HCTL Open Int. J. of Technology Innovations and Research
HCTL Open IJTIR, Volume 1, January 2013
e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Implementation of Custom Precision Floating Point Arithmetic on FPGAs

Raj Gaurav Mishra and Amit Kumar Shrivastava
rajgmishra@hctl.org and amitshrivastava@hctl.org

Abstract

Floating point arithmetic is a common requirement in signal processing, image processing and real time data acquisition & processing algorithms. Implementation of such algorithms on FPGA requires an efficient implementation of floating point arithmetic core as an initial process. We have presented an empirical result of the implementation of custom-precision floating point numbers on an FPGA processor using the rules of IEEE standards defined for single and double precision floating point numbers. Floating point operations are difficult to implement on FPGAs because of their complexity in calculations and their hardware utilization for such calculations. In this paper, we have described and evaluated the performance of custom-precision, pipelined, floating point arithmetic core for the conversion to and from signed binary numbers. Then, we have assessed the practical implications of using these algorithms on the Xilinx Spartan 3E FPGA boards.

HCTL Open Int. J. of Technology Innovations and Research
HCTL Open IJTIR, Volume 1, January 2013
e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Keywords

Field Programmable Gate Array (FPGA), floating point arithmetic, custom-precision floating point, floating point conversion.

Introduction

With the advent of sensor technology, it is now possible to measure and monitor a large number of parameters and to carefully use them in a number of fields such as medical, defence, commercial etc. for various applications. Real-time implementation of sensor based application requires a system which can read, store and process the sensor data using micro-controllers or FPGAs as processors. Figure 1 shown below, represents a real-time data acquisition system based on a FPGA processor. Such a system comprises of a single or multiple sensors, signal conditioning unit (filters and amplifiers) and analog to digital converters. The output of the analog to digital converter is generally connected to the input of the processor (FPGA device in our case) for further signal acquisition and processing.

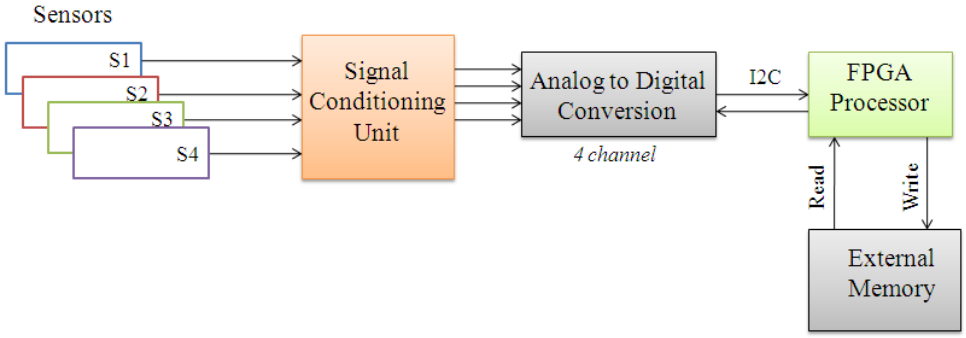

Figure 1: *Block diagram of a FPGA processor based real-time sensor data acquisition system.*

It is important for FPGA processor to store the real time sensor values to an external memory device for signal processing using custom algorithms. For example, analysing sound/speech in real time requires recording of sound signals using a microphone (sensor) using a high speed FPGA processor and then storing the resultant sensor values into a floating point format to maintain a specific accuracy and resolution. Floating point number system in comparison with binary number system have a better dynamic range and are better in

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

handling underflow and overflow situations during mathematical calculations (signal processing). In this way, when a sensor value is stored in floating point format provides a base for accurate signal processing.

In this paper, a pipelined implementation and hardware verification of custom precision floating point arithmetic on FPGA have been reported and discussed. Here, we assume that the reader is familiar with FPGA [1], its programming using Verilog HDL [2]-[3] and the single precision, double precision floating point standards [4] defined by IEEE. Comparatively, floating point number systems have a better dynamic range than a fixed point number system; also they are better in handling underflow and overflow situations during mathematical calculations however the speed and complexity issues rises when an implementation on FPGA processors comes into consideration. Research has been done to experiment various optimized implementations of IEEE single precision [7]-[10] and double precision [11]-[15] floating point arithmetic on FPGA. Algorithms for floating point implementation are complex in nature and with the number of bits used in single or double precision made them utilize a large area of the FPGA chip with a considerable processing time. Need of a custom precision floating point system arises when a real-time image or digital signal processing applications are to be implemented on an FPGA processor, where a requirement of high throughput in calculation and a balanced time-area-power implementation of the algorithm becomes an important requirement. In this way, an embedded designer can choose a suitable custom floating-point format depending upon the available FPGA space for the required embedded application.

Table 1 shows the basic comparison between the 17-bits custom precision, IEEE standards of single, double and quadruple precision floating point numbers.

This paper is organized as section 2 presents the custom precision floating point format in details. In section 3, we have described the algorithm and flowchart for the conversion of 12 bit signed binary number to 17 bits custom precision floating point number, the algorithm and flowchart for the conversion of 17 bits custom precision floating point number to a 12 bit signed binary number, along with their simulation results, synthesis summary and hardware verifications. Section 4 concludes this paper with the scope of future work which can be extended in various ways.

HCTL Open Int. J. of Technology Innovations and Research
HCTL Open IJTIR, Volume 1, January 2013
e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Table 1: *Different floating point formats.*

	Custom Precision	Single Precision	Double Precision	Quadruple Precision
Word Length	17 bits	32 bits	64 bits	128 bits
Mantissa	10 bits	23 bits	52 bits	112 bits
Exponent	6 bits	8 bits	11 bits	15 bits
Sign	1 bit	1 bit	1 bit	1 bit
Bias	2^{6-1}-1=31	2^{8-1}-1=127	2^{11-1}-1=1023	2^{15-1}-1=16383
Range	About $4.3x10^{9}$=(2^{32})	About $3.5x10^{38}$=(2^{128})	About $1.8x10^{308}$=(2^{1024})	About $1.2x10^{4932}$=(2^{16384})

Custom Precision Floating Point Format

Floating-point systems were developed to provide high resolution over a large dynamic range. Floating-point systems can often provide a solution when fixed-point systems, with their limited dynamic range, fail. Floating-point systems, however, bring a speed and complexity penalty. Most microprocessor floating-point systems comply with the published single- or double-precision IEEE floating-point standard; while in FPGA-based systems often employ custom formats.

A standard floating-point word consists of a sign-bit S, exponent E, and an unsigned (fractional) normalized mantissa M, arranged as shown in the figure 2.

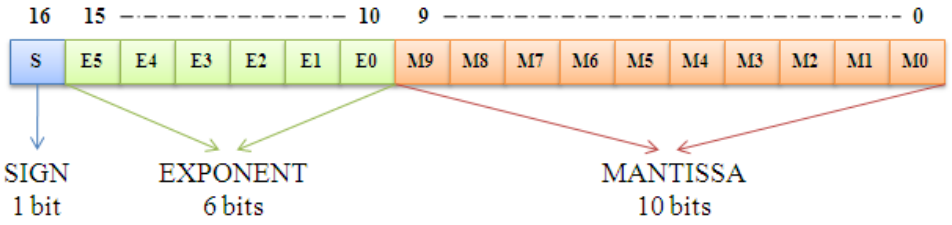

Figure 2: *17 bits Custom Precision Floating Point format.*

The minimum number custom precision floating point number (1,6,10) format can represent is:

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

$(\pm 0.00000000046566128730773925781 25)_{10}$ or $(\pm 0.00000000000000000000000000000001)_2$. Minimum number representation in custom precision floating point format is $(0000000000000000)_{1,6,10}$ for a positive number and $(1000000000000000)_{1,6,10}$ for a negative number.

The maximum number custom precision floating point number (1,6,10) format can represent is: $(\pm 4,29,49,67,296.00)_{10}$ or $(\pm 10000000000000000000000000000000)_2$. Maximum number representation in custom precision floating point format is $(0111111000000000)_{1,6,10}$ for a positive number and $(1111111000000000)_{1,6,10}$ for a negative number.

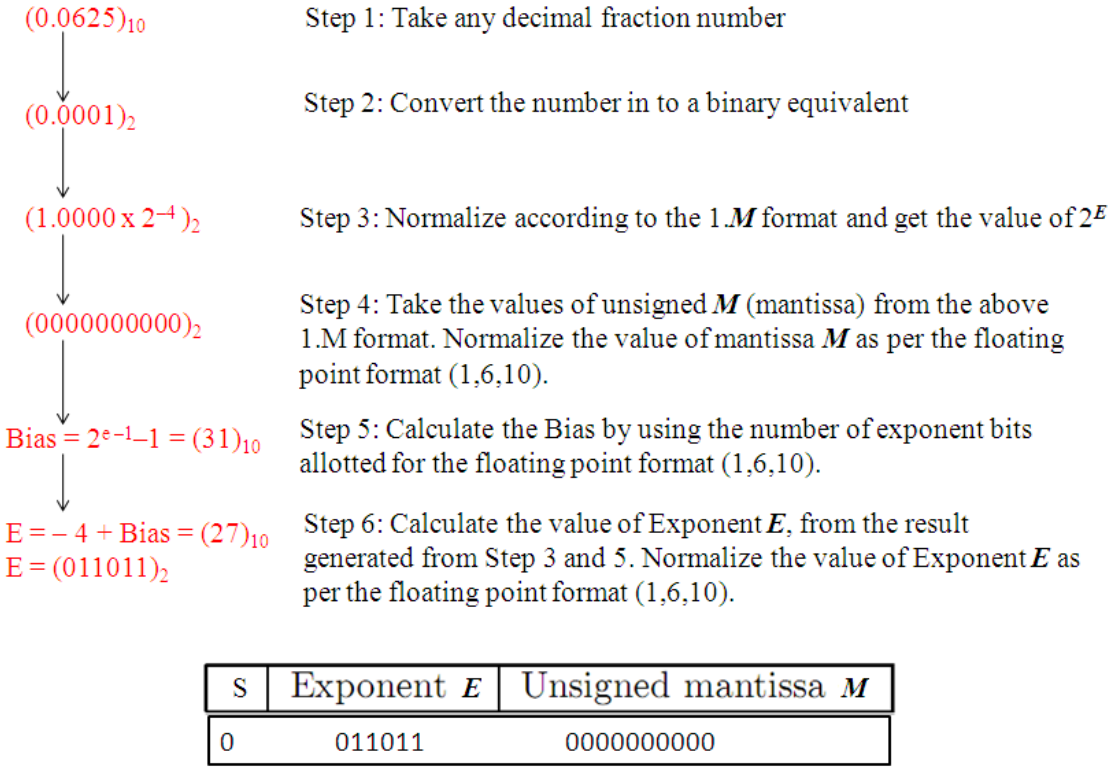

Figure 3: *Method of converting a fixed point decimal number in to a custom precision floating point number (1,6,10) format [5]-[6].*

HCTL Open Int. J. of Technology Innovations and Research
HCTL Open IJTIR, Volume 1, January 2013
e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Table 2: *Some Examples values of 17-bit Custom Precision Floating-Point Format.*

S.No.	17-bit custom precision floating-point format (1,6,10)	Equivalent decimal values
1	0 000000 0000000000	Represents a minimum number (+0)
2	1 000000 0000000000	Represents a minimum number (−0)
3	0 011111 0000000000	+1.0
4	1 011111 0000000000	−1.0
5	0 111111 0000000000	Represents a maximum number ($+\infty$)
6	1 111111 0000000000	Represents a maximum number ($-\infty$)

Floating Point Operations on FPGA

Signed Binary to Custom Precision Floating Point Conversion

Considering the system defined in the figure 1, an embedded designer can choose 8-bit, 12-bit or 16-bit analog to digital converters to adjust the required resolution of the sensor value for an application. Texas Instruments ADS7828 [16] or any other similar 12-bit ADC is more suitable for the algorithm developed and presented in the following section.

Algorithm 1 describes the step-wise approach of programming an FPGA for the conversion of a 12-bit signed binary number in to a 17-bit custom-precision floating point number.

The flow diagram of the algorithm 1 is shown in the figure 4. Table 3 shows the synthesis summary of hardware utilization and speed for the algorithm 1 on different FPGA processors. Figure 5 shows the simulation results for the algorithm 1 implemented using verilog HDL on Xilinx ISE Project Navigator Ver. 13.4 and ISE Simulator [17].

Custom Precision Floating Point to Signed Binary Conversion

Algorithm 2 defines the step-wise approach of programming an FPGA for the conversion of a 17-bit custom-precision floating point number in to a 12-bit signed binary number. The flow diagram of the algorithm 2 is shown in the

HCTL Open Int. J. of Technology Innovations and Research
HCTL Open IJTIR, Volume 1, January 2013
e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Algorithm 1 Converting a Signed Binary number (12 bits) in to a Custom-Precision (17 bits) Floating Point number (1,6,10) format.

Require: 12 bits signed binary number as input.
Ensure: 17 bit custom-precision floating point number (1,6,10) format as output.

1: Store the value of input to a temporary register R1 (size 12 bits).
2: Check for the sign-bit:
3: **if** Sign-bit is equal to 1 (Input is a negative number): **then** Take 2's complement of the values stored in R1 register (input value) and store the results to a temporary register R2 (size 12 bits).
4: **else if** Sign-bit is equal to 0 (Input is a positive number): **then** Store the value of temporary register R1 to temporary register R2 without any modifications.
5: **end if**
6: Scan all the bit values of register R2 starting from (MSB − 1) towards LSB and search for first HIGH (1) bit value.
7: Count of the total bits towards right side (towards LSB) from the first HIGH (1) bit found, and store this count to a temporary register R3 (size 4 bits).
8: Store all the bits towards right side (towards LSB) from the first HIGH (1) bit found, to a temporary register R4 (size 10 bits).
9: Calculate the addition of the count stored in temporary register R3 with the value of fixed bias* and store the results to a temporary register R5 (size 6 bits). This forms the exponent value. Calculation of Fixed Bias* = $2^{E-1} - 1 = 2^{6-1} - 1 = 31_{10}$ or 011111_2. (E is the number of bits allocated for exponent in the floating point format).
10: Normalize the value of register R4 (bit shifting towards MSB to fit the values completely in 10 bits format). Store the resultant value to a temporary register R6. This forms the mantissa value.
11: Store the sign bit from the input value (as stored in register R1), value of register R5 (exponent) and value of register R6 (mantissa) in a 17 bits custom (1,6,10) format to a temporary register R7 (size 17 bits).
12: Connect the temporary register R7 to the output.
13: 17 Bit custom-precision floating point number (1,6,10) format.

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Table 3: *Synthesis Summary for 12 bit Signed Binary Number to 17 bit Custom Precision Floating Point Conversion.*

FPGA Processor	Speed Grade	Number of Slices Used	Number of Slice Flip Flops Used	Number of 4 input LUTs Used	Number of bonded IOBs Used	Maximum Frequency
Spartan 3E XC3S500E	-5	71 out of 4656	70 out of 9312	134 out of 9312	30 out of 232	174.304 MHz
Spartan 3E XC3S1200E	-5	71 out of 8672	70 out of 17344	134 out of 17344	30 out of 250	174.304 MHz
Spartan 6 XC6SLX25	-3	77 out of 30064	138 out of 15032	162 out of 486	30 out of 226	212.770 MHz
Virtex 4 XC4VFX100	-12	83 out of 42176	70 out of 84352	157 out of 84352	30 out of 576	338.324 MHz
Virtex 5 XC5VFX100T	-3	70 out of 64000	94 out of 64000	113 out of 339	30 out of 680	394.120 MHz
Virtex 6 XC6VCX130T	-2	69 out of 160000	120 out of 80000	136 out of 408	30 out of 240	433.529 MHz

figure 8.

Table 4 shows the synthesis summary of hardware utilization and speed for the algorithm 2 on different FPGA processors. Figure 6 shows the simulation results for the algorithm 2 implemented using verilog HDL on Xilinx ISE Project Navigator Ver. 13.4 and ISE Simulator [17].

Algorithms 1 and 2 have been tested and verified on Digilent NEXYS 2 FPGA board [18] containing Spartan 3E [19] XC3S1200E FPGA processor as shown in figure 7. To verify the correctness of the algorithm different inputs were given through different combinations of on-board switches and output was received through the LEDs connected to the I/O pins of the FPGA processor.

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Algorithm 2 Converting a Custom-Precision (17 bits) Floating Point number (1,6,10) format in to a Signed Binary number (12 bits).

Require: 17 bit custom-precision floating point number (1,6,10) format as input.

Ensure: 12 bits signed binary number as output.

1: Store the MSB value of input to a temporary register R1 (size 1 bit). This is for the purpose of defining sign bit.
2: Store the 10 bits from the LSB towards MSB to a temporary register R2 (size 11 bits). This is to be used as mantissa for further calculations.
3: Assign the MSB bit of temporary register R2 as HIGH (1) to incorporate the hidden 1 bit.
4: Store the remaining 6 bits from the input to a temporary register R3 (size 6 bits). This is to be used as exponent for further calculations.
5: Calculation of exponent value (in order to normalize the mantissa): Values stored in register R3− Bias* = Value of exponent by which mantissa is to be normalized. Store this value to a temporary register R4 (size 8 bits). Calculation of Fixed Bias* $= 2^{E-1} - 1 = 2^{6-1} - 1 = 31_{10}$ or 011111_2. (E is the number of bits allocated for exponent in the floating point format).
6: Normalization of mantissa value to fit in 11 bits:
7: **if** the value stored in register R3 (exponent value) is equals to 0 (zero). **then** the value of mantissa will become 0 (zero).
8: **end if**
9: **if** the value stored in register R4 (exponent count) is equals to 0. **then** the value of mantissa is to be bit-shifted 10 times towards left (MSB) for normalization.
10: **end if**
11: **if** the value stored in register R4 (exponent value) is equals to 1. **then** the value of mantissa is to be bit-shifted 9 times towards left (MSB) for normalization.
12: **end if**
13: **if** the value stored in register R4 (exponent value) is equals to 2. **then** the value of mantissa is to be bit-shifted 8 times towards left (MSB) for normalization.
14: **end if**
15: **if** the value stored in register R4 (exponent count) is equals to 3. **then** the value of mantissa is to be bit-shifted 7 times towards left (MSB) for normalization.
16: **end if**
17: **if** the value stored in register R4 (exponent value) is equals to 4. **then** the value of mantissa is to be bit-shifted 6 times towards left (MSB) for normalization.
18: **end if**

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

19: **if** the value stored in register R4 (exponent value) is equals to 5. **then** the value of mantissa is to be bit-shifted 5 times towards left (MSB) for normalization.
20: **end if**
21: **if** the value stored in register R4 (exponent value) is equals to 6. **then** the value of mantissa is to be bit-shifted 4 times towards left (MSB) for normalization.
22: **end if**
23: **if** the value stored in register R4 (exponent value) is equals to 7. **then** the value of mantissa is to be bit-shifted 3 times towards left (MSB) for normalization.
24: **end if**
25: **if** the value stored in register R4 (exponent value) is equals to 8. **then** the value of mantissa is to be bit-shifted 2 times towards left (MSB) for normalization.
26: **end if**
27: **if** the value stored in register R4 (exponent value) is equals to 9. **then** the value of mantissa is to be bit-shifted 1 time towards left (MSB) for normalization.
28: **end if**
29: Store the resultant value of mantissa after suitable bit-shifting (normalization) to a temporary register R5 (size 11 bits).
30: Check for the sign-bit stored in the register R1:
31: **if** Sign-bit is equal to 1 (Input is a negative number): **then** Take 2's complement of the values stored in R5 register (normalized value of mantissa) and store the results to a temporary register R6 (size 12 bits).
32: **else if** Sign-bit is equal to 0 (Input is a positive number): **then** Store the value of temporary register R5 to temporary register R6 without any modifications.
33: **end if**
34: Connect the temporary register R6 to the output.
35: 12 bits signed binary number.

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Table 4: *Synthesis Summary for 17 bit Custom Precision Floating Point to 12 bit Signed Binary Number Conversion.*

FPGA Processor	Speed Grade	Number of Slices Used	Number of Slice Flip Flops Used	Number of 4 input LUTs Used	Number of bonded IOBs Used	Maximum Frequency
Spartan 3E XC3S500E	-5	56 out of 4656	47 out of 9312	100 out of 9312	30 out of 232	199.222 MHz
Spartan 3E XC3S1200E	-5	56 out of 4656	47 out of 9312	100 out of 9312	30 out of 232	199.222 MHz
Spartan 6 XC6SLX25	-3	48 out of 30064	72 out of 15032	90 out of 486	30 out of 226	248.738 MHz
Virtex 4 XC4VFX100	-12	55 out of 42176	47 out of 84352	99 out of 84352	30 out of 576	340.833 MHz
Virtex 5 XC5VFX100T	-3	47 out of 64000	61 out of 64000	80 out of 339	30 out of 680	440.567 MHz
Virtex 6 XC6VCX130T	-2	47 out of 160000	72 out of 80000	90 out of 408	30 out of 240	461.563 MHz

Conclusion and Future work

We have successfully implemented and tested the functionality of custom precision floating point numbers on FPGAs. The main objective of this research is to develop and implement a real-time sensor data acquisition system based on FPGA. In order to achieve it, the following activities are planned be carried out in future: Implementation of Multiplication, Addition/Subtraction and Division algorithms on custom precision numbers on FPGAs, Implementation of I^2C protocol to read serial ADC data on FPGA, Implementation of SD card and display module on FPGA to store and display real-time sensor data. These algorithms would be helpful in handling physical connections, storage and display of incoming sensor data and implementation of some basic digital signal processing techniques.

References

[1] Xilinx documentation on Field Programmable Gate Arrays (FPGA), Available online at: http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm (last accessed on 26-Oct-2012.

[2] Samir Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, ISBN: 0-13-451675-3.

[3] Uwe Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays (Signals and Communication Technology), ISBN: 978-3-540-72613-5.

[4] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008, pp.1-58, Aug. 29 2008.

[5] Decimal to Floating Point Conversion, Tutorial from Computer Science group in the Department of Mathematics and Computer Science, Mississippi College, Clinton, Mississippi, USA. Available online at: http://sandbox.mc.edu/~bennet/cs110/flt/dtof.html (last accessed on 03-Oct-2012).

[6] Floating Point to Decimal Conversion, Tutorial from Computer Science group in the Department of Mathematics and Computer Science, Mississippi College, Clinton, Mississippi, USA. Available online at: http://sandbox.mc.edu/~bennet/cs110/flt/ftod.html (last accessed on 03-Oct-2012).

[7] L. Louca, T. A. Cook, W. H. Johnson, Implementation of IEEE single precision floating point addition and multiplication on FPGAs, FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on, pp.107-116, 17-19 Apr 1996.

[8] W. B. Ligon, S. McMillan, G. Monn, K. Schoonover, F. Stivers, K. D. Underwood, A re-evaluation of the practicality of floating-point operations on FPGAs, FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on, pp.206-215, 15-17 Apr 1998.

[9] A. Guntoro and M. Glesner, High-performance fpga-based floating-point adder with three inputs, Field Programmable

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

Logic and Applications, 2008. FPL 2008. International Conference on, pp.627-630, 8-10 Sept. 2008.

[10] M. Al-Ashrafy, A. Salem, W. Anis, `An efficient implementation of floating point multiplier`, Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International, pp.1-5, 24-26 April 2011.

[11] Diniz, P.C.; Govindu, G., `Design of a Field-Programmable Dual-Precision Floating-Point Arithmetic Unit`, Field Programmable Logic and Applications, 2006. FPL '06. International Conference on, pp.1-4, 28-30 Aug. 2006.

[12] Shao Jie; Ye Ning; Zhang Xiao-Yan, `An IEEE Compliant Floating-Point Adder with the Deeply Pipelining Paradigm on FPGAs`, Computer Science and Software Engineering, 2008 International Conference on, vol.4, pp.50-53, 12-14 Dec. 2008.

[13] Kumar Jaiswal, M.; Chandrachoodan, N., `Efficient Implementation of Floating-Point Reciprocator on FPGA`, VLSI Design, 2009 22nd International Conference on, pp.267-271, 5-9 Jan. 2009.

[14] Ould Bachir, T.; David, J.-P., `Performing Floating-Point Accumulation on a Modern FPGA in Single and Double Precision`, Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on, pp.105-108, 2-4 May 2010.

[15] Kumar, Y.; Sharma, R.K., `Clock-less Design for Reconfigurable Floating Point Multiplier`, Computational Intelligence, Modelling and Simulation (CIMSiM), 2011 Third International Conference on, pp.222-226, 20-22 Sept. 2011.

[16] `Texas Instruments ADS7828 datasheet`, Available online at: www.ti.com/lit/ds/symlink/ads7828.pdf (last accessed on 26-Oct-2012).

[17] `Xilinx ISE Project Navigator Ver. 13.4 and Xilinx ISE Simulator`, Available online at: http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm (last accessed on 26-Oct-2012).

[18] `Digilent NEXYS 2 FPGA board`, Available online at: http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS2 (last accessed on 26-Oct-2012).

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

[19] `Xilinx Spartan3E FPGA Datasheet`, Available online at: `www.xilinx.com/support/documentation/data_sheets/ds312.pdf` (last accessed on 03-Oct-2012).

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

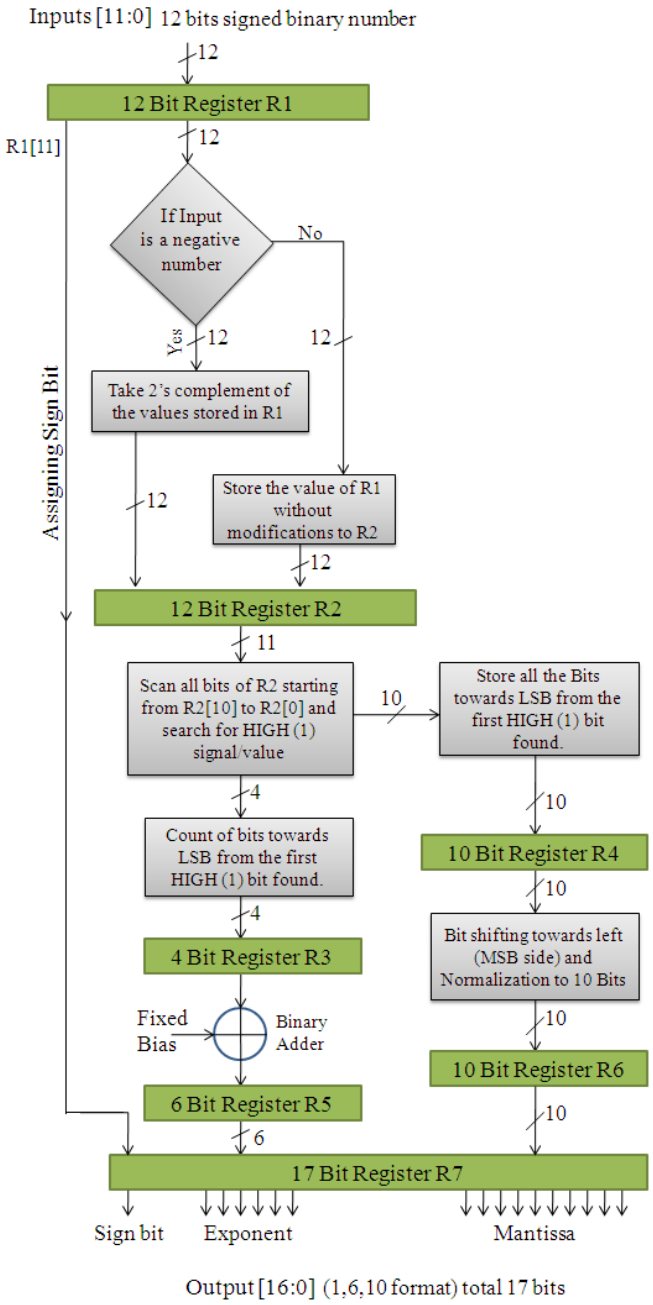

Figure 4: *Flow diagram for conversion of a 12 bit Signed Binary Number in to a 17 bit Custom Precision Floating Point Number - Pipelined approach*

HCTL Open Int. J. of Technology Innovations and Research
HCTL Open IJTIR, Volume 1, January 2013
e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

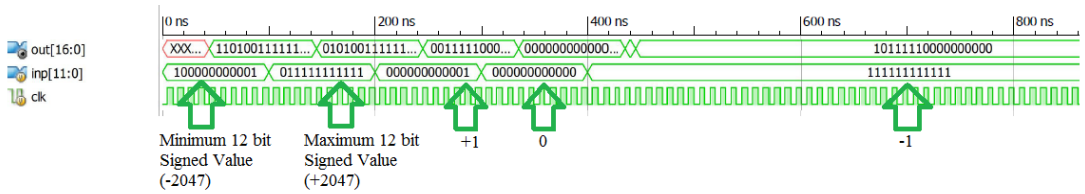

Figure 5: *Simulation Results for 12 bit Signed Binary Number to 17 bit Custom Precision Floating Point Conversion*

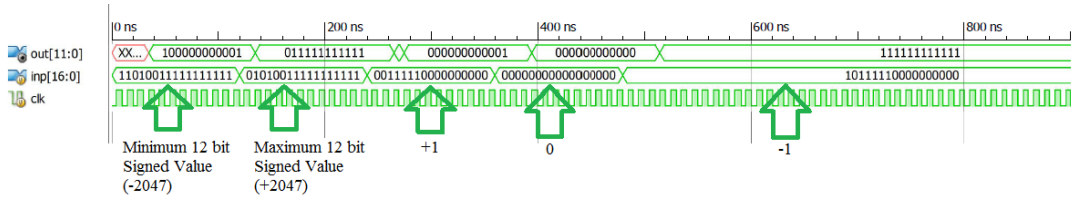

Figure 6: *Simulation Results for 17 bit Custom Precision Floating Point to 12 bit Signed Binary Number Conversion*

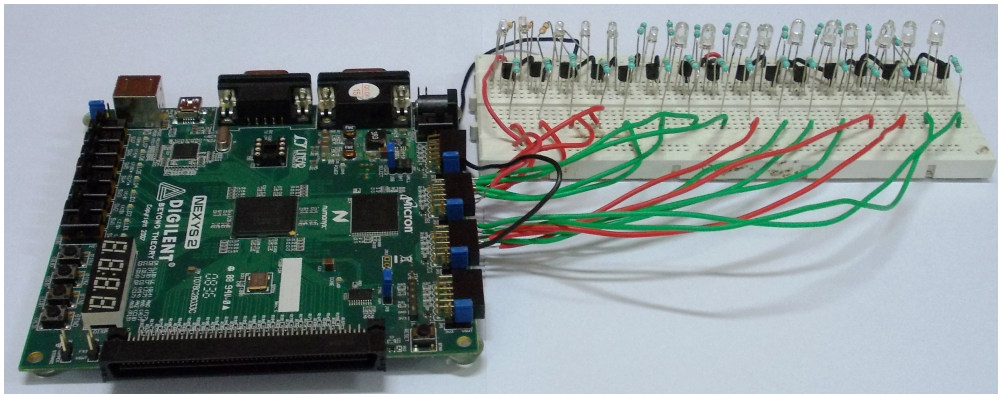

Figure 7: *Digilent NEXYS 2 FPGA board hardware setup for algorithm testing and verification*

e-ISSN: 2321-1814
ISBN (Print): 978-1-62776-012-6

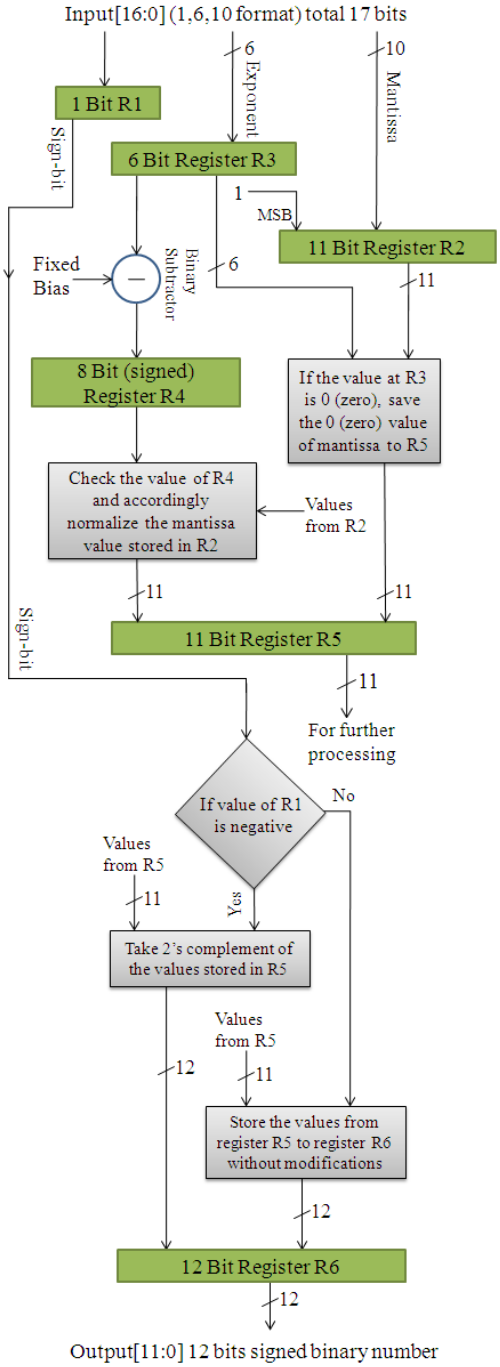

Figure 8: *Flow diagram for conversion of a 17 bit Custom Precision Floating Point Number in to a 12 bit Signed Binary Number - Pipelined approach*