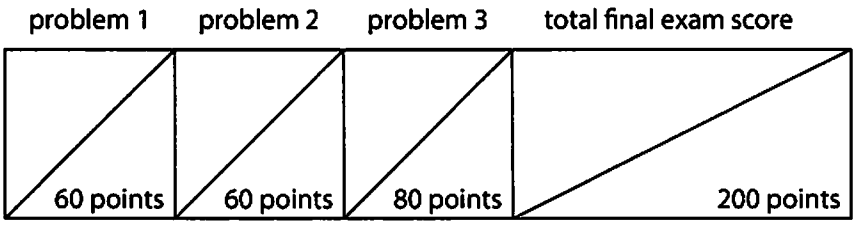
ECE4703 Final Exam

Your Name: SOLUTION Your box #: _____

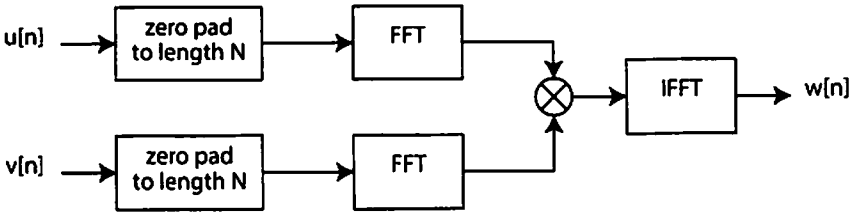
December 17, 2015

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.



1. 60 points total. Suppose we wish to linearly convolve two discrete-time sequences, each of length M. The output of the linear convolution will have length $2M-1$. This convolution can be computed in two ways: "conventional convolution" or "fast convolution". The block diagram below shows the fast convolution system.



(a) 20 points. Why is the zero padding step necessary? What happens if we don't zero pad?

We zero pad so this performs linear convolution. If we don't zero pad, we will get circular convolution which results in temporal aliasing

(b) 20 points. Suppose $M = 3$ with $u[n] = [1, 2, 3]$ and $v[n] = [1, 1, 1]$. Assuming the FFT and IFFT only work with values of $N \in \{2, 4, 8, 16, \dots\}$, specify an appropriate value for N and compute $w[n]$. Note that $w[n]$ will have length N.

We need $N \geq 8$ here since N must be $\geq 2M-1 = 5$.

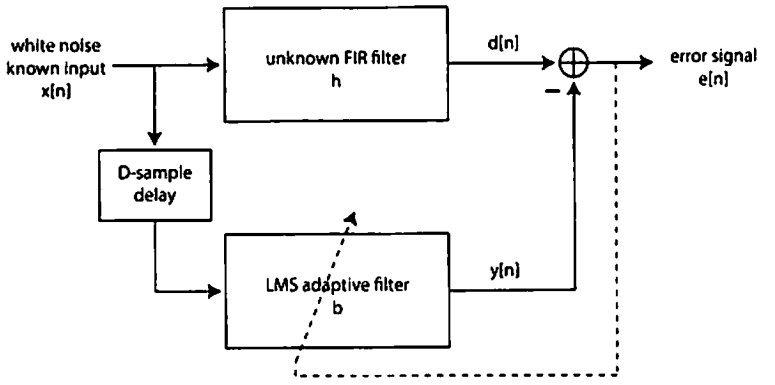
If $N=8$ then we effectively get linear convolution with some zeros on the end. Hence

$w[n] = [1, 3, 6, 5, 3, 0, 0, 0]$

(c) 20 points. Suppose "fast convolution" were implemented with DFT/IDFT rather than FFT/IFFT as shown above. What would change? What would be the same?

The output would be the same since the DFT and FFT generate the same results, but the complexity would be higher. The DFT has asymptotic complexity $\Theta(N^2)$ whereas the FFT has a.c. $\Theta(N \log N)$. If "fast convolution" were implemented with a DFT/IDFT, it probably wouldn't be any faster than conventional convolution.

2. 60 points total. Consider the system identification adaptive filtering system shown below.



For the following questions, assume that

- the mean squared value of the input noise $x[n]$ is one, i.e. $\mathrm{E}\left[x^2[n]\right] = 1$,
- the unknown filter coefficients are $\boldsymbol{h} = [0.2, 1.0, 0.3]$,
- the LMS adaptive filter $\boldsymbol{b}$ coefficients 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 that $D = 0$ (no delay) and that $\boldsymbol{b}$ has three coefficients. What will $\boldsymbol{b}$ be after convergence of the LMS algorithm? What will the MMSE be after convergence?

b will simply converge to h since this causes the MSE → 0.

Hence $b \to [0.2, 1.0, 0.3]$ and the MMSE = 0.

(b) 20 points. Now suppose $D = 0$ (no delay) and that $\boldsymbol{b}$ has **two** coefficients. What will $\boldsymbol{b}$ be after convergence of the LMS algorithm? What will the MMSE be after convergence?

This is an undermodeled scenario. b will converge to match the first two taps of h: $b \to [0.2, 1.0]$

$e[n] = 0.3\,x[n-3]$ $\quad MSE = E[e^2[n]] = 0.09 \cdot E[x^2[n-3]] = \underline{\underline{0.09}}$

(c) 20 points. For the case when $\boldsymbol{b}$ has **two** coefficients, find the value of D that leads to the lowest possible MMSE. For this value of D, what will $\boldsymbol{b}$ be after convergence of the LMS algorithm? For this value of D, what will the MMSE be after convergence?

If we set $D=1$, the LMS filter will converge to match the 2nd and 3rd coefficients of h. Then $b \to [1.0, 0.3]$

$e[n] = 0.2\,x[n] \Rightarrow MSE = E[e^2[n]] = \underline{\underline{0.04}}$

3. 80 points total. Suppose your DSP is running the assembly code given on the last page of this exam (this is the fully-pipelined dot product discussed in lecture).

(a) 10 points. Write a C prototype for this function.

float dotproduct2 (float *x, float *y, int N)

✓(b) 10 points. Draw a box around the instruction(s) in the second execute packet. Label it EP2. (Lines 10-11)

(c) 10 points. Suppose the SUB instruction on line 12 is currently in pipeline stage E1. Put a pound sign (#) next to any instruction(s) currently in pipeline stage E2. If there are no such instructions, explain below.

Lines 10-11 are in pipeline stage E2.
These LDW instructions have not completed yet.

(d) 10 points. Notice the "???" on line 16. What value should be here?

NOP 3 since MPYSP has 3 delay slots and the result of the MPYSP is needed by the ADDSP on line 17.

(e) 20 points. How many cycles does this function require to execute as a function of N (including the cycles required to execute the final branch back to the calling function)?

prolog: 1 cycle
loop: 10 cycles
epilog: 6 cycles

} total cycles = $7+10N$

(f) 10 points. If we use total cycles as a measure of complexity, what is the asymptotic complexity of this function?

$$\lim_{N\to\infty} \frac{\text{total cycles}}{N} = \frac{7+10N}{N} = 10 > 0$$

hence the asymptotic complexity is $\Theta(N)$

(g) 10 points. Suggest at least one way you could make this function faster. You do not need to write any code but be specific.

The dot product can be fully pipelined by doing double word loads (LDDW) and doing half the multiply accumulates on the A side while doing the other half on the B side.

The fully pipelined code has a single-cycle loop and the total cycles scales like $C + \frac{1}{2}N$

(C: total prolog & epilog cycles; $\frac{1}{2}N$: two MACs per loop)

```
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 ???                                 ; 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
```

EP2