Digital Systems Electronics
Academic year 2023 - 2024
Delivery date: 09/05/2024

Group 11
Laboratory n° 6
A simple digital filter

Authors:
Michela Cipriano
Simone Rossi
Riccardo Davide Cannone

Contents

1 Design Entry

The aim of this laboratory was to deal with a realistic design problem: the realization of a digital filter, a circuit that performed a series of scheduled arithmetic operations on input data. In the following, we document our logic procedure to implement the circuit, providing descriptions in terms of schemes and Modelsim simulations' fragments.

Source files:
Main: ACTIVE_FILTER.vhd;
Datapath: DP.vhd; full_adder.vhd, adder.vhd, extender.vhd, generic_reg.vhd, D_flip_flop.vhd, mux2to1.vhd, Filtering_function.vhd, saturator.vhd, MEM_A_WR_COUNTER.vhd, MEM_B_WR_COUNTER.vhd, MEM_A_RD_COUNTER.vhd;
Control Unit: CU.vhd;
Memories: MEMORY_A.vhd, MEMORY_B.vhd;
Simulation: ACTIVE_FILTER_TB.vhd, TB_GM.vhd, GM_MAT.m, REFERENCE_MODEL.m.

1.1 Pseudocode and ASM Chart of the Algorithm

We report the pseudocode description of the circuit we implemented. Since pseudocode is a form of very high-level syntax, we describe only essential operations, detailing every step in **Section 1.2**.

```
if start = 1 or restart = 1
address_A = 0
i = 0
memory_A_has_written = 0
number_of_process = 0
sum = 0
restart = 0
while (address_A < 1024):
    Memory_A(address_A) = data_in[i]
    address_A ++
    i ++
end while

address_A = 0
address_B = 0

while (address_B < 1024)
    Y = 0
    for index in range(3):
        N = address_B
        if (N - index < 0)
            X(N - index) = "000...0"
        else
            X(N - index) = Memory_A(N - index)
        X_shifted(N - index) = shift(X(N - index), LD(0), SH(0), N_SHIFT)
        X_load1(N - index) = load(X_shifted(N - index), LD(1), SH(1))
        if (index = 0 or index = 1)
            X_xor(N - index) = invert(X_load1(N - index))
        else
            X_xor(N - index) = X_load1(N)
        partial_sum = X_xor(N - index) + sum
        sum = load(partial_sum, LD(2), SH(2))
    Y = sat(sum)
    Memory_B(address_B) = Y
    Address_B ++
```

```
end while
```

We also provide the ASM Chart of the overall implemented function in Figure 1.

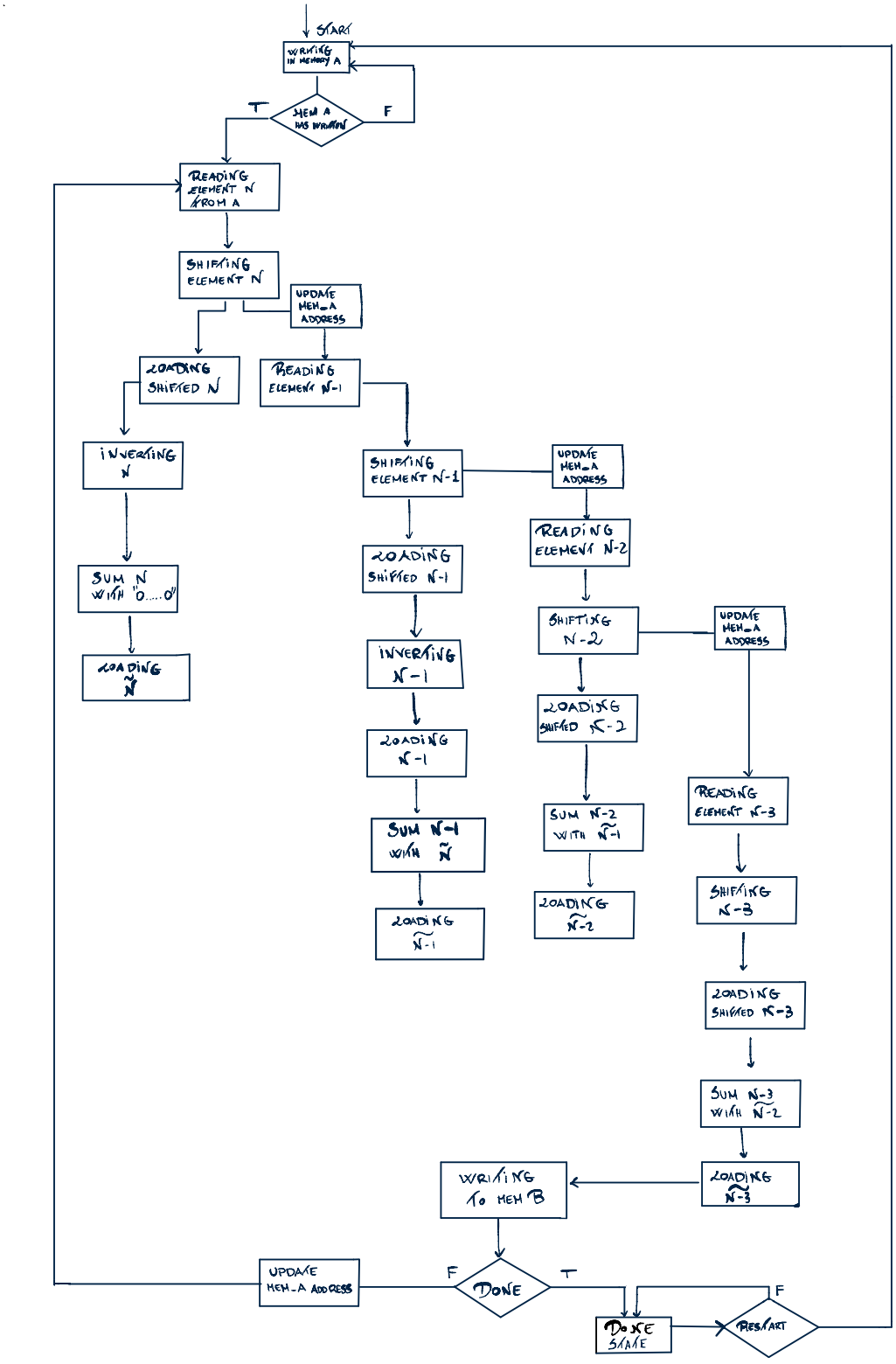

Figure 1: ASM Chart - Algorithm

1.2 Description and drawing of the circuit

In the following, we detail the sequence of operations we implemented in out circuit.

1. START = 1 makes CS_A = 1, i.e. memory A is selected and ready to operate;

2. WR_A = 1 activates a Modulo 1024 up-counter, MEM_A_WR_COUNTER, which counts the occurrence of every rising edge of the clock and gives in output ADDRESS_MEM_A_WR;

3. DATA_IN is given in input to MEM_A and stored at the location MEM_A_ADDRESS;

4. The counter stops at the 1024^{th} clock cycle, so the process stops after 1024 8-bit vectors DATA_IN has been stored into MEM_A. This number of data corresponds to the maximum capacity of the storage element and can be translated into the condition $MEM\_A\_HAS\_WRITTEN = 1$, which is a status signal sent to the control unit. Then, we start reading data from MEM_A in a combinational way, with the aim to send them in input to the sequential circuit filtering_function. The circuit filtering_function is made of an extender, one shift register, two load registers load_1 and load_2, one 11-bit RCA, two multiplexers, a D-flip-flop and a XOR gate;

5. The component extender is used to concatenate the input vector with "000" or "111" in order to extend the vector's length and avoid overflow problems that can occur both with the shifting and addition operations;

6. An external counter, MEM_A_RD_COUNTER, driven by the CU, provides the value of the addresses *n*, *n-1*, *n-2*, *n-3*;

7. The full-adder takes in input *add_mux*, *sum_buffer_mux* and the carry *inv*, and gives in output the sum *partial_sum*. Our aim is to sum the 4 values X(n), X(n-1), X(n-2), X(n-3) in 2's complement form, properly inverted and/or shifted, so the full-adder will perform 4 partial sums.
   The overall function implemented by filtering_functions falls within the range of 10 clock cycles, each one of them corresponding to a control unit state. We will now describe the computation of the output for a generic process *m*, one of the 1024 that the circuit is going to perform in the same way before reaching the DONE STATE.

   - **Iteration 1 (also READING_A1 STATE),** $1^{st} - 2^{nd}$ ⌐↓
     X_m(n) is read from the memory A, and the instruction to properly shift this value are given to the the shift register by the CU. This passage is essential, since the shifter needs to receive information about the operation to perform at least one clock cycle earlier.
   - **Iteration 2 (also ELABORATING_A1 STATE),** $2^{rd} - 3^{th}$ ⌐↓
     X_m(n) is shifted according to the received settings and given as input of the first load register load_1, which has been already set to perform its action in the previous clock cycle. Meanwhile the CU sends the signal to update MEM_A_ADDRESS in order to perform a second reading in the next iteration.
   - **Iteration 3 (also READING_A2 STATE),** $3^{th} - 4^{th}$ ⌐↓
     The shifted value of X_m(n) is finally ready to be summed (with its carry *inv)*, after being sent as output from the first load register to the adder. The second

addendum, in this state, is set to the value "0". Before that, the value passes through a XOR gate, which is able to perform an inversion according to the extended value of *inv*. Meanwhile X_m(n-1) is read from the memory and the instructions to perform the second shifting are given by the CU.

- **Iteration 4 (also ELABORATING_A2 STATE),** $4^{th} - 5^{th}$ ⌐↓
  The value of the first partial sum passes through the second load register, and it should immediately be summed with the output of the first load register. However, during this state, a D-flip-flop (together with a 2-to-1 multiplexer) controlled by the CU avoids modifications of the partial sum: this is necessary because the value of X_m(n-1) has just been shifted and so we need to wait for the next clock cycle to perform the right addition. As before, MEM_A_ADDRESS is updated to perform a third reading.
- **Iteration 5,6,7,8 (also READING_A3, ELABORATING_A3, READING_A4 AND ELABORATING_A4 STATES),** $5^{th}-6^{th}$ **,** $6^{th}-7^{th}$**,** $7^{th}-8^{th}$**,** $8^{th} - 9^{th}$ ⌐↓ ⌐↓ ⌐↓ ⌐↓
  In the next iterations the concept stays the same: a new value is read in the READING STATE while the previous one is loaded in the first load register and the sum is performed. The new value is shifted in the ELABORATING STATE, when the value of the address (of MEM_A) is updated by the CU and the process of addition is skipped (or to be more precise, a zero values vector is added to partial sum). However, during an ELABORATION STATE the value of the latest sum is sent as output of the second load register and so it's ready to be updated in the next clock cycle, when the D-flip-flop will let that happen.
- **Iteration 9 (also COMPLETING_SEQUENCE STATE),** $9^{th} - 10^{th}$ ⌐↓
  In this iteration the sum is completed with the last value X_m(N-4), so the last loading process in load1 is performed.
- **Iteration 10,11 (also PREPARATION_TO_WRITE AND WRITING_B STATES),** $9^{th} - 10^{th}$ **,** $10^{th} - 11^{th}$ ⌐↓ ⌐↓
  In these last two iterations the final sum is initially loaded in the load2 register and then sent to the memory B after being properly saturated. Also a new address for writing in B is provided in the clock cycle that anticipate the writing process.

8. When also the process of writing into MEM_B is completed, the Control Unit sends the signal "DONE", and the the circuit enters stand-by. The overall function starts again whenever the signal START returns to "1", and the new values provided as input data overwrite the previous ones.

A drawing of the overall circuit is shown in Figure 2. Finally, we report the description of the Control Unit in terms of State diagram and ASM Chart (Figures 3 and 4).

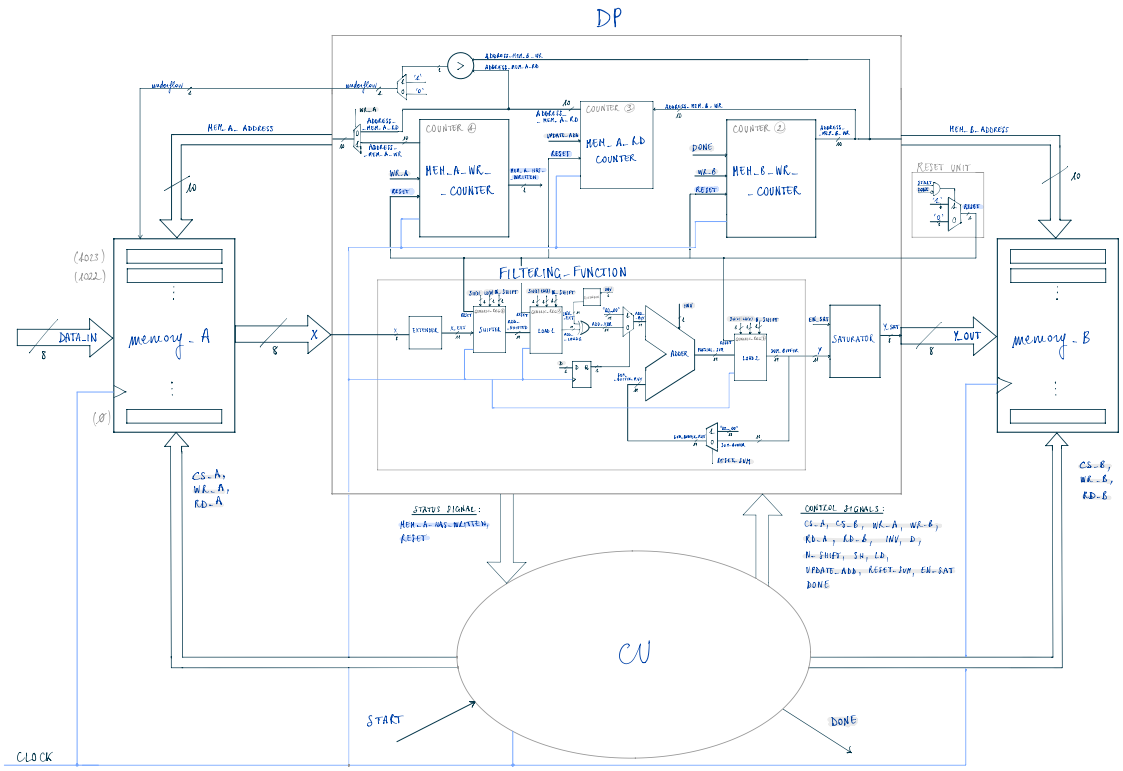

Figure 2: Drawing of the circuit

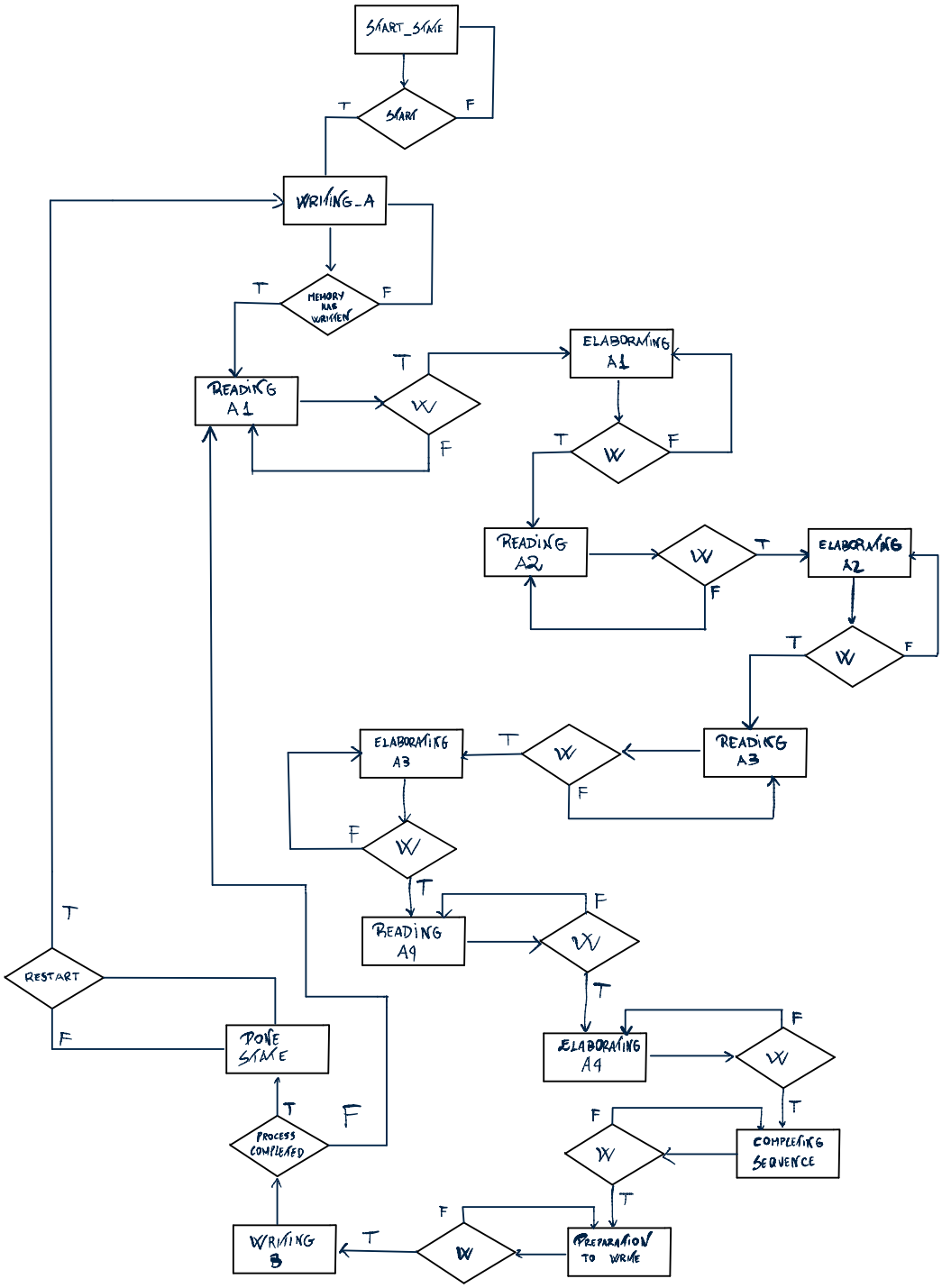

Figure 3: ASM Chart - Control Unit

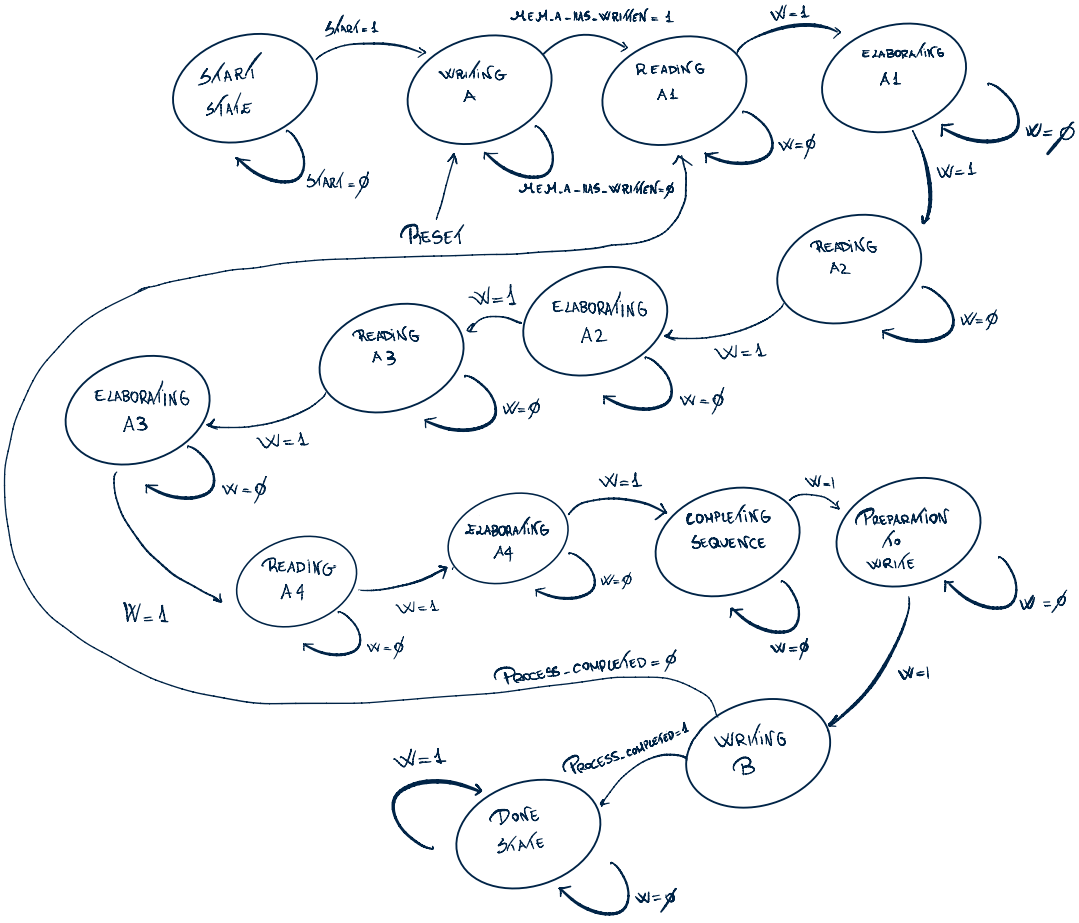

Figure 4: State diagram CU

2 Functional simulation

2.1 Testbench

Since we couldn't print our circuit on a FPGA (which could have given us more proof of the correct behavior of the circuit), it was of the uttermost importance to create a proper bench and simulate the correct behavior of the circuit in all of its parts. Yet, to kickstart the circuit by means of a usual testbench, we would have needed to manually insert 1024 values; even then, there would have been so many conditions to check: eventual timing problems, the correct output from all possible combinations of numbers (even and odd, positive and negative), the correct saturation, truncation and so much more!
So, in order to facilitate this process, we extended the scope of a regular testbench and created a more complex one, able to communicate with Matlab, both to receive the inputs and to send back the outputs, so that a program could make a comparison with a Golden Model GM (also known as Reference Model RM), checking that every output of our circuit coincided with the theoretical one computed by the GM. The golden model was extremely useful, as our circuit is complex: data and signals go inside many components which interact between each other, where they are processed before being sent back to other parts of the circuit. The risk of a fault somewhere in this system is not negligible.
Comparing the outputs of our circuit with others realized by means of simple high-level functions allowed us to be certain of the correct behavior of the project. Yet, implementing the GM wasn't as easy as recreating the given digital filter's function, as the reference circuit needed to have the same limitations of our circuit, mostly for what concerns the truncation of floating point numbers: in order to divide by 2 (or 4), we needed to shift our binary vectors by 1 or 2 positions to the right, losing in the process any information about the last (or last 2) bits. A signed number, in 8-bit 2's complement representation, is in the form $-2^7 \cdot b_7 + 2^6 \cdot b_6 + ... + b_0$.
Removing the last bit b_0 leads to a rounding down in absolute value for all floating positive numbers (ex: $7 = 0111$, divided by $2 \rightarrow 0011 = 3$). For a negative number, instead, removing the last bit b_0 leads to removing a "+1" from a negative amount, so that is actually increasing its absolute value (ex: - $7 = 1001$, divided by $2 \rightarrow 1100 =$ - 4). So any floating negative number will be rounded "up" (in its absolute value).
Hadn't we recreated the real behavior of our circuit, it would have been impossible for us to distinguish between flags caused by different approximations (between the Matlab GM and our circuit) and flags caused by real problems and errors in the code, unless we manually checked every flag.

To start our testbench, one must follow 4 steps:

1. Run the first Matlab code (GM_MAT.m), which creates two text files ('dati.txt' for the first fill and 'dati2.txt' for the restart), loading them with 1024 lines of 8 random bits, our testbench's inputs, and then it also computes through the Golden Model the expected theoretical output, writing it to another file ('dati_ref.txt').

2. After checking that all the files mentioned above have been created, run the simulation on Modelsim for a period of time which is enough for 'DONE_AUX' to rise up for the second time after the restart ($\sim$ 3ms). The use of the 'Textio' libraries allowed us to import the data we created in step 1.

3. After the simulation is completed, observe carefully that all signals of the internal circuit and components work as expected, as well as the restart process.

4. Finally, we can check if the final output is correct. 'dati out.txt' should have been created and filled with 1024 lines of binary vectors. All unnecessary spaces and newline characters will be removed by our final Matlab code ('REFERENCE MODEL.m'), preparing the file for the comparison with 'dati_ref.txt'. The number of flags (mismatches) is indicated by the console, and if no flags are detected, the system prints "okay".

The waveforms associated to our simulation are in Figures 5 and 6.

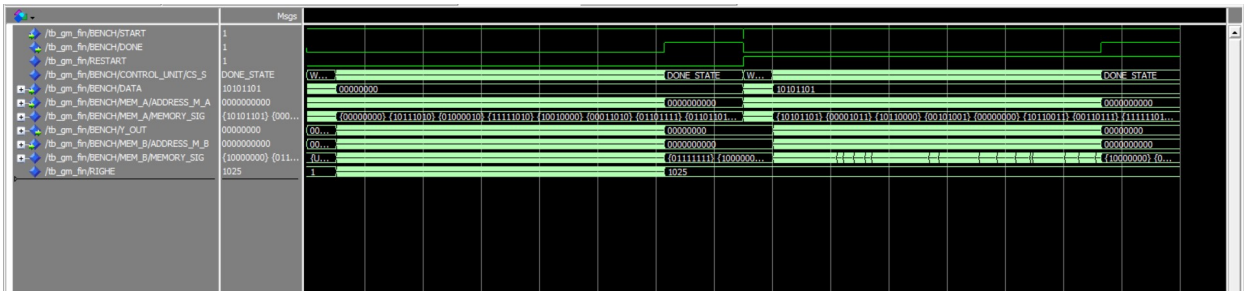

Figure 5: Functional simulation

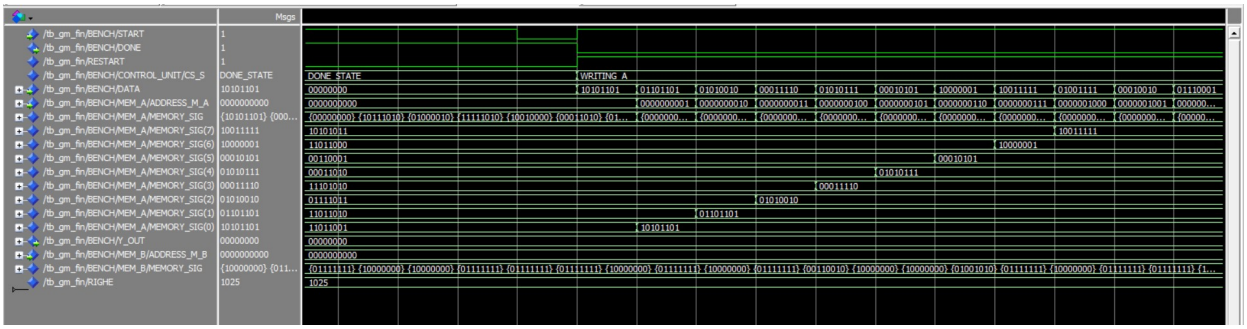

Figure 6: Restart of the circuit during the Functional simulation

2.2 Timing of the circuit

In Figure 7 and 8 is reported the by-hand timing of the load of samples in Memory A and B, compared with the one obtained from the Functional simulation, in Figures 9 - 12.

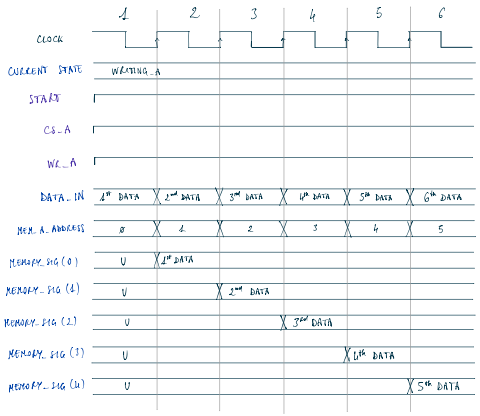

Figure 7: By-hand timing for the load of the samples in Memory A

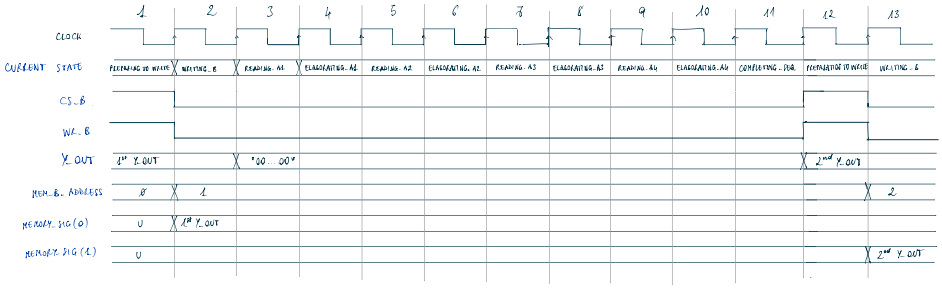

Figure 8: By-hand timing for the load of the samples in Memory B

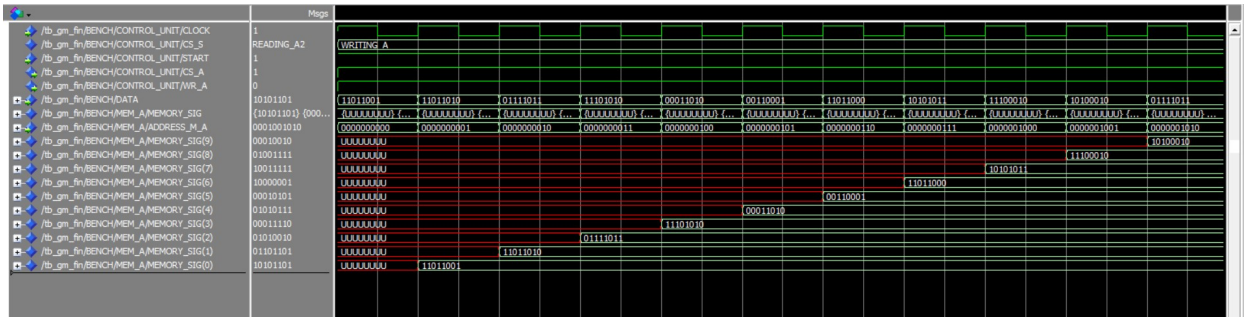

Figure 9: Load of the samples in Memory A

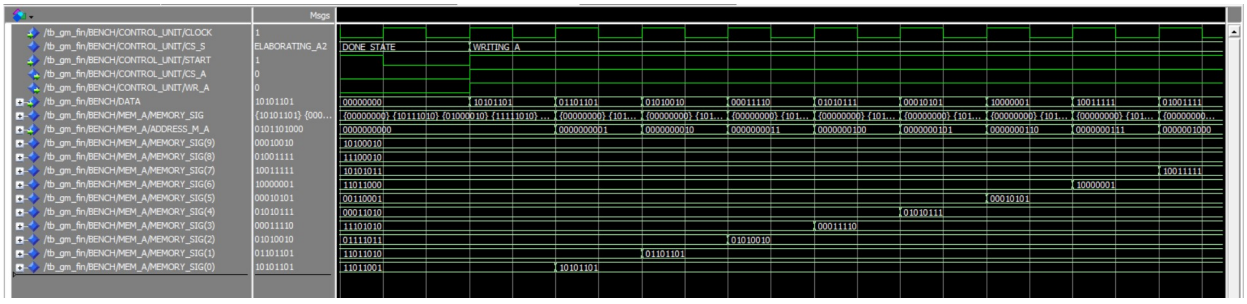

Figure 10: Load of the samples in Memory A after the circuit has restarted. The new values read from a second text file overwrite the previous ones.

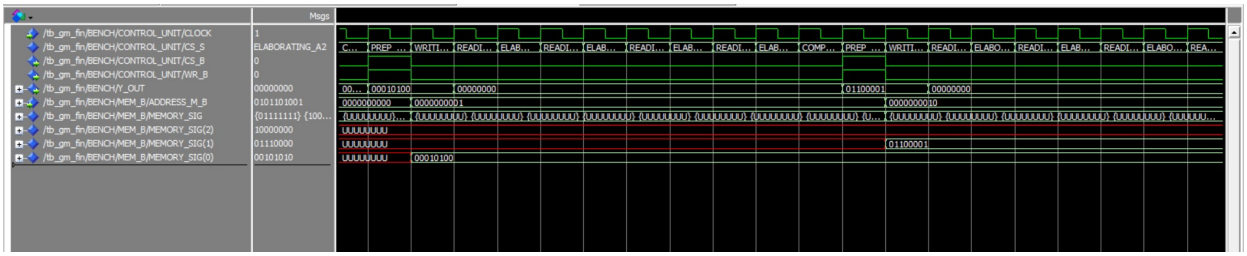

Figure 11: Load of the samples in Memory B, after they have been computed

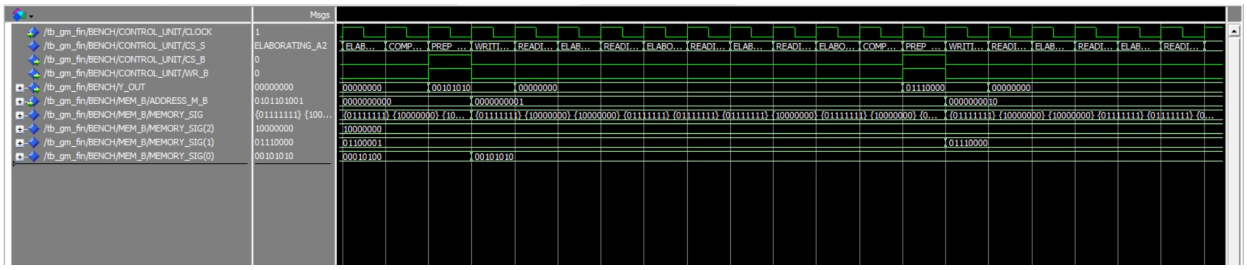

Figure 12: Load of the samples in Memory B after the circuit has restarted

We also report in Figure 13 the by-hand timing for the computation of Y_out, and, in Figures 14 - 15, the same timing observed on Modelsim.

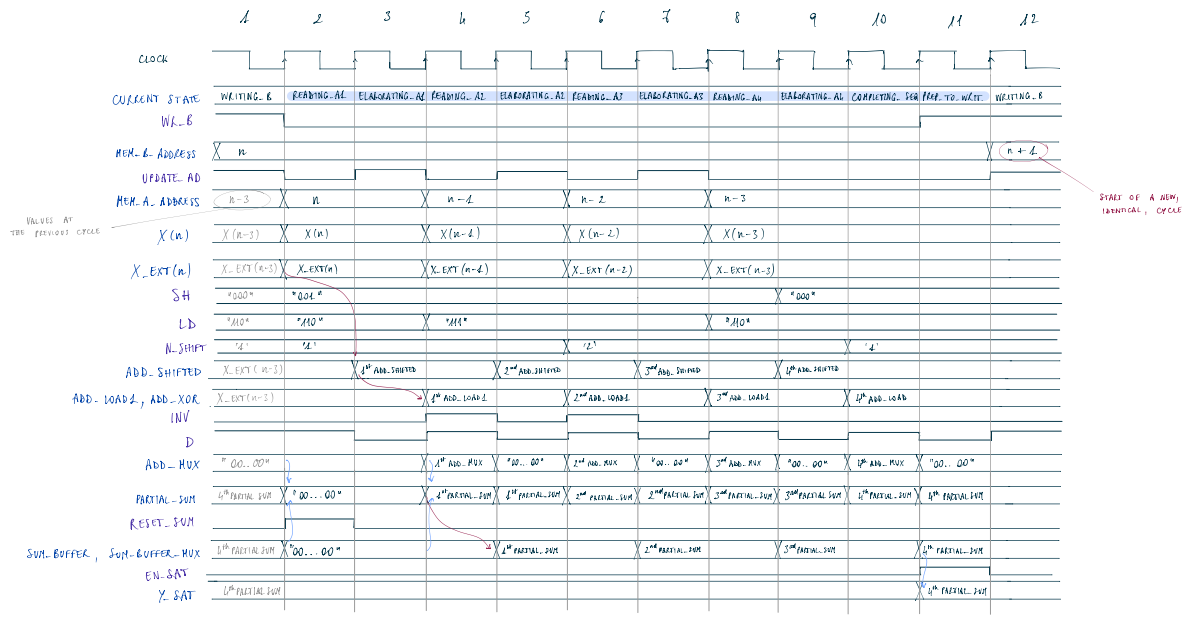

Figure 13: By-hand timing for computing Y_out

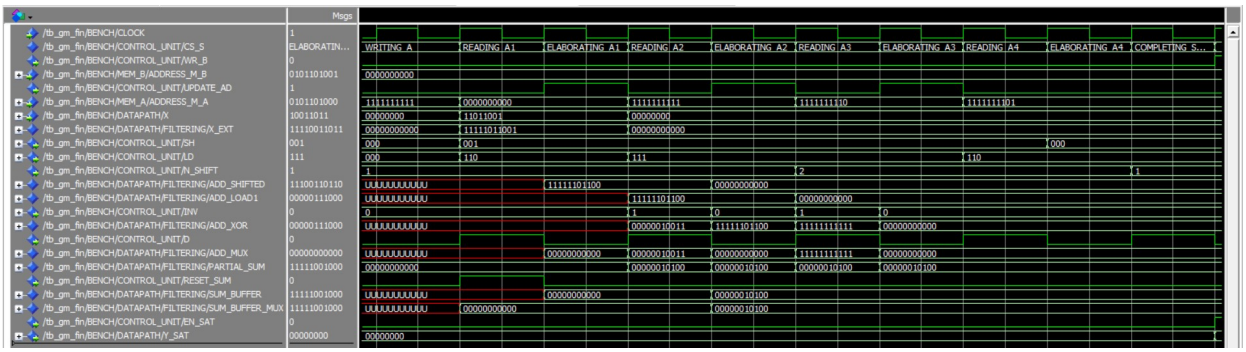

Figure 14: Computation of the first Y_out

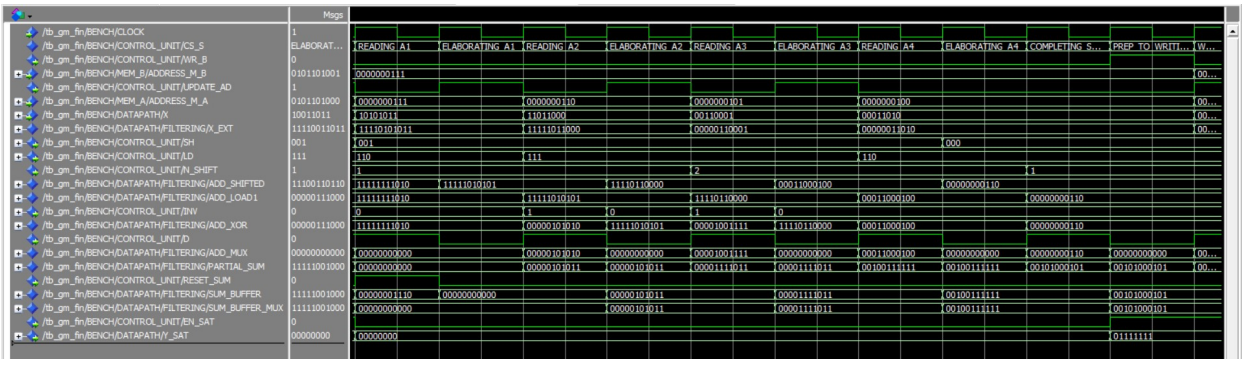

Figure 15: Computation of a generic Y_out

3 Timing simulation

Even if we were not requested to download our circuit on the FPGA board, it was still possible to synthesize it on Quartus, letting the software automatically assign the pins. So, we compiled our code after choosing a generic "Cyclone IV" FPGA device, in order to obtain the *.vho* and the *.sdo* files, i.e. the synthesized circuit code and the default propagation delays data, and to perform the Timing Simulation. The delay of the outputs with respect to the inputs, for a possible implementation of our circuit, is displayed by the following waveforms in Figure 16.

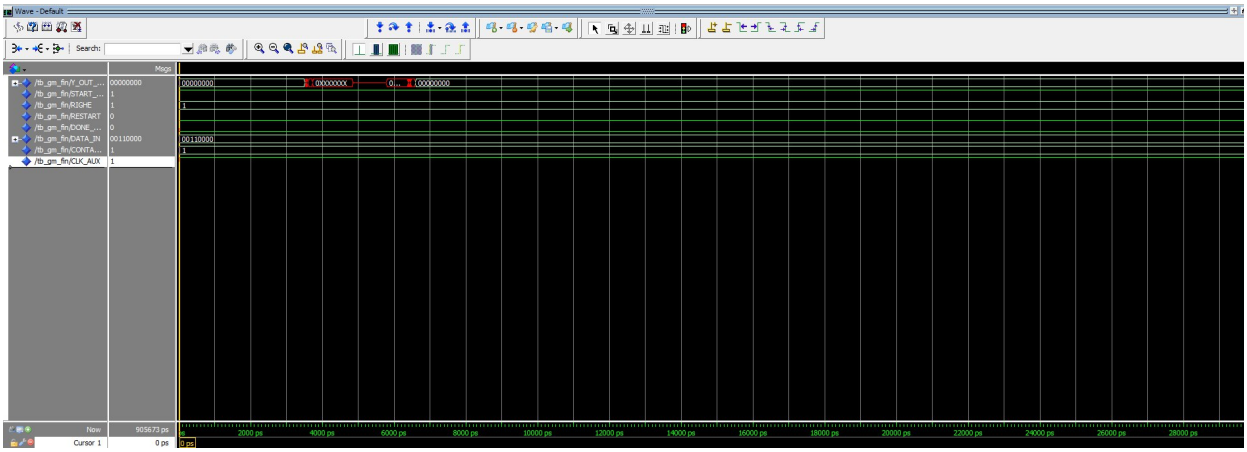

Figure 16: Timing simulation

4 Appendices

4.1 Main Code

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;


ENTITY ACTIVE_FILTER IS
  PORT(DATA: IN SIGNED(7 DOWNTO 0);
       CLOCK,START: IN STD_LOGIC;
       DONE: OUT STD_LOGIC);
END ACTIVE_FILTER;

ARCHITECTURE BEH OF ACTIVE_FILTER IS
SIGNAL RESET_SIG: STD_LOGIC;
SIGNAL X_SIG, Y_SAT_SIG: SIGNED(7 DOWNTO 0);
SIGNAL WR_A_SIG, WR_B_SIG, CS_A_SIG, CS_B_SIG, RD_A_SIG, RD_B_SIG:
   STD_LOGIC;
SIGNAL N_SHIFT_SIG: NATURAL;
SIGNAL INV_SIG: SIGNED(0 DOWNTO 0);
SIGNAL SH_SIG, LD_SIG : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL UPDATE_ADD_SIG, DONE_SIG, D_SIG, RESET_SUM_SIG, EN_SAT_SIG:
   STD_LOGIC;
SIGNAL MEM_A_HAS_WRITTEN_SIG, UNDERFLOW_SIG: STD_LOGIC;
SIGNAL MEM_A_ADDRESS_SIG, MEM_B_ADDRESS_SIG: UNSIGNED(9 DOWNTO 0);

COMPONENT CU IS
  PORT( START ,MEM_A_HAS_WRITTEN, CLOCK, RESET: IN STD_LOGIC;
  WR_A, RD_A, WR_B, RD_B, CS_A, CS_B, DONE, D, UPDATE_AD, RESET_SUM,
    EN_SAT: OUT STD_LOGIC;
  SH, LD: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
  INV: OUT SIGNED;
  N_SHIFT: OUT NATURAL);
END COMPONENT;


COMPONENT MEMORY_A IS
  PORT (ADDRESS_M_A: IN UNSIGNED(9 DOWNTO 0);
        UNDERFLOW: IN STD_LOGIC;
        DATA_IN_M: IN SIGNED(7 DOWNTO 0);
        CS_M_A,WR_M_A,RD_M_A,CLK_A: IN STD_LOGIC;
        Q_OUT: OUT SIGNED (7 DOWNTO 0));
END COMPONENT;

COMPONENT MEMORY_B IS
  PORT (ADDRESS_M_B: IN UNSIGNED(9 DOWNTO 0);
        DATA_OUT: IN SIGNED(7 DOWNTO 0);
        CS_M_B,WR_M_B,CLK_B: IN STD_LOGIC);
END COMPONENT;

COMPONENT DP IS
  PORT(X: IN SIGNED(7 DOWNTO 0);
       CLOCK, START, RESET: IN STD_LOGIC;
       WR_A, WR_B, CS_A, CS_B, RD_A, RD_B: IN STD_LOGIC;
       N_SHIFT: IN NATURAL;
```

```
51        INV: IN SIGNED (0 DOWNTO 0);
52        SH, LD : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
53        UPDATE_ADD, DONE, D, RESET_SUM, EN_SAT: IN STD_LOGIC;
54        MEM_A_HAS_WRITTEN_DP, UNDERFLOW: OUT STD_LOGIC;
55        MEM_A_ADDRESS, MEM_B_ADDRESS: OUT UNSIGNED(9 DOWNTO 0);
56        Y_SAT: OUT SIGNED(7 DOWNTO 0));
57 END COMPONENT;
58
59 BEGIN
60
61 CONTROL_UNIT: CU PORT MAP(START, MEM_A_HAS_WRITTEN_SIG, CLOCK, RESET_SIG,
       WR_A_SIG, RD_A_SIG, WR_B_SIG, RD_B_SIG, CS_A_SIG, CS_B_SIG,
62          DONE_SIG, D_SIG, UPDATE_ADD_SIG, RESET_SUM_SIG, EN_SAT_SIG, SH_SIG,
       LD_SIG, INV_SIG, N_SHIFT_SIG);
63
64 MEM_A: MEMORY_A PORT MAP(MEM_A_ADDRESS_SIG, UNDERFLOW_SIG, DATA, CS_A_SIG,
        WR_A_SIG, RD_A_SIG, CLOCK, X_SIG);
65
66 MEM_B: MEMORY_B PORT MAP(MEM_B_ADDRESS_SIG, Y_SAT_SIG, CS_B_SIG, WR_B_SIG,
        CLOCK);
67
68 DATAPATH: DP PORT MAP(X_SIG, CLOCK, START, RESET_SIG, WR_A_SIG, WR_B_SIG,
       CS_A_SIG, CS_B_SIG, RD_A_SIG, RD_B_SIG,
69          N_SHIFT_SIG, INV_SIG, SH_SIG, LD_SIG, UPDATE_ADD_SIG, DONE_SIG,
       D_SIG, RESET_SUM_SIG, EN_SAT_SIG,
70          MEM_A_HAS_WRITTEN_SIG, UNDERFLOW_SIG, MEM_A_ADDRESS_SIG,
       MEM_B_ADDRESS_SIG, Y_SAT_SIG);
71
72 RESET_SIG <= '1' WHEN (START = '1' AND START'EVENT AND DONE_SIG = '1')
       ELSE '0';
73
74 DONE <= DONE_SIG;
75
76 END BEH;
```

4.2 VHDL code for the Datapath

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5
6  ENTITY DP IS
7    PORT(--INPUT
8         X: IN SIGNED(7 DOWNTO 0);
9         CLOCK, START, RESET: IN STD_LOGIC;
10        --CONTROL SIGNALS
11        WR_A, WR_B, CS_A, CS_B, RD_A, RD_B: IN STD_LOGIC;
12        N_SHIFT: IN NATURAL;
13        INV: IN SIGNED (0 DOWNTO 0);
14        SH, LD : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
15        UPDATE_ADD, DONE, D, RESET_SUM, EN_SAT: IN STD_LOGIC;
16        --STATUS SIGNALS
17        MEM_A_HAS_WRITTEN_DP: OUT STD_LOGIC;
18        --OUTPUT
19        UNDERFLOW: OUT STD_LOGIC;
```

```
20        MEM_A_ADDRESS, MEM_B_ADDRESS: OUT UNSIGNED(9 DOWNTO 0);
21        Y_SAT: OUT SIGNED(7 DOWNTO 0));
22 END DP;
23
24 ARCHITECTURE BEH OF DP IS
25
26 SIGNAL ADDRESS_SIG_MEM_A_WR, ADDRESS_SIG_MEM_B_WR, ADDRESS_SIG_MEM_A_RD:
      UNSIGNED(9 DOWNTO 0) := (OTHERS => '0');
27 SIGNAL Y: SIGNED (10 DOWNTO 0);
28 SIGNAL MEM_A_HAS_WRITTEN_SIG: STD_LOGIC := '0';
29
30 COMPONENT MEM_A_WR_COUNTER IS
31   PORT(CLK, RST, EN, START: IN STD_LOGIC;
32   ADDR: OUT UNSIGNED(9 DOWNTO 0);
33   MEM_A_HAS_WRITTEN_M: OUT STD_LOGIC);
34 END COMPONENT;
35
36 COMPONENT MEM_B_WR_COUNTER IS
37   PORT(CLK, RST, EN, DONE: IN STD_LOGIC;
38         ADDR: OUT UNSIGNED(9 DOWNTO 0));
39 END COMPONENT;
40
41 COMPONENT MEM_A_READ_COUNTER IS
42   PORT(CLK, RST, EN: IN STD_LOGIC;
43   ADDR_B: IN UNSIGNED(9 DOWNTO 0);
44   ADDR: OUT UNSIGNED(9 DOWNTO 0));
45 END COMPONENT;
46
47
48 COMPONENT FILTERING_FUNCTION IS
49 PORT(X : IN SIGNED (7 DOWNTO 0);
50      inv: IN SIGNED;
51      LD, SH: IN STD_LOGIC_VECTOR(2 downto 0);
52      loc: IN NATURAL;
53      CLOCK, RESET, D, RST_SUM : IN STD_LOGIC;
54      Y : OUT SIGNED (10 DOWNTO 0));
55 END COMPONENT;
56
57 COMPONENT SATURATOR IS
58   PORT(EN: IN STD_LOGIC;
59        inp: IN SIGNED (10 DOWNTO 0);
60        outp : OUT SIGNED (7 DOWNTO 0));
61 END COMPONENT;
62
63
64 BEGIN
65
66 C0: MEM_A_WR_COUNTER PORT MAP(CLOCK, RESET, WR_A, START,
      ADDRESS_SIG_MEM_A_WR, MEM_A_HAS_WRITTEN_SIG);
67
68 MEM_A_HAS_WRITTEN_DP  <=  MEM_A_HAS_WRITTEN_SIG WHEN (WR_A = '1') ELSE
      '0';
69
70 MEM_A_ADDRESS <= ADDRESS_SIG_MEM_A_WR WHEN WR_A = '1' ELSE
71          ADDRESS_SIG_MEM_A_RD;
72
73 UNDERFLOW <= '1' WHEN( ADDRESS_SIG_MEM_A_RD > ADDRESS_SIG_MEM_B_WR) ELSE
```

```
74        '0';
75
76 C1: MEM_A_READ_COUNTER PORT MAP(CLOCK, RESET, UPDATE_ADD,
      ADDRESS_SIG_MEM_B_WR , ADDRESS_SIG_MEM_A_RD);
77
78 C2: MEM_B_WR_COUNTER PORT MAP(CLOCK, RESET, WR_B, DONE,
      ADDRESS_SIG_MEM_B_WR);
79
80 FILTERING: FILTERING_FUNCTION PORT MAP(X, INV, LD, SH, N_SHIFT, CLOCK,
      RESET, D, RESET_SUM, Y);
81
82 SAT: SATURATOR PORT MAP(EN_SAT, Y, Y_SAT);
83
84 MEM_B_ADDRESS <= ADDRESS_SIG_MEM_B_WR;
85
86 END BEH;
```

4.3 VHDL code for the Control Unit

```
1  LIBRARY IEEE;
2  USE ieee.std_logic_1164.all;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY CU IS
6    PORT( START ,MEM_A_HAS_WRITTEN, CLOCK, RESET: IN STD_LOGIC;
7      WR_A, RD_A, WR_B, RD_B, CS_A, CS_B, DONE, D, UPDATE_AD, RESET_SUM,
       EN_SAT: OUT STD_LOGIC;
8      SH, LD: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
9      INV: OUT SIGNED;
10     N_SHIFT: OUT NATURAL);
11 END CU;
12
13 ARCHITECTURE BEHAVIOR OF CU IS
14 TYPE STATE_TYPE IS (START_STATE, WRITING_A , READING_START, READING_A1 ,
      READING_A2 ,  READING_A3,  READING_A4, DONE_STATE, ELABORATING_A1,
      ELABORATING_A2,  ELABORATING_A3, ELABORATING_A4, COMPLETING_SEQUENCE,
       PREP_TO_WRITING, WRITING_B);
15 SIGNAL CS_S: STATE_TYPE := START_STATE;
16 SIGNAL FS: STATE_TYPE;
17 SIGNAL NUMBER_OF_PROCESS: UNSIGNED (9 DOWNTO 0):= (OTHERS => '0');
18 SIGNAL W: STD_LOGIC := '1';
19
20 BEGIN
21
22 MEM: PROCESS(CLOCK, RESET)
23     BEGIN
24         IF RESET = '1' THEN
25             CS_S <= WRITING_A;
26         ELSE
27             IF (CLOCK = '1' AND CLOCK'EVENT) THEN
28                 CS_S <= FS;
29             END IF;
30         END IF;
31     END PROCESS MEM;
32
33 CC1: PROCESS(CS_S, START, MEM_A_HAS_WRITTEN, W)
```

```
34      BEGIN
35          CASE CS_S IS
36              WHEN START_STATE =>
37                  IF (START = '1') THEN
38                      FS <= WRITING_A;
39                  ELSE
40                      FS <= START_STATE;
41                  END IF;
42
43              WHEN WRITING_A =>
44                  IF (MEM_A_HAS_WRITTEN = '1') THEN
45                      FS <= READING_A1;
46                  ELSE
47                      FS <= WRITING_A;
48                  END IF;
49
50              WHEN READING_A1 =>
51                  IF ( W = '1') THEN
52                      FS <= ELABORATING_A1;
53                  ELSE
54                      FS <= READING_A1;
55                  END IF;
56
57              WHEN ELABORATING_A1 =>
58                  IF (W = '1') THEN
59                      FS <= READING_A2;
60                  ELSE
61                      FS <= ELABORATING_A1;
62                  END IF;
63
64              WHEN READING_A2 =>
65                  IF ( W = '1') THEN
66                      FS <= ELABORATING_A2;
67                  ELSE
68                      FS <= READING_A2;
69                  END IF;
70
71              WHEN ELABORATING_A2 =>
72                  IF (W = '1') THEN
73                      FS <= READING_A3;
74                  ELSE
75                      FS <= ELABORATING_A2;
76                  END IF;
77
78              WHEN READING_A3 =>
79                  IF ( W = '1') THEN
80                      FS <= ELABORATING_A3;
81                  ELSE
82                      FS <= READING_A3;
83                  END IF;
84
85              WHEN ELABORATING_A3 =>
86                  IF (W = '1') THEN
87                      FS <= READING_A4;
88                  ELSE
89                      FS <= ELABORATING_A3;
90                  END IF;
```

```
91
92              WHEN READING_A4 =>
93                  IF ( W = '1') THEN
94                      FS <= ELABORATING_A4;
95                  ELSE
96                      FS <= READING_A4;
97                  END IF;
98
99              WHEN ELABORATING_A4 =>
100                 IF (W = '1') THEN
101                     FS <= COMPLETING_SEQUENCE;
102                 ELSE
103                     FS <= ELABORATING_A4;
104                 END IF;
105
106             WHEN COMPLETING_SEQUENCE =>
107                 IF (W = '1') THEN
108                     FS <= PREP_TO_WRITING;
109                 ELSE
110                     FS <= COMPLETING_SEQUENCE;
111                 END IF;
112
113             WHEN PREP_TO_WRITING =>
114                 IF (W = '1') THEN
115                     FS <= WRITING_B;
116                 ELSE
117                     FS <= PREP_TO_WRITING;
118                 END IF;
119
120             WHEN WRITING_B =>
121                 IF (W = '1') THEN
122
123                     IF NUMBER_OF_PROCESS < TO_UNSIGNED(1023,10) THEN
124                         FS <= READING_A1;
125                         NUMBER_OF_PROCESS <= NUMBER_OF_PROCESS + 1;
126                     ELSE
127                         FS <= DONE_STATE;
128                         NUMBER_OF_PROCESS <= (OTHERS => '0');
129                     END IF;
130
131                 ELSE
132
133                     FS <= COMPLETING_SEQUENCE;
134                     NUMBER_OF_PROCESS <= NUMBER_OF_PROCESS;
135
136                 END IF;
137
138             WHEN DONE_STATE =>
139                 FS <= DONE_STATE;
140
141             WHEN OTHERS =>
142                 FS <= CS_S;
143
144             END CASE;
145
146         END PROCESS CC1;
147
```

```
148 CC2: PROCESS (CS_S,FS)
149   BEGIN
150     CASE CS_S IS
151
152       WHEN START_STATE =>
153
154         EN_SAT <= '0';
155         RESET_SUM <= '0';
156         D <= '0';
157         WR_A <= '0';
158         RD_A <= '0';
159         UPDATE_AD <= '0';
160         RD_B <= '0';
161         WR_B <= '0';
162         CS_A <= '0';
163         CS_B <= '0';
164         SH <= "000";
165         LD <= "000";
166         INV <= TO_SIGNED(0,1);
167         DONE <= '0';
168         N_SHIFT <= 0;
169
170       WHEN WRITING_A =>
171
172         EN_SAT <= '0';
173         RESET_SUM <= '0';
174         D <= '0';
175         WR_A <= '1';
176         RD_A <= '0';
177         UPDATE_AD <= '0';
178         RD_B <= '0';
179         WR_B <= '0';
180         CS_A <= '1';
181         CS_B <= '0';
182         SH <= "000";
183         LD <= "000";
184         INV <= TO_SIGNED(0,1);
185         DONE <= '0';
186         N_SHIFT <= 1;
187
188       WHEN READING_A1 =>
189
190         EN_SAT <= '0';
191         RESET_SUM <= '1';
192         D <= '1';
193         WR_A <= '0';
194         RD_A <= '1';
195         UPDATE_AD <= '0';
196         RD_B <= '0';
197         WR_B <= '0';
198         CS_A <= '1';
199         CS_B <= '0';
200         SH <= "001";
201         LD <= "110";
202         INV <= TO_SIGNED(0,1);
203         DONE <= '0';
204         N_SHIFT <= 1;
```

```
205
206         WHEN ELABORATING_A1 =>
207
208           EN_SAT <= '0';
209           RESET_SUM <= '0';
210           D <= '0';
211           UPDATE_AD <= '1';
212           WR_A <= '0';
213           RD_A <= '1';
214           RD_B <= '0';
215           WR_B <= '0';
216           CS_A <= '1';
217           CS_B <= '0';
218           SH <= "001";
219           LD <= "110";
220           INV <= TO_SIGNED(0,1);
221           DONE <= '0';
222           N_SHIFT <= 1;
223
224         WHEN READING_A2 =>
225
226           EN_SAT <= '0';
227           RESET_SUM <= '0';
228           D <= '1';
229           WR_A <= '0';
230           UPDATE_AD <= '0';
231           RD_A <= '1';
232           RD_B <= '0';
233           WR_B <= '0';
234           CS_A <= '1';
235           CS_B <= '0';
236           SH <= "001";
237           LD <= "111";
238           INV <= TO_SIGNED(1,1);
239           DONE <= '0';
240           N_SHIFT <= 1;
241
242         WHEN ELABORATING_A2 =>
243
244           EN_SAT <= '0';
245           RESET_SUM <= '0';
246           D <= '0';
247           WR_A <= '0';
248           RD_A <= '1';
249           UPDATE_AD <= '1';
250           RD_B <= '0';
251           WR_B <= '0';
252           CS_A <= '0';
253           CS_B <= '0';
254           SH <= "001";
255           LD <= "111";
256           INV <= TO_SIGNED(0,1);
257           DONE <= '0';
258           N_SHIFT <= 1;
259
260         WHEN READING_A3 =>
261
```

```
262           EN_SAT <= '0';
263           RESET_SUM <= '0';
264           D <= '1';
265           UPDATE_AD <= '0';
266           WR_A <= '0';
267           RD_A <= '1';
268           RD_B <= '0';
269           WR_B <= '0';
270           CS_A <= '1';
271           CS_B <= '0';
272           SH <= "001";
273           LD <= "111";
274           INV <= TO_SIGNED(1,1);
275           DONE <= '0';
276           N_SHIFT <= 2;
277
278         WHEN ELABORATING_A3 =>
279
280           EN_SAT <= '0';
281           RESET_SUM <= '0';
282           D <= '0';
283           UPDATE_AD <= '1';
284           WR_A <= '0';
285           RD_A <= '1';
286           RD_B <= '0';
287           WR_B <= '0';
288           CS_A <= '1';
289           CS_B <= '0';
290           SH <= "001";
291           LD <= "111";
292           INV <= TO_SIGNED(0,1);
293           DONE <= '0';
294           N_SHIFT <= 2;
295
296         WHEN READING_A4 =>
297
298           EN_SAT <= '0';
299           RESET_SUM <= '0';
300           D <= '1';
301           UPDATE_AD <= '0';
302           WR_A <= '0';
303           RD_A <= '1';
304           RD_B <= '0';
305           WR_B <= '0';
306           CS_A <= '1';
307           CS_B <= '0';
308           SH <= "001";
309           LD <= "110";
310           INV <= TO_SIGNED(0,1);
311           DONE <= '0';
312           N_SHIFT <= 2;
313
314         WHEN ELABORATING_A4 =>
315
316           EN_SAT <= '0';
317           RESET_SUM <= '0';
318           D <= '0';
```

```
319           UPDATE_AD <= '0';
320           WR_A <= '0';
321           RD_A <= '1';
322           RD_B <= '0';
323           WR_B <= '0';
324           CS_A <= '1';
325           CS_B <= '0';
326           SH <= "000";
327           LD <= "110";
328           INV <= TO_SIGNED(0,1);
329           DONE <= '0';
330           N_SHIFT <= 2;
331
332         WHEN DONE_STATE =>
333
334           EN_SAT <= '0';
335           D <= '0';
336           RESET_SUM <= '0';
337           UPDATE_AD <= '0';
338           WR_A <= '0';
339           RD_A <= '0';
340           RD_B <= '0';
341           WR_B <= '0';
342           CS_A <= '0';
343           CS_B <= '0';
344           SH <= "000";
345           LD <= "110";
346           INV <= TO_SIGNED(0,1);
347           DONE <= '1';
348           N_SHIFT <= 0;
349
350         WHEN WRITING_B =>
351
352           EN_SAT <= '1';
353           RESET_SUM <= '0';
354           UPDATE_AD <= '1';
355           D <= '1';
356           WR_A <= '0';
357           RD_A <= '0';
358           RD_B <= '0';
359           WR_B <= '0';
360           CS_A <= '0';
361           CS_B <= '0';
362           SH <= "000";
363           LD <= "110";
364           INV <= TO_SIGNED(0,1);
365           DONE <= '0';
366           N_SHIFT <= 1;
367
368         WHEN COMPLETING_SEQUENCE =>
369
370           EN_SAT <= '0';
371           RESET_SUM <= '0';
372           UPDATE_AD <= '0';
373           D <= '1';
374           WR_A <= '0';
375           RD_A <= '0';
```

```
376           RD_B <= '0';
377           WR_B <= '0';
378           CS_A <= '0';
379           CS_B <= '0';
380           SH <= "000";
381           LD <= "110";
382           INV <= TO_SIGNED(0,1);
383           DONE <= '0';
384           N_SHIFT <= 1;
385
386         WHEN PREP_TO_WRITING =>
387
388           EN_SAT <= '1';
389           RESET_SUM <= '0';
390           UPDATE_AD <= '0';
391           D <= '0';
392           WR_A <= '0';
393           RD_A <= '0';
394           RD_B <= '0';
395           WR_B <= '1';
396           CS_A <= '0';
397           CS_B <= '1';
398           SH <= "000";
399           LD <= "110";
400           INV <= TO_SIGNED(0,1);
401           DONE <= '0';
402           N_SHIFT <= 1;
403
404         WHEN OTHERS =>
405
406           EN_SAT <= '0';
407           D <= '0';
408           RESET_SUM <= '0';
409           UPDATE_AD <= '0';
410           WR_A <= '0';
411           RD_A <= '0';
412           RD_B <= '0';
413           WR_B <= '0';
414           CS_A <= '0';
415           CS_B <= '0';
416           SH <= "000";
417           LD <= "110";
418           INV <= TO_SIGNED(0,1);
419           DONE <= '0';
420           N_SHIFT <= 0;
421
422       END CASE;
423           END PROCESS CC2;
424
425
426 END BEHAVIOR;
```