Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Math2mat

Base cell

Departement TIN – Electrical Engineering

16. juin 2009

Author :	Samuel Tâche (samuel.tache@hefr.ch)
Project leader :	Claude Magliocco (claude.magliocco@hefr.ch)
Others invovled :	Philippe Crausaz (philippe.crausaz@hefr.ch)

Revision	Date	Author
1	24.03.09	Samuel Tâche

Table of Contents

1. Introduction

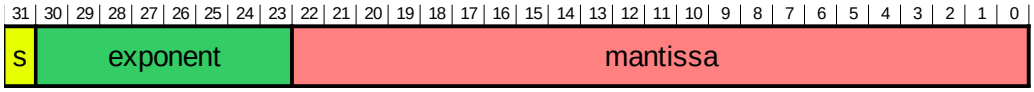
1.1. Purpose of the project

The base cells of Math2mat project are to perform basic operations such as +, -, *, /, sqrt. The project's goal is to optimize the speed of operations, that's why a pipelined version of these operators should be considered. Cells must manipulate floating point numbers. IEEE standard 754 has been imposed (single and double precision).

1.2. IEEE 754 format

L'IEEE 754 is a standard for the description of floating point numbers. Single precision format includes 32 bits, 1 for the sign, 8 for the exponent and 23 bits for the mantissa. Double precision format includes 64 bits, 1 for the sign, 11 for the exponent and 52 bits for the mantissa.

Single precision format:



With the standard IEEE 754, the equation 1.1 allows to compute the real number.

$$number = (-1)^{s} * 1.\text{mantissa} * 2^{(exponent - 127)}$$ **Equ. 1.1**

Round :

The IEEE 754 defines four rounding modes :

- Round toward +∞
- Round toward -∞
- Round toward 0
- Round to nearest

For the math2mat project, we only implemented one rounding mode, round to nearest.

This rounding is done by checking the value of the 24th bit generated during a computation of the mantissa in single precision.

If this bit is '1', the mantissa is incremented by '1', otherwise she's unchanged.

Example :

1.1100...101 —round→ 1.1100...11

1.1100...100 (Round bit) —round→ 1.1100...10

Exceptions :

Type	Exponent	Mantissa
Zeros	0	0
Denormalized numbers	0	different to 0
Normalized numbers	1 à $2^e - 2$	any
Infinity	$2^e - 1$	0
NaNs	$2^e - 1$	different to 0

Tableau 1: Exceptions IEEE 754 format

Math2mat project doesn't handle the denormalized numbers described by the IEEE 754 standard.

1.3. Cell interface

The interface of base cells (+, -, *, / and sqrt) was defined as follows (for square root, only one input of data) :

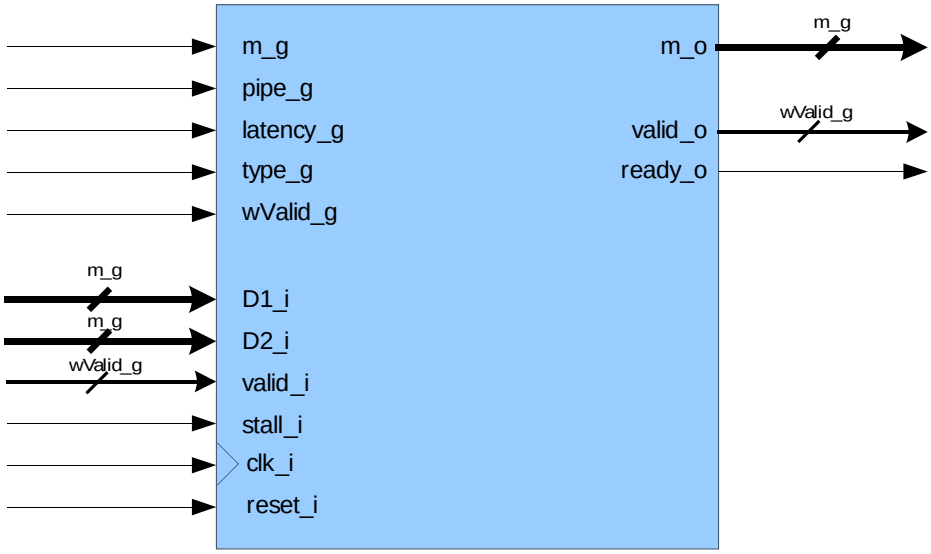


Illustration 1: Cellule de base

D1_i : Data 1 (IEEE 754 format).

D2_i : Data 2(IEEE 754 format).

valid_i : When high, the inputs are valid.

clk_i : System clock.

reset_i : System reset.

stall_i : When high, stop the operation (!enable).

m_g : Generic constant, defines the computation in single (m_g=32) or double (m_g=64) precision.

pipe_g : Generic constant, defines the combinatorial (= 0) or pipelined (!= 0) version.

latency_g : Generic constant, defines the duration of the operation for the combinatorial version.

type_g : Generic constant, defines the algorithm used for the operation.

wValid_g : Generic constant, defines the width of the signal valid_i

m_o : Result of the operation in floating point (IEEE 754 format).

valid_o : When high, the result is valid.

ready_o : When high, a new computation can be done.

Even if the component allows to set generically the width of data bus, some algorithms have so far been implemented in single precision, ie for a fixed bus width of 32 bits.

For some operation, it's useful to have a cell, which one must delay a computation that must be synchronized with another, so the next block was created. It's a shift register (delay_g = nb of registers) where each output m_o(i) has its validation signal valid_o(i).

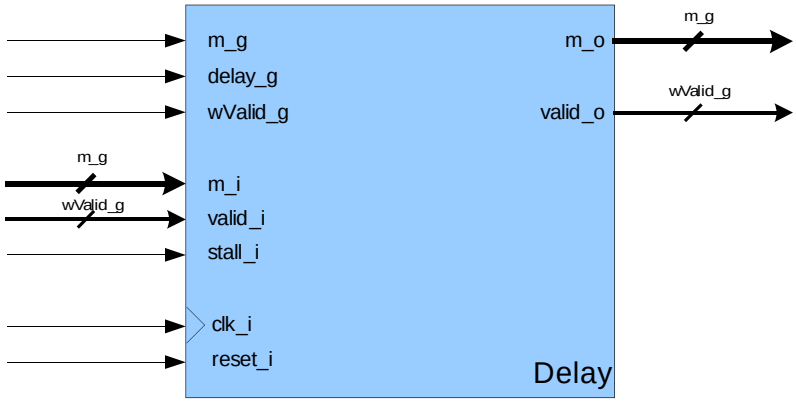


Illustration 2: Delay

1.4. Floating point algorithms

1.4.1. Addition / Subtraction

1. Take the largest exponent (= exponent of the result).
2. Shift the mantissa of the smallest number, the gab is the value of the difference of the exponents.
3. Addition / Subtraction of the mantissas.
4. Normalization of the result : mantissa between [1:2[, modification of the exponent if necessary
5. Round to nearest.

1.4.2. Multiplication

1. Addition of the exponents.
2. Multiplication of the mantissas.
3. Round to nearest.
4. Normalization of the result : mantissa between [1;2[, modification of the exponent if necessary.

1.4.3. Division

1. Subtraction of the exponents.
2. Division of the mantissas.
3. Round to nearest.
4. Normalization of the result: mantissa between[1;2[, modification of the exponent if necessary.

1.4.4. Square root

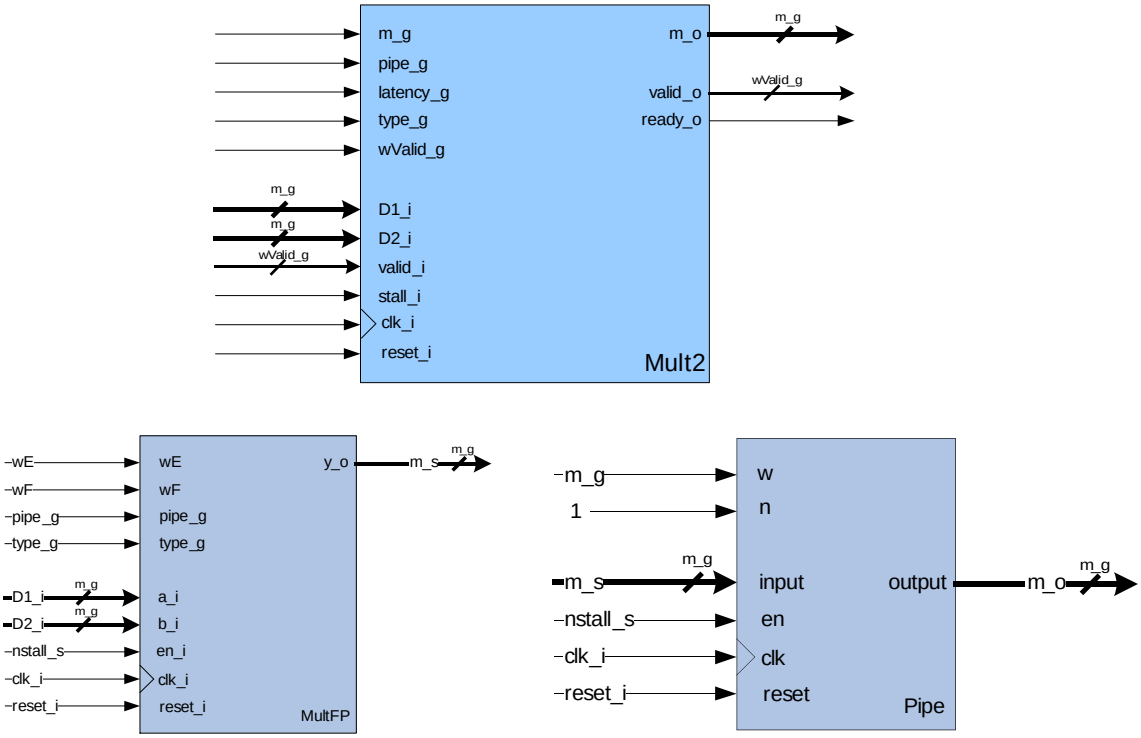
1. Division by 2 of the exponent.
2. Square root of the mantissa.
3. Round to nearest.
4. Normalization of the result : mantissa between [1;2[, modification of the exponent if necessary.

2. VHDL description of the base cells

2.1. Top level

The top level of a base cell (Add2, Mult2, Div2 ou Sqrt) is a structural description composed of 2 or more components :

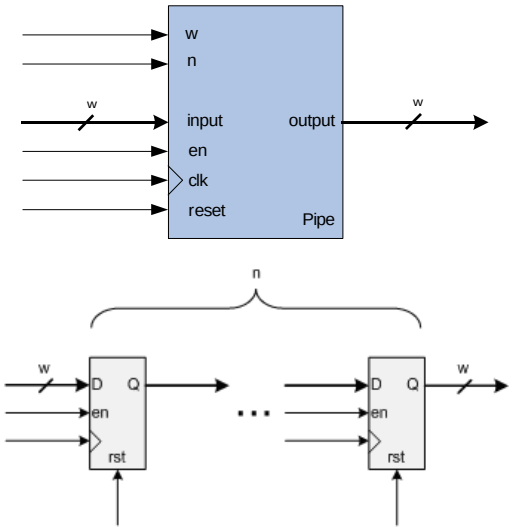
1. **AddFP, MultFP, DivFP, SqrtFP** : floating point computation (combinatorial or pipelined).
2. **Pipe** : shift register used in various cases.
3. *(AddFP_1, MultFP_1,...) : another algorithm. If so, the first algorithm becomes AddFP_0, MultFP_0.*



wE and wF generic constants define the width of exponent(wE) and mantissa(wF, F for Fraction).

2.2. Pipe

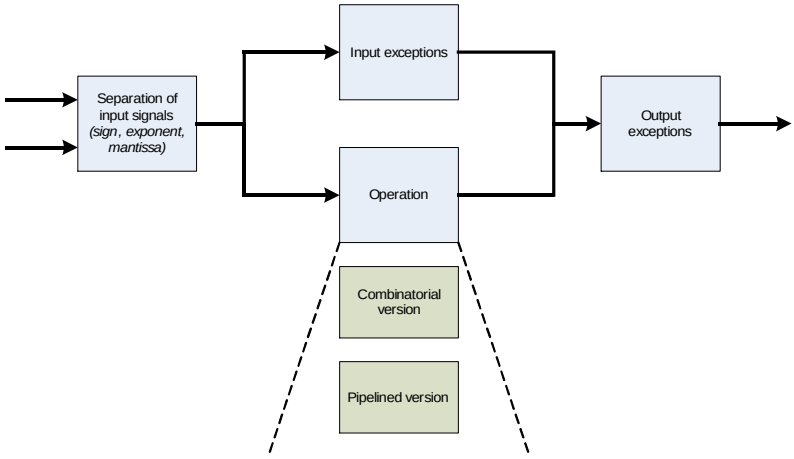
A component Pipe was created to generate a variable shift register with a variable width bus.



2.3. AddFP, MultFP, DivFP, SqrtFP

The operation in floating point was implemented as following :

- The input signals are separated in sign, exponent and mantissa.
- The operation is executed in combinatorial or pipelined version. The depth of the pipeline depends on the operation.
- In parallel, a test of inputs is executed to check the exceptions values . The exception is propagated to output to force the result.
- Another block determines if an exception occurred during the computation and force the result in output if necessary.



2.3.1. Input exceptions

Addition / Subtraction

	Nb	**0**	**+∞**	**-∞**	**NaN**
Nb	Nb	Nb	+∞	-∞	NaN
0	Nb	0	+∞	-∞	NaN
+∞	+∞	+∞	+∞	NaN	NaN
-∞	-∞	-∞	NaN	-∞	NaN
NaN	NaN	NaN	NaN	NaN	NaN

Multiplication

	Nb	**0**	**+∞**	**-∞**	**NaN**
Nb	Nb	0	+∞	-∞	NaN
0	0	0	NaN	NaN	NaN
+∞	+∞	NaN	+∞	-∞	NaN
-∞	-∞	NaN	-∞	+∞	NaN
NaN	NaN	NaN	NaN	NaN	NaN

Division

A / B	**Nb**	**0**	**+∞**	**-∞**	**NaN**
Nb	Nb	NaN	0	0	NaN
0	0	NaN	0	0	NaN
+∞	+∞	NaN	NaN	NaN	NaN
-∞	-∞	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN

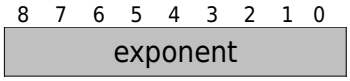
Square root

A	$\sqrt{A}$
Nb	Nb
<0	NaN
+∞	+∞
0	0
NaN	NaN

2.3.2. Output exceptions

During an operation, the result may be outside the limits of the standard IEEE 754 single or double precision. In this case, the result must be limited to values 0 or ∞.

For the single precision, the computation of exponents is realized on 9 bits (1 bit for carry). The treatment of exceptions is done as following :



1. exponent = "011111111" => result = ∞
2. exponent= "000000000" => result = 0
3. bit 8 and 7 = "10" => result = ∞
4. bit 8 and 7 = "11" => result = 0

Explications point 3 and 4 :

- *The largest possible result is obtained by multiplying two numbers with the exponent 011111110". During this operation, the exponent becomes :*

$$011111110+011111110-1111111=101111101$$

This example shows that when the result is too large and must apply the infinite, the two most significant bits of the exponent are "10".

- *When the division of a small number to a large one, the exponent result is :*

$$000000011-011000000+1111111=111000010$$

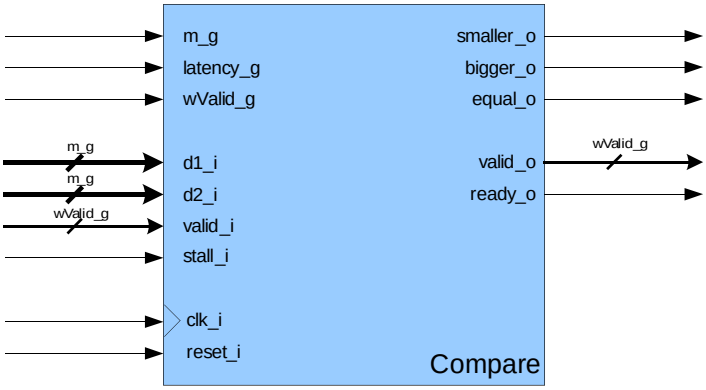
This example shows that when the result is too large and must apply the infinite, the two most significant bits of the exponent are "11".

2.3.3. Operation

A detailed description of the various operators are in the folder \doc.

2.4. Compare block

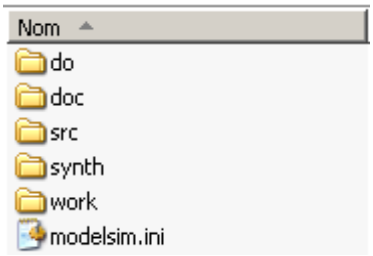
For the if...else implementation, we need to compare two values and know which number is bigger, smaller or if the two signals are equal. This block, completely combinatorial, performs this operation. It also includes the component pipe to propagate the signal valid_i to indicate when the result is valid (depending on the latency).



3. VHDL code organization

3.1. Project organization

The folder math2mat contains the following subfolders :



do : contains the files *.do (script Modelsim).
doc : project documentation.
src : contains the files *.vhd.
synth : contains the synthesis files.
work : library work.

3.2. Source files

Files name :

- A file with pkg_ is a package.
- A file with _tb is a testbench.
- A file without this two terms is a source code.

Description :

misc.vhd : this file contains the component pipe. If another useful component must be created, insert it in this file.

delay.vhd : cell delay to delay an output and its validation signal.

add2.vhd : cell addition.
mult2.vhd : cell multiplication.
div2.vhd : cell division.
sqrt.vhd : cell square root.

pkg_definition.vhd : Constants and functions declaration.

pkg_cellule.vhd : components add2, mult2, div2, sqrt and delay declaration.

Compilation script:

A script compile.do was realized to execute the compilation of source files in the correct order. This file is in the folder \do.

3.3. Version of the operators

The version of the operator is determined using the generic constants of the component. The generic constant type_g is used to determine the algorithm. Only the addition is implemented with one algorithm and therefore doesn't have this constant. The generic constant pipe_g determines the combinatorial or pipelined version. Latency_g constant determines the latency time (number of clock period) before the output is stable for the combinatorial version. And finally the generic constant m_g choose the single or double precision (be careful, not all algorithms are implemented in double precision).

The algorithms SRT4 for the division and SRT2 for the square root were taken from the library FPLibrary directed at ENS Lyon.

http://www.ens-lyon.fr/LIP/Arenaire/Ware/FPLibrary/

Add2 :

3 generic constants : m_g, pipe_g et latency_g

- m_g = 32 for single precision and m_g = 64 for double precision
- latency_g = 0 and pipe_g != 0 , pipelined version
- pipe_g = 0, combinatorial version, latency_g determines the time (number of clock period) before the output is stable.

Mult2 :

4 generic constants : m_g, pipe_g, latency_g and type_g

- m_g = 32 for single precision and m_g = 64 for double precision (double only for wired multiplier).
- type_g = 0 : multiplier carry save adder
  - pipe_g != 0 and latency_g = 0, pipelined version.
  - pipe_g = 0, combinatorial version, latency_g determines the time (number of clock period) before the output is stable.
- type_g = 1 : wired multiplier
  - pipe_g != 0 and latency_g = 0, pipelined version.
  - pipe_g = 0, combinatorial version, latency_g determines the time (number of clock period) before the output is stable.

Div2 :

4 generic constants : m_g, pipe_g, latency_g and type_g

- m_g = 32 for single precision and m_g = 64 for double precision (double only for SRT4 algorithm).
- type_g = 0 : SRT4 algorithm
  - pipe_g != 0 and latency_g = 0, pipelined version.

- pipe_g = 0, combinatorial version, latency_g determines the time (number of clock period) before the output is stable.

- type_g = 1 : algorithm "Array of soustracteurs"
  - pipe_g != 0, pipelined version.
  - pipe_g = 0, combinatorial version
- type_g = 2 : algorithm "Successive approximations".
  - pipe_g = 0, combinatorial version.

Sqrt :

4 generic constants : m_g, pipe_g, latency_g and type_g

- m_g = 32 for single precision and m_g = 64 for double precision
- type_g = 0 : SRT2 algorithm
  - pipe_g != 0 and latency_g = 0, pipelined version.
  - pipe_g = 0, combinatorial version, latency_g determines the time (number of clock period) before the output is stable.
- type_g = 1 : non-restoring algorithm
  - pipe_g != 0 and latency_g = 0, pipelined version.
  - pipe_g = 0, combinatorial version, latency_g determines the time (number of clock period) before the output is stable.

4. Synthesis results

All the synthesis results were made with the synthesizer XST (Xilinx) and the FPGA VirtexII xc2v1000-6bg575.

	Single precision				Double precision			
TYPE	**Slices**	**%**	**F_{max}**	**Pipes**	**Slices**	**%**	**F_{max}**	**Pipes**
Add2								
Combinatorial	664	12%			1801	35%		
Pipelined	408	7%	170 MHz	7	2714	53%	21 MHz	7
Mult2								
Wired multiplier								
Combinatorial	105	2%			345	6%		
Pipelined	155	3%	112 MHz	5	442	8%	76 MHz	5
Carry save adder								
Combinatorial	885	17%						
Pipelined	956	18%	203 MHz	7				
Div2								
SRT4								
Combinatorial	598	11%			2490	48%		
Pipelined	1049	20%	139 MHz	16	4166	81%	100 MHz	30
Array of substractor								
Combinatorial	2056	40%	6.8 MHz					
Pipelined	1904	37%	139 MHz	30				
Successive approximations								
Pipelined	3195	62%	6.3 MHz					
Sqrt								
SRT2								
Combinatorial	244	4%			901	17%		
Pipelined	400	7%	132 MHz	16	1438	28%	96 MHz	32
Non-restoring								
Combinatorial	426	8%			1698	33%		
Pipelined	701	13%	163 MHz	28	2778	54%	122 MHz	57

Synthesis with Virtex xc2v8000-5ff1152 :

1x Add2(pipelined) : 393 Slices, 1%

5x Add2(pipelined) : 1971 Slices, 4%

1x Mult2(pipelined) : 979 Slices, 2%

5x Mult2(pipelined) : 4900 Slices, 10%

The following results were achieved with the synthesizer Precision and the FPGA Virtex5 5VLX110FF676. (17280 Slices)

	Single precision				Double precision			
TYPE	**Slices**	**%**	**F_{max}**	**Pipes**	**Slices**	**%**	**F_{max}**	**Pipes**
Add2								
Combinatorial	106	0.61%	80 MHz	1	278	1.61%	60 MHz	1
Pipelined	122	0.71%	267 MHz	7	276	1.6%	170 MHz	7
Mult2								
Wired multiplier								
Combinatorial	27	0.16%	97 MHz	1	98	0.57%	60 MHz	1
Pipelined	47	0.30%	206 MHz	5	102	0.59%	76 MHz	5
Carry save adder								
Combinatorial	416	2.41%	87 MHz	1				
Pipelined	318	1.84%	338 MHz	7				
Div2								
SRT4								
Combinatorial	211	1.22%	23 MHz	1	863	5.00%	9 MHz	1
Pipelined	321	1.86%	242 MHz	16	1295	7.49%	201 MHz	30
Array of subtractors								
Combinatorial	263	1.52%	7 MHz	1				
Pipelined	547	3.17%	156 MHz	30				
Successive approximations								
Combinatorial	500	2.89%	7 MHz	1				
Sqrt								
SRT2								
Combinatorial	112	0.65%	24 MHz	1	434	2.51%	9 MHz	1
Pipelined	147	0.85%	230 MHz	16	530	3.07%	163 MHz	32
Non-restoring								
Combinatorial	183	1.06%	18 MHz	1	781	4.52%	6.7 MHz	1
Pipelined	261	1.51%	265 MHz	28	990	5.73%	216 MHz	57
Compare								
Combinatorial	17	0.10%	333 MHz					
Mult2 with a constant value (wired mulplier)								
Pipelined	40	0.23%	206 MHz					

5. Square root algorithm

5.1. Principle

Number representation in floating point is calculated as following:

$$F = M * 2^{E}$$

The square root of a number in floating point :

If E is even : $\sqrt{F} = \sqrt{M} * 2^{(E/2)}$

If E is odd : $\sqrt{F} = \sqrt{M/2} * 2^{((E/2)+1)}$

The square root of the mantissa was implemented with two different algorithms. The first was taken from an existing library (SRT2 algorithm) and the other was realized by ourselves (non-restoring algorithm).

5.2. Non-restoring square root

The square root of the mantissa is calculated through the following recurrence equation:

$$\begin{cases} X_0 = 0, \quad r_0 = M \\ q_0 = 1 \\ r_{i+1} = 2 * r_i - 2 * X_i * q_{i+1} - 2^{-(i+1)} \end{cases} \qquad q_{i+1} = \begin{cases} +1 & si\ r_i \geqslant 0 \\ -1 & sinon \end{cases}$$

Explication :

- r_i is the i_{th} partial remainder.
- X_i is the i_{th} bit of square root result.

The computation is divided into four stages :

1. the value of the partial remainder is shifted a bit left to get $2*r_i$.
2. The value of q_{i+1} is inferred by the sign of the partial remainder. If it's positive, $q_{i+1} = 1$ and if it's negative, $q_{i+1} = -1$.
3. the value of the square root is shifted a bit left to get $2*X_i$.
4. The computation of the partial remainder r_{i+1} can be done using the results of the first three stages.

Computation of X at the i_{th} step, example :

$$X_1 = 0.1_2$$
$$si\ q_2 = -1,\ X_2 = X_1 - 0.01_2 = 0.01_2$$
$$si\ q_2 = 1,\ \ X_2 = X_1 + 0.01_2 = 0.11_2$$

Example

Square root of 0.5 on 6 bits : *theoretical value*= $\sqrt{0.5} = 0.707107$

R_0 = 0.5		0	0	.	1	0	0	0	0		X_0=0
$2*r_0$		0	1	.	0	0	0	0	0	q_1=1	X_1=0.1
$-(2*X_0+2^{-1})$	-	0	0	.	1	0	0	0	0		
r_1		0	0	.	1	0	0	0	0		
$2*r_1$		0	1	.	0	0	0	0	0	q_2=1	X_2=0.11
$-(2*X_1+2^{-2})$	-	0	1	.	0	1	0	0	0		
r_2		1	1	.	1	1	0	0	0		
$2*r_2$		1	1	.	1	0	0	0	0	q_3=-1	X_3=0.101
$+(2*X_2-2^{-3})$	+	0	1	.	0	1	1	0	0		
r_3		0	0	.	1	1	1	0	0		
$2*r_3$		0	1	.	1	1	0	0	0	q_4=1	X_4=0.1011
$-(2*X_3+2^{-4})$	-	0	1	.	0	1	0	1	0	0	
r_4		0	0	.	0	1	1	1	0	0	
$2*r_4$		0	0	.	1	1	1	0	0	q_5=1	X_5=0.10111
$-(2*X_4+2^{-5})$	-	0	1	.	0	1	1	0	1	0	
r_4		1	1	.	0	1	1	1	1	0	
$2*r_4$		1	0	.	1	1	1	1	0	0	q_6=-1	$X_6=0.101101_2 = 0.703125$