Jonathan W. Valvano

October 10, 2008, 2:00pm-2:50pm.

(5) Question 1. I am looking to see if there is a correlation between EE319K instructor and EE345L grades. So far, the two seem uncorrelated (which of course is good).

(10) Question 2. These read-modify-write sequences do not constitute a critical section because their execution is atomic. In particular, both ISRs run with interrupts disabled.

(10) Question 3. Output high turns on the LED. V_{OH} is the output high voltage of the 9S12
Voltage across resistor is V_{OH} - V_D Current through resistor is (V_{OH} - V_D)/R
Desired LED circuit is I_D, set desired to equal actual. I_D = (V_{OH} - V_D)/R. Solve R = (V_{OH} - V_D)/ I_D

(15) Question 4. For each number, we define **x4 = I4**/16, **x3 = I3**/16, **x2 = I2**/16, and **x1 = I1**/16. We start with desired function

x4 = x1*x2 + x3

then, we plug in each definition.

(I4/16) = (I1/16)*(I2/16) + (I3/16)

Solve for **I4** and simplify. We want to divide last to reduce effect of dropout. The following code is algebraically correct and properly minimizes the error due to dropout.

```
I4= (I1*I2)/16 + I3;
```

However, as you can see from the machine code produced by Metrowerks, it does have a potential overflow error, because the divide by 16 is a 16-bit divide, rather than a 32-bit divide.

```
0000 fc0000       [3]     LDD   I1
0003 fd0000       [3]     LDY   I2
0006 13           [3]     EMUL ;RegY:D is 32-bit product
0007 49           [1]     LSRD ;***neglects most significant bits
0008 49           [1]     LSRD
0009 49           [1]     LSRD
000a 49           [1]     LSRD
000b f30000       [3]     ADDD  I3
000e 7c0000       [3]     STD   I4
```

To handle overflow, you need to promote the multiply to 32 bits, and perform the /16 in 32-bits. I define a 32-bit temporary variable called `Product` to perform the 16 by 16 into 32 bit multiply.

```
Product = (unsigned long)I1*(unsigned long)I2;
I4= (unsigned short)(Product/16) + I3;
```

Notice now the product and /16 are 32 bits

```
0000 fc0000       [3]     LDD   I1
0003 fd0000       [3]     LDY   I2
0006 13           [3]     EMUL
0007 7c0000       [3]     STD   Product:2
000a 7d0000       [3]     STY   Product
000d c604         [1]     LDAB  #4
000f b765         [1]     TFR   Y,X
0011 fd0000       [3]     LDY   Product:2
0014 160000       [4]     JSR   _LSHRU ;(RegXY)>>4
0017 f30000       [3]     ADDD  I3
001a 7c0000       [3]     STD   I4
```

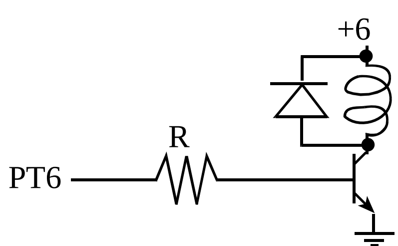
Hand execute your code with **x1**=1.5, **x2**=2, **x3**=0.25 to verify it calculates **x4**=3.25.

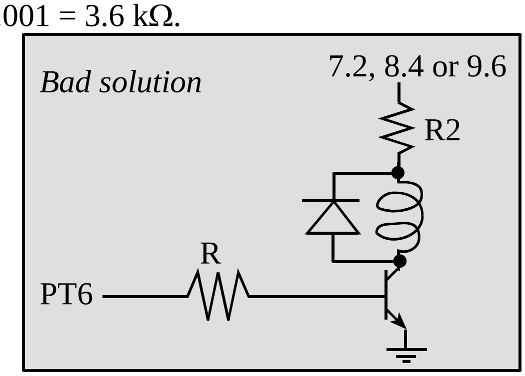
```
I1 = 16*1.5 = 24    1.50 = 24/16
I2 = 16*2 = 32      2.00 = 32/16
I3 = 16*0.25 = 4    0.25 =  4/16
I4 = (24*32)/16+4 = 48+4 = 52 (3.25 = 52/16)
```

(5) Question 5. Notice the output is low for 200, and high for 100
state is A 1) wait 100; 2) input=0; 3) output =0; then 4) set next state =B
state is B 1) wait 200; 2) input=0; 3) output =1; then 4) set next state =A
state is A 1) wait 100; 2) input=0; 3) output =0; then 4) set next state =B
state is B 1) wait 200; 2) input=0; 3) output =1; then 4) set next state =A
F) The system oscillates between state A and state B with the output toggling high and low, with the output being low for a longer time than the output is high.

(10) Question 6.
Part a) **`v1`** is allocated in C) EEPROM
Part b) **`v2`** is allocated in A) Global
Part c) **`v3`** is allocated on the B) Stack
Part d) **`v4`** is allocated in A) Global

(10) Question 7. 100 mA will require the 2N2222 (because it can handle up to 500 mA of I_{CE}). We could have used any NPN with I_{CE} > 100mA, e.g., TIP120, IRF540. The V_{CE} on voltage of the 2N2222 is 0.3V. Because the current gain is 100 (h_{fe}) the base current needs to be 100mA/100 = 1mA. The I_{OH} of the 9S12 can supply this 1mA (I_{OH} can be up to 10 mA). Because the V_{OH} of the 9S12 is 4.2V (or greater) and the V_{BE} if the 2N2222 is 0.6V (or less), the resistor from the 9S12 to the 2N2222 base must be less than (4.2-0.6V)/1mA = 3.6/0.001 = 3.6 kΩ.

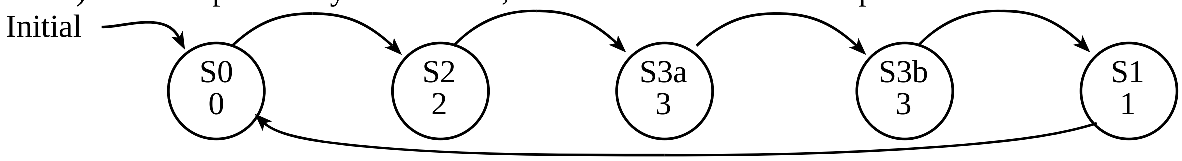




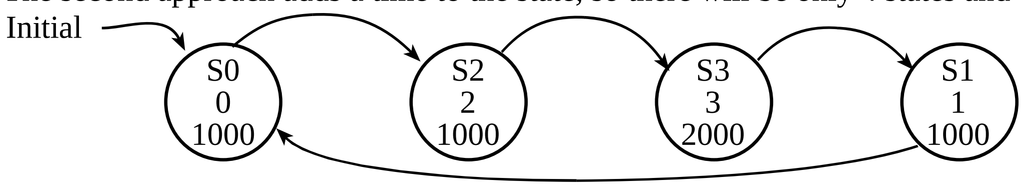
I suggest making R much less than 3.6 kΩ (e.g., 1 kΩ) because it will force the NPN into saturation, independent of the V_{OH} of the 9S12, the V_{BE} of the 2N2222, the h_{fe} of the 2N2222, and the resistance of the coil. Therefore, when the digital output is high, the voltage across the relay will be 5.7V. You might have been tempted to use a higher voltage supply, like the "Bad solution" and use a series resistor (R2) to drop the voltage down to 6V. There are two fundamental problems with the "Bad solution". First, the solution wastes power, the power delivered into R2 is lost as heat. Second, the resistance of the coil is a function of the mechanical load on the electromagnet. The coil resistance can not be assumed to be constant.

(35) Question 8. Spin a 2-phase synchronous motor (I fabricated this problem).

Part a) The first possibility has no time, but has two states with output = 3.



The second approach adds a time to the state, so there will be only 4 states and one less interrupt.



Part b) Definition of the structure.
```
const struct State{
  unsigned char Output;     // 0,2,3,1 sequence
  unsigned short Time;      // delay in usec
  const struct State *Next; // no inputs, one next state
};
typedef const struct State StateType;
typedef StateType *StatePtr;
```
Part c) Definition of the FSM.
```
#define S0 &FSM[0]
#define S2 &FSM[1]
#define S3 &FSM[2]
#define S1 &FSM[3]
StateType FSM[4]={
  {0,1000,S2},  // S0
  {2,1000,S3},  // S2
  {3,2000,S1},  // S3
  {1,1000,S0}}; // S1
```
Part d) The main program
```
StatePtr Pt;     // pointer to current state
void main(void){
  TSCR1 = 0x80;  // Enable TCNT 4 MHz in run mode
  TSCR2 = 0x02;  // divide by 4 TCNT prescale, 1us (0x03 for 9S12DP512)
  TIOS |= 0x80;  // activate TC1 as output compare
  TIE |= 0x80;   // arm OC1
  DDRT |= 0x03;  // PT1,PT0 outputs
  Pt = S0;                 // first state
  PTT = Pt->Output;        // perform output for first state
  TC7 = TC7+Pt->Time;      // time for first state
  Pt = Pt->Next;           // second state
  asm cli
  for(;;);
}
```
Part e) The output compare interrupt 7 service routine that outputs to PT1, and PT0.
```
interrupt 15 void TC7handler(void){
  TFLG1 = 0x80;            // acknowledge OC7
  PTT = Pt->Output;        // perform output for this state
  TC7 = TC7+Pt->Time;      // time for this state
  Pt = Pt->Next;           // next state
}
```