Lab session #4

Activity #1: Majority-vote ALU

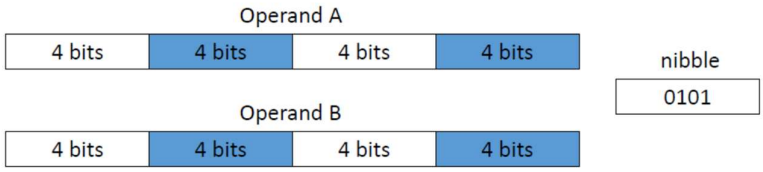
Design a hierarchical circuit consisting of 3 ALUs that perform the same operation, selected by the *ctrl* signal, on 3 distinct sets of 8-bit signals (*A* and *B* for ALU_0, *C* and *D* for ALU_1 and *E* and *F* for ALU_2). Use the ALU designed in activity #4 of Lab. 3. If:

- the 3 ALUs generate the same value, the circuit outputs that value on the 8-bit signal *decision* and the *data_valid* output is set to 1
- any 2 of the ALUs generate the same value, the circuit outputs that value on the 8-bit signal *decision* and the *data_valid* output is set to 1
- the 3 ALUs generate different values, the circuit outputs value Z on the 8-bit signal *decision* and the *data_valid* output is set to 0.

Create a suitable testbench with assertions to test the circuits on relevant values.

Activity #2: Reconfigurable ALU

Design a **reconfigurable ALU** composed of 4 4-bit ALUs processing two operands *A* and *B* whose size ranges from 4 to 16 bits. The operations are the same of the ALU designed in activity #4 of Lab. 3. An extra 4-bit input *nibble* selects the 4-bit group (a.k.a. nibble) that compose operands *A* and *B*. If the i-th bit in *nibble* is set to 1, the i-th group is part of the operand, as shown in the example of the following figure:



Modify the ALU designed in activity #4 of Lab. 3 so as to include an *enable* signal and to take into account carry-rippling between nibbles. When not enabled, each ALU outputs an all-0 result, carry-out included.

Create a testbench to simulate the circuit on relevant inputs. Use assertions.

Activity #3: Combinational adder-based multiplier

Designing a simple, though not optimal, combinational adder-based multiplier that resorts to the algorithm learned in elementary school.

The multiplication of two 4-bit numbers is illustrated in the following figure:

					a_3	a_2	a_1	a_0	**multiplicand**
×					b_3	b_2	b_1	b_0	**multiplier**
					a_3b_0	a_2b_0	a_1b_0	a_0b_0	
				a_3b_1	a_2b_1	a_1b_1	a_0b_1		
			a_3b_2	a_2b_2	a_1b_2	a_0b_2			
+		a_3b_3	a_2b_3	a_1b_3	a_0b_3				
	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	**product**

The algorithm works in 3 steps:

1. Multiply the digits of the multiplier (b_3, b_2, b_1 and b_0) by the multiplicand *A*. This corresponds to logically AND-ing b_i and the digits of *A*

2. Shift $b_i \cdot A$ to the left by i positions
3. Add column by column the shifted $b_i \cdot A$ terms to obtain the final product.

Design a **combinational adder-based multiplier** for 2 operands A and B on 8 bits. Operands are unsigned. Create a testbench to simulate the circuit on relevant inputs. Use assertions.

Activity #4: Optimized Combinational adder-based multiplier

Adders are the major components of this design. The previous implementation uses N-1 2N-bit adders. One way to reduce the size of this circuit is to add shifted $b_i \cdot A$ terms in sequence, reducing the width of the adder to N + 1 bits (see figure for 4-bit numbers).

				a_3	a_2	a_1	a_0	multiplicand
×				b_3	b_2	b_1	b_0	multiplier
				a_3b_0	a_2b_0	a_1b_0	a_0b_0	
			$pp0_4$	$pp0_3$	$pp0_2$	$pp0_1$	$pp0_0$	partial product $pp0$
		+	a_3b_1	a_2b_1	a_1b_1	a_0b_1		
		$pp1_4$	$pp1_3$	$pp1_2$	$pp1_1$	$pp1_0$		partial product $pp1$
	+	a_3b_2	a_2b_2	a_1b_2	a_0b_2			
	$pp2_4$	$pp2_3$	$pp2_2$	$pp2_1$	$pp2_0$			partial product $pp2$
+	a_3b_3	a_2b_3	a_1b_3	a_0b_3				
$pp3_4$	$pp3_3$	$pp3_2$	$pp3_1$	$pp3_0$				partial product $pp3$
$pp3_4$	$pp3_3$	$pp3_2$	$pp3_1$	$pp3_0$	$pp2_0$	$pp1_0$	$pp0_0$	product $prod$

A partial product pp_i is obtained from $b_i \cdot A$ appending an extra bit to the left of $b_i \cdot A$ to accommodate the carry out of the future addition. Note that in the addition phase only the upper bits are used. All N-1 adders are thus working on N+1 bits.

Design a **combinational adder-based multiplier** based on this remark, create a testbench to simulate the circuit on relevant inputs. Use assertions.