UNIVERSITY OF MANITOBA
Final Exam

Winter 2007

COMPUTER SCIENCE

Real-time Systems

Date: Fri, 20th April 2007
Time: 09:00 - 12:00
Room: Frank Kennedy Brown Gym (314-345)
(Time allowed: 180 Minutes)

NOTE: Attempt all questions.
This is a closed book examination.
Use of non-programmable calculators is permitted.
Use of any other electronic equipment is strictly forbidden.
You must show your work to receive full marks.

SURNAME:_____________________________
FORENAME(S):_________________________
STUDENT ID:___________________________

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>100</td>
</tr>
</tbody>
</table>

CONTINUED
Section A: Notes

The following assumptions are valid for all questions throughout this exam

- the code is executed on an AVR Butterfly development board,
- the processor on the AVR Butterfly is running at 2MHz,
- the first instruction of the interrupt service routine (ISR) will be executed exactly 6 clock cycles (4 clock cycles to acknowledge the interrupt and save the return address on the stack, and 2 clock cycles for the jump from the interrupt vector table to the first instruction of the ISR) after the interrupt occurs, assuming that interrupts are enabled and no other higher priority interrupts are pending.

Section B: Ports and Timers

1. The code below generates square waves on pins PE5 and PE6. `initWaveGen` was called before interrupts were enabled.

```c
void initWaveGen( void ) {
    DDRE |= (1 << DDE6) | (1 << DDE5);
    PORTE = 0x00;
    TCCR0A = (0 << FOC0A) | (0 << WGM00) | (0x01 << COM0A0) | (0x01 << CS00);
    TIFR0 = (1 << TOV0);
    TIMSK0 = (1 << TOIE0);
}
ISR(TIMER0_OVF_vect) {
    static uint8_t count = 0;
    if ( count == 0 ) {
        PINE = (1 << PE5);
        TCNT0 = 170;
    } else if ( count == 1 ) {
        PORTE = (1 << PE6);
        TCNT0 = 100;
    } else if ( count == 2 ) {
        PORTE = 0x00;
        TCNT0 = 200;
        count = 0xFF;
    }
    count++;
}
```
Show the waveform generated on pins E5 and E6 respectively in the timing diagram below. You start your analysis on the rising edge of pin E5, which occurred at time $t_0 = 125$ microseconds. Show the timing of each raising or falling edge of the pins E5 or E6 until the next raising edge of pin E5.

You can ignore any effects on the timing due to the code above not being balanced. The first transition (raising edge on pin E5 is already shown in the answer box.)

[6 marks]

2. You are an embedded systems designer and are given the following code. Timer 0 is the only interrupt source in the system and interrupts are always enabled.

```c
TIMERO_OVF_vect __attribute__ ((naked));
ISR(TIMER0_OVF_vect) {
    secret_prologue();
    your_routine1();
    secret_epilogue()
}
```

The routines `secret_prologue()` and `secret_epilogue()` are impossible to reverse engineer and top secret. You do not have the security clearance to access the source code. The only thing you can change is `your_routine1`. 

CONTINUED
The \texttt{TIMER0} overflow ISR has been set up correctly with a prescalar of 1.

Implement a routine \texttt{your_routine1} which stores the number of clock ticks used by the routine \texttt{secret_prologue} in the global variable \texttt{ticks_prologue}.

If it is impossible to implement such a routine then say so in your answer and explain why.

[7 marks]

```c
volatile uint8_t ticks_prologue;
inline void your_routine1( void ) {
  ticks_prologue = TCNT0-6-4;
}
/* Assuming that \texttt{secret_prologue} does not modify \texttt{TCNT0}.
Otherwise impossible. */
```

3. What is the frequency of the square wave generated on pin E5 given the code below. Timer 0 has been initialized correctly with a pre-scalar of 1.

[7 marks]
The frequency of the generated wave is: ____________ Hz

33 clock cycles, 2Mhz/66 = 30 kHz

Section C: Sound Generation

4. Given below is a small synthesized sound sample.
SAMPLE_NAME[9] = { 0x80, 0xBC, 0xDD, 0xD5, 0xAB, 0x75, 0x4E, 0x43, 0x54 }; 

Calculate the frequency of the resulting wave if the sound is played back with a sample rate of 12,000 Hz.

If it is impossible to determine the frequency of the sound wave, then say so in your answer and explain why.

[2 marks]

12000 Hz/ 9 Bytes per Wave = 1333.33 Hz

5. Delta modulation is a modulation technique where every 8 bit sample is replaced by a single bit based on whether the next sample is larger (turn speaker on) or less than or equal to the current sample (turn speaker off).

Given below is the listing file of the interrupt handler which plays back a 16 sample delta modulated waveform on the speaker of the AVR Butterfly.

23:sound_gen.c **** volatile uint16_t sample = 0x157A;
25:sound_gen.c **** ISR(TIMER0_OVF_vect) {
125 0010 1F92  push __zero_reg__
126 0012 0F92  push __tmp_reg__
127 0014 0FB6  in __tmp_reg__,__SREG__
128 0016 0F92  push __tmp_reg__
129 0018 0FB6  in __tmp_reg__,__SREG__
130 0019 0F92  push __tmp_reg__
125 0010 1F92  push __zero_reg__
126 0012 0F92  push __tmp_reg__
127 0014 0FB6  in __tmp_reg__,__SREG__
128 0016 0F92  push __tmp_reg__
129 0018 1124  clr __zero_reg__
130 001a 2F93  push r18
131 001c 3F93  push r19
132 001e 8F93  push r24
133 0020 9F93  push r25
26:sound_gen.c **** uint8_t bit;
27:sound_gen.c **** uint8_t const mask = ~( 1 << PB5 );
29:sound_gen.c **** bit = 0;
30:sound_gen.c **** if ( ( sample & 0x8000 ) != 0 ) {
137 0022 8091 0000  lds r24,sample
138 0026 9091 0000  lds r25,(sample)+1
139 002a 292F  mov r18,r25
CONTINUED
Calculate the execution time of the code above assuming that sample is equal to 0x157A when entering the ISR. Include the time needed to enter the ISR after an overflow on timer 0 occurred.

[7 marks]

The execution time of the code is 50 usec

CONTINUED
6. Add ballast code to the code shown above to make sure that the resulting code has a constant delay through all possible execution paths.
If the code is already balanced, or if it impossible to balance the code so that all execution paths have the same duration, then say so in your answer and explain why.

[4 marks]

No balancing is needed since the first if..then..else is replaced by rol,clr,rol and the second brne is constant (PB5).

7. What is the LCD output of the following program when executed on an AVR Butterfly. You can assume that the LCD has been initialized correctly.

```c
void sub2( void ) { 
    char * xx = "DADC";
    char * y = "BDCA";
    volatile char * x = xx;

    asm volatile ( "
        
        1: ld r24,%a0   \n
        1dd r25,%a2+3  \n
        cp r24,r25  \n
        breq 2f  \n
        st %a0+,r25  \n
        sbiw %A2,1  \n
        rjmp 1b  \n
4     \n
CONTINUED
```
"=b"(x):"O"(x),"b"(y):"r24","r25");
LCD_puts(xx,1); /* line: LCD_1 */
LCD_puts(x,1); /* line: LCD_2 */
}

[7 marks]

Output on LCD in line LCD_1 is: ACDC

Output on LCD in line LCD_2 is: DC

Section D: Threads, Context Switches

8. Given below are code fragments from an application with three threads: ThreadA, ThreadB, and ThreadC. Each thread will execute the body of the loop and then return. Once all threads have returned, the value of the variable buffer is printed out by the main thread.
A counting semaphore `sem` is used to synchronize access to shared memory buffer `buffer`. Although the code for the threads is very similar, they use different methods for dealing with synchronization and semaphores.

All threads run at normal priority. A context switch can only occur at the beginning or end of each line of code, that is you may assume that no context switch occurs in the middle of a line of code.

```c
volatile uint8_t index = 0;
struct Semaphore sem;

int main( ) {
  ...
  initSemaphore( & sem );
  startThread( ThreadA, NORMAL_PRIORITY );
  startThread( ThreadB, NORMAL_PRIORITY );
  startThread( ThreadC, NORMAL_PRIORITY );
  waitForThreadsToFinish();
  printBuffer( buffer );
}

void ThreadA( void ) {
  char const s1 = "AB";
  uint8_t i;

  for(i=0; i< strlen( s1 ); i++ ) {
    TA1 while( try_and_wait( & sem ) );
    TA2 buffer[index++]=s1[i];
    TA3 release( & sem );
    TA4 if ( index >= sizeof( buffer ) )
    TA5 index = 0;
  }
}

void ThreadB( void ) {
  char const s2 = "ab";
  uint8_t i;

  for(i=0; i< strlen( s2 ); i++ ) {
    TB1 while( try_and_wait( & sem ) );
    TB2 buffer[index++]=s2[i];
    TB3 if ( index >= sizeof( buffer ) )
    TB4 index = 0;
    TB5 release( & sem );
  }
}
```

CONTINUED
The content of the variable buffer is shown in the answer box below. A '?' indicates that the value at this memory location is unknown. Writing to index -1 or 5 indicates a memory underflow or overflow respectively.

If it is possible that the given content of the variable buffer is the result of the code above, then show one sequence of instructions of the statements numbered TA1-TA5, TB1-TB5, and TC1-TC3 that leads to the given content of the variable buffer. You can ignore the other instructions in the code. If it is impossible that the code above resulted in this content for the variable buffer then say so in your answer and explain why.

[7 marks]

Impossible since TA2 and TB2 are both atomic and will increment the index variable.
is impossible that the code above resulted in this content for the variable buffer then say so in your answer and explain why.

[7 marks]

<table>
<thead>
<tr>
<th>Index</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>?</td>
<td>a</td>
<td>A</td>
<td>1</td>
<td>2</td>
<td>B</td>
<td>b</td>
</tr>
</tbody>
</table>

TB1-5, TA1-5, TC1-5, TC1-5, TA1-3, TB1-5, TA4-5.

10. A thread has allocated stack memory in the range 0x150 to 0x173. The content of this memory is shown below.

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x173</td>
<td>0xCD</td>
</tr>
<tr>
<td>0x172</td>
<td>0xDC</td>
</tr>
<tr>
<td>0x171</td>
<td>0xAC</td>
</tr>
<tr>
<td>0x170</td>
<td>0xBC</td>
</tr>
<tr>
<td>0x16F</td>
<td>0x9A</td>
</tr>
<tr>
<td>0x16E</td>
<td>0xE4</td>
</tr>
<tr>
<td>0x16D</td>
<td>0x81</td>
</tr>
<tr>
<td>0x16C</td>
<td>0x56</td>
</tr>
<tr>
<td>0x16B</td>
<td>0x11</td>
</tr>
<tr>
<td>0x16A</td>
<td>0x34</td>
</tr>
<tr>
<td>0x169</td>
<td>0x20</td>
</tr>
<tr>
<td>0x168</td>
<td>0x06</td>
</tr>
<tr>
<td>0x167</td>
<td>0x1E</td>
</tr>
<tr>
<td>0x166</td>
<td>0x5A</td>
</tr>
<tr>
<td>0x165</td>
<td>0xB7</td>
</tr>
<tr>
<td>0x164</td>
<td>0x00</td>
</tr>
<tr>
<td>0x163</td>
<td>0x01</td>
</tr>
<tr>
<td>0x162</td>
<td>0x97</td>
</tr>
<tr>
<td>0x161</td>
<td>0x87</td>
</tr>
<tr>
<td>0x160</td>
<td>0x00</td>
</tr>
<tr>
<td>0x15F</td>
<td>0xAB</td>
</tr>
<tr>
<td>0x15E</td>
<td>0xCC</td>
</tr>
<tr>
<td>0x15D</td>
<td>0xA0</td>
</tr>
<tr>
<td>0x15C</td>
<td>0xEE</td>
</tr>
<tr>
<td>0x15B</td>
<td>0xBE</td>
</tr>
<tr>
<td>0x15A</td>
<td>0x10</td>
</tr>
<tr>
<td>0x159</td>
<td>0x02</td>
</tr>
<tr>
<td>0x158</td>
<td>0x34</td>
</tr>
<tr>
<td>0x157</td>
<td>0x00</td>
</tr>
<tr>
<td>0x156</td>
<td>0x45</td>
</tr>
<tr>
<td>0x155</td>
<td>0x03</td>
</tr>
<tr>
<td>0x154</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x153</td>
<td>0x00</td>
</tr>
<tr>
<td>0x152</td>
<td>0x50</td>
</tr>
<tr>
<td>0x151</td>
<td>0x04</td>
</tr>
<tr>
<td>0x150</td>
<td>0x66</td>
</tr>
</tbody>
</table>
```

The following routine is used to restore the thread context.

```
ldi    __SP_L__, 0x5A
ldi    __SP_H__, 0x01
pop    r25
pop    r24
pop    r19
pop    r18
pop    __tmp_reg__
out    __SREG__, __tmp_reg__
pop    __tmp_reg__
pop    __zero_reg__
retl
```
Show the value of the indicated registers after the execution of the `reti` instruction in the code above. [6 marks]

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP_L</td>
<td>0x63</td>
</tr>
<tr>
<td>SP_H</td>
<td>0x01</td>
</tr>
<tr>
<td>r25</td>
<td>0x10</td>
</tr>
<tr>
<td>r24</td>
<td>0xBE</td>
</tr>
<tr>
<td>r19</td>
<td>0xEE</td>
</tr>
<tr>
<td>r18</td>
<td>0xA0</td>
</tr>
<tr>
<td>r1</td>
<td>0x23</td>
</tr>
<tr>
<td>r0</td>
<td>0x00</td>
</tr>
<tr>
<td>SREG</td>
<td>0xCC</td>
</tr>
<tr>
<td>Instruction pointer</td>
<td>0x9787</td>
</tr>
</tbody>
</table>

**Section E: AD Converter**

11. An 8-bit cascaded AD converter uses two 4-bit AD converters, a 4-bit DA converter and a subtract and multiply by 16 circuit.

The components for the cascaded AD converter are shown below. Enter all necessary connections into schematic below so that the 8-bit conversion result of input $A_0$ appears on output $O_0..O_7$. [10 marks]
12. Implement an interrupt driven AD converter routine for an analog sensor. The sensor’s output is in the range of 0 to 1 Volt and it is connected to pin F2.

The accuracy of the result as well as power consumption is important so implement the most accurate result with the least amount of power that you can.

You must implement the following three parts:

- `ad_result`: global `uint8_t` variable which holds the result of the last AD conversion
- `void init( void )`: initializes the AD subsystem and sets up Port F2.
- `ISR(ADC_vect)`: interrupt that is called when the AD conversion is completed.

[10 marks]
extern volatile uint8_t ad_result;

void init( void ) {
    ADMUX = (0x03 << REFS0) | (1 << ADLAR) | 0x02 /* Port F2 */;
    ADCSRA = (1 << ADEN) | (0 << ADSC) | (1 << ADIE) | (0x00 << ADPS0);
    ADCSRB = 0x00; /* Free running mode, fastest pre-scalar */
    DIDR0 = DIDR0 | (1 << ADC2D); /* Turn off digital IO to save power */
    ADCSRA = ADCSRA & (~ (1 << ADIF)); /* Turn off IF */
    ADCSRA = ADCSRA | (1 << ADSC ); /* Start the free run conv. */
}

ISR(ADC_vect) {
    ad_result = ADCH;
}

Section F: Scheduling

13. Given below are three periodic tasks: T1, T2, and T3 with the following characteristics. The operating system uses a 10 millisecond timer.

<table>
<thead>
<tr>
<th>Task</th>
<th>Priority</th>
<th>Workload</th>
<th>Period</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>10</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>T2</td>
<td>2</td>
<td>20</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>T3</td>
<td>3</td>
<td>20</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

Show the output of fixed priority scheduling (FPS) on the workload above for time $t = 0$ to $t = 120$. The first scheduled task is already shown in the answer box for you. If it is impossible to complete the FPS schedule, then say so in your answer and explain why.

[6 marks]
14. Show the output of the earliest deadline first (EDF) scheduler given the workload above for time $t = 0$ and $t = 120$. In case more than one task has the earliest deadline, schedule the highest priority task with the same deadline.

If it is impossible to complete the EDF schedule, then say so in your answer and explain why.

[6 marks]
15. Given are two tasks 1b and 2b shown below. The period and deadline for Task 2b is unknown. What is the minimum period in multiples of 10 milliseconds for Task 2b, such that there exists at least one schedule that does not violate any deadlines.

<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
<th>Workload</th>
<th>Period</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1b</td>
<td>1</td>
<td>20</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Task 2b</td>
<td>2</td>
<td>40</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

[8 marks]

The minimum period for Task 2b such that a schedule without deadline violation exists is **120 milliseconds**.
Additional work pages