EE316 Digital Logic Design
Department of Electrical and Computer Engineering
University of Texas at Austin

Laboratory #5
EE 316
Spring 2021

LABORATORY # 5

Calculator Design with Adders and Registers

PART 1 **Calculator utilizing Ripple Carry Adder (RCA)**

PART 2 **Calculator utilizing Carry Lookahead Adder (CLA)**

PART 3 **Speed and Area Comparison of RCA v/s CLA**

Objectives

This FPGA application development assignment builds on your knowledge of datapath component design to implement a calculator that adds two 4-bit. The numbers are to be entered using the Basys3 board switches and the result of addition will be stored on a 5-bit register controlled by a button. The number stored on the register is displayed on the LEDs.

The design of the calculator will be done using two different adder designs, namely, ripple carry adder (RCA) and carry look-ahead adder (CLA). The designs will be followed by a small analysis exercise to understand the differences in complexity and delay of the two architectures.

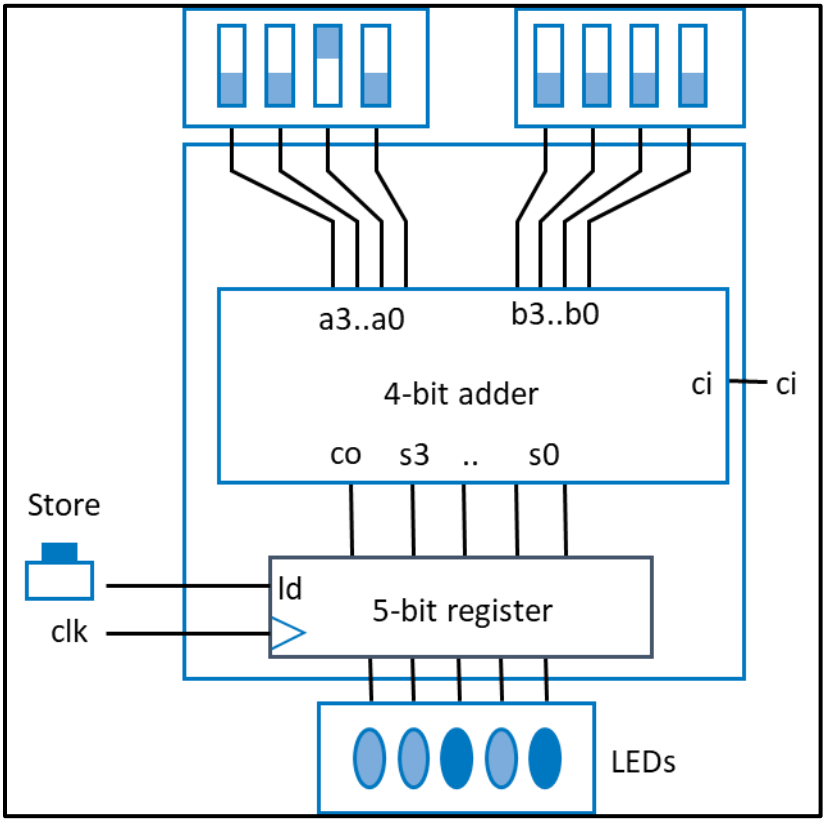

Figure 1. Adder, Register controlled by Store Button, and Output LEDs

Equipment

- PC or compatible
- Digilent's Basys3 FPGA Evaluation Board

Software

- Xilinx Vivado Design Software Suite

Part 1. Ripple Carry Adder

The basic building block of a great variety of adders is the full adder. The full adder adds two input bits (a and b) and a carry-in bit (c), and outputs a sum bit (s) and a carry-out bit (co). The adder is defined by the following equations –

$$co = bc + ac + ab$$
$$s = a \oplus b \oplus c$$

Multiple full adders can be cascaded to enable the addition of two N-bit numbers. A ripple carry adder is an adder in which the carry-out of one adder is the carry-in of the next adder. Figure 2 shows an adder design that adds two 4-bit numbers [a3…a0] & [b3…b0] and carry-in, and returns a sum and carry-out.

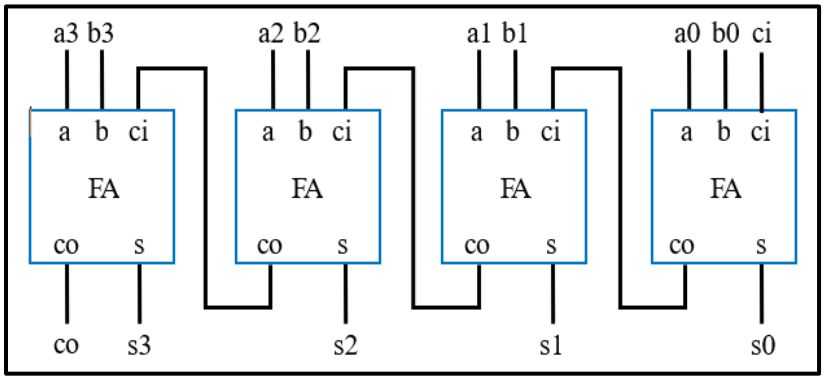

Figure 2. Ripple Carry Adder – Block Diagram

Procedure

Step 1 (Design) **-** Design a 4-bit ripple carry adder using the **data-flow style of** Verilog. The design should consist of three modules -

- Full-Adder
- 4-bit Adder
- Register Logic

As shown in Figure 1, it is required to load the result of the adder (Cout and Sum bits) into a 5-bit register (the MSB bit must be loaded with Cout and the remaining four bits with the Sum bits). Consider a 5-bit register as a set of 5 flip-flops, which loads the value on its input to the output at the positive edge of the clock if it is enabled. The I/O definitions for the modules is given in Figure 3, and this information should be used to build the code.

Note that the full_adder and register_logic blocks needs to be instantiated inside the RCA_4bits module, and hence, RCA_4bits module will be the top-level block for this design. The loaded value on the register ("*Q*") will be a concatenation of the output Sum bits and the Cout bit. The value won't be reflected on the LEDs unless the enable button is pressed.

```
module RCA_4bits(
    input clk,
    input enable,
    input [3:0] A,B,
    input Cin,
    output [4:0] Q //load registers, should contain the 4 sum bits and Cout
    );
```

```
module register_logic(
  input clk,
  input enable ,
  input [4:0] Data ,
  output reg [4:0]  Q
  ) ;
```

```
module full_adder(
    input A,B,Cin,
    output S,Cout
    );
```

Figure 3. Module I/O definitions for RCA, full adder and register logic modules

Step 2 (Verification) **–** To verify the design, create a testbench that tests the input combinations mentioned as part of Table 1.

A[3:0]	B[3:0]	Cin	Sum[3:0]	Cout
0001	0101	0	?	?
0111	0111	0	?	?
1000	0111	1	?	?
1100	0100	0	?	?
1000	1000	1	?	?
1001	1010	1	?	?
1111	1111	0	?	?

Table 1. Testcases for Ripple Carry Adder Verification

Step 3 (Synthesis and Implementation) **-** After behavioral simulation, the constraints file needs to be created according to the following mapping specifications –

- *"clk"* input port to internal clock W5
- *"enable"* input port to button U18
- *"A[3:0]"* input ports to switches SW[3:0]
- *"B[3:0]"* input ports to switches SW[7:4]
- *"Cin"* input port to switch SW8
- *"Q[4:0]"* output ports to LEDs LD[4:0]

Part 2. Carry Lookahead Adder

In this part of the lab, a faster adder called the Carry Look-Ahead Adder will be designed. In the case of the ripple carry adder, the sum and the output carry cannot be produced until the input carry of the previous stage is available. This leads to delay which grows linearly with the number of bits. The carry look-ahead adder is a faster adder: it reduces the propagation delay at the cost of using more gates. (In Part 3 of this

lab, the comparison between the delay and the complexity of the two adders needs to be done!)

Here, the basic information about the CLA design is summarized. *The lecture notes and the textbook contain a more extensive description of the CLA.*

Per-bit Propagate and Generate Variables

In addition to the sum and carry signals, the CLA introduces two additional values which are defined per bit in the following manner -

- Propagate $P_i = a_i \oplus b_i$
- Generate $G_i = a_i b_i$

The Sum and Carry outputs can now be defined in terms of the propagate and generate values in the following manner -

- Sum $S_i = P_i \oplus C_i$
- Carry $C_{i+1} = G_i + P_i C_i$

Procedure

Step 1 (Design) - To start the design of the CLA adder, the first step is to derive the equations for Sum bits (S_3, S_2, S_1, S_0) and C_4, C_3, C_2, C_1. It should be noted that C_0 will be the input carry and C_4 will be the output carry for the adder.

For example, $C_2 = P_1C_1 + G_1 = P_1(P_0C_0 + G_0) + G_1 = P_1P_0C_0 + P_1G_0 + G_1$

Once the equations are obtained, the adder needs be designed in Verilog using either Dataflow modeling or Structural modeling or a combination of both. Similar to part 1, it is required to load the result of the adder (Cout and Sum bits) into a 5-bit register so that the value is no reflected on the LEDs unless the enable button is pressed. The module I/O definitions is given in Figure 4, and this information should be used to build the code.

Note → Only the expanded equations (equations depending only on C_0) must be implemented in Verilog, as that ensures that carry look ahead adder delay is minimized.

```
module CLA_4bits(
    input clk,
    input enable,
    input [3:0] A,B,
    input Cin,
    output [4:0] Q //load registers, should contain the 4 sum bits and Cout
    );

    wire [3:0] G, P, S; //Generate, Propagate, Sum results bits
    wire [4:0] C; //Carry bits
    wire [3:0] Sum;
    wire Cout;
    assign C[0] = Cin;
```

```
module register_logic(
  input clk,
  input enable ,
  input [4:0] Data ,
  output reg [4:0]  Q
  ) ;
```

Figure 4. Module I/O definitions for CLA and register modules

Step 2 (Verification) **–** To verify the design, create a testbench that tests the input combinations mentioned as part of Table 2.

A[3:0]	B[3:0]	Cin	Sum[3:0]	Cout
0000	0101	0	?	?
0101	0111	0	?	?
1000	0111	1	?	?
1001	0100	0	?	?
1000	1000	1	?	?
1101	1010	1	?	?
1110	1111	0	?	?

Table 2. Testcases for Carry Lookahead Adder Verification

Step 3 (Synthesis and Implementation) **-** After behavioral simulation, the constraints file needs to be created according to the following mapping specifications –

- *"clk"* input port to internal clock W5
- *"enable"* input port to button U18
- *"A[3:0]"* input ports to switches SW[3:0]
- *"B[3:0]"* input ports to switches SW[7:4]
- *"Cin"* input port to switch SW8
- *"Q[4:0]"* output ports to LEDs LD[4:0]

Part 3. Comparison

In parts 1 and 2, two different techniques to add binary numbers, namely the ripple carry adder and the carry lookahead adder have been designed. This part is meant for a comprehensive understanding of the two designs, and pros & cons of each technique. The two techniques will be evaluated in terms of their speed (lesser delay for faster addition) and area. The sample values given for delay (time taken by a gate to give a stable output) and area (proportional to the number of transistors a gate uses) are given in Table 3, and need to be used for the calculation of delay and area of the two architectures.

Gate	Delay (ns)	Area
XOR	3	6
AND	3	4
OR	2	4

Table 3. Sample Delay and Area Values for Various Gates

Procedure

Step 1 (View Gate Level Schematic) **–** The first step is to generate the gate-level schematic for the design. Vivado's capability to give the schematic from the design

description will be used for the same. After the simulation of the two designs is complete, click on “Open Elaborated Design” under RTL Analysis in the left pane, as shown in Figure 5. The gate-level schematic for the design will show up. For the Ripple Carry Adder, the schematic will come up as four blocks for the four full adders. Double-clicking on each block will expand the design.

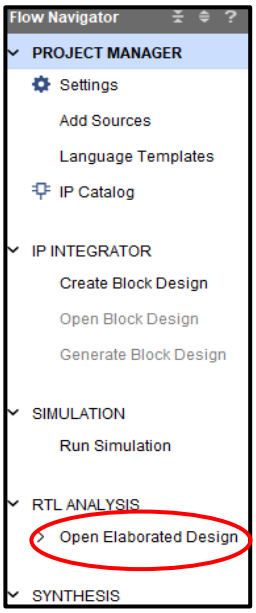

Figure 5. Opening Elaborated Design in Vivado

Step 2 (Calculating Delay and Area) – For each of the adder designs, find the delay of the critical path, i.e., the path with the maximum delay and the area using the schematic obtained and the information provided in Table 3. All the calculations need to be shown and it is required to draw a conclusion regarding the pros and cons of each of the techniques.