(2 pts) Name:

AME 3623: Embedded Real-Time Systems: Final Exam

Solution Set

May 10, 2006

Problem	Topic	Max	Grade
0	Name	2	
1	Interrupts and I/O	65	
2	Finite State Machines I	20	
3	Finite State Machines II	25	
4	Analog Processing	25	
5	Microprocessor Design	35	
6	Logic	30	
Total		202	

1. **Interrupts and I/O** (65 pts)

   (a) (15 pts) Below is an interrupt service routine that is supposed to produce a signal of some frequency on port D, pin 2 (counting from 0). However, there exist several bugs (errors). Make the necessary changes to this code to remove these bugs. The "half period" is specified by the variable **duration** (i.e., the signal changes state every **duration** counts). You may assume that the I/O hardware has been initialized correctly.

```
volatile uint16_t counter;
volatile uint16_t duration;   // Half period

ISR(TIMER0_OVF_vect) {

  ++counter;

  if(counter == duration) {

     PORTD |= 0x10;

  }
}
```

The repaired code is as follows:

```
volatile uint16_t counter;
volatile uint16_t duration;   // Half period

ISR(TIMER0_OVF_vect) {

  ++counter;

  if(counter >= duration) {   // MINOR SYNCHRONIZATION BUG (NO PENALTY
                              //    FOR THIS ONE)
     PORTD ^= 0x4;            // FIXED TWO THINGS
     counter = 0;             // ADDED
  }
}
```

Key changes: use XOR instead of OR; bit 2 (not bit 4); reset counter.

(b) (5 pts) Are there any *shared data problems* with the above code (in association with a main program)? Explain.

Shared data problems occur when two segments of code access a common data structure (specifically, one has to be making changes to it) and when the execution of one segment can be interrupted by the execution of the other. In this case - **yes**. *The* **duration** *variable would be modified by the main program. Unless appropriate steps are taken by the main program, it could be interrupted in the middle of modifying this variable (resulting in a corrupted value from the perspective of the ISR).*

(c) (10 pts) Assume that the code above has been corrected. Given a prescalar of 256 and *duration* = 20, what is the resulting signal frequency? (set up the fraction, but do not reduce it)

$$
\begin{aligned}
freq &= \frac{16000000}{256 * 256 * 20 * 2} \\
&= 6.1035 Hz
\end{aligned}
$$

(d) (5 pts) Briefly define *serial communication.*

At its minimum, serial communication involves a single line between two devices that can carry one bit of information at any one time. Transfer of multiple bits requires that each bit be written to the line for some period of time (hence, the first bit is written, then the next, etc.).

(e) (5 pts) What is the function of a *start bit* in serial communication (be specific)?

A start bit is used in asynchronous serial communication. Here, each device has its own internal clock that tells it which bit is being transferred at any instant. The start bit is transmitted by the sender to indicate that some data is about to be sent; this allows the two devices to synchronize their clocks for the duration of this transfer.

(f) (5 pts) Briefly define *polling.*

Polling is the process of a program continually checking the state of some device for a particular event (e.g., the change in state of an input line or the reception of a character in a buffer).

(g) (5 pts) What is one disadvantage of polling?

Time that could otherwise have been spent doing other things is wasted through the continual checking. In some implementations, we might not be able to do anything until the event is detected (potentially missing other events).

(h) (10 pts) Define a *buffer.* Briefly explain why it is useful.

A buffer is a temporary storage location for data (this data can be either incoming or outgoing).

In the case of a program receiving data, a buffer is useful because the program does not have to process the data as soon as it arrives. Instead, the program can finish what it is doing before addressing the incoming data (and these data are not lost in the mean time).

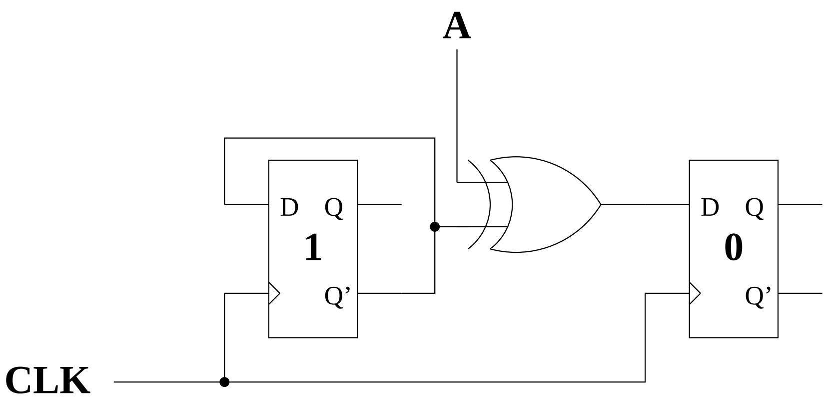
(i) (5 pts) Briefly explain why it is a problem to wait inside of an ISR for a digital I/O pin to change state.

Because ISRs interrupt other programs (and, in the case of the mega8, prevent the execution of other ISRs), it is important for an ISR to be as short as possible (otherwise, it could interfere with other timing activities).

In this case, the ISR is **busy waiting** *on this input line. The amount of time that is spent inside of this ISR is determined by this external input (which could be an arbitrary amount of time).*

2. **Finite State Machines I** (20 pts)

Consider the following sequential logic circuit:



Answer the following questions in the context of describing this circuit in terms of a finite state machine.

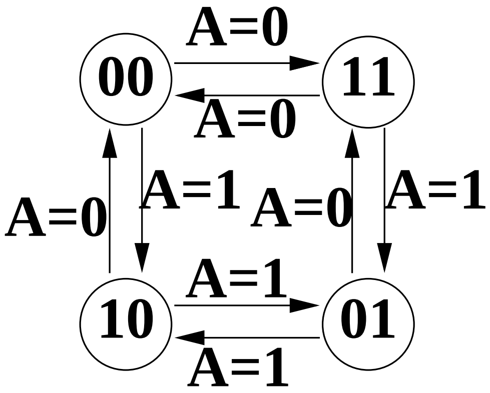
(a) (5 pts) What are the states?

00, 01, 10, 11

(b) (5 pts) What are the events?

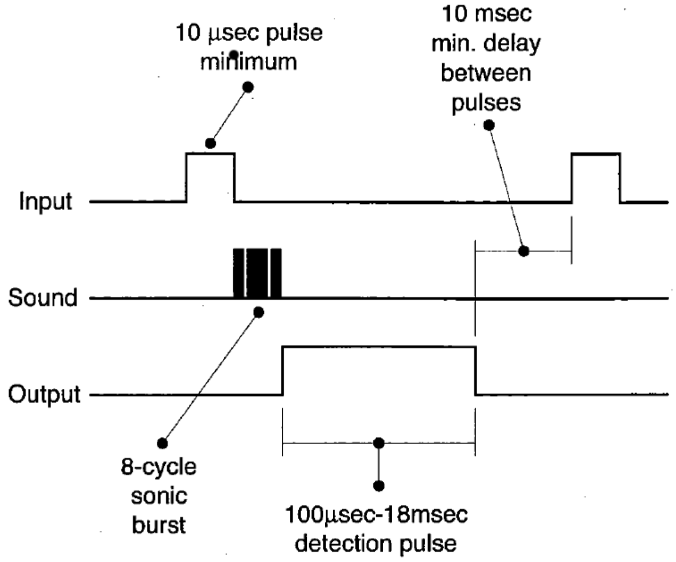
- $A = 0$ *(and clock goes from high to low)*
- $A = 1$ *(and clock goes from high to low)*

(c) (10 pts) Draw the finite state machine diagram. (hint: for each state and event, you must show what the next state is)



3. **Finite State Machines II** (25 pts)

Consider the specification for the behavior of the sonar unit that we used for projects 3 and 4. Over the next several questions, we are going to develop a finite state machine that is responsible for controlling the sensor and for processing the sensed distance. You may use a counter, but it **should not** be considered part of the state for our purposes (instead, consider it an input). You can assume that the FSM is implemented using an ISR that is called once every $16\mu sec$ and that the counter increments on each execution of the ISR. Also, you can define reasonably named constants without giving explicit values for them (e.g., you don't have to compute the number of counter steps that make up $5ms$).



(a) (5 pts) What are the states? (hint: recall that the FSM is in one state at any given time. You should be able to show on the above timing diagram what state the FSM is in at any given moment)

- *Command (generating the trigger signal)*
- *Wait-for-pulse*
- *Pulse*
- *Pause*

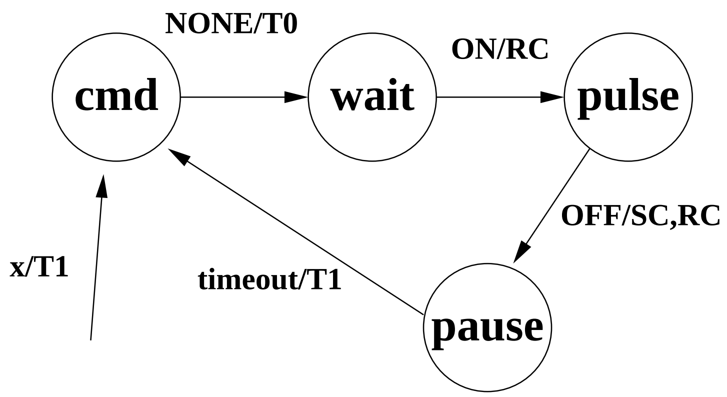
(b) (5 pts) What are the events? (in addition to **NONE**)

- *Pulse on (ON)*
- *Pulse off (OFF)*
- *counter == pause_length (timeout)*
- *Nothing (NONE)*

(c) (5 pts) What are the actions?

- *Trigger on (T1)*
- *Trigger off (T0)*
- *Reset counter (RC)*
- *Save counter (SC) (in global_duration)*

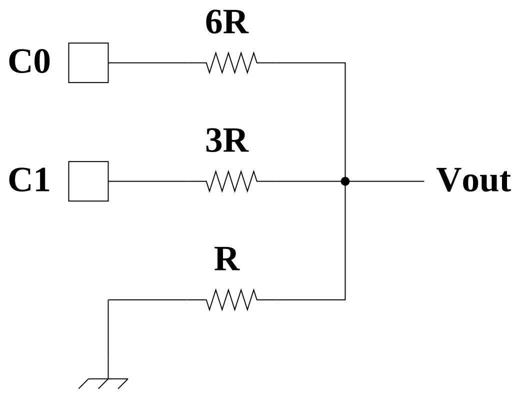
(d) (10 pts) Show the FSM.



In all states (other than **cmd***), if the event is* **NONE** *the FSM remains in the same state. Other events will not occur (so they are not shown)*

4. **Analog Processing** (25 pts)

Consider the following digital-to-analog conversion circuit:



(a) (15 pts) Give the expression for V_{out} in terms of binary digits $C0$ and $C1$. Show your derivation

Assume: current flows from left to right.
The fundamental equations (given by Ohm's law and Kirkoff's current law):

$$\begin{aligned} 5C_0 - V_{out} &= 6RI_0 \\ 5C_1 - V_{out} &= 3RI_1 \\ 0 - V_{out} &= RI \\ I + I_0 + I_1 &= 0 \end{aligned}$$

Therefore:

$$\begin{aligned} -\frac{V_{out}}{R} + \frac{5C_0 - V_{out}}{6R} + \frac{5C_1 - V_{out}}{3R} &= 0 \\ -6V_{out} + 5C_0 - V_{out} + 10C_1 - 2V_{out} &= 0 \\ 5C_0 + 10C_1 &= 9V_{out} \\ \frac{5}{9}(C_0 + 2C_1) &= V_{out} \end{aligned} \tag{1}$$

(b) (10 pts) Given that a program "wants" to generate a voltage $V_{out} = 1V$. What choice of binary values would best approximate this value?

$C[1,0] = 10$ would yield a voltage of $10/9V$ (which is within $5/18V$ of $1V$)

5. **Microprocessor Design** (35 pts)

(a) (10 pts) List two special purpose registers and briefly describe the function of each.

The **program counter** *contains the address of the memory location that contains the instruction that is currently being executed.*

The **instruction register** *contains the value of the instruction that is currently being decoded and executed.*

The **status register** *contains a memory of recent operations (e.g., zero result, carry, etc.).*

The **stack pointer** *stores the address to the memory location that is at the top of the stack.*

The **DDRx** *register (in the Mega8) sets the direction of a digital I/O port.*

The **PORTx** *register (in the Mega8) determines the values that are produced on the corresponding output lines.*

The **PINx** *register (in the Mega8) contains the value of the corresponding input lines.*

(b) (5 pts) During a read operation from a memory chip, which microprocessor component is responsible for driving the data bus?

The memory chip.

(c) (5 pts) (True/False) In the Atmel Mega8, special purpose registers provide the values to the Arithmetic Logical Unit (e.g., the values to be added together). Explain.

False. The general purpose registers provide these values (called **operands***).*

(d) (5 pts) Briefly state the function of an *instruction decoder.*

The instruction decoder is responsible for translating the binary representation of the currently executing instruction into the control signals that determine how the processor will respond (e.g., memory addresses and other control signals, ALU control signals, etc.).

(e) (10 pts) What is the value of variable *baz* at the end of this segment of code (in hexadecimal)?

```
foo = 0x40;

bar = 0xF3;

baz = foo & bar;

baz = baz ^ 0xA5;  // XOR
```

The values are shown below:

```
baz = foo & bar;
    = 0x40

baz = baz ^ 0xA5;
    = 0xE5
```

6. **Logic** (30 pts)

Given the following truth table:

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

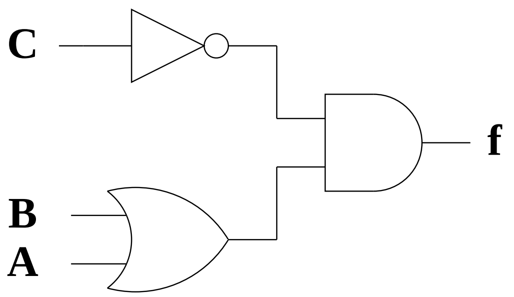
(a) (5 pts) Give the "minterm" form of the corresponding algebraic expression.

$f = \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

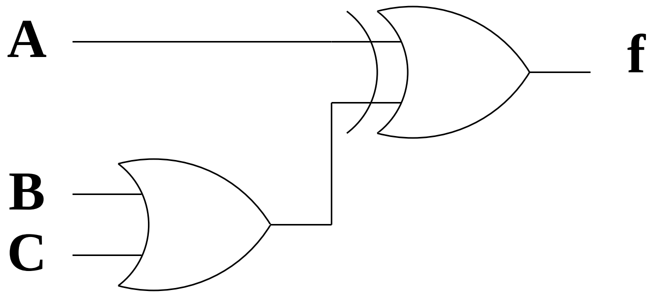
(b) (10 pts) Derive a simplified algebraic description for f. Justify each step (provide the name of the rule that you are using).

$\bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$	
$(\bar{A}B + A\bar{B} + AB)\bar{C}$	*Distributive (and associative)*
$(A + B)\bar{C}$	*Definition of OR*

(c) (5 pts) Draw the corresponding circuit.



Given the following circuit:



(d) (10 pts) What is the truth table?

A	*B*	*C*	*f*
0	*0*	*0*	*0*
0	*0*	*1*	*1*
0	*1*	*0*	*1*
0	*1*	*1*	*1*
1	*0*	*0*	*1*
1	*0*	*1*	*0*
1	*1*	*0*	*0*
1	*1*	*1*	*0*