



**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

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
<b>2</b>	<b>Calculating the division of fixed point numbers.....</b>	<b>2</b>
2.1	Newton Rapshon algorithm for calculating the division .....	2
2.2	IP block design .....	3
2.2.1	Top module design .....	3
2.2.2	Allocation and Timing .....	4
2.2.3	Data Path Module .....	5
2.2.4	Control Unit .....	6
2.3	Calculating number of bits to shift the denominator.....	7
2.4	Simulation results .....	7
	<b>Conclusion .....</b>	<b>11</b>
	<b>References .....</b>	<b>14</b>
<b>Appendix A</b>	<b>List of symbols and abbreviations .....</b>	<b>15</b>
A.1	List of abbreviations.....	15
A.2	List of symbols .....	16

## LIST OF FIGURES

2 - 1	Top module design for the IP block design. ....	4
2 - 2	Allocation and timing diagram for the Data Path Unit part of the IP. ....	5
2 - 3	Register transfer level RTL scheme of the IP Data Path Unit part of the IP. ....	6
2 - 4	Selected signals of simulation of division $N/D = 10 / 7$ . The correct result in $R0$ is obtained after two iterations (reg numberOfIterations). ....	8
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). ....	9
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). ....	9
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). ....	10
2 - 8	Selected signals of simulation of division $N/D = 10 / (519)$ . The correct result in $R0$ is obtained after two iterations (reg numberOfIterations). ....	10

**LIST OF TABLES**

2 - 1      Control signal encoding table for instructions to be processed by the Division Module. . . . . 6



# **1 Introduction**

This is the introduction.

## 2 Calculating the division of fixed point numbers

Usually, when using numerical methods to solve the transcendental 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 which 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 Raphson (NR) algorithm was developed. [2]

### 2.1 Newton Raphson 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*.

$$x_0 = \frac{48}{17} - \frac{32}{17}D, \quad (2 - 1)$$

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 synthesized 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.

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

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

$$\frac{dF(x_i)}{dx} = F'(x_i) = \frac{F(x_{i+1}) - F(x_i)}{x_{i+1} - x_i}. \quad (2 - 3)$$

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].

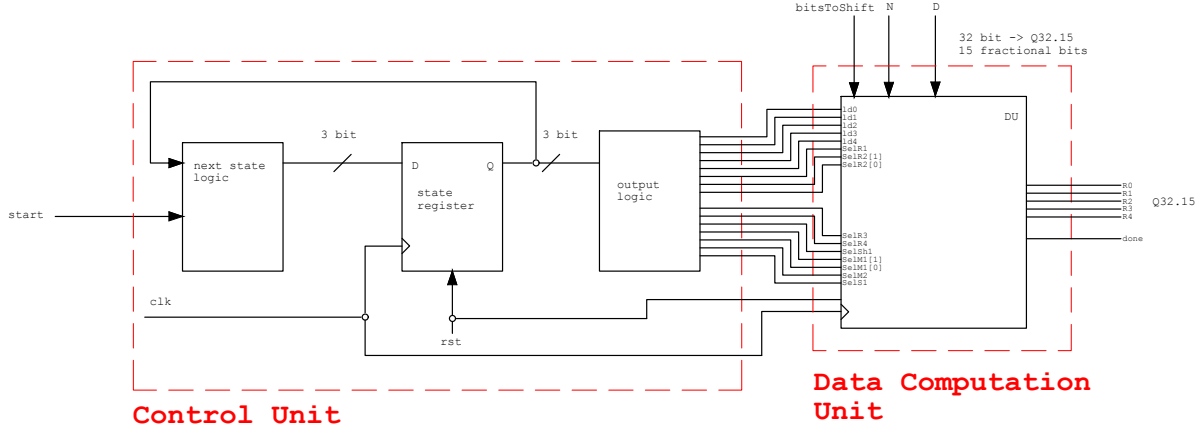


Figure 2 - 1 Top module design for the IP block design.

### 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.

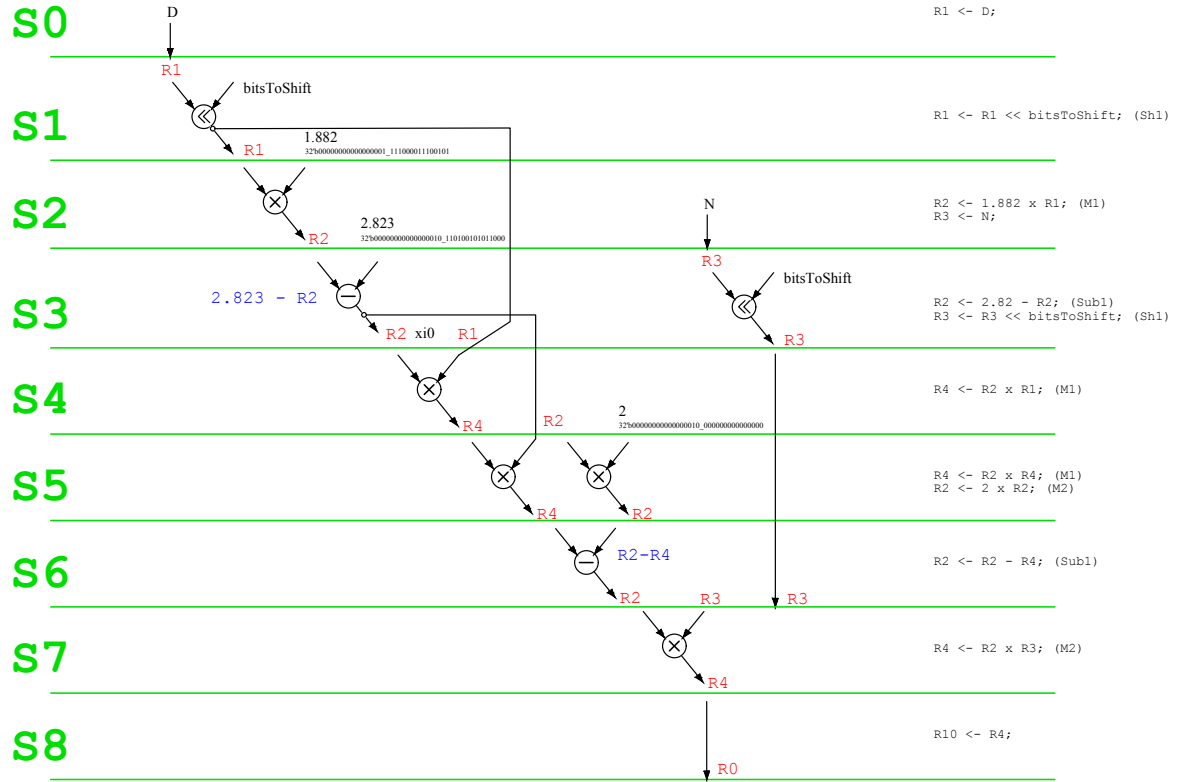


Figure 2 - 2 Allocation and timing diagram for the Data Path Unit part of the IP.

### 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 is 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 its actual sign and the 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.

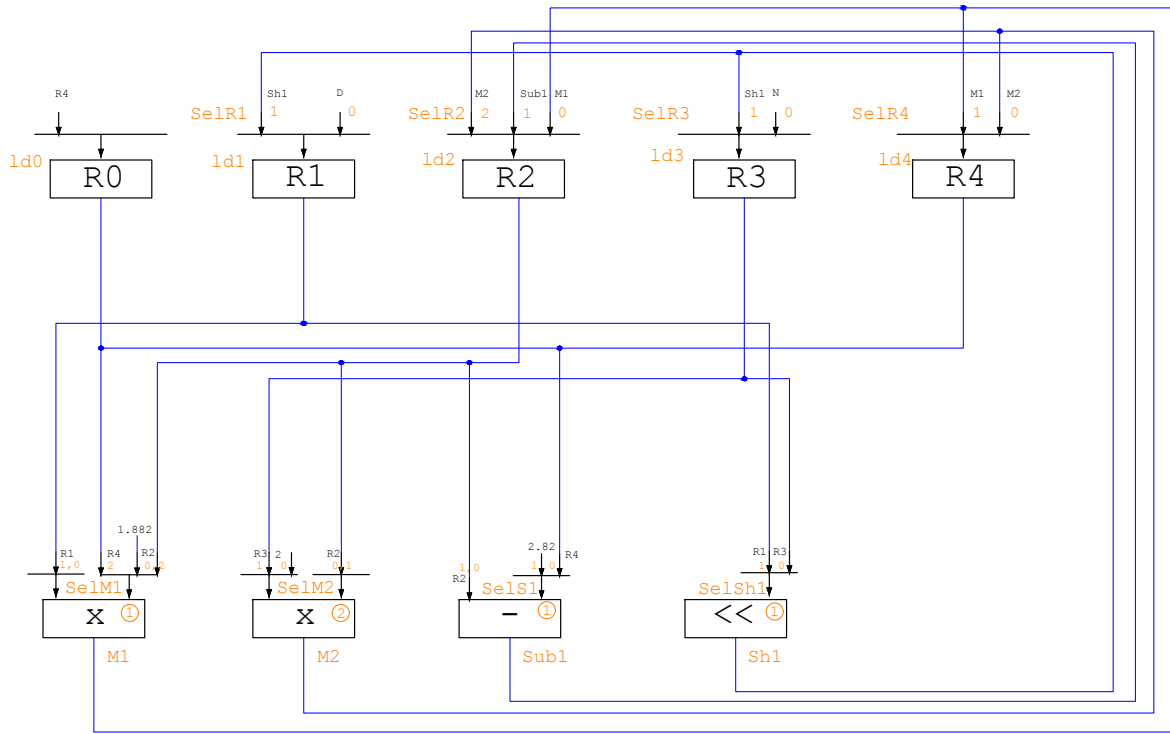


Figure 2 - 3 Register transfer level RTL scheme of the IP Data Path Unit part of the IP.

## 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  $S_0$ – $S_8$  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  $S_8$ , the FSM restarts at the state  $S_4$  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 ld0	13 ld1	12 ld2	11 ld3	10 ld4	9 SelR1	8 SelR2[1]	7 SelR2[0]	6 SelR3	5 SelR4	4 SelSh1	3 SelM1[1]	2 SelM1[0]	1 SelM2	0 SelS1	CV
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	2210h
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	1804h
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	18C0h
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	1528h
S6	$R2 \leftarrow R2 - R4; (S1)$	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1081h
S7	$R4 \leftarrow R2 \times R3; (M2)$	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	402h
S8	$R0 \leftarrow R4;$	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.

```
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 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 proper 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 in the 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.

```

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'
   b11111111111111111111_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 2 - 2 Pseudocode snippet for the Verilog simulation test bench.

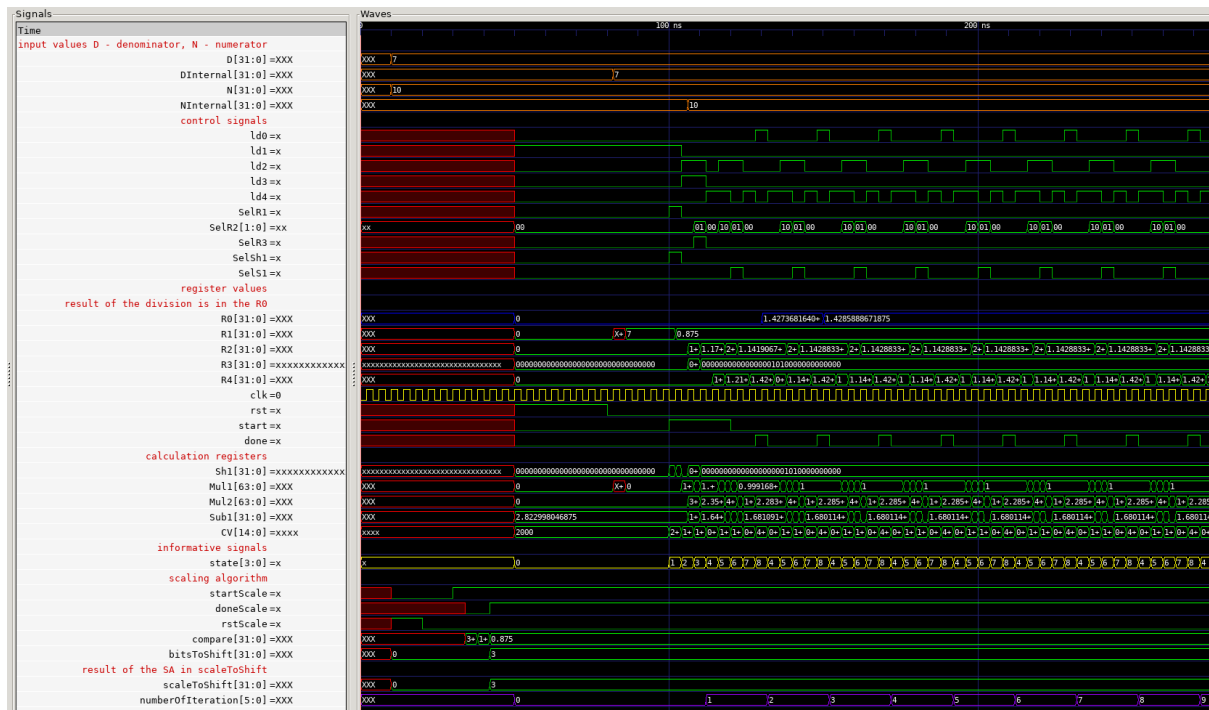


Figure 2 - 4 Selected signals of simulation of division  $N/D = 10 / 7$ . The correct result in  $R0$  is obtained after two iterations (reg numberOfIterations).

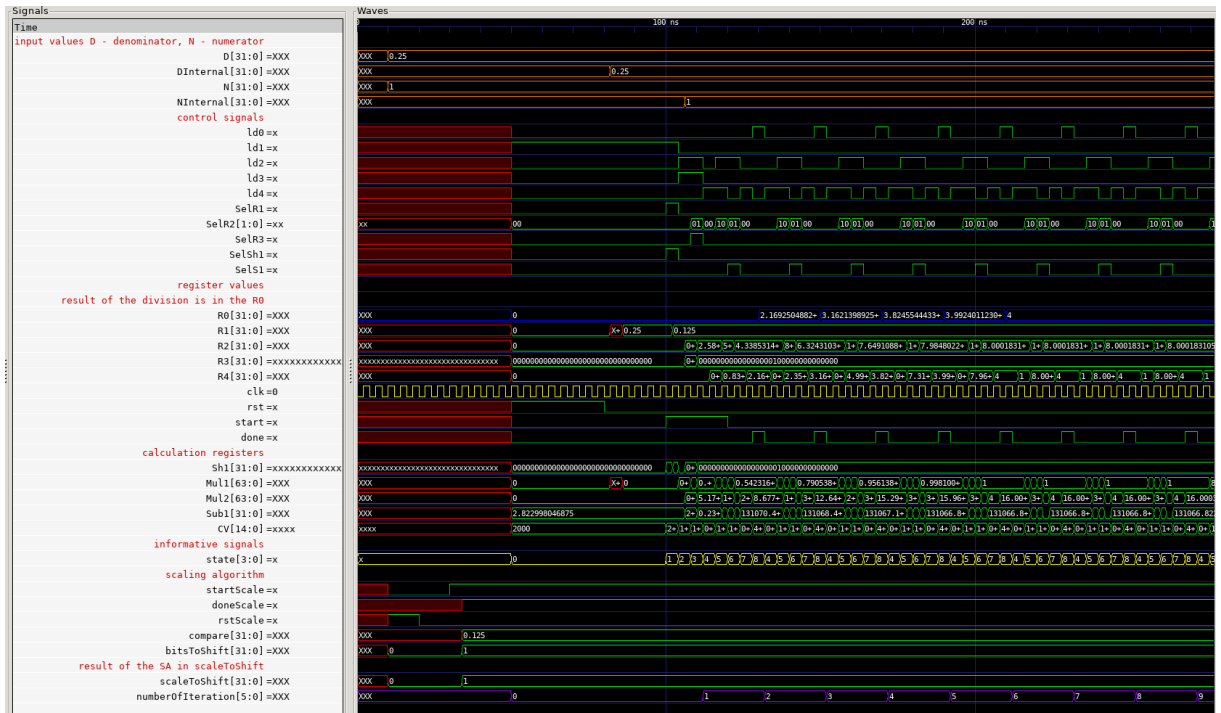


Figure 2 - 5 Selected signals of simulation of division  $N/D = 1 / 0.25$ . The correct result in R0 is obtained after five iterations (reg numberOfIterations).

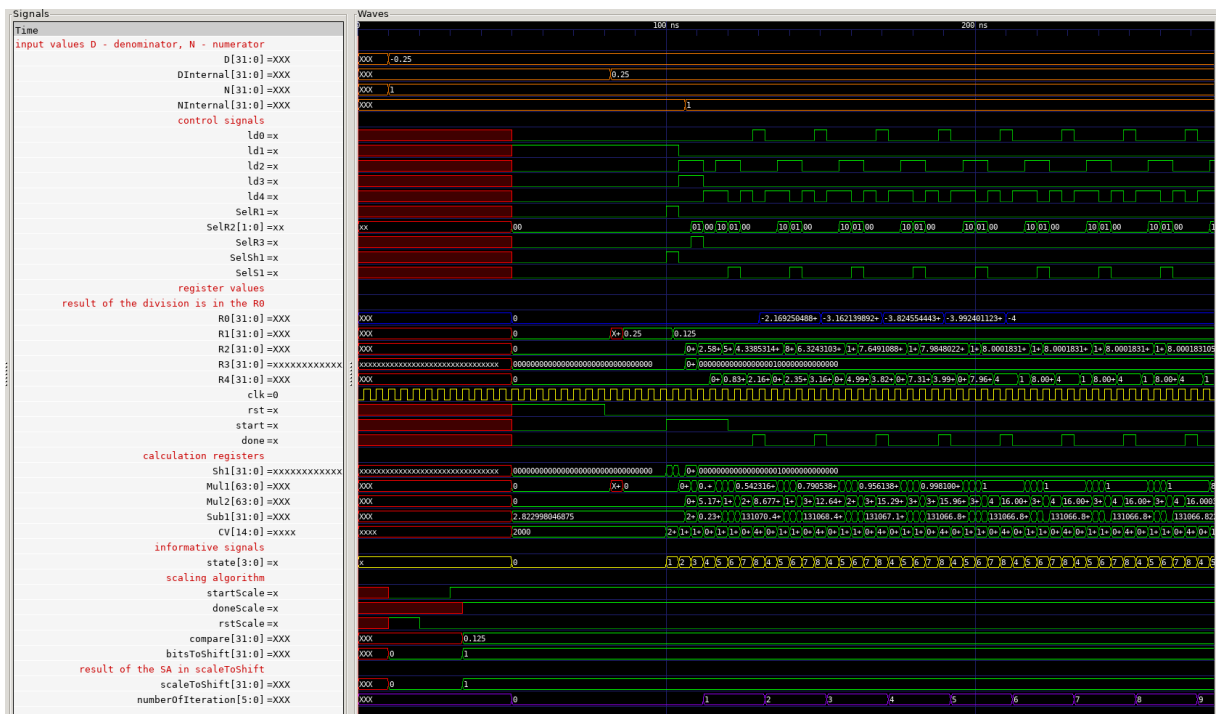


Figure 2 - 6 Selected signals of simulation of division  $N/D = 1 / (-0.25)$ . The correct result in R0 is obtained after five iterations (reg numberOfIterations).

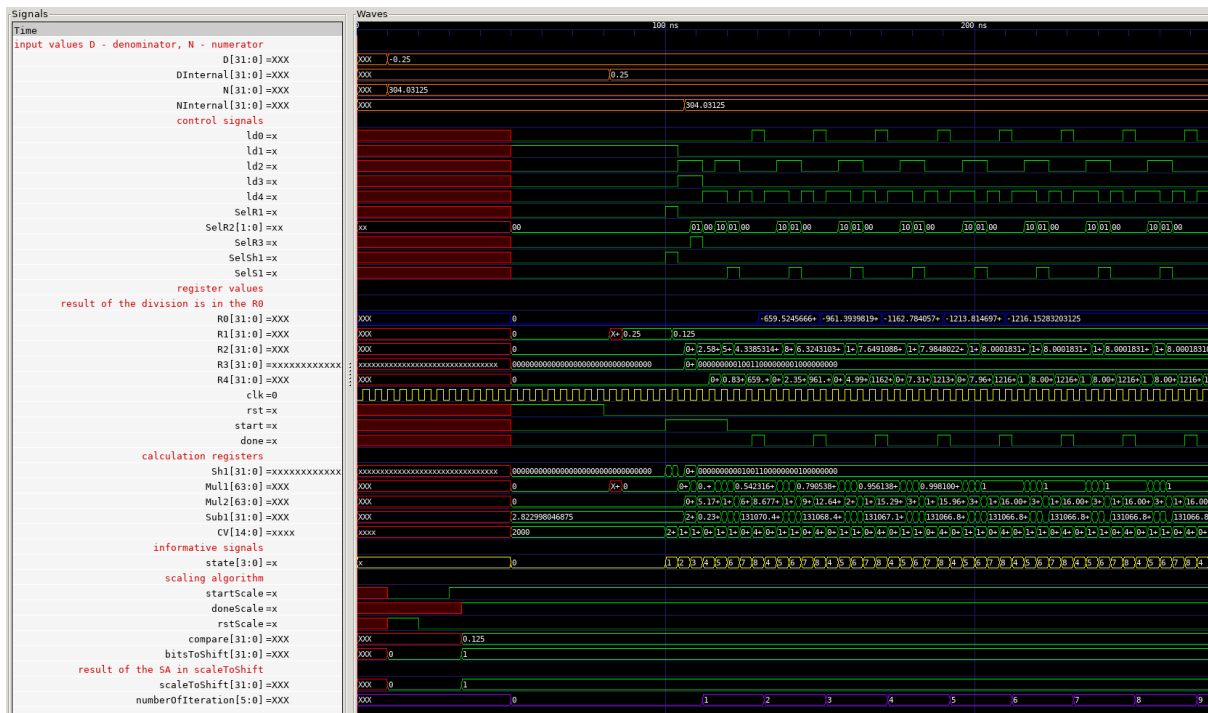


Figure 2 - 7 Selected signals of simulation of division  $N/D = 304.03215 / (-0.25)$ . The correct result in R0 is obtained after five iterations (reg numberOfIterations).

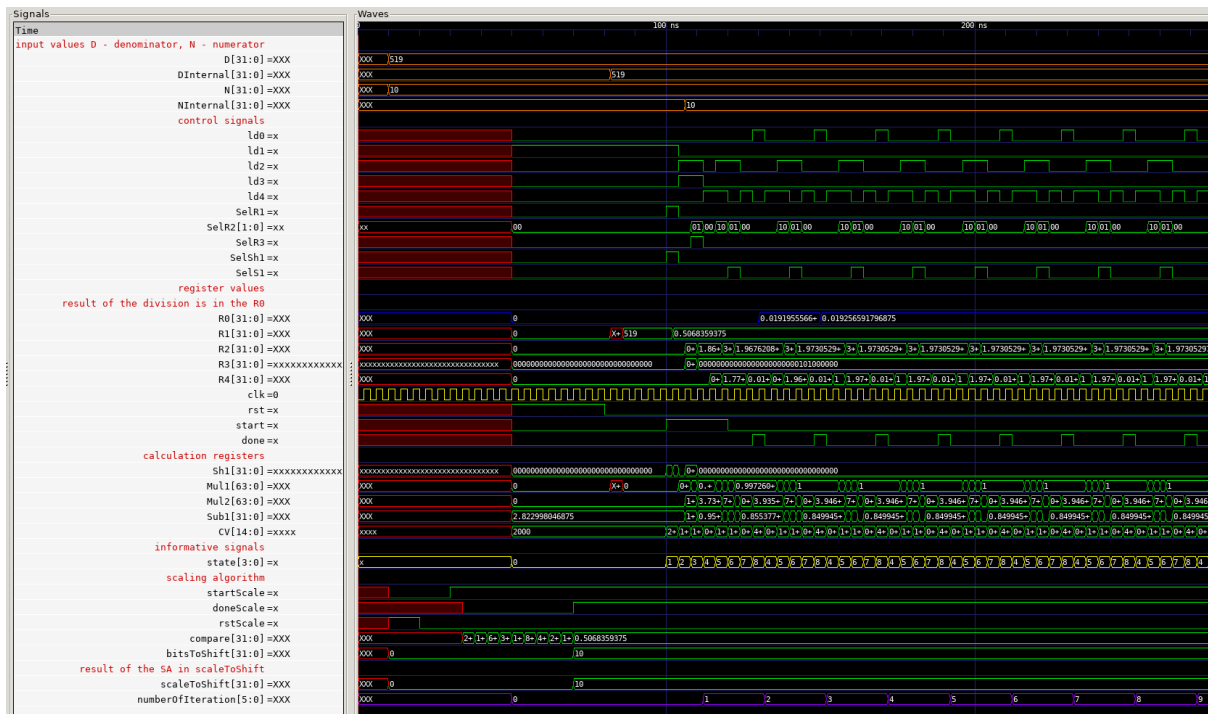


Figure 2 - 8 Selected signals of simulation of division  $N/D = 10 / (519)$ . The correct result in R0 is obtained after two iterations (reg numberOfIterations).



## 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](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] 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.
- [6] DIGILENT, Inc. Zybo. In: *Digilent Documentation* [online]. [B.r.] [visited on 2022-11-11]. Available from: <https://digilent.com/reference/programmable-logic/zybo/start>.
- [7] 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>.
- [8] 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>.
- [9] 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>.
- [10] 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>.
- [11] 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/>.
- [12] 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>.
- [13] XILINX, Inc. Downloads. In: *AMD Xilinx Downloads* [online]. [B.r.] [visited on 2022-11-19]. Available from: <https://www.xilinx.com/support/download.html>.
- [14] 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>.

- [15] 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>.
- [16] 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>.
- [17] 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>.
- [18] 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>.
- [19] 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>.
- [20] 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>.
- [21] 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>.
- [22] 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>.
- [23] 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>.
- [24] 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>.
- [25] 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>.
- [26] 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>.

- [27] 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](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**

<b>FPGA</b>	Field Programmable Gate Array
<b>FSM</b>	Finite State Machine
<b>IP</b>	Intellectual property
<b>NR</b>	Newton Raphson
<b>RTL</b>	Register Transfer Level

## A.2 List of symbols

$P_n$  (W) jmenovitý výkon stroje