# Solutions to Homework Problems

**ENCE361 Embedded Systems 1**

Course Coordinator: Le Yang (le.yang@canterbury.ac.nz)

Lecturer: Ciaran Moore (ciaran.moore@canterbury.ac.nz)

Department of Electrical and Computer Engineering

# Lecture 2

- Homework: What does the following code do?

```
CMP R0, #9;              // Compare the variable in R0 with 9
ITE GT;
ADDGT R1, R0, #55;       // If R0 > 9, R1 = R0 + 55, converting R0 (A ~ F) to its ASCII code ('A' is 65)
ADDLE R1, R0, #48;       // If R0 ≤ 9, R1 = R0 + 48, converting R0 (0 ~ 9) to its ASCII code ('0' is 48)
```

# Lecture 4 (1)

1. According to Shannon-Nyquist theorem, we must sample at more than twice the highest frequency to reproduce the original waveform. Why not sampling **at** twice the highest frequency?

Suppose we sample a sine wave $\sin(2\pi t)$ at twice the highest frequency which is 1Hz. The sampling frequency is 2Hz and the obtained samples are $\sin(2\pi*0.5n) = \sin(\pi n) = 0$, $n = 0, 1, 2, \ldots$ This means that we may misinterpret a signal if we sample at just twice the highest frequency. Increasing the sampling rate can mitigate this problem.

2. What is the condition called that refers to the distortion caused by sampling below the Nyquist rate?

Aliasing

# Lecture 4 (2)

1. Slide 9 outlines and mathematically defines ideal sampling. What does $1/T_s$ represent and what units does it have?

$T_s$ : sampling interval in seconds.

$1/T_s$ : sampling frequency in Hz or samples/second.

2. If signal averaging (see slide 14) is used for sampling an input signal of with a highest frequency $f_{max}$, what is the relationship between $M$, $f_{max}$ and the sampling rate $f_s$ to ensure that the signal averaging operation does not introduce significant aliasing?

We need to sample with a sampling rate of at least $2*M*f_{max}$

# Lecture 5

1. What type of ADC is used on the Tiva C-Series Launchpad?
   Successive-approximation based ADC

2. What common instrument uses a dual-slope integrating ADC?
   Instrument such as digital multimeter (DMM)

3. How many analog channels are supported by the Tiva C-Series Launchpad ADC?
   8

4. What is the range of quantisation error in volts for an 8-bit quantiser with an input range 0 – 3 V?
   Quantisation step $\Delta = (V_{max} - V_{min})/ 2^N = 3/2^8 = 11.7$ mV
   Range of quantisation error $[- \Delta/2, \Delta/2] = [-5.85, 5.85]$ mV

5. If the maximum signal frequency is 500 Hz and 12-bit quantization is being performed with signal averaging over 15 consecutive samples, what is the minimum sampling rate?
   2 * 500 * 15 = 15 kHz

# Lecture 6 (1)

1. Can the ADCdemo1.c program achieve a comparable performance without using interrupts? If so, how? If not, why not?

The ADC can be triggered using a general-purpose timer with periodic time-outs. This would be as effective as using SysTick interrupts to trigger the ADC.

The difficulty lies with storing the sampling results once they are obtained. Polling of the ADC to check if an ADC conversion is completed would have to be at least as frequent as the sampling rate. This leads to waste of processor clock cycles.

# Lecture 6 (2)

1. In general, how can a compiler decide which of the registers need to have their contents saved as part of the context?

The compiler can look at which of the general purpose registers are being used in the compiled ISR; in principle only the contents of those registers (in addition to the Program Counter, Status Register and Link Register) need to be included in the context.

If a function is called within an ISR, it would be difficult for the compiler to discover the information about which registers are used within the function, so it has to assume that all registers will be used and need to be saved as context at the beginning of the ISR.

# Lecture 7 (1)

1. Can static variables be used for inter-thread communication?
    - If so, how? If not, why?

    Yes,  because static variables would only need to be initialized once and can be accessed throughout the program.

# Lecture 7 (2)

2. In the vending machine example, if initially, there are indeed 21 chocolate bars, the sold out information may not be displayed

- Why? How to fix it?

```
// Main function
 uint32_t main(void)
 {
       while (1)
       {
           if (chocolateCnt <= 0)
           {
              // Display sold out information
           }


       }
 }
```

# Lecture 8 (1)

1. With reference to Slide 9, explain how the RC debouncer works and the role of the Schmitt trigger. What is a disadvantage of this hardware method of debouncing?

When the switch is open, the capacitor is fully charged and the output of the Schmidt trigger is HIGH. When the switch is closed, the capacitor is discharged through R2. The output of the Schmidt trigger would remain HIGH, until the capacitor voltage drops lower than a threshold. Switch bounce, during this period, will not cause multiple transitions between HIGH and LOW.

Disadvantage: extra RC circuitry and latency in switching.

2. Why are pin-change interrupts for debouncing button pushes **not** recommended?

A few rapid switchings can occur. Thus, multiple pin-change interrupts can be generated, which makes the processor be fully occupied for a period of several ms.

3. Why are timer generated interrupts recommended for button polling (Slide 12)?

Timer generated interrupts allow the polling to be approximately regular and independent of other program activities.

# Lecture 8 (2)

4. Write a modified version of **updateButtons()** which implements the dual count ($N_1$, $N_2$) algorithm described on Slide 14.

```
for (i = 0; i < NUM_BUTS; i++) {
    if (but_value[i] != but_state[i]) {
        but_count[i]++;
    if (but_count[i] >= NUM_BUT_PUSH && but_state[i] = RELEASED) ||
       (but_count[i] >= NUM_BUT_RELEASED && but_state[i] = PUSHED) {
        but_state[i] = but_value[i];
        but_count[i] = 0;
    }
  }
  else
    but_count[i] = 0;
}
```

# Lecture 9 (1)

1. Using **circBufT.c** as a model, write C code to implement the function **initDbleBuf ()** whose prototype is given in Slide 16.

```
# include<dbleBuf.h>
uint32_t *initDbleBuf(dbleBuf_t *buffer, uint32_t size)
{
buffer->size = size;          # Double buffer has 2 buffers, each stores size samples
buffer->windex = 0;           # Write index at the beginning of the first buffer
buffer->rindex = size;        # Read index at the beginning of the second buffer
buffer->data = (uint32_t *) calloc(2*size, sizeof(uint32_t));
return buffer->data;
}
```

# Lecture 9 (2)

2. Write C code to implement the function **writeDbleBuf ()** which has the prototype given on Slide 16.

```
void writeDbleBuf(dbleBuf_t *buffer, int32_t entry)
{
    buffer->data[buffer->windex] = entry;
    buffer->windex ++;                          # Increase write index
    if (buffer->windex >= 2*buffer->size)       # Put the write index at the beginning of the double buffer
      buffer->windex = 0;
}
```

# Lecture 9 (3)

3. Could you use the functions prototyped in **circBufT.h** and implemented in **circBufT.c** to implement a FIFO buffer (Slide 17)? If so, how? If not, why not?

In principle, yes. In this case, after the FIFO buffer (implemented using a circular buffer) is filled, the windex should point to the oldest sample. The rindex should also point to that sample. If a read operation is performed, the oldest sample is processed and rindex should be relocated to the next oldest sample. If no new data arrives, windex should continue pointing to the oldest sample we have just read out. If a new data sample arrives, it moves with the rindex. But windex cannot overtake the rindex. Otherwise, data loss would occur.

4. In the call to **displayMeanVal ()** on Slide 13, the expression in red finds a rounded value for the mean using only integer arithmetic. Comment on the order of operations in the expression, does the order matter?

The line in red realizes round(sum/BUF_size).

# Lecture 10 (1)

1. A particular UART-to-UART transmission is set for 1 start bit, 8-bit data, odd parity and 2 stop bits. Which of the following 4 strings are received without detectable error and what is the data word in each of those cases?

i. 010101011111, start 0 + data 10101011 + parity 1 + stop 11, detectable error from parity

ii. 001110001111, start 0 + data 01110001 + parity 1 + stop 11, no detectable error

iii. 001001101111, start 0 + data 01001101 + parity 1 + stop 11, no detectable error

iv. 011110001001, start 0 + data 11110001 + parity 0 + stop 01, detectable error from stop bits

# Lecture 10 (2)

2. UART units on the Tiva Microcontroller have Tx and Rx FIFO buffers. What is the difference between a FIFO buffer and a circular buffer?

The FIFO buffer has a first-in first-out rule. It could be realized as a circular buffer. In this case, after the FIFO buffer is filled, the windex should point to the oldest sample. The rindex should also point to that particular sample. If a read operation is performed, the oldest sample is processed and rindex should be relocated to the next oldest sample. If no new data arrives, windex should continue pointing to the oldest sample we have just read out. If a new data sample arrives, it moves with the rindex. But windex cannot overtake the rindex. Otherwise, data loss would occur.

3. What are the advantages and disadvantages of using a *blocking* Tx function?

Advantage: simple and no return values
Disadvantage: CPU clocks may be wasted on waiting for the busy UART to finish the current transmission request.

# Lecture 11 (1)

1. A primary source of noise in resistors is due to thermal (Johnson) noise
   - What is the noise RMS voltage of a 1 MΩ resistor at 300° K over a 100 kHz bandwidth?

   40.7 μV

   - What is the noise RMS voltage of a 1 MΩ resistor in parallel with a 1 kΩ resistor at 300° K over a 100 kHz bandwidth?

   1.29 μV

# Lecture 11 (2)

2. In the example with SNR = 20 dB shown on slide 12, the average power of the signal is 0.9 W (ignoring the DC offset). What is the average power of the noise?

   20 dB = $10\log_{10}$(signal power/noise power)

   Signal power = 0.9 W = 100 * noise power

   Noise power = 0.9W/100 = 0.009 W

3. In slide 13, explain why the noise amplitude on the right (obtained via signal averaging with $M$=32) is significantly less than that on the left (obtained via signal averaging with $M$=16).

   The noise is periodic with a period of 100/2.5Hz = 40 samples. When $M$ = 16, the noise samples within the averaging window may have the same sign, which leads to greatly reduced noise attenuation capability. When $M$ = 32, the noise samples within the averaging window would cancel one another, which indicates strong noise suppression.

# Lecture 12 (1)

1. Why might it be preferable to use PWM to pass information about a continuous variable, such as the position of the steering wheel on slide 3, rather than using an analogue voltage directly?

The information carried in the analogue voltage could be corrupted by additive noise during signal transmission. But it would be less distorted if it is carried in the duty cycle of a PWM signal. This is because the duty cycle is found at the receiver through averaging/integrating the received PWM signal, which could reduce the additive noise (cf. digital signal conditioning).

2. Are the average voltages shown on the RHS of Slide 4 the same as the RMS voltages (see next slide) for the respective waveforms? Calculate the RMS values for the waveforms shown to corroborate your answer.

No. The RMS voltage of a PWM signal with duty circle $d$ (0-1), high voltage $A$ and low voltage 0 is equal to sqrt($d$)*$A$, where sqrt() denotes the square root operator. On the other hand, the average voltage of the same PWM signal is $d$*$A$.

# Lecture 12 (2)

3. Show the calculation that leads to the conclusion that the minimum PWM frequency that can be achieved on the Tiva C-Series Launchpad with processor clock frequency 20 MHz and the default (no prescale) for the PWM clock is about 160 Hz.

According to slide 13, the load register of the PWM generator has 16 bits. The minimum PWM frequency is thus achieved by using the count-up/down mode. In this case, the PWM frequency is $20MHz/(2*2^{16}) \approx 152.6Hz$.

4. What is the minimum PWM frequency that can be achieved on the Tiva C-Series Launchpad with processor clock frequency 20 MHz and with a PWM prescale of 32?

In this case, the minimum frequency is $20MHz/(32*2*2^{16}) \approx 4.77Hz$.

# Lecture 14 (1)

1. A 16-bit count-down timer is used to measure intervals up to 200 ms. If the microprocessor clock rate is 20 MHz and the timer clock prescaler is an 8-bit unsigned integer, what is the best time resolution possible for the interval measurements and what is the value of the prescaler to achieve that?

Suppose the prescaler takes a value of K.

The time resolution would be 1/(20MHz/K) = K/20MHz second.

To maximize the time resolution, we have

$$2^{16}*K/20MHz = 200\ ms$$

This gives that K = 61.03. We then have that K = 61. This gives a time resolution of 3.05us.

2. With reference to Slide 6, describe what happens when an edge is detected on pin “PTn”, indicating the end of an interval.

In this case, the value in the counter TCNT of the free-running timer will be stored in the TCn register. If the interrupt is enabled, an interrupt request is issued and the ISR can read the value from TCn.

# Lecture 14 (2)

3. The code on Slide 12 will return an incorrect interval sometimes. Explain what may cause the problem. Assuming a 16-bit counter, modify the code to overcome the problem.

The timer works in the count-down mode. Therefore, if the timer is reloaded before the return function is called, the returned interval would be negative, which is incorrect.

```
uint32_t  second_time = TimerValueGet(TIMER0_BASE, TIMER_A);
if (second_time <= first_time) return (first_time - second_time);
else
return (MAX_16BIT_UVAL + first_time – second_time); // MAX_16BIT_UVAL = 2^16 - 1
```

# Lecture 15 (1)

1. With reference to Slide 10, what does it mean to “mask” an interrupt?

Masking refers to applying a pattern of bits to control the enabling and disabling of individual interrupts.

2. With reference to Slide 17, explain carefully why the version which uses API calls within the ISRs is slower to service an interrupt.

Every API function call involves the use of the stack to pass arguments, as well as branching and returning to the instructions implementing the function call. These operations take processor cycles that are not needed when directly setting the registers.

# Lecture 15 (2)

3. What happens if the *prototype* for a function, for which the implementation appears in a source file 'fancy_functions.c', does not appear in 'fancy_functions.h'?

If the compiler encounters a call for a function without any prototype, it **assumes** that the return type is **`int`** and it guesses the actual type for each argument from the values used in the call. This can cause errors at execution.

4. In the case mentioned in Q3, where else, other than in a header file, might a prototype for a function correctly appear and why?

For functions that are designed to be used only within a particular module (.c file), their prototypes should be placed at the top of the file (immediately after the # include<> statements).

# Lecture 16

1. Consider Slide 6. If the function **lSecondsSinceMidnight()** starts being executed at the time when the global variables have values **iSeconds** = 59, **iMinutes** = 59 and **iHours** = 3. After an interrupt occurs, what is the biggest difference possible between the value the function returns and the actual number of seconds since midnight?

   The biggest difference between the function return and the actual number of seconds would be 3600.

2. Why is there a shared data problem associated with the conventional *read-modify-write* operation used on many microprocessors to write to (i.e., to change) individual bits on I/O ports? Describe a scenario in a program using interrupts where the shared data problem exists.

Setting/clearing a single bit in a register associated with a peripheral requires three operations on many MCUs, namely read-modify-write. If an interrupt occurs and its ISR happens to modify the same register, there could be a shared data problem leading to in the register being left in an undecided state.

# Lecture 20 (1)

1. What is the major difference between the time-triggered and interrupt-triggered schedulers?

Both schedulers use interrupts to control which task is to be executed. The difference lies in how this decision is made. With the time-triggered system, tasks are scheduled according to how often they need to run. With the interrupt-triggered system, tasks are scheduled to run at the next possible opportunity when external hardware signals that certain events have happened via an interrupt.

2. What determines the resolution required for the real-time clock in the implementation of the ABS braking system?

The major factor affecting the time resolution of the real-time clock (RTC) is the rotation rate of the wheels. If e.g., 5% accuracy is required in measuring the individual pulse-to-pulse interval for each wheel at the maximum speed (200 kph), 20 clock ticks are required for each pulse. This corresponds to 20*1320 = 26,400 ticks per second. We may therefore set the working frequency of 30 kHz for the RTC.

# Lecture 20 (2)

3. In the time-triggered version of the ABS system, what is the maximum amount of time that could be spent on executing a chunk of the monitorDriver and displayABS tasks without causing the deadlines for wheel speed processing to be missed?

According to slide 6, if we ignore the time consumed on RTC (ISR_SysTick()) and wheel pulses (ISR_Wheels ()), the amount of time left for monitorDriver and displayABS is

3.788 ms – 0.12 ms (from 4 wheel speed estimator) – 0.25 ms (from relative speed estimator) = 3.418 ms

If we take into account the time consumed on RTC (ISR_SysTick()) and wheel pulses (ISR_Wheels()), we only have 2.07 ms for monitorDriver and displayABS.