ECE4703 Final Exam

Your Name: SOLUTION Your box #: ________

December 15, 2011

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.

problem 1	problem 2	problem 3	problem 4	total final exam score
40 points	50 points	60 points	50 points	200 points

1. 40 points total. Suppose you write a frame-based RT-DSP program processes a buffer of N input samples and lights up an LED if a certain type of signal is detected in the buffer. After profiling the code for various values of N, you determine that the number of cycles to process the length-N buffer follows the trend

$$\text{CPU cycles} = 90 + 6N + 3N^2.$$

(a) 10 points. If "CPU cycles" = "operations", what is the asymptotic complexity of your algorithm?

The asymptotic complexity is $\Theta(N^2)$ since

$$\lim_{N \to \infty} \frac{90 + 6N + 3N^2}{N^2} = 3 > 0$$

↑ not a function of N.

(b) 30 points. If your sampling rate is $f_s = 8000$Hz, your DSP clock rate is 225MHz, and all processing is performed on a frame-by-frame basis, how large can N be before your program will no longer run in real-time? Explain your answer. Hint: You may derive an approximate solution by assuming N is large enough such that $\underline{\text{CPU cycles} \approx 3N^2}$.

In general, we require

$$\frac{\text{CPU cycles}}{225 \times 10^6} \leq \frac{N}{f_s}$$

We will use the approximation to make the analysis easier.

$$\frac{3N^2}{225 \times 10^6} \leq \frac{N}{8000}$$

$$24000\, N \leq 225 \times 10^6$$

$$\boxed{N \leq \frac{225 \times 10^6}{24000} = 9375}$$

The exact solution (which requires finding the roots of a quadratic equation) is $N \leq 9373$ which is pretty close to the approximation.

2. 50 points. Suppose you have an array $\boldsymbol{x} = [1, 2]$ and another array $\boldsymbol{y} = [3, 4]$.

(a) 10 points. Compute $\boldsymbol{z} = \boldsymbol{x} \star \boldsymbol{y}$ where $\star$ denotes linear convolution.

$$z = [3, 10, 8]$$

(b) 20 points. Compute $\boldsymbol{X} = \text{FFT}_2(\boldsymbol{x})$, $\boldsymbol{Y} = \text{FFT}_2(\boldsymbol{y})$, $\boldsymbol{Z} = \boldsymbol{X} \cdot \boldsymbol{Y}$ where $\cdot$ is an element-wise product, and $\tilde{\boldsymbol{z}} = \text{IFFT}_2(\boldsymbol{Z})$. Explain why $\tilde{\boldsymbol{z}}$ is not the same as the $\boldsymbol{z}$ in part (a).

Two-point FFTs are easy.

$X[0] = x_0 + x_1 = 3 \qquad Y[0] = y_0 + y_1 = 7$

$X[1] = x_0 - x_1 = -1 \qquad Y[1] = y_0 - y_1 = -1$

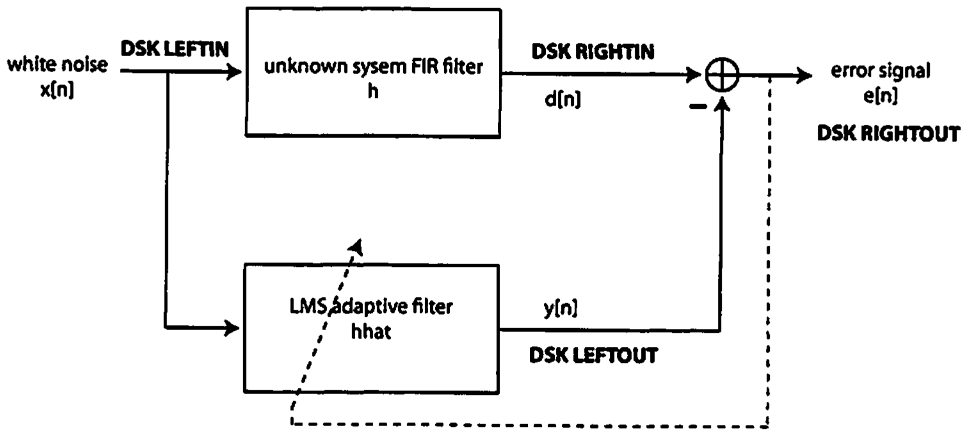
$$\left.\begin{array}{l} Z[0] = X[0]\,Y[0] = 21 \\ Z[1] = X[1]\,Y[1] = 1 \end{array}\right\} \xrightarrow{\text{IFFT}} \begin{array}{l} \tilde{z}_0 = \frac{1}{2}\left(Z[0] + Z[1]\right) = 11 \\ \tilde{z}_1 = \frac{1}{2}\left(Z[0] - Z[1]\right) = 10 \end{array}$$

$\tilde{z} \neq z$ because $\tilde{z} = x \circledast y$ where $\circledast$ is circular convolution, not linear convolution.

(c) 20 points. Describe how you would modify the procedure in part (b) so that the final result $\tilde{\boldsymbol{z}} = \boldsymbol{z}$. You do not need to perform all of the steps, but be specific.

1. zero pad x so that $\bar{x} = [1, 2, 0, 0]$
2. zero pad y so that $\bar{y} = [3, 4, 0, 0]$
3. compute $\bar{X} = \text{FFT}_4(\bar{x})$
4. compute $\bar{Y} = \text{FFT}_4(\bar{y})$
5. compute $\bar{Z} = \bar{X} \cdot \bar{Y}$ (element-wise product)
6. compute $\bar{z} = \text{IFFT}_4(\bar{Z})$
   this will be equal to $[3, 10, 8, 0]$
7. Discard the last element, yielding $\tilde{\bar{z}} = [3, 10, 8] = z$.

3. 60 points total. Consider the system identification adaptive filtering system shown below.



For the following questions, assume that

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

(a) 20 points. Suppose you write correct code for the LMS algorithm and let it run for a while. What will $\hat{\boldsymbol{h}}$ be when you halt execution?

$$\hat{h} = [1, -2, 1, 0, 0]$$

This will result in $y[n] = d[n]$ and $e[n] = 0$.

continued...

(b) 20 points. Recall that $e[n] = d[n] - y[n]$. Suppose you accidentally write your LMS code to compute $e[n] = y[n] - d[n]$ but your LMS code is otherwise correct. What will happen?

you are effectively using $-e[n]$ in the LMS update

$\underline{\hat{h}}[n+1] = \underline{\hat{h}}[n] + \mu e[n] \underline{x}[n]$ ← correct LMS update

$\underline{\hat{h}}[n+1] = \underline{\hat{h}}[n] - \mu e[n] \underline{x}[n]$ ← your LMS update.

so instead of doing "approximate gradient descent" you are doing "approximate gradient <u>ascent</u>".

In this case $\underline{\hat{h}}$ will just go off in some random direction toward infinity and the coefficients will not converge

(c) 20 points. Suppose you accidentally write your LMS code to compute $e[n] = x[n] - y[n]$, e.g. you accidentally swapped your input channels, but your LMS code is otherwise correct. What will happen?

The LMS filter is going to try to minimize the MSE, which is now defined as

$$E\{e^2[n]\} = E\{(x[n] - y[n])^2\}$$

i.e., the LMS adaptive filter output $y[n]$ will become like $x[n]$ to make the MSE small.

This is easy to do since $\underline{\hat{h}} = [1, 0, 0, 0, 0]$ causes $y[n] = x[n]$ (the LMS adapted filter input and output are the same). Hence MSE → 0 and $\underline{\hat{h}} \rightarrow [1, 0, 0, 0, 0]$ will be the final coefficients for the LMS adapted filter after it converges.

4. 50 points total. Consider the C6713 C-callable assembly function on the next page. This function computes $y = ax^2 + bx + c$ where all parameters a, b, c, x are floats and passed into the function through the function call with prototype

```
float q(float a, float b, float c, float x)
```

(a) 10 points. How many total cycles does it take for this function to execute? Include all cycles from the beginning of the function through the NOP 5 instruction at the end.

22 cycles

(b) 40 points. Rewrite this C-callable assembly function to improve its efficiency while still producing the correct result. Credit will be awarded based on the number of cycles in your ASM function. Your answer must be orderly to receive full credit.

Here is one possibility:

```
_q:     MPYSP   .M2     B6,B6,B7      ; B7 = x^2
        MPYSP   .M2     B4,B6,B8      ; B8 = bx
        NOP 2                         ; wait for B7 to become valid
        MPYSP   .M1x    B7,A4,A7      ; A7 = ax^2
        ADDSP   .L1x    B8,A6,A4      ; A4 = bx+c
        NOP 3                         ; wait for A4 and A7 to become valid
        ADDSP   .L1     A4,A7,A4      ; A4 = ax^2+bx+c
        B               B3            ; return
        NOP             5             ; branch pad
```

This takes 16 cycles. Note that the final ADDSP becomes valid during the NOPs after the branch instruction, which is fine. We could actually start the branch before the final ADDSP to save a few more cycles.

```
                .def    _q

                ; this function computes y = ax^2+bx+c
                ; input parameters
                ;   A4 = a
                ;   B4 = b
                ;   A6 = c
                ;   B6 = x
                ; output parameters
                ;   A4 = y

1     _q:       MPYSP   .M2     B6, B6, B7   ; compute x^2
3               NOP             3
1               MPYSP   .M1x    A4, B7, A5   ; compute A5 = ax^2
      ||        MPYSP   .M2     B4, B6, B8   ; compute B8 = bx
3               NOP     3
1               ADDSP   .L1x    A5, B8, A8   ; compute A8 = ax^2+bx
3               NOP     3
1               ADDSP   .L1     A8, A6, A4   ; compute A4 = ax^2+bx+c
3               NOP             3
1               B               B3           ; branch back to calling function
5               NOP             5

                .end
```