



CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Electric Drives and Traction

**Low Abstraction Real-Time FPGA Implementation of Selective Harmonic
Elimination Algorithm for Voltage Source Inverters Designed Using State
of The Art Free and Open Source Software**

Technical report

Petr Zakopal
Prague 2024

TABLE OF CONTENTS

1	Introduction	1
2	Note on the circuit Verilog designs	2
3	Calculating the division of fixed point numbers.....	3
3.1	Newton Rapshon algorithm for calculating the division	3
3.2	IP Block Design	4
3.2.1	Top module design	4
3.2.2	Allocation and Timing	5
3.2.3	Data Path Module	6
3.2.4	Control Unit	7
3.3	Calculating number of bits to shift the denominator.....	8
3.4	Simulation results	8
4	Using CORDIC to calculate trigonometric functions	13
4.1	Theory	13
4.1.1	Example of calculation	16
4.2	Python Implementation	16
4.3	IP Block Design	20
4.3.1	Top module design	20
4.3.2	Allocation and Timing	21
4.3.3	Data Path Module	22
4.3.4	Control Unit	25
4.4	Simulation results	25
5	Simple set of nonlinear equations solved by a Newton-Raphson algorithm using a custom circuit implementation	29
5.1	Theory	29
5.2	IP Block Design	30
5.2.1	Top module design	30
5.2.2	Allocation and Timing	31
5.2.3	Data Path Unit	32
5.2.4	Control Unit	34
5.3	Simulation results	34
6	Selective Harmonic Elimination	36
6.1	Theory	36
6.2	Simplification for Verilog and High level implementation	38
6.3	High level implementation	39
6.4	IP Block Design	41
6.4.1	Algorithm Block Diagram	41
6.4.2	Top module design	43

6.4.3	Allocation and Timing	43
6.4.4	Data Path Unit	45
6.4.5	Control Unit	45
6.4.6	Inverter output voltage analysis for Verilog implementation	46
6.5	Simulation results	47
	Conclusion	50
	References	52
Appendix A	List of and Abbreviations	53
A.1	List of abbreviations	53

LIST OF FIGURES

3 - 1	Top module design for the division unit module block.....	5
3 - 2	Allocation and timing diagram for the Data Path part of the division module.	6
3 - 3	Register Transfer Level (RTL) scheme of the Division module Data Path.....	7
3 - 4	Selected signals from simulation of division $N/D = 10 / 7$. The correct result in R0 is obtained after two iterations (register numberOfIterations).	10
3 - 5	Selected signals from simulation of division $N/D = 1 / 0.25$. The correct result in R0 is obtained after five iterations (register numberOfIterations).	10
3 - 6	Selected signals from simulation of division $N/D = 1 / (-0.25)$. The correct result in R0 is obtained after five iterations (register numberOfIterations).....	11
3 - 7	Selected signals from simulation of division $N/D = 304.03215 / (-0.25)$. The correct result in R0 is obtained after five iterations (register numberOfIterations).	11
3 - 8	Selected signals from simulation of division $N/D = 10 / 519$. The correct result in R0 is obtained after two iterations (register numberOfIterations).	12
4 - 1	Top module design for the CORDIC module.	21
4 - 2	Allocation and timing diagram for the Data Path Unit part of the CORDIC module.	22
4 - 3	Register transfer level (RTL) scheme of the CORDIC module Data Path.....	23
4 - 4	The Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = -1.2479$ rad. The actual scaled value of sine and cosine is calculated every iteration with this algorithm approach. The result is passed to registers R9 and R10.....	26
4 - 5	The detail of the last iteration of the Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = -1.2479$ rad. The actual scaled value of sine and cosine is calculated every iteration with this algorithm approach. The result is passed to registers R9 and R10.....	27
4 - 6	The Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = 10.7195129$ rad. The actual scaled value of sine and cosine is calculated every iteration with this algorithm approach. The result is passed to registers R9 and R10.	27
4 - 7	The Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = -6.7195129$ rad. The value of sinus and cosinus based on the current iteration is also calculated in this algorithm approach. The result is passed to registers R9 and R10.	28
5 - 1	Top module design for the simple Newton-Raphson (NR) calculation module block.	31
5 - 2	Allocation and timing diagram for the Data Path of the simple Newton-Raphson (NR) module.	32
5 - 3	Register Transfer Level (RTL) scheme of the Data Path part of the simple Newton-Raphson (NR) calculation module.....	33
5 - 4	The complete Verilog simulation of a simple Newton-Raphson (NR) algorithm. The result may be seen in registers R1 and R2 after the fifth iteration of the algorithm.	35
6 - 1	Six-Step voltage waveform and harmonic analysis.	36
6 - 2	Block Diagram of the Selective Harmonic Elimination (SHE) algorithm using Newton-Raphson algorithm (NR). Design suitable for hardware implementation.	42
6 - 3	Top module design for the Selective Harmonic Elimination (SHE) module.	43

6 - 4	Allocation and Timing diagram for the Data Path part of Selective Harmonic Elimination (SHE) module.	44
6 - 5	Register transfer level (RTL) scheme of the Selective Harmonic Elimination Data Path...	45
6 - 6	Selective Harmonic Elimination voltage waveform and harmonic analysis.	46
6 - 7	Comparison of a Six-Step and Selective Harmonic Elimination voltage waveforms and harmonic analyses.	46
6 - 8	The ending part of the Verilog simulation the Selective Harmonic Elimination (SHE) algorithm. The result are in registers R0 and R1.....	48
6 - 9	The complete Verilog simulation of Selective Harmonic Elimination (SHE) algorithm. The result are in registers R0 and R1.	49

LIST OF TABLES

3 - 1 Control signal encoding table for instructions to be processed by the Division Module. ... 7

4 - 1 Control signal encoding table for instructions to be processed by the CORDIC Module. ... 25

5 - 1 Control signal encoding table for instructions to be processed by the simple Newton-
Raphson (NR) algorithm Module. 34

6 - 1 Control signal encoding table for instructions to be processed by the Selective Harmonic
Elimination (SHE) algorithm Module. 46

1 Introduction

This paper presents the design of multiple FPGA units, which are designed to suit near real-time constraints of controlling the electric drives or for Hardware In Loop systems.

The goal of this paper also was to investigate how to design the speed optimized units using open source toolchain. The final designed unit is capable of solving the Selective Harmonic Elimination (SHE) algorithm. Many researches opt for proprietary design software, which very often offers premade Intellectual Property (IP) blocks, which can be used to design the specified circuit. However in this paper the design was created, tested and analyzed solely using the State of The Art Open Source software without any IP catalogs. This platformless solution ensures, that the designed units may possibly be synthesized for various FPGA chips without any major barriers.

The structure of the paper is as follows: Section 3 presents a unit for division of two arbitrary values by utilizing the Newton-Raphson (NR) algorithm. Section 4 presents design of the Coordinate Rotation Digital Computer (SHE) optimized for speed, rather than lesser complexity. Section 5 introduces design which solves two non-linear equations with a Newton-Raphson (NR) algorithm, presenting suitability of FPGA designs for iterative algorithms. Section 6 presents unit for solving the Selective Harmonic Elimination problem using previously developed modules.

2 Note on the circuit Verilog designs

All of the designs presented in this paper are created using pure Verilog code and tested through Free and Open-Source Software (FOSS). The decision to opt for FOSS was deliberate, aiming to prevent any vendor-locking to a specific hardware or predefined IPs. Predefined IPs are often optimized by a specific hardware vendor and intended for use with that vendor's hardware. However, the hardware may not always be available or suitable for the application.

Once the design and algorithm are thoroughly understood, they can be implemented without any specific platform in mind. Later, when selecting the device vendor, the design can be modified to suit the specific hardware requirements. That is why Verilog, with Cocotb [1] (Test Bench creation tool) and Verilator [2] (simulator) were used for designing the circuits presented in this paper.

3 Calculating the division of fixed point numbers

Typically, when employing numerical methods to solve transcendental equations, the calculation of the division of two input numbers becomes necessary. This requirement persists even when applying the Newton-Raphson (NR) method to solve a set of two equations, because computing the reciprocal value of the Jacobian determinant is necessary.

There are some IP blocks available, which are capable of calculating the division of two numbers, but the blocks are usually either vendor specific intellectual property (IP) [3] or feature low performance [4].

The drawback of vendor-specific IPs lies in their limited compatibility, often preventing their use with FPGA chips from different vendors. On the other hand the vendor specific IPs are usually optimized and able to use the specific type of resources available at the vendor's chip which resolve in better performance.

To preserve the compatibility of the design with chips from multiple vendors, the custom solution for division design based on the very well known Newton Raphson (NR) algorithm was developed. [4]

3.1 Newton Rapshon algorithm for calculating the division

General Newton Raphson (NR) algorithm is a well known approach to numerically solve equations. It is the reason why it is utilized in many algorithms. However, the negative aspect of NR is that it's convergency strongly depends on initial values of variables. When the initial values are chosen poorly, the performed number of iterations before the convergency is reached can be high.

To reach the fastest convergency possible (determined in number of iterations) apart from the scaling the dominator into the interval [0.5,1] the initial value calculation formula should be utilized. [4]

The Equation 3 - 1 for calculating the initial value is applied after the scaling of denominator is performed. The algorithm developed for the appropriate scaling is explained in the *Calculating number of bits to shift the denominator*.

$$x_0 = \frac{48}{17} - \frac{32}{17}D, \quad (3 - 1)$$

where the x_0 is the initial value for the NR algorithm, D is the denominator value for calculating the expression N/D , where N is the numerator.

Because in the module design implemented via Verilog the fixed point number format $Q32.15$ is used, the fractional numbers from Equation 3 - 1 are rounded to

2.8229 (32'sb000000000000000010_110100101011000 in binary)

and 1.8819 (32'sb000000000000000001_111000011100101 in binary) respectively.

After the initial value x_0 is calculated, the NR algorithm is performed. The idea of using NR algorithm to calculate the division of N/D is to trade the division for a multiplication which can be synthetized in the FPGA fabric. When employing the NR algorithm for finding the values of N/D the function with root is $1/D$ is essential. After the root of the function is found, it is then multiplied by the numerator value, and the solution N/D is obtained. There may be many functions, which root is the searched value $1/D$ but the most trivial is Equation 3 - 2.

$$F(x_i) = \frac{1}{x_i} - D. \quad (3 - 2)$$

For the derivative at the point of x_i then applies Equation 3 - 3.

$$\frac{dF(x_i)}{dx} = F'(x_i) = \frac{F(x_{i+1}) - F(x_i)}{x_{i+1} - x_i}. \quad (3 - 3)$$

Because finding root of the Equation 3 - 2, the value of $F(x_{i+1})$ is set to be zero. After separating the x_{i+1} value of the Equation 3 - 3 and derivating the function $F(x_i)$ the obtained algorithm for a value x_{i+1} is obtained from eq. 3 - 4.

$$x_{i+1} = -\frac{F(x_i)}{F'(x_i)} + x_i = -\frac{F(x_i)}{-\frac{1}{x_i^2}} + x_i = (\frac{1}{x_i} - D)x_i^2 + x_i = x_i - Dx_i^2 + x_i = 2x_i - Dx_i^2. \quad (3 - 4)$$

Usually, the iterative algorithm is stopped, when the value $F(x_{i+1}) - F(x_i)$ (called the defect) reaches certain value set by the stop condition. However, in this algorithm, the stop condition is not yet implemented. Based on the observation carried on the NR algorithm the obtained result is sufficient after 5 iterations.

The mathematically expressed algorithm is then transformed into programmable algorithm suitable for FPGA implementation. The top module design for this algorithm are presented in the section *Top module design*, the control and data unit for calculating the value x_{i+1} is presented in the *Allocation and Timing*

3.2 IP Block Design

The design of this unit consists of 4 main modules:

- the **data unit module**, used for manipulating data and making calculation operations,
- the **control unit module**, used for controlling the **data unit module** and **scaling unit module**,
- **scaling unit module**, used for calculating the number of bits needed for shifting the denominator value to the interval $[0.5, 1]$.

3.2.1 Top module design

The top module wraps all of the presented modules (**data unit module**, **control unit module**, **scaling unit module**). The basic structure of connected modules of this top design is depicted in the Figure 3 - 1. Thanks to this wrapper it is possible to test the created modules with Verilog Testbench, Verilator [2] or Cocotb [1].



Figure 3 - 1 Top module design for the division unit module block

3.2.2 Allocation and Timing

The diagram of the data flow and timing of the algorithm is displayed in the Figure 3 - 2.

The complete algorithm comprises nine steps. The initial four steps are used for calculating the initial value of x_0 as presented in the Equation 3 - 1. The steps $S4$ to $S8$ are for calculating the next search value of x_{i+1} , thus the root of the Equation 3 - 2 which in fact is the searched value of $1/D$. The following iteration begins at the step labeled as $S5$. The iterative process continues until a predefined stop condition is satisfied, such as reaching a specified number of iterations.

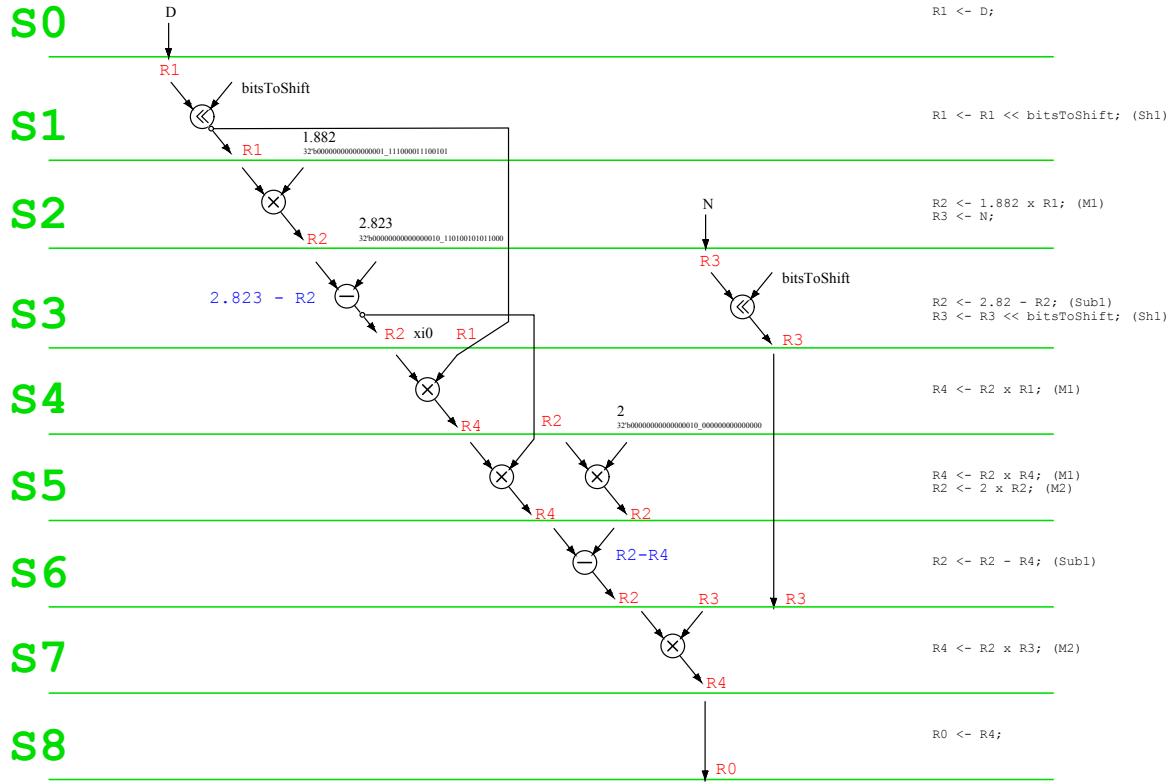


Figure 3 - 2 Allocation and timing diagram for the Data Path part of the division module.

3.2.3 Data Path Module

The structure of the Data Path Module is depicted in the Figure 3 - 3. The module was specifically designed to serve the needs of the division algorithm. It comprises five registers labeled $R0$ through $R4$, two multipliers $M1$, $M2$ and one bit shifter.

The module is controlled by the control unit using the control signal labeled as CS . The encoding table with the labels corresponding to the Data Path Unit module is presented in the section *Control Unit*.

The result of each iteration from the division algorithm is passed to a register $R0$.

The Data Path Module unit also covers the possibility of using negative denominator and numerator. Because the values are stored in a custom $Q32.15$ fixed point format (whole number comprises of 32 bits, 15 bits fractional part, 17 bits integer part), the algorithm checks if the D or N values are higher than $0h8000$ and determine it's actual sign and the sets sign of the result. If the analyzed number is determined negative, it is transformed to value positive and then used in the presented division algorithm. This transformation is needed because of the algorithm calculating the bits to shift the denominator in the interval.

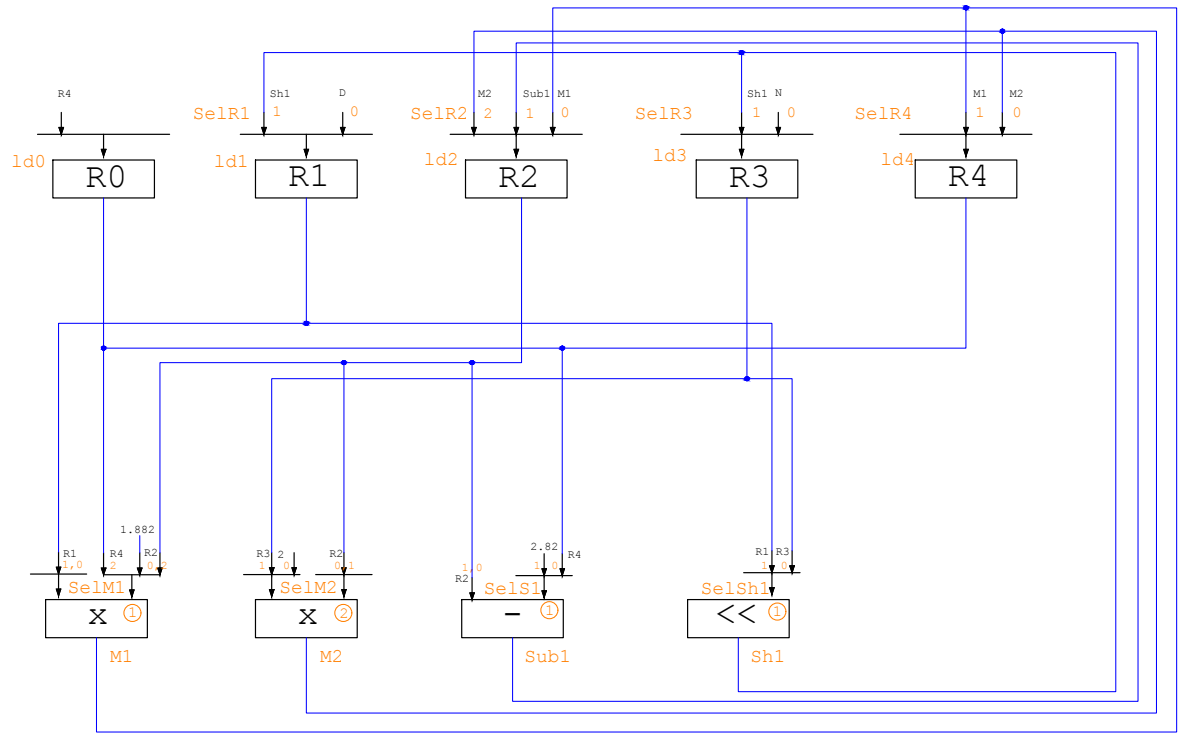


Figure 3 - 3 Register Transfer Level (RTL) scheme of the Division module Data Path.

3.2.4 Control Unit

Signals from Control Unit to Data Path Module are encoded in the CS signal. Table 3 - 1 displays the CS signal along with the corresponding instructions for steps $S0$ – $S8$ of the FSM. To enhance code clarity the signal is defined in the Control Unit in the hexadecimal format.

The number of the iteration of the Finite State Machine (FSM) is also set in the Control Unit. This iteration number is subsequently used in the module to check for the stop condition of the calculation loop.

As stated in the *Allocation and Timing* section, after the step $S8$, the FSM restarts at the state $S4$ with new x_i values as inputs. This state change is not depicted in the Table 3 - 1 for CS signal.

Table 3 - 1 Control signal encoding table for instructions to be processed by the Division Module.

State	RTL Code	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS
		ld0	ld1	ld2	ld3	ld4	SelR1	SelR2[1]	SelR2[0]	SelR3	SelR4	SelSh1	SelM1[1]	SelM1[0]	SelM2	SelS1	
S0	$R1 \leftarrow D;$	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000h
S1	$R1 \leftarrow R1 \ll 32; (Sh1)$	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	15'h2210
S2	$R2 \leftarrow 1.882 \times R1; (M1)$ $R3 \leftarrow N;$	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	15'h1804
S3	$R2 \leftarrow 2.82 - R2; (Sub1)$ $R3 \leftarrow R3 \ll 32; (Sh1)$	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	15'h18C0
S4	$R4 \leftarrow R2 \times R1; (M1)$	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	420h
S5	$R4 \leftarrow R2 \times R4; (M1)$ $R2 \leftarrow 2 \times R2; (M2)$	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	15'h1528
S6	$R2 \leftarrow R2 - R4; (S1)$	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	15'h1081
S7	$R4 \leftarrow R2 \times R3; (M2)$	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	15'h402
S8	$R0 \leftarrow R4;$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000h

3.3 Calculating number of bits to shift the denominator

As presented in the section *Newton Rapshon algorithm for calculating the division* the denominator must be appropriately scaled for the division algorithm to work. This section presents algorithm for scaling the denominator specified in the fixed point number format *Q32.15*. After the scaling value is successfully determined, the numerator is scaled accordingly.

The presented algorithm shifts the value of denominator at every positive edge of the clock signal and saves the shifted value in the `compare` register. Then the combinational circuit is utilized to compare the shifted value in `compare` register with the number 1 specified in *Q32.15* format. If the compared value is the same or lower than 1 the shifting algorithm is done and the value `scaleToShift` is successfully found. If not, the inner value of shifting bits is incremented and the algorithm proceeds to the next iteration.

The presented algorithm is realized in the *denominatorSizeScaleUnit* module and it's pseudocode is depicted in the code 3 - 1.

```
1 at every negative edge of clock or positive edge of reset
2   if(rst)
3       scaleToShift = 0;
4       scaleToShiftInternal = 1;
5       started = 0;
6   end if
7   else if (start)
8       started = 1;
9   end else if
10
11 at every positive edge of clock
12   if (compare <= 32'b000000000000000001_0000000000000000)
13       done = 1;
14       started = 0;
15       scaleToShift = scaleToShiftInternal;
16   end if
17   else
18       done = 0;
19       scaleToShiftInternal = scaleToShiftInternal + 1;
20   end if
21
22 at every positive edge of clock
23   if(start)
24       compare <= DInternal >> scaleToShiftInternal;
25   end if
```

Code 3 - 1 Pseudocode for the *denominatorSizeScaleUnit* module algorithm.

3.4 Simulation results

The simulation via Verilog testbench was made to determine the correctness of presented division module. The Icarus Verilog simulator was used to simulate the module and GTKWave was used to display the VCD simulation output file.

The simulation output confirms that the module operates correctly for positive and negative numbers

in the fixed-point format $Q32.15$. The algorithm used in this module can compute the correct result in significantly fewer clock cycles compared to the full division algorithm utilized in the division module within the package [4]. As a result, the module can be freely used as a submodule in more complex modules.

The rendered VCD simulation output waveforms are depicted on the following Figures. The simulations were conducted for arbitrarily selected values of numerator N and denominator D , with clock frequency set to 250 MHz. Pseudocode Verilog snippet for the test bench is provided in the Listing 3 - 2. In the test bench, one unit of time corresponds to 1 ns. (based on the set timescale settings) The division unit algorithm starts at the next positive edge of clock signal after successful determination of the value *bitsToShift*.

```

1  timescale 1ns/1ns
2  #10; // wait for 10 units of time
3  #0 rstScale = 1; startScale = 0; // reset unit for determining the
   number of bits to shift in the denominator and do not start the unit yet
4  N = 32'b000000000100110000_0000100000000000; D=32'
   b1111111111111111_1100000000000000; // set the numerator to N =
   304.03125, denominator to D = -0.25
5  #10 rstScale = 0; // wait for 10 units of time and stop the reset of
   scaling unit
6  #10 startScale = 1; // start the algorithm for scaling unit
7  #20 rst = 1; start = 0; // reset the division unit
8  #30 rst = 0; // stop resetting of the division unit
9  #20 start = 1; // start the division unit
10 #20 start = 0;
11 #1000; // wait 1000 units of time
12 $finish; // finish the simulation

```

Code 3 - 2 Pseudocode snippet for the Verilog simulation test bench.



Figure 3 - 4 Selected signals from simulation of division $N/D = 10 / 7$. The correct result in $R0$ is obtained after two iterations (register numberOfIterations).

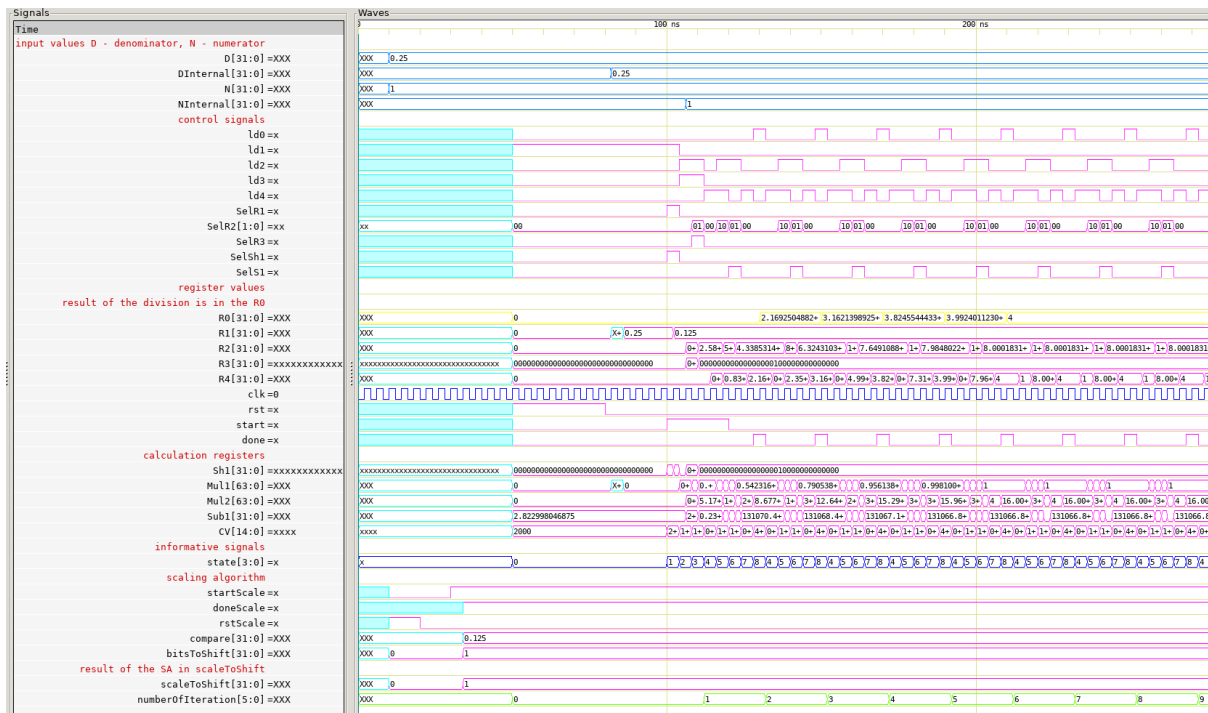


Figure 3 - 5 Selected signals from simulation of division $N/D = 1 / 0.25$. The correct result in $R0$ is obtained after five iterations (register numberOfIterations).



Figure 3 - 8 Selected signals from simulation of division $N/D = 10 / 519$. The correct result in R0 is obtained after two iterations (register numberOfIterations).

4 Using CORDIC to calculate trigonometric functions

There are numerous methods calculating trigonometric functions. To enhance flexibility of the design, the Coordinate Rotation Digital Computer (CORDIC) was selected over the Look-Up Table (LUT) implementation.

While the LUT method may be fast, its accuracy depends on the size of the table. In contrast, when using the CORDIC the precision depends on number of performed iterations of the algorithm. The modified algorithm is versatile and may be used to calculate non-trivial functions, including hyperbolic functions, square roots, multiplications, divisions, exponentials and logarithms. [5]

In this work only the calculation of *sine* and *cosine* functions is performed.

4.1 Theory

The theory of the first CORDIC was introduced by Volder in [6]. This algorithm computes a coordinate conversion between rectangular (x, y) and polar (R, θ) coordinates. The algorithm was then extended by Walther in [7] to include circular, linear and hyperbolic transforms. In this paper, only circular transforms are employed to calculate *sine* and *cosine* functions. The presentation will focus on the fundamental aspects of the algorithm.

The rotation of a vector in the rectangular coordinate system (x, y) may be described by matrix-vector multiplication depicted in the Equation 4 - 1.

$$\begin{pmatrix} x_R \\ y_R \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_{in} \\ y_{in} \end{pmatrix}, \quad (4 - 1)$$

where x_R and y_R are coordinates of a rotated vector, θ is the angle for which the vector with coordinates x_{in} and y_{in} is rotated.

Then when simplifying the Equation 4 - 1

$$\begin{pmatrix} x_R \\ y_R \end{pmatrix} = \cos(\theta) \begin{pmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{pmatrix} \begin{pmatrix} x_{in} \\ y_{in} \end{pmatrix} \quad (4 - 2)$$

it can be seen, that only multiplication by scaling factor of precalculated values of $\cos(\theta)$, multiplication by $\tan(\theta)$, subtraction and addition operations are needed to perform the rotation. However, the multiplication by $\tan(\theta)$ can be replaced. The replacement may be done for angles θ for which the Equation 4 - 3 is true. When implementing the algorithm to the FPGA the multiplication may be swapped for signed right bit shift, which is faster operation than multiplication.

$$\tan(\theta) = 2^{-1}. \quad (4 - 3)$$

When the initial values $x_{in} = 1$ and $y_{in} = 0$ are used, the result for *sine* and *cosine* may be easily obtained from x_R and y_R as expressed in the Equation 4 - 4.

$$\begin{aligned} x_R &= x_{in} \cos(\theta) - y_{in} \sin(\theta) = \cos(\theta), \\ y_R &= x_{in} \sin(\theta) + y_{in} \cos(\theta) = \sin(\theta). \end{aligned} \quad (4 - 4)$$

The algorithm can be further simplified by assuming that it is designed to undergo more than 6 iterations and thus the scaling constant, represented by multiplying *cosine* of different θ values, converges to the value 0.60725. If this condition is true, there is no necessity to precalculate all the scaling values and

only the convergent value may be used for the multiplication. In this paper the precalculated scaling values are passed from the custom LUT module to the main algorithm.

As evident from the *Example of calculation* section or the algorithm theory itself, it is essential to establish whether the angle for which the vector is rotated in the next iteration should be in a positive direction (counter-clockwise) or negative direction (clockwise). To address this, the set of the equations is expanded, and new variable z_i is introduced. The complete set of equations utilized in the implementation is as follows:

$$\begin{aligned} x[i+1] &= x[i] - \sigma_i 2^{-i} y[i], \\ y[i+1] &= y[i] + \sigma_i 2^{-i} x[i], \\ z[i+1] &= z[i] - \sigma_i \operatorname{atan}(2^{-i}). \end{aligned} \quad (4-5)$$

The σ_{i+1} is determined based on the sign of the z_{i+1} variable

$$\sigma_{i+1} = \begin{cases} -1, & \text{if } z_{i+1} < 0 \\ 1, & \text{if } z_{i+1} > 0 \\ 0, & \text{if } z_{i+1} = 0 \end{cases} \quad (4-6)$$

The algorithm, as presented, accurately computes values for *sine* and *cosine* functions only in the first and fourth quadrants ($-\pi/2$ to $\pi/2$ counter-clockwise). To expand its applicability across the entire 2π range, specific actions must be taken before the actual looped algorithm.

The algorithm must determine the quadrant, where the desired angle θ for which the *sine* and *cosine* functions are to be calculated is. This determination is made through `if` statements during the initialization of the algorithm values and at the final value calculation. If the reference angle θ falls outside the first or fourth quadrant, then the angle is rotated from its original quadrant to either the first or fourth quadrant. Depending on the quadrant, to which the angle is rotated, the σ_i value is set accordingly. The corresponding `if` statements during the algorithm initialization are provided in Pseudocode 4 - 1. Similar statements used at the final values calculation are presented in Pseudocode 4 - 2.

The pseudocodes use *initialZValue* as a reference angle θ , for which to calculate the *sine* and *cosine* function values, *zValue* as a temporary value for calculating the iterations for z_i variables, *sigmaValue* for temporary value, which holds the current iteration value of σ_i , the *resultCos* and *resultSin* variables are used for storing the temporary and final values of the $\cos(\theta)$ and $\sin(\theta)$ values respectively.

```

1 if (initialZValueCordic > 1.5707) and (initialZValueCordic < 3.141592):
2     zValue = initialZValueCordic - 3.141592
3     sigmaValue = -1
4     print("value in second q")
5     print("zValue:", zValue)
6 elif (initialZValueCordic > 3.141592) and (initialZValueCordic < 4.7123):
7     zValue = initialZValueCordic - 3.141592
8     sigmaValue = 1
9     print("value in third q")
10    print("zValue:", zValue)
11 elif (initialZValueCordic < 0) and (initialZValueCordic > - 1.5707):
12     sigmaValue = -1
13     zValue = initialZValueCordic

```

```

14     print("value in fourth q")
15     print("zValue:", zValue)
16 elif (initialZValueCordic < -1.5707) and (initialZValueCordic > - 3.141592)
17 :
18     sigmaValue = 1
19     zValue = initialZValueCordic + 3.141592
20     print("value in third q")
21     print("zValue:", zValue)
22 elif (initialZValueCordic < - 3.141592) and (initialZValueCordic > - 4.7129
23 ):
24     sigmaValue = - 1
25     zValue = initialZValueCordic + 3.141592
26     print("value in second q")
27     print("zValue:", zValue)
28 elif (initialZValueCordic < - 4.7129) and (initialZValueCordic > - 6.28318)
29 :
30     sigmaValue = initialSigmaValueCordic
31     zValue = initialZValueCordic + 2*3.141592
32     print("value in first q")
33     print("zValue:", zValue)
34 elif (initialZValueCordic > 4.7123) and (initialZValueCordic < 6.28318):
35     sigmaValue = -1
36     zValue = initialZValueCordic - 2*3.141592
37     print("value in fourth q")
38     print("zValue:", zValue)
39 else:
40     zValue = initialZValueCordic # For angle
41     sigmaValue = initialSigmaValueCordic # For +- next angle
42     print("value in first")
43     print("zValue:", zValue)

```

Code 4 - 1 Pseudocode for if statements used at the value initialization of the CORDIC algorithm.

```

1 if (initialZValueCordic > 1.5707) and (initialZValueCordic < 3.141592):
2     resultCos = - resultCos
3     resultSin = - resultSin
4 elif (initialZValueCordic > 3.141592) and (initialZValueCordic < 4.7123):
5     resultCos = - resultCos
6     resultSin = - resultSin
7 elif (initialZValueCordic < 0) and (initialZValueCordic > - 1.5707):
8     resultCos = resultCos
9     resultSin = resultSin
10 elif (initialZValueCordic < -1.5707) and (initialZValueCordic > - 3.141592)
11 :
12     resultCos = -resultCos
13     resultSin = -resultSin
14 elif (initialZValueCordic < - 3.141592) and (initialZValueCordic > - 4.7129
15 ):
16     resultCos = - resultCos

```

```

15     resultSin = - resultSin
16 elif (initialZValueCordic < - 4.7129) and (initialZValueCordic > -
    2*3.141592):
17     resultCos = resultCos
18     resultSin = resultSin

```

Code 4 - 2 Pseudocode for if statements used at the final sinus and cosinus value calculation.

4.1.1 Example of calculation

The CORDIC algorithm's general approach can be illustrated by calculating the *sine* and *cosine* values for the reference angle $\theta = 57,535^\circ$ (1.0041 rad). Initially, the angle is decomposed into its base angles, satisfying the Equation 4 - 3. In this example the decomposition is $57,535 = 45 + 25,565 - 14,03$.

The index i of the variables x_i and y_i in the following equations means the number of iteration of the algorithm.

$$0. \text{ iteration } \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \cos(45^\circ) \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_{in} \\ y_{in} \end{pmatrix}, \quad (4 - 7)$$

$$1. \text{ iteration } \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \cos(25,565^\circ) \begin{pmatrix} 1 & -2^{-1} \\ 2^{-1} & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}, \quad (4 - 8)$$

$$2. \text{ iteration } \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \cos(-14,03^\circ) \begin{pmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}. \quad (4 - 9)$$

Then values x_2 and y_2 may be obtained.

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \cos(45^\circ) \cos(25,565^\circ) \cos(-14,03^\circ) \begin{pmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{pmatrix} \begin{pmatrix} 1 & -2^{-1} \\ 2^{-1} & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_{in} \\ y_{in} \end{pmatrix}. \quad (4 - 10)$$

The values x_2 and y_2 in the Equation 4 - 10 correspond to $\cos(57,535^\circ)$ and $\sin(57,535^\circ)$ respectively.

4.2 Python Implementation

For simplicity, the CORDIC algorithm was prototyped in Python. This proved highly beneficial, as the debugging of the Python code is much more straightforward compared to debugging the Verilog design without prepared and debugged algorithm in a higher level language.

The Python code was used to precalculate the LUT for scaling factor and *arcus tangens* values for z_i calculations.

For clarity, the Python implementation is provided in Code 4 - 3. The presented Code also calculates the error between the CORDIC-calculated value and the Python math library functions.

```

1 import math
2
3
4 # Defining starting values and empty arrays
5 totalNumberOfIterations = 12 # 12 - best tradeof between value and
    iterations

```



```

6 atanValues = []
7 scalingValues = [1]
8 initialXValueCordic = 1
9 initialYValueCordic = 0
10 # initialZValueCordic = 1.248 # angle for which to calculate cordic
11 # initialZValueCordic = - 1.248 # angle for which to calculate cordic
12 # initialZValueCordic = - 6.7194 # angle for which to calculate cordic
13 # initialZValueCordic = 6.7194 # angle for which to calculate cordic
14 # initialZValueCordic = 5.8 # angle for which to calculate cordic
15 # initialZValueCordic = 10.7194824 # angle for which to calculate cordic
16 # initialZValueCordic = - 10.7194824 # angle for which to calculate cordic
17 # initialZValueCordic = 5.8 # angle for which to calculate cordic
18 # initialZValueCordic = - 5 # angle for which to calculate cordic
19 # initialZValueCordic = - 5 # angle for which to calculate cordic
20 # initialZValueCordic = - 20.3948 # angle for which to calculate cordic
21 # initialZValueCordic = - 1.8 # angle for which to calculate cordic
22 # initialZValueCordic = 1.6 # angle for which to calculate cordic
23 # initialZValueCordic = 1.8 # angle for which to calculate cordic
24 # initialZValueCordic = 3.5 # angle for which to calculate cordic
25 initialZValueCordic = - 3.5 # angle for which to calculate cordic
26 initialSigmaValueCordic = 1
27
28 for x in range(totalNumberOfIterations):
29     # Generating arcus tangens values of precalculated angles based on
    number of iterations
30     atanValues.append(math.atan(1*2**(-x)))
31     # Generating precalculated scaling values based on a number of
    iterations
32     scalingValues.append(scalingValues[x]*math.cos(atanValues[x]))
33
34 print("atanValues: ", atanValues)
35 print("scalingValues: ", scalingValues)
36
37 print("*-+-+-+*")
38 print("\n")
39 print("initialZValue original: ", initialZValueCordic)
40
41 # Moving angle to interval [0,2Pi]
42 if initialZValueCordic > 0:
43     while initialZValueCordic > (2*3.141592):
44         initialZValueCordic = initialZValueCordic - 2*3.141592
45 else:
46     while initialZValueCordic < (-2*3.141592):
47         initialZValueCordic = initialZValueCordic + 2*3.141592
48
49
50 print("initialZValue after moving to [0,2Pi] interval: ",
    initialZValueCordic)

```



```

96 # Passing starting values to the calculation values
97 xValue = initialXValueCordic # For cos
98 yValue = initialYValueCordic # For sin
99
100
101 # CORDIC ALGORITHM
102 for x in range(totalNumberOfIterations):
103
104     # Calculating next values of the current iteration x
105     xNextValue = xValue - (sigmaValue*yValue)*2**(-x)
106     yNextValue = yValue + (sigmaValue*xValue)*2**(-x)
107     zNextValue = zValue - sigmaValue * atanValues[x]
108
109     # Determining the signum of next angle (addition or subtraction)
110     if zNextValue >= 0:
111         sigmaNextValue = 1
112     else:
113         sigmaNextValue = -1
114
115     # Values for new iteration
116     xValue = xNextValue
117     yValue = yNextValue
118     zValue = zNextValue
119     sigmaValue = sigmaNextValue
120
121     print("iteration:", x, "xValue:", xValue, "yValue:", yValue, "zValue:",
122           zValue, "sigmaValue:", sigmaValue, "\n")
123
124 # Calculating results by scaling the result values from CORDIC by the
125 # scalingValue which depends on number of iterations which were made
126
127 resultCos = scalingValues[x-1] * xValue
128 resultSin = scalingValues[x-1] * yValue
129
130 # Changing results sign based on the rotation of the initialZValueCordic
131 if (initialZValueCordic > 1.5707) and (initialZValueCordic < 3.141592):
132     resultCos = - resultCos
133     resultSin = - resultSin
134 elif (initialZValueCordic > 3.141592) and (initialZValueCordic < 4.7123):
135     resultCos = - resultCos
136     resultSin = - resultSin
137 elif (initialZValueCordic < 0) and (initialZValueCordic > - 1.5707):
138     resultCos = resultCos
139     resultSin = resultSin
140 elif (initialZValueCordic < -1.5707) and (initialZValueCordic > -
141       3.141592):
142     resultCos = -resultCos
143     resultSin = -resultSin
144 elif (initialZValueCordic < - 3.141592) and (initialZValueCordic > -

```

```

    4.7129):
141     resultCos = - resultCos
142     resultSin = - resultSin
143 elif (initialZValueCordic < - 4.7129) and (initialZValueCordic > -
    2*3.141592):
144     resultCos = resultCos
145     resultSin = resultSin
146
147 # Calculating values based on the math library
148 mathResultCos = math.cos(initialZValueCordic)
149 mathResultSin = math.sin(initialZValueCordic)
150
151 # Calculating the error of CORDIC calculated values from the python math
    functions
152 errorCos = abs(resultCos) - abs(mathResultCos)
153 errorSin = abs(resultSin) - abs(mathResultSin)
154
155 # Result printing
156 print("*-+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+*")
157 print("CORDIC results:")
158 print("cos: ", resultCos)
159 print("sin: ", resultSin)
160 print("scaleFactor: ", scalingValues[totalNumberOfIterations-1])
161
162 print("\n")
163
164 print("MATH results:")
165 print("cos: ", mathResultCos)
166 print("sin: ", mathResultSin)

```

Code 4 - 3 Python code of CORDIC implementation.

Once the Python implementation and debugging are completed, the Verilog implementation of the algorithm can safely be created. Similar to the Division Unit module, as presented in *Calculating the division of fixed point numbers* section, the Data Path, Control Unit and Top Module were designed. This application-specific circuit design approach should be faster and safer than creating a custom CPU with reduced and customized ISA for performing the CORDIC algorithm.

4.3 IP Block Design

4.3.1 Top module design

The top module design of the CORDIC IP is illustrated in Figure 4 - 1. As evident, the structure closely resembles that of the Division Unit top module. When using an approach to create a customized circuit for an algorithm, the process of developing the top modules is likely to be similar, only with minor differences in used control signal, inputs and variables.

The Data Path module incorporates precalculated values in LUTs for *atanValues* and *scalingValues*. In this implementation, the value of *totalNumberOfIterations* is set to 12, making the LUT 12x32 bits in size. It is worth noting that the previously introduced custom fixed-point format *Q32.15* is utilized.

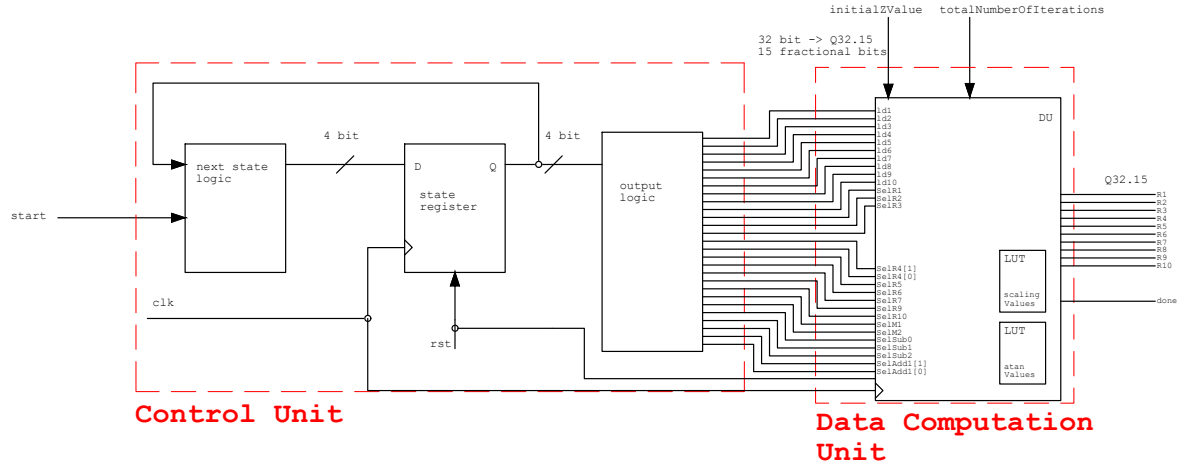


Figure 4 - 1 Top module design for the CORDIC module.

4.3.2 Allocation and Timing

In the Figure 4 - 2, the allocation and timing diagram is depicted. Notably, the `if` statements, implemented in the control unit, are documented within the diagram. The explanation, why the `if` statements are needed, is presented in the *CORDIC Theory* section.

As mentioned in the *CORDIC Control Unit* sections, there are two primary approaches to iteration cycles. The one is to proceed from *S4* to *S2* for a faster algorithm, while the other involves progressing from *S6* to *S2*. The latter approach is employed for demonstrative purposes, as it ensures that the final numerical values are always calculated.

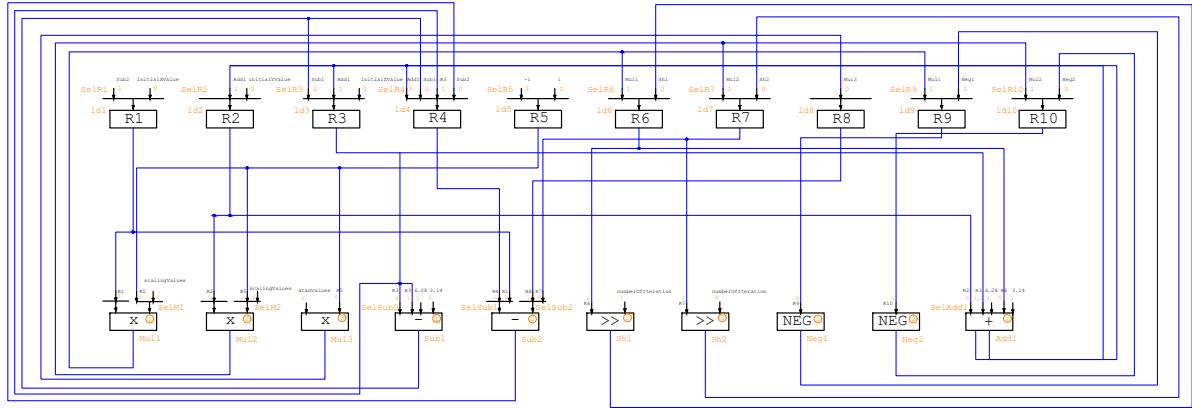


Figure 4 - 3 Register transfer level (RTL) scheme of the CORDIC module Data Path.

```

1 module atanValuesCordicLUT(index, returnValue);
2
3 input [3:0] index;
4 output reg signed [31:0] returnValue;
5
6
7 always@(index)
8 begin
9     case(index)
10         4'b0000: returnValue = 32'sb00000000000000000000_110010010000111; //
11             0.7853981633974483
12         4'b0001: returnValue = 32'sb00000000000000000000_011101101011000; //
13             0.4636476090008061
14         4'b0010: returnValue = 32'sb00000000000000000000_001111101011011; //
15             0.24497866312686414
16         4'b0011: returnValue = 32'sb00000000000000000000_000111111101010; //
17             0.12435499454676144
18         4'b0100: returnValue = 32'sb00000000000000000000_000011111111101; //
19             0.06241880999595735
20         4'b0101: returnValue = 32'sb00000000000000000000_000001111111111; //
21             0.031239833430268277
22         4'b0110: returnValue = 32'sb00000000000000000000_000000111111111; //
23             0.015623728620476831

```

```

17      4'b0111: returnValue = 32'sb000000000000000000_0000000111111111; //
0.007812341060101111
18      4'b1000: returnValue = 32'sb000000000000000000_0000000111111111; //
0.007812341060101111
19      4'b1001: returnValue = 32'sb000000000000000000_0000000011111111; //
0.0019531225164788188
20      4'b1010: returnValue = 32'sb000000000000000000_0000000001111111; //
0.0009765621895593195
21      4'b1011: returnValue = 32'sb000000000000000000_0000000000111111; //
0.0004882812111948983
22      default: returnValue = 32'sb000000000000000000_0000000000000000; // 0
23  endcase
24 end
25 endmodule

```

Code 4 - 4 Verilog code of the atanValuesCordicLUT lookup table (LUT) implementation.

```

1 module scalingValuesCordicLUT(index, returnValue);
2
3 input [3:0] index;
4 output reg signed [31:0] returnValue;
5
6 always@(index)
7 begin
8     case(index)
9         4'b0000: returnValue <= 32'sb000000000000000001_0000000000000000; //
1          1
10         4'b0001: returnValue <= 32'sb000000000000000000_101101010000010; //
0.7071067811865476
11         4'b0010: returnValue <= 32'sb000000000000000000_101000011110100; //
0.6324555320336759
12         4'b0011: returnValue <= 32'sb000000000000000000_100111010001001; //
0.6135719910778964
13         4'b0100: returnValue <= 32'sb000000000000000000_100110111101110; //
0.6088339125177524
14         4'b0101: returnValue <= 32'sb000000000000000000_100110111000111; //
0.6088339125177524
15         4'b0110: returnValue <= 32'sb000000000000000000_100110110111101; //
0.607351770141296
16         4'b0111: returnValue <= 32'sb000000000000000000_100110110111011; //
0.6072776440935261
17         4'b1000: returnValue <= 32'sb000000000000000000_100110110111010; //
0.6072591122988928
18         4'b1001: returnValue <= 32'sb000000000000000000_100110110111010; //
0.6072544793325625
19         4'b1010: returnValue <= 32'sb000000000000000000_100110110111010; //
0.6072533210898753
20         4'b1011: returnValue <= 32'sb000000000000000000_100110110111010; //
0.6072530315291345

```



```
21         default: returnValue <= 32'sb000000000000000000_0000000000000000; //
           0
22     endcase
23 end
24 endmodule
```

Code 4 - 5 Verilog code of the scalingValuesCordicLUT lookup table (LUT) implementation.

4.3.4 Control Unit

Similarly to the Division *Control Unit* section, the encoding of the control signal is presented in Table 4 - 1.

The branches of `if` statements used in the design have been color-coded to enhance clarity. Steps *S5* and *S6* are mainly focused on multiplying the result of iteration by the appropriate scaling value and on multiplying the calculated values by 1 or -1 based on the quadrant of the original reference angle.

Table 4 - 1 Control signal encoding table for instructions to be processed by the CORDIC Module.

State	RTL Code	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CV
MT	M2	M3	M4	M5	M6	M7	M8	M9	M10	ScR0	ScR1	ScR2	ScR3	ScR4	ScR5	ScR6	ScR7	ScR8	ScR9	ScR10	ScM1	ScM2	ScSub0	ScSub1	ScSub2	ScAdd1	ScAdd1	ScAdd1	ScAdd1	CV
S0	R0 ← totalNumberOfIterations; R1 ← initialValue; R2 ← initialValue; R3 ← initialValue;		1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28'b000000
S1	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1)		0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	28'b208010
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1)		0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	28'b208041
S2	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S2;																													28'b0
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
S3	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Sub1) due → nextState == S3; CS = 0;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b184 (Add1) due → nextState == S1;																													
	if(R3 <= 28'b184) R3 ← R3 - 6.28'b																													

4.4 Simulation results

The testbench for testing the design was developed using Cocotb [1] with the Verilator [2] as a simulator.

During the algorithm implementation process it becomes evident, that the number of cycles required for the final calculation can be determined as

$$NoCyc_{\text{result every iteration}} = \left\{ \begin{array}{l} 3, \text{ if } initialZValue \in [-2\pi, 2\pi] \\ 4, \text{ if } initialZValue \notin [-2\pi, 2\pi] \end{array} \right\} + 5NoIt, \quad (4 - 11)$$

where $NoCyc(-)$ is the number of cycles and $NoIt$ is the number of iterations for the CORDIC algorithm. The 4 value is caused by states $S0-S4$ and the multiplication by 5 is caused by iterating through states $S4-S8$. When the result of the CORDIC algorithm is calculated only once at the end of the algorithm, the number of iterations can be determined by

$$NoCyc_{\text{result at the end}} = \begin{cases} 3, & \text{if } initialZValue \in [-2\pi, 2\pi] \\ 4, & \text{if } initialZValue \notin [-2\pi, 2\pi] \end{cases} + 3NoIt + 2, \quad (4 - 12)$$

where the multiplication by value 3 is caused by iterating through states $S4-S6$, the addition of 4 is caused by states $S0-S4$ and the addition of the 2 is caused by states $S7-S8$ calculating the final result.

In the simulation the *numberOfCycles* displayed is an index of the cycle, so for angle $\theta = -1.247985$ rad is the number of iterations depicted on Figure 4 - 5 is 63.

The frequency of the clock signal in the simulation is currently set to 50 MHz.

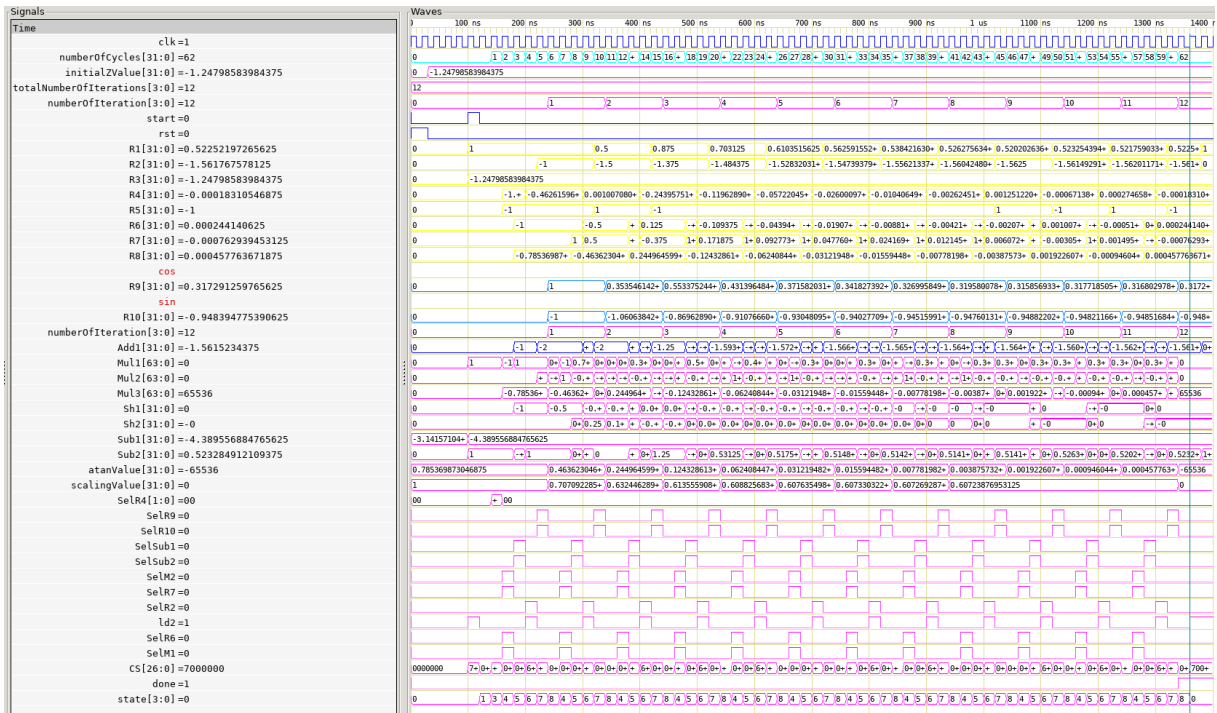


Figure 4 - 4 The Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = -1.2479$ rad. The actual scaled value of sine and cosine is calculated every iteration with this algorithm approach. The result is passed to registers R9 and R10.

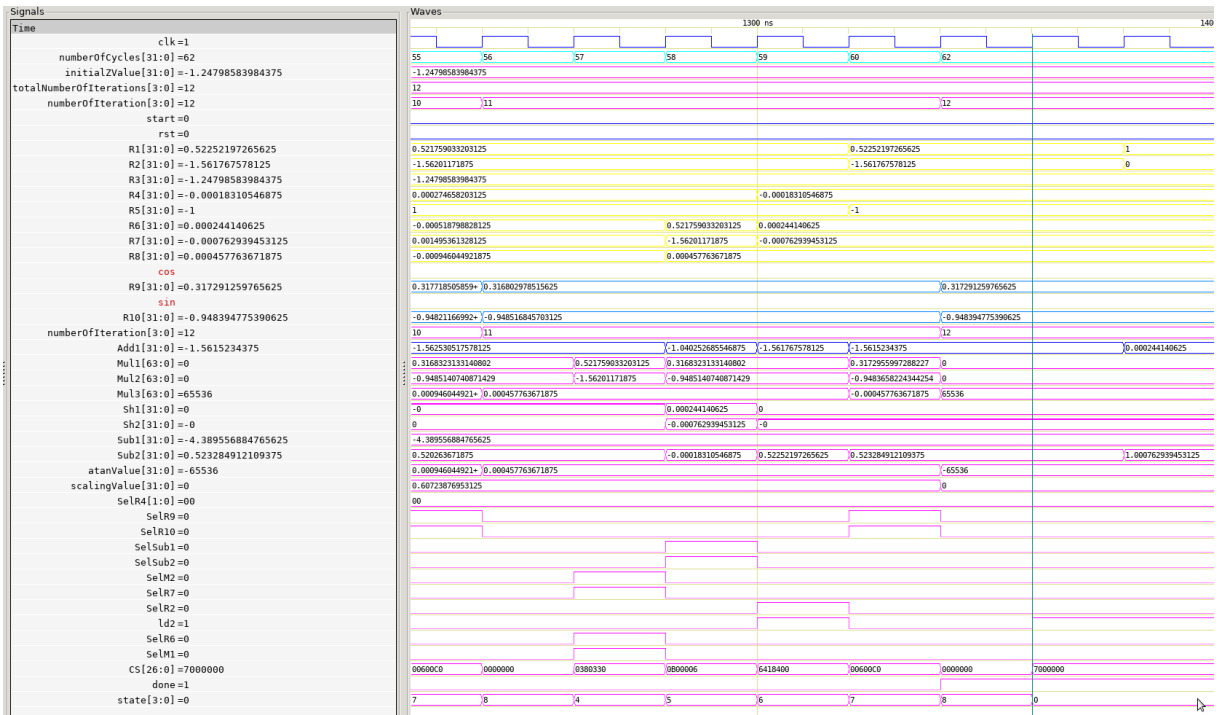


Figure 4 - 5 The detail of the last iteration of the Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = -1.2479$ rad. The actual scaled value of sine and cosine is calculated every iteration with this algorithm approach. The result is passed to registers R9 and R10.

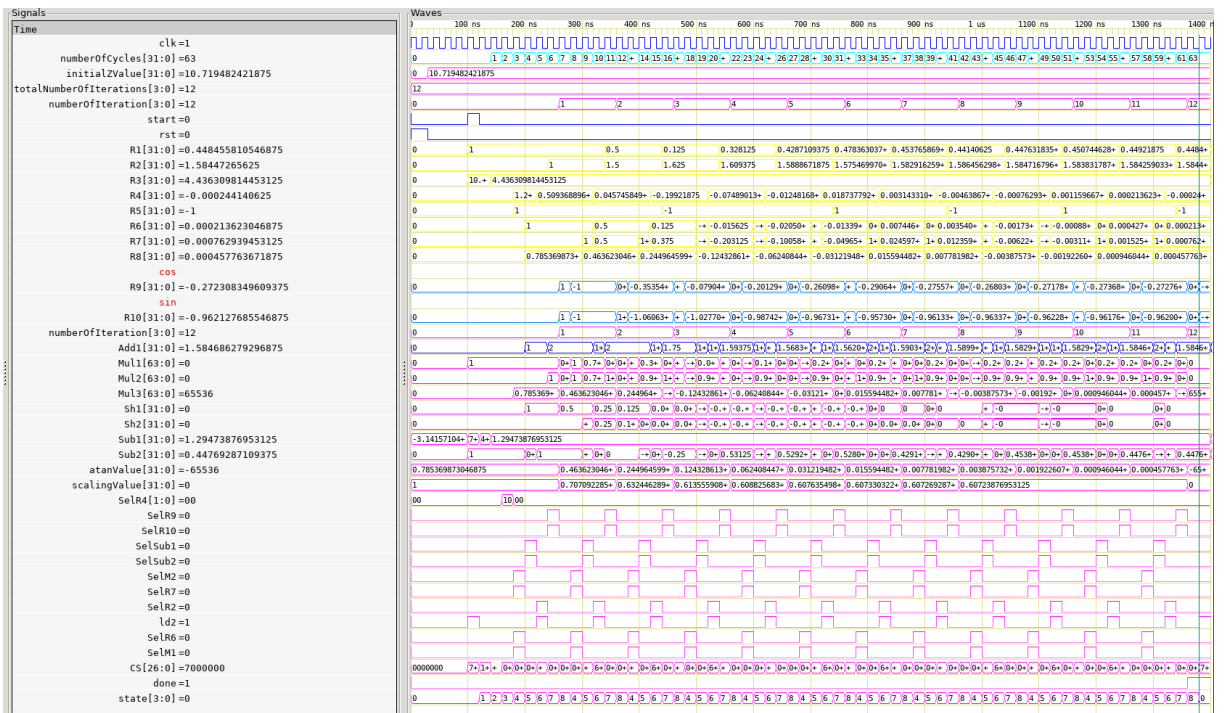


Figure 4 - 6 The Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = 10.7195129$ rad. The actual scaled value of sine and cosine is calculated every iteration with this algorithm approach. The result is passed to registers R9 and R10.

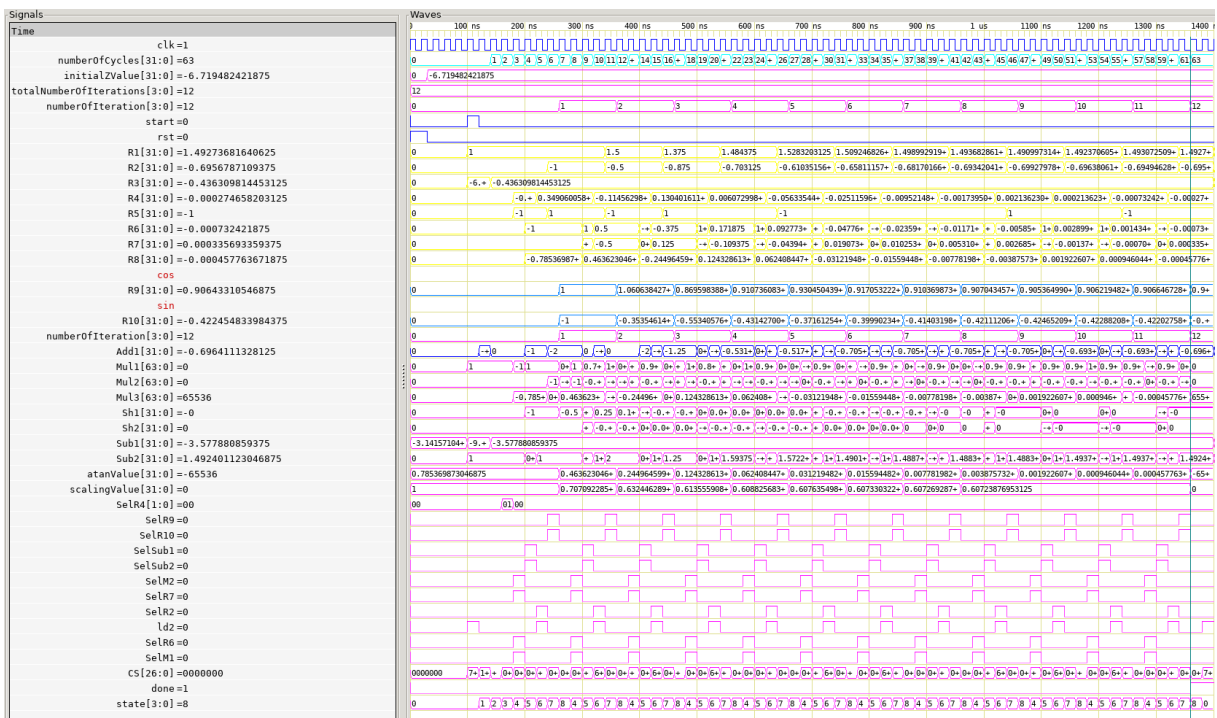


Figure 4 - 7 The Verilog simulation of CORDIC algorithm for determining the sine and cosine values of angle $\theta = -6.7195129$ rad. The value of sinus and cosinus based on the current iteration is also calculated in this algorithm approach. The result is passed to registers R9 and R10.

5 Simple set of nonlinear equations solved by a Newton-Raphson algorithm using a custom circuit implementation

Most of the modules presented in the preceding sections can be utilized as submodules to solve the system of nonlinear equations. Because this work aims to solve the transcendental equations for Selective Harmonic Elimination (SHE), the most effective approach is to initially solve a simpler set of equations to determine the difficulty and viability of the NR.

5.1 Theory

The objective of the NR algorithm is to solve the set of nonlinear equations

$$F_1(x_1, x_2) = x_1^3 - x_2 - 1, \quad (5 - 1)$$

$$F_2(x_1, x_2) = x_1 - 2x_2 - 2, \quad (5 - 2)$$

where one possible set of solutions x_1 and x_2 yields

$$F_1 = 0, \quad (5 - 3)$$

$$F_2 = 0. \quad (5 - 4)$$

The algorithm could have been implemented in a custom CPU with reduced instruction set. However, due to apparent reasons such as speed and complexity associated with developing own processor, chosen approach involved creating an application specific circuit design.

In order to integrate the algorithm into the custom design, the general NR algorithm approach had to be simplified to its most fundamental implementation. Every component that could be precalculated was set as a static value during the design phase.

To check if the implementation and algorithm was well designed, the solution by *Solve* function and a customized NR was made in Wolfram Mathematica.

Before initiating the algorithm, the starting values of x_1^0 and x_2^0 are set as inputs to the module. Based on that input the function values at selected starting points are calculated.

As a next step, the so called defect can be calculated using the newly found values of $F_1(x_1^0, x_2^0)$ and $F_2(x_1^0, x_2^0)$

$$\Delta \mathbf{F}^i = \begin{pmatrix} \Delta F_1^i \\ \Delta F_2^i \end{pmatrix} = \begin{pmatrix} F_1^i - F_1^{\text{known solution}} \\ F_2^i - F_2^{\text{known solution}} \end{pmatrix}, \quad (5 - 5)$$

where the superscript i is the number of iteration for which the defect is calculated. When the algorithm starts, the $i = 0$. So for example the input value for F_1^0 is x_1^0 and x_2^0 .

Next, the Jacobian matrix \mathbf{J} from vector of functions $(F)(x_1, x_2) = (F_1, F_2)$ is calculated as follows.

$$\mathbf{J}^i = \begin{pmatrix} \frac{dF_1}{dx_1^i} & \frac{dF_1}{dx_2^i} \\ \frac{dF_2}{dx_1^i} & \frac{dF_2}{dx_2^i} \end{pmatrix} = \begin{pmatrix} 3(x_1^i)^2 & -1 \\ 1 & -2 \end{pmatrix}. \quad (5 - 6)$$

As for the general NR algorithm, the inverted value Jacobian matrix needs to be calculated. The problem is, that when using general mathematical software, such as Wolfram Mathematica, the calculation of

the inversion is as easy as using function of inversion. When designing the circuit, the approach of manual calculation of inversion must be used. In this paper, the calculation is made possible by calculating the determinant of the Jacobian Matrix, its reciprocal value, adjugate matrix of the Jacobian Matrix and multiplication of the adjugate matrix elements by the calculated determinant reciprocal value.

Because the size of the Jacobian matrix is 2x2 the determinant may be easily calculated using the Sarrus Rule. When the matrix is more complicated, the expansion method may be utilized.

$$\det(\mathbf{J}) = 3(x_1^i)^2(-2) - (-1) = 3(x_1^i)^2(-2) + 1. \quad (5 - 7)$$

The reciprocal value of the determinant is then calculated by the Division Unit, created for calculating division of arbitrary real numbers. This Division Unit is presented in the section *Calculating the division of fixed point numbers*.

The adjugate matrix is calculated as follows

$$\text{adj}(\mathbf{J}) = \begin{pmatrix} \mathbf{J}_{11}(-1)^{1+1} & \mathbf{J}_{01}(-1)^{1+2} \\ \mathbf{J}_{10}(-1)^{1+2} & \mathbf{J}_{00}(-1)^{2+2} \end{pmatrix} = \begin{pmatrix} -2 & -1 \\ 1 & 3(x_1^i)^2 \end{pmatrix}. \quad (5 - 8)$$

After the calculation of the reciprocal value of the determinant of the Jacobian matrix and the adjugate matrix, the inverted Jacobian matrix may be finally calculated

$$\mathbf{J}^{-1i} = \frac{1}{\det(\mathbf{J}^i)} \begin{pmatrix} \text{adj}(\mathbf{J}_{00}^i) & \text{adj}(\mathbf{J}_{01}^i) \\ \text{adj}(\mathbf{J}_{10}^i) & \text{adj}(\mathbf{J}_{11}^i) \end{pmatrix} = \frac{1}{\det(\mathbf{J}^i)} \begin{pmatrix} -2 & -1 \\ 1 & 3(x_1^i)^2 \end{pmatrix}. \quad (5 - 9)$$

Next the $(\Delta x_1^i, \Delta x_2^i)$ can be calculated using the inverted Jacobian matrix and the defect.

$$\begin{pmatrix} \Delta x_1^i \\ \Delta x_2^i \end{pmatrix} = \begin{pmatrix} \mathbf{J}_{00}^{-1,i} \Delta F_1^i + \mathbf{J}_{01}^{-1,i} \Delta F_2^i \\ \mathbf{J}_{10}^{-1,i} \Delta F_1^i + \mathbf{J}_{11}^{-1,i} \Delta F_2^i \end{pmatrix}. \quad (5 - 10)$$

Now the next iteration value denoted as $i + 1$ of x_1 and x_2 may be calculated

$$\begin{pmatrix} x_1^{i+1} \\ x_2^{i+1} \end{pmatrix} = \begin{pmatrix} x_1^i + \Delta x_1^i \\ x_2^i + \Delta x_2^i \end{pmatrix}. \quad (5 - 11)$$

With these new iteration values x_1^{i+1} x_2^{i+1} the loop for calculation starts again at the calculation of new values F_1^{i+1} and F_2^{i+1} which is presented at the start of this section.

5.2 IP Block Design

5.2.1 Top module design

Figure 5 - 1 depicts the top module design of the circuit. The Control Unit sends control signals to the Data Path unit to make the calculations. As in all designs in this paper, the numbers are formatted in the *Q32.15* fixed point format.

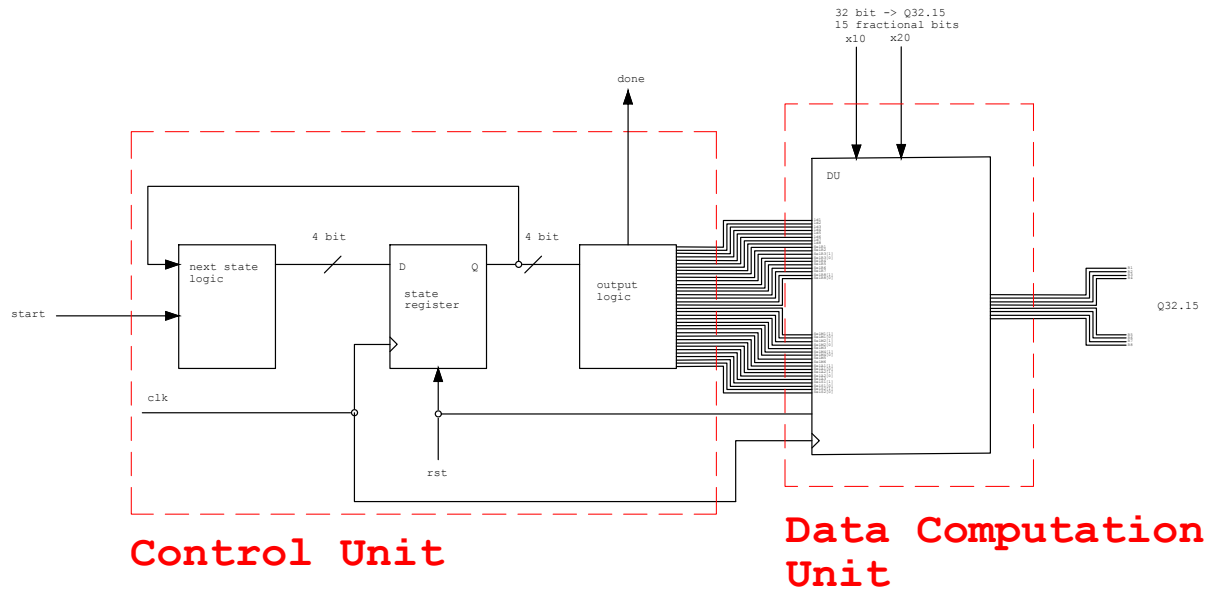


Figure 5 - 1 Top module design for the simple Newton-Raphson (NR) calculation module block.

5.2.2 Allocation and Timing

The algorithm structure for the Verilog implementation is depicted in the data flow diagram in the Figure 5 - 2.

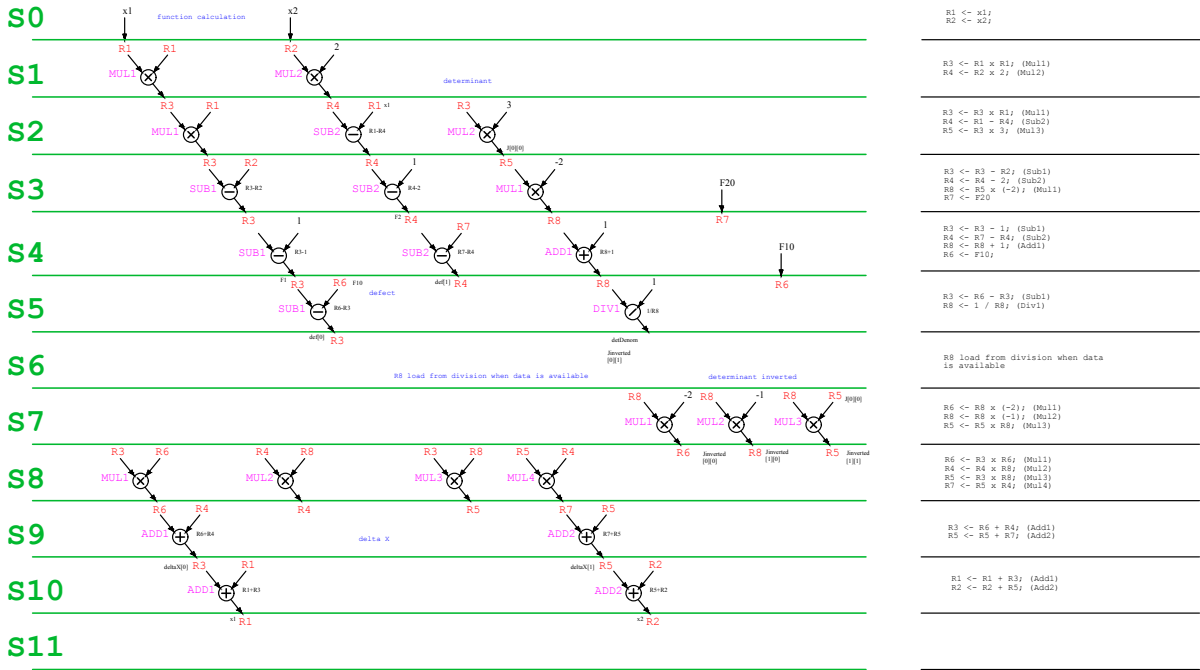


Figure 5 - 2 Allocation and timing diagram for the Data Path of the simple Newton-Raphson (NR) module.

5.2.3 Data Path Unit

The Data path unit for this simple NR algorithm consists of four multipliers, two adders, two subtractors and one divider. The divider is implemented using the Division Unit, presented in the section *Calculating the division of fixed point numbers*. Upon completion of the algorithm the results for x_1 and x_2 are saved in the R1 and R2 registers, the state transitions to S11 and signal *done* is set to 1. The results then can be driven to another module or unit for further usage.

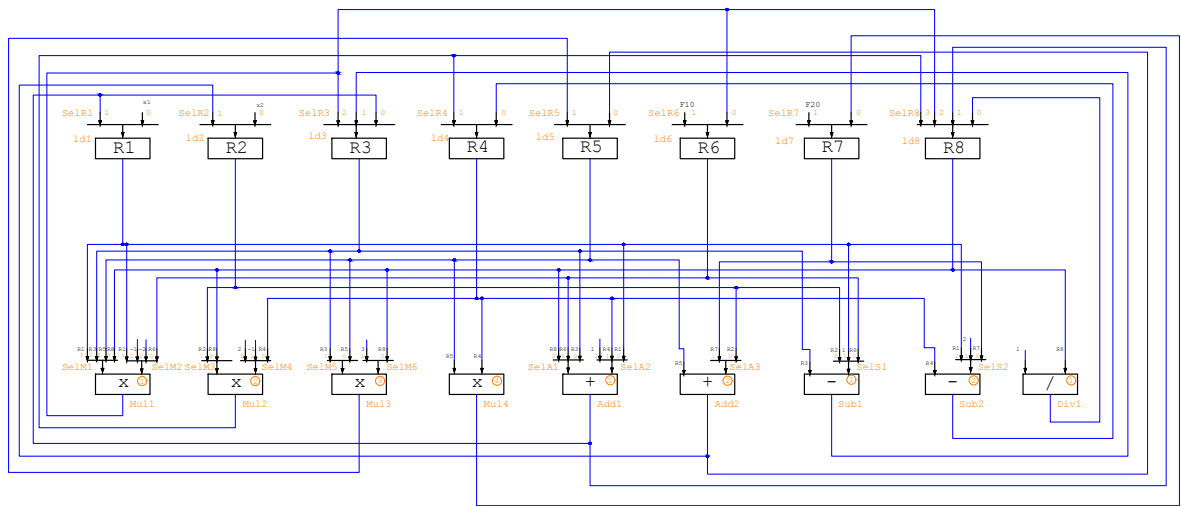


Figure 5 - 3 Register Transfer Level (RTL) scheme of the Data Path part of the simple Newton-Raphson (NR) calculation module.

5.2.4 Control Unit

The Table 5 - 1 shows encoding of a control signal for the Data Path unit.

The NR algorithm iteration transitions from the state *S10* to state *S1* when the iteration count is lower than the predetermined total number of iterations, value which is set in the Control Unit during the design phase. In this particular implementation, the total number of iterations is set to 5. It is worth noting that sometimes the termination of the NR algorithm is determined by the value of a defect. However, in this implementation the defect-check is not implemented.

Implementation of a defect-controlled algorithm would be straightforward. The values from registers holding the defect values, R3 and R4, would be connected to the control unit in the steps *S4* and *S5* respectively, and a comparison with the reference defect value would be executed. If the defect value was smaller than the reference value, the algorithm would transition to the state *S11* and therefore the calculation would end. Conversely, if the defect was larger than the reference value, the next state would be *S6* and the iteration would proceed normally, transitioning from state *S10* to *S1*.

Table 5 - 1 Control signal encoding table for instructions to be processed by the simple Newton-Raphson (NR) algorithm Module.

State	WFL Code	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS
S0	R1 ← x1; R2 ← x2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36'3C000000
S1	R3 ← R1 × R2 (1) R4 ← R2 × 2 (2)	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	36'3C200000
S2	R3 ← R1 × R2 (1) R4 ← R1 - R4 (2) R5 ← R3 × 3 (3)	0	0	0	1	1	1	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	36'3C24C002
S3	R3 ← R1 - R2 (1) R4 ← R4 - 2 (2) R5 ← R3 × (23) (3) R7 ← F200	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	36'3C310000
S4	R3 ← R3 - 1 (1) R4 ← R7 - R4 (2) R5 ← R5 + 1 (3) R6 ← F200	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	36'3C5120044
S5	R6 ← R6 - R5 (1) R9 ← 1 - R6 (1)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36'3C2100000
S6	R3 load from memory where data is available	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36'31000000
S7	R6 ← R9 × (2) (1) R7 ← R9 × (1) (2) R5 ← R5 × R6 (3)	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	36'3C04C000
S8	R6 ← R3 × R6 (1) R4 ← R4 × R6 (2) R5 ← R3 × R6 (3) R7 ← R5 × R6 (4)	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36'31E0C2000
S9	R3 ← R6 + R4 (1) R5 ← R5 + R7 (2)	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	36'3C2000000
S10	R3 ← R1 + R5 (1) R2 ← R2 + R5 (2)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36'3C3C00000
S11		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	36'3Cxxxxxx	

5.3 Simulation results

The test bench for simulation was made using Cocotb [1] with the Verilator [2] as a simulator. The results of the calculation may be seen in the registers R1 and R2. The results are $x_1 = -0.707489$ and $x_2 = -1.353759$.

The clock signal frequency in simulation was set to 20 MHz.

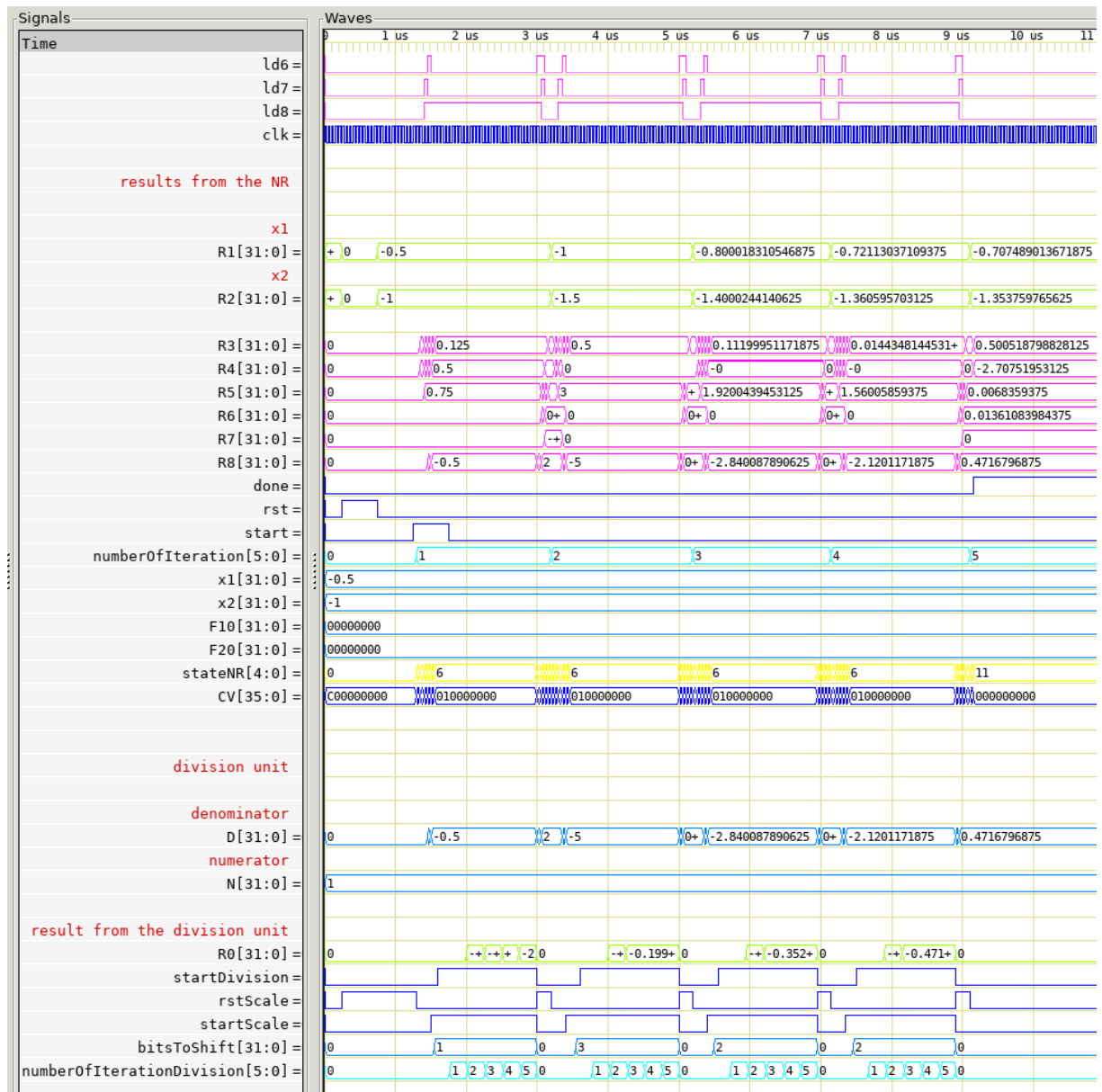


Figure 5 - 4 The complete Verilog simulation of a simple Newton-Raphson (NR) algorithm. The result may be seen in registers R1 and R2 after the fifth iteration of the algorithm.

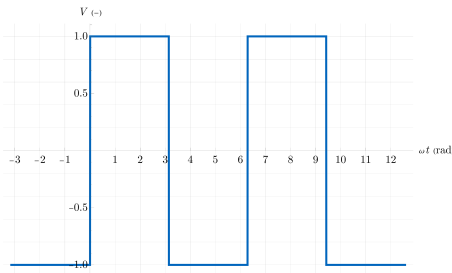
6 Selective Harmonic Elimination

6.1 Theory

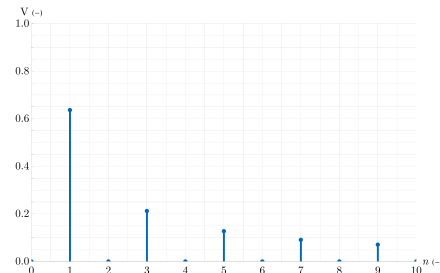
The original theory for Selective Harmonic Elimination was initially developed in [8, 9] and later adopted by numerous researchers for various voltage inverter topologies. Currently, the strategy is primarily employed in traction applications after start up state ends and the reference voltage for the drive is high enough so the six step output voltage is utilized. However, the general six step output signal produces high-order harmonics. When the motor is powered by these high-order voltage harmonics, the current with high-order harmonics (excluding triplen harmonics, considering the symmetric 3 phase motor) is observed. These current harmonics result in undesirable current ripple, torque ripple and losses [10], thereby decreasing the efficiency of the drive.

To control the output voltage and reduce unwanted harmonics, the Selective Harmonic Elimination (SHE) technique can be employed. The elimination is based on generating the output voltage by switching components at certain phase angles, thereby generating waveform with a number of pulses, to correspond the number of eliminated harmonics. The calculation which angles to use is based on the of Fourier coefficients theory. Equations describing the switching angles have been adapted for different types of converters, including multilevel, H-bridge converters or generic Voltage Source Inverters (VSI). In this paper, the regular two level VSI is considered.

The considered inverter phase voltage Six-Step output waveform is depicted in Figure 6 - 1a, while the harmonic analysis of the generic waveform is depicted in Figure 6 - 1b. It's worth noting that in a three-phase symmetrical system, the triplen harmonics are also eliminated.



(a) Generic Six-Step Waveform output of a two level Voltage Source Inverter. The Voltage value is normalized to a DC link voltage.



(b) Generic Six-Step Waveform harmonics analysis. The Voltage value is normalized to a DC link voltage.

Figure 6 - 1 Six-Step voltage waveform and harmonic analysis.

As previously mentioned, the SHE method is based on a Fourier coefficient analysis. When the odd quarter-wave symmetry of the waveform is assumed, the a_n Fourier coefficient is zero (as mentioned in the Equation 6 - 1), whereas the b_n coefficient may be written as Equation 6 - 2.

$$a_n = 0, \quad (6 - 1)$$

$$b_n = \frac{2}{T} \int_0^T x(n\omega t) \sin(\omega t) d\omega t, \quad (6 - 2)$$

where the T is signal periode, $x(\omega t)$ description of the VSI output voltage waveform and n is the order of the harmonics.

When assuming quarter-wave symmetry the Equation 6 - 2 may be rewritten as

$$b_n = \frac{8}{T} \int_0^{T/4} x(\omega t) \sin(n\omega t) d\omega t = \frac{8}{2\pi} \int_0^{2\pi/4} x(\omega t) \sin(n\omega t) d\omega t = \frac{4}{\pi} \int_0^{\pi/2} x(\omega t) \sin(n\omega t) d\omega t. \quad (6-3)$$

The function $x(\omega t)$ represents the normalized output voltage pulse in relation to a DC link voltage. The Equation 6 - 2 can be reformulated by substituting ωt with the angle α , which also characterizes the output waveform in terms of radians. The function $x(\alpha)$ yields 1 when the output voltage pulse is positive and -1 when negative. The reformulated Equation 6 - 2, assuming quarter-wave symmetry, is then as follows:

$$b_n = \sum_{k=1}^M \frac{8}{T} \int_{\alpha_k}^{\alpha_{k+1}} x(\alpha) \sin(n\alpha) d\alpha. \quad (6-4)$$

Here M represents number of pulses in half period of the output signal. Assuming that the integral is calculated for angles where $x(\alpha_k)$ is either 1 or -1 , the function may be replaced by a constant. As a result, when $x(\alpha_k) = 1$ the integral calculation becomes straightforward.

$$b_n = \frac{4}{\pi} \sum_{k=1}^M \frac{1}{n} [-\cos(n\alpha)]_{\alpha_k}^{\alpha_{k+1}} = \frac{4}{\pi n} \sum_{k=1}^M [\cos(n\alpha_k) - \cos(n\alpha_{k+1})]. \quad (6-5)$$

The Equation 6 - 5 can be further simplified by observing the results of the summation for $M = 2$.

$$\begin{aligned} b_n &= \frac{4}{\pi n} \sum_{k=1}^2 [\cos(n\alpha_k) - \cos(n\alpha_{k+1})] = \frac{4}{\pi n} [(\cos(n\alpha_1) - \cos(n\alpha_2)) + (\cos(n\alpha_2) - \cos(n\alpha_3))] = \\ &= \frac{4}{\pi n} (\cos(n\alpha_1) - \cos(n\alpha_3)). \end{aligned} \quad (6-6)$$

According to [8] and the example calculation for $M = 2$, the further simplification of the Equation 6 - 5 is Equation 6 - 7.

$$b_n = \frac{4}{\pi n} \sum_{k=1}^M (-1)^{k+1} \cos(n\alpha_k). \quad (6-7)$$

It can be said, that the number of eliminated odd harmonics is $N = M - 1$.

To maintain clarity of this paper only the 5th harmonics is being eliminated by the designed unit. The set of equations when incorporating the voltage level V_{DC} in a VSI DC link required to eliminate this harmonic is as follows.

$$\begin{aligned} V_1 &= b_1 = V_{DC} \frac{4}{\pi} [\cos(\alpha_1) - \cos(\alpha_2)], \\ V_5 &= b_5 = V_{DC} \frac{4}{5\pi} [\cos(5\alpha_1) - \cos(5\alpha_2)]. \end{aligned} \quad (6-8)$$

The amplitudes of the 1st and 5th harmonics are denoted as $V_1 = b_1$ and $V_5 = b_5$, respectively. For the elimination of the 5th harmonic, it is required that $b_5 = 0$. Consequently, the set of Equations 6 - 8 can be simplified as set of Equations 6 - 9.

$$\begin{aligned}\frac{\pi V_1}{4V_{DC}} &= \cos(\alpha_1) - \cos(\alpha_2), \\ 0 &= \cos(5\alpha_1) - \cos(5\alpha_2).\end{aligned}\tag{6 - 9}$$

Solving the nonlinear Equations 6 - 9 is not straightforward. Various methods can be employed for solving the problem, such as Genetic Algorithms [11, 12, 13] or algebraic methods [14, 15]. One commonly used algebraic method is Newton-Raphson (NR) algorithm [16]. In this paper, the solution is obtained solely using NR algorithm. However, it's worth noting that the success of this method depends on setting the initial conditions correctly; otherwise, a solution may not be found. In contrast, Genetic Algorithms also require setting initial values, but they often use random numbers from predefined intervals.

In real-time systems, the approach for solving the SHE equations may often be to precalculate the required switching angles offline and then utilize the LUT in a microprocessor to determine which set of angles use for the set reference voltage. Nowadays the FPGAs are more frequently utilized to calculate the solution. The calculation can be highly parallelized and optimized, enabling the solution to be obtained in near real-time. In the following sections the prototype implementation in Python and final implementation in Verilog are presented.

6.2 Simplification for Verilog and High level implementation

When implementing the solution in computational software like Wolfram Mathematica, optimizing the algorithm is unnecessary. However, when implementing the algorithm to an FPGA, higher-level constructs are not automatically available, so the simplification is necessary. Before creating the Verilog design, it is suitable, for clarity and prototyping purposes, to implement the algorithm in Python. In this section, the simplified algorithm of a NR algorithm is presented.

The set of equations for eliminating the 5th harmonics may be formulated as

$$\begin{aligned}F_1^i &= \cos(\alpha_1) - \cos(\alpha_2), \\ F_2^i &= \cos(5\alpha_1) - \cos(5\alpha_2), \\ \text{where } F_1^0 &= m \frac{\pi}{4}, F_2^0 = 0.\end{aligned}\tag{6 - 10}$$

Where $m = V_1/V_{DC}$ is modulation index.

Thus the Jacobian matrix is

$$\mathbf{J}^i = \begin{pmatrix} -\sin(\alpha_1^i) & \sin(\alpha_2^i) \\ -5\sin(5\alpha_1^i) & 5\sin(5\alpha_2^i) \end{pmatrix}.\tag{6 - 11}$$

Where i is the index of the iteration of the algorithm. Also the inverted Jacobian matrix is needed for further calculations.

$$\mathbf{J}^{-1,i} = \begin{pmatrix} \frac{5\sin(5\alpha_2^i)}{5\sin(5\alpha_1^i)\sin(\alpha_2^i) - 5\sin(\alpha_1^i)\sin(\alpha_2^i)} & -\frac{\sin(\alpha_2^i)}{5\sin(5\alpha_1^i)\sin(\alpha_2^i) - 5\sin(\alpha_1^i)\sin(\alpha_2^i)} \\ \frac{5\sin(\alpha_1^i)}{5\sin(5\alpha_1^i)\sin(\alpha_2^i) - 5\sin(\alpha_1^i)\sin(\alpha_2^i)} & -\frac{\sin(\alpha_1^i)}{5\sin(5\alpha_1^i)\sin(\alpha_2^i) - 5\sin(\alpha_1^i)\sin(\alpha_2^i)} \end{pmatrix}.\tag{6 - 12}$$

From the inverted Jacobian matrix in Equation 6 - 12, it is evident that it can be easily calculated by dividing corresponding components of Jacobian matrix by the determinant, expressed as

$$\det(\mathbf{J}) = 5\sin(5\alpha_1^i)\sin(\alpha_2^i) - 5\sin(\alpha_1^i)\sin(\alpha_2^i).\tag{6 - 13}$$

To avoid multiple division units, in which another NR algorithm is performed, the Verilog implementation takes advantage of calculating the value $1/\det(\mathbf{J})$ with one division unit and then multiplying the inverted Jacobian matrix elements with it. This solution saves a lot of clock cycles, thus makes the algorithm faster.

Next, the defect ΔF^i can be calculated

$$\begin{aligned}\Delta F_1^i &= F_1^0 - F_1^i, \\ \Delta F_2^i &= F_2^0 - F_2^i.\end{aligned}\tag{6 - 14}$$

After the successfully calculated defect of a current iteration, the $\Delta \alpha^i$ may be calculated.

$$\Delta \alpha^i = \mathbf{J}^{-1,i} \Delta \mathbf{F}^i,\tag{6 - 15}$$

thus rewritten in components notation which is more suitable for the Verilog implementation

$$\begin{aligned}\Delta \alpha_1^i &= \mathbf{J}_{00}^{-1,i} \Delta F_1^i + \mathbf{J}_{01}^{-1,i} \Delta F_2^i, \\ \Delta \alpha_2^i &= \mathbf{J}_{10}^{-1,i} \Delta F_1^i + \mathbf{J}_{11}^{-1,i} \Delta F_2^i.\end{aligned}\tag{6 - 16}$$

Finally the next iteration values of α_1^i and α_2^i may be calculated

$$\begin{aligned}\alpha_1^{i+1} &= \alpha_1^i + \Delta \alpha_1^i, \\ \alpha_2^{i+1} &= \alpha_2^i + \Delta \alpha_2^i.\end{aligned}\tag{6 - 17}$$

With the newly calculated values of α_1^i , α_2^i the algorithm may proceed with a new iteration ($i + 1$) for calculating the F_1^{i+1} and F_2^{i+1} values.

It is important to note, that for the NR algorithm to function correctly and yield viable results, suitable initial values F_1^0 and F_2^0 must be carefully chosen before the algorithm starts.

When eliminating the 5th harmonic with $m = 1$, the initial values of $F_2^0 = 0.08726$ rad and $F_2^0 = 1.3439$ rad yield satisfactory results.

The presented mathematical algorithm can then be transformed into an FPGA designed Verilog algorithm, visually represented as a block diagram in the section *Algorithm Block Design*.

6.3 High level implementation

The script allows changing the modulation index m at the beginning of the Python simulation. This feature enables generation of values that can be compared with results obtained from Verilog cocotb and Verilator simulation of the hardware-implemented algorithm.

The script may be run with command "`python3 she.py -mi <number>`", where `<number>` is the requested modulation index.

```
1 import math
2 import argparse # for parsing command line arguments
3
4 # colorama for colors, easier than init class, maybe later
5 # source: https://github.com/tartley/colorama
6 from colorama import init as colorama_init
7 from colorama import Fore
8 from colorama import Style
```

```

9
10 colorama_init(autoreset=True) # autoreset color on new line
11
12 # class with additional styles
13 class style:
14     BOLD = '\033[1m'
15     UNDERLINE = '\033[4m'
16     END = '\033[0m'
17
18 argParser = argparse.ArgumentParser() # new object
19 argParser.add_argument("-mi", "--modulationIndex", help="set the modulation
    index 0-1") # adding argument
20 args = argParser.parse_args() # parsing args
21 modulationIndex = args.modulationIndex
22
23 # Set the desired modulation index
24 if not modulationIndex:
25     print()
26     print(style.BOLD+Fore.RED + "You did not specify the modulation index
    with mi command, specify it now:\n" + style.END)
27     modulationIndex = input()
28
29 print("You have specified the modulation index: " + modulationIndex + ".\n"
    )
30
31 modulationIndex = float(modulationIndex)
32 totalNumberOfIterations = 10
33 f10 = modulationIndex * 0.7853981 # modulationIndex * pi/4
34 f20 = 0
35 x10 = 0.0872664 # 5 degree
36 x20 = 1.3439035 # 77 degree
37
38 x1 = x10
39 x2 = x20
40
41 # main NR-LOOP
42 for numberOfIteration in range(totalNumberOfIterations):
43     prepDeltaF1 = math.cos(x1) - math.cos(x2)
44     deltaF1 = f10 - prepDeltaF1
45
46     prepDeltaF2 = math.cos(5*x1) - math.cos(5*x2)
47     deltaF2 = f20 - prepDeltaF2
48
49     prepJ11 = math.sin(x1)
50     prepJ01 = math.sin(x2)
51     prepJ10 = 5 * math.sin(5*x1)
52     prepJ00 = 5 * math.sin(5*x2)
53

```



```

54
55     prepDet1 = prepJ10 * prepJ01
56     prepDet2 = 5 * prepJ11 * math.sin(5*x2)
57
58     prepDet = prepDet1 - prepDet2
59
60     divDet = 1 / prepDet
61
62     jInv00 = divDet * prepJ00
63     jInv01 = divDet * - prepJ01
64     jInv10 = divDet * prepJ10
65     jInv11 = divDet * - prepJ11
66
67
68     deltaX1 = (jInv00 * deltaF1) + (jInv01 * deltaF2)
69     deltaX2 = (jInv10 * deltaF1) + (jInv11 * deltaF2)
70
71     x1 = x1 + deltaX1
72     x2 = x2 + deltaX2
73
74     print(Fore.CYAN + "numberOfIteration: " + str(numberOfIteration) +
75           style.END)
76 # End of the main NR-LOOP
77
77 print(Fore.GREEN + "x1: " + str(x1) + style.END)
78 print(Fore.GREEN + "x2: " + str(x2) + style.END)

```

Code 6 - 1 Python implementation of the Selective Harmonic Elimination Algorithm with adjustable modulation index.

6.4 IP Block Design

6.4.1 Algorithm Block Diagram

The Figure 6 - 2 presents the hardware-implementation for SHE algorithm, mathematically expressed in the section *Simplification for Verilog and High level implementation*.

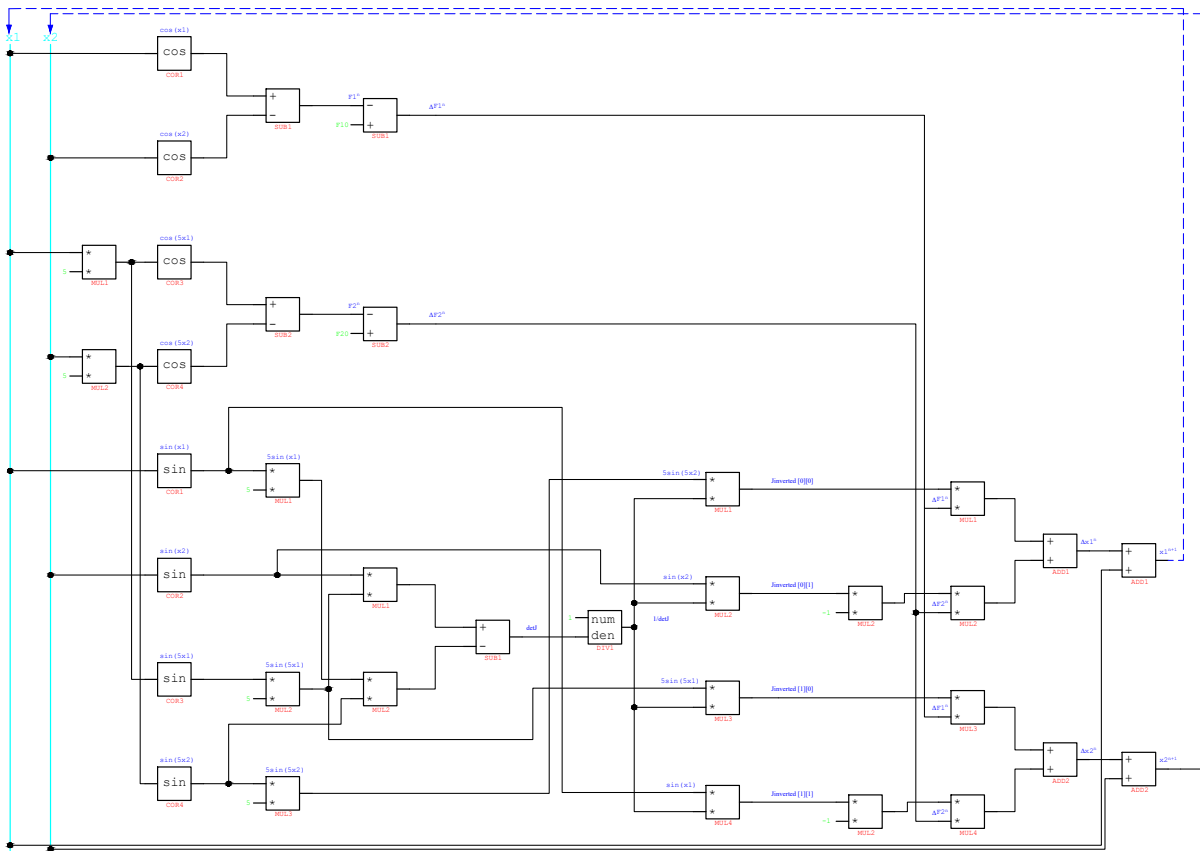


Figure 6 - 2 Block Diagram of the Selective Harmonic Elimination (SHE) algorithm using Newton-Raphson algorithm (NR). Design suitable for hardware implementation.

6.4.2 Top module design

The top module of this IP closely resembles other developed modules in this paper. The design consists of a Control Unit which sends control signals to the Data Unit. The Data Unit, which includes registers and computational units, incorporates few external sub-modules for additional calculations, such as CORDIC and division.

Consistent with every design presented, the units utilize the $Q32.15$ fixed point format for the computational units and registers. The exception is the multiplier computational units, which, by the principle of multiplication, use the $Q64.30$ format for results. When the multiplication results are transferred to registers, the values are rounded back to the globally used format.

The design is depicted in Figure 6 - 3.

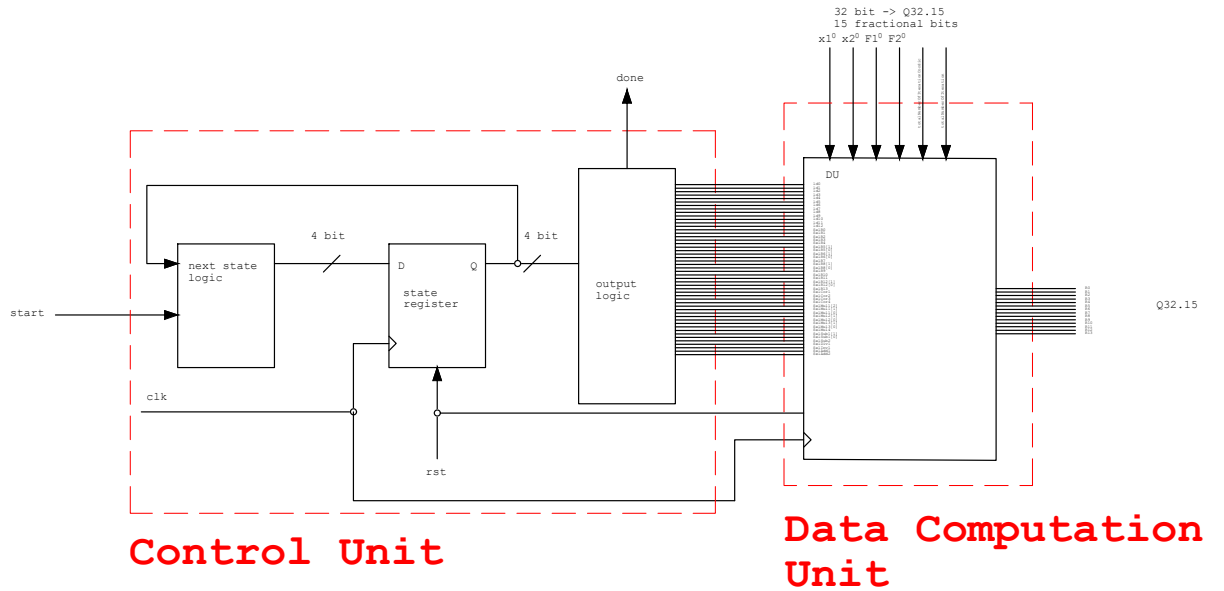


Figure 6 - 3 Top module design for the Selective Harmonic Elimination (SHE) module.

6.4.3 Allocation and Timing

The Allocation and Timing diagram, depicted in the Figure 6 - 4, outlines the algorithm presented in the *Theory* section. As evident from previous sections, this algorithm has been thoroughly tested before the Verilog implementation.

The Verilog implementation comprises a total of 13 states, labeled $S0$ - $S12$. Through states $S1$ - $S11$, the NR algorithm iterates to calculate the final results. The state $S0$ is a starting state after resetting the unit, and state $S12$ is the ending state reached after the successful calculation of the last algorithm iteration.

As previously stated, the SHE calculation module consists of various submodules, which may use other iterative algorithms. Iterations of these submodule algorithms are not in focus of this section and are implicitly accepted as a part of the SHE module algorithm.

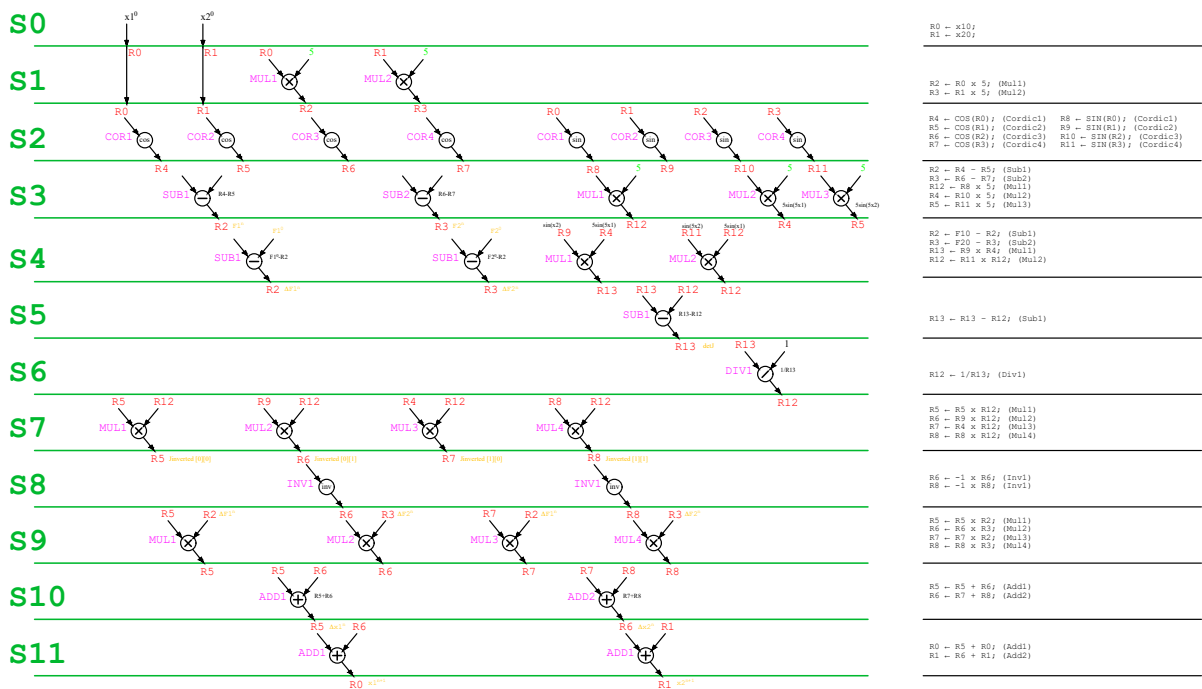


Figure 6 - 4 Allocation and Timing diagram for the Data Path part of Selective Harmonic Elimination (SHE) module.

6.4.4 Data Path Unit

As can be observed from the Figure 6 - 5 the Data Path unit for solving the transcendental equations is more complex than previously presented units. Obviously the design could be further simplified, i.e., reduce the number of registers and calculation units. This simplification would result in a trade of speed for less complexity. The less complex the design, the less FPGA resources, i.e., LUTs, is needed for the realization of the design. This paper mainly focuses on speed and clarity, so the design consists of thirteen data registers, four CORDIC units, four multiplication units, two adders, two subtractors, one division unit and one inverter unit, which is implemented directly in the registers logic.

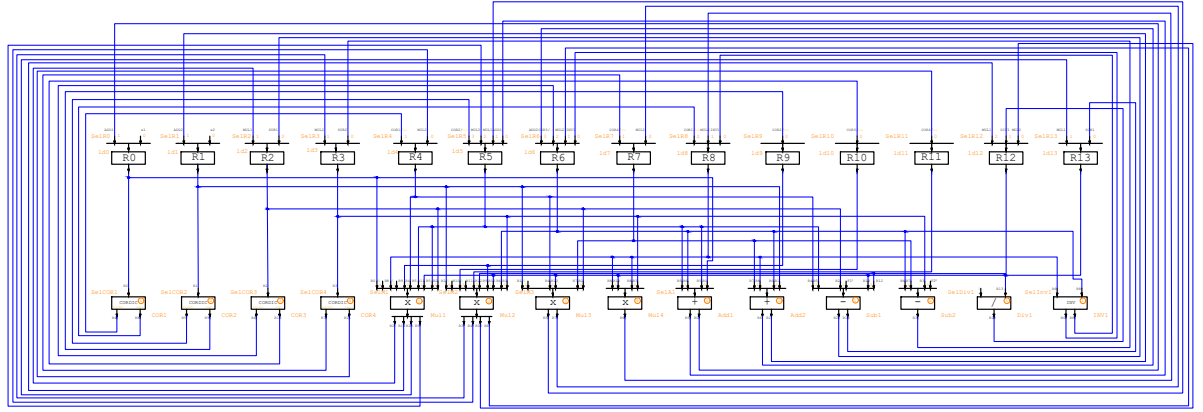


Figure 6 - 5 Register transfer level (RTL) scheme of the Selective Harmonic Elimination Data Path.

6.4.5 Control Unit

Control unit signal specification can be observed in the Table 6 - 1. If the unit design was less complex, i.e., with smaller amount of registers, the control signal length would be smaller, but the number of states would be higher.

Table 6 - 1 Control signal encoding table for instructions to be processed by the Selective Harmonic Elimination (SHE) algorithm Module.

Inst	Inst Code	u1	u2	u3	u4	u5	u6	u7	u8	u9	u10	u11	u12	u13	u14	u15	u16	u17	u18	u19	u20	u21	u22	u23	u24	u25	u26	u27	u28	u29	u30	u31	u32	u33	u34	u35	u36	u37	u38	u39	u40	u41	u42	u43	u44	u45	u46	u47	u48	u49	u50	u51	u52	u53	u54	u55	u56	u57	u58	u59	u60	u61	u62	u63	u64	u65	u66	u67	u68	u69	u70	u71	u72	u73	u74	u75	u76	u77	u78	u79	u80	u81	u82	u83	u84	u85	u86	u87	u88	u89	u90	u91	u92	u93	u94	u95	u96	u97	u98	u99	u100	u101	u102	u103	u104	u105	u106	u107	u108	u109	u110	u111	u112	u113	u114	u115	u116	u117	u118	u119	u120	u121	u122	u123	u124	u125	u126	u127	u128	u129	u130	u131	u132	u133	u134	u135	u136	u137	u138	u139	u140	u141	u142	u143	u144	u145	u146	u147	u148	u149	u150	u151	u152	u153	u154	u155	u156	u157	u158	u159	u160	u161	u162	u163	u164	u165	u166	u167	u168	u169	u170	u171	u172	u173	u174	u175	u176	u177	u178	u179	u180	u181	u182	u183	u184	u185	u186	u187	u188	u189	u190	u191	u192	u193	u194	u195	u196	u197	u198	u199	u200	u201	u202	u203	u204	u205	u206	u207	u208	u209	u210	u211	u212	u213	u214	u215	u216	u217	u218	u219	u220	u221	u222	u223	u224	u225	u226	u227	u228	u229	u230	u231	u232	u233	u234	u235	u236	u237	u238	u239	u240	u241	u242	u243	u244	u245	u246	u247	u248	u249	u250	u251	u252	u253	u254	u255	u256	u257	u258	u259	u260	u261	u262	u263	u264	u265	u266	u267	u268	u269	u270	u271	u272	u273	u274	u275	u276	u277	u278	u279	u280	u281	u282	u283	u284	u285	u286	u287	u288	u289	u290	u291	u292	u293	u294	u295	u296	u297	u298	u299	u300	u301	u302	u303	u304	u305	u306	u307	u308	u309	u310	u311	u312	u313	u314	u315	u316	u317	u318	u319	u320	u321	u322	u323	u324	u325	u326	u327	u328	u329	u330	u331	u332	u333	u334	u335	u336	u337	u338	u339	u340	u341	u342	u343	u344	u345	u346	u347	u348	u349	u350	u351	u352	u353	u354	u355	u356	u357	u358	u359	u360	u361	u362	u363	u364	u365	u366	u367	u368	u369	u370	u371	u372	u373	u374	u375	u376	u377	u378	u379	u380	u381	u382	u383	u384	u385	u386	u387	u388	u389	u390	u391	u392	u393	u394	u395	u396	u397	u398	u399	u400	u401	u402	u403	u404	u405	u406	u407	u408	u409	u410	u411	u412	u413	u414	u415	u416	u417	u418	u419	u420	u421	u422	u423	u424	u425	u426	u427	u428	u429	u430	u431	u432	u433	u434	u435	u436	u437	u438	u439	u440	u441	u442	u443	u444	u445	u446	u447	u448	u449	u450	u451	u452	u453	u454	u455	u456	u457	u458	u459	u460	u461	u462	u463	u464	u465	u466	u467	u468	u469	u470	u471	u472	u473	u474	u475	u476	u477	u478	u479	u480	u481	u482	u483	u484	u485	u486	u487	u488	u489	u490	u491	u492	u493	u494	u495	u496	u497	u498	u499	u500	u501	u502	u503	u504	u505	u506	u507	u508	u509	u510	u511	u512	u513	u514	u515	u516	u517	u518	u519	u520	u521	u522	u523	u524	u525	u526	u527	u528	u529	u530	u531	u532	u533	u534	u535	u536	u537	u538	u539	u540	u541	u542	u543	u544	u545	u546	u547	u548	u549	u550	u551	u552	u553	u554	u555	u556	u557	u558	u559	u560	u561	u562	u563	u564	u565	u566	u567	u568	u569	u570	u571	u572	u573	u574	u575	u576	u577	u578	u579	u580	u581	u582	u583	u584	u585	u586	u587	u588	u589	u590	u591	u592	u593	u594	u595	u596	u597	u598	u599	u600	u601	u602	u603	u604	u605	u606	u607	u608	u609	u610	u611	u612	u613	u614	u615	u616	u617	u618	u619	u620	u621	u622	u623	u624	u625	u626	u627	u628	u629	u630	u631	u632	u633	u634	u635	u636	u637	u638	u639	u640	u641	u642	u643	u644	u645	u646	u647	u648	u649	u650	u651	u652	u653	u654	u655	u656	u657	u658	u659	u660	u661	u662	u663	u664	u665	u666	u667	u668	u669	u670	u671	u672	u673	u674	u675	u676	u677	u678	u679	u680	u681	u682	u683	u684	u685	u686	u687	u688	u689	u690	u691	u692	u693	u694	u695	u696	u697	u698	u699	u700	u701	u702	u703	u704	u705	u706	u707	u708	u709	u710	u711	u712	u713	u714	u715	u716	u717	u718	u719	u720	u721	u722	u723	u724	u725	u726	u727	u728	u729	u730	u731	u732	u733	u734	u735	u736	u737	u738	u739	u740	u741	u742	u743	u744	u745	u746	u747	u748	u749	u750	u751	u752	u753	u754	u755	u756	u757	u758	u759	u760	u761	u762	u763	u764	u765	u766	u767	u768	u769	u770	u771	u772	u773	u774	u775	u776	u777	u778	u779	u780	u781	u782	u783	u784	u785	u786	u787	u788	u789	u790	u791	u792	u793	u794	u795	u796	u797	u798	u799	u800	u801	u802	u803	u804	u805	u806	u807	u808	u809	u810	u811	u812	u813	u814	u815	u816	u817	u818	u819	u820	u821	u822	u823	u824	u825	u826	u827	u828	u829	u830	u831	u832	u833	u834	u835	u836	u837	u838	u839	u840	u841	u842	u843	u844	u845	u846	u847	u848	u849	u850	u851	u852	u853	u854	u855	u856	u857	u858	u859	u860	u861	u862	u863	u864	u865	u866	u867	u868	u869	u870	u871	u872	u873	u874	u875	u876	u877	u878	u879	u880	u881	u882	u883	u884	u885	u886	u887	u888	u889	u890	u891	u892	u893	u894	u895	u896	u897	u898	u899	u900	u901	u902	u903	u904	u905	u906	u907	u908	u909	u910	u911	u912	u913	u914	u915	u916	u917	u918	u919	u920	u921	u922	u923	u924	u925	u926	u927	u928	u929	u930	u931	u932	u933	u934	u935	u936	u937	u938	u939	u940	u941	u942	u943	u944	u945	u946	u947	u948	u949	u950	u951	u952	u953	u954	u955	u956	u957	u958	u959	u960	u961	u962	u963	u964	u965	u966	u967	u968	u969	u970	u971	u972	u973	u974	u975	u976	u977	u978	u979	u980	u981	u982	u983	u984	u985	u986	u987	u988	u989	u990	u991	u992	u993	u994	u995	u996	u997	u998	u999	u1000	u1001	u1002	u1003	u1004	u1005	u1006	u1007	u1008	u1009	u1010	u1011	u1012	u1013	u1014	u1015	u1016	u1017	u1018	u1019	u1020	u1021	u1022	u1023	u1024	u1025	u1026	u1027	u1028	u1029	u1030	u1031	u1032	u1033	u1034	u1035	u1036	u1037	u1038	u1039	u1040	u1041	u1042	u1043	u1044	u1045	u1046	u1047	u1048	u1049	u1050	u1051	u1052	u1053	u1054	u1055	u1056	u1057	u1058	u1059	u1060	u1061	u1062	u1063	u1064	u1065	u1066	u1067	u1068	u1069	u1070	u1071	u1072	u1073	u1074	u1075	u1076	u1077	u1078	u1079	u1080	u1081	u1082	u1083	u1084	u1085	u1086	u1087	u1088	u1089	u1090	u1091	u1092	u1093	u1094	u1095	u1096	u1097	u1098	u1099	u1100	u1101	u1102	u1103	u1104	u1105	u1106	u1107	u1108	u1109	u1110	u1111	u1112	u1113	u1114	u1115	u1116	u1117	u1118	u1119	u1120	u1121	u1122	u1123	u1124	u1125	u1126	u1127	u1128	u1129	u1130	u1131	u1132	u1133	u1134	u1135	u1136	u1137	u1138	u1139	u1140	u1141	u1142	u1143	u1144	u1145	u1146	u1147	u1148	u1149	u1150	u1151	u1152	u1153	u1154	u1155	u1156	u1157	u1158	u1159	u1160	u1161	u1162	u1163	u1164	u1165	u1166	u1167	u1168	u1169	u1170	u1171	u1172	u1173	u1174	u1175	u1176	u1177	u1178	u1179	u1180	u1181	u1182	u1183	u1184	u1185	u1186	u1187	u1188	u1189	u1190	u1191	u1192	u1193	u1194	u1195	u1196	u1197	u1198	u1199	u1200	u1201	u1202	u1203	u1204	u1205	u1206	u1207	u1208	u1209	u1210	u1211	u1212	u1213	u1214	u1215	u1216	u1217	u1218	u1219	u1220	u1221	u1222	u1223	u1224	u1225	u1226	u1227	u1228	u1229	u1230	u1231	u1232	u1233	u1234	u1235	u1236	u1237	u1238	u1239	u1240	u1241	u1242	u1243	u1244	u1245	u1246	u1247	u1248	u1249	u1250	u1251	u1252	u1253	u1254	u1255	u1256	u1257	u1258	u1259	u1260	u1261	u1262	u1263	u1264	u1265	u1266	u1267	u1268	u1269	u1270	u1271	u1272	u1273	u1274	u1275	u1276	u1277	u1278	u1279	u1280	u1281	u1282	u1283	u1284	u1285	u1286	u1287	u1288	u1289	u1290	u1291	u1292	u1293	u1294	u1295	u1296	u1297	u1298	u1299	u1300	u1301	u1302	u1303	u1304	u1305	u1306	u1307	u1308	u1309	u1310	u1311	u1312	u1313	u1314	u1315	u1316	u1317	u1318	u1319	u1320	u1321	u1322	u1323	u1324	u1325	u1326	u1327	u1328	u1329	u1330	u1331	u1332	u1333	u1334	u1335	u1336	u1337	u1338	
------	-----------	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	--

6.5 Simulation results

The detail of the ending part of simulation with result of SHE algorithm after the 10th NR iteration can be seen in Figure 6 - 8. The whole simulation run is depicted in the Figure 6 - 9.

The clock signal frequency in simulation was set to 25 MHz to emulate low cost FPGA capabilities.



Figure 6 - 8 The ending part of the Verilog simulation the Selective Harmonic Elimination (SHE) algorithm. The result are in registers R0 and R1.

Conclusion

This paper introduces FPGA module designed for solving the SHE algorithm in near real-time. The module comprises two additional submodules, both discussed in this paper. These submodules include units for calculating the division of two arbitrary values and a CORDIC unit suitable for calculating *sine* and *cosine* functions.

The primary objective of this paper was to design speed-optimized modules capable of near real-time calculations. The outcomes of this paper can serve as a starting point for future research in designing FPGA modules for controlling electric drives or in creating the Hardware-in-Loop Systems.

References

- [1] LTD, Potential Ventures; INC, SolarFlare Communications. Cocotb. In: *Cocotb website* [online]. [B.r.] [visited on 2023-10-08]. Available from: <https://www.cocotb.org/>.
- [2] SNYDER, Wilson. Verilator. In: *Verilator website* [online]. [B.r.] [visited on 2023-10-08]. Available from: <https://www.veripool.org/verilator/>.
- [3] ADVANCED MICRO DEVICES, Inc. Divider Generator LogiCORE™ IP. In: *Intellectual Property* [online]. [B.r.] [visited on 2023-10-01]. Available from: <https://www.xilinx.com/products/intellectual-property/divider.html>.
- [4] BURKE, Tom. Verilog Fixed point math library. In: *GitHub* [online]. [B.r.] [visited on 2023-10-01]. Available from: https://github.com/freecores/verilog_fixed_point_math_library.
- [5] MEYER-BÄSE, Uwe. *Digital signal processing with field programmable gate arrays*. 4th ed. Berlin: Springer, 2014. ISBN 978-3-642-45308-3.
- [6] VOLDER, Jack E. The CORDIC Trigonometric Computing Technique. *IRE Transactions on Electronic Computers*. 1959, roč. EC-8, č. 3, pp. 330–334. Available from DOI: 10.1109/TEC.1959.5222693.
- [7] WALTHER, J. S. A Unified Algorithm for Elementary Functions. In: *Proceedings of the May 18-20, 1971, Spring Joint Computer Conference*. Atlantic City, New Jersey: Association for Computing Machinery, 1971, pp. 379–385. AFIPS '71 (Spring). ISBN 9781450379076. Available from DOI: 10.1145/1478786.1478840.
- [8] PATEL, Hasmukh S.; HOFT, Richard G. Generalized Techniques of Harmonic Elimination and Voltage Control in Thyristor Inverters: Part I--Harmonic Elimination. *IEEE Transactions on Industry Applications*. 05/1973, roč. IA-9, č. 3, pp. 310–317. ISSN 0093-9994. Available from DOI: 10.1109/TIA.1973.349908.
- [9] PATEL, Hasmukh S.; HOFT, Richard G. Generalized Techniques of Harmonic Elimination and Voltage Control in Thyristor Inverters: Part II --- Voltage Control Techniques. *IEEE Transactions on Industry Applications*. 09/1974, roč. IA-10, č. 5, pp. 666–673. ISSN 0093-9994. Available from DOI: 10.1109/TIA.1974.349239.
- [10] MÜLLNER, F.; NEUDORFER, H.; SCHMIDT, E. Modelling and precalculation of additional losses of inverter fed asynchronous induction machines for traction applications. In: *International Aegean Conference on Electrical Machines and Power Electronics and Electromotion, Joint Conference*. 2011, pp. 415–420. Available from DOI: 10.1109/ACEMP.2011.6490634.
- [11] TAGHIZADEH, H.; TARAFDAR HAGH, M. Harmonic elimination of multilevel inverters using particle swarm optimization. In: *2008 IEEE International Symposium on Industrial Electronics*. Cambridge, UK: IEEE, 06/2008, pp. 393–396. ISBN 978-1-4244-1665-3. Available from DOI: 10.1109/ISIE.2008.4677093.
- [12] ORTIZ-ESPINOZA, Alexandro; MENDEZ-FLORES, Efrain; PONCE-CRUZ, Pedro; MACIAS-HIDALGO, Israel; MOLINA-GUTIERREZ, Arturo. PWM with Selective Harmonic Elimination Using Optimization Inspired on Earthquakes for AC Electric Drives. In: *2020 5th International Conference on Control and Robotics Engineering (ICCRE)*. Osaka, Japan: IEEE, 04/2020, pp. 135–139. ISBN 978-1-72816-791-6. Available from DOI: 10.1109/ICCRE49379.2020.9096462.

- [13] ABDELQAWEE, I. M.; ABDEL-RAHIM, Naser M. B.; MANSOUR, Hajji. SELECTIVE HARMONIC ELIMINATION PWM VOLTAGE SOURCE INVERTER BASED ON GENETIC ALGORITHM. 2015. Available also from: <https://api.semanticscholar.org/CorpusID:38827373>.
- [14] WANG, Chenxu; ZHANG, Qi; YU, Wensheng; YANG, Kehu. A Comprehensive Review of Solving Selective Harmonic Elimination Problem with Algebraic Algorithms. *IEEE Transactions on Power Electronics*. 2023, pp. 1–20. ISSN 0885-8993, ISSN 1941-0107. Available from DOI: 10.1109/TPEL.2023.3327280.
- [15] CHIASSON, J.N.; TOLBERT, L.M.; MCKENZIE, K.J.; DU, Z. A Complete Solution to the Harmonic Elimination Problem. *IEEE Transactions on Power Electronics*. 03/2004, roč. 19, č. 2, pp. 491–499. ISSN 0885-8993. Available from DOI: 10.1109/TPEL.2003.823207.
- [16] BALOW, Writwik; HALDER, T. A Selective Harmonic Elimination (SHE) Technique for the Multi-Levelled Inverters. In: *2018 IEEE Electron Devices Kolkata Conference (EDKCON)*. Kolkata, India: IEEE, 11/2018, pp. 625–630. ISBN 978-1-5386-6415-5. Available from DOI: 10.1109/EDKCON.2018.8770415.
- [17] BURENEVA, Olga I.; KAIDANOVICH, Olga U. FPGA-based Hardware Implementation of Fixed-point Division using Newton-Raphson Method. In: *2023 IV International Conference on Neural Networks and Neurotechnologies (NeuroNT)*. 2023, pp. 45–47. Available from DOI: 10.1109/NeuroNT58640.2023.10175844.

Appendix A: List of and Abbreviations

A.1 List of abbreviations

CORDIC	Coordinate Rotation Digital Computer
CPU	Central Processing Unit
DC	Direct Current
FOSS	Free and Open-Source Software
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
IP	Intellectual property
ISA	Instruction Set Architecture
LUT	Look Up Table
NR	Newton Raphson
RTL	Register Transfer Level
SHE	Selective Harmonic Elimination
VCD	Value Change Dump
VSI	Voltage Source Inverter