Indian Journal of Science and Technology, *Vol 9(2), DOI: 10.17485/ijst/2016/v9i2/86343, January 2016*

*ISSN (Print) : 0974-6846*
*ISSN (Online) : 0974-5645*

# Design and Implementation of Reconfigurable ALU for Signal Processing Applications

**J. Thameema Begum[1]*, S. Harshavardhan Naidu[2], N. Vaishnavi[1], G. Sakana[1] and N. Prabhakaran[1]**

[1]Electronics and Communication Engineering, Vel Tech High Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Avadi, Chennai – 600062, Tamil Nadu, India; thameema84@gmail.com, vaishnaviec07@gmail.com, sasikdnl@gmail.com, captainprabhakar1982@gmail.com
[2]Electronics and Electrical Engineering, Vel Tech High Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Avadi, Chennai – 600062, Tamil Nadu, India; sapineniharsha@gmail.com

## Abstract

**Background/Objectives:** The main objective of the paper is to implement a reconfigurable ALU that is a combination of a 32-bit floating point adder/subtractor and integer ALU. The integer ALU performs integer functions and logical operations such as addition, subtraction, shifting and comparison. **Methods/Statistical analysis:** In this paper, a 32-bit single precision format based on IEEE754 standard for the floating-point unit, with a 23-bit mantissa, 8-bit exponent and 1-bit sign value is considered. **Findings:** Verilog Hardware Description Language (HDL) is used and simulated by model sim simulator and then synthesized with Spartan3E FPGA. The functional unit uses 25% number of slices, 9% number of slice flip-flops, 18% of 4 input LUTs. From the timing report, the maximum frequency obtained is 81.614MHz. The maximum power obtained by the system is 82.46mW. **Applications/Improvements:** This can be used for data-parallel and computation intensive applications and in multimedia applications.

**Keywords:** Field Programmable Gate Arrays (FPGA), Hardware Description Language (HDL), Reconfigurable Arithmetic Logic Unit

## 1. Introduction

With the development in the field of Programmable Logic Devices (PLD's), implementation of reconfigurable systems have become possible to a large extent. One such PLD is the Field Programmable Gate Array (FPGA), which is a hardware device with programmable logic, routing, memory, and I/O and its configuration is generally specified using a hardware description language[1]. Reconfigurable computing makes use of FPGA's for implementation of complex algorithms and when applied to an Arithmetic Logic Unit (ALU), makes it possible for the user to select the type of input and functionality to be performed. This paper describes the design and methodologies of the various functions performed by a reconfigurable ALU.

The IEEE (Institute of Electrical and Electronics Engineers) has given a standard to define floating-point representation. Even though there are other representations, it is the most commonly used for floating point numbers. IEEE 754 standard specifies formats and methods for floating-point arithmetic in computer systems - standard and extended functions with single, double, extended, and extendable precision - and recommends formats for data interchange[2]. Exceptions and the standard handling of these conditions have also been specified[3]. The IEEE 754 standard describes three formats for representing floating point numbers. These

**Author for correspondence*

include the 32 bit single precision, 64 bit double precision and 128 bit extended double precision[3].

A 32-bit single precision format for our floating-point addition and subtraction is considered in this paper. The single precision format has 1-sign bit, 8 exponent bits and 23 mantissa bits. The bias value is 127. The exponent, $e$ = E+127, has an exponent range of $E_{min}$ = -126 to $E_{max}$ = +127[3, 4].

## 2. Design Hierarchy

The reconfigurable ALU implemented here performs floating point and integer addition and subtraction, integer multiplication, shifting and comparison and the required function can be selected using a multiplexer[5,6]. The basic top-level block diagram is shown in Figure 1.

The reconfigurable ALU consists of the following modules: The floating point adder and subtractor which performs the addition and subtraction of the floating point numbers, the integer adder and subtractor which performs the addition and subtraction of the integer numbers, the shifter used to shift the input in either left or right direction, the comparator for logical comparison of the two inputs. All the inputs are 32 bits.

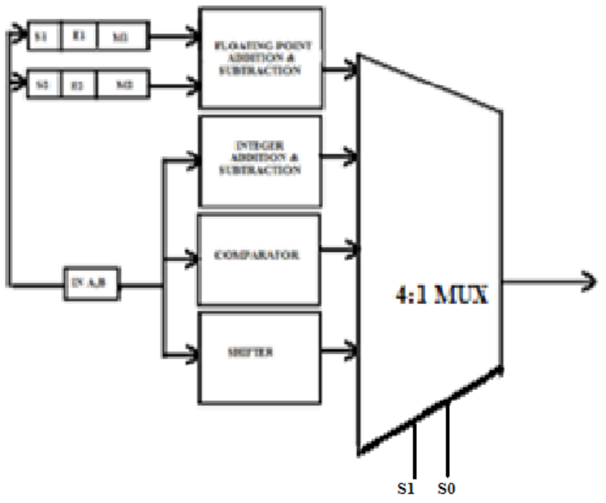
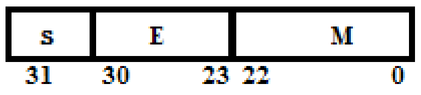

S1
E1
M1
S2
E2
M2
FLOATING POINT ADDITION & SUBTRACTION
INTEGER ADDITION & SUBTRACTION
COMPARATOR
SHIFTER
IN A,B
4:1 MUX
S1
S0

**Figure 1.** Top-level block diagram.

### 2.1 Floating Point Number Addition and Subtraction

According to the IEEE 754 single precision format, the 32-bit input can be divided into three parts namely sign, exponent and mantissa, which is presented in Figure 2.



S = SIGN (1-bit), E = EXPONENT (8-bits), M = MANTISSA (23-bits)

**Figure 2.** Three components of 32-bit input.

Two 32-bit inputs each of them divided into the sign, exponent and mantissa forms are considered (Figure 2). Comparison of the two exponent values and assigning of the larger and smaller exponents and fractions follow this. The smaller fraction is then right shifted by the difference value (of the two exponents). A control bit is used to decide whether addition or subtraction operation is to be performed. For addition, XOR the two sign bits and if the output bit is zero then perform addition else perform subtraction. For subtraction, take the complement of one sign and XOR the two sign bits. If the output bit is zero then perform addition else perform subtraction.

#### 2.1.1 Exponent Block

In the Figure 3, multiplexer-1(MUX1) is used to select the larger exponent. The select lines to the MUX1 are given from the control unit and the output from the MUX1, which is the larger exponent, goes into the normalization unit.

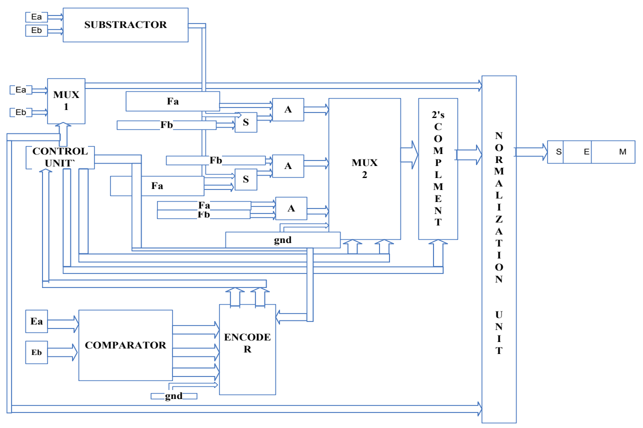


**Figure 3.** Floating point addition/ subtraction unit.

#### 2.1.2 Sign Block

The output sign is the sign bit of the input with the larger exponent.

#### 2.1.3 Mantissa Block

Comparison of the two-mantissa values is done using the comparator and there are three possible outputs: First mantissa is greater than second mantissa. Second mantissa is greater than first mantissa. Both the mantissa values are equal. These outputs are then given into an encoder and the encoder output is used to select the output from the multiplexer-2 (MUX2). A shifter is then used to shift the smaller mantissa by the difference in exponents, given by the exponent subtractor and the resultant mantissa is given into the normalization unit.

### *2.1.4 Normalization Block*

The output exponent and mantissa of the addition/subtraction process is given into the normalization unit where the normalization of the mantissa is done and the corresponding change in exponent occurs and the final exponent and mantissa is obtained.

## 2.2 Exception Handling

Exception handling has also been considered. The three basic exceptions checked here are: 1. Output zero, 2. Output infinity and 3. Not a Number (NaN). A three-bit register is used to indicate the occurrence of exceptions. The final output is given out as (sout, eout, fout). The flowchart for floating point adder/subtractor is given in Figure 4. By looking at the flowchart, the functionality of floating point addition and subtraction as explained for the block diagram is understood.

## 2.3 Integer Addition/ Subtraction

This module in Figure 5 performs the addition or subtraction of two integer inputs. It is implemented using four 8-bit full adders each of which is implemented using eight 1-bit full adders. Here each bit of the second input is XORed separately with the input carry bit. When input carry bit, cin, is low the module performs addition of the two inputs and when input cin is high, subtraction is performed. During addition the second input is fed as such to the adder whereas during subtraction the 2's complement form of the second input is fed to the adder.

## 2.4 Shifter

A barrel shifter shown in Figure 6 is used for shift application. Both left and right shifts are possible. The amount of shift can also be controlled. There are three inputs to the shifter module. A 32-bit input is provided to the shifter module. The direction of the shift is specified by the input d, d = 0 specifies left shift and d = 1 specifies

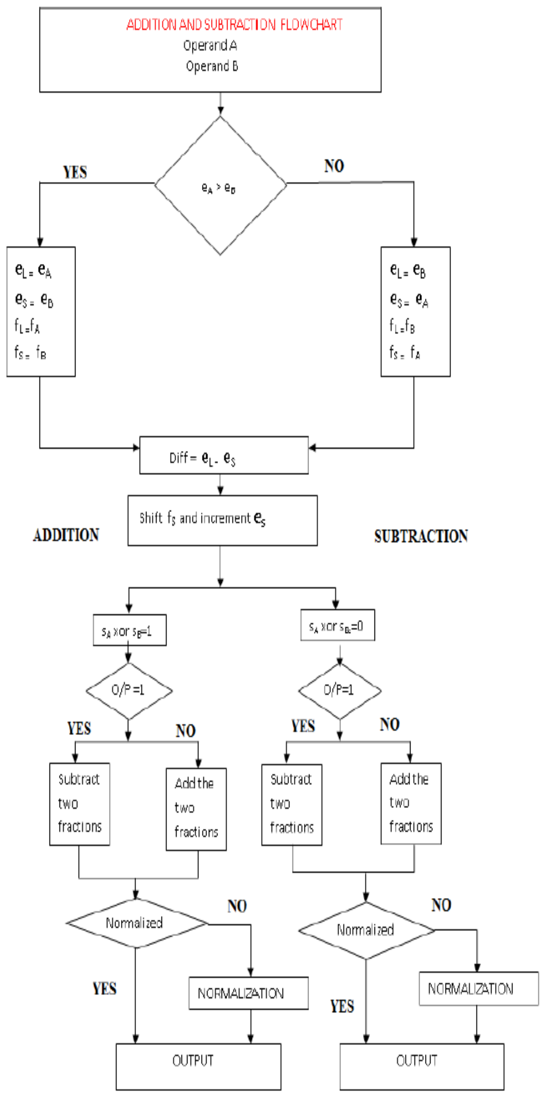


**Figure 4.** Flowchart of floating point adder/subtractor.

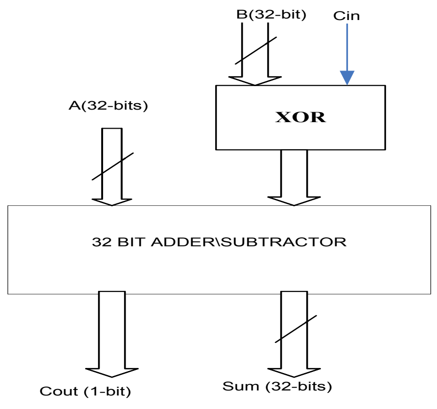


**Figure 5.** Integer adder/subtractor.

right shift. The amount of shift is determined by the input sc. Shifts of 1, 2, 4, 8 and 16 bits are possible; sc is a 5 bit register. When sc = 00001, it is a 1 bit shift. When sc = 00010, it is a 2 bit shift, sc = 00100 indicates a 4 bit shift and so on. Thus d = 1 and sc = 00010 indicates a shift of 2 bits to the right.

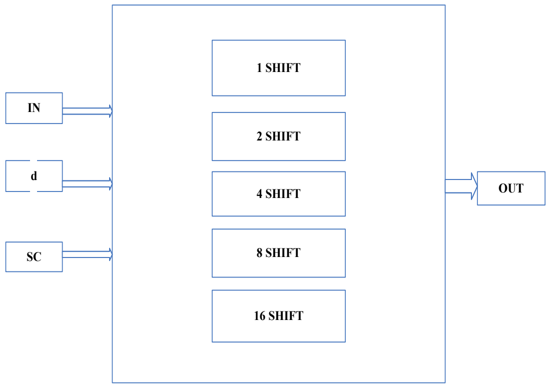


**Figure 6.** Barrel shifter.

## 2.5 Logical Comparator

A 32-bit comparator has been shown in Figure 7. There are 3 outputs to the module: greater, equal and lesser. The two inputs each are of size 32-bits and are compared bitwise. The comparator is implemented using AND, OR and NOR gates. The outputs of the comparator becomes one when A is greater than, equal to or lesser than B. Thus the output 'greater' becomes 1 when A is greater than B. The output 'equal' becomes 1 when A = B and 'lesser' becomes 1 when A<B.

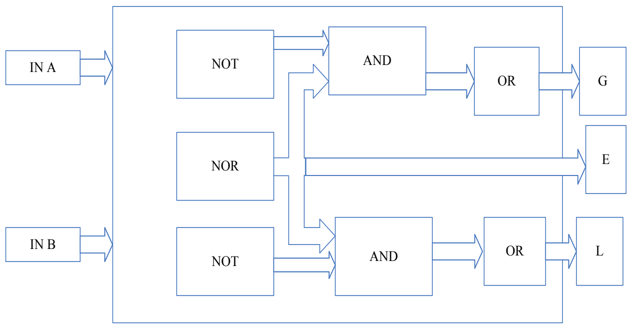


**Figure 7.** Logical comparator.

## 3. Implementation

Consider the two inputs 00111110100000000000000000000000 and 11000010110010000000000000000000. The implementation in each module is as follows.

In the floating-point unit the two inputs are split into the sign, exponent and mantissa. Thus sa = 0, ea = 01111101 and fa = 00000000000000000000000. Similarly sb = 1, eb = 10000101, fb = 10010000000000000000000. The exponent bits of the two inputs are then compared. Since ea<eb, el = eb = 10000101 and es = ea = 01111101 where el and es represents larger and smaller exponents respectively. Similarly fl = fb = 10010000000000000000000 and fs = 00000000000000000000000. Now the difference between the two exponents is calculated to be 00001000. fs is shifted by the difference in the exponents and is obtained as 00000001000000000000000 and es is incremented by 8 bits iees = 10000101 = el. For addition, the two sign bits are XORed to obtain the result as 1. Now the two mantissas are subtracted to obtain 10001111000000000000000. The output is then normalized to obtain the result 1100001 01100111100000000000000000. For subtraction the same procedure is followed except that the complemented form of sb is used.

In the integer adder/subtractor module addition and subtraction is performed on the basis of cin input considered. If the cin input is 0, then XOR operation of A, B, cin is done and sum = 100000001 01001000000000000000000 is obtained and when the cin input is 1, subtraction is performed and sum = 11001001011000000000000000000000 is obtained.

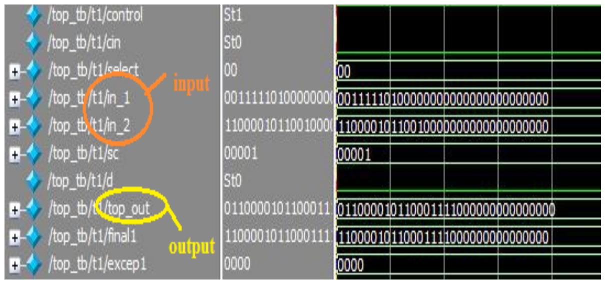
Let the shifter module be fed with the first input. Let the input be shifted to the left with a shift amount of 2 bits. The result of the shifter becomes 1111101000000000000000 000000000000.

The comparator compares the two inputs and gives three results as follows; greater, equal and lesser. Here since input B is greater than input A, the result of comparator becomes greater = 0, equal = 0 and lesser = 1.

## 4. Results and Discussion

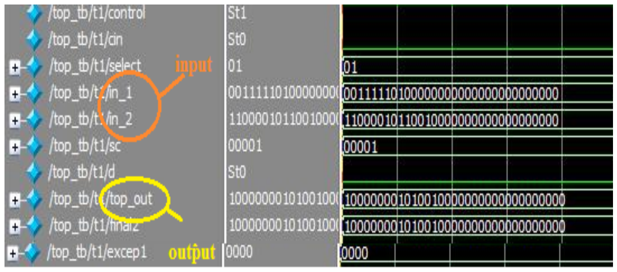
The proposed paper has been simulated in Model Sim and synthesized in Xilinx. According to the top-level diagram in Figure 1, when the select line of the mux is 00, the floating-point unit is selected. The floating-point output is shown in Figure 8. The final output is a 32-bit value.

The Figure 9 shows the integer adder/subtractor module output. The output is obtained when the select line of the mux is 01. The shifter output is shown in Figure 10. The sc value is 00010, and d is 0, which implies a left shift is done by two positions. Input, in_1 is taken as input for the shift. The Figure 11 shows the comparator output. Here the two inputs have been compared and the three LSB bits of the output indicates whether the result is greater, equal or lesser respectively.



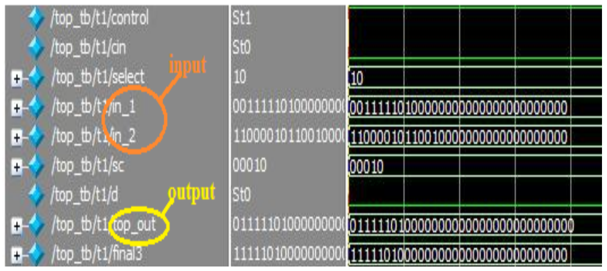
```
/top_tb/t1/control   St1
/top_tb/t1/cin       St0
/top_tb/t1/select    00                   00
/top_tb/t1/in_1      001111101000000000   00111110100000000000000000000000
/top_tb/t1/in_2      110000101100100000   11000010110010000000000000000000
/top_tb/t1/sc        00001                00001
/top_tb/t1/d         St0
/top_tb/t1/top_out   011000010110001111   011000010110001111000000000000000
/top_tb/t1/final1    110000101100011111   11000010110001111000000000000000
/top_tb/t1/excep1    0000                 0000
```

Figure 8. Floating-point output.



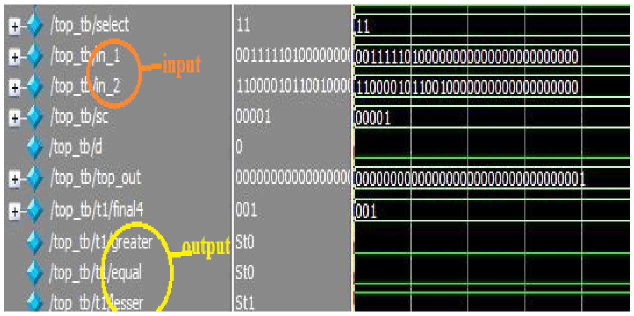
```
/top_tb/t1/control   St1
/top_tb/t1/cin       St0
/top_tb/t1/select    01                   01
/top_tb/t1/in_1      001111101000000000   00111110100000000000000000000000
/top_tb/t1/in_2      110000101100100000   11000010110010000000000000000000
/top_tb/t1/sc        00001                00001
/top_tb/t1/d         St0
/top_tb/t1/top_out   100000001010010000   100000001010010000000000000000000
/top_tb/t1/final2    100000001010010000   100000001010010000000000000000000
/top_tb/t1/excep1    0000                 0000
```

Figure 9. Integer adder and subtractor output.



```
/top_tb/t1/control   St1
/top_tb/t1/cin       St0
/top_tb/t1/select    10                   10
/top_tb/t1/in_1      001111101000000000   00111110100000000000000000000000
/top_tb/t1/in_2      110000101100100000   11000010110010000000000000000000
/top_tb/t1/sc        00010                00010
/top_tb/t1/d         St0
/top_tb/t1/top_out   011111010000000000   011111010000000000000000000000000
/top_tb/t1/final3    111110100000000000   11111010000000000000000000000000
```

Figure 10. Shifter output.



```
/top_tb/select       11                   11
/top_tb/in_1         001111101000000000   00111110100000000000000000000000
/top_tb/in_2         110000101100100000   11000010110010000000000000000000
/top_tb/sc           00001                00001
/top_tb/d            0
/top_tb/top_out      000000000000000000   000000000000000000000000000000001
/top_tb/t1/final4    001                  001
/top_tb/t1/greater   St0
/top_tb/t1/equal     St0
/top_tb/t1/lesser    St1
```

Figure 11. Comparator output.

This RTL schematic is similar to the top-level block diagram as shown in Figure 1. The top-level schematic of the modules are shown in Figure 12. The floating point adder/subtractor schematic has been generated and shown in Figure 13. It is seen that the output from the floating-point adder subtractor block is the first input into the final selection mux. The output is given out when the select line becomes 00.

Similar to the Figure 13, Figure 14 shows the integer adder subtractor block and the output from this block is the second input into the final selection mux. The output is shown when the select line becomes 01. The Figure 15 shows the barrel shifter. The output of this block is given as the third input of the mux, which is given as output from mux when select line is 10. The Figure 16 shows the RTL schematic of comparator block. The output is selected when the select line is 11.

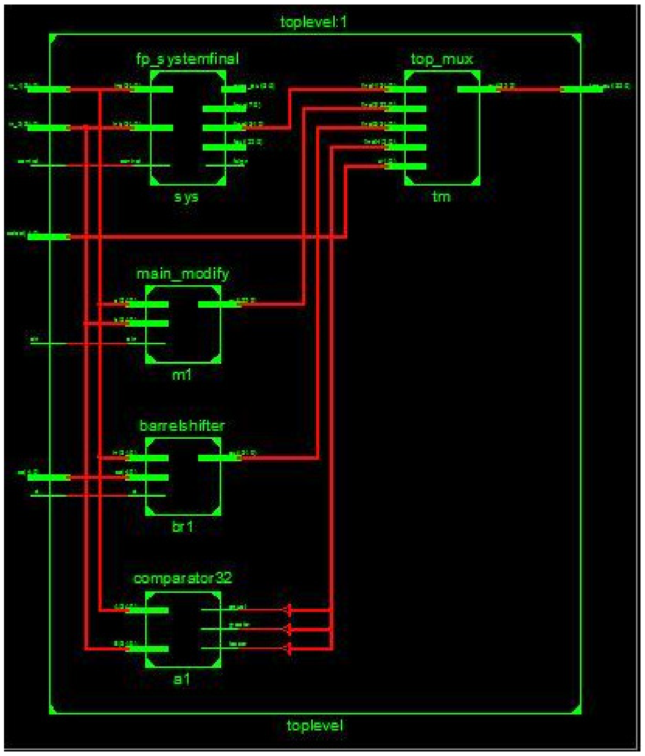


Figure 12. Top-level RTL schematic.

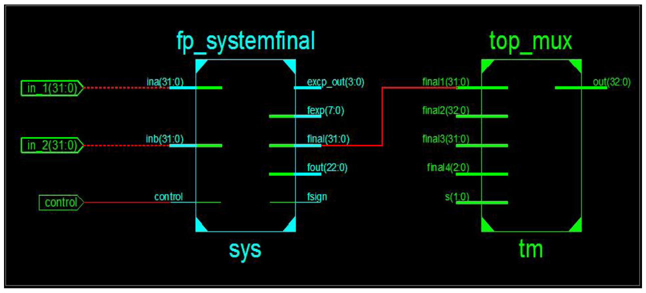


Figure 13. Floating point adder/subtractor RTL schematic.

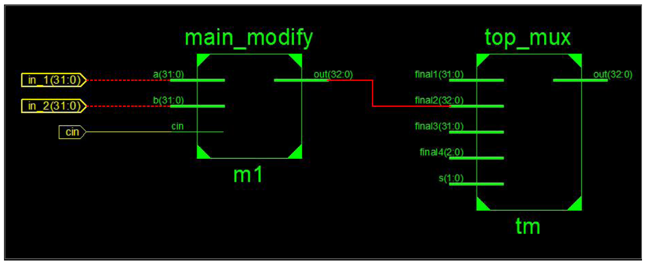


**Figure 14.** Integer adder/subtractor RTL schematic.

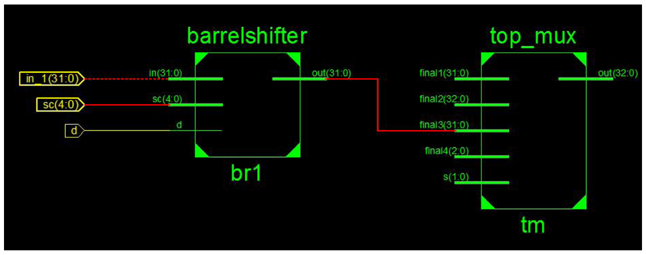


**Figure 15.** Barrel shifter RTL schematic.

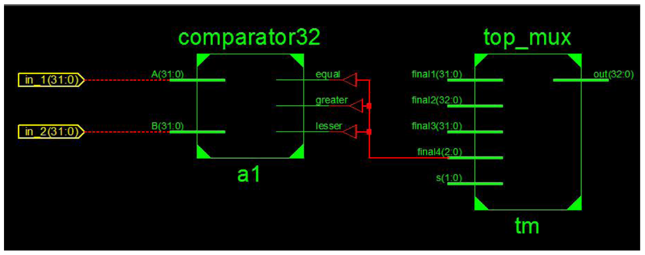
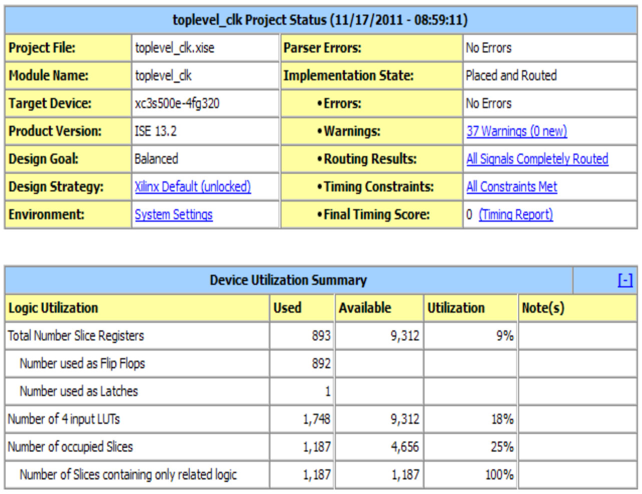


**Figure 16.** Comparator RTL schematic.



| toplevel_clk Project Status (11/17/2011 - 08:59:11) | | | |
|---|---|---|---|
| **Project File:** | toplevel_clk.xise | **Parser Errors:** | No Errors |
| **Module Name:** | toplevel_clk | **Implementation State:** | Placed and Routed |
| **Target Device:** | xc3s500e-4fg320 | **• Errors:** | No Errors |
| **Product Version:** | ISE 13.2 | **• Warnings:** | 37 Warnings (0 new) |
| **Design Goal:** | Balanced | **• Routing Results:** | All Signals Completely Routed |
| **Design Strategy:** | Xilinx Default (unlocked) | **• Timing Constraints:** | All Constraints Met |
| **Environment:** | System Settings | **• Final Timing Score:** | 0 (Timing Report) |

Device Utilization Summary [-]

| Logic Utilization | Used | Available | Utilization | Note(s) |
|---|---|---|---|---|
| Total Number Slice Registers | 893 | 9,312 | 9% | |
| Number used as Flip Flops | 892 | | | |
| Number used as Latches | 1 | | | |
| Number of 4 input LUTs | 1,748 | 9,312 | 18% | |
| Number of occupied Slices | 1,187 | 4,656 | 25% | |
| Number of Slices containing only related logic | 1,187 | 1,187 | 100% | |

**Figure 17.** Synthesis Report.

The Figure 17 shows the Synthesis report and the output shows the functional unit which uses 25% number of slices, 9% number of slice flip-flops, 18% of 4 input LUTs.

## 5. Conclusion

This system was implemented using the simulation tool and the functional unit uses 25% number of slices, 9% number of slice flip-flops, 18% of 4 input LUTs. From the timing report, the maximum frequency obtained as 81.614MHz. The power of the system was obtained as 82.46mW.As future enhancement the system can be implemented using a decoder instead of a mux. This will reduce the power consumption. Also multiplier and divider logic can be included in the floating point as well as integer units. More logical operations can also be implemented. This can be used for both floating point and integer operations. The system has been designed for 32 bit inputs and therefore must be generalized for other input bit sizes.

## 6. References

1. Brown SD. Field Programmable Gate Arrays. Kluwer Academic Publishers; 1992; 180.
2. IEEE. IEEE standard Floating-Point Arithmetic. IEEE Std 574-2008. 2008. p. 1–58.
3. Ismail RC, Coleman JN, Norzahiyah N, Sauli Z. A Comparative analysis between logarithmic number system and floating-point ALU. Advances in Environmental Biology. 2013; 7(12):3601–6.
4. Hollasch S. IEEE Standard 754 Floating Point Numbers. 2005. p. 1–8.
5. Suhaili S, Sidek O. Design and implementation of reconfigurable ALU on FPGA. 3rd International Conference on Electrical and Computer Engineering; Dhaka: Bangladesh. University of Malaysia; 2004. p. 28–30.
6. Khaladkar RB, Anita Angeline A, Kanchana Bhaaskaran VS. Dynamic logic ALU design with reduced switching power. Indian Journal of Science and Technology. 2015 Aug; 8(20):1–8.