Low Power and Efficient Floating-Point Architecture for Division and Square Root

Hema Priya K*, Arulmurugan L, Ramesh R, Nithya B
ECE-ME-VLSI Design, Bannari Amman Institute of Technology
hemakandasamy1997@gmail.com, arulmurugan@bitsathy.ac.in, ramesh.vd19@bitsathy.ac.in, nithya.vd19@bitsathy.ac.in
*hemakandasamy1997@gmail.com

Abstract. The modern processor requires multiple dedications, fully pipelined architecture uses instructional level parallelism. So, this pipeline is very important in it. The proposed system is implemented using the fully pipelined architecture FPU in FPGA’s. Square root and division have many complications in DSP but it is important to implement in FPGA because it is so tough. Some of them will not implement the add and multiply components when compared to the slow or large which will match the real time applications. In every clock cycle, pipeline should have the capability of accepting the square root and divider. In single precision floating point Verilog code is written and implemented using spartan 6 FPGA for sqrt/division. The operation latency and issue rate have 18 clock cycles and 15 clock cycles. It uses many adders/subtractors in other high pipelined implementations. The operation latency and issue rate have 15 clock cycles and 1 clock cycles. It reduces power dissipation. It has some advantages like area utilization will be less and latency bound.

1 Introduction
Floating-point support has becoming mandatory of microprocessors. For past few years, FPU (Floating Point Unit) is used as a leading architecture. Implementing addition and multiplication has increasingly effectively and efficiently. But the support of square root and division becomes uneven. There are variations in the types of algorithm, as well as quality and performance of implementation. But here they use radix-64 floating point division and a radix-16 floating point square root for implementing floating point division and square root unit. The carry skip logic is used in a 32-bit adder. And the adder is divided into several blocks for implementation. The sizes of each block is limited by the delays of carry in the signal and the final target. For calculating the maximum size of adders which is satisfying the target delay, an algorithm is used. The unit of measurement is used as the full adders delay in our analysis. The high frequency of floating-point operation has many modern applications in their performance and it is limited by the speed of FP hardware. So, that high performance FPU is essential component of these systems. Implementing the addition and multiplication has became36efficient by increasingly. But the division and other elementary functions support such as square root are uneven. In modern processors, the most representative FP functions is division and square root. They are less
frequency than the 2 basic arithmetic operations which are addition and multiplication, a poor performance when computing these operations can impact the processor global performance. Division is considered as mirror, member of the FP family.

1.1 Floating Point Architecture
The floating point is a real number in the format of binary. It has two formats in IEEE 754 [11] which are, Binary and Decimal interchange format. In this paper, they use binary format. Binary format consists of sign bit (1 bit), exponent (8 bit) and mantissa or 23-bit fraction (23 bit) have shown in figure 1.

Figure.1: Format for Binary in IEEE 754 of Single Precision
- S: 1-bit Sign
- E: 8-bit Exponent
- M: 23- Mantissa or Fraction

The number V have some values:
1. If F is not a zero and E=255, then V= Nan (“It will not work as a number”).
2. Suppose zero is considered as the value of F and E=255 and S is considered as 1, then the value of V will be equal to the -Infinity.
3. Suppose zero is considered as the value of F and E=255 and S is considered as 0, then the value of V will be equal to the Infinity.
4. Suppose let us consider that the 0<E<255 then the value of V is considered to be (-1) **S*2**(E-127) *(1.F).
   a. Range of exponent will be equal to -127 to +128
5. If F is nonzero and E=0, then V= (-1) **S *2 **(-126) * (0.F).
   a. The values are considered to be in “the un-normalized”.
6. Suppose F is zero, then E is equal to 0 and then S is considered as 1, then V is equal to -0.
7. Suppose M is zero, then E is equal to 0 and then S is considered as 0, then V is equal to 0.

Significant means when the extra bit is added to the mantissa in form. If it is in excess of 0 and fewer than 255 in E then 1 will be in MSB significand then that number will be called as normalized number. Consider that the real number is shown as equation (1)

\[ V=(-1)^S * 2^{(E-Bias)} *(1.M) \]  

So, Bias = 127 and M = m22 2-1 + m21 2-2 + m20 2-3+...+ m0 2-23.

1.2 General Algorithm for Division Floating Point
Explanations of algorithm in multiplication of floating point is given in the flow chart. Fig.2. There are two operands which are N1 and N2, has S1, E1, M1 and S2, E2, M2 of the respective sign bit, exponent and mantissa. Let x will be equal to N1 and D will be equal to N2 and the final result will be considered as q is equal to x/d.

Floating point division has 4 steps:
1. To determine, significant divide and subtract exponents.
   S=S1 XOR S2
   E=E1-E2
   M=M1/M2
2. Mantissa M (Shifted to the left or right by the value of 1) and E exponent has been updated.
3. For fitting in the given bits result should be rounded.
4. Special values and exceptional flags are determined.
For calculating the sign bit, exponent subtract, division mantissa (bias subtraction is not needed), for fitting in the given bits result should be rounded and similarly normalization is done to describe the square root operations.

Figure 2: Division of Floating-Point Flow Chart

1.3 General Algorithm for Square Root Floating Point

It mentioned that the square root positive attributes can incorporate in the algorithm. It understands the limits of the algorithm with the number of bit increases as it deals where it has 23-bit mantissa value.

The algorithm for the proposed design is:

1) Sign Bit:
   The number which is original is said to be as sign bit.

2) Exponent Computation:
   The number which has biased exponent in the result is called as exponent bit.
   If it is odd, then 127 is added and the result of final sum is shifted to the right. (divide it by the 2 operations).
   \[ E_r = \frac{E_a + 127}{2} \]  
   (2)
   If it is even, then 126 is added and the result of the final sum is shifted to the right. (divide it by the 2 operations).
   \[ E_r = \frac{E_a + 126}{2} \]  
   (3)
   In additional, to indicate the mantissa the shift flag is used to shift it to its side of left as considered by 1 bit which will be before its square root algorithm.

3) Evaluation of Mantissa:
   - Let us consider 2 registers which are TEMP and ANS with 27-bit and 26-bit. Considered as the square root example it evaluates will be taken as 16(100002). Imagine that the TEMP is considered as 0000...0000 and ANS is considered as 0100...0000. It carries out the process of iteration. In 1st iteration, let the Mantissa will be loaded with TEMP and compare with the value of ANS, on the dependent of shift operation the result is carried out.
ANS will be equal to or greater than, then TEMP values will get cancelled with the ANS, it is shifted to the left and TEMP will get stored. The left shifting is done by the ANS contents by 1 bit starting from the current position. Then, ANS in Bit position 1 is inserted.

In every iteration, the current bit will be pointer by the point which will replaces the ANS. If ANS is greater than TEMP, so the contents will shift to left and stores in TEMP. The ANS contents are right shifted by 1 bit which is starting from the position pointer. Then, 0 will be insert in ANS of current bit position. At the final iteration, ANS values, the bit is 2 least significant bits that is considered to be the final result.

Figure 3: Floating point Square Root Architecture

2 Existing Method
Division and square root are the 2 operations that are used in floating point. The digit iteration, pre-processing and post-processing are the 3 parts of digit-recurrence. The register is shared at the quotient/root partially, reminder and the signals in other in these parts. Pre-processing block, there are not packed operands, then carries out division/square root 1st iterations. In division, Dividend and divisor to put the division in narrow interval it is scaled around +1 so it simplified the selection of quotient-digit. In this stage normalization is done in any subnormal. It depends on the number of operands that which are subnormal; it uses normal logic for cycles 1 or 2. The initial value passes to digital iterations of partial quotient/root and reminder. The digit-iteration which have the part of iterative is called as recurrence algorithm. The shared register, digit iteration has differentiated on division and square root. (1) Calculation reminder and (2) Updating partial quotient/root and calculating the digit quotient/root. In introduction, the simpler iteration radix-4 of each cycle are obtaining by overlapping method by radix-16 square root and radix-64 division: 2 radix-4 in square root and 3 radix-4 iteration in division. It computes the reminder and the partial quotient/root of new value in every iteration. For both partial reminder and result, signed-digit representing the redundant radix-2. So, reminder and partial result can be used by the positive and negative words. In next cycle it passes quotient/root partially and the reminder which will be updated. At last, the post-processing stage are passed to the unrounded quotient/root and final reminders.

Logic of Digit iteration have separate square root and division. But some previous proposal which is used to combine the square root and division in the logic of digit iteration. It has small area; timing stage will be affected in our implementations. Particularly in same cycle it is impossible to have iteration of 3 radix-4 division. Update is done for reminder in 2 to 3 CSA in the division iterations and with 2 to 4 CSA in the square root iterations. For updating the division and square root, they use 4 to 2 carry save adder in combined units. At the same time in 1 cycle, it will not fit the 3 radix 4 iteration and also degrades the performance. Carry save adder is removed so it has small area only. The quotient/root has rounded the post-processing stage. Assimilated is needed for the reminder and unrounded result, it means that the subtraction is needed from the negative to positive word, to get the non-redundant unrounded and reminder sign result and it is rounded to the final root or quotient. Floating point division, the subnormal number is the final quotient. To get an IEEE std result mantissa should be right shifted. The traditional floating-point rounding stage will not be so longer in the post-processing stage.

The division and square root use an algorithm that is the algorithm of digit recurrence in the radix-4 with the iterations of 3 for division or 2 for square root/ cycle and represented to the digit that is signed with the digit set of quotient \( f_2; 1; 0; +1; +2g \); it is, \( a=2 \) and \( r=4 \), the digit and radix set respectively. Algorithm radix 4, in every iteration the quotient or root will be 2-bits. In division 3 performance of iteration radix-4 happens in every clock cycle, quotient of bits which have 6 in every cycle, equal to divider radix-64. Commonly, in square root clock per cycle is performed in 2 radix-4 iterations and every cycle is given by root of the 4 bits,
it is equal to a square root of radix-16.

2.1 Drawbacks of Using Existing System

- Utilization of hardware will be high.
- Number of adders used are higher.
- Area, Power and Delay will be high.
- Speed will be low.

3 Proposed System for Floating-Point in DIVIDER and SQUARE ROOT:

![Diagram of proposed system]

Figure 4: General Organization of the Unit
Figure 5: Proposed Diagram of floating-point square root and division

3.1 Overview of the Square Root and Division in Single Precision Floating Point
To get the high throughput in FPGS/ASIC by 32-bit IEEE std divider pipeline floating point. 32-bit divider in floating point operation.

- 754 IEEE compliant.
- 32-bit pipelined architecture.
- FIFO 33 clock cycle latency.
- Streaming inputs/outputs supports IP core.
- Rounding to the near even numbers.
- Overflow, underflow, divide by zero, zero, not a number, infinity and de-normalization are indicated using the status flags.
- Used to simplify the process of debugging.
- High speed and high performance.
- Positive edge clocking with fully synthesizable design synchronous.

3.2 DIVISION Algorithm for Proposed System
Sequence of subtract or add and operation shifting are considered. The quotient bit correction, different sign has the dividend and the reminder finally and in the later algorithm steps reminder restore has been postponed, up to the division restoration. This algorithm avoids the operation of restoration totally. Algorithm by using NRD gives some advantages, the negative number division should have the compatibility in 2’s notation of complement.

Algorithm for division:
Step 1: It is used to check the operation = 4’b0100
Step 2: “A” Register is set as 24 bit 0
Step 3: Divisor Register is M (24 bit)
Step 4: Dividend Q register is (24 bit)
Step 5: A and also Q is concatenate
Step 6: These steps are repeated for “n” = number of time (where n = divisor in number of bit): Suppose
A=0 in bit sign, combined A and Q are shifted, 1 bit left and subtract M-A. Or combined A and Q are shifted, 1-bit left and M+A is added if sign bit is A=0, then set that the q0 value as either q0=1 or q0=0.

Step 7: At last, if A=1 in sign bit then M+A.

Step 8: By zero exception the division should be checked in sec 2.3.4.1

Step 9: The value is given for A register of the output register to the reminder and the Q [22:0] register to the result division of register output in IEEE [22:0].

It approaches different for the negative numbers. The operand negative is converted into the complement form 2’s. Take the complement number and add 1 to the complement number to determine the 2’s complement number. Normal NRD is performed using two numbers, if the both are negative.

One operand is negative and the others are positive, follow the given algorithm:

Step 1: It is used to check the operation = 4'b0100

Step 2: “A” register is set as 24 bit 0

Step 3: Divisor Register is M in 2’s compliment is (24 bit)

Step 4: Dividend Q register is (24 bit)

Step 5: A and also Q is concatenate

Step 6: Positive number uses to perform NRD and negative of 2’s complement.

Step 7: If reminder is not equal to 0: the quotient is incremented by 1. Remainder has been calculated by the formula that is Reminder= divisor* quotient- dividend (these 3 are considered to be positive).

Step 8: Sign bit of quotient is 1.

Step 9: Division by 0 exception is checked.

Step 10: Value for register output and A is assigned and the register value Q [22:0] to result division in the register output in IEEE [22:0].

3.3 Algorithm of Proposed System for Square root Efficient is Determined

The algorithm for not restoring is “partial reminder” in each and every iteration and not in “square root each bit”. In every iteration one traditional add or subtract, it means that hardware is not required, as even multiplexors or multipliers. Gives the exact answer for the bit that present in the last also. Because of this, without any add or correct operation reminder is obtained. In fast rate clock it is implemented which have iterations on simple operations.

Initial Condition:

- Reminder is 0.
- Quotient is 0.
- The number of values of square root is register D.
- N is decreasing to 0, where n 15.

Step 1: Reminder register is equal to or greater than 0, keep the register reminder as (Reminder<>=(i+1)) &3). Give the reminder value as –((Quotient<>=(i+1)) &3). Again, give the reminder value as +((Quotient<><2) |1).

Step 2: Or give the reminder as (Reminder<>=(i+1)) &3). Give the reminder value as Reminder+((Quotient<><2) |3).

Step 3: Reminder value is greater or equal to 0, then give Quotient as ((Quotient<1) |1)

Step 4: Or Quotient value is ((Quotient<1) |0)

Step 5: Reminder value is less than 0, then give reminder as Reminder+((Quotient<1) |1)

Square root value is given from Q register and remainder is given by the register reminder. Getting the result of correct bit in every iteration that include the last also in this algorithm. It depends on the before iteration based on sign of result for add/subtract is based on every iteration. If it is negative also with the successive iteration in each iteration reminder is generated. At last, the final reminder is the partial positive reminder. By adding the partial the final reminder will be taken.

3.4 Efficient Shifting Algorithm for Proposed System

Combinational logic circuit which will shifts the input in only one clock cycle is called as Barrel shifting. It consists of 3 inputs

1) Shifted number
2) Direction that should be shifted
3) Value that should be shifted
4) The direction where the input is shifted and the value of input after the output given.

**Shifting Algorithm for Proposed System:**
Step 1: Operand 4 bit is 0101.
Step 2: The below should be done for the n number of times. The value of shift register is 5 bit which checks MSB.
When it is 1, register is copied [15:0] bits and saved to result register [31:16] bits and 1 is shifted when direction is rest [15:0]
When it is 0, register result is copied [31:16] and consists of [31:16] bits with the rest part.
When it is 0, for next iteration they will use the same without any changes.

3.5 Advantages of Using Proposed System:
- Hardware Utilization will be low.
- Number of adders used will be reduced.
- Area, Power and Delay will be decreased.
- Speed will be high.
- Complexity to design the square root and division will be less.

4 FPGA Implementation

4.1 RTL Schematic

![Figure 6: Top Module Connects](image)

Single precision number is developed using Verilog HDL and simulated using Xilinx 14.2 ISIM. So, this code is broken into various modules to deal the Exponent 8-bit and mantissa 23-bit. They give many sets of inputs to the top modular block to find the result. And finally, simulation and synthesis are done to find the result.

4.2 Device Summary

Hardware utilization is done in Xilinx. So, the information can be reported by the design divider in synthesized FPGA, with efficient hardware utilization 43.38%
4.3 Timing Summary

**Timing Summary:**
---

**Speed Grade:** -2

Minimum period: 7.192ns (Maximum Frequency: 139.043MHz)
Minimum input arrival time before clock: 6.564ns
Maximum output required time after clock: 4.277ns
Maximum combinational path delay: 48.803ns

4.4 Power

### 2.3. Power Supply Summary

| Power Supply Summary | Total | Dynamic | Quiescent |
|----------------------|-------|---------|-----------|
| Supply Power (mW)    | 81.07 | 0.51    | 80.56     |

4.5 Output Waveform:

Square root has ISIM simulator, a=256 (0100000110000000000000000000000000000000000000000) and for output b= 16 (010000001100000000000000000000000000000000000000001)
5 Results and Discussion

Table I: Comparison of Existing and Proposed Parameters

| Parameters                  | Method for Existing | Method for Proposed |
|-----------------------------|---------------------|---------------------|
| Look Up Table               | 1245                | 797                 |
| Registers in Slice          | 542                 | 378                 |
| Power that consumes in Watts| 90.47               | 80.56               |
| Frequency in Maximum MHz    | 124.356             | 139.094             |
| Less Time in ns             | 6.748               | 7.192               |
| Delay path in ns            | 440.125             | 48.503              |
| Latency in clock            | 18                  | 15                  |

The parameter table shows that it can be reduced easily for division and square root with the algorithm of conventional which is based on the proposed that will be more optimistic. Number of slices and LUT’s are less. Others can see that the number of slices in proposed is 378 when compared to the existing that uses the number of slices is 542. So nearly 43.38% is reduced, enhancing the density packages. The delay combination in maximum is 48.503 ns. The square root computational to conventional, the time that suffers is 7.192 ns. So, the result gets only less time to carry out, increases the speed.

Figure 7: Latency delay comparison (in cycles)

Figure 8: Delay and minimum Time comparison in ns

Hardware is converted into code and fit in the FPGA kit. Xilinx is the tool used for our existing and proposed. It helps to design synthesis, called as “Spartan 6 Device”. The program synthesis and displays the “Device Summary”. Synthesized by designed and reports the utilization of “Target Device”. Generates and displays the time by “Timing of the critical path”.

Figure 9: Overall Performance Comparison
6 Conclusion

Instead of using carry save adder in the proposed system carry skip adder is used, because in carry save adder carry will be automatically propagated. So, the adder used will be high and utilization of hardware will also be high. In the carry skip logic, carry will not be propagated automatically. It propagates when the output is 1, it will be added at the end and it will not propagate when the output is 0. Finally, by this method the hardware utilization will be high and number of adders used will be less. By using the floating-point format, it will not allow any discussion. Computer or electronics devices that operates with real number, used to implement and operation. In arithmetic, division is more important. In proposed, they use various architecture of iteration that improve the performance of ALU. Estimated result can be achieved by the logical shifting method. Better results can be gained through the synthesis in speed and area 43.38% utility, so they used silicon dye to reduce. And it reduces the size and cost. Fast and speedy calculation are happening by less time. The reduced power dissipation is 10.95% and it neglects the cooling mechanism for gels and blowers of thermal. So finally, Square root/ division will be more efficient than the existing system.

Reference:
[1] J.D Bruguera, 2018 Radix-64 Floating-Point Divider Proceedings of 25th IEEE international Symposium on Computer Arithmetic, pp. 8794.
[2] A. Fog. 2019 Instruction Tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs https://www.agner.org/optimize/instruction_tables.pdf
[3] C. Lichtenau, S. Carlough, and S.M. Mueller. 2016 Quad Precision Floating Point on the IBM z13TM Proceedings of 23th IEEE international Symposium on Computer Arithmetic
[4] A. Nannarelli. 2016 Performance/Power Space Exploration for Binary64 Division Units IEEE Transactions on Computers, 65. 5, pp. 1671-1677
[5] D. Piso and J.D. Bruguera. 2011 Variable Latency Goldschmidt Algorithm Based on a New Rounding Method and a Remainder Estimate IEEE Transactions on Computers, 60 11 pp. 1535-1546.
[6] J. Rupley, J. King, E. Quinnell, F. Galloway, K. Patton, P. M. Seidel, J. Dinh, H. Bui, A. Bhowmik. 2013 The Floating-Point Unit of the Jaguar x86 Core Proceedings of 21th IEEE international Symposium on Computer Arithmetic
[7] S. Srinivasan, K. Bhudiya, R. Ramanarayanan, P. S. Babu, T.Jacob, S.K.Mathew, R.Krishnamurthy and V.Erraguntla 2013 Split-pathFused Floating Point Multiply Accumulate (FPMAC) Proceedings of 21th IEEE international Symposium on Computer Arithmetic
[8] Pimentel, J., Bohmenstiehl, B., Baas, B. 2017 Hybrid hardware/soft-ware floating-point implementations for optimised area and through-put tradeoffs, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25, (1), pp. 100-113
[9] Tack-Jun, K., Sondene, J., Draper, J.:2017 Floating-point division and square root using a Taylor-Series expansion algorithm, 50th Mid-west Symposium on Circuits and Systems, , pp. 305-308
[10] Liang-Kai, W., Schulte, M.:2015 Decimal floating-point square root using Newton-Raphson iteration , IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP’05), pp. 309-315
[11] Cornea-Hasegan, M. Golliver, R., Markstein, P.: 2019 Correctness proofs outline for Newton-Raphson based floating-point divide and square root algorithms, IEEE Symposium on Computer Arithmetic, pp. 96-105