

SQRT_inverse - algorytm szybkiego obliczania odwrotności pierwiastka

Leszek Błach
Damian Brzeziński

2021

https://github.com/Haggs3/SDUP_InvSqrt

Spis treści

0	Algorytm	3
0.1	Wstęp	3
0.2	Działanie algorytmu	3
1	Algorytm w postaci diagramu	5
2	Model behawioralny algorytmu	8
2.1	Opis oraz listing algorytmu obliczania odwrotnego pierwiastka	8
2.2	Opis oraz listingi algorytmów mnożących	10
2.3	Opis oraz listing algorytmu odejmującego	16
3	Stworzenie modułu, określenie wejść/wyjść	19
4	Testbench modelu behawioralnego	20
4.1	Wstęp	20
4.2	Testbench oraz wyniki dla układu obliczającego odwrotność pierwiastka	21
4.3	Testbench oraz wyniki dla układów mnożących	24
4.4	Testbench oraz wyniki dla układu odejmującego	29
4.5	Podsumowanie otrzymanych wyników	31
5	Model syntezy algorytmu oraz jego testbench	32
6	AXI, Zynq	33
7	Uruchomienie na sprzęcie	35
8	Wersja potokowa układu	38
8.1	Wstęp	38
8.2	Algorytm z przetwarzaniem potokowym w postaci diagramu	39
8.3	Odwrotność pierwiastka	42
8.3.1	Układ	42
8.3.2	Testbench oraz wyniki	43
8.4	Układ inicjalizujący	45
8.5	Układ mnożący	47
8.5.1	Układ	47
8.5.2	Testbench oraz wyniki	52
8.6	Układ odejmujący	54
8.6.1	Układ	54
8.6.2	Testbench oraz wyniki	59
8.7	Podsumowanie uzyskanych wyników	61

8.8	AXI, Zynq	62
8.9	Uruchomienie na sprzęcie	64

0 Algorytm

0.1 Wstęp

Projekt opiera się o korzystanie istniejącego rozwiązania algorytmu do obliczania odwrotnego pierwiastka z zadanej liczby. Algorytm ten ma wiele praktycznych zastosowań (np. normalizacja długości wektora), ponieważ liczenie odwrotności pierwiastka w tradycyjny sposób zajmuje dużo czasu. Celem projektu jest zaimplementowanie tego algorytmu na układzie FPGA wchodzącego w skład Zynq Evaluation and Development Kit, z wykorzystaniem oprogramowania Xilinx Vivado 2018.3.

0.2 Działanie algorytmu

W normalnej sytuacji, aby policzyć odwrotny pierwiastek z danej liczby, trzeba go po prostu obliczyć używając następującego kodu:

```
1 float y = 1 / sqrt(x);
```

Kod ten jednak jest bardzo nieoptymalny w swojej szybkości działania. Twórcy gry Quake Arena stworzyli algorytm, który pozwolił znacznie przyspieszyć tę operację. Jeżeli policzymy logarytm o podstawie 2 z odwrotności pierwiastka otrzymamy:

$$\log_2 \left(\frac{1}{\sqrt{y}} \right) = \log_2 \left(y^{-\frac{1}{2}} \right) = -\frac{1}{2} \log_2 (y) \quad (0.1)$$

Wartość zmiennej typu floating point zapisana jest na 32 bitach w postaci:

[31] - bit znaku [S]

[30 : 23] - eksponenta [E]

[22 : 0] - mantysa [M]

Wartość liczby zapisanej w tym formacie obliczana jest jako:

$$(-1)^S \cdot \left(1 + \frac{M}{2^{23}} \right) \cdot 2^{E-127} \quad (0.2)$$

Wykorzystując przybliżenie $\log_2 (1 + x) \approx x + \mu$ dla $x < 1$, gdzie μ to współczynnik korygujący błąd przybliżenia, oraz zakładając, że $x > 0$ oraz obliczając logarytm o podstawie 2 z tej wartości otrzymujemy:

$$\log_2 \left(\left(1 + \frac{M}{2^{23}} \right) \cdot 2^{E-127} \right) = \log_2 \left(1 + \frac{M}{2^{23}} \right) + E - 127 \approx \frac{M}{2^{23}} + \mu + E - 127 \quad (0.3)$$

```

1  float Q_rsqrt( float number )
2  {
3      long i;
4      float x2, y;
5      const float threehalfs = 1.5F;
6
7      x2 = number * 0.5F;
8      y  = number;
9      i  = * ( long * ) &y;                                // evil floating point bit level hacking
10     i  = 0x5f3759df - ( i >> 1 );                          // what the duck?
11     y  = * ( float * ) &i;
12     y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
13     //    y  = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed
14
15     return y;
16 }

```

Listing 1: Pierwotna implementacja algorytmu do obliczania odwrotności pierwiastka

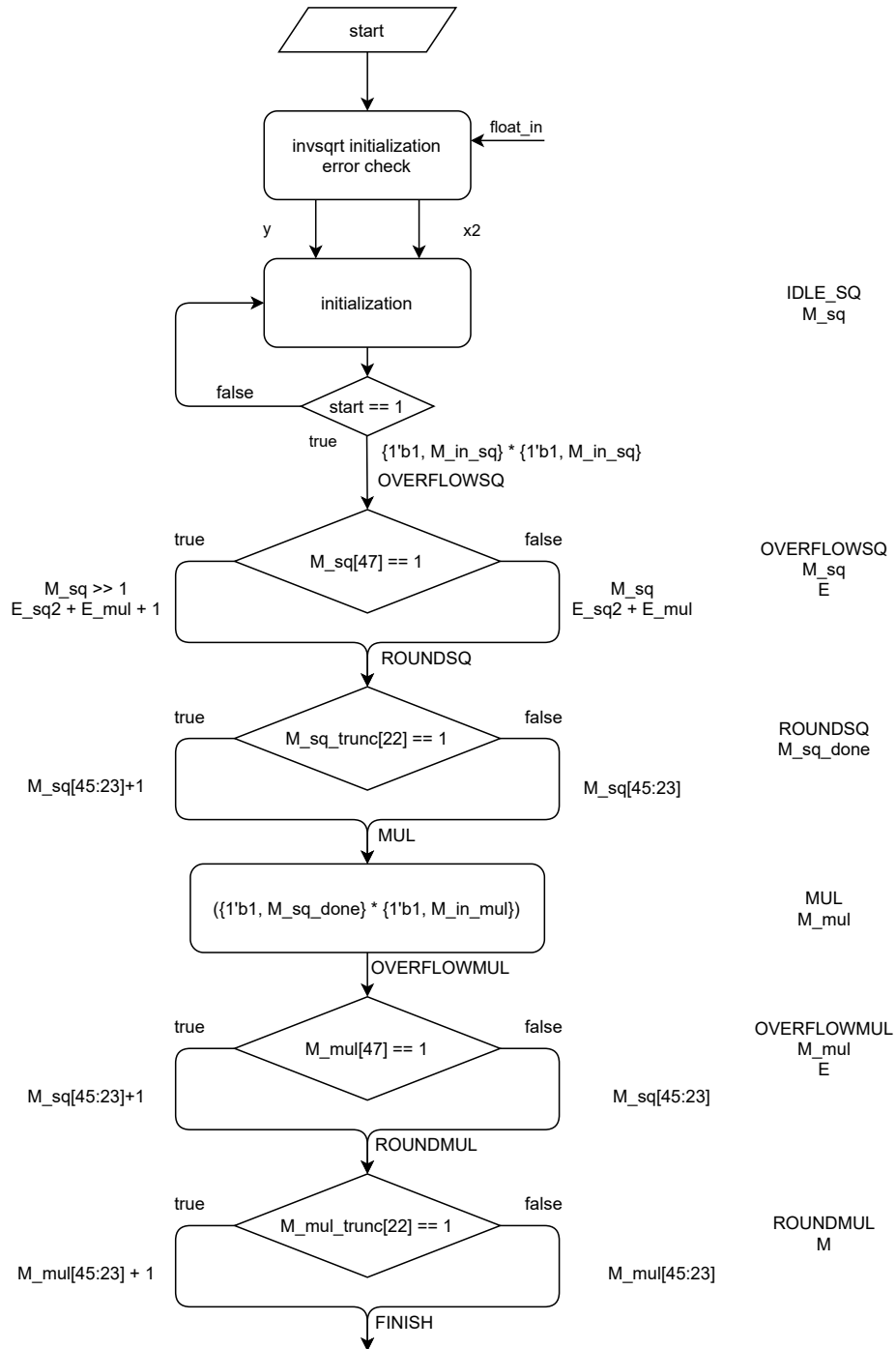
Przekształcając to równanie otrzymamy:

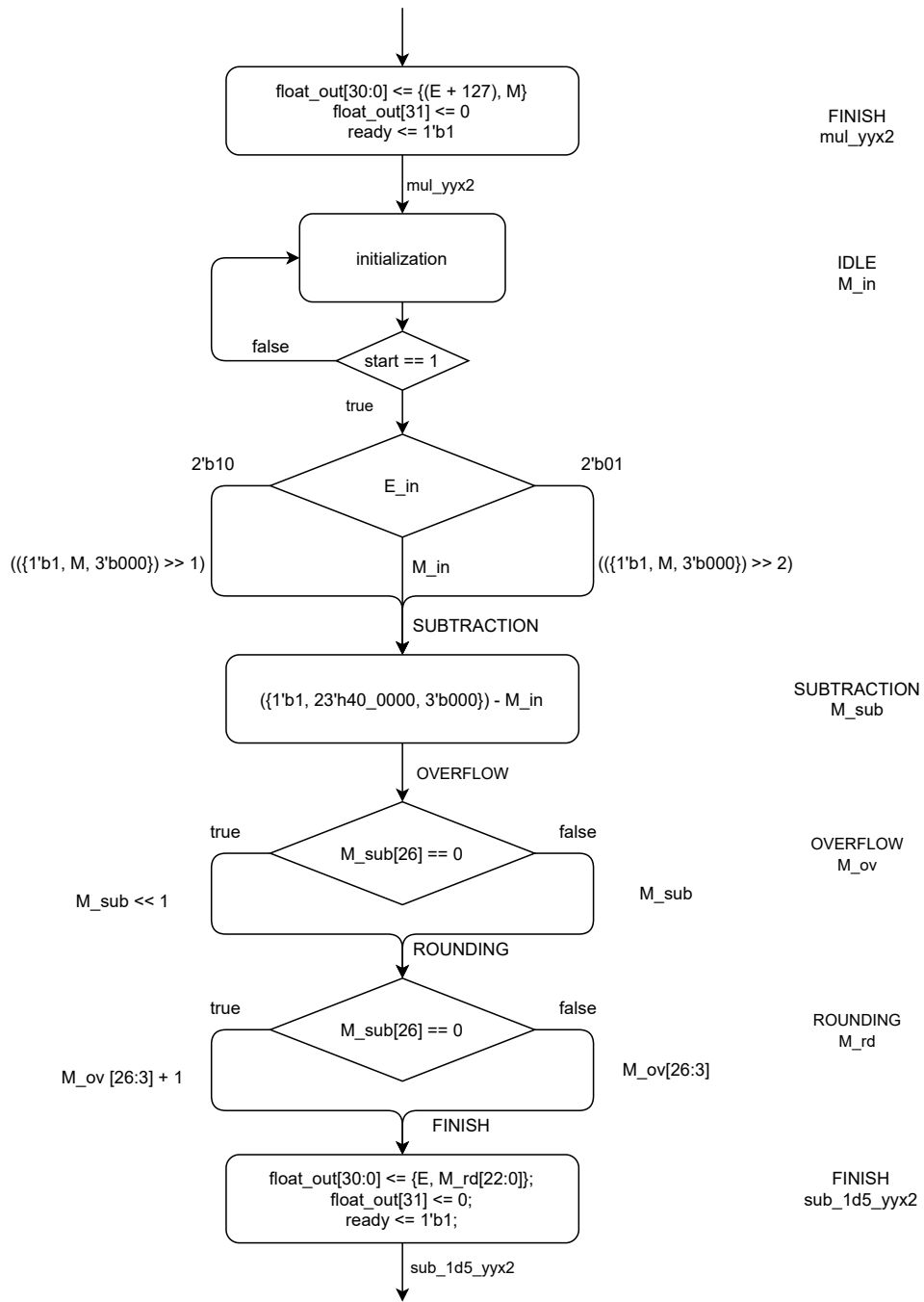
$$\frac{1}{2^{23}} \left(M + 2^{23} \cdot E \right) + \mu - 127 \quad (0.4)$$

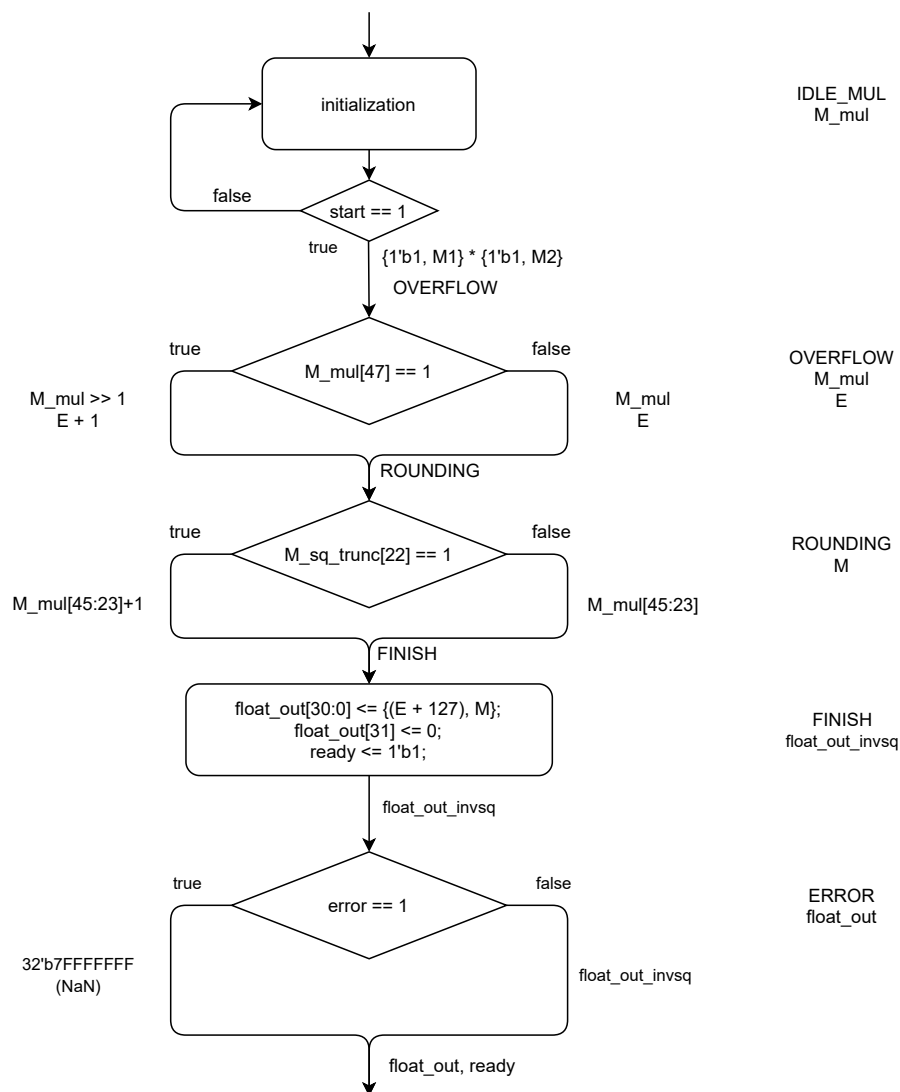
Wartość $M + 2^{23} \cdot E$ jest reprezentacją bitową dodatniej liczby w formacie floating point. Zatem reprezentacja bitowa liczby zmiennoprzecinkowej jest jej przeskalowanym logarytmem.

W kodzie przedstawionym wyżej przedstawiono oryginalny algorytm z gry Quake Arena. W 9 linii bitowa reprezentacja liczby zmiennoprzecinkowej zostaje przekonwertowana na zmienną typu long. W 10 linii następuje operacja przedstawiona w równaniu (0.1): $-(i \gg 1)$ to zanegowana reprezentacja bitowa danej liczby zmiennoprzecinkowej podzielona przez 2. Liczba heksadecymalna 0x5f3759df to obliczona wartość o którą trzeba przeskalować logarytm. W 11 linii następuje konwersja odwrotna do liczby zmiennoprzecinkowej. W tym momencie jest już obliczona przybliżona wartość odwrotnego pierwiastka. W 12 i 13 linii wykonywane jest przybliżenie metodą Newtona. Mając równanie $f(y) = \frac{1}{y^2} - x$ oraz jego pochodną $f'(y) = \frac{-2}{y^3}$ można wyznaczyć kolejne przybliżenia wartości odwrotnego pierwiastka: $y_{n+1} = y_n \cdot (1.5 - \frac{x}{2}y^2)$. Wartość zmiennej y jest na tyle blisko dokładnej wartości, że wystarczy jedna iteracja metodą Newtona aby uzyskać zadowalający wynik.

1 Algorytm w postaci diagramu







Rysunek 1: Diagram algorytmu obliczającego odwrotność pierwiastka

2 Model behawioralny algorytmu

2.1 Opis oraz listing algorytmu obliczania odwrotnego pierwiastka

Model algorytmu do obliczania odwrotności pierwiastka składa się z modułów, które służą do odejmowania, mnożenia dwóch liczb oraz mnożenia dwóch liczb, w tym jednej podniesionej do kwadratu.

Algorytm ten posiada 4 wejścia - wejście zegara `clk`, reset `rst`, wejście sygnału start `start` oraz wejściową liczbę typu floating point `float_in`, oprócz tego układ ma dwa wyjścia - wyjście `ready` informujące, że układ skoczył pracę oraz wyjściową liczbę zmiennoprzecinkową `float_out`, będącą odwrotnością pierwiastka kwadratowego z `float_in`:

$$\text{float_out} = \frac{1}{\sqrt{\text{float_in}}}$$

W nadrzędnym module zaimplementowany został algorytm detekcji błędów, którego zadaniem jest wykrywanie, czy z wejściowej liczby zmiennoprzecinkowej da się obliczyć odwrotność pierwiastka. W przypadku gdy na wejście podana zostanie liczba ujemna, zero, NaN, `+inf` lub `-inf` układ na wyjście podaje NaN.

Poszczególne moduły zostają uruchomione z użyciem sygnałów `start/ready`. Na wejście pierwszego modułu podawany jest sygnał `start`, który po skończeniu działania generuje sygnał `ready`, który jest sygnałem `start` dla kolejnego modułu.

Moduł `invsqrt.v` przygotowuje również wstępnie dane do dalszej obróbki, rozdziela wejściową liczbę zmiennoprzecinkową na eksponentę i mantysę - znak ze względu na obliczanie pierwiastka kwadratowego nie jest uwzględniany nigdzie oprócz sprawdzania poprawności danych wejściowych. Oprócz tego wejściowa liczba zostaje wstępnie podzielona na wartości `x2` oraz `y` jak w oryginalnym algorytmie. Poniżej znajduje się implementacja układu w języku Verilog.

Listing 2: moduł `invsqrt.v`

```
1  `timescale 1ns / 1ps
2
3  module invsqrt (clk, rst, start, float_in, float_out, ready);
4
5  input wire start;
6  input wire clk;
7  input wire rst;
8  input wire [31:0] float_in;
9
10 output wire [31:0] float_out;
```

```

11  output reg ready;
12
13  wire [31:0] x2, y, mul_yyx2, sub_1d5_yyx2, fp;
14  wire [22:0] M_in;
15  wire [7:0] E_in_x2;
16  wire rdy_1, rdy_2, rdy_3;
17  wire [31:0] float_out_invsq;
18  wire error;
19  assign M_in = float_in[22:0];
20  assign E_in_x2 = float_in[30:23] - 1;
21
22  assign x2 = {1'b0, E_in_x2, M_in};
23  assign y = 32'h5f3759df - (float_in >> 1);
24
25  assign error = ((float_in == 32'b0) || (float_in[30:23] == 8'hFF) ||
    ↪ (float_in[31] == 1'b1)) ? 1'b1 : 1'b0;
26  assign float_out = (error == 1'b0) ? float_out_invsq : 32'h7FFFFFFF;
27
28  float_sq_mul  multiply_yyx2      (clk, rst, start, y,          x2, mul_yyx2,
    ↪ rdy_1);
29  float_sub_1d5 subtract_1d5_yyx2 (clk, rst, rdy_1, mul_yyx2, sub_1d5_yyx2,
    ↪ rdy_2);
30  float_mul     multiply_y_sub     (clk, rst, rdy_2, y,          sub_1d5_yyx2,
    ↪ float_out_invsq, rdy_3);
31
32  always @(posedge clk)
33      if(rst == 1'b1) begin
34          ready <= 0;
35      end else begin
36          if (start == 1'b1) begin
37              ready <= 0;
38          end else if (rdy_3 == 1'b1) begin
39              ready <= 1;
40          end else begin
41              ready <= ready;
42          end
43      end
44  endmodule

```

2.2 Opis oraz listingi algorytmów mnożących

Matematycznie mnożenie liczb typu floating point można zapisać jako:

$$\left(1 + \frac{M_1}{2^{23}}\right) 2^{E_1-127} \cdot \left(1 + \frac{M_2}{2^{23}}\right) 2^{E_2-127}$$

Co po przemnożeniu daje nam:

$$2^{E_1-127+E_2-127} \left(1 + \frac{M_1}{2^{23}} + \frac{M_2}{2^{23}} + \frac{M_1 \cdot M_2}{2^{46}}\right)$$

Jednak ze względu na złożoność powyższej operacji (trzy dodawania i mnożenie), działania na mantysach ograniczone zostały do:

$$\left(1 + \frac{M_1}{2^{23}}\right) \cdot \left(1 + \frac{M_2}{2^{23}}\right)$$

Gdzie występuje tylko jedno mnożenie (dodawanie 1 można zrealizować jako konkatencję bitów).

Moduły te opierają się o tę samą zasadę działania, z tym, że moduł `float_sq_mul.v` zawiera w sobie dodatkowo obliczanie kwadratu z jednej z liczb wejściowych. Obydwa moduły mają taką samą ilość wejść i wyjść. Wejścia tak jak w module `invsqrt`: wejście zegara `clk`, reset `rst`, wejście sygnału start `start`, do tego wejściowe liczby do przemnożenia. W przypadku zwykłego układu mnożącego wejścia te to `float_in_1` oraz `float_in_2`, ponieważ mnożenie jest naprzemienne i wejście nie ma znaczenia. W przypadku układu z podnoszeniem do kwadratu wejścia mają znaczenie - zostały nazwane bardziej sugestywnie - `float_in_sq` oraz `float_in_mul`.

Układy mnożące zostały zrealizowane jako maszyny stanów, które w swojej budowie mają znamiona układu pracującego potokowo - każda dana przechodzi przez każdy ze stanów maszyny, niezależnie od tego czy będzie na niej wykonywana operacja w danym kroku.

W układach mnożących wykorzystany został algorytm do mnożenia liczb zmiennoprzecinkowych. W algorytmie tym wartości eksponent po odjęciu offsetu zostają do siebie dodane (mnożenie dwóch potęg o takiej samej podstawie realizowane jako suma wykładników), a mantysy są przez siebie mnożone. W przypadku mnożenia dwóch liczb 23 bitowych końcowy wynik ma 47 bitów. W przedstawionym algorytmie wynik mnożenia determinuje, czy otrzymany wynik zostanie poprawiony. Jeżeli wartość najstarszego bitu wyniku mnożenia jest równa 1, wynik tego mnożenia jest dzielony przez dwa, a wartość eksponenty zostaje zwiększona o jeden.

W przypadku układu z zaimplementowanym podnoszeniem liczby do kwadratu w porównaniu do układu mnożącego występuje jedno mnożenie więcej - działania na eksponentach zostały zamknięte jako `assign` w którym eksponenta liczby

podniesionej do kwadratu to pierwotna eksponenta przesunięta o jedno miejsce w lewo (odpowiednik mnożenia przez dwa).

Pozostałe kroki odbywają się jak w normalnym algorytmie do mnożenia (mnożenie mantys, sprawdzenie przepełnienia). W algorytmie zaimplementowane jest również zaokrąglanie liczb - jeżeli pierwszy z odrzuconych po mnożeniu bitów jest równy 1, wartość mantysy zostaje zwiększona o jeden (zaokrąglanie do najbliższej liczby). Kod obydwu modułów został umieszczony poniżej:

Listing 3: moduł float_mul.v

```
1  `timescale 1ns / 1ps
2
3  module float_mul (clk, rst, start, float_in_1, float_in_2, float_out, ready);
4
5  input wire start;
6  input wire clk;
7  input wire rst;
8  input wire [31:0] float_in_1;
9  input wire [31:0] float_in_2;
10
11 output reg [31:0] float_out;
12 output reg ready;
13
14 wire signed [7:0] E1, E2;
15 reg signed [7:0] E;
16 wire [22:0] M1, M2, M_trunc;
17 reg [22:0] M;
18 reg [47:0] M_mul;
19 reg [1:0] state;
20
21 localparam IDLE_MUL = 2'b00,
22             OVERFLOW = 2'b01,
23             ROUNDING = 2'b10,
24             FINISH = 2'b11;
25
26 assign E1 = float_in_1[30:23] - 127;
27 assign E2 = float_in_2[30:23] - 127;
28
29 assign M1 = float_in_1[22:0];
30 assign M2 = float_in_2[22:0];
31
32 assign M_trunc = M_mul[22:0];
```

```

33
34 always @ (posedge clk)
35     if(rst == 1'b1) begin
36         float_out <= 32'b0;
37         ready <= 1'b0;
38         state <= IDLE_MUL;
39         E <= 8'b0;
40         M <= 23'b0;
41         M_mul <= 48'b0;
42     end else begin
43         case(state)
44             IDLE_MUL: begin
45                 E <= E1 + E2;
46                 ready <= 1'b0;
47                 if (start == 1'b1) begin
48                     M_mul <= ({1'b1, M1} * {1'b1, M2});
49                     state <= OVERFLOW;
50                 end else begin
51                     state <= IDLE_MUL;
52                 end
53             end
54             OVERFLOW: begin
55                 if (M_mul[47] == 1) begin
56                     M_mul <= M_mul >> 1;
57                     E <= E + 1;
58                 end else begin
59                     M_mul <= M_mul;
60                     E <= E;
61                 end
62                 state <= ROUNDING;
63             end
64             ROUNDING: begin
65                 if(M_trunc[22] == 1) begin
66                     M <= M_mul[45:23] + 1;
67                 end else begin
68                     M <= M_mul[45:23];
69                 end
70                 state <= FINISH;
71             end
72             FINISH: begin
73                 float_out[30:0] <= {(E + 127), M};
74                 float_out[31] <= 0;

```

```

75         ready <= 1'b1;
76         state <= IDLE_MUL;
77     end
78 endcase
79 end
80 endmodule

```

Listing 4: modul float_sq_mul.v

```

1  `timescale 1ns / 1ps
2
3  module float_sq_mul (clk, rst, start, float_in_sq, float_in_mul, float_out,
4      ↪ ready);
5
6      input wire start;
7      input wire clk;
8      input wire rst;
9      input wire [31:0] float_in_sq;
10     input wire [31:0] float_in_mul;
11
12     output reg [31:0] float_out;
13     output reg ready;
14
15     wire signed [7:0] E_sq, E_mul;
16     wire signed [7:0] E_sq2;
17     reg signed [7:0] E;
18     wire [22:0] M_in_sq, M_in_mul, M_sq_trunc, M_mul_trunc;
19     reg [22:0] M_sq_done, M;
20     reg [47:0] M_mul, M_sq;
21     reg [2:0] state;
22
23     localparam IDLE_SQ = 3'b000,
24                OVERLOWSQ = 3'b001,
25                ROUNDSQ = 3'b010,
26                MUL = 3'b011,
27                OVERFLOWMUL = 3'b100,
28                ROUNDMUL = 3'b101,
29                FINISH = 3'b110;
30
31     assign E_sq = float_in_sq[30:23] - 127;
32     assign E_mul = float_in_mul[30:23] - 127;

```

```

32  assign E_sq2 = E_sq << 1;
33
34  assign M_in_sq = float_in_sq[22:0];
35  assign M_in_mul = float_in_mul[22:0];
36
37  assign M_sq_trunc = M_sq[22:0];
38  assign M_mul_trunc = M_mul[22:0];
39
40  always @(posedge clk)
41      if(rst == 1'b1) begin
42          float_out <= 32'b0;
43          E <= 8'b0;
44          M_sq_done <= 23'b0;
45          M <= 23'b0;
46          M_mul <= 48'b0;
47          M_sq <= 48'b0;
48          ready <= 1'b0;
49          state <= IDLE_SQ;
50      end else begin
51          case(state)
52              IDLE_SQ: begin
53                  ready <= 1'b0;
54                  if (start == 1'b1) begin
55                      M_sq <= ({1'b1, M_in_sq} * {1'b1, M_in_sq});
56                      state <= OVERLOWSQ;
57                  end else
58                      state <= IDLE_SQ;
59              end
60              OVERLOWSQ: begin
61                  if (M_sq[47] == 1'b1) begin
62                      M_sq <= (M_sq >> 1);
63                      E <= E_sq2 + E_mul + 1;
64                  end else begin
65                      M_sq <= M_sq;
66                      E <= E_sq2 + E_mul;
67                  end
68                  state <= ROUNDSQ;
69              end
70              ROUNDSQ: begin
71                  if(M_sq_trunc[22] == 1'b1)
72                      M_sq_done <= M_sq[45:23] + 1;
73                  else

```

```

74         M_sq_done <= M_sq[45:23];
75         state <= MUL;
76     end
77     MUL: begin
78         M_mul <= ({1'b1, M_sq_done} * {1'b1, M_in_mul});
79         state <= OVERFLOWMUL;
80     end
81     OVERFLOWMUL: begin
82         if (M_mul[47] == 1'b1) begin
83             M_mul <= (M_mul >> 1);
84             E <= E + 1;
85         end else begin
86             M_mul <= M_mul;
87             E <= E;
88         end
89         state <= ROUNDMUL;
90     end
91     ROUNDMUL: begin
92         if(M_mul_trunc[22] == 1'b1)
93             M <= M_mul[45:23] + 1;
94         else
95             M <= M_mul[45:23];
96
97         state <= FINISH;
98     end
99     FINISH: begin
100         float_out[30:0] <= {(E + 127), M};
101         float_out[31] <= 0;
102         ready <= 1'b1;
103         state <= IDLE_SQ;
104     end
105 endcase
106 end
107 endmodule

```

2.3 Opis oraz listing algorytmu odejmującego

Matematycznie działanie modułu można opisać jako:

$$\begin{aligned} & \left(1 + \frac{M_1}{2^{23}}\right) 2^{E_1-127} - \left(1 + \frac{M_2}{2^{23}}\right) 2^{E_2-127} \\ & \left(1 + \frac{M_1}{2^{23}}\right) 2^{E_1-127} - \left(1 + \frac{M_2}{2^{23}}\right) 2^{-a} \cdot 2^{E_1-127} \\ & E_2 = E_1 - a \\ & 2^{E_1-127} \left[\left(1 + \frac{M_1}{2^{23}}\right) - \left(1 + \frac{M_2}{2^{23}}\right) 2^{-a} \right] \end{aligned}$$

Układ odejmujący tak jak układy mnożące został zrealizowany w postaci maszyny stanów, ma on takie same wejścia i wyjścia jak moduł `invsqrt`, ze względu na fakt, iż przyjmuje tylko jedną liczbę na wejście: wejście zegara `clk`, reset `rst`, start `start`, wejściową liczbę typu floating point `float_in`, dwa wyjścia - wyjście `ready` oraz wyjściową liczbę zmiennoprzecinkową `float_out`, będąca wynikiem odjęcia `float_in` od wartości 1.5.

Operacja odejmowania została zrealizowana w oparciu o algorytm odejmowania liczb zmiennoprzecinkowych, w którym odejmowana liczba jest sprowadzana do takiej samej wartości eksponenty, z odpowiednim przesunięciem mantysy. W pierwszym kroku sprawdzane jest czy i o ile trzeba przesunąć wejściową liczbę. Sprawdzane jest tylko przesunięcie w zakresie od 0 do 3 bitów, ponieważ liczby otrzymywane w wyniku użycia algorytmu zamykają się w zakresie od 0.4 do 0.6 (w zakresie od 1 do 0.5 eksponenta jest równa -1, w zakresie 0.5 do 0.25 eksponenta jest równa -2. Sprawdzanie 3 bitu jest dodane w celu zabezpieczenia się przed skrajnymi przypadkami.

W stanie odejmowania na końcu dodane są trzy bity, których dodanie ma na celu przeciwdziałać utracie precyzji w trakcie wykonywania operacji odejmowania.

W stanie `OVERFLOW` sprawdzane jest, czy w wyniku odejmowania najstarszy bit mantysy ma wartość 1. Jeżeli nie, to wartość mantysy jest przesuwana o jeden w lewo, a wartość eksponenty zostaje zmniejszona o jeden. W następnym kroku wykonywana jest operacja zaokrąglania.

W układzie tym na wejście podawana jest tylko jedna odejmowana liczba, Kod modułu został zamieszczony poniżej.

Listing 5: moduł `float_mul.v`

```
1 `timescale 1ns / 1ps
2 `define EXP_SHIFT 23
3 `define ROUND_SHIFT 3
4
```

```

5  module float_sub_1d5 (clk, rst, start, float_in, float_out, ready);
6
7  input wire start;
8  input wire clk;
9  input wire rst;
10 input wire [31:0] float_in;
11
12 output reg [31:0] float_out;
13 output reg ready;
14
15 wire [1:0] E_in;
16 wire [7:0] E;
17 reg [`EXP_SHIFT + `ROUND_SHIFT:0] M_in, M_sub, M_ov;
18 reg [`EXP_SHIFT:0] M_rd;
19 wire E_ov;
20 reg [2:0] state;
21 wire [22:0] M;
22
23 localparam IDLE = 3'b000,
24             SUBTRACTION = 3'b001,
25             OVERFLOW = 3'b010,
26             ROUNDING = 3'b011,
27             FINISH = 3'b100;
28
29 assign E_in = float_in[24:23];
30 assign M = float_in[22:0];
31 assign E_ov = (M_sub[`EXP_SHIFT+`ROUND_SHIFT] == 1'b0) ? 1'b0 : 1'b1;
32 assign E = {7'b011_1111, E_ov};
33
34 always @(posedge clk)
35     if(rst == 1'b1) begin
36         float_out <= 32'b0;
37         M_in <= 27'b0;
38         M_sub <= 27'b0;
39         M_ov <= 27'b0;
40         M_rd <= 24'b0;
41         ready <= 1'b0;
42         state <= IDLE;
43     end else begin
44         case(state)
45             IDLE: begin
46                 ready <= 0;

```

```

47         if (start == 1'b1) begin
48             if (E_in == 2'b10)
49                 M_in <= (({1'b1, M, 3'b000}) >> 1);
50             else if (E_in == 2'b01)
51                 M_in <= (({1'b1, M, 3'b000}) >> 2);
52             else
53                 M_in <= M_in;
54             state <= SUBTRACTION;
55         end else begin
56             state <= IDLE;
57         end
58     end
59     SUBTRACTION: begin
60         M_sub <= ({1'b1, 23'h40_0000, 3'b000}) - M_in;
61         state <= OVERFLOW;
62     end
63     OVERFLOW: begin
64         if (M_sub[`EXP_SHIFT+`ROUND_SHIFT] == 1'b0)
65             M_ov <= M_sub << 1;
66         else
67             M_ov <= M_sub;
68         state <= ROUNDING;
69     end
70     ROUNDING: begin
71         if (M_ov[`ROUND_SHIFT-1] == 1'b1)
72             M_rd <= M_ov[`EXP_SHIFT+`ROUND_SHIFT:`ROUND_SHIFT] + 1;
73         else
74             M_rd <= M_ov[`EXP_SHIFT+`ROUND_SHIFT:`ROUND_SHIFT];
75         state <= FINISH;
76     end
77     FINISH: begin
78         float_out[30:0] <= {E, M_rd[22:0]};
79         float_out[31] <= 0;
80         ready <= 1'b1;
81         state <= IDLE;
82     end
83 endcase
84 end
85 endmodule

```

3 Stworzenie modułu, określenie wejść/wyjść

Układ posiada następujące wejścia: `clk`, `rst`, `start`, `float_in` oraz wyjścia `float_out` i `ready`. Sygnał `clk` to wejście zegara, `rst` to wejście sygnału reset, a sygnał `start` rozpoczyna pracę układu. Wejściowymi danymi w układzie jest 32 bitowa wartość `float_in`, będąca wejściową liczbą zmiennoprzecinkową.

```
1 input wire start;
2 input wire clk;
3 input wire rst;
4 input wire [31:0] float_in;
```

Obliczona wartość odwrotności pierwiastka pojawia się na 32 bitowym wyjściu `float_out`. Oprócz wyjściowej wartości liczbowej sygnał podaje sygnał `ready` informujący o tym, że układ zakończył pracę.

```
1 output wire [31:0] float_out;
2 output reg ready;
```

4 Testbench modelu behawioralnego

4.1 Wstęp

Wszystkie testbench'e zostały napisane w taki sposób, aby odczytywać dane z wektorów testowych oraz zapisywały wyniki wyjściowe do pliku tekstowego, przykładowa zawartość pliku sq_mul.in.tv:

```
.InSq   _InMul   _OutXpctd
3ea3d70a_3fc51eb8_3e217b0f
3e99999a_3fd851ec_3e1bc01a
3dfbe76d_404d70a4_3d46eb24
3f174bc7_3fddb22d_3f1ade50
3ea3d70a_3e99999a_3cfba882
3fc51eb8_3fd851ec_40804192
3dfbe76d_3dfbe76d_3af3e856
3dc49ba6_3fddb22d_3c835d98
3fd851ec_3fd851ec_409a7538
3fc51eb8_3fddb22d_4083718d
```

W każdym z wektorów testowych umieszczone zostały wejściowe wartości liczb zmiennoprzecinkowych oraz przewidywany wynik na wyjściu. Przewidywane rezultaty zostały zestawione w tabeli wraz z rzeczywistymi otrzymanymi wartościami, przedstawiona została również różnica jako wartość bezwzględna oraz moduł procentowej różnicy. Ze względu na znikomą czytelność wartości zapisanych jako liczby zmiennoprzecinkowe (np. $0.5 = 0.492799967527389$) nie zostały one zamieszczone w tabelach z wynikami. W celu jak najdokładniejszego odczytu/porównania wyników na wyjściach układów wartości wyjściowe rzeczywiste i przewidywane zostały przekonwertowane z wartości bitowych na dziesiętne z wykorzystaniem `$bitstoshortreal`, co pozwala na łatwe śledzenie i porównanie wyników na poziomie waveform'ów. Testbenche zostały napisane w języku SystemVerilog, dzięki czemu można było wykorzystać ww. funkcję.

4.2 Testbench oraz wyniki dla układu obliczającego odwrotność pierwiastka

Testując nadrzędny moduł, oprócz wyników obliczanych jako odwrotność pierwiastka przetestowane zostały również wszystkie kombinacje, które powinny striggerować błąd, w tabeli 1 kolejne wartości 7FFFFFFF oznaczające NaN (Not a Number) to odpowiedź na wartości (hex): (00000000 - zero), (80000000 - reakcja na bit znaku), (7fffffff - NaN), (7f800000 - +inf), (ff800000 - -inf), (c0a00000 - -5 - reakcja na liczbę ujemną). Testbench modułu został przedstawiony na listingu 6.

Listing 6: testbench invsqrt_tb.sv

```
1  `timescale 1ns / 1ps
2
3  module invsqrt_tb(
4
5  );
6
7  logic clk, rst, start;
8  logic [31:0] float_in;
9  logic [31:0] float_out;
10 logic ready;
11 real out, out_expected;
12
13 logic [31:0] float_out_expected;
14 logic [65:0] testvectors [12:0];
15 logic [31:0] vecnum;
16 integer f;
17
18 invsqrt invsqrt_TB(clk, rst, start, float_in, float_out, ready);
19
20 initial
21 begin
22     $readmemh("path/invsg_in.tv", testvectors);
23     f = $fopen("path/invsg_out.txt", "w");
24     clk <= 1'b0;
25     vecnum <= 0;
26     float_in = 32'b0;
27     start = 1'b0;
28     rst = 1'b1;
29     #100
```

```

30     rst = 1'b0;
31     #10
32     start = 1'b1;
33     #10;
34     start = 1'b0;
35 end
36 always begin
37     #5 clk <= ~clk;
38     out = $bitstoshortreal(float_out);
39     out_expected = $bitstoshortreal(float_out_expected);
40 end
41
42 always@(posedge ready)
43 begin
44     start = 1'b1;
45     @(negedge ready);
46     start = 1'b0;
47 end
48
49 always@(posedge start)
50 begin
51     vecnum = vecnum + 1;
52     {float_in, float_out_expected} = testvectors[vecnum];
53     $fwrite(f, "%h\n", float_out);
54
55     if (vecnum == 14) begin
56         $fclose(f);
57         $stop;
58     end
59
60 end
61
62 endmodule

```

Otrzymane oraz przewidywane wyniki przedstawiono w tabeli poniżej, zgodnie z opisem znajdującym się we wstępie do rozdziału.

out val	should be	abs diff	% abs diff
7ffffff	7ffffff	0,00E+00	0,00E+00
3eff910f	3F000000	8,46E-04	1,69E-01
7ffffff	7ffffff	0,00E+00	0,00E+00
3f13ac3c	3f13cd3a	5,03E-04	8,72E-02
7ffffff	7ffffff	0,00E+00	0,00E+00
3ea1a191	3ea1e89b	5,42E-04	1,71E-01
7ffffff	7ffffff	0,00E+00	0,00E+00
3d9d4447	3d9d89d9	1,33E-04	1,73E-01
7ffffff	7ffffff	0,00E+00	0,00E+00
3fff910f	40000000	3,39E-03	1,69E-01
7ffffff	7ffffff	0,00E+00	0,00E+00
402eb3e1	402ed5ad	2,06E-03	7,55E-02
3cdc8a33	3cdcae63	1,73E-05	6,41E-02

Tabela 1: Porównanie wyników na wyjściu z przewidywanymi - moduł invsqrt.v

4.3 Testbench oraz wyniki dla układów mnożących

Testbenche wszystkich układów są napisane w analogiczny sposób jak testbench modułu `invsqrt_tb.sv`. Dane wejściowe pobierane z odpowiednich plików z wektorami testowymi, zapis do pliku oraz wykorzystanie funkcji `$bitstoshortreal` do łatwiejszego śledzenia wyników. Pod kodem każdego z testbenchów, w postaci tabeli zostały przedstawione otrzymane dla danego modułu wyniki.

Listing 7: testbench `float_mul_tb.sv`

```
1  `timescale 1ns / 1ps
2
3  module float_mul_tb(
4
5  );
6
7  logic clk, rst, start;
8  logic [31:0] float_in_1;
9  logic [31:0] float_in_2;
10 logic [31:0] float_out;
11 logic ready;
12 real mul_module, mul_ideal;
13
14 logic [31:0] float_out_exp;
15 logic [95:0] testvectors [11:0];
16 logic [31:0] vecnum;
17 integer f;
18
19 float_mul float_mul_TB(clk, rst, start, float_in_1, float_in_2, float_out,
    ↪ ready);
20
21 initial
22 begin
23     $readmemh("path/mul_in.tv", testvectors);
24     f = $fopen("path/mul_out.txt", "w");
25     vecnum <= 32'b0;
26     clk <= 1'b1;
27     float_in_1 <= 32'b0;
28     float_in_2 <= 32'b0;
29     start = 1'b0;
30     rst = 1'b1;
31     #5
```

```

32     rst = 1'b0;
33     #5
34     start = 1'b1;
35     #10;
36     start = 1'b0;
37 end
38 always begin
39     #5 clk <= ~clk;
40 end
41
42 always@(posedge ready)
43 begin
44     vecnum = vecnum + 1;
45     mul_module = $bitstoshortreal(float_out);
46     mul_ideal = $bitstoshortreal($bitstoshortreal(float_in_1) *
47     ↪ $bitstoshortreal(float_in_2));
47     start = 1'b1;
48     @(negedge ready);
49     start = 1'b0;
50 end
51
52 always@(posedge start)
53 begin
54     {float_in_1 ,float_in_2, float_out_exp} = testvectors[vecnum];
55     $fwrite(f,"%h\n",float_out);
56
57     if (vecnum == 12) begin
58         $fclose(f);
59         $stop;
60     end
61 end
62 endmodule

```

out val	should be	abs diff	% abs diff
3efc5047	3efc5048	2,98E-08	6,05E-06
4036ca58	4036ca58	0,00E+00	0,00E+00
3f01cac1	3f01cac1	0,00E+00	0,00E+00
3eca2729	3eca2728	2,98E-08	7,55E-06
3f8305b8	3f8305b8	0,00E+00	0,00E+00
3dc49ba6	3dc49ba6	0,00E+00	0,00E+00
3e5a261c	3e5b280f	9,84E-04	4,60E-01
40269100	40269100	0,00E+00	0,00E+00
3c77dfa1	3c77dfa0	9,31E-10	6,16E-06
3e2a4335	3e2a4335	0,00E+00	0,00E+00
4036ca58	4036ca58	0,00E+00	0,00E+00
402ab4b7	402ab4b7	0,00E+00	0,00E+00

Tabela 2: Porównanie wyników na wyjściu z przewidywanymi - moduł float_mul.v

Listing 8: testbench float_sq_mul_tb.sv

```

1  `timescale 1ns / 1ps
2
3  module float_sq_mul_tb();
4
5  logic clk, rst, start;
6  logic [31:0] float_in_sq;
7  logic [31:0] float_in_mul;
8  logic [31:0] float_out;
9  logic ready;
10 real mul_module, mul_ideal;
11
12 logic [31:0] float_out_exp;
13 logic [95:0] testvectors [9:0];
14 logic [31:0] vecnum;
15 integer f;
16
17 float_sq_mul float_sq_mul_TB(clk, rst, start, float_in_sq, float_in_mul,
   ↪ float_out, ready);
18
19 initial
20 begin

```

```

21     $readmemh("path/sq_mul_in.tv", testvectors);
22     f = $fopen("path/sq_mul_out.txt", "w");
23     vecnum <= 32'b0;
24     clk <= 1'b1;
25     float_in_sq <= 32'b0;
26     float_in_mul <= 32'b0;
27     start = 1'b0;
28     rst = 1'b1;
29     #10
30     rst = 1'b0;
31     #5
32     start = 1'b1;
33     #10;
34     start = 1'b0;
35 end
36 always begin
37     #5 clk <= ~clk;
38 end
39
40 always@(posedge ready)
41 begin
42     vecnum = vecnum + 1;
43     mul_module = $bitstoshortreal(float_out);
44     mul_ideal = $bitstoshortreal($bitstoshortreal(float_in_sq) *
45     ↪ $bitstoshortreal(float_in_sq) * $bitstoshortreal(float_in_mul));
46     start = 1'b1;
47     @(negedge ready);
48     start = 1'b0;
49 end
50
51 always@(posedge start)
52 begin
53     {float_in_sq ,float_in_mul, float_out_exp} = testvectors[vecnum];
54     $fwrite(f, "%h\n", float_out);
55
56     if (vecnum == 10) begin
57         $fclose(f);
58         $stop;
59     end
60 end
61 endmodule

```

out val	should be	abs diff	% abs diff
3e217b0f	3e217b0f	0,00E+00	0,00E+00
3e1bc01b	3e1bc01a	1,49E-08	9,80E-06
3d46eb25	3d46eb24	3,73E-09	7,67E-06
3f1ade50	3f1ade50	0,00E+00	0,00E+00
3cfba883	3cfba882	1,86E-09	6,06E-06
40804192	40804192	0,00E+00	0,00E+00
3af3e857	3af3e856	1,16E-10	6,26E-06
3c82c2fa	3c835d98	7,37E-05	4,60E-01
409a7539	409a7538	4,77E-07	9,88E-06
4083718d	4083718d	0,00E+00	0,00E+00

Tabela 3: Porównanie wyników na wyjściu z przewidywanymi - moduł float_sq_mul.v

4.4 Testbench oraz wyniki dla układu odejmującego

Listing 9: testbench float_sub_1d5_tb.sv

```
1  `timescale 1ns / 1ps
2
3  module invsqr_ttb(
4
5  );
6
7  logic clk, rst, start;
8  logic [31:0] float_in;
9  logic [31:0] float_out;
10 logic ready;
11 real out, out_expected;
12
13 logic [31:0] float_out_expected;
14 logic [65:0] testvectors [12:0];
15 logic [31:0] vecnum;
16 integer f;
17
18 invsqr invsqr_TB(clk, rst, start, float_in, float_out, ready);
19
20 initial
21 begin
22     $readmemh("path/invsq_in.tv", testvectors);
23     f = $fopen("path/invsq_out.txt", "w");
24     clk <= 1'b0;
25     vecnum <= 0;
26     float_in = 32'b0;
27     start = 1'b0;
28     rst = 1'b1;
29     #100
30     rst = 1'b0;
31     #10
32     start = 1'b1;
33     #10;
34     start = 1'b0;
35 end
36 always begin
37     #5 clk <= ~clk;
38     out = $bitstoshortreal(float_out);
```

```

39     out_expected = $bitstoshortreal(float_out_expected);
40 end
41
42 always@(posedge ready)
43 begin
44     start = 1'b1;
45     @(negedge ready);
46     start = 1'b0;
47 end
48
49 always@(posedge start)
50 begin
51     vecnum = vecnum + 1;
52     {float_in, float_out_expected} = testvectors[vecnum];
53     $fwrite(f, "%h\n", float_out);
54
55     if (vecnum == 14) begin
56         $fclose(f);
57         $stop;
58     end
59
60 end
61
62 endmodule

```

out val	should be	abs diff	% abs diff
3f8ccccd	3f8ccccd	0,00E+00	0,00E+00
3f8cbc6b	3f8cbc6a	1,19E-07	1,08E-05
3f8cac08	3f8cac08	0,00E+00	0,00E+00
3f8c9ba6	3f8c9ba6	0,00E+00	0,00E+00
3f8c8b44	3f8c8b44	0,00E+00	0,00E+00
3f8c7ae1	3f8c7ae1	0,00E+00	0,00E+00
3f8c6a7f	3f8c6a7f	0,00E+00	0,00E+00
3f8c5a1d	3f8c5a1d	0,00E+00	0,00E+00
3f8c49ba	3f8c49ba	0,00E+00	0,00E+00
3f8c3958	3f8c3958	0,00E+00	0,00E+00

Tabela 4: Porównanie wyników na wyjściu z przewidywanymi - moduł fp_sub_1d5.v

4.5 Podsumowanie otrzymanych wyników

Wśród wyników wszystkich pod modułów można zauważyć, że ich pokrycie w przewidywanymi wynikami jest bardzo duże - największy procentowy błąd spośród otrzymanych wyników to 0,46% w przypadku mnożenia bardzo małych liczb - otrzymane 0.213036000728607 przy przewidywanym 0.214019998908042 w przypadku układu float_mul, skutkujące błędem na poziomie 0.46%, oraz skutkujące takim samym błędem wynik mnożenia z podnoszeniem do kwadratu - otrzymana wartość: 0,0159621126949787, przewidywana wartość: 0,0160358399152755. W przypadku nadrzędnego modułu obliczającego odwrotność pierwiastka wszystkie wyniki obarczone są błędem w zakresie od 0.064% do 0.1725%. Układ reaguje poprawnie na wszystkie kombinacje, które powinny zakończyć się errorem i podaniem na wyjście wartości NaN.

5 Model syntezy algorytmu oraz jego testbench

Moduły napisane jako moduły behawioralne są również modelami synteżowalnymi, nie ma więc potrzeby przedstawiania wyników dla nich jeszcze raz.

6 AXI, Zynq

Tworząc układ obliczający odwrotność pierwiastka jako moduł wykorzystujący AXI Lite. Schemat blokowy składa się z bloku procesora Zynq, układu obsługującego reset procesora, modułu AXI Interconnect oraz modułu `invsqrt`.

Podczas syntezy oraz implementacji układów wykorzystano następującą ilość zasobów układu:

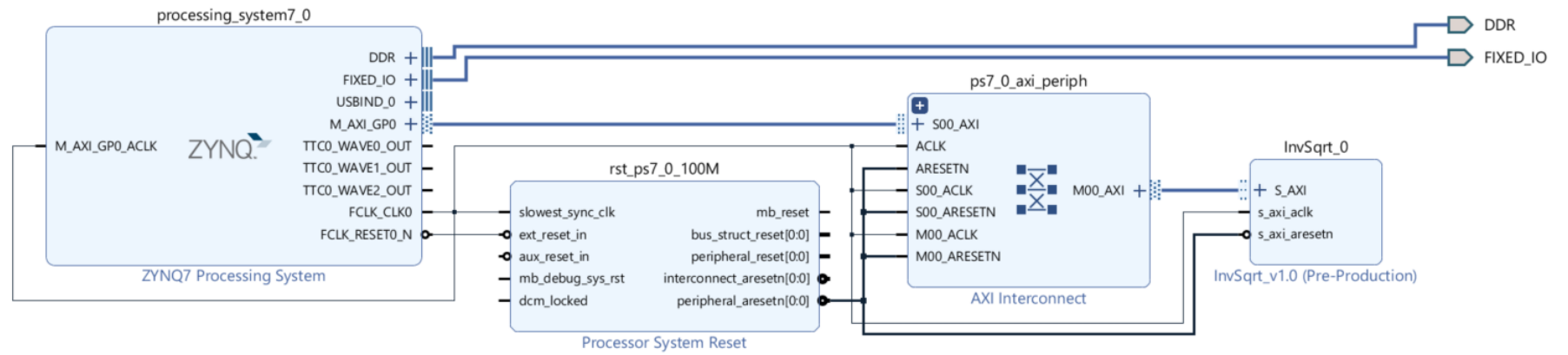
Resource	Utilization	Available	Utilization %
LUT	207	53200	0.39
FF	332	106400	0.31
DSP	6	220	2.73

Tabela 5: Zużycie zasobów - synteza

Resource	Utilization	Available	Utilization %
LUT	208	53200	0.39
FF	332	106400	0.31
DSP	6	220	2.73

Tabela 6: Zużycie zasobów - implementacja

Podczas implementacji udało się zmniejszyć zużycie o jedną tablicę LUT. Procentowo najwięcej zajmowanych jest układów mnożących (2.73%)



Rysunek 2: Block diagram układu invsqr

7 Uruchomienie na sprzęcie

W celu uruchomienia układu na sprzęcie napisany został kod, w którym została zaimplementowana pierwotna funkcja obliczająca odwrotność pierwiastka wykorzystana do porównania wyników oraz funkcja mająca za zadanie obliczyć ten pierwiastek z wykorzystaniem funkcji sprzętowej oraz pierwowzorem, wraz z ustawieniem odpowiednich bitów w wewnętrznych rejestrach modułu. Poniżej przedstawiono kod funkcji main oraz kod funkcji InvSqrt_Calculate.

Listing 10: funkcja InvSqrt_calculate oraz funkcja main dla invsqrt

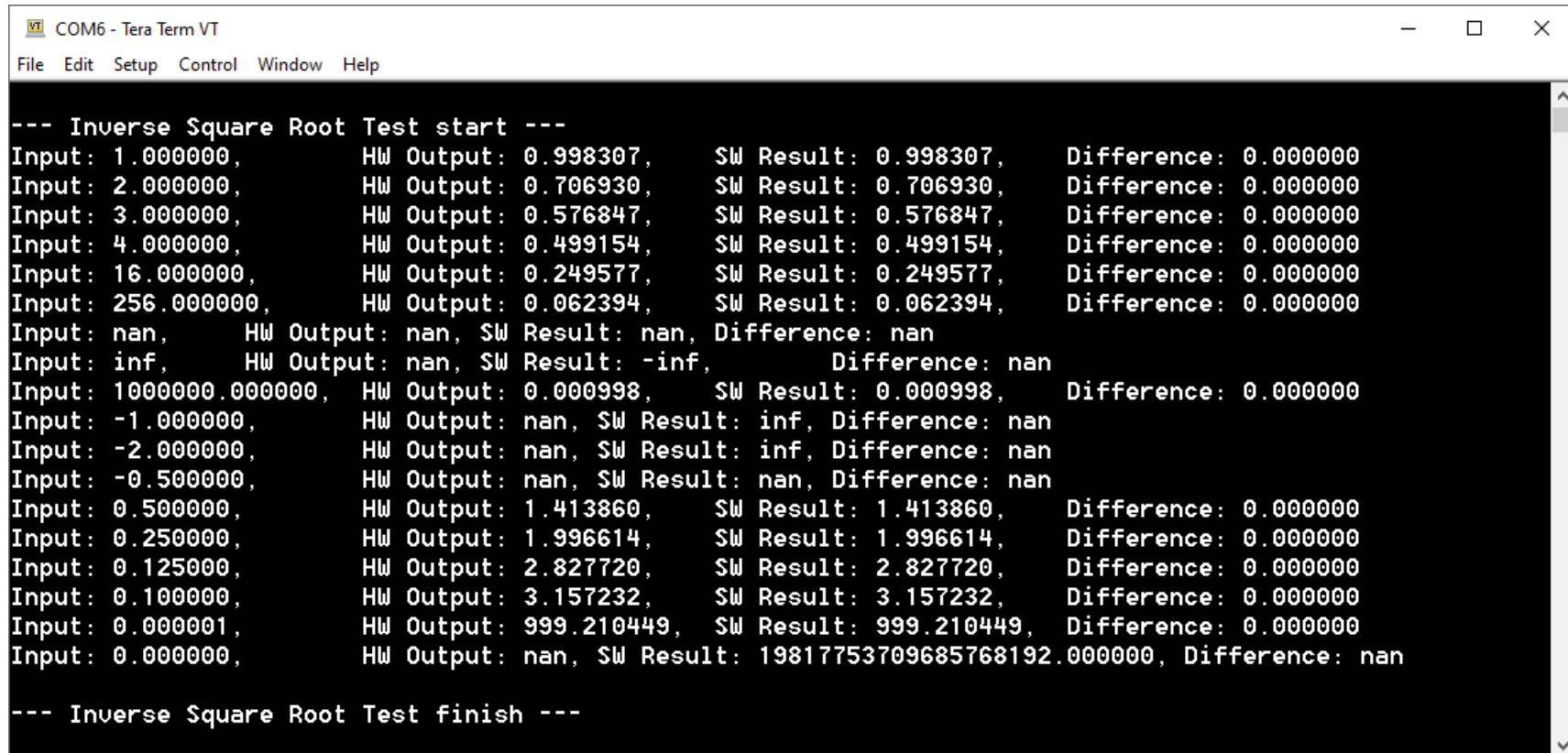
```
1 static float InvSqrt_Calculate(float number)
2 {
3     u32 result_u32;
4     float result_f;
5     INVSQRT_mWriteReg(INVSQRT_BASEADDR, INVSQRT_INPUT_REG, *(u32*)&number);
6     INVSQRT_mWriteReg(INVSQRT_BASEADDR, INVSQRT_START_REG, 1);
7     while(INVSQRT_mReadReg(INVSQRT_BASEADDR, INVSQRT_READY_REG) == 0);
8     result_u32 = INVSQRT_mReadReg(INVSQRT_BASEADDR, INVSQRT_OUTPUT_REG);
9     result_f = *(float*)&result_u32;
10    return result_f;
11 }
12
13 int main()
14 {
15     init_platform();
16     const float test_vector[ARRAY_LENGTH] = {
17         1.0, 2.0, 3.0, 4.0, 16.0, 256.0, NAN, INFINITY,
18         ↪ 1000000.0, -1.0, -2.0, -0.5, 0.5, 0.25, 0.125, 0.1,
19         ↪ 0.000001, 0.0
20     };
21
22     float hw_result[ARRAY_LENGTH];
23     float sw_result[ARRAY_LENGTH];
24
25     xil_printf("\r\n--- Inverse Square Root Test start ---\r\n");
26
27     for(u8 i = 0; i < ARRAY_LENGTH; i++) {
28         sw_result[i] = Q_rsqrt(test_vector[i]);
29         hw_result[i] = InvSqrt_Calculate(test_vector[i]);
30     }
```

```

30 //      Xil_DCacheDisable();
31
32 /*
33  * Test finished, check data
34  */
35 float in, hw, sw;
36 for (u8 i = 0; i < ARRAY_LENGTH; i++) {
37     in = test_vector[i];
38     hw = hw_result[i];
39     sw = sw_result[i];
40     printf("Input: %f,\tHW Output: %f,\tSW Result: %f,\tDifference:
    ↪ %f \n", in, hw, sw, hw - sw);
41 }
42
43 xil_printf("\r\n--- Inverse Square Root Test finish ---\r\n");
44
45 cleanup_platform();
46 return 0;
47 }

```

Na załączonym poniżej zrzucie ekranu przedstawione zostały wyniki użycia stworzonej funkcji. HW Output to wynik otrzymany na wyjściu układu, a SW Result to wynik otrzymany z wykorzystaniem oryginalnej funkcji.



```
COM6 - Tera Term VT
File Edit Setup Control Window Help

--- Inverse Square Root Test start ---
Input: 1.000000,      HW Output: 0.998307,      SW Result: 0.998307,      Difference: 0.000000
Input: 2.000000,      HW Output: 0.706930,      SW Result: 0.706930,      Difference: 0.000000
Input: 3.000000,      HW Output: 0.576847,      SW Result: 0.576847,      Difference: 0.000000
Input: 4.000000,      HW Output: 0.499154,      SW Result: 0.499154,      Difference: 0.000000
Input: 16.000000,     HW Output: 0.249577,      SW Result: 0.249577,      Difference: 0.000000
Input: 256.000000,    HW Output: 0.062394,      SW Result: 0.062394,      Difference: 0.000000
Input: nan,          HW Output: nan, SW Result: nan, Difference: nan
Input: inf,          HW Output: nan, SW Result: -inf,      Difference: nan
Input: 1000000.000000, HW Output: 0.000998,      SW Result: 0.000998,      Difference: 0.000000
Input: -1.000000,     HW Output: nan, SW Result: inf, Difference: nan
Input: -2.000000,     HW Output: nan, SW Result: inf, Difference: nan
Input: -0.500000,     HW Output: nan, SW Result: nan, Difference: nan
Input: 0.500000,      HW Output: 1.413860,      SW Result: 1.413860,      Difference: 0.000000
Input: 0.250000,      HW Output: 1.996614,      SW Result: 1.996614,      Difference: 0.000000
Input: 0.125000,      HW Output: 2.827720,      SW Result: 2.827720,      Difference: 0.000000
Input: 0.100000,      HW Output: 3.157232,      SW Result: 3.157232,      Difference: 0.000000
Input: 0.000001,      HW Output: 999.210449,    SW Result: 999.210449,    Difference: 0.000000
Input: 0.000000,      HW Output: nan, SW Result: 19817753709685768192.000000, Difference: nan

--- Inverse Square Root Test finish ---
```

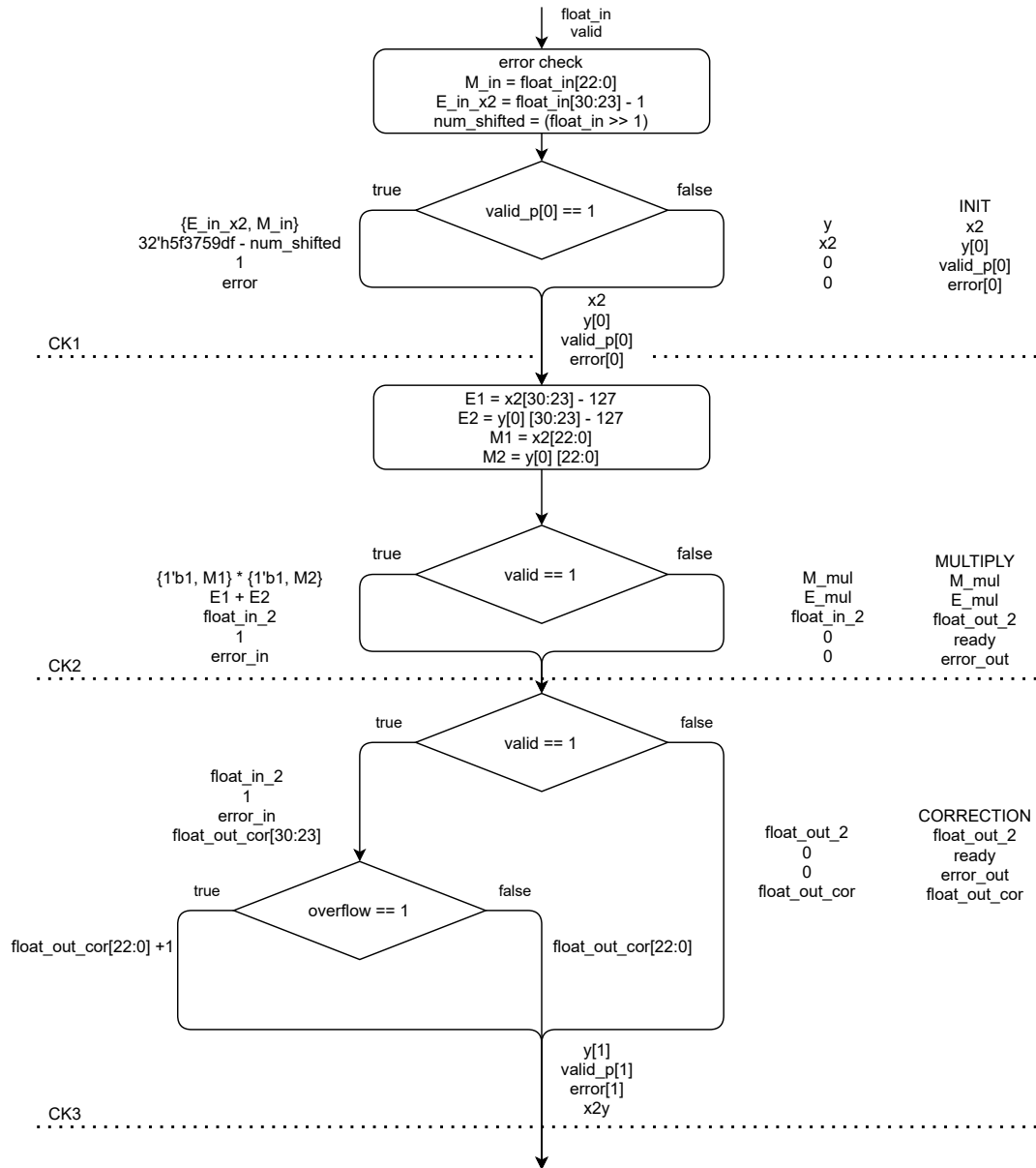
Rysunek 3: Wyniki otrzymane na sprzęcie - screenshot z konsoli

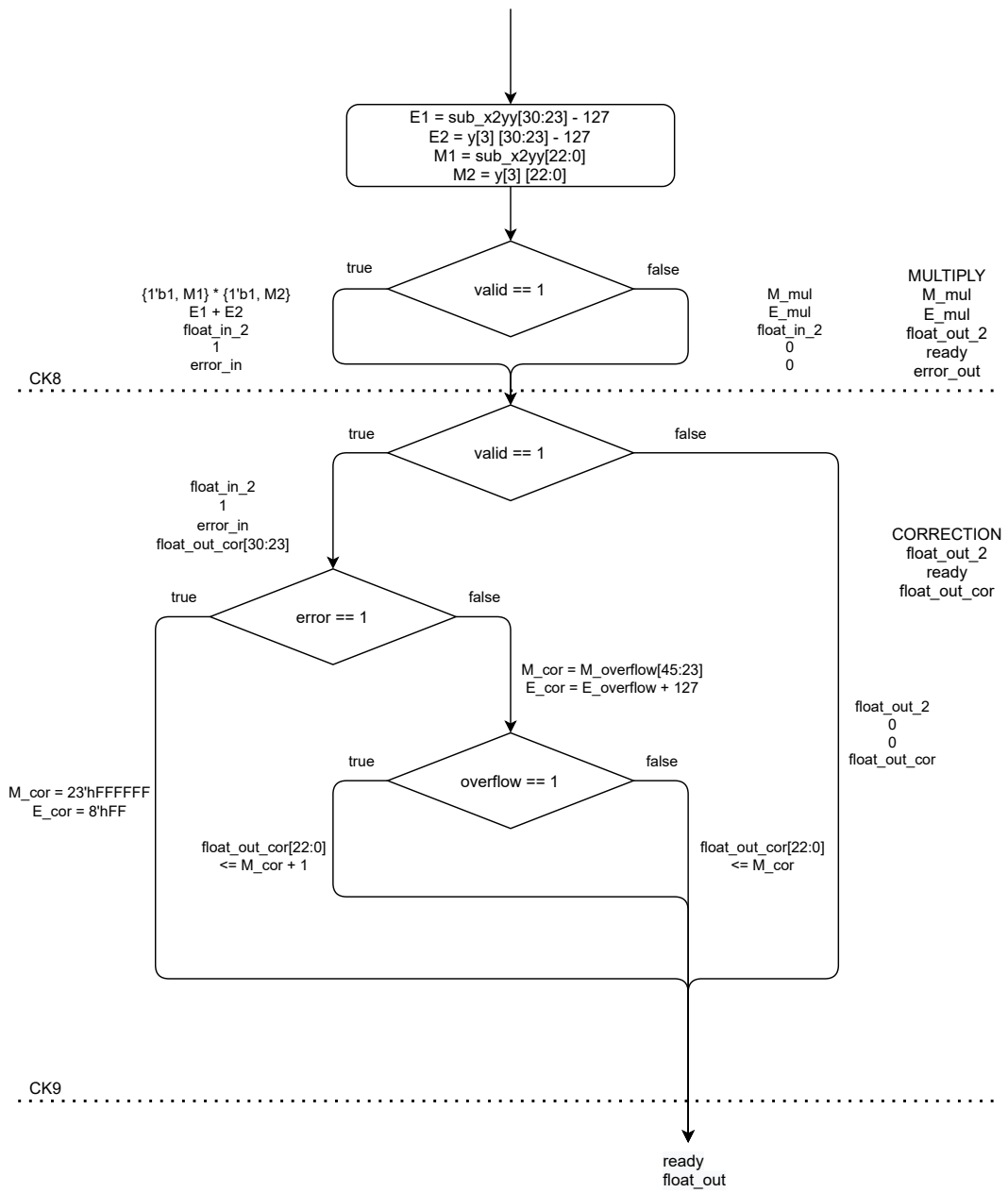
8 Wersja potokowa układu

8.1 Wstęp

W tym rozdziale opisany zostanie układ w wersji pipeline. W poszczególnych podrozdziałach zostaną poruszone poszczególne tematy poruszane w poprzednich rozdziałach, każdy z bloków funkcjonalnych wchodzących w skład układu z przetwarzaniem potokowym został opisany w osobnym podrozdziale. Podrozdziały te zawierają kod opisujący moduł, oraz testbench i wyniki jego użycia jeżeli takowe istnieją.

8.2 Algorytm z przetwarzaniem potokowym w postaci diagramu





8.3 Odwrotność pierwiastka

W układzie obliczającym odwrotność pierwiastka w wersji potokowej główny moduł składa jedyne wszystkie moduły w całość, bezpośrednio w nim nie dzieje się nic. W układzie z przetwarzaniem potokowym w porównaniu do poprzedniej wersji error handling, ze względu na sposób działania układu został zrealizowany w inny sposób - flaga błędu jest przekazywana pomiędzy poszczególnymi modułami do samego końca, gdzie układ reaguje w odpowiedni sposób na jej stan.

8.3.1 Układ

Listing 11: moduł float_mul.v

```
1  `timescale 1ns / 1ps
2
3  module invsqrt_pipeline (clk, rstn, backprn, valid, float_in, float_out, ready);
4
5  input wire clk;
6  input wire rstn;
7  input wire valid;
8  input wire [31:0] float_in;
9
10 input wire backprn;
11 output wire [31:0] float_out;
12 output wire ready;
13
14 wire [30:0] y [0:3];
15 wire [30:0] x2, x2y, x2yy, sub_x2yy;
16 wire valid_p [0:3];
17 wire error[3:0];
18
19 assign float_out[31] = 1'b0;
20
21 invsqrt_pipe_init invsqrt_pipe_init (clk, rstn, backprn, valid,      float_in,
   ↪  x2, y[0], valid_p[0], error[0]);
22 fp_mul_pipe      mul_pipe_x2y      (clk, rstn, backprn, valid_p[0], x2, y[0],
   ↪  x2y, y[1], valid_p[1], error[0], error[1]);
23 fp_mul_pipe      mul_pipe_x2yy     (clk, rstn, backprn, valid_p[1], x2y, y[1],
   ↪  x2yy, y[2], valid_p[2], error[1], error[2]);
24 fp_sub_1d5_pipe  sub_1d5_pipe      (clk, rstn, backprn, valid_p[2], x2yy, y[2],
   ↪  sub_x2yy, y[3], valid_p[3], error[2], error[3]);
```

```

25 fp_mul_pipe #(1) mul_pipe_sub_x2yy (clk, rstn, backprn, valid_p[3], sub_x2yy,
    ↪ y[3], float_out, , ready, error[3]);
26
27 endmodule

```

8.3.2 Testbench oraz wyniki

Testbenche układów z przetwarzaniem potokowym wyglądają podobnie do tych dla układów nie mających pipeline'ingu. Ze względu na potokowe działanie wszystkie wejścia i wyjścia zmieniają stan co cykl zegara a nie w momencie pojawienia się sygnału start/ stop.

Listing 12: moduł invsqrt_pipeline_tb.sv

```

1  `timescale 1ns / 1ps
2
3  module invsqrt_pipeline_tb(
4
5      );
6
7  real out, out_expected;
8  logic clk, ce;
9  logic [31:0] fp_in;
10 logic [30:0] float_out;
11 logic ready;
12 logic rstn, backprn;
13
14 logic [31:0] float_out_expected;
15 logic [65:0] testvectors [12:0];
16 logic [31:0] vecnum;
17 integer f;
18
19 invsqrt_pipeline invsqrt_pipelineTB(clk, rstn, backprn, ce, fp_in[31:0],
    ↪ float_out, ready);
20
21 initial
22 begin
23     $readmemh("invsq_pipe_in.tv", testvectors);
24     f = $fopen("invsq_pipe_out.txt","w");
25     rstn <= 1;
26     backprn <= 1;

```

```

27     vecnum <= 0;
28     ce = 0;
29     clk <= 1'b1;
30     #10
31     rstn <= 0;
32     #130
33     rstn <= 1;
34     fp_in <= 0;
35     @(negedge clk);
36     {fp_in, float_out_expected} = testvectors[vecnum];
37     vecnum = vecnum + 1;
38     ce = 1'b1;
39     repeat(10) @(posedge clk);
40     ce = 1'b0;
41     repeat(5) @(posedge clk);
42     ce = 1'b1;
43 end
44 always begin
45     #5 clk <= ~clk;
46 end
47
48 always@(posedge clk)
49 begin
50
51     out <= $bitstoshortreal({1'b0, float_out});
52     out_expected = $bitstoshortreal(float_out_expected);
53
54     $fwrite(f, "%h\n", float_out);
55     if (ce == 1'b1) begin
56         {fp_in, float_out_expected} = testvectors[vecnum];
57         vecnum = vecnum + 1;
58     end
59
60     if (vecnum == 35) begin
61         $fclose(f);
62         $stop;
63     end
64 end
65
66 endmodule

```

out val	should be	abs diff	% abs diff
7fffffff	7fffffff	0,00E+00	0,00E+00
3eff910f	3F000000	8,46E-04	1,69E-01
7fffffff	7fffffff	0,00E+00	0,00E+00
3f13ac3c	3f13cd3a	5,03E-04	8,72E-02
7fffffff	7fffffff	0,00E+00	0,00E+00
3ea1a191	3ea1e89b	5,42E-04	1,71E-01
7fffffff	7fffffff	0,00E+00	0,00E+00
3d9d4447	3d9d89d9	1,33E-04	1,73E-01
7fffffff	7fffffff	0,00E+00	0,00E+00
3ff910f	40000000	3,39E-03	1,69E-01
7fffffff	7fffffff	0,00E+00	0,00E+00
402eb3e1	402ed5ad	2,06E-03	7,55E-02
3cdc8a33	3cdcae63	1,73E-05	6,41E-02

Tabela 7: Porównanie wyników na wyjściu z przewidywanymi - moduł `invsqrt_pipeline.v`

8.4 Układ inicjalizujący

Układ ten to prosty układ mający za zadanie przyjąć dane wejściowe i wstępnie je przygotować do dalszych operacji. W układzie tym również jest generowana flaga błędu, przekazywana na resztę układu.

Listing 13: moduł `invsqrt_pipe_init.v`

```

1  `timescale 1ns / 1ps
2
3  module invsqrt_pipe_init (clk, rstn, backprn, valid, number, x2, y, ready,
   ↪  error_out);
4
5  input wire clk;
6  input wire rstn;
7  input wire valid;
8  input wire [31:0] number;
9
10 input wire backprn;
11 output reg [30:0] x2;
12 output reg [30:0] y;
13 output reg ready;
14 output reg error_out;
15

```

```

16  wire [22:0] M_in;
17  wire [7:0] E_in_x2;
18  wire [30:0] num_shifted;
19  wire error;
20
21  assign M_in = number[22:0];
22  assign E_in_x2 = number[30:23] - 1;
23  assign num_shifted = (number >> 1);
24
25  assign error = ((number == 31'h00000000) || (number[30:23] == 8'hFF) ||
    ↪ (number[31] == 1'b1)) ? 1'b1 : 1'b0;
26
27  always @(posedge clk) begin
28      if(rstn == 1'b0) begin
29          ready <= 0;
30          error_out <= 0;
31      end else begin
32          if(backprn == 1'b0) begin
33              y <= y;
34              x2 <= x2;
35              ready <= ready;
36              error_out <= error_out;
37          end else begin
38              if(valid == 1'b1) begin
39                  y <= 32'h5f3759df - num_shifted;
40                  x2 <= {E_in_x2, M_in};
41                  ready <= 1;
42                  error_out <= error;
43              end else begin
44                  y <= y;
45                  x2 <= x2;
46                  ready <= 0;
47                  error_out <= 0;
48              end
49          end
50      end
51  end
52
53  endmodule

```

8.5 Układ mnożący

8.5.1 Układ

W przypadku układu z pipeline'ingiem nie ma podziału na układ mnożący i układ mnożący z podnoszeniem do kwadratu - te trzy operacje mnożenia wcześniej zamknięte w dwóch blokach w obecnej wersji są zamknięte jako trzykrotne użycie układu mnożącego. Układ sam w sobie został podzielony na dwa moduły - moduł obliczający wynik mnożenia oraz moduł korygujący wyjściowy wynik. Kod wszystkich modułów został przedstawiony poniżej.

Listing 14: moduł fp_mul_pipe.v

```
1  `timescale 1ns / 1ps
2  module fp_mul_pipe (clk, rstn, backprn, valid, float_in_1, float_in_2, float_out,
   ↪ float_out_delay, ready, error_in, error_out);
3  parameter END = 0;
4
5  input wire clk;
6  input wire rstn;
7  input wire valid;
8  input wire [30:0] float_in_1;
9  input wire [30:0] float_in_2;
10 input wire error_in;
11
12 input wire backprn;
13 output wire [30:0] float_out;
14 output wire [30:0] float_out_delay;
15 output wire ready;
16 output wire error_out;
17
18 wire [47:0] M_mul;
19 wire signed [7:0] E_mul;
20 wire [30:0] float_delay;
21 wire valid_p;
22 wire error_mid;
23
24 fp_mul_multiply_pipe multiply(clk, rstn, backprn, valid, float_in_1, float_in_2,
   ↪ float_delay, M_mul, E_mul, valid_p, error_in, error_mid);
25 fp_mul_correction_pipe #(END) correction(clk, rstn, backprn, valid_p, M_mul,
   ↪ E_mul, float_delay, float_out, float_out_delay, ready, error_mid, error_out);
26 endmodule
```

Listing 15: modul fp_mul_multiply_pipe.v

```

1  `timescale 1ns / 1ps
2
3  module fp_mul_multiply_pipe (clk, rstn, backprn, valid, float_in_1, float_in_2,
   ↪ float_out_2, M_mul, E_mul, ready, error_in, error_out);
4
5  input wire clk;
6  input wire rstn;
7  input wire valid;
8  input wire [30:0] float_in_1;
9  input wire [30:0] float_in_2;
10 input wire error_in;
11
12 input wire backprn;
13 output reg [30:0] float_out_2;
14 output reg [47:0] M_mul;
15 output reg signed [7:0] E_mul;
16 output reg ready;
17 output reg error_out;
18
19 wire signed [7:0] E1, E2, Etmp1, Etmp2;
20 wire [22:0] M1, M2;
21
22 assign E1 = float_in_1[30:23] - 127;
23 assign E2 = float_in_2[30:23] - 127;
24
25 assign M1 = float_in_1[22:0];
26 assign M2 = float_in_2[22:0];
27
28 always @(posedge clk) begin
29     if(rstn == 1'b0) begin
30         ready <= 1'b0;
31         error_out <= 1'b0;
32     end else begin
33         if(backprn == 1'b0) begin
34             float_out_2 <= float_out_2;
35             M_mul <= M_mul;
36             E_mul <= E_mul;
37             ready <= ready;

```

```

38         error_out <= error_out;
39     end else begin
40         if(valid == 1'b1) begin
41             float_out_2 <= float_in_2;
42             M_mul <= {1'b1, M1} * {1'b1, M2};
43             E_mul <= E1 + E2;
44             ready <= 1'b1;
45             error_out <= error_in;
46         end else begin
47             float_out_2 <= float_out_2;
48             M_mul <= M_mul;
49             E_mul <= E_mul;
50             ready <= 1'b0;
51             error_out <= 1'b0;
52         end
53     end
54 end
55 endmodule
56
57

```

Ze względu na fakt, iż moduł mnożący jest ostatnim modulem w torze obliczania odwrotności pierwiastka, został on sparametryzowany w taki sposób, aby w przypadku podania odpowiedniego parametru układ sprawdzał wartość wejściowego bitu błędu i w zależności od jego wartości podawał na wyjście obliczony wynik, jeżeli `error == 0`, w przeciwnym wypadku wyjściowym wynikiem jest 7FFFFFFF (NaN)

Listing 16: moduł `fp_mul_correction_pipe.v`

```

1  `timescale 1ns / 1ps
2
3  module fp_mul_correction_pipe (clk, rstn, backprn, valid, M_in_mul, E_in_mul,
4      ↪ float_in_2, float_out_cor, float_out_2, ready, error_in, error_out);
5
6      parameter END = 0;
7
8      input wire clk;
9      input wire rstn;
10     input wire valid;
11     input wire [47:0] M_in_mul;

```

```

11  input wire signed [7:0] E_in_mul;
12  input wire [30:0] float_in_2;
13  input wire error_in;
14  output reg error_out;
15
16  input wire backprn;
17  output reg [30:0] float_out_cor;
18  output reg [30:0] float_out_2;
19  output reg ready;
20
21  wire [22:0] M_trunc, M_cor;
22  wire overflow;
23  wire signed [7:0] E_cor;
24  reg [46:0] M_overflow;
25  reg signed [7:0] E_overflow;
26
27
28  generate begin
29      if (END == 0) begin
30          assign overflow = M_trunc[22];
31          assign M_cor = M_overflow[45:23];
32          assign E_cor = E_overflow + 127;
33      end else begin
34          assign overflow = (error_in == 0) ? M_trunc[22] : 1'b0;
35          assign M_cor = (error_in == 0) ? M_overflow[45:23] : 23'hFFFFFF;
36          assign E_cor = (error_in == 0) ? (E_overflow + 127) : 8'hFF;
37      end
38  end
39  endgenerate
40
41  assign M_trunc = M_overflow[22:0];
42
43  always @(posedge clk) begin
44      if(rstn == 1'b0) begin
45          ready <= 1'b0;
46          error_out <= 1'b0;
47      end else begin
48          if(backprn == 1'b0) begin
49              float_out_2 <= float_out_2;
50              ready <= ready;
51              error_out <= error_out;
52              float_out_cor <= float_out_cor;

```

```

53     end else begin
54         if(valid == 1'b1) begin
55             float_out_2 <= float_in_2;
56             ready <= 1'b1;
57             error_out <= error_in;
58             float_out_cor[30:23] <= E_cor;
59             if(overflow == 1'b1)
60                 float_out_cor[22:0] <= M_cor + 1;
61             else
62                 float_out_cor[22:0] <= M_cor;
63         end else begin
64             float_out_cor <= float_out_cor;
65             float_out_2 <= float_out_2;
66             ready <= 1'b0;
67             error_out <= 0;
68         end
69     end
70 end
71 end
72
73 always @* begin
74     if(M_in_mul[47] == 1'b1) begin
75         M_overflow = M_in_mul >> 1;
76         E_overflow = E_in_mul + 1;
77     end else begin
78         M_overflow = M_in_mul;
79         E_overflow = E_in_mul;
80     end
81 end
82
83 endmodule

```

8.5.2 Testbench oraz wyniki

Listing 17: testbench fp_mul_pipe.tb.sv

```
1  `timescale 1ns / 1ps
2
3  module float_mul_pipe_tb(
4
5  );
6
7  real out, out_expected;
8  logic clk, valid, ready, error_in, error_out;
9  logic [31:0] float_in_1, float_in_2;
10 logic [30:0] float_out, float_out_delay;
11 logic rstn, backprn;
12
13 logic [31:0] float_out_exp;
14 logic [99:0] testvectors [9:0];
15 logic [31:0] vecnum;
16 logic [3:0] error_4b;
17 integer f;
18
19 fp_mul_pipe #(1) float_mul_pipe_TB(clk,rstn, backprn, valid, float_in_1,
   ↪ float_in_2, float_out, float_out_delay, ready, error_in, error_out);
20
21 initial
22 begin
23     $readmemh("mul_pipe_in.tv", testvectors);
24     f = $fopen("mul_pipe_out.txt","w");
25     rstn <= 1'b1;
26     backprn <= 1'b1;
27     vecnum <= 32'b0;
28     valid <= 1'b0;
29     error_in <= 1'b0;
30     float_in_1 <= 32'b0;
31     float_in_2 <= 32'b0;
32     clk <= 1'b1;
33     #10;
34     rstn <= 1'b0;
35     #10;
36     rstn <= 1'b1;
37     #10;
```

```

38     {float_in_1 ,float_in_2, float_out_exp, error_4b} = testvectors[vecnum];
39     vecnum <= vecnum + 1;
40     error_in <= error_4b[0];
41     valid = 1'b1;
42 end
43
44 always begin
45     #5 clk <= ~clk;
46 end
47
48 always@(posedge clk)
49 begin
50
51     out = $bitstoshortreal(float_out);
52     out_expected = $bitstoshortreal($bitstoshortreal(float_in_1) *
53     ↪ $bitstoshortreal(float_in_2));
54
55     if (valid == 1) begin
56         {float_in_1 ,float_in_2, float_out_exp, error_4b} = testvectors[vecnum];
57         vecnum <= vecnum + 1;
58         error_in <= error_4b[0];
59     end
60
61     $fwrite(f, "%h\n",float_out);
62
63     if (vecnum == 13) begin
64         $fclose(f);
65         $stop;
66     end
67 end
68 endmodule

```

out val	should be	abs diff	% abs diff
3efc5047	3efc5048	2,98E-08	6,05E-06
3f01cac1	3f01cac1	0,00E+00	0,00E+00
3eca2729	3eca2728	2,98E-08	7,55E-06
3f8305b8	3f8305b8	0,00E+00	0,00E+00
3dc49ba6	3dc49ba6	0,00E+00	0,00E+00
40269100	40269100	0,00E+00	0,00E+00
3c77dfa1	3c77dfa0	9,31E-10	6,16E-06
3e2a4335	3e2a4335	0,00E+00	0,00E+00
4036ca58	4036ca58	0,00E+00	0,00E+00
402ab4b7	402ab4b7	0,00E+00	0,00E+00

Tabela 8: Porównanie wyników na wyjściu z przewidywanymi - moduł `fp_mul_pipe.v`

8.6 Układ odejmujący

8.6.1 Układ

Tak jak w przypadku układu mnożącego, układ odejmujący został podzielony na część obliczającą różnicę oraz układ korygujący wynik.

Listing 18: moduł `float_1d5_sub_pipe.v`

```

1  `timescale 1ns / 1ps
2  `define EXP_SHIFT 23
3  `define ROUND_SHIFT 3
4
5  module fp_sub_1d5_pipe (clk, rstn, backprn, valid, float_in, float_in_delay,
6    ↪ float_out, float_out_delay, ready, error_in, error_out);
7
8  input wire clk;
9  input wire rstn;
10 input wire valid;
11 input wire [30:0] float_in;
12 input wire [30:0] float_in_delay;
13
14 input wire backprn;
15 output wire [30:0] float_out;
16 output wire [30:0] float_out_delay;
17 output wire ready;

```

```

18 output wire error_out;
19
20 wire [`EXP_SHIFT + `ROUND_SHIFT:0] M_sub;
21 wire [30:0] float_delay;
22 wire valid_p;
23 wire error_mid;
24
25 fp_1d5_sub_subtract_pipe subtract (clk, rstn, backprn, valid, float_in,
    ↪ float_in_delay, M_sub, float_delay, valid_p, error_in, error_mid);
26 fp_1d5_sub_correction_pipe correction(clk, rstn, backprn, valid_p, M_sub,
    ↪ float_delay, float_out, float_out_delay, ready, error_mid, error_out);
27
28 endmodule

```

Listing 19: modul fp_1d5_sub_subtract.v

```

1 `timescale 1ns / 1ps
2 `define EXP_SHIFT 23
3 `define ROUND_SHIFT 3
4
5 module fp_1d5_sub_subtract_pipe (clk, rstn, backprn, valid, float_in,
    ↪ float_in_delay, M_sub, float_out_delay, ready, error_in, error_out);
6
7 input wire clk;
8 input wire rstn;
9 input wire valid;
10 input wire [30:0] float_in;
11 input wire [30:0] float_in_delay;
12 input wire error_in;
13
14 input wire backprn;
15 output reg [`EXP_SHIFT + `ROUND_SHIFT:0] M_sub;
16 output reg [30:0] float_out_delay;
17 output reg ready;
18 output reg error_out;
19
20 wire [1:0] E_in;
21 reg [`EXP_SHIFT + `ROUND_SHIFT:0] M_in;
22 wire [22:0] M;
23
24 assign E_in = float_in[24:23];

```



```

25  assign M = float_in[22:0];
26
27  always @(posedge clk) begin
28      if(rstn == 1'b0) begin
29          ready <= 1'b0;
30          error_out <= 1'b0;
31      end else begin
32          if(backprn == 1'b0) begin
33              float_out_delay <= float_out_delay;
34              M_sub <= M_sub;
35              ready <= ready;
36              error_out <= error_out;
37          end else begin
38              if(valid == 1'b1) begin
39                  float_out_delay <= float_in_delay;
40                  M_sub <= ({1'b1, 23'h40_0000, 3'b000}) - M_in;
41                  ready <= 1'b1;
42                  error_out <= error_in;
43              end else begin
44                  float_out_delay <= float_in_delay;
45                  M_sub <= M_sub;
46                  ready <= 1'b0;
47                  error_out <= 1'b0;
48              end
49          end
50      end
51  end
52
53  always @* begin
54      if (E_in == 2'b10)
55          M_in = (({1'b1, M, 3'b000}) >> 1);
56      else if (E_in == 2'b01)
57          M_in = (({1'b1, M, 3'b000}) >> 2);
58      else
59          M_in = {1'b1, M, 3'b000};
60  end
61
62  endmodule

```

Listing 20: modul fp_1d5_sub_correction_pipe.v

```

1  `timescale 1ns / 1ps
2  `define EXP_SHIFT 23
3  `define ROUND_SHIFT 3
4
5  module fp_1d5_sub_correction_pipe (clk, rstn, backprn, valid, M_sub,
   ↪ float_in_delay, float_out, float_out_delay, ready, error_in, error_out);
6
7  input wire clk;
8  input wire rstn;
9  input wire valid;
10 input wire [`EXP_SHIFT + `ROUND_SHIFT:0] M_sub;
11 input wire [30:0] float_in_delay;
12 input wire error_in;
13
14 input wire backprn;
15 output reg [30:0] float_out;
16 output reg [30:0] float_out_delay;
17 output reg ready;
18 output reg error_out;
19
20 reg [`EXP_SHIFT + `ROUND_SHIFT:0] M_ov;
21 wire E_ov;
22 wire [7:0] E;
23
24 assign E = {7'b0111_111, E_ov};
25 assign E_ov = (M_sub[`EXP_SHIFT + `ROUND_SHIFT] == 1'b0) ? 1'b0 : 1'b1;
26
27 always @(posedge clk) begin
28     if(rstn == 1'b0) begin
29         ready <= 1'b0;
30         error_out <= 1'b0;
31     end else begin
32         if(backprn == 1'b0) begin
33             float_out_delay <= float_out_delay;
34             float_out <= float_out;
35             ready <= ready;
36             error_out <= error_out;
37         end else begin

```

```

38         if(valid == 1'b1) begin
39             float_out_delay <= float_in_delay;
40             float_out[30:23] <= E;
41             ready <= 1'b1;
42             error_out <= error_in;
43             if (M_ov[`ROUND_SHIFT-1] == 1'b1)
44                 float_out[22:0] <= M_ov[`EXP_SHIFT+`ROUND_SHIFT:`ROUND_SHIFT]
45                     ↪ + 1;
46             else
47                 float_out[22:0] <=
48                     ↪ M_ov[`EXP_SHIFT+`ROUND_SHIFT:`ROUND_SHIFT];
49             end else begin
50                 float_out_delay <= float_out_delay;
51                 float_out <= float_out;
52                 ready <= 1'b0;
53                 error_out <= 1'b0;
54             end
55         end
56     end
57 end
58
59 always @* begin
60     if (M_sub[`EXP_SHIFT+`ROUND_SHIFT] == 1'b0)
61         M_ov = M_sub << 1;
62     else
63         M_ov = M_sub;
64     end
65 end
66
67 endmodule

```

8.6.2 Testbench oraz wyniki

Listing 21: testbench float_1d5_sub_pipe_tb.sv

```
1  `timescale 1ns / 1ps
2
3  module float_1d5_sub_pipe_tb(
4
5  );
6
7  real out, out_expected;
8  logic clk, valid, ready, error_in, error_out;
9  logic [31:0] float_in;
10 logic [30:0] float_out, float_out_delay;
11 logic rstn;
12
13 logic [31:0] float_out_expected;
14 logic [67:0] testvectors [9:0];
15 logic [31:0] vecnum;
16 logic [3:0] error_4b;
17 integer f;
18
19
20 fp_sub_1d5_pipe fp_sub_1d5_pipeTB(clk, rstn, backprn, valid, float_in[30:0], 0,
   ↪ float_out, float_out_delay, ready, error_in, error_out);
21
22 initial
23 begin
24     $readmemh("sub_pipe_in.tv", testvectors);
25     f = $fopen("sub_pipe_out.txt", "w");
26     rstn <= 1;
27     vecnum <= 0;
28     valid <= 0;
29     clk <= 1'b1;
30     error_in <= 1'b0;
31     float_in <= 0;
32     #10
33     rstn <= 0;
34     #10
35     rstn <= 1;
36     #10
37     {float_in, float_out_expected, error_4b} = testvectors[vecnum];
```

```

38     vecnum = vecnum + 1;
39     error_in <= error_4b[0];
40     valid = 1'b1;
41 end
42
43 always begin
44     #5 clk <= ~clk;
45 end
46
47 always@(posedge clk)
48 begin
49
50     out <= $bitstoshortreal({1'b0, float_out});
51     out_expected = $bitstoshortreal(float_out_expected);
52     $fwrite(f, "%h\n", float_out);
53
54     if (valid == 1) begin
55         {float_in, float_out_expected, error_4b} = testvectors[vecnum];
56         vecnum = vecnum + 1;
57         error_in <= error_4b[0];
58     end
59
60     if (vecnum == 14) begin
61         $fclose(f);
62         $stop;
63     end
64 end
65
66 endmodule

```

out val	should be	abs diff	% abs diff
3f8cccd	3f8cccd	0,00E+00	0,00E+00
3f8cbc6b	3f8cbc6a	1,19E-07	1,08E-05
3f8cac08	3f8cac08	0,00E+00	0,00E+00
3f8c9ba6	3f8c9ba6	0,00E+00	0,00E+00
3f8c8b44	3f8c8b44	0,00E+00	0,00E+00
3f8c7ae1	3f8c7ae1	0,00E+00	0,00E+00
3f8c6a7f	3f8c6a7f	0,00E+00	0,00E+00
3f8c5a1d	3f8c5a1d	0,00E+00	0,00E+00
3f8c49ba	3f8c49ba	0,00E+00	0,00E+00
3f8c3958	3f8c3958	0,00E+00	0,00E+00

Tabela 9: Porównanie wyników na wyjściu z przewidywanymi - moduł sub_pipe.v

8.7 Podsumowanie uzyskanych wyników

Wyniki otrzymane w układzie z przetwarzaniem potokowym nie odbiegają jakością w porównaniu do układu bez pipeline'ingu. Dla wybranych wartości bitowych w podmodułach wyniki w większości są identyczne z przewidywanymi, w przypadku nadrzędnego modułu obliczającego odwrotność pierwiastka otrzymane wyniki dla takich samych kombinacji bitowych dokładność również waha się w podobnym zakresie - od 0.064% do 0.173%. Porównywalny poziom dokładności jest doskonałym dowodem na to, że wersja układu z przetwarzaniem jest pod względem algorytmu dokładnie tym samym, co pierwotny moduł invsqrt.v, układy te różnią się tylko szybkością działania.

8.8 AXI, Zynq

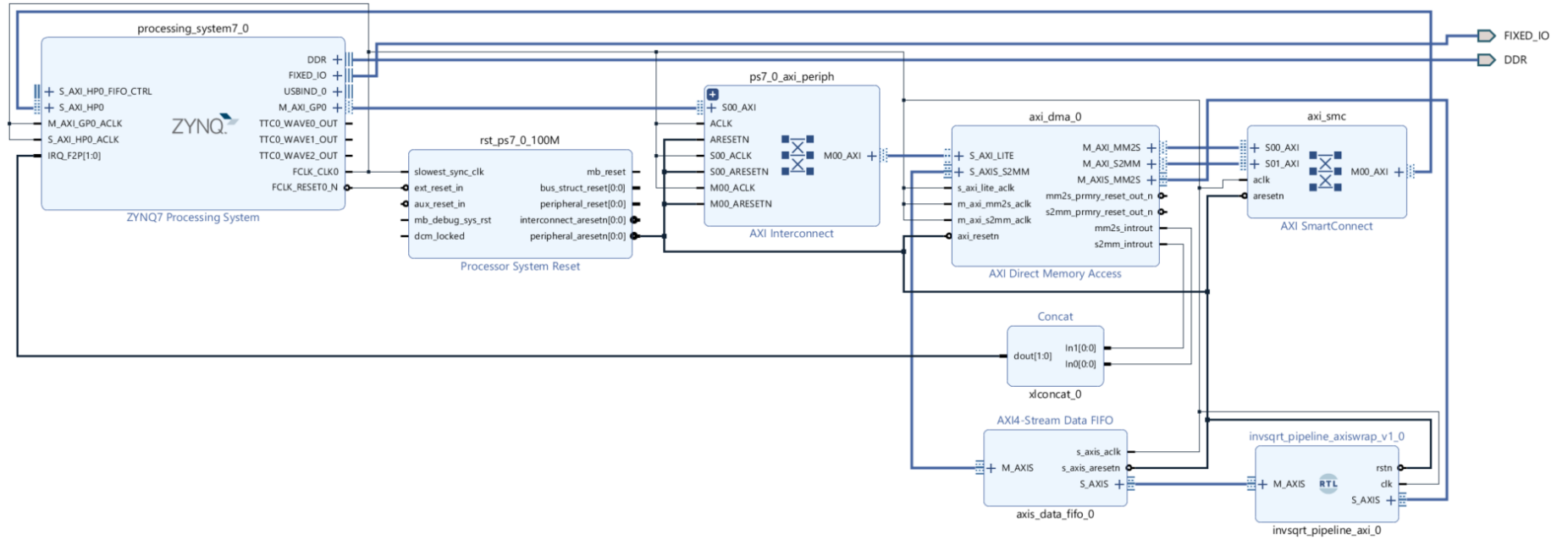
Układ obliczający odwrotność pierwiastka w wersji z przetwarzaniem potokowym wykorzystuje AXI Stream. Tak jak wcześniej układ składa się z procesora i układu obsługującego reset w układzie. Procesor przez AXI Interconnect przesyła rozkaz do DMA, aby wysłać określone dane w zadanej ilości do podłączonego modułu obliczającego odwrotność pierwiastka. Następnie dane te mają zostać odebrane z układu invsqrt z użyciem DMA. Do wejścia DMA podłączony jest układ invsqrt wraz z FIFO (AXI4-Stream Data FIFO), aby zakolejkować dane na wyjściu układu. Smart Connect podłączony na wyjściu DMA, które zapisują dane do wejścia procesora AXI High Performance Slave Interface. Układ ma zaimplementowaną obsługę przerw z DMA.

Podczas syntezy oraz implementacji układów wykorzystano następującą ilość zasobów układu:

Resource	Utilization	Available	Utilization %
LUT	194	53200	0.39
FF	351	106400	0.31
DSP	6	220	2.73

Tabela 10: Zużycie zasobów - synteza i implementacja

Podczas syntezy i implementacji zużyto taką samą ilość poszczególnych bloków. Procentowo tak jak w poprzednim rozwiązaniu najwięcej zajmowanych jest układów mnożących (2.73%), mniej tablic LUT, za to więcej przerzutników.



Rysunek 4: Block diagram układu z przetwarzaniem potokowym

8.9 Uruchomienie na sprzęcie

W celu uruchomienia układu na sprzęcie napisany został kod, w którym m.in. zawarte zostały funkcje do obsługi DMA. Wykorzystany został również kod z pierwszego listingu, zawierający pierwotną implementację algorytmu obliczającego odwrotność pierwiastka - została ona wykorzystana w celu weryfikacji poprawności otrzymanych wyników. Poniżej przedstawiono zawartość funkcji main napisanego programu.

Listing 22: funkcja main dla invsqrt_pipe

```
1  int main()
2  {
3      init_platform();
4      XAxiDma AxiDma;
5
6      xil_printf("\r\n--- Inverse Square Root Pipelined DMA Test start
   ↪  ---\r\n");
7
8      dma_input = (u32*)malloc(ARRAY_LENGTH*sizeof(u32));
9      hw_results = (u32*)calloc(ARRAY_LENGTH, sizeof(u32));
10     RxDone = 0;
11     TxDone = 0;
12
13     for(u8 i = 0; i < ARRAY_LENGTH; i++) {
14         dma_input[i] = *((u32*)&test[i]);
15     }
16     for(u8 i = 0; i < ARRAY_LENGTH; i++) {
17         sw_results[i] = Q_rsqrt(test[i]);
18     }
19
20     SetupDMA(&AxiDma);
21     Xil_DCacheDisable();
22
23     /* Send a packet */
24     XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR) hw_results,
   ↪  sizeof(u32)*ARRAY_LENGTH, XAXIDMA_DEVICE_TO_DMA);
25     XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR) dma_input,
   ↪  sizeof(u32)*ARRAY_LENGTH, XAXIDMA_DMA_TO_DEVICE);
26
27     while (!TxDone || !RxDone);
28     /*
```

```

29      * Test finished, check data
30      */
31      float in, out, sw_result;
32      for (u8 i = 0; i < ARRAY_LENGTH; i++) {
33          in = *(float*)&dma_input[i];
34          out = *(float*)&hw_results[i];
35          sw_result = sw_results[i];
36          printf("Input: %f,\tHW Output: %f,\tSW Result: %f,\tDifference:
37              ↪ %f \n", in, out, sw_result, out - sw_result);
38      }
39      xil_printf("\r\n--- Inverse Square Root Pipelined DMA Test finish
40              ↪ ---\r\n");
41      free(dma_input);
42      free(hw_results);
43      cleanup_platform();
44      return 0;
45  }

```

Na załączonym poniżej zrzucie ekranu przedstawione zostały wyniki użycia stworzonej funkcji. HW Output to wynik otrzymany na wyjściu układu, a SW Result to wynik otrzymany z wykorzystaniem oryginalnej funkcji.

```

COM6 - Tera Term VT
File Edit Setup Control Window Help

--- Inverse Square Root Pipelined DMA Test start ---
Input: 1.000000,      HW Output: 0.998307,      SW Result: 0.998307,      Difference: 0.000000
Input: 2.000000,      HW Output: 0.706930,      SW Result: 0.706930,      Difference: 0.000000
Input: 3.000000,      HW Output: 0.576847,      SW Result: 0.576847,      Difference: 0.000000
Input: 4.000000,      HW Output: 0.499154,      SW Result: 0.499154,      Difference: 0.000000
Input: 16.000000,     HW Output: 0.249577,      SW Result: 0.249577,      Difference: 0.000000
Input: 256.000000,    HW Output: 0.062394,      SW Result: 0.062394,      Difference: 0.000000
Input: nan,           HW Output: nan, SW Result: nan, Difference: nan
Input: inf,           HW Output: nan, SW Result: -inf,      Difference: nan
Input: 1000000.000000, HW Output: 0.000998,      SW Result: 0.000998,      Difference: 0.000000
Input: -1.000000,     HW Output: nan, SW Result: inf, Difference: nan
Input: -2.000000,     HW Output: nan, SW Result: inf, Difference: nan
Input: -0.500000,     HW Output: nan, SW Result: nan, Difference: nan
Input: 0.500000,      HW Output: 1.413860,      SW Result: 1.413860,      Difference: 0.000000
Input: 0.250000,      HW Output: 1.996614,      SW Result: 1.996614,      Difference: 0.000000
Input: 0.125000,      HW Output: 2.827720,      SW Result: 2.827720,      Difference: 0.000000
Input: 0.100000,      HW Output: 3.157232,      SW Result: 3.157232,      Difference: 0.000000
Input: 0.000001,      HW Output: 999.210449,    SW Result: 999.210449,    Difference: 0.000000
Input: 0.000000,      HW Output: nan, SW Result: 19817753709685768192.000000, Difference: nan

--- Inverse Square Root Pipelined DMA Test finish ---

```

Rysunek 5: Wyniki otrzymane na sprzęcie - screenshot z konsoli

List of Listings

1	Pierwotna implementacja algorytmu do obliczania odwrotności	
	pierwiastka	4
2	moduł invsqrt.v	8
3	moduł float_mul.v	11
4	moduł float_sq_mul.v	13
5	moduł float_mul.v	16
6	testbench invsqrt_tb.sv	21
7	testbench float_mul_tb.sv	24
8	testbench float_sq_mul_tb.sv	26
9	testbench float_sub_1d5_tb.sv	29
10	funkcja InvSqrt_calculate oraz funkcja main dla invsqrt	35
11	moduł float_mul.v	42
12	moduł invsqrt_pipeline_tb.sv	43
13	moduł invsqrt_pipe_init.v	45
14	moduł fp_mul_pipe.v	47
15	moduł fp_mul_multiply_pipe.v	48
16	moduł fp_mul_correction_pipe.v	49
17	testbench fp_mul_pipe_tb.sv	52
18	moduł float_1d5_sub_pipe.v	54
19	moduł fp_1d5_sub_subtract.v	55
20	moduł fp_1d5_sub_correction_pipe.v	57
21	testbench float_1d5_sub_pipe_tb.sv	59
22	funkcja main dla invsqrt_pipe	64

Spis tabel

1	Porównanie wyników na wyjściu z przewidywanymi - moduł invsqrt.v	23
2	Porównanie wyników na wyjściu z przewidywanymi - moduł float_mul.v	26
3	Porównanie wyników na wyjściu z przewidywanymi - moduł float_sq_mul.v	28
4	Porównanie wyników na wyjściu z przewidywanymi - moduł fp_sub_1d5.v	30
5	Zużycie zasobów - synteza	33
6	Zużycie zasobów - implementacja	33
7	Porównanie wyników na wyjściu z przewidywanymi - moduł invsqrt_pipeline.v	45
8	Porównanie wyników na wyjściu z przewidywanymi - moduł fp_mul_pipe.v	54
9	Porównanie wyników na wyjściu z przewidywanymi - moduł sub_pipe.v	61

10	Zużycie zasobów - synteza i implementacja	62
----	---	----