B EE 427 PA #3
Due: 03/05/2023 11:59pm

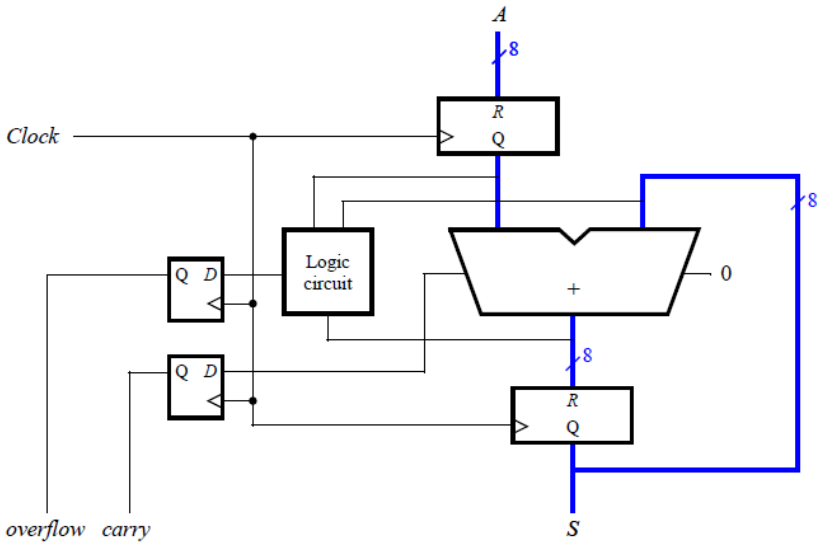
Note: There are two parts. Create a Quartus Prime project file for each part. **You should submit your Verilog source code, test bench code, and simulation screenshots to Canvas.** In addition, you should show me your demo using the DE1-SoC board. There is no report for this PA.

Part 1

If the "+" symbol is used in Verilog, we can simply write the code of a ripple carry adder (RCA) as follows:

```
input [N-1:0] a, b;
output [N-1:0] sum;
output c_out;
assign {c_out, sum} = a + b; // assume that c_in is always 0 (always addition)
```

By using an RCA module, implement the circuit below.



This circuit, which is often called an *accumulator*, is used to add the value of input A to itself repeatedly. The circuit has an output *carry* from the adder, as well as an *overflow* output signal. An arithmetic overflow occurs when an addition between positive numbers generates a negative number or an addition between negative numbers generates a positive number.

To implement an 8-bit accumulator design, perform the following steps:

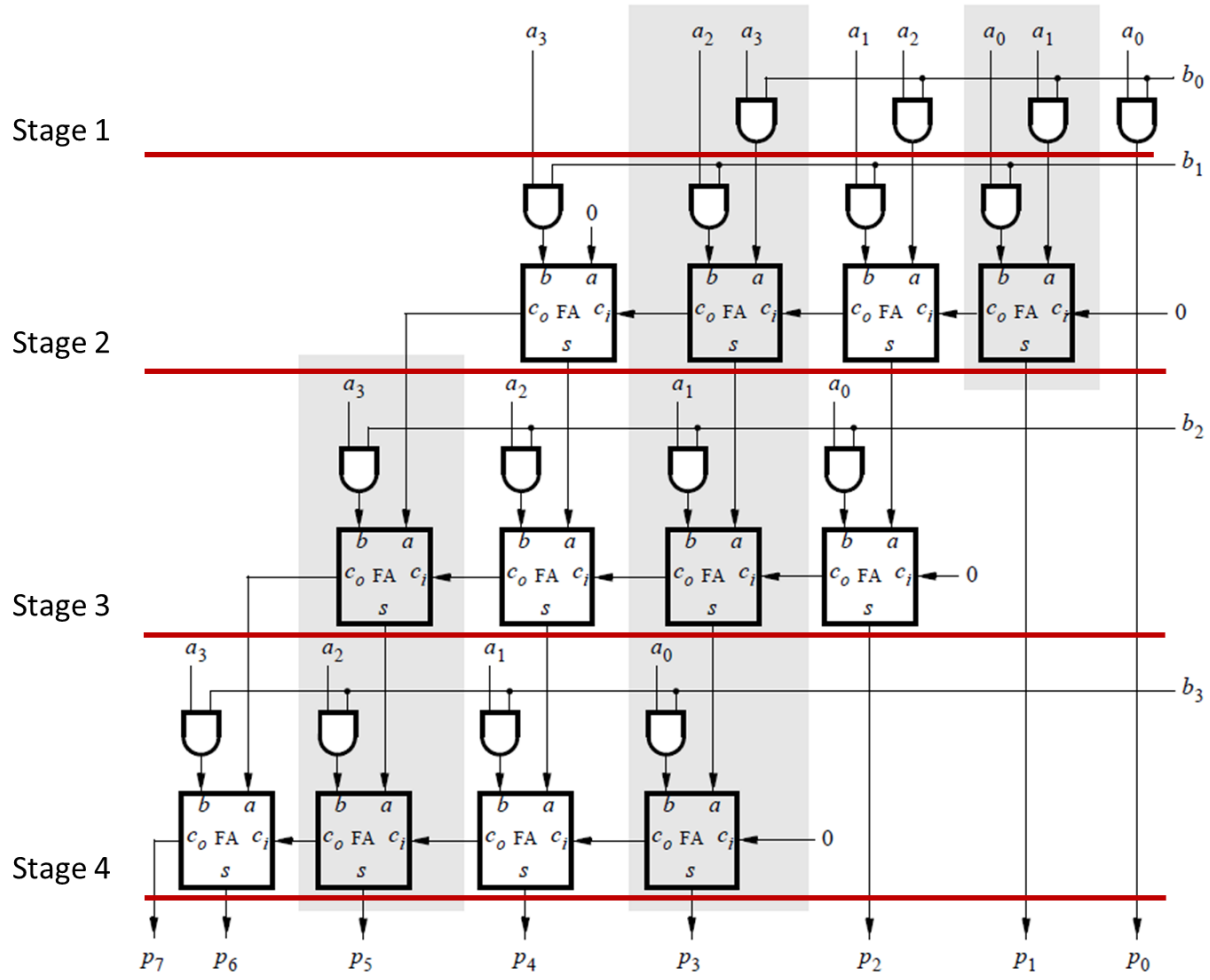
1. Create a new Quartus Prime project using the system builder program. You'll use switches, buttons, LEDs, and 7-segment displays.
2. Connect input A to switches SW7-0. A is an 8-bit binary number represented in 2's complement, but assume that each A for testing is a positive number. Use SW9 as an asynchronous reset (either

negative or positive one is fine) and KEY0 as a manual clock input. The sum from the adder (not registered) should be displayed on the red lights LEDR7-0, the registered carry signal should be displayed on LEDR8, and the registered overflow signal should be displayed on LEDR9. Show the registered values of A and S as hexadecimal numbers on the 7-segment displays HEX3-2 and HEX1-0, respectively.

3. Include the Verilog code in your project and compile the circuit.
4. Simulate the designed circuit to verify its functionality.
5. Verify that your circuit works properly by observing the lights.

Part 2

The figure below shows a block diagram of a 4-bit unsigned multiplier architecture. This time, you should convert it into a **4-stage pipelined** architecture.



If operands Ai and Bi (i = 1, 2, 3, …) are inputted every cycle, the operation of this circuit is as follows:

	Cycle 1	Cycle 2	Cycle 3	Cycle 4	…
Stage 1	**A1[3:0] x B1[0]**	**A2[3:0] x B2[0]**	**A3[3:0] x B3[0]**	**A4[3:0] x B4[0]**	…
Stage 2		(A1[3:0] x B1[0]) **+ (A1[3:0] x B1[1])**	(A2[3:0] x B2[0]) **+ (A2[3:0] x B2[1])**	(A3[3:0] x B3[0]) **+ (A3[3:0] x B3[1])**	…
Stage 3			(A1[3:0] x B1[0]) + (A1[3:0] x B1[1]) **+ (A1[3:0] x B1[2])**	(A2[3:0] x B2[0]) + (A2[3:0] x B2[1]) **+ (A2[3:0] x B2[2])**	…
Stage 4				(A1[3:0] x B1[0]) + (A1[3:0] x B1[1]) + (A1[3:0] x B1[2]) **+ (A1[3:0] x B1[3])**	…

From Cycle 4, all the full adders (FAs) and AND gates in this circuit work in parallel and we can get multiplication results every cycle.

To implement this design, perform the following steps:

1. Create a new Quartus Prime project using the system builder program. You'll use switches, buttons, LEDs, and 7-segment displays.
2. Write Verilog code. Use switches SW7-4 to represent operand A and switches SW3-0 to represent operand B. The A and B values are to be displayed as decimal numbers on HEX3-2 and HEX1-0, respectively. Use SW9 as an asynchronous reset and KEY0 as a manual clock input. You should always set new SW7-0 values before pressing KEY0. The 8-bit multiplication result is to be displayed on LEDR7-0. You can use the "+" symbol for an addition.
3. Include the Verilog code in your project and compile the circuit.
4. Simulate the designed circuit to verify its functionality.
5. Verify that your circuit works properly by observing the lights.

Reference

[1] Intel, "Teaching Materials for the Intel FPGA Academic Program"