CSE 466 Exam I 10 February 2006

1. Microcontrollers (30 points)

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

a) Pulse-width modulation

Pulse width modulation involves generating a periodic signal with adjustable duty cycle. The main parameters are the period and the percentage that the signal is high during that period. For example, we used this in generating different light levels on the tri-color LEDs.

b) General-purpose I/O pins

General-purpose I/O pins are used to control or sense a binary value connected to the microcontroller. A GPIO pin can be used to sense a pulse-width modulated signal and determine the percentage duty-cycle. This is how the accelerometer transmits its x-y forces to the microcontroller.

c) SPI slave interrupt

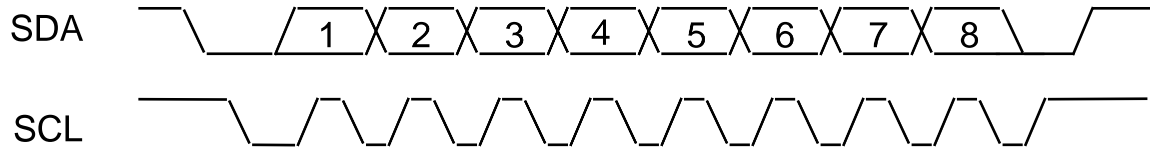
This interrupt is signaled whenever an SPI slave device receives a new byte of data from the SPI master. In our labs, we used the SPI interrupt to determine when new data is exchanged between the microcontroller and the USB interface chip.

d) Auto trigger for A-to-D conversion

An automatic trigger for A-to-D conversion causes the A-to-D circuitry to sample and convert and analog value periodically. This allows the microcontroller to read a register for the converted value whenever it needs the most recent conversion. This is useful for determining potentiometer values as for the dimmer pot in our project.

2. Decoding (40 points)

Below is an example of an 8-bit packet of encoded data similar to an I2C (or TWI) serial interface. Recall that this interface has two open collector wires. Either the transmitter or receiver can hold them low.



Note that the transmitter starts the byte with a "start" falling transition on SDA while the clock (SCL) is high. This is then followed by 8 pulses on SCL corresponding to the 8 bits of data. Assume the receiver will sample the signal on the rising edge of the clock. Finally, after the 8th bit, the transmitter signals a "stop" bit by holding SCL high while placing a rising edge on SDA. The receiver can slow down the transmitter by holding SCL low, thus preventing a rising transition on SCL to signal a new bit.

Show how you would write the code to receive and decode such a signal using the features of the ATmega16 microcontroller. Do not be concerned with syntax. In fact, you can use English (e.g., "read timer value"). However, you must label interrupt routines so that it is clear what condition will cause them to be executed. Also, make sure to clearly show how timers are set. Make sure to describe the timers/input-capture/output-compare/GPIO-pins/port-configurations or other devices you would use internally to the microcontroller.

a) Describe how a receiver would detect the "start" of a packet.

A "start" bit is signaled by a falling transition on SDA while the clock is high. The easiest way to detect this is to connect the SDA and SCL lines to interrupts. To detect a start bit, the SDA interrupt can be set to detect a falling edge (and check SCL is high) and the interrupt service routine can set a PACKET_FLAG.

b) Describe how a receiver would decode each bit. What would a receiver have to do if it was very slow – and could not receive the bits as fast as the transmitter would like to send them? Your routines should call a function called "got_bit(bit_value, bit_type)". The parameters are the value of the bit just decoded and a bit type which is one of START, DATA, or STOP. For a START the value is assumed to be 0, for a STOP the value is assumed to be one, for DATA it is the value of the SDA line when the SCL line went high.

To decode the bits, we would need an interrupt on the rising edge of every clock pulse. The interrupt routine would hold the clock line low to make the sender wait (remember that it is an open-collector line, meaning that the GPIO would have to be set to output with a value of zero) and sample the value on SDA as the value of the bit. Once this is done, the SCL line can be released by making its GPIO pin an input again.

c) Describe how a receiver would detect the "stop" at the end of a packet.

In an analogous way to detecting the "start" bit, we would need an interrupt service routing for a rising edge on SDA that checked that SCL was high. This routine could then clear the PACKET_FLAG. Of course, SDA will rise and fall at other times, but then the SCL line should be low between bits. That is why we must check that SCL is high for the "start" and "stop" bits.

d) If you assume the receiver has a clock that runs at 10MHz, what is the fastest a packet can be properly received? Explain your conclusion and clearly state your assumptions about how long each routine may take.

Assuming that the interrupt routine for decoding each bit takes X instructions to execute, the data transfer rate will 10MHz/X. For example, if there are 20 instruction cycles in the service routine (a reasonable guess given the interrupt overhead, switch of SCL from input to output and back to input, and the placing of a new bit value into the received byte buffer), then we could handle transitions that cause interrupts at the rate of 500K/sec. A byte-long packet (as in the diagram) consists of a worst-case 28 transitions (2 for the start bit, 3x8 for the data bits – 2 clock edge and one data edge, and 2 for the stop bit). This gives us a total of 500K/28 or approximately 15-20K packets/sec.

<u>*Pseudo-code:*</u>

<u>*In main:*</u>

```
set input capture for any edge (rising or falling) on SCL and SDA pins
set PACKET_FLAG = FALSE;
loop to do other things including waiting for a full-packet to be received
```

<u>*In service routine for SDA interrupt:*</u>

```
if (SCL == TRUE) {
        if (SDA == FALSE) { got_bit(0, START); PACKET_FLAG = TRUE; }
        else { got_bit(1, STOP); PACKET_FLAG = FALSE;  }
} else { ignore transition }
}
```

<u>*In service routine for SCL interrupt:*</u>

```
if ( (SCL == TRUE) && (PACKET_FLAG == TRUE) ) {
        set SCL to output a 0
        got_bit(SDA, DATA);
        set SCL to input;
} else { ignore transition }
}
```

3. Measuring time (30 points)

You are building a sonar range finder. It will make use of a compass and an ultrasound transceiver. The compass is only an input to the microcontroller and has 4 wires, one for each of the cardinal directions. Up to two wires can be high at one time so that the compass can specify any of N, NE, E, SE, S, SW, W, NW, and W. The ultrasound range finder uses just two wires. One is an output from the microcontroller that tells the range finder to emit a sound pulse. The other is an input to the microcontroller that will be high when an echo is detected. Your device is to take a range reading every time the compass changes direction. At the completion of the reading it should call a function `new_range` with arguments of direction and time until the echo returns in timer units (don't worry about conversions), e.g., `new_range(direction, return_time)` where `direction` is the state of the compass when the distance measurement was started and `return_time` is the timer's counter value when the echo returned. Assume that sound travels at 343m/sec and that you have a 10MHz clock for your microcontroller.

a) Write pseudo-code for how you would implement this system. You should have a routine for detecting a change in the compass heading and starting an ultrasound pulse and a routine for determining the time from starting the ultrasound pulse until the sensor hears its echo.

On every transition (falling or rising) of any of the compass wires, record the compass direction and take a new range reading with the sonar. If our microcontroller doesn't have four separate interrupt wires, then we can use some Boolean logic to create one signal that changes every time the compass changes direction (e.g., a parity circuit made of 3 XOR gates).

<u>In main:</u>
set up an interrupt for any edge on the compass signal(s)
set up an interrupt on rising edge of ECHO signal from ultrasound rangefinder

<u>In service routine for compass signal(s):</u>
```
direction = compass direction;
clear and start timer;
if (WAITING_FOR_PULSE == FALSE) {
        raise signal to generate ultrasound pulse;
        WAITING_FOR_PULSE = TRUE;
}
```

<u>In service routine for ECHO signal rising edge:</u>
```
return_time = value of timer;
new_range (direction, return_time);
disable timer;
lower signal that generated ultrasound pulse
WAITING_FOR_PULSE = FALSE;
```

b) How fast can a user move the compass (i.e., minimum time between changes in heading)? How does it affect the possible range of distance measurements?

If we assume we want to measure up to 10m distance then the compass can move as fast as 20m/343m (20m for the round-trip) or approximately .06 seconds for every change in heading. This assumes our instruction execution overhead is negligible (which it likely is given the slow speed of sound). This should be reasonable for even some fast movements that change the compass direction. If we want to measure up to 100m (if we even could with this ultrasound rangefinder) then it would be every 0.6 seconds. This is less likely to be adequate.