24th IEEE International Conference on Application-specific Systems, Architectures and Processors, June 2013.

# FPGA and ASIC Square Root Designs for High Performance and Power Efficiency

Shashank Suresh, Spiridon F. Beldianu and Sotirios G. Ziavras
Department of Electrical and Computer Engineering
New Jersey Institute of Technology, Newark, NJ 07102
ziavras@njit.edu

***Abstract* - Floating-point square root is a fundamental operation in signal processing and various HPC applications. Since this is an expensive operation in resource and energy consumption, its efficient implementation should be of priority in future multicores that will face dark silicon issues. This paper presents a low-cost, low-power consumption design to calculate the square root using the IEEE754 single-precision floating-point format. Two versions of the design are investigated with and without clock gating (CG), respectively. Evaluation involves FPGA and ASIC technologies at 40 and 65 nm. Substantial performance growth and reduced power consumption are gained as compared to a popular iterative solution. The ASIC design demonstrates much lower power consumption, which at 40 nm is lower than that at 65 nm by about a threefold. At 40 nm, CG for the ASIC realization is justified primarily for low activity rates.**

***Keywords—FPGA, ASIC, floating-point square root, energy consumption, multicore processors.***

## I. INTRODUCTION

FPGAs are commonly used to prototype processing or communication modules which are then sometimes converted into ASIC implementations [1, 2, 14]. Apart from the basic arithmetic operations, such as addition, subtraction and multiplication, the square root is a key operation often required for the realization of signal and image processing algorithms. These operations are often implemented on high-performance embedded platforms that have resource and power constraints. The square root operation is also present in several high-performance computing applications, such as Cholesky decomposition [9], LU factorization [12, 13], solution of quadratic equations, and other modern applications [10]. Hardware support for this operation is also expected to be common in future heterogeneous multicore processors since different cores will have different capabilities and various types of hardware accelerators will be present for intensive arithmetic operations [16]. Since the square root is an expensive operation in terms of resource and energy consumption [7], its efficient implementation should be of high priority in future multicore designs that will also face major dark silicon issues [17].

SSE vector support may not be needed for every core in multicore processors [16], and thus the sharing of vector-processing hardware among multiple cores is highly justified [18]. Some shared vector accelerators in future multicores will implement directly the square root [15]. Square root computation requires more specialized hardware than multiplication or division due to its complexity [7]; it usually involves iterative approximation techniques that are computationally intense [3, 7].

We introduce an enhanced square root calculation which is based on a basic non-restoring iterative approach [5]. This basic approach was chosen for improvement due to its simplicity that can render it beneficial for efficient realizations of heterogeneous multicores using FPGA and ASIC technologies. A major objective of our work is to compare square root realizations that are fine tuned for these two technologies. We do not pursue complex square root designs, such as fused multiply/divide/square root [10] that provides the same latency for division and square root since we do not want to penalize the former. We also ignore approaches using integer hardware [7]. Although the basic design is pipelined, it uses a non-pipelined iterative stage that converges closer to the result after every iteration that consumes one clock cycle. This iterative process applies small increments to a variable, which is initialized appropriately, and then squares the variable in order to compare it with the initial input. Based on the sign of the difference, a small value is added or subtracted accordingly.

After introducing the design for 40 nm and 65 nm target FPGAs, we then optimize it to target 40 nm and 65 nm ASIC processes for higher performance and lower energy consumption. Finally, we enhance our ASIC design using CG that reduces the power consumption by deactivating the clock signal for inactive modules at any time. The results highlight the effectiveness of our design and performance-power gaps between FPGAs and ASICs.

## II. SQUARE ROOT ARCHITECTURE

The design of the floating-point square root module is divided into the pre-normalization, computational and post-normalization stages. Pre-normalization splits the 32-bit single-precision input into exponent and mantissa/significand components. The leading zeros in the mantissa are counted by logic that uses basic gates and muxes, and operates on the 24 bits in the fractional part of the input. The mantissa is shifted left until there are no leading zeros and the exponent is adjusted accordingly. The shifting of the mantissa is done with a 24-bit barrel shifter; intermediate results during an iteration are stored in 52 bits.

The computational core employs an iterative non-restoring algorithm [5, 8] that converges closer to the result after every iteration. Two registers are used. The 26-bit *reg1* register will hold the mantissa of the square root while *reg2* will contain the square of the value stored in the former register. *reg1* is initialized by setting its MSB to 1 and clearing all other bits. This is the best initial approximation for a random input as it represents half of the maximum value. *reg2* has 52 bits, that is twice as many bits as in *reg1*, and it is initialized with the square of the former constant. A loop is started with a counter equal to the number of bits in *reg1*. A comparison is made between *reg2* and the input mantissa *fract*. Based on the sign of the result, the shifted constant is either added to or subtracted from *reg1*. A similar operation is performed on *reg2* which at the end of the iteration becomes the square of *reg1*. This process is applied per iteration, converging each time closer to the desired result. When the loop is exited after 26 iterations, *reg1* will hold the square root of the input mantissa. The final phase rounds up the fraction and adjusts the exponent. Fig. 1 shows pseudocode to compute the mantissa. The result from the core is appended to the exponent and the sign bit in the post-normalization phase, thus obtaining the square root of the input. Lines 5, 6, 8 and 9 require only additions/subtractions since multiplication by two is implemented simply with a left shift of the operand.

Our redesign process had its first focus to maximize the operating frequency, and reduce the static and dynamic power. Low-cost pipeline stages were added between large blocks in order to synchronize the clock signals with the data flows in various design parts. In synthesizing the original design for FPGAs, it was observed that the pre-normalization stage had the highest cost in terms of resource utilization, counted in number of look-up tables (LUTs).

```
1     reg1=2^25
2     reg2=2^50
3     for (count=26;count >0;count --){
4       if (reg2 > fract) {
5         reg1=reg1-2^(count-1)
6         reg2=reg2+2^2(count-1) reg1*2^count
7       } else
        {
8         reg1=reg1+2^(count-1)
9         reg2= reg2+2^2(count-1)+reg1*2^count
10      }
11    }
```

Fig. 1. Pseudocode for the computational core.

The targeted Xilinx FPGAs are the 65 nm Virtex-5 (XC5VLX85T) and the 40 nm Virtex-6 (XC6VLX75T). Pre-normalization of the basic design consumed 824 and 622 LUTs on the Virtex-5 and Virtex-6, respectively. Each LUT*i* type, for *i*=1-6, accepts *i* inputs to implement functions with $2^i$ possible outputs. More complicated functionality is realized by combining the outputs of LUTs using multiplexers. Due to its complexity, the pre-normalization stage affected the design by limiting the maximum operating frequency and increasing the static power. Bottlenecks in pre-normalization were alleviated by optimizing the design. The next objective was to have low static and dynamic power consumption for the modules involved in iterations. The design was clock gated [6] to achieve low dynamic power consumption. This was more effective on the Xilinx Virtex-6, since, unlike Virtex-5, Virtex-6 offers fine-grain CG [4]. A similar CG approach was followed in the ASIC implementation.

The Virtex-5 and Virtex-6 FPGAs were chosen to have comparable resources. The design was then converted for 65 nm and 40 nm ASIC technologies using Synopsys Design compiler, VCS and PrimeTime [11]. Both designs were also simulated separately, with and without CG, for analyzing the dynamic power. The simulations were performed under various activity rates for functional units.

TABLE I. FPGA MAX. FREQUENCIES OF BASIC DESIGN.

| | Virtex-5 | Virtex-6 |
|---|---|---|
| **Max. MHz** | 75.04 | 176.65 |

TABLE II. FPGA MAX. FREQUENCIES OF ENHANCED DESIGN.

| | Virtex-5 | | Virtex-6 | |
|---|---|---|---|---|
| **Variant** | **w/o CG** | **CG** | **w/o CG** | **CG** |
| **Max. MHz** | 199.76 | 196.12 | 205.68 | 203.25 |

TABLE III. FPGA RESOURCES IN BASIC DESIGN.

| | Virtex-5 | | | Virtex-6 | | |
|---|---|---|---|---|---|---|
| | **Used** | **Avail.** | **Utilized** | **Used** | **Avail.** | **Utilized** |
| **Registers** | 452 | 51840 | 0.87% | 526 | 93120 | 0.56% |
| **Slice LUTs** | 1550 | 51840 | 2.98% | 804 | 46560 | 1.72% |
| **Slices** | 597 | 12960 | 4.6% | 270 | 11640 | 2.3% |

TABLE IV. FPGA RESOURCES IN ENHANCED DESIGN W/O CG.

| | Virtex-5 | | | Virtex-6 | | |
|---|---|---|---|---|---|---|
| | **Used** | **Avail.** | **Utilized** | **Used** | **Avail.** | **Utilized** |
| **Registers** | 475 | 51840 | 0.91% | 557 | 93120 | 0.59% |
| **LUTs** | 1055 | 51840 | 2.03% | 971 | 46560 | 2.08% |
| **Slices** | 331 | 12960 | 2.55% | 297 | 11640 | 2.3% |

TABLE V. FPGA RESOURCES IN ENHANCED DESIGN WITH CG.

| | Virtex-5 | | | Virtex-6 | | |
|---|---|---|---|---|---|---|
| | **Used** | **Avail.** | **Utilized** | **Used** | **Avail.** | **Utilized** |
| **Registers** | 449 | 51840 | 0.86% | 531 | 93120 | 0.57% |
| **LUTs** | 953 | 51840 | 1.83% | 819 | 46560 | 1.75% |
| **Slices** | 338 | 12960 | 2.60% | 276 | 11640 | 2.55% |

TABLE VI. FPGA STATIC POWER (mW) OF BASIC DESIGN.

| | Virtex-5 | Virtex-6 |
|---|---|---|
| **Static Power** | 900.27 | 1008.59 |
| **Normalized per Slices** | 41.41 | 23.19 |

TABLE VII. FPGA STATIC POWER (mW) OF ENHANCED DESIGN.

| | Virtex-5 | Virtex-6 | Virtex-6 |
|---|---|---|---|
| | **w/o CG** | **w/o CG** | **CG** |
| **Static Power** | 903.42 | 1045.54 | 1043.99 |
| **Normalized per Slices** | 23.03 | 26.66 | 24.74 |

## III. SIMULATION RESULTS

### *A. FPGA Design*

Square root calculation requires up to 40 clock cycles. Two cycles are spent in pre-normalization for an input that does not conform to IEEE754. The computation core takes

31 clock cycles; 26 cycles for the iterative algorithm and five cycles for initialization (this overhead can be zeroed with specialized circuitry, but we assume that the involved registers can also be used by other processes). Post-normalization requires three cycles. Finally, the top module that controls the three stages adds four cycles. To keep clock skew to a minimum, clock management block modules were embedded in the FPGA design. The Virtex-5 designs employ DCM (Digital Clock Module) while the Virtex-6 designs use MMCM (Mixed Mode Clock Manager) to generate a clock tree. They are instantiated using the Xilinx LogicCORE IP clocking wizard.

CG changes the clock tree structure. Due to the embedded clock blocks in Virtex-6 that increase the FPGA resources, there is slightly higher static power consumption than with the PLL in Virtex-5. A lot of the logic is synthesized using many LUTs, especially for Virtex-5. This also affects the timing and power consumption. Beyond the optimizations during design, the tool also does a better job in synthesizing the design on Virtex-6 than on Virtex-5. Hence, there are more gaps in terms of timing, area and power when targeting Virtex-5 as compared to Virtex-6.

TABLE VIII. FPGA DYNAMIC POWER (mW) OF COMPUTATIONAL CORE FOR BASIC DESIGN (CG: AUTO).

| Activity (%) | Virtex-5 | Virtex-6 |
|---|---|---|
| **0** | 39.91 | 19.44 |
| **25** | 46.88 | 22.39 |
| **50** | 55.41 | 26.14 |
| **75** | 62.66 | 28.73 |
| **100** | 67.61 | 30.75 |

TABLE IX. FPGA DYNAMIC POWER (mW) OF COMPUTATIONAL CORE FOR ENHANCED DESIGN.

| Activity (%) | Virtex-5 | Virtex-6 | |
|---|---|---|---|
| | w/o CG | w/o CG | CG |
| **0** | 18.87 | 11.81 | 12.03 |
| **25** | 24.34 | 17.98 | 16.43 |
| **50** | 29.84 | 24.26 | 21.38 |
| **75** | 35.26 | 30.21 | 26.41 |
| **100** | 40.66 | 34.82 | 30.91 |

Tables I-II show the maximum operating frequencies of the FPGA designs after synthesis; it also shows CG for the enhanced design. The operating frequency of the enhanced design is much higher for Virtex-5; it more than doubles compared to the basic design. Tables III-V compare various implementations in FPGA resource utilization. Each Virtex-5 slice contains four LUTs and four flip-flops, whereas each Virtex-6 slice contains four LUTs and eight flip-flops. Without CG, enabling circuitry is embedded by the design tool within individual flip flops; CG is facilitated selectively by the developer using external clock gates. CG generally reduces the numbers of consumed resources for both the FPGA and ASIC designs.

To obtain the average power consumed by an FPGA, the Xilinx XPower analyzer tool was used. Native Circuit Description (NCD) and Physical Constraints File (PCF) files generate the estimated power figures. For accurate estimation of the dynamic power a Value Change Dump (VCD) file is also generated to monitor all the signals and their toggling in the enhanced architecture. A testbench was designed to generate simulations for activity rates 0%-100% (in increments of 25%), and the average dynamic power was measured. For the basic design, the CG setting in Xilinx ISE is set to *Auto (*the tool attempts to optimize the design for power). For the enhanced design the flag is set to *Yes* (with CG) or *No* (without CG).

Tables VI-VII report the static power. Due to increased resources compared to the basic design, the static power increases slightly. Tables VIII-IX show the average dynamic power of the computational core under various activity rates. The enhanced design is better than the basic design on Virtex-5, and also on Virtex-6 without CG. With CG, it is still better at lower than peak activity rate. CG on Virtex-6 is the best choice for total power consumption.

TABLE X. ASIC AREA OF ENHANCED DESIGN W/O CK.

| | 40 nm | 65 nm |
|---|---|---|
| **Library: Temp ($^0$C), Volts** | 125, 1.21 | 125, 1.1 |
| **Combinational Area (μm$^2$)** | 3254.76 | 8353.44 |
| **Non-combinational Area (μm$^2$)** | 2423.56 | 6309.12 |
| **Total Cell Area (μm$^2$)** | 5678.32 | 14662.56 |

### *B. ASIC Design*

40 nm and 65 nm NLDM (Non Linear Delay Model) design libraries of Synopsys were used for synthesis, simulation, and timing/area/power estimation in order to match the feature sizes of the target FPGAs. Based on the fan-out, NLDM estimates the parasitic capacitances and resistances of wires and the loads of gates. Propagation delays are presented in a non-linear format and are calculated with a cell's output load (wire and fan-out loads) and slew rate. The data is stored in a two-dimensional LUT and intermediate values are interpolated. Apart from the transistor channel lengths, the two libraries (shown in Tables X-XI) vary with respect to the operating conditions.

TABLE XI. ASIC AREA OF ENHANCED DESIGN WITH CG.

| | 40 nm | 65 nm |
|---|---|---|
| **Library: Temp ($^0$C), Volts** | 125, 1.21 | 125, 1.1 V |
| **Combinational Area (μm$^2$)** | 3190.37 | 8076.48 |
| **Non-combinational Area (μm$^2$)** | 2389.87 | 6145.92 |
| **Total Cell Area (μm$^2$)** | 5580.24 | 14222.40 |

Design synthesis was performed using Synopsys Design Compiler. RTL and Netlist simulations used VCS (Verilog Compiler and Simulator). For power consumption reports, the design run at 900 MHz for both feature sizes, and the Switching Activity was monitored. PrimeTime from Synopsys was used to analyze the design, and the results were extracted using the SAIF file obtained from netlist simulations. Tables X-XI show area statistics for the ASIC enhanced design with and without CG. CG reduces the area needs since modules do not contain embedded clock enable ports and associated interfaces.

TABLE XII. ASIC POWER (mW) AT 40 nm W/O CG.

| Activity (%) | Net Switching | Cell Internal | Cell Leakage | Total Power |
|---|---|---|---|---|
| **0** | 0; 0% | 3.84;83.4% | 0.76; 16.6% | 4.61 |
| **25** | 0.34;6.5% | 4.12; 78.9% | 0.77; 14.7% | 5.22 |
| **50** | 0.69;11.8% | 4.40; 75.1% | 0.77;13.1% | 5.86 |
| **75** | 1.04; 16% | 4.70; 72.2% | 0.77;11.8% | 6.51 |

| | | | | |
|---|---|---|---|---|
| **100** | 1.40;19.5% | 5.01;69.8% | 0.77;10.8% | 7.18 |

TABLE XIII. ASIC POWER (mW) AT 65 nm W/O CG.

| Activity (%) | Net Switching | Cell Internal | Cell Leakage | Total Power |
|---|---|---|---|---|
| **0** | 0.02;0.2% | 9;61.8% | 5.54;38% | 14.56 |
| **25** | 0.5;3.2% | 9.5;61% | 5.6;35.8% | 15.65 |
| **50** | 1; 6% | 10.1;60.4% | 5.64;33.6% | 16.74 |
| **75** | 1.5;8.4% | 10.7;59.9% | 5.67;31.7% | 17.86 |
| **100** | 1.98;10.5% | 11.2;59.4% | 5.69;30.1% | 18.86 |

TABLE XIV. ASIC POWER (mW) AT 40 nm WITH CG.

| Activity Rate (%) | Net Switching | Cell Internal | Cell Leakage | Total Power |
|---|---|---|---|---|
| **0** | 0;0% | 2.42; 75.76% | 0.77;24.24% | 3.19 |
| **25** | 0.51; 12.56% | 2.8; 68.52% | 0.77;18.92% | 4.08 |
| **50** | 1.05; 20.81% | 3.2; 63.69% | 0.78;15.50% | 5.03 |
| **75** | 1.58; 26.43% | 3.63; 60.57% | 0.78;13.00% | 5.99 |
| **100** | 2.12; 30.39% | 4.07; 58.43% | 0.78;11.19% | 6.97 |

Tables XII-XIV show the ASIC static and dynamic power. The power at 40 nm is significantly lower than that at 65 nm by about a threefold. At 40 nm, the power with CG is lower than without CG. The difference is higher at lower activity rates. At low rates, clock gates capture most of the clock power consumption of the CG design, while at high activity rates both designs rely on distributed clocks. Cell static power is consumed by a gate when it is not switching; it is caused by currents flowing through the transistors even when they are turned off. Dynamic power has two components: switching power and cell internal power. Switching power is dissipated when charging or discharging the load capacitance at the cell output. Its amount depends on the switching activity (and obviously the operating frequency) of the cell. The larger the number of logic transitions on the cell output, the larger the switching power. Internal power is consumed in a cell for charging or discharging internal cell capacitances.

Fig. 2 plots its energy consumption per square-root computation. 40 nm with CG provides the best choice in terms of total power and energy consumption per square-root evaluation. The lower the activity rate is, the more striking are the benefits of CG. For very high activity rates, the 40 nm design without CG competes favorably since all parts of the design have to be active during execution.

## IV. CONCLUSIONS

The proposed low-cost design for square root computation with IEEE754 single-precision compliant floating-point data uses a non-restoring iterative approach that provides a high operating efficiency. 40 nm and 65 nm FPGA and ASIC designs were optimized for reduced resource and energy consumption. Several versions of the ASIC design were investigated, including designs with the absence or presence of CG. The results show substantial performance growth and reduced power consumption for both FPGA and ASIC technologies in comparison to the realization of the basic non-restoring iterative algorithm. ASIC designs demonstrate much lower power consumption than FPGA designs. ASIC power at 40 nm is lower than that at 65 nm by about a threefold. At 40 nm, CG for the ASIC design is justified for low activity rates.

## REFERENCES


[1] V. Hopkin and B. Kirk, "FPGA Migration to ASICs," WESCON Conf. Record Microel. Comm. Techn., 1995.

[2] S.G. Ziavras, et al., "Coprocessor Design to Support MPI Primitives in Configurable Multiprocessors," Integration, the VLSI Journal, Vol. 40, No. 3, 2007, pp. 235-252.

[3] Y. Li and W. Chu, "Implementation of Single Precision Floating Point Square Root on FPGAs", 5th IEEE symposium on FPGA-based Custom Computing Machines, April 1997.

[4] F. Rivoallon, "Reducing Switching Power with Intelligent Clock Gating", Xilinx white paper, March 2011.

[5] J. Bannur and A. Varma, "The VLSI Implementation of Square Root Algorithm," IEEE Symp. Comp. Arith., 1985.

[6] S. Churiwala, S. Garg and M. Gianfagna, Principles of VLSI RTL Design: A Practical Guide, Springer publ., May 2011.

[7] I. Sajid, M.M. Ahmed and S.G. Ziavras, "Novel Pipelined Architecture for Efficient Evaluation of the Square Root Using a Modified Non-Restoring Algorithm," Journal Signal Proces. Systems, Vol. 67, No. 2, May 2012, pp. 157-166.

[8] J. Al-Eryani, http://opencores.org/project,fpu100.

[9] S.G. Haridas and S.G. Ziavras, "FPGA Implementation of a Cholesky Algorithm for a Shared-Memory Multiprocessor Architecture," Parallel Alg. Applic., Dec. 2004, pp. 211-226.

[10] T.-J. Kwon, et al., "Floating-Point Division and Square Root Implementation using a Taylor-Series Expansion Algorithm," Midwest Symp. Circuits Syst., 2008.

[11] www.synopsys.com, Synopsys, Inc.

[12] X. Wang and S.G. Ziavras, "A Configurable Multiprocessor and Dynamic Load Balancing for Parallel LU Factorization," 18th Intern. Parallel Distrib, Proc. Symp., April 2004.

[13] X. Wang, S.G. Ziavras, et al., "Parallel Solution of Newton's Power Flow Equations on Configurable Chips," Intern. Journal Electr. Power Energy Syst., June 2007, pp. 422-431.

[14] R.F. Woods, et al., "High Performance DSP ASIC for Multiply, Divide and Square Root," 5th IEEE Intern. ASIC Conf. Exh., 1992.

[15] S.F. Beldianu, C. Dahlberg, T. Steele and S.G. Ziavras, "Versatile Design of Shared Vector Coprocessors for Multicores," Micropr. Micros., Oct. 2012, pp. 543–554.

[16] M. Arora, et al., "Redefining the Role of the CPU in the Era of CPU-GPU Integration," IEEE Micro, Nov. 2012, pp. 4-16.

[17] H. Esmaeilzadehy, et al., "Dark Silicon and the End of Multicore Scaling," IEEE Micro, May 2012, pp. 122-134

[18] S.F. Beldianu and S.G. Ziavras, "On-Chip Vector Coprocessor Sharing for Multicores," 19th Euromicro Intern. Conf. Paral. Distr. Netw.-based Comp., Febr. 2011.


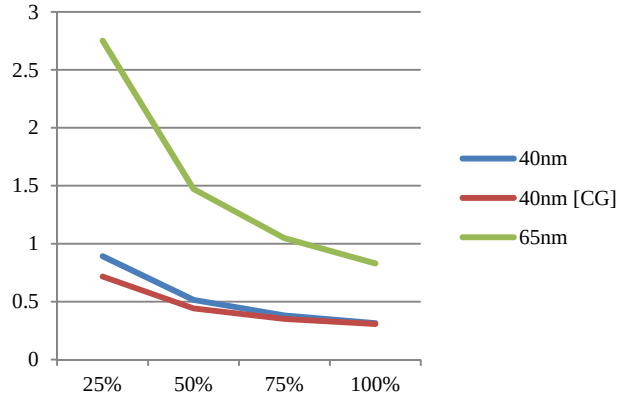


Fig. 2. ASIC energy consumption (nJ) per square-root computation of enhanced design for various activity rates.