Credit Hours System
CMPN111 Logic Design-2
Fall 2022

Cairo University
Faculty of Engineering

Lab Requirement 2
Carry-Select Adder

If we want to implement a 16-bit full-adder (FA) circuit using 4-bit full adders, the go-to method of implementation would be as follows:

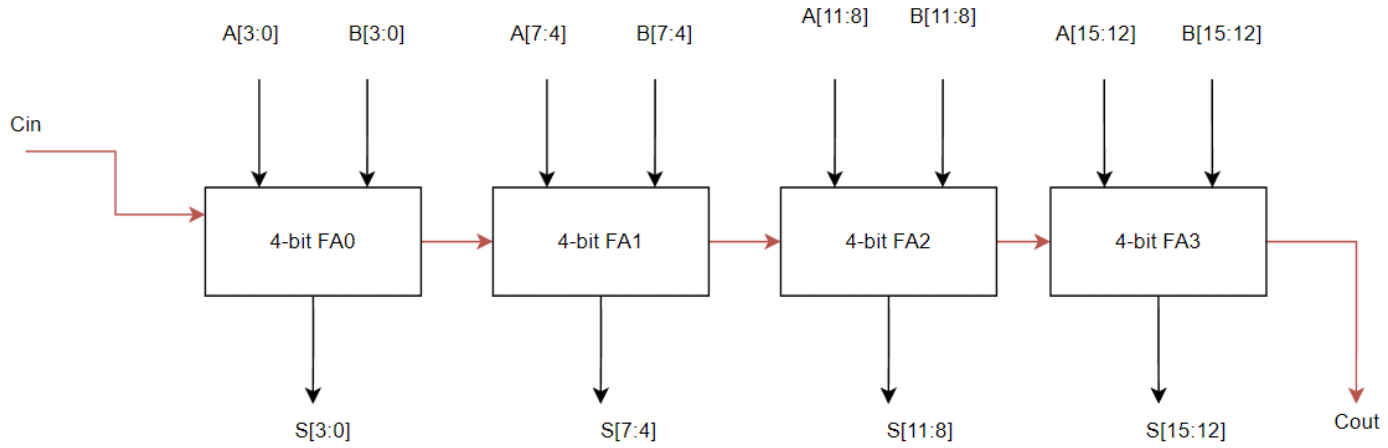

Fig. 1 Ripple-Carry Adder

This is called the "ripple-carry" adder (as carry ripples from each module to the next, just like a wave).

While this implementation is functionally correct, it's not fast. This is because of 2 main reasons: 1) nothing is ideal in real life and every FA module has an internal propagation delay (PD) which is the time between input given and output produced, and 2) every FA must wait for the carry-out of the previous FA before it can produce a proper output.

We consider the PD for the longest input-output path in the circuit (colored in red).

For example, if we assume each FA has a 10ns PD, FA0 will produce it's correct output (S [3:0] and Cout) at t=10, but then at that time, FA1 will have just received the proper Cout for it's calculations, so it'll take another 10ns to produce the correct result. So now FA1 will have proper input at t=10 and proper output produced after another 10ns which is t=20. This goes on and on until you reach the final FA module (FA3 in our case) which will output the correct output at t=40).

It's clear that for this implementation, the PD is linearly proportional to the number of FAs you have. If we have 4 FAs(each having tPD=tFA=10ns delay), we get a delay of 4*10ns which is 40 (4*tFA). For N FAs, we get N*tPD delay. This is not efficient for adders of larger sizes (think 128-bit FA and above).

There are several different FA designs that solve this problem, the one we will discuss is the "Carry-Select" adder (CSA) illustrated below:

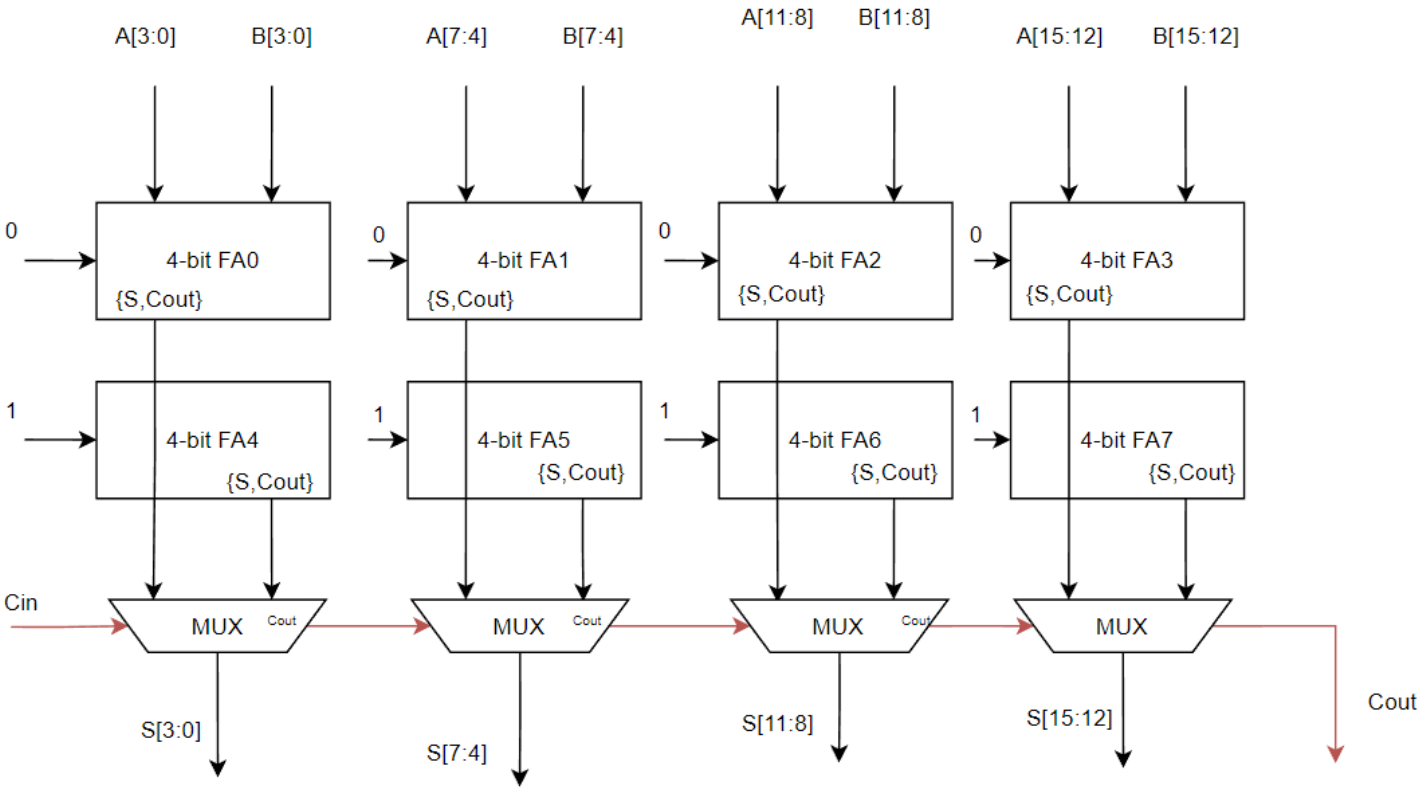

Fig. 2 Carry-Select Adder

There are a few differences in design from the normal ripple adder, they are as follows:

- For every 4 bits of input, 2 FAs calculate (in parallel) the output sum. One FA is fed Cin=0 and the other Cin=1.
- The output of each parallel FA pair (vertical column) is fed to a MUX which decides to choose {S,Cout} of the upper FA if Cin=0, and {S,Cout} of the lower FA if Cin=1. This ensures that if Cin=0 we output the result of the upper adder and if Cin=1 we output the lower adder result (Select the proper adder output according to Cin)
- Both CSA and Ripple-Carry adders have the exact same functional output for the same input, the only difference is the extra hardware and the different delas.
- Our PD depends on 2 factors: FA PD and MUX PD. Since both upper-row and lower-row FAs calculate in parallel, they will have the same delay equal to a 4-bit FA delay "tFA".
- Our longest data path from input to output now is highlighted in red, with a PD of 4 MUX delays "4tMUX".
- As such, the final PD will be tFA+ 4tMUX = tFA + N*tMux, N=4.
- The ripple-carry adder had a delay of 4*tfA=N*tFA, N=4.
- If we assume tMUX to be less than tFA (which is usually the case), the CSA will always calculate and output faster than the ripple-carry adder.
- In this design, we have optimized the performance of the adder at the cost of extra hardware (another row of FAs and MUXs).

The Requirement

You are required to implement the CSA illustrated in Fig.2. Your input/output size is exactly the same as illustrated (A, B, and S are 16-bits). Each sub-FA is 4-bit (same as illustrated). Each MUX is a 4-bit 2x1 MUX.

Deliverables

You should submit the following 2 files:

1. Verilog implementation of the CSA.
2. Verilog testbench for the CSA.

Implementation Notes

As this is your first lab for the course, we recommend following these notes for your implementation:

1. Be modular. You should first try to implement 4-bit FA, then, 4-bit 2x1 MUX, and reuse the 4-bit FA module and MUX module four times inside your CSA module.
2. Name your Verilog modules and Verilog module files the same name (module "full_addr" should be in a file named "full_addr.v"). It's also usually better to name modules as if they were variable names (don't start with numbers, don't have special characters or spaces inside the name, etc.).
3. There are two approaches for implementing the FAs and MUXs:
    1. Using logic gates, as explained in the tutorial.
    2. Using assign statements, if you want to add two 1-bit numbers and consider carry you can do as follows:
       assign {carry,sum} = (a + b);
       if you want to implement 1-bit mux, you can do as follows:
       assign out = (sel) ? in1: in0;
4. When developing the testbench, try to consider several test cases to be sure that your hardware implementation is working properly.
5. Quartus is not needed for this requirement. You can use Modelsim.
6. Feel free to contact us for any inquiries and good luck!