Laboratory Three

Combinational Logic Design: Adders

Basic Concepts

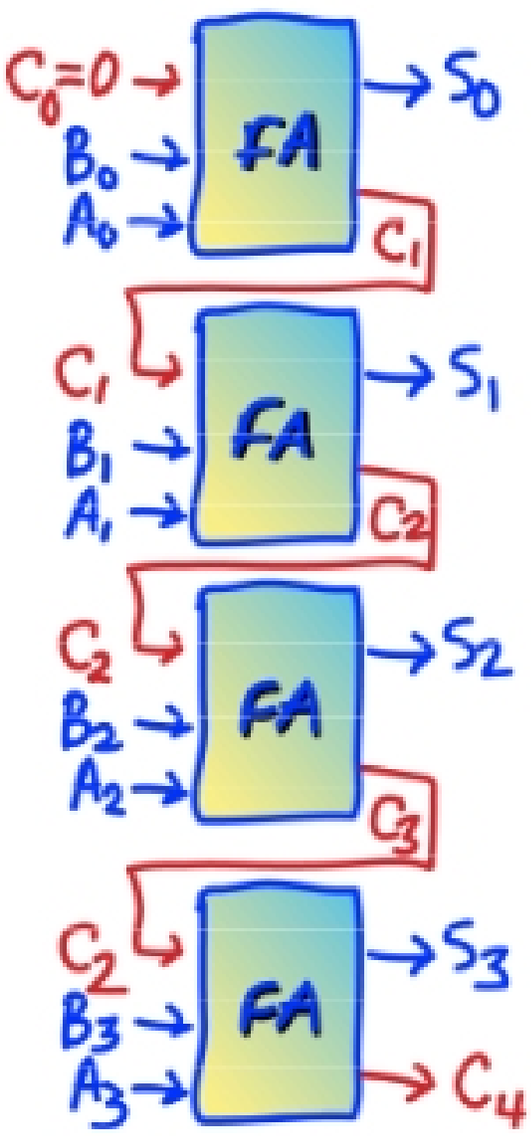
1. The central element of the ALU is the adder, which adds the contents of registers. Adders are built through the use of exclusive-OR gates, also called **XORs**.
2. A Full Adder (**FA**), adds three 1-bit operands, to produce a sum and an output carry bit. This is 1-bit full addition. A *ripple* adder may then be constructed by cascading n **FA**s.
3. Two level (**SOP**) n-bit *parallel* adders may be derived from n-bit cascaded adders by substituting the output equations of one stage into the next. This produces a very fast circuit, however, it also results in many product terms. Fortunately, we can speed up the entry of these big circuits in ***Xilinx*** through the use of ***VHDL*** code.
4. Faster carry *lookahead* adder designs reduce the size of the carry equations by looking ahead at what the carrys might be.
5. Review these video tutorials to see how to set up a new project, enter a design and simulate your design.

Note: This prelab assignment is worth 10 pts. See the (Pre)s in Tasks 2 and 3 below.

Task One: Logic Design of a Ripple Adder

The simplest adder that can be built is the 4-bit ripple adder shown at right. To save time, we will build this adder using the **ADD1** full adder logic block in ***Xilinx's*** **Schematic Editor.**

1. Copy and paste the circuitry for a **Full Adder** (see Lab 2, Task 2) into a schematic. Paste three more **FA** vertically down, as shown at right. We will label each stage with a number i. There will be four stages, for $0 \le i \le 3$. Label all inputs as A_i, B_i, C_i and outputs as S_i and C_{i+1}, as shown at right, for $0 \le i \le 3$, and connect the stages with the "C_i" (Please set C_0 to **0** and reserve C_4 to form a fourth carry output). You now have made a 4-stage recursive (ripple) adder with 13 non-zero terminal (pin) signals total.
2. Go to the **Project Navigator** and click the Design tab. In the **View pane** at the top, select Simulation. Select your schematic in the **Hierarchy panel**, and start the **Simulate Behavioral Model**. Go to **File->Load**. Browse to the location of the test benches and open the appropriate ***.do file**. Verify that you can properly add some 4-bit numbers, such as: 6 + 6 = 12.



Task Two: Three-bit Two Level Adder

The cascaded **FA** circuit of **Task One** above is built upon the equations of the **Full Adder** circuit of **Lab 2**, which for the *ith* stage may be expressed as:

$$S_i = A_i \ominus B_i \ominus C_i$$

$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$$

The resulting circuit is slow, because carrys must ripple through the stages of the adder. In particular, the *ith +1* stage depends upon the carry of the prior *ith* stage. We will eliminate the carry propagation by reducing the 4-bit ripple adder of **Task One** to two levels of logic. To do this, we will substitute the equations of one stage into the equations of the next stage and repeat this process for all four stages. This will produce a 2-level circuit capable of adding two 4-bit numbers *in parallel*.

1. (Pre) The output equations for the first (full adder) stage S_0, and C_1 (when $i = 0$) are computed for you at right. For $i = 1$, you must substitute the expression for C_1 into the equations for S_1, and C_2; and for $i = 2$, substitute the resulting expression for C_2 into S_2 and C_3. Repeat this once more for the $i = 3$ case. The equations that result from this work should be just in terms of A_i, B_i only.
2. It would be very cumbersome to implement a schematic in ***Xilinx***, for the expressions derived in Step 1. So, your instructor will now illustrate an alternative design entry method, using the equation-based ***VHDL*** HDL language, which can directly input the design equations for S_i and C_{i+1}. Test your resulting circuit.

$C_0 = 0$, so

$i = 0$: $S_0 = A_0 \oplus B_0$, $C_1 = A_0 \cdot B_0$

$i = 1$: $S_1 = ?$, $C_2 = ?$

$i = 2$: $S_2 = ?$, $C_3 = ?$

$i = 3$: $S_3 = ?$, $C_4 = ?$

Task Three: Carry Lookahead Adder

Suggested reading assignment: 'carry lookahead' at end of Section 5.1 of text. From the results of doing **Task Two**, it can be seen that the two-level expressions for the parallel outputs S_i and C_{i+1}, are very cumbersome. The problem is the increasingly complex expressions for the C_i that result when the equations are expanded out. Fortunately, the complexity of the carry outputs may be reduced by using the idea of *carry-lookahead*, which helps retain the simplicity of the iterative (recursive) design of **Task One** while incorporating some of the two-level nonrecursive design speed of **Task Two**. Here is how we will do this: define, for the ith stage, a *carry-generate* signal $G_i = A_i * B_i$ (i.e. generate a carry if both input bits are high) and *carry-propagate* signal $P_i = A_i + B_i$ (i.e. propagate a carry if either input bit is high). In terms of the carry generate and propagate definitions, the carryout condition may now be expressed recursively as $C_{i+1} = G_i + P_i * C_i$.

1. (Pre) Using the carry-lookahead expressions, and assuming that $C_0 = 0$, derive expressions for each of the carries C_{i+1}, for $0 \le i \le 3$, using the G_i and P_i expressions, by substituting forward the C_i, as in **Task Two**. Subsitute the A_i and B_i variables into the result to complete the carry derivation.
2. (Pre) Finally, write down your expressions for the S_i $0 \le i \le 3$, from **Task Two,** which will not change.
3. Enter the equations for the four-bit carry lookahead adder into ***VHDL***.
4. Simulate your fast adder circuit and test it as before and show your lab instructor the results.

$$G_i = A_i \cdot B_i$$

$$P_i = A_i + B_i$$

$$C_1 = G_0$$

$$C_2 = ?$$

$$C_3 = ?$$

$$C_4 = ?$$

$$S_i = ?$$