Embedded Real-Time Systems (AME 3623)
Homework 4 Solutions

April 30, 2008

Question 1

1. (10pts) Briefly explain why *polling* can be undesirable when performing input/output operations.

   Polling is undesirable because the processor can potentially spend a long time waiting for something to happen (we refer to this as busy waiting*). This time could otherwise be used to do other things.*

   If we try to be clever about switching between polling and doing other things, we run the risk of not responding to input quick enough.

2. (10pts) Define an *Interrupt Service Routine*.

   An Interrupt Service Routine is a small piece of code that is executed in response to an interrupt (an event).

Question 2

1. (15pts) Suppose we want a function – called $donow()$ – to be executed once every $21s$. What is the timer1 prescaler configuration and the (pseudo)code for the interrupt routine (the code does not need to be syntactically correct)?

   We will use a prescaler of 1024. This gets us down to an interrupt every 4.1943 s. We then need an interrupt routine with an additional counter that expires at 5. So, we are left with an interrupt interval of: $5 * 1024 * 256 * 256/16000000 = 20.9715s$.

```
ISR(TIMER1_OVF_vect) {
    static uint8_t counter = 0;

    ++counter;
    if(counter == 5) {
     donow();
     counter = 0;
   };
};

Somewhere in the main program:


// Interrupt occurs every
//     (1024*256*256)/16000000 = 4.1943 sec
timer1_config(TIMER1_PRE_1024);
// Enable the timer interrupt
timer1_enable();
// Enable global interrupts
sei();
```

Consider the following code.

```
volatile uint8_t duration;

ISR(TIMER0_OVF_vect) {
    static uint8_t counter = 0;

    ++counter;
    if(counter >= duration) {
     donow();
     counter = 0;
   };
};

Somewhere in the main program:

// Interrupt occurs every
//     (64*256)/16000000 = 1.024 ms
timer0_config(TIMER0_PRE_64);
// Enable the timer interrupt
timer0_enable();
// Enable global interrupts
sei();

while(1)
{
    <change the value of duration>
}
```

2. (5 pts) What does the ISR do?

   It calls **donow()** *at a period that is specified by the variable* **duration**.

3. (5 pts) What does the main program do (in the while() loop)?

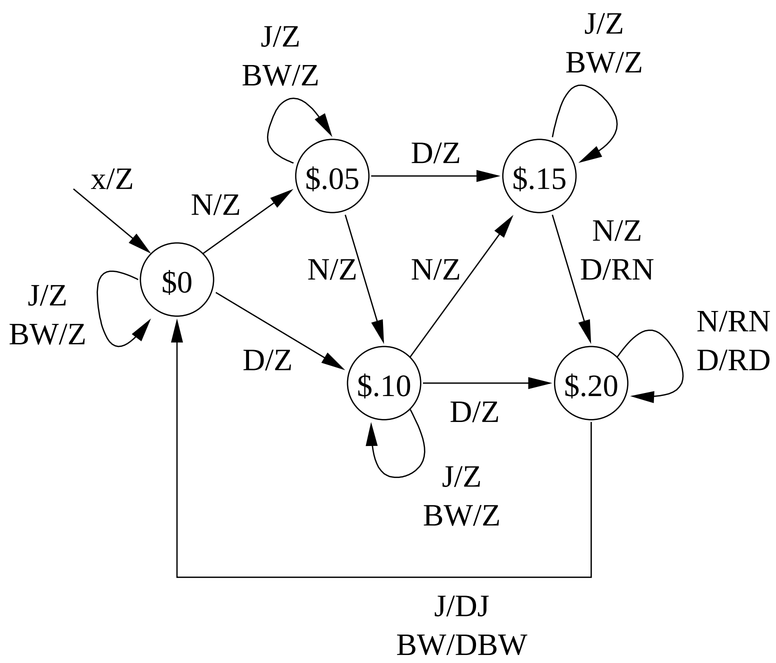
   It is responsible for setting the frequency of the donow() call.

4. (5 pts) Does a shared data problem exist between the ISR and the main program?

   No. The shared data consists of a single byte. So - all accesses to this byte are done atomically. Since the ISR only reads from this variable, the main program is free to do what it wants with it (either reading or writing).

Question 3

(20pts) Below is the FSM for the vending machine that we discussed in class.



Alter this vending machine such that if three nickels are inserted (starting from state $0), then a dime is returned immediately and the vending machine still maintains a state of \$.15.

We have to introduce a second state with a value of 10 cents (we call it $.10∗*). Starting from the initial state, this new state distinguishes between inserting a dime from inserting two nickels, and sets us up to exhibit two different behaviors when a nickel is then inserted.*