CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Electric Drives and Traction

Name of the report

Technical report

Petr Zakopal
Prague 2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

This is the introduction.

2 Calculating the division of fixed point numbers

Usually, when using numerical methods to solve the transcendetal equations, there is a need to calculate the division of two input numbers. Even for solving one set of two equations with Newton Raphson (NR) method, the calculation of reciprocal value of the Jacobian determinant is needed.

There are available some IP blocks, which are capable of calculating the division of two numbers, but the blocks are usually vendor specific intellectual property IP [1] or feature low performance [2].

The negative side of vendor specific IP is, that it is hard to use them with any other FPGA chip than the vendor specific. On the other hand the vendor specific IP is usually optimized to use the specific type of resources available at the vendor's chip whci resolves in better performance.

To preserve the compatibility of the design with multiple vendors, the custom solution for division design based on the very known Newton Rapshon (NR) algorithm was developed. [2]

2.1 Newton Rapshon algorithm for calculating the division

General Newton Raphson (NR) algorithm is a well known way how to solve equations the numerical way. 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 unknown variables. When the initial variables 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. [2] The initial value formula 2 - 1 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*.

$$x_0 = \frac{48}{17} - \frac{32}{17}D, \tag{2 - 1}$$

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

Because of the fixed point number format *Q32.15* is used, the fractional numbers in equation 2 - 1 are rounded to 2.8229 (32'b00000000000000010_110100101011000 in binary)
and 1.8819 (32'b00000000000000001_111000011100101 in binary) respectively.

After the initial value x_0 is calculated, the NR algorithm is performed. The idea for 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. For the NR algorithm the function which root is $1/D$ is crucial. There may be many functions, which root is the searched value $1/D$ but the most trivial is eq. 2 - 2.

$$\mathrm{F}(x) = \frac{1}{x} - D. \tag{2 - 2}$$

For the derivative at the point of x_i then applies eq. 2 - 3.

$$\frac{\mathrm{d}F(x_i)}{\mathrm{d}x} = F'(x_i) = \frac{F(x_{i+1}) - F(x_i)}{x_{i+1} - x_i}. \tag{2 - 3}$$

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

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

Usually, the iterative algorithm is stopped, when the value $F(x_{i+1}) - F(x_i)$ (called 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 N-R algorithm the obtained result is sufficient after 5 iterations.

The mathematically expressed algorithm is then transformed into calculation flow suitable for implementing in the FPGA. The top module design for this algorithm is 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*

2.2 IP block design

The design of this division unit is separated into 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**, this unit is a Finite State Machine (FSM),
- **scaling unit module**, used for calculating the number of bits needed for shifting the denominator value to the interval [0.5,1].

2.2.1 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 in this top design is depicted in the fig. 2 - 1. Thanks to this wrapper it is possible to test the created modules with Verilog Testbench, Verilator [3] or Cocotb [4].

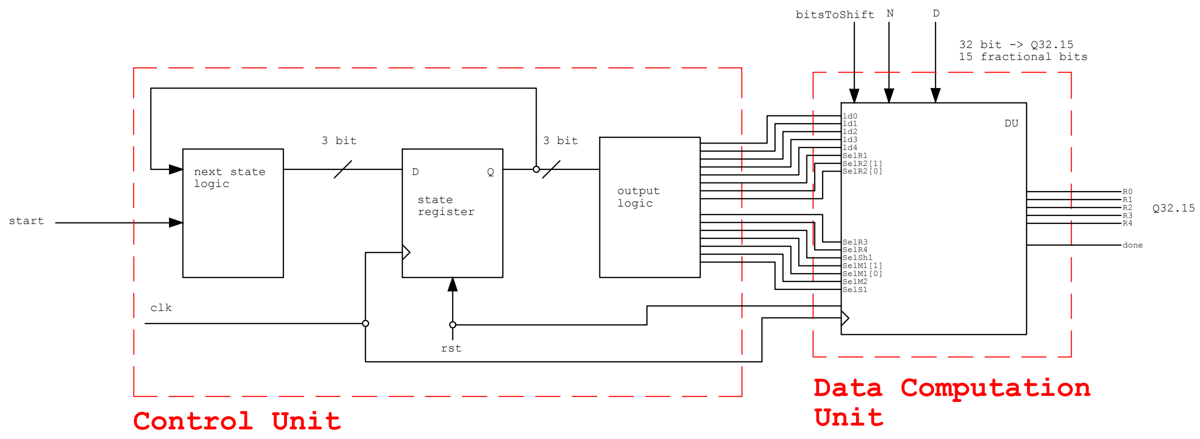

Figure 2 - 1 Top module design for the IP block design.

2.2.2 Allocation and Timing

The diagram which describes the data flow and timing by steps of the algorithm is displayed in the figure 2 - 2.

The whole algorithm consists of nine steps. The first four steps are used for calculating the initial value of x_0 as described in the equation 2 - 1. The steps *S4* to *S8* are for calculating the next search value of x_{i+1}, the root of the equation 2 - 2 so the searched value of $1/D$. The next iteration starts at the step labeled as *S5*. The iterative process continues till the stop condition (eg. number of iterations) is met.

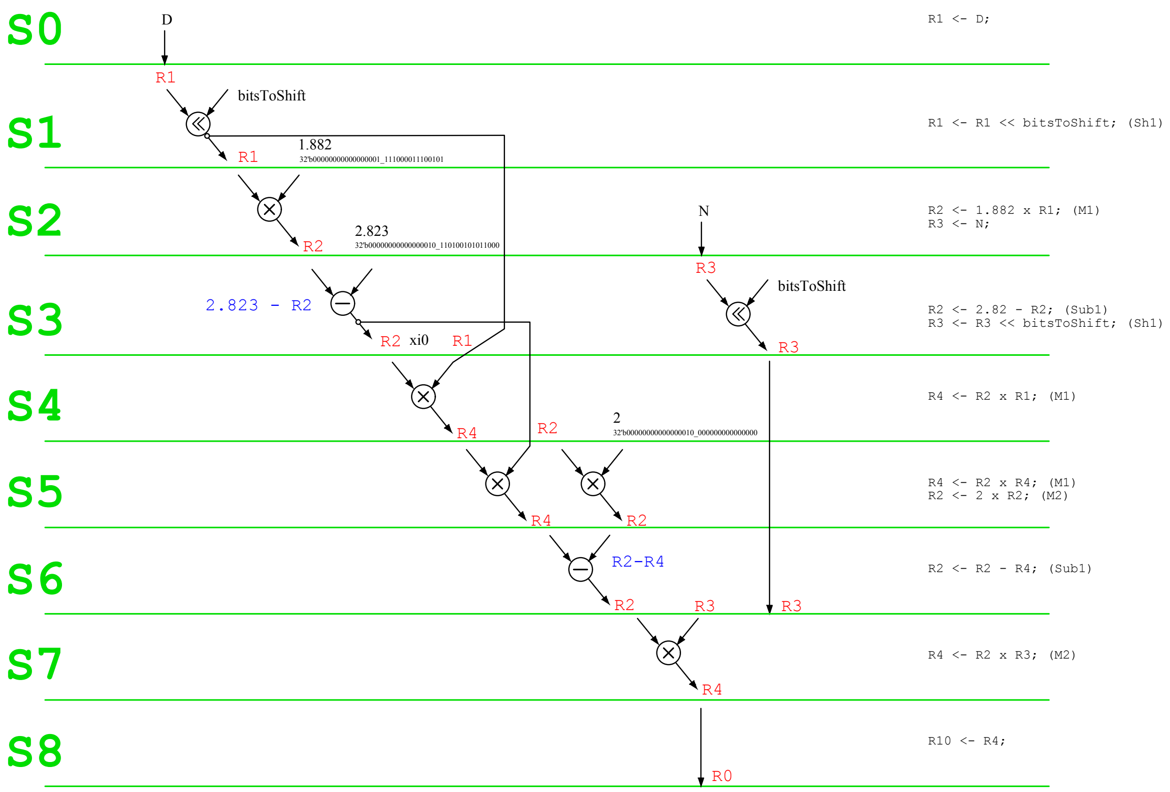

Figure 2 - 2 Alloccation and timing diagram for the Data Path Unit part of the IP.

2.2.3 Data Path Module

The structure of created Data Path Module is depicted in the figure 2 - 3. The module was specifically designed to serve the needs of the division algorithm. It consists of five registers labeled $R0$ through $R4$, two multipliers $M1$, $M2$ and one bit shifter.

The module si controlled by the presented control unit FSM with the control signal labeled as CV. The encoding table with the labels which corresponds with 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 negative denominator and numerator. Because the values are stored in a custom *Q32.15* fixed point format, the algorithm checks if the D or N values are higher than $0h8000$ value and determine it's actual sign and the sign if 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.

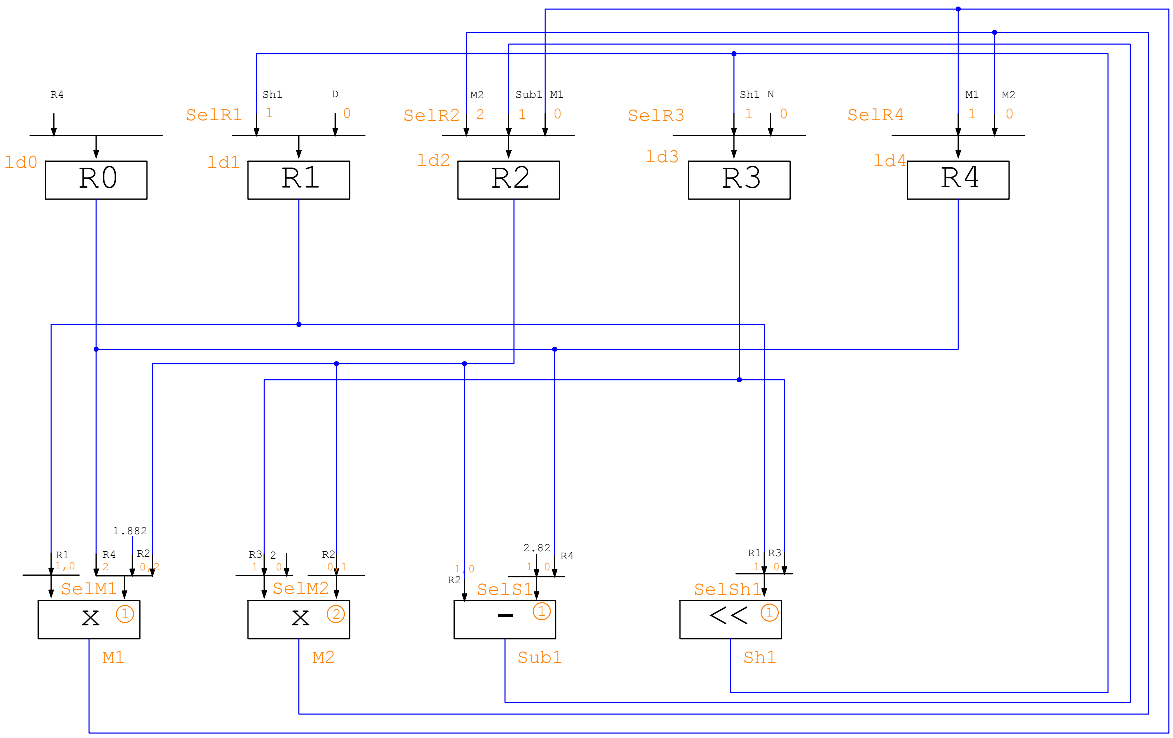

Figure 2 - 3 Register transfer level RTL scheme of the IP Data Path Unit part of the IP.

2.2.4 Control Unit

The signals from Control Unit to Data Path Module are encoded in the CV signal. The CV signal with the corresponding instructions for the steps $S0$–$S8$ of the FSM is presented in the table 2 - 1. For cleaner code, the signal is passed to the Control Unit in the hexadecimal format.

The number of the iteration is also set in the Control Unit. The value is used in this module to determine the stop condition of the calculation.

As stated in the *Allocation and Timing* section, after the step *S8*, the FSM restarts at the state *S4* with new x_i values to be used in the current iteration. This jump is not depicted in the table for CV signal.

Table 2 - 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	CV
		ld0	ld1	ld2	ld3	ld4	SelR1	SelR2[1]	SelR2[0]	SelR3	SelR4	SelSh1	SelM1[1]	SelM1[0]	SelM2	SelS1	
S0	R1 ← D;	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000h
S1	R1 ← R1 « 32; (Sh1)	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	2210h
S2	R2 ← 1.882 x R1; (M1) R3 ← N;	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1804h
S3	R2 ← 2.82 - R2; (Sub1) R3 ← R3 « 32; (Sh1)	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	18C0h
S4	R4 ← R2 x R1; (M1)	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	420h
S5	R4 ← R2 x R4; (M1) R2 ← 2 x R2; (M2)	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	1528h
S6	R2 ← R2 - R4; (S1)	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1081h
S7	R4 ← R2 x R3; (M2)	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	402h
S8	R0 ← R4;	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000h

2.3 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 2 - 1.

```
at every negative edge of clock or positive edge of reset
if(rst)
  scaleToShift = 0;
  scaleToShiftInternal = 1;
  started = 0;
end if
else if (start)
started = 1;
end else if

at every positive edge of clock
if (compare <= 32'b00000000000000001_000000000000000)
done = 1;
started = 0;
scaleToShift = scaleToShiftInternal;
end if
else
done = 0;
scaleToShiftInternal = scaleToShiftInternal + 1;
end else
```

Code 2 - 1 Pseudocode for the denominatorSizeScaleUnit module algorithm.

2.4 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.

As for the simulation output it can be stated, that the module works correctly for positive and negative numbers of fixed point format *Q32.15*.

The algorithm used in this module is able to calculate the propper result in much less clock cycles than the full division algorithm used in the division module in the package [2].

Thus the presented module may be used as a submodule in more complex modules.

Following figures present VCD wave simulation outputs for selected N and D. The clock frequency was set 250 MHz. Pseudocode Verilog snippet for the test bench is presented int he listing 2 - 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* when the *start* signal is set on low.

```
timescale 1ns/1ns
#10; // wait for 10 units of time
#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
N = 32'b0000000100110000_0000100000000000; D=32'b1111111111111111_1100000000000000; // set the numerator to N = 304.03125, denominator to D = -0.25
#10 rstScale = 0; // wait for 10 units of time and stop the reset of scaling unit
#10 startScale = 1; // start the algorithm for scaling unit
#20 rst = 1; start = 0; // reset the division unit
#30 rst = 0; // stop reseting of the division unit
#20 start = 1; // start the division unit
#20 start = 0;
#1000; // wait 1000 units of time
$finish; // finish the simulation
```

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

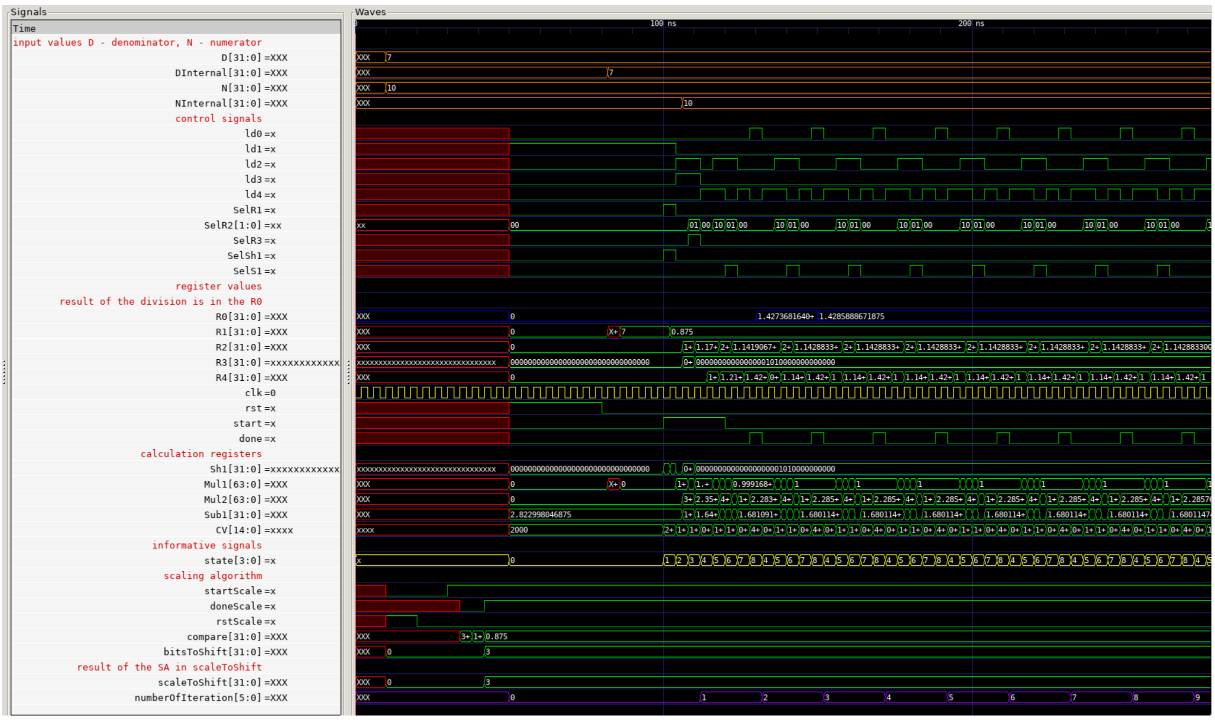

Figure 2 - 4 Selected signals of simulation of division N/D = 10 / 7. The correct result in R0 is obtained after two iterations (reg numberOfIterations).

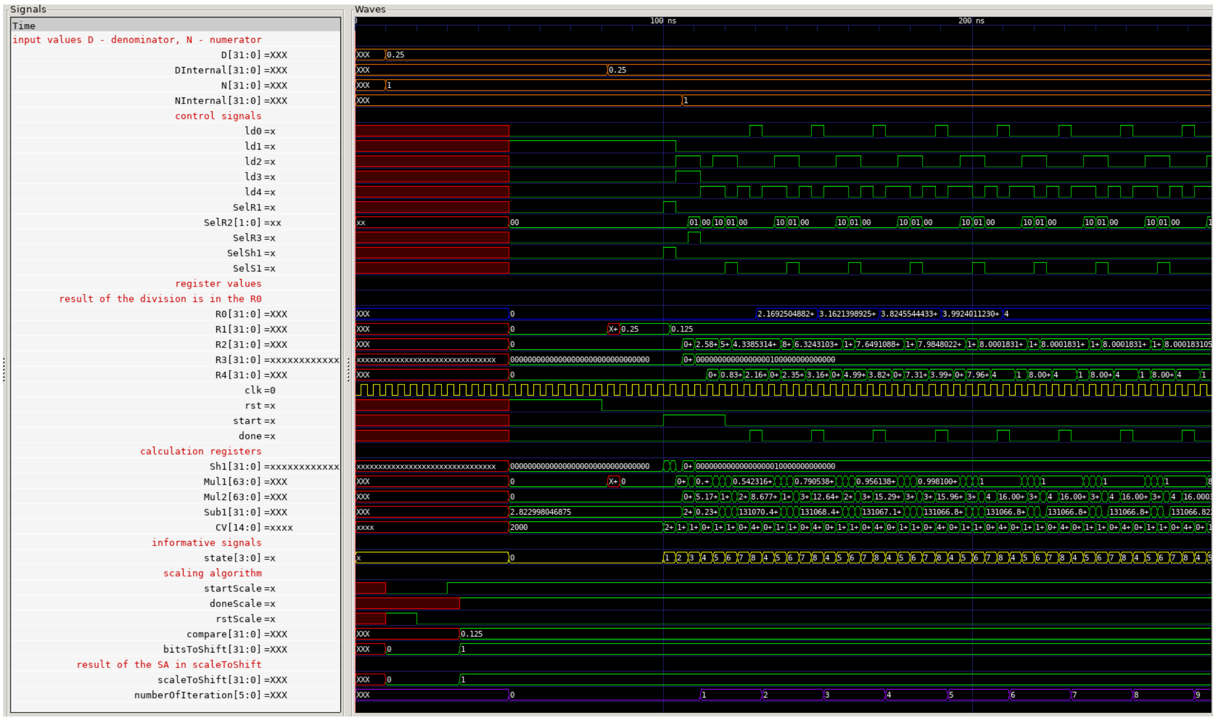

Figure 2 - 5 Selected signals of simulation of division N/D = 1 / 0.25. The correct result in R0 is obtained after five iterations (reg numberOfIterations).

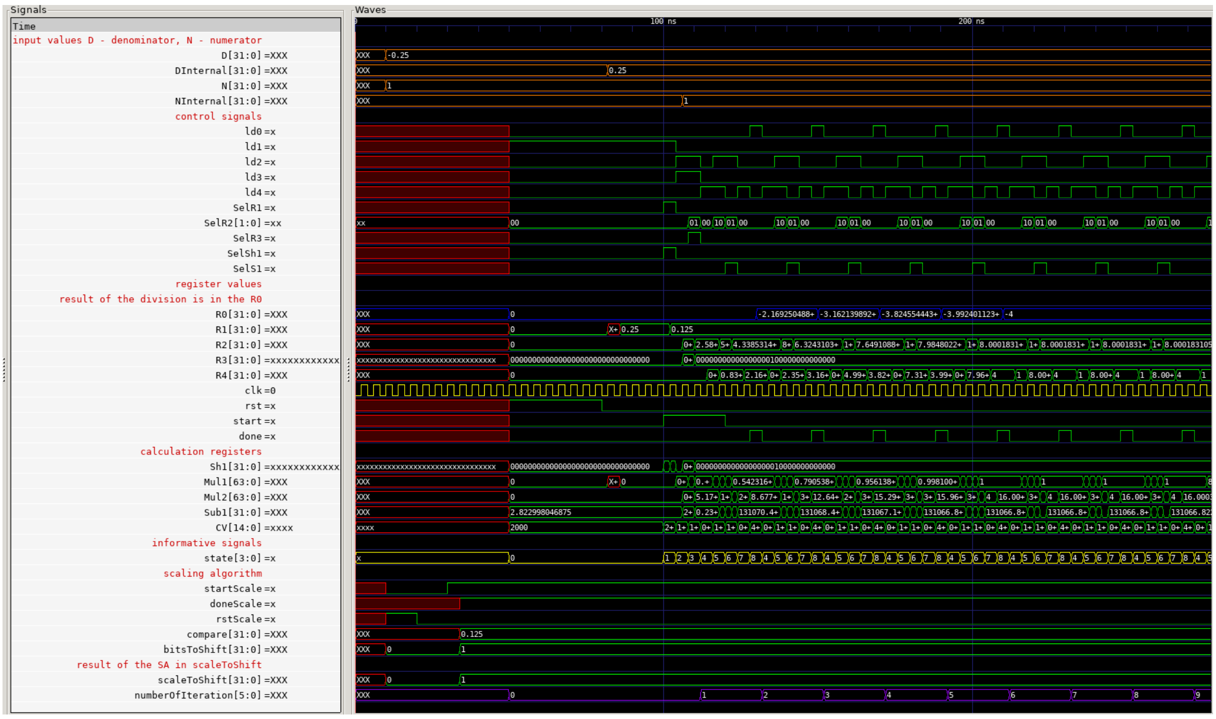

Figure 2 - 6 Selected signals of simulation of division N/D = 1 / (-0.25). The correct result in R0 is obtained after five iterations (reg numberOfIterations).

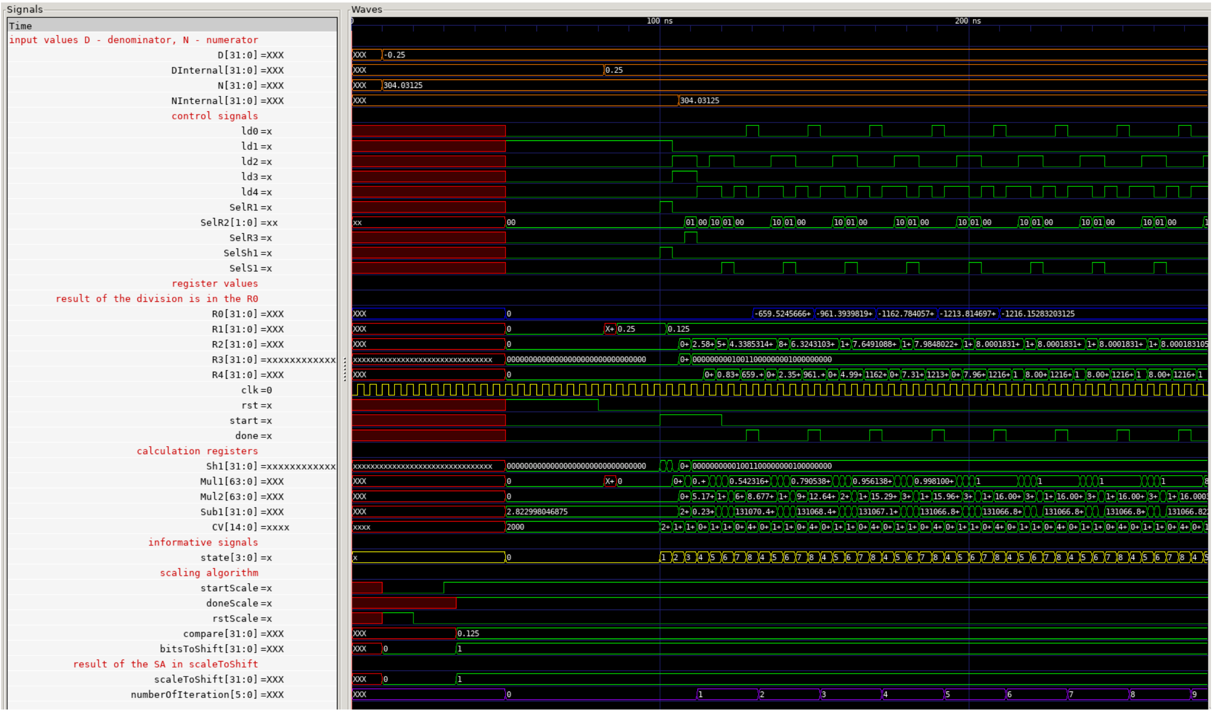

Figure 2 - 7 Selected signals of simulation of division N/D = 304.03215 / (-0.25). The correct result in R0 is obtained after five iterations (reg numberOfIterations).

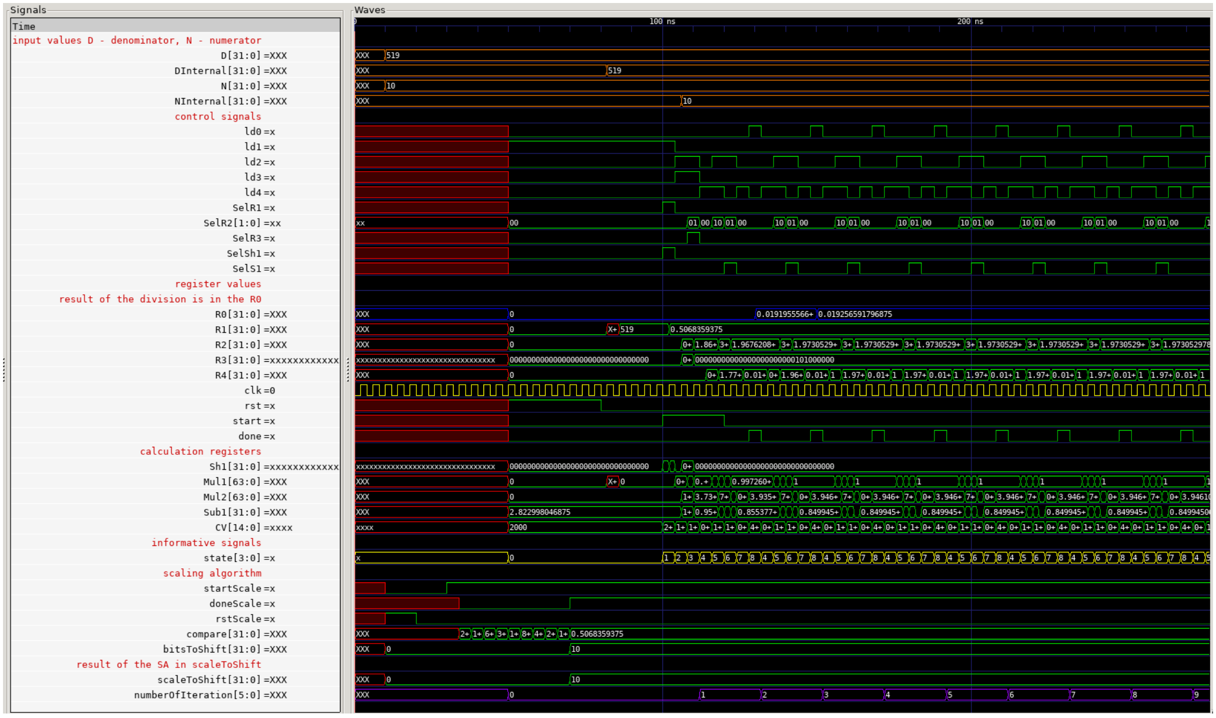

Figure 2 - 8 Selected signals of simulation of division N/D = 10 / (519). The correct result in R0 is obtained after two iterations (reg numberOfIterations).

3 Using CORDIC to calculate trigonometric functions

There are few approaches when there is a need to calculate the trigonometric functions. In this work, the CORDIC was used. To get more flexibility in the calculation the implementation of CORDIC algorithm was chosen above the LUT for calculation.

The LUT method may be fast, but the accuracy depends on the size of the table. When using the CORDIC one can influence the precision by using more iterations of the algorithm. The modified algorithm may be used to calculate non-trivial functions, such as hyperbolic functions, square roots, multiplications, divisions, exponentials and logarithms. [5] In this work only the calculation of $sinus$ and $cosinus$ functions is used.

3.1 Theory

The theory of the first CORDIC was proposed by Volder in [6]. This algorithm computes a coordinate conversion between rectangular (x, y) and polar (R, θ) coordinates. The algorithm was then generalized by Walther in [7] to include circular, linear and hyperbolic transforms. This paper utilizes only circular transforms to calculate $sinus$ and $cosinus$ functions. Only the most basic approach to the algorithm will be presented.

The rotation of a vector in the rectangular coordinate system (x, y) may be described by matrix-vector multiplication depicted in the eq. 3 - 1.

$$\begin{pmatrix} x_\mathrm{R} \\ y_\mathrm{R} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_\mathrm{in} \\ y_\mathrm{in} \end{pmatrix}, \tag{3 - 1}$$

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 was rotated.

Then when simplifying the equation

$$\begin{pmatrix} x_\mathrm{R} \\ y_\mathrm{R} \end{pmatrix} = \cos(\theta) \begin{pmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{pmatrix} \begin{pmatrix} x_\mathrm{in} \\ y_\mathrm{in} \end{pmatrix} \tag{3 - 2}$$

it can be seen, that only multiplication by scaling factor of precalculated values of $\cos(\theta)$, multiplication by $\tan(\theta)$, subtraction and addiction operations are needed. However, the multiplication by $\tan(\theta)$ can be interchanged. The interchange may be done for angles θ for which the equation 3 - 3 is true. The when implementing the algorithm to the FPGA the multiplication may be swapped for signed right bit shift.

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

Then when the values $x_\mathrm{in} = 1$ and $y_\mathrm{in} = 0$ are used, the result for $sinus$ and $cosinus$ can be easily obtained from x_R and y_R as expressed in the equation 3 - 4.

$$x_\mathrm{R} = x_\mathrm{in}\cos(\theta) - y_\mathrm{in}\sin(\theta) = |\theta = 0| = \cos(\theta) y_\mathrm{R} = x_\mathrm{in}\sin(\theta) + y_\mathrm{in}\cos(\theta) = |\theta = 0| = \sin(\theta) \tag{3 - 4}$$

The algorithm may be further simplified by expecting that the algorithm is designed to use more than 6 iterations and thus the scaling constant represented by multipliying $cosinus$ of different θ values converges to $0,60725$. So there is no need to precalculate all the scaling values and use only the value of the convergence. In this paper the scaling values are precalculated and passed from the custom LUT module.

As can be seen from the section *Example of calculation* section or the algortihm teory itself, it needs to be determined, if the angle for which the vector is rotated in the next iteration should be in a positive direction (counter-clockwise) or negative direction (clockwise). For that, the set of the equations is expanded with value z_i. The complete set of equations which are used in the *Implementation* are as follows.

$$\begin{aligned} x[i+1] &= x[i] - \sigma_i 2^{-i} y[i], \\ y[i+1] &= y[i] + \sigma_i 2^{-i} x[i], \\ z[i+1] &= z[i] - \sigma_i \operatorname{atan}(2^{-i}). \end{aligned} \tag{3 - 5}$$

The σ is determined based on the sign of the z variable

$$\sigma_i = \begin{Bmatrix} -1, \text{ if } z_i \leq 0 \\ 1, \text{ if } z_i \geq 0 \\ 0, \text{ if } z_i = 0 \end{Bmatrix} \tag{3 - 6}$$

3.1.1 Example of calculation

The general approach of CORDIC algorithm may be explained on the example for calculating the $sinus$ and $cosinus$ values for the angle $\theta = 57,535\ ^\circ$. Firstly, the angle may be destructurized in the base angles, for which the equation 3 - 3 is true. In this example the is destructurized as $57,535 = 45 + 25,565 - 14,03$.

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

$$0.\text{ iteration} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \cos(45\ ^\circ) \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_{\text{in}} \\ y_{\text{in}} \end{pmatrix}, \tag{3 - 7}$$

$$1.\text{ iteration} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \cos(26,565\ ^\circ) \begin{pmatrix} 1 & -2^{-1} \\ 2^{-1} & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}, \tag{3 - 8}$$

$$2.\text{ iteration} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \cos(-14,03\ ^\circ) \begin{pmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}. \tag{3 - 9}$$

Then after substitution the value of x_2 and y_2 may be obtained.

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \cos(45\ ^\circ)\cos(25,565\ ^\circ)\cos(-14,03\ ^\circ) \begin{pmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{pmatrix} \begin{pmatrix} 1 & -2^{-1} \\ 2^{-1} & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_{\text{in}} \\ y_{\text{in}} \end{pmatrix}. \tag{3 - 10}$$

From the equation 3 - 10 the values x_2 and y_2 represent the value of $\cos(57,535\ ^\circ)$ and $\sin(57,535\ ^\circ)$ respectively.

3.2 Implementation

3.3 Simulation results

Conclusion

And this is the conclusion of my report. P_{n}.

References

[1] 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.

[2] 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.

[3] SNYDER, Wilson. Verilator. In: *Verilator website* [online]. [B.r.] [visited on 2023-10-08]. Available from: https://www.veripool.org/verilator/.

[4] LTD, Potential Ventures; INC, SolarFlare Communications. Cocotb. In: *Cocotb website* [online]. [B.r.] [visited on 2023-10-08]. Available from: https://www.cocotb.org/.

[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] 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.

[9] DIGILENT, Inc. Zybo. In: *Digilent Documentation* [online]. [B.r.] [visited on 2022-11-11]. Available from: https://digilent.com/reference/programmable-logic/zybo/start.

[10] DIGILENT, Inc. Zybo Z7 Migration Guide. In: *Digilent Documentation* [online]. [B.r.] [visited on 2022-11-11]. Available from: https://digilent.com/reference/programmable-logic/zybo-z7/migration-guide.

[11] XILINX, Inc. SoCs with Hardware and Software Programmability. In: *Xilinx Website* [online]. [B.r.] [visited on 2022-11-11]. Available from: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[12] XILINX, Inc. Zynq-7000 SoC Technical Reference Manual. In: *Xilinx Documentation* [online]. 02. 04. 2021 [visited on 2022-11-11]. Available from: https://docs.xilinx.com/v/u/en-US/ug585-Zynq-7000-TRM.

[13] DIGILENT, Inc. Zybo Reference Manual. In: *Digilent Documentation* [online]. [B.r.] [visited on 2022-11-11]. Available from: https://digilent.com/reference/programmable-logic/zybo/reference-manual.

[14] XILINX, Inc. Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973). In: *AMD Xilinx Documentation Portal* [online]. [B.r.] [visited on 2022-11-18]. Available from: https://docs.xilinx.com/r/en-US/ug973-vivado-release-notes-install-license/.

[15] XILINX, Inc. PetaLinux Tools Documentation: Reference Guide (UG1144). In: *AMD Xilinx Documentation Portal* [online]. [B.r.] [visited on 2022-11-18]. Available from: https://docs.xilinx.com/r/en-US/ug1144-petalinux-tools-reference-guide.

[16] XILINX, Inc. Downloads. In: *AMD Xilinx Downloads* [online]. [B.r.] [visited on 2022-11-19]. Available from: https://www.xilinx.com/support/download.html.

[17] XILINX, Inc. Downloads. In: *AMD Xilinx PetaLinux Tools* [online]. [B.r.] [visited on 2022-11-19]. Available from: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[18] INC., Xilinx. Zynq-7000 SoC Technical Reference Manual (UG585). In: *Xilinx Documentation Portal* [online]. 02. 04. 2021 [visited on 2023-02-28]. Available from: https://docs.xilinx.com/v/u/en-US/ug585-Zynq-7000-TRM.

[19] XILINX, Inc. System-on-Modules (SOMs): How and Why to Use Them. In: *Xilinx Website* [online]. [B.r.] [visited on 2023-03-10]. Available from: https://www.xilinx.com/products/som/what-is-a-som.html.

[20] XILINX, Inc. Kria KR260 Robotics Starter Kit. In: *Xilinx Website* [online]. [B.r.] [visited on 2023-03-10]. Available from: https://www.xilinx.com/products/som/kria/kr260-robotics-starter-kit.html.

[21] XILINX, Inc. Kria SOM Carrier Card Design Guide (UG1091). In: *AMD Xilinx Documentation Portal* [online]. 27. 07. 2022 [visited on 2023-03-18]. Available from: https://docs.xilinx.com/r/en-US/ug1091-carrier-card-design/Introduction.

[22] XILINX, Inc. Kria K26 SOM Data Sheet (DS987). In: *AMD Xilinx Documentation Portal* [online]. 26. 07. 2022 [visited on 2023-03-18]. Available from: https://docs.xilinx.com/r/en-US/ds987-k26-som.

[23] XILINX, Inc. Kria KR260 Robotics Starter Kit User Guide (UG1092). In: *AMD Xilinx Documentation Portal* [online]. 17. 05. 2022 [visited on 2023-04-05]. Available from: https://docs.xilinx.com/r/en-US/ug1092-kr260-starter-kit/Interfaces.

[24] XILINX, Inc. Kria K26 System-on-Module. In: *AMD Xilinx Product Brief* [online]. [B.r.] [visited on 2023-04-05]. Available from: https://www.xilinx.com/content/dam/xilinx/publications/product-briefs/xilinx-k26-product-brief.pdf.

[25] XILINX, Inc. XTP743 - Kria KR260 Starter Kit Carrier Card Schematics (v1.0). In: *AMD Xilinx Board Files* [online]. 09. 06. 2022 [visited on 2023-04-06]. Available from: https://www.xilinx.com/member/forms/download/design-license.html?cid=bad0ada6-9a32-427e-a793-c68fed567427&filename=xtp743-kr260-schematic.zip.

[26] XILINX, Inc. XTP685 - Kria K26 SOM XDC File (v1.0). In: *AMD Xilinx Board Files* [online]. 14. 05. 2021 [visited on 2023-04-06]. Available from: https://www.xilinx.com/member/forms/download/design-license.html?cid=29e0261a-9532-4a47-bb06-38c83bbbb8c0&filename=xtp685-kria-k26-som-xdc.zip.

[27] ADMIN, Confluence Wiki; ROY, Debraj; DYLAN. Embedded SW Support. In: *Xilinx Wiki* [online]. 28. 02. 2023 [visited on 2023-04-06]. Available from: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841631/Embedded+SW+Support.

[28] FOUNDATION, Linux. Real-Time Linux. In: *Linux Foundation DokuWiki* [online]. [B.r.] [visited on 2023-04-06]. Available from: https://wiki.linuxfoundation.org/realtime/start.

[29] XILINX, Inc. Zynq UltraScale+ MPSoC Processing System Product Guide (PG201). In: *Xilinx Documentation* [online]. 11. 05. 2021 [visited on 2023-04-13]. Available from: https://docs.xilinx.com/r/en-US/pg201-zynq-ultrascale-plus-processing-system/Fabric-Reset-Enable.

[30] ZAKOPAL, Petr et al. [Kria SOM KR260 Starter Kit] Schematic (pdf) vs constrains (xdc) pin confusion. Possible explanation on fan pinout. In: *Xilinx Support Community Forum*. 18. 03. 2023. Available also from: https://support.xilinx.com/s/question/0D54U00006alUwcSAE/kria-som-kr260-starter-kit-schematic-pdf-vs-constrains-xdc-pin-confusion-possible-explanation-on-fan-pinout?language=en_US.

Appendix A: List of symbols and abbreviations

A.1 List of abbreviations

CORDIC	Coordinate Rotation Digital Computer
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
IP	Intellectual property
LUT	Look Up Table
NR	Newton Raphson
RTL	Register Transfer Level

A.2 List of symbols

P_n (W) jmenovitý výkon stroje