EE 4427/5527 – Embedded Systems Engineering

Midterm Exam

March 17, 2015 (9:30 – 10:30 a.m.)

Your Name: ____________________    Total Points: __ _______
Last 5 of Bengal ID: _____________

There are 4 groups of questions. Full score is 100 points. For maximum credit, show and organize all your work neatly and clearly.

Topics covered in this Midterm Exam include:

A. Fundamentals of PIC32MX (45 points)
B. Registers for ipl7 interrupts (10 points)
C. Configuration of peripheral timers (10 points)
D. Multi-vectored ISR code (35 points)
Q1. (45%) Respond to the following statements with True or False. If your response to a statement is False, explain briefly but in sufficient details why the statement is False.

1. The MPLAB X/C32 compiler supports both integer and floating-point complex data types.  
   T

2. At power-up or after a reset, the special initialization code crt0 is executed before main().  
   T

3. On the PIC32MX platform, integer type long is treated the same as the integer type int.  
   T

4. The Program Counter (PC) of the processor always contains the address of the next instruction.  
   T

5. The MPLAB “configuration bits” can be used to configure the PIC32MX peripheral bus speed.  
   T

6. Single-vectored INT allows every interrupt source to have its own Interrupt Service Routine.  
   F  
   Only 1 ISR is allowed. Vector 0 (i.e. the Core Timer) is the only permitted interrupt source.

7. Nonmaskable interrupts cannot be ignored by the processor under any circumstances.  
   T

8. An Interrupt Service Routine returns an integer value 0 upon its successful completion.  
   F  
   ISRs are type void, i.e. they do not return a value even upon successful completion.

9. PIC32MX's interrupt control module (ICM) can handle up to 32 independent INT events.  
   F   
   The ICM can handle up to 96 interrupt events from 64 different sources, not 32 events.

10. Once an interrupt flag (-IF) is set, it must be cleared to allow the same ISR to be invoked again.  
    T

11. The LEDs on the PIC32 board can be directly controlled through PORTB pins.  
    F  
    PORTD, not PORTB, directly controls the I/O of the PIC32 LEDs.

12. If two interrupt events with the same priority and sub-priority occur at the same time, these two interrupt events will be served in a random order as determined by the ICM hardware.  
    F  
    A "Natural Order" table (see p. 86 of textbook) is used, not random order, for final arbitration.

13. Nesting of interrupt calls allows a higher-priority interrupt event to overtake a lower-priority event.  
    F  
    Nesting is provided for both the single- and multi-vectored ISRs. "ei" is used for lower-end MCUs.

14. PORTB is by default for analog I/O, and AD1PCFG needs to be set to 0xffff to make it digital.  
    T

15. For PIC32MX, the division operation for type long long is handled by the MIPS instruction div.  
    F  
    A subroutine from library module libgcc2.c is used by way of a jal call, not the MIPS instruction div.

Q2. (10%) Why does PIC32MX provide two sets of 32 registers for its interrupt handling? What are the benefits and limits of this?

The second set of 32 registers is called the "shadow set". They are used ONLY for the highest priority interrupts, i.e. at level 7. The benefits: cut down on context switching time, since fewer instructions are then needed for an ISR of priority level 7. The limits: you only have one extra set, so can't have more than one priority-7 interrupt arriving at the same time; otherwise they will be serialized.
Q3. This question relates to the configurations of peripheral timers Timer1 and Timer4.

a. (5%) Timer1 has been configured as T1CON = 0x8010; and the constant DELAY is set to 64000, while frequency of the peripheral bus clock is configured at 64 kHz. What will be Timer1’s real-time delay in seconds?

\[
T_{\text{delay}} = \frac{1}{F_{\text{pb}}} \times \text{Prescaler} \times \text{DELAY} \\
T_{\text{delay}} = \frac{1}{(64 \times 10^3)} \times 8 \times 64000 = 8 \text{ sec}
\]

b. (5%) Using the peripheral library <plib.h>, we can specify the following code for Timer4:

```c
WriteTimer4(0);
OpenTimer4(T4_ON|T4_PS_1_256, 0xFF);
```

Complete the following C statements to make them equivalent to the above code.

```c
TMR4 = ___0___;
T4CON = ___0x8070___;
PR4 = ___0xFF___;
```

### Timer1 Pre-scaler Scheme:

![Timer1 Pre-scaler Scheme Diagram](image1)

### Timer2/Timer4 Pre-scaler Scheme:

![Timer2/Timer4 Pre-scaler Scheme Diagram](image2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit Range</th>
<th>Bit 31/23/15/7</th>
<th>Bit 30/22/14/6</th>
<th>Bit 29/21/13/5</th>
<th>Bit 28/20/12/4</th>
<th>Bit 27/19/11/3</th>
<th>Bit 26/18/10/2</th>
<th>Bit 25/17/9/1</th>
<th>Bit 24/16/8/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1CON(3,4,5)</td>
<td>31:24</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>23:16</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>15:8</td>
<td>ON</td>
<td>FRZ</td>
<td>SIDL</td>
<td>TWDIS</td>
<td>TWIP</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>7:0</td>
<td>T Gate</td>
<td>—</td>
<td>TCKPS&lt;1:0&gt;</td>
<td>—</td>
<td>TSYNC</td>
<td>TCS</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>TxCON(3,4,5)</td>
<td>31:24</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>TSYNC</td>
<td>TCS</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>23:16</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>15:8</td>
<td>ON</td>
<td>FRZ</td>
<td>SIDL</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>7:0</td>
<td>T Gate</td>
<td>—</td>
<td>TCKPS&lt;2:0&gt;</td>
<td>T32(1)</td>
<td>—</td>
<td>TCS</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Q4. (35 points) Write a complete multi-vectored PIC32MX interrupt-handling program in correct MPLAB C32 language syntax, i.e. with the needed ISRs, #include, #define, main( ), comments, etc., as required. The program will satisfy the design requirements listed below. Each of these requirements (i.e. R1 through R7) carries a full credit of 5 points when met. The program:

(R1) Uses Timer2 as an ipl-3 interrupt source, with a period register value of 9000 (in decimal).
(R2) Uses Timer2 ISR to increment a global integer variable count. count will be incremented by 1 each time this ISR is executed.
(R3) Uses Timer4 as an ipl-7 interrupt source with a period register value of 18000 (in decimal).
(R4) Uses Timer4 ISR and PORTD to display all illumination combinations of the 3 LEDs on the PIC32MX board each time this ISR is executed. One such illumination combination is when both the Red and Green LEDs are lit. Each of these illumination combinations must last (i.e. be delayed) for 256 ms. This ISR need not consider the situation when all the 3 LEDs are OFF. Hence, there should be 7 different LED illumination combinations. (Hint: The 3 LEDs are directly connected to PORTD RD0, RD1, and RD2 pins. RD0 controls the Red LED, RD1 the Orange LED, and RD2 the Green LED.)
(R5) Stops both Timer2 and Timer4 interrupts and turns all the LEDs OFF, after the “LED illumination combinations” action specified in (R4) has been performed five times. The program must use the Timer2 ISR, the global variable count, and appropriate code in main( ) to satisfy this requirement.
(R6) All the peripheral timers will use a 36-MHz bus clock rate, along with a prescaler value of 256.
(R7) Program code is written properly with indentation, and well documented in accordance with the structured programming style typically seen in high-level programming languages.

```
#include <p32xxxx.h>
#include <plib.h>
#define DELAY 36000
int count = 0;

void __ISR(_TIMER_2_VECTOR, ipl3) T2InterruptHandler(void)
{
    count++; // Increment count
    mT2ClearIntFlag(); // Clear T2 flag and exit
} // Timer2 ISR

void __ISR(_TIMER_4_VECTOR, ipl7) T4InterruptHandler(void)
{
    int i;
    for (i=1; i<=7; i++)
    {
        PORTD = i;
```
// 256ms delay - LEDs ON
TMR1 = 0;
while (TMR1 < DELAY) {};

mT4ClearIntFlag(); // Clear T4 flag and exit
} // Timer4 ISR

main()
{
    // Initialize PORTD
    TRISD = 0;    // PORTD as output

    // Configure Timers
    T1CON = 0x8030;   // TMR1 on, pre-scale 1:256 PB=36 MHz
    T2CON = 0x8070;   // TMR2 on, pre-scale 1:256 PB=36 MHz
    T4CON = 0x8070;   // TMR4 on, pre-scale 1:256 PB=36 MHz
    PR2 = 9000;    // TMR2 period
    PR4 = 18000;    // TMR4 period

    // For the ICM
    mT2SetIntPriority(3);  // Set T2 pr
    mT4SetIntPriority(7);
    INTEnableSystemMultiVectoredInt();
    mT2IntEnable(1);
    mT4IntEnable(1);
    while (count < 6) {};  // Main loop for T2/T4 interrupts
    mT2IntEnable(0);   // Stop T2 interrupts
    mT4IntEnable(0);   // Stop T4 interrupts

    // ISRs completed
    PORTD = 0;    // Done, turn all LEDs OFF
}