Jonathan W. Valvano First Name: _____ Last Name:_____

December 11, 2002, 9 am to 12 noon

This is an open book, open notes exam. You may put answers on the backs of the pages, but please don't turn in any extra sheets.

(15) Question 1. Design a low-noise analog circuit with the following specifications

differential input	(V_1-V_2), neither V_1,V_2 is ground
transfer function	$V_{out} = 500 \bullet (V_1-V_2)$
constrained input	$-0.01V \le (V_1-V_2) \le 0.01V$
constrained output	$-5V \le V_{out} \le +5V$
large input impedance	$Z_{in} \ge 1\ M\Omega$

Give chip numbers but not pin numbers. Specify all resistor values. You may use +12 and –12V analog supply voltages.

(15) Question 2. Consider the DS1230 32K by 8 bit static RAM. The timing diagrams from its data sheets are attached.

Part a) Develop an equation for **Read Data Available** using terms like AdV, AdN, $\downarrow \overline{CE}$, $\uparrow \overline{CE}$, $\downarrow \overline{WE}$, $\uparrow \overline{WE}$, $\downarrow \overline{OE}$, and $\uparrow \overline{OE}$. You will also have to use symbols from the timing diagram like t_{ACC}.

Part b) Assume the rise of $\overline{CE}$ occurs before the rise of $\overline{WE}$ during a write cycle access to this chip. Develop an equation for **Write Data Required** using terms like AdV, AdN, $\downarrow \overline{CE}$, $\uparrow \overline{CE}$, $\downarrow \overline{WE}$, $\uparrow \overline{WE}$, $\downarrow \overline{OE}$, and $\uparrow \overline{OE}$. You will also have to use symbols from the timing diagram like t_{WC}.

(55) Question 3. The objective of this problem is to implement a PI motor controller. There are three starter programs taken directly from the book. Your job will be to edit them into a system that maintains the motor speed at a constant 500 rpm. The first component of the control system is the state estimator. A tachometer is used to estimate the rotational speed of the motor shaft. Assume the tachometer produces a square wave that is connected to PT0. You will directly measure period of this input with a range of 200 to 65535 μsec and a resolution of 1 μsec. If `Period` is the period in μsec, then the estimated speed in rpm can be calculated as `200000/Period`.

```
unsigned short Speed;  // motor speed in rpm
```

You may assume the `Speed` varies from 0 to 1000 rpm, so the `Period` will always be greater than 200 μsec. There are two problems to consider. First, if the motor is stopped there will be no tachometer input (no input capture interrupts), because the period will be infinite. If the motor is spinning very slowly, the period may be larger than the 65535 μsec upper limit on the period measurement. These two problems will be solved with a TOF interrupt.

(10) Part a) When the `Speed` is about 500 rpm, what is the speed resolution of the state estimator?

(10) Part b) The following two programs are taken directly from the book. You will modify the input capture system to measure period on input PT0, and change the resolution to 1 μsec. Calculate `Speed` in rpm after each period measurement. Add code to the TOF interrupt handler so that the `Speed` is set to 0 if you get two consecutive TOF interrupts without any IC interrupts.

```
// Program 6.6, pages 301-302
unsigned short Period;  // units of 500 ns
unsigned short First;   // TCNT first edge
void IC_Ritual(void){
  asm(" sei");    // make atomic
  TIOS &= 0xFD;   // PT1 input capture
  DDRT &= 0xFD;   // PT1 is input
  TSCR = 0x80;    // enable TCNT
  TMSK2= 0x32;    // 500ns clock
  TCTL4 = (TCTL4&0xF3)|0x04; // rising
  First = TCNT;   // first will be wrong
  TFLG1 = 0x02;   // Clear C1F
  TMSK1 |= 0x02;  // Arm IC1
  asm(" cli");}
#pragma interrupt_handler IChandler()
void IChandler(void){
  Period = TC1-First;
  First = TC1;   // Setup for next
  TFLG1 = 0x02;  // ack by clearing C1F


}
```

```
#pragma abs_address:0xffec
void (*ICvector[])() = {IChandler};
#pragma end_abs_address

// Program 4.51, page 259
unsigned short Time;
void TOF_Ritual(void){
  asm(" sei");      // Make ritual atomic
  TMSK2 = 0xB2 ;    // Arm, Set PR to 010, 30.517Hz
  TSCR = 0x80;      // enable counter
  Time = 0;         // Initialize global data structures
  asm(" cli");
}


#pragma interrupt_handler TOFHan()
void TOFHan(void){
  TFLG2 = 0x80;       // Acknowledge by clearing TOF
  Time++;



}
#pragma abs_address:0xffde
void (*TOFvector[])() = {TOFHan};
#pragma end_abs_address
```

(10) Part c) The DC motor must be interfaced to PT1. The motor voltage is +12 V and its current is 0.5 A. Show the hardware interface circuit (do not isolate it using an optocoupler.)

(10) Part d) Pulse width modulation will be used to adjust the power delivered to the motor. Modify the following program so that the output is on PT1 and the basic period is 50 ms (20Hz). `High+Low` will be fixed at 50000 by the control software. Change this ritual so that it does not conflict with any of the other rituals in this question.

```
// Program 6.27 page 331
unsigned short High; // Num of Cycles High
unsigned short Low;  // Num of Cycles Low
// Period is High+Low Cycles
void OC_Ritual(void){
  asm(" sei");    // make atomic
  TIOS |= 0x08;   // enable OC3
  DDRT |= 0x08;   // PT3 is output
  TSCR = 0x80;    // enable
  TMSK2 = 0x32;   // 500 ns clock
  TMSK1 |= 0x08;  // Arm output compare 3
  TFLG1 = 0x08;   // Initially clear C3F
  TCTL2 |= 0xC0;  // PT3 set on next int
  TC3 = TCNT+50;  // first right away
  asm(" cli");
}
#pragma interrupt_handler OChandler()
void OChandler(void){
//**the controller software in part e) will be called from here*****
  TFLG1 = 0x08;       // ack OC3F
  if(TCTL2&0x40){     // PT3 is now high
    TC3 = TC3+High;   // 1 for High cyc
    TCTL2 &= 0xBF;    // clear on next
  }
  else {              // PT3 is now low
    TC3 = TC3+Low;    // 0 for Low cycles
    TCTL2 |= 0x40;    // set on next int
  }
}
#pragma abs_address:0xffe8
void (*OCvector[])() = {OChandler};
#pragma end_abs_address
```

(15) Part e) The digital controller is executed every $\Delta t = 0.05$ s. Show the control software to be executed in the above output compare interrupt handler at the place specified. You must deal with overflow/underflow. You may define additional variables as you need them.

$e = 500 - Speed$

proportional term

$Up = 134.5 \bullet e$

integral term

$Ui = Ui + 7890 \bullet e \bullet \Delta t$

put together

`High` = Up + Ui

`Low` = 50000-`High` (maintain High+Low always equal to 50000)

Use **binary** fixed-point math. Add anti-reset windup. Limit `High` and `Low` to values between 200 and 49800.

(15) Question 4. Consider a problem of running two foreground threads (`producer` and `consumer`) using a preemptive scheduler with semaphore synchronization (like Lab 17.) There is a shared fifo data structure. The `producer` thread will create data and call `PutFifo`. The `consumer` thread will call `GetFifo` and process the data. The basic fifo is Program 4.22 found on pages 212-3 of the book. Define one or more semaphores, then add calls to the following three functions in order to properly synchronize the interactions between the `producer` and `consumer`. You do not have to implement the semaphore functions, just call them.

```
int OS_InitSemaphore(Sema4Type *semaPt, short value);
void OS_Wait(Sema4Type *semaPt);
void OS_Signal(Sema4Type *semaPt);
```

You will draw a line through code no longer needed (delete), and add calls to the semaphore functions, otherwise no other changes are allowed. For each semaphore you add, explain what it means to be 0, 1 etc. Assume `InitFifo` is run first.

```
/* Index,counter implementation of the FIFO */
#define FifoSize 10    /* Number of 8 bit data in the Fifo */
unsigned char PutI;    /* Index of where to put next */
unsigned char GetI;    /* Index of where to get next */
unsigned char Size;    /* Number of elements currently in the FIFO */
/* FIFO is empty if Size=0     FIFO is full if Size=FifoSize */
char Fifo[FifoSize];      /* The statically allocated fifo data */
void InitFifo(void){char SaveSP;

  asm(" tpa\n staa %SaveSP\n sei");   /* make atomic, entering critical*/

  PutI=GetI=Size=0;                   /* Empty when Size==0 */

  asm(" ldaa %SaveSP\n tap");         /* end critical section */

}
```

```
int PutFifo(char data){ char SaveSP;

  if (Size == FifoSize )

    return(0);    // Failed, was full

  else{

    asm(" tpa\n staa %SaveSP\n sei");

    Size++;

    Fifo[PutI++]=data;  //  put data

    if (PutI == FifoSize) PutI = 0;

    asm(" ldaa %SaveSP\n tap");

    return(-1);    /* Successful */

  }

}
```

```
int GetFifo(char *datapt){char SaveSP;

  if (Size == 0 )

    return(0);     // Empty if Size=0

  else{

    asm(" tpa\n staa %SaveSP\n sei");

    *datapt=Fifo[GetI++];

    Size--;

    if (GetI == FifoSize) GetI = 0;

    asm(" ldaa %SaveSP\n tap");

    return(-1);

  }

}
```

Extra Credit (bonus points)
Throughout EE345L and EE345M, I have espoused the need to know assembly language to understand how the software works and to optimize performance for time-critical tasks.

(1) Bonus 1. Which assembly language instruction allows you to write a spinlock binary semaphore without disabling interrupts? Don't write the program, just state the assembly op code.

(1) Bonus 2. Which assembly language instruction does the C compiler use to implement the following minimally intrusive, noncritical, and friendly debugging instrument?

```
PORTT &= ~0x40;   // turn off LED
```

(1) Bonus 3. Which two assembly language instructions allow you to quickly execute the defuzzification step in a fuzzy logic controller? Don't write any code, just state the two assembly op codes.

(1) Bonus 4. The C code, shown below, properly executes the desired operation, but runs too slow:

```
out=0.902*xx-1.81*yy+0.045*zz
```

If you were asked to optimize this C function, which two assembly language instructions perform these basic calculations in a very efficient manner? Don't write the assembly program, just state the two assembly op codes.

```
short calculate(short xx, short yy, short zz){ short out;
  out = (short)((902*(long)xx-1810*(long)yy+45*(long)zz)/1000);
  return out;
}
```

(1) Bonus 5. Which assembly language instruction allows you to implement linear interpolation in a very efficient manner? Don't write any code, just state the assembly op code.

(1) Bonus 6. Assume the software is currently executing a background thread. Which assembly language instruction is used to switch the control back to the foreground? Don't write the program, just state the assembly op code.