A

Project Report On

**32-BIT ALU USING REVERSIBLE LOGIC GATES**

Submitted in partial fulfilment of the requirements for award of the degree of

**BACHELOR OF TECHNOLOGY**

In

**ELECTRICAL AND ELECTRONICS ENGINEERING**

By

**MOUNA KRISHNA G (107117060)**

**REDDY VISHWAS (108117081)**

Under the guidance of

**Mr. SRIDHRA SHETTY**

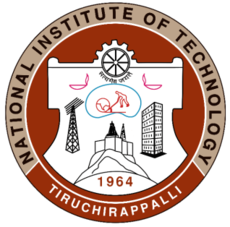
**In charge, CED**

Of

ECIL-ECIT

**ELECTRONICS CORPORATION OF INDIA LIMITED**

(A Government of India Enterprise)



**DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING, NIT TRICHY**

**CONTENTS**

1. **Abstract**
2. **Organization Profile**
3. **VLSI Introduction**
4. **FPGA Design Flow**
5. **Xilinx Procedure**
6. **Project Description**
7. **RTL Schematic**
8. **Waveforms**
9. **Applications**
10. **References**

## ABSTRACT

Reversible logic is one of the emerging technologies having promising applications in quantum computing. Reversible or information-lossless circuits have applications in digital signal processing, communication, computer graphics and cryptography. Reversibility plays an important role when energy efficient computations are considered. Reversible logic is used to reduce the power dissipation that occurs in classical circuits by preventing the loss of information. This project will deal with the design of a 32-bit reversible Arithmetic Logic Unit (ALU) with 16 operations is presented by making use of Peres Full Adder gate (PFAG gate), Fredkin gate (FRG gate), Toffoli gate, DKG gate and NOT gate. A new VLSI architecture for ALU using reversible logic gates is proposed.

ALU is one of the most important components of CPU that can be part of a programmable reversible computing device such as a quantum computer. A first single bit reversible ALU is designed and then 32 single bit ALU’s are cascaded together taking carry out of ALU performing LSB operation as an input to carry in of ALU performing next LSB operation. Design is implemented and verified in Xilinx.

# Organization profile

ECIL was setup under the department of Atomic Energy in the year 1967 with a view to generate a strong indigenous capability in the field of professional grade electronic. The initial accent was on self-reliance and ECIL was engaged in the Design Development Manufacture and Marketing of several products emphasis on three technology lines viz. Computers, control systems and communications. ECIL thus evolved as a multi-product company serving multiple sectors of Indian economy with emphasis on import of country substitution and development of products and services that are of economic and strategic significance to the country.

Electronics Corporation of India Limited (ECIL) entered into collaboration with OSI Systems Inc. (www.osi-systems.com) and set up a joint venture "ECIL\_RAPSICAN LIMITED". This Joint Venture manufacture the equipment’s manufactured by RAPSICAN, U.K, U.S.A with the same state of art Technology, Requisite Technology is supplied by RAPSICAN and the final product is manufactured at ECIL facility.

Recognizing the need for generating quality IT professionals and to meet the growing demand of IT industry, a separate division namely CED has been established to impart quality and professional IT training under the brand name of ECIT. ECIT, the prestigious offshoot of ECIL is an emerging winner and is at the fore front of IT education in the country.

**Mission**

ECIL’s mission is to consolidate its status as a valued national asset in the area of strategic electronics with specific focus on Atomic Energy, Defense, Security and such critical sectors of strategic national importance.

**Objectives**

* To continue services to the country’s needs for the peaceful uses Atomic Energy. Special and Strategic requirements of Defence and Space, Electronics Security System and Support for Civil aviation sector.
* To establish newer Technology products such as Container Scanning Systems and Explosive Detectors.
* To re-engineer the company to become nationally and internationally competitive by paying particular attention to delivery, cost and quality in all its activities.
* To explore new avenues of business and work for growth in strategic sectors in addition to working realizing technological solutions for the benefit of society in areas like Agriculture, Education, Health, Power, Transportation, Food, Disaster Management etc.

**Divisions**

The Company is organized into divisions serving various sectors, national and Commercial Importance. They are Divisions serving nuclear sector like Control & Automation Division (CAD), Instruments & Systems Division (ISD), Divisions Serving defence sector like Communications Division (CND), Antenna Products Division (APD), Servo Systems Division (SSD) etc., Divisions handling Commercial Products are Telecom Division (TCD), Customer Support Division (CSD), Computer Education Division (CED).

**Exports**

ECIL is currently operating in major business EXPORT segments like Instruments and systems design, Industrial/Nuclear, Servo Systems, Antenna Products, Communication, Control and Automation and several other components.

**Services**

The company played a very significant role in the training and growth of high calibre technical and managerial manpower especially in the fields of Computers and Information Technology. Though the initial thrust was on meeting the Control & Instrumentation requirements of the Nuclear Power Program, the expanded scope of self-reliance pursued by ECIL enabled the company to develop various products to cater to the needs of Defence, Civil Aviation, Information & Broadcasting, Tele communications, etc.

# VLSI Introduction

Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.

Hardware description language

In electronics, a hardware description language (HDL) is a specialized computer language used to program the structure, design and operation of electronic circuits, and most commonly, digital logic circuits.

A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis, simulation, and simulated testing of an electronic circuit. It also allows for the compilation of an HDL program into a lower level specification of physical electronic components, such as the set of masks used to create an integrated circuit.

Verilog

Verilog is a Hardware Description Language; a textual format for describing electronic circuits and systems. Applied to electronic design, Verilog is intended to be used for verification through simulation, for timing analysis, for test analysis (testability analysis and fault grading) and for logic synthesis.

The Verilog HDL is an IEEE standard - number 1364. The first version of the IEEE standard for Verilog was published in 1995. A revised version was published in 2001; this is the version used by most Verilog users. The IEEE Verilog standard document is known as the Language Reference Manual, or LRM. This is the complete authoritative definition of the Verilog HDL.

IEEE Std 1364 also defines the Programming Language Interface, or PLI. This is a collection of software routines which permit a bidirectional interface between Verilog and other languages.

Verilog as both a language and a simulator. At the same time, Synopsys was marketing the top-down design methodology, using Verilog. This was a powerful combination.

Modelling Styles in Verilog HDL

Normally we use Three type of Modelling Style in Verilog HDL -

1. Data Flow Modelling Style.

2. Gate level Modelling Style.

3. Behavioural Modelling Style.

1. Data Flow Modelling Style - Data Flow Modelling Style Shows that how the data / signal flows from input output through the registers / Components. Data Flow Modelling Style works on Concurrent Execution.

2. Gate level Modelling Style:

Gate level Modelling Style shows the Graphical Representation of modules/ instances / components with their Interconnection. In Gate Modelling Style We defines that how our Components / Registers / Modules are Connected to each other using Nets/ Wires. Gate level Modelling Style works on Concurrent Execution.

3.Behavioural Modelling Style -

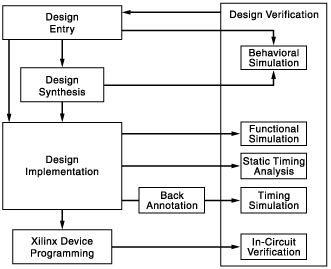
Behavioural Modelling Style shows that how our system performs according to current input values. In Behavioural Modelling, we define that what value we get at the output corresponding to input values.

We Defines the function / Behaviour of our Digital Systems in Behavioural Modelling Style.

Behavioural Modelling Style works on Sequential Execution.

# FPGA Design Flow Overview

The ISE® design flow comprises the following steps: design entry, design synthesis, design implementation, and Xilinx® device programming. Design verification, which includes both functional verification and timing verification, takes places at different points during the design flow. This section describes what to do during each step. For additional details on each design step, click on a link below the following figure.



* Design Entry
* Design Synthesis and verification
* Design Implementation and verification
* Device Programming
* In-Circuit Verification

**Design Entry**

Create an ISE project as follows:

1. Create a project.
2. Create files and add them to your project, including a user constraints (UCF) file.
3. Add any existing files to your project.
4. Assign constraints such as timing constraints, pin assignments, and area constraints.

**Functional Verification**

You can verify the functionality of your design at different points in the design flow as follows:

* Before synthesis, run behavioural simulation (also known as RTL simulation).
* After Translate, run functional simulation (also known as gate-level simulation), using the SIMPRIM library.
* After device programming, run in-circuit verification.

**Design Synthesis**

Synthesize your design.

**Design Implementation**

Implement your design as follows:

1. Implement your design, which includes the following steps:
   * Translate
   * Map
   * Place and Route
2. Review reports generated by the Implement Design process, such as the Map Report or Place & Route Report, and change any of the following to improve your design:
   * Process properties
   * Constraints
   * Source files
3. Synthesize and implement your design again until design requirements are met.

**Timing Verification**

You can verify the timing of your design at different points in the design flow as follows:

* Run static timing analysis at the following points in the design flow:
  + After Map
  + After Place & Route
* Run timing simulation at the following points in the design flow:
  + After Map (for a partial timing analysis of CLB and IOB delays)
  + After Place and Route (for full timing analysis of block and net delays)

**Xilinx Device Programming**

Program your Xilinx device as follows:

1. Create a programming file (BIT) to program your FPGA.
2. Generate a PROM or ACE file for debugging or to download to your device.  Optionally, create a JTAG file.
3. Use impact to program the device with a programming cable.

# Xilinx ISE 14.7 Procedure

Xilinx ISE means Xilinx® Integrated Software Environment (ISE). This Xilinx® design software suite allows you to take your design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes your design through several steps in the ISE design flow. These steps are Design Entry, Synthesis, Implementation, Simulation/Verification, and Device Configuration.

So, the steps are as follows;

**Open Xilinx ISE**

To open the Xilinx ISE 14.7, click on the **Xilinx** icon or **ISE Simulator** on the desktop or go to the **Start -> Programs -> Xilinx ISE Design Suit 14.7 -> ISE -> Project Navigator**. You can close the window for the "Tip of the Day".

**Create a project**

In this section, you will create a new ISE project. A project is a collection of all files necessary to create and to download a design to a selected FPGA or CPLD device.

1. Select **File > New Project**. The New Project Wizard appears.
2. First, enter a location (directory path) for the new project, then give a name for the project. For example, we name it **Counter01**.
3. Select **HDL** from the Top-Level Module Type list, indicating that the top-level file in your project will be HDL, rather than Schematic or other stuffs.
4. Click on **Next** to move to the project properties page.
5. Fill in the properties in the table as shown below:
6. Device Family: Spartan3E
7. Device: XC3S100E
8. Package: VQ100
9. Speed: -5
10. Synthesis Tool: XST [VHDL/Verilog]
11. Simulator: ISE Simulator [VHDL/Verilog]
12. Preferred Language: VERILOG
13. All the rest should be default
14. Click **Next** to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, your new project will be created.

**Create VERILOG Source**

In this section, you will create a top-level HDL file for your design. You are going to design an up-down counter which is the same as what you did in the previous lab.

Click **New Source** in the New Project Wizard to add to one new source to your project.

1. Type in the file name.
2. Select **VERILOG Module** as the source type in the New Source Dialog box.
3. Verify that the **Add to Project** checkbox is selected.
4. Click **Next**.
5. Define the ports for your VERILOG source. For example,
   * In the Port Name column, type the port names on three separate rows: **CLOCK, DIRECTION**, and **COUNT\_OUT**.
   * In the Direction column, indicate whether each port is an input, output, or inout. For CLOCK and DIRECTION, select **in** from the list. For the COUNT\_OUT, select **out** from the list.
   * To indicate that COUNT\_OUT is a 4-bit bus, use the arrows to select **3** in the MSB (Most Significant Bit) field, and select **0** in the LSB (Least Significant Bit) field.
6. Click **Next** in the Define Module dialog box.
7. Click **Finish** in the New Project Wizard - Summary dialog box to complete the new source file template.
8. If the following window appears, click on "yes".
9. Click **Next** in the New Project Wizard.
10. Click **Next** again.
11. Click **Finish** in the New Project Wizard - Project Summary dialog box.   
    ISE creates and displays the new project in the Source in Project window and adds the file to the project.

The file contains:

* + Module declaration for counter with input and output ports.

**Enter and Edit VERILOG Code**

After entering the code proceed to Checking the Syntax of the New Counter.

**Check the Syntax of your VERILOG source - Synthesize Your Code**

When source files are complete, the next step is to check the syntax of the design. Syntax errors and typos can be found using this step.

1. Select the **counter** design source in the ISE Sources window to display the related processes in the Processes for Source window.
2. Click **+** next to the Synthesize-XST process to expand the hierarchy.
3. Double-click on the **Synthesize -XST** process.   
   When an ISE process completes, you will see a status indicator next to the process name.
   * If the process completed successfully, a green check mark appears.
   * If there were errors and the process failed, a red X appears.
   * A yellow exclamation point means that the process completed successfully, but some warnings occurred.
   * An orange question marks means the process is out of date and should be run again.
4. Look in the **Console** tab of the Transcript window and read the output and status messages produced by any process you run.
5. You must correct any errors found in your source files. If you continue without valid syntax, you will not be able to simulate or synthesize your design.
6. You would like to see "Process 'synthesis' completed successfully". or "Process 'Check Syntax' completed successfully".

**Design Simulation - Simulate the Module Using the ISE Simulator**

**Create a Test Bench for Simulation**

This test bench waveform is a graphical view of a test bench. It is used with simulator to verify that the design meets both Behavioural and timing design requirements. You will use the waveform editor to create a test bench waveform (TBW) file.

1. Select the **counter** HDL file in the Sources in Project window.
2. Create a new source by selecting **project -> New Source**.
3. In the New Source window, select **Test Bench Waveform** as the source type, and fill the File Name field.
4. Make sure the box for Add to Project is checked.
5. Click **Next**.
6. The Continue File dialog box shows that you are associating the test bench with the source file: counter. Click **Next**.
7. Click **Finish**.   
   You need to set the initial values for test bench waveform in the Initialize Timing dialog box before the test bench waveform editing window opens.
8. Fill in the fields in the Initialize Timing dialog box using the information below:
   * Clock Time High: 20 ns.
   * Clock Time Low: 20 ns.
   * Input Setup Time: 10 ns.
   * Output Valid Delay: 10 ns.
   * Initial Offset: 100 ns.
   * Global Signals: GSR (FPGA). Note: The GSR value of 100 is added to the Initial Offset value automatically.
   * initial Length of Test Bench: 1000 ns.
   * Leave the remaining fields with their default values.
9. Click **Finish** to open the waveform editor.

The blue shaded areas are associated with each input signal and corresponding to the Input Setup Time in the Initialize Timing dialog box. In this design, the input transitions occur at the edge of the blue cells located under each rising edge of the CLOCK input.

1. Look at the following picture for the setup of the DIRECTION port.
2. Select **File -> Save** to save the waveform.
3. Select the **Behavioural Simulation**in the Source window.
4. On the Sources in Project Window, the TBW file is automatically added to your project.

**Simulating Behavioural Model (ISE Simulator)**

To run the integrated simulation process in ISE:

1. Select the **testbench file** waveform in the Sources in Project window. You can see Xilinx ISE Simulator processes in the Processes for Source window.
2. Double-click on the **Simulate Behavioural Model**process in the Project window. The ISE Simulator opens and run the simulation to the end of the test bench.

# Project Description

Now-a-days, the electronic systems are an integral part of a human’s life. As technology develops, the complexity of the system also increases. This gives rise to increase in power consumption, which is a great issue faced by the world today.

Energy dissipation due to information loss in high technology circuits and systems constructed using irreversible hardware was demonstrated by R. Landauer in the year 1960. According to Lindauer’s principle, the loss of one bit of information lost will dissipate kTln2 joules of energy where, k is the Boltzmann’s constant, T is the absolute temperature. In 1973, Bennett showed that to avoid kTln2 joules of energy dissipation in a circuit it must be built from reversible circuits. Reversible circuits (gates) have the same number of inputs and outputs and there is a one to one mapping between vectors of inputs and outputs. Thus, the vector of input states can be always uniquely reconstructed from the vector of output states.

Arithmetic Logic Unit (ALU)

ALU is essentially the heart of a CPU. Different kinds of computers have different ALU’s but all the ALU’s contain arithmetic unit and logic unit, which are the basic structures. This allows the computer to add, subtract, and to perform basic logical operations such as AND, OR etc.

Reversible Logic

A function is reversible if each input vector produces a unique output vector. Reversible logic is of growing importance to many future computer technologies. It is not possible to realize quantum computing without implementation of reversible logic. The main purposes of designing reversible logic are to decrease quantum cost, depth of the circuits and the number of garbage outputs. The reversible circuits form the basic building block of quantum computers as all quantum operations are reversible. Reversible logic is gaining importance in areas of CMOS design because of its low power dissipation. The traditional gates like AND, OR, XOR are all irreversible gates. Consider the case of traditional AND gate. It consists of two inputs and one output. As a result, one bit is lost each time a computation is carried out. According to the truth table shown in Fig.1, there are three inputs (1, 0), (0, 1) and (0, 0) that corresponds to an output zero. Hence it is not possible to determine a unique input that resulted in the output zero. In order to make a gate reversible additional input and output lines are added so that a one to one mapping exists between the input and output. This prevents the loss of information that is main cause of power dissipation in irreversible circuits. The input that is added to an m x n function to make it reversible is known as constant input (CI). All the outputs of a reversible circuit need not be used in the circuit. Those outputs that are not used in the circuit is called as garbage output (GO). The number of garbage output for a particular reversible gate is not fixed. The two main constraints of reversible logic circuit are

* Fan out not allowed.
* Feedbacks or loops not allowed.

Basic Definitions related to reversible logic

**Reversible logic gate:**

Reversible Gates are circuits in which number of outputs is equal to the number of inputs and there is a one to one correspondence between the vector of inputs and outputs. It not only helps us to determine the outputs from the inputs but also helps us to uniquely recover the inputs from the outputs.

**Constant inputs:**

This refers to the number of inputs that are to be maintaining constant at either 0 or 1 in order to synthesize the given logical function.

**Garbage outputs:**

Garbage is the number of outputs added to make an n-input k-output function reversible. We use the words constant inputs to denote the present value inputs that were added to an (n:k) function to make it reversible. The following simple formula shows the relation between the number of garbage outputs and constant inputs.

Input + constant input = output + garbage.

**Quantum cost:**

Quantum cost refers to the cost of the circuit in terms of the cost of a primitive gate. It is calculated knowing the number of primitive reversible logic gates (1\*1 or 2\*2) required to realize the circuit. The quantum cost of a circuit is the minimum number of 2\*2 unitary gates to represent the circuit keeping the output unchanged. The quantum cost of a 1\*1 gate is 1 and that of any 2\*2 gate is the same, which is 1.

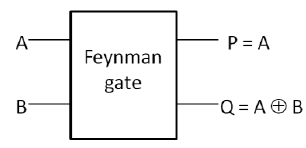
**List of reversible logic gates:**

**NOT Gate**

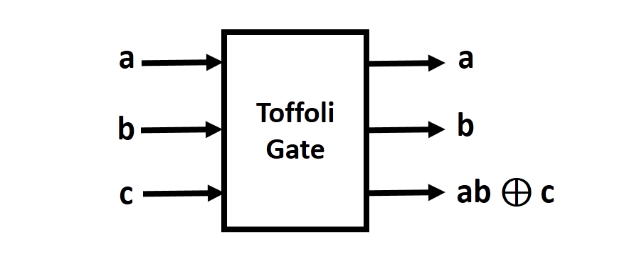
The simplest Reversible gate is NOT gate and is a 1\*1 gate. The Reversible 1\*1 gate is NOT Gate with zero Quantum Cost is as shown in the Figure. If input is A then output is P=.

P

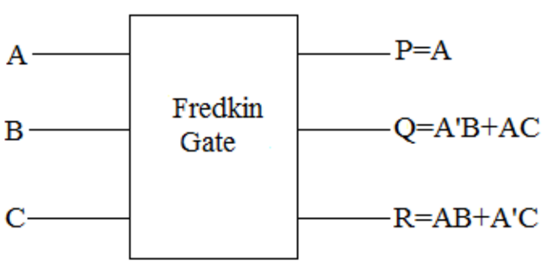
**FEYNMAN Gate**

The Feynman gate which is a 2\*2 gate and is also called as Controlled NOT and it is widely used for fan out purposes. The inputs (A, B) and outputs P=A, Q= A XOR B. It has Quantum cost one. It is the only 2x2 reversible gates available and is commonly used for fan out purposes.

**TOFFOLI Gate**

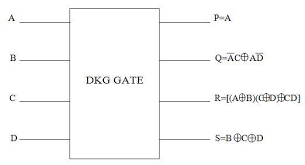
Toffoli gate is a 3\*3 gate 1with inputs (A, B, C) and outputs P=A, Q=B, R=AB XOR C. It has quantum cost of 5 and garbage outputs of 2. It is also called as CCNOT gate. It was invented by Tommaso Toffoli, is a universal reversible logic gate, which means that any reversible circuit can be constructed from Toffoli gates.

**FREDKIN Gate**

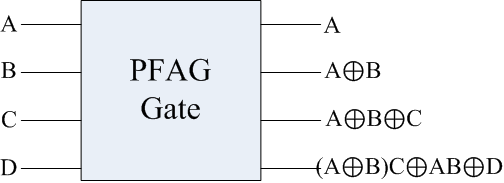
Fredkin gate which is a 3\*3 gate with inputs (A, B, C) and outputs P=A, Q=A'B+AC, R=AB+A'C. It has Quantum cost 5.

**DKG Gate**

It is 4\* 4 reversible gates that can work singly as a reversible Full adder and a reversible Full subtractor is shown in Figure. It can be verified that input pattern corresponding to a particular output pattern can be uniquely determined. If input A=0, the proposed gate works as a reversible Full adder, and if input A=1, then it works as a reversible Full subtractor. Its Quantum cost is 6.



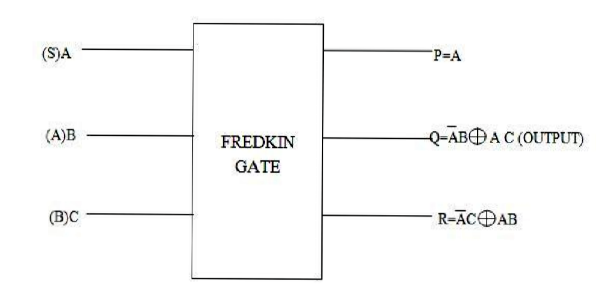
**PFAG Gate**

Peres Full Adder Gate (PFAG) shown in figure. The gate is achieved by cascading two 3\*3 Peres gate. The quantum realization cost of this gate is 8 since it includes two 3\*3 Peres gates. The gate can work singly as a reversible full adder circuit when its fourth input is set to zero (D=0). This gate requires only one clock cycle and produces no extra garbage outputs.

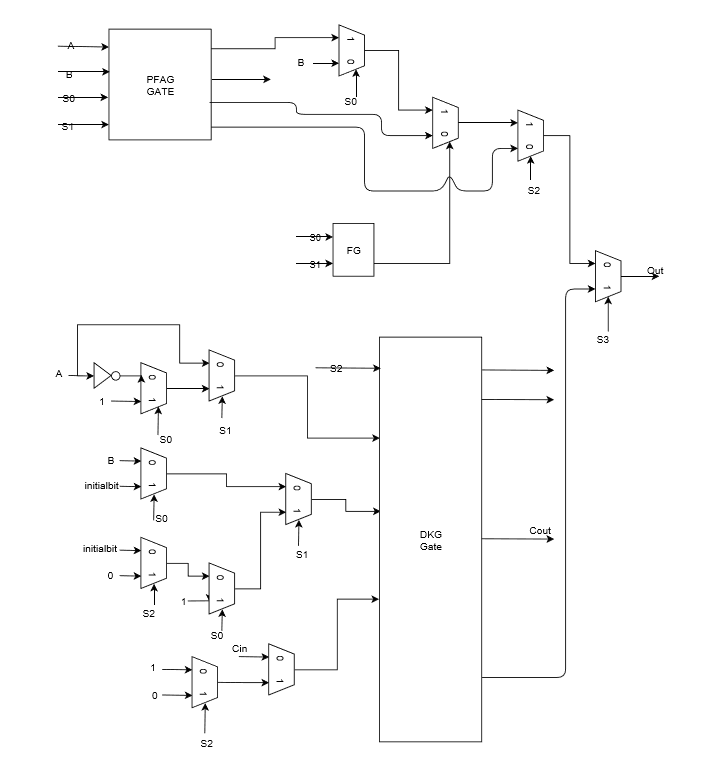
**ALU Design**

ALU has 2 parts, 1st which has PFAG Gate as base of the circuit and is selected when select line s3 is zero. The operations performed here are buffer, AND, OR, NAND, NOR, EX-OR, and EX-NOR. 2nd part has DKG Gate as base of the circuit and is selected when select line s3 is one. The operations performed here are add, increment, 2’s complement, set, subtract, decrement, not, and clear. The operations selected depending on various select lines are shown in the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operations | S3 | S2 | S1 | S0 |
| AND | 0 | 0 | 0 | 0 |
| OR | 0 | 0 | 0 | 1 |
| NAND | 0 | 0 | 1 | 0 |
| NOR | 0 | 0 | 1 | 1 |
| XOR | 0 | 1 | 0 | 0 |
| Buffer A | 0 | 1 | 0 | 1 |
| Buffer B | 0 | 1 | 1 | 0 |
| XNOR | 0 | 1 | 1 | 1 |
| Addition | 1 | 0 | 0 | 0 |
| Increment | 1 | 0 | 0 | 1 |
| 2’s Complement | 1 | 0 | 1 | 0 |
| SET | 1 | 0 | 1 | 1 |
| Subtraction | 1 | 1 | 0 | 0 |
| Decrement | 1 | 1 | 0 | 1 |
| NOT | 1 | 1 | 1 | 0 |
| CLEAR | 1 | 1 | 1 | 1 |

Multiplexer is designed using Fredkin gate when we make A as select line and (B & C) as input. B or C is selected depending on A is 0 or 1 respectively. The block diagram is shown in figure.

**BLOCK DIAGRAM OF 1-BIT ALU**



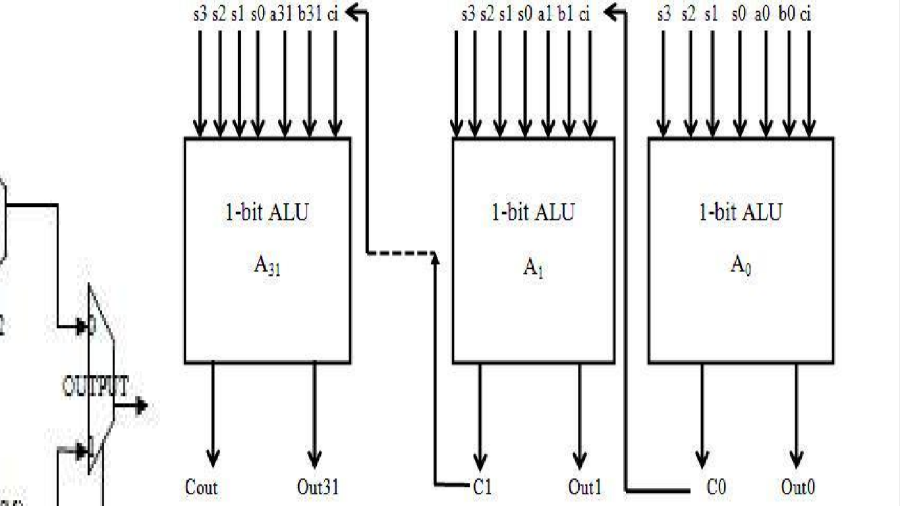
Block Diagram Explanation

* The 2:1 MUX are made using Fredkin gate (FRG).
* The Toffoli gate is used as a AND gate to produce S0.S1 product.
* The Feynman gate is used at two places to produce XOR of S0 and S1.
* The DKG gate is used as an adder/subtractor where sum is S and carry is R.
* Using the PFAG gate at the top the logic operations

AND, OR, NAND, NOR, XOR, XNOR

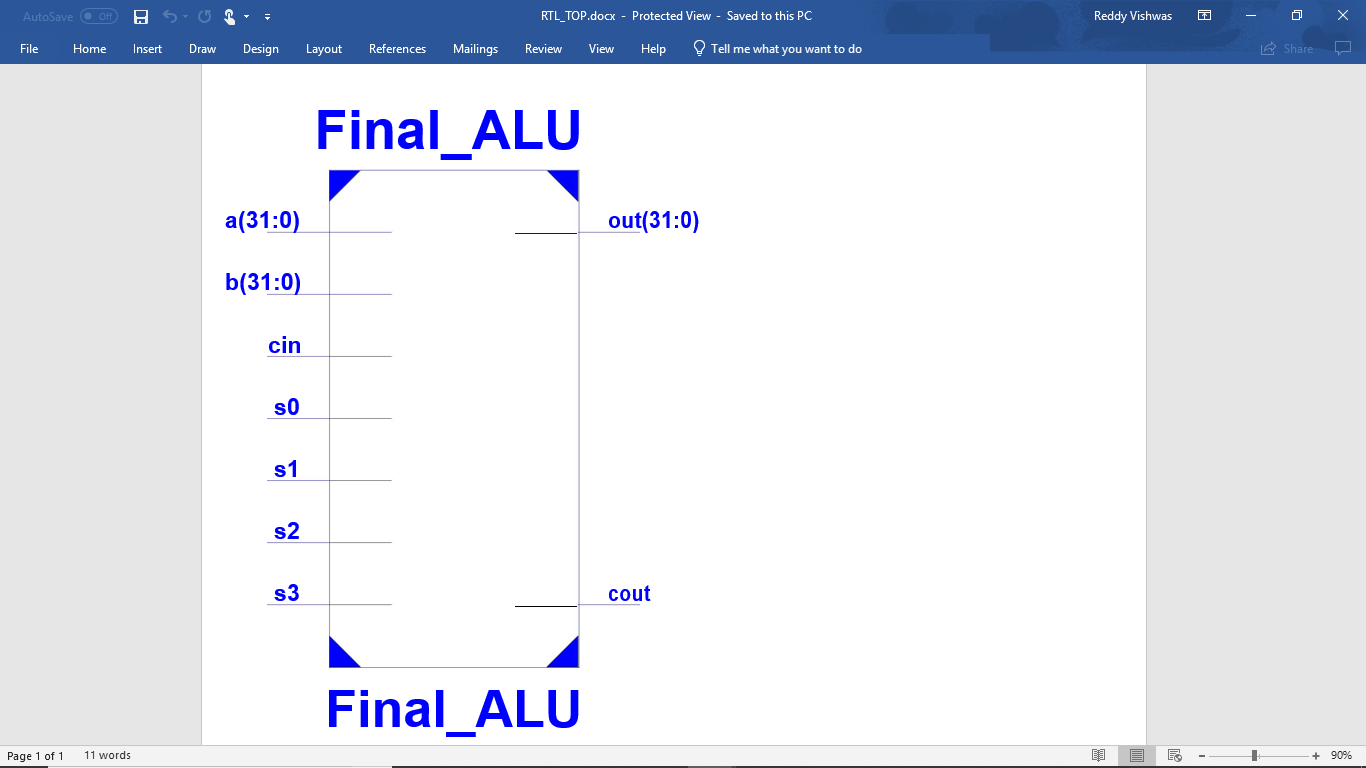
are performed with C(S1) and D(S0) as select lines with A and B as inputs.

* The BUFFER-A and BUFFER-B operation or performed using the MUXs near PFAG gate.
* The addition operation is performed by sending inputs A, B to B, C and Cin to D of DFK gate and setting A to 1’b0.
* Increment operation is performed by sending input A to B, 1’b1 to C, 0 to D of DKG in addition mode gate for the 1st bit ALU and changing C to 1’b0 for the other ALUs.
* 2’s complement is performed by sending NOT (A) to B, 1’b1 to C, 0 to D of DKG gate for the 1st bit ALU and changing C to 1’b0 for the other ALUs.
* Set operation is performed by sending 1’b1 to B, C, D of DKG gate.
* The subtraction operation is performed by sending input A, B to B, C and Bin to D pf DKG gate and setting a to 1’b1.
* Decrement operation is performed by sending input A to B, 1’b1 to C, 0 to D of DKG gate in subtraction mode for the 1st bit ALU and changing C to 1’b0 for the other ALUs.
* NOT operation is performed by sending NOT(A) to B, and 1’b0 to C, D of DKG gate.
* Clear operation is performed by sending 1’b1 to B, C and 1’b0 to D of DKG gate in addition mode.

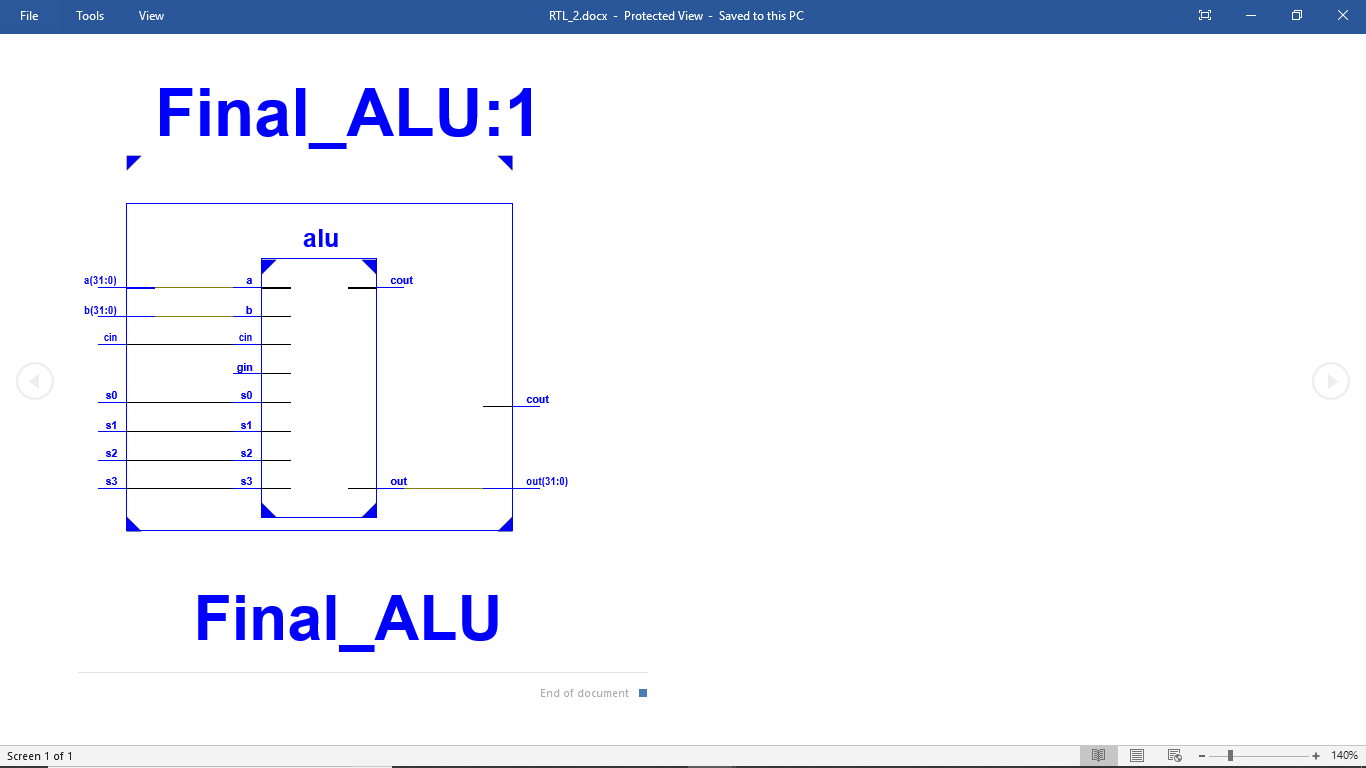
Each 1-bit ALU is cascaded as shown in the figure to form a 32-bit ALU. The initalbit value is given as 1 for first bit and for the rest of the bits it is given as zero.

# RTL Schematic

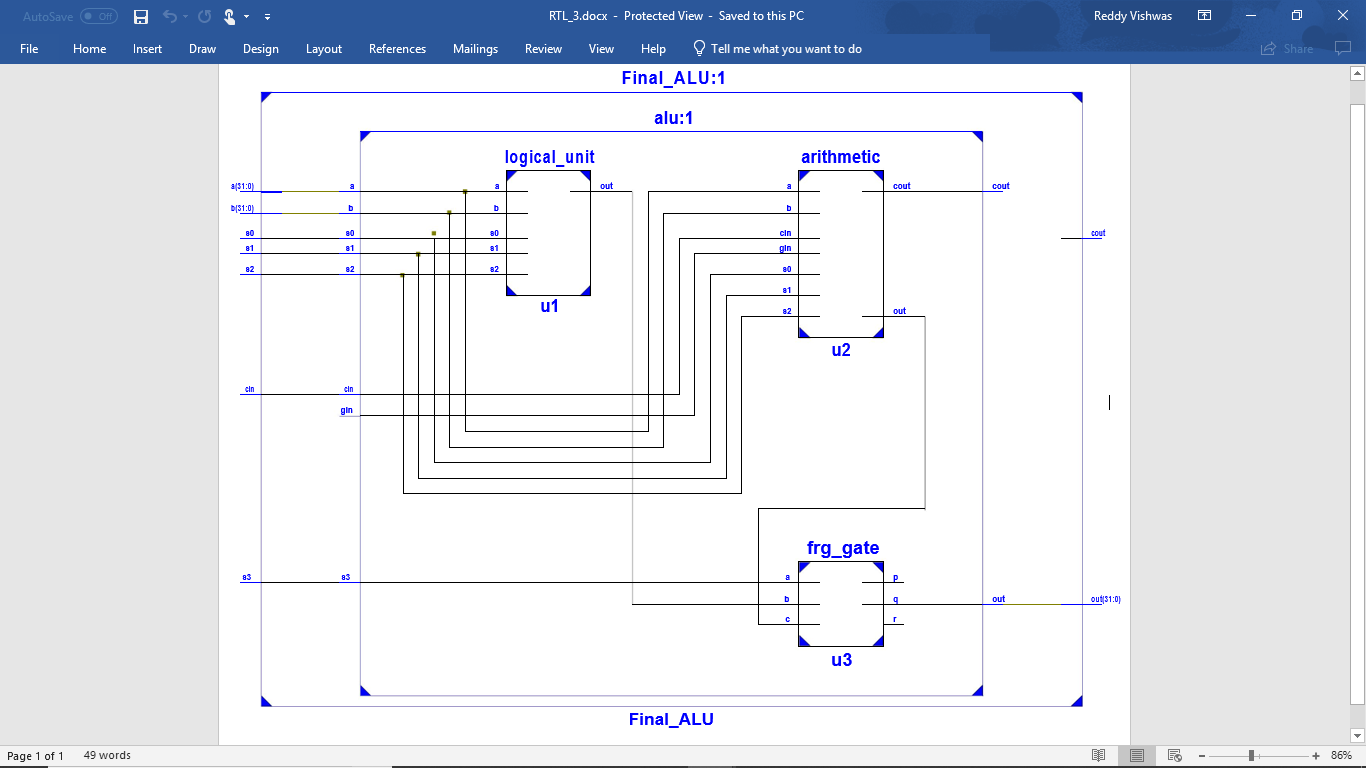
* Top level RTL



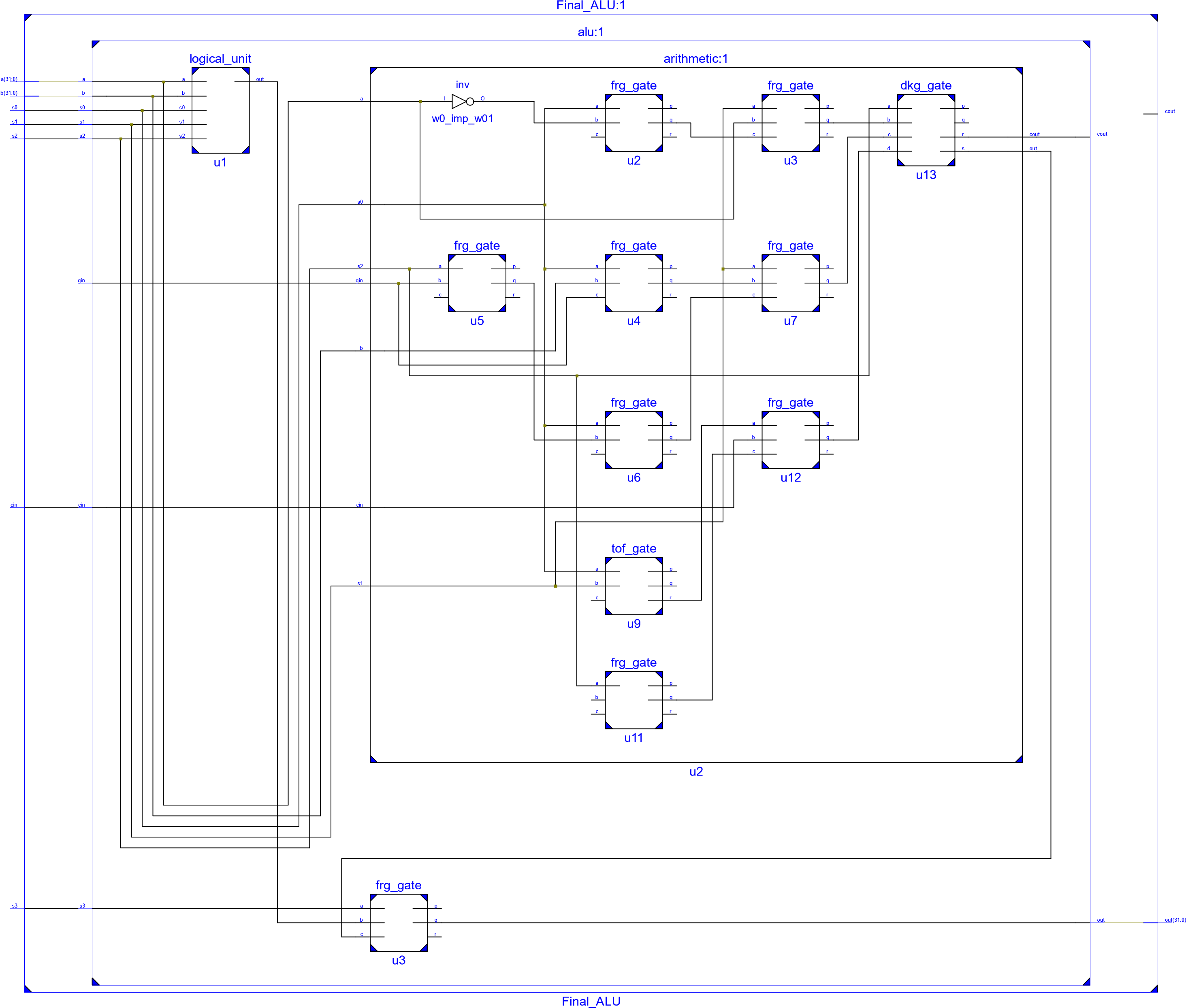
* Each 1-bit ALU



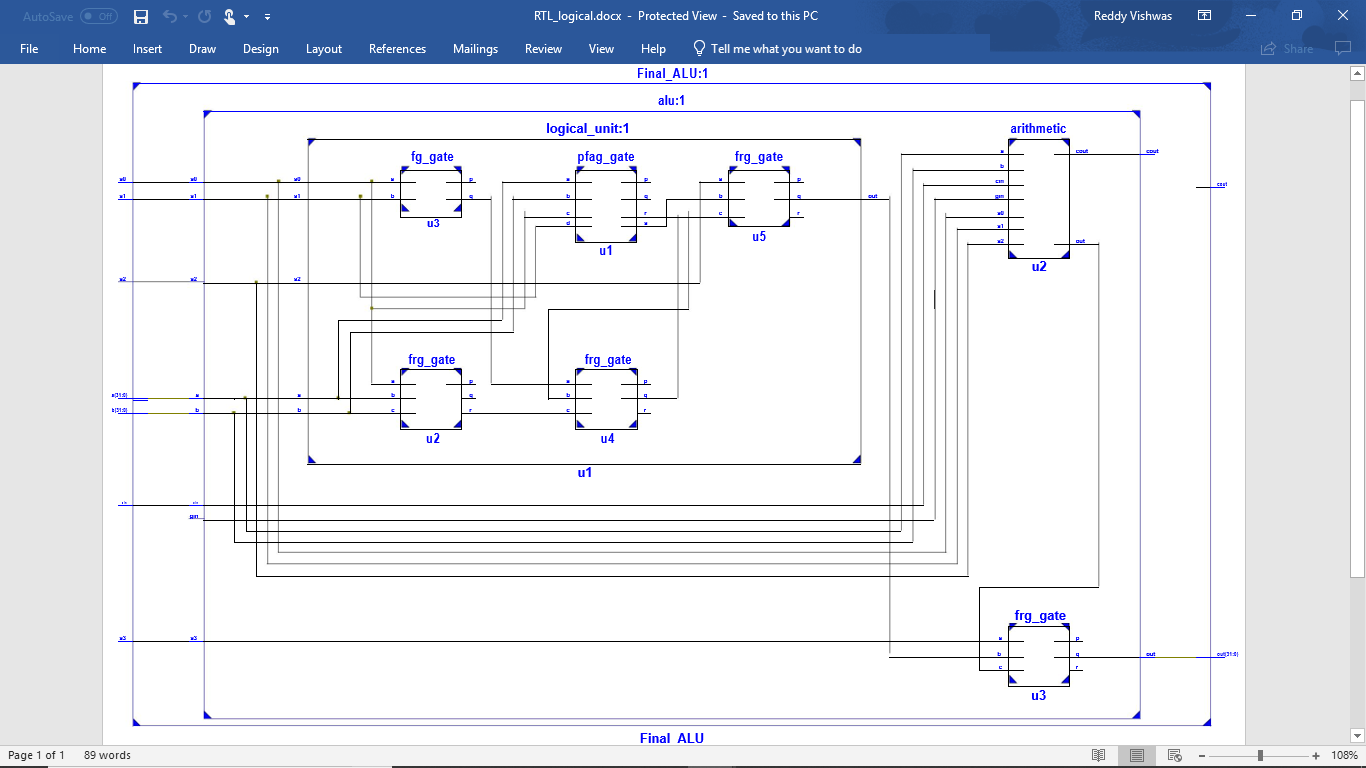
* Logical Unit and Arithmetic Units

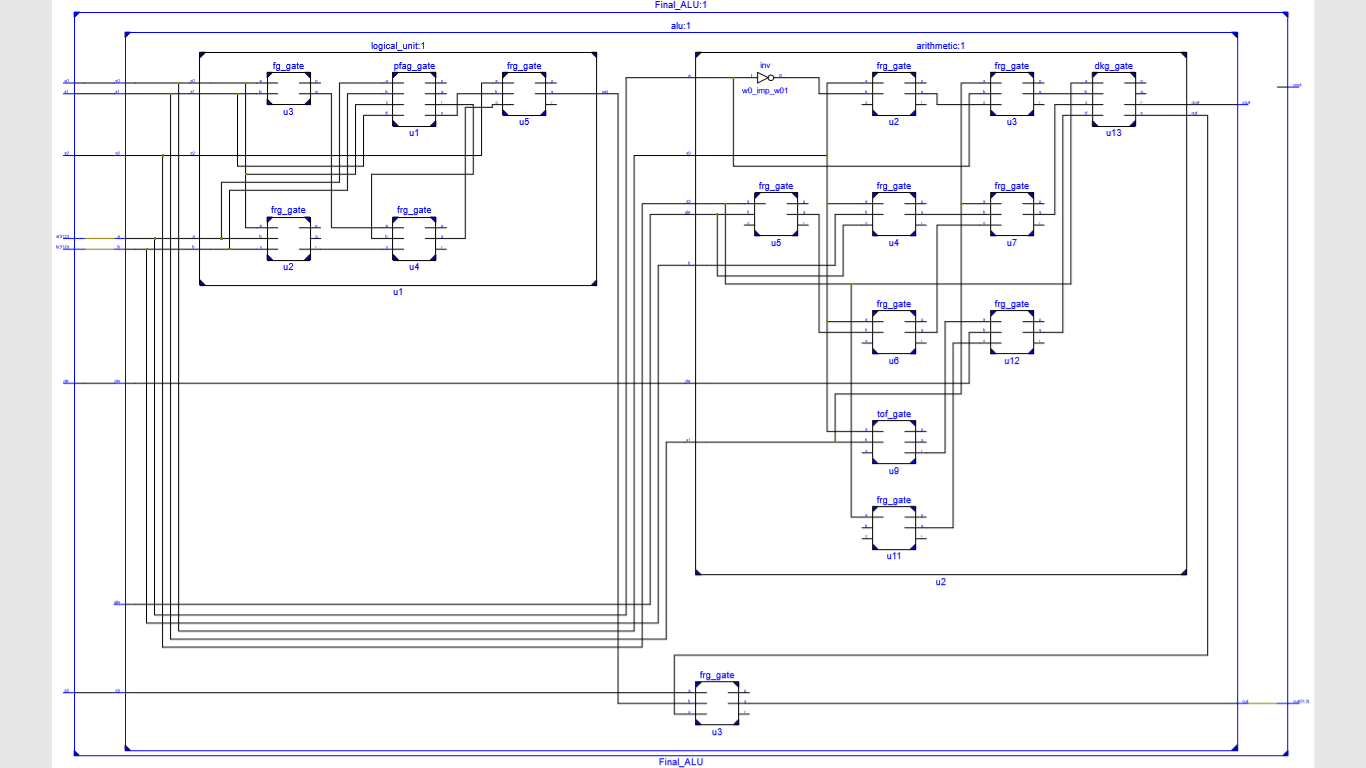


* Arithmetic unit



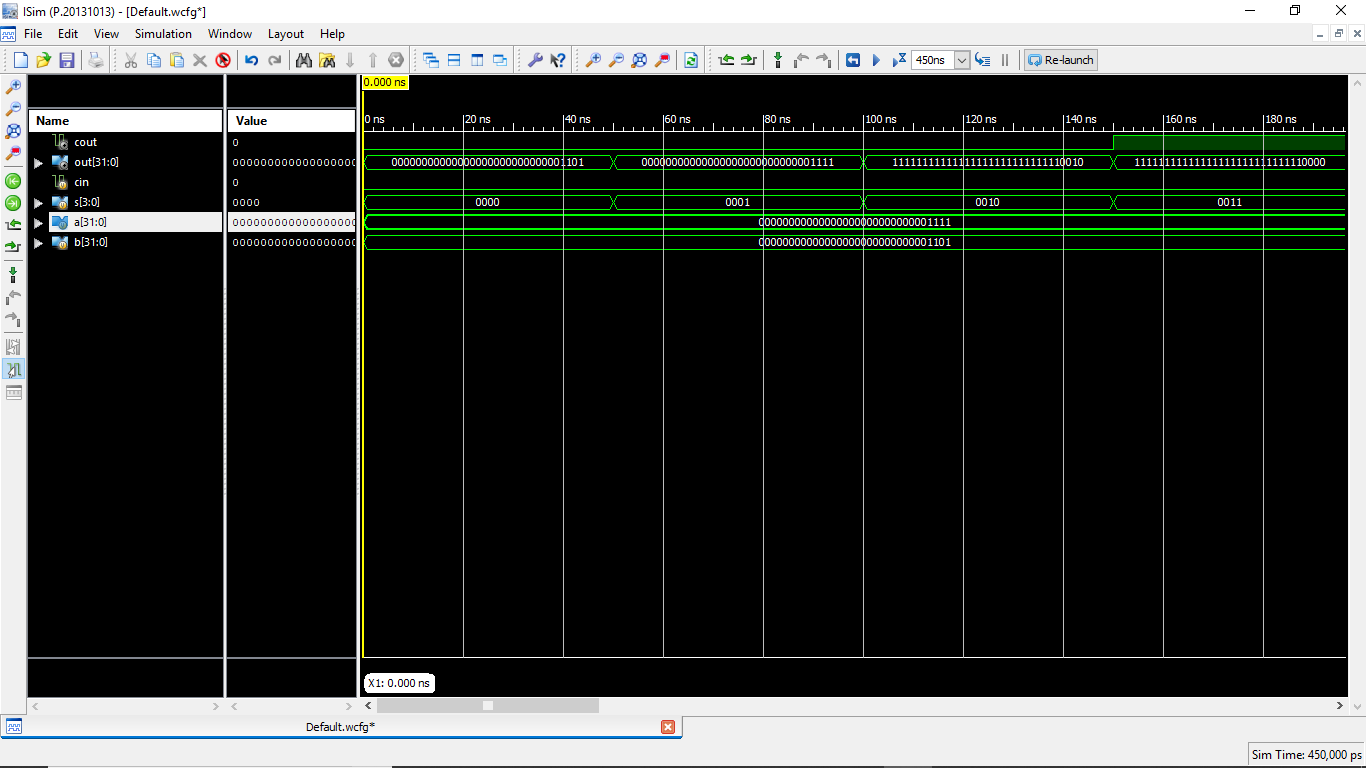
* Logical Unit

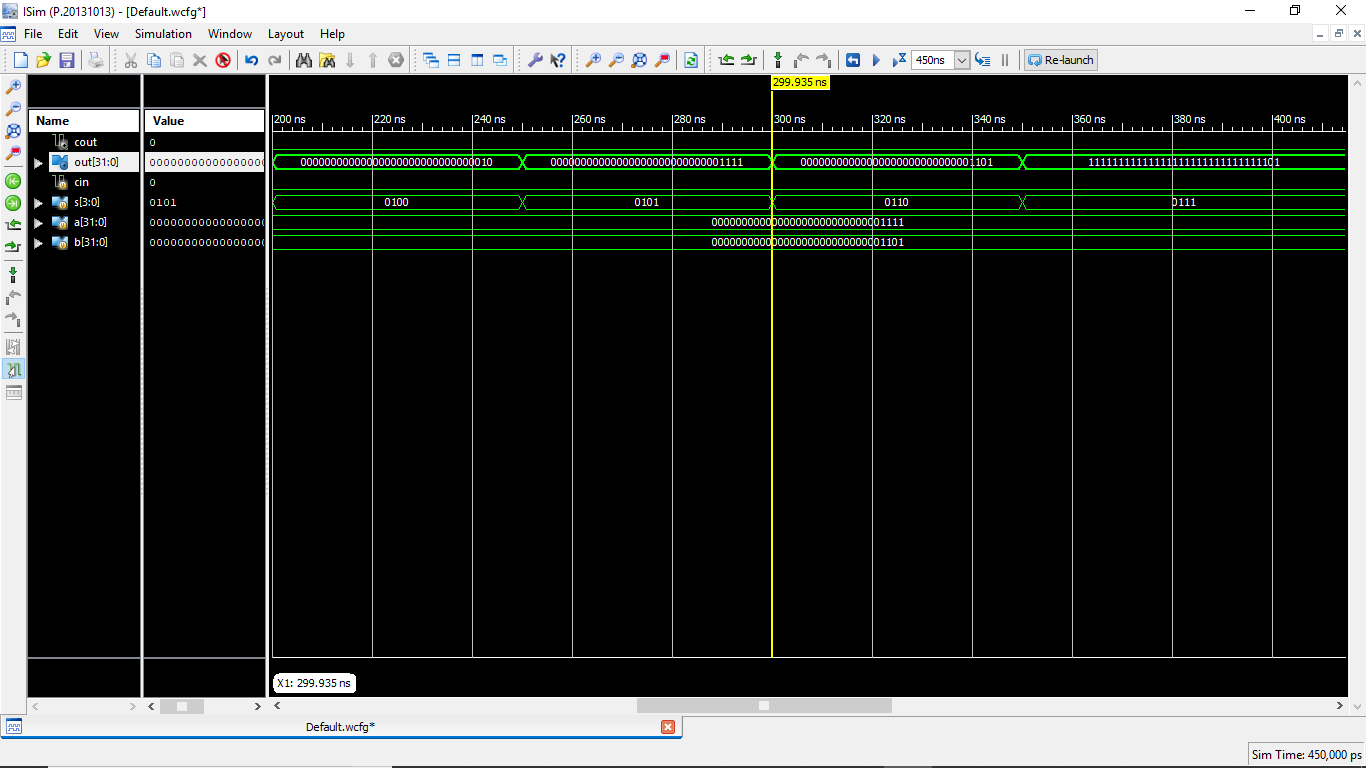
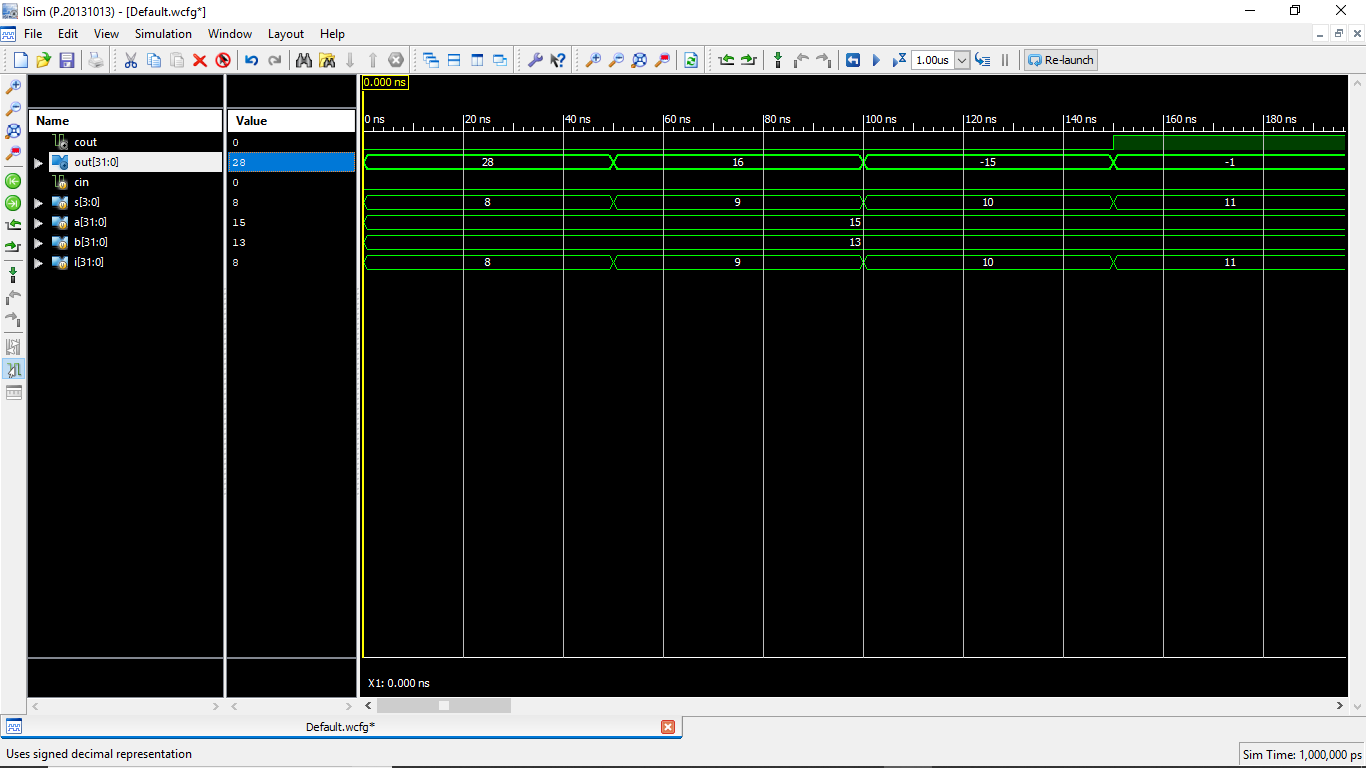


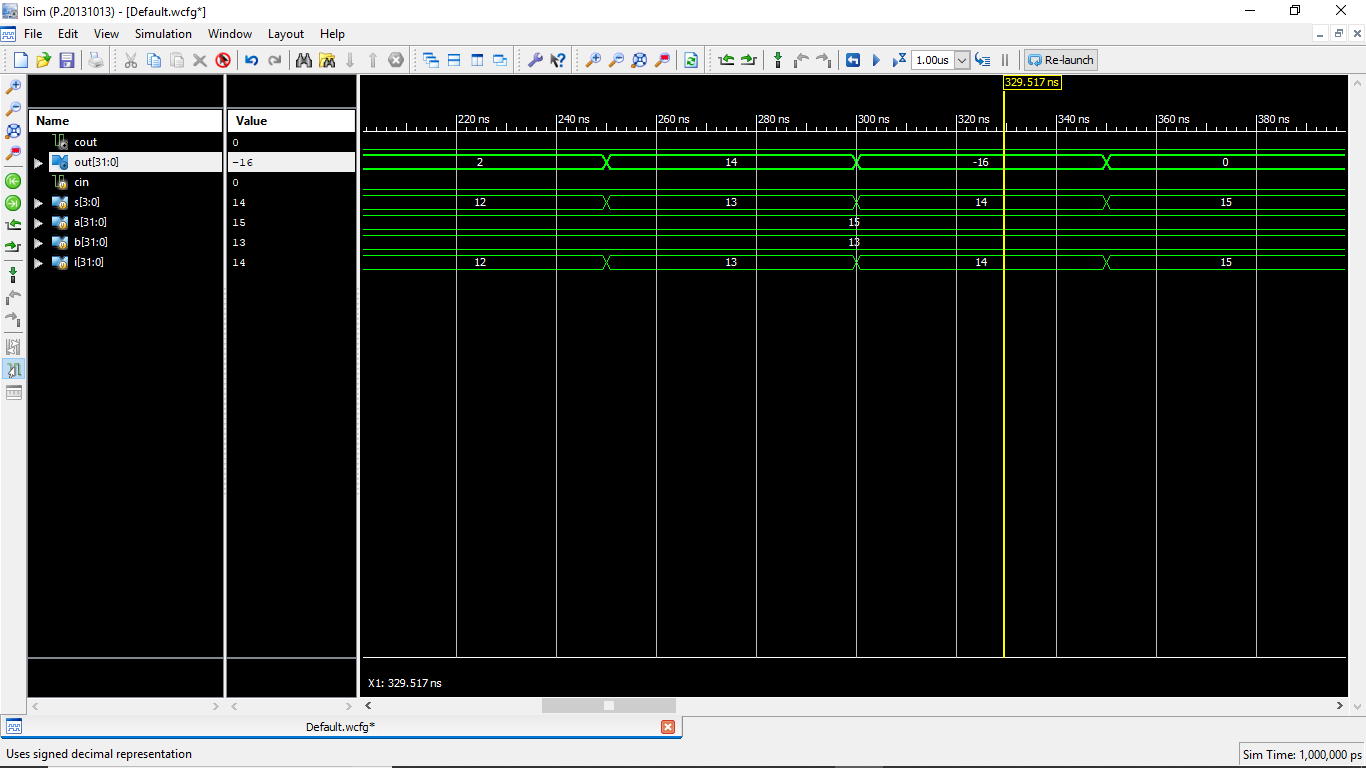
* Complete RTL Schematic

# Waveforms

* The following is a wave form for AND, OR, NAND and NOR operations



* The following is a waveform for XOR, Buffer A, Buffer B and XNOR operations.
* The following are the waveforms of Addition, Increment, 2’s Complement and SET operations.

* The following are the waveforms of Subtraction, Deccrement, NOT and CLEAR operations.

# Applications

The reversible 32-bit ALU may have applications in computer security and transaction processing, but the main long-term benefit will be felt very well in those areas which require high energy efficiency, speed and performance. It includes the area like

1. Low power CMOS.
2. Quantum computer.
3. Nanotechnology
4. Optical computing
5. Design of low power arithmetic and data path for digital signal processing (DSP).
6. Field Programmable Gate Arrays (FPGAs) in CMOS technology for extremely low power, high testability and self-repair.
7. Microprocessors and Microcontrollers.

# Refrences

* C. H. Bennett, “Logical reversibility of computation”, IBM J. Research and Development, 17, pp. 525 – 532, November1973.
* Feynman, R., “Quantum mechanical computers” Optics, News, and pp.: 11-20 11, 1985.
* Fredkin, E. and T. Toffolli, “Conservative logic”, Intl. J. Theoretical Phy., 21, pp.: 219-253, 1982.
* Lihui Ni, Zhijin Guan, and Wenying Zhu, “A General Method of Constructing the Reversible Full-Adder”, Third International Symposium on Intelligent Information Technology and Security Informatics, pp.109-113, 2010 5
* R. Landauer, “Irreversibility and heat generation in the computing process”, IBM J. Research and Development, 5(3), pp.: 183-191, 1961.
* Toffoli T, “Reversible computing”, Technical Memo MIT/LCS/TM-151, MIT Laboratory for Computer Science (February), 1980.