CSE 466 Exam I 9 February 2007

1. (30 points)

Define the following terms related to features of your Atmega16 microcontroller and provide an example of when each feature would be used. Make SURE to relate your examples to the projects you worked on for Labs 3 and 4, if at all possible. If not, explain why not.

a) A-to-D conversion

Analog to digital conversion is the process of converting a continuous signal into a discretely quantized one. The ATmega microcontrollers have built-in A-to-D converters that can be triggered explicitly or on a periodic basis and generate an interrupt when they have completed the conversion. In the labs, this was used to convert the potentiometer value for the tri-color LED brightness.

b) Output compare

Timers can be set to trigger an interrupt when they reach a pre-specified value. Output compare adds the ability to automatically change the value of an output pin without having to run interrupt handler code. This was useful in the labs in generating the PWM signals for the tri-color LED.

c) Memory-mapped I/O registers

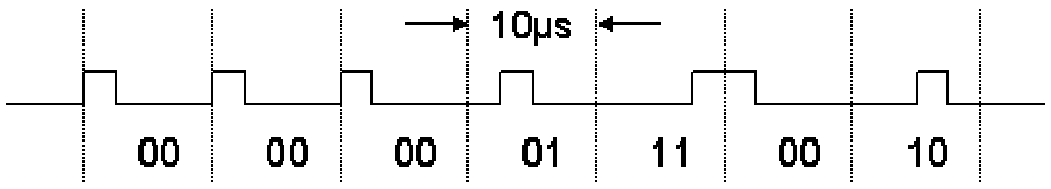
All the I/O subsystems in the ATmega microcontroller are controlled through memory-mapped I/O registers. This allows us to use the well-defined memory abstraction to sense and control what is going on in all of these parallel units. They were used extensively throughout the labs whenever a timer, A-to-D converter, or GPIO pin was used.

d) Interrupt service routine dispatching

Interrupt service routing dispatching is the process of finding the appropriate ISR code to run for a particular interrupt. In the ATmega microcontrollers, there is a “jump table” at the beginning of program memory that holds the first instruction of every possible ISR (usually this is a long jump instruction to the real start of the variable length ISR). This was used in the labs for all interrupt handling routines.

2. (70 points)

Quadrature phase is a common method for encoding information. It is used in the most common infrared communication protocols (e.g., IrDA). Below is an example of a 14-bit packet of encoded data. The basic idea is that two bits of information are encoded in every cycle (between adjacent dashed lines). If the pulse is in the first fourth of the cycle then the value is 00, if it is in the second quarter it is 01, 10 for the third quarter, and 11 if the pulse is in the final fourth of the cycle. Assume that a cycle is 10 microseconds.



Part I (short answers – 20 points)

(a) What is the shortest possible duration for the signal being high between lows? __*2.5μs*___

"01" or "10"

(b) What is the longest possible duration for the signal being high between lows? __*5.0μs*___

"11" followed by "00"

(c) What is the shortest possible duration for the signal to be low between highs? __*2.5μs*___

"10" followed by "00"

(d) What is the longest possible duration for the signal to be low between highs? _*15.0μs*___

"00" followed by "11"

(e) What is the maximum number of bits per second can be transmitted this way? _*200Kb/sec*

*2 bits/10.0μs = 2 bits/10*10^{-6} sec = 2 bits/10^{-5} sec = 2*10^{5} bits/sec = 200Kb/sec*

Part II (decoding – 35 points)

Write pseudo-code you would use to decode this signal. It should be possible to do entirely in interrupt service routines. Do not be concerned with syntax. In fact, you can use English (e.g., "read timer0 value"). However, you must label interrupt routines so that it is clear which condition will cause them to be executed. Make sure to describe the configuration of all devices and I/O capabilities you would use (you should only use a single timer).

<u>Option I:</u>

General idea: *Use the first bits (a "00") to synchronize a timer to go off every 2.5µs. That will generate four samples of the signal per 10µs period. If the signal is high for the first sample, then it is a "00"; second sample, then it is "01"; etc.*

Initialization:
Set to interrupt on rising edge of input signal

Interrupt service routine for rising edge:
Set timer to periodically interrupt in 1.25µs to sample value in the middle of quadrature pulse
Set quadrant counter to 0
Disable interrupt on rising edge

Interrupt service routine for timer:

```
First time, change timer to interrupt every 2.5µs
Sample signal and place in 4-element sample array indexed by quadrant counter
If (quandrant counter == 3) then
        Add two bits to input buffer
        (00 if 0th sample is true, 01 if 1st sample is true, 10 if 2nd sample is true, 11 if 3rd sample is true)
        Reset quadrant counter
Else
        Increment quadrant counter
```

Option II:

General idea: Interrupt on rising edges of signal and measure time since last interrupt to determine which bits to add to input buffer. Need a “00” at start to know how to interpret first rising edge.

Initialization:
Set to interrupt on rising edge of input signal

Interrupt service routine for rising edge:

```
First time, set last_edge_time to 0, set last_bits to “00”, start counter
Every other time, compute spread = current_time – last_edge_time
If last_bits were “00”
        And if spread approx. 10.0µs then append “00” to input buffer
        Or if spread approx. 12.5µs then append “01” to input buffer
        Or if spread approx. 15.0µs then append “10” to input buffer
        Or if spread approx. 17.5µs then append “11” to input buffer
If last_bits were “01”
        And if spread approx. 7.5µs then append “00” to input buffer
        Or if spread approx. 10.0µs then append “01” to input buffer
        Or if spread approx. 12.5µs then append “10” to input buffer
        Or if spread approx. 15.0µs then append “11” to input buffer
If last_bits were “10”
        And if spread approx. 5.0µs then append “00” to input buffer
        Or if spread approx. 7.5µs then append “01” to input buffer
        Or if spread approx. 10.0µs then append “10” to input buffer
        Or if spread approx. 12.5µs then append “11” to input buffer
If last_bits were “11”
        And if spread greater than 10.0µs then append “00” to input buffer,
                Set last_bits to “00” and last_edge_time to last_edge_time + 2.5µs
                Run this interrupt routine again
        Or if spread approx. 5.0µs then append “01” to input buffer
        Or if spread approx. 7.5µs then append “10” to input buffer
        Or if spread approx. 10.0µs then append “11” to input buffer
Set last_edge_time to current time
Set last_bits to bits last appended to input buffer
```

Part III (discussion – 15 points)

(a) Is it possible to look for only the rising edges during decoding? If so, why? If not, why not?

Option I: No need to look for rising edges after the first one that is used to start up the timer.

Option II: Yes, only rising edges are fine because we are keeping track of the last two bits that were decoded.

(b) How much jitter can your approach tolerate? Please explain. Jitter is the amount by which an edge can move from its ideal position.

Option I: Jitter can be +/- 1.25 µs as it relies on sampling in the middle of each quadrature pulse

Option II: Jitter can be +/-1.25 µs and this can be embodied in the approximately time separation comparision.

(c) Does your decoding approach rely on an initial pattern in the data (e.g., a preamble)? If so, why? If not, why not?

Option I: Yes, an initial "00" (or any other known bit pair) is needed to synchronize the timer to sample at the right times.

Option II: Yes, an initial "00" (or any other known bit pair) is needed to allow for appropriate comparison of separation between successive rising edges.