Politecnico di Torino
Dipartimento di Elettronica e Telecomunicazioni

Laboratory #2
Integrated Systems Architecture
Optimization of Digital Arithmetic in a Digital Filter

Master Degree in Electronic Engineering

Authors: Group 1

Alberto Aimaro 253196
Beatrice Bussolino 251190
Alessio Colucci 251197
Fabio Zanoni 232113

January 31, 2019

Contents

CHAPTER 1

Digital Arithmetic and Logic Synthesizers

1.1 Adders and Multipliers

The last developed architecture in LAB1, a FIR filter optimized with unfolding and pipelining, is re-synthesized forcing Design Compiler to use Design Ware adders and multipliers for all the cells of the design. The possible adders that can be used are: ripple-carry adder, carry-look-ahead adder and parallel-prefix adder; the available multipliers are: carry-save multiplier and parallel-prefix multiplier.

All the possible combinations of adders-multipliers are synthesized to find the maximum operating frequency. Then all the details as area and power consumption are get at $f_{max}/4$. In Table 1.1 all results are reported.

Add/Mult	$T_{min}[ns]$	$f_{max}[MHz]$	$f_{max}/4$ [MHz]	Area [μm^2]	Dynamic P. [mW]	Leakage P. [μW]
Ripple-Carry / Carry-Save	2.2	454.55	113.64	19715.654297	2.5099	369.1261
Ripple-Carry/Parallel-Prefix	2.0	500.00	125.00	17326.441406	2.5140	391.0926
Carry-Look-Ahead/Carry-Save	2.2	454.55	113.64	20004.263672	2.6082	384.0351
Carry-Look-Ahead/Parallel-Prefix	2.25	444.44	111.11	17400.656250	2.5584	398.9084
Parallel-Prefix/Carry-Save	2.2	454.55	443.64	19680.541016	2.5073	368.9240
Parallel-Prefix/Parallel-Prefix	2.2	454.55	113.64	17461.037109	2.5984	397.7968
Original Architecture	2.0	500	125	18244.408203	2.3873	421.1755

Table 1.1: Comparison of the same architecture implemented with different combinations of adders and multipliers

By testing all possible combinations of adders and multipliers it is found out that there are no significant variations in maximum frequency, area or power consumption. This is due also to the fact that the filter has a low data parallelism, that does not allow to see eventual improvements due to a good choice of adders and multipliers.

1.2 Pipelining

The performance of the design is then optimized inserting additional pipeline registers after the multiplier and issuing the command `compile_ultra`. To find the ideal number of registers, different tests are performed. In Table 1.2 is reported the minimum period for different levels of pipeline:

4 pipe registers	1.20 ns
5 pipe registers	1.02 ns
6 pipe registers	1.02 ns

Table 1.2: Minimum period for different numbers of pipe registers

From Table 1.2 it is possible to see that the optimal number of pipe registers is 5, since further increasing it does not lead to performance improvements.

The optimized architecture is then synthesized and simulated at frequency $f_{max}/4$. The results are reported in Table 1.3, where there is also a comparison with the original design.

	Original Architecture	Pipelined Architecture
T_{min}	2 ns	1.02 ns
f_{max}	500.00MHz	980.39MHz
$T_{min} * 4$	8 ns	4.08 ns
$f_{max}/4$	125.00MHz	245.10MHz
Area μm^2	18244.408203	24389.539062
Dynamic Power [mW]	2.9841	7.9386
Leakage Power [μW]	421.1755	482.7730

Table 1.3: Comparison between original architecture and pipelined architecture

CHAPTER 2

Manual Multiplier Optimization

After using Design Compiler to try to automatically improve the design by pipelining and manually mapping the arithmetic operators, a manually optimized multiplier is implemented: the basic principle relies on a Dadda tree fed on Modified Booth Encoding's partial products.

2.1 Version 1

The first version is a standard implementation of a correct multiplier: a Modified Booth Encoding (Figure 2.1) is used to reduce the number of partial products (in particular there are 10-bit inputs, so with MBE the number of partial products is decreased from 10 to 5). However, since the inputs are signed, it would be necessary to process also the sign extension: following Roorda's paper, these passages can be improved, optimizing only the bits in common with the following partial product. It is also necessary to add the MSB - 1 bit, as well as the LSB for Roorda's optimization, thus another partial product is inserted to contain all the added bits.

$$PP_j = (X_{2i} \oplus X_{2i-1})(X_{2i+1} \oplus Y_j) + \overline{(X_{2i} \oplus X_{2i-1})}(X_{2i+1} \oplus X_i)(X_{2i+1} \oplus Y_{j-1}).$$

Figure 2.1: Modified Booth Encoding equation implemented in the Dadda trees

Then Dadda's approach is followed, using the minimum number of Full Adders and Half Adders as compressors to reduce the number of partial products from 6 to 4, then to 3 and finally to 2;then the automatically optimized "+" operator is used to obtain the final result.

To check the improvements of Roorda's approach over the standard MBE, a standard Dadda tree is used, covering all the partial products with FAs and HAs, also for the sign extension. Then, to avoid the overhead of the FAs, a second version is developed, where the full adders for the sign extension are optimized by hand.

The result for the Dadda tree can be seen in Figure 2.2.

The two architectures are synthesized and tested. In Table 2.1 are reported the results of the synthesis.

As happened when trying the different combinations of adders and multipliers, on a small design it is not easy to appreciate timing, area or power optimizations.

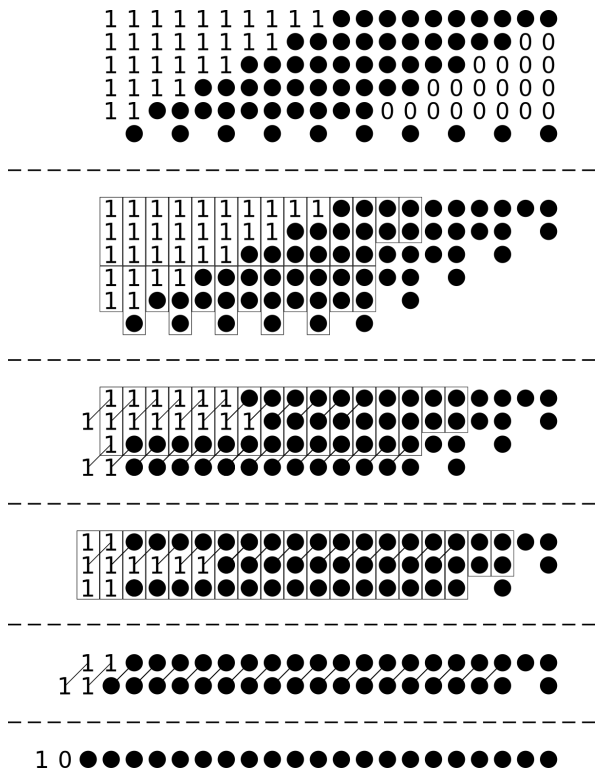


Figure 2.2: Standard Dadda tree

	$T_{min}[ns]$	$f_{max}[MHz]$	$f_{max}/4$ [MHz]	Area [μm^2]	Dynamic P. [mW]	Leakage P. [μW]
Manually optimized Dadda tree	2.22	450.45	112.61	17538.708984	3.0552	386.3891
Standard Dadda tree	2.20	454.55	113.64	17551.210938	3.1046	381.7069

Table 2.1: Synthesis results for manually optimized Dadda tree and standard Dadda tree

2.2 Version 2

Then, to improve performances and area/power, the adders up to the 6^{th} LSB are removed, introducing an error but reducing the area and the power. The introduced error is very small, since only 10 MSB are used in the final result.

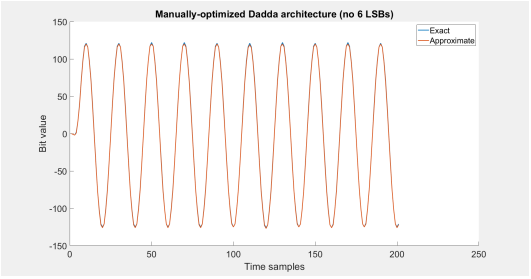
Both the Roorda implementation and the standard one are tested, to observe the differences.

Synthesis results of these two designs are reported in Table 2.2. results confirm the decrease of area and power consumption with respect to Version 1 architectures.

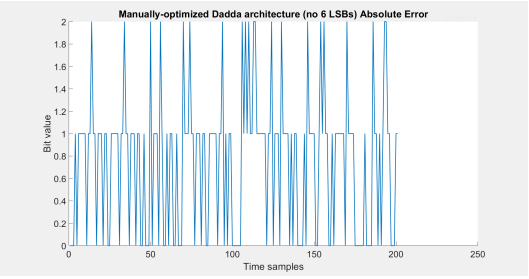
However it is necessary to highlight an error that surfaced while simulating these designs: the errors coming from the 4-2 compressors are so high that when used inside the FIR filter they make the whole architecture overflow with respect to the correct results, leading to possible problems in correctly synthesizing and placing the design.

In Figure 2.3 and 2.4 are reported the comparisons with exact output values and the absolute error of manually optimized Dadda tree and of standard Dadda tree without 6 LSBs. The errors, assuming

a normal inputs distribution, are shown respectively in Figure 2.5 and 2.6

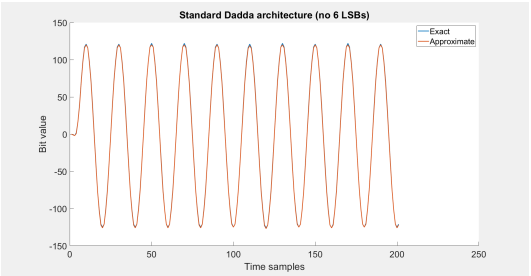


(a) *Manually-optimized Dadda tree without 6 LSBs, signal comparison with exact implementation*

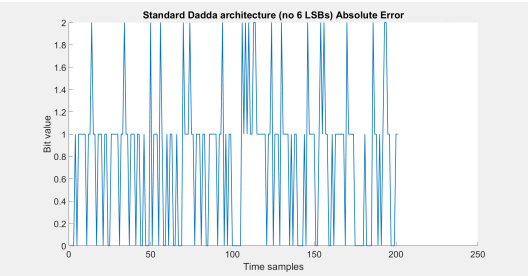


(b) *Manually-optimized Dadda tree without 6 LSBs, maximum absolute error*

Figure 2.3: Results and errors of manually optimized Dadda tree without 6 LSBs



(a) *Standard Dadda tree without 6 LSBs, signal comparison with exact implementation*



(b) *Standard Dadda tree without 6 LSBs, maximum absolute error*

Figure 2.4: Results and errors of standard Dadda tree without 6 LSBs

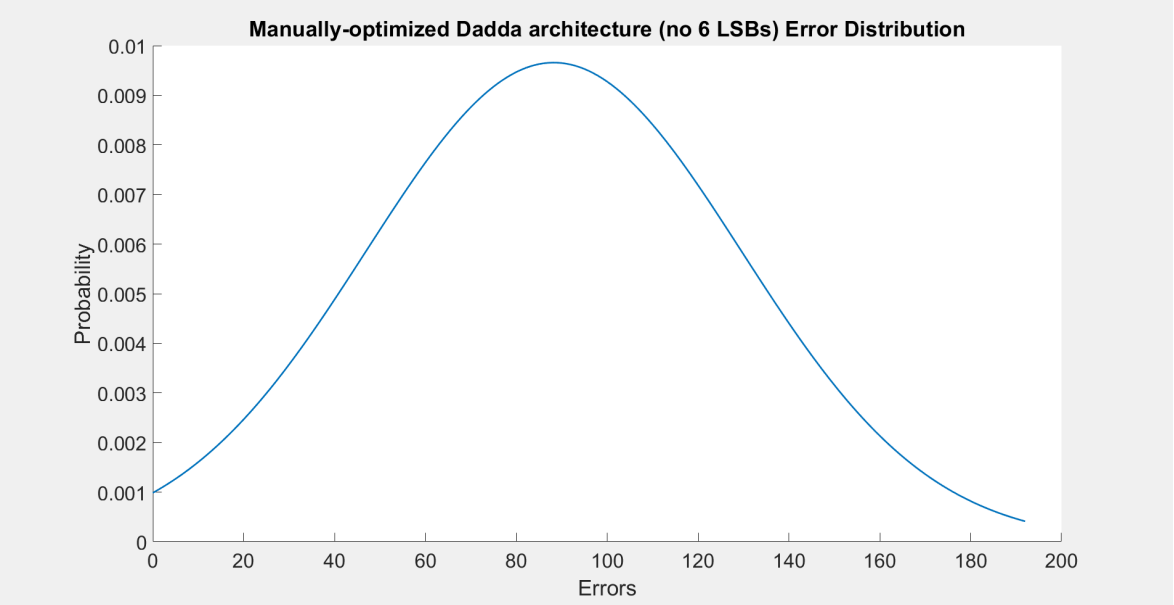


Figure 2.5: Manually-optimized Dadda tree without 6 LSBs, error distribution

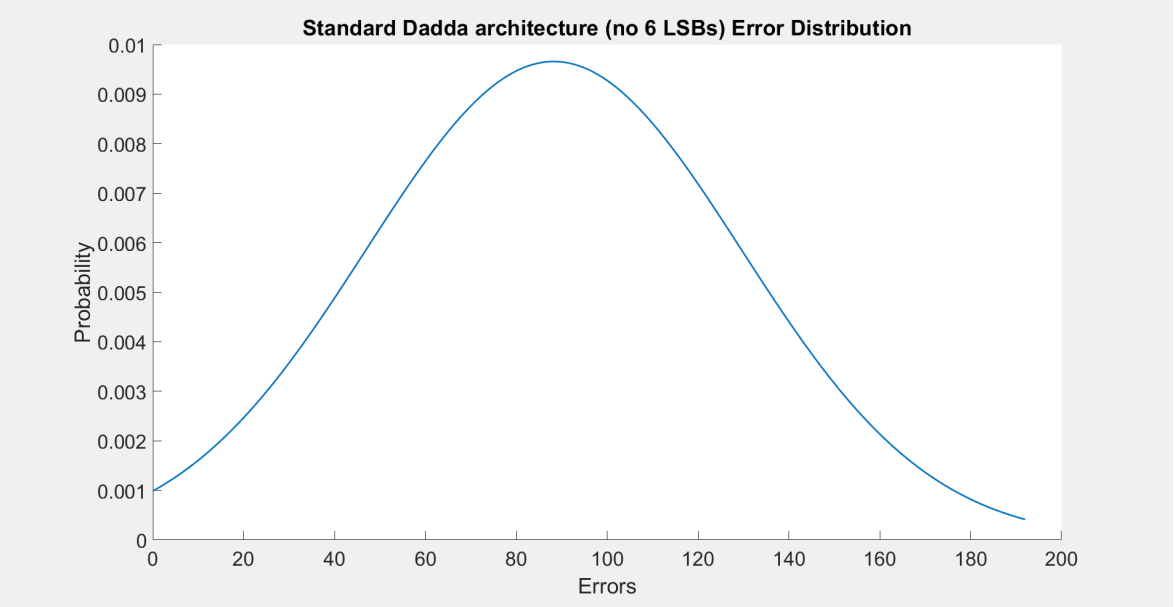


Figure 2.6: Standard Dadda tree without 6 LSBs, error distribution

	$T_{min}[ns]$	$f_{max}[MHz]$	$f_{max}/4$ [MHz]	Area [μm^2]	Dynamic P. [mW]	Leakage P. [μW]
Manually optimized Dadda tree	2.02	495.05	123.38	15517.109375	2.7685	334.2000
Standard Dadda tree	2.01	497.51	124.38	15483.859375	2.7908	333.9016

Table 2.2: Synthesis results for manually optimized Dadda tree and standard Dadda tree, with adders up to 6^{th} LSB removed

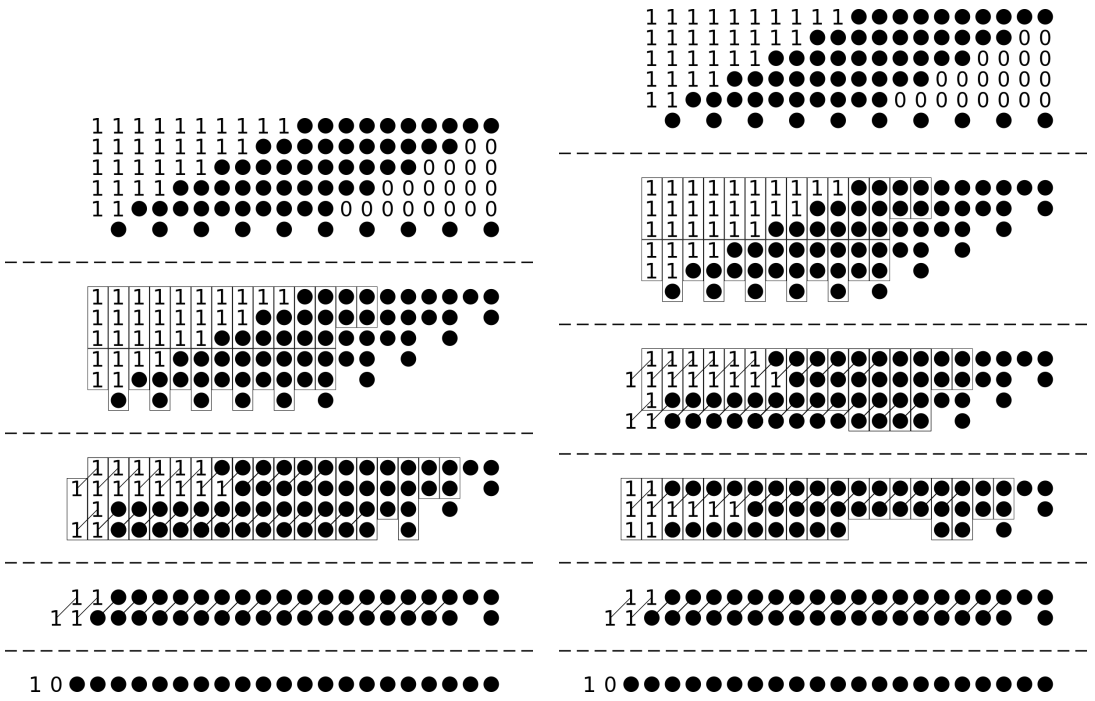
2.3 Version 3

As a final version, approximate 4-2 compressors are used as well as an approximate MBE, to try and reduce the area/power/timing cost of the multiplier without losing too much on precision.

The full-approximate structures are implemented using AMBE with manually optimized standard Dadda tree, using 4-2 compressors in a Dadda tree layer (reducing by 1 the total number of layers, as can be seen in Figure 2.7a) and using both at the same time.

Then a customizable model and testbench are created for testing the performances of different approximations: the AMBE can be mixed with the standard MBE, obtaining a range of architectures from MBE-only to AMBE-only; for example, an architecture AMBE1 will have an AMBE applied to the first row of partial products. Also the 4-2 compressors can be inserted in the second layer of the Dadda tree, obtaining architectures in a range from standard Dadda tree to almost-fully approximated, with approximations starting from the LSBs.

Doing a full simulation and crunching the errors in MATLAB, using a normal distribution of all the errors since each possible input is passed only once, it is possible to find the best approximating architecture: higher AMBE approximations leads to lower mean error (negative), while higher 4-2 compressor approximation leads to higher positive mean error. Thus, choosing a mixed approximation (in this case two levels with AMBE and four 4-2 compressors) can lead to the best result, since the



(a) *Dadda tree with 4-2 compressors*

(b) *Final version of Dadda tree with fewer 4-2 compressors*

Figure 2.7: Dadda tree with approximated compressors

	$T_{min}[ns]$	$f_{max}[MHz]$	$f_{max}/4$ [MHz]	Area [μm^2]	Dynamic P. [mW]	Leakage P. [μW]
Manually optimized Dadda tree with 4-to-2 compressors in second layer	2.01	497.51	124.38	14213.975586	2.2762	301.4014
Manually optimized Dadda tree with AMBE in partial products generation	2.20	454.55	113.64	14487.423828	2.0337	319.8428
Manually optimized Dadda tree with AMBE in partial product generation and 4-to-2 compressors in second layer	2.00	500.00	125.00	11938.877930	1.5957	254.1680
Manually optimized Dadda tree with AMBE2 and four 4-to-2 compressors in second layer	2.20	454.55	113.64	15771.406250	2.6467	344.4026

Table 2.3: Synthesis results for different approximating architectures

average error must be divided by 2^{10}, since only the 10 highest bits are used. This result will also be validated using Design Compiler to check the improvements on area/power. The comparison between

various approximating architectures can be seen in Figure 2.8.The distributions for the solutions at the extremes are present in 2.9, 2.10, 2.11

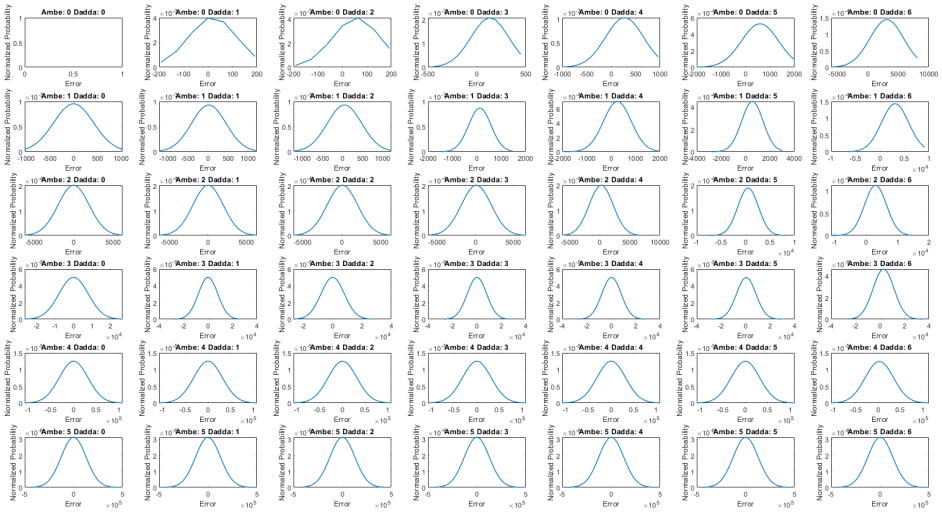


Figure 2.8: Comparison of different Dadda trees using AMBE and 4-2-compressors variable approximations

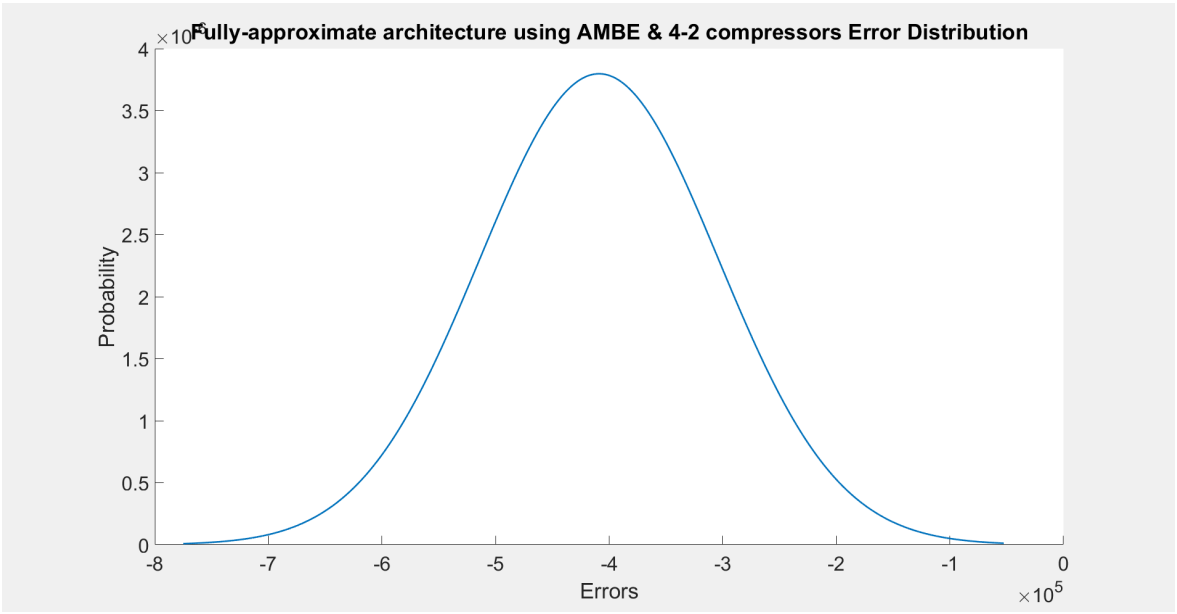


Figure 2.9: Error distribution for AMBE with 4-2 compressors

All the errors are compared, along with a comparison with the exact output signal: for the full approximations there are huge distortions, while the chosen approximation is quite faithful to the exact one; depending on the application, this choice could be acceptable. These comparisons are shown in Figure 2.12a & 2.12b, 2.13a & 2.13b, 2.14a & 2.14b

The synthesis results show that the more the architecture is approximate, the higher the clock and lower the area and the power: obviously also the errors increase, as shown in the comparison figures

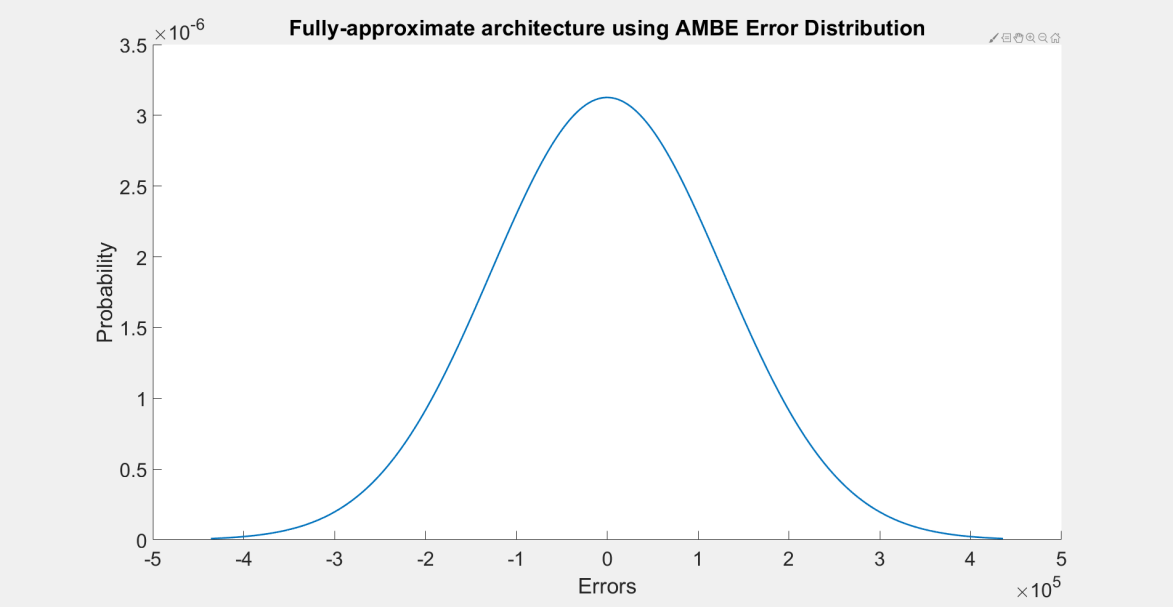


Figure 2.10: Error distribution for AMBE

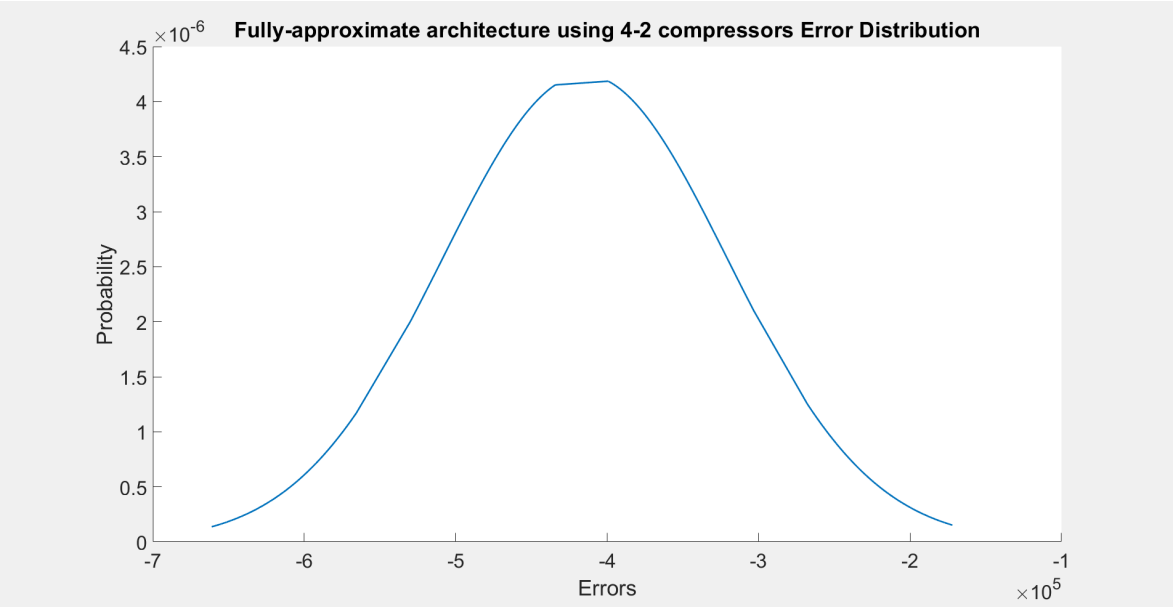
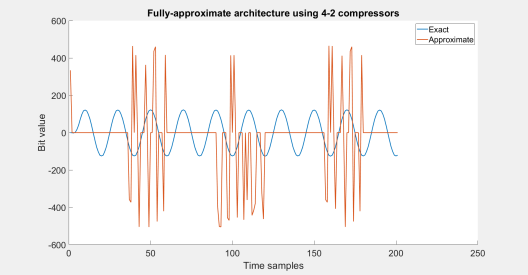


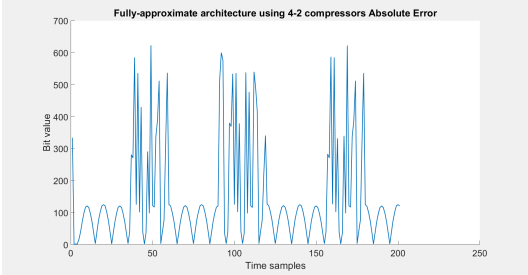
Figure 2.11: Error distribution for MBE with 4-2 compressors

mentioned above. That is why a mixed solution is chosen, to decrease the maximum absolute value of the error (as it is shown in Figure 2.15a & 2.15b).

Summing up, the automatically optimized and pipelined design obtains the best performance (achieved also using high-order approximations), but area and power are amongst the worse ones. On the other side, manually-approximate architectures are much smaller but obviously the results are not always exact. So the best solution would be to allow some small inexact results to reduce area, power and period, without affecting too much the final shape of the signal.

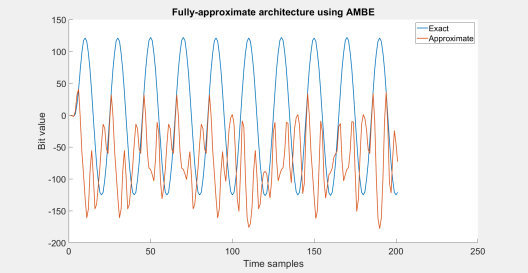


(a) *Fully-approximated Dadda tree using 4-2 compressors, signal comparison with exact implementation*

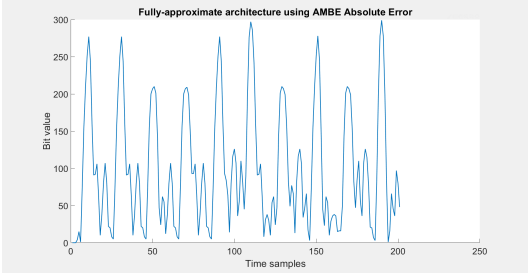


(b) *Fully-approximated Dadda tree using 4-2 compressors, maximum absolute error*

Figure 2.12: Results and error for Dadda tree with 4-to-2 compressors

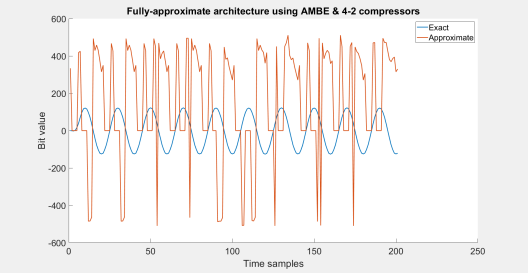


(a) *Fully-approximated Dadda tree using AMBE, signal comparison with exact implementation*

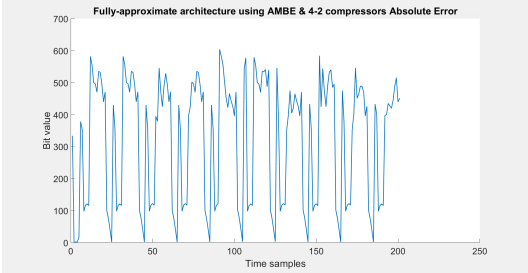


(b) *Fully-approximated Dadda tree using AMBE, maximum absolute error*

Figure 2.13: Results and error for Dadda tree with AMBE

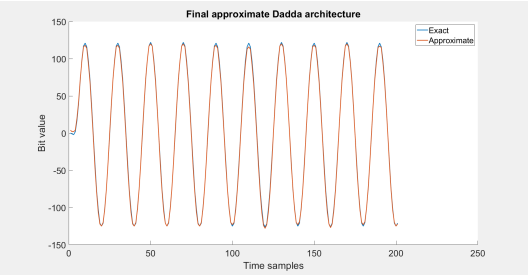


(a) *Fully-approximated Dadda tree using 4-2 compressors and AMBE, signal comparison with exact implementation*

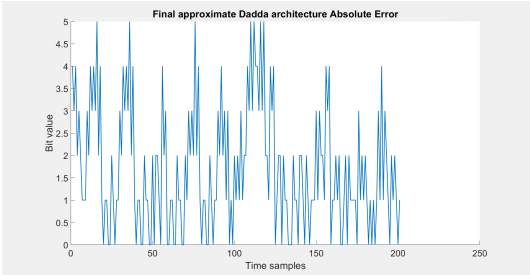


(b) *Fully-approximated Dadda tree using 4-2 compressors and AMBE, maximum absolute error*

Figure 2.14: Results and error for Dadda tree with AMBE and 4-to-2 compressors



(a) *Final architecture for the Dadda tree using 2 AMBE levels and 4 4-2 compressors, signal comparison with exact implementation*



(b) *Final architecture for the Dadda tree using 2 AMBE levels and 4 4-2 compressors, maximum absolute error*

Figure 2.15: Results and error for Dadda tree with AMBE

Appendices

APPENDIX A

Manually-pipelined FIR filter

The first manually optimized version with all the pipes.

A.0.1 D-Flip-Flop

./Code/dff.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

-- Flip Flop di tipo D, con parallelismo N e reset asincrono

ENTITY dff IS
PORT (D : IN STD_LOGIC;      -- ingresso
     Clock, Resetn, EN : IN STD_LOGIC;            -- clock, reset, enable
     Q : OUT STD_LOGIC);      -- uscita
END dff;

ARCHITECTURE Behavior OF dff IS
BEGIN
PROCESS (Clock)
BEGIN

     IF (Clock'EVENT AND Clock = '1') THEN  -- se c'è il fronte
     IF (resetn = '0') then
          q <= '0';
     elsIF (EN='1') THEN  -- e enable è attivo
          Q <= D;
          END IF;
END IF;
END PROCESS;
END Behavior;
```

A.0.2 Register

./Code/regn.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
USE ieee.numeric_std.all;

-- Flip Flop di tipo D, con parallelismo N e reset asincrono

ENTITY regn IS
GENERIC (N: INTEGER := 16);                    -- numero di bit del registro
PORT (D : IN SIGNED (N-1 DOWNTO 0);      -- ingresso
     Clock, Resetn, EN : IN STD_LOGIC;          -- clock, reset, enable
     Q : OUT SIGNED (N-1 DOWNTO 0));      -- uscita
END regn;

ARCHITECTURE Behavior OF regn IS
BEGIN
PROCESS (Clock, Resetn)
BEGIN

     IF (Clock'EVENT AND Clock = '1') THEN  -- se c'è il fronte
     IF (resetn = '0') then
          q <= (others => '0');
     elsIF (EN='1') THEN  -- e enable è attivo
          Q <= D;
          END IF;
END IF;
END PROCESS;
END Behavior;
```

A.0.3 Common constants - Util

./Code/util.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.math_real.all;

PACKAGE util IS

     CONSTANT Nbit: INTEGER := 10;
     CONSTANT N: INTEGER := 9;-- number of coefficients
     CONSTANT P: INTEGER := 3; --- unfolding factor
     CONSTANT W1: INTEGER := 2; --- MAX NUMBER OF INTERMEDIATE REGISTER FOR DIN(P*K)
     CONSTANT W2: INTEGER := 3; --- MAX NUMBER OF INTERMEDIATE REGISTER FOR DIN(P*K+1)
     CONSTANT W3: INTEGER := 3; --- MAX NUMBER OF INTERMEDIATE REGISTER FOR DIN(P*K+2)
     --CONSTANT Nbit_result: INTEGER := Nbit+integer(ceil(log2(real(N))));
     CONSTANT Nbit_result: INTEGER := Nbit;

     CONSTANT FINAL_DELAY : integer := N + 5;

     CONSTANT T: time := 20 ns; -- PERIODO DEL CLOCK
     CONSTANT start_time: time := 101 ns; -- momento di inizio della prima operazione,
     viene dato il I start;

     TYPE LIST_N IS ARRAY (0 to W3) OF SIGNED(Nbit-1 downto 0);
     TYPE LIST_mult IS ARRAY (0 to N-1) OF SIGNED(Nbit+Nbit-1 downto 0);
     TYPE LIST_mult_resize IS ARRAY (0 to N-1) OF SIGNED(Nbit downto 0);

     TYPE LIST_sum_1 IS ARRAY (0 to (N/2)-1) OF SIGNED((Nbit+1)-1 downto 0);
     TYPE LIST_sum_2 IS ARRAY (0 to ((((N*P)/2)-1)/2)+1) OF SIGNED(Nbit downto 0);
     ----array used in folding structure-------------------------------------------
     TYPE input_format_type          IS ARRAY (0 to P-1) OF SIGNED(Nbit-1 downto 0);
```

```
30      TYPE OUT_PIPES_TYPE              IS ARRAY (0 TO P-1) OF LIST_N;
31      TYPE mult_array_TYPE             IS ARRAY (0 TO P-1) OF LIST_mult;
32      TYPE mult_resize_array_TYPE      IS ARRAY (0 TO P-1) OF LIST_mult_resize;
33
34      type partial_array is array(Nbit / 2 downto 0) of signed(2 * Nbit - 1 downto 0);
35      --type internal_partial_array is array(12 + N / 2 downto 0) of signed(2 * N + 2
        downto 0);
36      type internal_partial_array is array(integer range <>) of signed(2 * Nbit + 2
        downto 0);
37      type multiple_factoring_array is array(Nbit / 2 - 1 downto 0) of signed(Nbit
        downto 0);
38
39  END util;
```

A.0.4 FIR filter code

./Code/FIR_filter_pipe.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   library std;
6   USE work.util.all;
7
8   ENTITY FIR_filter IS
9
10  PORT(    CLK:     in std_logic;
11           RST_n:   in std_logic;
12
13           VIN:     in std_logic;
14           VOUT:    out std_logic;
15
16           -- we have Nbit * P types for Verilog, which does not support custom types
17           DIN:     in std_logic_vector(Nbit * P - 1 downto 0);
18           DOUT:    out std_logic_vector(Nbit * P -1 downto 0);
19
20           b:       in std_logic_vector(N * Nbit - 1 downto 0));
21
22  END ENTITY FIR_filter;
23
24  ARCHITECTURE behavior OF FIR_filter IS
25
26  COMPONENT regn IS
27  GENERIC (N: INTEGER := 16);                    -- numero di bit del registro
28  PORT (D : IN SIGNED (N-1 DOWNTO 0);      -- ingresso
29      Clock, Resetn, EN : IN STD_LOGIC;          -- clock, reset, enable
30      Q : OUT SIGNED (N-1 DOWNTO 0));      -- uscita
31  END COMPONENT regn;
32
33  component dff IS
34  PORT (D : IN STD_LOGIC;      -- ingresso
35      Clock, Resetn, EN : IN STD_LOGIC;          -- clock, reset, enable
36      Q : OUT STD_LOGIC);      -- uscita
37  END component;
38
39  SIGNAL xz: OUT_PIPES_TYPE;
40  SIGNAL mult: mult_array_TYPE;
41  signal mult_out_pipe: mult_resize_array_TYPE;
42
```

```
43 SIGNAL mult_resize: LIST_mult_resize;
44 SIGNAL sum_1_in,sum_2_in,sum_3in,sum_1_out,sum_2_out,sum_3out: LIST_sum_1;
45 SIGNAL PIPE_REG_MULT_2_8:SIGNED(Nbit+1-1 downto 0);
46
47 type sum1_reg_type is array (0 to 4-1) of SIGNED((Nbit+1)-1 downto 0);
48 type sum1_type is array (0 to P-1) of sum1_reg_type;
49 SIGNAL sum1_reg_in,sum1_reg_out : sum1_type;
50
51 type sum2_reg_type is array (0 to 2-1) of SIGNED((Nbit+2)-1 downto 0);
52 type sum2_type is array (0 to P-1) of sum2_reg_type;
53 SIGNAL sum2_reg_in,sum2_reg_out : sum2_type;
54
55 type sum3_type is array (0 to P-1) of SIGNED((Nbit+3)-1 downto 0);
56 SIGNAL sum3_reg_in,sum3_reg_out : sum3_type;
57 SIGNAL REG8_EXTENDED : SIGNED(Nbit+3-1 DOWNTO 0);
58
59 type sum_final_type is array (0 to P-1) of SIGNED((Nbit+4)-1 downto 0);
60 SIGNAL sum_final : sum_final_type;
61
62 type reg_8_reg_type is array (0 to 4-1) of SIGNED((Nbit+1)-1 downto 0);
63 type reg_8_type is array (0 to P-1) of  reg_8_reg_type;
64 SIGNAL reg_8_value : reg_8_type;
65 signal reg_8_value_not_aggregate: signed(Nbit-1 downto 0);
66
67 type pipelined_type is array (0 to 4) of std_logic;
68 SIGNAL en_shift_p,vout_p : pipelined_type;
69
70 SIGNAL VIN_retard : std_logic;
71
72 TYPE state IS (RESET,IDLE,DATA_CYCLE1,DATA_CYCLE2,LAST_DATA1);
73 SIGNAL present_state : state;
74 SIGNAL EN_REG_1,EN_REG_OUT,EN_SHIFT,RST_INT_n,EN_FIRST_REG,VOUT1 : STD_LOGIC;
75
76 SIGNAL DOUT1,DOUT2,DOUT3 :SIGNED(Nbit-1 DOWNTO 0 );
77 BEGIN
78 ---- DATAPATH-----------------------------------------------------
79 EN_FIRST_REG<=(EN_REG_1 or EN_SHIFT);
80 in_reg_3k: regn      generic map (N => Nbit)
81                 port map (D => signed(DIN(3 * Nbit - 1 downto 2 * Nbit)), Clock => CLK
       , Resetn => RST_INT_n, EN => EN_FIRST_REG , Q => xz(0)(0));
82 in_reg_3k_plus_1: regn  generic map (N => Nbit)
83                 port map (D => signed(DIN(2 * Nbit - 1 downto Nbit)), Clock => CLK,
       Resetn => RST_INT_n, EN => EN_FIRST_REG , Q => xz(1)(0));
84 in_reg_3k_plus_2: regn  generic map (N => Nbit)
85                 port map (D => signed(DIN(Nbit -1 downto 0)), Clock => CLK, Resetn =>
       RST_INT_n, EN => EN_FIRST_REG , Q => xz(2)(0));
86 ---------------out register          ------------------------------------------
87 out_reg_1: regn generic map (N => Nbit)
88                 port map (D => sum_final(0)((Nbit+1)-1 downto (Nbit+1)-1-Nbit+1),
       Clock => CLK, Resetn => RST_INT_n, EN => en_shift_p(4), Q => DOUT1(Nbit -1 downto
       0));
89 out_reg_2: regn generic map (N => Nbit)
90                 port map (D => sum_final(1)((Nbit+1)-1 downto (Nbit+1)-1-Nbit+1),
       Clock => CLK, Resetn => RST_INT_n, EN => en_shift_p(4), Q => DOUT2(Nbit - 1 downto
        0));
91 out_reg_3: regn generic map (N => Nbit)
92                 port map (D => sum_final(2)((Nbit+1)-1 downto (Nbit+1)-1-Nbit+1),
       Clock => CLK, Resetn => RST_INT_n, EN => en_shift_p(4), Q => DOUT3(Nbit - 1 downto
        0));
93 DOUT( Nbit - 1  downto 0)       <=std_logic_vector(DOUT3);
94 DOUT(2 * Nbit-1 downto Nbit)    <=std_logic_vector(DOUT2);
95 DOUT(3 * Nbit-1 downto 2*Nbit)  <=std_logic_vector(DOUT1);
```

```
96
97  ----------------end modify-----------------------------------------
98  shift_reg_3k: for i in 0 to W1-1 generate
99      reg_i: regn generic map (N => Nbit)
100                 port map (D => xz(0)(i), Q => xz(0)(i+1), Clock => CLK, Resetn =>
        RST_INT_n, EN => EN_SHIFT);
101 end generate shift_reg_3k;
102
103 shift_reg_3k_plus_1: for i in 0 to W2-1 generate
104     reg_i: regn generic map (N => Nbit)
105                 port map (D => xz(1)(i), Q => xz(1)(i+1), Clock => CLK, Resetn =>
        RST_INT_n, EN => EN_SHIFT);
106 end generate shift_reg_3k_plus_1;
107
108 shift_reg_3k_plus_2: for i in 0 to W3-1 generate
109     reg_i: regn generic map (N => Nbit)
110                 port map (D => xz(2)(i), Q => xz(2)(i+1), Clock => CLK, Resetn =>
        RST_INT_n, EN => EN_SHIFT);
111 end generate shift_reg_3k_plus_2;
112 -----
113 mult(0)(0)<= signed(b((0 + 1) * Nbit - 1 downto 0 * Nbit)) * xz(0)(0);
114 mult(0)(1)<= signed(b((1 + 1) * Nbit - 1 downto 1 * Nbit)) * xz(2)(1);
115 mult(0)(2)<= signed(b((2 + 1) * Nbit - 1 downto 2 * Nbit)) * xz(1)(1);
116 mult(0)(3)<= signed(b((3 + 1) * Nbit - 1 downto 3 * Nbit)) * xz(0)(1);
117 mult(0)(4)<= signed(b((4 + 1) * Nbit - 1 downto 4 * Nbit)) * xz(2)(2);
118 mult(0)(5)<= signed(b((5 + 1) * Nbit - 1 downto 5 * Nbit)) * xz(1)(2);
119 mult(0)(6)<= signed(b((6 + 1) * Nbit - 1 downto 6 * Nbit)) * xz(0)(2);
120 mult(0)(7)<= signed(b((7 + 1) * Nbit - 1 downto 7 * Nbit)) * xz(2)(3);
121 mult(0)(8)<= signed(b((8 + 1) * Nbit - 1 downto 8 * Nbit)) * xz(1)(3);
122
123 mult(1)(0)<= signed(b((0 + 1) * Nbit - 1 downto 0 * Nbit)) * xz(1)(0);
124 mult(1)(1)<= signed(b((1 + 1) * Nbit - 1 downto 1 * Nbit)) * xz(0)(0);
125 mult(1)(2)<= signed(b((2 + 1) * Nbit - 1 downto 2 * Nbit)) * xz(2)(1);
126 mult(1)(3)<= signed(b((3 + 1) * Nbit - 1 downto 3 * Nbit)) * xz(1)(1);
127 mult(1)(4)<= signed(b((4 + 1) * Nbit - 1 downto 4 * Nbit)) * xz(0)(1);
128 mult(1)(5)<= signed(b((5 + 1) * Nbit - 1 downto 5 * Nbit)) * xz(2)(2);
129 mult(1)(6)<= signed(b((6 + 1) * Nbit - 1 downto 6 * Nbit)) * xz(1)(2);
130 mult(1)(7)<= signed(b((7 + 1) * Nbit - 1 downto 7 * Nbit)) * xz(0)(2);
131 mult(1)(8)<= signed(b((8 + 1) * Nbit - 1 downto 8 * Nbit)) * xz(2)(3);
132
133 mult(2)(0)<= signed(b((0 + 1) * Nbit - 1 downto 0 * Nbit)) * xz(2)(0);
134 mult(2)(1)<= signed(b((1 + 1) * Nbit - 1 downto 1 * Nbit)) * xz(1)(0);
135 mult(2)(2)<= signed(b((2 + 1) * Nbit - 1 downto 2 * Nbit)) * xz(0)(0);
136 mult(2)(3)<= signed(b((3 + 1) * Nbit - 1 downto 3 * Nbit)) * xz(2)(1);
137 mult(2)(4)<= signed(b((4 + 1) * Nbit - 1 downto 4 * Nbit)) * xz(1)(1);
138 mult(2)(5)<= signed(b((5 + 1) * Nbit - 1 downto 5 * Nbit)) * xz(0)(1);
139 mult(2)(6)<= signed(b((6 + 1) * Nbit - 1 downto 6 * Nbit)) * xz(2)(2);
140 mult(2)(7)<= signed(b((7 + 1) * Nbit - 1 downto 7 * Nbit)) * xz(1)(2);
141 mult(2)(8)<= signed(b((8 + 1) * Nbit - 1 downto 8 * Nbit)) * xz(0)(2);
142 -----
143
144 mult_reg: for i in 0 to P-1 generate
145             sec:for k in 0 to N-1 generate
146                 mult_reg_i: regn generic map (N => Nbit+1)
147                                     port map (D => mult(i)(k)(Nbit+Nbit-1 downto Nbit
        -1) , Q => mult_out_pipe(i)(k), Clock => CLK, Resetn => RST_INT_n, EN =>
        en_shift_p(0));
148             end generate;
149         end generate mult_reg;
150
151 ----primo strato di adders ------------------------------------------------------------
152 adders_1_1: process(mult_out_pipe)
```

```
153             variable temp1,temp2,temp3: integer;
154             begin
155                 for i in 0 to P-1 loop
156                     for k in 0 to 3 loop
157                         temp1:=to_integer(mult_out_pipe(i)(2*k+1));
158                         temp2:=to_integer(mult_out_pipe(i)(2*k));
159                         temp3:=temp1+temp2;
160                         sum1_reg_in(i)(k)<=to_signed(temp3,Nbit+1);
161                     end loop;
162                 end loop;
163             end process;
164 
165 -- for i in 0 to P-1 generate
166             -- sec:for k in 0 to ((N-1)/2)-1 generate
167                 -- ADD1: sum1_reg_in(i)(k)<=mult_out_pipe(i)(2*k)+mult_out_pipe(i)(2*k
    +1);
168             -- end generate;
169         -- end generate adders_1_1;
170 ----secondo strato di adders ------------------------------------------------------------
171 adders_2: process(sum1_reg_out)
172             variable temp1,temp2,temp3: integer;
173             begin
174                 for i in 0 to P-1 loop
175                     for k in 0 to 1 loop
176                         temp1:=to_integer(sum1_reg_out(i)(2*k+1));
177                         temp2:=to_integer(sum1_reg_out(i)(2*k));
178                         temp3:=temp1+temp2;
179                         sum2_reg_in(i)(k)<=to_signed(temp3,Nbit+2);
180                     end loop;
181                 end loop;
182             end process;
183 -- for i in 0 to P-1 generate
184             -- sec:for k in 0 to 1 generate
185                 -- ADD2: sum2_reg_in(i)(k)<=sum1_reg_out(i)(2*k)+sum1_reg_out(i)(2*k
    +1);
186             -- end generate;
187 -- end generate adders_2;
188 ---terzo strato di adders-----------------------------------------------------------------
189 adders_3: process(sum2_reg_out)
190             variable temp1,temp2,temp3: integer;
191             begin
192                 for i in 0 to P-1 loop
193                     temp1:=to_integer(sum2_reg_out(i)(0));
194                     temp2:=to_integer(sum2_reg_out(i)(1));
195                 temp3:=temp1+temp2;
196                 sum3_reg_in(i)<=to_signed(temp3,Nbit+3);
197 
198                 end loop;
199             end process;
200 -- for i in 0 to P-1 generate
201                 -- ADD3: sum3_reg_in(i) <= sum2_reg_out(i)(0)+sum2_reg_out(i)(1);
202         -- end generate adders_3;
203 ---quarto strato di adders
    ------------------------------------------------------------
204 
205 adders_4:process(reg_8_value,sum3_reg_out)
206             variable temp1,temp2,temp3: integer;
207             begin
208                 for i in 0 to P-1 loop
209                     temp1:=to_integer(reg_8_value(i)(3));
210                     temp2:=to_integer(sum3_reg_out(i));
211                 temp3:=temp1+temp2;
```

```
212                 sum_final(i)<=to_signed(temp3,Nbit+4);
213
214                 end loop;
215             end process;
216
217
218 -- for i in 0 to P-1 generate
219                     -- reg_8_value_not_aggregate<=reg_8_value(i)(3);
220                     -- REG8_EXTENDED<=to_signed( reg_8_value_not_aggregate,Nbit+3);
221                 -- ADD3: sum_final(i) <= sum3_reg_out(i) + REG8_EXTENDED;
222         -- end generate adders_4;
223 ------------------------------------------------------
224 ---------------reg_8 shift register-------------------
225 shift_reg8: for i in 0 to P-1 generate
226                 reg_8_value(i)(0)<=mult_out_pipe(i)(8);
227             sec:for k in 0 to 2 generate
228                 shift_reg_8: regn generic map (N => Nbit+1)
229                 port map (D => reg_8_value(i)(k), Q => reg_8_value(i)(k+1), Clock =>
    CLK, Resetn => RST_INT_n, EN => en_shift_p(k+1));
230             end generate;
231         end generate shift_reg8;
232 ------------------------------------------------------
233 registri_vari: for i in 0 to P-1 generate
234             sum1_cycle:for k in 0 to 3 generate
235                 sum1_reg: regn generic map (N => Nbit+1)
236                 port map (D => sum1_reg_in(i)(k), Q => sum1_reg_out(i)(k), Clock =>
    CLK, Resetn => RST_INT_n, EN => en_shift_p(1));
237             end generate;
238             sum2_cycle:for k in 0 to 1 generate
239                 sum2_reg: regn generic map (N => Nbit+2)
240                 port map (D => sum2_reg_in(i)(k), Q => sum2_reg_out(i)(k), Clock =>
    CLK, Resetn => RST_INT_n, EN => en_shift_p(2));
241             end generate;
242
243             sum3_reg: regn generic map (N => Nbit+3)
244                 port map (D => sum3_reg_in(i), Q => sum3_reg_out(i), Clock => CLK,
    Resetn => RST_INT_n, EN => en_shift_p(3));
245
246         end generate registri_vari;
247
248 -----PIPE OF CONTROL SIGNALS-------------------------------------------
249 pipe_registers_en_shift: for i in 0 to 3 generate
250     reg_i: dff  port map (D => en_shift_p(i), Q => en_shift_p(i+1), Clock => CLK,
    Resetn => RST_INT_n, EN => '1');
251 end generate pipe_registers_en_shift;
252
253 pipe_registers_vout: for i in 0 to 3 generate
254     reg_i: dff port map (D => vout_p(i), Q => vout_p(i+1), Clock => CLK, Resetn =>
    RST_INT_n, EN => '1');
255 end generate pipe_registers_vout;
256
257 en_shift_p(0)<=EN_SHIFT;
258 vout_p(0)<=VOUT1;
259 VOUT<=vout_p(4);
260
261
262 ---------CONTROL UNIT-----------------------------------------------------
263 state_process: PROCESS (CLK, RST_n, VIN)
264 BEGIN
265 IF (RST_n='0') THEN present_state<=RESET;
266 ELSIF (CLK'EVENT AND CLK='1') THEN
267     CASE (present_state) IS
```

```
268         -- reset
269         WHEN RESET => present_state<= IDLE;
270         WHEN IDLE => IF (VIN='1') THEN present_state <= DATA_CYCLE1;
271                      ELSE present_state <= IDLE;
272                      END IF;
273         WHEN DATA_CYCLE1 => IF (VIN='0') THEN present_state <= LAST_DATA1;
274                      ELSE present_state<=DATA_CYCLE2;
275                      END IF;
276         WHEN DATA_CYCLE2 => IF (VIN='0') THEN present_state <= LAST_DATA1;
277                      ELSE present_state<=DATA_CYCLE2;
278                      END IF;
279         WHEN LAST_DATA1 =>  present_state <= IDLE;
280
281       END CASE;
282 END IF;
283 END PROCESS state_process;
284
285 output_process: PROCESS (present_state)
286 BEGIN
287 VOUT1<='0';
288 EN_REG_1<='0';
289 EN_REG_OUT<='0';
290 EN_SHIFT<='0';
291 RST_INT_n<='1';
292     CASE (present_state) IS
293         -- reset
294         WHEN RESET => RST_INT_n<='0';
295         WHEN IDLE => EN_REG_1<='1';
296         WHEN DATA_CYCLE1 => EN_REG_OUT<='1';
297                             EN_SHIFT<='1';
298         WHEN DATA_CYCLE2 => EN_REG_OUT<='1';
299                             EN_SHIFT<='1';
300                             VOUT1<='1';
301         WHEN LAST_DATA1 =>  VOUT1<='1';
302       END CASE;
303 END PROCESS output_process;
304
305 END ARCHITECTURE behavior;
```

A.0.5 Clock generator

./Code/clk_gen.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity clk_gen is
7    port (
8      END_SIM : in  std_logic;
9      CLK     : out std_logic;
10     RST_n   : out std_logic);
11 end clk_gen;
12
13 architecture beh of clk_gen is
14
15   constant Ts : time := 10 ns;
16
17   signal CLK_i : std_logic;
```

```
18
19 begin  -- beh
20
21   process
22   begin  -- process
23     if (CLK_i = 'U') then
24       CLK_i <= '0';
25     else
26       CLK_i <= not(CLK_i);
27     end if;
28     wait for Ts/2;
29   end process;
30
31   CLK <= CLK_i and not( END_SIM );
32
33   process
34   begin  -- process
35     RST_n <= '0';
36     wait for 2*Ts/2;
37     RST_n <= '1';
38     wait;
39   end process;
40
41 end beh;
```

A.0.6 Data sink

./Code/data_sink.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  use ieee.std_logic_textio.all;
6
7  library std;
8  use std.textio.all;
9
10 library work;
11 use work.util.all;
12
13 entity data_sink is
14   port (
15     CLK    : in std_logic;
16     RST_n  : in std_logic;
17     VIN    : in std_logic;
18     DIN    : in std_logic_vector(P * Nbit - 1 downto 0));
19 end data_sink;
20
21 architecture beh of data_sink is
22
23 begin  -- beh
24
25   process (CLK, RST_n)
26     file res_fp : text open WRITE_MODE is "./results.txt";
27     variable line_out : line;
28   begin  -- process
29     if RST_n = '0' then                 -- asynchronous reset (active low)
30       null;
31     -- it works on falling edges and divides the three outputs
```

```
32      elsif CLK'event and CLK = '0' then  -- falling clock edge
33        if (VIN = '1') then
34          write(line_out, conv_integer(signed(DIN(3 * Nbit - 1 downto 2 * Nbit))));
35          writeline(res_fp, line_out);
36          write(line_out, conv_integer(signed(DIN(2 * Nbit - 1 downto Nbit))));
37          writeline(res_fp, line_out);
38          write(line_out, conv_integer(signed(DIN(Nbit - 1 downto 0))));
39          writeline(res_fp, line_out);
40        end if;
41      end if;
42    end process;
43
44  end beh;
```

A.0.7 Signal Generator

./Code/signal_generator.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4  USE STD.Textio.all;
5
6  library std;
7  USE work.util.all;
8
9  ENTITY signal_generator IS
10     PORT(    VIN:     out std_logic;
11              DIN:     out std_logic_vector(P*Nbit-1 downto 0);
12              RST_n : in std_logic;
13              CLK : in std_logic;
14              END_SIM : out std_logic;
15              b:       out std_logic_vector(Nbit * N - 1 downto 0));
16 END ENTITY signal_generator;
17
18 ARCHITECTURE behavior OF signal_generator is
19
20 SUBTYPE word IS STD_LOGIC_VECTOR (Nbit-1 DOWNTO 0);
21 -- Array in cui salvare tutti i dati del file DATI_AB
22 TYPE word_array IS ARRAY (1 TO 1000) OF word;
23 SIGNAL input_data: word_array;
24
25 signal global_line_count : integer := 0;
26
27 BEGIN
28
29 data_reading_process: PROCESS
30     FILE in_file: TEXT open READ_MODE is "./samples_bin.txt";
31     VARIABLE buf: LINE;
32     VARIABLE d_v: CHARACTER;
33     variable line_count : integer := 1;
34     --VARIABLE i: INTEGER:=1;
35     BEGIN
36     -- we first read all the file containing all the samples
37     while not endfile(in_file) loop
38         readline(in_file,buf);
39         for h in Nbit-1 downto 0 loop
40          read(buf,d_v);
41          -- we convert each line bit by bit
42          IF d_v='1' THEN
```

```
43              input_data(line_count)(h)<='1';
44          ELSE
45              input_data(line_count)(h)<='0';
46          END IF;
47         end loop;
48         -- we increase the count of the lines
49         line_count := line_count + 1;
50     END LOOP;
51     file_close(in_file);
52     -- to communicate the total line count to the other process
53     global_line_count <= line_count;
54     wait;
55 END PROCESS data_reading_process;
56 
57 data_generation_process: PROCESS(CLK, RST_n)
58     variable j : integer := 3;
59     variable delay : integer := 0;
60     variable end_int : boolean := false;
61     variable w : boolean := false;
62     variable wait_reset : boolean := false;
63 BEGIN
64 -- we avoid working during reset
65 if (RST_n = '0') then
66     DIN<=(OTHERS => '0');
67     VIN<='0';
68     END_SIM <= '0';
69 else
70     -- we work on falling edges to avoid simulation errors
71     if (falling_edge(CLK)) then
72         -- we wait 1 cycle after issuing the reset to ensure the correct
73         -- resetting of the whole machine
74         if (wait_reset = true) then
75             -- standard cycle, 3 by 3
76             if (j < global_line_count and end_int = false) then
77                 -- this if is needed to stop the execution for some time
78                 -- to check the correct behaviour while VIN = 0
79                 if (w = false and (j < 100 and j > 90)) then
80                     VIN <= '0';
81                     j := j + 1;
82                     if (j = 99) then
83                         w := true;
84                         j := 93;
85                     end if;
86                 else
87                     -- input data at each cycle
88                     VIN<='1';
89                     DIN<=input_data(j-2) & input_data(j-1) & input_data(j);
90                     END_SIM <= '0';
91                     j := j + 3;
92                 end if;
93             -- check for the end of the simulation after passing all inputs
94             elsif (end_int = false) then
95                 end_int := true;
96                 VIN <= '0';
97             -- we wait for some time before stopping the simulation
98             elsif (end_int = true and delay < FINAL_DELAY) then
99                 delay := delay + 1;
100            -- stop the simulation
101            elsif (end_int = true and delay >= FINAL_DELAY) then
102                END_SIM <= '1';
103            end if;
104        else
```

```
105             wait_reset := true;
106         end if;
107     end if;
108 end if;
109 END PROCESS data_generation_process;
110 
111 b <= "1111111100" & "1111111001" & "0000011010" & "0010001000" & "0011001111" & "
    0010001000" & "0000011010" & "1111111001" & "1111111100";
112 
113 END ARCHITECTURE behavior;
```

A.0.8 Testbench

./Code/tb_fir.v

```
1  //`timescale 1ns
2  
3  
4  module tb_fir ();
5  
6      wire CLK_i;
7      wire RST_n_i;
8      wire [29:0] DIN_i;
9      wire VIN_i;
10     wire [89:0] b_i;
11     wire [29:0] DOUT_i;
12     wire VOUT_i;
13     wire END_SIM_i;
14 
15     clk_gen CG(.END_SIM(END_SIM_i),
16             .CLK(CLK_i),
17             .RST_n(RST_n_i));
18 
19     signal_generator SM(.CLK(CLK_i),
20                 .RST_n(RST_n_i),
21             .VIN(VIN_i),
22             .DIN(DIN_i),
23             .END_SIM(END_SIM_i),
24             .b(b_i));
25 
26     // FIR_filter_dadda UUT(.CLK(CLK_i),
27      //         .RST_n(RST_n_i),
28      //         .DIN(DIN_i),
29     //          .VIN(VIN_i),
30      //         .b(b_i),
31     //          .DOUT(DOUT_i),
32     //          .VOUT(VOUT_i));
33 
34     // FIR_filter_no_dadda UUT(.CLK(CLK_i),
35      //         .RST_n(RST_n_i),
36      //         .DIN(DIN_i),
37     //          .VIN(VIN_i),
38      //         .b(b_i),
39     //          .DOUT(DOUT_i),
40     //          .VOUT(VOUT_i));
41 
42     // FIR_filter_dadda_standard UUT(.CLK(CLK_i),
43      //         .RST_n(RST_n_i),
44      //         .DIN(DIN_i),
45     //          .VIN(VIN_i),
```

```
46      //        .b(b_i),
47      //        .DOUT(DOUT_i),
48      //        .VOUT(VOUT_i));
49
50      // FIR_filter_dadda_standard_no6LSB UUT(.CLK(CLK_i),
51      //        .RST_n(RST_n_i),
52      //        .DIN(DIN_i),
53      //        .VIN(VIN_i),
54      //        .b(b_i),
55      //        .DOUT(DOUT_i),
56      //        .VOUT(VOUT_i));
57
58      // FIR_filter_dadda_no6LSB UUT(.CLK(CLK_i),
59      //        .RST_n(RST_n_i),
60      //        .DIN(DIN_i),
61      //        .VIN(VIN_i),
62      //        .b(b_i),
63      //        .DOUT(DOUT_i),
64      //        .VOUT(VOUT_i));
65
66      // FIR_filter_dadda_approx_layer2 UUT(.CLK(CLK_i),
67      //        .RST_n(RST_n_i),
68      //        .DIN(DIN_i),
69      //        .VIN(VIN_i),
70      //        .b(b_i),
71      //        .DOUT(DOUT_i),
72      //        .VOUT(VOUT_i));
73
74      // FIR_filter_ambe_approx_layer2 UUT(.CLK(CLK_i),
75      //        .RST_n(RST_n_i),
76      //        .DIN(DIN_i),
77      //        .VIN(VIN_i),
78      //        .b(b_i),
79      //        .DOUT(DOUT_i),
80      //        .VOUT(VOUT_i));
81
82      // FIR_filter_ambe UUT(.CLK(CLK_i),
83      //        .RST_n(RST_n_i),
84      //        .DIN(DIN_i),
85      //        .VIN(VIN_i),
86      //        .b(b_i),
87      //        .DOUT(DOUT_i),
88      //        .VOUT(VOUT_i));
89
90      FIR_filter_dadda_final_approx UUT(.CLK(CLK_i),
91         .RST_n(RST_n_i),
92         .DIN(DIN_i),
93         .VIN(VIN_i),
94         .b(b_i),
95         .DOUT(DOUT_i),
96         .VOUT(VOUT_i));
97
98      data_sink DS(.CLK(CLK_i),
99           .RST_n(RST_n_i),
100          .VIN(VOUT_i),
101          .DIN(DOUT_i));
102
103     initial begin
104     $read_lib_saif("../saif/NangateOpenCellLibrary.saif");
105     $set_gate_level_monitoring("on");
106     $set_toggle_region(UUT);
107     $toggle_start;
```

```
108     end
109
110     always @ ( END_SIM_i ) begin
111     if (END_SIM_i) begin
112     $toggle_stop;
113     $toggle_report("../saif/FIR_filter_switching.saif", 1.0e-9, "tb_fir.UUT");
114     end
115     end
116
117 endmodule
```

APPENDIX B

Manually-optimized FIR filter

This is the code of the manually-optimized filter, using Dadda trees and various optimizations, as indicated in each section.

B.1 Standard Dadda tree

This section contains the code for the standard Dadda tree, using Full Adders and Half Adders to cover all the bits. The filter code and the multiplier **will not** be repeated for the other configurations, since they are practically the same except for the name of the components.

B.1.1 FIR filter code

./Code/FIR_filter_dadda_standard.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  library std;
6  USE work.util.all;
7
8  ENTITY FIR_filter_dadda_standard IS
9
10 PORT(    CLK:     in std_logic;
11         RST_n:   in std_logic;
12
13         VIN:     in std_logic;
14         VOUT:    out std_logic;
15
16         -- we have Nbit * P types for Verilog, which does not support custom types
17         DIN:     in std_logic_vector(Nbit * P - 1 downto 0);
18         DOUT:    out std_logic_vector(Nbit * P -1 downto 0);
19
20         b:       in std_logic_vector(N * Nbit - 1 downto 0));
21
22 END ENTITY FIR_filter_dadda_standard;
23
24 ARCHITECTURE behavior OF FIR_filter_dadda_standard IS
25
```

```
COMPONENT regn IS
GENERIC (N: INTEGER := 16);                          -- numero di bit del registro
PORT (D : IN SIGNED (N-1 DOWNTO 0);      -- ingresso
     Clock, Resetn, EN : IN STD_LOGIC;               -- clock, reset, enable
     Q : OUT SIGNED (N-1 DOWNTO 0));     -- uscita
END COMPONENT regn;

component dff IS
PORT (D : IN STD_LOGIC;      -- ingresso
     Clock, Resetn, EN : IN STD_LOGIC;               -- clock, reset, enable
     Q : OUT STD_LOGIC);      -- uscita
END component;
component mul_dadda_standard is
     port (a, b : in signed(Nbit -1  downto 0);
           product : out signed(2 * Nbit - 1 downto 0));
end component;

SIGNAL xz: OUT_PIPES_TYPE;
SIGNAL mult: mult_array_TYPE;
signal mult_out_pipe: mult_resize_array_TYPE;

SIGNAL mult_resize: LIST_mult_resize;
SIGNAL sum_1_in,sum_2_in,sum_3in,sum_1_out,sum_2_out,sum_3out: LIST_sum_1;
SIGNAL PIPE_REG_MULT_2_8:SIGNED(Nbit+1-1 downto 0);

type sum1_reg_type is array (0 to 4-1) of SIGNED((Nbit+1)-1 downto 0);
type sum1_type is array (0 to P-1) of sum1_reg_type;
SIGNAL sum1_reg_in,sum1_reg_out : sum1_type;

type sum2_reg_type is array (0 to 2-1) of SIGNED((Nbit+2)-1 downto 0);
type sum2_type is array (0 to P-1) of sum2_reg_type;
SIGNAL sum2_reg_in,sum2_reg_out : sum2_type;

type sum3_type is array (0 to P-1) of SIGNED((Nbit+3)-1 downto 0);
SIGNAL sum3_reg_in,sum3_reg_out : sum3_type;
SIGNAL REG8_EXTENDED : SIGNED(Nbit+3-1 DOWNTO 0);

type sum_final_type is array (0 to P-1) of SIGNED((Nbit+4)-1 downto 0);
SIGNAL sum_final : sum_final_type;

type reg_8_reg_type is array (0 to 4-1) of SIGNED((Nbit+1)-1 downto 0);
type reg_8_type is array (0 to P-1) of  reg_8_reg_type;
SIGNAL reg_8_value : reg_8_type;
signal reg_8_value_not_aggregate: signed(Nbit-1 downto 0);

type pipelined_type is array (0 to 4) of std_logic;
SIGNAL en_shift_p,vout_p : pipelined_type;

SIGNAL VIN_retard : std_logic;

TYPE state IS (RESET,IDLE,DATA_CYCLE1,DATA_CYCLE2,LAST_DATA1);
SIGNAL present_state : state;
SIGNAL EN_REG_1,EN_REG_OUT,EN_SHIFT,RST_INT_n,EN_FIRST_REG,VOUT1 : STD_LOGIC;

SIGNAL DOUT1,DOUT2,DOUT3 :SIGNED(Nbit-1 DOWNTO 0 );
BEGIN
---- DATAPATH------------------------------------------------------
EN_FIRST_REG<=(EN_REG_1 or EN_SHIFT);
in_reg_3k: regn      generic map (N => Nbit)
                port map (D => signed(DIN(3 * Nbit - 1 downto 2 * Nbit)), Clock => CLK
    , Resetn => RST_INT_n, EN => EN_FIRST_REG  , Q => xz(0)(0));
in_reg_3k_plus_1: regn   generic map (N => Nbit)
```

```
87                  port map (D => signed(DIN(2 * Nbit - 1 downto Nbit)), Clock => CLK,
    Resetn => RST_INT_n, EN => EN_FIRST_REG , Q => xz(1)(0));
88 in_reg_3k_plus_2: regn  generic map (N => Nbit)
89                  port map (D => signed(DIN(Nbit -1 downto 0)), Clock => CLK, Resetn =>
    RST_INT_n, EN => EN_FIRST_REG , Q => xz(2)(0));
90 ----------------out register          ------------------------------------------------
91 out_reg_1: regn generic map (N => Nbit)
92                  port map (D => sum_final(0)((Nbit+1)-1 downto (Nbit+1)-1-Nbit+1),
    Clock => CLK, Resetn => RST_INT_n, EN => en_shift_p(4), Q => DOUT1(Nbit -1 downto
    0));
93 out_reg_2: regn generic map (N => Nbit)
94                  port map (D => sum_final(1)((Nbit+1)-1 downto (Nbit+1)-1-Nbit+1),
    Clock => CLK, Resetn => RST_INT_n, EN => en_shift_p(4), Q => DOUT2(Nbit - 1 downto
     0));
95 out_reg_3: regn generic map (N => Nbit)
96                  port map (D => sum_final(2)((Nbit+1)-1 downto (Nbit+1)-1-Nbit+1),
    Clock => CLK, Resetn => RST_INT_n, EN => en_shift_p(4), Q => DOUT3(Nbit - 1 downto
     0));
97 DOUT( Nbit - 1  downto 0)       <=std_logic_vector(DOUT3);
98 DOUT(2 * Nbit-1 downto Nbit)    <=std_logic_vector(DOUT2);
99 DOUT(3 * Nbit-1 downto 2*Nbit)  <=std_logic_vector(DOUT1);
100
101 ----------------end modify-------------------------------------
102 shift_reg_3k: for i in 0 to W1-1 generate
103     reg_i: regn generic map (N => Nbit)
104                  port map (D => xz(0)(i), Q => xz(0)(i+1), Clock => CLK, Resetn =>
    RST_INT_n, EN => EN_SHIFT);
105 end generate shift_reg_3k;
106
107 shift_reg_3k_plus_1: for i in 0 to W2-1 generate
108     reg_i: regn generic map (N => Nbit)
109                  port map (D => xz(1)(i), Q => xz(1)(i+1), Clock => CLK, Resetn =>
    RST_INT_n, EN => EN_SHIFT);
110 end generate shift_reg_3k_plus_1;
111
112 shift_reg_3k_plus_2: for i in 0 to W3-1 generate
113     reg_i: regn generic map (N => Nbit)
114                  port map (D => xz(2)(i), Q => xz(2)(i+1), Clock => CLK, Resetn =>
    RST_INT_n, EN => EN_SHIFT);
115 end generate shift_reg_3k_plus_2;
116 -----
117 mult_0_0 : mul_dadda_standard
118                  port map (a => signed(b((0 + 1) * Nbit - 1 downto 0 * Nbit)),
119                            b => xz(0)(0),
120                            product => mult(0)(0));
121 mult_0_1 : mul_dadda_standard
122                  port map (a => signed(b((1 + 1) * Nbit - 1 downto 1 * Nbit)),
123                            b => xz(2)(1),
124                            product => mult(0)(1));
125 mult_0_2 : mul_dadda_standard
126                  port map (a => signed(b((2 + 1) * Nbit - 1 downto 2 * Nbit)),
127                            b => xz(1)(1),
128                            product => mult(0)(2));
129 mult_0_3 : mul_dadda_standard
130                  port map (a => signed(b((3 + 1) * Nbit - 1 downto 3 * Nbit)),
131                            b => xz(0)(1),
132                            product => mult(0)(3));
133 mult_0_4 : mul_dadda_standard
134                  port map (a => signed(b((4 + 1) * Nbit - 1 downto 4 * Nbit)),
135                            b => xz(2)(2),
136                            product => mult(0)(4));
137 mult_0_5 : mul_dadda_standard
```

```
138                 port map (a => signed(b((5 + 1) * Nbit - 1 downto 5 * Nbit)),
139                           b => xz(1)(2),
140                           product => mult(0)(5));
141 mult_0_6 : mul_dadda_standard
142                 port map (a => signed(b((6 + 1) * Nbit - 1 downto 6 * Nbit)),
143                           b => xz(0)(2),
144                           product => mult(0)(6));
145 mult_0_7 : mul_dadda_standard
146                 port map (a => signed(b((7 + 1) * Nbit - 1 downto 7 * Nbit)),
147                           b => xz(2)(3),
148                           product => mult(0)(7));
149 mult_0_8 : mul_dadda_standard
150                 port map (a => signed(b((8 + 1) * Nbit - 1 downto 8 * Nbit)),
151                           b => xz(1)(3),
152                           product => mult(0)(8));
153
154 mult_1_0 : mul_dadda_standard
155                 port map (a => signed(b((0 + 1) * Nbit - 1 downto 0 * Nbit)),
156                           b => xz(1)(0),
157                           product => mult(1)(0));
158 mult_1_1 : mul_dadda_standard
159                 port map (a => signed(b((1 + 1) * Nbit - 1 downto 1 * Nbit)),
160                           b => xz(0)(0),
161                           product => mult(1)(1));
162 mult_1_2 : mul_dadda_standard
163                 port map (a => signed(b((2 + 1) * Nbit - 1 downto 2 * Nbit)),
164                           b => xz(2)(1),
165                           product => mult(1)(2));
166 mult_1_3 : mul_dadda_standard
167                 port map (a => signed(b((3 + 1) * Nbit - 1 downto 3 * Nbit)),
168                           b => xz(1)(1),
169                           product => mult(1)(3));
170 mult_1_4 : mul_dadda_standard
171                 port map (a => signed(b((4 + 1) * Nbit - 1 downto 4 * Nbit)),
172                           b => xz(0)(1),
173                           product => mult(1)(4));
174 mult_1_5 : mul_dadda_standard
175                 port map (a => signed(b((5 + 1) * Nbit - 1 downto 5 * Nbit)),
176                           b => xz(2)(2),
177                           product => mult(1)(5));
178 mult_1_6 : mul_dadda_standard
179                 port map (a => signed(b((6 + 1) * Nbit - 1 downto 6 * Nbit)),
180                           b => xz(1)(2),
181                           product => mult(1)(6));
182 mult_1_7 : mul_dadda_standard
183                 port map (a => signed(b((7 + 1) * Nbit - 1 downto 7 * Nbit)),
184                           b => xz(0)(2),
185                           product => mult(1)(7));
186 mult_1_8 : mul_dadda_standard
187                 port map (a => signed(b((8 + 1) * Nbit - 1 downto 8 * Nbit)),
188                           b => xz(2)(3),
189                           product => mult(1)(8));
190
191
192
193 mult_2_0 : mul_dadda_standard
194                 port map (a => signed(b((0 + 1) * Nbit - 1 downto 0 * Nbit)),
195                           b => xz(2)(0),
196                           product => mult(2)(0));
197 mult_2_1 : mul_dadda_standard
198                 port map (a => signed(b((1 + 1) * Nbit - 1 downto 1 * Nbit)),
199                           b => xz(1)(0),
```

```
                        product => mult(2)(1));
mult_2_2 : mul_dadda_standard
                port map (a => signed(b((2 + 1) * Nbit - 1 downto 2 * Nbit)),
                        b => xz(0)(0),
                        product => mult(2)(2));
mult_2_3 : mul_dadda_standard
                port map (a => signed(b((3 + 1) * Nbit - 1 downto 3 * Nbit)),
                        b => xz(2)(1),
                        product => mult(2)(3));
mult_2_4 : mul_dadda_standard
                port map (a => signed(b((4 + 1) * Nbit - 1 downto 4 * Nbit)),
                        b => xz(1)(1),
                        product => mult(2)(4));
mult_2_5 : mul_dadda_standard
                port map (a => signed(b((5 + 1) * Nbit - 1 downto 5 * Nbit)),
                        b => xz(0)(1),
                        product => mult(2)(5));
mult_2_6 : mul_dadda_standard
                port map (a => signed(b((6 + 1) * Nbit - 1 downto 6 * Nbit)),
                        b => xz(2)(2),
                        product => mult(2)(6));
mult_2_7 : mul_dadda_standard
                port map (a => signed(b((7 + 1) * Nbit - 1 downto 7 * Nbit)),
                        b => xz(1)(2),
                        product => mult(2)(7));
mult_2_8 : mul_dadda_standard
                port map (a => signed(b((8 + 1) * Nbit - 1 downto 8 * Nbit)),
                        b => xz(0)(2),
                        product => mult(2)(8));
-----

mult_reg: for i in 0 to P-1 generate
            sec:for k in 0 to N-1 generate
                mult_reg_i: regn generic map (N => Nbit+1)
                                        port map (D => mult(i)(k)(Nbit+Nbit-1 downto Nbit
    -1) , Q => mult_out_pipe(i)(k), Clock => CLK, Resetn => RST_INT_n, EN =>
    en_shift_p(0));
            end generate;
        end generate mult_reg;

----primo strato di adders ------------------------------------------------------
adders_1_1: process(mult_out_pipe)
            variable temp1,temp2,temp3: integer;
            begin
                for i in 0 to P-1 loop
                    for k in 0 to 3 loop
                        temp1:=to_integer(mult_out_pipe(i)(2*k+1));
                        temp2:=to_integer(mult_out_pipe(i)(2*k));
                        temp3:=temp1+temp2;
                        sum1_reg_in(i)(k)<=to_signed(temp3,Nbit+1);
                    end loop;
                end loop;
            end process;

-- for i in 0 to P-1 generate
            -- sec:for k in 0 to ((N-1)/2)-1 generate
                -- ADD1: sum1_reg_in(i)(k)<=mult_out_pipe(i)(2*k)+mult_out_pipe(i)(2*k
    +1);
            -- end generate;
        -- end generate adders_1_1;
----secondo strato di adders -----------------------------------------------------
adders_2: process(sum1_reg_out)
```

```
259             variable temp1,temp2,temp3: integer;
260             begin
261                 for i in 0 to P-1 loop
262                     for k in 0 to 1 loop
263                         temp1:=to_integer(sum1_reg_out(i)(2*k+1));
264                         temp2:=to_integer(sum1_reg_out(i)(2*k));
265                         temp3:=temp1+temp2;
266                         sum2_reg_in(i)(k)<=to_signed(temp3,Nbit+2);
267                     end loop;
268                 end loop;
269             end process;
270 -- for i in 0 to P-1 generate
271             -- sec:for k in 0 to 1 generate
272                 -- ADD2: sum2_reg_in(i)(k)<=sum1_reg_out(i)(2*k)+sum1_reg_out(i)(2*k
    +1);
273             -- end generate;
274 -- end generate adders_2;
275 ---terzo strato di adders-----------------------------------------------------
276 adders_3: process(sum2_reg_out)
277             variable temp1,temp2,temp3: integer;
278             begin
279                 for i in 0 to P-1 loop
280                     temp1:=to_integer(sum2_reg_out(i)(0));
281                     temp2:=to_integer(sum2_reg_out(i)(1));
282                 temp3:=temp1+temp2;
283                 sum3_reg_in(i)<=to_signed(temp3,Nbit+3);
284 
285                 end loop;
286             end process;
287 -- for i in 0 to P-1 generate
288                 -- ADD3: sum3_reg_in(i) <= sum2_reg_out(i)(0)+sum2_reg_out(i)(1);
289         -- end generate adders_3;
290 ---quarto strato di adders
    ------------------------------------------------------------
291 
292 adders_4:process(reg_8_value,sum3_reg_out)
293             variable temp1,temp2,temp3: integer;
294             begin
295                 for i in 0 to P-1 loop
296                     temp1:=to_integer(reg_8_value(i)(3));
297                     temp2:=to_integer(sum3_reg_out(i));
298                 temp3:=temp1+temp2;
299                 sum_final(i)<=to_signed(temp3,Nbit+4);
300 
301                 end loop;
302             end process;
303 
304 
305 -- for i in 0 to P-1 generate
306                     -- reg_8_value_not_aggregate<=reg_8_value(i)(3);
307                     -- REG8_EXTENDED<=to_signed( reg_8_value_not_aggregate,Nbit+3);
308                 -- ADD3: sum_final(i) <= sum3_reg_out(i) + REG8_EXTENDED;
309         -- end generate adders_4;
310 ------------------------------------------------------------
311 ------------reg_8 shift register-----------------
312 shift_reg8: for i in 0 to P-1 generate
313                 reg_8_value(i)(0)<=mult_out_pipe(i)(8);
314             sec:for k in 0 to 2 generate
315                 shift_reg_8: regn generic map (N => Nbit+1)
316                 port map (D => reg_8_value(i)(k), Q => reg_8_value(i)(k+1), Clock =>
    CLK, Resetn => RST_INT_n, EN => en_shift_p(k+1));
317             end generate;
```

```
318         end generate shift_reg8;
319 ------------------------------------------------------------
320 registri_vari: for i in 0 to P-1 generate
321             sum1_cycle:for k in 0 to 3 generate
322                 sum1_reg: regn generic map (N => Nbit+1)
323                 port map (D => sum1_reg_in(i)(k), Q => sum1_reg_out(i)(k), Clock =>
    CLK, Resetn => RST_INT_n, EN => en_shift_p(1));
324             end generate;
325             sum2_cycle:for k in 0 to 1 generate
326                 sum2_reg: regn generic map (N => Nbit+2)
327                 port map (D => sum2_reg_in(i)(k), Q => sum2_reg_out(i)(k), Clock =>
    CLK, Resetn => RST_INT_n, EN => en_shift_p(2));
328             end generate;
329
330             sum3_reg: regn generic map (N => Nbit+3)
331                 port map (D => sum3_reg_in(i), Q => sum3_reg_out(i), Clock => CLK,
    Resetn => RST_INT_n, EN => en_shift_p(3));
332
333         end generate registri_vari;
334
335 ------PIPE OF CONTROL SIGNALS-------------------------------------------
336 pipe_registers_en_shift: for i in 0 to 3 generate
337     reg_i: dff  port map (D => en_shift_p(i), Q => en_shift_p(i+1), Clock => CLK,
    Resetn => RST_INT_n, EN => '1');
338 end generate pipe_registers_en_shift;
339
340 pipe_registers_vout: for i in 0 to 3 generate
341     reg_i: dff port map (D => vout_p(i), Q => vout_p(i+1), Clock => CLK, Resetn =>
    RST_INT_n, EN => '1');
342 end generate pipe_registers_vout;
343
344 en_shift_p(0)<=EN_SHIFT;
345 vout_p(0)<=VOUT1;
346 VOUT<=vout_p(4);
347
348
349 ---------CONTROL UNIT-----------------------------------------------------
350 state_process: PROCESS (CLK, RST_n, VIN)
351 BEGIN
352 IF (RST_n='0') THEN present_state<=RESET;
353 ELSIF (CLK'EVENT AND CLK='1') THEN
354     CASE (present_state) IS
355         -- reset
356         WHEN RESET => present_state<= IDLE;
357         WHEN IDLE => IF (VIN='1') THEN present_state <= DATA_CYCLE1;
358                      ELSE present_state <= IDLE;
359                      END IF;
360         WHEN DATA_CYCLE1 => IF (VIN='0') THEN present_state <= LAST_DATA1;
361                      ELSE present_state<=DATA_CYCLE2;
362                      END IF;
363         WHEN DATA_CYCLE2 => IF (VIN='0') THEN present_state <= LAST_DATA1;
364                      ELSE present_state<=DATA_CYCLE2;
365                      END IF;
366         WHEN LAST_DATA1 =>  present_state <= IDLE;
367
368       END CASE;
369 END IF;
370 END PROCESS state_process;
371
372 output_process: PROCESS (present_state)
373 BEGIN
374 VOUT1<='0';
```

```
375 EN_REG_1<='0';
376 EN_REG_OUT<='0';
377 EN_SHIFT<='0';
378 RST_INT_n<='1';
379     CASE (present_state) IS
380         -- reset
381         WHEN RESET => RST_INT_n<='0';
382         WHEN IDLE => EN_REG_1<='1';
383         WHEN DATA_CYCLE1 => EN_REG_OUT<='1';
384                             EN_SHIFT<='1';
385         WHEN DATA_CYCLE2 => EN_REG_OUT<='1';
386                             EN_SHIFT<='1';
387                             VOUT1<='1';
388         WHEN LAST_DATA1 =>  VOUT1<='1';
389       END CASE;
390 END PROCESS output_process;
391
392 END ARCHITECTURE behavior;
```

B.1.2 Multiplier

./Code/mul_dadda_standard.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  library work;
6  use work.util.all;
7
8  entity mul_dadda_standard is
9      port (a, b : in signed(Nbit -1  downto 0);
10           product : out signed(2 * Nbit - 1 downto 0));
11 end mul_dadda_standard;
12
13 architecture behav of mul_dadda_standard is
14     component  dadda_standard
15         port (partial : in partial_array;
16             out1, out2 : out signed(2 * Nbit - 1 downto 0));
17     end component;
18
19     component mbe
20         port (a, b : in signed(Nbit - 1 downto 0);
21             partial : out partial_array);
22     end component;
23
24     signal partial : partial_array;
25     signal out1, out2 : signed(2 * Nbit - 1 downto 0);
26
27     begin
28         part_prod_gen : mbe port map (a => a, b => b, partial => partial);
29         part_prod_red : dadda_standard port map (partial => partial, out1 => out1,
       out2 => out2);
30         product <= out1 + out2;
31     end behav;
```

B.1.3 Partial Product Generation - MBE

./Code/mbe.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity mbe is
    port (a, b : in signed(Nbit - 1 downto 0);
          partial : out partial_array);
end mbe;

architecture behav of mbe is
    signal q, p_temp : multiple_factoring_array;
    signal extended_b : signed(Nbit downto 0);
    begin
        -- zero right extension for index -1
        extended_b <= b & '0';

        -- q generation from MBE
        multiple_factoring : for i in 0 to Nbit / 2 - 1 generate
            q(i) <= a(Nbit -1) & a when (extended_b(2 * i + 1) xor extended_b(2 * i))
    = '1' else
                    a & '0' when ((not (extended_b(2 * i + 1) xor extended_b(2 * i)))
    and (extended_b(2 * i + 2) xor extended_b(2 * i + 1))) = '1' else
                    (others => '0');
        end generate;

        partial_product : for i in 0 to Nbit / 2 - 1 generate
            -- xor 1 bit for MBE
            p_temp_gen : for j in 0 to Nbit generate
                p_temp(i)(j) <= q(i)(j) xor b(2 * i + 1);
            end generate;
            -- partial product without sign for Roorda
            partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto 0);
            -- Roorda extension
            partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
            -- Zero padding on the right
            if_padding : if i > 0 generate
                partial(i)(2 * i - 1 downto 0) <= (others => '0');
            end generate;
            -- MBE carry
            --p(N / 2)(2 * i) <= b(2 * i + 1);
            -- Roorda carry
            --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
            -- these last two could be compacted in one, MBE carry should be
            -- max in position N - 1, while Roorda carry starts from position
            -- N
            -- MBE + Roorda carry in same line
            partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
            partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
            partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
            partial(Nbit / 2)(2 * i + 1) <= '0';
        end generate;
    end behav;
```

B.1.4 Partial Product Reduction - Dadda tree

./Code/dadda_standard.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity dadda_standard is
    port (partial : in partial_array;
          out1, out2 : out signed(2 * Nbit - 1 downto 0));
end dadda_standard;

architecture behav of dadda_standard is
    component fa
        port (cin, a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component ha
        port (a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
    signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
    signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 3 downto 0);
    signal partial_temp_layer3 : internal_partial_array(Nbit / 2 - 4 downto 0);

    begin
        partial_association : for i in 0 to Nbit / 2 generate
            partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
            partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
        end generate;

        -- first stage --> from 6 to 4
        partial_temp_layer1(0)(0) <= partial_temp(0)(0);
        partial_temp_layer1(1)(0) <= partial_temp(5)(0);

        partial_temp_layer1(0)(1) <= partial_temp(0)(1);

        partial_temp_layer1(0)(2) <= partial_temp(0)(2);
        partial_temp_layer1(1)(2) <= partial_temp(1)(2);
        partial_temp_layer1(2)(2) <= partial_temp(5)(2);

        partial_temp_layer1(0)(3) <= partial_temp(0)(3);
        partial_temp_layer1(1)(3) <= partial_temp(1)(3);

        partial_temp_layer1(0)(4) <= partial_temp(0)(4);
        partial_temp_layer1(1)(4) <= partial_temp(1)(4);
        partial_temp_layer1(2)(4) <= partial_temp(2)(4);
        partial_temp_layer1(3)(4) <= partial_temp(5)(4);

        partial_temp_layer1(0)(5) <= partial_temp(0)(5);
        partial_temp_layer1(1)(5) <= partial_temp(1)(5);
        partial_temp_layer1(2)(5) <= partial_temp(2)(5);

        ha1_1 : ha port map (a => partial_temp(0)(6),
                             b => partial_temp(1)(6),
                             cout => partial_temp_layer1(0)(7),
                             s => partial_temp_layer1(0)(6));
        partial_temp_layer1(1)(6) <= partial_temp(2)(6);
```

```
62          partial_temp_layer1(2)(6) <= partial_temp(3)(6);
63          partial_temp_layer1(3)(6) <= partial_temp(5)(6);
64
65          ha1_2 : ha port map (a => partial_temp(0)(7),
66                               b => partial_temp(1)(7),
67                               cout => partial_temp_layer1(0)(8),
68                               s => partial_temp_layer1(1)(7));
69          partial_temp_layer1(2)(7) <= partial_temp(2)(7);
70          partial_temp_layer1(3)(7) <= partial_temp(3)(7);
71
72          fa1_1 : fa port map (a => partial_temp(0)(8),
73                               b => partial_temp(1)(8),
74                               cin => partial_temp(2)(8),
75                               cout => partial_temp_layer1(0)(9),
76                               s => partial_temp_layer1(1)(8));
77          ha1_3 : ha port map (a => partial_temp(3)(8),
78                               b => partial_temp(4)(8),
79                               cout => partial_temp_layer1(1)(9),
80                               s => partial_temp_layer1(2)(8));
81          partial_temp_layer1(3)(8) <= partial_temp(5)(8);
82
83
84          fa_ha : for i in 9 to 2 * Nbit - 1 generate
85              even_case : if ((i mod 2) = 0) generate
86                  fa_up : fa port map (a => partial_temp(0)(i),
87                                       b => partial_temp(1)(i),
88                                       cin => partial_temp(2)(i),
89                                       cout => partial_temp_layer1(0)(i + 1),
90                                       s => partial_temp_layer1(2)(i));
91                  fa_down : fa port map (a => partial_temp(3)(i),
92                                         b => partial_temp(4)(i),
93                                         cin => partial_temp(5)(i),
94                                         cout => partial_temp_layer1(1)(i + 1),
95                                         s => partial_temp_layer1(3)(i));
96              end generate;
97
98              odd_case : if ((i mod 2) = 1) generate
99                  fa_odd : fa port map (a => partial_temp(0)(i),
100                                        b => partial_temp(1)(i),
101                                        cin => partial_temp(2)(i),
102                                        cout => partial_temp_layer1(0)(i + 1),
103                                        s => partial_temp_layer1(2)(i));
104                 ha_odd : ha port map (a => partial_temp(3)(i),
105                                       b => partial_temp(4)(i),
106                                       cout => partial_temp_layer1(1)(i + 1),
107                                       s => partial_temp_layer1(3)(i));
108             end generate;
109         end generate;
110
111         -- layer 2
112
113         partial_temp_layer2(0)(0) <= partial_temp_layer1(0)(0);
114         partial_temp_layer2(1)(0) <= partial_temp_layer1(1)(0);
115
116         partial_temp_layer2(0)(1) <= partial_temp_layer1(0)(1);
117
118         partial_temp_layer2(0)(2) <= partial_temp_layer1(0)(2);
119         partial_temp_layer2(1)(2) <= partial_temp_layer1(1)(2);
120         partial_temp_layer2(2)(2) <= partial_temp_layer1(2)(2);
121
122         partial_temp_layer2(0)(3) <= partial_temp_layer1(0)(3);
123         partial_temp_layer2(1)(3) <= partial_temp_layer1(1)(3);
```

```
124
125         ha2_3 : ha port map (a => partial_temp_layer1(0)(4),
126                              b => partial_temp_layer1(1)(4),
127                              cout => partial_temp_layer2(0)(5),
128                              s => partial_temp_layer2(0)(4));
129         partial_temp_layer2(1)(4) <= partial_temp_layer1(2)(4);
130         partial_temp_layer2(2)(4) <= partial_temp_layer1(3)(4);
131
132         ha2_1 : ha port map (a => partial_temp_layer1(0)(5),
133                              b => partial_temp_layer1(1)(5),
134                              cout => partial_temp_layer2(0)(6),
135                              s => partial_temp_layer2(1)(5));
136         partial_temp_layer2(2)(5) <= partial_temp_layer1(2)(5);
137
138         -- ha2_2 : ha port map (a => partial_temp_layer1(0)(6),
139         --                      b => partial_temp_layer1(1)(6),
140         --                      cout => partial_temp_layer2(0)(7),
141         --                      s => partial_temp_layer2(1)(6));
142         -- partial_temp_layer2(2)(6) <= partial_temp_layer1(2)(6);
143
144         fa_gen2 : for i in 6 to 2 * Nbit - 1 generate
145             fa2 : fa port map (a => partial_temp_layer1(0)(i),
146                                b => partial_temp_layer1(1)(i),
147                                cin => partial_temp_layer1(2)(i),
148                                cout => partial_temp_layer2(0)(i + 1),
149                                s => partial_temp_layer2(1)(i));
150
151             partial_temp_layer2(2)(i) <= partial_temp_layer1(3)(i);
152         end generate;
153
154         partial_temp_layer2(1)(2 * Nbit) <= partial_temp_layer1(0)(2 * Nbit);
155         partial_temp_layer2(2)(2 * Nbit) <= partial_temp_layer1(1)(2 * Nbit);
156
157
158         -- layer 3
159
160         partial_temp_layer3(0)(0) <= partial_temp_layer2(0)(0);
161         partial_temp_layer3(1)(0) <= partial_temp_layer2(1)(0);
162
163         partial_temp_layer3(0)(1) <= partial_temp_layer2(0)(1);
164
165         ha3_1 : ha port map (a => partial_temp_layer2(0)(2),
166                              b => partial_temp_layer2(1)(2),
167                              cout => partial_temp_layer3(0)(3),
168                              s => partial_temp_layer3(0)(2));
169         partial_temp_layer3(1)(2) <= partial_temp_layer2(2)(2);
170
171         ha3_2 : ha port map (a => partial_temp_layer2(0)(3),
172                              b => partial_temp_layer2(1)(3),
173                              cout => partial_temp_layer3(0)(4),
174                              s => partial_temp_layer3(1)(3));
175
176         fa_gen3 : for i in 4 to 2 * Nbit generate
177             fa3 : fa port map (a => partial_temp_layer2(0)(i),
178                                b => partial_temp_layer2(1)(i),
179                                cin => partial_temp_layer2(2)(i),
180                                cout => partial_temp_layer3(0)(i + 1),
181                                s => partial_temp_layer3(1)(i));
182         end generate;
183
184         partial_temp_layer3(1)(1) <= '0'; -- to avoid errors
185
```

```
186         -- cut down version of 23 bits to 20
187         out1 <= partial_temp_layer3(0)(2 * Nbit - 1 downto 0);
188         out2 <= partial_temp_layer3(1)(2 * Nbit - 1 downto 0);
189 
190     end behav;
```

B.2 Manually-optimized Dadda tree

This section contains the code for the manually-optimized Dadda tree, using custom connections to cover some configurations like 1-1-1 which becomes 1-1.

B.2.1 Partial Product Generation - MBE

./Code/mbe.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  
5  library work;
6  use work.util.all;
7  
8  entity mbe is
9      port (a, b : in signed(Nbit - 1 downto 0);
10           partial : out partial_array);
11 end mbe;
12 
13 architecture behav of mbe is
14     signal q, p_temp : multiple_factoring_array;
15     signal extended_b : signed(Nbit downto 0);
16     begin
17         -- zero right extension for index -1
18         extended_b <= b & '0';
19 
20         -- q generation from MBE
21         multiple_factoring : for i in 0 to Nbit / 2 - 1 generate
22             q(i) <= a(Nbit -1) & a when (extended_b(2 * i + 1) xor extended_b(2 * i))
    = '1' else
23                     a & '0' when ((not (extended_b(2 * i + 1) xor extended_b(2 * i)))
    and (extended_b(2 * i + 2) xor extended_b(2 * i + 1))) = '1' else
24                     (others => '0');
25         end generate;
26 
27         partial_product : for i in 0 to Nbit / 2 - 1 generate
28             -- xor 1 bit for MBE
29             p_temp_gen : for j in 0 to Nbit generate
30                 p_temp(i)(j) <= q(i)(j) xor b(2 * i + 1);
31             end generate;
32             -- partial product without sign for Roorda
33             partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto 0);
34             -- Roorda extension
35             partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
36             -- Zero padding on the right
37             if_padding : if i > 0 generate
38                 partial(i)(2 * i - 1 downto 0) <= (others => '0');
39             end generate;
40             -- MBE carry
```

```
            --p(N / 2)(2 * i) <= b(2 * i + 1);
            -- Roorda carry
            --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
            -- these last two could be compacted in one, MBE carry should be
            -- max in position N - 1, while Roorda carry starts from position
            -- N
            -- MBE + Roorda carry in same line
            partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
            partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
            partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
            partial(Nbit / 2)(2 * i + 1) <= '0';
        end generate;
    end behav;
```

B.2.2 Partial Product Reduction - Dadda tree

./Code/dadda.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity dadda is
    port (partial : in partial_array;
          out1, out2 : out signed(2 * Nbit - 1 downto 0));
end dadda;

architecture behav of dadda is
    component fa
        port (cin, a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component ha
        port (a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
    signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
    signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 3 downto 0);
    signal partial_temp_layer3 : internal_partial_array(Nbit / 2 - 4 downto 0);

    begin
        partial_association : for i in 0 to Nbit / 2 generate
            partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
            partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
        end generate;

        -- first stage --> from 6 to 4
        -- HA --> carry in 0 and sum in 1
        ha1_1 : ha port map (a => partial_temp(0)(6),
                             b => partial_temp(1)(6),
                             cout => partial_temp_layer1(0)(7),
                             s => partial_temp_layer1(1)(6));
        ha1_2 : ha port map (a => partial_temp(0)(7),
                             b => partial_temp(1)(7),
```

```
43                              cout => partial_temp_layer1(0)(8),
44                              s => partial_temp_layer1(1)(7));
45          ha1_3 : ha port map (a => partial_temp(3)(8),
46                              b => partial_temp(4)(8),
47                              cout => partial_temp_layer1(0)(9),
48                              s => partial_temp_layer1(1)(8));
49          ha1_4 : ha port map (a => partial_temp(3)(9),
50                              b => partial_temp(4)(9),
51                              cout => partial_temp_layer1(0)(10),
52                              s => partial_temp_layer1(1)(9));
53          ha1_5 : ha port map (a => partial_temp(3)(11),
54                              b => partial_temp(4)(11),
55                              cout => partial_temp_layer1(0)(12),
56                              s => partial_temp_layer1(1)(11));
57          ha1_6 : ha port map (a => partial_temp(3)(15),
58                              b => partial_temp(4)(15),
59                              cout => partial_temp_layer1(0)(16),
60                              s => partial_temp_layer1(1)(15));
61          ha1_7 : ha port map (a => partial_temp(3)(13),
62                              b => partial_temp(4)(13),
63                              cout => partial_temp_layer1(0)(14),
64                              s => partial_temp_layer1(1)(13));
65          -- FA --> carry in 2/0 and sum in 3/1
66          fa1_1 : fa port map (a => partial_temp(0)(8),
67                              b => partial_temp(1)(8),
68                              cin => partial_temp(2)(8),
69                              cout => partial_temp_layer1(2)(9),
70                              s => partial_temp_layer1(3)(8));
71          fa1_2 : fa port map (a => partial_temp(0)(9),
72                              b => partial_temp(1)(9),
73                              cin => partial_temp(2)(9),
74                              cout => partial_temp_layer1(2)(10),
75                              s => partial_temp_layer1(3)(9));
76          fa1_3 : fa port map (a => partial_temp(0)(10),
77                              b => partial_temp(1)(10),
78                              cin => partial_temp(2)(10),
79                              cout => partial_temp_layer1(2)(11),
80                              s => partial_temp_layer1(3)(10));
81          fa1_4 : fa port map (a => partial_temp(0)(11),
82                              b => partial_temp(1)(11),
83                              cin => partial_temp(2)(11),
84                              cout => partial_temp_layer1(2)(12),
85                              s => partial_temp_layer1(3)(11));
86          fa1_5 : fa port map (a => partial_temp(3)(10),
87                              b => partial_temp(4)(10),
88                              cin => partial_temp(5)(10),
89                              cout => partial_temp_layer1(0)(11),
90                              s => partial_temp_layer1(1)(10)); -- row 3 already taken
      by FA, free for HA
91          fa1_6 : fa port map (a => partial_temp(3)(12),
92                              b => partial_temp(4)(12),
93                              cin => partial_temp(5)(12),
94                              cout => partial_temp_layer1(2)(13),
95                              s => partial_temp_layer1(3)(12));
96          fa1_7 : fa port map (a => partial_temp(3)(14),
97                              b => partial_temp(4)(14),
98                              cin => partial_temp(5)(14),
99                              cout => partial_temp_layer1(2)(15),
100                             s => partial_temp_layer1(3)(14));
101         fa1_8 : fa port map (a => partial_temp(3)(16),
102                             b => partial_temp(4)(16),
103                             cin => partial_temp(5)(16),
```

```
104                             cout => partial_temp_layer1(2)(17),
105                             s => partial_temp_layer1(3)(16));
106         -- propagations
107         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
108         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
109         partial_temp_layer1(1)(5 downto 2) <= partial_temp(1)(5 downto 2);
110         partial_temp_layer1(2)(7) <= partial_temp(2)(7);
111         --partial_temp_layer1(2)(6) <= partial_temp(2)(6);
112         partial_temp_layer1(2)(5) <= partial_temp(2)(5);
113         partial_temp_layer1(2)(4) <= partial_temp(2)(4);
114         partial_temp_layer1(3)(7) <= partial_temp(3)(7);
115         partial_temp_layer1(2)(6) <= partial_temp(3)(6); -- because 3 will be used for
        Roorda and MBE carries
116         partial_temp_layer1(0)(6) <= partial_temp(2)(6);
117         partial_temp_layer1(3)(6) <= partial_temp(5)(6);
118         partial_temp_layer1(2)(8) <= partial_temp(5)(8); -- used by FA
119         roorda_mbe_carries : for i in 0 to 3 generate
120             partial_temp_layer1(3)(2 * i) <= partial_temp(5)(2 * i);
121         end generate;
122         -- custom connections
123         -- positioned in HA sum 1
124         -- three ones
125         partial_temp_layer1(1)(14) <= '1';
126         partial_temp_layer1(1)(16) <= '1';
127         partial_temp_layer1(1)(19) <= '1';
128         -- two ones
129         partial_temp_layer1(1)(18) <= partial_temp(5)(18);
130         --partial_temp_layer1(1)(12) <= partial_temp(3)(12);
131         -- one one
132         partial_temp_layer1(1)(17) <= not partial_temp(4)(17);
133         -- position in HA carry 0
134         -- three ones
135         partial_temp_layer1(0)(15) <= '1';
136         partial_temp_layer1(0)(17) <= '1';
137         -- two ones
138         partial_temp_layer1(0)(19) <= '1';
139         partial_temp_layer1(0)(13) <= '1';
140         partial_temp_layer1(0)(20) <= '1';
141         -- one one
142         partial_temp_layer1(0)(18) <= partial_temp(4)(17);
143         -- positioned in FA sum 3
144         -- three ones
145         partial_temp_layer1(3)(15) <= '1';
146         partial_temp_layer1(3)(17) <= '1';
147         partial_temp_layer1(3)(18) <= '1';
148         -- two ones
149         partial_temp_layer1(1)(12) <= partial_temp(2)(12); -- taken by FA
150         partial_temp_layer1(3)(13) <= partial_temp(2)(13);
151         -- positioned in FA carry 2
152         -- three ones
153         partial_temp_layer1(2)(16) <= '1';
154         partial_temp_layer1(2)(18) <= '1';
155         partial_temp_layer1(2)(19) <= '1';
156         -- two ones
157         partial_temp_layer1(2)(14) <= partial_temp(2)(14);
158         partial_temp_layer1(2)(20) <= '1';
159
160
161
162
163
164
```

```
165
166         -- layer 2
167         -- HA --> sum 0 carry 1
168         ha2_1 : ha port map (a => partial_temp_layer1(0)(4),
169                              b => partial_temp_layer1(3)(4),
170                              cout => partial_temp_layer2(1)(5),
171                              s => partial_temp_layer2(0)(4));
172         ha2_2 : ha port map (a => partial_temp_layer1(0)(5),
173                              b => partial_temp_layer1(1)(5),
174                              cout => partial_temp_layer2(1)(6),
175                              s => partial_temp_layer2(0)(5));
176         -- ha2_4 : ha port map (a => partial_temp_layer1(0)(16),
177         --                      b => partial_temp_layer1(1)(16),
178         --                      cout => partial_temp_layer2(1)(17),
179         --                      s => partial_temp_layer2(0)(16));
180         -- FA --> sum 0 carry 1
181         fa2_1 : fa port map (a => partial_temp_layer1(0)(6),
182                              b => partial_temp_layer1(1)(6),
183                              cin => partial_temp_layer1(3)(6),
184                              cout => partial_temp_layer2(1)(7),
185                              s => partial_temp_layer2(0)(6));
186         fa2_2 : fa port map (a => partial_temp_layer1(0)(7),
187                              b => partial_temp_layer1(1)(7),
188                              cin => partial_temp_layer1(3)(7),
189                              cout => partial_temp_layer2(1)(8),
190                              s => partial_temp_layer2(0)(7));
191         fa2_3 : fa port map (a => partial_temp_layer1(0)(8),
192                              b => partial_temp_layer1(1)(8),
193                              cin => partial_temp_layer1(2)(8),
194                              cout => partial_temp_layer2(1)(9),
195                              s => partial_temp_layer2(0)(8));
196         fa2_4 : fa port map (a => partial_temp_layer1(0)(9),
197                              b => partial_temp_layer1(1)(9),
198                              cin => partial_temp_layer1(2)(9),
199                              cout => partial_temp_layer2(1)(10),
200                              s => partial_temp_layer2(0)(9));
201         fa2_5 : fa port map (a => partial_temp_layer1(0)(10),
202                              b => partial_temp_layer1(1)(10),
203                              cin => partial_temp_layer1(2)(10),
204                              cout => partial_temp_layer2(1)(11),
205                              s => partial_temp_layer2(0)(10));
206         fa2_6 : fa port map (a => partial_temp_layer1(0)(11),
207                              b => partial_temp_layer1(1)(11),
208                              cin => partial_temp_layer1(2)(11),
209                              cout => partial_temp_layer2(1)(12),
210                              s => partial_temp_layer2(0)(11));
211         fa2_7 : fa port map (a => partial_temp_layer1(0)(12),
212                              b => partial_temp_layer1(1)(12),
213                              cin => partial_temp_layer1(2)(12),
214                              cout => partial_temp_layer2(1)(13),
215                              s => partial_temp_layer2(0)(12));
216         fa2_8 : fa port map (a => partial_temp_layer1(0)(14),
217                              b => partial_temp_layer1(2)(14),
218                              cin => partial_temp_layer1(3)(14),
219                              cout => partial_temp_layer2(1)(15),
220                              s => partial_temp_layer2(0)(14));
221         fa2_9 : fa port map (a => partial_temp_layer1(2)(13),
222                              b => partial_temp_layer1(3)(13),
223                              cin => partial_temp_layer1(1)(13),
224                              cout => partial_temp_layer2(1)(14),
225                              s => partial_temp_layer2(0)(13));
226
```

```
227         -- propagation
228         partial_temp_layer2(0)(3 downto 0) <= partial_temp_layer1(0)(3 downto 0);
229         partial_temp_layer2(1)(0) <= partial_temp_layer1(3)(0);
230         partial_temp_layer2(1)(4 downto 2) <= partial_temp_layer1(1)(4 downto 2);
231         partial_temp_layer2(2)(2) <= partial_temp_layer1(3)(2);
232         partial_temp_layer2(2)(7 downto 4) <= partial_temp_layer1(2)(7 downto 4);
233         partial_temp_layer2(2)(12 downto 8) <= partial_temp_layer1(3)(12 downto 8);
234         partial_temp_layer2(2)(14 downto 13) <= (others => '1');
235         partial_temp_layer2(2)(15) <= partial_temp_layer1(2)(15);
236         --partial_temp_layer2(0)(16) <= partial_temp_layer1(1)(16);
237         partial_temp_layer2(0)(16) <= partial_temp_layer1(0)(16);
238         partial_temp_layer2(2)(16) <= partial_temp_layer1(3)(16);
239         --partial_temp_layer2(1)(17) <= partial_temp_layer1(1)(17);
240         partial_temp_layer2(2)(17) <= partial_temp_layer1(2)(17);
241         partial_temp_layer2(2)(18) <= partial_temp_layer1(1)(18);
242         partial_temp_layer2(2)(19) <= partial_temp_layer1(2)(19);
243         partial_temp_layer2(0)(20) <= partial_temp_layer1(0)(20);
244         partial_temp_layer2(2)(20) <= partial_temp_layer1(2)(20);
245
246
247         -- custom connection
248         partial_temp_layer2(0)(15) <= partial_temp_layer1(1)(15);
249         partial_temp_layer2(1)(16) <= '1';
250
251         partial_temp_layer2(0)(17) <= partial_temp_layer1(1)(17);
252         partial_temp_layer2(1)(17) <= '1';
253         partial_temp_layer2(1)(18) <= '1';
254
255         partial_temp_layer2(0)(18) <= partial_temp_layer1(0)(18);
256         partial_temp_layer2(1)(19) <= '1';
257
258         partial_temp_layer2(0)(19) <= '0';
259         partial_temp_layer2(1)(20) <= '1';
260
261
262
263
264
265         -- layer 3
266         -- HA
267         ha3_1 : ha port map (a => partial_temp_layer2(0)(2),
268                              b => partial_temp_layer2(1)(2),
269                              cout => partial_temp_layer3(1)(3),
270                              s => partial_temp_layer3(0)(2));
271         ha3_2 : ha port map (a => partial_temp_layer2(0)(3),
272                              b => partial_temp_layer2(1)(3),
273                              cout => partial_temp_layer3(1)(4),
274                              s => partial_temp_layer3(0)(3));
275
276         -- FA
277         fa3_1 : fa port map (a => partial_temp_layer2(0)(4),
278                              b => partial_temp_layer2(1)(4),
279                              cin => partial_temp_layer2(2)(4),
280                              cout => partial_temp_layer3(1)(5),
281                              s => partial_temp_layer3(0)(4));
282
283         fa3_2 : fa port map (a => partial_temp_layer2(0)(5),
284                              b => partial_temp_layer2(1)(5),
285                              cin => partial_temp_layer2(2)(5),
286                              cout => partial_temp_layer3(1)(6),
287                              s => partial_temp_layer3(0)(5));
288
```

```
289         fa3_3 : fa port map (a => partial_temp_layer2(0)(6),
290                              b => partial_temp_layer2(1)(6),
291                              cin => partial_temp_layer2(2)(6),
292                              cout => partial_temp_layer3(1)(7),
293                              s => partial_temp_layer3(0)(6));
294
295         fa3_4 : fa port map (a => partial_temp_layer2(0)(7),
296                              b => partial_temp_layer2(1)(7),
297                              cin => partial_temp_layer2(2)(7),
298                              cout => partial_temp_layer3(1)(8),
299                              s => partial_temp_layer3(0)(7));
300
301         fa3_5 : fa port map (a => partial_temp_layer2(0)(8),
302                              b => partial_temp_layer2(1)(8),
303                              cin => partial_temp_layer2(2)(8),
304                              cout => partial_temp_layer3(1)(9),
305                              s => partial_temp_layer3(0)(8));
306
307         fa3_6 : fa port map (a => partial_temp_layer2(0)(9),
308                              b => partial_temp_layer2(1)(9),
309                              cin => partial_temp_layer2(2)(9),
310                              cout => partial_temp_layer3(1)(10),
311                              s => partial_temp_layer3(0)(9));
312
313         fa3_7 : fa port map (a => partial_temp_layer2(0)(10),
314                              b => partial_temp_layer2(1)(10),
315                              cin => partial_temp_layer2(2)(10),
316                              cout => partial_temp_layer3(1)(11),
317                              s => partial_temp_layer3(0)(10));
318
319         fa3_8 : fa port map (a => partial_temp_layer2(0)(11),
320                              b => partial_temp_layer2(1)(11),
321                              cin => partial_temp_layer2(2)(11),
322                              cout => partial_temp_layer3(1)(12),
323                              s => partial_temp_layer3(0)(11));
324
325         fa3_9 : fa port map (a => partial_temp_layer2(0)(12),
326                              b => partial_temp_layer2(1)(12),
327                              cin => partial_temp_layer2(2)(12),
328                              cout => partial_temp_layer3(1)(13),
329                              s => partial_temp_layer3(0)(12));
330
331         fa3_10 : fa port map (a => partial_temp_layer2(0)(13),
332                               b => partial_temp_layer2(1)(13),
333                               cin => partial_temp_layer2(2)(13),
334                               cout => partial_temp_layer3(1)(14),
335                               s => partial_temp_layer3(0)(13));
336
337         fa3_11 : fa port map (a => partial_temp_layer2(0)(14),
338                               b => partial_temp_layer2(1)(14),
339                               cin => partial_temp_layer2(2)(14),
340                               cout => partial_temp_layer3(1)(15),
341                               s => partial_temp_layer3(0)(14));
342
343         fa3_12 : fa port map (a => partial_temp_layer2(0)(15),
344                               b => partial_temp_layer2(1)(15),
345                               cin => partial_temp_layer2(2)(15),
346                               cout => partial_temp_layer3(1)(16),
347                               s => partial_temp_layer3(0)(15));
348
349         fa3_13 : fa port map (a => partial_temp_layer2(0)(17),
350                               b => partial_temp_layer2(1)(17),
```

```
351                             cin => partial_temp_layer2(2)(17),
352                             cout => partial_temp_layer3(1)(18),
353                             s => partial_temp_layer3(0)(17));
354
355         fa3_14 : fa port map (a => partial_temp_layer2(0)(18),
356                             b => partial_temp_layer2(1)(18),
357                             cin => partial_temp_layer2(2)(18),
358                             cout => partial_temp_layer3(1)(19),
359                             s => partial_temp_layer3(0)(18));
360
361         fa3_15 : fa port map (a => partial_temp_layer2(0)(16),
362                             b => partial_temp_layer2(1)(16),
363                             cin => partial_temp_layer2(2)(16),
364                             cout => partial_temp_layer3(1)(17),
365                             s => partial_temp_layer3(0)(16));
366
367
368
369         -- propagation
370         partial_temp_layer3(0)(0) <= partial_temp_layer2(0)(0);
371         partial_temp_layer3(1)(0) <= partial_temp_layer2(1)(0);
372
373         partial_temp_layer3(0)(1) <= partial_temp_layer2(0)(1);
374
375         -- partial_temp_layer3(1)(1) <= partial_temp_layer2(2)(1);
376         partial_temp_layer3(1)(1) <= '0'; -- to avoid U
377         partial_temp_layer3(1)(2) <= partial_temp_layer2(2)(2);
378
379         partial_temp_layer3(0)(20) <= partial_temp_layer2(0)(20);
380
381         -- custom connections
382         -- partial_temp_layer3(0)(16) <= partial_temp_layer2(2)(16);
383         -- partial_temp_layer3(1)(17) <= '1';
384
385         partial_temp_layer3(0)(19) <= '0';
386         partial_temp_layer3(1)(20) <= '1';
387
388         partial_temp_layer3(1)(21) <= '1';
389
390         -- cut down version of 23 bits to 20
391         out1 <= partial_temp_layer3(0)(2 * Nbit - 1 downto 0);
392         out2 <= partial_temp_layer3(1)(2 * Nbit - 1 downto 0);
393
394     end behav;
```

B.3 Standard Dadda tree without 6 LSBs

This section contains the code for the standard Dadda tree with no FA/HA in the last 6 LSBs.

B.3.1 Partial Product Generation - MBE

./Code/mbe.vhd

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 library work;
```

```
use work.util.all;

entity mbe is
    port (a, b : in signed(Nbit - 1 downto 0);
          partial : out partial_array);
end mbe;

architecture behav of mbe is
    signal q, p_temp : multiple_factoring_array;
    signal extended_b : signed(Nbit downto 0);
    begin
        -- zero right extension for index -1
        extended_b <= b & '0';

        -- q generation from MBE
        multiple_factoring : for i in 0 to Nbit / 2 - 1 generate
            q(i) <= a(Nbit -1) & a when (extended_b(2 * i + 1) xor extended_b(2 * i))
    = '1' else
                    a & '0' when ((not (extended_b(2 * i + 1) xor extended_b(2 * i)))
    and (extended_b(2 * i + 2) xor extended_b(2 * i + 1))) = '1' else
                    (others => '0');
        end generate;

        partial_product : for i in 0 to Nbit / 2 - 1 generate
            -- xor 1 bit for MBE
            p_temp_gen : for j in 0 to Nbit generate
                p_temp(i)(j) <= q(i)(j) xor b(2 * i + 1);
            end generate;
            -- partial product without sign for Roorda
            partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto 0);
            -- Roorda extension
            partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
            -- Zero padding on the right
            if_padding : if i > 0 generate
                partial(i)(2 * i - 1 downto 0) <= (others => '0');
            end generate;
            -- MBE carry
            --p(N / 2)(2 * i) <= b(2 * i + 1);
            -- Roorda carry
            --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
            -- these last two could be compacted in one, MBE carry should be
            -- max in position N - 1, while Roorda carry starts from position
            -- N
            -- MBE + Roorda carry in same line
            partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
            partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
            partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
            partial(Nbit / 2)(2 * i + 1) <= '0';
        end generate;
    end behav;
```

B.3.2 Partial Product Reduction - Dadda tree

./Code/dadda_standard_no6LSB.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
```

```
6  use work.util.all;
7
8  entity dadda_standard_no6LSB is
9      port (partial : in partial_array;
10           out1, out2 : out signed(2 * Nbit - 1 downto 0));
11 end dadda_standard_no6LSB;
12
13 architecture behav of dadda_standard_no6LSB is
14     component fa
15         port (cin, a, b : in std_logic;
16             cout, s : out std_logic);
17     end component;
18
19     component ha
20         port (a, b : in std_logic;
21             cout, s : out std_logic);
22     end component;
23
24     signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
25     signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
26     signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 3 downto 0);
27     signal partial_temp_layer3 : internal_partial_array(Nbit / 2 - 4 downto 0);
28
29     begin
30         partial_association : for i in 0 to Nbit / 2 generate
31             partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
32             partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
33         end generate;
34
35         -- first stage --> from 6 to 4
36         ha1_1 : ha port map (a => partial_temp(0)(6),
37                              b => partial_temp(1)(6),
38                              cout => partial_temp_layer1(0)(7),
39                              s => partial_temp_layer1(0)(6));
40         partial_temp_layer1(1)(6) <= partial_temp(2)(6);
41         partial_temp_layer1(2)(6) <= partial_temp(3)(6);
42         partial_temp_layer1(3)(6) <= partial_temp(5)(6);
43
44         ha1_2 : ha port map (a => partial_temp(0)(7),
45                              b => partial_temp(1)(7),
46                              cout => partial_temp_layer1(0)(8),
47                              s => partial_temp_layer1(1)(7));
48         partial_temp_layer1(2)(7) <= partial_temp(2)(7);
49         partial_temp_layer1(3)(7) <= partial_temp(3)(7);
50
51         fa1_1 : fa port map (a => partial_temp(0)(8),
52                              b => partial_temp(1)(8),
53                              cin => partial_temp(2)(8),
54                              cout => partial_temp_layer1(0)(9),
55                              s => partial_temp_layer1(1)(8));
56         ha1_3 : ha port map (a => partial_temp(3)(8),
57                              b => partial_temp(4)(8),
58                              cout => partial_temp_layer1(1)(9),
59                              s => partial_temp_layer1(2)(8));
60         partial_temp_layer1(3)(8) <= partial_temp(5)(8);
61
62
63         fa_ha : for i in 9 to 2 * Nbit - 1 generate
64             even_case : if ((i mod 2) = 0) generate
65                 fa_up : fa port map (a => partial_temp(0)(i),
66                                      b => partial_temp(1)(i),
67                                      cin => partial_temp(2)(i),
```

```
68                                      cout => partial_temp_layer1(0)(i + 1),
69                                      s => partial_temp_layer1(2)(i));
70              fa_down : fa port map (a => partial_temp(3)(i),
71                                      b => partial_temp(4)(i),
72                                      cin => partial_temp(5)(i),
73                                      cout => partial_temp_layer1(1)(i + 1),
74                                      s => partial_temp_layer1(3)(i));
75          end generate;
76
77          odd_case : if ((i mod 2) = 1) generate
78              fa_odd : fa port map (a => partial_temp(0)(i),
79                                    b => partial_temp(1)(i),
80                                    cin => partial_temp(2)(i),
81                                    cout => partial_temp_layer1(0)(i + 1),
82                                    s => partial_temp_layer1(2)(i));
83              ha_odd : ha port map (a => partial_temp(3)(i),
84                                    b => partial_temp(4)(i),
85                                    cout => partial_temp_layer1(1)(i + 1),
86                                    s => partial_temp_layer1(3)(i));
87          end generate;
88      end generate;
89
90      -- layer 2
91      fa_gen2 : for i in 6 to 2 * Nbit - 1 generate
92          fa2 : fa port map (a => partial_temp_layer1(0)(i),
93                             b => partial_temp_layer1(1)(i),
94                             cin => partial_temp_layer1(2)(i),
95                             cout => partial_temp_layer2(0)(i + 1),
96                             s => partial_temp_layer2(1)(i));
97
98          partial_temp_layer2(2)(i) <= partial_temp_layer1(3)(i);
99      end generate;
100
101     partial_temp_layer2(0)(6) <= '0';
102
103     partial_temp_layer2(1)(2 * Nbit) <= partial_temp_layer1(0)(2 * Nbit);
104     partial_temp_layer2(2)(2 * Nbit) <= partial_temp_layer1(1)(2 * Nbit);
105
106
107     -- layer 3
108
109     fa_gen3 : for i in 6 to 2 * Nbit generate
110         fa3 : fa port map (a => partial_temp_layer2(0)(i),
111                            b => partial_temp_layer2(1)(i),
112                            cin => partial_temp_layer2(2)(i),
113                            cout => partial_temp_layer3(0)(i + 1),
114                            s => partial_temp_layer3(1)(i));
115     end generate;
116
117     partial_temp_layer3(0)(6) <= '0';
118
119
120     -- cut down version of 23 bits to 20
121     out1(5 downto 0) <= (others => '0');
122     out2(5 downto 0) <= (others => '0');
123     out1(2 * Nbit - 1 downto 6) <= partial_temp_layer3(0)(2 * Nbit - 1 downto 6);
124     out2(2 * Nbit - 1 downto 6) <= partial_temp_layer3(1)(2 * Nbit - 1 downto 6);
125
126 end behav;
```

B.4 Manually-optimized Dadda tree without 6 LSBs

This section contains the code for the standard Dadda tree with no FA/HA in the last 6 LSBs.

B.4.1 Partial Product Generation - MBE

./Code/mbe.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity mbe is
    port (a, b : in signed(Nbit - 1 downto 0);
          partial : out partial_array);
end mbe;

architecture behav of mbe is
    signal q, p_temp : multiple_factoring_array;
    signal extended_b : signed(Nbit downto 0);
    begin
        -- zero right extension for index -1
        extended_b <= b & '0';

        -- q generation from MBE
        multiple_factoring : for i in 0 to Nbit / 2 - 1 generate
            q(i) <= a(Nbit -1) & a when (extended_b(2 * i + 1) xor extended_b(2 * i)) = '1' else
                    a & '0' when ((not (extended_b(2 * i + 1) xor extended_b(2 * i))) and (extended_b(2 * i + 2) xor extended_b(2 * i + 1))) = '1' else
                    (others => '0');
        end generate;

        partial_product : for i in 0 to Nbit / 2 - 1 generate
            -- xor 1 bit for MBE
            p_temp_gen : for j in 0 to Nbit generate
                p_temp(i)(j) <= q(i)(j) xor b(2 * i + 1);
            end generate;
            -- partial product without sign for Roorda
            partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto 0);
            -- Roorda extension
            partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
            -- Zero padding on the right
            if_padding : if i > 0 generate
                partial(i)(2 * i - 1 downto 0) <= (others => '0');
            end generate;
            -- MBE carry
            --p(N / 2)(2 * i) <= b(2 * i + 1);
            -- Roorda carry
            --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
            -- these last two could be compacted in one, MBE carry should be
            -- max in position N - 1, while Roorda carry starts from position
            -- N
            -- MBE + Roorda carry in same line
            partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
            partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
            partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
```

```
            partial(Nbit / 2)(2 * i + 1) <= '0';
        end generate;
    end behav;
```

B.4.2 Partial Product Reduction - Dadda tree

./Code/dadda_no6LSB.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity dadda_no6LSB is
    port (partial : in partial_array;
          out1, out2 : out signed(2 * Nbit - 1 downto 0));
end dadda_no6LSB;

architecture behav of dadda_no6LSB is
    component fa
        port (cin, a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component ha
        port (a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
    signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
    signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 3 downto 0);
    signal partial_temp_layer3 : internal_partial_array(Nbit / 2 - 4 downto 0);

    begin
        partial_association : for i in 0 to Nbit / 2 generate
            partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
            partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
        end generate;

        -- first stage --> from 6 to 4
        -- HA --> carry in 0 and sum in 1
        ha1_1 : ha port map (a => partial_temp(0)(6),
                             b => partial_temp(1)(6),
                             cout => partial_temp_layer1(0)(7),
                             s => partial_temp_layer1(1)(6));
        ha1_2 : ha port map (a => partial_temp(0)(7),
                             b => partial_temp(1)(7),
                             cout => partial_temp_layer1(0)(8),
                             s => partial_temp_layer1(1)(7));
        ha1_3 : ha port map (a => partial_temp(3)(8),
                             b => partial_temp(4)(8),
                             cout => partial_temp_layer1(0)(9),
                             s => partial_temp_layer1(1)(8));
        ha1_4 : ha port map (a => partial_temp(3)(9),
                             b => partial_temp(4)(9),
                             cout => partial_temp_layer1(0)(10),
                             s => partial_temp_layer1(1)(9));
```

```
53          ha1_5 : ha port map (a => partial_temp(3)(11),
54                               b => partial_temp(4)(11),
55                               cout => partial_temp_layer1(0)(12),
56                               s => partial_temp_layer1(1)(11));
57          ha1_6 : ha port map (a => partial_temp(3)(15),
58                               b => partial_temp(4)(15),
59                               cout => partial_temp_layer1(0)(16),
60                               s => partial_temp_layer1(1)(15));
61          ha1_7 : ha port map (a => partial_temp(3)(13),
62                               b => partial_temp(4)(13),
63                               cout => partial_temp_layer1(0)(14),
64                               s => partial_temp_layer1(1)(13));
65          -- FA --> carry in 2/0 and sum in 3/1
66          fa1_1 : fa port map (a => partial_temp(0)(8),
67                               b => partial_temp(1)(8),
68                               cin => partial_temp(2)(8),
69                               cout => partial_temp_layer1(2)(9),
70                               s => partial_temp_layer1(3)(8));
71          fa1_2 : fa port map (a => partial_temp(0)(9),
72                               b => partial_temp(1)(9),
73                               cin => partial_temp(2)(9),
74                               cout => partial_temp_layer1(2)(10),
75                               s => partial_temp_layer1(3)(9));
76          fa1_3 : fa port map (a => partial_temp(0)(10),
77                               b => partial_temp(1)(10),
78                               cin => partial_temp(2)(10),
79                               cout => partial_temp_layer1(2)(11),
80                               s => partial_temp_layer1(3)(10));
81          fa1_4 : fa port map (a => partial_temp(0)(11),
82                               b => partial_temp(1)(11),
83                               cin => partial_temp(2)(11),
84                               cout => partial_temp_layer1(2)(12),
85                               s => partial_temp_layer1(3)(11));
86          fa1_5 : fa port map (a => partial_temp(3)(10),
87                               b => partial_temp(4)(10),
88                               cin => partial_temp(5)(10),
89                               cout => partial_temp_layer1(0)(11),
90                               s => partial_temp_layer1(1)(10)); -- row 3 already taken
      by FA, free for HA
91          fa1_6 : fa port map (a => partial_temp(3)(12),
92                               b => partial_temp(4)(12),
93                               cin => partial_temp(5)(12),
94                               cout => partial_temp_layer1(2)(13),
95                               s => partial_temp_layer1(3)(12));
96          fa1_7 : fa port map (a => partial_temp(3)(14),
97                               b => partial_temp(4)(14),
98                               cin => partial_temp(5)(14),
99                               cout => partial_temp_layer1(2)(15),
100                              s => partial_temp_layer1(3)(14));
101         fa1_8 : fa port map (a => partial_temp(3)(16),
102                              b => partial_temp(4)(16),
103                              cin => partial_temp(5)(16),
104                              cout => partial_temp_layer1(2)(17),
105                              s => partial_temp_layer1(3)(16));
106         -- propagations
107         partial_temp_layer1(2)(7) <= partial_temp(2)(7);
108         --partial_temp_layer1(2)(6) <= partial_temp(2)(6);
109         partial_temp_layer1(2)(4) <= partial_temp(2)(4);
110         partial_temp_layer1(3)(7) <= partial_temp(3)(7);
111         partial_temp_layer1(2)(6) <= partial_temp(3)(6); -- because 3 will be used for
      Roorda and MBE carries
112         partial_temp_layer1(0)(6) <= partial_temp(2)(6);
```

```
113         partial_temp_layer1(3)(6) <= partial_temp(5)(6);
114         partial_temp_layer1(2)(8) <= partial_temp(5)(8); -- used by FA
115         roorda_mbe_carries : for i in 0 to 3 generate
116             partial_temp_layer1(3)(2 * i) <= partial_temp(5)(2 * i);
117         end generate;
118         -- custom connections
119         -- positioned in HA sum 1
120         -- three ones
121         partial_temp_layer1(1)(14) <= '1';
122         partial_temp_layer1(1)(16) <= '1';
123         partial_temp_layer1(1)(19) <= '1';
124         -- two ones
125         partial_temp_layer1(1)(18) <= partial_temp(5)(18);
126         --partial_temp_layer1(1)(12) <= partial_temp(3)(12);
127         -- one one
128         partial_temp_layer1(1)(17) <= not partial_temp(4)(17);
129         -- position in HA carry 0
130         -- three ones
131         partial_temp_layer1(0)(15) <= '1';
132         partial_temp_layer1(0)(17) <= '1';
133         -- two ones
134         partial_temp_layer1(0)(19) <= '1';
135         partial_temp_layer1(0)(13) <= '1';
136         partial_temp_layer1(0)(20) <= '1';
137         -- one one
138         partial_temp_layer1(0)(18) <= partial_temp(4)(17);
139         -- positioned in FA sum 3
140         -- three ones
141         partial_temp_layer1(3)(15) <= '1';
142         partial_temp_layer1(3)(17) <= '1';
143         partial_temp_layer1(3)(18) <= '1';
144         -- two ones
145         partial_temp_layer1(1)(12) <= partial_temp(2)(12); -- taken by FA
146         partial_temp_layer1(3)(13) <= partial_temp(2)(13);
147         -- positioned in FA carry 2
148         -- three ones
149         partial_temp_layer1(2)(16) <= '1';
150         partial_temp_layer1(2)(18) <= '1';
151         partial_temp_layer1(2)(19) <= '1';
152         -- two ones
153         partial_temp_layer1(2)(14) <= partial_temp(2)(14);
154         partial_temp_layer1(2)(20) <= '1';
155
156
157
158
159
160
161
162         -- layer 2
163         partial_temp_layer2(1)(6) <= '0';
164         -- FA --> sum 0 carry 1
165         fa2_1 : fa port map (a => partial_temp_layer1(0)(6),
166                              b => partial_temp_layer1(1)(6),
167                              cin => partial_temp_layer1(3)(6),
168                              cout => partial_temp_layer2(1)(7),
169                              s => partial_temp_layer2(0)(6));
170         fa2_2 : fa port map (a => partial_temp_layer1(0)(7),
171                              b => partial_temp_layer1(1)(7),
172                              cin => partial_temp_layer1(3)(7),
173                              cout => partial_temp_layer2(1)(8),
174                              s => partial_temp_layer2(0)(7));
```

```
175         fa2_3 : fa port map (a => partial_temp_layer1(0)(8),
176                              b => partial_temp_layer1(1)(8),
177                              cin => partial_temp_layer1(2)(8),
178                              cout => partial_temp_layer2(1)(9),
179                              s => partial_temp_layer2(0)(8));
180         fa2_4 : fa port map (a => partial_temp_layer1(0)(9),
181                              b => partial_temp_layer1(1)(9),
182                              cin => partial_temp_layer1(2)(9),
183                              cout => partial_temp_layer2(1)(10),
184                              s => partial_temp_layer2(0)(9));
185         fa2_5 : fa port map (a => partial_temp_layer1(0)(10),
186                              b => partial_temp_layer1(1)(10),
187                              cin => partial_temp_layer1(2)(10),
188                              cout => partial_temp_layer2(1)(11),
189                              s => partial_temp_layer2(0)(10));
190         fa2_6 : fa port map (a => partial_temp_layer1(0)(11),
191                              b => partial_temp_layer1(1)(11),
192                              cin => partial_temp_layer1(2)(11),
193                              cout => partial_temp_layer2(1)(12),
194                              s => partial_temp_layer2(0)(11));
195         fa2_7 : fa port map (a => partial_temp_layer1(0)(12),
196                              b => partial_temp_layer1(1)(12),
197                              cin => partial_temp_layer1(2)(12),
198                              cout => partial_temp_layer2(1)(13),
199                              s => partial_temp_layer2(0)(12));
200         fa2_8 : fa port map (a => partial_temp_layer1(0)(14),
201                              b => partial_temp_layer1(2)(14),
202                              cin => partial_temp_layer1(3)(14),
203                              cout => partial_temp_layer2(1)(15),
204                              s => partial_temp_layer2(0)(14));
205         fa2_9 : fa port map (a => partial_temp_layer1(2)(13),
206                              b => partial_temp_layer1(3)(13),
207                              cin => partial_temp_layer1(1)(13),
208                              cout => partial_temp_layer2(1)(14),
209                              s => partial_temp_layer2(0)(13));
210 
211         -- propagation
212         partial_temp_layer2(2)(7 downto 6) <= partial_temp_layer1(2)(7 downto 6);
213         partial_temp_layer2(2)(12 downto 8) <= partial_temp_layer1(3)(12 downto 8);
214         partial_temp_layer2(2)(14 downto 13) <= (others => '1');
215         partial_temp_layer2(2)(15) <= partial_temp_layer1(2)(15);
216         --partial_temp_layer2(0)(16) <= partial_temp_layer1(1)(16);
217         partial_temp_layer2(0)(16) <= partial_temp_layer1(0)(16);
218         partial_temp_layer2(2)(16) <= partial_temp_layer1(3)(16);
219         --partial_temp_layer2(1)(17) <= partial_temp_layer1(1)(17);
220         partial_temp_layer2(2)(17) <= partial_temp_layer1(2)(17);
221         partial_temp_layer2(2)(18) <= partial_temp_layer1(1)(18);
222         partial_temp_layer2(2)(19) <= partial_temp_layer1(2)(19);
223         partial_temp_layer2(0)(20) <= partial_temp_layer1(0)(20);
224         partial_temp_layer2(2)(20) <= partial_temp_layer1(2)(20);
225 
226 
227         -- custom connection
228         partial_temp_layer2(0)(15) <= partial_temp_layer1(1)(15);
229         partial_temp_layer2(1)(16) <= '1';
230 
231         partial_temp_layer2(0)(17) <= partial_temp_layer1(1)(17);
232         partial_temp_layer2(1)(17) <= '1';
233         partial_temp_layer2(1)(18) <= '1';
234 
235         partial_temp_layer2(0)(18) <= partial_temp_layer1(0)(18);
236         partial_temp_layer2(1)(19) <= '1';
```

```
237
238         partial_temp_layer2(0)(19) <= '0';
239         partial_temp_layer2(1)(20) <= '1';
240
241
242
243
244
245         -- layer 3
246         partial_temp_layer3(1)(6) <= '0';
247         -- FA
248         fa3_3 : fa port map (a => partial_temp_layer2(0)(6),
249                              b => partial_temp_layer2(1)(6),
250                              cin => partial_temp_layer2(2)(6),
251                              cout => partial_temp_layer3(1)(7),
252                              s => partial_temp_layer3(0)(6));
253
254         fa3_4 : fa port map (a => partial_temp_layer2(0)(7),
255                              b => partial_temp_layer2(1)(7),
256                              cin => partial_temp_layer2(2)(7),
257                              cout => partial_temp_layer3(1)(8),
258                              s => partial_temp_layer3(0)(7));
259
260         fa3_5 : fa port map (a => partial_temp_layer2(0)(8),
261                              b => partial_temp_layer2(1)(8),
262                              cin => partial_temp_layer2(2)(8),
263                              cout => partial_temp_layer3(1)(9),
264                              s => partial_temp_layer3(0)(8));
265
266         fa3_6 : fa port map (a => partial_temp_layer2(0)(9),
267                              b => partial_temp_layer2(1)(9),
268                              cin => partial_temp_layer2(2)(9),
269                              cout => partial_temp_layer3(1)(10),
270                              s => partial_temp_layer3(0)(9));
271
272         fa3_7 : fa port map (a => partial_temp_layer2(0)(10),
273                              b => partial_temp_layer2(1)(10),
274                              cin => partial_temp_layer2(2)(10),
275                              cout => partial_temp_layer3(1)(11),
276                              s => partial_temp_layer3(0)(10));
277
278         fa3_8 : fa port map (a => partial_temp_layer2(0)(11),
279                              b => partial_temp_layer2(1)(11),
280                              cin => partial_temp_layer2(2)(11),
281                              cout => partial_temp_layer3(1)(12),
282                              s => partial_temp_layer3(0)(11));
283
284         fa3_9 : fa port map (a => partial_temp_layer2(0)(12),
285                              b => partial_temp_layer2(1)(12),
286                              cin => partial_temp_layer2(2)(12),
287                              cout => partial_temp_layer3(1)(13),
288                              s => partial_temp_layer3(0)(12));
289
290         fa3_10 : fa port map (a => partial_temp_layer2(0)(13),
291                               b => partial_temp_layer2(1)(13),
292                               cin => partial_temp_layer2(2)(13),
293                               cout => partial_temp_layer3(1)(14),
294                               s => partial_temp_layer3(0)(13));
295
296         fa3_11 : fa port map (a => partial_temp_layer2(0)(14),
297                               b => partial_temp_layer2(1)(14),
298                               cin => partial_temp_layer2(2)(14),
```

```
299                             cout => partial_temp_layer3(1)(15),
300                             s => partial_temp_layer3(0)(14));
301
302         fa3_12 : fa port map (a => partial_temp_layer2(0)(15),
303                             b => partial_temp_layer2(1)(15),
304                             cin => partial_temp_layer2(2)(15),
305                             cout => partial_temp_layer3(1)(16),
306                             s => partial_temp_layer3(0)(15));
307
308         fa3_13 : fa port map (a => partial_temp_layer2(0)(17),
309                             b => partial_temp_layer2(1)(17),
310                             cin => partial_temp_layer2(2)(17),
311                             cout => partial_temp_layer3(1)(18),
312                             s => partial_temp_layer3(0)(17));
313
314         fa3_14 : fa port map (a => partial_temp_layer2(0)(18),
315                             b => partial_temp_layer2(1)(18),
316                             cin => partial_temp_layer2(2)(18),
317                             cout => partial_temp_layer3(1)(19),
318                             s => partial_temp_layer3(0)(18));
319
320         fa3_15 : fa port map (a => partial_temp_layer2(0)(16),
321                             b => partial_temp_layer2(1)(16),
322                             cin => partial_temp_layer2(2)(16),
323                             cout => partial_temp_layer3(1)(17),
324                             s => partial_temp_layer3(0)(16));
325
326
327
328         -- propagation
329         partial_temp_layer3(0)(20) <= partial_temp_layer2(0)(20);
330
331         -- custom connections
332         -- partial_temp_layer3(0)(16) <= partial_temp_layer2(2)(16);
333         -- partial_temp_layer3(1)(17) <= '1';
334
335         partial_temp_layer3(0)(19) <= '0';
336         partial_temp_layer3(1)(20) <= '1';
337
338         partial_temp_layer3(1)(21) <= '1';
339
340         -- cut down version of 23 bits to 20
341         out1(5 downto 0) <= (others => '0');
342         out2(5 downto 0) <= (others => '0');
343         out1(2 * Nbit - 1 downto 6) <= partial_temp_layer3(0)(2 * Nbit - 1 downto 6);
344         out2(2 * Nbit - 1 downto 6) <= partial_temp_layer3(1)(2 * Nbit - 1 downto 6);
345
346     end behav;
```

B.5 Fully-approximated Dadda tree using 4-2 compressors

This section contains the code for the fully-approximated Dadda tree with 4-2 compressors in the second layer, which allow reducing the numbers of Dadda layers from 3 to 2.

B.5.1 Partial Product Generation - MBE

./Code/mbe.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity mbe is
    port (a, b : in signed(Nbit - 1 downto 0);
          partial : out partial_array);
end mbe;

architecture behav of mbe is
    signal q, p_temp : multiple_factoring_array;
    signal extended_b : signed(Nbit downto 0);
    begin
        -- zero right extension for index -1
        extended_b <= b & '0';

        -- q generation from MBE
        multiple_factoring : for i in 0 to Nbit / 2 - 1 generate
            q(i) <= a(Nbit -1) & a when (extended_b(2 * i + 1) xor extended_b(2 * i))
    = '1' else
                    a & '0' when ((not (extended_b(2 * i + 1) xor extended_b(2 * i)))
    and (extended_b(2 * i + 2) xor extended_b(2 * i + 1))) = '1' else
                    (others => '0');
        end generate;

        partial_product : for i in 0 to Nbit / 2 - 1 generate
            -- xor 1 bit for MBE
            p_temp_gen : for j in 0 to Nbit generate
                p_temp(i)(j) <= q(i)(j) xor b(2 * i + 1);
            end generate;
            -- partial product without sign for Roorda
            partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto 0);
            -- Roorda extension
            partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
            -- Zero padding on the right
            if_padding : if i > 0 generate
                partial(i)(2 * i - 1 downto 0) <= (others => '0');
            end generate;
            -- MBE carry
            --p(N / 2)(2 * i) <= b(2 * i + 1);
            -- Roorda carry
            --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
            -- these last two could be compacted in one, MBE carry should be
            -- max in position N - 1, while Roorda carry starts from position
            -- N
            -- MBE + Roorda carry in same line
            partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
            partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
            partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
            partial(Nbit / 2)(2 * i + 1) <= '0';
        end generate;
    end behav;
```

B.5.2 Partial Product Reduction - Dadda tree

./Code/dadda_four_to_two_approx_layer2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity dadda_four_to_two_approx_layer2 is
    port (partial : in partial_array;
          out1, out2 : out signed(2 * Nbit - 1 downto 0));
end dadda_four_to_two_approx_layer2;

architecture behav of dadda_four_to_two_approx_layer2 is
    component fa
        port (cin, a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component ha
        port (a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component four_to_two_approx
        port (a, b, c, d : in std_logic;
              cout, s : out std_logic);
    end component;

    signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
    signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
    signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 4 downto 0);

    begin
        partial_association : for i in 0 to Nbit / 2 generate
            partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
            partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
        end generate;

        -- first stage --> from 6 to 4
        -- HA --> carry in 0 and sum in 1
        ha1_1 : ha port map (a => partial_temp(0)(6),
                             b => partial_temp(1)(6),
                             cout => partial_temp_layer1(0)(7),
                             s => partial_temp_layer1(1)(6));
        ha1_2 : ha port map (a => partial_temp(0)(7),
                             b => partial_temp(1)(7),
                             cout => partial_temp_layer1(0)(8),
                             s => partial_temp_layer1(1)(7));
        ha1_3 : ha port map (a => partial_temp(3)(8),
                             b => partial_temp(4)(8),
                             cout => partial_temp_layer1(0)(9),
                             s => partial_temp_layer1(1)(8));
        ha1_4 : ha port map (a => partial_temp(3)(9),
                             b => partial_temp(4)(9),
                             cout => partial_temp_layer1(0)(10),
                             s => partial_temp_layer1(1)(9));
        ha1_5 : ha port map (a => partial_temp(3)(11),
                             b => partial_temp(4)(11),
                             cout => partial_temp_layer1(0)(12),
                             s => partial_temp_layer1(1)(11));
        ha1_6 : ha port map (a => partial_temp(3)(15),
```

```
62                              b => partial_temp(4)(15),
63                              cout => partial_temp_layer1(0)(16),
64                              s => partial_temp_layer1(1)(15));
65          ha1_7 : ha port map (a => partial_temp(3)(13),
66                              b => partial_temp(4)(13),
67                              cout => partial_temp_layer1(0)(14),
68                              s => partial_temp_layer1(1)(13));
69          -- FA --> carry in 2/0 and sum in 3/1
70          fa1_1 : fa port map (a => partial_temp(0)(8),
71                              b => partial_temp(1)(8),
72                              cin => partial_temp(2)(8),
73                              cout => partial_temp_layer1(2)(9),
74                              s => partial_temp_layer1(3)(8));
75          fa1_2 : fa port map (a => partial_temp(0)(9),
76                              b => partial_temp(1)(9),
77                              cin => partial_temp(2)(9),
78                              cout => partial_temp_layer1(2)(10),
79                              s => partial_temp_layer1(3)(9));
80          fa1_3 : fa port map (a => partial_temp(0)(10),
81                              b => partial_temp(1)(10),
82                              cin => partial_temp(2)(10),
83                              cout => partial_temp_layer1(2)(11),
84                              s => partial_temp_layer1(3)(10));
85          fa1_4 : fa port map (a => partial_temp(0)(11),
86                              b => partial_temp(1)(11),
87                              cin => partial_temp(2)(11),
88                              cout => partial_temp_layer1(2)(12),
89                              s => partial_temp_layer1(3)(11));
90          fa1_5 : fa port map (a => partial_temp(3)(10),
91                              b => partial_temp(4)(10),
92                              cin => partial_temp(5)(10),
93                              cout => partial_temp_layer1(0)(11),
94                              s => partial_temp_layer1(1)(10)); -- row 3 already taken
    by FA, free for HA
95          fa1_6 : fa port map (a => partial_temp(3)(12),
96                              b => partial_temp(4)(12),
97                              cin => partial_temp(5)(12),
98                              cout => partial_temp_layer1(2)(13),
99                              s => partial_temp_layer1(3)(12));
100         fa1_7 : fa port map (a => partial_temp(3)(14),
101                             b => partial_temp(4)(14),
102                             cin => partial_temp(5)(14),
103                             cout => partial_temp_layer1(2)(15),
104                             s => partial_temp_layer1(3)(14));
105         fa1_8 : fa port map (a => partial_temp(3)(16),
106                             b => partial_temp(4)(16),
107                             cin => partial_temp(5)(16),
108                             cout => partial_temp_layer1(2)(17),
109                             s => partial_temp_layer1(3)(16));
110         -- propagations
111         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
112         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
113         partial_temp_layer1(1)(5 downto 2) <= partial_temp(1)(5 downto 2);
114         partial_temp_layer1(2)(7) <= partial_temp(2)(7);
115         --partial_temp_layer1(2)(6) <= partial_temp(2)(6);
116         partial_temp_layer1(2)(5) <= partial_temp(2)(5);
117         partial_temp_layer1(2)(4) <= partial_temp(2)(4);
118         partial_temp_layer1(3)(7) <= partial_temp(3)(7);
119         partial_temp_layer1(2)(6) <= partial_temp(3)(6); -- because 3 will be used for
    Roorda and MBE carries
120         partial_temp_layer1(0)(6) <= partial_temp(2)(6);
121         partial_temp_layer1(3)(6) <= partial_temp(5)(6);
```

```
122         partial_temp_layer1(2)(8) <= partial_temp(5)(8); -- used by FA
123         roorda_mbe_carries : for i in 0 to 3 generate
124             partial_temp_layer1(3)(2 * i) <= partial_temp(5)(2 * i);
125         end generate;
126         -- custom connections
127         -- positioned in HA sum 1
128         -- three ones
129         partial_temp_layer1(1)(14) <= '1';
130         partial_temp_layer1(1)(16) <= '1';
131         partial_temp_layer1(1)(19) <= '1';
132         -- two ones
133         partial_temp_layer1(1)(18) <= partial_temp(5)(18);
134         --partial_temp_layer1(1)(12) <= partial_temp(3)(12);
135         -- one one
136         partial_temp_layer1(1)(17) <= not partial_temp(4)(17);
137         -- position in HA carry 0
138         -- three ones
139         partial_temp_layer1(0)(15) <= '1';
140         partial_temp_layer1(0)(17) <= '1';
141         -- two ones
142         partial_temp_layer1(0)(19) <= '1';
143         partial_temp_layer1(0)(13) <= '1';
144         partial_temp_layer1(0)(20) <= '1';
145         -- one one
146         partial_temp_layer1(0)(18) <= partial_temp(4)(17);
147         -- positioned in FA sum 3
148         -- three ones
149         partial_temp_layer1(3)(15) <= '1';
150         partial_temp_layer1(3)(17) <= '1';
151         partial_temp_layer1(3)(18) <= '1';
152         -- two ones
153         partial_temp_layer1(1)(12) <= partial_temp(2)(12); -- taken by FA
154         partial_temp_layer1(3)(13) <= partial_temp(2)(13);
155         -- positioned in FA carry 2
156         -- three ones
157         partial_temp_layer1(2)(16) <= '1';
158         partial_temp_layer1(2)(18) <= '1';
159         partial_temp_layer1(2)(19) <= '1';
160         -- two ones
161         partial_temp_layer1(2)(14) <= partial_temp(2)(14);
162         partial_temp_layer1(2)(20) <= '1';
163
164
165
166
167
168
169
170
171
172         -- layer 2
173         -- HA --> sum 0 carry 1
174         ha2_1 : ha port map (a => partial_temp_layer1(0)(2),
175                              b => partial_temp_layer1(1)(2),
176                              cout => partial_temp_layer2(1)(3),
177                              s => partial_temp_layer2(0)(2));
178         ha2_2 : ha port map (a => partial_temp_layer1(0)(3),
179                              b => partial_temp_layer1(1)(3),
180                              cout => partial_temp_layer2(1)(4),
181                              s => partial_temp_layer2(0)(3));
182         -- FA --> sum 0 carry 1
183         fa2_1 : fa port map (a => partial_temp_layer1(0)(5),
```

```
                                b => partial_temp_layer1(1)(5),
                                cin => partial_temp_layer1(2)(5),
                                cout => partial_temp_layer2(1)(6),
                                s => partial_temp_layer2(0)(5));
        -- 4to2 --> sum 0 carry 1
        four_to_two_2_1 : four_to_two_approx port map (
                        a => partial_temp_layer1(0)(4),
                        b => partial_temp_layer1(1)(4),
                        c => partial_temp_layer1(2)(4),
                        d => partial_temp_layer1(3)(4),
                        cout => partial_temp_layer2(1)(5),
                        s => partial_temp_layer2(0)(4)
        );

        four_to_two_gen : for i in 6 to 2 * N - 1 generate
            four_to_two_inst : four_to_two_approx port map (
                        a => partial_temp_layer1(0)(i),
                        b => partial_temp_layer1(1)(i),
                        c => partial_temp_layer1(2)(i),
                        d => partial_temp_layer1(3)(i),
                        cout => partial_temp_layer2(1)(i + 1),
                        s => partial_temp_layer2(0)(i)
                    );
        end generate;

        -- propagation
        partial_temp_layer2(0)(0) <= partial_temp_layer1(0)(0);
        partial_temp_layer2(1)(0) <= partial_temp_layer1(3)(0);

        partial_temp_layer2(0)(1) <= partial_temp_layer1(0)(1);

        partial_temp_layer2(1)(2) <= partial_temp_layer1(3)(2);

        --partial_temp_layer2(1)(6) <= partial_temp_layer1(3)(6);

        -- custom connection
        partial_temp_layer2(0)(2 * N) <= '1';
        partial_temp_layer2(1)(2 * N + 1) <= '1';

        partial_temp_layer2(0)(2 * N + 1) <= '0';
        partial_temp_layer2(1)(2 * N + 2) <= '1';

        -- to avoid errors
        partial_temp_layer2(1)(1) <= '0';

        -- cut down version of 23 bits to 20
        out1 <= partial_temp_layer2(0)(2 * Nbit - 1 downto 0);
        out2 <= partial_temp_layer2(1)(2 * Nbit - 1 downto 0);

    end behav;
```

B.6 Fully-approximated Dadda tree using AMBE

This section contains the code for the Dadda tree with Approximated MBE as partial production generation.

B.6.1 Partial Product Generation - AMBE

./Code/ambe.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity ambe is
    port (a, b : in signed(Nbit - 1 downto 0);
          partial : out partial_array);
end ambe;

architecture behav of ambe is
    signal q, p_temp : multiple_factoring_array;
    signal extended_b : signed(Nbit downto 0);
    begin
        -- zero right extension for index -1
        extended_b <= b & '0';

        partial_product : for i in 0 to Nbit / 2 - 1 generate
            -- xor 1 bit for MBE
            p_temp_gen : for j in 0 to Nbit - 1 generate
                p_temp(i)(j) <= (a(j) xor extended_b(2 * i + 1)) and (extended_b(2 * i
    ) or extended_b(2 * i + 1));
            end generate;
            -- sign extension
            p_temp(i)(Nbit) <= p_temp(i)(Nbit - 1);
            -- partial product without sign for Roorda
            partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto 0);
            -- Roorda extension
            partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
            -- Zero padding on the right
            if_padding : if i > 0 generate
                partial(i)(2 * i - 1 downto 0) <= (others => '0');
            end generate;
            -- MBE carry
            --p(N / 2)(2 * i) <= b(2 * i + 1);
            -- Roorda carry
            --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
            -- these last two could be compacted in one, MBE carry should be
            -- max in position N - 1, while Roorda carry starts from position
            -- N
            -- MBE + Roorda carry in same line
            partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
            partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
            partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
            partial(Nbit / 2)(2 * i + 1) <= '0';
        end generate;
    end behav;
```

B.6.2 Partial Product Reduction - Dadda tree

./Code/dadda.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
```

```
6  use work.util.all;
7
8  entity dadda is
9      port (partial : in partial_array;
10           out1, out2 : out signed(2 * Nbit - 1 downto 0));
11 end dadda;
12
13 architecture behav of dadda is
14     component fa
15         port (cin, a, b : in std_logic;
16               cout, s : out std_logic);
17     end component;
18
19     component ha
20         port (a, b : in std_logic;
21               cout, s : out std_logic);
22     end component;
23
24     signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
25     signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
26     signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 3 downto 0);
27     signal partial_temp_layer3 : internal_partial_array(Nbit / 2 - 4 downto 0);
28
29     begin
30         partial_association : for i in 0 to Nbit / 2 generate
31             partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
32             partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
33         end generate;
34
35         -- first stage --> from 6 to 4
36         -- HA --> carry in 0 and sum in 1
37         ha1_1 : ha port map (a => partial_temp(0)(6),
38                              b => partial_temp(1)(6),
39                              cout => partial_temp_layer1(0)(7),
40                              s => partial_temp_layer1(1)(6));
41         ha1_2 : ha port map (a => partial_temp(0)(7),
42                              b => partial_temp(1)(7),
43                              cout => partial_temp_layer1(0)(8),
44                              s => partial_temp_layer1(1)(7));
45         ha1_3 : ha port map (a => partial_temp(3)(8),
46                              b => partial_temp(4)(8),
47                              cout => partial_temp_layer1(0)(9),
48                              s => partial_temp_layer1(1)(8));
49         ha1_4 : ha port map (a => partial_temp(3)(9),
50                              b => partial_temp(4)(9),
51                              cout => partial_temp_layer1(0)(10),
52                              s => partial_temp_layer1(1)(9));
53         ha1_5 : ha port map (a => partial_temp(3)(11),
54                              b => partial_temp(4)(11),
55                              cout => partial_temp_layer1(0)(12),
56                              s => partial_temp_layer1(1)(11));
57         ha1_6 : ha port map (a => partial_temp(3)(15),
58                              b => partial_temp(4)(15),
59                              cout => partial_temp_layer1(0)(16),
60                              s => partial_temp_layer1(1)(15));
61         ha1_7 : ha port map (a => partial_temp(3)(13),
62                              b => partial_temp(4)(13),
63                              cout => partial_temp_layer1(0)(14),
64                              s => partial_temp_layer1(1)(13));
65         -- FA --> carry in 2/0 and sum in 3/1
66         fa1_1 : fa port map (a => partial_temp(0)(8),
67                              b => partial_temp(1)(8),
```

```
68                              cin => partial_temp(2)(8),
69                              cout => partial_temp_layer1(2)(9),
70                              s => partial_temp_layer1(3)(8));
71          fa1_2 : fa port map (a => partial_temp(0)(9),
72                              b => partial_temp(1)(9),
73                              cin => partial_temp(2)(9),
74                              cout => partial_temp_layer1(2)(10),
75                              s => partial_temp_layer1(3)(9));
76          fa1_3 : fa port map (a => partial_temp(0)(10),
77                              b => partial_temp(1)(10),
78                              cin => partial_temp(2)(10),
79                              cout => partial_temp_layer1(2)(11),
80                              s => partial_temp_layer1(3)(10));
81          fa1_4 : fa port map (a => partial_temp(0)(11),
82                              b => partial_temp(1)(11),
83                              cin => partial_temp(2)(11),
84                              cout => partial_temp_layer1(2)(12),
85                              s => partial_temp_layer1(3)(11));
86          fa1_5 : fa port map (a => partial_temp(3)(10),
87                              b => partial_temp(4)(10),
88                              cin => partial_temp(5)(10),
89                              cout => partial_temp_layer1(0)(11),
90                              s => partial_temp_layer1(1)(10)); -- row 3 already taken
    by FA, free for HA
91          fa1_6 : fa port map (a => partial_temp(3)(12),
92                              b => partial_temp(4)(12),
93                              cin => partial_temp(5)(12),
94                              cout => partial_temp_layer1(2)(13),
95                              s => partial_temp_layer1(3)(12));
96          fa1_7 : fa port map (a => partial_temp(3)(14),
97                              b => partial_temp(4)(14),
98                              cin => partial_temp(5)(14),
99                              cout => partial_temp_layer1(2)(15),
100                             s => partial_temp_layer1(3)(14));
101         fa1_8 : fa port map (a => partial_temp(3)(16),
102                             b => partial_temp(4)(16),
103                             cin => partial_temp(5)(16),
104                             cout => partial_temp_layer1(2)(17),
105                             s => partial_temp_layer1(3)(16));
106         -- propagations
107         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
108         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
109         partial_temp_layer1(1)(5 downto 2) <= partial_temp(1)(5 downto 2);
110         partial_temp_layer1(2)(7) <= partial_temp(2)(7);
111         --partial_temp_layer1(2)(6) <= partial_temp(2)(6);
112         partial_temp_layer1(2)(5) <= partial_temp(2)(5);
113         partial_temp_layer1(2)(4) <= partial_temp(2)(4);
114         partial_temp_layer1(3)(7) <= partial_temp(3)(7);
115         partial_temp_layer1(2)(6) <= partial_temp(3)(6); -- because 3 will be used for
    Roorda and MBE carries
116         partial_temp_layer1(0)(6) <= partial_temp(2)(6);
117         partial_temp_layer1(3)(6) <= partial_temp(5)(6);
118         partial_temp_layer1(2)(8) <= partial_temp(5)(8); -- used by FA
119         roorda_mbe_carries : for i in 0 to 3 generate
120             partial_temp_layer1(3)(2 * i) <= partial_temp(5)(2 * i);
121         end generate;
122         -- custom connections
123         -- positioned in HA sum 1
124         -- three ones
125         partial_temp_layer1(1)(14) <= '1';
126         partial_temp_layer1(1)(16) <= '1';
127         partial_temp_layer1(1)(19) <= '1';
```

```
128         -- two ones
129         partial_temp_layer1(1)(18) <= partial_temp(5)(18);
130         --partial_temp_layer1(1)(12) <= partial_temp(3)(12);
131         -- one one
132         partial_temp_layer1(1)(17) <= not partial_temp(4)(17);
133         -- position in HA carry 0
134         -- three ones
135         partial_temp_layer1(0)(15) <= '1';
136         partial_temp_layer1(0)(17) <= '1';
137         -- two ones
138         partial_temp_layer1(0)(19) <= '1';
139         partial_temp_layer1(0)(13) <= '1';
140         partial_temp_layer1(0)(20) <= '1';
141         -- one one
142         partial_temp_layer1(0)(18) <= partial_temp(4)(17);
143         -- positioned in FA sum 3
144         -- three ones
145         partial_temp_layer1(3)(15) <= '1';
146         partial_temp_layer1(3)(17) <= '1';
147         partial_temp_layer1(3)(18) <= '1';
148         -- two ones
149         partial_temp_layer1(1)(12) <= partial_temp(2)(12); -- taken by FA
150         partial_temp_layer1(3)(13) <= partial_temp(2)(13);
151         -- positioned in FA carry 2
152         -- three ones
153         partial_temp_layer1(2)(16) <= '1';
154         partial_temp_layer1(2)(18) <= '1';
155         partial_temp_layer1(2)(19) <= '1';
156         -- two ones
157         partial_temp_layer1(2)(14) <= partial_temp(2)(14);
158         partial_temp_layer1(2)(20) <= '1';
159
160
161
162
163
164
165
166         -- layer 2
167         -- HA --> sum 0 carry 1
168         ha2_1 : ha port map (a => partial_temp_layer1(0)(4),
169                              b => partial_temp_layer1(3)(4),
170                              cout => partial_temp_layer2(1)(5),
171                              s => partial_temp_layer2(0)(4));
172         ha2_2 : ha port map (a => partial_temp_layer1(0)(5),
173                              b => partial_temp_layer1(1)(5),
174                              cout => partial_temp_layer2(1)(6),
175                              s => partial_temp_layer2(0)(5));
176         -- ha2_4 : ha port map (a => partial_temp_layer1(0)(16),
177         --                      b => partial_temp_layer1(1)(16),
178         --                      cout => partial_temp_layer2(1)(17),
179         --                      s => partial_temp_layer2(0)(16));
180         -- FA --> sum 0 carry 1
181         fa2_1 : fa port map (a => partial_temp_layer1(0)(6),
182                              b => partial_temp_layer1(1)(6),
183                              cin => partial_temp_layer1(3)(6),
184                              cout => partial_temp_layer2(1)(7),
185                              s => partial_temp_layer2(0)(6));
186         fa2_2 : fa port map (a => partial_temp_layer1(0)(7),
187                              b => partial_temp_layer1(1)(7),
188                              cin => partial_temp_layer1(3)(7),
189                              cout => partial_temp_layer2(1)(8),
```

```
190                             s => partial_temp_layer2(0)(7));
191         fa2_3 : fa port map (a => partial_temp_layer1(0)(8),
192                             b => partial_temp_layer1(1)(8),
193                             cin => partial_temp_layer1(2)(8),
194                             cout => partial_temp_layer2(1)(9),
195                             s => partial_temp_layer2(0)(8));
196         fa2_4 : fa port map (a => partial_temp_layer1(0)(9),
197                             b => partial_temp_layer1(1)(9),
198                             cin => partial_temp_layer1(2)(9),
199                             cout => partial_temp_layer2(1)(10),
200                             s => partial_temp_layer2(0)(9));
201         fa2_5 : fa port map (a => partial_temp_layer1(0)(10),
202                             b => partial_temp_layer1(1)(10),
203                             cin => partial_temp_layer1(2)(10),
204                             cout => partial_temp_layer2(1)(11),
205                             s => partial_temp_layer2(0)(10));
206         fa2_6 : fa port map (a => partial_temp_layer1(0)(11),
207                             b => partial_temp_layer1(1)(11),
208                             cin => partial_temp_layer1(2)(11),
209                             cout => partial_temp_layer2(1)(12),
210                             s => partial_temp_layer2(0)(11));
211         fa2_7 : fa port map (a => partial_temp_layer1(0)(12),
212                             b => partial_temp_layer1(1)(12),
213                             cin => partial_temp_layer1(2)(12),
214                             cout => partial_temp_layer2(1)(13),
215                             s => partial_temp_layer2(0)(12));
216         fa2_8 : fa port map (a => partial_temp_layer1(0)(14),
217                             b => partial_temp_layer1(2)(14),
218                             cin => partial_temp_layer1(3)(14),
219                             cout => partial_temp_layer2(1)(15),
220                             s => partial_temp_layer2(0)(14));
221         fa2_9 : fa port map (a => partial_temp_layer1(2)(13),
222                             b => partial_temp_layer1(3)(13),
223                             cin => partial_temp_layer1(1)(13),
224                             cout => partial_temp_layer2(1)(14),
225                             s => partial_temp_layer2(0)(13));
226 
227         -- propagation
228         partial_temp_layer2(0)(3 downto 0) <= partial_temp_layer1(0)(3 downto 0);
229         partial_temp_layer2(1)(0) <= partial_temp_layer1(3)(0);
230         partial_temp_layer2(1)(4 downto 2) <= partial_temp_layer1(1)(4 downto 2);
231         partial_temp_layer2(2)(2) <= partial_temp_layer1(3)(2);
232         partial_temp_layer2(2)(7 downto 4) <= partial_temp_layer1(2)(7 downto 4);
233         partial_temp_layer2(2)(12 downto 8) <= partial_temp_layer1(3)(12 downto 8);
234         partial_temp_layer2(2)(14 downto 13) <= (others => '1');
235         partial_temp_layer2(2)(15) <= partial_temp_layer1(2)(15);
236         --partial_temp_layer2(0)(16) <= partial_temp_layer1(1)(16);
237         partial_temp_layer2(0)(16) <= partial_temp_layer1(0)(16);
238         partial_temp_layer2(2)(16) <= partial_temp_layer1(3)(16);
239         --partial_temp_layer2(1)(17) <= partial_temp_layer1(1)(17);
240         partial_temp_layer2(2)(17) <= partial_temp_layer1(2)(17);
241         partial_temp_layer2(2)(18) <= partial_temp_layer1(1)(18);
242         partial_temp_layer2(2)(19) <= partial_temp_layer1(2)(19);
243         partial_temp_layer2(0)(20) <= partial_temp_layer1(0)(20);
244         partial_temp_layer2(2)(20) <= partial_temp_layer1(2)(20);
245 
246 
247         -- custom connection
248         partial_temp_layer2(0)(15) <= partial_temp_layer1(1)(15);
249         partial_temp_layer2(1)(16) <= '1';
250 
251         partial_temp_layer2(0)(17) <= partial_temp_layer1(1)(17);
```

```
252         partial_temp_layer2(1)(17) <= '1';
253         partial_temp_layer2(1)(18) <= '1';
254
255         partial_temp_layer2(0)(18) <= partial_temp_layer1(0)(18);
256         partial_temp_layer2(1)(19) <= '1';
257
258         partial_temp_layer2(0)(19) <= '0';
259         partial_temp_layer2(1)(20) <= '1';
260
261
262
263
264
265         -- layer 3
266         -- HA
267         ha3_1 : ha port map (a => partial_temp_layer2(0)(2),
268                              b => partial_temp_layer2(1)(2),
269                              cout => partial_temp_layer3(1)(3),
270                              s => partial_temp_layer3(0)(2));
271         ha3_2 : ha port map (a => partial_temp_layer2(0)(3),
272                              b => partial_temp_layer2(1)(3),
273                              cout => partial_temp_layer3(1)(4),
274                              s => partial_temp_layer3(0)(3));
275
276         -- FA
277         fa3_1 : fa port map (a => partial_temp_layer2(0)(4),
278                              b => partial_temp_layer2(1)(4),
279                              cin => partial_temp_layer2(2)(4),
280                              cout => partial_temp_layer3(1)(5),
281                              s => partial_temp_layer3(0)(4));
282
283         fa3_2 : fa port map (a => partial_temp_layer2(0)(5),
284                              b => partial_temp_layer2(1)(5),
285                              cin => partial_temp_layer2(2)(5),
286                              cout => partial_temp_layer3(1)(6),
287                              s => partial_temp_layer3(0)(5));
288
289         fa3_3 : fa port map (a => partial_temp_layer2(0)(6),
290                              b => partial_temp_layer2(1)(6),
291                              cin => partial_temp_layer2(2)(6),
292                              cout => partial_temp_layer3(1)(7),
293                              s => partial_temp_layer3(0)(6));
294
295         fa3_4 : fa port map (a => partial_temp_layer2(0)(7),
296                              b => partial_temp_layer2(1)(7),
297                              cin => partial_temp_layer2(2)(7),
298                              cout => partial_temp_layer3(1)(8),
299                              s => partial_temp_layer3(0)(7));
300
301         fa3_5 : fa port map (a => partial_temp_layer2(0)(8),
302                              b => partial_temp_layer2(1)(8),
303                              cin => partial_temp_layer2(2)(8),
304                              cout => partial_temp_layer3(1)(9),
305                              s => partial_temp_layer3(0)(8));
306
307         fa3_6 : fa port map (a => partial_temp_layer2(0)(9),
308                              b => partial_temp_layer2(1)(9),
309                              cin => partial_temp_layer2(2)(9),
310                              cout => partial_temp_layer3(1)(10),
311                              s => partial_temp_layer3(0)(9));
312
313         fa3_7 : fa port map (a => partial_temp_layer2(0)(10),
```

```
314                         b => partial_temp_layer2(1)(10),
315                         cin => partial_temp_layer2(2)(10),
316                         cout => partial_temp_layer3(1)(11),
317                         s => partial_temp_layer3(0)(10));
318
319         fa3_8 : fa port map (a => partial_temp_layer2(0)(11),
320                         b => partial_temp_layer2(1)(11),
321                         cin => partial_temp_layer2(2)(11),
322                         cout => partial_temp_layer3(1)(12),
323                         s => partial_temp_layer3(0)(11));
324
325         fa3_9 : fa port map (a => partial_temp_layer2(0)(12),
326                         b => partial_temp_layer2(1)(12),
327                         cin => partial_temp_layer2(2)(12),
328                         cout => partial_temp_layer3(1)(13),
329                         s => partial_temp_layer3(0)(12));
330
331         fa3_10 : fa port map (a => partial_temp_layer2(0)(13),
332                          b => partial_temp_layer2(1)(13),
333                          cin => partial_temp_layer2(2)(13),
334                          cout => partial_temp_layer3(1)(14),
335                          s => partial_temp_layer3(0)(13));
336
337         fa3_11 : fa port map (a => partial_temp_layer2(0)(14),
338                          b => partial_temp_layer2(1)(14),
339                          cin => partial_temp_layer2(2)(14),
340                          cout => partial_temp_layer3(1)(15),
341                          s => partial_temp_layer3(0)(14));
342
343         fa3_12 : fa port map (a => partial_temp_layer2(0)(15),
344                          b => partial_temp_layer2(1)(15),
345                          cin => partial_temp_layer2(2)(15),
346                          cout => partial_temp_layer3(1)(16),
347                          s => partial_temp_layer3(0)(15));
348
349         fa3_13 : fa port map (a => partial_temp_layer2(0)(17),
350                          b => partial_temp_layer2(1)(17),
351                          cin => partial_temp_layer2(2)(17),
352                          cout => partial_temp_layer3(1)(18),
353                          s => partial_temp_layer3(0)(17));
354
355         fa3_14 : fa port map (a => partial_temp_layer2(0)(18),
356                          b => partial_temp_layer2(1)(18),
357                          cin => partial_temp_layer2(2)(18),
358                          cout => partial_temp_layer3(1)(19),
359                          s => partial_temp_layer3(0)(18));
360
361         fa3_15 : fa port map (a => partial_temp_layer2(0)(16),
362                          b => partial_temp_layer2(1)(16),
363                          cin => partial_temp_layer2(2)(16),
364                          cout => partial_temp_layer3(1)(17),
365                          s => partial_temp_layer3(0)(16));
366
367
368
369         -- propagation
370         partial_temp_layer3(0)(0) <= partial_temp_layer2(0)(0);
371         partial_temp_layer3(1)(0) <= partial_temp_layer2(1)(0);
372
373         partial_temp_layer3(0)(1) <= partial_temp_layer2(0)(1);
374
375         -- partial_temp_layer3(1)(1) <= partial_temp_layer2(2)(1);
```

```
376         partial_temp_layer3(1)(1) <= '0'; -- to avoid U
377         partial_temp_layer3(1)(2) <= partial_temp_layer2(2)(2);
378 
379         partial_temp_layer3(0)(20) <= partial_temp_layer2(0)(20);
380 
381         -- custom connections
382         -- partial_temp_layer3(0)(16) <= partial_temp_layer2(2)(16);
383         -- partial_temp_layer3(1)(17) <= '1';
384 
385         partial_temp_layer3(0)(19) <= '0';
386         partial_temp_layer3(1)(20) <= '1';
387 
388         partial_temp_layer3(1)(21) <= '1';
389 
390         -- cut down version of 23 bits to 20
391         out1 <= partial_temp_layer3(0)(2 * Nbit - 1 downto 0);
392         out2 <= partial_temp_layer3(1)(2 * Nbit - 1 downto 0);
393 
394     end behav;
```

B.7 Fully-approximated Dadda tree using AMBE and 4-2 compressors

This section contains the code for the fully-approximated Dadda tree with Approximated MBE as partial production generation, along with the whole second layer implemented using 4-2 compressors.

B.7.1 Partial Product Generation - AMBE

./Code/ambe.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  
5  library work;
6  use work.util.all;
7  
8  entity ambe is
9      port (a, b : in signed(Nbit - 1 downto 0);
10           partial : out partial_array);
11 end ambe;
12 
13 architecture behav of ambe is
14     signal q, p_temp : multiple_factoring_array;
15     signal extended_b : signed(Nbit downto 0);
16     begin
17         -- zero right extension for index -1
18         extended_b <= b & '0';
19 
20         partial_product : for i in 0 to Nbit / 2 - 1 generate
21             -- xor 1 bit for MBE
22             p_temp_gen : for j in 0 to Nbit - 1 generate
23                 p_temp(i)(j) <= (a(j) xor extended_b(2 * i + 1)) and (extended_b(2 * i
       ) or extended_b(2 * i + 1));
24             end generate;
25             -- sign extension
26             p_temp(i)(Nbit) <= p_temp(i)(Nbit - 1);
```

```
            -- partial product without sign for Roorda
            partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto 0);
            -- Roorda extension
            partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
            -- Zero padding on the right
            if_padding : if i > 0 generate
                partial(i)(2 * i - 1 downto 0) <= (others => '0');
            end generate;
            -- MBE carry
            --p(N / 2)(2 * i) <= b(2 * i + 1);
            -- Roorda carry
            --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
            -- these last two could be compacted in one, MBE carry should be
            -- max in position N - 1, while Roorda carry starts from position
            -- N
            -- MBE + Roorda carry in same line
            partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
            partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
            partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
            partial(Nbit / 2)(2 * i + 1) <= '0';
        end generate;
    end behav;
```

B.7.2 Partial Product Reduction - Dadda tree

./Code/dadda_four_to_two_approx_layer2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity dadda_four_to_two_approx_layer2 is
    port (partial : in partial_array;
          out1, out2 : out signed(2 * Nbit - 1 downto 0));
end dadda_four_to_two_approx_layer2;

architecture behav of dadda_four_to_two_approx_layer2 is
    component fa
        port (cin, a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component ha
        port (a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component four_to_two_approx
        port (a, b, c, d : in std_logic;
              cout, s : out std_logic);
    end component;

    signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
    signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
    signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 4 downto 0);

    begin
```

```
34          partial_association : for i in 0 to Nbit / 2 generate
35              partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
36              partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
37          end generate;
38
39          -- first stage --> from 6 to 4
40          -- HA --> carry in 0 and sum in 1
41          ha1_1 : ha port map (a => partial_temp(0)(6),
42                               b => partial_temp(1)(6),
43                               cout => partial_temp_layer1(0)(7),
44                               s => partial_temp_layer1(1)(6));
45          ha1_2 : ha port map (a => partial_temp(0)(7),
46                               b => partial_temp(1)(7),
47                               cout => partial_temp_layer1(0)(8),
48                               s => partial_temp_layer1(1)(7));
49          ha1_3 : ha port map (a => partial_temp(3)(8),
50                               b => partial_temp(4)(8),
51                               cout => partial_temp_layer1(0)(9),
52                               s => partial_temp_layer1(1)(8));
53          ha1_4 : ha port map (a => partial_temp(3)(9),
54                               b => partial_temp(4)(9),
55                               cout => partial_temp_layer1(0)(10),
56                               s => partial_temp_layer1(1)(9));
57          ha1_5 : ha port map (a => partial_temp(3)(11),
58                               b => partial_temp(4)(11),
59                               cout => partial_temp_layer1(0)(12),
60                               s => partial_temp_layer1(1)(11));
61          ha1_6 : ha port map (a => partial_temp(3)(15),
62                               b => partial_temp(4)(15),
63                               cout => partial_temp_layer1(0)(16),
64                               s => partial_temp_layer1(1)(15));
65          ha1_7 : ha port map (a => partial_temp(3)(13),
66                               b => partial_temp(4)(13),
67                               cout => partial_temp_layer1(0)(14),
68                               s => partial_temp_layer1(1)(13));
69          -- FA --> carry in 2/0 and sum in 3/1
70          fa1_1 : fa port map (a => partial_temp(0)(8),
71                               b => partial_temp(1)(8),
72                               cin => partial_temp(2)(8),
73                               cout => partial_temp_layer1(2)(9),
74                               s => partial_temp_layer1(3)(8));
75          fa1_2 : fa port map (a => partial_temp(0)(9),
76                               b => partial_temp(1)(9),
77                               cin => partial_temp(2)(9),
78                               cout => partial_temp_layer1(2)(10),
79                               s => partial_temp_layer1(3)(9));
80          fa1_3 : fa port map (a => partial_temp(0)(10),
81                               b => partial_temp(1)(10),
82                               cin => partial_temp(2)(10),
83                               cout => partial_temp_layer1(2)(11),
84                               s => partial_temp_layer1(3)(10));
85          fa1_4 : fa port map (a => partial_temp(0)(11),
86                               b => partial_temp(1)(11),
87                               cin => partial_temp(2)(11),
88                               cout => partial_temp_layer1(2)(12),
89                               s => partial_temp_layer1(3)(11));
90          fa1_5 : fa port map (a => partial_temp(3)(10),
91                               b => partial_temp(4)(10),
92                               cin => partial_temp(5)(10),
93                               cout => partial_temp_layer1(0)(11),
94                               s => partial_temp_layer1(1)(10)); -- row 3 already taken
    by FA, free for HA
```

```
95          fa1_6 : fa port map (a => partial_temp(3)(12),
96                               b => partial_temp(4)(12),
97                               cin => partial_temp(5)(12),
98                               cout => partial_temp_layer1(2)(13),
99                               s => partial_temp_layer1(3)(12));
100         fa1_7 : fa port map (a => partial_temp(3)(14),
101                              b => partial_temp(4)(14),
102                              cin => partial_temp(5)(14),
103                              cout => partial_temp_layer1(2)(15),
104                              s => partial_temp_layer1(3)(14));
105         fa1_8 : fa port map (a => partial_temp(3)(16),
106                              b => partial_temp(4)(16),
107                              cin => partial_temp(5)(16),
108                              cout => partial_temp_layer1(2)(17),
109                              s => partial_temp_layer1(3)(16));
110         -- propagations
111         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
112         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
113         partial_temp_layer1(1)(5 downto 2) <= partial_temp(1)(5 downto 2);
114         partial_temp_layer1(2)(7) <= partial_temp(2)(7);
115         --partial_temp_layer1(2)(6) <= partial_temp(2)(6);
116         partial_temp_layer1(2)(5) <= partial_temp(2)(5);
117         partial_temp_layer1(2)(4) <= partial_temp(2)(4);
118         partial_temp_layer1(3)(7) <= partial_temp(3)(7);
119         partial_temp_layer1(2)(6) <= partial_temp(3)(6); -- because 3 will be used for
        Roorda and MBE carries
120         partial_temp_layer1(0)(6) <= partial_temp(2)(6);
121         partial_temp_layer1(3)(6) <= partial_temp(5)(6);
122         partial_temp_layer1(2)(8) <= partial_temp(5)(8); -- used by FA
123         roorda_mbe_carries : for i in 0 to 3 generate
124             partial_temp_layer1(3)(2 * i) <= partial_temp(5)(2 * i);
125         end generate;
126         -- custom connections
127         -- positioned in HA sum 1
128         -- three ones
129         partial_temp_layer1(1)(14) <= '1';
130         partial_temp_layer1(1)(16) <= '1';
131         partial_temp_layer1(1)(19) <= '1';
132         -- two ones
133         partial_temp_layer1(1)(18) <= partial_temp(5)(18);
134         --partial_temp_layer1(1)(12) <= partial_temp(3)(12);
135         -- one one
136         partial_temp_layer1(1)(17) <= not partial_temp(4)(17);
137         -- position in HA carry 0
138         -- three ones
139         partial_temp_layer1(0)(15) <= '1';
140         partial_temp_layer1(0)(17) <= '1';
141         -- two ones
142         partial_temp_layer1(0)(19) <= '1';
143         partial_temp_layer1(0)(13) <= '1';
144         partial_temp_layer1(0)(20) <= '1';
145         -- one one
146         partial_temp_layer1(0)(18) <= partial_temp(4)(17);
147         -- positioned in FA sum 3
148         -- three ones
149         partial_temp_layer1(3)(15) <= '1';
150         partial_temp_layer1(3)(17) <= '1';
151         partial_temp_layer1(3)(18) <= '1';
152         -- two ones
153         partial_temp_layer1(1)(12) <= partial_temp(2)(12); -- taken by FA
154         partial_temp_layer1(3)(13) <= partial_temp(2)(13);
155         -- positioned in FA carry 2
```

```
156         -- three ones
157         partial_temp_layer1(2)(16) <= '1';
158         partial_temp_layer1(2)(18) <= '1';
159         partial_temp_layer1(2)(19) <= '1';
160         -- two ones
161         partial_temp_layer1(2)(14) <= partial_temp(2)(14);
162         partial_temp_layer1(2)(20) <= '1';
163
164
165
166
167
168
169
170
171
172         -- layer 2
173         -- HA --> sum 0 carry 1
174         ha2_1 : ha port map (a => partial_temp_layer1(0)(2),
175                              b => partial_temp_layer1(1)(2),
176                              cout => partial_temp_layer2(1)(3),
177                              s => partial_temp_layer2(0)(2));
178         ha2_2 : ha port map (a => partial_temp_layer1(0)(3),
179                              b => partial_temp_layer1(1)(3),
180                              cout => partial_temp_layer2(1)(4),
181                              s => partial_temp_layer2(0)(3));
182         -- FA --> sum 0 carry 1
183         fa2_1 : fa port map (a => partial_temp_layer1(0)(5),
184                              b => partial_temp_layer1(1)(5),
185                              cin => partial_temp_layer1(2)(5),
186                              cout => partial_temp_layer2(1)(6),
187                              s => partial_temp_layer2(0)(5));
188         -- 4to2 --> sum 0 carry 1
189         four_to_two_2_1 : four_to_two_approx port map (
190                     a => partial_temp_layer1(0)(4),
191                     b => partial_temp_layer1(1)(4),
192                     c => partial_temp_layer1(2)(4),
193                     d => partial_temp_layer1(3)(4),
194                     cout => partial_temp_layer2(1)(5),
195                     s => partial_temp_layer2(0)(4)
196         );
197
198         four_to_two_gen : for i in 6 to 2 * N - 1 generate
199             four_to_two_inst : four_to_two_approx port map (
200                     a => partial_temp_layer1(0)(i),
201                     b => partial_temp_layer1(1)(i),
202                     c => partial_temp_layer1(2)(i),
203                     d => partial_temp_layer1(3)(i),
204                     cout => partial_temp_layer2(1)(i + 1),
205                     s => partial_temp_layer2(0)(i)
206                 );
207         end generate;
208
209         -- propagation
210         partial_temp_layer2(0)(0) <= partial_temp_layer1(0)(0);
211         partial_temp_layer2(1)(0) <= partial_temp_layer1(3)(0);
212
213         partial_temp_layer2(0)(1) <= partial_temp_layer1(0)(1);
214
215         partial_temp_layer2(1)(2) <= partial_temp_layer1(3)(2);
216
217         --partial_temp_layer2(1)(6) <= partial_temp_layer1(3)(6);
```

```
218
219         -- custom connection
220         partial_temp_layer2(0)(2 * N) <= '1';
221         partial_temp_layer2(1)(2 * N + 1) <= '1';
222
223         partial_temp_layer2(0)(2 * N + 1) <= '0';
224         partial_temp_layer2(1)(2 * N + 2) <= '1';
225
226         -- to avoid errors
227         partial_temp_layer2(1)(1) <= '0';
228
229         -- cut down version of 23 bits to 20
230         out1 <= partial_temp_layer2(0)(2 * Nbit - 1 downto 0);
231         out2 <= partial_temp_layer2(1)(2 * Nbit - 1 downto 0);
232
233     end behav;
```

B.8 Final Dadda tree

This section contains the code for the final chosen architecture for the Dadda tree, usign 4 4-2 compressors, along with 2 partial products generated by the AMBE, the remaining by the MBE.

B.8.1 Partial Product Generation - AMBE/MBE

./Code/ambe_mbe_approx.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  library work;
6  use work.util.all;
7
8  entity ambe_mbe_approx is
9      generic (limit : integer := 0);
10     port (a, b : in signed(Nbit - 1 downto 0);
11           partial : out partial_array);
12 end ambe_mbe_approx;
13
14 architecture behav of ambe_mbe_approx is
15     signal q, p_temp : multiple_factoring_array;
16     signal extended_b : signed(Nbit downto 0);
17     begin
18         -- zero right extension for index -1
19         extended_b <= b & '0';
20
21         -- q generation from MBE
22         multiple_factoring : for i in 0 to Nbit / 2 - 1 generate
23             q(i) <= a(Nbit -1) & a when (extended_b(2 * i + 1) xor extended_b(2 * i))
       = '1' else
24                     a & '0' when ((not (extended_b(2 * i + 1) xor extended_b(2 * i)))
       and (extended_b(2 * i + 2) xor extended_b(2 * i + 1))) = '1' else
25                     (others => '0');
26         end generate;
27         mix_gen : if limit > 0 and limit < 5 generate
28             partial_product_mbe : for i in limit to Nbit / 2 - 1 generate
29                 -- xor 1 bit for MBE
```

```
30                  p_temp_gen : for j in 0 to Nbit generate
31                      p_temp(i)(j) <= q(i)(j) xor b(2 * i + 1);
32                  end generate;
33                  -- partial product without sign for Roorda
34                  partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto
      0);
35                  -- Roorda extension
36                  partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
37                  -- Zero padding on the right
38                  if_padding : if i > 0 generate
39                      partial(i)(2 * i - 1 downto 0) <= (others => '0');
40                  end generate;
41                  -- MBE carry
42                  --p(N / 2)(2 * i) <= b(2 * i + 1);
43                  -- Roorda carry
44                  --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
45                  -- these last two could be compacted in one, MBE carry should be
46                  -- max in position N - 1, while Roorda carry starts from position
47                  -- N
48                  -- MBE + Roorda carry in same line
49                  partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
50                  partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
51                  partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
52                  partial(Nbit / 2)(2 * i + 1) <= '0';
53              end generate;
54
55              partial_product_ambe : for i in 0 to limit - 1 generate
56                  -- xor 1 bit for MBE
57                  p_temp_gen : for j in 0 to Nbit - 1 generate
58                      p_temp(i)(j) <= (a(j) xor extended_b(2 * i + 1)) and (extended_b(2
      * i) or extended_b(2 * i + 1));
59                  end generate;
60                  -- sign extension
61                  p_temp(i)(Nbit) <= p_temp(i)(Nbit - 1);
62                  -- partial product without sign for Roorda
63                  partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto
      0);
64                  -- Roorda extension
65                  partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
66                  -- Zero padding on the right
67                  if_padding : if i > 0 generate
68                      partial(i)(2 * i - 1 downto 0) <= (others => '0');
69                  end generate;
70                  -- MBE carry
71                  --p(N / 2)(2 * i) <= b(2 * i + 1);
72                  -- Roorda carry
73                  --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
74                  -- these last two could be compacted in one, MBE carry should be
75                  -- max in position N - 1, while Roorda carry starts from position
76                  -- N
77                  -- MBE + Roorda carry in same line
78                  partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
79                  partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
80                  partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
81                  partial(Nbit / 2)(2 * i + 1) <= '0';
82              end generate;
83          end generate;
84
85          full_approx_gen : if limit >= 5 generate
86              partial_product_ambe : for i in 0 to limit - 1 generate
87                  -- xor 1 bit for MBE
88                  p_temp_gen : for j in 0 to Nbit - 1 generate
```

```
89                      p_temp(i)(j) <= (a(j) xor extended_b(2 * i + 1)) and (extended_b(2
    * i) or extended_b(2 * i + 1));
90                  end generate;
91                  -- sign extension
92                  p_temp(i)(Nbit) <= p_temp(i)(Nbit - 1);
93                  -- partial product without sign for Roorda
94                  partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto
    0);
95                  -- Roorda extension
96                  partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
97                  -- Zero padding on the right
98                  if_padding : if i > 0 generate
99                      partial(i)(2 * i - 1 downto 0) <= (others => '0');
100                 end generate;
101                 -- MBE carry
102                 --p(N / 2)(2 * i) <= b(2 * i + 1);
103                 -- Roorda carry
104                 --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
105                 -- these last two could be compacted in one, MBE carry should be
106                 -- max in position N - 1, while Roorda carry starts from position
107                 -- N
108                 -- MBE + Roorda carry in same line
109                 partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
110                 partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
111                 partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
112                 partial(Nbit / 2)(2 * i + 1) <= '0';
113             end generate;
114         end generate;
115
116         no_approx_gen : if limit <= 0 generate
117             partial_product_mbe : for i in 0 to Nbit / 2 - 1 generate
118                 -- xor 1 bit for MBE
119                 p_temp_gen : for j in 0 to Nbit generate
120                     p_temp(i)(j) <= q(i)(j) xor b(2 * i + 1);
121                 end generate;
122                 -- partial product without sign for Roorda
123                 partial(i)(Nbit + 2 * i - 1 downto 2 * i) <= p_temp(i)(Nbit - 1 downto
    0);
124                 -- Roorda extension
125                 partial(i)(2 * Nbit - 1 downto Nbit + 2 * i) <= (others => '1');
126                 -- Zero padding on the right
127                 if_padding : if i > 0 generate
128                     partial(i)(2 * i - 1 downto 0) <= (others => '0');
129                 end generate;
130                 -- MBE carry
131                 --p(N / 2)(2 * i) <= b(2 * i + 1);
132                 -- Roorda carry
133                 --p(N / 2 + 1)(N + 2 * i) <= p_temp(N);
134                 -- these last two could be compacted in one, MBE carry should be
135                 -- max in position N - 1, while Roorda carry starts from position
136                 -- N
137                 -- MBE + Roorda carry in same line
138                 partial(Nbit / 2)(Nbit + 2 * i) <= not p_temp(i)(Nbit);
139                 partial(Nbit / 2)(2 * i) <= b(2 * i + 1);
140                 partial(Nbit / 2)(Nbit + 2 * i + 1) <= '0';
141                 partial(Nbit / 2)(2 * i + 1) <= '0';
142             end generate;
143         end generate;
144     end behav;
```

B.8.2 Partial Product Reduction - Dadda tree

./Code/dadda_four_to_two_variable_approx.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.util.all;

entity dadda_four_to_two_variable_approx is
    generic (limit : integer := 0);
    port (partial : in partial_array;
          out1, out2 : out signed(2 * Nbit - 1 downto 0));
end dadda_four_to_two_variable_approx;

architecture behav of dadda_four_to_two_variable_approx is
    component fa
        port (cin, a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component ha
        port (a, b : in std_logic;
            cout, s : out std_logic);
    end component;

    component four_to_two_approx
        port (a, b, c, d : in std_logic;
              cout, s : out std_logic);
    end component;

    signal partial_temp : internal_partial_array(Nbit / 2 downto 0);
    signal partial_temp_layer1 : internal_partial_array(Nbit / 2 - 2 downto 0);
    signal partial_temp_layer2 : internal_partial_array(Nbit / 2 - 3 downto 0);
    signal partial_temp_layer3 : internal_partial_array(Nbit / 2 - 4 downto 0);
    signal temp_partial_2 : signed(22 downto 0);

    begin
        partial_association : for i in 0 to Nbit / 2 generate
            partial_temp(i)(2 * Nbit - 1 downto 0) <= partial(i);
            partial_temp(i)(2 * Nbit + 2 downto 2 * Nbit) <= (others => '0');
        end generate;

        -- first stage --> from 6 to 4
        -- HA --> carry in 0 and sum in 1
        ha1_1 : ha port map (a => partial_temp(0)(6),
                             b => partial_temp(1)(6),
                             cout => partial_temp_layer1(0)(7),
                             s => partial_temp_layer1(1)(6));
        ha1_2 : ha port map (a => partial_temp(0)(7),
                             b => partial_temp(1)(7),
                             cout => partial_temp_layer1(0)(8),
                             s => partial_temp_layer1(1)(7));
        ha1_3 : ha port map (a => partial_temp(3)(8),
                             b => partial_temp(4)(8),
                             cout => partial_temp_layer1(0)(9),
                             s => partial_temp_layer1(1)(8));
        ha1_4 : ha port map (a => partial_temp(3)(9),
                             b => partial_temp(4)(9),
                             cout => partial_temp_layer1(0)(10),
```

```
59                              s => partial_temp_layer1(1)(9));
60          ha1_5 : ha port map (a => partial_temp(3)(11),
61                               b => partial_temp(4)(11),
62                               cout => partial_temp_layer1(0)(12),
63                               s => partial_temp_layer1(1)(11));
64          ha1_6 : ha port map (a => partial_temp(3)(15),
65                               b => partial_temp(4)(15),
66                               cout => partial_temp_layer1(0)(16),
67                               s => partial_temp_layer1(1)(15));
68          ha1_7 : ha port map (a => partial_temp(3)(13),
69                               b => partial_temp(4)(13),
70                               cout => partial_temp_layer1(0)(14),
71                               s => partial_temp_layer1(1)(13));
72          -- FA --> carry in 2/0 and sum in 3/1
73          fa1_1 : fa port map (a => partial_temp(0)(8),
74                               b => partial_temp(1)(8),
75                               cin => partial_temp(2)(8),
76                               cout => partial_temp_layer1(2)(9),
77                               s => partial_temp_layer1(3)(8));
78          fa1_2 : fa port map (a => partial_temp(0)(9),
79                               b => partial_temp(1)(9),
80                               cin => partial_temp(2)(9),
81                               cout => partial_temp_layer1(2)(10),
82                               s => partial_temp_layer1(3)(9));
83          fa1_3 : fa port map (a => partial_temp(0)(10),
84                               b => partial_temp(1)(10),
85                               cin => partial_temp(2)(10),
86                               cout => partial_temp_layer1(2)(11),
87                               s => partial_temp_layer1(3)(10));
88          fa1_4 : fa port map (a => partial_temp(0)(11),
89                               b => partial_temp(1)(11),
90                               cin => partial_temp(2)(11),
91                               cout => partial_temp_layer1(2)(12),
92                               s => partial_temp_layer1(3)(11));
93          fa1_5 : fa port map (a => partial_temp(3)(10),
94                               b => partial_temp(4)(10),
95                               cin => partial_temp(5)(10),
96                               cout => partial_temp_layer1(0)(11),
97                               s => partial_temp_layer1(1)(10)); -- row 3 already taken
    by FA, free for HA
98          fa1_6 : fa port map (a => partial_temp(3)(12),
99                               b => partial_temp(4)(12),
100                              cin => partial_temp(5)(12),
101                              cout => partial_temp_layer1(2)(13),
102                              s => partial_temp_layer1(3)(12));
103         fa1_7 : fa port map (a => partial_temp(3)(14),
104                              b => partial_temp(4)(14),
105                              cin => partial_temp(5)(14),
106                              cout => partial_temp_layer1(2)(15),
107                              s => partial_temp_layer1(3)(14));
108         fa1_8 : fa port map (a => partial_temp(3)(16),
109                              b => partial_temp(4)(16),
110                              cin => partial_temp(5)(16),
111                              cout => partial_temp_layer1(2)(17),
112                              s => partial_temp_layer1(3)(16));
113         -- propagations
114         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
115         partial_temp_layer1(0)(5 downto 0) <= partial_temp(0)(5 downto 0);
116         partial_temp_layer1(1)(5 downto 2) <= partial_temp(1)(5 downto 2);
117         partial_temp_layer1(2)(7) <= partial_temp(2)(7);
118         --partial_temp_layer1(2)(6) <= partial_temp(2)(6);
119         partial_temp_layer1(2)(5) <= partial_temp(2)(5);
```

```
120         partial_temp_layer1(2)(4) <= partial_temp(2)(4);
121         partial_temp_layer1(3)(7) <= partial_temp(3)(7);
122         partial_temp_layer1(2)(6) <= partial_temp(3)(6); -- because 3 will be used for
        Roorda and MBE carries
123         partial_temp_layer1(0)(6) <= partial_temp(2)(6);
124         partial_temp_layer1(3)(6) <= partial_temp(5)(6);
125         partial_temp_layer1(2)(8) <= partial_temp(5)(8); -- used by FA
126         roorda_mbe_carries : for i in 0 to 3 generate
127             partial_temp_layer1(3)(2 * i) <= partial_temp(5)(2 * i);
128         end generate;
129         -- custom connections
130         -- positioned in HA sum 1
131         -- three ones
132         partial_temp_layer1(1)(14) <= '1';
133         partial_temp_layer1(1)(16) <= '1';
134         partial_temp_layer1(1)(19) <= '1';
135         -- two ones
136         partial_temp_layer1(1)(18) <= partial_temp(5)(18);
137         --partial_temp_layer1(1)(12) <= partial_temp(3)(12);
138         -- one one
139         partial_temp_layer1(1)(17) <= not partial_temp(4)(17);
140         -- position in HA carry 0
141         -- three ones
142         partial_temp_layer1(0)(15) <= '1';
143         partial_temp_layer1(0)(17) <= '1';
144         -- two ones
145         partial_temp_layer1(0)(19) <= '1';
146         partial_temp_layer1(0)(13) <= '1';
147         partial_temp_layer1(0)(20) <= '1';
148         -- one one
149         partial_temp_layer1(0)(18) <= partial_temp(4)(17);
150         -- positioned in FA sum 3
151         -- three ones
152         partial_temp_layer1(3)(15) <= '1';
153         partial_temp_layer1(3)(17) <= '1';
154         partial_temp_layer1(3)(18) <= '1';
155         -- two ones
156         partial_temp_layer1(1)(12) <= partial_temp(2)(12); -- taken by FA
157         partial_temp_layer1(3)(13) <= partial_temp(2)(13);
158         -- positioned in FA carry 2
159         -- three ones
160         partial_temp_layer1(2)(16) <= '1';
161         partial_temp_layer1(2)(18) <= '1';
162         partial_temp_layer1(2)(19) <= '1';
163         -- two ones
164         partial_temp_layer1(2)(14) <= partial_temp(2)(14);
165         partial_temp_layer1(2)(20) <= '1';
166
167
168
169
170
171
172
173         -- layer 2
174         -- HA --> sum 0 carry 1
175         ha2_1 : ha port map (a => partial_temp_layer1(0)(4),
176                              b => partial_temp_layer1(3)(4),
177                              cout => partial_temp_layer2(1)(5),
178                              s => partial_temp_layer2(0)(4));
179         ha2_2 : ha port map (a => partial_temp_layer1(0)(5),
180                              b => partial_temp_layer1(1)(5),
```

```
181                         cout => partial_temp_layer2(1)(6),
182                         s => partial_temp_layer2(0)(5));
183         -- ha2_4 : ha port map (a => partial_temp_layer1(0)(16),
184         --                      b => partial_temp_layer1(1)(16),
185         --                      cout => partial_temp_layer2(1)(17),
186         --                      s => partial_temp_layer2(0)(16));
187         -- FA --> sum 0 carry 1
188
189         fa2_8 : fa port map (a => partial_temp_layer1(0)(14),
190                         b => partial_temp_layer1(2)(14),
191                         cin => partial_temp_layer1(3)(14),
192                         cout => partial_temp_layer2(1)(15),
193                         s => partial_temp_layer2(0)(14));
194         fa2_9 : fa port map (a => partial_temp_layer1(2)(13),
195                         b => partial_temp_layer1(3)(13),
196                         cin => partial_temp_layer1(1)(13),
197                         cout => partial_temp_layer2(1)(14),
198                         s => partial_temp_layer2(0)(13));
199
200         -- propagation
201         partial_temp_layer2(0)(3 downto 0) <= partial_temp_layer1(0)(3 downto 0);
202         partial_temp_layer2(1)(0) <= partial_temp_layer1(3)(0);
203         partial_temp_layer2(1)(4 downto 2) <= partial_temp_layer1(1)(4 downto 2);
204         partial_temp_layer2(2)(2) <= partial_temp_layer1(3)(2);
205         temp_partial_2(12 downto 4) <= partial_temp_layer1(3)(12 downto 8) &
206                                        partial_temp_layer1(2)(7 downto 4);
207         partial_temp_layer2(2)(5 downto 4) <= temp_partial_2(5 downto 4);
208         partial_temp_layer2(2)(14 downto 13) <= (others => '1');
209         partial_temp_layer2(2)(15) <= partial_temp_layer1(2)(15);
210         --partial_temp_layer2(0)(16) <= partial_temp_layer1(1)(16);
211         partial_temp_layer2(0)(16) <= partial_temp_layer1(0)(16);
212         partial_temp_layer2(2)(16) <= partial_temp_layer1(3)(16);
213         --partial_temp_layer2(1)(17) <= partial_temp_layer1(1)(17);
214         partial_temp_layer2(2)(17) <= partial_temp_layer1(2)(17);
215         partial_temp_layer2(2)(18) <= partial_temp_layer1(1)(18);
216         partial_temp_layer2(2)(19) <= partial_temp_layer1(2)(19);
217         partial_temp_layer2(0)(20) <= partial_temp_layer1(0)(20);
218         partial_temp_layer2(2)(20) <= partial_temp_layer1(2)(20);
219
220
221         -- custom connection
222         partial_temp_layer2(0)(15) <= partial_temp_layer1(1)(15);
223         partial_temp_layer2(1)(16) <= '1';
224
225         partial_temp_layer2(0)(17) <= partial_temp_layer1(1)(17);
226         partial_temp_layer2(1)(17) <= '1';
227         partial_temp_layer2(1)(18) <= '1';
228
229         partial_temp_layer2(0)(18) <= partial_temp_layer1(0)(18);
230         partial_temp_layer2(1)(19) <= '1';
231
232         partial_temp_layer2(0)(19) <= '0';
233         partial_temp_layer2(1)(20) <= '1';
234
235
236
237
238
239         -- layer 3
240         -- HA
241         ha3_1 : ha port map (a => partial_temp_layer2(0)(2),
242                         b => partial_temp_layer2(1)(2),
```

```
243                         cout => partial_temp_layer3(1)(3),
244                         s => partial_temp_layer3(0)(2));
245         ha3_2 : ha port map (a => partial_temp_layer2(0)(3),
246                         b => partial_temp_layer2(1)(3),
247                         cout => partial_temp_layer3(1)(4),
248                         s => partial_temp_layer3(0)(3));
249
250
251
252         -- FA
253         fa3_1 : fa port map (a => partial_temp_layer2(0)(4),
254                         b => partial_temp_layer2(1)(4),
255                         cin => partial_temp_layer2(2)(4),
256                         cout => partial_temp_layer3(1)(5),
257                         s => partial_temp_layer3(0)(4));
258
259         fa3_2 : fa port map (a => partial_temp_layer2(0)(5),
260                         b => partial_temp_layer2(1)(5),
261                         cin => partial_temp_layer2(2)(5),
262                         cout => partial_temp_layer3(1)(6),
263                         s => partial_temp_layer3(0)(5));
264
265
266
267         fa3_10 : fa port map (a => partial_temp_layer2(0)(13),
268                          b => partial_temp_layer2(1)(13),
269                          cin => partial_temp_layer2(2)(13),
270                          cout => partial_temp_layer3(1)(14),
271                          s => partial_temp_layer3(0)(13));
272
273         fa3_11 : fa port map (a => partial_temp_layer2(0)(14),
274                          b => partial_temp_layer2(1)(14),
275                          cin => partial_temp_layer2(2)(14),
276                          cout => partial_temp_layer3(1)(15),
277                          s => partial_temp_layer3(0)(14));
278
279         fa3_12 : fa port map (a => partial_temp_layer2(0)(15),
280                          b => partial_temp_layer2(1)(15),
281                          cin => partial_temp_layer2(2)(15),
282                          cout => partial_temp_layer3(1)(16),
283                          s => partial_temp_layer3(0)(15));
284
285         fa3_13 : fa port map (a => partial_temp_layer2(0)(17),
286                          b => partial_temp_layer2(1)(17),
287                          cin => partial_temp_layer2(2)(17),
288                          cout => partial_temp_layer3(1)(18),
289                          s => partial_temp_layer3(0)(17));
290
291         fa3_14 : fa port map (a => partial_temp_layer2(0)(18),
292                          b => partial_temp_layer2(1)(18),
293                          cin => partial_temp_layer2(2)(18),
294                          cout => partial_temp_layer3(1)(19),
295                          s => partial_temp_layer3(0)(18));
296
297         fa3_15 : fa port map (a => partial_temp_layer2(0)(16),
298                          b => partial_temp_layer2(1)(16),
299                          cin => partial_temp_layer2(2)(16),
300                          cout => partial_temp_layer3(1)(17),
301                          s => partial_temp_layer3(0)(16));
302
303
304
```

```
305         -- propagation
306         partial_temp_layer3(0)(0) <= partial_temp_layer2(0)(0);
307         partial_temp_layer3(1)(0) <= partial_temp_layer2(1)(0);
308
309         partial_temp_layer3(0)(1) <= partial_temp_layer2(0)(1);
310
311         -- partial_temp_layer3(1)(1) <= partial_temp_layer2(2)(1);
312         partial_temp_layer3(1)(1) <= '0'; -- to avoid U
313         partial_temp_layer3(1)(2) <= partial_temp_layer2(2)(2);
314
315
316         partial_temp_layer3(0)(20) <= partial_temp_layer2(0)(20);
317
318         -- custom connections
319         -- partial_temp_layer3(0)(16) <= partial_temp_layer2(2)(16);
320         -- partial_temp_layer3(1)(17) <= '1';
321
322         partial_temp_layer3(0)(19) <= '0';
323         partial_temp_layer3(1)(20) <= '1';
324
325         partial_temp_layer3(1)(21) <= '1';
326
327         -- cut down version of 23 bits to 20
328         out1 <= partial_temp_layer3(0)(2 * Nbit - 1 downto 0);
329         out2 <= partial_temp_layer3(1)(2 * Nbit - 1 downto 0);
330
331
332
333
334
335
336
337
338
339
340
341         mix_gen : if limit > 0 and limit < 6 generate
342             four_to_two_gen : for i in 6 to 6 + limit - 1 generate
343                 four_to_two_comp : four_to_two_approx
344                             port map (a => partial_temp_layer1(0)(i),
345                                     b => partial_temp_layer1(1)(i),
346                                     c => partial_temp_layer1(2)(i),
347                                     d => partial_temp_layer1(3)(i),
348                                     cout => partial_temp_layer2(1)(i + 1),
349                                     s => partial_temp_layer2(0)(i));
350             end generate;
351             fa_gen_layer2 : for i in 6 + limit to 12 generate
352             fa_comp : fa port map (a => partial_temp_layer1(0)(i),
353                                 b => partial_temp_layer1(1)(i),
354                                 cin => partial_temp_layer1(2)(i),
355                                 cout => partial_temp_layer2(1)(i + 1),
356                                 s => partial_temp_layer2(0)(i));
357             end generate;
358
359             fa_gen_layer3 : for i in 6 + limit to 12 generate
360                 fa_comp_layer3 : fa port map (a => partial_temp_layer2(0)(i),
361                                             b => partial_temp_layer2(1)(i),
362                                             cin => partial_temp_layer2(2)(i),
363                                             cout => partial_temp_layer3(1)(i + 1),
364                                             s => partial_temp_layer3(0)(i));
365             end generate;
366
```

```
367             ha_gen : for i in 6 to 6 + limit - 1 generate
368                 ha_comp : ha port map (a => partial_temp_layer2(0)(i),
369                                        b => partial_temp_layer2(1)(i),
370                                        cout => partial_temp_layer3(1)(i + 1),
371                                        s => partial_temp_layer3(0)(i));
372             end generate;
373 
374             partial_temp_layer2(2)(12 downto 6 + limit) <=
375                                        temp_partial_2(12 downto 6 + limit);
376         end generate;
377 
378         no_approx_gen : if limit <= 0 generate
379             fa_gen_6_7 : for i in 6 to 7 generate
380                 fa_comp : fa port map (a => partial_temp_layer1(0)(i),
381                                        b => partial_temp_layer1(1)(i),
382                                        cin => partial_temp_layer1(3)(i),
383                                        cout => partial_temp_layer2(1)(i + 1),
384                                        s => partial_temp_layer2(0)(i));
385             end generate;
386 
387             fa_gen_others : for i in 8 to 12 generate
388                 fa_comp : fa port map (a => partial_temp_layer1(0)(i),
389                                        b => partial_temp_layer1(1)(i),
390                                        cin => partial_temp_layer1(2)(i),
391                                        cout => partial_temp_layer2(1)(i + 1),
392                                        s => partial_temp_layer2(0)(i));
393             end generate;
394 
395             fa_gen_layer3 : for i in 6 to 12 generate
396                 fa_comp_layer3 : fa port map (a => partial_temp_layer2(0)(i),
397                                        b => partial_temp_layer2(1)(i),
398                                        cin => partial_temp_layer2(2)(i),
399                                        cout => partial_temp_layer3(1)(i + 1),
400                                        s => partial_temp_layer3(0)(i));
401             end generate;
402 
403             partial_temp_layer2(2)(12 downto 6) <=
404                                        temp_partial_2(12 downto 6);
405         end generate;
406 
407         full_approx_gen : if limit >= 6 generate
408             four_to_two_gen : for i in 6 to 12 generate
409                 four_to_two_comp : four_to_two_approx
410                             port map (a => partial_temp_layer1(0)(i),
411                                       b => partial_temp_layer1(1)(i),
412                                       c => partial_temp_layer1(2)(i),
413                                       d => partial_temp_layer1(3)(i),
414                                       cout => partial_temp_layer2(1)(i + 1),
415                                       s => partial_temp_layer2(0)(i));
416             end generate;
417 
418             ha_gen : for i in 6 to 12 generate
419                 ha_comp : ha port map (a => partial_temp_layer2(0)(i),
420                                        b => partial_temp_layer2(1)(i),
421                                        cout => partial_temp_layer3(1)(i + 1),
422                                        s => partial_temp_layer3(0)(i));
423             end generate;
424         end generate;
425 
426 
427     end behav;
```

B.9 Extra files

These files were used to generate all the graphs and checks all the results.

B.9.1 Testbench for results generation - parallel for variable Dadda tree

./Code/testbench_dadda_final_approx.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;

library std;
use std.textio.all;

library work;
use work.util.all;

entity testbench_dadda_final_approx is
    generic (max_limit_ambe : integer := 5;
             max_limit_dadda : integer := 6);
end testbench_dadda_final_approx;

architecture behav of testbench_dadda_final_approx is

    component mul_ambe_mbe_dadda_four_to_two_with_enable
        generic (limit_ambe : integer := 0;
                 limit_dadda : integer := 0);
        port (a, b : in signed(Nbit -1  downto 0);
            product : out signed(2 * Nbit - 1 downto 0);
            en : in std_logic);
    end component;

    signal a, b : signed(9 downto 0) := "0000000000";
    signal a_int, b_int, correct_int : integer;
    signal correct_bin : signed(19 downto 0);
    --type subfile_array is array(max_limit_dadda downto 0) of file;
    --type file_array is array(max_limit_ambe downto 0) of subfile_array;
    type subproduct_array is
            array(max_limit_dadda downto 0) of signed(2 * Nbit - 1 downto 0);
    type product_array is
        array(max_limit_ambe downto 0) of subproduct_array;
    type subproduct_int_array is array(max_limit_dadda downto 0) of integer;
    type product_int_array is
         array(max_limit_ambe downto 0) of subproduct_int_array;
    type substring_array is array(max_limit_dadda downto 0) of string(40 downto 1);
    type string_array is array(max_limit_ambe downto 0) of substring_array;
    type correct_array is array(max_limit_ambe downto 0) of
                            std_logic_vector(max_limit_dadda downto 0);

    signal product : product_array;
    signal product_int, dist : product_int_array;
    --signal files : file_array;
    signal filenames : string_array;
    signal basedir : string(24 downto 1) := "Comparisons_final_approx";
    signal correct, en_array : correct_array;

    begin
        if_gen : if max_limit_ambe >= 0 and max_limit_ambe <= 5 and
```

```
53                  max_limit_dadda >= 0 and max_limit_dadda <= 6 generate
54          ambe_gen : for i in 0 to max_limit_ambe generate
55              dadda_gen : for j in 0 to max_limit_dadda generate
56                  test : mul_ambe_mbe_dadda_four_to_two_with_enable
57                      generic map (limit_ambe => i, limit_dadda => j)
58                      port map (a => a, b => b, product => product(i)(j),
59                                en => en_array(i)(j));
60                  product_int(i)(j) <= to_integer(product(i)(j));
61                  dist(i)(j) <= correct_int - product_int(i)(j);
62                  filenames(i)(j) <= basedir & "/ambe" & integer'image(i) &
63                                     "dadda" & integer'image(j) & ".txt";
64              end generate;
65          end generate;
66      end generate;
67
68      a <= to_signed(a_int, 10);
69      b <= to_signed(b_int, 10);
70      correct_bin <= to_signed(correct_int, 20);
71
72
73  process
74      variable line_out : line;
75      file output : text;
76      begin
77          a_int <= 0;
78          b_int <= 0;
79          en_array <= (others => (others => '0'));
80
81          wait for 10 ns;
82
83          if max_limit_ambe >= 0 and max_limit_ambe <= 5 and
84                          max_limit_dadda >= 0 and max_limit_dadda <= 6 then
85              for k in 0 to max_limit_ambe loop
86                  for l in 0 to max_limit_dadda loop
87                      file_open(output, filenames(k)(l), write_mode);
88                      en_array(k)(l) <= '1';
89                      for i in -2 ** 9 to 2 ** 9 - 1 loop
90                          for j in -2**9 to 2 ** 9 - 1 loop
91                              a_int <= i;
92                              b_int <= j;
93                              wait for 1 ps;
94                              correct_int <= i * j;
95                              if (i * j - product_int(k)(l)) = 0 then
96                                  correct(k)(l) <= '1';
97                              else
98                                  correct(k)(l) <= '0';
99                              end if;
100                             wait for 2500 ps;
101                             --write(line_out, string'("a bin: "));
102                             --write(line_out, a, right, 10);
103                             --write(line_out, string'(" ; a int: "));
104                             --write(line_out, string'("a int: "));
105                             write(line_out, a_int);
106                             --write(line_out, string'(" ; b bin: "));
107                             --write(line_out, b, right, 10);
108                             --write(line_out, string'(" ; b int: "));
109                             write(line_out, string'(";"));
110                             write(line_out, b_int);
111                             --write(line_out, string'(" ; product bin: "));
112                             --write(line_out, product, right, 20);
113                             --write(line_out, string'(" ; product int: "));
114                             write(line_out, string'(";"));
```

```
                                write(line_out, product_int(k)(l));
                                --write(line_out, string'(" ; correct bin: "));
                                --write(line_out, correct_bin, right, 20);
                                --write(line_out, string'(" ; correct int: "));
                                write(line_out, string'(";"));
                                write(line_out, correct_int);
                                --write(line_out, string'(" ; correct bool: "));
                                write(line_out, string'(";"));
                                write(line_out, correct(k)(l));
                                --write(line_out, string'(" ; distance: "));
                                write(line_out, string'(";"));
                                write(line_out, dist(k)(l));
                                writeline(output, line_out);
                                wait for 2499 ps;
                            end loop;
                        end loop;
                        wait for 0.5 us;
                        file_close(output);
                        en_array(k)(l) <= '0';
                        wait for 0.5 us;
                    end loop;
                end loop;
            end if;

            -- a <= "0000000001";
            -- b <= "0000000001";

            -- wait for 10 ns;

            -- a <= "1111111111";
            -- b <= "1111111111";

            -- wait for 10 ns;

            -- a <= "1111111110";
            -- b <= "0000000001";

            -- wait for 10 ns;

            -- a <= "1010010100";
            -- b <= "1010111001";

            -- wait for 10 ns;


            wait;
        end process;
    end behav;
```

B.9.2 Testbench for results generation - single

./Code/testbench_dadda.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;

library std;
use std.textio.all;
```

```
8
9  library work;
10 use work.util.all;
11
12 entity testbench_dadda is
13 end testbench_dadda;
14
15 architecture behav of testbench_dadda is
16
17     component mul_dadda_standard_no6LSB
18         port (a, b : in signed(Nbit -1  downto 0);
19             product : out signed(2 * Nbit - 1 downto 0));
20     end component;
21
22     signal a, b : signed(9 downto 0) := "0000000000";
23     signal a_int, b_int, product_int, correct_int, dist : integer;
24     signal product, correct_bin : signed(19 downto 0);
25     signal correct : std_logic;
26
27     begin
28         test : mul_dadda_standard_no6LSB port map (a => a, b => b, product => product)
       ;
29
30         a <= to_signed(a_int, 10);
31         b <= to_signed(b_int, 10);
32         product_int <= to_integer(product);
33         correct_bin <= to_signed(correct_int, 20);
34         dist <= correct_int - product_int;
35
36
37     process
38         variable line_out : line;
39         file output : text open WRITE_MODE is "./Matlab/PDFs/
       results_dadda_standard_no6LSB.txt";
40         begin
41             a_int <= 0;
42             b_int <= 0;
43
44             wait for 10 ns;
45
46             for i in -2 ** 9 to 2 ** 9 - 1 loop
47                 for j in -2**9 to 2 ** 9 - 1 loop
48                     a_int <= i;
49                     b_int <= j;
50                     wait for 1 ps;
51                     correct_int <= i * j;
52                     if (i * j - product_int) = 0 then
53                         correct <= '1';
54                     else
55                         correct <= '0';
56                     end if;
57                     wait for 2499 ps;
58                     --write(line_out, string'("a bin: "));
59                     --write(line_out, a, right, 10);
60                     --write(line_out, string'(" ; a int: "));
61                     --write(line_out, string'("a int: "));
62                     write(line_out, a_int);
63                     --write(line_out, string'(" ; b bin: "));
64                     --write(line_out, b, right, 10);
65                     --write(line_out, string'(" ; b int: "));
66                     write(line_out, string'(";"));
67                     write(line_out, b_int);
```

```
68                  --write(line_out, string'(" ; product bin: "));
69                  --write(line_out, product, right, 20);
70                  --write(line_out, string'(" ; product int: "));
71                  write(line_out, string'(";"));
72                  write(line_out, product_int);
73                  --write(line_out, string'(" ; correct bin: "));
74                  --write(line_out, correct_bin, right, 20);
75                  --write(line_out, string'(" ; correct int: "));
76                  write(line_out, string'(";"));
77                  write(line_out, correct_int);
78                  --write(line_out, string'(" ; correct bool: "));
79                  write(line_out, string'(";"));
80                  write(line_out, correct);
81                  --write(line_out, string'(" ; distance: "));
82                  write(line_out, string'(";"));
83                  write(line_out, dist);
84                  writeline(output, line_out);
85                  deallocate(line_out);
86                  wait for 2500 ps;
87              end loop;
88          end loop;
89
90          -- a <= "0000000001";
91          -- b <= "0000000001";
92
93          -- wait for 10 ns;
94
95          -- a <= "1111111111";
96          -- b <= "1111111111";
97
98          -- wait for 10 ns;
99
100         -- a <= "1111111110";
101         -- b <= "0000000001";
102
103         -- wait for 10 ns;
104
105         -- a <= "1010010100";
106         -- b <= "1010111001";
107
108         -- wait for 10 ns;
109
110
111         wait;
112     end process;
113 end behav;
```

B.9.3 Matlab script for comparisons

./Code/dadda_final_approx_error.m

```
1 clear all
2 close all
3 clc
4
5 res_correct = importdata("../results_correct.txt");
6 file_names = ["dadda_4to2_layer2"
7               "dadda_ambe"
8               "dadda_ambe_4to2_layer2"
9               "dadda_final_approx"
```

```
                "dadda_no_6LSB"
                "dadda_standard_no6LSB"];
titles = ["Fully-approximate architecture using 4-2 compressors"
          "Fully-approximate architecture using AMBE"
          "Fully-approximate architecture using AMBE & 4-2 compressors"
          "Final approximate Dadda architecture"
          "Manually-optimized Dadda architecture (no 6 LSBs)"
          "Standard Dadda architecture (no 6 LSBs)"];

dest_folder = "Images";
dest_filetype = ".png";
appendices = ["signal" "abs_error"];
folder = ".";
filetype = ".txt";
files_to_process = ls(folder + "/*" + filetype);
len = size(files_to_process, 1);

for i=1:len
    res_final = importdata(files_to_process(i, :));

    figure('units','normalized','outerposition',[0 0 1 1],"DefaultAxesFontSize", 24)
    xlabel("Time samples");
    ylabel("Bit value");
    title(titles(i, :));
    hold on
    plot(res_correct, "LineWidth", 2);
    hold on
    plot(res_final, "LineWidth", 2);
    legend("Exact", "Approximate");
    F = getframe(gcf);
    imwrite(F.cdata, dest_folder + "/" + file_names(i, :) + "_" + appendices(1) +
    dest_filetype);
    close(gcf);

    figure('units','normalized','outerposition',[0 0 1 1],"DefaultAxesFontSize", 24)
    plot(res_correct - res_final);
    close(gcf);

    figure('units','normalized','outerposition',[0 0 1 1],"DefaultAxesFontSize", 24)
    xlabel("Time samples");
    ylabel("Bit value");
    title(titles(i, :) + " Absolute Error");
    hold on
    plot(abs(res_correct - res_final), "LineWidth", 2);
    F = getframe(gcf);
    imwrite(F.cdata, dest_folder + "/" + file_names(i, :) + "_" + appendices(2) +
    dest_filetype);
    close(gcf);
end
```

B.9.4 Matlab script for distributions - variable

./Code/dadda_matlab.m

```
close all
clear all
clc

folder = "C:/Users/colucci/Desktop/Comparisons_final_approx";
filetype = ".txt";
```

```
ambe_pos = 5;
dadda_pos = 11;

files_to_process = ls(folder + "/*" + filetype);
indexes = [str2num(files_to_process(:, ambe_pos)) str2num(files_to_process(:,
    dadda_pos))];
max_ = max(indexes);
max_ambe = max_(1) + 1;
max_dadda = max_(2) + 1;
figure('Name', 'AMBE and Dadda Analysis')
title('AMBE and Dadda Analysis')
for k=1:size(files_to_process, 1)
    file = files_to_process(k, :)
    index = [str2num(file(ambe_pos)) str2num(file(dadda_pos))];
    data = importdata(folder + "/" + file, ";");
    data_to_plot = data(:, end);
    max_ = max(data_to_plot)
    mean_ = mean(data_to_plot)
    var_ = var(data_to_plot)
    pd = fitdist(data_to_plot, 'Normal')
    subplot(max_ambe, max_dadda, k);
    plot(sort(data_to_plot), pdf(pd, sort(data_to_plot)), 'LineWidth', 1)
    title("Ambe: " + string(index(1)) + " Dadda: " + string(index(2)));
    ylabel('Normalized Probability');
    xlabel('Error');
    %figure
    %plot(data(1:10000, end))
end
```

B.9.5 Matlab script for distributions - multiple

./Code/dadda_pdf_error.m

```
clear all
close all
clc

titles = ["Fully-approximate architecture using AMBE"
          "Fully-approximate architecture using AMBE & 4-2 compressors"
          "Fully-approximate architecture using 4-2 compressors"
          "Manually-optimized Dadda architecture (no 6 LSBs)"
          "Standard Dadda architecture (no 6 LSBs)"];

file_names = ["dadda_ambe"
              "dadda_ambe_4to2_layer2"
              "dadda_4to2_layer2"
              "dadda_no_6LSB"
              "dadda_standard_no6LSB"];
dest_folder = "Images";
dest_filetype = ".png";

folder = ".";
filetype = ".txt";
appendix = "pdf";
files_to_process = ls(folder + "/*" + filetype);
len = size(files_to_process, 1);

for i=1:len
    res_final = importdata(files_to_process(i, :));
    pd = fitdist(res_final(:, end), 'Normal');
```

```
28
29      figure('units','normalized','outerposition',[0 0 1 1],"DefaultAxesFontSize", 24)
30      xlabel("Errors");
31      ylabel("Probability");
32      title(titles(i, :) + " Error Distribution");
33      hold on
34      plot(sort(res_final(:, end)), pdf(pd, sort(res_final(:, end))), "LineWidth", 2);
35      F = getframe(gcf);
36      imwrite(F.cdata, dest_folder + "/" + file_names(i, :) + "_" + appendix +
        dest_filetype);
37      close(gcf);
38  end
```

B.9.6 Python script for result comparison

./Code/check_equality.py

```
1  import sys
2
3  with open(sys.argv[1], 'r') as f:
4      with open(sys.argv[2], 'r') as g:
5          a = g.read().splitlines()
6          b = f.read().splitlines()
7          print("-" * 100)
8          print("|" + ''.rjust(5) + "|" + sys.argv[1].rjust(
9              30) + "|" + sys.argv[2].ljust(30) + "|" + "distance".center(30)
       + "|")
10         print("-" * 100)
11         for pos, c in enumerate(a):
12             if b[pos] != c:
13                 print("|" + str(pos).rjust(5) + "|" +
14                       b[pos].rjust(30) + "|" + c.ljust(30) + "|" + str(int(b
       [pos]) - int(c)).center(30) + "|")
15                 print("-" * 100)
```