1. **NOTE 1:** Full names on each page. Number each page. Single sided. Stapled.
2. Failure to follow (2) precisely will result in your assignment not being graded.
3. **** NOTE 2 **** This lab may be done in pairs. However there is to be no collaboration between pairs. **The academic code of integrity will be strictly enforced.**
4. **** NOTE 3 **** The code you submit should be crystal clear, well documented, with the use of proper, self-explanatory identifiers. The plots should be similarly clear, properly labeled and should be accompanied by the corresponding table of values, and a summary of the results or findings.

The purpose of this assignment is two-fold. First, it is to gain experience in writing Verilog behavioral and structural descriptions of a most common component in digital systems, namely, combinational adders, and to perform functional and (abstract) timing simulation. Second is to learn some remarkably simple ways of increasing the performance of adders.

The lab may appear to be very long and tedious. It isn't. After solving the first problem, you can repeatedly reuse the code with simple modifications to solve the others. Then with click of a button, you can simulate each. Read the entire assignment first before you start and design the code for maximum reuse.

1. Figure 1 shows the classic ripple carry adder (RCA), consisting of N full adders (FA). Normally, you are used to seeing each full adder supplied with the bits A_i and B_i of the operands. A minor variaition is to first compute two functions $G_i(A_i, B_i)$ and $P_i(A_i, B_i)$, and then use the G_i and P_i to compute the carry out and sum.

   The Ps and Gs shown in the figure are simply functions of the operands A and B. Consider the carry-out C_{i+1} of stage i of a FA. $C_{i+1} = 1$ can be *generated* **independent** of the carry-in C_i if both the operand bits $A_i = 1$ and $B_i = 1$. We refer to this as the *generate* condition and denote it by $G_i = A_i B_i$. The other possibility is that the carry-out can be the same as the carry-in, $C_{i+1} = C_i$ if $A_i \neq B_i$. We refer to this as the *propagate* condition and denote it as $P_i = A_i \oplus B_i$. Therefore, $C_{i+1} = G_i + P_i C_i$. The important thing to note is that G_i and P_i are functions only of the operands. The sum $S_i = P_i \oplus C_i$

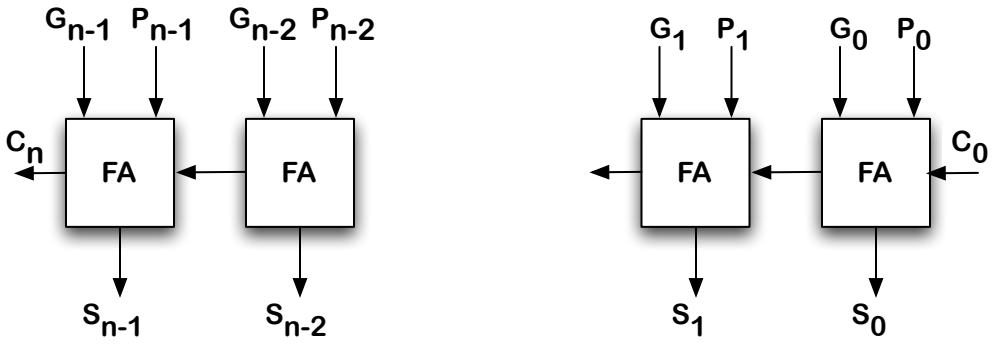

Figure 1: N-bit Ripple Carry Adder (RCA)

   (a) Write an Verilog behavioral description of a FA. Assume that each two input logic operation (e.g. AND(x,y)) takes 1ns, and that each FA is implemented as a sum of products expresson. In this case, an $XOR(a, b) = ab' + a'b$ would take 2ns from the time of arrival of the latest input.

   (b) Write a Verilog structural description of a N-bit RCA shown in Figure 1. This is a very simple exercise, and the description of an RCA can probably be found in almost any document on Verilog.

(c) Construct a testbench and simulate RCAs for N = 4, 8, 16, 32, 64 and 128 bits. **** NOTE **** Use Verilog constructs that help make the code compact. Where applicable, parameterize the description.

**** Note: **** When simulating each RCA, first determine the input operands that will result in the maximum possible delay in the circuit - the time to compute the last bit of the sum and the final carry out. In an RCA it is the carry propagation that determines the delay of the circuit. Simulate your designs for the worst case inputs. Use the simulator to *measure* the delay. **Plot the delay as a function of N**.

2. This the first simple variation of the RCA, and is shown in Figure 2. The basic idea here is that the carry-in (C_0) into the first stage can be propagated directly to the end (without having to wait for it to ripple through) if all the propagate signals $P_i = 1$, for $i = 0, 1, \ldots, M-1$. Hence, as the figure shows, if $P_o P_1 \cdots P_{M-1} = 1$, then we can simply propagate the carry-in directly to the end, and use $P_o P_1 \cdots P_{M-1}$ as the select bit of a multiplexor, whose output will the carry-out (C_M) of the M-bit adder. We will refer to this design as a CBA.

   Figure 3 shows a block diagram of an M-bit CBA. It consists of a unit called SetUp, which takes as input the operands A and B, and computes all the G_i and P_i. The second unit, called CarryPropagate, computes each carry-out given the G_is and P_is. Up to now, nothing is really different from an RCA. t_{setup}, t_{mux}, t_{sum} and t_{carry} denote the delay of the corresponding blocks.

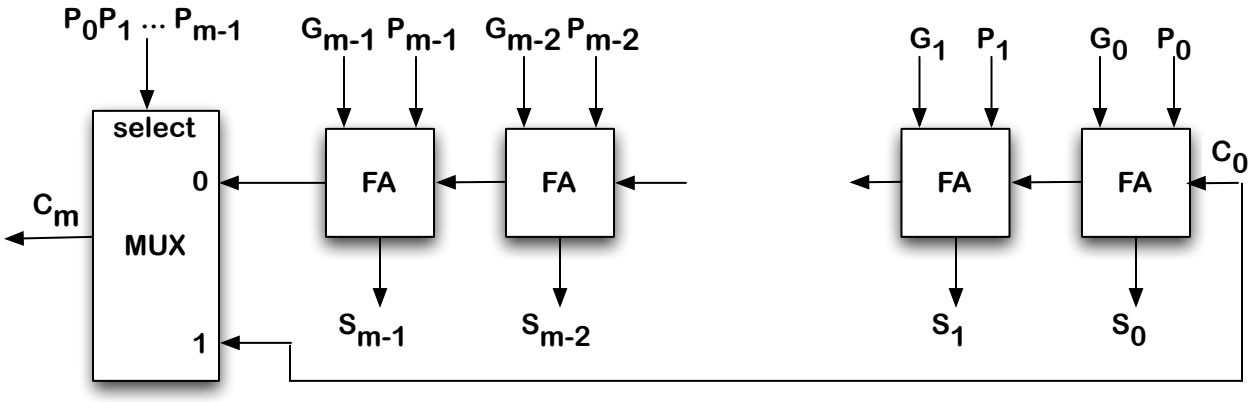

Figure 2: M-bit CBA

The CBA is not much of an improvement over an RCA. The worst case delay is still caused by the carry having to ripple through each stage. The average delay of the CBA will be lower than that of an RCA, but that is of no advantage, as we have to only be concerned with the worst case delay. The advantage of the CBA becomes apparent when M-bit CBAs are cascaded to form a (large) N-bit adder ($N = kM$, where k is the number of M-bit CBA stages).

Figure 4 shows a cascade of k, M-bit CBAs, for which you are to write a Verilog description. Use $M = 4$. Therefore $N = kM$. Perform the same steps as in question (1).

(a) Simulate the CBA for $k = 1, 2, 4, 8, 16, 32$ (i.e. for $N = 4, 8, 16, 32, 64, 128$. As in problem (1), identify the input that will result in the worst case delay for the N-bit CBA. Plot the delay as a function of N, just as in the case of question (1).

(b) Derive an expression for the delay of this cascade in terms of M, N, and the delay parameters of the M-bit CBA. This should be consistent with the trend shown in the plot.

3. In an RCA every FA has to wait for the incoming carry before an outgoing carry can be generated. One way to get around this dependency is to compute the result of both possibilities in advance. Once the real value of the incoming carry is known, the correct result is selected using a muliplexer. This variation

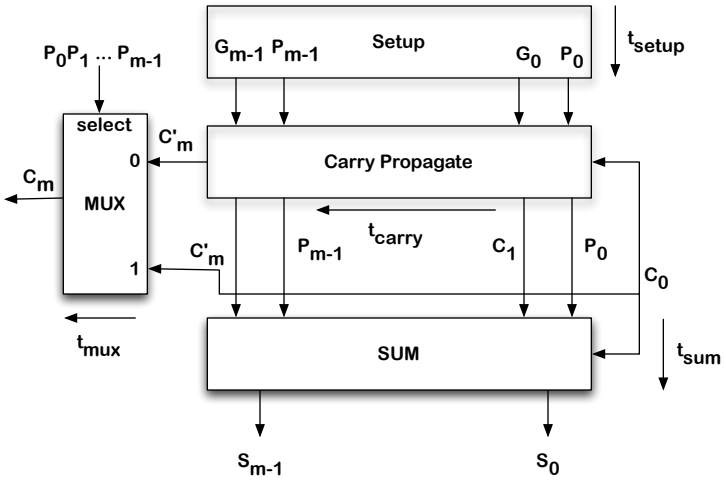

Figure 3: Block diagram of an M-bit CBA)

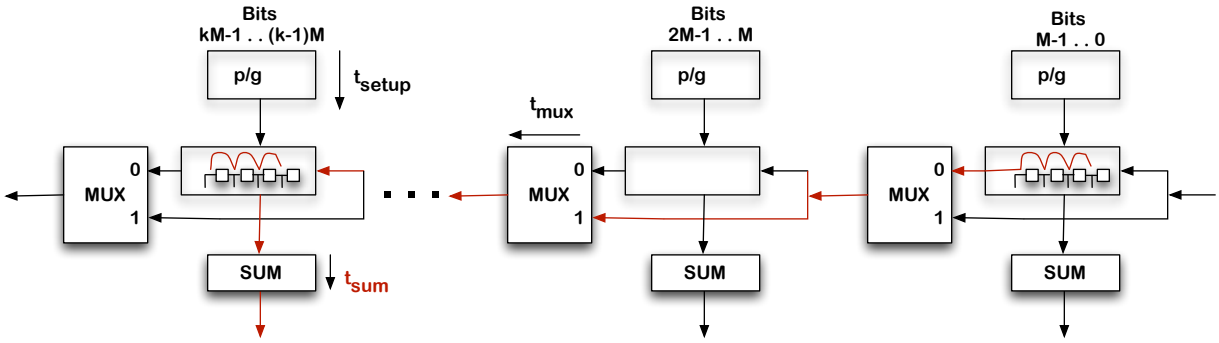

Figure 4: N-bit CBA using a cascade of k, M-bit CBAs)

will be referred to as CSA. Figure 5 shows a block diagram of the one stage of an M-bit CSA. A full N-bit CSA ($N = kM$) is built by cascading k M-bit CSAs, in a manner identical to the CBA. This is shown in Figure 6.

Perform what was requested for problem (2) for this design.

(a) Simulate the CSA for $k = 1, 2, 4, 8, 16, 32$ (i.e. for $N = 4, 8, 16, 32, 64, 128$. As in problem (1), identify the input that will result in the worst case delay for the N-bit CBA. Plot the delay as a function of N, just as in the case of question (1).

(b) Derive an expression for the delay of this cascade in terms of M, N, and the delay parameters of the M-bit CSA. This should be consistent with the trend shown in the plot.

4. This is a minor variation of the CSA of problem 3. In problem 3, each CSA stage was M-bits wide. In this design, which we refer to as SQCSA, the first stage will be a 2-bit CSA stage (for bits 0, 1), the second will be a 3-bit CSA stage (for bits 2, 3, 4), the third will be a 4-bit CSA stage (bits 5, 6, 7, 8), and so on. Each stage gets progressively wider by one bit. Perform what was requested for problems (2) and (3) for this design.

(a) Simulate the SQCSA for $k = 3, 5, 7, 10, 15$ (i.e. for $N = 9, 20, 35, 65, 135$ bits). As in the earlier problems, identify the input that will result in the worst case delay for the N-bit SQCSA. Plot the delay as a function of N.

(b) Explain the results.

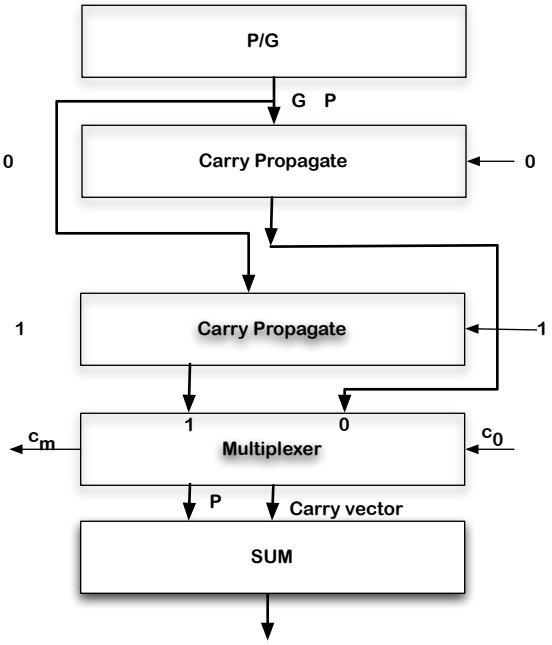

Figure 5: M-bit CSA stage)

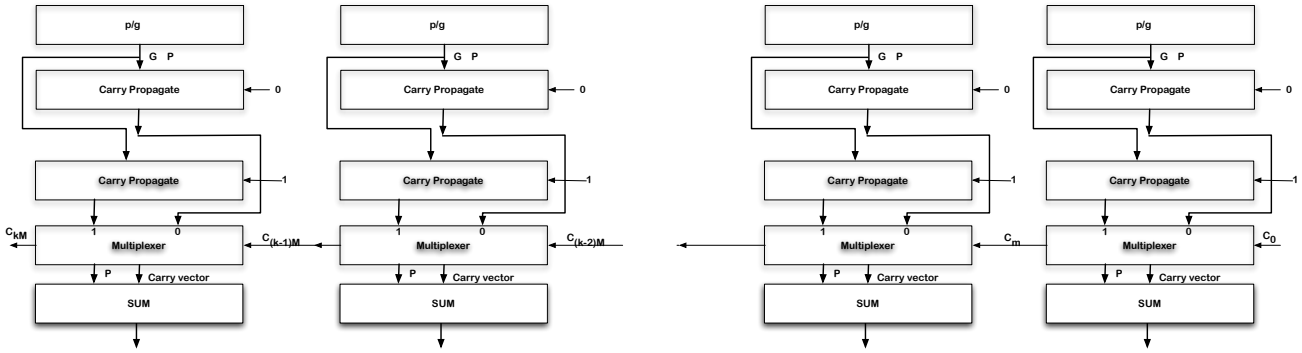

Figure 6: N-bit CSA using a cascade of k, M-bit CSAs)