

# DESIGN AND IMPLEMENTATION OF RECONFIGURABLE ALU ON FPGA.

Shamsiah Suhaili and Othman Sidek

School of Electrical & Electronic Engineering, Engineering Campus, Universiti Sains Malaysia,  
14300 Nibong tebal, Seberang Perai Selatan, Pulau Pinang  
E-mail: eeshamsiah@eng.usm.my, dean\_ee@eng.usm.my

## ABSTRACT

Reconfigurable Computing has grown to become an important and large field of research. This paper describes the implementation of a reconfigurable ALU that combines 32-bit single precision floating-point adder and integer ALU (arithmetic logic unit) into a single unit on FPGA. Reconfigurable ALU can perform both integer operations and floating-point additions. Simply extending the operand width and adding a few switches enable the unit to be reconfigured.

## 1. INTRODUCTION

With rapid advances in FPGA technologies, architectures based on configurable computing engines, in which the Arithmetic Logic Unit (ALU) can be modified on-the-fly during computation, are becoming popular. FPGAs are logic devices that offer in-circuit hardware re-programmability. Since their programming can be changed quickly without any rewiring or refabrication, they can be used in more flexible manner than standard gate arrays [1]. Reconfigurable Computing (RC) is an emerging technology that utilizes FPGAs to implement computation intensive algorithms at hardware level [2].

With the technological progress of programmable devices and programming methodologies, reconfigurable devices become more and more interesting and important. In this paper, such technology is applied for designing reconfigurable ALU. ALU is the heart of microprocessors as it executes binary instructions of a program.

Floating-point arithmetic is very useful in many applications. However, non-numerical applications usually have very few floating-point computations, and as a result, the floating-point resources are mostly in idle mode. During this idle mode, the floating-point still consume power and the die area is

wasted [3]. Therefore, reconfigurable ALU can be used in microprocessor to reduce power consumption and avoid waste die area.

## 2. RECONFIGURABLE ALU

This section presents the architecture of ALU, which perform both integer and floating-point addition operations. Lavinier, Solihin and Cameron [4, 5] proposed the idea of an ALU that can serve both integer and floating-point instructions. However, in this paper, reconfigurable ALU was designed as 32-bit integer and 32-bit single precision floating-point adder using Verilog HDL based on FPGA.

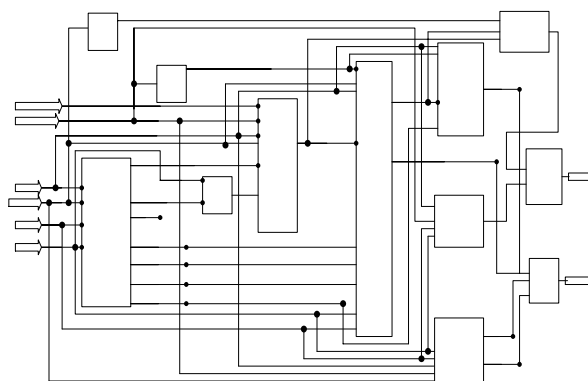


Fig. 1 Reconfigurable ALU

### 2.1 Floating-point number

The IEEE representation divides the number of bits into three groups, the sign bit, the exponent and the mantissa part. A floating point number can be represented as

s	e	m
31	30	23 22
		0

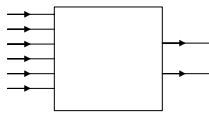
- S: Sign bit
- M: Mantissa, coded with 23 bits
- E: Exponent, coded with 8 bits

The sign bit field,  $s$  is bit 31 and is used to specify the sign of the number. Bit 30 to 23 are the exponent field. This 8-bit quantity is a signed number represented by using a bias of 127, which gives an exponent range of -126 to 127. Bits 22 down to 0 are used to store the binary representation of the floating-point number. The mantissa is always normalized, and its most significant bit is always 1. The leading one in the mantissa, 1.m, does not appear in the representation; therefore, the leading one is implicit.

IEEE 754 standard deals with floating point representation and proposes the following data format, for a single-precision number, the equation is given by,

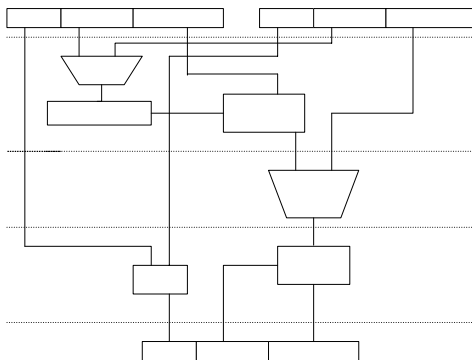
$$(-1)^s \times 1.m \times 2^{(e-127)} \quad (1)$$

## 2.2 Architecture



**Fig. 2** Reconfigurable ALU top level

Before describing the architecture of reconfigurable ALU, the architecture of integer arithmetic logic unit and 32-bit floating-point adder will be described. In arithmetic logic unit mode, reconfigurable ALU performs 32-bit integer operation, addition, subtraction, shift left and shift right and logic operations. In floating point adder mode, reconfigurable ALU performs single precision 32-bit floating-point adder.



**Fig. 3** Floating-point adder algorithm

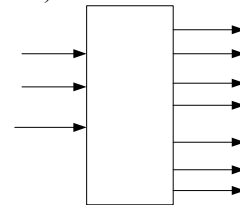
The architecture of reconfigurable ALU follows the architecture of floating point adder. However, the hardware has been modified to allow integer operations. It was modelled in Verilog HDL

language. The modifications over the architecture of floating-point adder are:

- Extension of adder from 27 bits to 32 bits.
- Substitution of the 27 bits shifter by a 32 bits barrel shifter.
- Insertion of some multiplexer inside the module.

### 2.2.1 EXPCOMP module

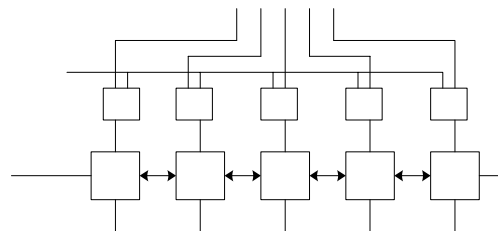
EXPCOMP module is the first module, which decomposes floating-point number with largest exponent into sign, exponent ( $e_2$ ) and mantissa ( $um_2$ ). Besides, the EXPCOMP module calculates the absolute value of difference between exponents (shift). This module also determines expected sign of output ( $sm$ ) and determines whether overall calculation is effectively addition or subtraction operation (addsub).



**Fig. 4** EXPCOMP module

### 2.2.2 SHIFTER module

SHIFTER module is a shifter, which shifts mantissa ( $um_2$ ) or  $in\_b$ . It depends to the reconfigure selector. If the shifter is in integer ALU mode, the input of the shifter is  $in\_b$ , but if shifter is in floating-point mode, the input of the shifter is  $um_2$ . The shift amount of ALU depends from user input, ( $s[4:0]$ ) and (Dir) whether shift to left or right. However, for floating-point adder, the shifter shifts mantissa ( $um_2$ ) of operand with smaller exponent right by shift bits ( $alu\_a[4:0]$ ) to align its decimal point. The output value of shifter module is  $barsht\_out$ .



**Fig. 5** SHIFTER module

### 2.2.3 ADDER module

When configured as a floating-point adder, the inputs of the adder come from first stage of the pipeline. In order to get the output of mantissa

(fp\_adder), the ADDER module depends on (addsub), whether to add or subtract the value of mantissa (m1, fp\_barshft). When configured as an integer ALU, the two inputs of the adder are respectively connected to in\_a and in\_b by multiplexer to get the integer output (alu\_adder). The output of the adder module is adder\_out. The ADDER module also determines the sign of the result (s\_add).

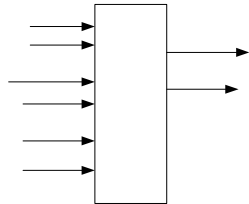


Fig. 6 ADDER module

**2.2.4 ROUND module**

The IEEE standard provides four precisely defined rounding modes. The ROUND module only rounds to zero mode. It shifts the mantissa (fp\_adder) to normalize it. The exponent of the floating-point adder will increase or decrease depending on the value of shift amount. This module also produces the final mantissa and exponent of the result (fp\_adder1).

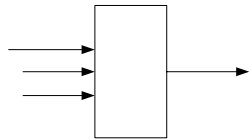


Fig. 7 ROUND module

**2.2.5 INVALIDCASE module**

INVALIDCASE module handles special cases of the input either being infinity, zero, or NaN (Not a Number). The output result will be sent to MUX31 module when en is logic high.

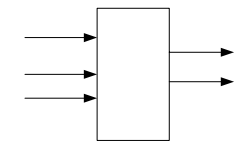


Fig. 8 INVALIDCASE module

**2.2.6 MUX31 module**

This module configures whether the output of ROUND module or INVALIDCASE module will be the final result of floating-point adder.

**2.2.7 INT\_OUT module**

INT\_OUT module is a module that generates the integer ALU operation. It consists of ALU\_ARITH module and ALU\_LOGIC module. ALU\_ARITH

module contains the shifter and adder output from integer ALU. Floating-point adder also uses the same shifter and adder module.

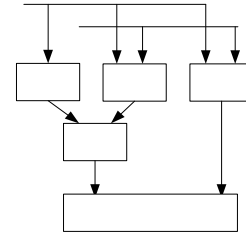


Fig. 9 Integer Arithmetic Logic unit

There are 3-pipeline stages of floating-point adder; adder, shifter, adder, and the final result. Figure 10 shows the flow chart of reconfigurable ALU.

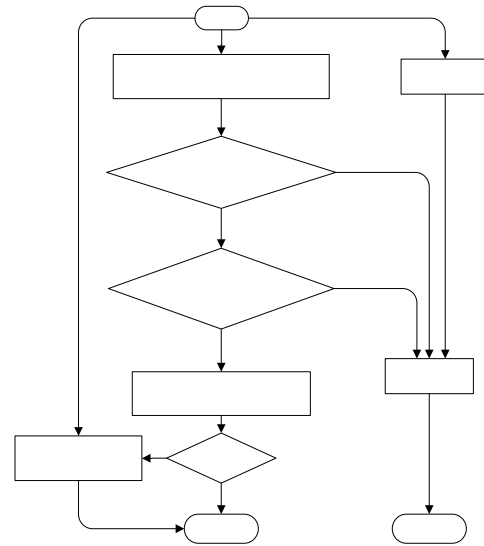


Fig. 10 Reconfigurable ALU flow chart.

**3. RESULTS**

Figure 11 shows the simulation of reconfigurable ALU. In the simulation, there are two types of reconfiguration. The first type switches from integer mode to floating-point mode. The second type switches from floating-point mode to integer mode. Due to the 3-stage pipeline when configured as a floating-point unit and 1-stage pipeline when configured as an integer ALU, the time for switching from integer to floating-point and vice versa is not constant.

**3.1 Switching from integer to floating-point mode**

In the integer mode, the reconfigure input is '0'. When the reconfigure input change to '1', the floating-point mode will display the result. If the integer performs add/sub or logic unit operation

in\_a  
in\_b  
reconfigure  
add sub  
adder  
m1  
fp\_barshft

before switching to the floating-point mode, there are no extra cycles for reconfiguring the reconfigure ALU. However, if the integer mode performs shift operation before switching to the floating-point mode, one cycle is lost for reconfiguration purpose. Since shift operation uses the barrel shifter, which is also used in the first step of the floating-point adder.

### 3.2 Switching from floating-point to integer mode

In the floating-point mode, when it switches to the integer mode, which is in shift operation, one reconfiguration cycle is needed. Since a shift operation uses the same barrel shifter, which is also used in the first stage of the floating-point adder. If the add/sub operation done after the floating-point mode, two cycle are needed for switching from the floating-point to the integer mode. However, there is no extra cycle needed for reconfiguring the reconfigurable ALU if switches to logic unit operation.

## 4. CONCLUSION

The module was implemented in the Verilog Hardware description language (HDL), tested in simulation using Model Tech's ModelSIM XE II v5.6a, placed and routed on a Spartan IIE XC2s300e FPGA from Xilinx. When synthesized, this reconfigurable functional unit used 31% number of slices, 5% number of slice flip-flops, and 28% number of 4 input LUTs. For timing report, this module used 20.457 MHz (48.840 ns).

The re-programmability is provided by a few multiplexer. Due to the pipeline of the floating-point adder, the switching from floating-point to integer needs two cycles, while switching from integer to floating-point does not require extra cycles. This penalty is due to the need to wait for the floating-point pipeline before using it for integer operation.

## REFERENCES

- [1] S. Hauck, "The Roles of FPGAs in reprogrammable System", Proceedings of the IEEE, Vol. 86, No. 4, pp 615-638, April 1998.
- [2] C. Gloster, Jr., Ph.D., P.E., E. Dickens, "A Reconfigurable Instruction Set", MAPLD International Conference 2002.
- [3] Y. Solihin, W. Cameron, L. Yong, D. Lavenier, M. Gokhale, "Mutable Functional Unit and Their Application on Microprocessors", ICCD 2001, International Conference on Computer Design, Austin, Texas, sep 23-26, 2001.
- [4] D.Lavenier, Y.Solihin, W.Cameron, "Integer/floating-point Reconfigurable ALU", Technical Report LA-UR #99-5535, Los Alamos National Laboratory, sep 1999.
- [5] D.Lavenier, Y.Solihin, W. Cameron, "Reconfigurable arithmetic and logic unit". SympA'6: 6th Symposium on New Machine Architectures, Besancon, France, June 2000.

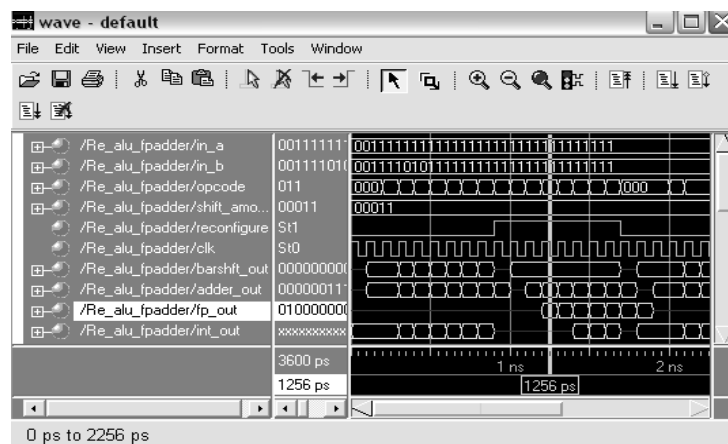


Fig. 11 Simulation of Reconfigurable ALU