Case Study for 32-Bit Pipelined CPU Design with New ALU Architecture

Adam Sumner

Illinois Institute of Technology

ECE 429-01

December 4th, 2015

Contents

1 Introduction **2**
- 1.1 Circuit Description . . . 2
- 1.2 Memory File . . . 3
- 1.3 ALU . . . 3
- 1.4 Synchronization . . . 4

2 32-Bit CPU Design with Different Adders **5**
- 2.1 Introduction . . . 5
- 2.2 Carry Ripple Adder . . . 5
  - 2.2.1 RTL Simulation . . . 6
  - 2.2.2 Logic Synthesis and Post-Synthesis Simulation . . . 7
  - 2.2.3 Place & Route and Post-P&R Simulation . . . 8
  - 2.2.4 Test Bench . . . 9
- 2.3 Carry Lookahead Adder . . . 10
  - 2.3.1 RTL Simulation . . . 11
  - 2.3.2 Logic Synthesis and Post-Synthesis Simulation . . . 11
  - 2.3.3 Place & Route and Post-P&R Simulation . . . 11
  - 2.3.4 Test Bench . . . 11
- 2.4 Carry Skip Adder . . . 11
  - 2.4.1 RTL Simulation . . . 12
  - 2.4.2 Logic Synthesis and Post-Synthesis Simulation . . . 12
  - 2.4.3 Place & Route and Post-P&R Simulation . . . 12
  - 2.4.4 Test Bench . . . 12
- 2.5 Carry Select Adder . . . 12
  - 2.5.1 RTL Simulation . . . 12
  - 2.5.2 Logic Synthesis and Post-Synthesis Simulation . . . 12
  - 2.5.3 Place & Route and Post-P&R Simulation . . . 12
  - 2.5.4 Test Bench . . . 12

3 32-Bit CPU Design with New ALU Architecture **12**

4 Conclusions **12**

1 Introduction

The objective of this project is to understand how a 32-bit pipelined Central Processing Unit (CPU) functions and how it is both physically and logically constructed. As the name suggests, the length of a word is 32 bits wide. Furthermore, due to the wonderful invention of pipelining, multiple instructions can be executed simultaneously. A CPU's normal processes (excluding interrupts) are synchronized, therefore the CPU will be in sync using an external clock signal. The instruction signals for addressing the memory file, Arithmetic Logic Unit (ALU) operands, and the ALU operation are also external.

1.1 Circuit Description

The complete overview of the primary building blocks and signals of the CPU are shown in Figure 1. The two main components are the memory file and the ALU. The external clock signal synchronizes the capture and release of data within the 32 x 32 memory file block. Because of pipelining, each instruction executed can be performed in 2 clock cycles. The first cycle involves the two decoders translating the external address selection signals used for specifying the contents of the memory file that should be read. In addition to this, the multiplexer blocks are used to select the operands for the ALU. In the second clock cycle, the ALU performs the specified operation. The output of the ALU can then be read from outside the CPU through a tri-state buffer, and they can also be written back into the memory file in the word specified by Address B.

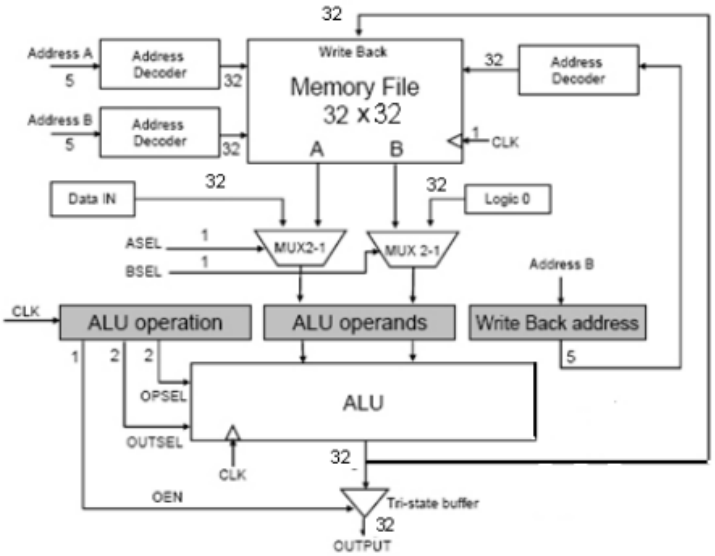

Figure 1: Circuit Overview

1.2 Memory File

The memory file stores 32 32-bit words. There are two read ports and one write port. The words that are read in each clock cycle are specified in Address A and Address B from Figure 1. The primary storage element used in this design is a D-register. The output of each register is connected to the two output read ports through tri-state buffers. The buffers are enabled through the decoded address A and address B. The value of address B specifies the word address in the memory file where the results of the ALU will be stored after the computation is complete in the second clock cycle.

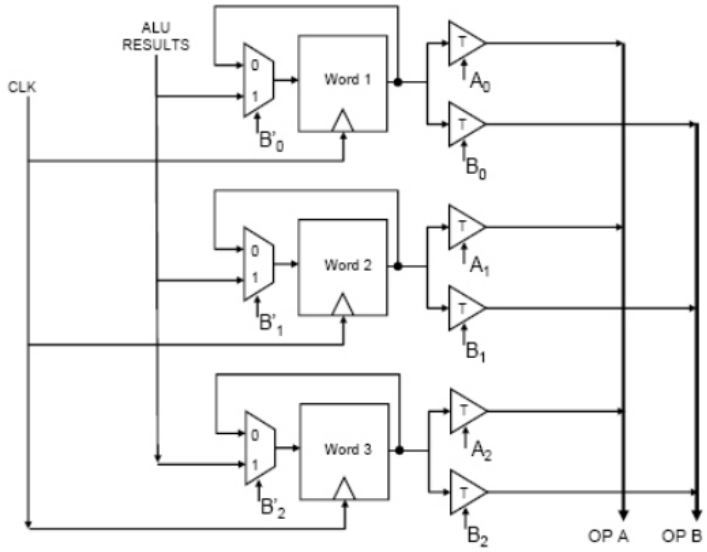

Figure 2: Memory File

1.3 ALU

The ALU of the circuit has two operands A and B, and it implements the following functions:

- A * B : multiplication
- A + B : addition
- A - B : subtraction
- B - A : subtraction
- A or B : logic OR function
- A and B : logic AND function
- A xor B: exclusive XOR function

- A xnor B : logic XNOR function

The ALU has three primary operation blocks: the multiplier, the adder, and the logic function block. This is shown in figure 3. The multiplier carries out the multiplication function. Because multiplication results in a 64-bit result, 2 clock cycles are used to store the result. This means that to execute the multiply instruction, 3 cycles instead of 2 are used in total.

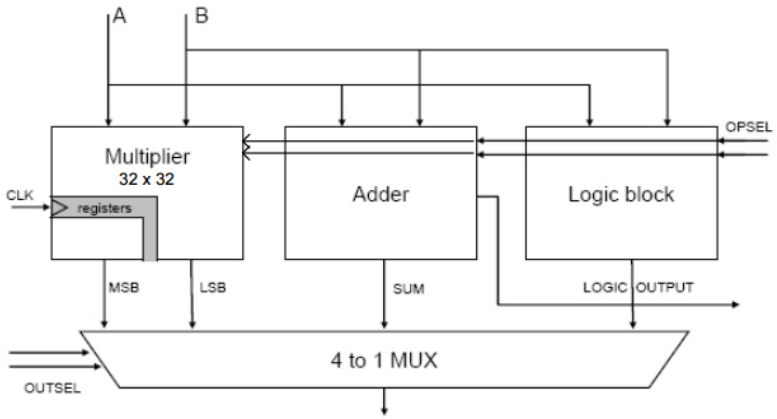

Figure 3: ALU Overview

The adder circuit within the ALU is a 32-bit adder/subtractor circuit. It executes the addition and subtraction operations. The selection of the operation is done by the two externally defined operation select signals OPSEL. The same signals are used to specify the operation executed within the logic function of the ALU. The final output select signal OUTSEL specifies the result of the ALU for output using the 4-to-1 multiplexer.

1.4 Synchronization

Each instruction executed by the CPU is determined by the external control signals. An instruction word is shown in Figure 4. Once the clock signals high, the instruction is applied to the signals that control the CPU operation. Since each instruction is executed in two steps, some control signals need to be stored at the internal registers of the CPU. In the first step of the instruction, the instruction specifies the contents of the memory file that will be read from the read ports A and B, and the operands of the ALU. In the second cycle, the control signals determine the operation to be executed internally in the ALU and the ALU output. The results of the ALU will be available if the OEN signal is set, and the result will also be written back into the memory file. This address is specified by address B. Data is written at the next positive edge of the clock. The period of the clock signal is determined by the longest data path delay in the circuit.

Figure 4: Instruction Word Contents

2 32-Bit CPU Design with Different Adders

2.1 Introduction

The objective of this project is to carry out the logical and physical synthesis of a 32-bit cpu using 4 separate types of adder circuits.

2.2 Carry Ripple Adder

An n-bit CRA is formed by concatenating n full adders in cascade with the carry output from one adder connected to the carry input of the next. This is shown in Figure 5.

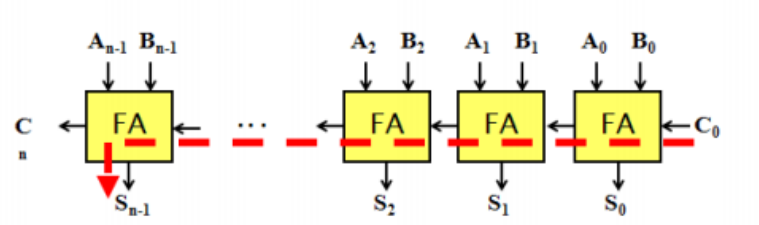

Figure 5: Carry Ripple Adder

2.2.1 RTL Simulation

```
Compiling source file "tb_cpu.v"
Compiling source file "cpu_CRA.v"

Warning!  Too few module port connections                    [Verilog-TFNPC]
          "cpu_CRA.v", 557: cra0(.sum(sum[7:0]), .c_out(c7)
          , .a(a[7:0]), .b(b[7:0]), .c_in(c_in))

Warning!  Too few module port connections                    [Verilog-TFNPC]
          "cpu_CRA.v", 558: cra1(.sum(sum[15:8]), .c_out(
          c15), .a(a[15:8]), .b(b[15:8]), .c_in(c7))

Warning!  Too few module port connections                    [Verilog-TFNPC]
          "cpu_CRA.v", 559: cra2(.sum(sum[23:16]), .c_out(
          c23), .a(a[23:16]), .b(b[23:16]), .c_in(c15))
Highest level modules:
stimulus

L30 "tb_cpu.v": $finish at simulation time 501
3 warnings
0 simulation events (use +profile or +listcounts option to count) + 21024 accelerated events
CPU time: 0.0 secs to compile + 0.1 secs to link + 0.8 secs in simulation
End of Tool:    VERILOG-XL      08.20.001-p   Dec  1, 2015  15:21:56
```

Figure 6: Display

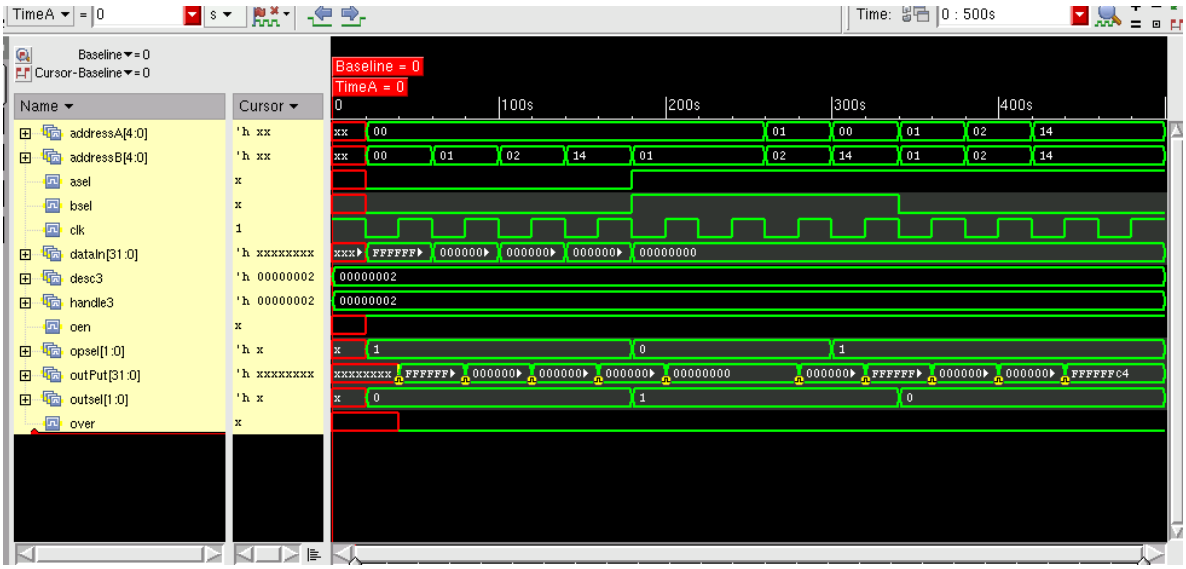

Figure 7: Simvision

2.2.2 Logic Synthesis and Post-Synthesis Simulation

```
Compiling source file "gscl45nm.v"
Compiling source file "tb_cpu.v"
Compiling source file "cpu.vh"
Highest level modules:
AOI21X1
BUFX4
CLKBUF1
CLKBUF2
CLKBUF3
DFFNEGX1
DFFSR
FAX1
HAX1
INVX2
INVX4
INVX8
LATCH
MUX2X1
NAND3X1
NOR3X1
OAI22X1
OR2X2
TBUFX1
stimulus


"gscl45nm.v", 299: Timing violation in stimulus.proj.\a/13/rc30/qout_reg
    $setup( negedge D:35999, posedge CLK:36000,  0.09 : 9 );

L30 "tb_cpu.v": $finish at simulation time 50100
0 simulation events (use +profile or +listcounts option to count) + 201694 accelerated events + 345566 timing check events
CPU time: 0.1 secs to compile + 0.3 secs to link + 0.2 secs in simulation
End of Tool:    VERILOG-XL      08.20.001-p   Dec  1, 2015  15:26:39
```

Figure 8: Display

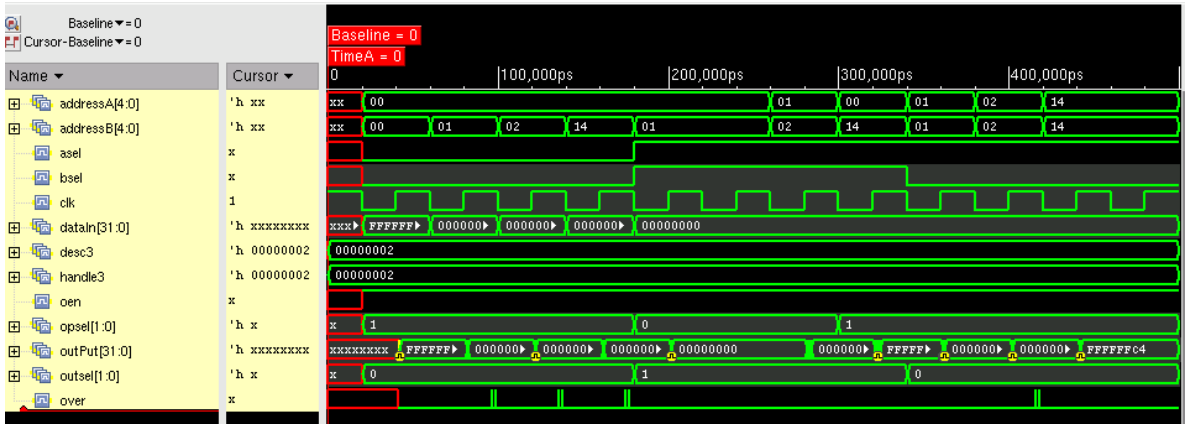

Figure 9: Simvision

2.2.3 Place & Route and Post-P&R Simulation

```
Compiling source file "gscl45nm.v"
Compiling source file "tb_cpu.v"
Compiling source file "final.v"
Highest level modules:
AOI21X1
BUFX4
CLKBUF1
CLKBUF2
CLKBUF3
DFFNEGX1
DFFSR
FAX1
HAX1
INVX2
INVX4
LATCH
MUX2X1
NAND3X1
NOR3X1
OAI22X1
OR2X2
TBUFX1
stimulus


"gscl45nm.v", 299: Timing violation in stimulus.proj.\a/13/rc30/qout_reg
    $setup( negedge D:36017, posedge CLK:36018,  0.09 : 9 );

L30 "tb_cpu.v": $finish at simulation time 50100
0 simulation events (use +profile or +listcounts option to count) + 205645 accelerated events + 345566 timing check events
CPU time: 0.2 secs to compile + 0.2 secs to link + 0.3 secs in simulation
End of Tool:    VERILOG-XL      08.20.001-p   Dec  1, 2015  15:44:00
```

Figure 10: Display

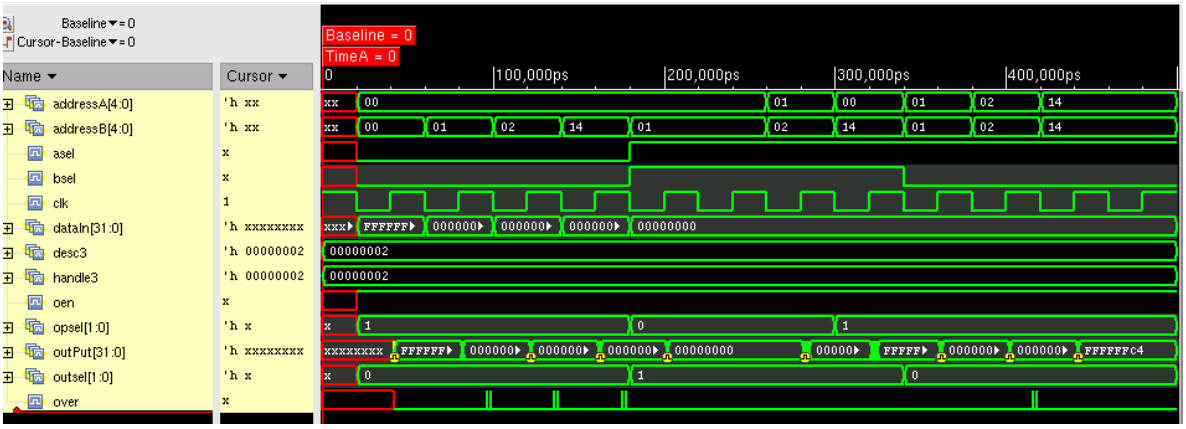

Figure 11: Simvision

2.2.4 Test Bench

```
Compiling source file "tb_test.v"
Compiling source file "cpu_CRA.v"

Warning!  Too few module port connections                    [Verilog-TFNPC]
          "cpu_CRA.v", 557: cra0(.sum(sum[7:0]), .c_out(c7)
          , .a(a[7:0]), .b(b[7:0]), .c_in(c_in))

Warning!  Too few module port connections                    [Verilog-TFNPC]
          "cpu_CRA.v", 558: cra1(.sum(sum[15:8]), .c_out(
          c15), .a(a[15:8]), .b(b[15:8]), .c_in(c7))

Warning!  Too few module port connections                    [Verilog-TFNPC]
          "cpu_CRA.v", 559: cra2(.sum(sum[23:16]), .c_out(
          c23), .a(a[23:16]), .b(b[23:16]), .c_in(c15))
Highest level modules:
stimulus

L30 "tb_test.v": $finish at simulation time 1101
3 warnings
0 simulation events (use +profile or +listcounts option to count) + 96988 accelerated events
CPU time: 0.0 secs to compile + 0.1 secs to link + 1.2 secs in simulation
End of Tool:    VERILOG-XL      08.20.001-p   Dec  1, 2015  15:46:34
```

Figure 12: RTL Display

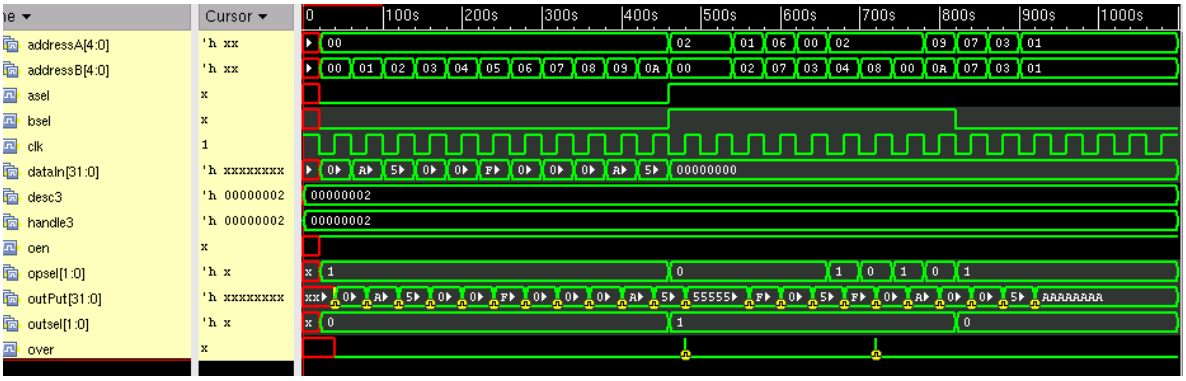

Figure 13: RTL Simvision

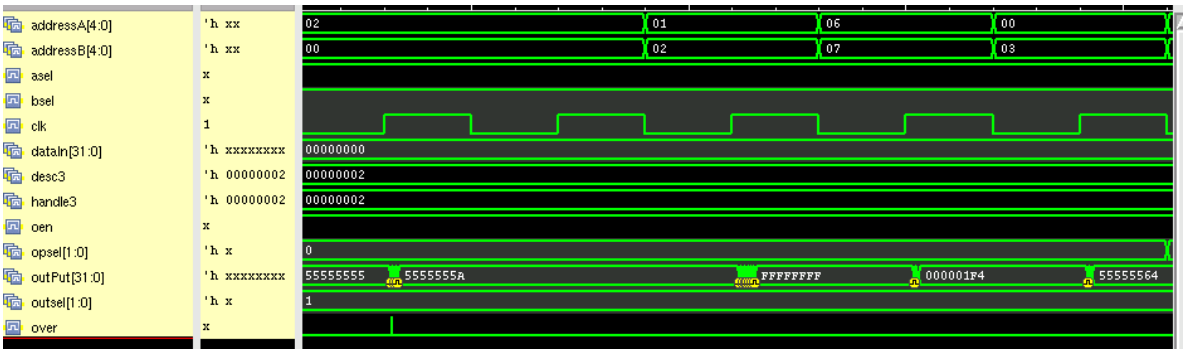

Figure 14: ALU Operations

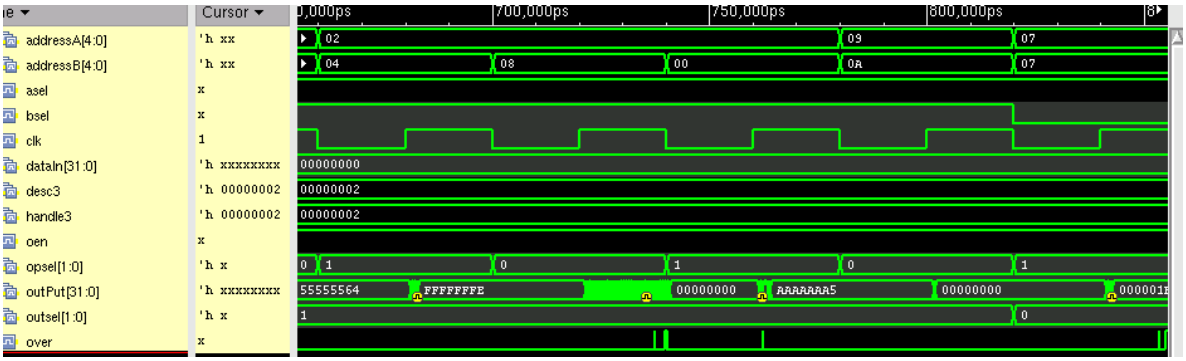

Figure 15: ALU Operations

2.3 Carry Lookahead Adder

Carry Lookahead circuits are special logic circuits that can dramatically reduce the time to perform addition at the price of more complex hardware. This is done by transforming the ripple carry design so that the carry logic over a fixed group of bits are reduced to two-level logic. This is shown in Figure 16.

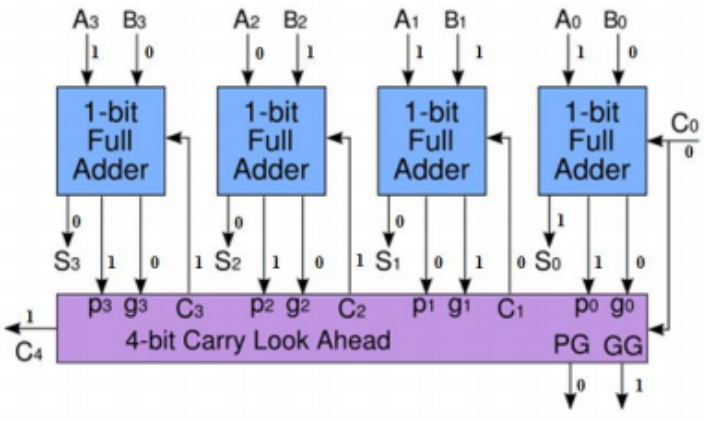

Figure 16: Carry Lookahead Adder

2.3.1 RTL Simulation

2.3.2 Logic Synthesis and Post-Synthesis Simulation

2.3.3 Place & Route and Post-P&R Simulation

2.3.4 Test Bench

2.4 Carry Skip Adder

In the Carry Skip Adder, the operands are divided into blocked of r bits blocks. In each block, a ripple carry adder can be utilized to produce the sum bits and a carryout bit for the block. This is shown in Figure 17.

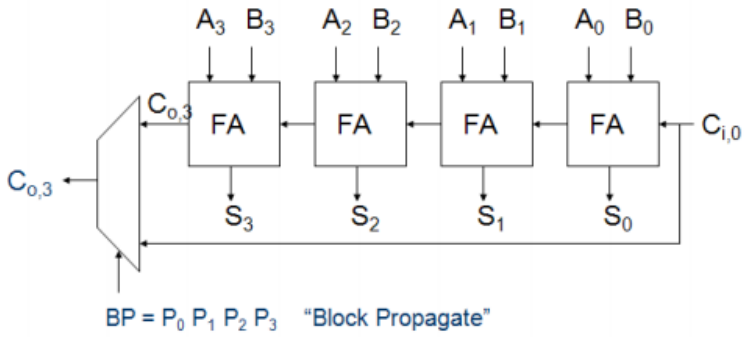

Figure 17: Carry Skip Adder

2.4.1 RTL Simulation

2.4.2 Logic Synthesis and Post-Synthesis Simulation

2.4.3 Place & Route and Post-P&R Simulation

2.4.4 Test Bench

2.5 Carry Select Adder

The Carry Select Adder divides the operands to be added into r bit blocks similar to the Carry Skip Adder. For each block, two r-bit Ripple Carry Adders operate in parallel to form two sets of sum bits and carry out signals. Each Ripple Carry Adder has two sets of hard-coded carry-in signals. One Ripple Carry Adder has a carry-in of 0, while the other has a carry-in of 1. This is shown in Figure 18.

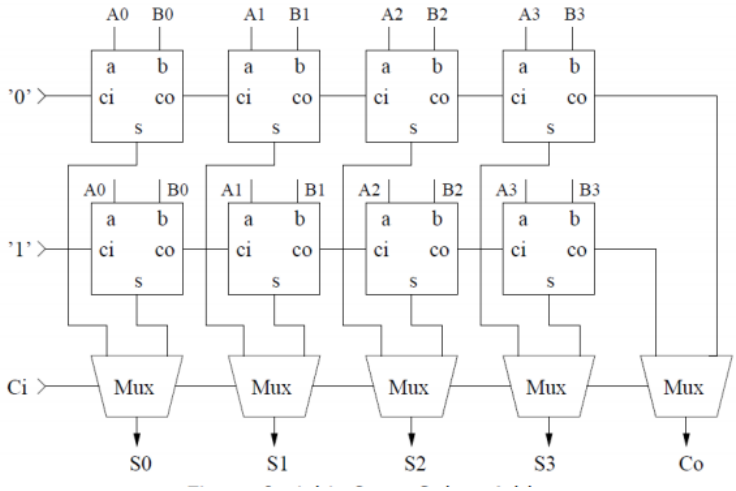

Figure 18: Carry Select Adder

2.5.1 RTL Simulation

2.5.2 Logic Synthesis and Post-Synthesis Simulation

2.5.3 Place & Route and Post-P&R Simulation

2.5.4 Test Bench

3 32-Bit CPU Design with New ALU Architecture

4 Conclusions