
Department of Electronics and Telecommunications
Bachelor degree in Electronic Engineering

# Digital Systems Electronics

# LAB 6: A simple digital filter.

Due date: May 06, 2024
- extended to: May 15, 2024
Delivery date: May 15, 2024

Group: 19

Contributions:

- Valtorta Alexander James
- Tedesco Angelo
- Urgu Sara

The members of the group listed above declare under their own responsibility that no part of this document has been copied from other documents and that the associated code is original and has been developed expressly for the assigned project.

# Contents

**0 Introduction** **2**
0.1 Aim of this project . . . . . 2
0.2 Content and structure of the document . . . . . 2

**1 Digital Filter** **3**
1.1 Datapath and Control Unit scheme . . . . . 3
1.2 Design entry . . . . . 6
1.3 Functional simulation . . . . . 7
1.4 Synthesis . . . . . 9

# 0 Introduction

## 0.1 Aim of this project

The purpose of this laboratory is to develop digital design skills, by dealing with a realistic problem starting from the functional specification in order to develop a circuit made of several blocks including memories, control unit and data path.

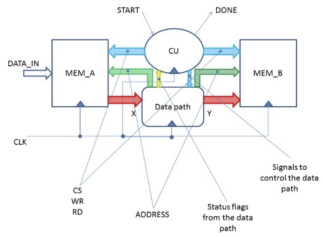
## 0.2 Content and structure of the document

The report is divided into four sections:
- The complete scheme of the Datapath and the Control Unit in order to implement a Printed Circuit Board for the design starting uniquely form the schemes.
- Design Entry, which describes the circuit to be implemented on the FPGA device, in particular it is indicated the overall architecture of the circuit, by reporting the name of the source file or files and the key role of each of them.
- Functional Simulation, which describes the testbench and the approach used to assess the completed design. (provided when requested)
- Synthesis, which shows the results of Modelsim functional simulation and that of the timing simulation.

# 1 Digital Filter

## 1.1 Datapath and Control Unit scheme

DIGITAL FILTER




$$Y(n) = -0{,}5\,X(n) - 2\,X(n-1) + 4\,X(n-2) + 0{,}25\,X(n-3)$$

PSEUDOCODE

```
CNT=0
SUM = 0
IF START = 1
    For CNT_A  0  to 1023
        X(CNT_A)=DATA_IN
    end for
    For CNT  0 to 1023
        shiFF8_dx (reg_S)
        reg_X =X(CNT)
        shift_dx (reg_X)
        sum = sum - reg_X
        shiFt_sx (reg_X1)
        sum = sum - reg_X1
        shift2_dx (reg_X2)
        sum = sum + reg_X2
        shiFt2_sx (reg_X3)
        sum = sum + reg_X3
        Y(CNT_A) = sum
        sum=0
    end for
    DONE = 1
```

Figure 1: Pseudocode of the circuit

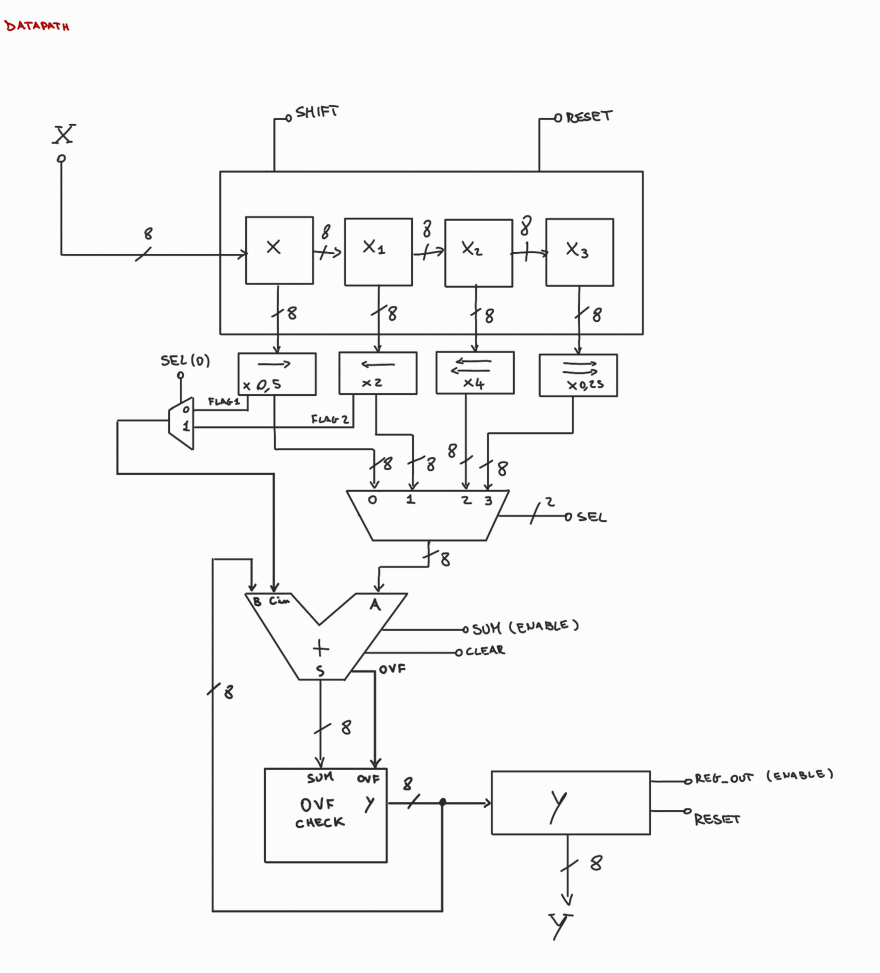



Figure 2: Design of the datapath

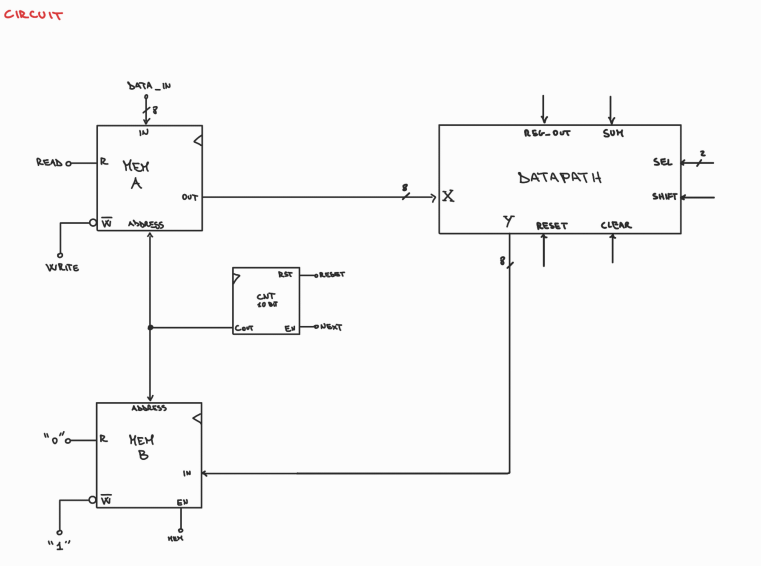



Figure 3: Design of the circuit

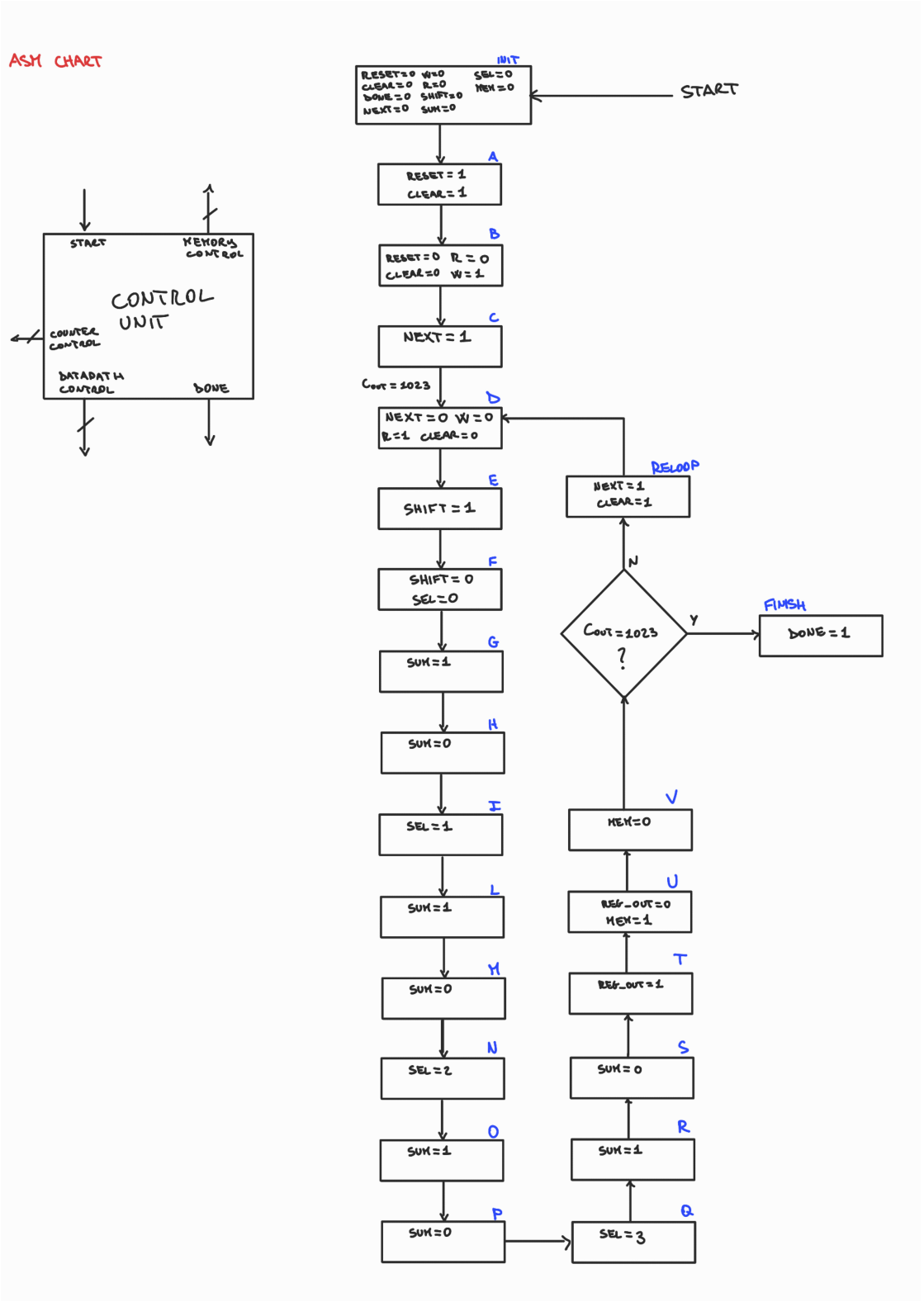



Figure 4: Algorithmic State Machine chart of the Control Unit

## 1.2 Design entry

In order to design the filter, the data structure used is an 8-bit shift register and a register to store the result only when it's ready. One multiplexer is employed to select which data needs to be added and another to set the carry-in of the adder to 1, in order to ensure correct two's complement subtraction. Finally, a 10-bit counter is utilized to select memory addresses.

Now going into more detail:

The memory is defined in *RAM.vhd* by means of a behavioural architecture in which the *Outputs* signal represent the memory content and is defined as a *memory_type*: an array type representing a memory matrix with 1024 locations, each capable of storing an 8-bit signed value. Then in the process are tested the two conditions provided in the assignment, in order to store the datum available at the *Data_in* input in the location pointed by *Address* or assign to the *Data_out* output the value stored at the location pointed by *Address*.

- The counter, which is defined in *n_bit_counter.vhd* using a T-type flipflop, defined in *T_flipflop.vhd*; both already described in previous assignments.

For the datapath it is necessary to define:

- The 8-bit register, which is defined in *reg8bit.vhd* using a D-type flipflop, defined in *d_flipflop.vhd*; both already described in previous assignments.

- The 8-bit shift register is defined in *reg_shift8.vhd*, using four registers as components (defined in *reg.vhd* and already described in previous assignments) and *s, s1, s2* as 8-bit signed internal signals. In the descriptive part of the structural architecture the registers are instantiated by mapping their inputs and outputs to those of the shift register, declared in the entity, in particular connecting them in a chain-like manner so that the output of the first register is the input of the second one and so on.

- The 8-bit wide 4-to-1 multiplexer is defined in *mux_4_1_8bit.vhd* with a behavioural architecture that makes use of a process which consider all the four possible cases and for each of them assign to the output $M$ the corresponding input, for any other case (which would mean an error) it assigns a 8-bit vector of zeros.

- The 8-bit adder for signed numbers, with indication of overflow, is defined in *signed_adder_8bit.vhd*, using as components the flipflop, the register and the 8-bit Ripple-Carry-Adder, defined respectively in *ff.vhd, reg.vhd, rca_8bit.vhd* and described for previous assignments (in particular the rca is described using 8 full adders *f_adder.vhd*, which are defined using 1-bit wide 2-to-1 multiplexes *mux_2_1_1bit.vhd*) and *r1, r2, Q1, Q2, Y* as internal signals. In the descriptive part of the structural architecture the inputs and outputs values of the components are mapped to those defined in the entity and the internal signals (setting the value of the *Enable* to '1').

- In order to implement the multiplications it is necessary to perform four different types of shifting:
1. the multiplication by (0.5) is performed by right shifting 1 bit, it is defined in *shift_dx.vhd*. In particular in the behavioural architecture the process checks of the input $R$ zero in order to assign the proper value to the output, because if it is not then it performs a right shift operation by one position bit (2's complement operation) and raises the *Sub_flag*; then it checks if the MSB has changed in order to restore it to the original value and update the output $Q$.
2. the multiplication by (-2) is performed by left shifting 1 bit, it is defined in *shift_sx.vhd*. In par-

ticular in the behavioural architecture the process checks of the input $R$ zero in order to assign the proper value to the output, because if it is not then it performs a left shift operation by one position bit and raises the *Sub_flag*; then it checks if the MSB has changed in order to restore it to the original value and update the output $Q$.
3. the multiplication by (+4) is implemented by performing a left shift operation by two positions, it is defined in *shift2_sx.vhd*.
4. the multiplication by (+0,25) is implemented by performing a left shift operation by two positions, it is defined in *shift2_dx.vhd*.

- Finally the whole Datapath is described in *datapath.vhd*, by mapping the inputs and outputs values of the components defined in *reg.vhd, reg_shift.vhd, mux_2_1_1bit.vhd, mux_4_1_8bit.vhd, signed_adder_8bit.vhd, shift_dx.vhd, shift_sx.vhd, shift2_sx.vhd, shift2_dx.vhd*, to those declared in the entity (*X, SEL, Clock, Reset, Shift, SUB, Clear, Sum, Reg_out, Y*) and the internal signals (*X_t, X1_t, X2_t, X3_t, S, S1, S2, S3, ADD, SM*).

The Control Unit is defined in *ControlUnit.vhd* following the structure elaborated in the Algorithmic State Machine, keeping in mind that the adder needs two clock cycles, thus the *sum* is performed accordingly.
- Lastly, the completed circuit is defined in *circuit.vhd*, where in the declarative part of the architecture the *RAM.vhd, datapath.vhd, n_bit_counter.vhd, ControlUnit.vhd* are defined as components and *X, Y, Cout, Reset, Wr, Rd, Clear, Mem, DONE, Nxt, Reg_out, Shift, Sum, SUB, SEL* as internal signals. In the descriptive part all the components can be connected following the design by mapping their ports with the appropriate internal signal and the inputs and outputs declared in the entity (*DATA_IN, START, Clock*).

## 1.3 Functional simulation

With the testbenches it is possible to check the functionalities of the completed designs.
- In the *tb_ram.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *RAM.vhd*), and the initialized internal signals for the simulation (*CStb, WRtb, RDtb, Clk, Add, Dtb, Qtb*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones , the simulation is performed through means of two processes: one for the simulation of the address, the input value and the selection of the read or write operation, and the other for the simulation of the clock. The former is performed by first simulating the writing (*CStb, WRtb* set to '1', *RDtb* seto to '0' and some values assigned to *Dtb, Add*), next the values for *Dtb, Add* are changed twice to simulate the storing of more data, then the simulation of the read operation (*WRtb, RDtb* toggle their value to '0' and '1' respective and a value is assigned to *Add* from which the reading needs to be done), next the value of *Add* are changed twice to simulate the reading of the data previously stored. All the commutations are performed by waiting a short amount of time in between. The latter is performed by assigning to the internal signal (*Clk*) first the logic value '0' and then '1', waiting a short amount of time in between.

- In the *tb_cnt.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *n_bit_counter.vhd*), and the initialized internal signals for the simulation (*Cout_t, En_t, Clock_t, Clear_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones and an integer valued is assigned to the generic $N$, the simulation is performed through means of two processes: one for the simulation of the enable and clear, and the other for the simulation of the

clock. The former is performed by assigning to the *En_t* signal first the logic value '1' then it toggles to '0', the same is done for the *Clear_t* signal, waiting in between each commutation. The latter is performed by assigning to the internal signal (*Clock_t*) first the logic value '0' and then '1', waiting a short amount of time in between.
- In the *tb_reg.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *reg.vhd*), and the initialized internal signals for the simulation (*R_t, Q_t, Clock_t, Reset_t, Enable_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of two processes: one for the simulation of the enable, the reset and input signal and the other for the simulation of the clock. The former is performed by assigning to the *Enable_t* signal first the logic value '1' and a 8-bit value to *R_t*, then changing its value and toggling *Enable_t, Reset_t*, waiting in between each commutation, to simulate all the different behaviours. The latter is performed by assigning to the internal signal (*Clock_t*) first the logic value '0' and then '1', waiting a short amount of time in between.

- In the *tb_reg_shift8.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *register1024.vhd*), and the initialized internal signals for the simulation (*Xin_t, X_t, X1_t, X2_t, X3_t, Clock_t, Reset_t, Shift_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of three processes: one for the simulation of the reset and input signal, the other for the simulation of the clock and the last one for the simulation of the shifting. The first one is performed by assigning to the *Xin_t* signal a 8-bit value, and changing it three times, the it is assigned to *Reset_t* first the logic value '1'then it toggles to '0', waiting in between each commutation, this is repeated enough times in order to simulate all the possible behaviours. The second is performed by assigning to the internal signal (*Clock_t*) first the logic value '0' and then '1', waiting a short amount of time in between. The last one is performed by assigning to the internal signal (*Shift_t*) first the logic value '1' and then '0', waiting a short amount of time in between.

- In the *tb_adder_8bit.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *signed_adder_8bit.vhd*), and the initialized internal signals for the simulation (*A_t, B_t, SUM_t, Clock_t, SUB_t, OVF_t, RST_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of two processes: one for the simulation of the subtraction option and the input signals and the other for the simulation of the clock. The former is performed by assigning to the *A_t, B_t* signals some values, then they are changed and the it is assigned to *SUB_t* first the logic value '1' then it toggles to '0', waiting in between each commutation, this is repeated enough times in order to simulate all the possible behaviours. The latter is performed by assigning to the internal signal (*Clock_t*) first the logic value '0' and then '1', waiting a short amount of time in between.

- In the *tb_shift_dx.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *shift_dx.vhd*), and the initialized internal signals for the simulation (*R_t, Q_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of a process, where different 8-bit values are assigned to the *R_t* signal waiting in between each commutation.

- The whole datapath is tested in the *tb_datapath.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *datapath.vhd*), and

the initialized internal signals for the simulation (*X_t, Y_t, SEL_t, Clock_t, Reset_t, Shift_t, SUB_t, Clear_t, Sum_t, Reg_out_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of two processes. The first one simulates the initial reset by assigning to the *Reset_t, Clear_r* signals first the logic value '1' then '0', then it simulates a 8-bit value for *X_t* followed by the simulation of the *Shift_t* signal (it toggles from '1' to '0' to store the value), next are the four simulations of the *SEL_t* to select all four inputs of the multiplexer and performing the addition (*Sum_t* toggles from '1' to '0'), next is the simulation of the *Reg_out_t* (which toggles from '1' to '0' to store the value in the output register), lastly the *SEL_t* is set to '00' (to restore the first input of the multiplexer) and the *Cleat_t* toggles from '1' to '0'; all the commutations are performed waiting some time in between each of them. This is repeated four times changing the 8-bit value of *X_t*. The second process is for the simulation of the clock, which is performed by assigning to the internal signal (*Clock_t*) first the logic value '0' and then '1', waiting a short amount of time in between.
- The Control Unit is tested in the *tb_controlunit.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *ControlUnit.vhd*), and the initialized internal signals for the simulation (*SEL_t, Cout_t, START_t, Clock_t, RESET_t, WR_t, RD_t, CLEAR_t, MEM_t, DONE_t, NXT_t, Reg_out_t, SHIFT_t, SUM_t, SUB_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of two processes: one for the simulation of the output of the counter and the other for the simulation of the clock. The former is performed by assigning to the *Cout_t* signal first the value '1111111111' (meaning that the counting is complete) then '0000000000' (to reset the counting), and doing it again, waiting in between each commutation. The latter is performed by assigning to the internal signal *Clock_t* first the logic value '0' and then '1', waiting a short amount of time in between.

- The complete circuit is tested in the *tb_circuit.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *circuit.vhd*), and the initialized internal signals for the simulation (*DATA_t, START_t, Clock_t*).
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of two processes: one for the simulation of the start and the input data and the other for the simulation of the clock. The former is performed by assigning to the *DATA_t* signal a 8-bit value then setting *START_t* to '1', next different values are assigned to *DATA_t* and *START_t* is set to '0'; waiting in between each commutation. The latter is performed by assigning to the internal signal *Clock_t* first the logic value '0' and then '1', waiting a short amount of time in between.

## 1.4 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run the Modelsim functional simulation.

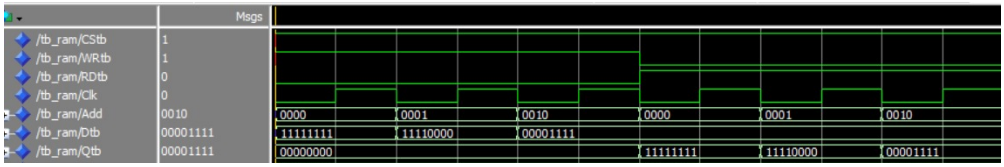



Figure 5: Modelsim functional simulation of the memory

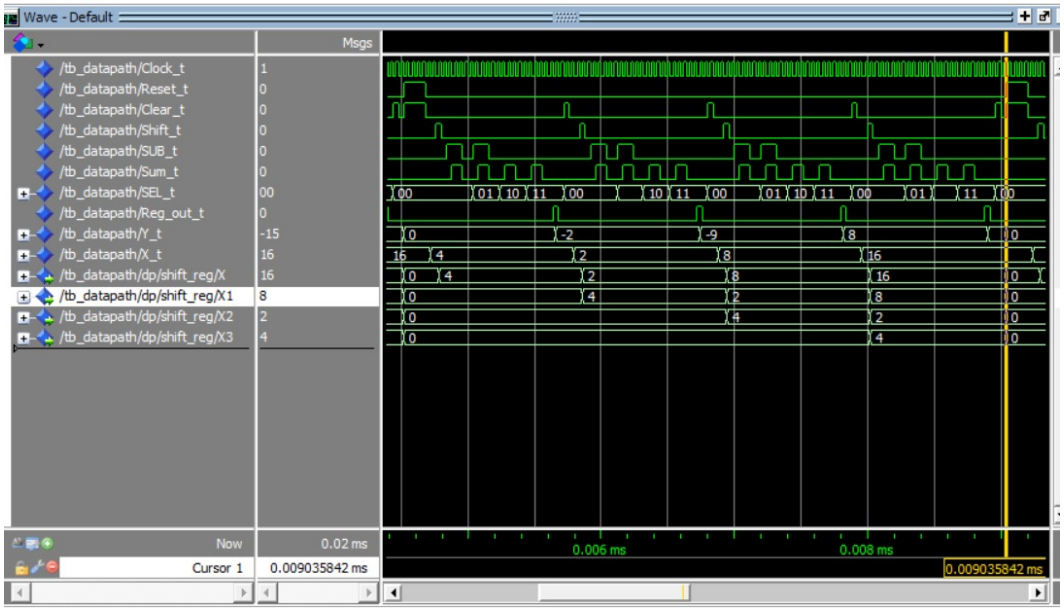



Figure 6: Modelsim functional simulation of the DP

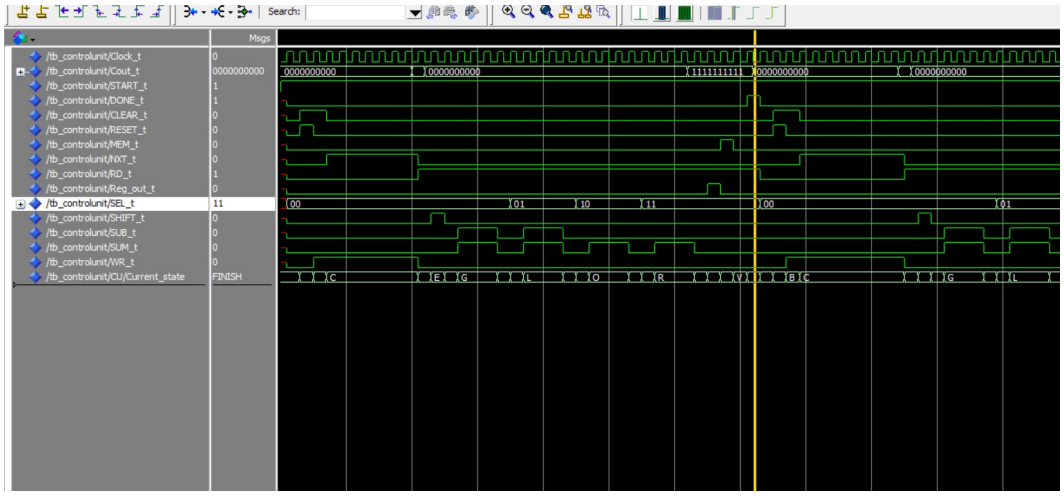



Figure 7: Modelsim functional simulation of the CU

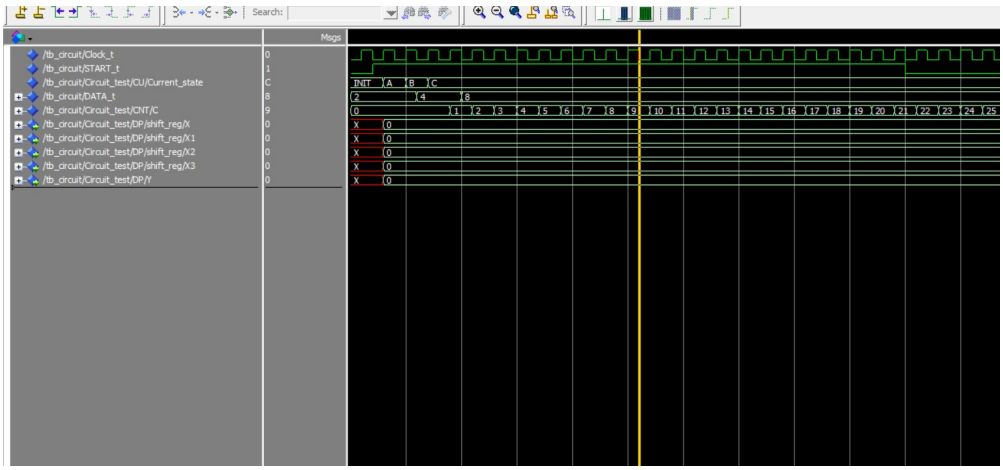



Figure 8: Modelsim functional simulation of the completed design

(it crashes before the counter can reach 1023)

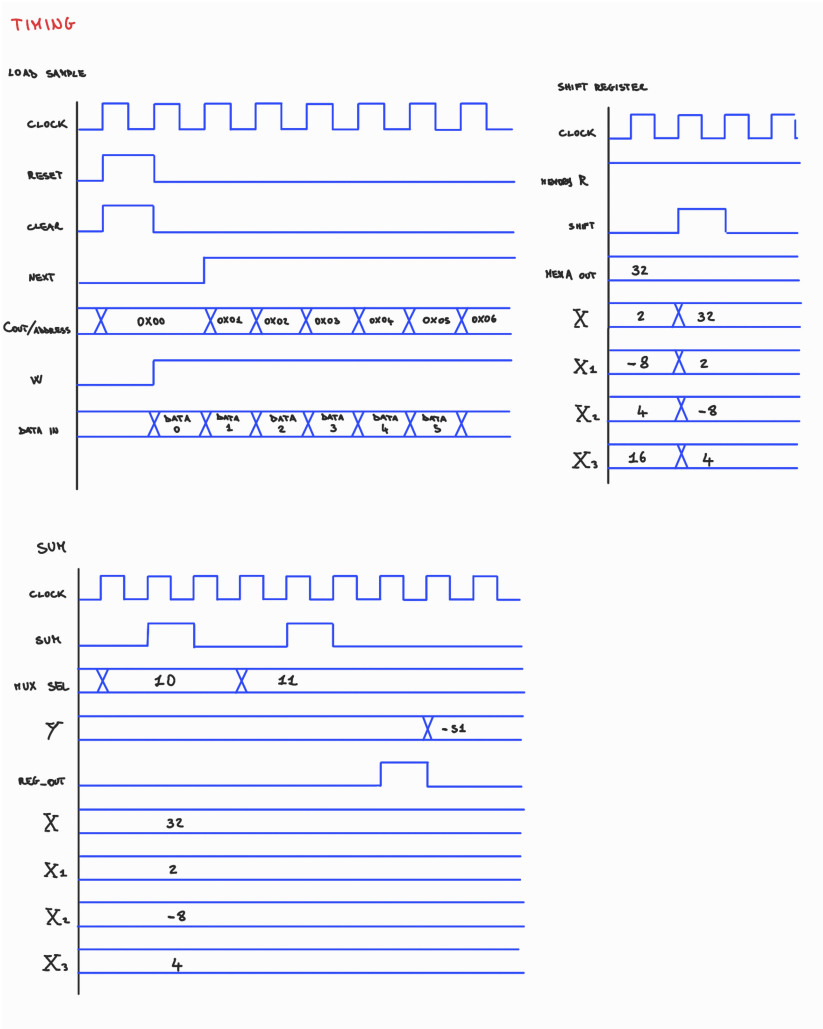



Figure 9: By-hand timing