Digital Logic and Electronic Systems Design Company

Floating Point Verilog RTL Library User Guide

Pulse Logic

www.pulselogic.com.pl

e-mail: info@pulselogic.com.pl

Document version: 1.0

Document date: May 2009

Table of Contents

Introduction....3
Floating Point Adder/Subtractor – Float32Add....4
Floating Point Multiplier – Float32Mul....6
Floating Point Divider – Float32Div....8
Floating Point Square Root – Float32Sqrt....10
Know Problems and Solutions....12
License....13

Introduction

The IEEE has standardized the computer representation for binary floating-point numbers in IEEE 754 standard. It is the most commonly used representation in modern computing machines. The standard defines at least five different formats for floating point numbers. However, the following two seems to be most popular:

- Single precision. This is a binary format that occupies 32 bits (4 bytes)
- Double precision. This is a binary format that occupies 64 bits (8 bytes)

This library provides arithmetic operations for single precision floating point numbers represented using 32 bits.

Floating point numbers are represented in a binary form using the following fields:

- sign bit
- exponent field
- significant (mantissa) field

The following figure shows structure of single precision floating point number:

Figure 1. Single precision numbers.

Exact result of a floating point operation may need more mantissa bits to be stored.

In such case the result must be rounded before it can be stored in the above defined fields.

The IEEE standards defines several rounding schemes. However, rounding to nearest seams to be most popular and it is used in this library.

Floating point operations can run into problems like illegal operation, underflow or overflow.

Such exceptions are signaled using special codes like INF (infinity) or NAN (Not A Number).

The exceptions are fully supported in this library.

Result of an operation can be also to small to be represented correctly in IEEE floating point format.

In such case hardware changes representation and provides result as denormalized number.

Denormalized numbers are fully supported in this library. They are also accepted as valid operands to arithmetic operations.

Floating Point Adder/Subtractor – Float32Add

The Float32Add core is provided for free in the form of Verilog obfuscated code. The code is difficult to read because of removed text formatting and identifiers replaced with automatically generated strings. However, the code is fully functional and should work correctly in any simulation and synthesis tool. You can simulate it or synthesize to you FPGA or ASIC technology. Modifications of the code are extremely difficult. In case of any bugs please send a report to: info@pulselogic.com.pl

Features:

- Technology independent Verilog RTL design (obfuscated code)
- Synchronous design optimized for area and performance
- IEEE 754 single precision compliant
- Exceptions supported
- Denormal numbers supported including denormal arguments
- Rounding to nearest supported
- Output latency: 12 clock cycles

The following picture shows ports of Float32Add module ports:

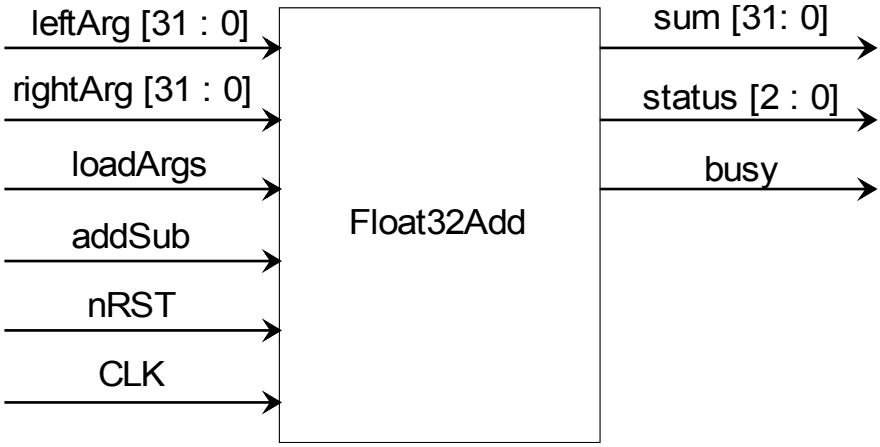

Figure 2. Float32Add ports

Port	Direction	Description
CLK	IN	Global clock. Rising edge active.
nRST	IN	Global reset. Active low.
leftArg [31 : 0]	IN	Left argument. Bit 31 - Sign, Bits 30:23 – Exponent, Bits 22:0 - Mantissa.
rightArg [31 : 0]	IN	Right argument. Bit 31 - Sign, Bits 30:23 - Exponent, Bits 22:0 - Mantissa.
addSub	IN	Operation selection. 1 - Addition, 0 – Subtraction.
loadArgs	IN	Arguments load strobe. Active high.
status [2 : 0]	OUT	Status output. Bit 2 - Not a Number, Bit 1 - Infinity, Bit 0 - Denormal.
busy	OUT	Busy output - high means performing calculations, low means result ready.
sum [31 : 0]	OUT	Result. Valid if busy bit is low. Status bits denote exceptions.

Table 1. Float32Add ports description

Both input arguments should be applied to *leftArg* and *rightArg* inputs along with *loadArgs* strobe on rising clock edge. The Float32Add module loads the arguments on the next rising clock edge and sets *busy* flag. High state on the *busy* output means that calculations are performed and result is not ready. Result is available to read when busy is low. At the same time you can load new arguments. Please, take a look at the waveform below:

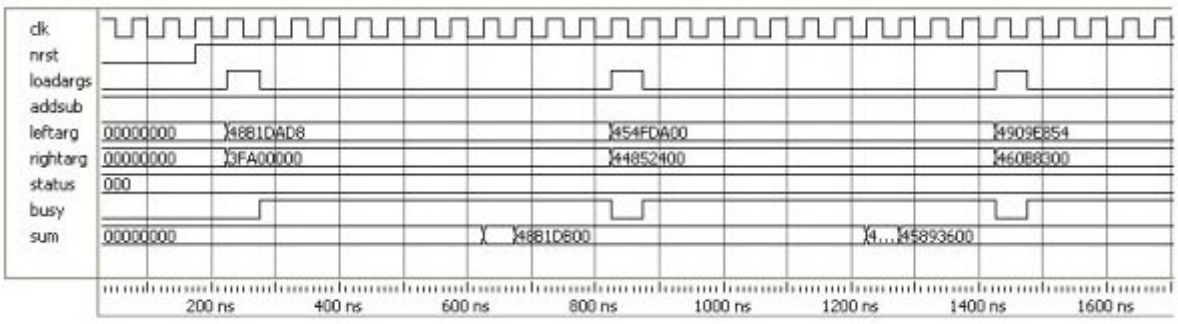

Figure 3. Float32Add simulation waveform

Floating Point Multiplier – Float32Mul

The Float32Mul core is provided for free in the form of Verilog obfuscated code. The code is difficult to read because of removed text formatting and identifiers replaced with automatically generated strings. However, the code is fully functional and should work correctly in any simulation and synthesis tool. You can simulate it or synthesize to you FPGA or ASIC technology. Modifications of the code are extremely difficult. In case of any bugs please send a report to: info@pulselogic.com.pl

Features:

- Technology independent Verilog RTL design (obfuscated code)
- Synchronous design optimized for area and performance
- IEEE 754 single precision compliant
- Exceptions supported
- Denormal numbers supported including denormal arguments
- Rounding to nearest supported
- Output latency: 10 clock cycles

The following picture shows ports of Float32Mul module ports:

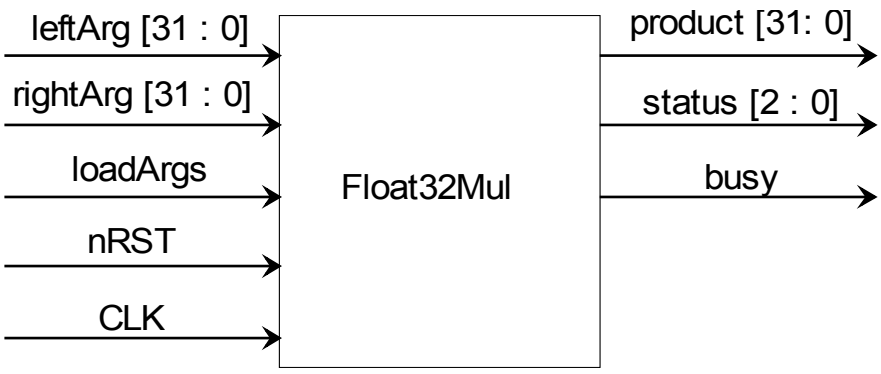

Figure 4. Float32Mul ports

Port	Direction	Description
CLK	IN	Global clock. Rising edge active.
nRST	IN	Global reset. Active low.
leftArg [31 : 0]	IN	Left argument. Bit 31 - Sign, Bits 30:23 – Exponent, Bits 22:0 - Mantissa.
rightArg [31 : 0]	IN	Right argument. Bit 31 - Sign, Bits 30:23 - Exponent, Bits 22:0 - Mantissa.
loadArgs	IN	Arguments load strobe. Active high.
status [2 : 0]	OUT	Status output. Bit 2 - Not a Number, Bit 1 - Infinity, Bit 0 - Denormal.
busy	OUT	Busy output - high means performing calculations, low means result ready.
product [31 : 0]	OUT	Result. Valid if busy bit is low. Status bits denote exceptions.

Table 2. Float32Mul ports description

Both input arguments should be applied to *leftArg* and *rightArg* inputs along with *loadArgs* strobe on rising clock edge. The Float32Mul module loads the arguments on the next rising clock edge and sets *busy* flag. High state on *busy* output means that calculations are performed and result is not ready. Result is available to read when busy is low. At the same time you can load new arguments. Please, take a look at the waveform below:

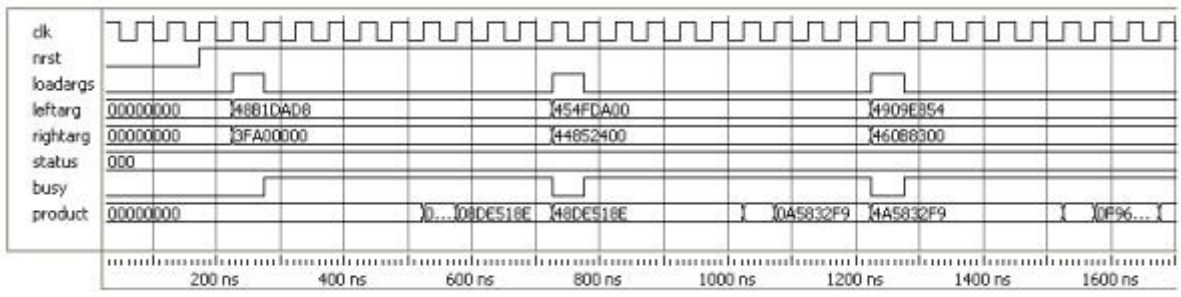

Figure 5. Float32Mul simulation waveform

Floating Point Divider – Float32Div

The Float32Div core is provided for free in the form of Verilog obfuscated code. The code is difficult to read because of removed text formatting and identifiers replaced with automatically generated strings. However, the code is fully functional and should work correctly in any simulation and synthesis tool. You can simulate it or synthesize to you FPGA or ASIC technology. Modifications of the code are extremely difficult. In case of any bugs please send a report to: info@pulselogic.com.pl

Features:

- Technology independent Verilog RTL design (obfuscated code)
- Synchronous design optimized for area and performance
- IEEE 754 single precision compliant
- Exceptions supported
- Denormal numbers supported including denormal arguments
- Rounding to nearest supported
- Output latency: 67 clock cycles

The following picture shows ports of Float32Div module ports:

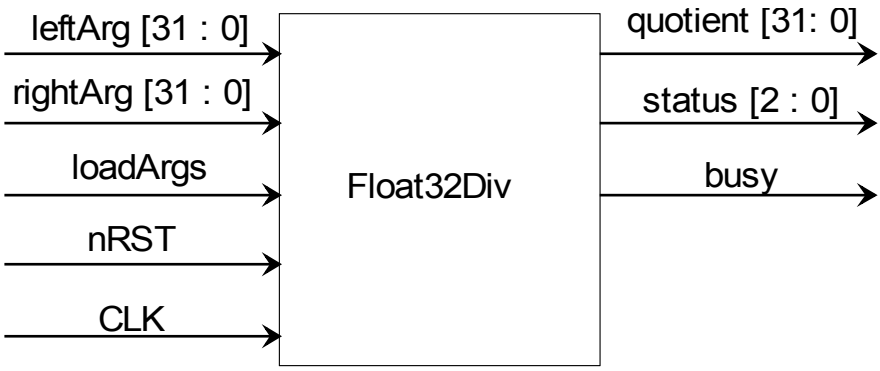

Figure 6. Float32Div ports

Port	Direction	Description
CLK	IN	Global clock. Rising edge active.
nRST	IN	Global reset. Active low.
leftArg [31 : 0]	IN	Left argument. Bit 31 - Sign, Bits 30:23 – Exponent, Bits 22:0 - Mantissa.
rightArg [31 : 0]	IN	Right argument. Bit 31 - Sign, Bits 30:23 - Exponent, Bits 22:0 - Mantissa.
loadArgs	IN	Arguments load strobe. Active high.
status [2 : 0]	OUT	Status output. Bit 2 - Not a Number, Bit 1 - Infinity, Bit 0 - Denormal.
busy	OUT	Busy output - high means performing calculations, low means result ready.
quotient [31 : 0]	OUT	Result. Valid if busy bit is low. Status bits denote exceptions.

Table 3. Float32Div ports description

Both input arguments should be applied to *leftArg* and *rightArg* inputs along with *loadArgs* strobe on rising clock edge. The Float32Div module loads the arguments on the next rising clock edge and sets *busy* flag. High state on *busy* output means that calculations are performed and result is not ready. Result is available to read when busy is low. At the same time you can load new arguments. Please, take a look at the waveform below:

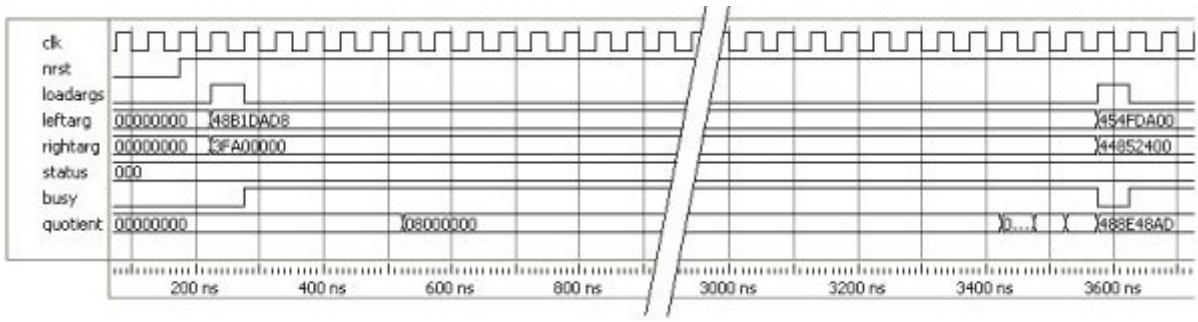

Figure 7. Float32Div simulation waveform

Floating Point Square Root – Float32Sqrt

The Float32Sqrt core is provided for free in the form of Verilog obfuscated code. The code is difficult to read because of removed text formatting and identifiers replaced with automatically generated strings. However, the code is fully functional and should work correctly in any simulation and synthesis tool. You can simulate it or synthesize to you FPGA or ASIC technology. Modifications of the code are extremely difficult. In case of any bugs please send a report to: info@pulselogic.com.pl

Features:

- Technology independent Verilog RTL design (obfuscated code)
- Synchronous design optimized for area and performance
- IEEE 754 single precision compliant
- Exceptions supported
- Denormal numbers supported including denormal arguments
- Rounding to nearest supported
- Output latency:
  - 894 clock cycles – version with radix two division
  - 257 clock cycles – version with high radix division

The square root core is provided in two versions. The two version differ in the algorithm used for integers division. The first version uses standard radix two division algorithm. It is resources efficient core but needs significant number of clock cycles to calculate result. The second version uses parallelized division algorithm. It needs less clock cycles but utilizes significantly more hardware resources.

The following picture shows ports of Float32Sqrt module ports:

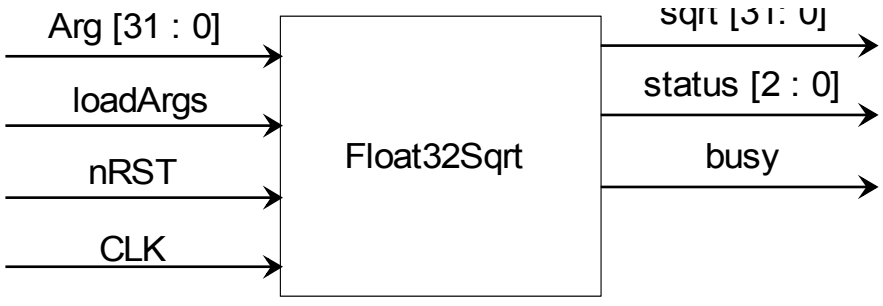

Figure 8. Float32Sqrt ports

Port	Direction	Description
CLK	IN	Global clock. Rising edge active.
nRST	IN	Global reset. Active low.
arg [31 : 0]	IN	Input argument. Bit 31 - Sign, Bits 30:23 – Exponent, Bits 22:0 - Mantissa.
loadArgs	IN	Argument load strobe. Active high.
status [2 : 0]	OUT	Status output. Bit 2 - Not a Number, Bit 1 - Infinity, Bit 0 - Denormal.
busy	OUT	Busy output - high means performing calculations, low means result ready.
sqrt [31 : 0]	OUT	Result. Valid if busy bit is low. Status bits denote exceptions.

Table 4. Float32Sqrt ports description

Input argument should be applied to a*rg* input along with *loadArgs* strobe on rising clock edge. The Float32Sqrt module loads the argument on the next rising clock edge and sets *busy* flag. High state on *busy* output means that calculations are performed and result is not ready. Result is available to read when busy is low. At the same time you can load new argument. Please, take a look at the waveform below:

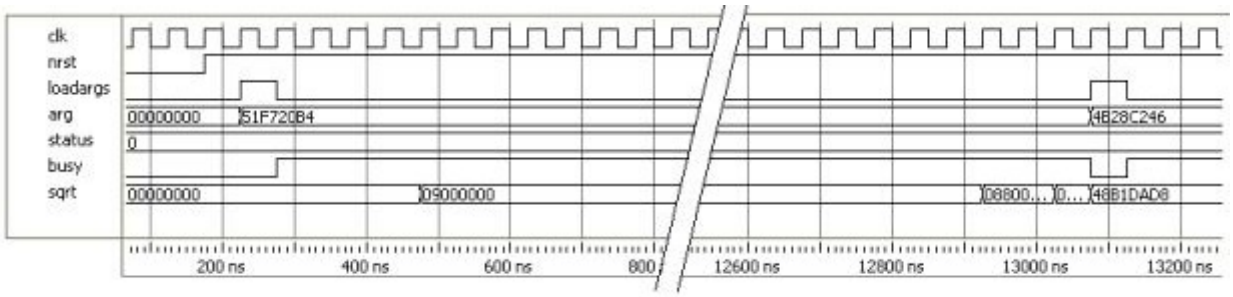

Figure 9. Float32Sqrt simulation waveform

Know Problems and Solutions

1. **Differences detected while comparing Verilog simulation results against C/C++ program executed on PC machine.**

Solution:

C/C++ compilers provide *float* type compatible with IEEE 754 single precision numbers. Variables of the *float* type are 32 bits wide as defined in IEEE standard. However, PC machines perform all calculations with full precision numbers, which are 64 bits wide. The 64-bits results are rounded to 32-bits once the operation is finished. Results of an arithmetic operation performed on 64-bits and rounded to 32 bits can be different than result of an operation performed on 32 bits. In case of single addition or multiplication mantissa may differ +/- 1. If you have more operations performed serially the difference may be larger. You should perform calculations on 32-bits to get results matching Verilog simulation. You can use VHDL behavioral models provided in IEEE library in the floating point package.

2. Result of a single addition or multiplication differs much more than +/-1 while comparing Verilog simulation results against C/C++.

Solution:

Check if the difference is encountered with NaN (Not a Number). IEEE standard defines NaN as a number with all bits set to one in the exponent and non zero mantissa. The contents of the mantissa is not important but must be different than zero. Usually, mantissa MSB bit is set to one while signaling NaN. Remaining bits of mantissa are left untouched and may contain values from previous calculations. Check always status flags and do not compare NaNs if encountered.

License