Department of Electrical and Computing Engineering

UNIVERSITY OF CONNECTICUT

ECE 3411 Microprocessor Application Lab: Fall 2015

Quiz VI

There are 3 questions in this quiz. There are 13 pages in this quiz booklet. Answer each question according to the instructions given.

You have **45 minutes** to answer the questions.

Some questions are harder than others and some questions earn more points than others—you may want to skim all questions before starting.

If you find a question ambiguous, be sure to write down any assumptions you make.
Be neat and legible. If we can't understand your answer, we can't give you credit!

Write your name in the space below. Write your initials at the bottom of each page.

THIS IS A CLOSED BOOK, CLOSED NOTES QUIZ.
PLEASE TURN YOUR NETWORK DEVICES OFF.

Any form of communication with other students is considered cheating and will merit an F as final grade in the course.

Do not write in the boxes below

1 (x/30)	2 (x/40)	3 (x/30)	Total (xx/100)

Name:

Student ID:

1. **[30 points]:** Below is a program layout with comments explaining what happens during program execution. Also the meaning of all register initializations is given (there is no need to look into the ATmega328P data sheet).

You should pay attention to the main body which initializes Timer1, and polls its value before an ADC measurement in sleep mode starts and after the execution of an "ADC task" finishes. The difference of the two values is converted to micro seconds and added to a variable busy. The goal for busy is to measure the time during which the MCU is doing "useful" work. The code that is related to busy is highlighted with vertical bars.

After the program layout below, the first subproblem asks you what is truly measured by busy in the program and the second subproblem asks you to explain the code which converts the difference to micro seconds.

```
... we assume a clock frequency of 20MHz    ...
... inclusion of packages                   ...
... declaration of global variables         ...

// ------------------------------------------------- //

ISR (TIMER0_COMPA_vect)
{
    /* Update task timer */
    if (taskADC_timer >0 )  {--taskADC_timer;}
}

// ------------------------------------------------- //

ISR (ADC_vect)
{
    /* Read a 10-bit conversion */
    AinLow = (int)ADCL;
    Ain = (int)ADCH*256;
    Ain = Ain + AinLow;
}

// ------------------------------------------------- //

void taskADC(void)
{
    /* Reset task timer */
    taskADC_timer = 400;

    //Convert Ain into a voltage
    voltage = ((1.0*Ain)/1024.0)*5.0;

    ... Some more computation: sometimes taking more and sometimes taking less time ...
    ... However, no matter how long taskADC() takes, its execution is always <= 200 ms ...
}
```

Initials:

```
int main(void)
{
    ... initialization variables ...

|    // set up timer 1 for 3.2 micro second counter increments
|    TCCR1B = 3;           //set prescalar to divide by 64

    //set up timer 0 such that ISR(TIMER0_COMPA_vect) is called every 1 milli second
    OCR0A  = 77;          //Set the compare reg to 78 time ticks
    TIMSK0 = (1<<OCIE0A); //Turn on timer 0 cmp match ISR
    TCCR0B = 4;           //Set prescalar to divide by 256
    TCCR0A = (1<<WGM01);  //Turn on clear-on-match

    // initialize the ADC
    ADMUX  = 6;                          // Select ADC Channel 6
    ADCSRA = (1<<ADEN) | (1<<ADIE) + 7 ; // Enable AD converter, enable its interrupt,
                                         // set prescalar (notice that the ADSC bit is
                                         // not set, so no ADC conversion is started)
    SMCR   = (1<<SM0) ;                  // Choose ADC sleep mode

    sleep_enable();
    sei();

    while (1)
    {
        if (taskADC_timer == 0)
        {
|           // Measure timer 1
|           T1poll_before = TCNT1;

            //Perform an ADC measurement in sleep mode, and execute taskADC
            sleep_cpu();
            taskADC();

|           //Measure timer 1 again and update busy with the amount of micro seconds that
|           //have passed: every TCNT1 to TCNT1+1 increment takes 3.2 micro seconds.
|           T1poll_after = TCNT1;
|
|           if (T1poll_after > T1poll_before) {
|               busy += (T1poll_after - T1poll_before)*3.2;
|           } else {
|               busy += ( (T1poll_after - T1poll_before) + 65536 ) * 3.2;
|           }
        } /* end of if (taskADC_timer == 0) */
    } /* end of while(1) */
} /* end of main() */
```

Initials:

The data sheet (section 9.4) writes for the ADC Noise Reduction Mode that "... the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the external interrupts, 2-wire Serial Interface address match, Timer/Counter2 and the Watchdog to continue operating (if enabled) ...". This means that all other hardware modules stop working, in particular, the other timers/counters stop incrementing.

A. (20 points) Answer with "never", "sometimes", or "always", whether the execution times (measured in micro seconds) of the following procedures are added into `busy` variable. Explain your answers.

(a) `ISR(TIMER0_COMPA_vect)`

(b) `ISR(ADC_vect)`

(c) `sleep_cpu()`

(d) `taskADC()`

Initials:

B. (10 points) The program assumes that `taskADC()` always takes $\leq$ 200 ms. Use this assumption to explain why the code

```
if (T1poll_after > T1poll_before) {
    busy += ( T1poll_after - T1poll_before ) * 3.2;
} else {
    busy += ( (T1poll_after - T1poll_before) + 65536 ) * 3.2;
}
```

correctly adds to `busy` the time in micro seconds that passed between the polling of `T1poll_before` and the polling of `T1poll_after`.

Initials:

2. [40 points]: Given that the clock frequency ($clk_{I/O}$) of ATmega328P is 16MHz, write a program that uses watchdog timer in interrupt and reset modes simultaneously. The detailed functionality of the program is as follows:

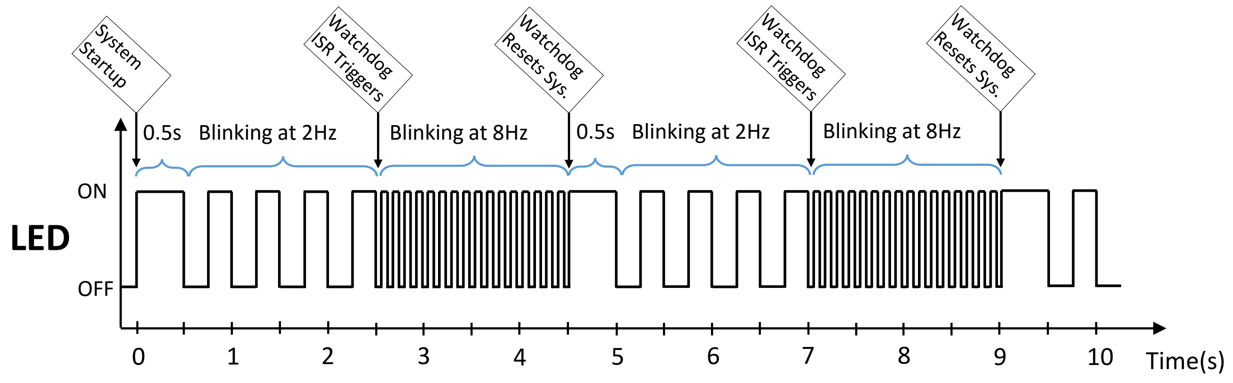
(a) Upon the system startup/reset, a LED connected to PB5 lights up for $0.5s$ and then turns off.

(b) After this, the main function starts blinking the LED at approximately $2Hz$.

(c) After 2 seconds, the watchdog interrupt occurs and it keeps blinking the LED at $8Hz$ until the system reset occurs.

To simplify the implementation, use `_delay_ms()` or `_delay_us()` routines inside `while(1)` loops to implement the LED blinking function.

The following figure shows the detailed timing of the LED for the desired system. Notice that, after the watchdog interrupt, it takes another watchdog timeout period for the system reset to occur.



Implement this system by filling the gaps in the provided code layouts of the subsections A, B and C. You may use the provided data sheet for your reference.

Initials:

A. Initialization: (15 points)
Complete the function `initialize_all(void)` as instructed below:

```
/*********** ECE3411 Quiz 6, Task 2 ************/
// Define any variables here if needed




/* Initialization function */
void initialize_all(void)
{
    /* Configure the LED pin and implement the functionality of step (a) */









    /* Configure the Watchdog timer in Reset & Interrupt mode */
    /* Set a prescaler such that watchdog times out after 2 seconds */









    /* Any other initializations here if needed */







} /* End of initialize_all() */
```

Initials:

B. Watchdog timeout ISR Implementation: (10 points)
Write the ISR `ISR(WDT_vect)` to achieve the desired functionality.

```
/* Watchdog timeout ISR */
ISR(WDT_vect)
{
    /* Blink the LED at 8Hz using _delay_ms() or _delay_us() function */




} /* end of Watchdog timeout ISR */
```

Initials:

C. Main Function Implementation: (15 points)
Write the function `main()` to complete the system functionality.

```
/* Main Function */
int main(void)
{
    /* Cleanup any aftereffects of Watchdog timeout reset */









    initialize_all();     // Initialize everything
    sei();                // Enable Global Interrupts

    /* Event loop */
    while(1)
    {
        /* Blink the LED at 2Hz using _delay_ms() */














    }

} /* End of main() */
```

Initials:

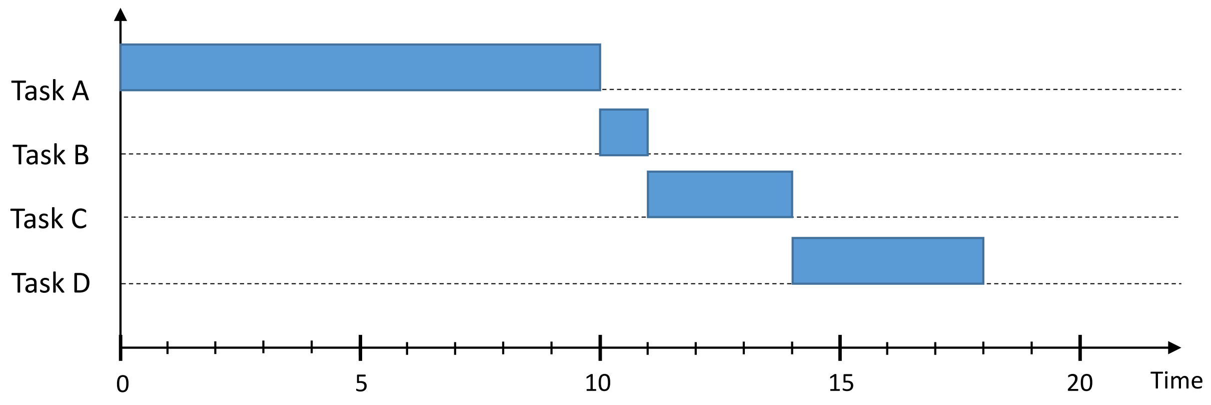
3. [30 points]: Table 1 shows the characteristics of four tasks that need to be scheduled on the MCU.

"Ready Time" indicates when the corresponding task is ready to execute. In this example, all the tasks are ready and want to execute as soon as the system starts, i.e. at time 0. "Required CPU Time" indicates how many time units are needed for the task to finish.

Table 1: Task Specifications

Task	Ready Time	Required CPU Time
A	0	10
B	0	1
C	0	3
D	0	4

The following figure shows an example of First Come First Serve scheduling of these tasks assuming the order A, B, C, and then D.



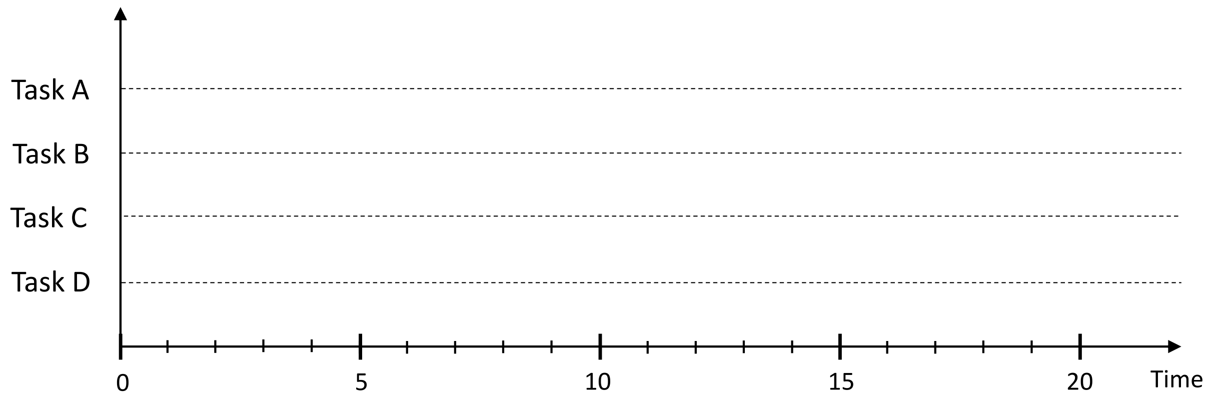
The following table shows the completion times of the tasks and their corresponding wait times (i.e. while the task is suspended and waiting for the CPU) under First Come First Serve scheduling scheme.

Table 2: Analysis under First Come First Serve Scheduling

Task	Ready Time	Required CPU Time	Completion Time	Wait Time
A	0	10	10	0
B	0	1	11	10
C	0	3	14	11
D	0	4	18	14
Average			13.25	8.75

Initials:

A. Round Robin Scheduling: (15 points)
Plot how the tasks A, B, C, and D will be scheduled on the CPU under **Round Robin Scheduling** with a **time slice of 1 time unit**. I.e. assuming an order of A, B, C, and D; the tasks take turns for the CPU and each task gets the CPU for 1 time unit in each turn until the task finishes. Assume that no time is wasted in context switching between the tasks.



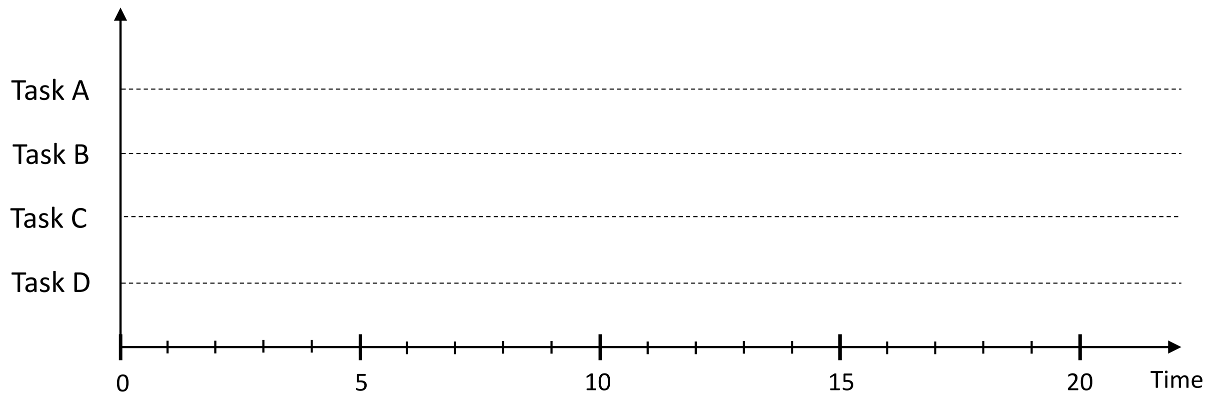
Use the plot above to complete the following table.

Table 3: Analysis under Round Robin Scheduling

Task	Ready Time	Required CPU Time	Completion Time	Wait Time
A	0	10		
B	0	1		
C	0	3		
D	0	4		
Average				

Initials:

B. Shortest Remaining Time First Scheduling: (15 points)
Plot how the tasks A, B, C, and D will be scheduled on the CPU under **Shortest Remaining Time First Scheduling**. I.e. whichever task needs the shortest amount of CPU time to finish gets the CPU first. Once this task is finished, another task that needs the smallest CPU time is executed. Assume that no time is wasted in scheduling a new task once a task finishes.



Use the plot above to complete the following table.

Table 4: Analysis under Shortest Remaining Time First Scheduling

Task	Ready Time	Required CPU Time	Completion Time	Wait Time
A	0	10		
B	0	1		
C	0	3		
D	0	4		
Average				

End of Quiz

Please double check that you wrote your name on the front of the quiz.

Initials: