International Conference on Advances in Materials, Computing and Communication Technologies
In Association withInternational Journal of Scientific Research in Science and Technology
Volume 9 | Issue 1 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)

HUB Floating-Point Adder Using Double Path

Dr. S. Ramesh[1], Saravanavel K[2]
[1]Professor Department of ECE, KPR Institute of Technology Coimbatore, Tamil Nadu, India
[2]Student (ME – VLSI) KPR Institute of Technology, Coimbatore, Tamil Nadu, India

ABSTRACT

Several previous publications have shown the area and delay reduction when implementing real number computation using HUB formats for both floating-point and fixed-point. In this paper, we present a HUB floating-point adder for FPGA which greatly improves the speed of previous proposed HUB designs for these devices. Our architecture is based on the double path technique which reduces the execution time since each path works in parallel. We also deal with the implementation of unbiased rounding in the proposed adder. Experimental results are presented showing the goodness of the new HUB adder for FPGA

Keywords—Component, Formatting, Style, Styling, Insert

I. INTRODUCTION

to When specific Floating-Point (FP) is needed for some applications, Field-Programmable Gate Array (FPGA) design allows to meet the required features. Thus, nowadays many systems are not implemented in ASIC but using FPGAs. Traditionally FPGA implementations use fixed-point arithmetic mainly because many of the Digital Signal Processing (DSP) applications tolerate error precision providing low-cost implementation at the same time. However, in the last years a fast growth of floating-point implementations and studies has been seen in the literature. There are more DSP applications implementing complex algorithms which require extended dynamic range and higher precision. The drawback is that the implied implementations on FPGA are costlier than their fixed-point counterparts. However, there are some promising researches proposing designs of adders and multipliers (key units on most DSP applications) which use other format than the IEEE-754 standard for binary floating point with lower cost. Specifically, the implementations on FPGA of an adder and multiplier are analyzed in having simultaneously less area and delay (compared to conventional implementations). In this brief, we use the same format as that used in named HUB format.

Half Unit Biased

HUB is the acronym of Half-Unit-Biased format and it is based on shifting the standard numbers by half unit in the last place (ULP). Some of its important features are that the two's complement is carried out by bit-wise inversion, the round to- nearest is performed by simple truncation, and requires the

same number of bits for storage as its conventional counterpart for the same precision. Thanks to those characteristics, it is possible to eliminate the rounding logic which significantly reduces both area and delay.

HUB in adder

A floating-point HUB number, x, has a similar representation as the binary floating-point standard. The difference between both representations is in the mantissa. A normalized HUB mantissa, Mx, has both a value $1 < Mx < 2$ and an implicit least significant bit (ILSB) which equals one. The efficiency of using HUB formats for floating-point approach has been demonstrated in several works, the authors analyze the benefits of using HUB format for floating-point adders, multipliers, and converters from a quantitative point of view for ASIC implementation. The HUB adder proposed in is optimized for FPGA devices in achieving excellent results. In this paper we present new architectures based on the double-path technique that speed up the previous results of HUB addition in FPGA devices. Moreover, the problem of bias when rounding in the previous architectures is overcome by adapting the proposal in.

Rounding in IEEE754

The rounding operation is performed in almost all arithmetic operations involving real numbers. There are several ways to perform this operation, although unbiased rounding-to-nearest (RN) has the best characteristics. It provides the closest possible number to the original exact value, but if the exact value is exactly halfway between two numbers, then it is selected randomly. The most commonly used approach is the tie-to-even method, which is the default mode of the floating-point (FP) IEEE-754 standard . However, the implementation of this rounding mode is relatively complex, and the area and the delay introduced for rounding circuits may be very large, since they normally lead in the critical path. For this reason, it is generally used in the FP circuits. Many researchers have proposed different architectures to reduce the impact of this delay by merging rounding with other operations or removing it from the critical path, wherein if the result of an addition is the input to another one, the addition required for rounding up is postponed until the next operation. In a dedicated circuit to compute the sticky bit in parallel with the main path was proposed with the aim of accelerating the implementation of multiplication.

A compound adder (a circuit that, having a carry-save input, delivers the results and the result plus one) was proposed in to generate the rounded result of any operation. In three different methods were compared for multipliers that simplify rounding decisions and merge the rounding up with the computation of the operation. Similarly, Burgess proposed combining rounding with the final addition to convert the carry-save solution into the conventional representation.

RN in HUB

A totally different approach would be to use a new real-number encoding, in order to simplify the implementation of RN. Thus, the problem would change from optimizing the rounding operation to dealing with arithmetic operations under the new number representation. This proposal is found with RN representations bandwidth half-unit-biased (HUB) formats. Together with other advantages, these new formats allow performing RN simply by truncation. On the other hand, these new formats are based on the simple modifications of the conventional formats and so could be applied to practically any conventional format. In this paper, we focus on the HUB FP formats.

The efficiency of using the HUB formats for a fixed-point representation has been demonstrated By reducing the bit width while maintaining the same accuracy, the area cost and delay of finite impulse response filter implementations have been dramatically reduced and similarly for the QR decomposition . In this paper, we perform a quantitative estimation of the benefit obtained using

the HUB formats to implement the FP computation systems under RN. Some preliminary results for half-precision FP adders and multipliers were presented. The work in shows that the area and power consumption of a basic FP adder could be improved by up to 70% for high frequencies when using HUB formats, whereas they remain the same for the basic FP multiplier. In addition to a deeper analysis, in this paper, we extend these results to other sizes and circuits, such as converters.

In comparison with the work i, the main contributions of this paper are as follows:

1) A detailed architecture for basic adders and multipliers to deal with HUB numbers;
2) A study of the conversions between different FP formats and the corresponding architectures;
3) The experimental comparison of accuracy between HUB and conventional formats;
4) Measures of improvements in area, speed, and power consumption for single- and double-precision adders, multipliers, and converters.

FPGA

Advancements in Field Programmable Gate Array (FPGA) technology are continuously being reported. High speed and flexibility, possibility of taking advantage of the inherent parallelism of many systems and algorithms, short time-to-market, good cost- performance tradeoff, large amount of embedded resources, and availability of specialized intellectual property (IP) cores make FPGAs the preferred implementation platform in many industrial applications. Until a relatively recent time there were two separate approaches to design digital systems for industrial control applications: a sequential (software) approach based on either microcontrollers or Digital Signal Processors (DSPs) and associated embedded peripherals, and a parallel (hardware) approach, usually restricted to solve specific parts of problems requiring high-performance solutions.

Industrial penetration of this second approach was conditioned by the limited knowledge of the technology and the design tools, lack of maturity of these tools, price, and lack of some specialized hardware functionalities. One of the major impediments to a wider adoption of reconfigurable computing as a new paradigm is the complexity of programming FPGAs and the need for some hardware design expertise to tame them. As FPGAs evolved taking advantage of fabrication technology scaling down, vendors started to develop soft processor cores that may be implemented from standard FPGA resources, as well as to integrate embedded (hard) processors in their devices. This trend has seen a tremendous continuous development, to the extent that current solutions are countless.

Because of this, the past dichotomy of design approaches resulted in a paradigm shift that constitutes the major current asset of FPGAs, which cannot be just seen as hardware accelerators anymore, but as very powerful System-on-Chip (SoC) platforms. The combination in a single chip of embedded (or soft) processors with custom optimized, high performance hardware peripherals has open the door for the unlimited application of FPGAs in all areas of digital design for industrial applications. Not surprisingly, the characteristics, design tools and methodologies, and application areas of these devices have been extensively analyzed over the past years , just to mention the works specifically focused on industrial systems. However, being a relatively mature but also still young area, new features are continuously being developed.

Therefore, in the authors' opinion a review of the most recent advancements in FPGA technology will be useful for the industrial informatics research community. Thus, the aim of this article is to provide such review together with an analysis of the impact the new features of current devices may have in the design of digital systems for industrial applications. The analysis is mainly carried out in three areas, namely digital real-time simulation, advanced control techniques, and electronic instrumentation, with

focus on mechatronics, robotics, and power systems design.

One major issue when evaluating new architectures is determining how a fair comparison to existing commercial FPGA architectures can be made. The Versatile Place and Route (VPR) tool is widely used in FPGA architecture research; however, the computer-aided design (CAD) algorithms used within are different from those of modern FPGAs, as is its underlying island- style FPGA architecture. As examples, VPR does not support retiming, nor does it support carry chains that are present in all major FPGA devices. To enable modeling of our FPFPGA and comparison with a standard island-style FPGA, we propose a methodology to evaluate an architecture based on an existing FPGA device. The key element of our methodology is to adopt virtual embedded blocks (VEBs), created from the reconfigurable fabric of an existing FPGA, to model the area, placement, and delay of the embedded blocks to be included in the FPGA fabric. Using this method, the impact of incorporating embedded elements on performance and area can be quickly evaluated, even if an actual implementation of the element is not available.

II. PROPOSED METHOD

BASIC OPERATIONS UNDER HUB FORMATS

An HUB FP format should include a significand that is represented using an HUB fixed-point number and an exponent that is represented in any conventional way . An HUB fixed-point format is produced when the exactly represented numbers (ERNs) of a conventional representation are increased (or biased) by half unit in the last place (ULP). This shifting of the ERNs could be seen as an implicit least significant bit (ILSB) set to one. For example, the HUB version of the IEEE-754 single precision has 25 bits for the significand, where the first and last bits are implicit and equal to 1, but only 23 bits are stored, as in the conventional version.

Fig. 2.1 shows an example for a binary FP format with 3-bit significand. Given a real value, its representation using either conventional or HUB formats will produce different rounding errors, although the accuracy of both the formats is the same. In fact, the rounding errors for both the formats are complementary (i.e., the addition of both the rounding errors equals 0.5 ULP). The main advantage of computing the HUB formats is that the two's complement operation is implemented simply by a bitwise inversion and RN by truncation. The unbiased rounding may require forcing the LSB to zero, when all the discarded bits are zero .

The general procedure to operate with the HUB numbers involves the following steps : First, the ILSB is explicitly appended to the significand of input operands. Second, the operation is performed in a classic way, such that all the bits of the significand result before rounding are obtained. Finally, the significand is rounded simply by truncation. However, since the ILSB is a constant value, the datapath could be further optimized depending on the specific operation. Next,

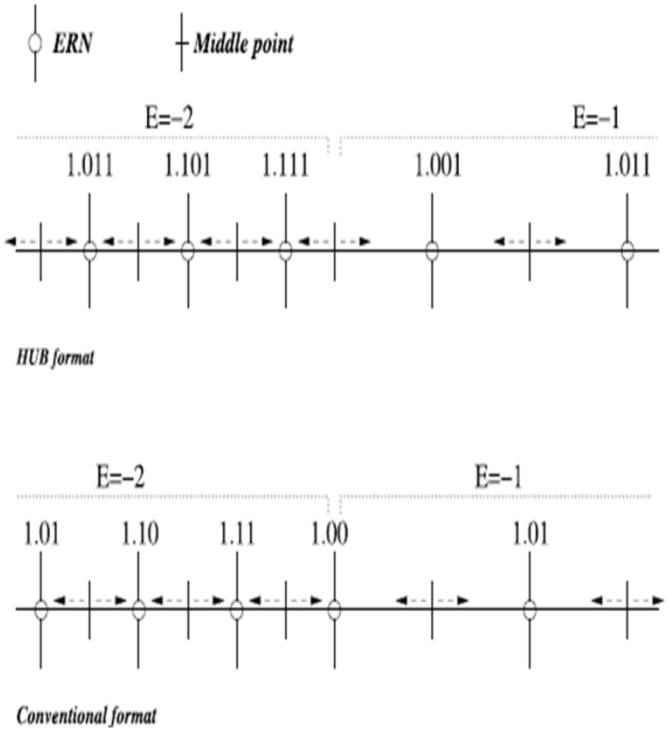


Fig. 2.1. Example of ERNs for a conventional FP format and its HUB version.

we develop several architectures to support the HUB numbers. These architectures are adapted from the basic architectures described . We are aware that many optimizations to these architectures have been proposed in the literature. However, none can be selected as the best, since this selection depends on many different factors. Even if the more relevant ones were selected, it would not be possible to review all of them in this paper. Thus, we simply describe the adaptation of these basic architectures with the aim of their being used as examples to investigate the implementation of other much more sophisticated approaches.

FP ADDERS

A basic FP addition for conventional numbers requires several steps, which could be implemented using the significand datapath shown in Fig. 2.2(a). First, the operand exponents (d = Ex – Ey) are compared and the significands are aligned accordingly. The latter is usually performed by right shifting (|d| bits) the significand corresponding to the number with the lowest exponent, which is selected using the swap module and the sign of d. The computation of the sticky bit corresponding to the bits shifted beyond the precision of the significand is also performed. The sticky bit is required for the computation of two's complement and rounding. Second, either the effective addition or the subtraction of the aligned significands is performed for the m + 3 MSBs (the significand plus guard, round, and sticky bits). In general, in order to perform subtraction, the significand corresponding to the lower exponent is previously one's complemented using the significand comparator and the conditional bit inverters.

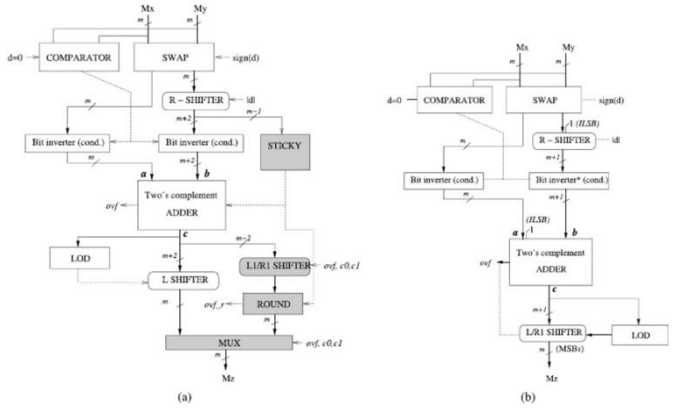


Fig. 2.2. Basic FP adder architectures for (a) standard and (b) HUB numbers.

Moreover, the sticky bit is introduced in the adder to complete the two's complement transformation. The result of addition has to be normalized by shifting 1 bit to the right, if an overflow is produced. Otherwise, it is shifted to the left if there are leading zeros whose number is computed in the leading one detector. Besides normalization, the result has to be rounded based on the two LSBs of the result and the sticky bit. However, no roundup is required when the result has at least two leading zeros . Thus, left shifting and rounding are performed in parallel paths. On the other hand, the new exponent is also generated in a parallel path. If the result of addition is rounded up, an overflow may be produced, which requires a new correction of the exponent. The same basic architecture could be used for the HUB numbers, although the significands are 1 bit larger and the rounding circuit is removed. However, knowing that the ILSB always equals 1, the significand datapath is further optimized, as shown in Fig. 3.2(b). The first difference is in the right shifter used to align the significands, because the ILSB has to be included at the input to obtain a correct result if no shifting is performed. Furthermore, the sticky-bit computation logic has been removed. Given that the ILSBs of both the significands equals 1, the sticky bit is always one for nonaligned significands. Moreover, the sticky bit is not required for aligned significands because shifting is not performed.

In this HUB architecture, we should note that the conditional bit inverters directly perform the two's complement, as explained in Section II, and no carry input is required in the fixed-point adder. The conditional inverter at the output of the right shifter has to be modified to control when shifting is not performed. In this case, the ILSB is explicitly represented by the LSB of the output. It then has to be set to 1 after the inversion to complete the two's complement operation.

On the other hand, Fig. 2.2(b) shows that the ILSB of the second operand is appended at the corresponding input of the fixed-point adder. Despite this, the fixed-point adder is slightly shorter than the one shown in Fig. 2.2(a). The latter requires two guard bits and the carry input for the sticky bit (i.e., m +2 bits) due to the rounding operation. However, in the HUB approach shown in Fig. 3.2(b), no guard bits are required because rounding is performed by truncation. Thus, the fixed-point adder has only m +1 bits (one additional bit to support the ILSB). In fact, the ILSB is shown in the architecture to simplify the explanation, although, this fixed-point addition can be implemented using an m-bit adder and an inverter.

Finally, the rounding path [gray in Fig. 2.2(a)] is removed, because rounding is simply performed by truncation. Consequently, given that explicit rounding up is not performed for the HUB architecture, overflow could not occur after rounding. Thus, the additional correction of the exponent required in Fig. 2.2(a) is eliminated, which also simplifies the exponent data path.

PROPOSED DOUBLE-PATH HUB ADDER

The proposed double-path HUB adder is presented in Fig.3.3. It has a global structure similar to that of the classic double- path implementation, except that the circuits required for rounding have been eliminated to handle floating point HUB numbers. The left adder forms the Close path, and the right adder forms the Far path, both including only one variable shifter (R-SHIFTER in the Far path and L-SHIFTER in the Close path, shadowed in Fig. 3.3). The variable right shifter, R-SHIFTER of Fig. 3.2, is now placed in the Far path and the variable left- and one-bit right shifter L/R1-SHIFTER of Fig. 3.2 is now placed in the Close path. Apart from the aforementioned differences, when comparing the double-path approach (Fig. 3.3) with the previous single path approach (Fig. 3.2) we can see that the comparator of Fig. 3.2 has been prevented and the inversion of one of the operands (if required) is performed before shifting in the double- path architecture (Fig. 3.3). We can also see that there is a fixed R1- SHIFTER in the Close path and a fixed L1/R1-SHIFTER in the Far path. These modules are explained later and the logic needed to implement them is very simple (similar to one multiplexer).

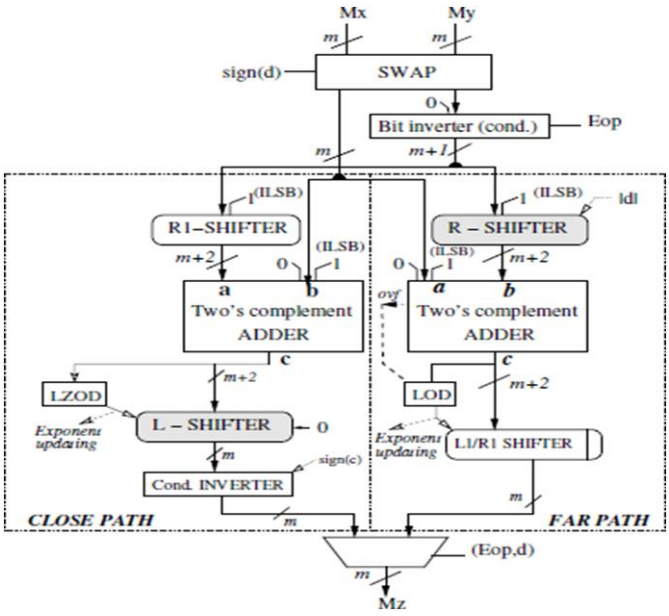


Fig. 2.3. Double-path HUB adder

Before entering in Close and Far path, the swap module of Fig. 3.3 places the mantissa of the highest exponent in the left output depending on the sign of the difference of exponents (d). This ensures that the mantissa of the greatest number is located at the left output of the swap module except when the difference of exponent is 0 (d = 0), in which case the position of the greatest operand is unknown. The top conditional inverter of Fig. 2.3 inverts the operand if the Effective Operation (Eop) is a subtraction only. Next, the operands arrive to both the Close and Far paths.

Note that in both paths the Implicit Least Significant Bit (ILSB) of the operands is incorporated as the Least Significant Bit (LSB) of the operation in the suitable modules in the architecture (R1-SHIFTER, R-SHIFTER and both adders). Note, as well, that sign extension (bit 0) is incorporated in the top conditional inverter and at the second input of the two adders.

Let us deal with the Close and the Far paths separately:

CLOSE PATH.

The Close path is intended for effective subtraction of two floating-point HUB numbers with a difference between the exponents less than two (d = 0; 1). In this case, aligning the input operands is almost not required, but normalizing the results may required large left shifting. The R1-SHIFTER module in the Close path of Fig. 3.3 is used when d = 1 to perform a fixed shift of one position to the right (if d = 0 it allows the data to go through without shifting).

The output c of the adder is suitably shifted by the variable left-shifter, L-SHIFTER. The number of bits to be shifted is calculated by the module LZOD which detects the leading 0 or 1 depending on the sign of the output c. Note that the prevention of the comparator of Fig. 3.2 makes a negative output c possible. This happens when the input operands have the same exponent (d=0) and the swap module place the mantissa of the lowest operand in the left output. Thus, a conditional inverter is required at the end of the Close path (Cond. Inverter module in the bottom of Fig. 3.3). The left shift is filled with 0's.

FAR PATH.

This path covers the rest of the cases (i.e. subtraction with d > 1 and addition). In this case, aligning the input operands may be required but, at most, one-bit shifting may be required for normalization. Therefore, a variable shifter is required at the input of the adder for alignment (R-SHIFTER in the Far path of Fig. 3.3). The final L1/R1-SHIFTER module corrects a possible overflow in the addition operation (one-bit right shift) or the case of having a pattern 0.1xxx in the result of a subtraction operation (one-bit left shift). Note that for the subtraction of the mantissas of two HUB numbers, it is not necessary to calculate the sticky bit since it is always one due to the ILSB of the second operand (the right shifted operand), and thus, it always involves an incoming carry to the adder (sticky=1). After the Close and Far paths, the final multiplexer of Fig. 3.3 selects the result of either the Close or the Far path depending of the effective operation (Eop) and the difference of exponents (d).

UNBIASED ROUND-TO-NEAREST

When the result of an operation is just in the middle of two exactly representable numbers (tie case), rounding may be performed in any direction. However, the careless election of this direction may produce a statistical bias in the results. To avoid annoying statistical anomalies of some applications due to this bias, we should round either up or down with similar probability for the tie case. In a deep analysis of the three source of bias for HUB-FP addition is presented. A bias can be produced under any of the next operations:

1) aligned addition (d = 0),
2) aligned subtraction (d = 0),
3) subtraction with difference of exponents of one (d = 1).

In the algorithms to reduce, or even, prevent the bias for the tie case are also proposed. Next, we introduce this solution to our double-path architecture. The first source of bias (aligned addition) occurs only in the Far path (which is devoted to perform subtractions with d > 1 and additions), whereas the other two always happen in the Close path (which is devoted to perform subtractions with d = 0; 1). Hence, the adaptation of the solution proposed to our architecture is almost straightforward. Fig. 2.4 shows the modifications required in the architecture of Fig. 2.3 to support the unbiased solution. The differences with Fig. 2.3 are the two modules entitled" unbiased" (shadowed in Fig. 2.4). These modules involve a very

simple logic such that the hardware cost and the penalty time are very small. On the one hand, the logic to prevent the bias for the aligned addition is included at the output of the Far path. On the other hand, the bias for the aligned subtraction is, in fact, prevented in the original double-path architecture and only the logic to eliminate the bias for the third source is needed. In this case, depending on the LSB of the result, the pattern 0111... is inserted in the left shifting.

We should note that in the simple HUB adder (Fig. 3.2) it is possible to prevent the bias only for the case of aligned addition (if the corresponding logic is appended). To eliminate the other two sources of bias a negative result at the output of the adder is required. This is

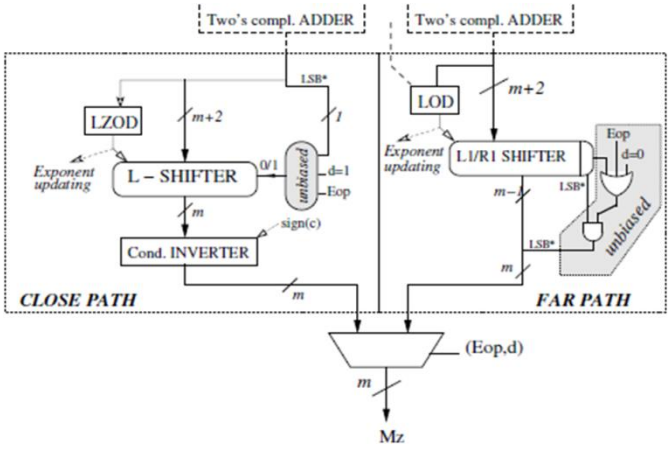


Fig. 2.4. Unbiased Double-path HUB adder

not possible in the simple path approach due to the fact that the comparator in the architecture of Fig. 3.2 ensures a positive result at the output.

III. METHODOLOGY AND RESULT

XILINX ISE DESIGN SUITE

The Xilinx ISE tools allow the design to be entered several ways including graphical schematics, state machine diagrams, VHDL, and Verilog. The ISE®Design Suite controls all aspects of the design flow. Through the Project Navigator interface, you can access all of the design entry and design implementation tools. You can also access the files and documents associated with your project.

Project Navigator Interface

By default, the Project Navigator interface is divided into four panel subwindows, as seen in Fig.4.2. On the top left are the Start, Design, Files, and Libraries panels, which include display and access to the source files in the project as well as access to running processes for the currently selected source. The Start panel provides quick access to opening projects as well as frequently access reference material, documentation and tutorials. At the bottom of the Project Navigator are the Console, Errors, and Warnings panels, which display status messages, errors, and warnings. To the right is a multi-document interface (MDI) window referred to as the Workspace. The Workspace enables you to view design reports, text files, schematics, and simulation waveforms. Each window can be resized, undocked from Project Navigator, moved to a new location within the main Project Navigator window, tiled, layered, or closed. You can use the View >Panels menu commands to open or close panels. You can use the Layout > Load Default Layout to restore the default window layout.

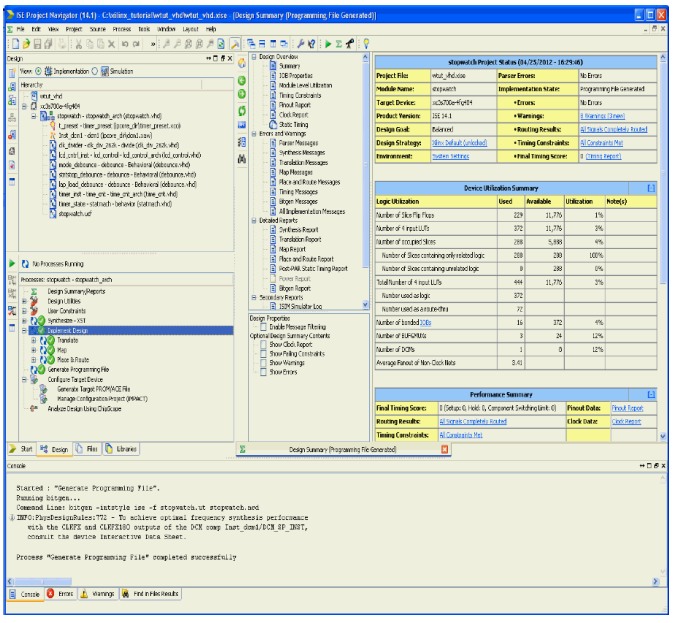


Fig.3.1.Project Navigator Interface

Design Panel

The Design panel provides access to the

- View pane
- Hierarchy pane
- Processes pane

View Pane

The View pane radio buttons enable you to view the source modules associated with the implementation or Simulation Design View in the Hierarchy pane. If you select Simulation, you must select a simulation phase from the drop down list.

Hierarchy Pane

The Hierarchy pane displays the project name, the target device, user documents, and design source files associated with the selected Design View. The View pane at the top of the Design panel allows you to view only those source files associated with the selected Design View, such as Implementation or Simulation. Each file in the Hierarchy pane has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file, for example). For a complete list of possible source types and their associated icons, see the ―Source File Types‖ topic in the ISE Help. From Project Navigator, select Help > Help Topics to view the ISE Help. If a file contains lower levels of hierarchy, the icon has a plus symbol (+) to the left of the name. You can expand the hierarchy by clicking the plus symbol (+). You can open a file for editing by double-clicking on the filename.

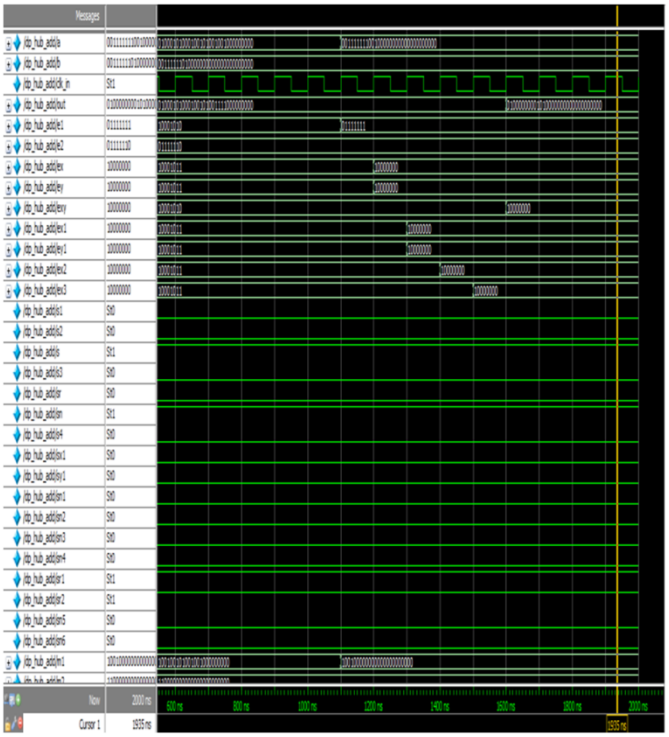
Processes Pane

The Processes pane is context sensitive, and it changes based upon the source type selected in the Sources pane and the top-level source in your project. From the Processes pane, you can run the functions necessary to define, run, and analyze your design.

3.1 Workspace

The Workspace is where design editors, viewers, and analysis tools open. These include ISE Text Editor, Schematic Editor, Constraint Editor, Design Summary/Report Viewer, RTL and Technology Viewers, and Timing Analyzer. Other tools such as the PlanAhead™ tool for I/O planning and floorplanning, ISim, third- party text editors, XPower Analyzer, and iMPACT open in separate windows outside the main Project Navigator environment when invoked.

The Above mentioned method is applied in Xilinx and the output are obtained from the simulation window



IV. CONCLUSION

A double-path based HUB-FP adder to speed up the computation on FPGA devices is proposed. Compared to the single-path adder, the cost in area and energy of the double-path approach is reasonable, giving a substantial speedup, especially for short bit-width mantissas. In addition, we analyze the impact of adding the hardware required to produce unbiased addition in the proposed architecture. In this case, with a slight increment in the delay and area, the new architecture prevents the three sources of bias.

V. REFERENCES

[1]. J. J. Rodriguez-Andina, M. D. Valdes-Pena, and M. J. Moure, "Advanced features and industrial applications of FPGAs: A review," IEEE Trans. on Industrial Informatics, vol. 11, no. 4, pp. 853–864, aug 2015.

[2]. L. Zhuo and V. K. Prasanna, "High-performance designs for linear algebra operations on reconfigurable hardware," IEEE Transactions on Computers, vol. 57, no. 8, pp. 1057–1071, aug 2008.

[3]. C. H. Ho, C. W. Yu, P. Leong, W. Luk, and S. J. E. Wilton, "Floating point FPGA: Architecture and modeling," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 17, no. 12, pp. 1709–1718, 2009.

[4]. G. Caffarena and D. Menard, "Quantization noise power estimation for floating-point dsp circuits," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 63, no. 6, pp. 593–597, jun 2016.

[5]. J. Hormigo and J. Villalba, "HUB floating point for improving FPGA implementations of DSP applications," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 64, no. 3, pp. 319–323, mar 2017.

[6]. B. Catanzaro and B. Nelson, "Higher radix floating-point representations for FPGA-based arithmetic," in 13th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05), 2005, pp. 161 – 170.

[7]. A. Ehliar, "Area efficient floating-point adder and multiplier with IEEE- 754 compatible semantics," in Proc. International Conference on Field- Programmable Technology (FPT'14), dec 2014, pp. 131– 138.

[8]. M. Langhammer, "Floating point datapath synthesis for FPGAs," in Proc. International Conference on Field Programmable Logic and Applications (FPL'08), sept 2008, pp. 355–360.

[9]. J. Hormigo and J. Villalba, "Measuring improvement when using HUB formats to implement floating-point systems under round-to- nearest," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 6, pp. 2369–2377, 2016.

[10]. IEEE Standard for Floating-Point Arithmetic, IEEE Std. 754, 2008.

[11]. J. Villalba, J. Hormigo, and S. Gonzalez-Navarro, "Unbiased rounding for HUB floating-point addition," IEEE Transactions on Computers, vol. Early access, no. 99, pp. 1–1, 2018.

[12]. Farmwald and P. Michael, "On the design of high performance digital arithmetic units," Ph.D. dissertation, Stanford University, 1981.

[13]. M. D. Ercegovac and T. Lang, Digital Arithmetic. Morgan Kaufmann, San Francisco, 2004