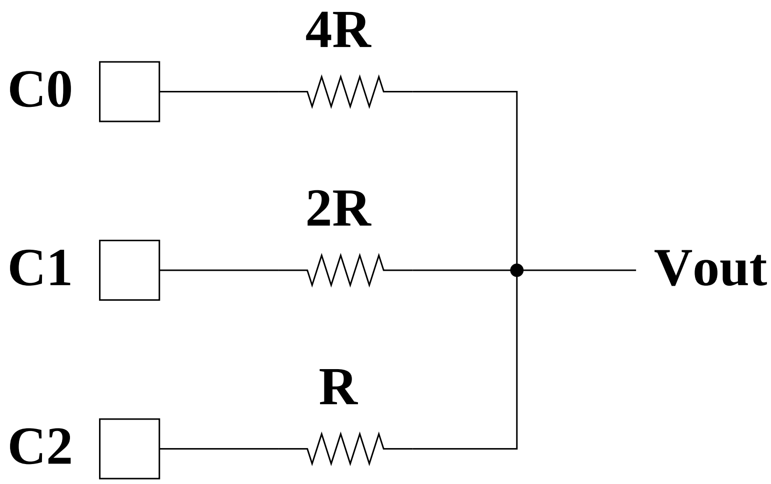
Embedded Real-Time Systems (AME 3623)
Homework 6 Solutions

May 2, 2006

Question 1

Consider the following circuit in which three digital outputs ($C0$, $C1$, $C2$) drive an analog circuit. For bit values 0 and 1, each pin will be driven at $0V$ and $+5V$, respectively.



(20pts) Starting from Kirkoff's law, derive an equation for $Vout$ in terms of the three digital signals.

By Kirkoff's (current) law, we know that the sum of the three currents must be zero (note that we assume that no current is moving to the right hand side of the figure from the node):

$\sum_i I_i = 0$

By Ohm's law, we have:

$\sum_i \frac{\Delta V_i}{R_i} = 0$

The voltage at pin i is $5C_i$. So:

$\sum_i \frac{5C_i - V_{out}}{R_i} = 0$

Reorganizing, we have:

$\frac{5}{R} \sum_i \frac{C_i}{2^{2-i}} = \frac{V_{out}}{R} \sum_i \frac{1}{2^{2-i}}$

Therefore:

$V_{out} = \frac{20}{7} (C_0/4 + C_1/2 + C_2)$

or:

$V_{out} = \frac{5}{7} (C_0 + 2C_1 + 4C_2)$

(10pts) For each possible combination of boolean values for the Ci's, give the actual value of $Vout$.

C_2	C_1	C_0	V_{out}
0	*0*	*0*	0
0	*0*	*1*	$5/7 = 0.7143$
0	*1*	*0*	$10/7 = 1.4286$
0	*1*	*1*	$15/7 = 2.1429$
1	*0*	*0*	$20/7 = 2.8571$
1	*0*	*1*	$25/7 = 3.5714$
1	*1*	*0*	$30/7 = 4.2857$
1	*1*	*1*	$35/7 = 5.0$

Question 2

(10pts) Give two disadvantages for performing I/O through polling.

1. *Processor time can be wasted while the processor waits for signal lines to change state.*
2. *Polling must be done at a high enough frequency (or else events might be missed). This can be a problem if the processor is also trying to accomplish another (possibly unrelated) task.*

(10pts) Explain (in brief) how the use of *interrupts* solves these two problems.
These answers parallel the ones from above.

1. *When using an interrupt to handle an event, there is no need for the processor to spend time repeatedly checking the signal lines (since an interrupt is only raised when things have changed).*
2. *The use of an interrupt (in most cases) enables the processor to respond almost immediately to an event.*

(10pts) List two necessary conditions for there to be a shared data problem.

1. *A data structure is accessed both by the main program and by an interrupt routine (more generally: a data structure is accessed by two or more processes).*

2. *The interrupt routine can be executed while the main program is in the middle of an access to the shared data structure (more generally: the processes accessing the shared data structure can execute asynchronously).*

(15pts) Suppose we want a small segment of code – called *donow*() – to be executed precisely once every 5.12ms. What is the timer0 prescalar configuration and the (psuedo)code for the interrupt routine?

We will use a prescalar of 64. This gets us down to an interrupt every 1.024 ms. We then need an interrupt routine with an additional counter that expires at 5. So, we are left with an interrupt interval of: $5 * 256 * 64/16000000 = 5.12ms$.

```
SIGNAL(SIG_OVERFLOW0) {
    ++counter;
    if(counter == 5) {
      donow();
      counter = 0;
   };
};

Somewhere in the main program:

// Initialize counter
counter = 0;
// Interrupt occurs every (64*256)/16000000 = 1.024 ms
timer0_config(TIMER0_PRE_64);
// Enable the timer interrupt
timer0_enable();
// Enable global interrupts
sei();
```

Question 3

Consider a hybrid priority and round-robin scheduler that is **non-preemptive**. Consider also three regularly-scheduled processes:

Task 1 executes at $2Hz$/priority 2 and requires $50ms$ of processing time (it moves from the *waiting* to the *ready* state at $t = 0, .5s, 1s, 1.5s$, etc.).

Task 2 executes at $4Hz$/priority 2 and requires $100ms$ of processing time (and moves from waiting to ready at $t = 0.01, .251s, .501s, .751s, 1.01s$, etc.).

Task 3 executes at $1Hz$/priority 1 and requires $300ms$ of processing (and moves from waiting to ready at $t = 0, 1s, 2s$, etc.).

Assume that priority 2 is the highest priority (this is not the case for all OS's).

(20pts) At 50ms intervals, show which process is occupying the processor at any given time for the interval $t = [0s, 1.5s]$.

Time	*Execution*	*Ready (left is top of queue)*	*Notes*
0	*T1a*	*T2a, T3a*	*Time interval: 0 to 50*
50	*T2a*	*T3a*	
100	*T2a*	*T3a*	
150	*T3a*	-	
200	*T3a*	-	
250	*T3a*	*T2b*	
300	*T3a*	*T2b*	
350	*T3a*	*T2b*	
400	*T3a*	*T2b*	
450	*T2b*	-	
500	*T2b*	*T1b, T2c*	*T2b misses deadline!*
550	*T1b*	*T2c*	
600	*T2c*	-	
650	*T2c*	-	
700	-	-	
750	*T2d*	-	*T2d actually doesn't start execution*
800	*T2d*	-	*until 751; and it completes at 851*
850	-	-	
900			
950			
1000	*T1c*	*T2e, T3b*	
1050	*T2e*	*T3b*	
1100	*T2e*	*T3b*	
1150	*T3b*	-	
1200	*T3b*	-	
1250	*T3b*	*T2f*	
1300	*T3b*	*T2f*	
1350	*T3b*	*T2f*	
1400	*T3b*	*T2f*	
1450	*T2f*		
1500	*T2f*	*T1d, T2g*	*T2f misses deadline!*