Journal of Computer Science & Computational Mathematics, Volume 11, Issue 3, September 2021
DOI: 10.20967/jcscm.2021.03.004

Intel/Altera FPGA Implementation of CORDIC Square Root Algorithm

Muhazam Mustapha[*], Burinieamman Jayabalan, Anis Shahida Niza Mokhtar

Department of Electrical & Electronics Engineering, Faculty of Engineering,National Defense University of Malaysia, Sungai Besi Camp,57000 Kuala Lumpur, Malaysia
[*]Corresponding author email: muhazam.mustapha@gmail.com

Abstract: There is no freely available HDL module / library that calculates square root. To solve this problem this paper is presenting a modified Verilog RTL module to calculate square root that is simulated on ModelSim and Quartus Prime. The algorithm used is a CORDIC algorithm. Since the algorithm is a bit-wise processor, the accuracy is rather high, i.e. ± 0.00002 magnitude of error with [16.16] fixed point format, while the duration is rather long at 0.5μs for the same fixed point format.

Keywords: FPGA, Verilog, square root, RTL, CORDIC.

1. Introduction and Motivation

Square root is the reverse function of square, i.e. the multiplication a number with itself. Even though squaring is easy but square root is rather hard to compute since it is not representable as a finite polynomial. However, square root appears in many calculations like hypoteneus, bearing, 3D spherical distance, etc. Hence there is a need for square root implementation on HDL (hardware description language) so that such calculations can be done on hardware.

2. Related Work

Some significant work had been done on producing digital hardware circuit to calculate square root. Yunong et al. for example wrote it using MATLAB Simulink to model and verify the different GDS models for computing square roots scalar towards the final result of field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) realization [1]. John O'Leary et al. explain that hardware implementation can be very effective if it is transformed from square root of non-restoring integer. It also discusses about multiple level of abstraction of performing square root function with various techniques [2]. Y. Chandu et al. discuss that Dvanda yoga is an ancient Vedic script used to find square root function and binary digit is used in this principle. 32 binary bits of input and 16 binary bits if output is used to perform this function. It is implemented in Verilog and run it in Xilinx 14.5 software [3].

This paper on the other hand is presenting a work in [4] by Burinieamman J. to implement square root function in Verilog on Quartus Prime using a direct bit-wise CORDIC algorithm which normally used to calculate trigonometric functions bit-by-bit [5]. The work in [4] however was focused on ModelSim simulation and faced a problem in Quartus software and the delay had to be estimated. This paper shows an improvement in the code that it can be simulated with Quartus Prime.

3. CORDIC Algorithm for Square Root

CORDIC is the name given to the family of vector rotating algorithms that can be reduced to bit-wise data manipulations.

3.1 Pseudocode and C/C++ Algorithm

This algorithm is the algorithm that checks the bits in result register one-by-one if it should be set to 1 or reset to 0. This algorithm can be written in C/C++ as shown in Listing 1 for 8-bit integer [6]:

```
int sqrt(int x)
{
   int base, i, y;
   base = 128;
   y = 0;
   for (i = 1; i ${<}$= 8; i++) {
       y += base;
       if ((y*y) ${>}$ x) {
            y -= base;
       }      base >>= 1;
   }   return y;
}
```

Listing 1. Pseudo-C Code for CORDIC Square Root

The work presented in this paper is basically converting the above code into an HDL code in Verilog.

3.2 Data Flow Design

The next step in the RTL (register transfer level) design is to construct a data flow for the algorithm given in Listing 1. One modification was made on the algorithm by removing the reverting subtraction to simplify the algorithm. This can be done since we can use a comparator and only add y with base if $y * y$ is less than or equal to x. The resulting circuit is in Figure 1. The controller part of the RTL design reduces to a counter of mod-$m + n$ (total bits in input).

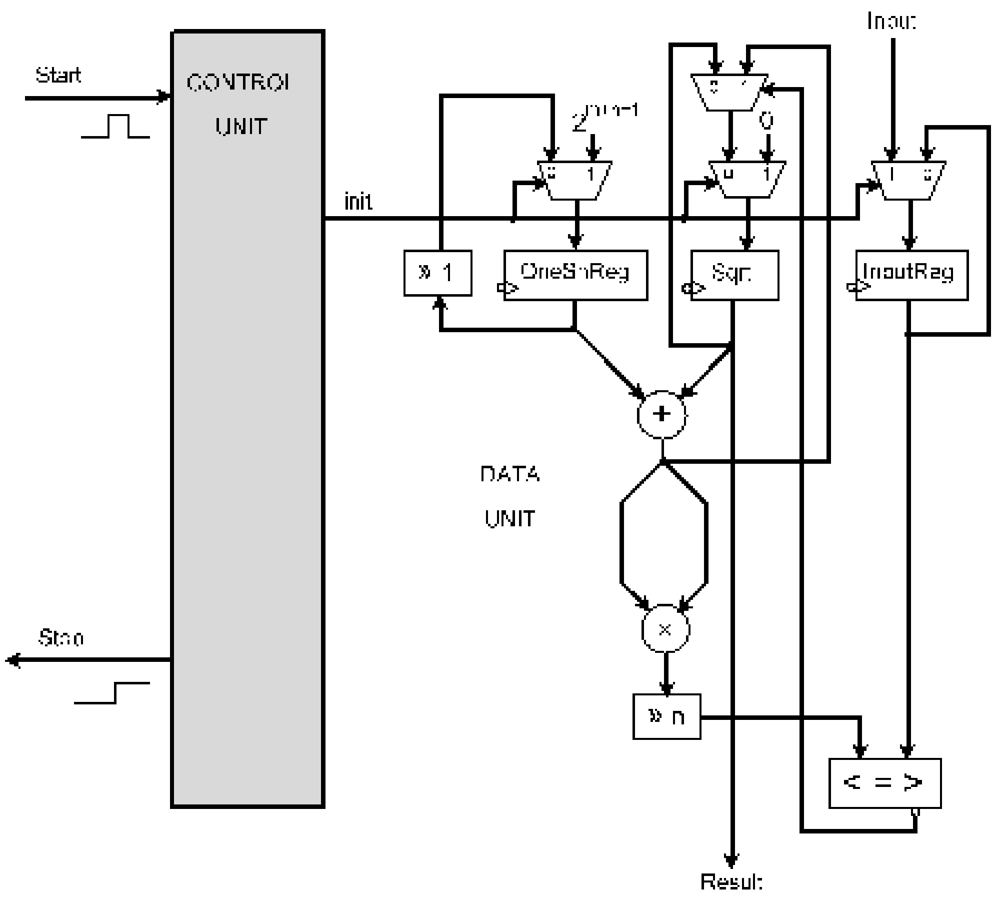

Figure 1. RTL Design of Square Root CORDIC System with Data Flow Details

3.3 Coding Methodology

The summary of coding strategy for this algorithm is as follows[1].

3.3.1 Data Unit: Assignments at Operation Junctions

For the data unit, the coding was done by *writing assignments at a few essential operation junctions*. The details follow in this subsection.

3.3.2 Control Unit: A Mod-`(m+n)` Counter

For the control unit, it was coded as a single round counter to count `m+n` (total number of bits in the input) clock pulses starting at the falling edge of `Start` signal. Once the counting complete, it stops and the `Stop` signal raised.

3.3.3 Gated Clock for Data Unit

Data unit will be triggered with clock that is gated with control unit's stop signal as shown in Listing 2. This is to ensure no operation take place when control unit issues stop signal; this behavior can also save power in the data unit.

```
assign dataClock = sysClock & ~Stop;
```

Listing 2. Gating Data Unit Clock

[1]The complete and running code of this project is published in GitHub with full documentation

3.3.4 Phase Shifted Clocking

If it is a registered assignment in data unit, it is triggered by the *negative* edge of the gated clock. The counting process in control unit, on the other hand, will be counting at *positive* edges of the system clock. This phase shifting in triggering registers between control and data units is performed in order to allow some propagation delay of signals from control unit into the data unit.

3.3.5 Shift Register

This is the shift register that shifts the single bit 1 to be added into accumulator in normal operation, or initialized to only one bit 1 at the beginning.

```
always@(negedge dataClock)
begin
   OneShReg = (init == 1) ? (1 << 31)
                          : OneShReg >> 1;
end
```

Listing 3. Shift Register Operation

3.3.6 Square Root Accumulator

This is the accumulator that stores the square root result. It is either to be added with the one bit 1 from shift register or not, depending on whether or not the addition results in a value, when squared, is less than or greater than the input.

```
always@(negedge dataClock)
begin
   Sqrt = (init == 1) ? 0 :
        ((gt == 0) ? Sqrt : AddedSqrt);
end
```

Listing 4. Result Accumulator Operation

3.3.7 Input Register

This register is to latch the input value so that it constant throughout the process.

```
always@(negedge dataClock)
begin
   InputReg=(init==1)? Input : InputReg;
end
```

Listing 5. Input Register Latching

3.3.8 Adding, Squaring and Fixed Point Bit Restoration

The squaring result needs to reduce the number of bits so that the processing is kept at the same bit size throughout the process. This is done by flushing out the right hand side (LSB) n bits (the fraction part of the fixed point format).

```
assign AddedSqrt=OneShReg + Sqrt;
assign SqrtRest=(AddedSqrt*AddedSqrt)>>16;
```

Listing 6. Adding, Squaring and Bit Restoration

3.3.9 Comparator

The comparator checks if the square is greater than the input. This status is used to select the multiplexer input at accumulator.

```
assign gt=(SqrtRest > InputReg) ? 1 : 0;
```

Listing 7. Comparison Status

4. Simulation and Results

This paper is to publish the combined results of ModelSim simulation from the [4] and the new results of Quartus Prime simulations. Tabulated progressive ModelSim simulation result is in Table 1.

As a bit-wise manipulation algorithm, it is expected that this algorithm produces a very high accuracy result that only depends on the number of bits used in the output register. At [16.16] fixed point notation, the magnitude of error is around $2^{-16} = 0.0002$.

The result of Quartus Prime simulation is in Figure 2. This gives the average delay for computation at each clock as 17ns and total delay of that amount times the number of bits processed, which is $17 \times 32 = 0.544\mu$s for [16.16] fixed point.

5. Conclusions

The work in this paper provides a freely available open source HDL code for one of the most common non-polynomial function, i.e. square root, that appears in many formula calculations. With this charitable contribution, it is hoped that research community would make a progress in designing more complicated digital system that involves heavy computations.

Table 1. Sample Simulation Result of Square Root of 2000.45

start	clk	Shift Register	stop	Result
0	0	X	0	0.000000
1	1	X	0	0.000000
1	0	8388608	0	0.000000
1	1	8388608	0	0.000000
1	0	8388608	0	0.000000
1	1	8388608	0	0.000000
0	0	4194304	0	0.000000
0	1	4194304	0	0.000000
0	0	2097152	0	0.000000
0	1	2097152	0	0.000000
0	0	1048576	0	32.000000
0	1	1048576	0	32.000000
0	0	524288	0	32.000000
0	1	524288	0	32.000000
0	0	262144	0	40.000000
0	1	262144	0	40.000000
0	0	131072	0	44.000000
0	1	131072	0	44.000000
0	0	65536	0	44.000000
0	1	65536	0	44.000000
0	0	32768	0	44.000000
0	1	32768	0	44.000000
0	0	16384	0	44.500000
0	1	16384	0	44.500000
0	0	8192	0	44.500000
0	1	8192	0	44.500000
0	0	4096	0	44.625000
0	1	4096	0	44.625000
0	0	2048	0	44.687500
0	1	2048	0	44.687500
0	0	1024	0	44.718750
0	1	1024	0	44.718750
0	0	512	0	44.718750
0	1	512	0	44.718750
0	0	256	0	44.718750
0	1	256	0	44.718750
0	0	128	0	44.722656
0	1	128	0	44.722656
0	0	64	0	44.754609
0	1	64	0	44.754609
0	0	32	0	44.725586
0	1	32	0	44.725586
0	0	16	0	44.726074
0	1	16	0	44.726074
0	0	8	0	44.726318
0	1	8	0	44.726318
0	0	4	0	44.726318
0	1	4	0	44.726318
0	0	2	0	44.726379
0	1	2	0	44.726379
0	0	1	0	44.726379
0	1	1	0	44.726379
0	0	0	0	44.726379
0	1	0	0	44.726379
0	0	0	1	44.726379

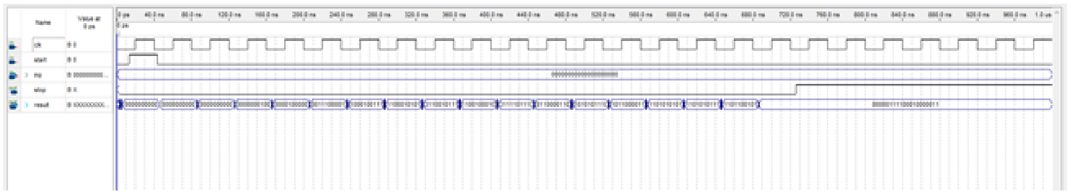

Figure 2. Quartus Prime Timing Diagram of the Entire Square Root RTL System

The result in term of accuracy and error follow the expected values according to number of bits, in this case it is 2^{-16} for 16-bit fraction. The delay also depends on the total number of bit processed times the per clock delay that is mostly due to the squaring process that is around 17ns each.

References

[1] Y. Zhang, D. Chen, B. Liao, D. Guo, and Z. Ke, "Different energy functions leading to different gds models illustrated via square roots computing," in *2013 Chinese Automation Congress*. IEEE, 2013, pp. 403–408.

[2] J. O'Leary, M. Leeser, J. Hickey, and M. Aagaard, "Non-restoring integer square root: A case study in design by principled optimization," in *International Conference on Theorem Provers in Circuit Design*. Springer, 1994, pp. 52–71.

[3] Y. Chandu and M. Megha, "Design and implementation of high efficiency square root circuit using vedic mathematics," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, 2017, pp. 1148–1151.

[4] B. Jayabalan, "Design of CORDIC Based Square Root Calculation," Master's thesis, Department of Electrical & Electronics Engineering, Faculty of Engineering, National Defense University of Malaysia, the Malaysia, 2018.

[5] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on electronic computers*, no. 3, pp. 330–334, 1959.

[6] Square Root Based on CORDIC, "internet source."