ECE4703 Final Exam

Your Name: SOLUTION Your box #: _____

December 19, 2013

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! Points may be deducted for a disorderly presentation of your solution.
- 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
30 points	40 points	60 points	70 points	200 points

1. 30 points. A "bubble sort" is an algorithm that sorts a list of numbers in ascending order by going through the list, comparing neighboring elements, and swapping them if the second item is smaller than the first. It is performed iteratively by making passes through the list until no more swaps occur. Pseudocode for a bubble sort (from Wikipedia) is given below.

```
procedure bubbleSort( A : list of sortable items )
   repeat
     swapped = false
     for i = 1 to length(A) - 1 inclusive do:
       /* if this pair is out of order */
       if A[i-1] > A[i] then
         /* swap them and remember something changed */
         swap( A[i-1], A[i] )
         swapped = true
       end if
     end for
   until not swapped
end procedure
```

As an example, consider the unsorted list $A = \{5, 3, 1, 2\}$. The following table shows how the bubble sort works on this list.

Pass	Current list	Compare	Swap?	New list
1	$\{\mathbf{5,3},1,2\}$	$A[0] > A[1]$?	yes	$\{\mathbf{3,5},1,2\}$
	$\{3,\mathbf{5,1},2\}$	$A[1] > A[2]$?	yes	$\{3,\mathbf{1,5},2\}$
	$\{3,1,\mathbf{5,2}\}$	$A[2] > A[3]$?	yes	$\{3,1,\mathbf{2,5}\}$
2	$\{\mathbf{3,1},2,5\}$	$A[0] > A[1]$?	yes	$\{\mathbf{1,3},2,5\}$
	$\{1,\mathbf{3,2},5\}$	$A[1] > A[2]$?	yes	$\{1,\mathbf{2,3},5\}$
	$\{1,2,\mathbf{3,5}\}$	$A[2] > A[3]$?	no	unchanged
3	$\{\mathbf{1,2},3,5\}$	$A[0] > A[1]$?	no	unchanged
	$\{1,\mathbf{2,3},5\}$	$A[1] > A[2]$?	no	unchanged
	$\{1,2,\mathbf{3,5}\}$	$A[2] > A[3]$?	no	unchanged

Counting each compare as an "operation" and denoting the list length as N, it is clear that the *best-case* asymptotic complexity is $\mathcal{O}(N)$ since $N-1$ compares must be performed even if the list is pre-sorted. What is the *worst-case* asymptotic complexity of the bubble sort algorithm for an unsorted list? Explain (use the back of this page if necessary).

Worst-case occurs when the smallest element is at the end. This then requires N-1 passes, each pass requiring N-1 compares.

Total compares (worst case) $= (N-1)^2$

Asymptotic complexity $= \Theta(N^2)$ (worst case)

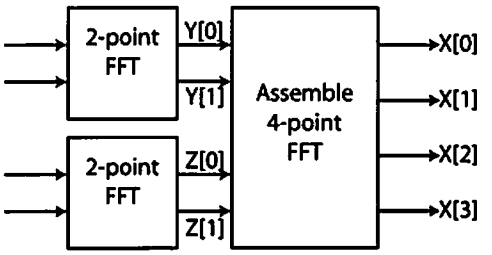
2. 40 points total. Suppose you wish to compute the 4-point FFT of the array

$$x = \{x[0], x[1], x[2], x[3]\}.$$

The output of the FFT is denoted as

$$X = \text{FFT}(x) = \{X[0], X[1], X[2], X[3]\}.$$

As shown in the figure below, the 4-point FFT is implemented by calling a 2-point FFT function twice. Denote the output of the first 2-point FFT call as $Y = \{Y[0], Y[1]\}$ and the output of the second 2-point FFT call as $Z = \{Z[0], Z[1]\}$.

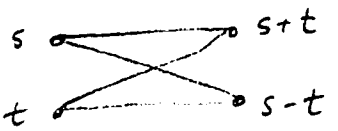


Given

$$\begin{aligned} x[0] &= a \\ x[1] &= b \\ x[2] &= c \\ x[3] &= d, \end{aligned}$$

compute $Y[0], Y[1], Z[0], Z[1], X[0], X[1], X[2]$, and $X[3]$. Show your work and explain your reasoning.

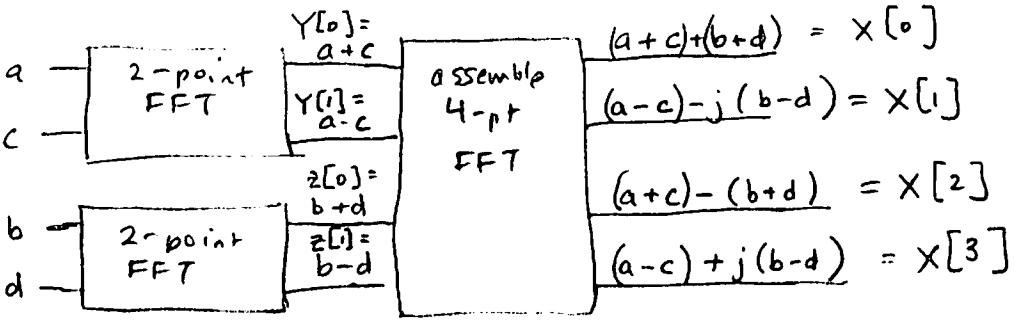
two-point FFT:



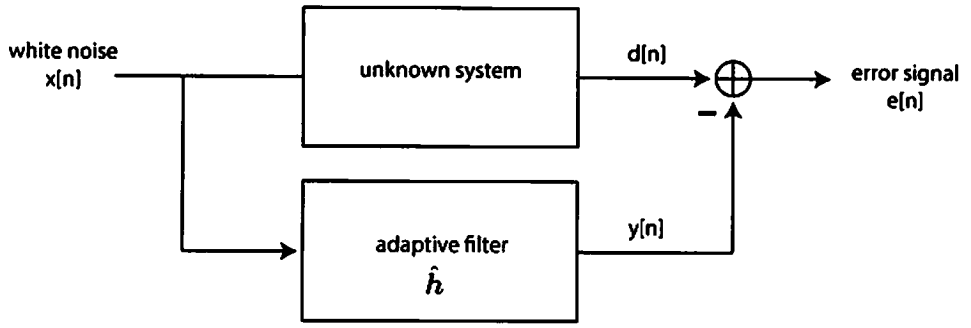
The first two-point FFT would be for the even elements, i.e. $x[0]$ and $x[2]$

The second two-point FFT would be for the odd elements ie $x[1]$ and $x[3]$

so



3. 60 points total. Consider the system identification problem shown below.



Suppose you collect a large amount of input/output data and compute the following auto- and cross-correlations:

- $r_{k,\ell} = \mathrm{E}\{x[n-k]x[n-\ell]\} = \begin{cases} 0.1 & k = \ell \\ 0 & \text{otherwise} \end{cases}$
- $p_k = \mathrm{E}\{x[n-k]d[n]\} = \begin{cases} 0.2 & k = 0 \\ \frac{1}{k} & k = 1, \ldots, 5 \\ 0 & \text{otherwise} \end{cases}$

(a) 20 points. Determine the minimum mean squared error (MMSE) adaptive filter $\hat{h}$ assuming $\hat{h}$ is a scalar (a single coefficient).

If $\hat{h}$ has only one coefficient, then

$$\hat{h}_{MMSE} = (r_{0,0})^{-1} p_0 = \frac{1}{0.1} \cdot 0.2 = 2.$$

$$\boxed{\hat{h}_{MMSE} = 2}$$

(b) 20 points. Find an adaptive filter $\hat{h}$ that causes the mean squared error to become zero.

We need at least 6 coefficients

$$\hat{h}_{MMSE} = \begin{bmatrix} 0.1 & & \\ & \ddots & \\ & & 0.1 \end{bmatrix}^{-1} \begin{bmatrix} 0.2 \\ 1 \\ 1/2 \\ 1/3 \\ 1/4 \\ 1/5 \end{bmatrix}$$

$$= \begin{bmatrix} 10 & & \\ & \ddots & \\ & & 10 \end{bmatrix} \begin{bmatrix} 0.2 \\ 1 \\ 1/2 \\ 1/3 \\ 1/4 \\ 1/5 \end{bmatrix} = \begin{bmatrix} 2 \\ 10 \\ 5 \\ 10/3 \\ 2.5 \\ 2 \end{bmatrix}$$

$$\hat{h}_{MMSE} = \left[2,\ 10,\ 5,\ \frac{10}{3},\ 2.5,\ 2\right]$$

(c) 20 points. Suppose you initialize an LMS-adapted filter exactly at the MMSE solution, i.e., $\hat{h}[0] = \hat{h}_{\text{MMSE}}$. What will happen in the case of part (a) and part (b)? Explain.

In part (a), since the error term $e[n] = d[n] - y[n]$ will not be zero (the system is undermodeled), the LMS-adapted filter will "bounce around" the MMSE solution.

In part (b), since the system is exactly modeled, $e[n] = 0$ for all n and the LMS-adapted filter will not change.

4. 70 points total. Consider the C6713 C-callable assembly function on the last page. The function prototype is given as

```
float doit(float, float);
```

(a) 20 points. Suppose the global variables $x = 5$, $y = 2$, and the function is called as `z = doit(3,4)`. What value does the function return?

A4 = 3
B4 = 4

A1 = 5 when X is loaded

A2 = 2 when Y is loaded

A2 is then squared twice → A2 = 16

A1 × A4 → A5 : A5 = 5 × 3 = 15

A5 + B4 → A4 : A4 = 15 + 4 = 19

A4 + A2 → A4 : A4 = 19 + 16 = 35

A4 is returned, hence the function returns the value $\boxed{z = 35}$

(b) 20 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 and explain your result.

$\boxed{40 \text{ cycles}}$ (nothing in parallel, so its pretty easy to just count)

(c) 30 points. Rewrite this function to save at least 10 cycles and have the same functionality.

lines 1–10 dont change

★ idea #1 load x and y simultaneously

```
      MVKL .S1 -x, A0
||    MVKL .S2 -y, B0
      MVKH .S1 -x, A0
||    MVKH .S2 -y, B0
      LDW  .D1 *A0, A1
||    LDW  .D2 *B0, B1
      NOP  4
```

This replaces lines 11–18

7 cycles total (used to require 14)

y is now in B1

```
      MPYSP  .M2  B1, B1, B1
||    MPYSP  .M1  A1, A4, A5
      NOP 3
      MPYSP  .M2  B1, B1, B1
||    ADDSP  .L1  A5, B4, A4
      NOP 3
```

★ idea #2: multiply y while doing other multiply/adds

this replaces lines 19–26

8 cycles total (used to require 16)

```
      ADDSP  .L1x A4, B1, A4
      NOP 3
      B   B3
      NOP 5
```

Same # of cycles as lines 27–30 (could have moved the branch up to save cycles too).

Saved 15 cycles ✓

```
1   ; float doit(float a, float b)
2   ; a -> A4 = 3
3   ; b -> B4 = 4
4
5           .def    _linear
6           .ref    _x                  ; refer to global variable x (float)
7           .ref    _y                  ; refer to global variable y (float)
8
9   _doit:
10
11          MVKL    .S1     _x, A0      ; put the address of the global variable x in A0
12          MVKH    .S1     _x, A0
13          LDW     .D1     *A0, (A1)   ; load x into A1        A1 = 5
14          NOP     4                   ; wait for result
15          MVKL    .S1     _y, A0      ; put the address of the global variable y in A0
16          MVKH    .S1     _y, A0
17          LDW     .D1     *A0, (A2)   ; load y into A2        A2 = 2
18          NOP     4                   ; wait for result
19          MPYSP   .M1     A2,A2,A2    ; multiply              A2 = 4
20          NOP     3                   ; wait for result
21          MPYSP   .M1     A2,A2,A2    ; multiply              A2 = 16
22          NOP     3                   ; wait for result
23          MPYSP   .M1     A1,A4,A5    ; multiply              5 x 3 = 15  (A5 = 15)
24          NOP     3                   ; wait for result
25          ADDSP   .L1     A5,B4,A4    ; add                   15 x 4 = 60  (A4 = 60)
26          NOP     3                   ; wait for result
27          ADDSP   .L1     A4,A2,A4    ; add                   60 + 16 = 76  (A4 = 76)
28          NOP     3                   ; wait for result
29          B       B3                  ; branch back to calling function
30          NOP     5
31
32          .end
```