Jonathan W. Valvano

First Name: _____ Last Name:_____

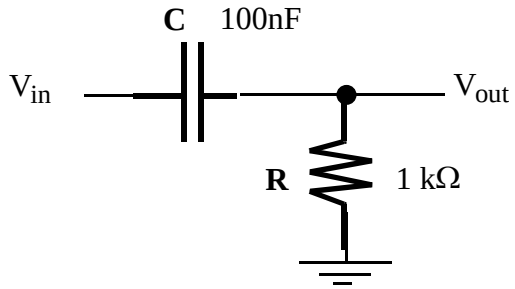
April 2, 2010, 10:00 to 10:50am

Open book, open notes, calculator (no laptops, phones, devices with screens larger than a TI-89 calculator, devices with wireless communication). Please don't turn in any extra sheets.

(10) Question 1. Let **x(t)** be the input signal interfaced to a 12-bit ADC. 256 data points are sampled at 10 kHz using the ADC, and the collected data are converted to the frequency domain by calculating the FFT. Consider a situation where for **k**=0 to 128, all FFT output terms **y(k)** are zero except **k** equals 5. Furthermore, the real part of **y(5)** is zero, but the imaginary part of **y(5)** is 0.5 V.
Part a) What was the **x(t)** input signal in the time domain? I.e., give the amplitude(s), frequency(s) and phase(s) of the input signal.

Part b) Which of the **y(k)** terms for **k**= 129 to 255 are nonzero? For each nonzero term specify its value.

(20) Question 2. Consider the following RC circuit.



Part a) At time less than zero $\mathbf{V_{in}}$ is zero, and for times greater than zero $\mathbf{V_{in}}$ equals 3.3V, what will be the transient output? Give both an explicit equation $\mathbf{V_{out}(t)}$ and a rough sketch of $\mathbf{V_{out}}$ versus time.

Part b) Derive the gain versus frequency response of this circuit? Give both an explicit equation for gain as a function of frequency and a rough plot of this function in the frequency domain.

(25) Question 3. The goal of this problem is to implement the following IIR digital filter. The sampling rate is 20 kHz, and the ADC is a 12-bit unsigned 0 to +3.3V range converter.

y(n) = 0.3333333333x(n) – 0.2222222222y(n-1)

You will develop C code to implement this filter.

Part a) Write C code to define data structure(s) needed to implement this IIR filter. Full credit will to the simplest data structure**.**

Part b) Show the C function that implements this filter using 16-bit signed integer math. 32-bit integer and floating point are not allowed. Choose integer constants that give a good implementation and will have no overflow when using 16-bit arithmetic. You cannot cast any numbers into **long.** The prototype for the function you are asked to write is

```
short IIR(short input);  // input is the new ADC
```

An example usage of this filter is

```
void Producer(void){ short data,result;
  data = ADC_In(1);      // sample channel 1
  result = IIR(data);    // call your function
  LCD_Plot(result);      // display filter output
}
```

(30) Question 4. Consider an autonomous vehicle that logs debugging information on a flash EEPROM-based disk. Each time the vehicle is powered up, a new file is created. Assume **`name`** is unique each time the vehicle starts, calling:

```
eFile_Create(name);
eFile_WOpen(name);
```

In this application, debugging data exists as 4096-byte chunks. This means whenever data is stored, **`eFile_Write`** is called exactly 4096 times. While the vehicle runs, a stream of debugging data is saved into the file using code like:

```
for(i=0; i<4096; i++)
  eFile_Write(DebugData[i]);  // save one 8-bit character
```

Just before the vehicle is turned off, the file is closed, calling:

```
eFile_WClose();
```

In this system, files are *never deleted*. The size of the disk will be large enough to hold all data for the life of the vehicle. If problem with the vehicle occurs (accident, fire, explosion etc.), the contents of the files will be printed out for legal reasons. For most systems, the data are never read from the disk. Assume the block size is 4 kibibytes, the disk size is 128 mebibytes, the directory fits in one block, and the disk contains all the free-space management you need to store.

Part a) What is the simplest way to manage free space? You may assume there will be no bad disk blocks. Draw pictures to describe your free space management scheme.

Part b) What is the simplest way to organize data on this disk so that there is no external fragmentation. Draw pictures to describe your allocation scheme.

Part c) Describe the directory structure you would need. Assume the file names are exactly 4 characters (fits in 32 bits). How many files can you store on this disk?

Part d) In this application is there any internal fragmentation? Justify your answer.

(15) Question 5. You are asked to design a file system that is extremely reliable. You will consider individual block errors and not systematic failures like power loss, fire or explosions. In other words, on your eDisk, there is a small probability p ($p<0.0000001$) that when you write a block then later read that block one or more bytes will not have been properly recorded. You may assume once a block fails on the disk, it will remain nonfunctional. However, the damage may occur immediately or may occur up to years after the write. You want to reduce the probability of lost data to less than p^2. If an eDisk error were to occur, some data in the block during read would not equal the data previously written, Other than the data being wrong no other warning or error is reported by the hardware. You can assume that disk errors in one part of the disk are not correlated to errors in another part of the disk, and the disk errors are uniformly distributed across the disk. You may assume the disk size is many times larger than the application requires or you may use multiple disks. These are the prototypes for the eDisk functions, like Lab 5, that are given.

```
DRESULT eDisk_WriteBlock (const BYTE *buff, DWORD sector);
DRESULT eDisk_ReadBlock (BYTE *buff,  DWORD sector);
```

Your solution will exist as software layer between eFile and eDisk.

Part a) Briefly describe your new **`nDisk_WriteBlock`** you will design. This new function will be called in eFile in all places the eFile used to call **`eDisk_WriteBlock`**. Your **`nDisk_WriteBlock`** will call **`eDisk_WriteBlock`**.

Part b) Briefly describe your new **`nDisk_ReadBlock`** you will design. This new function will be called in eFile in all places the eFile used to call **`eDisk_ReadBlock`**. Your **`nDisk_ReadBlock`** will call **`eDisk_ReadBlock`**.