 + 11) have elapsed, meaning the remaining number of seconds indicates the current time is on February 12th.

42 days, 10 hours, 4 min, 4 sec is February 12th 10:04:04 AM

Problem 3

Since our timer interrupts at 25ms intervals, we need to modify `main` such that `do_thing` is called every 500ms, or $\frac{500ms}{25ms/interrupt} = 20\ interrupts$.

We can do this by recording the time count of the last update—when the current time exceeds the last update time by 20 ticks, we should do the thing.

```
volatile unsigned long time_count = 0;

#pragma vector=TIMER2_A0_VECTOR
__interrupt void TIMER_ISR(void) {
   time_count++;
}

void main(void) {
  unsigned long last_update = 0;

  configure_everything();
  start_25ms_timer();
  _enable_interrupt();

  while(1) { // Compare the current time to the time of the last update
    if ((time_count - last_update) >= 20) {
      do_thing();

      // Record the time of the last update for the next iteration
      last_update = time_count;
    }

  }
}
```

A potential alternative would be to use an if statement with a modulo operation to run the function on every 20th tick, as follows:

```
while(1) {
  if((time_count % 20) == 0) {
    do_thing();
  }
}
```

However, this approach may miss a call to the function if `do_thing`, or another part of the program, is running while the counter is a multiple of 20. If the `if` statement is only evaluated at time 19 and time 21, the function will miss its deadline. Using this "every N'th tick" method is primarily suitable `when scheduling tasks in an ISR`, since we know that the statement will be evaluated on each interrupt.