Jonathan W. Valvano First:_____ Last:_____

October 8, 2010, 2:00pm-2:50pm. This is a closed book exam. You have 50 minutes, so please allocate your time accordingly. ***Please read the entire quiz before starting***.

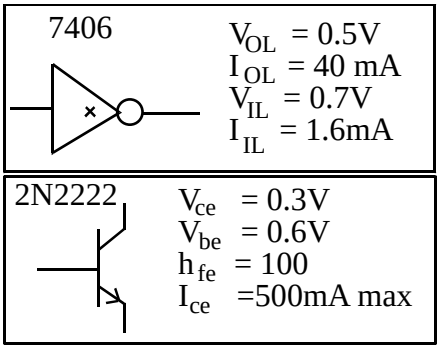
**PTT** is 8-bit bi-directional I/O port
**DDRT** is the associated direction register for Port T (0 means input, 1 means output)
**PTM** is 8-bit bi-directional I/O port
**DDRM** is the associated direction register for Port M (0 means input, 1 means output)
**PTP** is 8-bit bi-directional I/O port
**DDRP** is the associated direction register for Port P (0 means input, 1 means output)
**TSCR1** is the first 8-bit timer control register
bit 7 **TEN**, 1 allows the timer to function normally, 0 means disable timer including **TCNT**
**TSCR2** is the second 8-bit timer control register
bits 2,1,0 are **PR2**, **PR1**, **PR0**, which select the rate, let **n** be the 3-bit number formed by **PR2**, **PR1**, **PR0**
9S12DP512 without PLL **TCNT** is 8MHz/$2^{\mathbf{n}}$, with PLL **TCNT** is 24MHz/$2^{\mathbf{n}}$, **n** ranges from 0 to 7
**TCNT** is 16-bit up counter
**TIOS** is the 8-bit output compare select register, one bit for each channel (1 = output compare, 0 = input capture)
**TIE** is the 8-bit output compare arm register, one bit for each channel (1 = armed, 0 = disarmed)
**TC0 TC1 TC2… TC7** are the eight 16-bit output compare registers, one register for each channel
**TFLG1** is the 8-bit flag register, one bit for each channel:
With input capture, flags are set on the active edge of the input **TC0 TC1 TC2… TC7**
With output compare, flags are set when **TCNT** equals **TC0 TC1 TC2… TC7**
Flags become zero when software writes a 1 to it. E.g., **TFLG1=0x08;** clears channel 3 flag

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **TCTL1** | **OM7** | **OL7** | **OM6** | **OL6** | **OM5** | **OL5** | **OM4** | **OL4** |
| **TCTL2** | **OM3** | **OL3** | **OM2** | **OL2** | **OM1** | **OL1** | **OM0** | **OL0** |
| **TCTL3** | **EDG7B** | **EDG7A** | **EDG6B** | **EDG6A** | **EDG5B** | **EDG5A** | **EDG4B** | **EDG4A** |
| **TCTL4** | **EDG3B** | **EDG3A** | **EDG2B** | **EDG2A** | **EDG1B** | **EDG1A** | **EDG0B** | **EDG0A** |

If OMn=OLn=0 then an output compare event will not directly affect the output pin. If the pair (OMn,OLn) equals (0,1) then the output pin will toggle on each output compare. If the pair (OMn,OLn) equals (1,0) then the output pin will clear on each output compare. If the pair (OMn,OLn) equals (1,1) then the output pin will set on each output compare.

If EDGnB=EDGnA=0 then no input capture event will occur. If EDGnB=0 and EDGnA=1 then an input capture event will occur on the rising edge of the input. If EDGnB=1 and EDGnA=0 then an input capture event will occur on the falling edge of the input. If EDGnB=1 and EDGnA=1 then an input capture event will occur on both the rising and falling edges of the input.

```
0xFFD6    interrupt 20 SCI
0xFFDE    interrupt 16 timer overflow
0xFFE0    interrupt 15 timer channel 7
0xFFE2    interrupt 14 timer channel 6
0xFFE4    interrupt 13 timer channel 5
0xFFE6    interrupt 12 timer channel 4
0xFFE8    interrupt 11 timer channel 3
0xFFEA    interrupt 10 timer channel 2
0xFFEC    interrupt 9  timer channel 1
0xFFEE    interrupt 8  timer channel 0
0xFFF0    interrupt 7  real time interrupt
```




**9S12DP512 parameters**

$I_{OL}$ = 10mA, $I_{OH}$ = 10mA, $I_{IL}$ = 1μA, $I_{IH}$ = 1μA,
$V_{OL}$ = 0.8V, $V_{OH}$ = 4.2V, $V_{IL}$ = 1.75V, $V_{IH}$ = 3.25 V

**(10) Question 1.** `Buffer` is a shared permanently-allocated array containing 32 16-bit elements. **`N`** is a shared permanently-allocated 16-bit index, initialized to 0 during startup. The 9S12 is running at 8 MHz (E clock period is 125 ns). We added the following debugging instrument to an interrupt service routine to record the times of the last 32 interrupts. We expect the interrupt to occur about once every 1 ms:

```
Buffer[N] = TCNT; N = (N+1)&0x1F;
```

The compiler generated the following assembly code

```
0009 fc0000       [3]     LDD   N
000c 59           [1]     LSLD
000d de00         [3]     LDX   _TCNT
000f b746         [1]     TFR   D,Y
0011 6eea0000     [3]     STX   Buffer,Y
0015 f60000       [3]     LDAB  N+1   ;least significant byte of N
0018 52           [1]     INCB
0019 c41f         [1]     ANDB  #31
001b 87           [1]     CLRA
001c 7c0000       [3]     STD   N
```

Part a) Is this debugging monitor *nonintrusive*, *minimally intrusive*, or *highly intrusive*?

Part b) Justify your answer.

**(Bonus)** Give an equation that relates current through to the voltage across an ideal inductor, with inductance L.

**(15) Question 2.** You are given the following interrupt service routine and the following regular function. The **`RegularFunction()`** is called from the main program. Add minimally-intrusive debugging instruments to determine which occurs first: the first execution of the interrupt service routine or the first time the function is started. In particular, set **`bOC1`** if the first execution of the output compare ISR occurs before the function is started for the first time. Set **`bReg`**, if the function is started for the first time before the first interrupt. Interrupts will be enabled when the **`RegularFunction()`** is called from the main program. Be careful not to set both flags (i.e., do not introduce any critical sections).

```
short bOC1=0; // true if first ISR occurred first
short bReg=0; // true if first call to function occurred first
interrupt 9 void IC1handler(void){









  TFLG1 = 0x02;       // acknowledge OC1
  Stuff1();
}
void RegularFunction(void){









  Stuff2();
}
```

**(20) Question 3.** Draw a Moore FSM graph to solve the following problem. Your FSM graph must have less than 10 states and there should be NO wait parameter for this machine. There are two positive-logic switches as inputs, and there are two positive-logic outputs to LEDs. You may assume the switches do not bounce. The order in which the switches are touched is irrelevant. LED1 will turn on (and stay on forever) if Switch 1 goes from touched to released before Switch 2 goes from touched to released. LED2 will turn on (and stay on forever) if Switch 2 goes from touched to released before Switch 1 goes from touched to released. Only the first touch and release matters, subsequent touching and releasing will be ignored. If the first time the switches go from touched to released occurs such that both switches are released at the same time, turn on (and leave on) both LEDs. For each state give: the state name, the output, and four next states. The FSM controller sequence is output, input, change to next state. Just draw the FSM graph, no software is required. Full credit will be given to the machine with the fewest states that solves the problem. You may not include a time delay for each state.

**(25) Question 4.** You are given two tasks: the subroutine **`Task1()`** should be executed every 1 ms. It takes 10 to 100 μs to execute **`Task1`**, The subroutine **`Task2()`** should be executed every 2 ms. It takes 10 to 50 μs to execute **`Task2`**. The goal is to minimize jitter. This means **`Task1()`** is never delayed by the running of **`Task2`**. Similarly, **`Task2()`** is never delayed by the running of **`Task1`**. You should not activate the PLL. The 9S12DP512 E clock is 8 MHz. You may assume there are no other interrupts.
**Part a)** Show the ritual to initialize this system. You will use one or more output compare interrupts.

**Part b)** Show the interrupt service routine(s). No backward jumps are allowed.

**(10) Question 5.** Write C code to implement the following equation

**out = in*√½** (i.e., **out = in* 0.707106781186547524400844362104 85**)

where **`in`** comes from the ADC (0 to 1023). The variables **`in`** and **`out`** are defined as

```
unsigned short in,out;
```

Here are some possible approximations to consider

| n | m | n/m | \|error\| |
|---|---|---|---|
| 11 | 16 | 0.6875 | 0.0196 |
| 45 | 64 | 0.7031 | 0.0040 |
| 71 | 100 | 0.7100 | 0.0029 |
| 12 | 17 | 0.7059 | 0.0012 |
| 707 | 1000 | 0.7070 | 0.0001 |
| 181 | 256 | 0.7070 | 0.0001 |

You should minimize error, dropout, and eliminate overflow using only 16-bit arithmetic (no **`long float`** or **`double`** types are allowed). Write it entirely in C, without assembly code.

**(20) Question 6.** The NiMH battery cell voltage is 1.2V. Using multiple cells, we can create a power source at integer multiples of 1.2V. Interface a DC motor to the 9S12. To turn the motor on, the motor needs more than 5 V at 100 mA. Include protection against back EMF. Label part numbers for all interface components and resistor values. You can specify resistor values using an equation, rather than calculating the exact number. You must select which NiMH battery to use.