CSE 466 Exam I 5 Nov 2004

1. Timers (20 points)

Define the following terms related to features of the timers in your Atmega16 microcontroller and provide an example of when each feature would be used. Relate your examples to the Ball Lightning project, if possible.

a) Overflow (5 points – 3 for definition, 2 for example)

When a timer rolls over from its maximum value back to 0. An overflow interrupt can be used to generate time intervals greater than a single timer's counting ability by counting the number of overflows. For example, in generating a very long period for a PWM signal.

b) Auto-reload (5 points – 3 for definition, 2 for example)

When a timer reaches its maximum values and reload to a set value. Used to generate periodic interrupts of a particular duration. For example, in generating a PWM signal.

c) Input capture (5 points – 3 for definition, 2 for example)

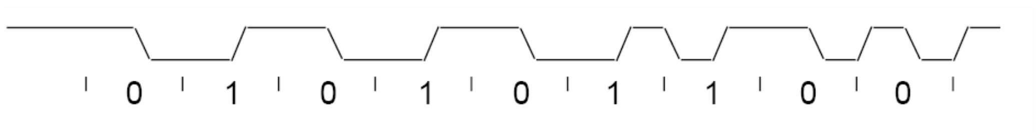
Captures the value of the counter when an input changes value. The interrupt can be serviced later but the time saved is exact to within one timer clock cycle rather than to when the ISR actually executes. For example, in measuring the duty-cycle of a PWM input.

d) Output compare (5 points – 3 for definition, 2 for example)

Changes an output when a timer reaches a specified value. For example, to change the value of a PWM output signal so as to generate the right duty-cycle. Used in conjunction with auto-reload, it makes it unnecessary to have an interrupt service routine to generated a PWM signal.

2. Decoding (50 points)

Below is an example of a short packet of Manchester encoded data.



Note that each bit has two parts: a 0 is a high followed by a low, a 1 is a low followed by a high. Each bit within a packet is of the same duration but this duration may vary from packet to packet. The transition in the middle of a bit may jitter from bit to bit by at most 10% of the bit time. Show how you would write the code to decode such a signal using the timer 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.

a) Describe how you would detect a starting bit (which is always a 0)? (10 points)

A start bit is signaled by a falling transition. Set an input capture interrupt to catch a falling edge.

b) Describe how you would decode each bit. Do you need to have alternating bits at the start of the packet? Why or why not? (10 points)

We need to have alternating bits at the start of the packet (a preamble) so that we can measure the bit period as it can vary. Once we have a running average of the bit period, we can set an input capture for any transition. If the transition is too early, we ignore it and wait for the next one. The next should be exactly one bit period and its direction (the current value of the wire) tells us the value of the bit.

c) Describe how you would detect the end of the packet (when the line goes high)? (10 points)

At the end of the packet the line stays high. We set another timer to overflow after more than one bit time. If this timeout counter generates an interrupt, then the line stayed quiet and we have a stop bit at the end of the packet. The time out counter needs to be reset at every transition.

d) Write some pseudo-code for decoding a packet one bit at a time. 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_BIT, PACKET_BIT, or STOP_BIT. For a START_BIT the value is assumed to be 0, for a STOP_BIT the value is assumed to be one. (10 points)

```
In main:
set input capture for any edge (rising or falling)
set timeout timer using an output capture timer that will interrupt at a certain value to be set later
set packetflag = FALSE;
set avg_bit_time = 0;
set new_bit = FALSE;
loop forever
if ( new_bit == TRUE ) {
        compute_average_bit_time (bit_time);
        got_bit(bit_value, bit_type);
        new_bit = FALSE;}
}
```

```
In input capture service routine:
if ( (packetflag == TRUE ) && ( .75*avg_bit_time > counter_value > .25*avg_bit_time ) ) {
        ignore transition;
} else {
        bit_time = counter value;
        bit_value = value of wire;
        if packetflag {
                bit_type = PACKET_BIT;
        } else {
                bit_type = START_BIT;
                packetflag = TRUE;
        }
        new_bit = TRUE;
        reset timer (so that we can count next bit time and also reset timeout with output compare);
}
```

```
In output capture service routine:
bit_value = value of wire;
bit_type = STOP_BIT;
new_bit = TRUE;
packetflag = FALSE;
```

```
In compute_average_bit_time:
keep track of first N (minimum number of bits of preamble that will be seen, say 8) transitions
set avg_bit_time to this value
then for every next call keep running average (avg_bit_time = avg_bit_time * 7 + bit_time) / 8;
```

e) If you assume your timer has a clock at 10MHz, how fast can the packets be? That is, what is the shortest bit time for which you can still decode the signal correctly? Explain your conclusion and clearly state your assumptions. (10 points)

*We may have up two transition per bit time. If we execute approximately 100 instructions – 100 cycles at 10MHz = 100 * 1 / 10MHz = 10usec. Therefore our fastest packets can go no faster than 20usec bit times which is equivalent to 1 / 20usec = 50Kbits/sec.*

3. Filtering (30 points)

You have a pin on your ATmega16 microcontroller connected to a vibration sensor. It goes high and low with each oscillation. Write pseudo-code for a routine that can determine the frequency of the vibration and reports if it is within the range of 10-20Hz. It should set a variable when the vibration is present and clear when the vibration is at a frequency outside this range. This is called a band-pass detector – selecting only frequencies within a certain range and not those lower or higher. Your filter should make its decision within 1 sec. The vibration sensor is quite noisy and its period may vary by as much as a factor of 3 – that is, a period that should be 100ms may be as great as 300ms or as small as 33ms – but will be correct, on average, over 10 periods.

On every rising (or falling) transition measure the time since the last transition to get a period. Keep track of 10 of these with a running average. If the average is ever between 10 and 20 Hz, that is, 50-100msec then set in_band to TRUE to indicate that a vibration is present. If it is outside that range then reset in_band to FALSE.

<u>In main:</u>

```
set up input capture for any rising edge
for each time the input capture handler reports a new edge
{
    compute average period keeping track of the average of the last 10
    period = count of overflows * timer maximum + current timer value
    reset timer
    reset overflow counter
    if (average is in the band of 50-100msec) in_range = TRUE; else in_range = FALSE;
}
```

<u>In input capture:</u>

```
report a new edge
```

<u>In overflow handler:</u>

```
increment overflow counter
```