CE1901 LABORATORY PROJECT

SUMMARY

Addition is the fundamental arithmetic operation. Addition of binary numbers A and B relies on a basic component called a full-adder. The full-adder adds one column of A and B to produce a sum bit and a carry bit. An n-bit adder is implemented using n full-adder components.

Addition algorithms differ by how they generate the carry bits. The simplest scheme directly matches the paper and pencil approach. This scheme ripples the carry from one column to the next. Unfortunately, this ripple-carry adder (RCA) is slow because carry energy must ripple through all n stages of the number. Thus, the time complexity of this algorithm is linear. We say that the time complexity is “order n” or “big-O n” and write the complexity in mathematical symbols as O(n).

Carry look-ahead addition, on the other hand, uses algebra to expand the iterative carry equations to equations that only depend on C0, the carry that arrives with A and B at the adder inputs. These expanded logic equations use AND and OR gates to calculate all carries simultaneously and thus remove the ripple dependency. Thus, the time complexity of a carry look-ahead adder (CLA) is constant provided gates can have any number of inputs. This is written in big-O notation as O(1). This constant time algorithm is not dependent on the number of bits. It sacrifices semiconductor space for speed.

Carry-select adders compromise by using ripple carry-adders to save on gate space. Consider an 8-bit carry-select adder. The lower nibble is added using one ripple-carry. Due to the gate delays of the full adder, the carry into column four (C4) will be stable after four full-adder delays. The upper nibble does not wait for this carry to arrive. Rather, two full adders add both possible results. One full adder adds the upper nibble assuming that C4 will be 0 and the other adds the upper nibble assuming that C4 will be 1. Both results wire to a bus multiplexer. When C4 arrives, C4 energizes the select signal of the bus multiplexer to pass the correct upper nibble as the output. This technique breaks the linearity of ripple carry addition because the upper nibble does not wait. Complexity analysis for n-bit carry-select adders gives O($\sqrt{n}$).

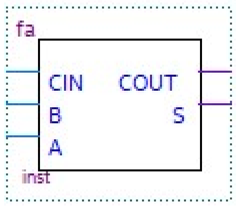
Many other addition algorithms exist. All algorithms support subtraction by using XOR gates to pass $\bar{B}$ to each full adder when C0 = 1. This results in –B presented as the second input.

This laboratory exercises focuses on ripple-carry addition.

CE1901 LABORATORY PROJECT

PRELABORATORY EXERCISES

1. **Create** a new Quartus project called RCAS4.
   a. **Use** File → New Project Wizard
   b. **Name** the project **rcas4**.
   c. This will be the only project you create.
   d. All design files will be stored in the same project.
2. **Create** a schematic full adder component.
   a. **Use** File → New → Block Diagram/Schematic File.
   b. **Add and connect** the inputs, outputs, and gates for the full adder component. **Use** the functional block symbol and truth-table to guide your K-map work.

FUNCTIONAL BLOCK SYMBOL	**INPUTS**			**OUTPUTS**	
	A	B	CIN	COUT	S
					

   c. **Save** your file as **fa.bdf.**
   d. **Use** File → Create/Update → Create Symbol Files for Current File. This will create a functional block symbol in your project folder for the full adder component you have just built.
3. **Create** a schematic 4-bit ripple-carry adder-subtractor in the same project.
   a. **Use** File → New → Block Diagram/Schematic File.
   b. **Add and connect** inputs, outputs, four XOR gates, and four full adder components. If you completed step 2d, you will have your full adder component available when you double-click the canvas to place a part. It will be available in the “Project” library of the part selection tool.
   c. **Use** the block diagram in Figure 1 to guide your work.
   d. **Save** your file as **rcas4.bdf**.
   e. RCAS4 is the top-level entity. If you did not name your project rcas4 then you will have to set rcas4.bdf as the top-level design file. **Use** Project → Set as Top-Level Entity.

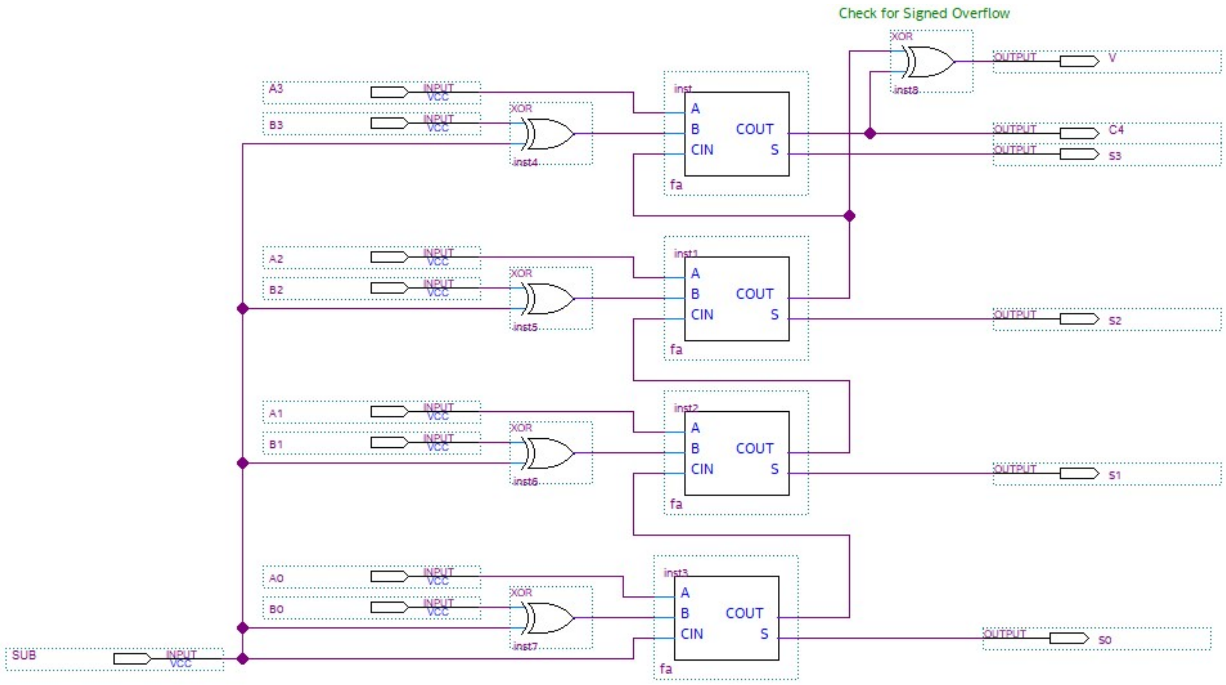


Figure 1: A 4-bit ripple-carry add-subtract circuit with signed overflow detection

4. **Simulate** your design in Quartus.
   a. **Use** File → New → University Program VWF.
   b. **Insert** all inputs and outputs.
   c. **Group** A3, A2, A1, A0, B3, B2, B1, and B0 into signed decimal busses called A and B.
   d. **Place** 16 random values on A and B using the Random Values toolbar icon.
   e. **Select** a region of time on SUB and then use the logic-1 toolbar icon to force subtraction.
   f. **Run** functional simulation to verify operation.
   g. **Use** Figure 2 as a guide for what a good simulation would look like.

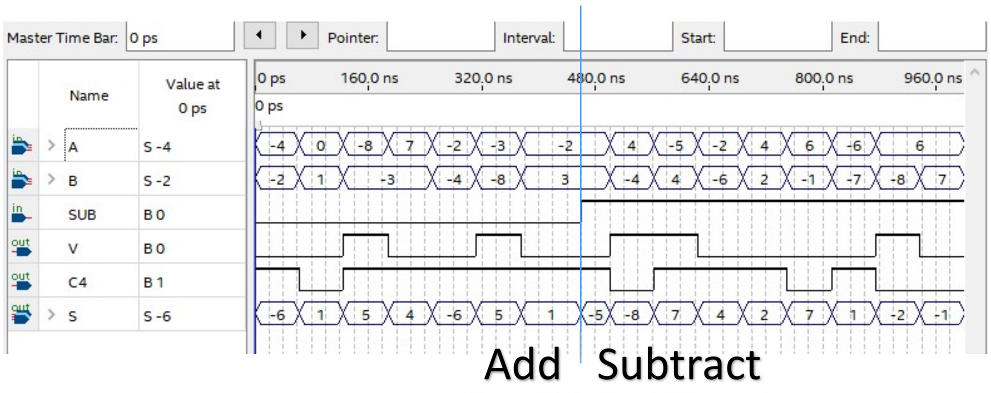


Figure 2: An RCAS4 simulation with both addition and subtraction

5. **Assign** DE10-Lite I/O slider switches and LEDs to design inputs and outputs as shown in Table 1.

Table 1: Pin Assignments

INPUT	DE10	INPUT	DE10	INPUT	DE10	OUTPUT	DE10	OUTPUT	DE10
A3	SW7	B3	SW3	SUB	SW9	S3	LEDR3	C4	LEDR8
A2	SW6	B2	SW2			S2	LEDR2	V	LEDR9
A1	SW5	B1	SW1			S1	LEDR1		
A0	SW4	B0	SW0			S0	LEDR0		

6. **Compile** and **download** to the DE10.
7. **Test** your design by completing this test table on paper first and then comparing against the DE10 results.

INPUTS				OUTPUTS		
A	B	SUB	BEHAVIOR	C4	V	SUM
3	4	0	3 + 4			
3	4	1	3 – 4			
0	1	1	0 – 1			
6	6	1	6 – 6			
4	-4	1	4 - - 4			
-5	8	0	-5 + 8			
-4	7	0	-4 + 7			
-3	5	1	-3 – 5			
2	8	0	2 + 8			
5	2	0	5 + 2			

DEMONSTRATION AND SUBMISSION

1. **Demonstrate** completed work to the instructor during the lab period.
2. **Submit** laboratory documentation through your instructor's preferred submission method.