Module 4: Lecture Questions
Partial Solutions

Module 4 – Lecture 13(Friday – 4/16)

1) What is the difference between a Mailbox and a Queue? How is their configuration different in the TI-RTOS GUI interface?
2) What are the conditions under which the Mailbox_post() and Mailbox_pend() commands are blocked? Can these commands be called from Hwi or Swi threads? Explain.

Module 4 – Lecture 14(Tuesday – 4/20)

3) TI-RTOS does not support pipes. What is the difference between a queue and a pipe? How might we send a variable size message using a Mailbox object instead. What is the disadvantage of using the Mailbox object versus a real pipe?
4) Write your own mailbox_put and mailbox_get commands using binary semaphore Semaphore_post() and Semaphore_pend() commands to determine if the mailbox is full during the put command or empty during the get command and to resolve any shared data issues. Use the FIFO Queue as an example. The message being past is two integers one for sample time and one for sample value. If the mailbox is full, the mailbox_put should block until it is empty. If the mailbox is empty, the mailbox_get pend should bock until it is full.

```
int32_t gTimeSample;
int32_t gDataSample;
void mailbox_put(int32_t time, int32_t data) {

    Semaphore_pend(mailboxEmpty, BIOS_WAIT_FOREVER);
    Semaphore_pend(dataMutex, BIOS_WAIT_FOREVER);
    gTimeSample = time;
    gDataSample = data;
    Semaphore_post(dataMutex);
    Semaphore_post(mailboxFull);

}

void mailbox_get(int32_t *time, int32_t *data) {
    Semaphore_pend(mailboxFull, BIOS_WAIT_FOREVER);
    Semaphore_pend(dataMutex, BIOS_WAIT_FOREVER);
    *time = gTimeSample;
    *data = gDataSample;
    Semaphore_post(dataMutex);
    Semaphore_post(mailboxEmpty);
}
```

5) You are trying to insert an accurate 5 ms delay into a Task. What is the main advantage and the main disadvantage of calling the TI-RTOS Task_sleep() function versus polling a hardware timer in a while loop for this purpose? If the clock module is set for a 250 usec clock tick period, what should be entered in the argument for the Task_sleep() command.

6) If a finite value for the timeout argument in a Semaphore_pend() call is used, name two types of problems which may arise? Name two situations were finite timeout values maybe useful if handled properly.

7) There is an event object whose handle name is myEvent.

   a) Write a task called writeEvent which when called by the clock module every 10 msec uses an event_post function to signal that event 0 and event 5 and event 11 have occurred.

```
void writeEvent(UArg arg0) {
   Event_post(myEvent, Event_Id_00 | Event_Id_05 | Event_Id_11);
}
```

   b) Write a task called readEvent that uses the event_pend function to wait for event 0 and event 5 to be signaled or wait for event 5 or event 11 to be signaled before executing critical events.

```
void readEvent (void){
uint32_t posted;
   while(1) {
   posted = Event_pend(myEvent,
      Event_Id_00 | Event_Id_05,  /* andMask */
      Event_Id_05 | Event_Id_11,  /* orMask */
      BIOS_WAIT_FOREVER);
   //Do critical stuff.
   }
```

   c) Which condition will the readEvent be unblocked by? It is unblocked by the andMask and orMask condition.

   d) What will be the return value of readEvent be? It returns 0x821.

   e) What events will still be signaled in myEvent after readEvent is called? There will be no other bits still signaled after the call it consumes them all and clears them.

8) The following Task uses a TI-RTOS Mailbox object to handle multiple event types.

```
void Task1(UArg arg0, UArg arg1) {
    int msg;
    while (1) {
        Mailbox_pend(mailbox1, &msg, BIOS_WAIT_FOREVER);
        switch(msg) {
            case 0:
                // handle event 0
                break;
            case 1:
                // handle event 1
                break;
        }
    }
}
```

a) Write a Swi service routine called `Swi1` that signals `Task1` that "event 0" (referring to the comments) has occurred.

```
void Swi1(UArg arg0, UArg arg1) {
    int msg;
    msg = 0;  // specify event 0
    Mailbox_post(mailbox1, &msg, BIOS_NO_WAIT); // Swi can't block
}
```

b) Rewrite `Task1`, replacing the Mailbox with a TI-RTOS Event object to achieve approximately the same functionality (ability to handle two event types). Refer to the SYS/BIOS (TI-RTOS Kernel) User's Guide (link available on Canvas under Pages/Datasheets) and CCS online documentation for detailed information about Event objects.

Handling multiple events through Event objects is different in that more than one event may need to be handled per loop iteration. The return value of Event_pend() needs to be examined to determine which events to handle. We cannot use a single switch() statement here.

Create an Event object called `event1`.

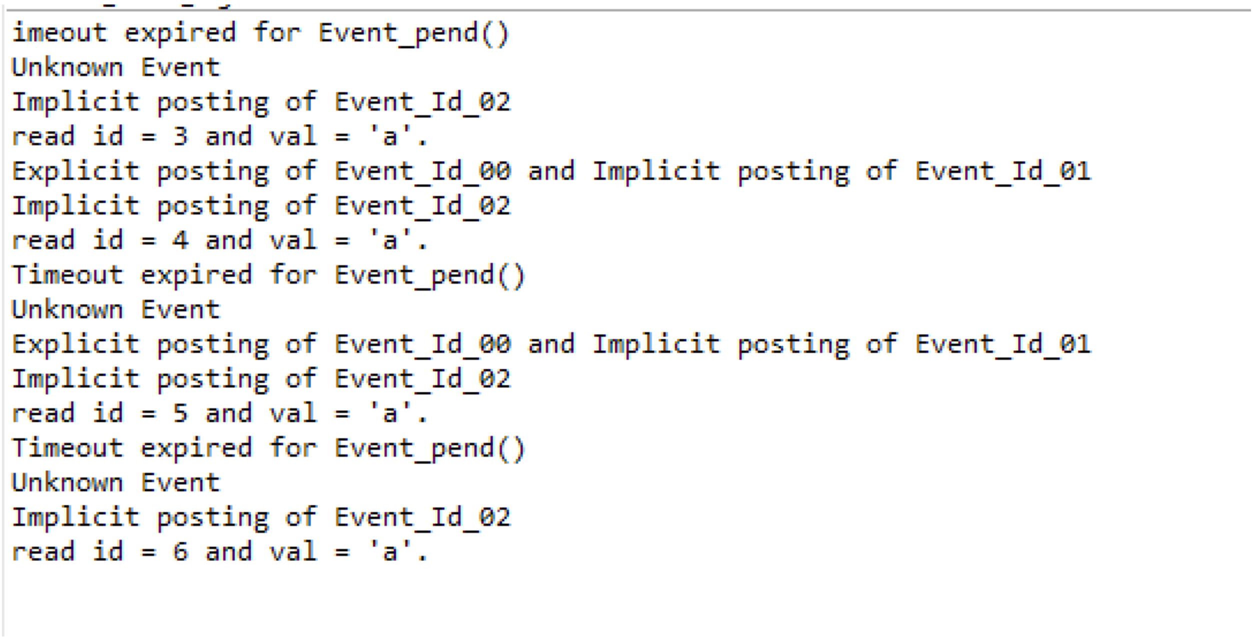
```
void Task1(UArg arg0, UArg arg1) {
    UInt posted;
    while (1) {
        posted = Event_pend(event1,
                     Event_Id_NONE,               // AND mask; 0 also OK
                     Event_Id_00 | Event_Id_01, // OR mask
                     BIOS_WAIT_FOREVER);

        if (posted & Event_Id_00) {
            // handle event 0
        }

        if (posted & Event_Id_01) {
            // handle event 1
        }
    }
}
```

c) Write a Clock callback function called `Clock1` that signals the new `Task1` from part (b) that "event 1" has occurred.

```
void Clock1(UArg arg) {
    Event_post(event1, Event_Id_01);
}
```

Module 4 – Lecture 15 (Friday – 4/23)

9) What are three ways to signal an event?
10) The mailbox object can be configured to signal two events. What action signals a reader event? What action signals a writer event?
11) The event3849_event example program had a task called readertask().
    a) Why does the readertask() do a Semaphore_pend() and a Mailbox_pend() with the BIOS_NO_WAIT argument instead of the traditional BIOS_WAIT_FOREVER argument?
    b) The original program had events happening at certain times. If the event timing was changed as described below. What would the readertask() printout messages report for the first 16 msec?
        - clk0Fxn task was periodically triggered every 2 msec
        - clk1Fnx task was periodically triggered every 6 msec
        - Mailbox_post() was issued every 4 msec with data always equal to 'a' with an incrementing id value.
        - The TIMEOUT value in the event_pend is 3 msec.



```
imeout expired for Event_pend()
Unknown Event
Implicit posting of Event_Id_02
read id = 3 and val = 'a'.
Explicit posting of Event_Id_00 and Implicit posting of Event_Id_01
Implicit posting of Event_Id_02
read id = 4 and val = 'a'.
Timeout expired for Event_pend()
Unknown Event
Explicit posting of Event_Id_00 and Implicit posting of Event_Id_01
Implicit posting of Event_Id_02
read id = 5 and val = 'a'.
Timeout expired for Event_pend()
Unknown Event
Implicit posting of Event_Id_02
read id = 6 and val = 'a'.
```

12) Your program is experiencing data corruption and you think it is due to a missed relative deadline in one of your Hwi / ISR's. Name three common debugging measurements you might make to figure out the problem? How would you go about making these measurements with the TI-RTOS tools?
13) What events can the TI-RTOS log?
14) How do you check the stack utilization of each task in your design?
15) We have looked at two ways to measure CPU Load. What are the advantages and disadvantages of each method?

Module 4 – Lecture 16 (Monday – 4/26)

16) What are the pros and cons of using built-in RTOS real-time debugging tools, like the TI-RTOS System Analyzer, versus implementing custom debugging code? List two pros and two cons.

Pros:

1. **Extensive functionality, powerful visualization tools.**
2. **Already debugged.**

Cons:

1. **Significant overhead, showing up as increased latency and CPU load.**
2. **Does not always measure what you need, e.g. response time.**
3. **OS-dependent.**

17) The RTOS is able to measure the CPU utilization of a real-time system. Even if this measurement is well below the RMS (rate-monotonic scheduling) CPU utilization bound, this still does not completely guarantee schedulability of your system. Why? What measurements would you take to guarantee schedulability?

18) One of your Tasks is triggered by a periodic hardware timer (through an ISR). How would you estimate the response time of this Task? Outline your approach.

- **Determine the time elapsed from the interrupt to the end of the event handler using the current timer count. This is the instantaneous response time.**
- **Keep track of the max response time in a global. Update it if the instantaneous response time is higher. Exercise code paths that may affect this task's response time.**

19) You are trying to measure the response time of an ISR. This ISR is triggered by a rising edge of an external signal. Outline your approach.

20) How would you measure the worst-case execution time of a C function using a logic analyzer or oscilloscope? Assume the execution time of this function varies from run to run.

- **Set a GPIO output to 1 before calling this function.**
- **Reset the GPIO output to 0 after this function returns.**
- **Run this code many times in a loop with a good delay between function calls so multiple events are not shown on the screen.**
- **Trigger the logic analyzer or oscilloscope on the rising edge of the GPIO signal.**
- **Turn on infinite persist mode and find the time from the trigger to the farthest falling edge.**

More questions on next page.

21) Using the System Tick timer, instrument the code below to measure the maximum execution time of the code in the while loop. Use ece3849_profiling for reference.

```
int main(void)
{
  uint32_t i = 0;
  uint32_t sum = 0;

  // Enable the Floating Point Unit, and permit ISRs to use it
  FPUEnable();
  FPULazyStackingEnable();

  // Initialize the system clock to 120 MHz
  gSystemClock = SysCtlClockFreqSet(SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
       SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480, 120000000);

  while (1) {
    for (i = 0; i < NUM_SAMP; i++) {
      sum += data_samples[i];
    }
  }
}
```

Solution on next page

```
int main(void)
{
   uint32_t i = 0;
   uint32_t sum = 0;

   //declare your profiling variables
   uint32_t bias;
   uint32_t start, finish;
   uint32_t execution_time = 0;
   uint32_t max_execution_time = 0;

   //disable interrupts
   IntMasterDisable();
   // Enable the Floating Point Unit, and permit ISRs to use it
   FPUEnable();
   FPULazyStackingEnable();

   // Initialize the system clock to 120 MHz
   gSystemClock = SysCtlClockFreqSet(SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480, 120000000);

   // initialize SysTick timer for the purpose of profiling
   SysTickPeriodSet(1 << 24); // full 24-bit counter
   SysTickEnable();

   //measure your measurement bias
   start = SysTickValueGet(); // read SysTick timer value
   finish = SysTickValueGet();
   bias = (start - finish) & 0xffffff;

   sum = 0;
   while (1) {
      start = SysTickValueGet();
      for (i = 0; i < NUM_SAMP; i++) {
         sum += data_samples[i];
      }
      finish = SysTickValueGet();
      execution_time  = ((start - finish) & 0xffffff) - bias;
      if (execution_time > max_execution_time) {
         max_execution_time = execution_time;
      }
   }
}
```

22) TIMER0 is used to trigger an interrupt handled by the ISR_Event0() function every 200 usec. The ISR_Event0 then signals the task0_func task. The system has two modes, in one mode it takes the average of the temperature values and in the other mode it takes the difference in measurements. The user presses a button to switch between modes. Assume the ISR_Event0 is the highest priority interrupt and that the code meets the relative deadline so there will be no rollover conditions.

a) Instrument the code below to measure the maximum latency and response time of the task0_func task . Be sure to account for any measurement bias. See the ece3849_rtos_latency example as a reference. See next page for solutions

b) What tests will you run to guarantee all the alternative code paths are exercised? Why is it important to exercise all paths?

c) You run your instrumented code and the execution time measurement returns a maximum response time 2000 and your latency measurement is 200. What units is this measurement in? If you have a 200 MHz System clock, what is the response time in usec? What is your relative deadline?

   Response time in usec = 2000 * 1/200MHz * 1e6 usec/sec = 10 usec.
   The relative deadline is the interrupt period of 200 usec as the ISR must complete before the next interrupt happens.

d) Your manager says seeing is believing and wants you to use lab equipment to show him the real-time latency and response time on an oscilloscope instead of using instrumented software. What is an advantage of using lab equipment? Explain how you setup the measurement in lab and what changes to the code are required to make the measurement?

e) While you are demonstrating your measurement in lab, your manager realizes that he wants to know what the maximum response time. What are two ways you can configure the oscilloscope to show maximum response time.

```
void ISR_Event0(UArg arg0)
{
  TIMER0_ICR_R = TIMER_ICR_TATOCINT; // clear interrupt flag
  Semaphore_post(semaphore0);
}

void task0_func(UArg arg0, UArg arg1)
{

  IntMasterEnable();
  TimerEnable(TIMER0_BASE, TIMER_A);

  while(1) {
    Semaphore_pend(semaphore0, BIOS_WAIT_FOREVER);

    if (mode_average) {
      calc_average();
    }
    else {
      calc_difference();
```

```
        }
    }
}
```

Solutions for 22a

```
void task0_func(UArg arg0, UArg arg1)
{
    unsigned long t = 0;
    unsigned long t_start = 0;
    unsigned long t_end = 0;

    IntMasterEnable();

    TimerEnable(TIMER0_BASE, TIMER_A);
   // measure your bias
    t_start = TIMER0_TAR_R;
    t = TIMER0_TAR_R;
    t_bias = t_start - t;

    while(1) {
        Semaphore_pend(semaphore0, BIOS_WAIT_FOREVER);

        t_start = TIMER0_PERIOD - TIMER0_TAR_R - t_bias;
        if (t_start > event0_latency) event0_latency = t_start;

        if (mode_average) {
            calc_average();
        }
        else {
            calc_difference();
        }
        t_end = TIMER0_PERIOD - TIMER0_TAR_R - t_bias;
        if (t_end > event0_response_time) event0_response_time = t_end;
    }
}
```

23) Both SysTick Timer and BIOS Timestamp Modules run off of the same CPU System clock.
   a) What is an advantage of using the BIOS Timestamp Module?
   b) What is a disadvantage of using the Timestamp Module?
   c) You can also configure a 16-bit hardware timer to take the same measurement using the same CPU System clock. What is an advantage of using the hardware timer? What is the disadvantage?
   d) If my CPU clock is 120 MHz and my maximum execution time is 200 msec which method should I use? Explain?

Module 4 – Lecture 17 (Tuesday– 4/27)

24) You are using a timer in capture mode to measure the frequency of an input square wave. You do not have control over the input signal's frequency. Why might this be a problem for your system?

25) You have a perfect sign wave whose frequency can vary from 10k to 100kHz with a DC offset of 1.5V and an amplitude that can vary from 50 mV to 1V.
    a) Explain how you would measure the frequency of this signal.
    b) For large signals, everything works perfectly and the period measured matches the expected value. For small periods, no ISRs are triggered. What is most likely the problem.

26) When using a timer in capture mode to measure period, what is the minimum measurable period determined by? What is the maximum measurable period determined by?

27) I have two signals, serial_select and serial_data. They operate at 500 kHz. I need to measure the delay between the falling edge of a serial_select and the either the rising or falling edge of serial_data. How would you setup this measurement? Hint: it is similar to measuring pulse width. What is the smallest delay you can measure?

28) In class we saw that by using two timers we could measure pulse widths as small as two system clock periods. Using this method, what is the maximum allowable frequency of that signal?

29) You are generating a pulse width modulated output signal to control the brightness of an LED bulb using a timer in PWM mode with a system clock frequency of 100 MHz . The LED response time is slow and requires that the frequency of the PWM signal be 100 Hz. For the maximum brightness the LED must be driven high for 90% of the time, for medium brightness 50% of the time and for the dim setting 10% of the time.
    a) What load value would you program?
    b) If TnPWML = 0 what match values would you program for each of the brightness settings?

30) A 24-bit timer is being used in edge time capture mode to measure the period of an input waveform. The following ISR places the measured period into a global variable. When the period of the input waveform decreases below 10 µs, you notice occasional erroneous period measurements that are too large.
    a) What is the most likely cause of these errors? How much do you expect an erroneous measurement to deviate from the actual period? Assume the ISR execution time is much less than 10 µs. Hint: Think of other parts of the system interfering.

10 µs is a relatively short period, which is also equal to the relative deadline of TimerCaptureISR1(). When the relative deadline shortens below a certain point, the erroneous period measurements are most likely due to missing the deadline and skipping one interrupt. Since the capture interrupts are periodic, by skipping one interrupt, the measurement is twice the actual period.

Since the ISR execution time is much less than 10 µs (the relative deadline), the culprit is latency. This latency can come from higher priority ISRs or from critical sections that disable interrupts.

b) How could the valid period measurement range be extended down to 5 µs? You may even suggest simple external hardware.

1. **Increase the priority of TimerCaptureISR1(). It may be possible to achieve lower latency by increasing the ISR priority, but this may cause another ISR to miss deadlines.**
2. **High latency could be caused by a critical section that disables interrupts. In such a case, optimize the critical section that causes high latency.**
3. **Pass the input signal through a frequency divider (prescaler). A 5 µs period becomes a 10 µs period after frequency division by 2. This requires just a single flip-flop. Some microcontrollers feature built-in prescalers for timer inputs. Software can then take the frequency division into account when calculating the period.**
4. **Use the timer edge count mode to count input edges in a fixed time interval, set by another timer. Then determine the period as the fixed time interval divided by the number of periods counted in that interval. This method works well for high-frequency inputs whose period does not need to be sampled often.**

```
uint32_t period; // measured period in clock cycles

void TimerCaptureISR1(void) { // ISR for capture interrupts
    static uint32_t last_count = 0;
    uint32_t count;

    <clear timer capture interrupt flag>;
    count = <read captured timer count>;
    period = (count - last_count) & 0xffffff;
    last_count = count;
}
```

31) Your embedded system is measuring the period of an analog signal that behaves as in the following figure. The signal is passed through an analog comparator, then fed to the input of a timer in capture mode. Your period measurements often come out much smaller than the actual analog signal period.

a) What adverse effect could the issue in part (a) have on the schedule of the real-time system? Explain.

The edges of the timer input signal are triggering interrupts. There can be many such edges in a short time interval during the threshold crossing of the analog signal, so many interrupts. The timer capture ISR can temporarily starve lower priority real-time tasks of the CPU, causing them to miss deadlines.

It could also cause the TimerCaptureISR to miss its deadlines give incorrect values.

b) What do you think is causing this problem?

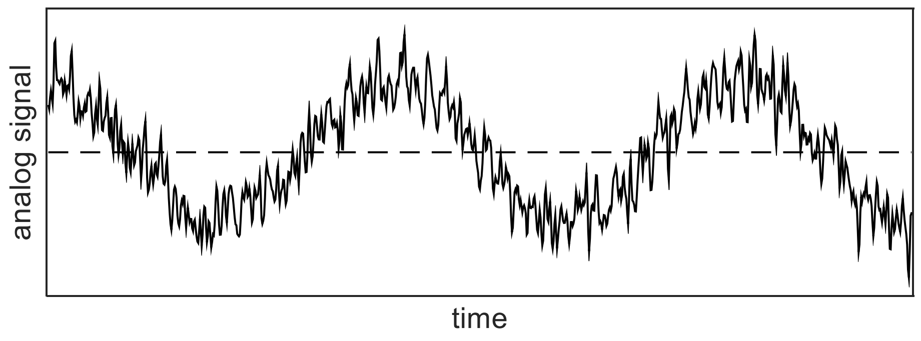
The noise in the input signal is causing multiple transitions in the timer input signal for each threshold crossing of the analog signal. These multiple transitions are being interpreted as very short waveform periods. Period is typically measured between two rising edges of the timer input signal.

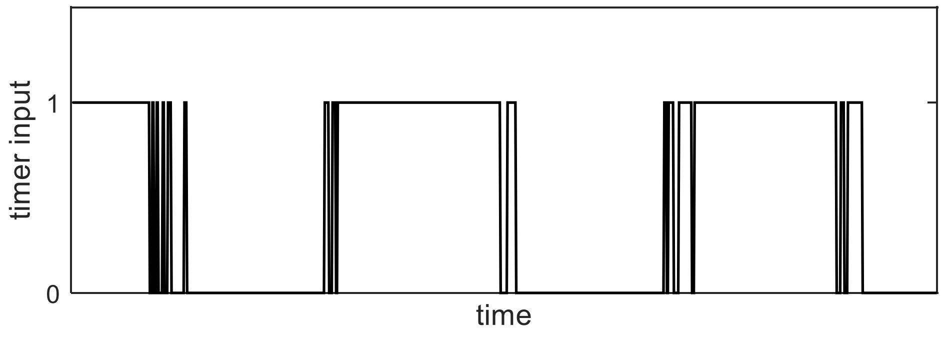
c) On the "timer input" axes below, **fill in** what you think the timer input signal approximately looks like. State any assumptions about the polarity of your comparator. See next page.

d) What can be done to eliminate this problem? Hint: The solution would likely require hardware changes.

Low-pass filter the analog signal to reduce high-frequency noise.

Add GPIO input conditioning that cleans up edges, similar to hardware debouncing to the timer input.


analog signal
time


timer input
time