Digital Systems Electronics Laboratory 06

Group 13

s295391 Giorgio Zoccatelli
s295567 Vittorio Macripò

Due Date:	May 6, 2024
Delivery date:	May 11, 2024
Instructor:	Professor Guido Masera

Politecnico di Torino
Accademic Year 2023/24

Contents

1 Introduction

The aim of this laboratory activity is to gather the experience acquired in the digital design on VHDL to implement a circuit working as a digital filter.

2 Design Entry

2.1 Preliminary steps

At the beginning of our work we have to prepare ourselves in order to have a clear idea of what we want to implement once we open Quartus Prime. In particular we will need:

- One ASM chart for the algorithm we want to implement
- One ASM chart outlining the relevant signals of the control unit
- One datapath schematic to have a clear idea of the connections between al the components
- A pseudocode of the algorithm to resume the workflow
- A wave analysis of the expected output

First of all we represent the two ASM chart to recognize which will be the functions of the control unit.

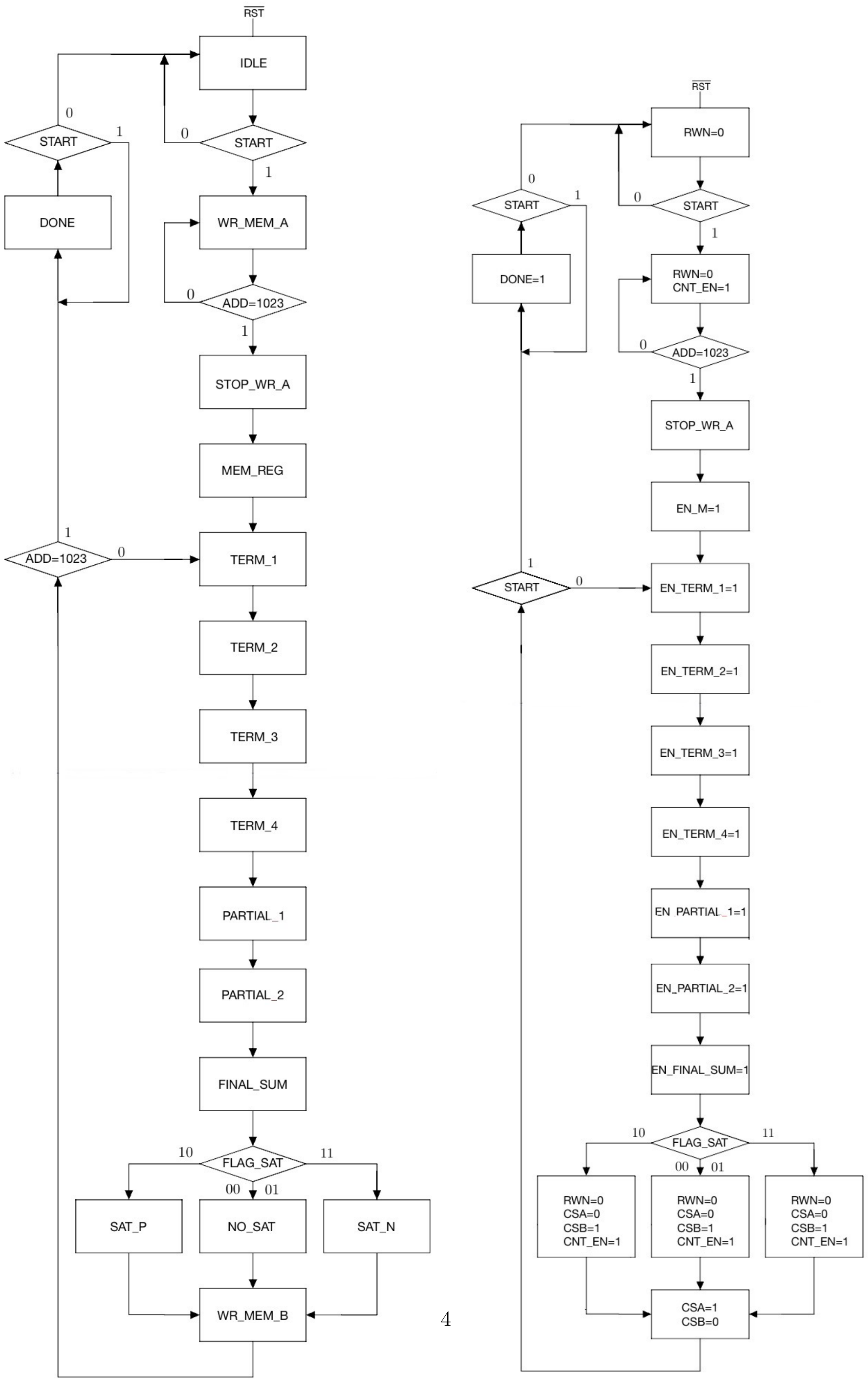


Figure 1: ASM charts

Then we can draw a detailed datapath showing the components used and the routes for the signals in the filter.

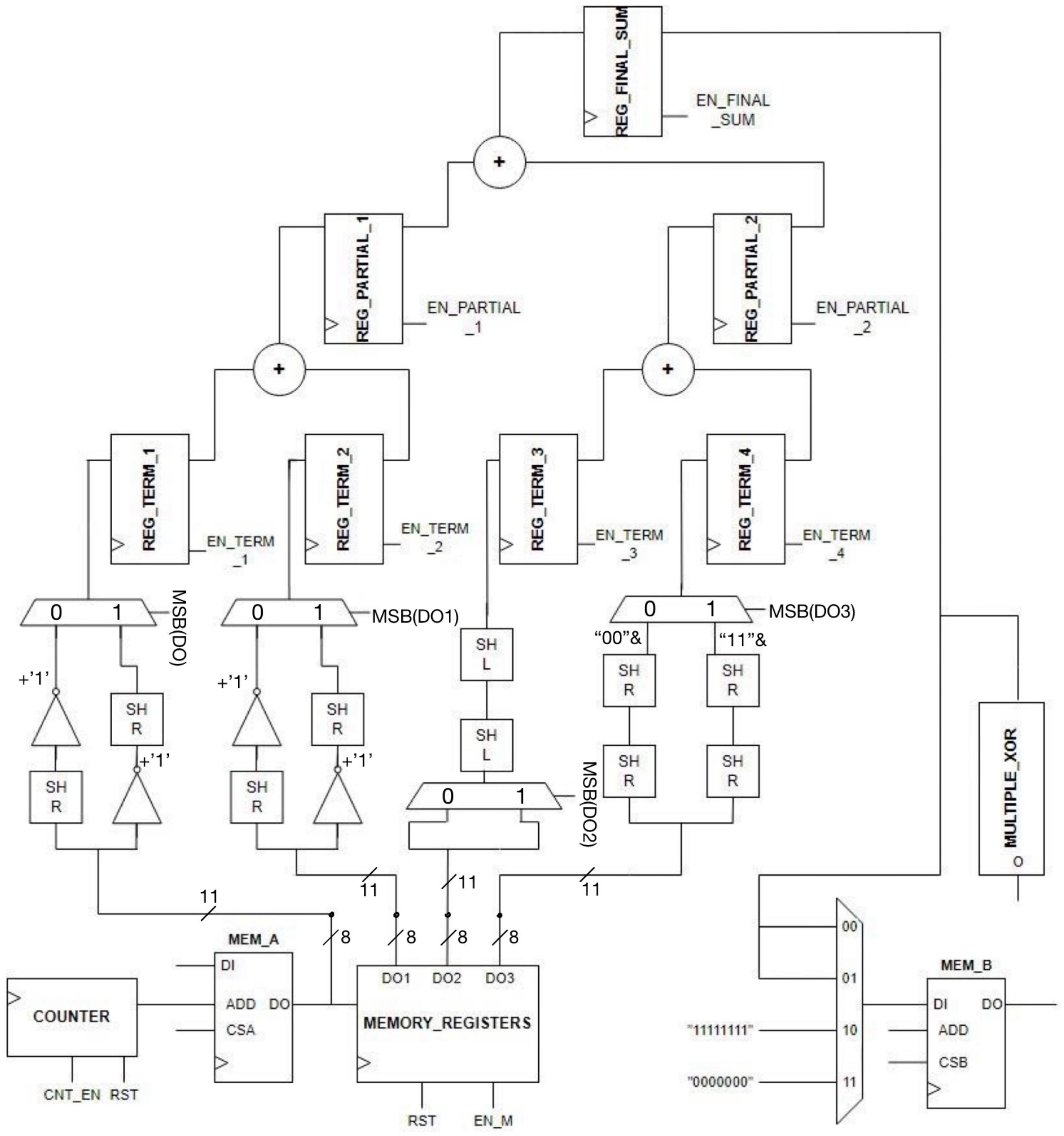


Figure 2: Datapath

Then in the following lines we summarized the tasks we want to implement in our circuit in order to obtain a digital filter through a pseudocode.

```
for add in 0 to 1023
    mem_a(add)<=data_in
end for

for add in 0 to 1023
    term_1<=f(INV,ADD,RS,mem_a(add))      --RS: Right Shift, INV: inverter
    term_2<=f(INV,ADD,LS,mem_a(add-1))    --LS: Left Shift
    term_3<=f(LS,mem_a(add-2))
    term_4<=f(RS,mem_a(add-3))

    partial_1<=term_1+term_2
    partial_2<=term_3+term_4
    final_sum<=partial_1+partial_2

    if final_sum>127
        data_out<=127
    elsif final_sum<-128
        data_out<=-128
    else
        data_out<=final_sum
    end if

    mem_b(add)<=data_out
end for
```

In practice we have a first phase for writing into *mem_a*, a second phase to read the elements of *mem_a* end elaborate them in order to satisfy the required function and a third phase to write the obtained output into *mem_b*. Finally we can draw a wave analysis for the expected outputs.

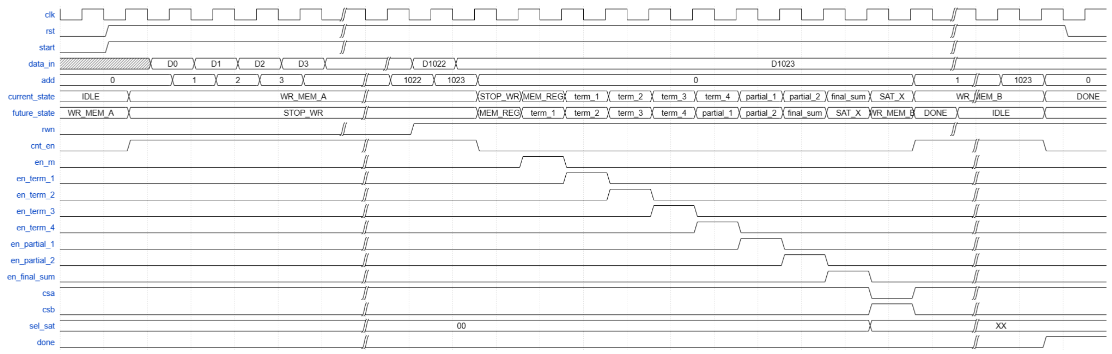


Figure 3: Expected Wave Analysis

2.2 Designs implemented

Once we have a clear idea of what we want to implement we can continue by writing our code for every needed design on Quartus Prime.

- **adder.vhd**: A generic adder has to be designed in order to add the partial terms of the final sum and to correctly manipulate the single terms as we will explain later. Please notice that the adder can sum two terms of 11 bits: this number represents the parallelism of our circuit calculated by considering the worst positive case and the worst negative case. After noticing that the values that can be represented in a 8 bit wide 2's complement notation are in a range [-128,127], we can calculate:
  - Worst positive case: $-0.5 \cdot (-128) - 2 \cdot (-128) + 4 \cdot (127) + 0.25 \cdot (127) \approx 860$
  - Worst negative case:$-0.5 \cdot (127) - 2 \cdot (127) + 4 \cdot (-128) + 0.25 \cdot (-128) \approx -862$

  So we can consider a parallelism of 11 bits to have a range [-1024,1023] and correctly represent all the values that can be stored in the memories.
- **counter.vhd**: A counter is needed to count from 0 to 1023 and succesfully fill every clock cycle a new address of *mem_a*.
- **memory.vhd**: This design represent the memory pinout given by the exercise text: it will be used of course both to represent *mem_a* and *mem_b*.
- **memory_registers.vhd**: This design is made for three shift registers, in fact the equation to implement requires to handle not only the value red at the current clock cycle but also the values red one past cycle, two past cycles and three past cycles. We also thought to set the default values of the registers to high impedance (*'Z'* data type) in order to distinguish the case when a vector of only zeros is passed to the register from the case when there is a reset or the enable is not high.
- **mux_4to1.vhd**: Generic N bit wide 4-to-1 multiplexer.
- **mux_2to1.vhd**: Generic N bit wide 2-to-1 multiplexer.
- **regn.vhd**: Generic N bit wide synchronous register.
- **multiple_xor.vhd**: This component gives as an output 1 if there's a underflow or overflow occurrence through a check of the first four bits of the value analyzed.
- **saturation.vhd**: This component has a double utility: through the use of the *multiple_xor.vhd* and the MSB of the value red it gives back a flag to determine the condition of overflow, underflow or regular value and it also manipulate the value through *mux_4to1.vhd* in order to set the value to -128 (underflow condition), 127 (overflow condition) or to the nominal value.
- **asm.vhd**: This design describes the real control unit of our digital filter. It is based on the two graphs already shown and has the function to decide according to the current state how to set different control signals.
- **datapath.vhd**: This design is the real heart of our circuit where all the main operations are made. Beyond the phase of filling *mem_a* with the counter and filling *mem_b* at the end which are pretty self-explanatory operations we want to underline the logic behind the equation elaboration. After changing the parallelism of our circuit from 8 bits to 11 bits for the reasons already explained we can manipulate the single terms.

  - Term 1: this term manipulates the present value red from *mem_a* and multiplies it by a factor of -0.5 so first of all we have to make sure that the data is divisible by 2 by setting the LSB to 0, then we have to distinguish two cases: if the MSB is equal to 0 (the value is positive) we have to shift the value toward right to divide it by two and then complement it and adding 1 to it in order to switch the sign. Otherwise if the MSB is equal to 1 (the value is negative) we have to invert the operations so we will first complement it and add 1 to it and then we will shift it toward right.
  - Term 2: this term manipulates the previous value red from *mem_a* and multiplies it by a factor of -2 so we have to distinguish two cases: if the MSB is equal to 0 (positive value) we have to shift the value toward left to obtain the double and then we have to complement it and add 1 to it in order to switch the sign while if the MSB is equal to 1 (negative value) we have to switch the operations, so first we complement and add 1 and then we shift toward left.
  - Term 3: this term manipulate the value red two past cycles from *mem_a* and multiplies it by a factor of 4 so we just have to shift the value toward left two times regardless of the value of the MSB.
  - Term 4: this term manipulate the value red three past cycles from *mem_a* and multiplies it by a factor of 0.25 so we have to distinguish two cases: if the MSB is equal to 1 (negative value) we have to shift the value two times toward right and then set the first two bits to 1 to maintain the sign, while if the MSB is equal to 0 (positive value) we just need to shift the value two times toward right.
  -

- **digital_filter.vhd**: This is the top entity of our circuit including *datapath.vhd* and *asm.vhd*.

3 Functional Simulation

After the design description we can implement a testbench for our simulation: this part is describen in the file *digital_filter_tb.vhd*, it is pretty simple because we just have to set the start and reset functions, the main particularity is that we want to import an external *.txt* file with the 1024 values to put into *mem_a* so we have to use some new VHDL syntax. In addiction to that we implemented the following piece of code in Python in order to generate the *.txt* file automatically:

```python
import random

with open('file_input.txt', 'w') as output_file:
  for _ in range(1024):
    random_binary_string = ''.join(random.choices(['0', '1'], k=8))
    output_file.write(random_binary_string + '\n')
```

4 Synthesis

In the Synthesis we can view the outputs of our simulations. First of all we can take a look to the RTL Viewer and to the State Machine Viewer to make some comparisons with the preliminary considerations.

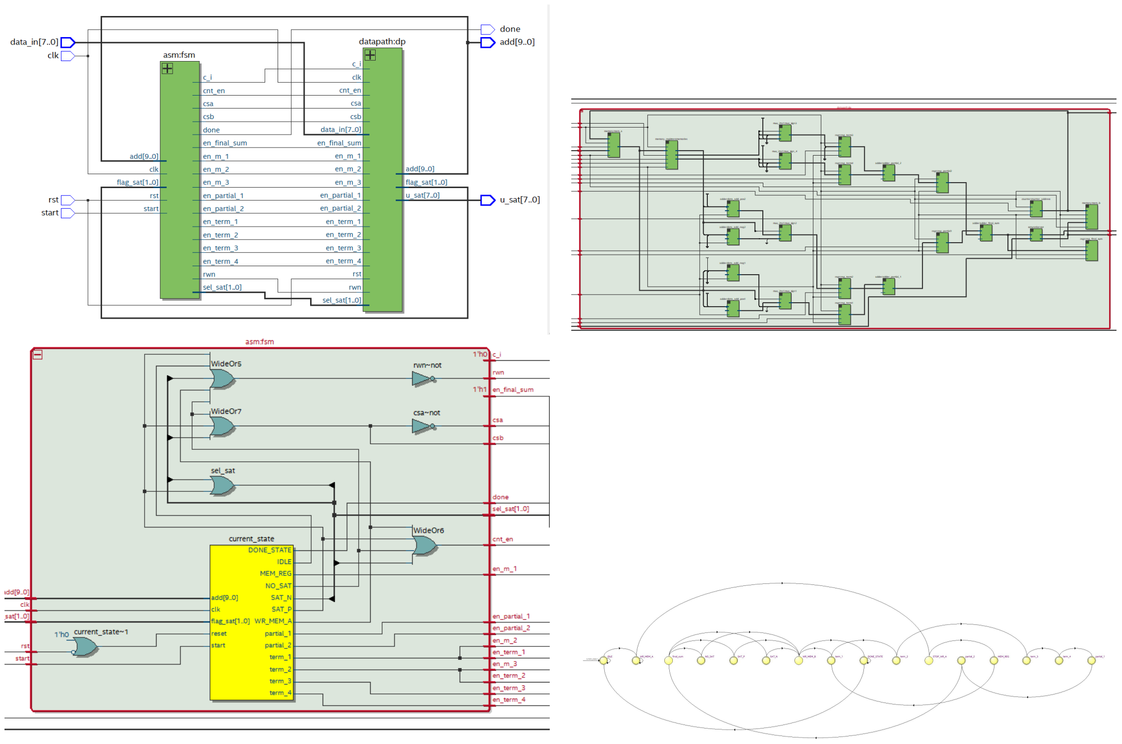
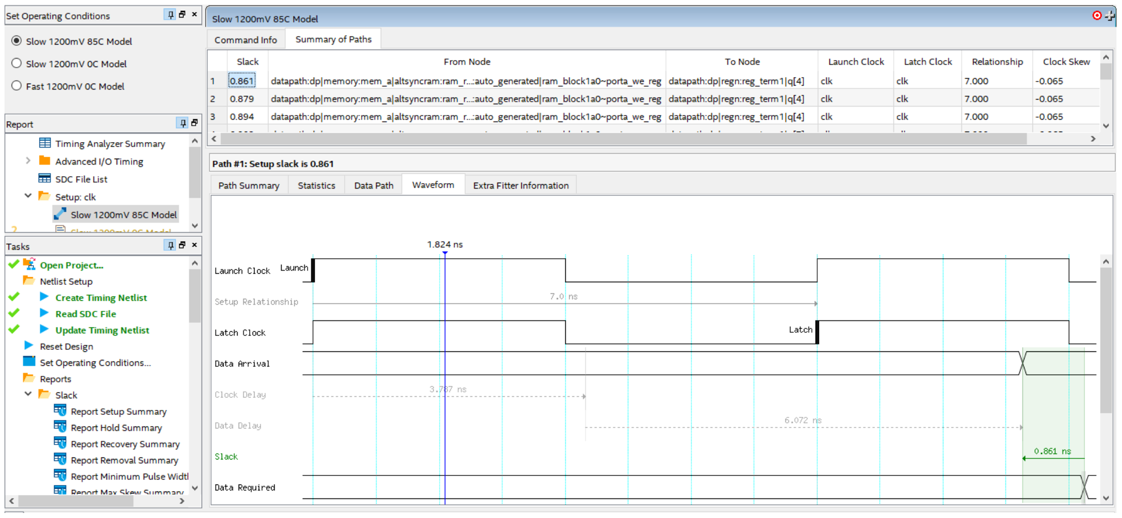


Figure 4: RTL and State Machine Viewers

Then we can have a look to the time analysis which gives us informations in terms of internal delays in our circuit: in our case the software gives us a maximum operative frequency $F_{max} = 162,89$ MHz.



Slow 1200mV 85C Model

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew
1	0.861	datapath:dp|memory:mem_a|altsyncram:ram_r...:auto_generated|ram_block1a0~porta_we_reg	datapath:dp|regn:reg_term1|q[4]	clk	clk	7.000	-0.065
2	0.879	datapath:dp|memory:mem_a|altsyncram:ram_r...:auto_generated|ram_block1a0~porta_we_reg	datapath:dp|regn:reg_term1|q[4]	clk	clk	7.000	-0.065
3	0.894	datapath:dp|memory:mem_a|altsyncram:ram_r...:auto_generated|ram_block1a0~porta_we_reg	datapath:dp|regn:reg_term1|q[4]	clk	clk	7.000	-0.065

Path #1: Setup slack is 0.861

Figure 5: Time Analysis

We should now have a look at the wave analysis on ModelSim but after several tries and research we didn't succeed in correctly upload it because we didn't manage to handle the relation between the testbench and the *.txt* file of the values generated. Despite the lack of this useful graph we documented in many ways our work and we are pretty confident that thanks to the effort we have put into it our solution should be very near to the expected one.