Name: SOLUTIONS
Perm #:

Lab Section TA:

ECE 152A-Fall 2004

Prof. Volkan Rodoplu

FINAL EXAMINATION

INSTRUCTIONS:

1. READ THIS PAGE THOROUGHLY WHEN YOU RECEIVE IT, BUT DO NOT START TURNING TO THE OTHER PAGES UNTIL YOU ARE INSTRUCTED TO DO SO.

2. WHENEVER INDICATED, YOU MUST WRITE YOUR ANSWERS ON THE ANSWER LINES PROVIDED IN CERTAIN PROBLEMS. NO PARTIAL CREDIT WILL BE GIVEN ON THESE PROBLEMS.

3. On other problems, PARTIAL CREDIT will be given only to true statements that make progress towards the correct answer. Partial credit may be given to correct reasoning in developing structures such as K-maps, truth tables, state diagrams and state tables in which the variables are clearly labeled. NO partial credit will be given for incorrect statements or statements to which no truth value can be assigned (such as a bunch of numbers or algebraic expressions). NO partial credit will be given for statements that use symbols that the problem statement or you have not defined. NO credit will be given for any work that is not clearly labeled with the part and problem number to which this work provides an answer.

4. All the exam rules in the course syllabus apply to this exam.

5. You may remove the staple from the exam pages, if is more convenient. We will provide a stapler at the end of the exam.

6. There are 30 pages in this exam (printed on two sides). When you are instructed to start, first check that you have all the pages.

7. There is a total of 100 points on this exam.

8. Write the solution of each problem in the space provided for that problem.

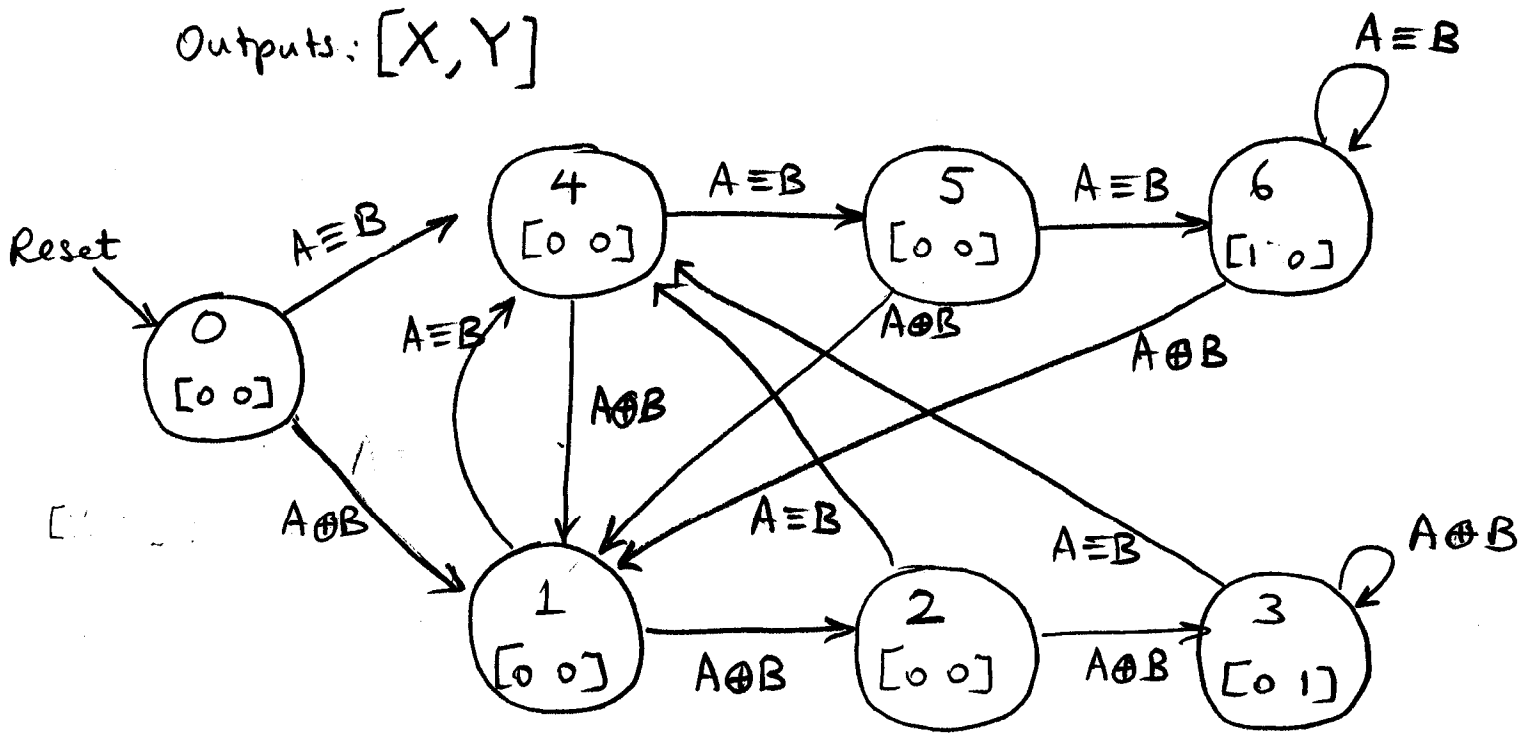
PROBLEM 1: FINITE STATE MACHINE DESIGN (20 points)

Construct a synchronous MOORE machine with two inputs, A and B, and two outputs, X and Y. The machine accepts data on two input lines synchronously with the clock. The output X is 1 if and only if the data on the two input lines have been identical (i.e. A and B are both 1, or A and B are both 0) for the last three or more consecutive clock cycles. Output Y is 1 if and only if the data on the two input lines have been complements of each other (i.e. A = 1 and B = 0, or A = 0 and B = 1) for the last three or more consecutive clock cycles. Assume that the machine has a synchronous Reset.

(a) **(10 points)** Draw a state diagram for this machine. Define the state variables for this machine, and explain the meaning of each state; that is, in English, what does each state correspond to, or represent? All of the transition arcs must be drawn and fully labeled (except that it is fine to show Reset with a single arrow into that state.) All variables used must be defined, if they are not defined in the problem statement. The number of states should not exceed 8.

Inputs : A, B

Outputs : [X, Y]



Meaning of states:

State 0 : Start/reset state

state 1 : A and B have been different once

state 4 : " " " " " same "

state 2 : " " different twice

state 5 : " " same "

state 3 : " " different thrice or more

state 6 : " " same " or more.

(b) **(10 points)** Implement this machine as a **Verilog module** using an edge-triggered design (i.e. a design based on flip-flops).

```
module Machine(A,B,X,Y,CLK,Reset);
    input A,B; CLK; Reset;
    output X,Y;
    reg[2:0] Q, Q_next;
    parameter s0 = 3'b000, s1 = 3'b001;
        s2 = 3'b010, s3 = 3'b011, s4 = 3'b100,
        s5 = 3'b101, s6 = 3'b110;

    always @ (posedge CLK)
        if (Reset)
            Q <= s0;
        else
            Q <= Q_next;

    // next-state combinational logic
    always @ (Q or A or B)
        case(Q)
            s0 : if (A==B) Q_next <= s4;
                 else Q_next <= s1;
            s1 : if (A==B) Q_next <= s4;
                 else Q_next <= s2;
            s2 : if (A==B) Q_next <= s4;
                 else Q_next <= s3;
```

```
        s3: if (A == B) Q_next <= s4;
            else  Q_next <= s3;
        s4: if(A == B) Q_next <= s5;
            else  Q_next <= s1;
        s5: if (A == B)  Q_next <= s6;
            else  Q_next <= s1;
        s6:  if (A == B)  Q_next <= s6;
            else  Q_next <= s1;
        default:  Q_next <= s0;
    endcase
//output logic:
  assign  X = Q[2] & Q[1] & ~Q[0];      //s6
  assign  Y = ~Q[2] & Q[1] & Q[0];      //s3

endmodule
```

PROBLEM 2: VERILOG (20 points)

The sections of this problem will be graded for correctness as well as style. We look for correct code that is simple, clean, synthesizable, and that uses the Verilog language well.

(a) **(4 points)** Write a complete Verilog module for a D flip-flop with synchronous, active-high Reset. Use behavioral Verilog. (An implementation at the gate level is not acceptable.) The D flip-flop should output both Q and Q_bar (the complement of Q).

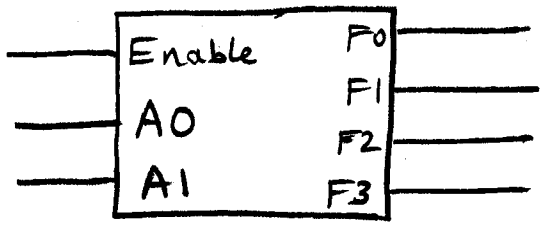
```
module D_flipflop(Reset, CLK, D, Q, Q_bar);
    input Reset, CLK, D;
    output Q, Q_bar;

    reg Q, Q_bar;

    always@(posedge CLK)
        if(Reset)
            begin
                Q<=0;
                Q_bar <=1;
            end
        else
            begin
                Q <= D;
                Q_bar <= ~D;
            end

endmodule
```

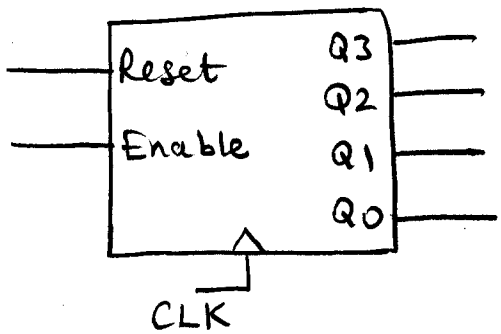
(b) **(6 points)** Write a complete Verilog module for a "2:4 decoder with Enable", **using procedural assignments in a 'case' or a 'casex' statement**. This decoder has an Enable input. If Enable is not asserted, the decoder outputs the zero vector. If Enable is asserted, the decoder decodes as usual.




```
module Decoder2_4 (Enable, A, F);
    input [1:0] A;
    input Enable;
    output [3:0] F;
    reg [3:0] F;
    always@ (Enable or A)
        case ({Enable, A})
            3'b100 :  F = 4'b0001;
            3'b101 :  F = 4'b0010;
            3'b110 :  F = 4'b0100;
            3'b111 :  F = 4'b1000;
            default :  F = 4'b0000;
        endcase
endmodule
```

<u>Note</u>: Using casex with 3'b0xx in the place of default is fine, too.

(c) **(10 points)** Draw the interface schematic and write a complete Verilog module for a synchronous, edge-triggered BCD down-counter that counts down from 9 to 0 and wraps around. This counter has a synchronous Reset that sets the counter to 0, as well as a synchronous Enable input. (Reset, Enable, CLK are the only inputs.) Enable overrides Reset; that is, if Enable == 0, then the counter holds its value (regardless of the value of Reset), and if Enable == 1, and Reset == 1, the counter resets.
You must use behavioral Verilog: your solution must solve the problem at the right level of abstraction without designing in detail the state diagram or a gate-level description of the counter. (Hint: Make use of the addition and subtraction operations in Verilog.)




```
module BCD_DownCounter (Reset, Enable, CLK, Q);
    input Reset, Enable, CLK;
    output [3:0] Q;
    reg [3:0] Q;

    always @ (posedge CLK)
        if (Enable)
            if (Reset)
                Q <= 4'b0000;
            else if (Q == 4'b0000)
                Q <= 4'b1001;
            else
                Q <= Q - 1;

endmodule
```

PROBLEM 3: TIMING OF SEQUENTIAL CIRCUITS (17 points)

NO PARTIAL CREDIT will be given on this problem. Your answer must be written clearly on the lines indicated. We will check both your answer and your work. If your answer is incorrect, you will get no credit. If your answer is correct, but your derivation is wrong, you will get no credit. Use the following data in this problem:

D flip-flop:

Set-up time: 2.3 ns
Hold time: 1.1 ns
CLK-to-Q propagation delay: 3.0 ns

T flip-flop:

Set-up time: 3.1 ns
Hold time: 0.3 ns
CLK-to-Q propagation delay: 1.5 ns

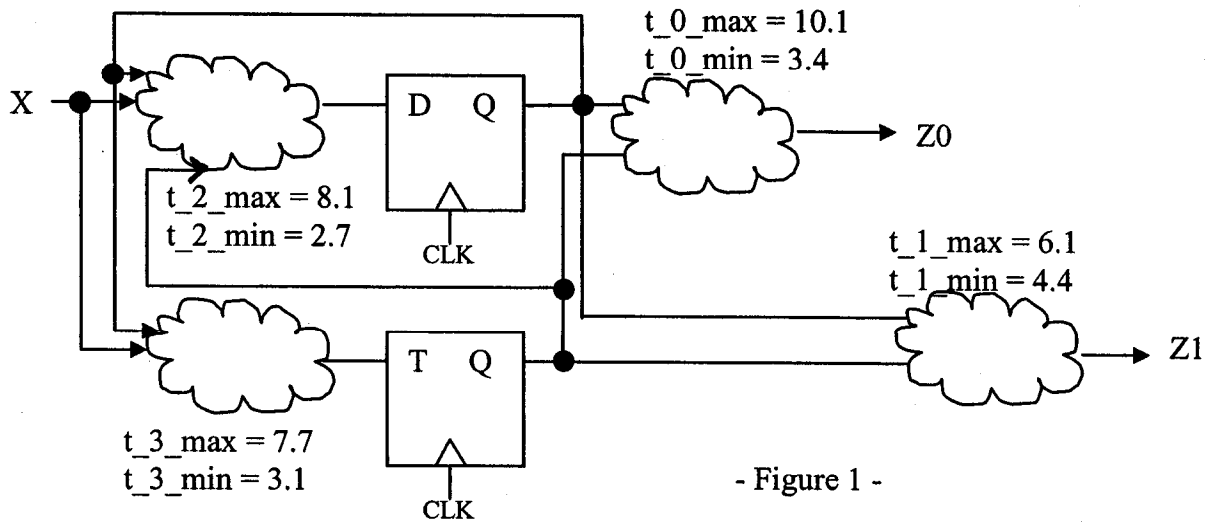
Clocked D latch:

Set-up time: 1.2 ns
Hold time: 0.5 ns
CLK-to-Q propagation delay: 1.3 ns
D-to-Q propagation delay: 1.8 ns

This problem has three parts : A, B, and C. These parts are not directly related to each other and can be done separately.

Problem 3-Part A:

In Figure 1, combinational logic blocks are shown as bubbles. This abstracts away the particular gates in the combinational logic block, and we assume that the minimum and maximum delay of each combinational logic block have already been computed. Note that the upper memory element is a D **flip-flop,** and the lower memory element is a **T flip-flop** in this circuit. (The Reset for the T flip-flop is not shown.) Assume that there is no clock skew (i.e. the clock edge arrives at the same time at all flip-flops). In this circuit, X is the only input, and Z0 and Z1 are the only outputs. All delays shown are in nanoseconds.



- Figure 1 -

(a) **(1 point)** Is the machine in Figure 1 a Mealy or Moore machine? (No explanation is needed.)

MOORE

(b) **(5 points)** Find the minimum clock period at which the machine in Figure 1 operates correctly.

Answer: 13.8 ns.

Show your work here (and on the next page, if needed):

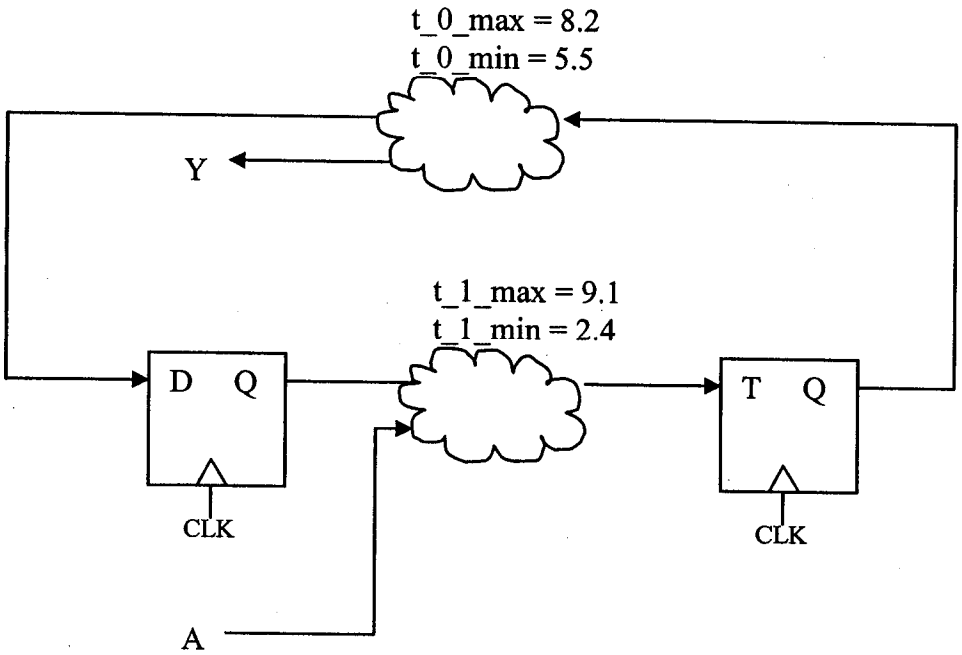
There are 3 loops to consider. The maximum delay loop determines the minimum clock period.

$$\max\{ t_{CLK\to Q}^{(D-ff)} + t_2_max + t_{su}^{(D-ff)},$$
$$t_{CLK\to Q}^{(D-ff)} + t_3_max + t_{su}^{(T-ff)},$$
$$t_{CLK\to Q}^{(T-ff)} + t_2_max + t_{su}^{(D-ff)} \}$$

$$= \max\{ 3.0 + 8.1 + 2.3,$$
$$3.0 + 7.7 + 3.1,$$
$$1.5 + 8.1 + 2.3\} = \max\{13.4, 13.8, 11.9\}$$

$$= \boxed{13.8}$$

Problem 3-Part B:
It is sometimes advantageous to divide up the combinational logic and distribute it between the flip-flops in sequential design. This way, a circuit can be run at a higher clock frequency. Figure 2 shows a circuit that has been designed using such a methodology. In this circuit, A is the only input and Y is the only output. Note that the memory element on the left is a D **flip-flop** and the memory element on the right is a **T flip-flop** in this circuit.



- Figure 2 -

(a) **(1 point)** Is the machine in Figure 2 a Mealy or Moore machine? (No explanation is needed.)

MOORE

(b) **(4 points)** Find the minimum clock period at which the machine in Figure 2 operates correctly.

Answer: 15.2 ns.

Show your work here (and on the next page, if needed):

There are two paths to consider.

$$\max\{ t_{CLK\to Q}^{(D-ff)} + t_1_max + t_{su}^{(T-ff)},\ t_{CLK\to Q}^{(T-ff)} + t_0_max + t_{su}^{(D-ff)} \}$$

$$= \max\{3.0 + 9.1 + 3.1,\ 1.5 + 8.2 + 2.3\}$$

$$= \max\{15.2,\ 12.0\} = \boxed{15.2 \text{ ns}}$$

Problem 3-Part C (6 points)

A new type of flip-flop is constructed from two clocked D latches as shown in Figure 3. X is a 1-bit input and Z is a 1-bit output.

Compute the following quantities for this new type of flip-flop, using the simple gate delay model as well as the definitions of set-up time, hold time and the propagation delay. (See the beginning of the problem for the data on the D latch.)

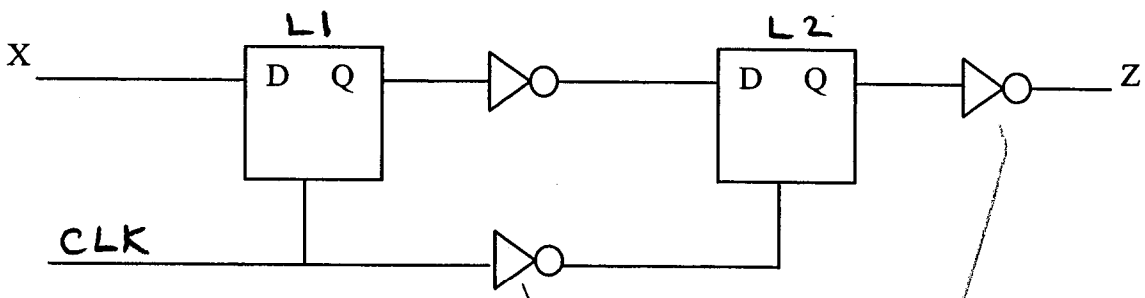
You must write your answer on the following lines.

Set-up time: 1.2 ns. (2 points)

Hold time: 0.5 ns. (2 points)

Clock-to-Z propagation delay: 3.7 ns (2 points)

Assume that the delay of an inverter is 1.2 ns.



- Figure 3 -

Show your work here (and on the next page, if needed):

(a) $t_{su}^{(ff)} = t_{su}^{(D\ latch\ L1)}$

$= \underline{1.2}$ ns.

(b) $t_h^{(ff)} = t_h^{(D\ latch\ L1)}$

$= \underline{0.5}$ ns

(c) $t_{CLK \to Z}^{(ff)} = t_{inv} + t_{CLK \to Q}^{(L2)} + t_{inv}$

$= 1.2 + 1.3 + 1.2 = \underline{\underline{3.7}}$ ns

PROBLEM 4: DELAY ANALYSIS OF ADDERS (23 points)

In this problem, we introduce an adder, which you have not seen before, called a "carry bypass adder". The main idea is the following: Consider a 16-bit adder made up of four 4-bit adders in cascade (i.e. hooked up as a ripple carry adder). There are two novelties in this particular design of the carry bypass adder:

1. If the group generate for a 4-bit adder is 1, then, there is no need to wait for the carry-in from the 4-bit adder in the previous stage. As soon as the group generate is computed to be 1, it is passed on as a carry-in to the next stage.

2. If the group propagate for a 4-bit-adder is 1, then, there carry-in from the previous stage can be propagated to the 4-bit-adder in the next stage. (This helps if the 4-bit adder itself is a ripple carry adder; the carry-in is said to "by-pass" the carry chain inside the 4-bit adder in this case by using the group propagate of the 4-bit adder.)

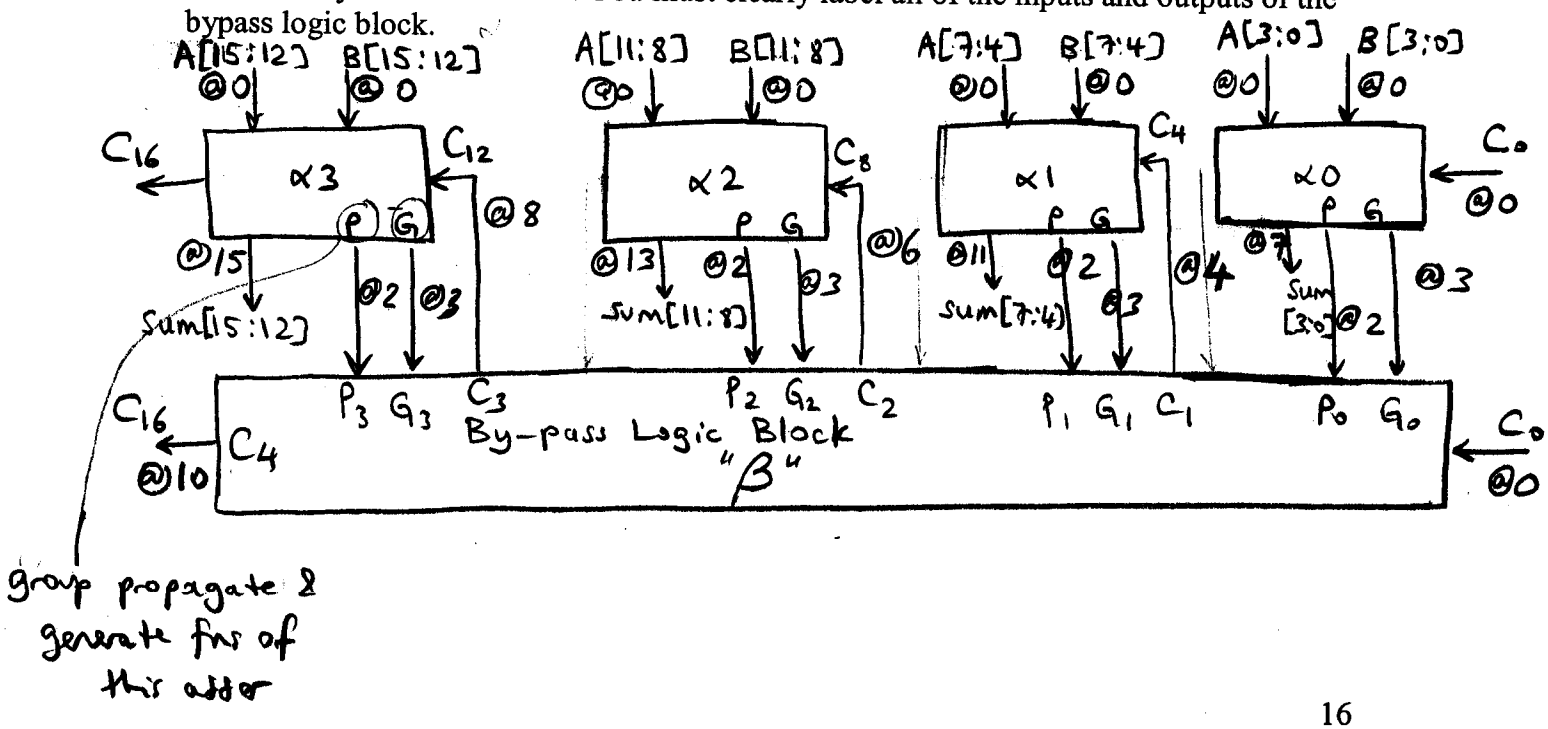
In this problem, you will analyze the worst-case delay of a 16-bit bypass adder made up of four 4-bit adders. But we will construct them in different ways.

Throughout the delay analysis in this problem, ASSUME THAT THE DELAY OF EACH GATE IS 1, REGARDLESS OF THE FAN-IN OR FAN-OUT OF THAT GATE.

Problem 4-Part A:

In this part, assume that the 4-bit adder block is a **ripple carry adder**. The 16-bit adder is constructed as a cascade of these 4-bit adders, and using the two bypass functions (described above as novelties 1. and 2.).

(a) **(2 points)** Draw a schematic of the 16-bit adder made up of four 4-bit adders. Clearly show all of the inputs and outputs. Further, show a block that says "bypass logic" (only its interface schematic, without showing what's inside) to implement the additional functionality described above. You must clearly label all of the inputs and outputs of the bypass logic block.



(b) **(8 points)** Write down the equations clearly for the bypass logic block, and show all of the delays for all of the intermediate and final outputs of this adder design (as we did for the carry look-ahead adder in class). In particular, we are interested in when the sum[15:0] bits finish the computation in the worst case. Assume that the two 16-bit inputs are available at time zero (@0) to the 16-bit adder. Use the @ notation on your diagram to show when the computations finish in the worst case.

(The main point is that the by-pass logic block does not do global computations as the CLA adder. Each by-pass logic computation requires access to only the carry-in from the previous stage and the group propagate and generate functions of the current stage. This results in higher delay, but less hardware.)

- <u>Logic equations inside the by-pass logic block β:</u>

$$\overset{@4}{C_1} = \overset{@3}{G_0} + \overset{@2}{P_0} \cdot \overset{@0}{C_0}$$

$$\overset{@6}{C_2} = \overset{@3}{G_1} + \overset{@2}{P_1} \cdot \overset{@4}{C_1}$$

$$\overset{@8}{C_3} = \overset{@3}{G_2} + \overset{@2}{P_2} \cdot \overset{@6}{C_2}$$

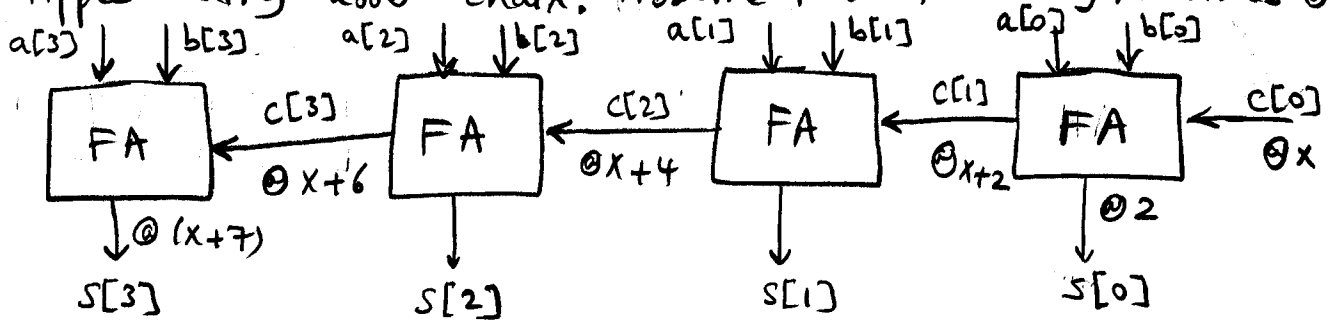
$$\overset{@10}{C_4} = \overset{@3}{G_3} + \overset{@2}{P_3} \cdot \overset{@8}{C_3}$$

- <u>Delay Analysis of the carrys:</u>

- As we showed in class, the group propagate and generate of each 4-bit adder is available @2 and @3 respectively.
- The delay analysis of the carrys is shown on the equations above. (Note that each AND and each OR gate is of delay 1.)

- <u>Delay analysis of the sum:</u>

Note that the sum bits go through the 4-bit ripple carry adder chain. Assume that the carry-in arrives @x.



The delay of the 4-bit sum vector is determined by the worst-case delay of s[3].

The carry chain has the delay:

$$\overset{@(x-i)+2}{c[i+1]} = \underbrace{\overset{@0}{a[i]} \cdot \overset{@0}{b[i]}}_{g[i]\ @1} + \underbrace{(\overset{@0}{a[i]} \oplus \overset{@0}{b[i]})}_{p[i]\ @1} \cdot \overset{@(x-i)}{c[i]}$$

Then

$$\overset{@(x+7)}{s[3]} = \underbrace{\overset{@0}{a[3]} \oplus \overset{@0}{b[3]}}_{p[3]\ @1} \oplus \overset{@(x+6)}{c[3]}$$

- The final results are shown on the diagram on p. 16.
- In the worst-case, the 16-bit sum vector is available @15 (since sum[15:12] becomes available @15.)

(c) **(4 points)** What is the critical path of this adder? Give a pair of 16-bit input vectors that achieve the worst-case delay.

- The critical path of this adder occurs through the by-pass logic carry chain of the first three stages plus the ripple carry chain of the last stage adder ($\alpha 3$).
- A pair that achieves the worst-case delay:

	$\alpha 3$	$\alpha 2$	$\alpha 1$	$\alpha 0$
A =	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1
B =	1 0 1 0	1 0 1 0	1 0 1 0	1 0 1 1

Note that this triggers the entire carry chain and assures that $\alpha 1$ and $\alpha 2$ do <u>not</u> group generate. (The fact that the $\alpha 0$ stage group generates is OK since it is the first stage and its carry-out is still available early @ 4.)

(d) **(3 points)** Explain how the group generate novelty (denoted as 1. in the problem description) helps the delay characteristics of the adder. Be precise. How is the worst-case delay affected by this novelty?

It lowers the <u>average</u> (and best-case) delays of the adder, but it does <u>not</u> affect the worst-case delay. (Note that the worst-case delay occurs when the carry has to be propagated all the way. So the bypass via the group-propagate helps in reducing the worst-case delay, but not the group generate.)

Problem 4-Part B (6 points)
In this part, assume that the 4-bit adder block is a carry look-ahead adder, just like the one we discussed in detail in class. Now, assume that we implement the same bypass functionality to build the 16-bit adder. What is the critical path of this adder? Compute the critical delay. (Note: You do not need to write down all the logic equations. However, show us how you obtain your result.) What are the differences that result compared to Part A?

- The critical path is constituted by the first three stages of the by-pass logic and the last stage CLA adder. ($\alpha 3$)

- <u>Critical delay</u>:
  C_{12} will still be ready @8. A 4-bit CLA ($\alpha 3$) takes 3 more delays to compute its sum. So, the result will be available @11.
  ∴ The critical delay is $\boxed{11 \text{ units}}$.

- The only difference is that we get a speed-up in the last-stage ($\alpha 3$) when the ripple carry chain to compute sum[15:12] is replaced by the CLA adder.

PROBLEM 5: DATAPATH AND CONTROL DESIGN (20 points)

In this problem, you will design a circuit that computes the "running average" of the four most recent words that come from a RAM chip. We use an 64 * 4 SRAM chip. The idea is to read out 4-bit words, **one at a time**, from the RAM chip, and have a digital filter to compute the average of the most recent four words that have been read out from the RAM chip. (Each 4-bit word is assumed to be a twos complement number.)
Once all the words in the RAM chip have been exhausted, the machine stops. We assume that the data in the RAM chip has been properly formatted such that it has 4-bit zero vectors in its 0th, 1st and 2nd words, and also 4-bit zero vectors in its 64th, 63rd, and 62nd words. This allows us not to worry about "edge effects". (If this were not the case, then we would have devise special circuitry to operate the machine for 3 more cycles at the head and the tail, and feed in zeros from outside so that the running average is not messed up.)

Therefore, the 4-bit words inside this RAM look like (in hexadecimal notation): 0x0, 0x0, 0x0, 0xA, 0x9, 0x7, ..., 0x6, 0x6, 0x0, 0x0, 0x0. (There are 64 such words in total. The values in the middle here are just an example.)

Your task is to design the whole system by following these steps:

(a) Think about how you will implement the computation of the running average. What components are you planning to use? Make a rough sketch of what the final circuit will look like. Show the RAM chip, what the circuitry will look like to compute the average, and any other components you need.

(b) Draw an ASM chart for this machine. (A rough chart showing the actions will suffice at this point.)

(c) Based on the ASM chart, define the control signals you need. (We need precise definitions.) Draw precise interface schematics of the components you will use (including the RAM chip's interface schematic.)

(d) Draw the datapath clearly. (This is the refinement of the sketch in Part (a).)

(e) Write down a refined ASM chart, or modify the ASM chart from Part (b). This ASM chart should use the control signals defined in Part (c).

(f) Illustrate the operation of your machine for the first 2 words that are read out from the RAM, by drawing a timing diagram with the control signals, showing the values of the outputs as well as the RAM addresses clearly. (You may use hexadecimal notation where needed.)