Institut für Parallele und Verteilte Systeme
Abteilung Parallele Systeme
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis Nr. 3071

Development of Stochastic Floating Point Unit for Numerical Accuracy Analysis

Moath Mheidat

Studiengang:	M.Sc. in Information Technology
Prüfer:	Prof. Dr.-Ing. Sven Simon
Betreuer:	M.Sc. Wenbin Li
Begonnen am:	01.06.2010
Beendet am:	01.12.2010
CR-Klassifikation:	B.5.1, C.3, G.1.0

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Stuttgart, 1.12.2010 Moath Mheidat

Dedicated to my beloved parents, sisters and brothers.
It is only under the shadow of their prayers and support
that I have achieved this milestone.

Abstract

Recently, fast computers have led to more complex scientific computations involving more and more arithmetic operations. Due to the use of finite precision arithmetic for Floating Point (FP) numbers, each elementary FP operation may induce a rounding error. Consequently, the computed results are affected by the rounding error propagation. The final result can be completely different from that which is expected.

Numerical accuracy analysis with respect to rounding error propagations is a general concern in both scientific and engineering applications. The Discrete Stochastic Arithmetic (DSA) is a powerful numerical accuracy analysis method that enables one to estimate the number of exact significant digits of any computed result. DSA is based on a probabilistic approach for round-off error analysis. Stochastic arithmetic operations are computed on a Stochastic Floating Point Unit (SFPU), where the rounding of the results is randomly chosen. However, software implementation of this method suffers from computational bottlenecks, while hardware alternative would accelerate the computation.

The purpose of this work is to develop a single-precision SFPU and to use it as the basis of the DSA method. The SFPU is then connected to the MicroBlaze processor via the Processor Local Bus (PLB). Virtex-5 Field-Programmable Gate Array (FPGA) is used to synthesis the system. In addition, The SFPU and the system have been verified to show the correctness, reliability as well as the performance of the system.

Contents

1 Introduction 17

2 Discrete Stochastic Arithmetic 21

2.1 Rounding Error in Floating Point Arithmetic 21

2.2 A Stochastic Approach of Rounding Errors 23

2.2.1 The CESTAC Method 23

2.2.2 Stochastic Arithmetic 24

2.3 Numerical Algorithms 25

3 Implementation Environment 27

3.1 Field Programmable Gate Array (FPGA) 27

3.1.1 Overview 27

3.1.2 Virtex-5 FPGAs 28

3.2 MicroBlaze 29

3.2.1 Instructions 30

3.2.2 Registers 31

3.2.3 Pipeline Architecture 32

3.2.4 User and Privileged Modes 32

3.2.5 Memory Architecture 33

3.2.6 Virtual-Memory Management 37

3.2.7 Reset, Interrupts, Exceptions and Break 38

3.2.8 Floating Point Unit (FPU) 39

3.2.9 Instruction and Data Caches 40
3.3 Hardware Description Language (HDL) 40
3.4 Xilinx Tools 40
3.5 ModelSim 42

4 Stochastic Floating Point Unit Implementation 43
4.1 SFPU Design Overview 43
4.2 Design Elements 44
4.2.1 Open-Source Elements 44
4.2.2 Registers 48
4.2.3 Control Elements 48
4.2.4 Random Rounding Unit 50
4.3 Stochastic Floating Point Components 51
4.3.1 SFP Addition/Subtraction Component 52
4.3.2 SFP Multiplication Component 56
4.3.3 SFP Division Component 59
4.4 How To Use The Components 62
4.5 Synthesis Details 65
4.6 Design Verification 66

5 Conclusion and Future Works 71

Bibliography 73

List of Figures

3.1 FPGA structure 28
3.2 MicroBlaze block diagram 30
3.3 Word data type 30
3.4 PLB interconnect diagram 34
3.5 OPB system 35
3.6 MicroBlaze system with two LMB modules 36
3.7 FSL bus block diagram 37
3.8 IEEE 754 single precision format 39
3.9 Basic Embedded System Design Process 41

4.1 SFP components attached to MicroBlaze 44
4.2 libhdlfltp library representation format of FP numbers 44
4.3 Addition Unit 45
4.4 Multiplication Unit 46
4.5 Division Unit 46
4.6 Register's block diagram 48
4.7 Select Register Input block diagram 48
4.8 Operation Select, Sign Changer, and Mux block diagrams 49
4.9 Pulse Generator, Ready Generator, Write Decoder, and Read Decoder block diagrams 50
4.10 Fibonacci-style LFSR 50

4.11 Rounding unit output 50
4.12 SFP Components block diagram 51
4.13 SFP Addition timing diagram 53
4.14 SFP Addition/Subtraction Component block diagram 55
4.15 SFP Multiplication timing diagram 57
4.16 SFP Multiplication Component block diagram 58
4.17 SFP Division timing diagram 60
4.18 SFP Division Component block diagram 61
4.19 A flow diagram of how to use the SFP Addition/Subtraction Component . 62
4.20 A flow diagram of how to use SFP Multiplication and Division Components 64
4.21 SFP Addition output 66
4.22 SFP Subtraction output 67
4.23 SFP Multiplication output 67
4.24 SFP Division output 68

List of Tables

3.1 MicroBlaze general information 29
3.2 Instructions functional categories 31
3.3 Special purpose registers description 32

4.1 libhdlfltp Exn field values and meanings 45
4.2 Input and output signals of addition, multiplication, and division units 47
4.3 Rounding modes 47
4.4 Floating point operations and their number of clock cycles 48
4.5 Input and output signals of the SFP Components 51
4.6 The elements and the number of instances for each SFP Component 52
4.7 SFP Addition/Subtraction registers 52
4.8 SFP Multiplication registers 56
4.9 SFP Division registers 59
4.10 XPS synthesis summary 65
4.11 Device utilization summary 66
4.12 Addition operation results 66
4.13 Subtraction operation results 67
4.14 Multiplication operation results 67
4.15 Division operation results 68
4.16 Working together results 68
4.17 Inner vector multiplication results 69

List of Abbreviations

FP	Floating Point
DSA	Discrete Stochastic Arithmetic
SFPU	Stochastic Floating Point Unit
FPGA	Field-Programmable Gate Array
CESTAC	Controle et Estimation Stochastique des Arrondis de Calculs
HDL	Hardware Description Language
RISC	Reduced Instruction Set Computer
VHDL	VHSIC Hardware Description Language
PLB	Processor Local Bus
OPB	On-Chip Peripheral Bus
LMB	Local Memory Bus
XCL	Xilinx CacheLink
FSL	Fast Simplex Link
MMU	Memory Management Unit
FPU	Floating Point Unit
ISE	Integrated Software Environment
EDK	Embedded Development Kit
XPS	Xilinx Platform Studio
SDK	Software Development Kit
GUI	Graphical User Interface
SFP	Stochastic Floating Point

LFSR	Linear Feedback Shift Register
LUT	Lookup Table
RAM	Random Access Memories
DSP	Digital Signal Processing

Chapter 1

Introduction

In complex scientific computations, the accuracy of results of floating point operations is degraded because of the induced rounding errors in the elementary operations and their propagation to the final result.

Generally, there are two approaches to analyze the propagation of rounding errors, one is deterministic and the second is stochastic [1].

An algorithm is an ordered sequence of equations of the form:

$$s = f(a_1, a_2, a_3, \ldots, a_n) \tag{1.1}$$

Where s represents the new quantity computed at ach step in the algorithm and applied to the quantities $a_1, a_2, a_3, \ldots, a_n$. s is represented on the computer as S because of the propagation of rounding errors.

Two types of deterministic analysis exist for rounding errors, forward analysis of errors and backward analysis of errors. The first type attempts to compare S and s at each step of computing, and to calculate a bound for $|S - s|$. It often proves to be

very difficult, and sometimes unfeasible, and does not offer a good way to approach the problem of analyzing rounding errors.

In backward analysis, which is practically simpler than forward analysis and helps to analyze the propagation of rounding errors in certain algorithms, at each step, *S* is expressed as the exact solution of the equation which is obtained by disturbance of the initial equation:

$$S = f(a_1 + e_1,\ a_2 + e_2,\ a_3 + e_3, \ldots,\ a_n + e_n) \tag{1.2}$$

The existence of the e_i is established and bounds are set in order to estimate the maximum size of the disturbance.

Moreover, some methods using extensions of the usual arithmetic have been introduced to obtain guarantees on the computed results, by the computer itself. These results are expressed in the form of intervals. In the interval arithmetic of Moore [2], a floating-point number *X* is considered to be the image of an interval, the interval of all the real numbers which, after rounding, produce the number *X*. Moore investigated how the operations and functions of a computer act on these new operands, which are no longer numbers but intervals, and how to compute the result interval containing the exact results.

With respect to deterministic methods, we can state that, while they are very fertile from the theoretical point of view for a close understanding of rounding errors, in practice, they lead to a pessimistic estimation of the rounding error on the results.

The CESTAC (Controle et Estimation Stochastique des Arrondis de Calculs) method, which has been developed by La Porte and Vignes [1], is based on a probabilistic approach of rounding errors and enables one to estimate the number of exact significant digits of any computed result. Discrete stochastic arithmetic (DSA) is the modelling of the synchronous implementation of the CESTAC method on a computer. DSA is a powerful method that helps to provide the accuracy of any computed result and to decide if the results are reliable or not.

A Floating Point Unit (FPU) is a part of a processor system specially designed to carry out operations on floating point numbers. Typical operations are addition, subtraction, multiplication and division. A Stochastic Floating Point Unit (SFPU) has an additional feature in which the rounding mode of the results is randomly selected from the modes that are defined by the designer. The SFPU is used as the basis for the DSA method to perform stochastic operations.

This work is organized as follow: Chapter 2 is dedicated to review the principle of the DSA. An overview about the implementation environment is given in chapter 3. Chapter 4 explains and details the stochastic floating point unit design and implementation; in addition, it shows the synthesis details and the verification of the design. The conclusion and future works are presented in chapter 5.

Chapter 2

Discrete Stochastic Arithmetic

In this chapter the principle of the DSA will be reviewed. In section 3.1, the round-off error in floating-point arithmetic is investigated. The main approaches to analyze the propagation of rounding errors are explained in section 3.2, where the CESTAC and the DSA methods are also explained. Section 3.3 talks about the categories of the numerical algorithms and the raised problems when these algorithms are implemented on a computer.

2.1 Rounding Error in Floating Point Arithmetic

Any arithmetic operation defined by:

$$z = x \, \omega \, y, \tag{2.1}$$

where x and y are real numbers and ω is the operation (+, -, *, /), is performed by a computer as:

$$Z = X \, \Omega \, Y \tag{2.2}$$

Where X and Y are in the set of floating point values that can be represented on the computer, and Ω is the floating point operation done by the computer.

The absolute error is defined by:

$$\varepsilon = Z - z = X \,\Omega\, Y - x \,\omega\, y. \tag{2.3}$$

Generally, x and y may not be coded exactly in the computer, so X and Y are given by:

$$X = x(1 + \alpha_x), \tag{2.4}$$

$$Y = y(1 + \alpha_y), \tag{2.5}$$

where α_x and α_y are the relative rounding errors.

When Z is computed, because of the rounding error, we obtain:

$$Z = X \,\Omega\, Y = X \,\omega\, Y + \alpha_z (X \,\omega\, Y), \tag{2.6}$$

and then $Z = (x + \alpha_x x) \,\omega\, (y + \alpha_y y) + \alpha_z\, ((x + \alpha_x x) \,\omega\, (y + \alpha_y y))$.

So, the absolute error is given by:

$$\varepsilon = Z - z = (x + \alpha_x x) \,\omega\, (y + \alpha_y y) + \alpha_z\, ((x + \alpha_x x) \,\omega\, (y + \alpha_y y)) - x \,\omega\, y \tag{2.7}$$

And Z is defined by:

$$Z = z + \varepsilon = z\,(1 + \varepsilon/z) = z\,(1 + \beta), \tag{2.8}$$

where β is the rounding error propagation.

At each step of computing, the error in a result is always composed of two parts, one is resulting from the round-off error propagation of the previous steps, and the other is resulting from the round-off error in the current step. The propagation of these errors means that the accuracy of the results evolves throughout the computation [3].

2.2 A Stochastic Approach of Rounding Errors

2.2.1 The CESTAC Method

The CESTAC method is based on a probabilistic approach of rounding errors and enables one to estimate the number of exact significant digits of any computed result.

The idea behind this method is to consider that every result *R* of a floating-point operation is surrounded by two consecutive floating-point results, one rounded off from below *F*-, the second rounded off from above *F*+, and each of them representing the exact arithmetical result *r*, with equal validity.

The same code is executed *N* times with different rounding error propagation for each run and the common parts of the results are estimated and considered to be representative of the exact result. These different rounding error propagations are practically obtained by using the random rounding mode which selects randomly either *F*- or *F*+ as a result with an equal probability of 1/2.

After running the code *N* times, *N* results are available. The final result of the computation is given by CESTAC as the mean value ($\overline{R}$) of these *N* results (R_i), and the number of significant digits of this final result is accordingly computed.

The mean value of the *N* executions is given by:

$$\overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \tag{2.9}$$

The number of exact significant digits of R' is estimated by:

$$C_{\overline{R}} = \log_{10} \frac{\sqrt{N}.|\overline{R}|}{\sigma.\tau_\beta} \tag{2.10}$$

with $$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(R_i - \overline{R})^2 \tag{2.11}$$

τ_β is the value of the Student's distribution for N - 1 degrees of freedom and a probability level 1 - β. In practice, $N = 3$, $\beta = 0.05$ and then $\tau_\beta = 4.4303$.

The result provided by equation (12) is reliable if the following two hypotheses hold in practice [1, 4]:

- Hypothesis 1. The elementary rounding errors of the floating-point arithmetic operations are random independent, centered and uniformly distributed variables.
- Hypothesis 2. The approximation of the first order in 2^{-p} (p is the number of bits of the mantissa) is legitimate.

The concept of informatical zero (also called computational zero and denoted by @.0) was introduced to treat non-significant results. The result provided by the CESTAC method is considered as an informatical zero if one of the following conditions is true:

1. For every i, $R_i = 0$, (2.12)
2. $C_{\overline{R}} \leq 0$.

The first condition means that all the results of the N runs are null, and the second one means that there is no significant digits.

The validity of the CESTAC method can be assured by having the results of multiplications and divisions always significant. In addition, the comparisons should be executed only on the significant part of the operands. So it is absolutely necessary to detect the emergence of informatical zeros on each operation during a run of code. To achieve this, a synchronous implementation of the CESTAC method is used in which each arithmetic operation is performed N times with the random rounding mode before performing another.

2.2.2 Stochastic Arithmetic

Two types of stochastic arithmetic actually exist; it can be either continuous or discrete.

By using the CESTAC method, so that the N runs of a code take place in parallel. The N results of each arithmetical operation can be considered as realizations of a Gaussian random variable centered on the exact mathematical result [1, 5].

Therefore a new number, called stochastic number, is defined as an *N*-tuple whose components are the *N* values provided by the CESTAC method, and then a new arithmetic, called continuous stochastic arithmetic, is applied to these numbers. Continuous stochastic arithmetic is the modelling of the synchronous implementation of the CESTAC method. The properties established in its theoretical framework can be applied on a computer via the practical use of the DSA [3].

The elements of the DSA are the stochastic numbers. An arithmetic stochastic operation is performed on these stochastic numbers by executing the corresponding floating-point operation on each component of the stochastic operands, and the result of each operation is also a stochastic number. Each result is rounded using the random rounding mode. The average result is computed using equation (2.9) and the number of exact significant digits of this average result can be estimated from equation (2.10).

Discrete stochastic arithmetic is a powerful method that helps to achieve the validation of scientific codes. It serves during the run of scientific code to:

- Estimate the accuracy of results with respect to exact numerical results.
- Detect the numerical instabilities.
- Eliminate the programming expedients that are absolutely unfounded.
- Decide if the provided results are reliable or not.

2.3 Numerical algorithms

From the mathematical perspective, numerical algorithms are divided into three categories: direct algorithms, iterative algorithms and approximate algorithms [3].

- Direct algorithms: These algorithms give exact results after finite and ordered sequences of arithmetic computations. The computing of discrete linear mathematical transforms, such as the Fast Fourier Transform, belongs to this category.

- Iterative algorithms: These algorithms provide results as limits of sequences. Jacobi methods, Raphson-Newton's method for solving nonlinear equation and the algorithms for solving the optimization problems can be classified in this category.

- Approximate algorithms: In these algorithms, an approximation is given for the exact results. For example, any finite method for solving differential systems belongs to this category.

From the computational perspective, when these algorithms are performed on a computer using the floating-point arithmetic of the computer, several problems arise and false results may be furnished to the user.

- For direct algorithms, the results given by the computer always have errors which are the consequences of the rounding error propagation. But when these algorithms are performed with the CESTAC method, numerical instabilities are detected and the result provided with its accuracy.

- For iterative algorithms, the results given by the computer contain a double error, one resulting from the termination criterion of the iterative process, and the other resulting from rounding error propagation. The implementation of the CESTAC method in any iterative algorithm is able to break off iterations correctly and to estimate the accuracy of the computed result.

- For approximate algorithms, the results given by the computer also contain a double error, one resulting from the approximation "method error" and the other resulting from rounding error propagation. The CESTAC method is a precious tool to deal with types of errors.

Chapter 3

Implementation Environment

This chapter introduces the implementation environment used in this work. Section 3.1 gives an overview about the FPGAs and specially Vertix-5 FPGA that is used in this work. Section 3.2 contains an overview of MicroBlaze processor features and some detailed information about MicroBlaze architecture including registers, memory architecture, virtual-memory management and floating point unit. A definition for a Hardware Description Language (HDL) is given in section 3.3. Section 3.4 presents a description about the Xilinx tools that deals with all aspects of embedded systems development.

3.1 Field-Programmable Gate Array (FPGA)

3.1.1 Overview

FPGA is an integrated circuit that is completely manufactured. It is designed to be configured and programmed in the field after manufacturing by the designer or customer. FPGAs are well suited for prototyping integrated circuit designs and small production volumes. When the design is ready, hardwired chips can be produced in large quantities and better performance. A hardware description

language (HDL) is used to configure FPGAs. Figure 3.1 shows a general FPGA structure.

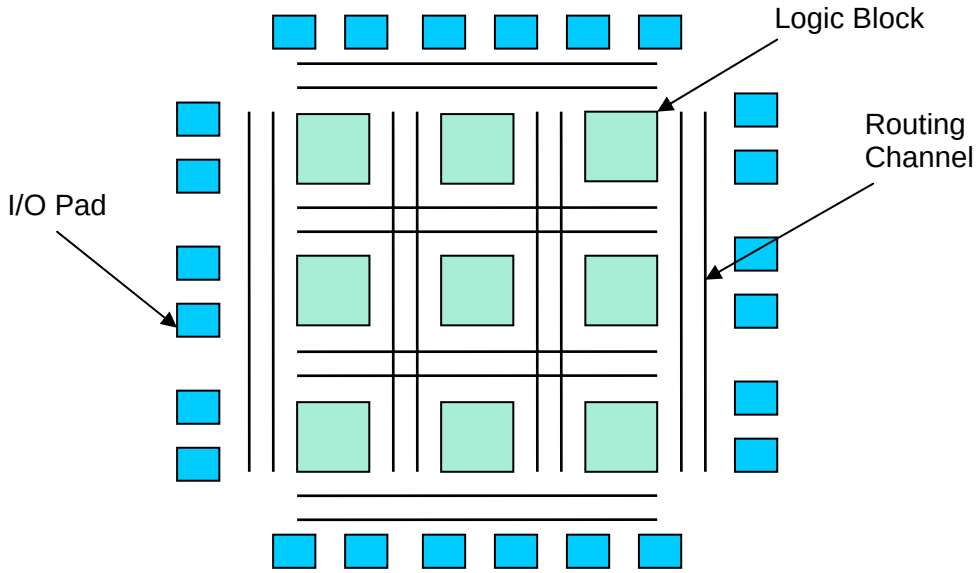

Figure 3.1: FPGA structure

An FPGA contains reconfigurable logic components (logic blocks) and programmable interconnects (routing channels). Logic blocks can be configured to execute any functionality, from basic logic gate to any complex combinational function or memory element. Interconnects are used to connect the logic blocks together.

3.1.2 Virtex-5 FPGAs

In 1998, Virtex FPGA family was introduced by Xilinx. Virtex-II, Virtex-E, Virtex-4 and Virtex-5 are some products of this series. The Virtex-5 family, which is used in this thesis, provides very powerful features in the FPGA market. This family is available in five distinct platforms LX, LXT, SXT, TXT, and FXT. For example, Virtex-5 SXT has high-performance signal processing applications with advanced serial connectivity. Generally, Virtex-5 FPGAs have the following main features:

- Available in five platforms LX, LXT, SXT, TXT, and FXT
- Most advanced, high-performance, optimal-utilization, FPGA fabric
- Powerful clock management tile (CMT) clocking

- 36-Kbit block RAM/FIFOs
- High-performance parallel SelectIO technology
- Flexible configuration options

3.2 MicroBlaze

The MicroBlaze is a 32-bit soft processor core designed by Xilinx as a reduced instruction set computer (RISC) and used in Xilinx FPGAs. Table 3.1 shows general information about MicroBlaze.

Designer	Xilinx
Type of core	Soft
Architecture	32-bit pipelined RISC
Address bus	32 bits
Instruction width	32 bits
Addressing modes	Two
Endianess	Big
General Purpose Registers	32 x 32 bits

Table 3.1: MicroBlaze general information

The MicroBlaze soft core is highly configurable, so it can be parameterized to slice off the unwanted functionality. The key benefits of using a soft processor include configurability to trade between price and performance, faster time to market, easy integration with the FPGA, improving and extending product life-cycles, creating application-specific coprocessors, and avoiding obsolescence.

The main fixed features of MicroBlaze include Thirty-two 32-bit general purpose registers, 32-bit address bus, 32-bit instruction word with two addressing modes, basic ALU and shift operations, and single issue pipeline.

Some of the features which can be customized are the cache size, pipeline depth (3-stage or 5-stage), embedded peripherals, memory management unit, and bus-interfaces. In addition, hardware support for certain operations, such as multiplication, division, and floating-point arithmetic, can be added or removed. Figure 3.2 shows the Microblaze block diagram and the optional MicroBlaze features.

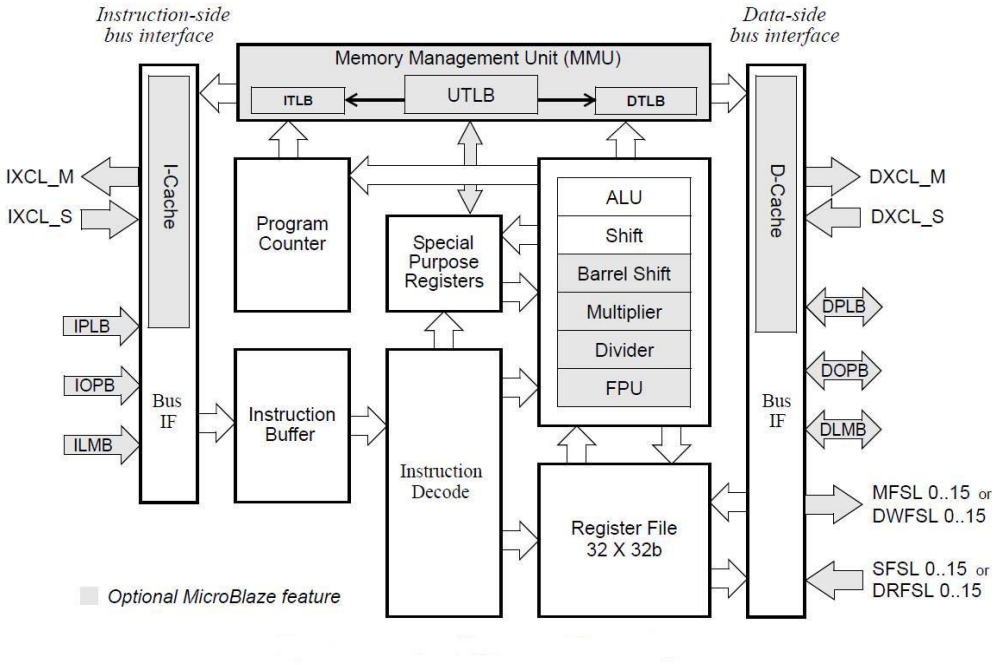

Figure 3.2: MicroBlaze block diagram [6]

The format that is used by MicroBlaze to represent data is Big-Endian and the hardware supports byte, half-word, and word data types. Figure 3.3 shows the organization of the word data type. Byte and half-word have the same organization.

Byte address	n	n+1	n+2	n+3
Byte label	0	1	2	3
Byte significance	MSByte			LSByte
Bit address	0			31
Bit significance	MSBit			LSBit

Figure 3.3: Word data type

3.2.1 Instructions

The width of all instructions in MicroBlaze is 32 bits. Generally, the MicroBlaze has two addressing modes. The first one has up to two source register operands and one destination register operand. The second one has one source register operand, one 16-bit immediate operand, and one destination register operand. The MicroBlaze has 124 instructions. Table 3.2 shows the functional categories of instructions.

Instruction category	Number of instructions
integer arithmetic	22
Logical	8
Shift	9
branch	36
floating point	11
load/store	12
return	4
other	22

Table 3.2: Instructions functional categories

3.2.2 Registers

MicroBlaze provides two types of registers, general-purpose registers and special purpose registers. MicroBlaze has thirty-two 32-bit general purpose registers and up to eighteen 32-bit special purpose registers, depending on the selected features. Table 3.3 provides a brief description of each special purpose register.

Register	Description
Program Counter (PC)	Address of the next instruction to be executed
Machine Status Register (MSR)	Contains control and status bits for the processor
Exception Address Register (EAR)	Stores the full load/store address that caused the exception
Exception Status Register (ESR)	Contains exception status bits for the processor
Branch Target Register (BTR)	Stores the branch target address for all delay slot branch instructions
Floating Point Status Register (FSR)	Contains status bits for the floating point unit
Exception Data Register (EDR)	Stores data read on an FSL link that caused an FSL exception

Process Identifier Register (PID)	Identifies a software process during MMU address translation
Zone Protection Register (ZPR)	Used to override MMU memory protection defined in TLB entries
Translation Look-Aside Buffer Low Register (TLBLO)	Used to access MMU Unified Translation Look-Aside Buffer (UTLB) entries
Translation Look-Aside Buffer High Register (TLBHI)	Used to access MMU Unified Translation Look-Aside Buffer (UTLB) entries
Translation Look-Aside Buffer Index Register (TLBX)	Used as an index to the Unified Translation Look-Aside Buffer (UTLB) when accessing the TLBLO and TLBHI registers
Translation Look-Aside Buffer Search Index Register (TLBSX)	Used to search for a virtual page number in the Unified Translation Look-Aside Buffer (UTLB)

Table 3.3: Special purpose registers description

3.2.3 Pipeline Architecture

MicroBlaze has a pipelined architecture. Depending on the area optimization feature, the pipeline can be three stages or five stages. If the area optimization is enabled, then the pipeline is divided into three stages which are Fetch, Decode, and Execute. If the area optimization is disabled, then the pipeline is divided into five stages which are Fetch (IF), Decode (OF), Execute (EX), Access Memory (MEM), and Writeback (WB). In addition, an instruction prefetch buffer is implemented by MicroBlaze to reduce the impact of memory instruction latency.

3.2.4 User and Privileged Modes

User mode and privileged mode are the execution modes used by MicroBlaze. The privileged instructions like (GET, PUT...etc) will cause a privileged instruction exception if it is executed in the user mode. If one of the following things happened, then the processor will switch from the user mode to the privileged mode:

- Hardware reset
- Hardware exception
- Non-maskable break or hardware break
- Interrupt
- Executing the instruction "BRALID Re, 0x8” to perform a user vector exception
- Executing the software break instructions “BRKI” jumping to physical address 0x8 or 0x18

3.2.5 Memory Architecture

MicroBlaze uses Harvard memory architecture in which it has two 32-bit range address spaces for instruction and data accesses. The accesses to instructions and data must be aligned. In addition, the following memory interfaces are supported by MicroBlaze:

- Processor Local Bus (PLB) or On-Chip Peripheral Bus (OPB)
- Local Memory Bus (LMB)
- Xilinx CacheLink (XCL)

Processor Local Bus (PLB)

The PLB, which is used in this work, provides bus infrastructure for connecting an optional number of PLB masters and slaves into an overall PLB system. It consists of a bus control unit, a watchdog timer, and separate address, write, and read data path units, as well as an optional DCR (Device Control Register) slave interface to provide access to its bus error status registers.

The PLB has the following features [7]:

- Arbitration support for a configurable number of PLB master devices
- PLB address and data steering support for all masters
- 128-bit, 64-bit, and 32-bit support for masters and slaves
- PLB address pipelining (supported in shared bus mode or point-to-point configuration)
- Three-cycle arbitration
- Four levels of dynamic master request priority
- Selectable round robin or fixed priority arbitration

- Configurable optimization for point-to-point topology
- PLB watchdog timer
- PLB architecture compatible
- Complete PLB bus structure provided
    - Supports a configurable number of slave devices
    - No external OR gates required for PLB slave input signals
- PLB Reset circuit
    - PLB Reset generated synchronously to the PLB clock
    - PLB Reset generated synchronously from external reset when external reset provided
    - Provides vectorized reset signal to reduce system fanout for improved timing
    - Active state of external reset selectable via a design parameter

Figure 3.4 shows an example of the PLB connections for a system with one master and one slave which is composed of central bus arbiter, the necessary bus control and gating logic, and all necessary bus OR/MUX structures.

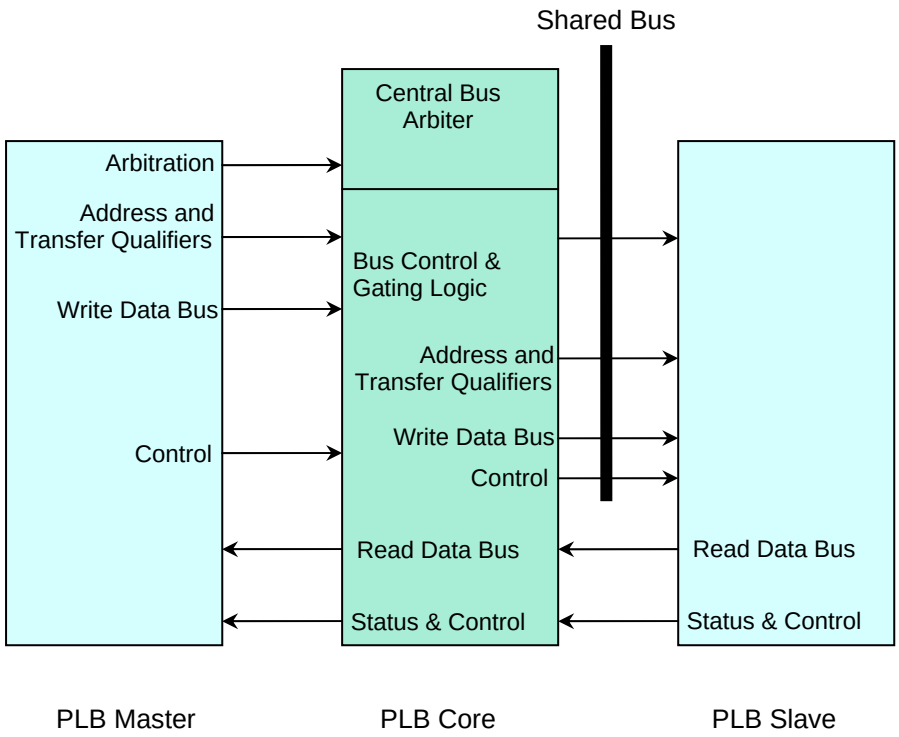

Figure 3.4: PLB interconnect diagram

On-Chip Peripheral Bus (OPB)

The OPB is based on the IBM's 64-Bit On-Chip Peripheral Bus which is designed for easy connection of on-chip peripheral devices. OPB interface provides a connection to both on-chip and off-chip peripherals and memory. The bus interconnect in the OPB is a distributed multiplexer implemented as an AND function in the master or slave driving the bus, and an OR function to combine the drivers into a single bus. It includes a parameterized OPB Arbiter and a parameterized I/O signals to support up to 16 masters and any number of slaves [8]. Figure 3.5 shows an OPB system which is composed of masters, slaves, a bus interconnect and an arbiter.

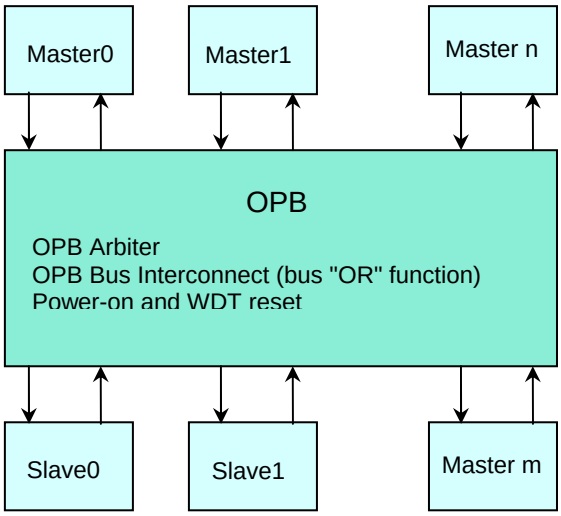

Figure 3.5: OPB system

Local Memory Bus (LMB)

The LMB has the following features:

- It is a highly optimized, fast, and efficient synchronous bus.
- It is used to access high-speed peripherals, mainly on-chip block RAM (BRAM).
- It has a separate read and write data buses.
- To ensure that local block RAM is accessed in a single clock cycle, a minimum number of control signals and a simple protocol are used.
- The FPGA resource utilization is low.
- Arbiter is not required because it uses a single bus master.

Figure 3.6 shows a MicroBlaze system with two LMB modules where both I and D side LMB buses are connected to a dual-ported BRAM Block via separate LMB BRAM interface controllers.

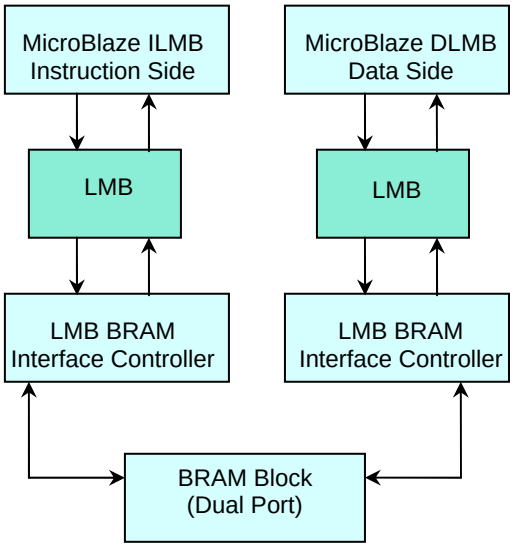

Figure 3.6: MicroBlaze system with two LMB modules

Fast Simplex Link (FSL)

The FSL has the following features [9]:

- It is a uni-directional point-to-point communication channel bus between an output FIFO and an input FIFO.
- It is used to perform fast communication between any two design elements on the FPGA when implementing an interface to the FSL bus.
- A mechanism for unshared and non-arbitrated communication is available. This can be used for fast transfer of data words.
- It provides an extra control bit that can be used, for example, to decode the word being transmitted as a control word or use the bit to indicate the start or end of the transmission of a frame.
- Both synchronous and asynchronous FIFO modes are supported. This allows the master and slave side of the FSL to clock at different rates.

Figure 3.7 shows FSL bus block diagram where FSL_M_Clk is the input clock to the master interface and FSL_S_Clk is the input clock to the slave interface and so on and so forth for all signals.

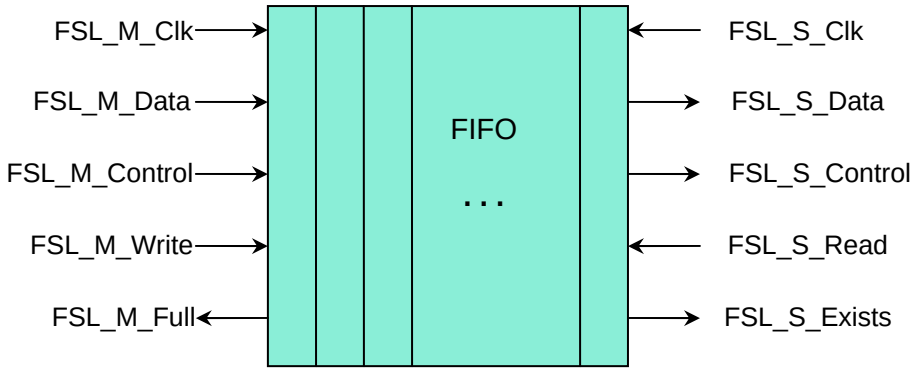

Figure 3.7: FSL bus block diagram

Xilinx CacheLink (XCL)

The XCL has the following features [6]:

- It is a high performance solution for external memory accesses.
- It is designed to connect directly to a memory controller with integrated FSL buffers.
- It is only available on MicroBlaze when caches are enabled.
- The CacheLink cache controllers handle 4 or 8-word cache lines with critical word first.

3.2.6 Virtual-Memory Management

When the memory in MicroBlaze is accessed by the programs, the addresses used by the programs (effective address) are translated to physical addresses according to two modes, real mode and virtual mode. In the real mode effective addresses are directly used as physical addresses to access the physical memory, while in the virtual mode the memory management unit (MMU) is used to translate effective addresses into the physical ones.

Virtual mode is implemented in MicroBlaze by MMU. MMU is a computer hardware component responsible for handling accesses to the memory requested by the processor. Its functions include translation of effective addresses to physical addresses (virtual memory management), memory protection, cache control, and page-level control during address translation. MMU supports eight page sizes from 1kB to 16 MB in a quadruple way.

3.2.7 Reset, Interrupts, Exceptions and Break

MicroBlaze supports reset, interrupts, exceptions, and break. All of these events can be summarized as follow where the priority increases when we go from the user vector exception to the reset:

- User Vector Exception:
  - It is caused by inserting a BRALID Rx, 0x8 instruction to the code
  - The user exception vector is located at address 0x8
  - R15 is recommended to store the return address
  - RTSD is used to return from the user exception handler

- Interrupt:
  - One external interrupt source is supported
  - MSR's interrupt enable bit must be set
  - The instruction in the execution stage is completed while the instruction in the decode stage is replaced by a branch to the interrupt vector
  - The return address (the address of the instruction in the decode stage) is loaded into R14
  - Future interrupts are disabled by clearing the IE bit in the MSR until executing RTID instruction
  - RTID is used to return from the handler
- Breaks:
  - There are two types, hardware (external) and software (internal)
  - Hardware breaks are performed by asserting the external break signal
  - The instruction in the execution stage is completed while the instruction in the decode stage is replaced by a branch to the break vector
  - The return address (the address of the instruction in the decode stage) is loaded into R16
  - Break In Progress (BIP) flag in MSR is set
  - Software breaks are performed by executing the BRK and BRKI instructions

- Hardware Exceptions:
  - It causes MicroBlaze to flush the pipeline and branch to the vector (address 0x20)
  - The instruction in the execution stage is not executed

- The return address is stored in R17
- Some examples are:
  - FSL exception
  - Instruction Bus Exception
  - Illegal Opcode Exception
  - FPU Exception

- Reset
  - When a Reset or Debug_Rst occurs, MicroBlaze:
    - flushes the pipeline
    - starts fetching instructions from the reset vector (address 0x0)
  - Both external reset signals are active high
  - They should be asserted for a minimum of 16 cycles

3.2.8 Floating Point Unit (FPU)

The floating point unit in Microblaze supports addition, subtraction, multiplication, division, comparison, conversion, and square root instructions. FPU uses the IEEE 754 single precision format to represent floating point numbers. The default round-to-nearest rounding mode is implemented and if the two represented values are equally near, then the value that has the least significant bit zero is returned. Figure 3.8 shows the IEEE single precision format which is composed of the following three elements:

- 1-bit sign
- 8-bit exponent
- 23-bit fraction

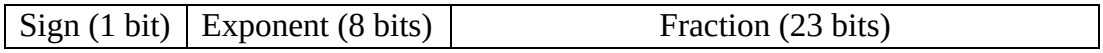

Figure 3.8: IEEE 754 single precision format

When the hardware exception in MicroBlaze is enabled, exceptions are generated for all IEEE 754 standard conditions which are: underflow, overflow, divide-by-zero, and illegal operation. In addition, an exception is generated for the denormalized operand error which is specified for MicroBlaze.

3.2.9 Instruction and Data Caches

Instruction cache and data cache are options that can be used by MicroBlaze in order to improve the performance. They share the following main features [6]:

- Direct mapped (1-way associative)
- User selectable cacheable memory address range
- Configurable cache size and tag size
- Caching over CacheLink (XCL) interface
- Option to use 4 or 8 word cache-lines
- Cache on and off controlled using a bit in the MSR
- Optional WIC and WDC instructions to invalidate cache lines

In addition, the MicroBlaze data cache implements a write-through policy in which every write to the cache causes a write to the main memory.

3.3 Hardware Description Language (HDL)

A Hardware Description Language (HDL) is a computer programming language used to describe and model the operation of digital logic circuits for documentation, simulation and logic synthesis. The two major standards, most widely-used and well-supported HDL languages are VHDL and Verilog.

3.4 Xilinx Tools

To simplify the design process of an embedded processor system, Xilinx provides several set of tools that deals with all aspects of embedded systems development. The main function of each of these tools can be described as follow:

- The Integrated Software Environment (ISE) is the foundation for Xilinx FPGA logic design. It consists of an integrated tool suite that enables the developer to produce, test, and implement designs for Xilinx FPGAs. The available tools cover all aspects of the work flow from design entry to bitstream generation and downloading. The ISE Design Suite features include synthesis and analysis of HDL designs, constraints entry, time analysis, logic placement and routing (PAR), simulation, and creation of the bit files that are used to configure the chip.

- The Embedded Development Kit (EDK) is an environment of tools and Intellectual Property (IP) covering all things related to embedded processor systems and their design. It enables the designer to design a complete embedded processor system for implementation in a Xilinx FPGA device. The Xilinx ISE software must also be installed to run EDK. EDK is then further divided into two separate environments: XPS and SDK [10].

- The Xilinx Platform Studio (XPS) is the development environment that deals with the hardware aspect of the project. It is used to configure and build the hardware part of the embedded processor system.

- Platform Studio Software Development Kit (SDK) is an integrated development environment that deals with the software aspect of the project. It is used for creating and verifying C/C++ embedded software applications.

Figure 3.9 shows a simplified flow for an embedded system development and gives an overview of how these tools work together.

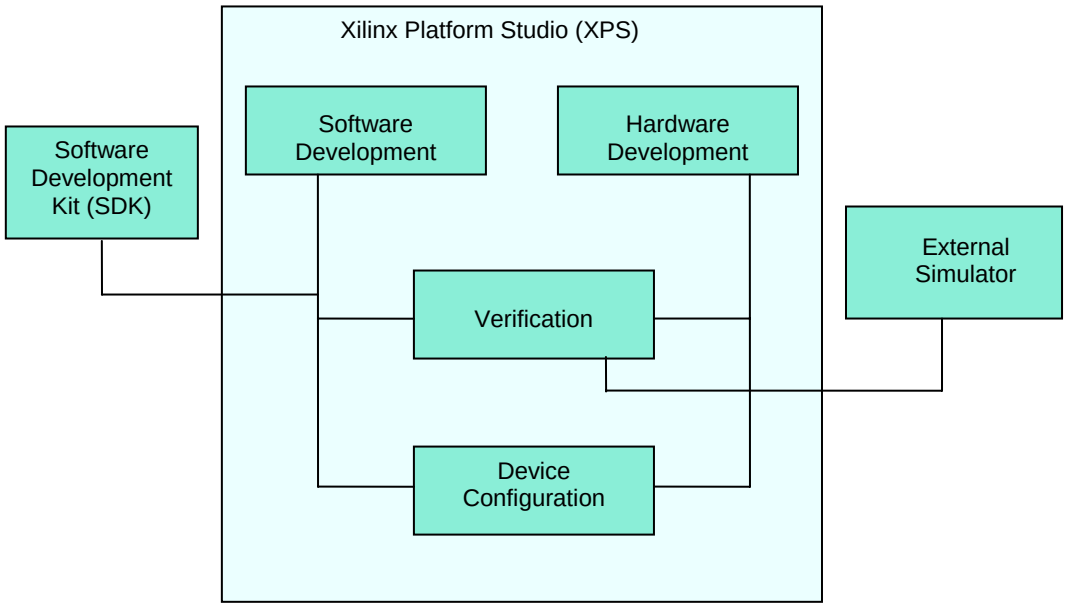

Figure 3.9: Basic Embedded System Design Process

XPS is used primarily for embedded processor system hardware development, and also to develop the basic embedded software applications. SDK is used for developing simple and complex embedded software applications. A Hardware

Description Language (HDL) simulator like ModelSim is used for verifying the correctness of the design.

3.5 ModelSim

ModelSim is a hardware simulation and debug environment by Mentor Graphics. Generally, it has the following features:

- It is a high-performance and scalable environment.
- It provides simulation solutions for a broad range of design sizes and complexities.
- It supports verification of multiple languages like VHDL, Verilog and SystemC.
- The graphical user interface (GUI) allows access to all the inputs, outputs and signals in the design.

Chapter 4

Stochastic Floating Point Unit Implementation

In this work, SFPU is implemented and attached to the MicroBlaze via the PLB bus. The SFPU design will be divided into three main components which are the addition/subtraction, multiplication, and division components. Section 4.1 gives an overview about the SFPU design. Section 4.2 will explain the design elements that will be used to build the components. In section 4.3, the complete design of each component will be demonstrated and explained in details. In order to know exactly how to use each component, section 4.4 gives some examples to clarify that. The synthesis details will be shown in section 4.5. The verification of the design is presented in section 4.6.

4.1 SFPU Design Overview

In this design, the operations that are implemented by SFPU are addition, subtraction, multiplication, and division. SFPU is divided into three different Stochastic Floating Point (SFP) components as follow:

- SFP Addition/Subtraction Component
- SFP Multiplication Component
- SFP Division Component

Each of these components is designed as a stand alone component. Figure 4.1 shows the main design of the system after attaching the three components to the MicroBlaze processor via the PLB bus. In this architecture, each component has a unique address and can be accessed independently.

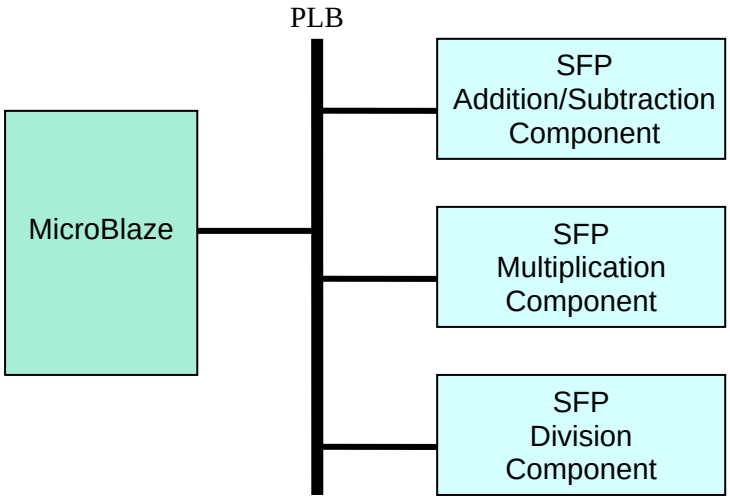

Figure 4.1: SFP components attached to MicroBlaze

4.2 Design Elements

4.2.1 Open-Source Elements

The main elements that are used to execute the floating point operations are taken from an open source library called libhdlfltp [11]. These elements are the addition, multiplication, and division units. The main characteristics of this library are that it is fully pipelined and parameterized.

The representation format of floating point numbers in libhdlfltp library is shown in the following figure:

Exn(2 bits)	Sign(1 bit)	Exponent(wE bits)	Fraction(wF bits)

Figure 4.2: libhdlfltp library representation format of FP numbers

The overall length of the number is:

$$Length = wE + wF + 3\ bits$$

The exponent and fraction have a variable sizes, where wE represents the size of exponent and wF represents the size of fraction. Exn is used for representation exceptions. Table 4.1 shows the values that can be set to Exn field and their meanings.

Exn Value	Meaning
00	Number = (-1)^sign * 0 = IEEE signed zero
01	Number = normalized number (normal case)
10	Number = (-1)^sign * infinity
11	Number = Not a number (NaN)

Table 4.1: libhdlfltp Exn field values and meanings

In this work, the IEEE 754 single precision format is used to represent floating point numbers. So the exponent size (wE) is fixed to be 8 bits and the fraction size (wF) is fixed to be 23 bits. The exception handler is not implemented in this design, so the Exn field is fixed to '01' which is the normal case. In this situation, the size of the represented numbers is 34-bit.

Figure 4.3, figure 4.4, and figure 4.5 show the block diagrams of the main three open-source elements taken from libhdlfltp library. These elements represent the core of the whole design.

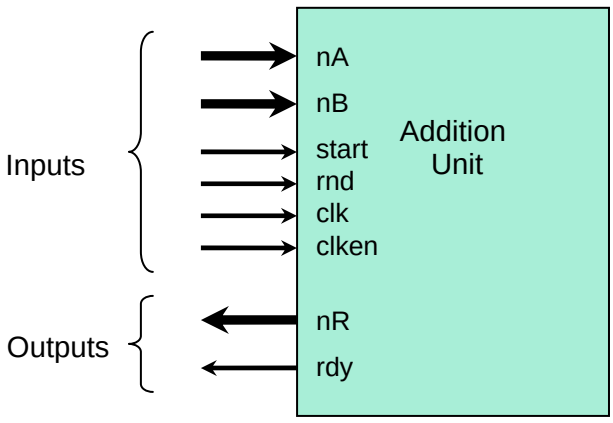

Figure 4.3: Addition Unit

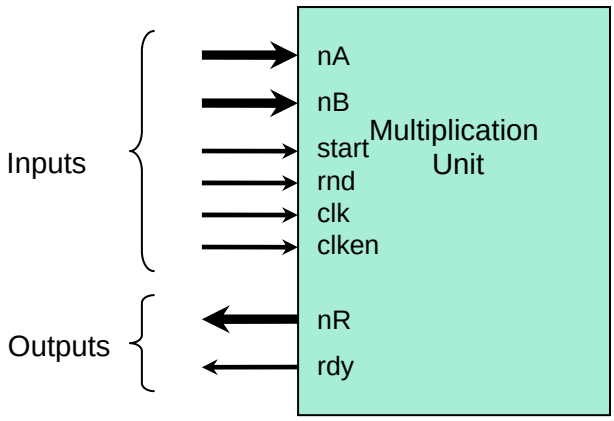

Figure 4.4: Multiplication Unit

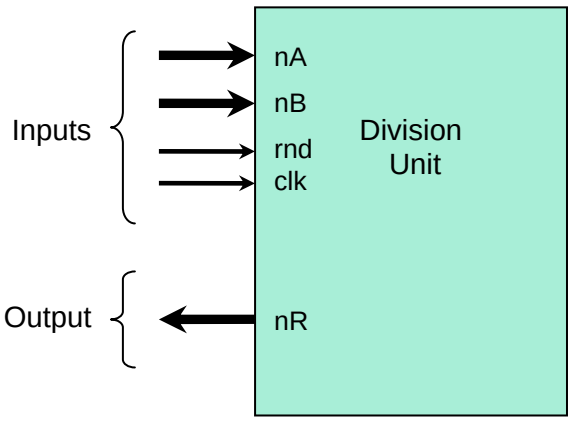

Figure 4.5: Division Unit

Table 4.2 explains the functions and the sizes of the input and output signals of all of these units. The addition and the multiplication units have the same input and output signals. But in the division unit, start, rdy, and clken signals are not existed. An additional logic is built outside the division unit in order to compensate the rdy signal and to tell when the result is ready.

The clken signal is fixed to '1' in the addition and multiplication units in order to make them always enabled. The disappearance of clken in the division unit means that it is always enabled.

Signal	Size in bits	Function
nA	34	First operand
nB	34	Second operand
start	1	Start operation signal
rnd	2	Rounding mode
clk	1	Clock
clken	1	Clock enable
rdy	1	The result is ready
nR	34	The result

Table 4.2: Input and output signals of addition, multiplication, and division units

The rounding modes that can be used and a brief description about them are shown in the following table:

rnd signal	Rounding mode	Description
00	Round to nearest even	Round to the nearer of the two possible result values. If the neither possibility is nearer, then the even alternative is chosen
10	Round to zero	Round to the possible value closer to zero. This is analogous to the Common truncate function
01	Round up	Round to the possible value closer to positive infinity. This is analogous to the common ceiling function
11	Round down	Round to the possible value closer to negative infinity. This is analogous to the common floor function

Table 4.3: Rounding modes

The number of clock cycles that each unit needs to calculate the result is listed in table 4.4.

Operation	Number of clock cycles
Addition	3
Multiplication	4
Division	15

Table 4.4: Floating point operations and their number of clock cycles

4.2.2 Registers

The MiceoBlaze accesses the components as normal memory locations. So the registers are created as part of these components in order to communicate with them. Figure 4.6 represents the block diagram of the register.

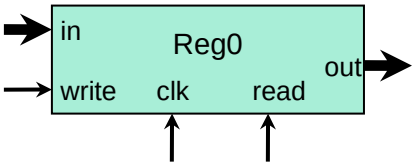

Figure 4.6: Register's block diagram

The register is 32-bit wide. It has a 32-bit input bus and a 32-bit output bus. In addition, it has a clock, write and read input signals.

4.2.3 Control Elements

Some of registers may have more than one source for data input. One source comes always from the processor and a second source may come from internal computations. The Select Register Input component is responsible for selecting the right input according to some control signals. The block diagram for this component is shown in figure 4.7 and table 4.5 shows the truth table for it.

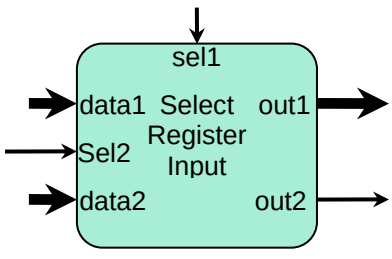

Figure 4.7: Select Register Input block diagram

The value of data1 input will be forwarded to out1 output if sel1 input is activated and the value of data2 input will be forwarded to out1 output if sel2 input is activated. Out2 output will be activated if one of sel1 and sel2 inputs is activated.

The Operation Select, Sign Changer, and Mux elements are used in the SFP Addition/Subtraction Component only which are shown in figure 4.8. The Operation Select element will set the output to '0' or '1' depending on the value of one of the registers, where '0' means addition operation is desired and '1' means subtraction operation is desired.

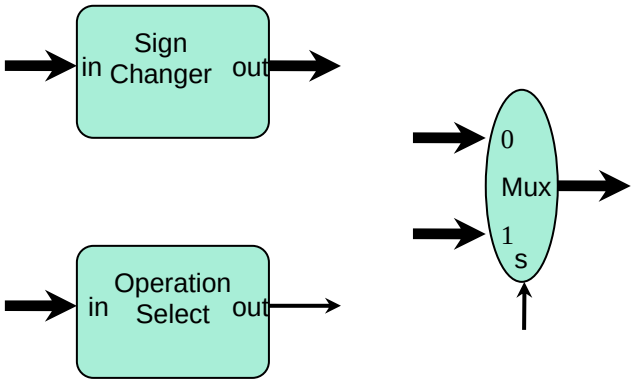

Figure 4.8: Operation Select, Sign Changer, and Mux block diagrams

Normally, any addition unit can be used to do addition and subtraction operations, just by changing the sign of the second operand and feeding it to the second input. The name of the Sign Changer element reflects its function; the sign of the floating point number can be changed by flipping the first bit. The Mux is used to forward one of two inputs to the output depending on a select signal.

Figure 4.9 shows the other control elements which are used in all components. The Pulse Generator element, depending on a trigger input, will send a one clock pulse signal that will be used afterward to start a floating point computation. The Ready Generator will start counting and after fifteen clock cycles the output ready signal of the Ready Generator will be activated. The Write Decoder and Read Decoder are used to activate only one of the output signals depending on the input value. This activated is used to "write to" or "read from" one of the registers. The decoders may have five or four outputs depending on the type of component.

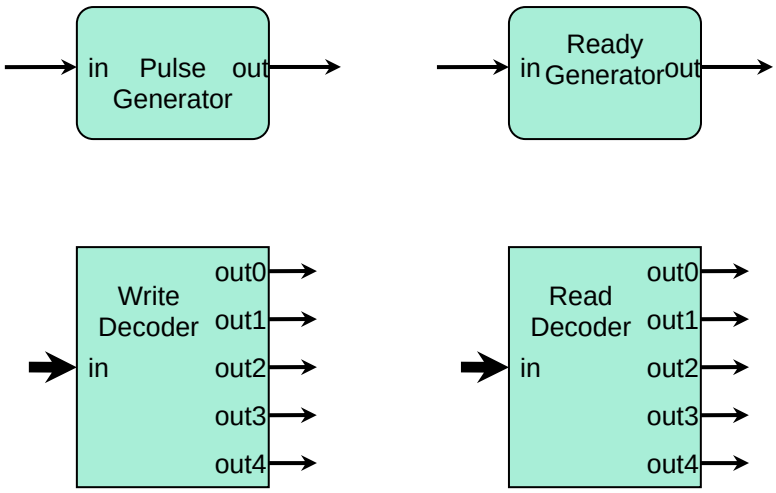

Figure 4.9: Pulse Generator, Ready Generator, Write Decoder, and Read Decoder block diagrams

4.2.4 Random Rounding Unit

The Random Rounding Unit generates two rounding modes which are the round up (01) and round down (11). A Linear Feedback Shift Register (LFSR, Figure 4.7) is used as the pseudo random number generator to change the rounding randomly. Figure 4.10 shows a Fibonacci-style LFSR with taps at 32th, 31th, 30th and 10th bits. It gives a repeating cycle of length 2^{32-1}. A sample of the Random Rounding Unit output is given in figure 4.11.

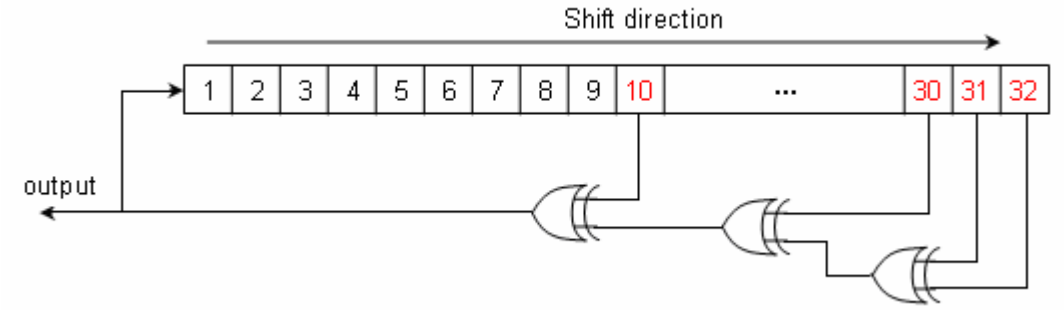

Figure 4.10: Fibonacci-style LFSR

Figure 4.11: Rounding unit output

4.3 Stochastic Floating Point Components

The three SFP components that implement the desired floating point operations are:

- SFP Addition/Subtraction Component
- SFP Multiplication Component
- SFP Division Component

All of these components have generally the same block diagram which is shown in figure 4.12. Table 4.5 explains the functions of input and output signals of these components.

Figure 4.12: SFP Components block diagram

Signal	Function
data_in	Input data
write_sel	Register to be written
read_sel	Register to be read
clk	Clock
data_out	Output data

Table 4.5: Input and output signals of the SFP Components

Table 4.6 shows the elements and the number of instances of each one that are used to build these components.

Element	Number of instance		
	SFP Addition/Subtraction	SFP Multiplication	SFP Division
Write Decoder	1	1	1
Select Register Input	2	2	2
Register	5	4	4
Sign Changer	1	0	0
Pulse Generator	1	1	0
Operation Select	1	0	0
Mux	1	0	0
Ready Generator	0	0	1
Random Rounding Unit	1	1	1
Addition Unit	1	0	0
Multiplication Unit	0	1	0
Division Unit	0	0	1

Table 4.6: The elements and the number of instances for each SFP Component

4.3.1 SFP Addition/Subtraction Component

This is one of the main three components that are used to do the floating point operations. This component is responsible for executing floating point addition and subtraction operations depending on a randomly selected rounding mode. The elements that form this component are shown in table 4.6.

Five registers are necessary to fulfill the functionality of this component. Table 4.7 shows the data that can be stored in each register.

Register	Data
Reg0	First operand
Reg1	Second operand
Reg2	The result of the floating point operation
Reg3	Set to 0x1 When the result is ready
Reg4	Set to 0x0 to select addition operation or 0x1 to select subtraction operation

Table 4.7: SFP Addition/Subtraction registers

Figure 4.14 shows a detailed block diagram for this component, the operation of this detailed data path can be explained as follow:

Reg0 and Reg1 should be written with operand1 and operand respectively. Reg4 should be set to 0x0 in order to select addition operation and to 0x1 in order to select subtraction operation. The value of Reg0 will be concatenated with '01' and will be forwarded to the first operand of the Addition Unit (nA).

The value of Reg1 will be forwarded to the first operand of the Mux and also to the Sign Changer block. The Sign Changer will change the sign of the input and the output will be put on the second operand of the Mux. After writing 0x0 or 0x1 to Reg4, the value will be forwarded to Operation Select block input. The Operation Select will set its output to '0' or '1' if the value of Reg4 is 0x0 or 0x1 respectively. This output will be sent to the select control signal (s) of the Mux. Depending on this signal, the Mux will select one of its inputs and put it on the output. The output of the Mux will be concatenated with '01' and will be forwarded to the second operand of the Addition Unit (nB).

In addition, upon writing to Reg1 the write signal that comes from the Write Decoder in order to write the second operand to Reg1 will also trigger the Pulse Generator block. This pulse will be sent to the start input of the Addition Unit and will cause it to start working and doing the computation. After three clock cycles the ready signal (rdy) of the Addition Unit will be activated which means that the floating point result is ready at the nR output. The main result nR(0-31) will be sent to Data2 input of Select Inputs block of Reg2. The ready signal will be sent to both Select Inputs blocks. This will cause the result to be written to Reg2 and 0x1 to Reg3.

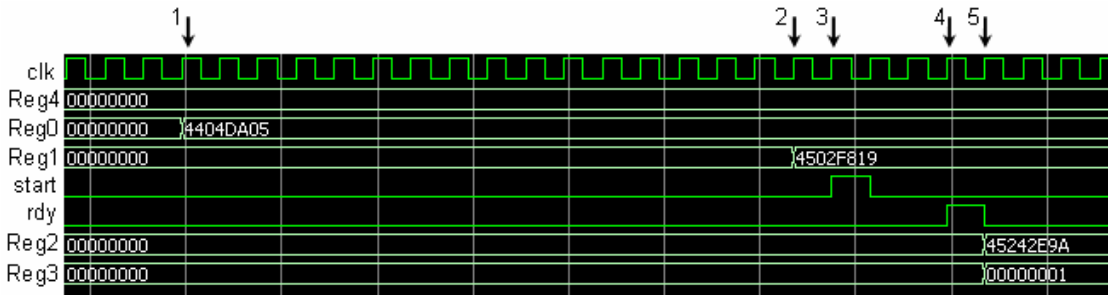

Figure 4.13 SFP Addition timing diagram

Figure 4.13 shows a timing diagram for the operation of this component. The two operands are written to the first two registers at time instants 1 and 2. The value

0x0 is written to Reg4 to select addition operation. Then the calculation of the result is started at 3 by asserting the start signal for one clock cycle, and after three clock cycles the rdy signal is activated at 4 which means that the result is ready. As a result, the result is written to Reg2 and 0x00000001 is written to Reg3 at 5. The values of the registers are expressed in IEEE 754 single precision format.

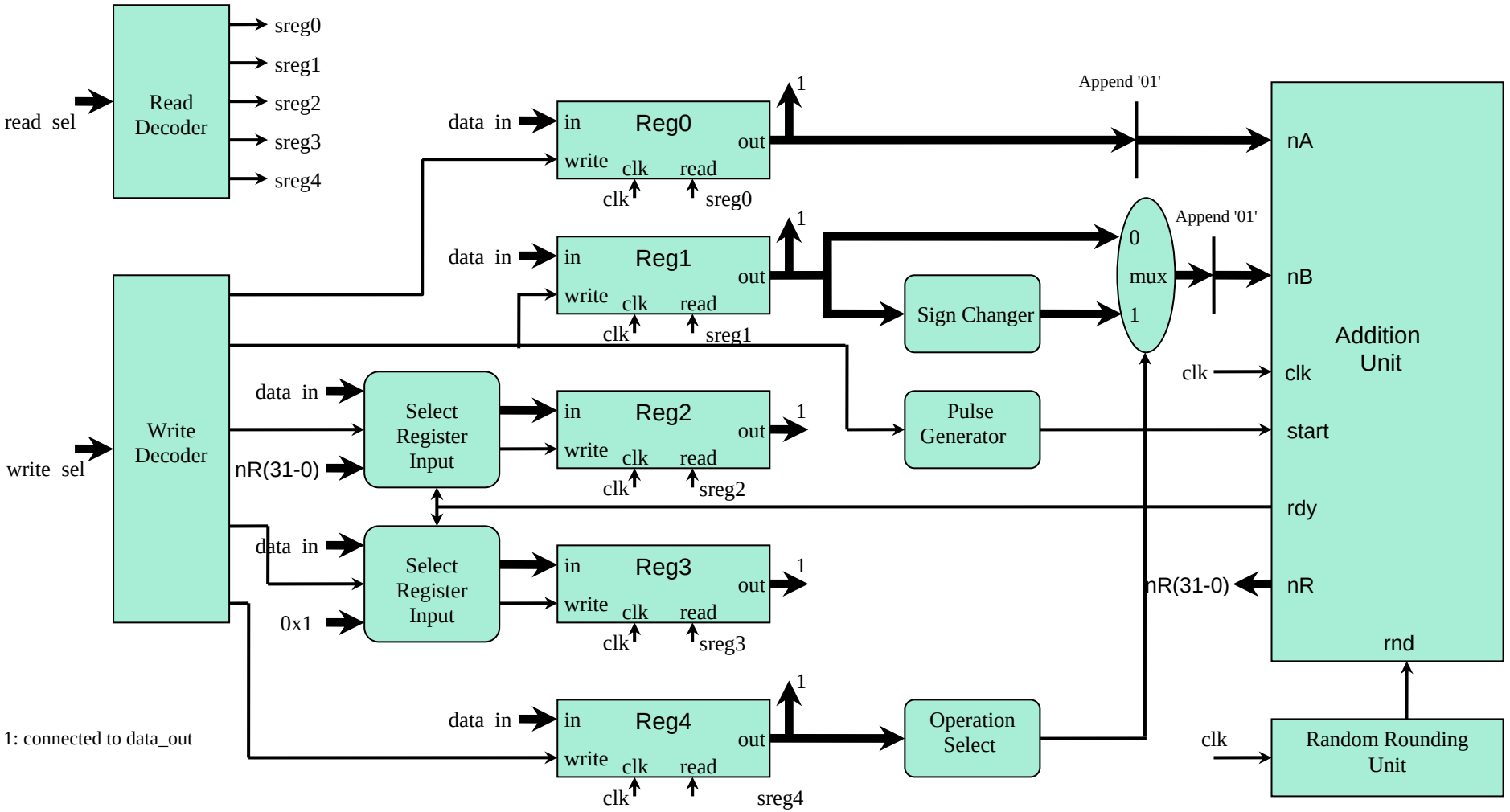

Figure 4.14: SFP Addition/Subtraction Component block diagram

4.3.2 SFP Multiplication Component

This component is responsible for executing floating point multiplication operation depending on a randomly selected rounding mode. The elements that form this component are shown in table 4.6.

Four registers are necessary to fulfill the functionality of this component. Table 4.8 shows the data that can be stored in each register.

Register	**Data**
Reg0	First operand
Reg1	Second operand
Reg2	The result of the floating point operation
Reg3	Set to 0x1 When the result is ready

Table 4.8: SFP Multiplication registers

Figure 4.16 shows a detailed block diagram for this component. The operation of this detailed data path can be explained as follow:

Reg0 and Reg1 should be written with operand1 and operand respectively. The value of Reg0 will be concatenated with '01' and will be forwarded to the first operand of the Multiplication Unit (nA). The value of Reg1 will be concatenated with '01' and will be forwarded to the second operand of the Multiplication Unit (nB).

In addition, upon writing to Reg1 the write signal, which comes from the Write Decoder in order to write the second operand to Reg1, will also trigger the Pulse Generator block. This pulse will be sent to the start input of the Multiplication Unit and will cause it to start working and doing the computation.

After four clock cycles the ready signal (rdy) of the Multiplication Unit will be activated which means that the floating point result is ready at the nR output. The main result nR(0-31) will be sent to Data2 input of Select Inputs block of Reg2. The ready signal will be sent to both Select Inputs blocks. This will cause the result to be written to Reg2 and 0x1 to Reg3.

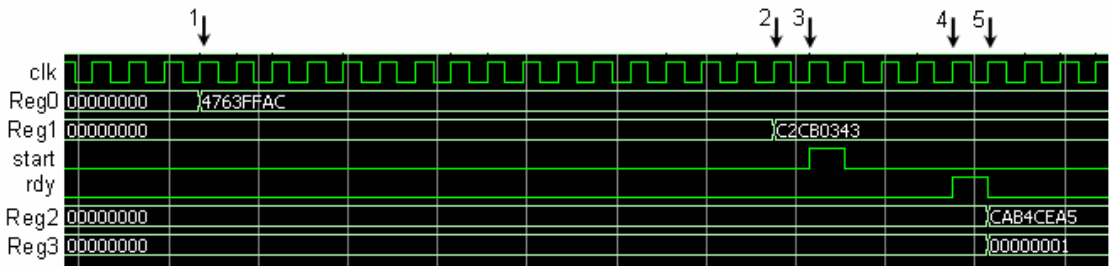

Figure 4.15: SFP Multiplication timing diagram

Figure 4.15 shows a timing diagram for the operation of this component which has similar timing behavior to the add/sub component. The two operands are written to the first two registers at time instants 1 and 2. Then the calculation of the result is started at 3 by asserting the start signal for one clock cycle, and after three clock cycles the rdy signal is activated at 4 which means that the result is ready. As a result, the result is written to Reg2 and 0x00000001 is written to Reg3 at 5. The values of the registers are expressed in IEEE 754 single precision format.

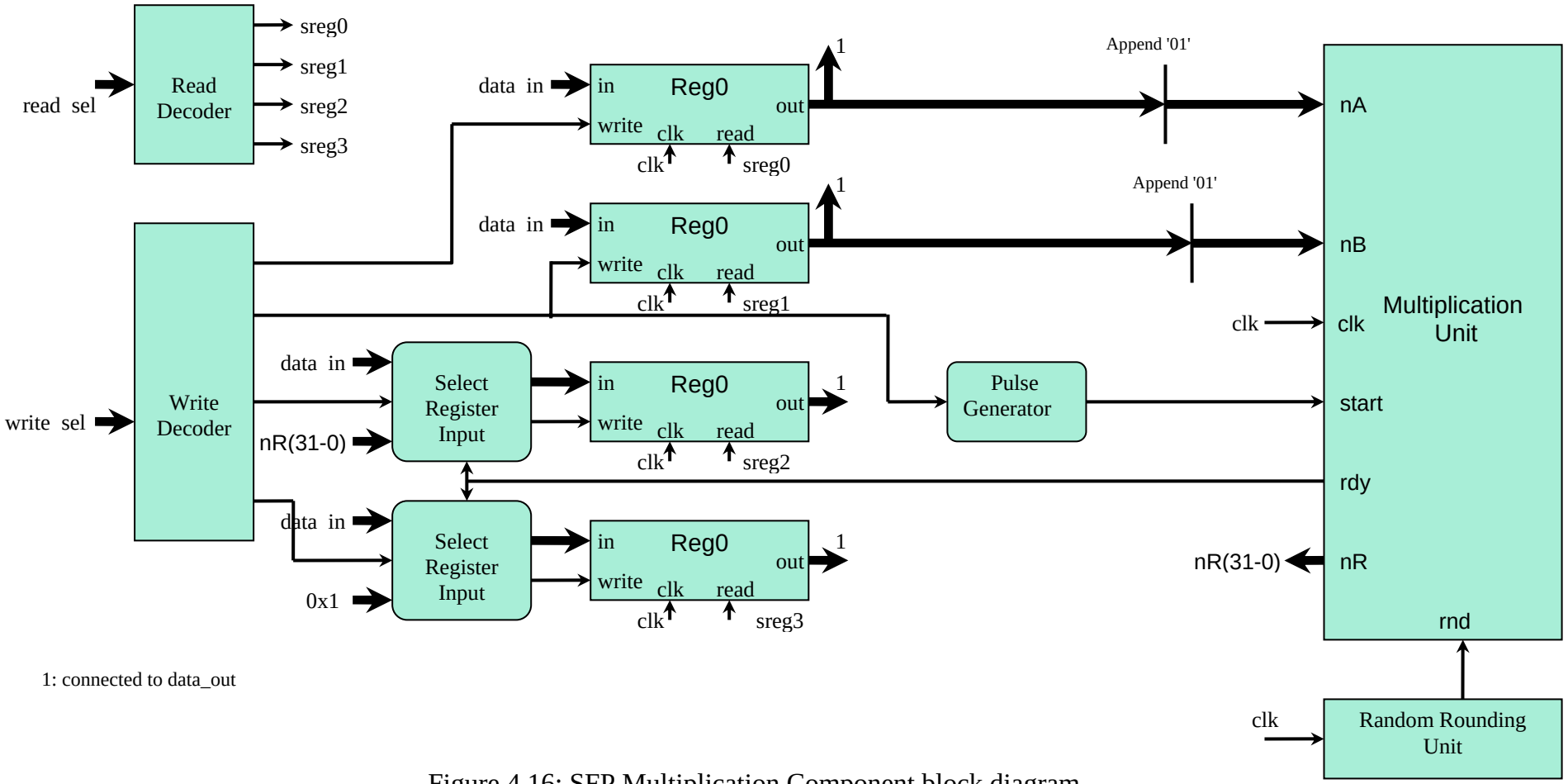

Figure 4.16: SFP Multiplication Component block diagram

4.3.3 SFP Division Component

This component is responsible for executing floating point division operation depending on a randomly selected rounding mode. The elements that form this component are shown in table 4.6.

Four registers are necessary to fulfill the functionality of this component. Table 4.9 shows the data that can be stored in each register.

Register	Data
Reg0	First operand
Reg1	Second operand
Reg2	The result of the floating point operation
Reg3	Set to 0x1 When the result is ready

Table 4.9: SFP Division registers

Figure 4.18 shows a detailed block diagram for this component. The operation of this detailed data path can be explained as follow:

Reg0 and Reg1 should be written with operand1 and operand respectively. The value of Reg0 will be concatenated with '01' and will be forwarded to the first operand of the Division Unit (nA). The value of Reg1 will be concatenated with '01' and will be forwarded to the second operand of the Division Unit (nB).

In addition, upon writing to Reg1 the write signal, which comes from the Write Decoder in order to write the second operand to Reg1, will also trigger the Ready Generator block.

The Ready Generator will start counting and after fifteen clock cycles the output ready signal of the Ready Generator will be activated which means that the floating point result is ready at the nR output. The main result nR(0-31) will be sent to Data2 input of Select Inputs block of Reg2. The ready signal will be sent to both Select Inputs blocks. This will cause the result to be written to Reg2 and 0x1 to Reg3.

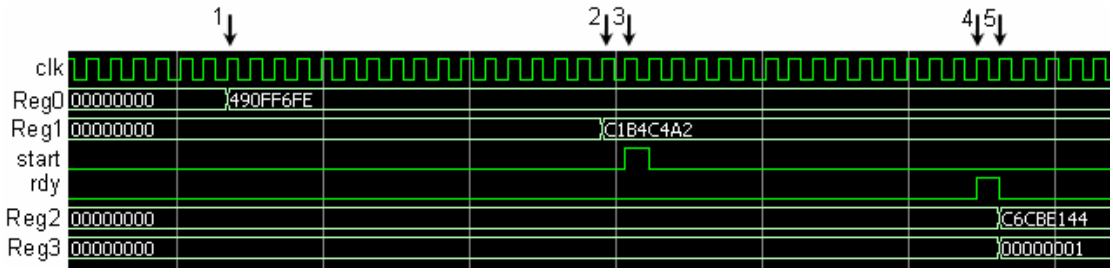

Figure 4.17 SFP Division timing diagram

Figure 4.17 shows a timing diagram for the operation of this component. The two operands are written to the first two registers at time instants 1 and 2. Then the calculation of the result is started at 3 by asserting the start signal for one clock cycle, and after fifteen clock cycles the rdy signal is activated at 4 which means that the result is ready. As a result, the result is written to Reg2 and 0x00000001 is written to Reg3 at 5. The values of the registers are expressed in IEEE 754 single precision format.

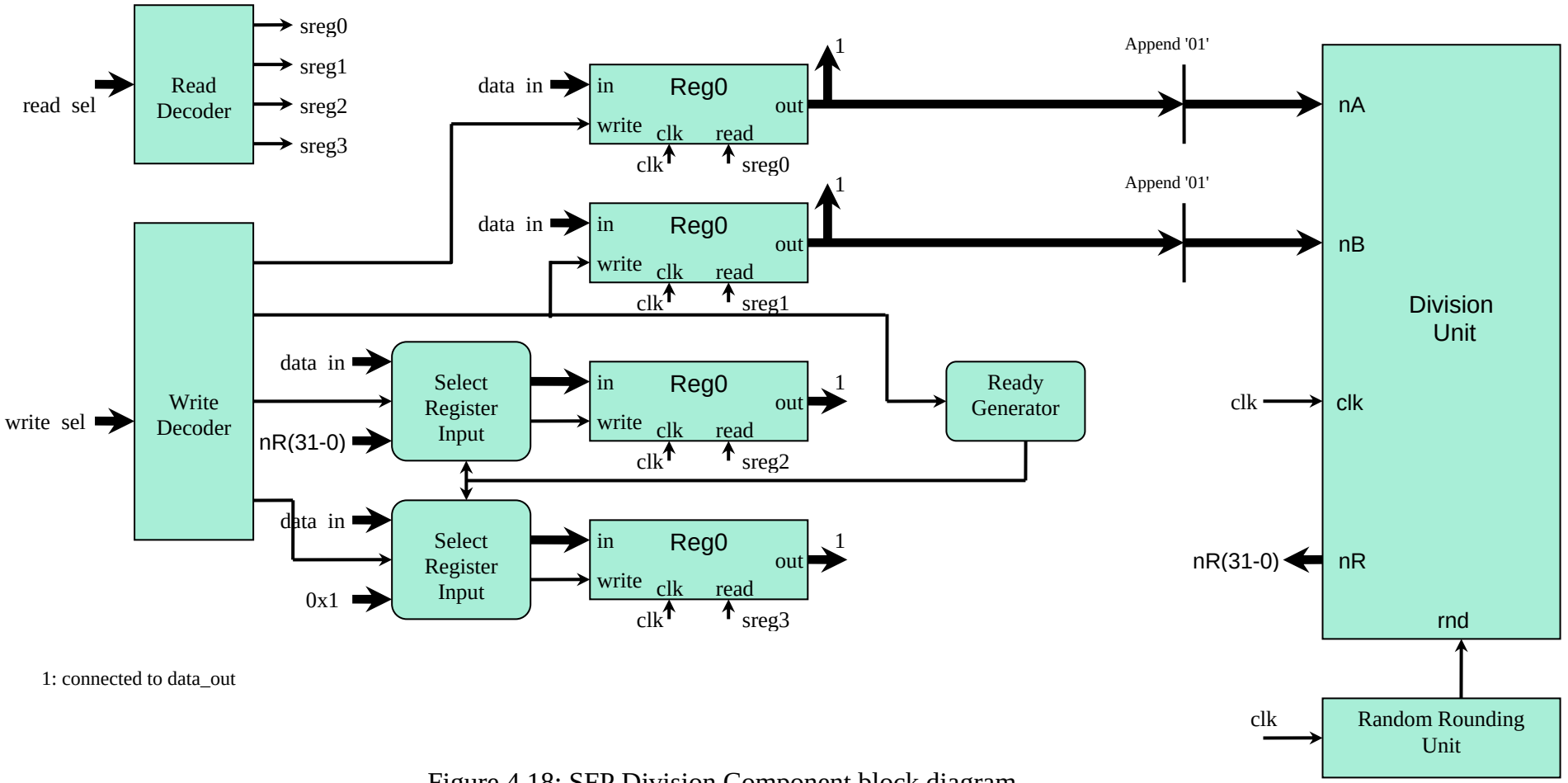

Figure 4.18: SFP Division Component block diagram

4.4 How To Use The Components

All three main components are connected to the MicroBlaze processor via the PLB bus. Each of them has a unique address and can be accessed independently. They can be used to execute floating point operations by writing the operands to the input registers and reading the result from the output register depending on a condition that the result is ready.

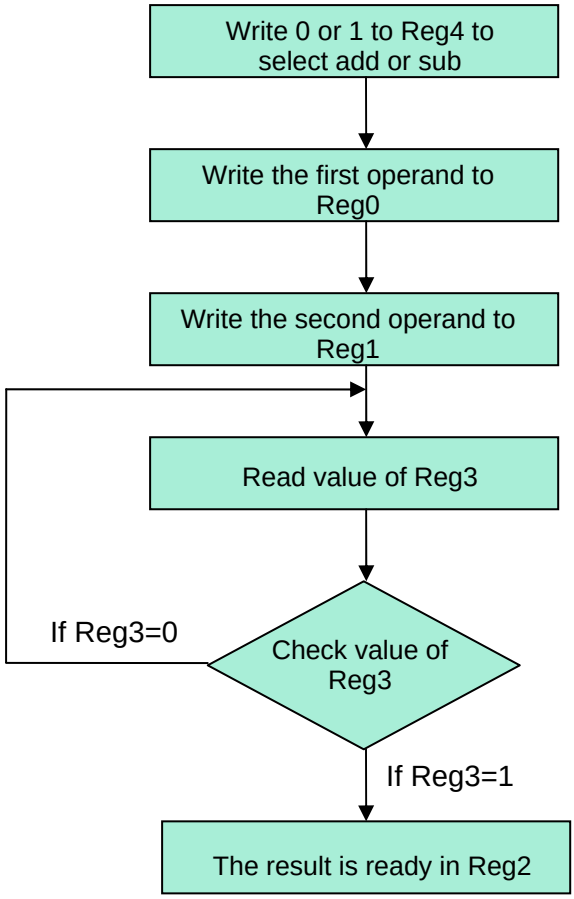

Figure 4.19: A flow diagram of how to use the SFP Addition/Subtraction Component

Figure 4.19 shows a flow diagram of how to use SFP Addition/Subtraction Component. First, Reg4 should be written by 0x0 or 0x1 in order to select addition or subtraction operations respectively. Then the operands are written to Reg0 and Reg1. The component will start computing the result and when it finishes it will write the result to Reg2 and also write 0x1 to Reg3 to indicate that the result is

ready. So the processor will keep checking this register until the value is set to 0x1 then it can read the result from Reg2.

A "C" code that explains how to use this addition/subtraction component can be written as follow:

```
Xuint32 *customLogicPtr;

int a0 = 0x4404DA05, a1 = 0x4502F819;

//        531,40656         2095,5061

int a2 = 0x0, addsub=0x00000000, ready=0x0;

// Base address of the addition/subtraction component

customLogicPtr = (Xuint32 *)XPAR_CUSTOM_ADDSUB_0_BASEADDR;

*(customLogicPtr + 0x4) = addsub; // Write o to select add
*(customLogicPtr) = a0;           // First operand
*(customLogicPtr + 0x1) = a1;     // Second operand

// Wait until the result is ready
do{
      ready = *(customLogicPtr + 0x3);
}while(ready!=0x00000001);

// read the result from Reg2
a2 = *(customLogicPtr + 0x2);
```

Figure 4.20 shows a flow diagram of how to use SFP Multiplication and Division Components. First, the operands should be written to Reg0 and Reg1. The component will start computing the result and when it finishes it will write the result to Reg2 and also write 0x1 to Reg3 to indicate that the result is ready. So the processor will keep checking this register until the value is set to 0x1 then it can read the result from Reg2.

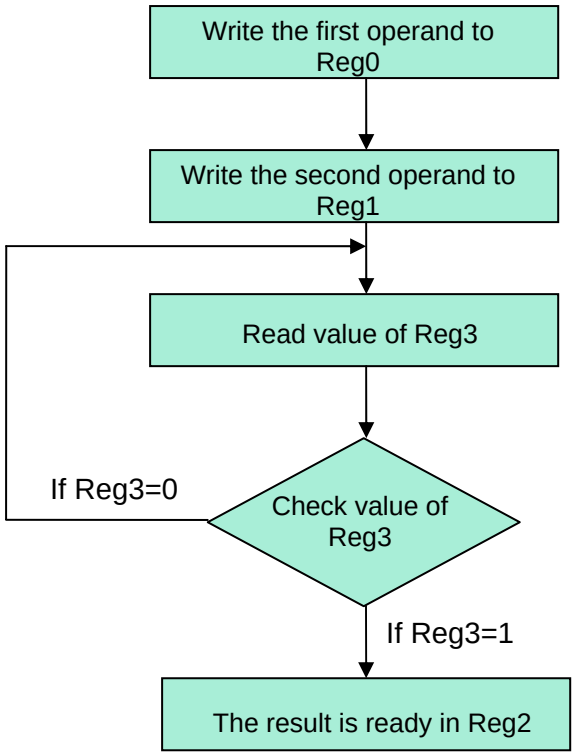

Figure 4.20: A flow diagram of how to use SFP Multiplication and Division Components

A "C" code that explains how to use the multiplication component can be written as follow:

```
Xuint32 *customLogicPtr;
int a0 = 0x4763FFAC, a1 = 0xC2CB0343;
//        58367,672       -101,50637
int a2 = 0x0, ready=0x0;

// Base address of the multiplication component
customLogicPtr = (Xuint32 *)XPAR_CUSTOM_MUL_0_BASEADDR;
*(customLogicPtr) = a0;           // First operand
*(customLogicPtr + 0x1) = a1;     // Second operand

// Wait until the result is ready
do{
     ready = *(customLogicPtr + 0x3);
}while(ready!=0x00000001);

// read the result from Reg2
a2 = *(customLogicPtr + 0x2);
```

A "C" code that explains how to use the division component can be written as follow:

```
Xuint32 *customLogicPtr;

int a0 = 0x490FF6FE, a1 = 0xC1B4C4A2;
//        589679,88       -22,596012

int a2 = 0x0, ready=0x0;

// Base address of the division component
customLogicPtr = (Xuint32 *)XPAR_CUSTOM_MUL_0_BASEADDR;

*(customLogicPtr) = a0;           // First operand
*(customLogicPtr + 0x1) = a1;     // Second operand

// Wait until the result is ready
do{
      ready = *(customLogicPtr + 0x3);
}while(ready!=0x00000001);

// read the result from Reg2
a2 = *(customLogicPtr + 0x2);
```

4.5 Synthesis Details

The system has been synthesized using the Vertix-5 FPGA ML506. The synthesis summary is presented in table 4.9 which shows the number of flip flops, lookup tables (LUTs) and block Random Access Memories (RAMs) used in the whole system and also in addition/subtraction, multiplication and division components. The device utilization summary for the registers, LUTs and Digital Signal Processing (DSP) blocks is presented in table 4.10. The Maximum frequency for the system is 117.041MHz.

Report	Flip Flops Used	LUTs Used	BRAMS Used
system	5599	5765	16
custom_addsub_0_wrapper	873	904	-
custom_mul_0_wrapper	804	925	-
custom_div_0_wrapper	1640	1333	-

Table 4.10: XPS synthesis summary

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	5267	32640	16%
Number of Slice LUTs	5439	32640	16%
Number of DSP48Es	8	288	2%

Table 4.11: Device utilization summary

4.6 Design Verification

Verification is normally done to show the correctness and reliability of the system. The first simple part of the verification is to examine each SFP component, one operation is executed on each of them by applying two operands and checking if the result of the computation is correct or not. This process has repeated many times on each component.

Table 4.12, table 4.13, table 4.14 and table 4.15 show the results of the first part of the verification. In each table, the IEEE single-precision value and the decimal value of the operands and the result of each operation are shown. In addition, the decimal double value calculated on the CPU is also shown.

Figure 4.21, figure 4.22, figure 4.23 and figure 24 show the timing diagrams for the addition, subtraction, multiplication, and division SFP operations respectively.

Addition Operation	IEEE single-precision value	Decimal value
First operand	0x4404DA05	531.40656
Second operand	0x4502F819	2095.5061
Result	0x45242E9A	2626.9126
Decimal double value calculated on the CPU: 2626.912660		

Table 4.12: Addition operation results

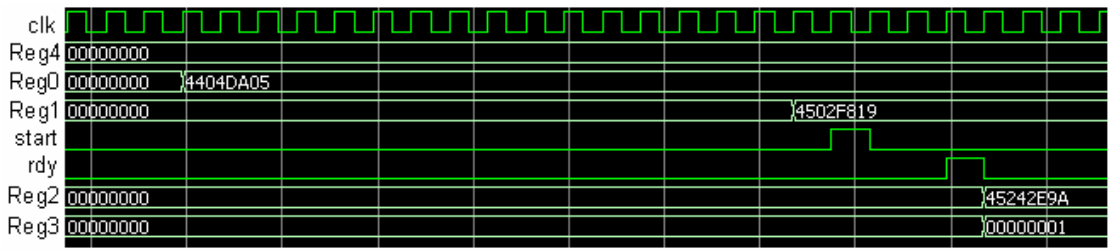

Figure 4.21: SFP Addition output

Subtraction Operation	IEEE single-precision value	Decimal value
First operand	0x4006C2E7	2.1056459
Second operand	0x3F9960DA	1.1982682
Result	0x3F6849E8	0.90737772
Decimal double value calculated on the CPU: 0.907378		

Table 4.13: Subtraction operation results

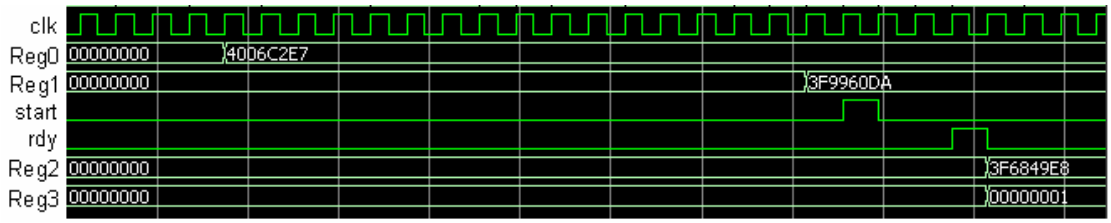

Figure 4.22: SFP Subtraction output

Multiplication Operation	**IEEE single-precision value**	**Decimal value**
First operand	0x4763FFAC	58367.672
Second operand	0xC2CB0343	-101.50637
Result	0xCAB4CEA5	-5924690.5
Decimal double value calculated on the CPU: -5924690.510071		

Table 4.14: Multiplication operation results

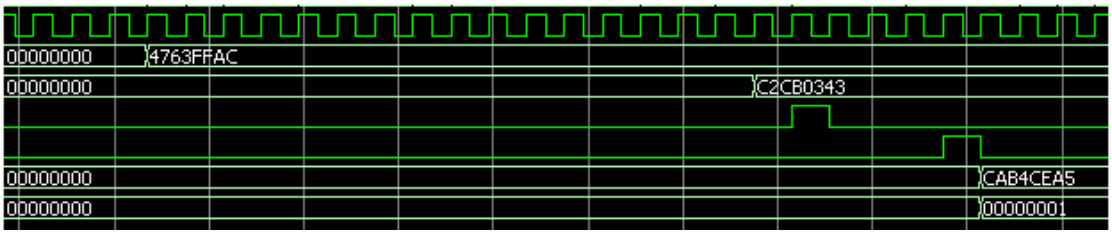

Figure 4.23: SFP Multiplication output

Division Operation	IEEE single-precision value	Decimal value
First operand	0x490FF6FE	589679.88
Second operand	0xC1B4C4A2	-22.596012
Result	0xC6CBE144	-26096.633
Decimal double value calculated on the CPU: -26096.635105		

Table 4.15: Division operation results

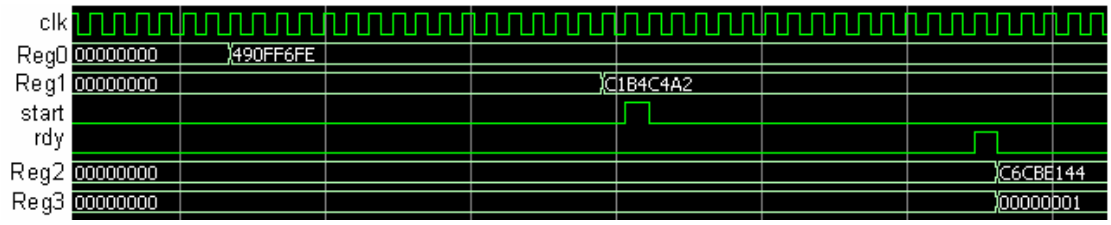

Figure 4.24: SFP Division output

Moreover, table 4.16 shows the operands and the result of the following equation in order to show that all components are working together correctly:

$$R = (op1 + op2) / (op1 * op2)$$

Division Operation	IEEE single-precision value	Decimal value
Op1	0x4763FFAC	58367.672
Op2	0xC2CB0343	-101.50637
Op3	0x4404DA05	531.40656
Op4	0x4502F819	2095.5061
Result	0x3D5651A7	0.052323963
Decimal double value calculated on the CPU: 0.052324		

Table 4.16: Working together results

The second part of the verification is done by executing the inner vector multiplication algorithm that takes two equal-length sequences of numbers (vectors) and returns a single number obtained by multiplying corresponding entries and adding up those products.

In this experiment, the two vectors were generated randomly from the range [-100000, 100000] and each one has 64 elements. The DSA method is used to estimate the accuracy of the result by providing the number of the exact significant digits.

In the DSA method, three results should be provided by running the code three times. Table 4.17 shows the IEEE single-precision value and the decimal value of these three results. In addition, the decimal double result calculated on the CPU is also shown.

Vector Multiplication	IEEE single-precision value	Decimal value
First run	0xCE1559E8	-626424320
Second run	0xCE155A00	-626425860
Third run	0xCE155A10	-626426880
Decimal double value calculated on the CPU: -626425991.635016		

Table 4.17: Inner vector multiplication results

The number of the exact significant digits is calculated as follow:

Using equation 2.9, the average value ($\overline{R}$) of the three results is -626425686.$\overline{6}$. σ is calculated using equation 2.11 which gives σ = 1288.772. Using equation 2.10, where τ_β = 4.4303, the number of the exact significant digits is equal to 5.3 digits.

If the double precision result calculated on the CPU is taken as a reference, the number of the exact significant digits can be calculated using the following equation:

$$C_{\overline{R}} = -\log_{10}\left|\frac{\overline{R}-r}{r}\right| \quad , r \text{ is the reference value} \tag{4.1}$$

In our case, r is the double value taken from table 4.17, and then the number of the exact significant digits is equal to 6.3 digits. If this value is compared to the one provided from the DSA, which 5.3 digits, then the results that are computed using this SFPU implementation are reliable.

Chapter 5

Conclusion and Future Works

In this work, the principle of the DSA has been reviewed, and a hardware design and implementation of a single-precision Stochastic Floating Point Unit (SFPU), which is used as the basis for the DSA method, has been investigated. The SFPU has been divided into three different Stochastic Floating Point (SFP) components to support addition/subtraction, multiplication and division floating point operations. Each of these components has a unique address and can be accessed independently. The SFPU was integrated to the MicroBlaze processor via the Processor Local Bus (PLB), and the system has been synthesized on a Virtex-5 FPGA. The achieved frequency was 117.041MHz.

The SFP components have been verified independently, and then the whole design has been verified by executing inner vector multiplication. Using the DSA method the number of exact significant digits of the result was 5.3. The double-precision value computed on the CPU was taken as reference and the number of exact significant digits of the result was 6.3. This shows that the results of this SFPU implementation are reliable.

Some improvements can be done to the design in order accelerate the computation by modifying the SFPU design to be pipelined and integrating the SFPU in the MicroBlaze.

Bibliography

[1] J. Vignes, A stochastic arithmetic for reliable scientific computation, Math. Comput. Simulation 35 (1993) 233-261.

[2] R.E. Moore, Interval Analysis (Prentice-Hall, Englewood Cliffs, NJ, 1966).

[3] J. Vignes, Review on stochastic approach to round-off error analysis, Math. Comput. Simulation 30 (1988) 481-491.

[4] J. Vignes, Discrete stochastic arithmetic for validating results of numerical software, Numerical Algorithms 37 (2004) 377-390.

[5] F. Jezequel, Dynamical strategies using discrete stochastic arithmetic for approximation methods, Laboratoire d'Informatique de Paris 6- CNRS UMR 7606 (2006).

[6] Xilinx, "MicroBlaze Processor Reference Guide". WWW page (accessed October 2010), http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[7] Xilinx, "LogiCORE IP Processor Local Bus (PLB) v4.6". WWW page (accessed October 2010), http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf

[8] Xilinx, "LogiCORE IP On-Chip Peripheral Bus V2.0 with OPB Arbiter". WWW page (accessed October 2010), http://www.xilinx.com/support/documentation/ip_documentation/opb_v20.pdf

[9] Xilinx, "LogiCORE IP Fast Simplex Link (FSL) V20 Bus". WWW page (accessed October 2010), http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf

[10] Xilinx, "EDK Concepts, Tools, and Techniques". WWW page (accessed October 2010), http://www.xilinx.com/support/documentation/sw_manuals/edk10_ctt.pdf

[11] libhdlfltp, FPGA coprocessor math library. WWW page (accessed November 2010), http://sourceforge.net/projects/libhdlfltp/