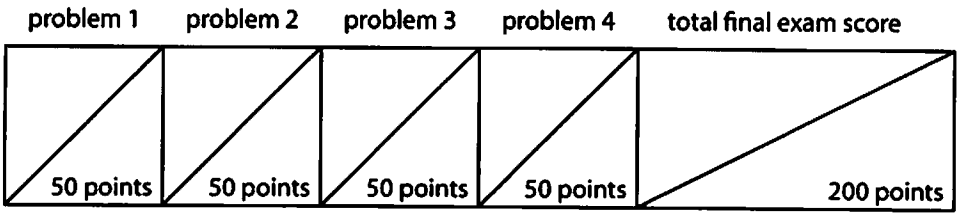
ECE4703 Final Exam

Your Name: SOLUTION Your box #: _____

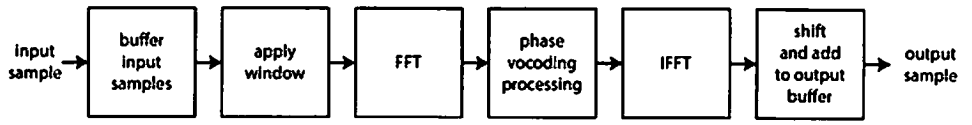
December 13, 2012

Tips:

- Look over all of the questions before starting.
- Budget your time to allow yourself enough time to work on each question.
- Write neatly and show your work!
- This exam is worth a total of 200 points.
- Attach your "cheat sheet" to the exam when you hand it in.



1. 50 points total. Suppose you write a phase vocoder using frame-by-frame processing as shown below. The window size is denoted as N.



(a) 30 points. Using the profiler, you compute the following table of results for different values of the window size parameter N. Assuming a sampling rate of $f_s = 44.1$ kHz, a DSP clock rate of $f_c = 225$ MHz, and a fixed window overlap of 50%, what is the largest value of N that you can use and run in real-time? Show your reasoning.

N	cycles
16	660
32	2244
64	8484
128	33252
256	131940
512	525924
1024	2100324
2048	8394852
4096	33566820
8192	134242404

$\frac{N}{2}$ samples between frames

1 sample = 5102 cycles

→ 256 samples = 1,306,112 cycles > 525924 so ok
→ 512 samples ≈ 2.6×10^6 > 2.1×10^6 still ok
→ 1024 samples ≈ 5.2×10^6 < 8.4×10^6 not ok

⇒ N = 1024 is the largest value of N that will run in real-time

(b) 20 points. Suppose you determine the asymptotic complexity of the "phase vocoding processing" block is $\mathcal{O}(N^2)$. What is the overall asymptotic complexity of the system? Explain.

Buffering $\Theta(N)$
windowing $\Theta(N)$
FFT $\Theta(N \log N)$
Phase vocoder $\Theta(N^2)$
IFFT $\Theta(N \log N)$
output buffer $\Theta(N)$

overall asymptotic complexity is $\Theta(N^2)$

2. 50 points. In many applications of the FFT, the input sequence to be processed $\boldsymbol{x} = \{x[0], \ldots, x[N-1]\}$ is real-valued. As we saw in Lab 5, complex-valued FFT algorithms can still be used by simply populating the imaginary elements of the input buffer with zeros. Suppose you have two real-valued length-N input sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ and form the length-N complex-valued input sequence

$$\boldsymbol{z} = \{z[0], \ldots, z[N-1]\} = \{x[0] + jy[0], \ldots, x[N-1] + jy[N-1]\} = \boldsymbol{x} + j\boldsymbol{y}$$

You then perform a length-N FFT on the complex-valued input sequence $\boldsymbol{z}$ to obtain $\boldsymbol{Z} = \{Z[0], \ldots, Z[N-1]\}$.

(a) 30 points. Show how you can obtain the FFTs $\boldsymbol{X} = \{X[0], \ldots, X[N-1]\}$ and $\boldsymbol{Y} = \{X[0], \ldots, X[N-1]\}$ from the FFT $\boldsymbol{Z}$.

Hint 1: The N-point FFT of a real valued signal satisfies the conjugate symmetry property that $X[k] = X^*[N-k]$ for $k = 1, \ldots, N-1$.

Hint 2: If you are having trouble deriving this for the general case, try $N = 4$.

The FFT is linear, so $Z[k] = X[k] + j\,Y[k]$ (1)

Note that $Z^*[k] = X^*[k] - j\,Y^*[k]$

and $Z^*[N-k] = X^*[N-k] - j\,Y^*[N-k]$

$= X[k] - j\,Y[k]$ (2)

From (1) and (2), we see that

$$Z[k] + Z^*[N-k] = 2X[k]$$
$$Z[k] - Z^*[N-k] = 2jY[k]$$

Hence

$$X[k] = \frac{1}{2}\left[Z[k] + Z^*[N-k]\right]$$
$$Y[k] = \frac{1}{2j}\left[Z[k] - Z^*[N-k]\right]$$

(b) 20 points. Does this "trick" save any computation with respect to separately computing the length-N FFTs of x and y, each with imaginary elements set to zero? You can assume that we measure computation as "real-valued multiplications" and that the FFT operation requires exactly $4N \log_2(4N)$ operations to compute, irrespective of whether the input is real or complex-valued. Explain.

computing the FFT of x and y separately will require exactly $8N \log_2(4N)$ operations.

Using the "trick" we get the FFT of z with exactly $4N \log_2(4N)$ operations, but it requires a few more operations to recover X and Y.

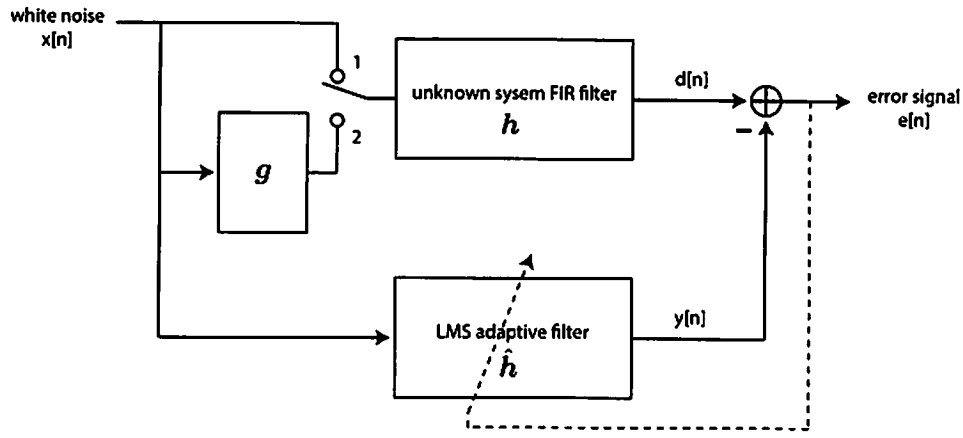
Each $X[k]$ requires two real multiplications $(k=0, \ldots, N-1)$
Each $Y[k]$ also requires two real multiplications $(k=0, \ldots, N-1)$

$\Rightarrow$ total operations of "trick" is

$$\boxed{4N \log_2(4N) + 4N \text{ operations}}$$

This is definitely more efficient than separate computation of the FFTs, especially for large values of N.

3. 50 points total. Consider the system identification adaptive filtering system shown below.



For the following questions, assume that

- The input power $\mathrm{E}\{x^2[n]\} = 1$,
- the unknown system filter coefficients are $\boldsymbol{h} = [1, 2]$,
- the system $\boldsymbol{g}$ has coefficients $\boldsymbol{g} = [1, 1]$
- the LMS adaptive filter $\hat{\boldsymbol{h}}$ has **three** coefficients that are initialized to zero prior to adaptation, and
- the LMS step-size is small enough to allow for convergence of the algorithm to the minimum mean squared error (MMSE) solution.

(a) 20 points. Suppose the switch is in position 1. What will $\hat{\boldsymbol{h}}$ be after convergence of the LMS algorithm? What will the MSE be after convergence?

$\hat{h}$ will be $[1,2,0]$ and the MSE will be zero, since $\hat{h}$ matches h perfectly.

(b) 30 points. Now suppose that the switch is moved to position 2. Before any adaptation of your LMS adapted filter occurs, what is the MSE? What will $\hat{\boldsymbol{h}}$ be after convergence of the LMS algorithm? What will the MSE be after convergence?

Before adaptation: $\hat{h} = [1,2,0]$ and $g * h = [1,3,2]$

$MSE = (1-1)^2 \cdot 1 + (2-3)^2 \cdot 1 + (0-2)^2 \cdot 1 = 5$

After adaptation: $\hat{h} = [1,3,2]$ and the MSE = 0.

4. 50 points total. Consider the C6713 C-callable assembly function on the next page. This function computes computes the dot product of two arrays of single-precision floats. The function prototype is given as

```
float dotproduct2(float*, float*, int);
```

(a) 25 points. Suppose $N = 10$. How many total cycles does it take for this function to execute? Include all cycles from the beginning of the function through the NOP 2 instruction at the end and explain your result.

prologue : 1 cycle
loop : 10 cycles
epilogue : 6 cycles

$\Rightarrow$ total cycles = 107

(b) 25 points. Suggest at least one way to improve the speed of this function. You don't need to write any code but you must be specific.

Double word loads would fetch 4 single-precision floats simultaneously. This would allow us to use both multipliers, both adders, and cut the number of loops required in half.

Everything else being kept the same, this would result in a savings of 50 cycles.

```
1   ; semi-parallel assembly code for N element floating point dot product
2   ; A4 and B4 contain pointers to arrays of floats
3   ; A6 contains an integer with N
4           .def    _dotproduct2
5
6   _dotproduct2:
7                   MV              .S1     A6, A1          ; copy count to register A1
8   ||              ZERO            .L1     A8              ; zero accumulator
9   LOOP:
10                  LDW             .D1     *A4++, A2       ; get element from first array
11  ||              LDW             .D2     *B4++, B2       ; get element from second array
12                  SUB             .S1     A1, 1, A1       ; decrement counter
13                  NOP 2                                   ; wait for data
14          [A1]    B               .S2     LOOP            ; conditional branch
15                  MPYSP           .M1x    A2, B2, A7      ; multiply
16                  NOP 3                                   ; wait for result
17                  ADDSP           .L1     A7, A8, A8      ; accumulate
18                  ; conditional branch occurs here
19                  B               B3                      ; branch back to calling function
20                  NOP 2
21                  MV              .S1     A8, A4          ; put result in proper register
22                  NOP 2
23
24          .end
```