Jonathan W. Valvano April 1, 2011, 11:00 to 11:50am

(15) Question 1.

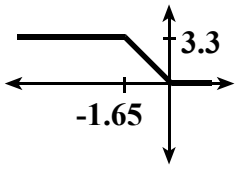
Part a) You add windowing to remove spectral leakage.

Part b) FFT size affects frequency resolution.

$f_s/\Delta f = 200kHz/100Hz = 2000$, N=2048, $1/f_s = 5us$, T =5μs*2048 = 10.24ms

Part c) 80 dB = 20 $\log_{10}$ V/ΔV, 4 = $\log_{10}$ V/ΔV, V/ΔV = 10000, 13 or 14 bits. An interesting fact is 20 $\log_{10}$ 2 is about 6dB. So there is about 6dB per bit. Notice that 80(dB)/6(dB/bit) = 13.3 (bits).

(10) Question 2. The op amp is rail to rail, so it works only if the inputs and outputs are between 0 and 3.3V. This circuit tries to make Vout = -2Vin. If the input is below -1.65V, the op amp will break. If the input is positive, the output is zero.

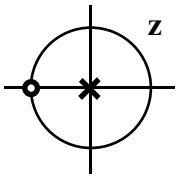


(10) Question 3. Convert to frequency domain using Z Transform

$$\mathbf{Y(z) = (X(z) + z^{-1} X(z))/2}$$

$$\mathbf{H(z) \equiv Y(z)/X(z) = (1 + z^{-1})/2 = 0.5*(1+z)/z}$$

There is a zero at **z**=-1, and a pole at **z**=0



(30) Question 4.

Part a) Given **y(n) = (x(n) + x(n-1) + x(n-2) + x(n-3) + x(n-4) + ... + x(n-63))/64**

Define **z(n)** as the sum of last 64 samples

z(n) = x(n) + x(n-1) + x(n-2) + x(n-3) + x(n-4) + ... + x(n-63)

Clearly, by the definitions of **y(n)** and **z(n)**

y(n) = z(n)/64

Shift **z(n)** in time (substitute **n-1** for **n**)

z(n-1) = x(n-1) + x(n-2) + x(n-3) + x(n-4) +...+ x(n-63) + x(n-64)

Subtract **z(n)** minus **z(n-1)**

z(n) - z(n-1) = x(n) - x(n-64)

Arrange to get the answer

z(n) = z(n-1) + x(n) - x(n-64)

Pointer solution

Part b) Show the C function that implements this faster filter using unsigned integer math.

```
unsigned long static x[65];
unsigned long static y;
unsigned long static *pt0;  // pointer to x(n)
unsigned long static *pt64; // pointer to x(n-64)
void Filter_Init(void){
  pt0 = &x[0];
  pt64 = &x[64];
}                   // enter new data into what used to be old data
unsigned long Filter(unsigned long data){
  if(pt0 == &x[0]){ // shift pointers  [1 read pointer]
    pt0 = &x[64];   // wrap            [1/64 store pointer]
  }
  else{
     pt0--;         //                 [63/64 store pointer]
  }
  if(pt8 == &x[0]){ // shift pointers  [1 read pointer]
    pt8 = &x[64];   // wrap            [1/64 store pointer]
  }
```

```
  else{
     pt8--;          //                [63/64 store pointer]
  }
  *pt0 = data;       //                [1 store data at pointer]
  y = (64*y + data – (*pt8))/64;    //  [2 reads, 1 store, 2 adds]
  return y;
}
```

Part c) This filter requires just 5 memory cycles to enter new data, and 5 cycles to run the filter. This is a total of 10 cycles.

Indexed solution

Part b) Show the C function that implements this faster filter using unsigned integer math.

```
unsigned long static x[65];
unsigned long static y;
unsigned long static I;  // index into buffer
void Filter_Init(void){
  I = 0;
}                   // enter new data into what used to be old data
unsigned long Filter(unsigned long data){ unsigned long x64;
  I = (I+1)%65;                //  [1 read index, 1 divide, 1 store]
  x[I] = data;                 //  [1 store data at index]
  x64 = x[(I+64)%65];          //  [1 read at index, 1 divide]
  y = (64*y + data – x64))/64; //  [1 read, 1 store, 2 adds]
  return y;
}
```

Part c) This filter requires just 4 memory cycles to enter new data, and 6 cycles to run the filter. This is a total of 10 cycles.

(20) Question 5. *Wear-leveling* attempts to erase/program all the blocks at the same rate.

Part a)

1) I would implement free space by placing all free blocks in a linked list. When a new block is needed, I would take one off the front of the list. When a new block is returned, I would put it at the end of the list.

2) I could have a counter for each block recording how many times it has been erased. When a new block is needed, I would choose the one with the smallest count.

Part b) Manage the **directory** to improve the life span of the disk

1) Put the directory in battery-backed RAM, and not store it on the disk at all.

2) Perform statistics on the minimum time it takes to generate 50,000 directory writes. Assume this time is 1 month. Allocate the first 1000 blocks for the directory, and use a formula mapping the month to the number from 0 to 999 (the system lasts for 1000 months or 83 years). This way, a new block is used for the directory before 50,000 writes have occurred.

3) Count the number of times the directory is written. The directory has an extra pointer. If the pointer is null, it means this is the real directory. If the pointer is valid it points to another directory block in a linked list that is more recent. After 50,000 writes, allocate a new block for the directory, and write once more to the old block putting a directory pointer to the new directory. After a million directory writes there will be a linked list of 20 directory blocks, with the last one most recent. Mounting the disk will require 20 block reads to find the real directory.

4) Run the system on battery power keeping the directory in RAM, and write the directory at most once a day. This system lasts 100,000 days or 274 years.
5) Use 1000 blocks for the directory in a structured and obvious way (i.e., the directory entry for each file is a separate block). If the file access is uniform, then directory block writes are reduced by a factor of 1000.
6) Implement a two-layer directory. The first block simply points to the location of the actual directory block. Count the number of times the directory is written. After 50,000 writes, allocate a new block for the directory, and write once the first block putting the directory pointer to the new directory. After many directory writes there will be exactly two directory blocks, with the second one containing the directory information. Mounting the disk will require exactly 2 block reads to find the real directory.

(15) Question 6. Let 2^m be the cluster size. 32 bits is 4 bytes. Since each cluster address is 4 bytes, a cluster can store 2^{m-2} addresses. Since there is no external fragmentation, we must allow for one file to occupy the entire disk. To minimize internal fragmentation we want to minimize cluster size. Minimize **m**, such that

$$2^m * 2^{m-2} \geq 2^{30}$$

which is

$$2^{2m-2} \geq 2^{30}$$

or

$2m-2 \geq 30$ => m=16, so the cluster size is 65536 bytes