# Exam 2 / Module 2

| Question # | Grader | Total Points | Points Received | Comments |
|---|---|---|---|---|
| #1:<br>Earliest Deadline First | | 30 | | |
| #2: Shared Data | | 35 | | |
| #3 Thread Synchronization | | 35 | | |
| Total | | 100 | | |

**Name: _____**

# Instructions & Restrictions

## Do's:

- Read the instruction on canvas before starting the test.
- Enter your name on page 1.
- This is an open book, open notes exam.
- If asked to write C code, you may use CCS as your editor to check your syntax.
- **Submit you test in pdf format to canvas by 10:15** or agreed to time if accommodations have been made.
    - Points may be deducted for lateness.
    - If you experience technical difficulties, contact the Professor immediately via zoom or by cell phone 508-523-0606.
- Make sure your answers are readable on your submitted version. Points cannot be given if the answers cannot be read.
- Review your pdf submission to verify all pages are present.
- All work must represent your knowledge and all code must be written by you.
- Read each question carefully, make sure you follow the specification when implementation details are given.
- You may ask questions of the instructor via zoom or by texting or calling 508-523-0606.

## Don'ts

- You many not collaborate with or communicate with others during the exam. Not verbally or in writing. Not in-person or electronically.
- Do not lose track of time. It is more important to answer every question then to get one question 100% correct.

# Question 1: Earliest Deadline First Scheduling (EDF)

A. What RMS Theory condition is not met when using the Earliest Deadline First scheduling strategy? [5]

The EDF scheduling strategy has dynamic scheduling. It changes the task priority based on which deadline is the closest. This violates the RMS condition that tasks have fixed priority.

B. What is the utilization upper bound for EDF that guarantees that the tasks will be schedulable? If this upper bound is exceeded, is there still a possibility that it can still be scheduled? Explain. [5]

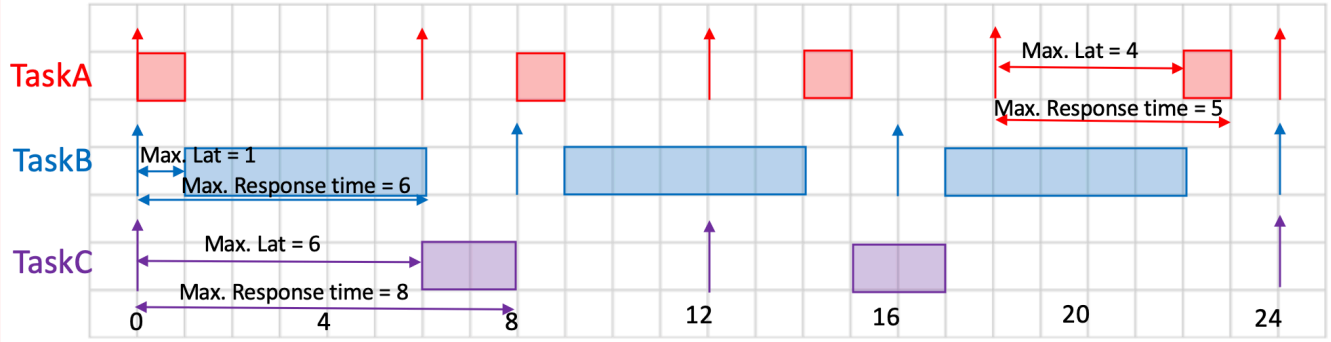
The utilization upper bound of the EDF is 100%. If the upper bound is exceeded it is impossible to schedule the event as the CPU utilization cannot be greater than 100%

The system below uses Earliest Deadline First Scheduling. It has three tasks with the characteristics shown in the table. If deadlines are the same, it uses a first come first serve rule to prioritize. For example, two events have a deadline of 10 msec. The release time of Task1 is 2 ms and the release time of the Task2 is 5 ms. Task1 will take priority because it happened first.

C. Use the graphical method to draw out the schedule either on the graph paper provided or using your own. Label the time axis and task names for full credit. [11]

D. From the graphical results determine the latency, response time, and if it is schedulable. Enter results in the table provided. [9]

| Event | Period | Execution Time | Latency | Response Time | Schedulable ? |
|---|---|---|---|---|---|
| TaskA | 6 ms | 1 ms | 4 ms | 5 ms | 5 ms < 6 ms, YES |
| TaskB | 8 ms | 5 ms | 1 ms | 6 ms | 6 ms < 8 ms, YES |
| TaskC | 12 ms | 2 ms | 6 ms | 8 ms | 8 ms < 12 ms, YES |



# Question 2: Shared Data

- Your system uses an interrupt InputISR() to take two ADC measurements one for x and one for y. **InputISR() is the highest priority interrupt in the system**.
- The **CalcSlope() function is called from main** to calculate the slope of the x and y values between when it was run last and the last sampled x and y values. If the slope is too large, its value is limited to MAX_SLOPE.
- **Other ISRs call the ReadSlope() function to read the slope** for use in their own calculations. This means that ReadSlope can interrupt CalcSlope at any time.
- The following code outlines the communication between the ISRs and main(). Assume that reads and writes to int variables are atomic.

```
#define NumSamp 1000
#define MAX_SLOPE 100
volatile int x[NumSamp];
volatile int y[NumSamp];
volatile int i = 0;
volatile int slope = 0;

void InputISR(void) { //highest priority interrupt
  // <clearing interrupt>
  // reading ADC0 and ADC1 to measure x & y values
      i++;
      if (i >=NumSamp) i = 0;
      x[i] = x_value;
      y[i] = y_value;
}

void CalcSlope(void) { //called from main program
      //save values from previous call.
      static int y_old = 0;
      static int x_old = 0;

      //calculate slope
      slope = (y[i]-y_old)/(x[i]-x_old);
      //update old values
      y_old = y[i];
      x_old = x[i];

     //limit the maximum slope
      if (slope > MAX_SLOPE) {
        slope = MAX_SLOPE;
      }
}
int ReadSlope(void) { //called from ISRs
  return slope;
}
```

A. Are there any shared data bugs in the **ReadSlope() function** itself? Explain.[5]

There are no shared data bugs in the ReadSlope() function. There is only a single atomic read in the function.

B. In lab, we are seeing errors and unexpected values in the slope, indicating there are shared data bugs in the **CalcSlope()** function. Find all the shared data bugs in this code. Explain the sequence(s) of events that can cause the problems seen. [10]

The CalcSlope() function has two shared data bugs.

Bug #1: Multiple Reads from the same global variable
The variable "i" is read multiple times. If the InputISR occurs between the "slope = (y[i]-y_old)/(x[i]-x_old);" line and the "x_old = x[i]" line. Then some of the reads will read the original i value and some will read the new i value. This will lead to an incorrect slope value to be calculated during the this function call or incorrect _old values during the next function call.

Bug #2: Multiple Writes to the same global variable
The variable slope is written twice. If ReadSlope data is called between the slope = (y[i]-y_old)/(x[i]-x_old);" line and the slope = MAX_SLOPE; line it may read an invalid value. The intermediate value may have slope > MAX_SLOPE.

C. Modify the CalcSlope() function below to fix the shared data bugs by disabling interrupts for the minimum time required to fix all bugs. Assume that interrupts are enabled entering the CalcSlope() function. [5]

```
void CalcSlope(void) { //called from main program
   //save values from previous call.
   static int y_old = 0;
   static int x_old = 0;

IntMasterDisable();
   //calculate slope
   slope = (y[i]-y_old)/(x[i]-x_old);
   //update old values
   y_old = y[i];
   x_old = x[i];

  //limit the maximum slope
   if (slope > MAX_SLOPE) {
      slope = MAX_SLOPE;
   }
IntMasterEnable();

}
```

D. Modify the CalcSlope() function below to fix the shared data bugs without disabling interrupts. Do **NOT** use synchronization techniques like Mailbox or FIFO [10]

```
void CalcSlopeFix(void) { //called from main program
   //save values from previous call.
   static int y_old = 0;
   static int x_old = 0;
   //calculate slope
   int j = i;
   int local_slope = (y[j]-y_old)/(x[j]-x_old);
   //update old values
   y_old = y[j];
   x_old = x[j];
  //limit the maximum slope
   if (local_slope > MAX_SLOPE) {
      slope = MAX_SLOPE;
   }
   else {
      slope = local_slope;
   }
}
```

E. There are two ways to fix the shared data bugs, one that requires disabling interrupts and one that does not require disabling interrupts. Which one would you choose to use? Explain. [5]

It is always better to fix the bug by changing the code. Disabling interrupts will increase the latency of all the interrupts in the system and lower the performance.

# Question 3: Synchronization Techniques

- You have a medical device that measures the amount of medicine delivered to the patient over a fixed period of time.
- The TimerISR() measures the dose since the last measurement and accumulates the value. TimerISR() is the highest priority interrupt.
- The main while loop waits a fixed amount of time and then calculates the total dose delivered over the entire procedure by adding the accumulated_dosage to the total dose. It then clears the accumulated value and prints the total dose to the LCD screen.
- The following code outlines the communication between the ISRs and main(). Assume that reads and writes to int variables are atomic.

```
volatile int accumulated_dosage = 0;
volatile int total_dosage =0;

void TimerISR(void) { //highest priority interrupt
// <clear interrupt>
// < measure ADC value of medicine dose since last interrupt>
   accumulated_dosage += last_dose;
}

main() {
   < initialization functions >
   while(1) {
         delay_us(FIXED_DELAY);
         total_dosage += accumulated_dosage;
         accumulated_dosage = 0;
         PrintDosage();
   }
}
```

A. Explain the shared data bug. Where in the while () loop would the TimerISR() need to occur to cause the shared data bug? How would the bug affect the value of the total_dosage? [10]

Both the while() loop and the TimerISR write to the same global variable accumulated_dosage. If the ISR is triggered after updating total_dosage but before clearing the accumulated dosage, it will cause a shared data bug.

TimerISR will modify and write the accumulated dosage in the ISR, upon returning to the while(1) loop this update will be overwritten back to zero. The total_dosage value will be missing the value from that interrupt and will under estimate the dosage delivered.

B. In lecture we looked at two synchronization methods, one using a Mailbox and one using a FIFO. For the conditions below, which of these methods would you use? If using a FIFO what is the minimum FIFO size needed to guarantee no data loss?

If the TimerISR() occurs every 1 msec and the fixed while loop delay is 500 usec. Which method would you use? Explain. [5]

The while() loop delay is less than the TimerISR period. For this case, I would use a Mailbox because its execution time and the resources used are smaller than the FIFO. Since the total_dosage is guaranteed to be updated between each ISR, the Mailbox will always be empty for the next TimerISR call and it will not experience any data loss.

If the TimerISR occurs every 1 msec and the fixed while loop delay is 10 msec. Which method would you use? Explain. [5]

The while() loop delay is greater than the TimerISR period. For this case, I would use a FIFO because the Mailbox could not read data fast enough. This would cause data loss. The TimerISR is called once every ms, 10 samples can be taken between updates in the while loop. The required fifo size = the number of samples + 1 = 11.

C. Fix the shared data problem using a Mailbox. Modify the TimerISR() and main function as well as add any required global and local variables to implement the fix. Assume that the fixed while loop delay guarantees that no data is lost. [15]

```
volatile int bMailboxFull = 0;

void TimerISR(void) { //highest priority interrupt
// <clear interrupt>
// < measure ADC value of medicine dose since last interrupt>
  if (!bMailboxFull) {
    accumulated_dosage = last_dose;
    bMailboxFull = 1;
  }
}


  while(1) {
    delay_us(FIXED_DELAY);
    if (bMailboxFull) {
      total_dosage += accumulated_dosage;
       // write to accumulated_dosage removed
      bMailboxFull = 0;
    }
    PrintDosage();
  }
```