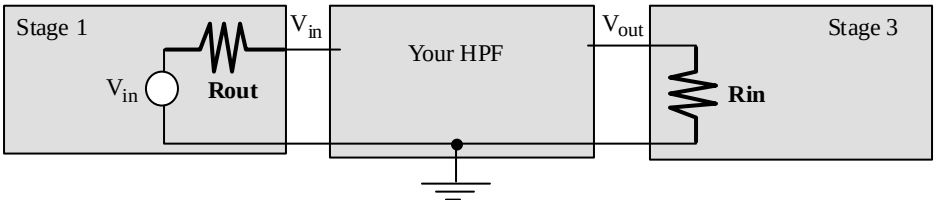
Jonathan W. Valvano

First Name: _____ Last Name:_____

April 6, 2012, 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.

(20) Question 1. Design a one pole analog high pass filter with a 1 kHz cutoff. Stages 1 and 3 use rail to rail op amps powered with 3.3V. The output impedance of Stage 1 is R_{out}=0.01Ω, and the input impedance of Stage 3 is R_{in} = 100 MΩ. The range of input voltage is 0 to 3V, and the range of the output voltage should also be 0 to 3 V (the signal plus a 1.50V offset). Show your work and include equations, part numbers, and values for resistors and capacitors. To make it easy for me to grade, use a 0.1 μF capacitor. However specify capacitor type (e.g., tantalum, C0G, Z5U or electrolytic)



(15) Question 2. First, derive the **H(z)** transform for this filter. Then, use the transform to determine the DC gain of this filter. Explain what this number means.

y(n) = x(n) + y(n-1)

(25) Question 3. Write software to implement this 60 Hz digital reject filter. The sampling rate is 1000 Hz,

y(n) = x(n) -1.8595529717765x(n-1) + x(n-2) + 1.76657532318768y(n-1)-0.9025y(n-2)

The prototype for the function is

```
long Filter(long data);
```

where the input parameter is the 16-bit ADC sample (-32768 to +32767), and the output parameter is the filter output. An example usage of this filter is (**`Producer()`** is called from the background at 1000 Hz)

```
void Producer(void){ long data,result;
  data = ADC_In(1);       // sample channel 1, 16 bits
  result = Filter(data);  // call your function
  oLED_Plot(result);      // display filter output
}
```

Assume each addition, each multiplication, each memory read, and each memory store takes one bus cycle. Logical shift operations require no time because of the barrel shifter. Division requires up to 12 cycles. Floating point operations will require up to 100 cycles each. We are also not counting bus cycles needed to fetch instructions, because fetching op codes often runs in parallel with data operations.

Part a) Rewrite the filter equation to minimize execution speed. I.e., show the fixed-point filter equation (part b will be the C code). You do not need to maximize accuracy, but you do need to avoid overflow. How many cycles do you estimate that it will require to execute?

Part b) Show the C function that implements this filter using integer math. Include the MACQ. If you need initialization, define a second function that initializes the data structure. I.e., implement in C the **`Filter`** function as shown on the last page.

(40) Question 4. In this question we will make an electronic disk is using 32k of internal flash EEPROM. Specifically ROM locations 0x0000.8000 to 0x0000.FFFF will be used to store data. You cannot write 1s to ROM. The write proper sequence is to erase the block (making bits all 1), then program data (setting the zeros as needed) You are given the following functions.

```
// initialize flash, freq is the crystal frequency
void Flash_Init(unsigned char freq);

// addr is aligned to 1k block from 0x0000.8000 to 0x0000.FFFF
// returns 0 if ok
int Flash_Erase(unsigned long addr); // erase 1024 bytes

// Write 32-bit data to flash
// Return 0 if successful
int Flash_Write(unsigned long addr, unsigned long data);

// Write a 1k block to flash
// pt is call by reference to 1024 bytes of data
// addr is aligned to 1k block from 0x0000.8000 to 0x0000.FFFF
// returns 0 if ok
int Flash_ProgramBlock(char *source, unsigned long addr);
```

If you wish to read ROM, you could define a fixed pointer, similar to the way ports are addressed.

```
#define Buf  (*((unsigned long *)0x00008000))
```

You can also write C code with the following syntax creating a variable pointer:

```
void MyRoutine(void){ unsigned long *pt; unsigned long num;
  pt = (unsigned long *)0x00008000;
  num = *(pt+1); // reads 32-bit word at 0x0000.8004
  num = Buf[1];  // reads 32-bit word at 0x0000.8004
}
```

Obviously you cannot write to ROM using these definitions, but you can read from ROM. The disk will be used in the following fashion. Once at the time the system is constructed and the main program is downloaded, your function **`File_Init()`** will be called. During the life time of the system your function **`File_Record()`** will be called to store 2000 bytes of data at time. Because there is only 32 kibibytes allocated to this disk, this function can be called only 15 times. Data is never deleted.

Part a) How would you design the **free space management**? It is about managing free space.

Part b) How would you allocate disk space? Draw a picture of where data will be stored

Part c) Show the implementation of **`File_Init`**. This function is the only one allowed to erase the flash. The prototype for your function is

```
int File_Init(void);
```

which will return 0 if successful.

Part d) Show the implementation of **`File_Record`**. The prototype is
`int File_Record(unsigned char *pt); // stores 2000 bytes`
which will return 0 if successful. **`pt`** is a pointer to a 2000-byte block of data. You may assume the user buffer that contains the 2000 bytes of data was allocated a full 2048 bytes of space.