Introduction to SystemVerilog, FPGA, CAD, and 16-bit Adders

Hongbo Zheng, Yuhao Yuan

Summer 2021

ECE 385

Section AB1

TA: Hanfei Wang

Introduction

The purpose of this lab is to get familiar with the basic syntax and constructs of SystemVerilog, as well as acquires the basic skill required to operate Quartus Prime and CAD tool for FPGA synthesis and simulation. There are 2 portions of this lab, the first part is extending a 4-bit logic processor to an 8-bit logic processor, and the second part is implementing 3 types of adders: Ripple-Carry Adder, Carry-Select Adder, and Carry Lookahead Adder. Then, the performance analysis and optimization tools of Quartus Prime will be explored in the process of checking the area, power, and maximum operating frequencies of each adder.

8-bit Serial Logic Processor

The 8-bit serial logic operation processor is capable of calculating one of 8 functions: AND, OR, XOR, set all bits to 1, NAND, NOR, XNOR, and set all bits to 0. The circuit first parallelly loads two different sets of 8-bit data into two 8-bit registers (register A and register B). Then 1 computation cycle (8 clock cycles) after Execute switch is triggered, register A and register B will store one of the three 8-bit values: A, B, or operational result F(A,B).

Ripple-Carry Adder

The ripple-carry adder uses the carry out value from the previous unit as the carry in value of the current unit. The final result is computed 1 bit at a time which is quite slow.

Carry-Select Adder

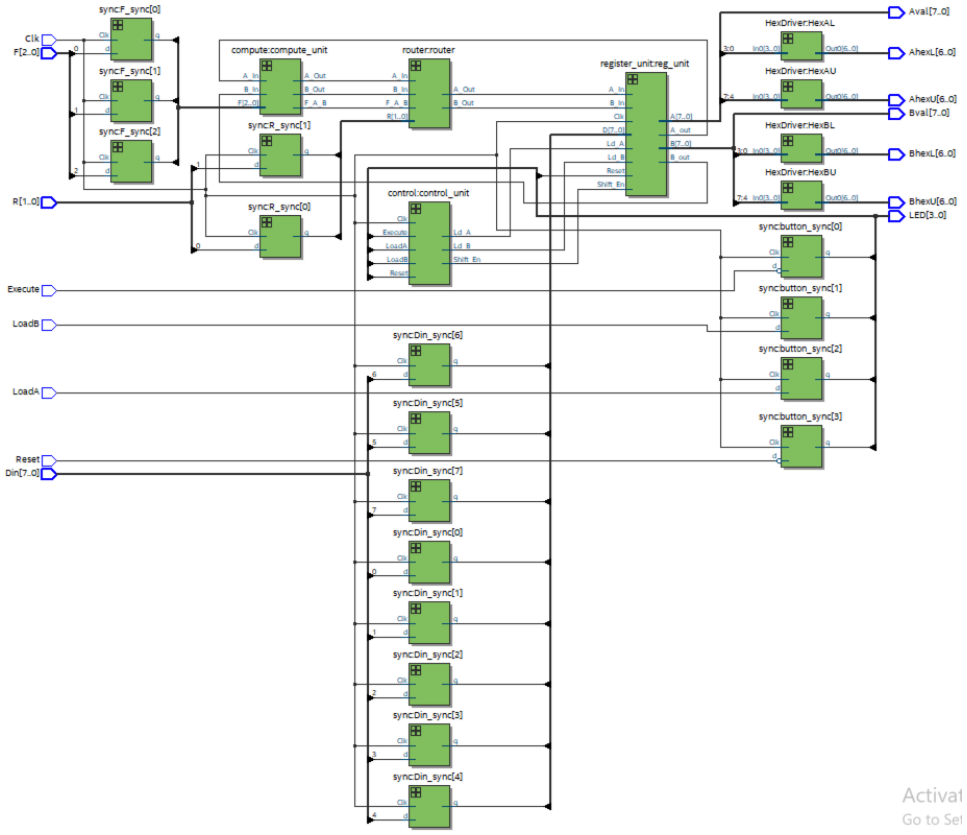
The carry-select adder has 2 sets of adders whose outcomes are pre-computed. The final result is selected using a multiplexer based on the initial carry in value.

Carry Lookahead Adder

The carry lookahead adder has the generate bit G and the propagated bit P which are calculated directly from the input A and input B. Then, the carry out value is computed from the G, P and the initial carry in value which speeds up the computation.

8-bit Serial Logic Processor

Block Diagram



Module Descriptions

Module: reg_8.sv

Input: Clk, Reset, Shift_In, Load, Shift_En, [7:0] D

Output: Shift_Out, [7:0] Data_Out

Description: This is a positive-edge triggered 8-bit right shift register with synchronous reset and synchronous load. When Reset is high, all the registers are cleared (store the binary value 0) on the positive edge of Clk. When Load is high, data is loaded from [7:0] D into the registers on the positive edge of Clk. When Shift_En is high, the values in the registers start to shift right on the positive edge of Clk, Shift_In updates the value of the left most register (register 7), and Shift_Out is the output of the right most register (register 0). Data_Out is the 8-bit output of all the register units (register 0 – register 7).

Purpose: This module is used to store 8-bit binary value of A and B in the 8-bit serial logic processor circuit.

Module: register_unit.sv

Input: Clk, Reset, A_In, B_In, Ld_A, Ld_B, Shift_En, [7:0] D

Output: A_out, B_out, [7:0] A, [7:0] B

Description: This component contains 2 positive-edge triggered 8-bit right shift registers (register A and register B) with synchronous reset and synchronous load. When Reset is high, both positive-edge triggered 8-bit right shift registers are cleared (store the binary value 0) on the positive edge of Clk. When Ld_A is high, data is loaded from [7:0] D into the 8-bit register A on the positive edge of Clk. When Ld_B is high, data is loaded from [7:0] D into the 8-bit register B on the positive edge of Clk. When Shift_En is high, the values in each register start to shift right on the positive edge of Clk, A_In updates the left most value of register A (register A7), B_In updates the left most value of register B (register B7), A_Out is the right most output of register A (register A0), and B_Out is the right most output of register B (register B0). [7:0] A is the 8-bit output of register A (register A0 – register A7). [7:0] B is the 8-bit output of register B (register B0 – register B7).

Purpose: This module is the Register Unit of the 8-bit serial logic processor. In addition, it also the high-level block of the register unit which contains 2 positive-edge triggered 8-bit right shift registers: register A and register B.

Module: compute.sv

Input: [2:0] F, A_In, B_In

Output: A_Out, B_Out, F_A_B

Description: This component contains a 1-bit 8-to-1 Multiplexer which outputs the operational result F_A_B of the ALU. In addition, A_In and B_In are not only used as two other outputs A_Out and B_Out, but also used as two inputs of the 8 ALU functions. [2:0] F is used as the functional select which is capable of choosing 1 of 8 functions: AND, OR, XOR, set all bits to 1, NAND, NOR, XNOR, and set all bits to 0.

Purpose: This module is used as the computational unit of the 8-bit serial logic processor which computes bitwise logic between A and B in 8 clock cycles based on the 3-bit functional select.

Module: Router.sv

Input: [1:0] R, A_In, B_In, F_A_B

Output: A_Out, B_Out

Description: This component contains two 1-bit 4-to-1 Multiplexers which output 1-bit A_Out and 1-bit B_Out to update the values in 8-bit register A and 8-bit register B. A_In, B_In, and F_A_B are the inputs of the 2 multiplexers. [1:0] R is used as the routing select which is capable of choosing 1 of 3 results (A, B, or F(A,B)) to update 8-bit register A and 8-bit register B.

Purpose: This module is used as the routing unit of the 8-bit serial logic processor which updates 8-bit register A and 8-bit register B with 3 options: A, B, or F(A,B), based on the 2-bit routing select.

Module: Control.sv

Input: Clk, Reset, LoadA, LoadB, Execute

Output: Shift_En, Ld_A, Ld_B

Description: This component contains both combinational logic and sequential logic. When Reset is high, the FSM is reset to its initial state. When Reset and LoadA are both high, the output Ld_A will be high which allows 8-bit register A to load 8-bit input. When Reset and LoadB are both high, the output Ld_B will be high which allows 8-bit register B to load 8-bit input. When Execute is high, Shift_En will be high for 8 clock cycles to ensure the entire operation is completed.

The Reset signal and Execute signal are finally modified to be active-low signals after connecting to the synchronizers.

Purpose: This module is used as the control unit of the 8-bit serial logic processor. The control unit makes sure the correct number of shifts are done in order for 1 computational cycle (8 clock cycles) to be completed. In addition, it controls the operating modes of register A and register B.

Module: HexDriver.sv

Input: [3:0] In0

Output: [6:0] Out0

Description: This component contains a decoder which decodes the each 4-bit binary value to its corresponding 7-bit binary LED control value. The input [3:0] In0 is translate to 7-bit LED control signals [6:0] Out0 which is capable of displaying the hexadecimal letter (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) on the LED units.

Purpose: This module is used as 7-Segment Display which can show the 4-bit hexadecimal (2-bit hexadecimal value representing 8-bit binary value in register A and 2-it hexadecimal value representing 8-bit binary value in register B) values to the user on the 4 LED units.

Module: Synchronizers.sv

Input: Clk, d

Output: q

Description: This component contains a positive-edge triggered DFF.

Purpose: This module is used to change the asynchronous signals to synchronous signals which are only triggered on the positive edge of Clk.

Module: Processor.sv

Input: Clk, Reset, LoadA, LoadB, Execute, [7:0] Din, [2:0] F, [1:0] R

Output: [3:0] LED, [7:0] Aval, [7:0] Bval, [6:0] AhexL, [6:0] AhexU, [6:0] BhexL, [6:0] BhexU

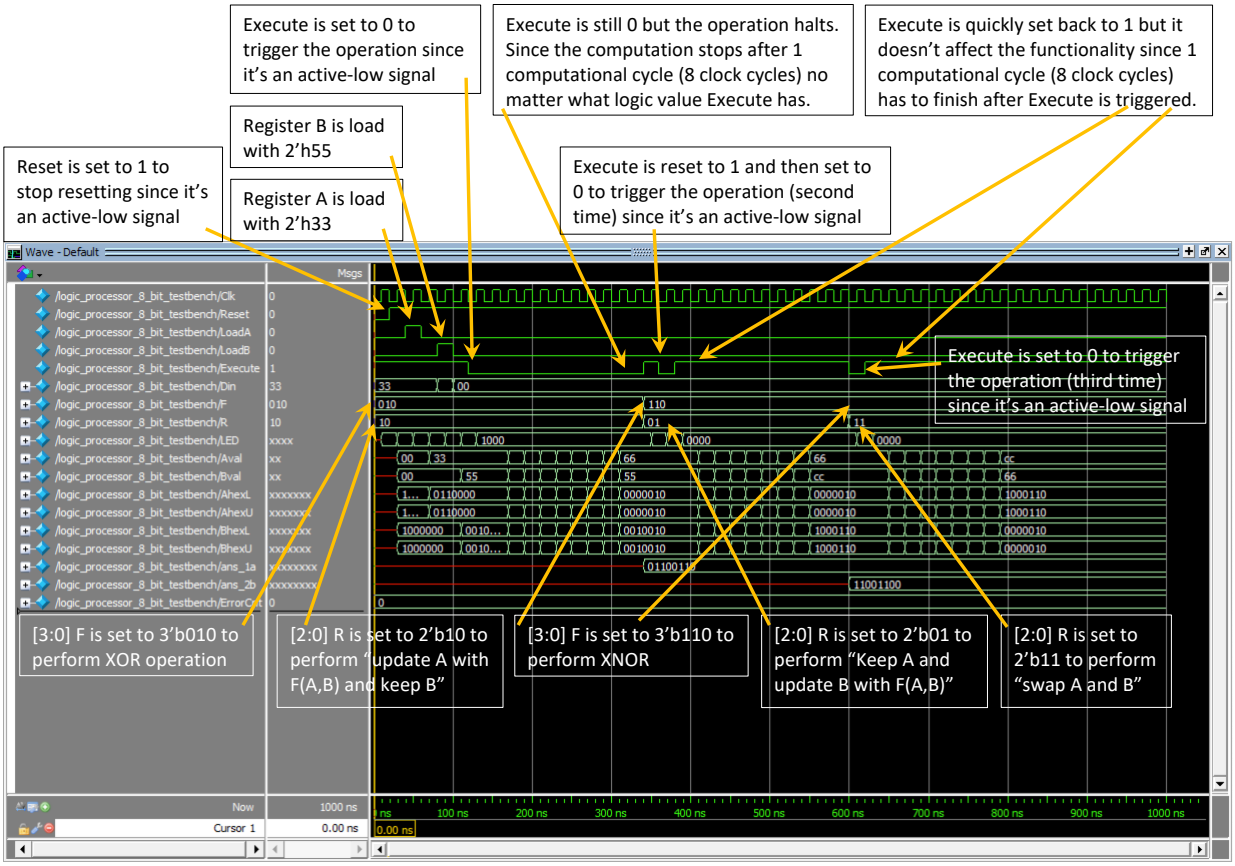
Description: This component contains register unit, computational unit, routing unit, control unit, and 7-segment display. The Reset signal and Execute signal are changed from active-high to active-low. All the inputs are all connected to synchronizers in order to change from asynchronous signals to synchronous signals. The 8-bit input [7:0] Din is the input to both register A and register B controlled by LoadA and LoadB. When Reset is low, the [7:0] Aval and [7:0] Bval are reset to 8-bit 0, and [6:0] AhexL, [6:0] AhexU, [6:0] BhexL, and [6:0] BhexU are set to 7-bit binary value which displays 0 on the LEDs. When Execute is low, the 8-bit serial logic processor starts 1 computational cycle (8 clock cycles) and then halt after finished. [2:0] F and [1:0] R are functional select and routing select. The final values of register A and register B are displayed either through [7:0] Aval and [7:0] Bval in the testbench or the decoded 7-bit values [6:0] AhexL, [6:0] AhexU, [6:0] BhexL, and [6:0] BhexU on the 4 LED units.

Purpose: This module is the high-level block of the 8-bit serial logic processor which contains register unit, computational unit, routing unit, control unit, and 7-segment display.

Changes made to the .SV files

1. input logic D and output logic Data_Out in Reg_4.sv from [3:0] to [7:0]
2. input logic D and output logic A, B in Register_unit.sv from [3:0] to [7:0]
3. input logic Din and output logic Aval in Processor.sv from [3:0] to [7:0]
4. enum logic from [2:0] to [3:0] and add G, H, I, J inside the curly braces
   F: next_state = G;
   G: next_state = H;
   H: next_state = I;
   I: next_state = J;

Quartus ModelSim Waveform



1st Execution

Initial 8-bit Aval stored in Register A: 8'h33

Initial 8-bit Bval stored in Register B: 8'h55

[2:0] F = 3'b010 --- A XOR B = 8'h66

[1:0] R = 2'b10 --- Update A with F(A,B) and Keep B

Final 8-bit Aval stored in Register A: 8'h66

Final 8-bit Bval stored in Register B: 8'h55

2nd Execution

Initial 8-bit Aval stored in Register A: 8'h66

Initial 8-bit Bval stored in Register B: 8'h55

[2:0] F = 3'b110 --- A XNOR B = 8'hCC

[1:0] R = 2'b01 --- Keep A and Update B with F(A,B)

Final 8-bit Aval stored in Register A: 8'h66

Final 8-bit Bval stored in Register B: 8'hCC

3rd Execution

Initial 8-bit Aval stored in Register A: 8'h66

Initial 8-bit Bval stored in Register B: 8'hCC

[2:0] F = 3'b110 --- A XNOR B = 8'h55

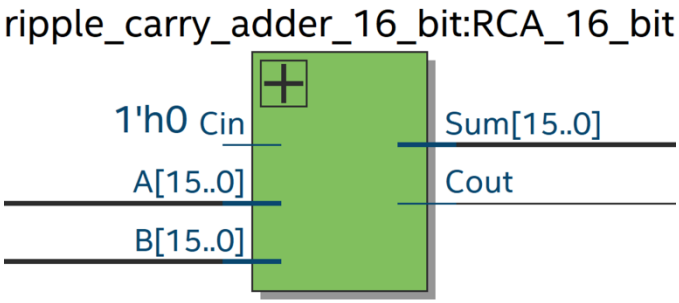
[1:0] R = 2'b01 --- Swap A and B

Final 8-bit Aval stored in Register A: 8'hCC

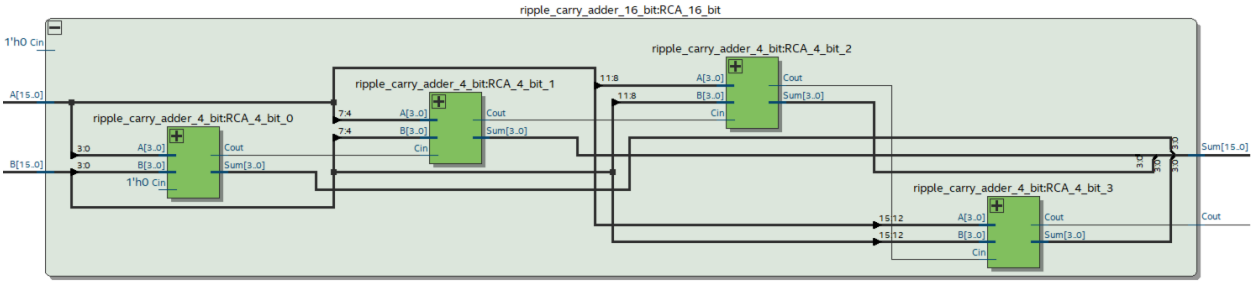
Final 8-bit Bval stored in Register B: 8'h66

16-bit Ripple-Carry Adder

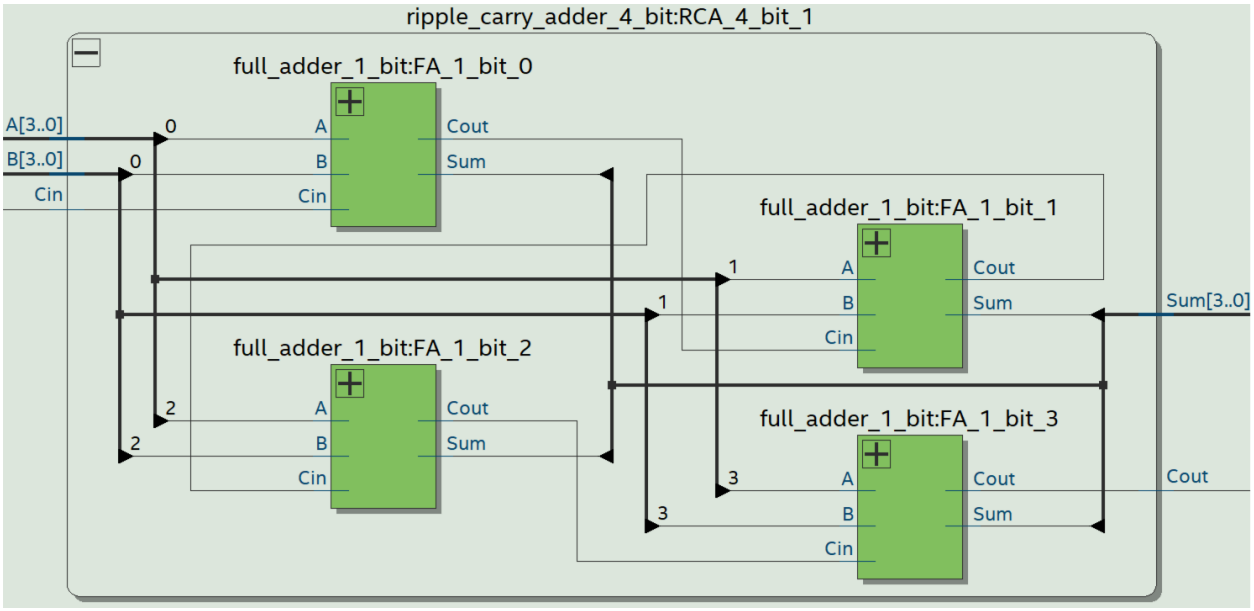
High-Level Block Diagram



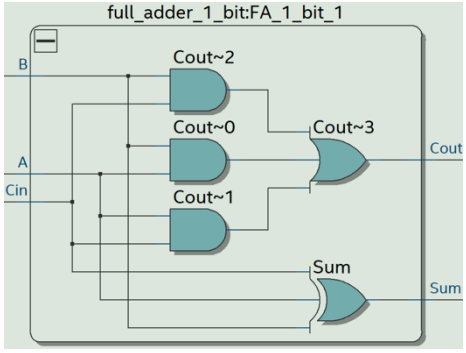
16-bit Ripple-Carry Adder Block Diagram



4-bit Ripple-Carry Adder Block Diagram



1-bit Full-Adder Block Diagram



Written Description

The ripple-carry adder is implemented by connecting a sequence of full-adders together with the carry out of one adder being the carry in of the lower-bit adder. Each full adder has 3 inputs: A, B, carry in Cin, and 1 output: carry out Cout. This design is limited in the computational speed because every Cin of the full-adder has to wait for the Cout of the lower-bit full-adder before it can correctly compute its sum and Cout.

Module Descriptions

Module: 1-bit_Full_Adder.sv

Input: A, B, Cin

Output: Sum, Cout

Description: This component is a 1-bit full-adder which add the three 1-bit inputs: A, B, Cin, and produce two 1-bit output: Sum and Cout. The boolean expression of Sum is $Sum = A \oplus B \oplus C_{in}$, and the boolean expression of Cout is $Sum = (A \cdot B) + (A \cdot C_{in}) + (B \cdot C_{in})$.

Purpose: This module is an 1-bit full-adder which is used as the fundamental building block of the 4-bit ripple-carry adder and the 4-bit carry-select adder.

Module: 4-bit_Ripple-Carry_Adder.sv

Input: [3:0] A, [3:0] B, Cin

Output: [3:0] Sum, Cout

Description: This component is a 4-bit ripple-carry adder which is consist of four 1-bit full-adder with the carry out of one adder being the carry in of the lower-bit adder. It can compute two 4-bit binary number addition: [3:0] A, [3:0] B, and the 4-bit binary result will display through [3:0] Sum and 1-bit Cout.

Purpose: This module is a 4-bit ripple-carry adder which is used as the fundamental building block of the 16-bit ripple-carry adder and the 16-bit carry-select adder.

Module: 16-bit_Ripple_Carry_Adder.sv

Input: [15:0] A, [15:0] B, Cin

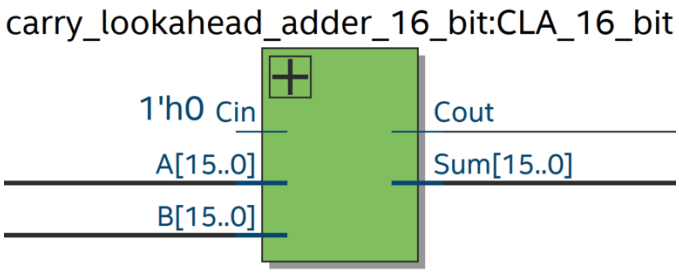
Output: [15:0] Sum, Cout

Description: This component is a 16-bit ripple-carry adder which is consist of four 4-bit ripple-carry adder with the carry out of one adder being the carry in of the lower-bit adder. It can compute two 16-bit binary number addition: [15:0] A, [15:0] B, and the 16-bit binary result will display through [15:0] Sum and 1-bit Cout.

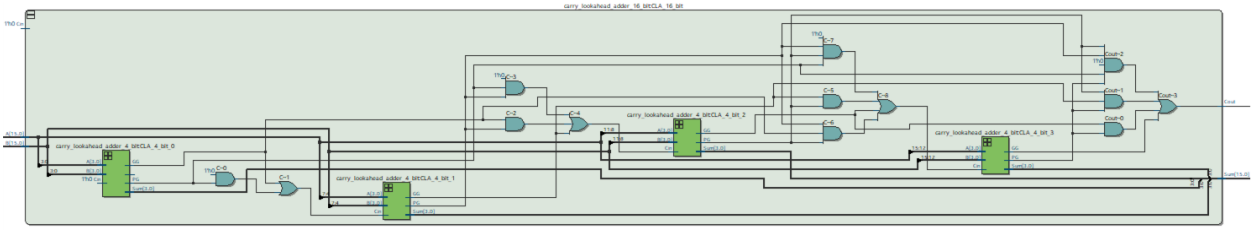
Purpose: This module is the high-level block of the 16-bit ripple-carry adder.

16-bit Carry Lookahead Adder

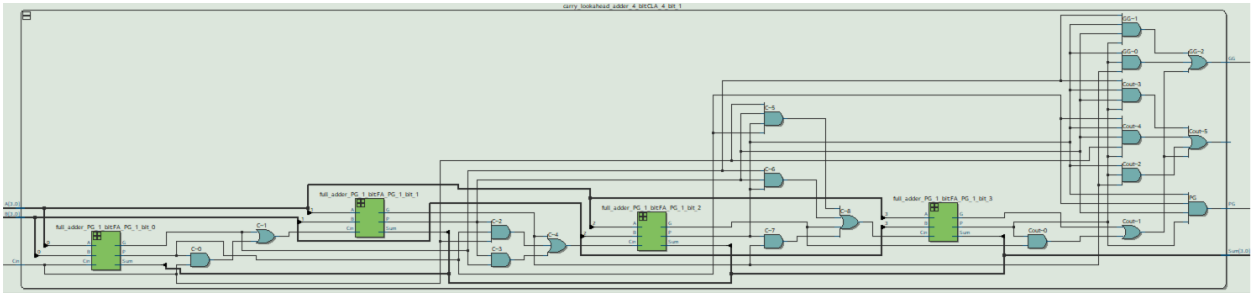
High-Level Block Diagram



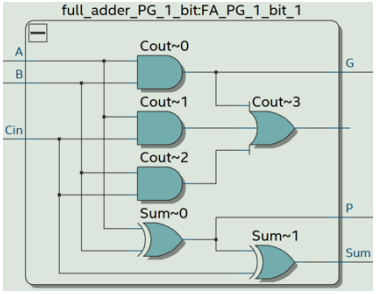
16-bit Carry Lookahead Adder Block Diagram



4-bit Carry Lookahead Adder Block Diagram



1-bit Full-Adder_PG Block Diagram



Written Description

The carry lookahead adder uses the concept of generating G and propagating P logic to predict what the value of its Cout would be for any value of its Cin. These signals are obtained from the immediate available input A and input B by using some combinational logics. A Cout is generated G if and only if both input A and input B are logic '1', regardless of the Cin. The boolean expression of G is $G(A,B) = A \cdot B$. On the other hand, there is possibility that Cout is propagated P if either A or B is logic '1'. The boolean expression of P is $P(A,B) = A \oplus B$. When G and P are calculated from the input A and input B, the boolean expression of C_{i+1} giving the value of C_i is $C_{i+1} = (P_i \cdot C_i) + G_i$. However, the boolean expression should be expanded since C_{i+1} still depends on C_i which will behave like a ripple-carry adder without any gain in speed. The example of expanding the boolean expressions of C_{i+1} is shown below.

$$C_0 = C_{in}$$

$$C_1 = C_0 \cdot P_0 + G_0 = C_{in} \cdot P_0 + G_0$$

$$C_2 = C_1 \cdot P_1 + G_1 = (C_{in} \cdot P_0 + G_0) \cdot P_1 + G_1 = C_{in} \cdot P_0 \cdot P_1 + G_0 \cdot P_1 + G_1$$

$$\begin{aligned} C_3 = C_2 \cdot P_2 + G_2 &= (C_{in} \cdot P_0 \cdot P_1 + G_0 \cdot P_1 + G_1) \cdot P_2 + G_2 \\ &= C_{in} \cdot P_0 \cdot P_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + G_1 \cdot P_2 + G_2 \end{aligned}$$

Using the method discussed above, the computation time of the carry lookahead adder is much faster than that of the ripple-carry adder. However, the boolean expression of C_{i+1} will become longer and longer after substituting. Therefore, the carry lookahead adder requires additional wide logic gates which will increase both the area and power consumption of the design.

The 16-bit carry lookahead adder has 4 groups of 4-bit carry lookahead adder, so each group of 4-bit carry lookahead adder will have 2 additional output signals called group propagate P_G and group generate G_G. The boolean expression of P_G is $P_G = P_0 \cdot P_1 \cdot P_2 \cdot P_3$, and the boolean expression of G_G is $G_G = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$. Then the Cin of each group of 4-bit carry lookahead adder are calculated using the same method which is used to obtain the Cout within the 4-bit carry lookahead adder. The boolean expressions are shown below.

$$C_0 = C_0$$

$$C_4 = G_{G0} + C_0 \cdot P_{G0}$$

$$C_8 = G_{G4} + G_{G0} \cdot P_{G4} + C_0 \cdot P_{G0} \cdot P_{G4}$$

$$C_{12} = G_{G8} + G_{G4} \cdot P_{G8} + G_{G0} \cdot P_{G8} \cdot P_{G4} + C_0 \cdot P_{G8} \cdot P_{G4} \cdot P_{G0}$$

Module Descriptions

Module: 1-bit_Full_Adder_PG

Input: A, B, Cin

Output: Sum, Cout, P, G

Description: This component is a modified version of 1-bit full-adder which has two more 1-bit outputs: G and P. The boolean expression of G is $G = A \cdot B$, and the boolean expression of P is $P = A \oplus B$.

Purpose: This module is an 1-bit full-adder with P and G signals which is used as the fundamental building block of the 4-bit carry lookahead adder.

Module: 4-bit_Carry_Lookahead_Adder.sv

Input: [3:0] A, [3:0] B, Cin

Output: [3:0] Sum, Cout, PG, GG

Description: This component is a 4-bit carry lookahead adder which is consist of four 1-bit full-adder with P and G signals. The 4-bit carry lookahead adder uses 1-bit P and 1-bit G from the 1-bit full-adder with P and G signals with some combinational logics as its Cout instead of using the normal 1-bit Cout.

Purpose: This module is a 4-bit carry lookahead adder which is used as the fundamental building block of the 16-bit carry lookahead adder.

Module: 16-bit_Carry_Lookahead_Adder

Input: [15:0] A, [15:0] B, Cin

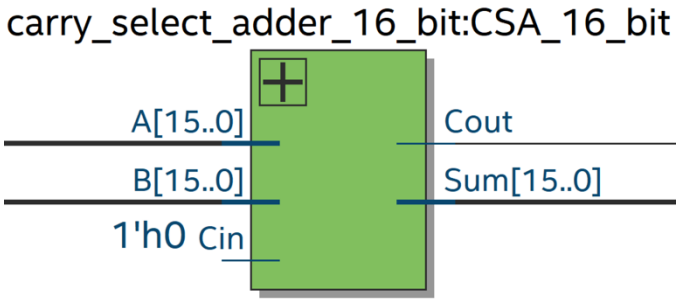
Output: [15:0] Sum, Cout

Description: This component is a 16-bit carry lookahead adder which is consist of four 4-bit carry lookahead adder with PG and GG signals. The 16-bit carry lookahead adder uses 1-bit PG and 1-bit GG from the 4-bit carry lookahead adder with some combinational logics as its Cout instead of using the normal 1-bit Cout.

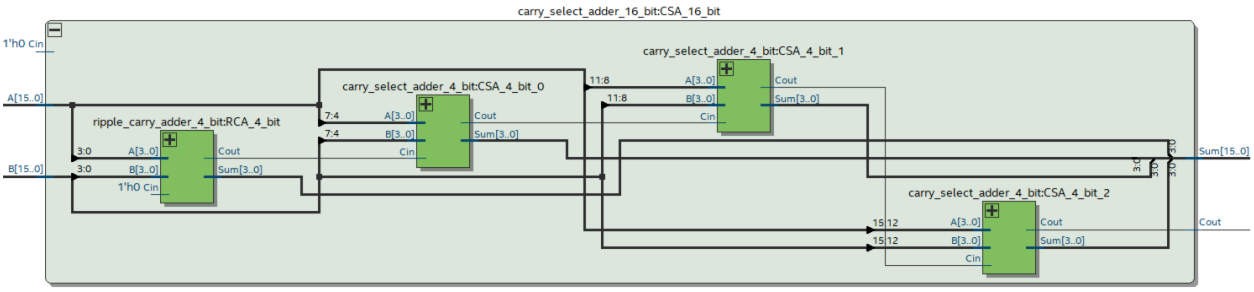
Purpose: This module is the high-level block of the 16-bit carry lookahead adder.

16-bit Carry-Select Adder

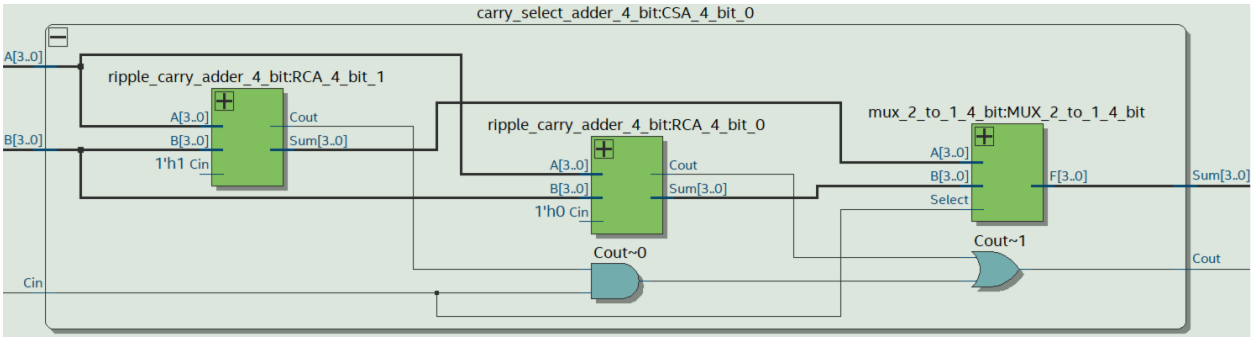
High-Level Block Diagram



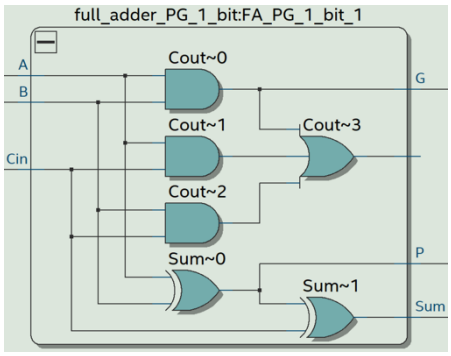
16-bit Carry-Select Adder Block Diagram



4-bit Carry-Select Adder Block Diagram



1-bit Full-Adder Block Diagram



Written Description

The carry-select adder is made of 2 ripple-carry adders and a multiplexer. One of the ripple-carry adders computes the Sum and Cout based on the assumption that the Cin is logic '0', and the other ripple-carry adder computes the Sum and Cout based on the assumption that the Cin is logic '1'. The Multiplexer takes the real Cin as the select signal to decide which Sum and Cout results to use, and the result of Cout serves the select signal for the next carry-select adder. However, starting with the second group of 4-bit carry-select adder, the Cout is calculated by first ANDing the Cin with the Cout of the 4-bit ripple-carry adder whose Cin is 1'b1, and then ORing with the Cout of the 4-bit ripple-carry adder whose Cin is 1'b0. Therefore, the 2 possible outcomes are pre-computed which speeds up the speed of calculation. However, the area and power consumption both increase significantly compared to the normal ripple-carry adder since the design requires twice the numbers of adders and several multiplexers.

Module Descriptions

Module: 4-bit_Carry_Select_Adder.sv

Input: [3:0] A, [3:0] B, Cin

Output: [3:0] Sum, Cout

Description: This component is a 4-bit carry-select adder which is consist of two 4-bit ripple-carry adders and a 4-bit 2-to-1 Multiplexer. One of the 4-bit ripple-carry adders is fed with Cin as 1'b0, and the other 4-bit ripple-carry adder is fed with Cin as 1'b1. Therefore, the 4-bit final result [3:0] Sum is chosen by the Select signal of the 4-bit 2-to-1 Multiplexer which is just the 1-bit Cin. The value of Cout is calculated by first ANDing the Cin with the Cout of the 4-bit ripple-carry adder whose Cin is 1'b1, and then ORing with the Cout of the 4-bit ripple-carry adder whose Cin is 1'b0.

Purpose: This module is a 4-bit carry-select adder which is used as the fundamental building block of the 16-bit carry-select adder.

Module: 16-bit_Carry_Select_Adder

Input: [15:0] A, [15:0] B, Cin

Output: [15:0] Sum, Cout

Description: This component is a 16-bit carry-select adder which is consist of four 4-bit carry-select adder with the carry out of one adder being the carry in of the lower-bit adder. It can compute two 16-bit binary number addition: [15:0] A, [15:0] B, and the 16-bit binary result will display through [15:0] Sum and 1-bit Cout.

Purpose: This module is the high-level block of the 16-bit carry-select adder.

Module: 4-bit_2-to-1_Multiplexer.sv

Input: [3:0] A, [3:0] B, Select

Output: [3:0] F

Description: This component is a 4-bit 2-to-1 Multiplexer which can select one of the 4-bit inputs: [3:0] A or [3:0] B as the 4-bit output [3:0] F based on the 1-bit Select.

Purpose: This module is used to build the 16-bit carry-select adder in order to choose the correct pre-computed 4-bit output result from the 4-bit carry-select adder.

Other Module Description

Module: reg_17.sv

Input: Clk, Reset, Load, [16:0] D

Output: [16:0] Data_Out

Description: The component is a 17-bit registers which is used to store the result of 16-bit adders. When Load is high, the 17-bit registers parallelly load the results of the adders. When Reset is high, the 17-bit value stored in the registers is cleared (store 17'b0).

Purpose: This module is used to store the results of the adders and display it on the LED units through the 7-segment decoder.

Module: router.sv

Input: R, [15:0] A_In, [16:0] B_In

Output: [16:0] Q_Out

Description: This component is a 17-bit multiplexer implemented using case statements

Purpose: The module is used to decide whether to put the sum of A and B or just B into the registers.

Module: control.sv

Input: Clk, Reset, Run

Output: Run_O

Description: The component is the control unit of the 16-bit adder. It has two 1-bit active-low inputs: Reset and Run. When Run is low, the adder adds the 8-bit input from the switch to the 16-bit value stored in the registers. When Reset is low, the 17-bit value stored in the registers is cleared (store 17'b0).

Purpose: The module is used to control the add and clear operation of the 16-bit adders on the DE10-Lite board.

Bugs encountered and corrective measures taken

The implementation of ripple-carry adder is pretty straight forward because the building process is just connecting four 1-bit full-adder in sequence to make it a 4-bit ripple-carry adder, and then use the 4-bit ripple carry adder as the building block for the 16-bit ripple-carry adder. Therefore, it doesn't have too many combinational logics included. In addition, the carry-select adder is not hard to implement since it consists of 2 ripple-carry adders and a multiplexer. However, the carry lookahead adder is a bit hard to build because it contains lengthy boolean expressions to compute the Ps and Gs. When the carry lookahead adder is first tested using testbench, the simulation gives the wrong results. This is quickly found to be some small mistakes in the boolean expressions of P and G because that's the most likely place to go wrong. After fixing the issues of the boolean expressions and programming each of them on the DE10-Lite board, the calculations do not seem to be correct after pressing the Execute pushbutton (KEY 1) for all of them. Since all the adders have passed several testbench cases, the mistake is suspect to be assigning the wrong PIN number to Execute. After carefully checking the Pin Planner, the Execute is assigned to some other pin which causes the error. Finally, all the three adders work correctly on the DE10-Lite board after the Execute pin is fixed.

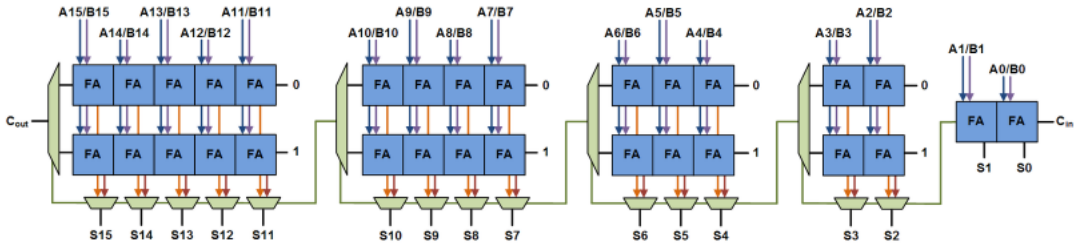
Post-Lab

The 4-bit serial logic processor schematic from lab 1 (virtual TTL chips design using Quartus) is compared with the 4-bit serial logic processor (FPGA design given by professor) after compiling the design on Quartus, since it is hard to get all the information in the table below from the physical TTL circuit, and it's also hard to extend the lab 1 schematic from a 4-bit schematic to an 8-bit schematic. The power section of TTL design is not accurate since it is not measuring the power of the physical circuit.

	TTL	FPGA
LUT	37	53
DSP	None	None
Memory (BRAM)	0	0
Flip-Flop	11	27
Frequency	362.45MHz	370.37MHz
Static Power	89.94mW	89.94mW
Dynamic Power	0.00mW	0.00mW
Total Power	98.52mW	98.66mW

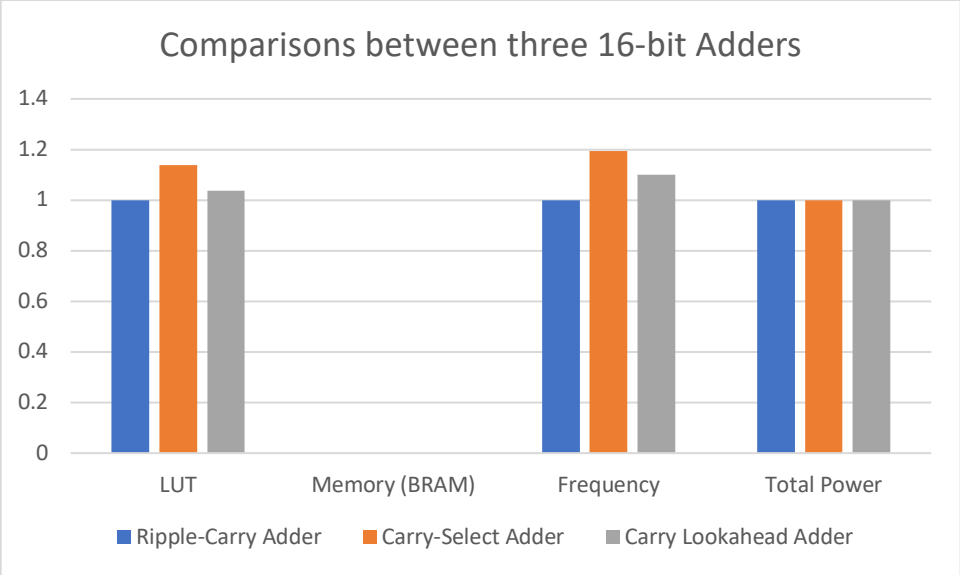
According to the table, the number of LUT and registers used in the TTL design is less than that of the FPGA design. In addition, all the combinational logics in FPGA are implemented with LUTs. For example, a 1-bit 2-to-1 Multiplexer requires 4 NAND gates for the TTL design. However, in FPGA, a 1-bit 2-to-1 Multiplexer requires a 3-input LUT which is an 8 × 1 RAM to store the results. The LUT is implemented with some combinational logics which is used to retrieve the results from the memory. Moreover, the implementations of the FSM in TTL design and FPGA design are different. In TTL design, the last 2 bits of the counter is used. However, in FPGA design, the FSM is using "one hot" encoding which requires more registers and combinational logics (more LUT). Therefore, the FPGA design takes more resources than TTL chips design.

The 4x4 hierarchy design of the carry-select adder in the lab manual is not ideal. In order to design the ideal hierarchy of carry-select adder on the FPGA, several information should be figured out, such as the propagational delay of the 1-bit full-adder, the propagational delay of the multiplexer, Frequency, and Total Power. For example, the most optimal design of 16-bit carry-select adder should have block sizes of 2-2-3-4-5 shown in the figure below when the full-adder delay is assumed to be equal to the multiplexer delay. It takes 2 clock cycles for the first two 1-bit full-adder to compute the sum and carry out. During these 2 clock cycles, the rest 4 groups of carry-select adders are also computing. After another clock cycle for the first multiplexer to produce output, the rest 3 carry-select adders are also computing. Same thing works for the second and the third multiplexer. Therefore, the total delay is 2 full-adders delay plus 4 multiplexers delay. Therefore, it is important to do experiments to figure out the delay of the 1-bit full adder and the delay of the 1-bit 2-to-1 multiplexer in order to adjust the block size of the carry-select adder.



	Ripple-Carry Adder	Carry Lookahead Adder	Carry-Select Adder
LUT	79	90	82
DSP	None	None	None
Memory (BRAM)	0	0	0
Flip-Flop	20	20	20
Frqeuency	256.08MHz	305.9MHz	281.93MHz
Static Power	89.94mW	89.94mW	89.94mW
Dynamic Power	0.00mW	0.00mW	0.00mW
Total Power	98.71mW	98.71mW	98.71mW

	Ripple-Carry Adder	Carry Lookahead Adder	Carry-Select Adder
LUT	79	90	82
Memory	0	0	0
Frequency	1	1.19455	1.10095
Power	1	1	1



The LUT section makes sense because the carry lookahead adder has the most combinational logics and the ripple-carry adder has the least combinational logics. All 3 adders do not have DSP and are not using any memory. The number of registers is the same because all of them are testing using the same circuit. The carry lookahead adder can operate under faster clock speed (high frequency) than that of the ripple-carry adder because it computes the results faster than the ripple-carry adder. The clock speed that the carry-select adder can operate is slightly lower than that of the carry lookahead adder may be because the 4x4 hierarchy design is not the optimal design of the carry-select adder. When the carry-select adder has its optimal design, it may be able to operate under approximately the same clock speed as the carry lookahead adder. All the power data are the same for the 3 adders. This happens probably because the three 16-bit adders design does not have huge differences, and they are all simple designs. However, the expect result of the power should be that the carry-select adder and the carry lookahead adder consumes more power than the ripple-carry adder because they are using more components. Therefore, besides the power section, all the data in the table and the bar chart make sense.

Conclusion

This lab not only allows us to understand the basic syntax and constructs of SystemVerilog, but also gets us familiar with the Quartus environment, such as the performance analysis of all adders. There are several difficulties of this lab. The first one is to understand the why the carry-select adder and carry lookahead adder are faster than ripple-carry adder under specific implementation. The other one is to know what the trade-offs are while designing the adders to compute faster. The last one is to figure out what is the most optimal design for the carry-select adder in terms of maximizing its computational speed. In addition, the lab manual is clear and well-written, so it helps a lot when designing the three 16-bit adders. Specifically, the boolean expressions of P and G for the carry lookahead adder is clearly explained and derived in the lab manual which is really helpful while designing the carry lookahead adder. Overall, this lab is a great introduction to the FPGA board and SystemVerilog.