

CSE 140 Homework Three

July 20, 2009

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on July 31, 2009 (Friday) at 3:00pm.*

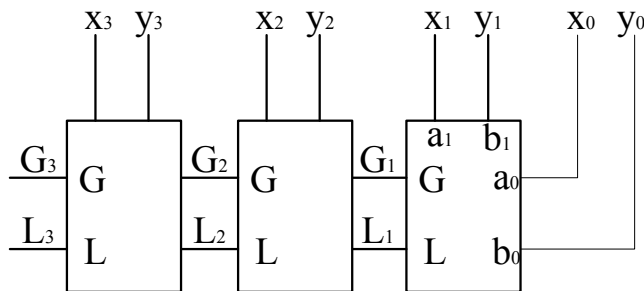
1 Problem Set Part A

- textbook 5.3
- textbook 5.9
- textbook 5.10
- textbook 5.13(c)
- textbook 5.16
- textbook 5.17(a)(c)
- textbook 6.2
- textbook 6.6
- textbook 6.10
- textbook 6.11
- textbook 6.13
- textbook 6.14
- textbook 6.19
- textbook 6.20

2 Problem Set Part B

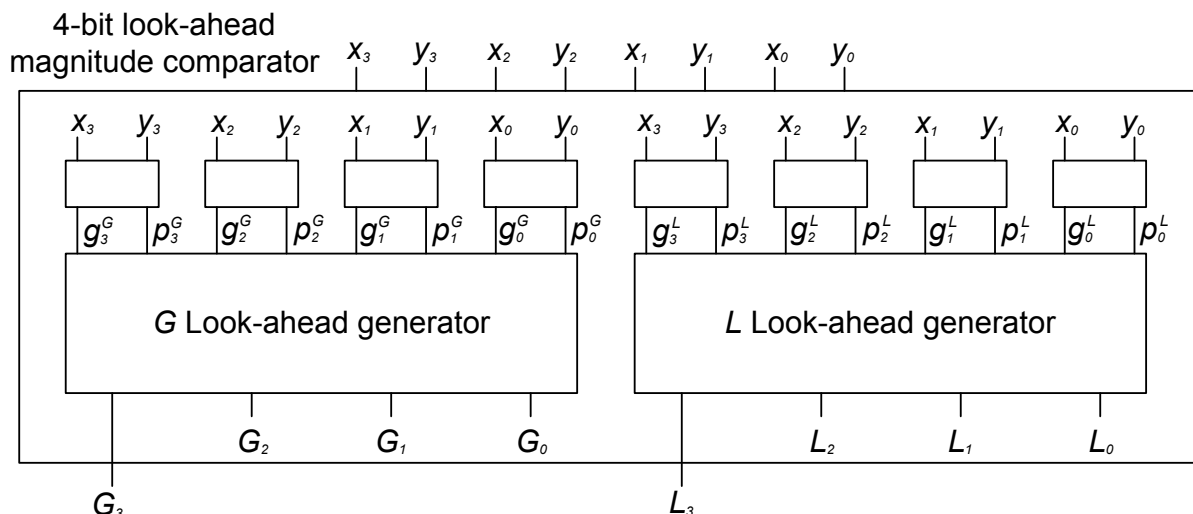
1 (Magnitude Comparator)

An n -bit magnitude comparator can be implemented by connecting $n-1$ 2-bit magnitude comparators in a serial manner, as shown in the figure below. The serial implementation, though, does have a major disadvantage in performance as its delay increases in proportion to the number of bits. In this question, you are asked to investigate the possibility of designing a **look-ahead magnitude comparator** that alleviates this performance bottleneck.



A 4-bit look-ahead magnitude comparator has a block level view as shown in the figure below. Since the G and L signals are both generated in a look-ahead manner, two sets of look-ahead logic are needed, one for signal G and the other for signal L . Each set of the look-ahead logic resembles a CLA adder in many aspects. To provide you a hint about how to approach the design, we outline a functional overview using the look-ahead logic for signal G as an example. For each bit i , a G generation function g_i^G and a G propagation function p_i^G are produced from the inputs at that bit, namely x_i and y_i . The g_i^G and the p_i^G signals are then fed into a look-ahead generator to produce the G signal for each bit in parallel. The look-ahead logic for signal L operates in a similar manner.

The G/L generation function g_i and propagation function p_i are defined in the following manner. If the G/L generation function g_i equals 1, the G/L signal at bit i will definitely be 1 without even needing to consult the G/L signal at the lower bit. On the other hand, the propagation function p_i lets us know whether it is necessary to consult the G/L signal at the lower bit, as when it is 1, the G/L signal at the current bit will directly assume the G/L value at the lower bit.



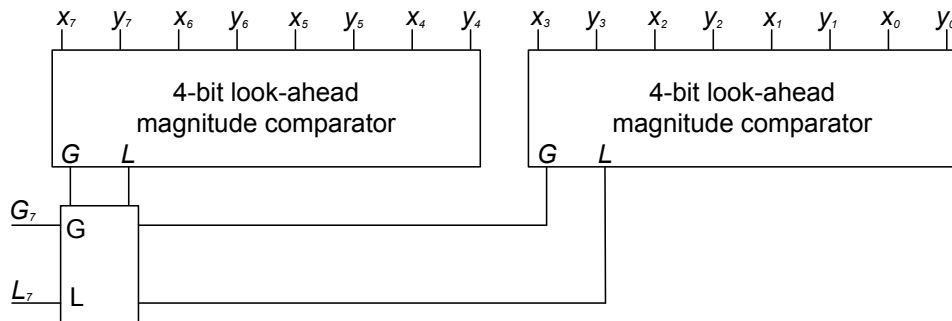
(Part A) Please derive the generation function g_i^G and propagation function p_i^G for the look-ahead logic for signal G .

(Part B) Please derive the generation function g_i^L and propagation function p_i^L for the look-ahead logic for signal L .

(Part C) Please derive the functions for G_0 through G_3 which define the logic in the G look-ahead generator. These functions should take g_i^G and p_i^G signals as inputs, and be constructed in a **sum-of-products** representation.

(Part D) Is the L look-ahead generator identical to the G look-ahead generator? Please provide a reasoning for your answer for full credit.

(Part E) An 8-bit magnitude comparator can be constructed by connecting two 4-bit look-ahead comparators to a 2-bit comparator, as shown in the figure below.



A smart engineer has found that it is possible to optimize the 2-bit comparator (i.e., implement it with less hardware cost) in this scheme. The optimized 2-bit comparator can be implemented with only *two* 2-input AND gates, *two* 2-input OR gates, and *two* inverters. In this part of the question, you are asked to explore such an optimization idea. Please derive the Boolean functions for the optimized implementation and draw its logic diagram.

2 (Carry Look-ahead Adder)

This question concerns the implementation of the *Carry Look-ahead* Adder (CLA), which utilizes the idea of *pre-computation* to reduce the delay of the carry chain in a ripple-carry adder. Recall that the carry equation of a single bit full adder can be written as:

$$c_{i+1} = g_i + p_i \cdot c_i \quad (1)$$

wherein $g_i = x_i \cdot y_i$ and $p_i = x_i \oplus y_i$. This equation can be furthermore expressed to generate the *sum-of-products* equations of c_{i+2} and c_{i+3} :

$$\begin{aligned} c_{i+2} &= g_{i+1} + p_{i+1}g_i + p_{i+1}g_i c_i \\ c_{i+3} &= g_{i+2} + p_{i+2}g_{i+1} + p_{i+2}p_{i+1}g_i + p_{i+2}p_{i+1}p_i c_i \end{aligned}$$

Furthermore, based on Equation (1), one can also generate the *sum-of-product* equations for the second level CLA as shown below:

$$c_{i+4} = G_{(i,i+3)} + P_{(i,i+3)} \cdot c_i \quad (2)$$

wherein

$$\begin{aligned} P_{(i,i+3)} &= p_{i+3}p_{i+2}p_{i+1}p_i \\ G_{(i,i+3)} &= g_{i+3} + p_{i+3}g_{i+2} + p_{i+3}p_{i+2}g_{i+1} + p_{i+3}p_{i+2}p_{i+1}g_i \end{aligned}$$

(Part A) In this part, we want to convert Equation (1), a *sum-of-products* expression, into a minimal *product-of-sums* expression for the carry, as shown in the following form:

$$c_{i+1} = \gamma_i \cdot (\delta_i + c_i) \quad (3)$$

We are furthermore providing you 4 possible options for the expressions of γ_i and δ_i :

- A** $\gamma_i = x_i + y_i$, $\delta_i = x_i \odot y_i$;
- B** $\gamma_i = x_i \oplus y_i$, $\delta_i = x_i \odot y_i$;
- C** $\gamma_i = x_i \oplus y_i$, $\delta_i = x_i \cdot y_i$;
- D** $\gamma_i = x_i + y_i$, $\delta_i = x_i \cdot y_i$;

Please help us by figuring out the correct solution(s) first (there may be multiple!). If you think multiple correct solutions exist, please let us know the solution that you think is best for this type of CLA adder, and tell us your reasoning. Please clearly show your answers in the appropriate position below.

The correct option(s) are:

The best option (if multiple correct solutions exist) is:

The reason is:

(Part B) The values of γ_i and δ_i can signify whether a carry-out can be directly generated at bit i without consulting its carry-in or whether the carry-in will be propagated to its carry-out with no change. Please specify the correct values of γ_i and δ_i for these two possibilities just described.

Case 1: carry-out directly generated at bit i without consulting carry-in

$$\gamma_i = \quad \quad \quad \delta_i =$$

Case 2: carry-in propagated to carry out with no change

$$\gamma_i = \quad \quad \quad \delta_i =$$

(Part C) Based on Equation (3) in (Part A), one can generate the *product-of-sums* form for a 2-level CLA adder in an analogous manner to Equation (2):

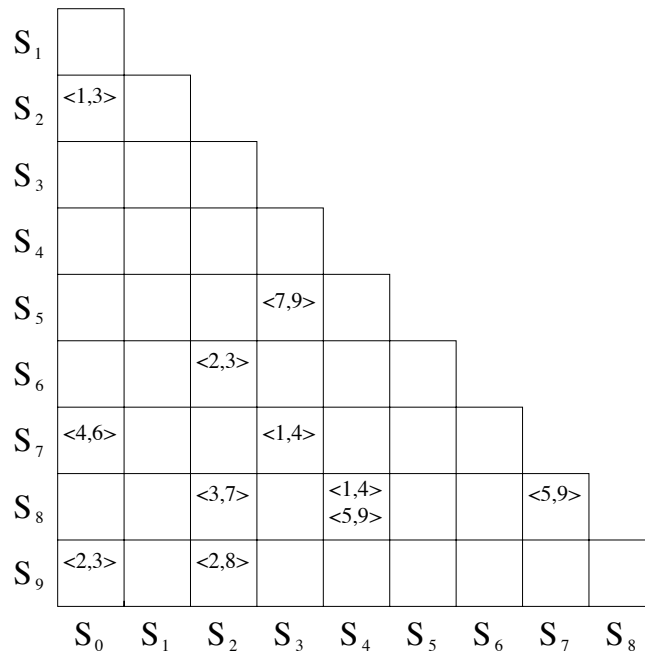
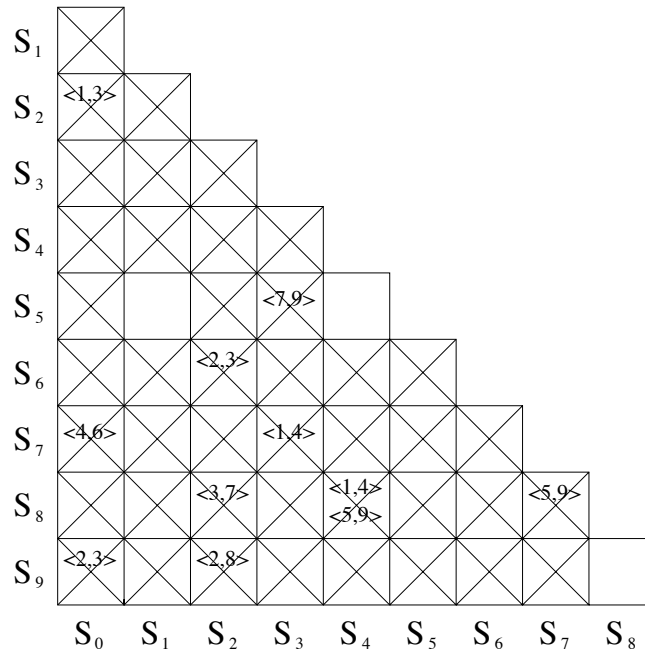
$$c_{i+4} = \Gamma_{(i,i+3)} \cdot (\Delta_{(i,i+3)} + c_i) \quad (4)$$

Your job in this part is to figure out the minimal *product-of-sums* expressions for $\Gamma_{(i,i+3)}$ and $\Delta_{(i,i+3)}$ that is used in Equation (4). Please clearly show your answer in the following table.

	Expression
$\Gamma_{(i,i+3)} =$	
$\Delta_{(i,i+3)} =$	

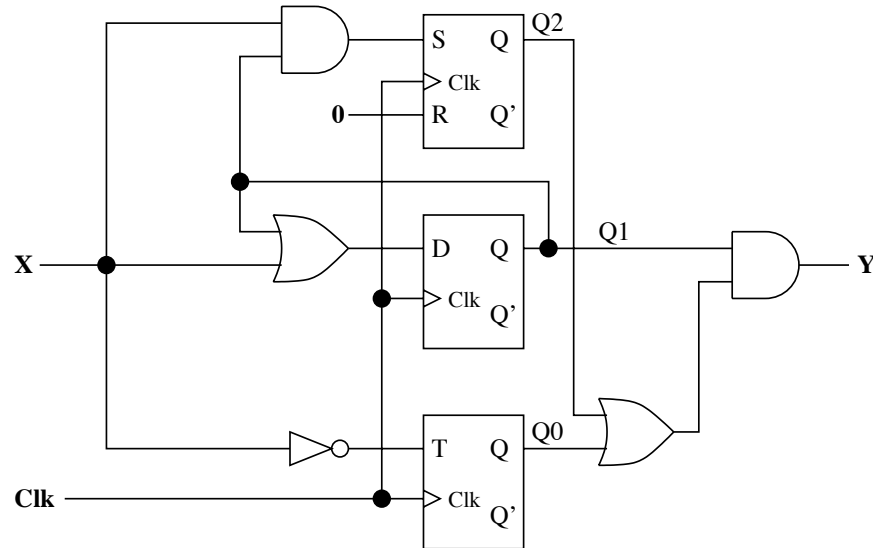
3 (State Minimization)

Our technical staff has been working on the following state minimization problem. In a moment of bleary-eyed, need more coffee time, they made a mistake and put an 'X' where it does not belong. Then they continued filling in the chart. Rather than start from scratch, we would like you to fix the problem on this diagram. Please identify the 'X' that was inadvertently placed and provide a reason for your answer. Next, please correct the ensuing inconsistencies that result from your fix. Finally, tell us the equivalent state classes of the final answer.

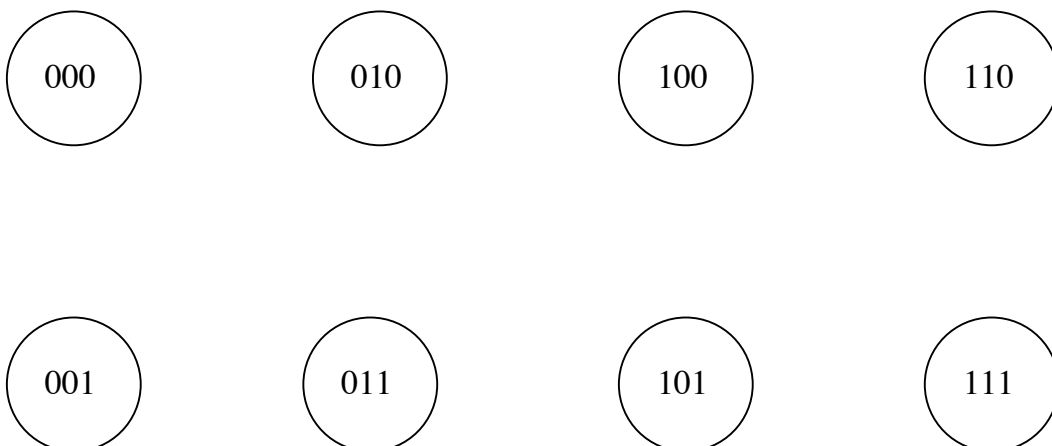


4 (FSM Analysis and Optimization)

The following sequential circuit consists of three different flip-flops and five gates. The input and output signals are X and Y , respectively. Each state is encoded as $Q_2Q_1Q_0$, with **SR** being the most significant bit and T being the least significant bit of the state representation.



(Part A) In this part, you are asked to figure out the state diagram for the FSM. Assuming all the three flip-flops are initialized to 0 at the beginning, please specify all the transitions, as well as the input and output values associated with each transition. *Please note that you may not need to include all the states in your state diagram.*



(Part B) A group of CSE 140 students (from a previous batch) is having a hot discussion about the optimization opportunities for the given sequential circuit.

- Student **A** thinks that the number of flip-flops can be reduced by performing state minimization.
- Student **B** thinks that the original state diagram already has the minimal number of states.
- Student **C** thinks that while state minimization can be performed to reduce the number of states, the number of flip-flops can not be reduced.

Which of the three statements do you think is correct? Please give a brief reasoning.

(Part C) While the previous three students are struggling with the idea of state minimization, another pair of students are busy with implementing the given sequential circuit in *LogicWorks*. For some reason their implementation is not working correctly. After spending a lot of time debugging their circuit, they find that the problem is caused by a bug in *LogicWorks*: the inverter provided in the *LogicWorks* standard library is not working correctly.

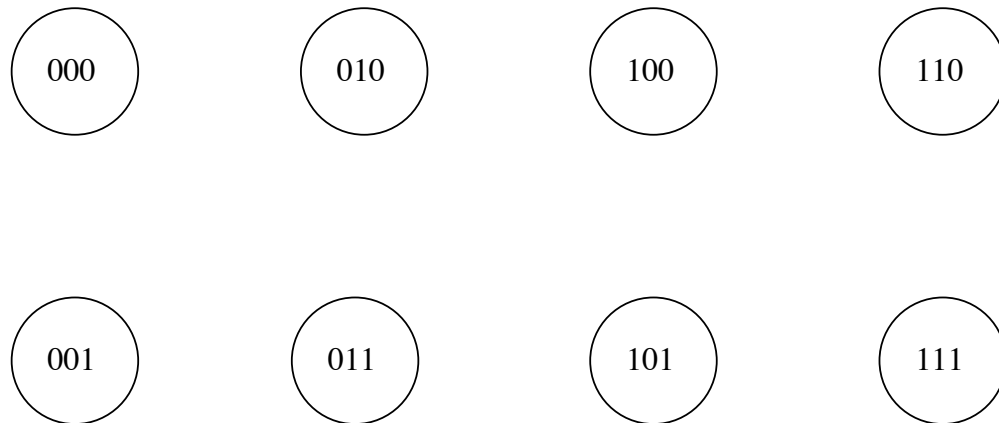
To make their implementation work correctly, Student **X** is wondering whether they should replace the inverter with an NAND gate or they should change the *T* flip-flop to a *D* flip-flop. Student **Y**, a somewhat smarter kid, proposes another idea. He has observed from the circuit that the inverter is only used by the *T* flip-flop and the next state logic of the other two flip-flops does not depend on the value of the *T* flip-flop. Therefore, he claims that without changing the flip-flop types, by only swapping the encodings of two states, the FSM can be implemented within only **two** 2-input **AND** and **two** 2-input **OR** gates and **no** inverters. Unfortunately, due to his highly limited communication skills, he cannot convince his partners about this idea. On the other hand, Student **X** does not believe in the statement of Student **Y**. Her argument is as follows: she has observed that some of the transformations on input '1' in the state diagram are self-loops, i.e., from state S_i to S_i . Therefore, she thinks that it is impossible to get rid of the inverter if the *T* flip-flop is still in use.

Since this group is in a deadlock situation with no one able to convince the other, they are asking for your help. As a student who is steeped in digital logic understanding, you immediately find that although the concern of Student **X** is well-founded, the statement of Student **Y** is correct. Please help Student **Y** to convince his partner by 1) enunciating clearly the two states whose encodings need to be swapped so as to get rid of the inverter, and 2) making appropriate modifications to the FSM that of course preserve the semantics yet can handle the self-loop problem. Furthermore, please help this group of students to quickly finish their implementation by figuring out the new state diagram and the new schematic of the circuit. (*Please clearly show your answers on the next page.*)

The two states whose encodings need to be swapped:

The way to handle the self-loop problem:

The new state diagram:



The new circuit implementation:

