SEA FINAL PROJECT:

POST-OP MONITORING SYSTEM

06/02/2023

Andrea Carboni 50036660

Matteo Comini 50035432

SUMMARY

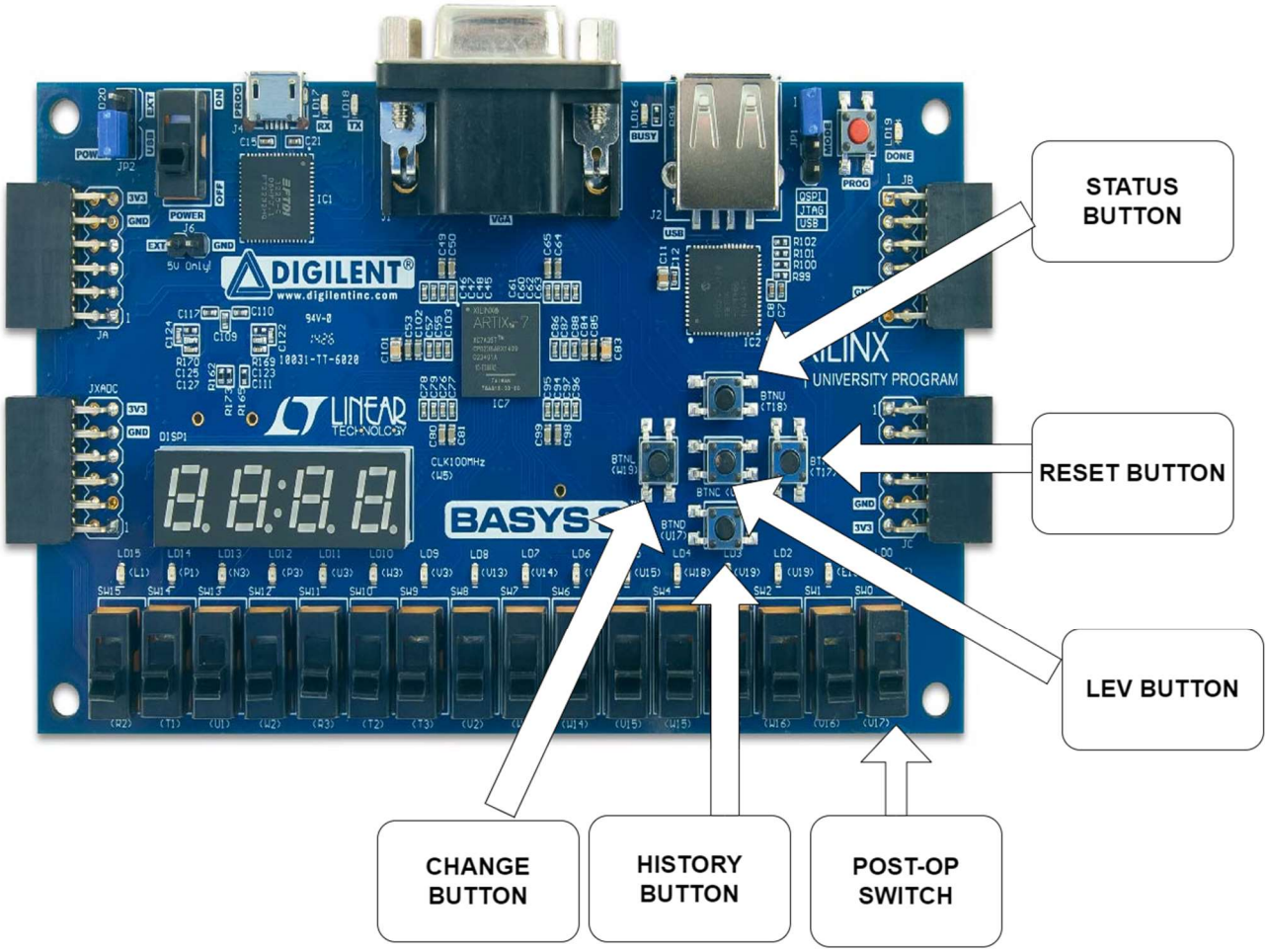
1) Introduction - page 3
2) SYSTEM DEMO – page 4
3) Design decisions – page 5
4) Main modules – page 6
5) State Machines – page 10
6) Schematic and testbench – page 39
7) Resource Utilization – page 57
8) Division of task – page 64
9) Warnings – page 65

1. Introduction

The project consists of a system for monitoring a patient in post-operative situations. Monitoring is done in 5 stages that handle the alarm situation differently.

Board inputs come in through buttons and switches, and have been assigned in this way:

- LEV BUTTON: **V18**
- STATUS BUTTON: **T18**
- CHANGE BUTTON: **W19**
- HISTORY BUTTON: **V17**
- RESET BUTTON: **T17**
- POST-OP SW: **SW0**



Project was made with Vivado 2020.2.

2. SYSTEM DEMO

The system works as follows:

Initially the 7-segment display will show the words "WAIT" and the only button to work will be the **RESET** button, all until we raise the switch that will bring the system into the monitoring phase and activate the rest of the buttons. At the beginning of the monitoring phase the word "WAIT" present in the 7-segment display will be replaced by the word "POST", from now on the buttons can be used and will work.

In the post-operation phase we will click the **LEV** button which will cause the words "the value of the bag increase" to appear on the screen and increase the level of liquid in the bag. The **STATUS** button will cause the words "LXXX" to appear in the display, which will tell us how much liquid is in the bag, when we let go of the button the value will be replaced by the previous words "POST". The **CHANGE** button will cause the words "the doctor has changed the bag" to appear in the screen and reset the level of liquid in the bag to zero.

The "HISTORY" button will allow us to see the sampling of the last time the "LEV" button was pressed, showing the bag level, stage and time. If a bag change occurs or the switch is lowered, there will be a reset of the value.

Clicking the **LEV** button could bring us to the alarm stage, this will turn on various LEDs based on the stage we are in and print the following phrase on the screen "alarm on: the drainage level is XXX in stage Y at hour KK:KK." Once we get to the error stage only the **LEV** button and **HISTORY** will work, which will not change their operation. Once **HISTORY** is pressed it will unlock the use of the other two buttons that initially did not work.

To exit the alarm phase, the doctor will have to press the **HISTORY** button initially and later unlock the **CHANGE** button, which if pressed will empty the bag and turn off the LEDs that were lit in the error phase, notifying everything with the phrase "the doctor has changed the bag" in the screen.

Following the passage of 24H the bag is automatically changed and will be notified all through the writing on the screen "automatic bag change (24H)", it will happen even if we are in the error phase. Once the 24H has passed we will move on to the next stage and the alarm threshold will also change.

Whenever the **RESET** button is pressed it will always return to the initial state.

3) Design decisions

In our design within the system core, the clear0 signal (clear that is generated when 24h passes) is the one that has the highest precedence over virtually every action: in fact, if the conta24h notifies that 24h has passed, the system immediately takes over communications with the spi to immediately perform stage advancement, with counter resets (such as the level one) and sometimes with screen printouts.

Another design choice is related to the history button: in fact it every time the switch goes down and the bag empties it deletes the information it contains, this is because it would be related to measurements that are no longer interesting, in fact when it goes down and raises the sw another measurement so the old sampling would have been wrong, same reasoning for the bag change.

We also wanted to specify that many states of the fsm that may seem superfluous are actually implemented and used to make sampling at the correct instants and to make the timings work the right way for the board to work the right way.

An other unusual design choice was to put in the fsm_stage in the IDLE state the error threshold at 1, this is because initially the threshold was not set in time and the system would go in the first instant of the post-op in alarm and we did not want that to happen, both for correctness of the system but also for a matter of false notification of the alarm on the screen.

In addition, as for the status button, we decided not to connect it to a debouncer because we want that when it is pressed, it instantly shows the level value in the 7-segment display, remaining displayed until the doctor wants to let go of the button.

4)Main Modules

Doctor

The doctor module has two debouncers, called db2 and db3, which serve for the history_button, which we use to view the patient's log, and the change_button, used to change the patient's bag. In addition to this we assign the switch and the button_status, so that when it remains pressed it displays the bag level on the screen.

SYSTEMCORE

The systemcore consists of several submodules, and a debouncer used for the data we receive from the SPI:

ContaLev

Contalev will keep track of the bag level by incrementing it each time the button_lev is pressed, and resetting the value if we lower the switch or there is a bag change

Conta24h

Count24h is for us to keep track of the passage of 24 hours, which will give us a signal that we can make the bag change every 24 hours and keep track in case we need the time.

On the basys3 demo, one day corresponds to about a minute and a half.

Led_Logic

Led_Logic creates the logic of raising the LEDs based on the stage and whether we have the alarm high or not.

FSM_SystemCore

FSM_SystemCore creates the general systemcore logic.

Based on whether buttons are pressed or alarms are raised, it will create output signals for us to operate the system.

FSM_Alarm

FSM_Alarm is created to handle the alarm logic after the history_button is pressed. Basically, using only the FSM_Systemcore, we could not handle the logic of the doctor being able to "unlock" the change and status buttons after he presses the

history, so this way we are able to handle it by generating a signal that will indicate that we are in this situation.

Soglia

Soglia will generate the alarm in cases where the level exceeds the threshold, and it will also handle the logic that when we are in an alarm situation and the doctor has pressed the history_button and , therefore, the patient is under control, it will also generate a clear signal in case the change_button is pressed.

FSMGC

FSMGC will handle the SPI communication, and select the slave we want to communicate with.

GCUart

GCUart will handle the communication of the uart, generating data for us that will tell us what we should go to print on the screen.

FSM_Stage

FSM_stage creates the logic of the passage of the stage, based on the passage of days.

CFSM

CFSM passes the received data of the SPI according to the slave to the respective signals.

display7segmenti

display7segments creates the labels that we are going to print on the 7-segment display, such as the label "WAIT" when we have the switch low, the label "POST" when we raise the switch, and the level value when we go to press the status_button . For the logic, we used the modules used in the ESD course.

Contalettere

Contalettere is used for sending one letter at a time for screen printing, the busy input generated by the UART is used.

FSM_Scritta

This module contains logic using a state machine that we use for on-screen printing of certain messages, such as alarm and post-op situations, because we want them to be written once on the screen, and since both switch and alarm are signals that usually stay high, they could not see whether they were just high or not, so we

generate this signal that stays high for one clock cycle and will be used by GCUart to print the message.

Simple_dual_one_clk

the simple_dual_one_clk module we use it to sample and print when we click history the last lev action performed, printing both the stage and the time.

Initially it was supposed to be a many-cell ram but the non-functioning of the printouts forced us to modify and remove some modules that had lost their meaning, this one in particular was modified.

DISPATCHER

The dispatcher governs communications within our system.

Within it we find various signals, the clock and reset, spi_r,spi_d,spi_e which respectively serve to indicate when the spi is ready, when it has finished communicating, and a signal to activate it. Then we find spi_code, spi_data1 and spi_data2 which are respectively the input and output data from the SPI, the 3 buttons status history and change, a series of signals that we will need in the uart_coding module such as stage, date, level, cntL, o_massimo, transmit instead we will need to warn the UART to start the communication and it will also be passed in the tr_module in order to start displaying on screen the writes. In input we also have a clear signal, which will be the one that will basically alert us that 24h has passed.

In output on the other hand we will need signals like o_status, o_history, o_change, o_clear to make an assignment if they happened as if they were through a cable that we will need in the systemcore.

The last outputs are busy and ready, signals from the UART that we will need in the systemcore, also we will have Out_Data_Serial which will be the signal that we will pass into the XDC so that the writing to the screen can happen.

Inside it we find the top_dut module, which would be the UART provided by the SEA course, UART_encoding, which will do the ascii encoding of what we want to print on the screen, such as if the change button should be pressed, on the screen we will be provided with this detail.

We also have the "tr_module", a module that was provided to us in the September call in which we participated by a colleague within our group. The module is used for serial transmission and to properly display messages on the screen.

Also inside we have SPI2Slave.

The SPI we use is the one provided by the SEA course.

It has two interfaces, which are top_lev and top_post.

Top_lev will send the lev_button value to the master, while top_post will send a stage increment signal to the master.

The timescanner logic is done in the systemcore, and this helps us know when to stay listening in the SPI and send the system to the next stage with a high bit.

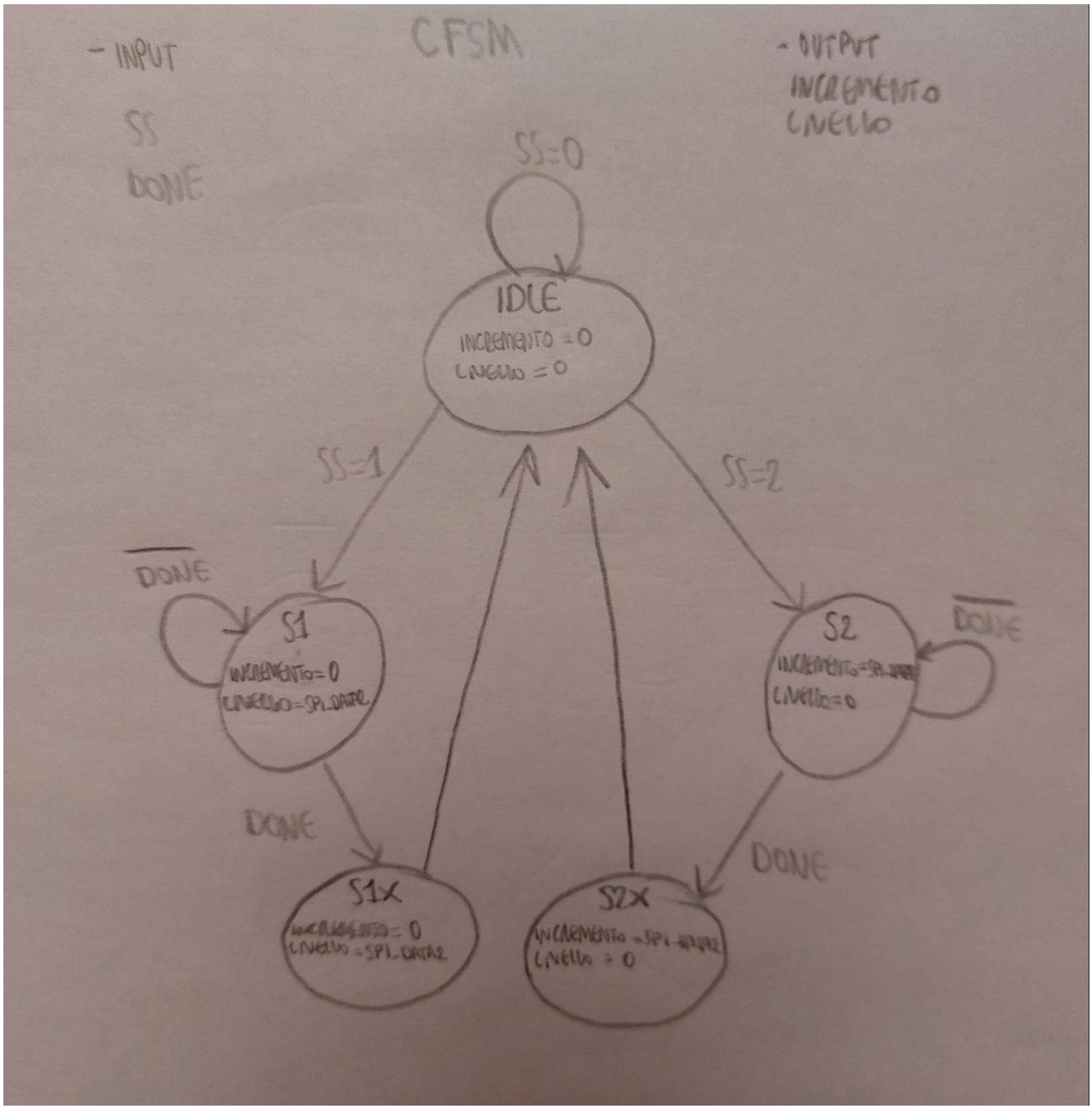
Topmodule

The topmodule will be the module that connects the various submodules of the system: system_Core, doctor and the Top_Dispatcher.

The signals we find inside it just the clock and reset, lev, history, change, status, which are the respective buttons of our system, sw, which indicates the switch we use to tell when a patient enters post-op, led, which will be the LEDs on the board that will rise according to the stage and alarm, seg and an which will be the outputs for the 7-segment display, and Out_Data_Serial which will be the output that will transfer the messages in the log screen

5) State machines

CFSM



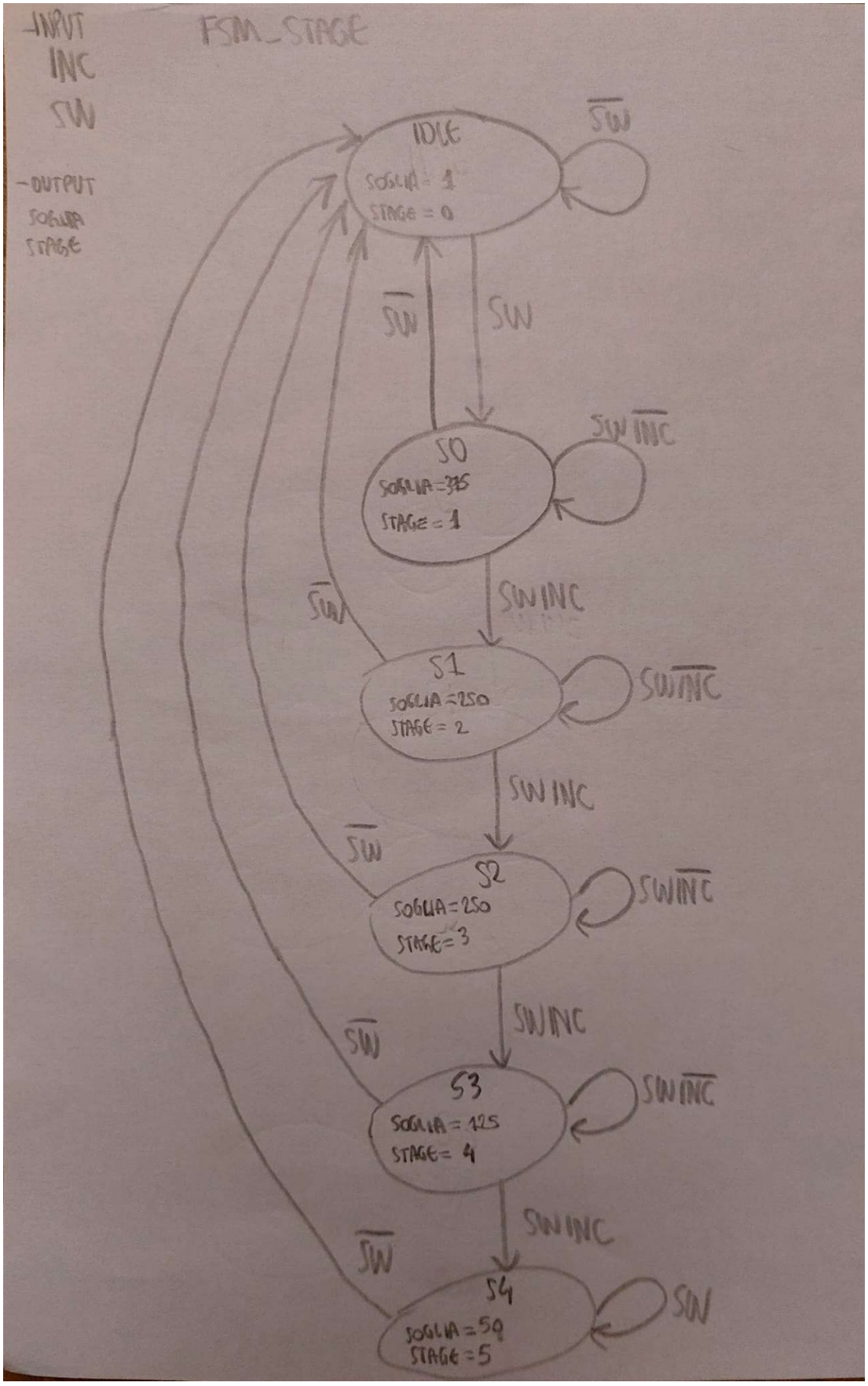
STATE	DESCRIPTION
IDLE	it does nothing
S1	State used to understand that the spi data is received from slave 1
S2	State used to understand that the spi data is received from slave 2

S1X	I use this state to sample the data correctly
S2X	I use this state to sample the data correctly

STATE	INPUT	NEXT STATE
IDLE	SS=0 DONE=X	IDLE
	SS=1 DONE=X	S1
	SS=2 DONE=X	S2
S1	SS=X DONE=1	S1X
	SS=X DONE=0	S1
S2	SS=X DONE=1	S2X
	SS=X DONE=0	S2
S1X	SS=X DONE=X	IDLE
	SS=X DONE=X	IDLE
S2X	SS=X DONE=X	IDLE
	SS=X DONE=X	IDLE

STATE	OUTPUT	
	incremento	livello
IDLE	0	0
S1	0	spi_data2
S2	spi_data2	0
S1X	0	spi_data2
S2X	spi_data2	0

FSM_STAGE


INPUT
INC
SW
OUTPUT
SOGLIA
STAGE
FSM_STAGE
IDLE
SOGLIA = 1
STAGE = 0
SW
SW
SW
S0
SOGLIA=375
STAGE = 1
SW INC
SW INC
S1
SOGLIA=250
STAGE = 2
SW INC
SW
SW INC
S2
SOGLIA=250
STAGE=3
SW INC
SW
SW INC
S3
SOGLIA = 125
STAGE = 4
SW INC
SW
S4
SOGLIA=50
STAGE=5
SW

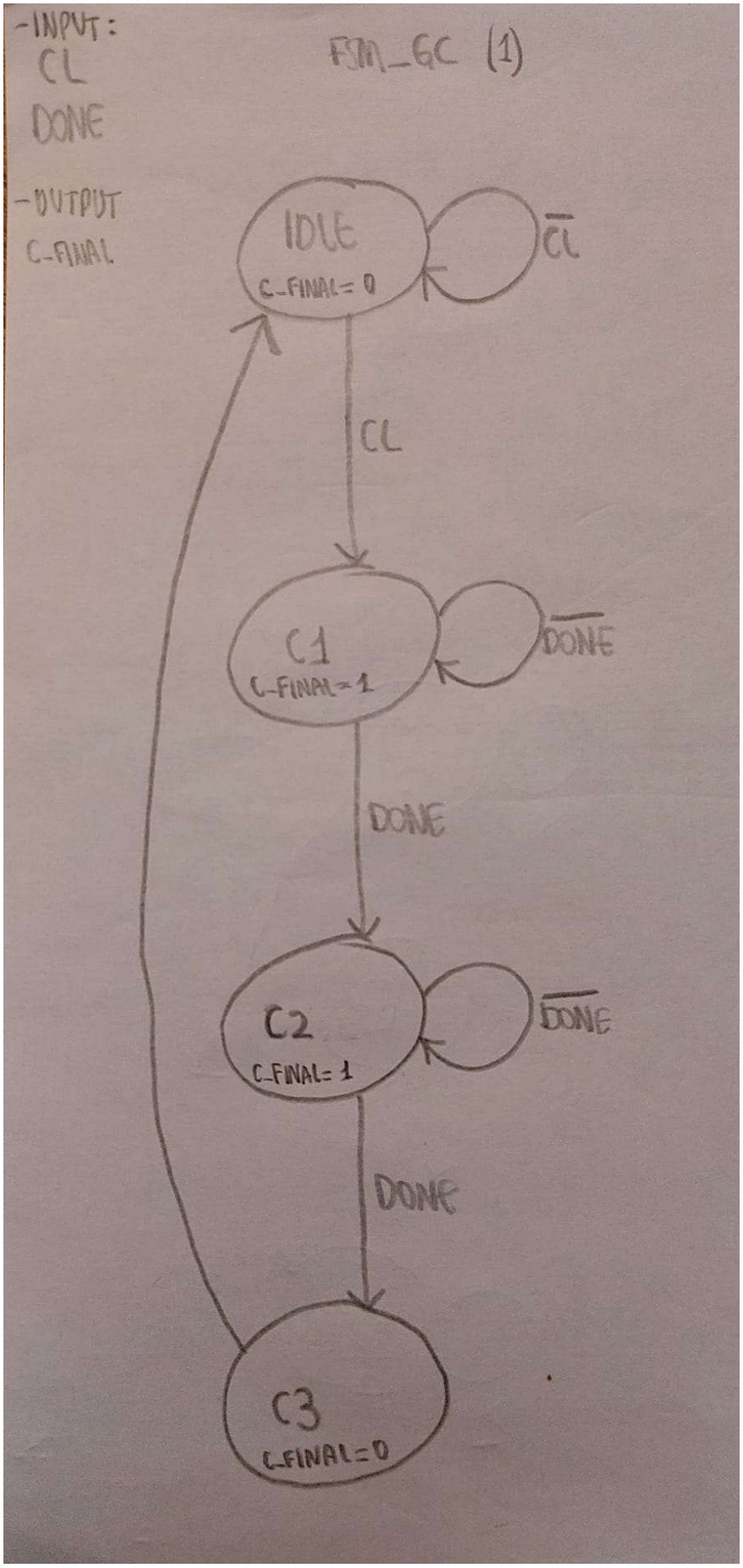
STATE	DESCRIPTION
IDLE	does nothing, wait for the high sw
S0	been used to make the status transition every 24h and to set the new alarm thresholds
S1	been used to make the status transition every 24h and to set the new alarm thresholds
S2	been used to make the status transition every 24h and to set the new alarm thresholds
S3	been used to make the status transition every 24h and to set the new alarm thresholds
S4	been used to make the status transition every 24h and to set the new alarm thresholds

- In IDLE threshold is set to 1 so as not to send the system into error the first instant the sw is raised, even before it goes into state S0

STATE	INPUT	STATE NEXT
IDLE	Sw=1 Incremento=x	S0
	Sw=0 Incremento=x	IDLE
S0	Sw=0 Incremento=x	IDLE
	Sw=1 Incremento=1	S1
	Sw=1 Incremento=0	S0
S1	Sw=0 Incremento=x	IDLE
	Sw=1 Incremento=1	S2
	Sw=1 Incremento=0	S1
S2	Sw=0 Incremento=x	IDLE
	Sw=1 Incremento=1	S3
	Sw=1 Incremento=0	S2
S3	Sw=0 Incremento=x	IDLE
	Sw=1 Incremento=1	S4
	Sw=1 Incremento=0	S3
S4	Sw=0 Incremento=x	IDLE
	Sw=1 Incremento=x	S4

STATE	OUTPUT	
	soglia	stage
IDLE	1	0
S0	375	1
S1	250	2
S2	250	3
S3	125	4
S4	50	5

FSM_GC1


-INPUT:
CL
DONE
-OUTPUT
C_FINAL
FSM_GC (1)
IDLE
C_FINAL = 0
$\overline{CL}$
CL
C1
C_FINAL = 1
$\overline{DONE}$
DONE
C2
C_FINAL = 1
$\overline{DONE}$
DONE
C3
C_FINAL = 0

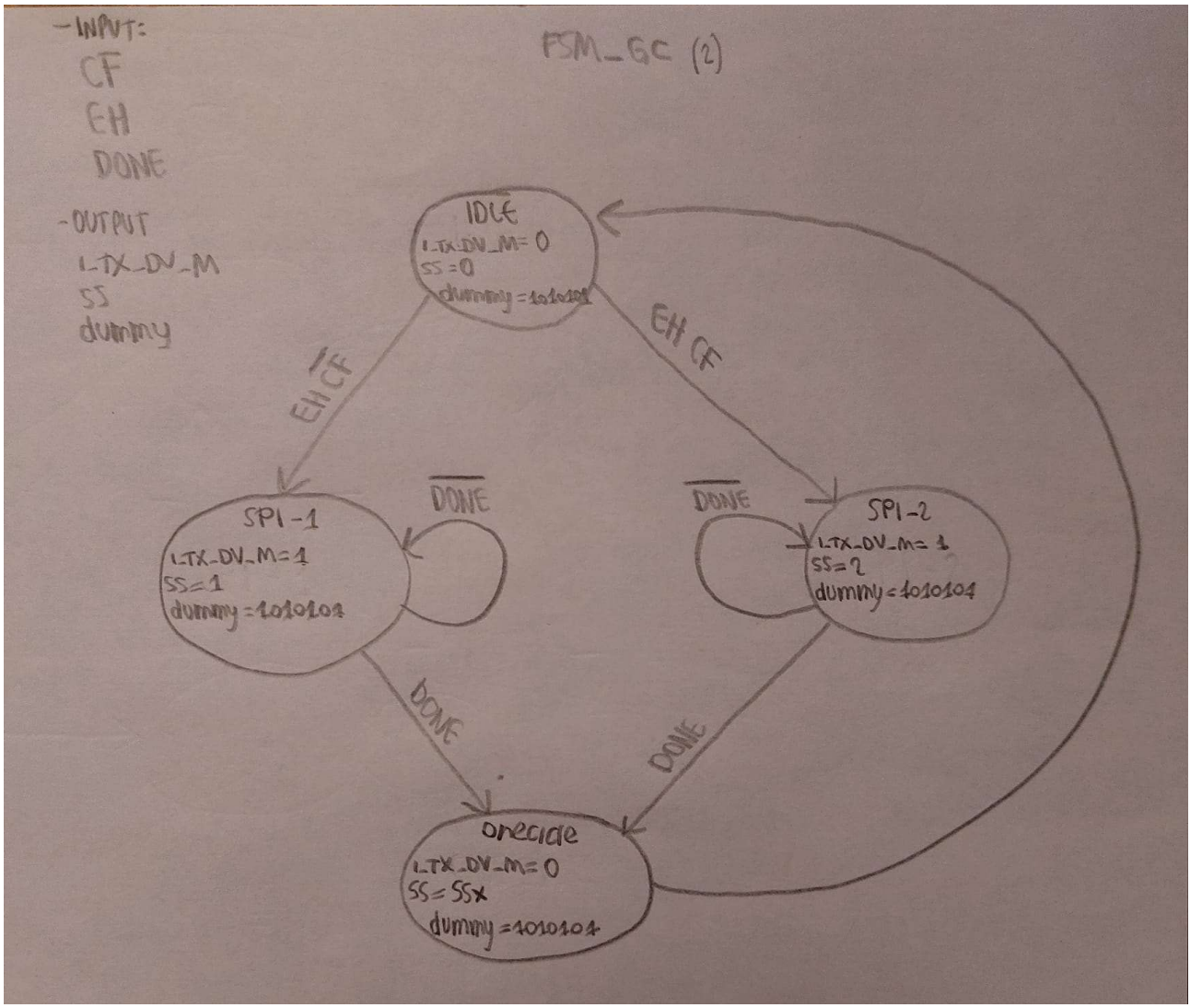
STATE	INPUT	STATE NEXT
IDLE	Clear=1 Done=X	C1
	Clear=0 Done=X	IDLE
C1	Clear=X Done=0	C1
	Clear=X Done=1	C2
C2	Clear=X Done=0	C2
	Clear=X Done=1	C3
C3	Clear=X Done=X	IDLE

- we use this state machine to communicate with the spi, in particular for handling the communication related to the 24h, in fact the clear always came up at times where it was communicating with the lev button and therefore we could not receive a bit that was related to the stage progress, so this state machine solves our problem by waiting for the previous communications to end and then taking precedence with the transmission

STATE	DESCRIPTION
IDLE	waits for the high clear received from conta24 to notify the spi
C1	Status to communicate with the spi at the right time to receive the stage change
C2	state to report the end of communication with the spi
C3	state to report the end of communication with the spi

STATE	OUTPUT
	C_final
IDLE	0
C1	1
C2	1
C3	0

FSM_GC2



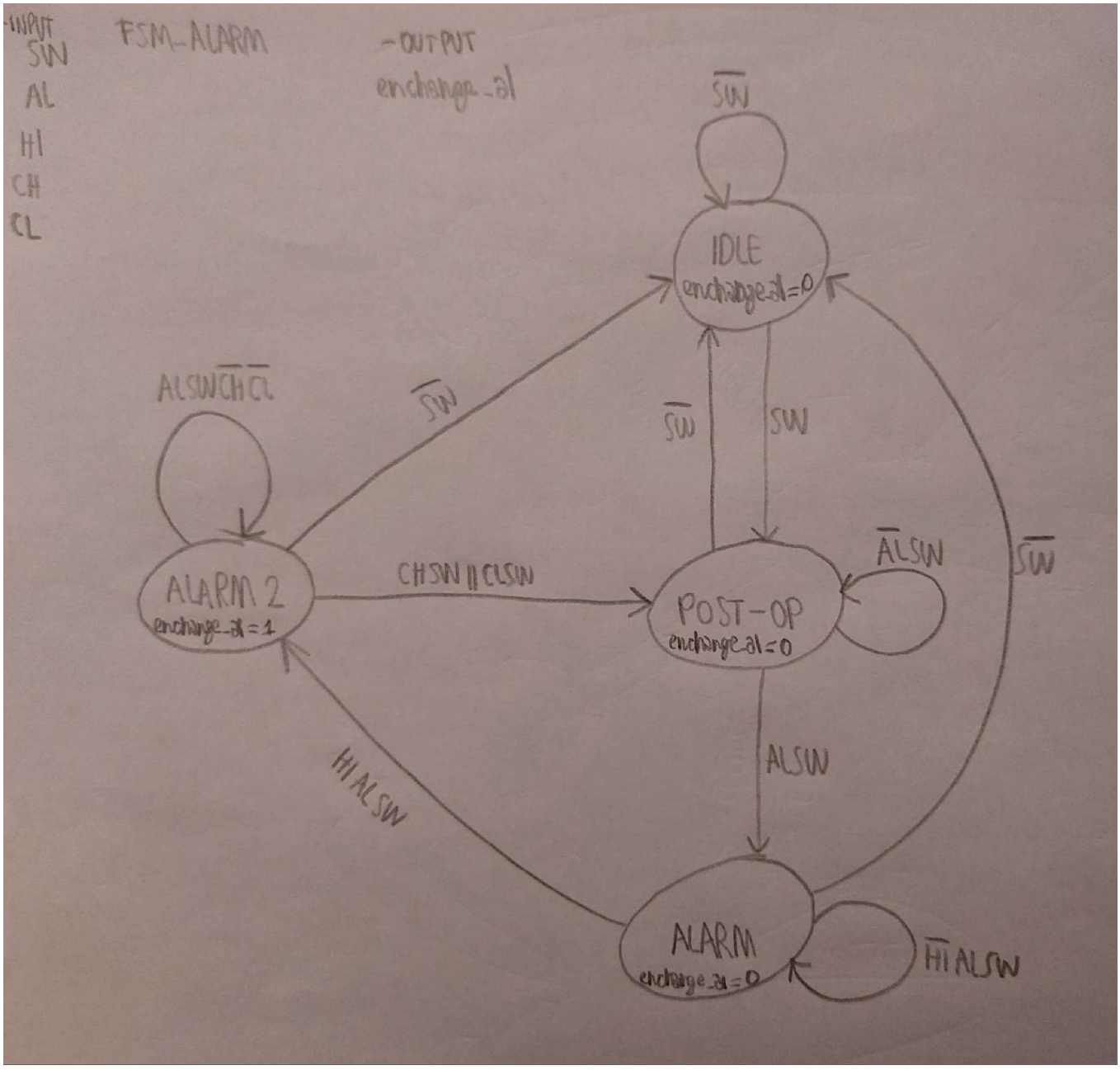
- this fsm uses the output of the previous one to synchronize communications with slaves

STATE	INPUT	STATE NEXT
IDLE	Enh=1 C_final=1 Done=X	SPI2
	Enh=1 C_final=0 Done=X	SPI1
	Enh=0 C_final=X Done=X	IDLE
SPI1	Enh=X C_final=X Done=0	SPI1
	Enh=X C_final=X Done=1	OneCicle
SPI2	Enh=X C_final=X Done=0	SPI2
	Enh=X C_final=X Done=1	OneCicle
OneCicle	Enh=X C_final=X Done=X	IDLE

STATE	DESCRIPTION
IDLE	remains waiting for the right inputs to communicate with the right slave
SPI1	waits for the done low to start communicating with the correct slave
SPI2	waits for the done low to start communicating with the correct slave
OneCicle	been additionally used to correctly sample the data received from the spi

STATE	OUTPUT		
	I_TX_DV_M	SS	dummy
IDLE	0	0	1010101
SPI1	1	1	1010101
SPI2	1	2	1010101
OneCicle	0	SSx	1010101

FSM_ALARM


INPUT
SW
AL
HI
CH
CL
FSM-ALARM
-OUTPUT
enchange_al
IDLE
enchange_al=0
ALARM 2
enchange_al=1
POST-OP
enchange_al=0
ALARM
enchange_al=0
ALSWCHCL
CHSW || CLSW
ALSW
HIALSW
HIALSW

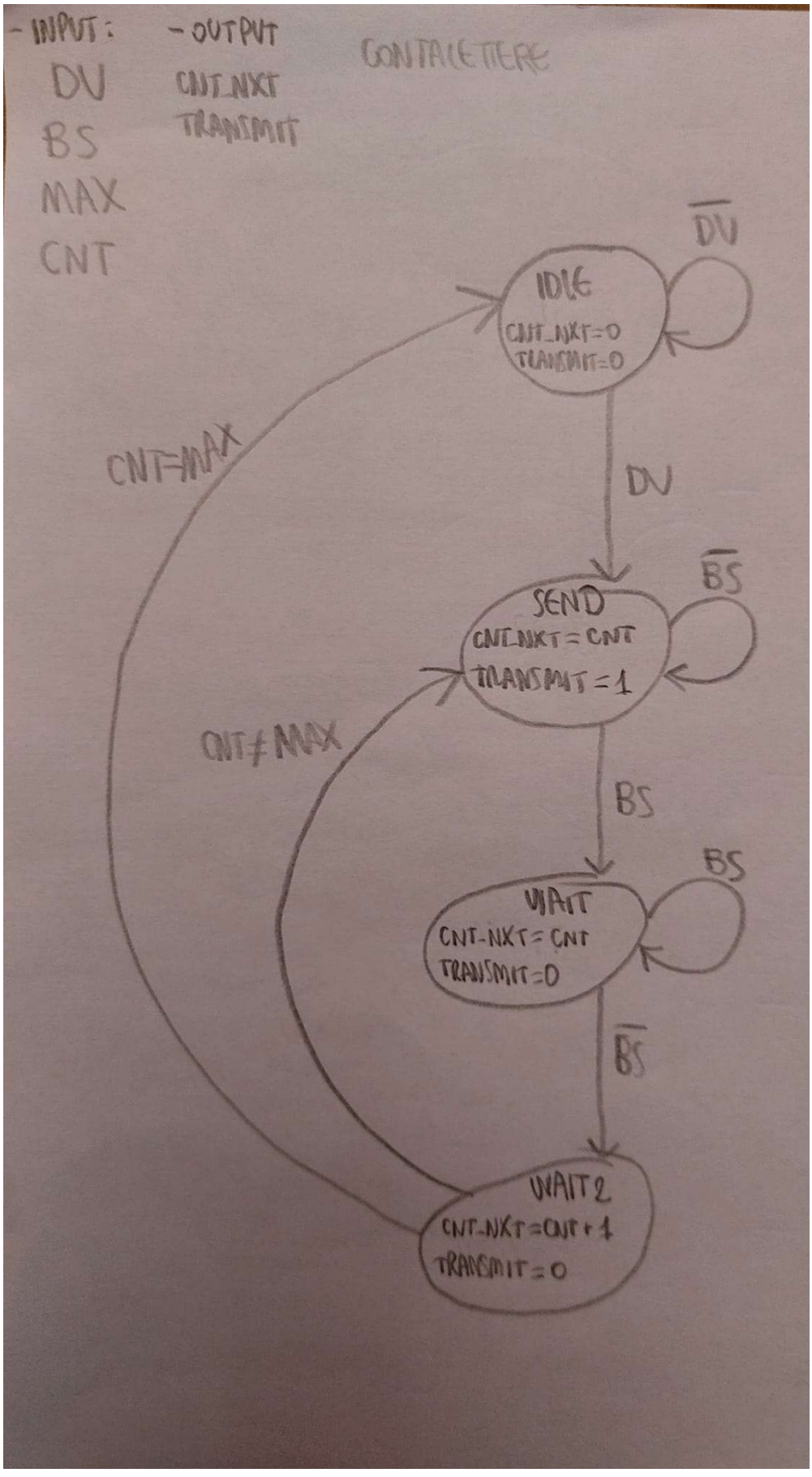
STATE	INPUT	STATE NEXT
IDLE	SW=1 Alarm=X History=X Change=X Clear0=X	POST_OP
	SW=0 Alarm=X History=X Change=X Clear0=X	IDLE
POST_OP	SW=0 Alarm=X History=X Change=X Clear0=X	IDLE
	SW=1 Alarm=1 History=X Change=X Clear0=X	ALARM
	SW=1 Alarm=1 History=X Change=X Clear0=X	POST_OP
ALARM	SW=0 Alarm=X History=X Change=X Clear0=X	IDLE
	SW=1 Alarm=1 History=1 Change=X Clear0=X	ALARM2
	SW=1 Alarm=1 History=0 Change=X Clear0=X	ALARM
ALARM2	SW=0 Alarm=X History=X	IDLE

	Change=X Clear0=X	
	SW=1 Alarm=X History=X Change=1 Clear0=X OR SW=1 Alarm=X History=X Change=X Clear0=1	POST_OP
	SW=1 Alarm=X History=X Change=0 Clear0=0	

STATE	DESCRIPTION
IDLE	nothing happens
POST-OP	nothing happens
ALARM	I remain waiting to receive a history so I can have access to the other two buttons that did not work during the error
ALARM2	I remain waiting for a click of change to empty the bag and lift the alarm situation, I can also click the status button

STATE	OUTPUT
	Enchange_al
IDLE	0
POST_OP	0
ALARM	0
ALARM2	1

Contalettere

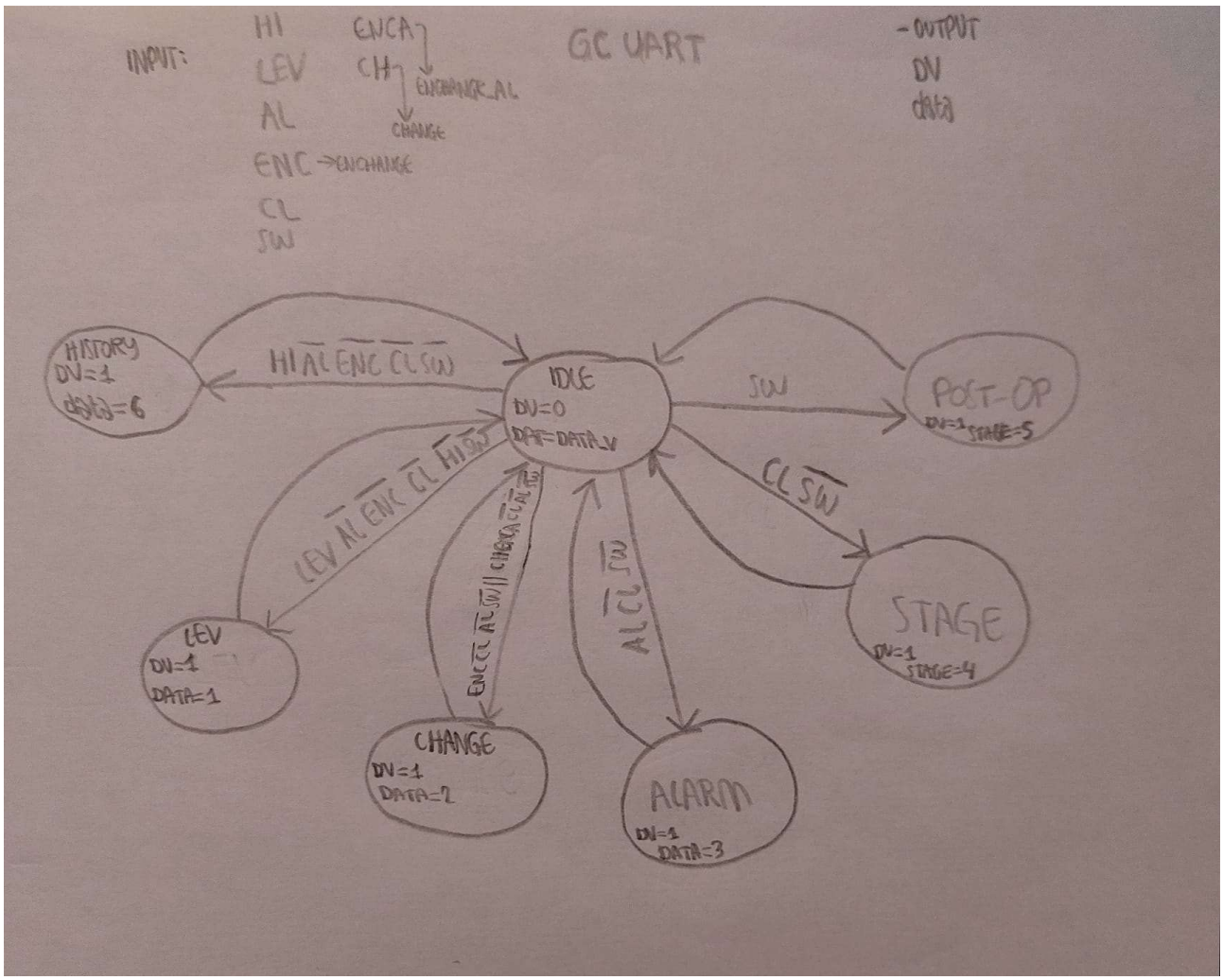


STATE	INPUT	STATE NEXT
IDLE	DV=1 Busy=X CNT=X	SEND
	DV=0 Busy=X CNT=X	IDLE
SEND	DV=X Busy=1 CNT=X	WAIT
	DV=X Busy=0 CNT=X	SEND
WAIT	DV=X Busy=0 CNT=X	WAIT2
	DV=X Busy=1 CNT=X	WAIT
WAIT2	DV=X Busy=X CNT=MAX	IDLE
	DV=X Busy=X CNT!=MAX	SEND

STATE	DESCRIPTION
IDLE	does nothing
SEND	Waits for the letter to be sent waiting for the busy high
WAIT	when busy goes down it means the signal has been sent, so we will move to the new state where we will figure out whether to continue sending or go back to idle
WAIT2	I send the counter forward by one and see if I should keep transmitting

STATE	OUTPUT	
	Cnt_nxt	transmit
IDLE	0	0
SEND	Cnt	1
WAIT	Cnt	0
WAIT2	Cnt + 1	0

GCUART


INPUT:
HI
LEV
AL
ENC
CL
SW
ENCA
CH
ENCHANGE_AL
CHANGE
ENCHANGE
GC UART
- OUTPUT
DV
data
HISTORY
DV=1
data=6
HI AL ENC CL SW
IDLE
DV=0
DATA=DATA_V
SW
POST-OP
DV=1 STAGE=5
LEV AL ENC CL HI SW
CL SW
ENC CL AL SW || CHENCA CL AL SW
AL CL SW
LEV
DV=1
DATA=1
CHANGE
DV=1
DATA=2
ALARM
DV=1
DATA=3
STAGE
DV=1
STAGE=4

STATE	INPUT	STATE NEXT
IDLE	Enlev=1 Alarm=0 Enchange=0 Clear0=0 History=0 Sw=0 Change=X Enchange_Al=X	LEV
	Enlev=X History=X Alarm=0 Enchange=1 Clear0=0 Sw=0 Change=X Enchange_Al=X OR Enlev=X History=X Alarm=0 Enchange=X Clear0=0 Sw=0 Change=1 Enchange_Al=1	CHANGE
	Enlev=X History=X Alarm=1 Enchange=X Clear0=0 Sw=0 Change=X Enchange_Al=X	ALARM
	Enlev=X History=X Alarm=X Enchange=X Clear0=1 Sw=0 Change=X Enchange_Al=X	STAGE
	Enlev=X History=1	HISTORY

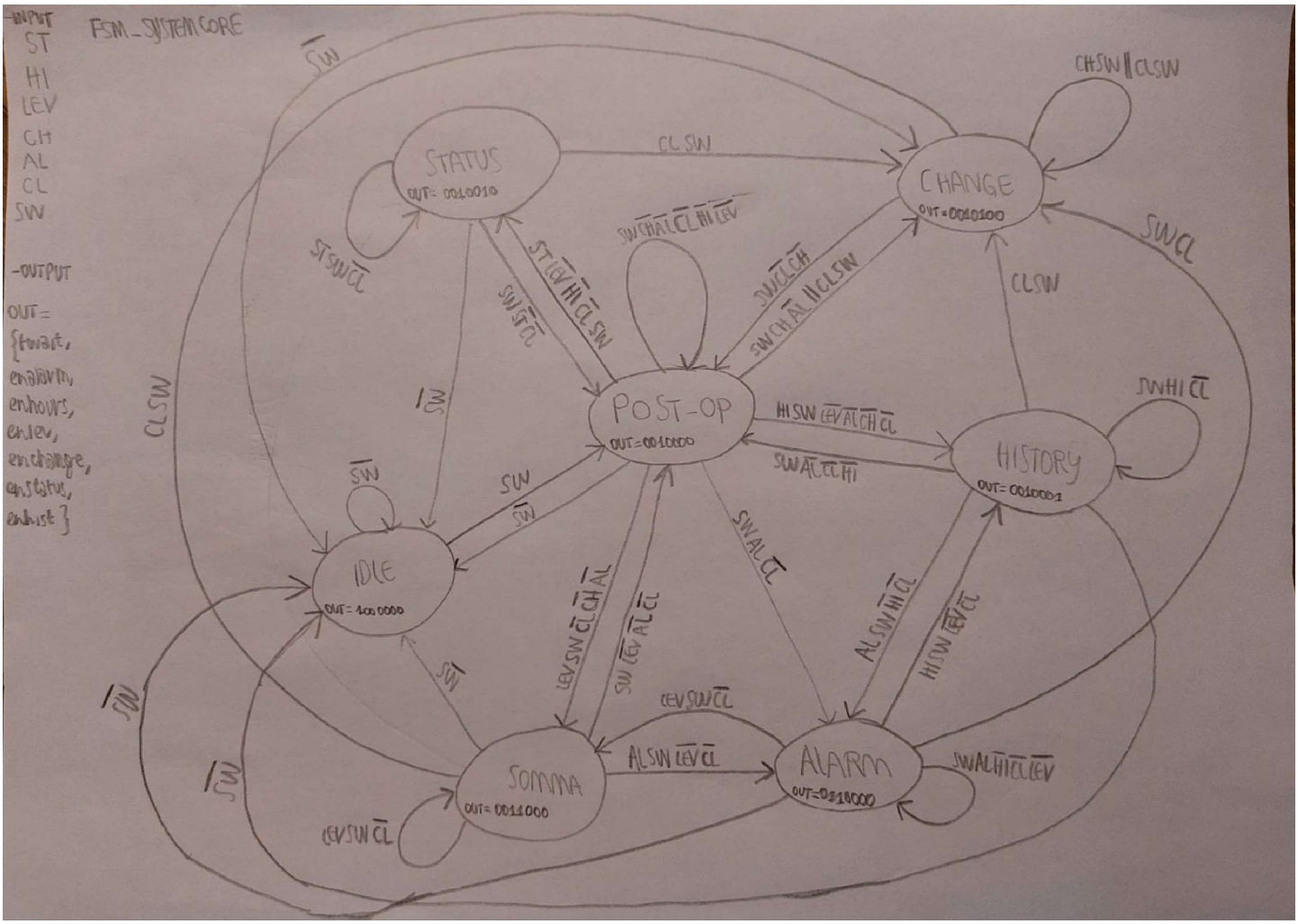
	Alarm=0 Enchange=0 Clear0=0 Sw=0 Change=X Enchange_Al=X	
	Enlev=X History=X Alarm=X Enchange=X Clear0=X Sw=1 Change=X Enchange_Al=X	POST-OP
HISTORY	Enlev=X History=X Alarm=X Enchange=X Clear0=X Sw=X Change=X Enchange_Al=X	IDLE
LEV	Enlev=X History=X Alarm=X Enchange=X Clear0=X Sw=X Change=X Enchange_Al=X	IDLE
CHANGE	Enlev=X History=X Alarm=X Enchange=X Clear0=X Sw=X Change=X Enchange_Al=X	IDLE
ALARM	Enlev=X History=X Alarm=X Enchange=X Clear0=X Sw=X	IDLE

	Change=X Enchange_Al=X	
STAGE	Enlev=X History=X Alarm=X Enchange=X Clear0=X Sw=X Change=X Enchange_Al=X	IDLE
POST-OP	Enlev=X History=X Alarm=X Enchange=X Clear0=X Sw=X Change=X Enchange_Al=X	IDLE

STATE	DESCRIPTION
IDLE	waits for high signals
LEV	performs the printing of the action in the screen (lev button)
CHANGE	performs the printing of the action in the screen (change bag)
ALARM	performs the printing of the action in the screen (alarm on)
STAGE	performs the printing of the action in the screen (next stage)
POST-OP	performs the printing of the action in the screen (post op started)
HISTORY	performs the printing of the action in the screen (history button)

STATE	OUTPUT	
	DV	data
LEV	1	1
CHANGE	1	2
ALARM	1	3
STAGE	1	4
POST-OP	1	5
HISTORY	1	6

FSM SYSTEM CORE



STATE	INPUT	STATE NEXT
IDLE	Status=X History=X Lev=X Change=X Alarmon=X Sw=1 Clear=X	POST-OP
	Status=X History=X Lev=X Change=X Alarmon=X Sw=0 Clear=X	IDLE

POST-OP	Status=X History=X Lev=X Change=X Alarmon=X Sw=0 Clear=X	IDLE
	Status=X History=X Lev=X Change=1 Alarmon=0 Sw=1 Clear=X OR Status=X History=X Lev=X Change=X Alarmon=X Sw=1 Clear=1	CHANGE
	Status=X History=X Lev=X Change=X Alarmon=1 Sw=1 Clear=0	ALARM
	Status=X History=X Lev=1 Change=0 Alarmon=0 Sw=1 Clear=0	SOMMA
	Status=1 History=0 Lev=0 Change=0 Alarmon=0 Sw=1	STATUS

	Clear=0	
	Status=X History=1 Lev=0 Change=0 Alarmon=0 Sw=1 Clear=0	HISTORY
	Status=0 History=0 Lev=0 Change=0 Alarmon=0 Sw=1 Clear=0	POST-OP
SOMMA	Status=X History=X Lev=X Change=X Alarmon=X Sw=0 Clear=X	IDLE
	Status=X History=X Lev=X Change=X Alarmon=X Sw=1 Clear=1	CHANGE
	Status=X History=X Lev=0 Change=X Alarmon=1 Sw=1 Clear=0	ALARM
	Status=X History=X Lev=1 Change=X Alarmon=X Sw=1 Clear=0	SOMMA
	Status=X	POST-OP

	History=X Lev=0 Change=X Alarmon=X Sw=1 Clear=0	
STATUS	Status=X History=X Lev=X Change=X Alarmon=X Sw=0 Clear=X	IDLE
	Status=X History=X Lev=X Change=X Alarmon=X Sw=1 Clear=1	CHANGE
	Status=1 History=X Lev=X Change=X Alarmon=X Sw=1 Clear=0	STATUS
	Status=0 History=X Lev=X Change=X Alarmon=X Sw=1 Clear=0	POST-OP
HISTORY	Status=X History=X Lev=X Change=X Alarmon=X Sw=0 Clear=X	IDLE
	Status=X History=X Lev=X	CHANGE

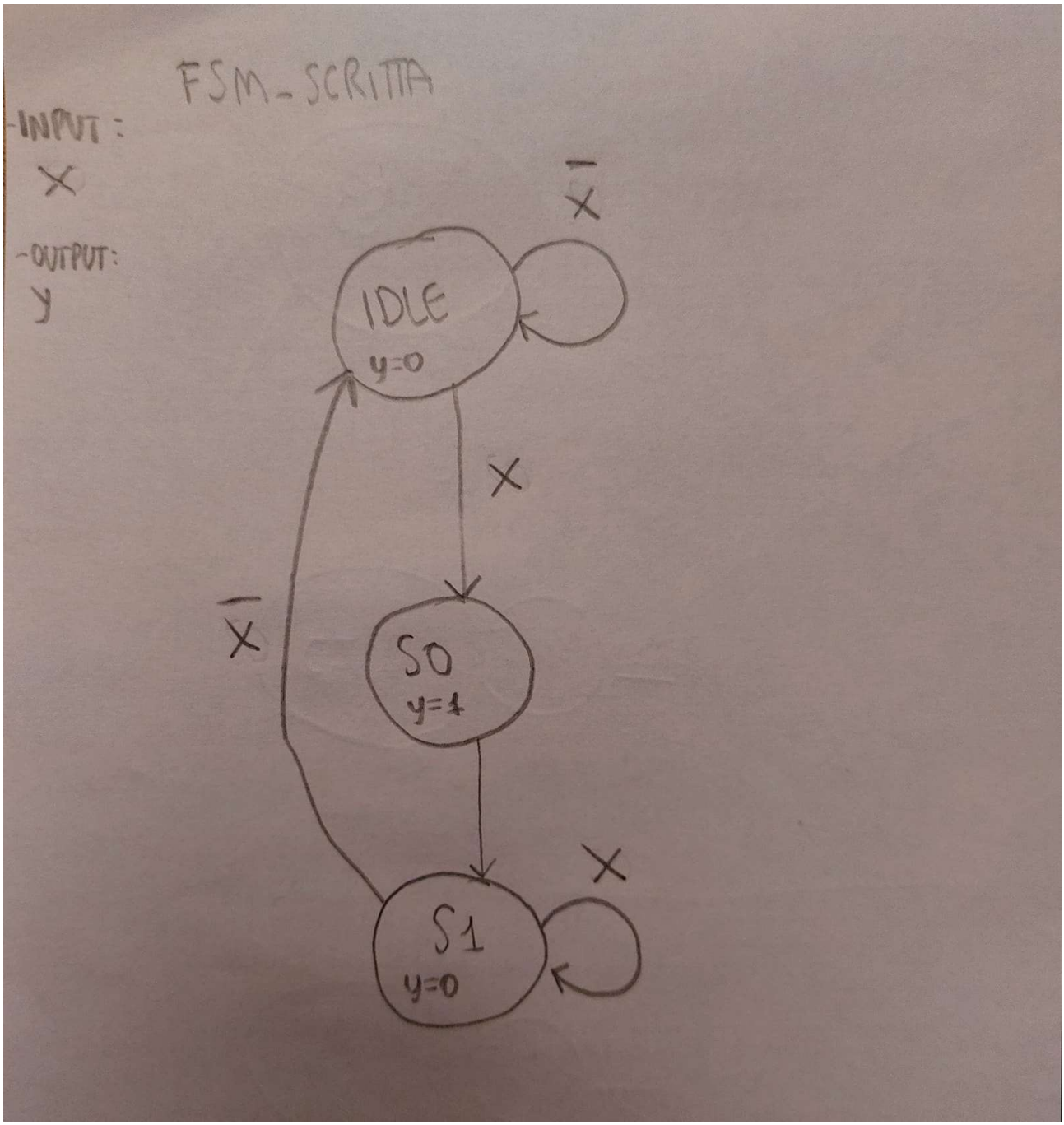
	Change=X Alarmon=X Sw=1 Clear=1	
	Status=X History=0 Lev=X Change=X Alarmon=1 Sw=1 Clear=0	ALARM
	Status=X History=1 Lev=X Change=X Alarmon=X Sw=1 Clear=0	HISTORY
	Status=X History=0 Lev=X Change=X Alarmon=X Sw=1 Clear=0	POST-OP
ALARM	Status=X History=X Lev=1 Change=X Alarmon=X Sw=1 Clear=0	SOMMA
	Status=X History=X Lev=X Change=X Alarmon=X Sw=0 Clear=X	IDLE
	Status=X History=X Lev=X Change=X Alarmon=X	CHANGE

	Sw=1 Clear=1	
	Status=X History=1 Lev=0 Change=X Alarmon=X Sw=1 Clear=0	HISTORY
	Status=X History=X Lev=1 Change=X Alarmon=X Sw=1 Clear=0	SOMMA
	Status=X History=0 Lev=0 Change=X Alarmon=1 Sw=1 Clear=0	ALARM

STATE	DESCRIPTIONS
IDLE	Writes wait on the 7-segment display
POST-OP	Writes post in the 7-segment display and stands by for actions
SOMMA	increases the value contained in the bag
STATUS	Shows the current bag level on the 7-segment display
HISTORY	Shows in the screen the last lev increment, with level, stage and time
ALARM	situation that does not make us go into change (through the button) and status
CHANGE	Bag change due to button click or 24h

STATE	OUTPUT
	Out={fwait,enalarm,enhours,enlev,enchange,enstatus,enhist}
IDLE	1000000
POST-OP	0010000
SOMMA	0011000
STATUS	0010010
HISTORY	0010001
ALARM	0110000
CHANGE	0010100

FSM_SCRITTA



STATE	INPUT	STATE NEXT
IDLE	x=0	IDLE
	x=1	S0
S0	x=X	S1
S1	x=1	S1
	x=0	IDLE

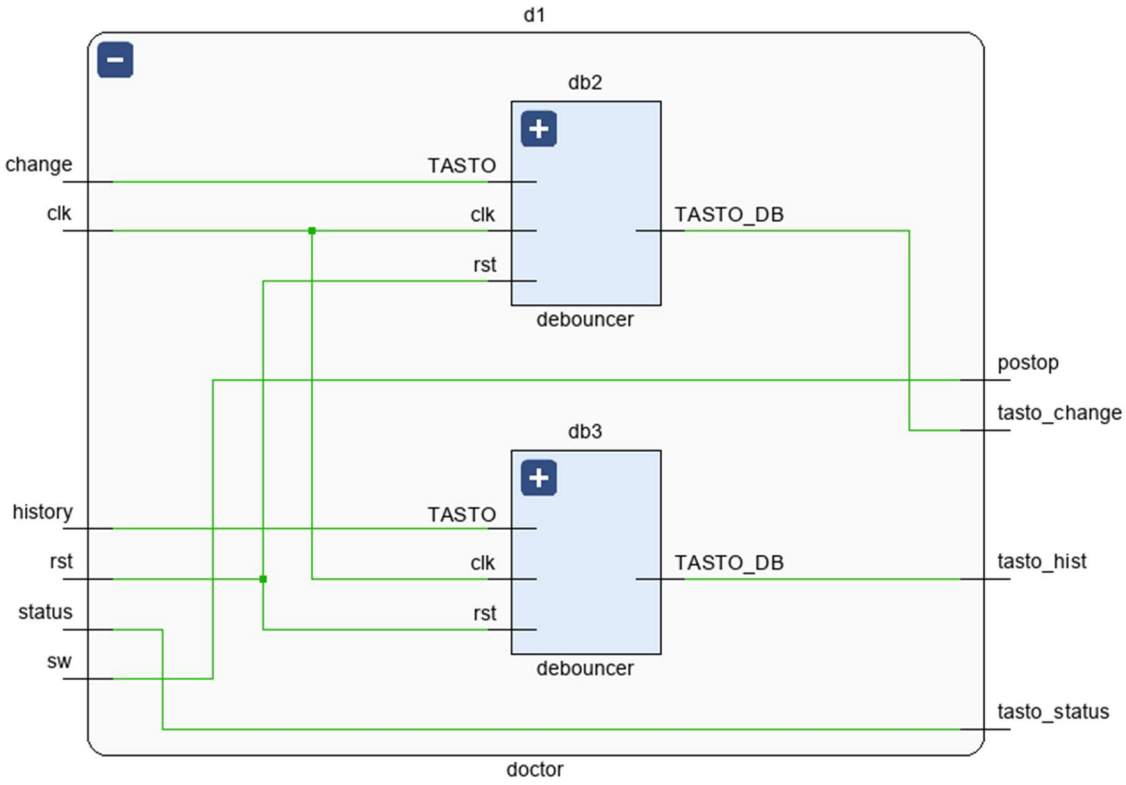
STATE	DESCRIPTION
IDLE	does nothing, wait for the high signal
S0	keeps the output high one clock
S1	Wait for the signal to drop to go into the idle state and wait for it to rise again

STATE	OUTPUT
	y
IDLE	0
S0	1
S1	0

6) Testbench and Schematic

Doctor

Schematic

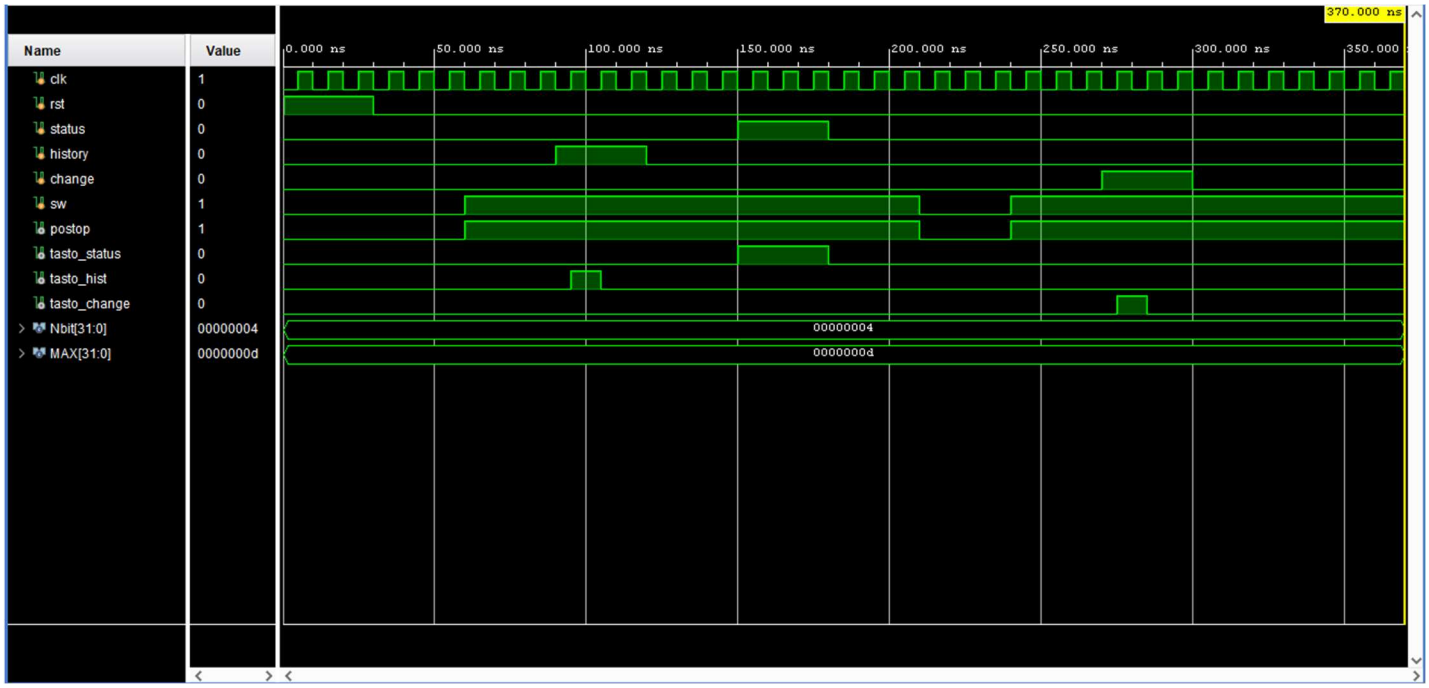


Testbench of the module Doctor

The testbench shows the correct operation of the two debouncers present, the status button, and the sw switch.

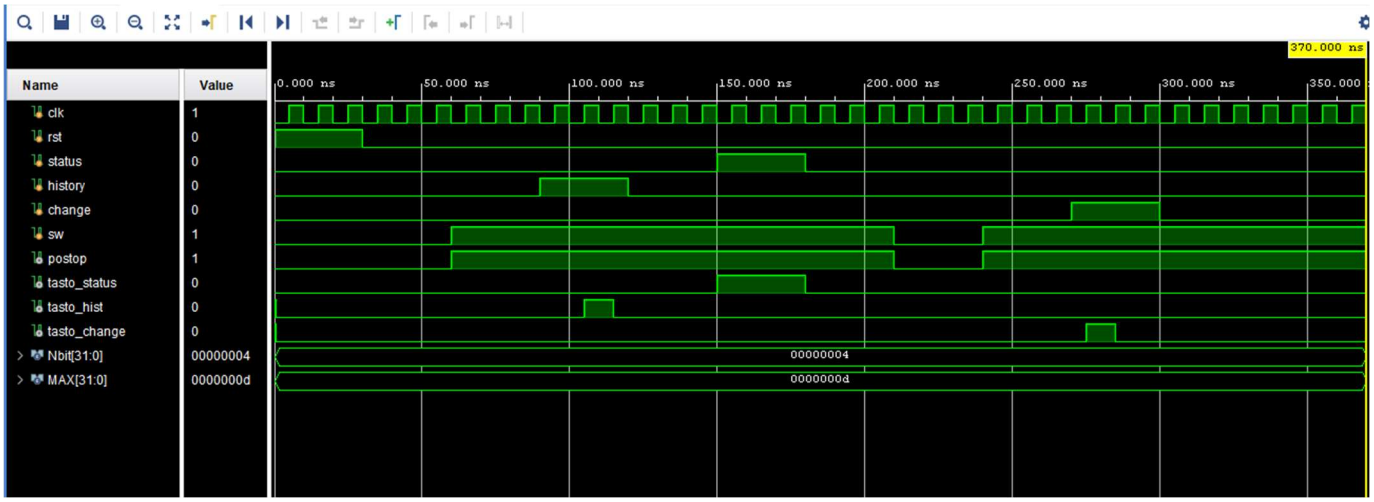
Compared to the operation on the board, we used ideal parameters that could make sense in the simulation, in fact Nbit will have value 4 and nMAX value 13.

Behavioral:

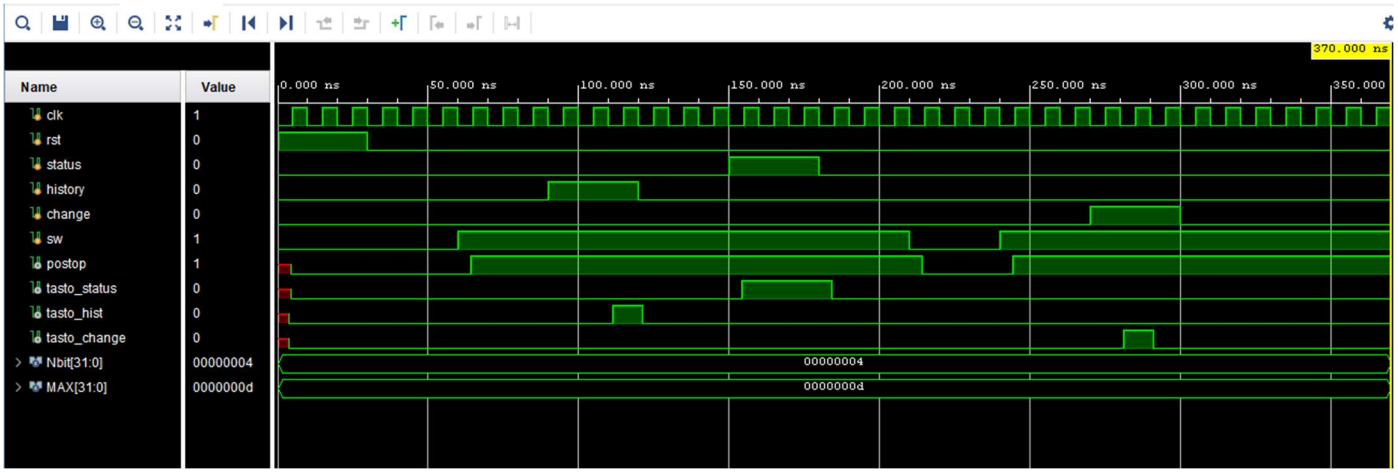


Post Synthesis Functional and Timing:

Functional



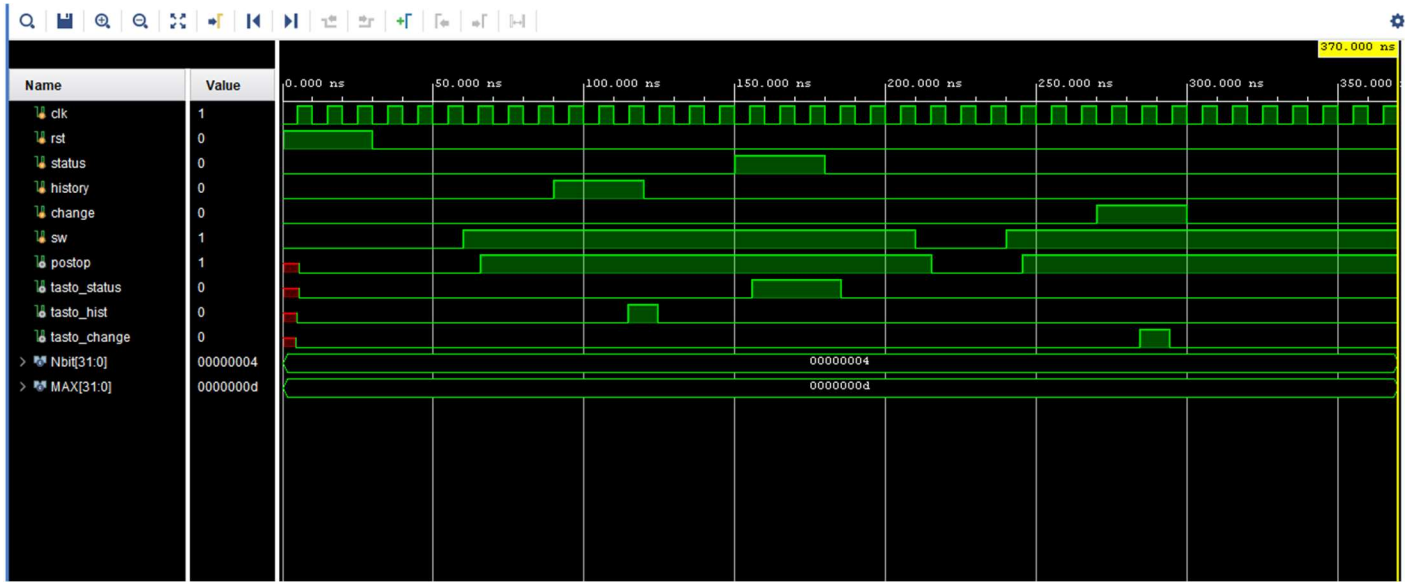
Timing:



Post Implementation Functional and Timing:

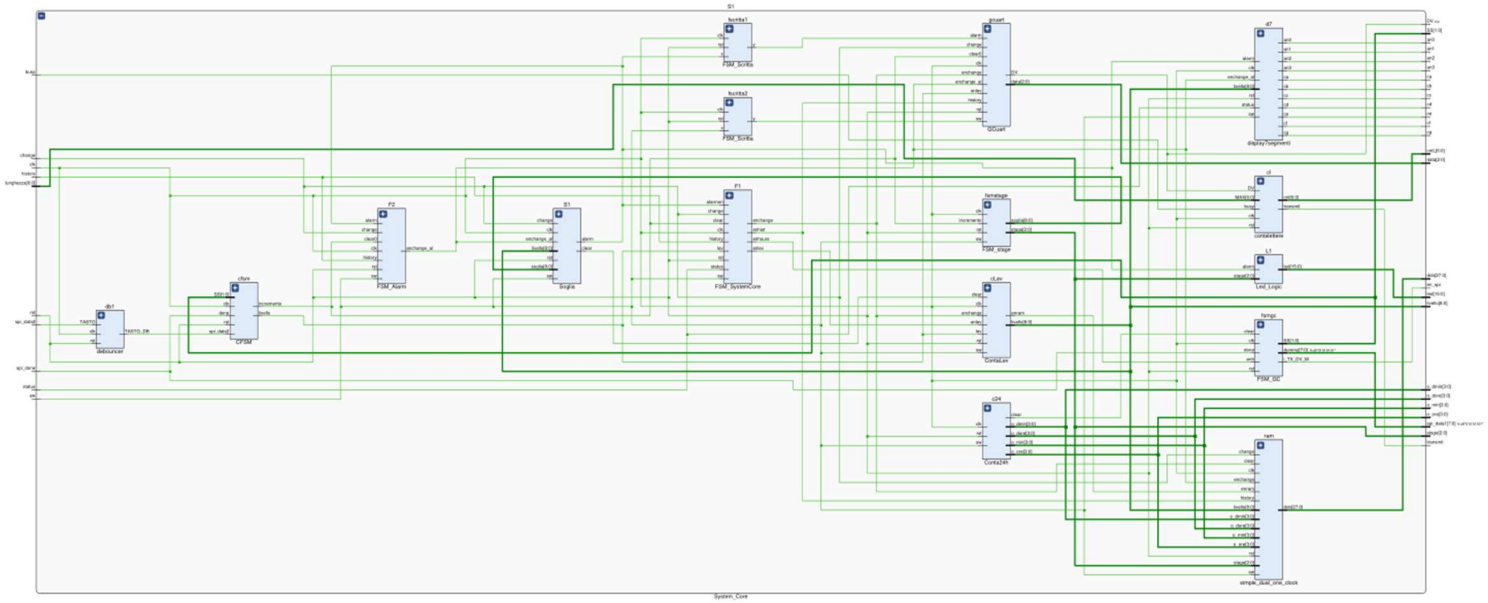
Functional:

Timing:



System Core

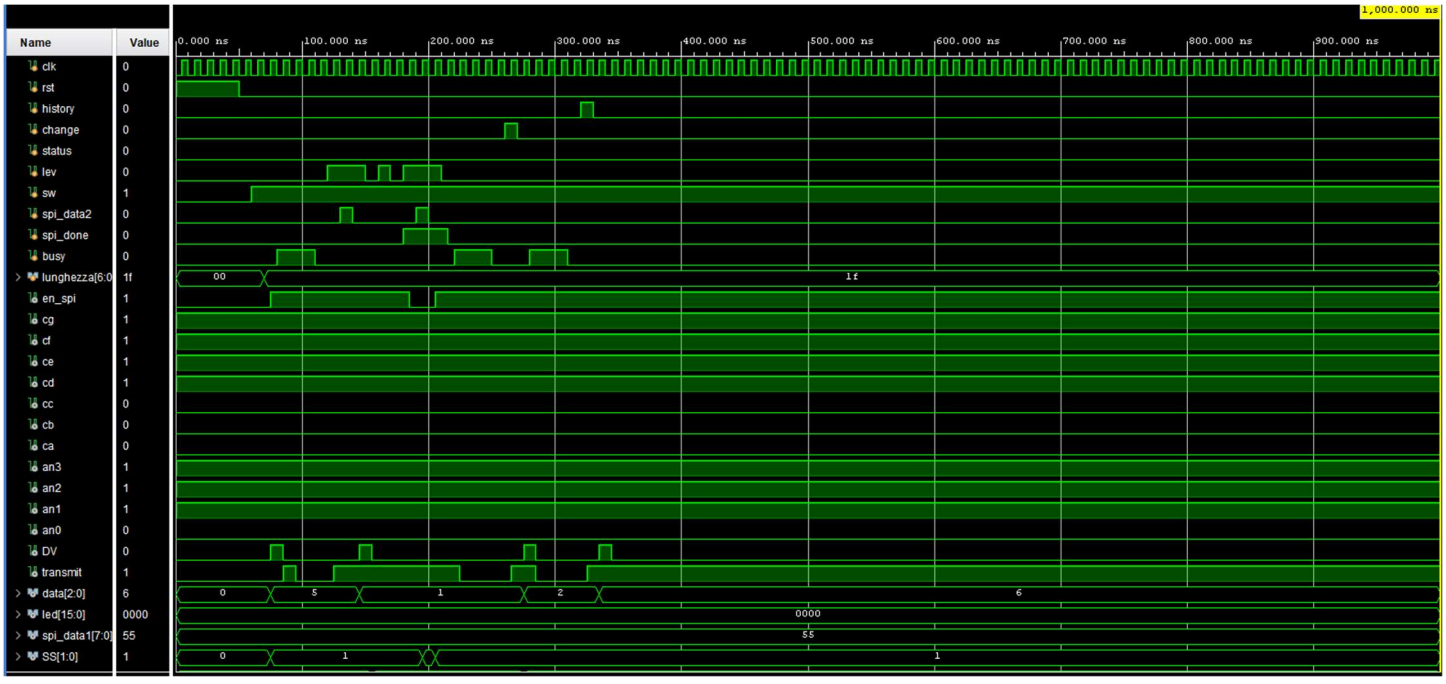
Schematic

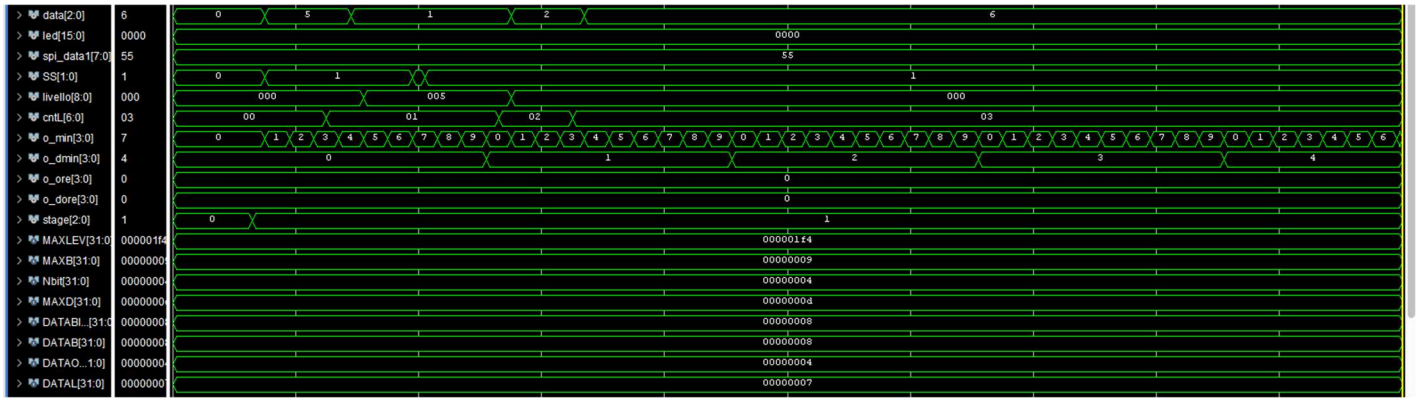


Testbench:

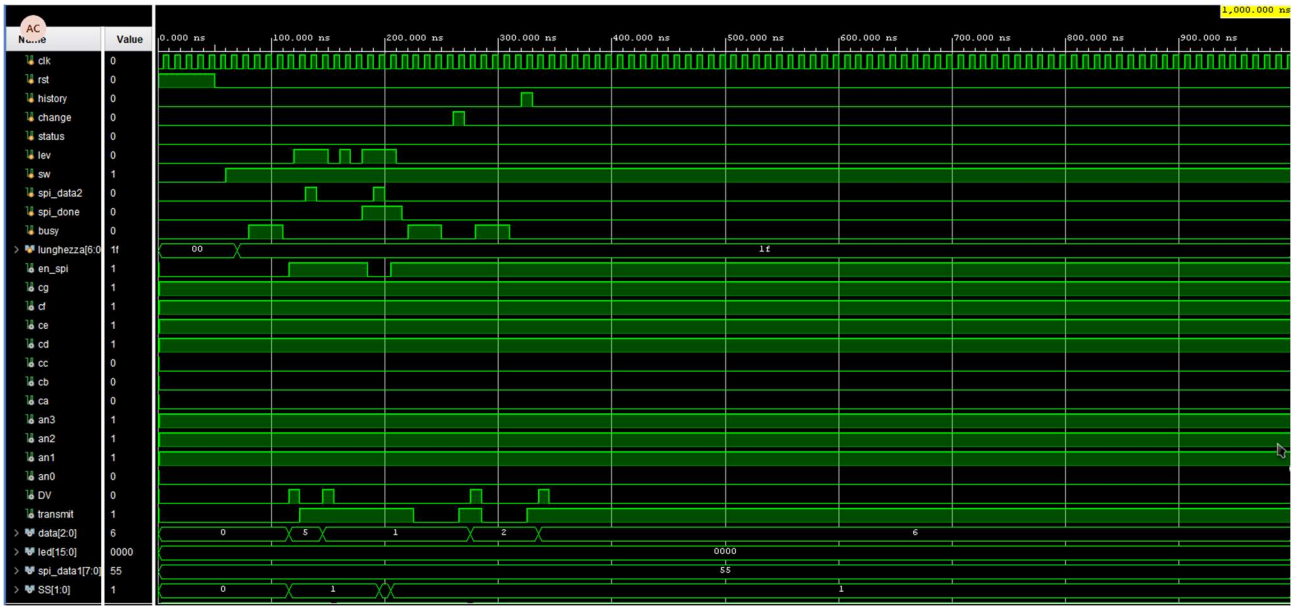
Testbench that simulates the behavior of the system core, trying to show how it works

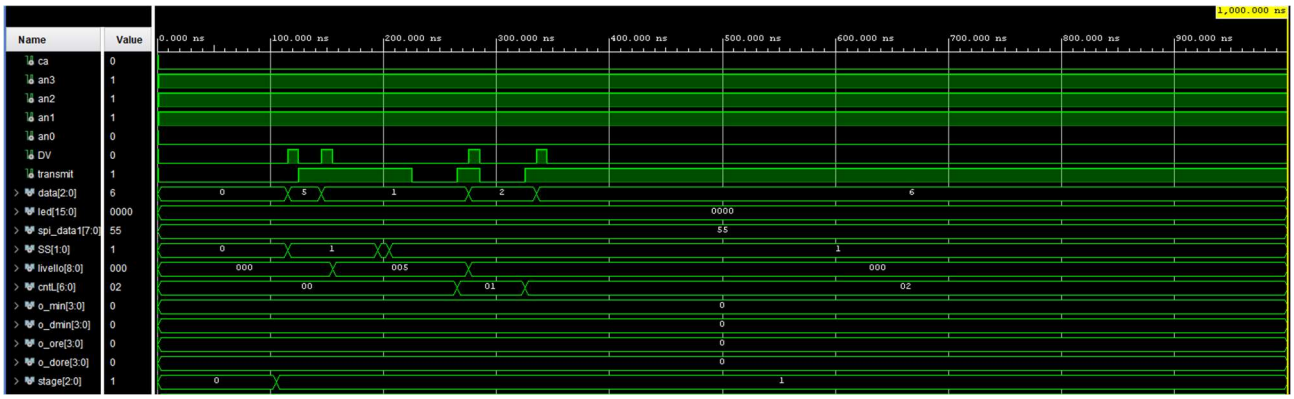
Behavioral:

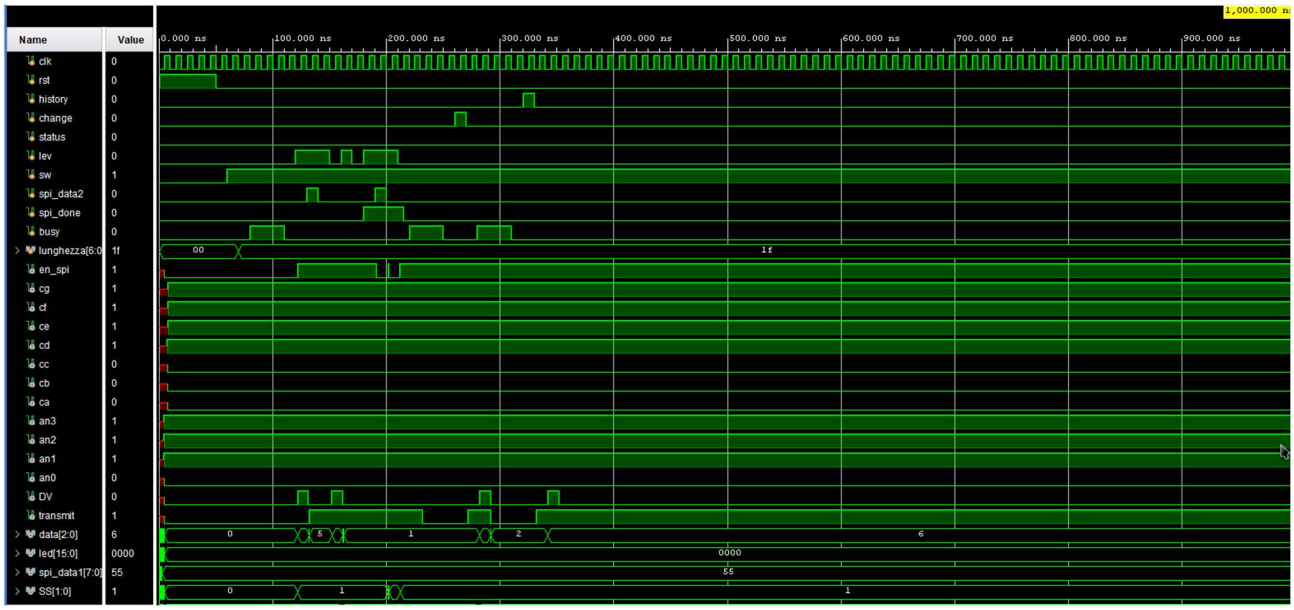


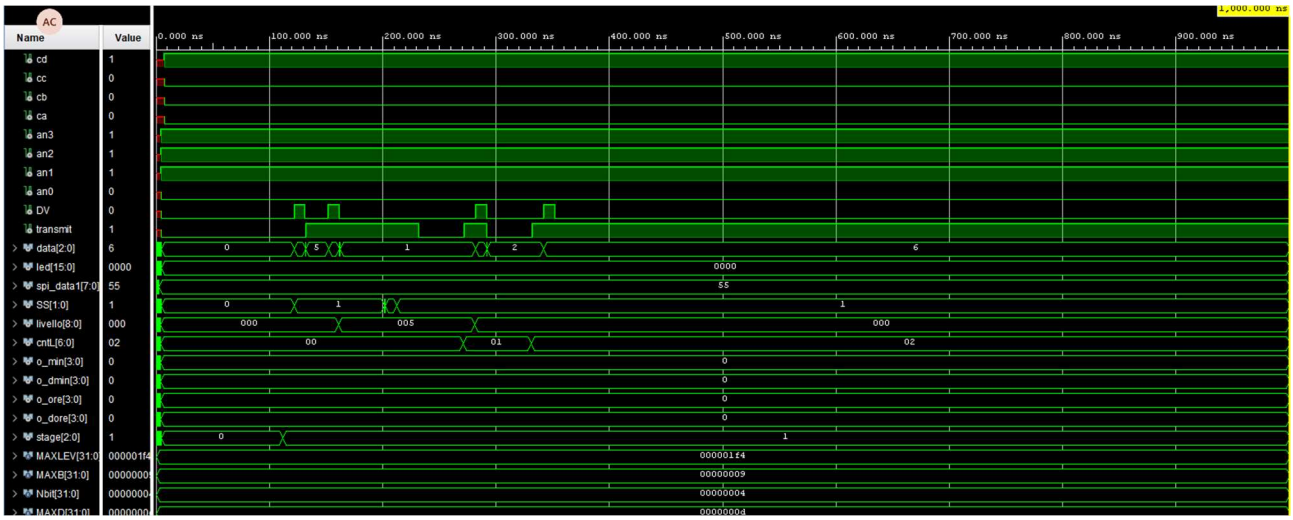


Post Synthesis Functional and Timing:





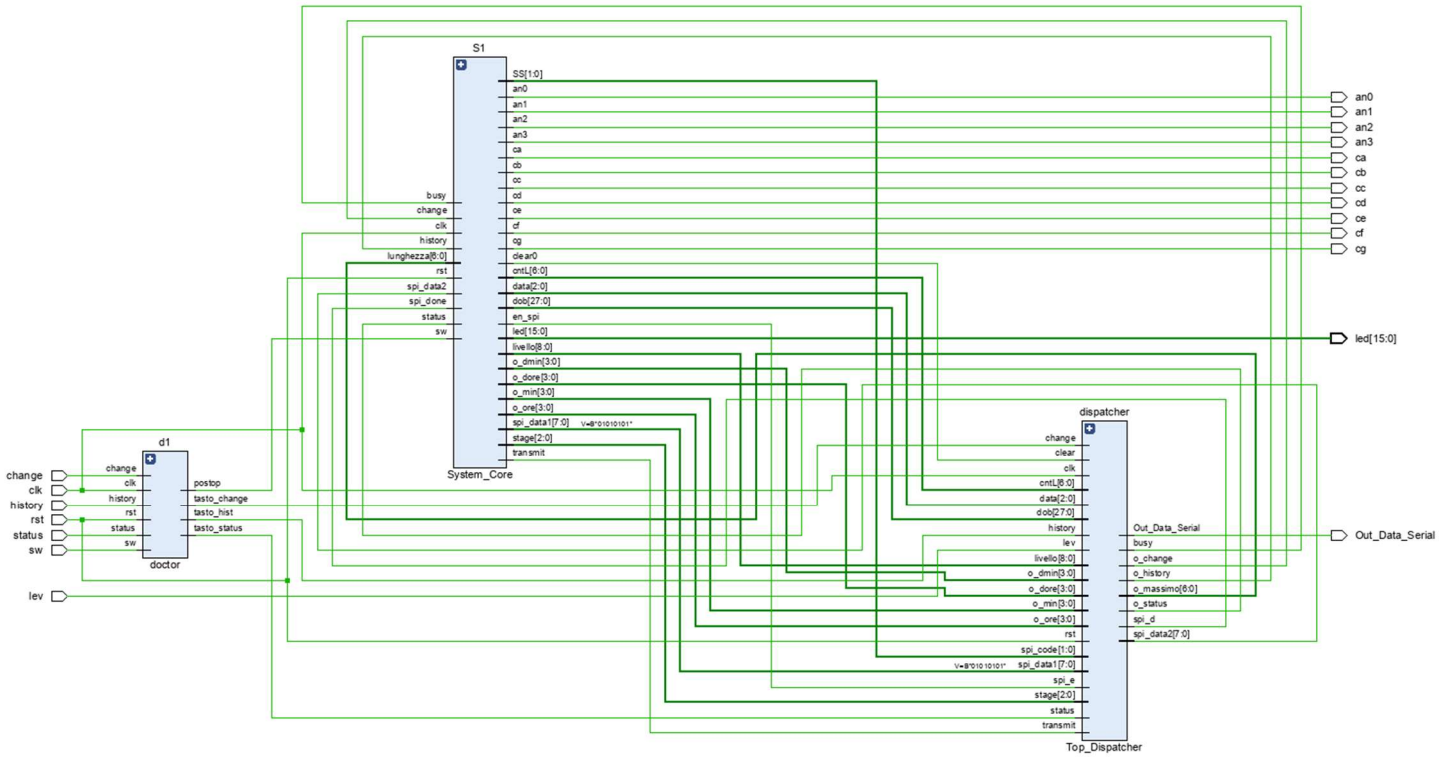




We had problems with the post implementation testbench, failing to provide screenshots

Topmodule

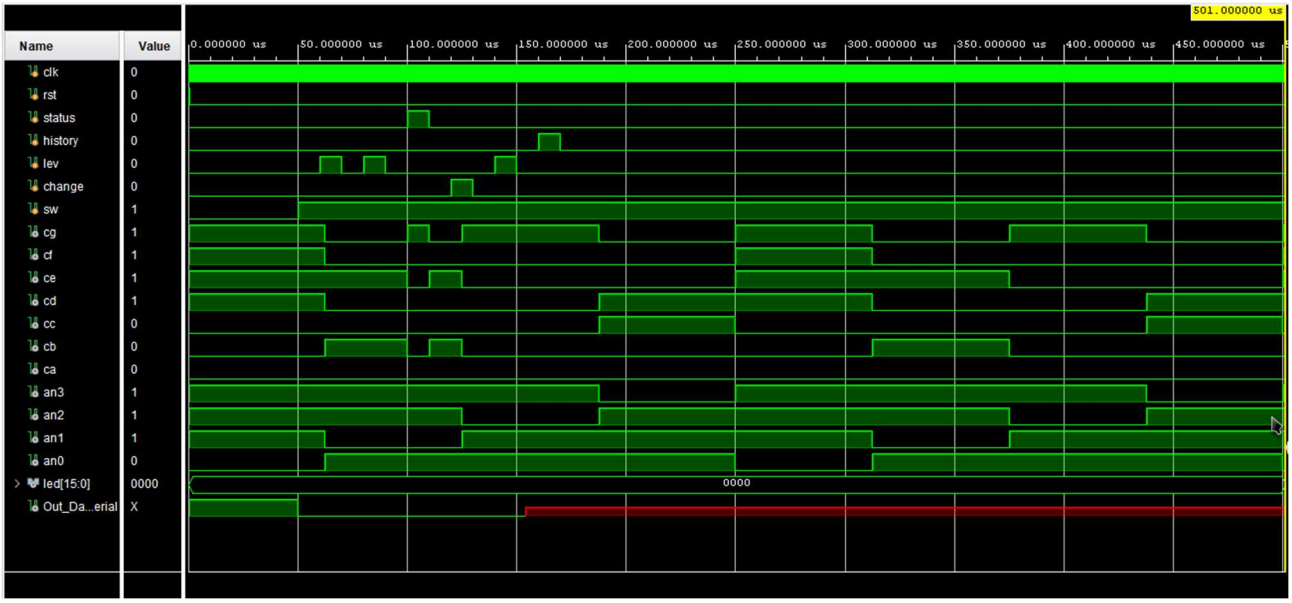
Schematic:



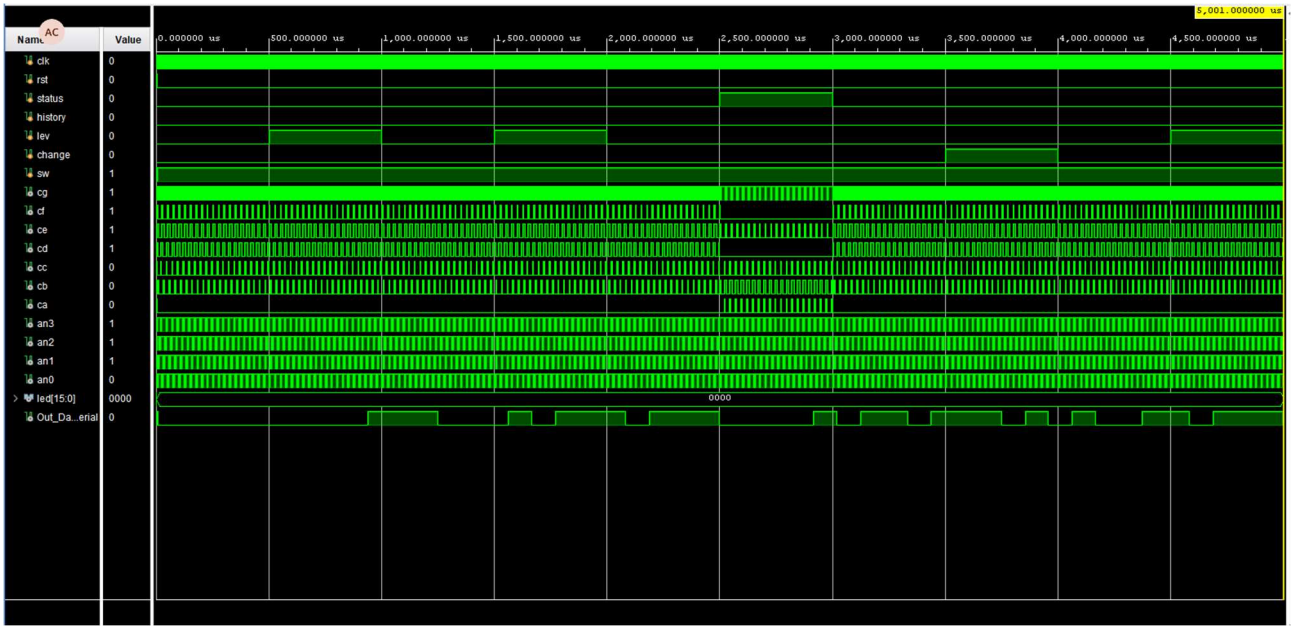
Testbench

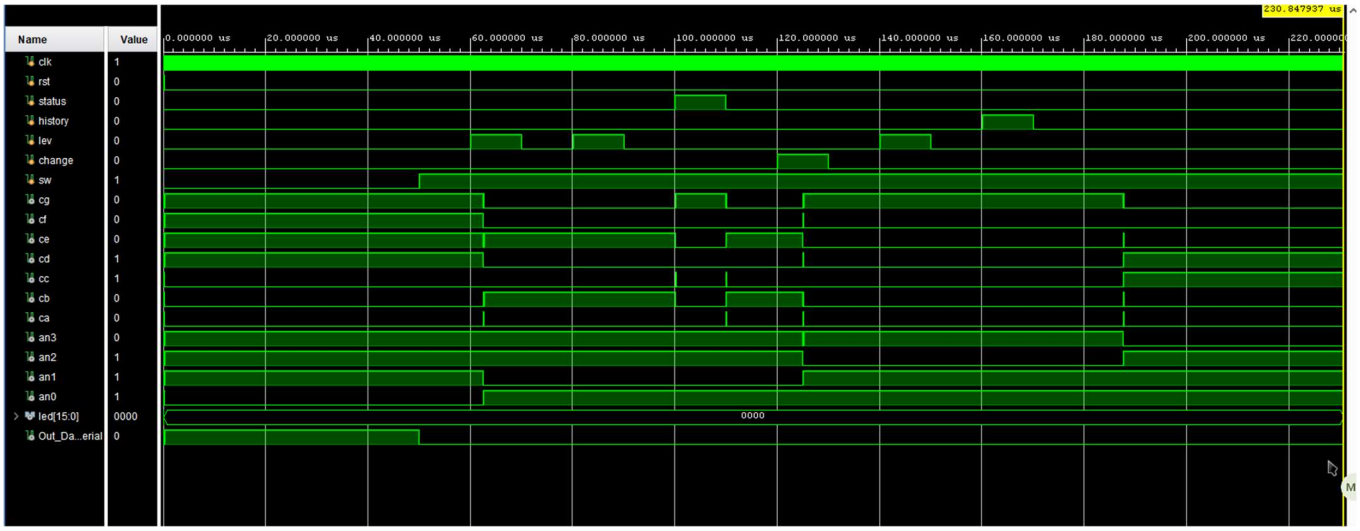
The testbench of the topmodule tries to show its operation, we noticed a glitch in the behavioral of the Out_Data_serial signal, which we do not notice in the other simulations. We tried to have a decent timing

Behavioral

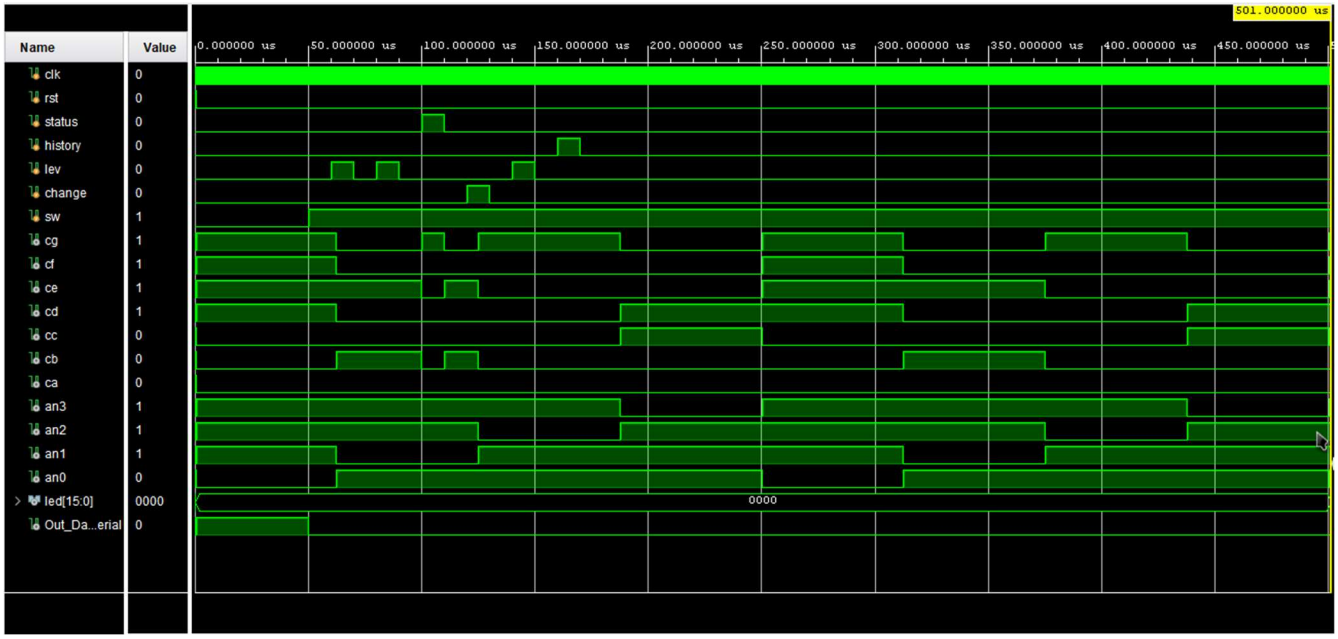


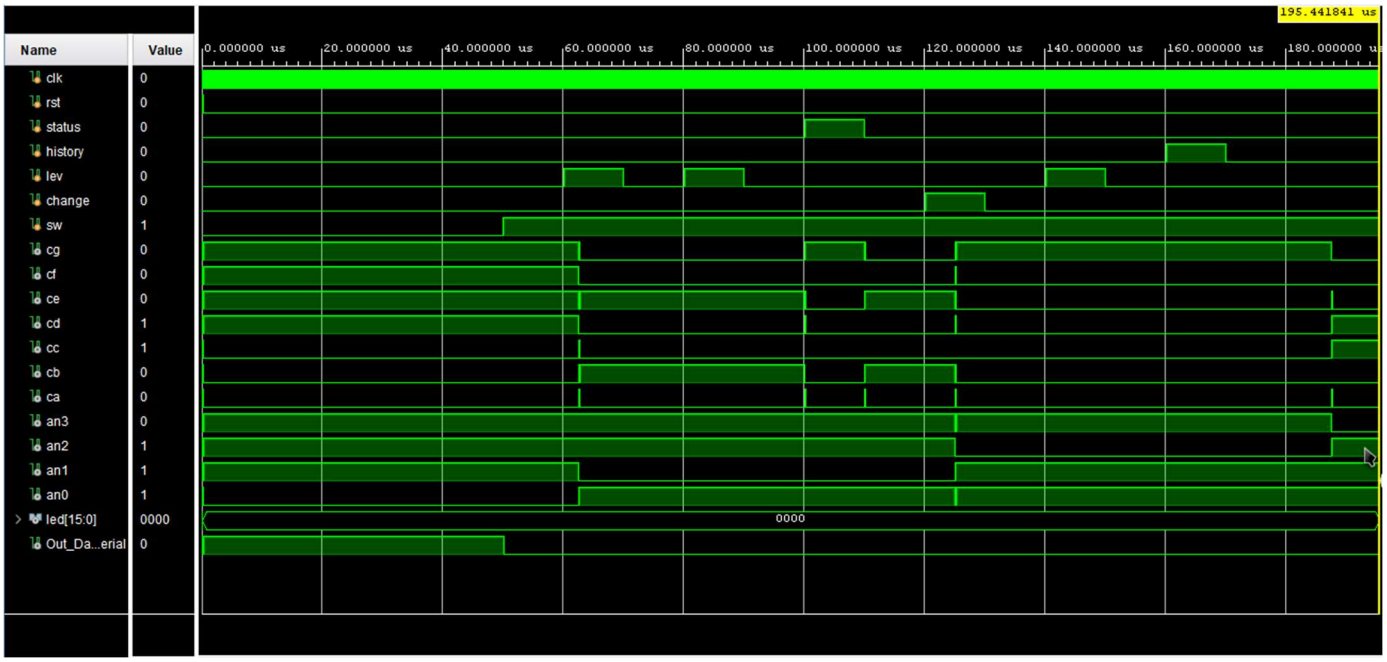
Post Synthesis Functional and Timing:





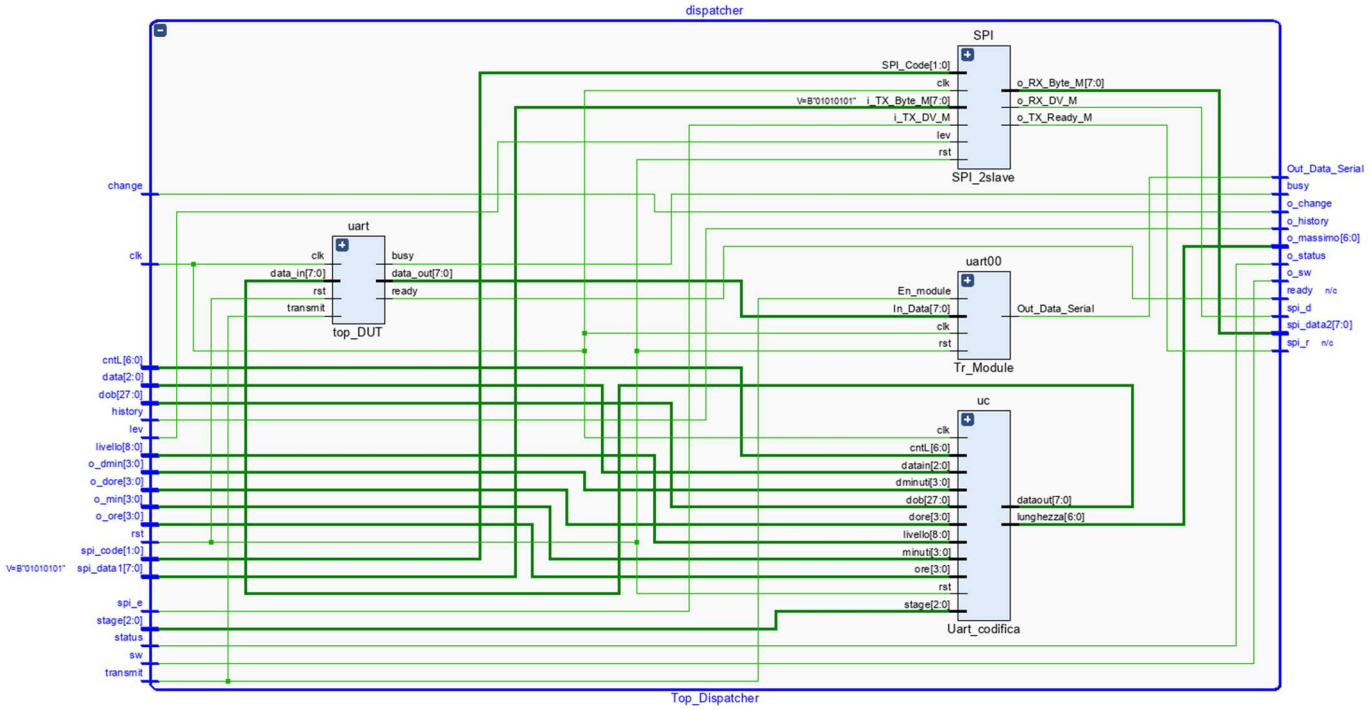
Post Implementation Functional and Timing:




195.441841 us
Name
Value
0.000000 us
20.000000 us
40.000000 us
60.000000 us
80.000000 us
100.000000 us
120.000000 us
140.000000 us
160.000000 us
clk 0
rst 0
status 0
history 0
lev 0
change 0
sw 1
cg 0
cf 0
ce 0
cd 1
cc 1
cb 0
ca 0
an3 0
an2 1
an1 1
an0 1
led[15:0] 0000
0000
Out_Da...erial 0

Dispatcher

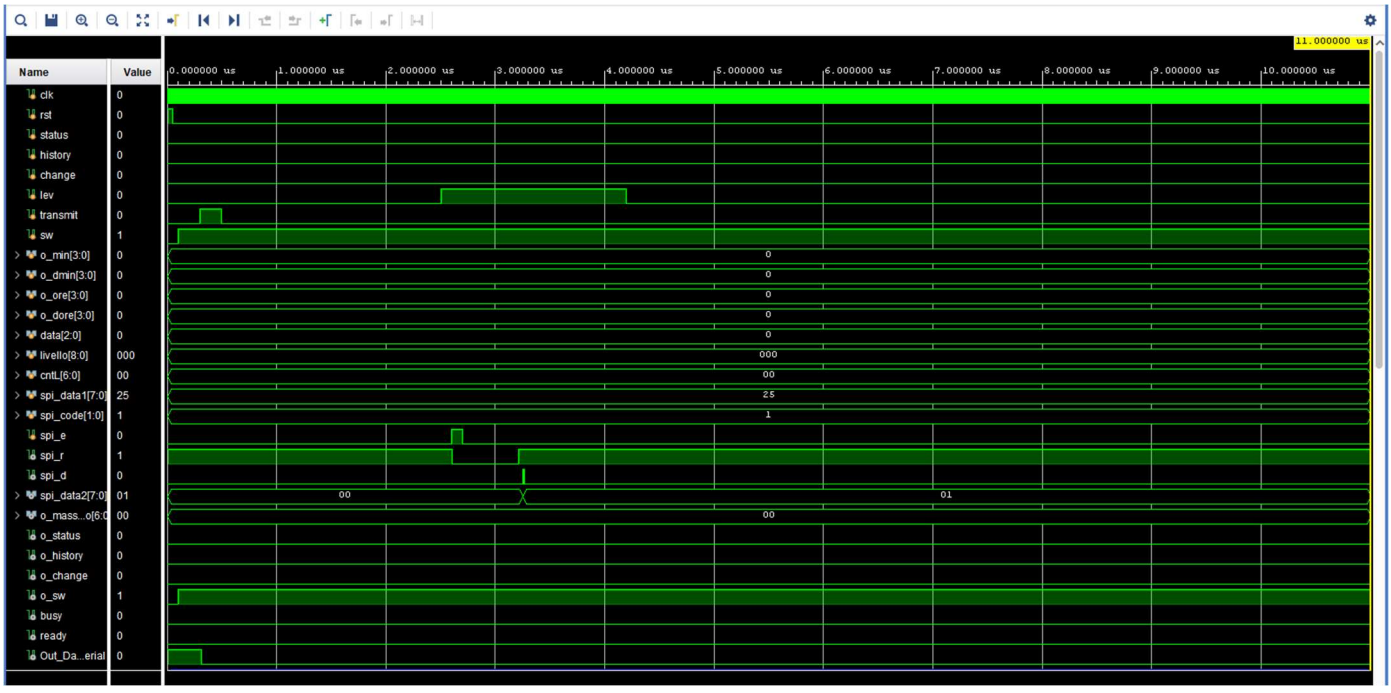
Schematic



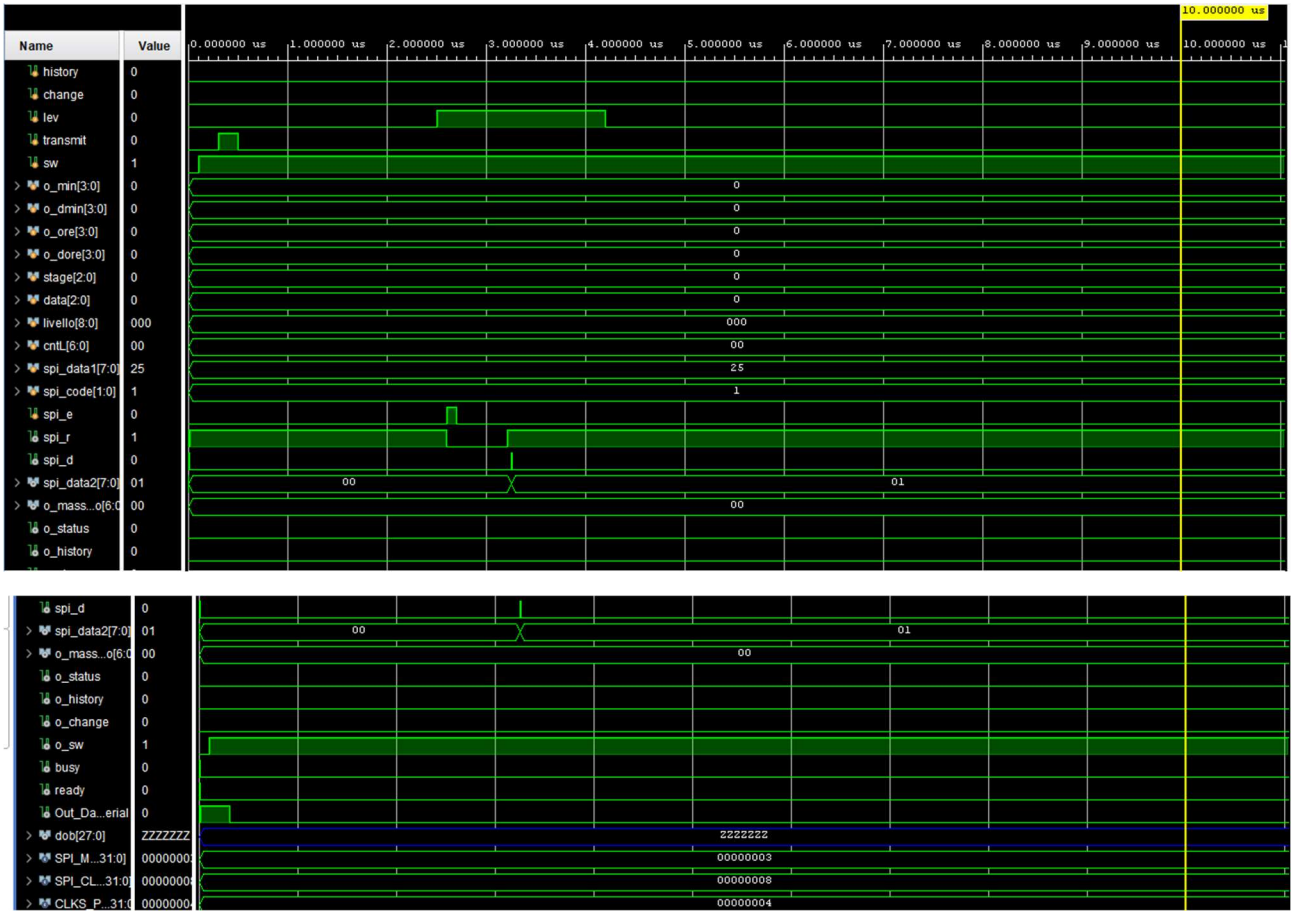
Testbench

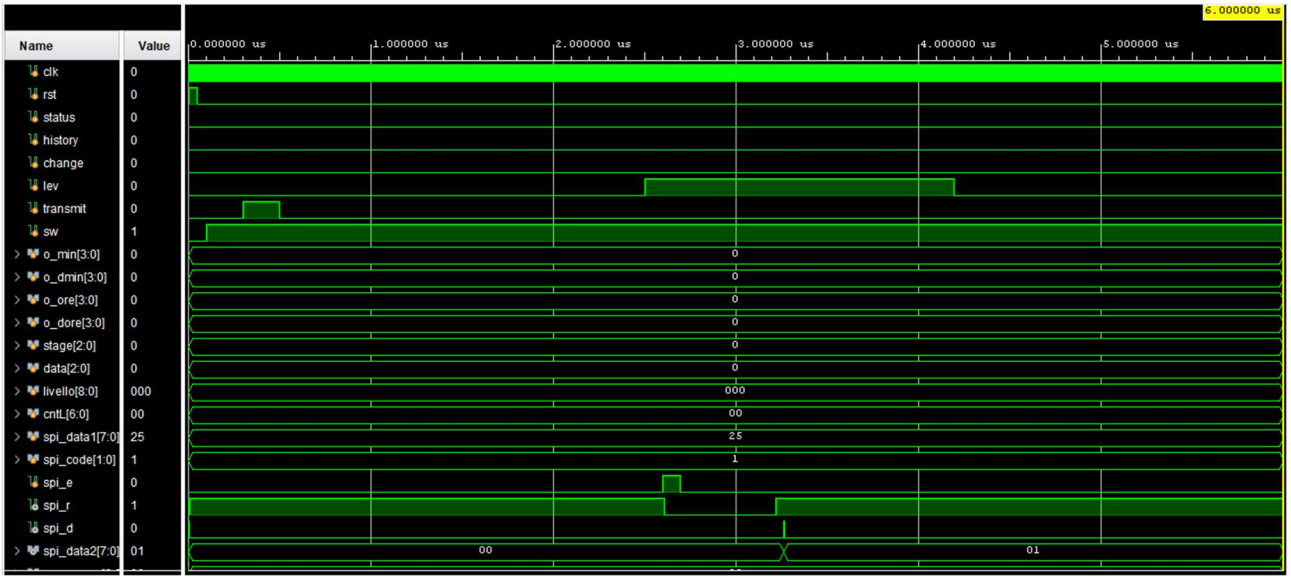
The dispatcher testbench comes in handy to see if the links are working properly

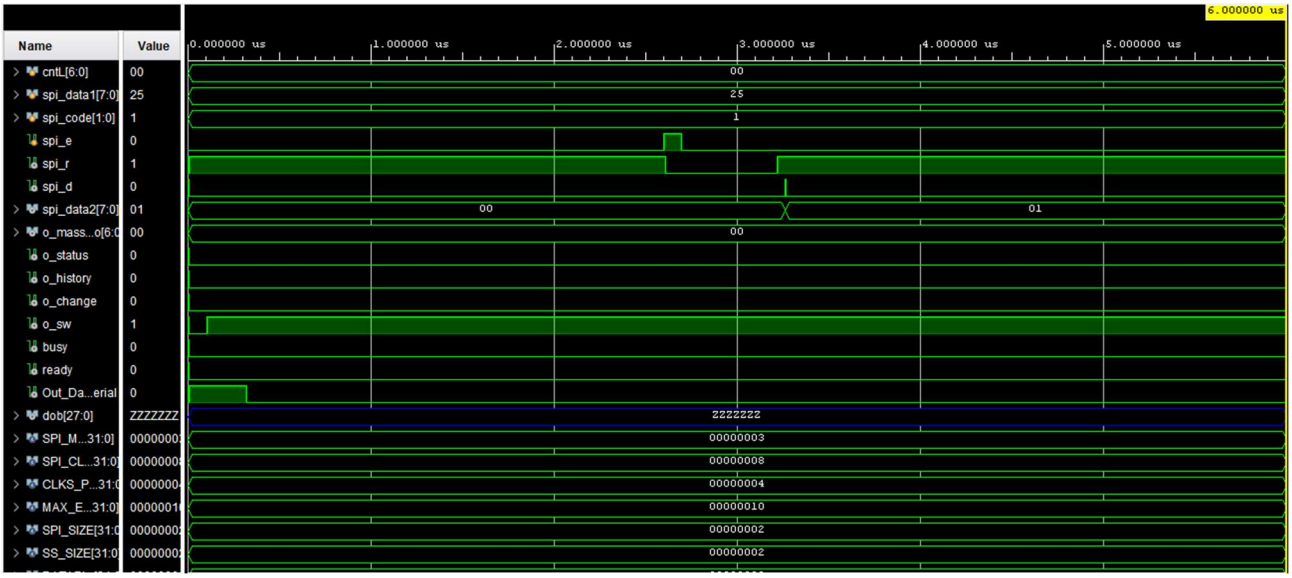
Behavioral



Post Synthesis Functional and Timing:



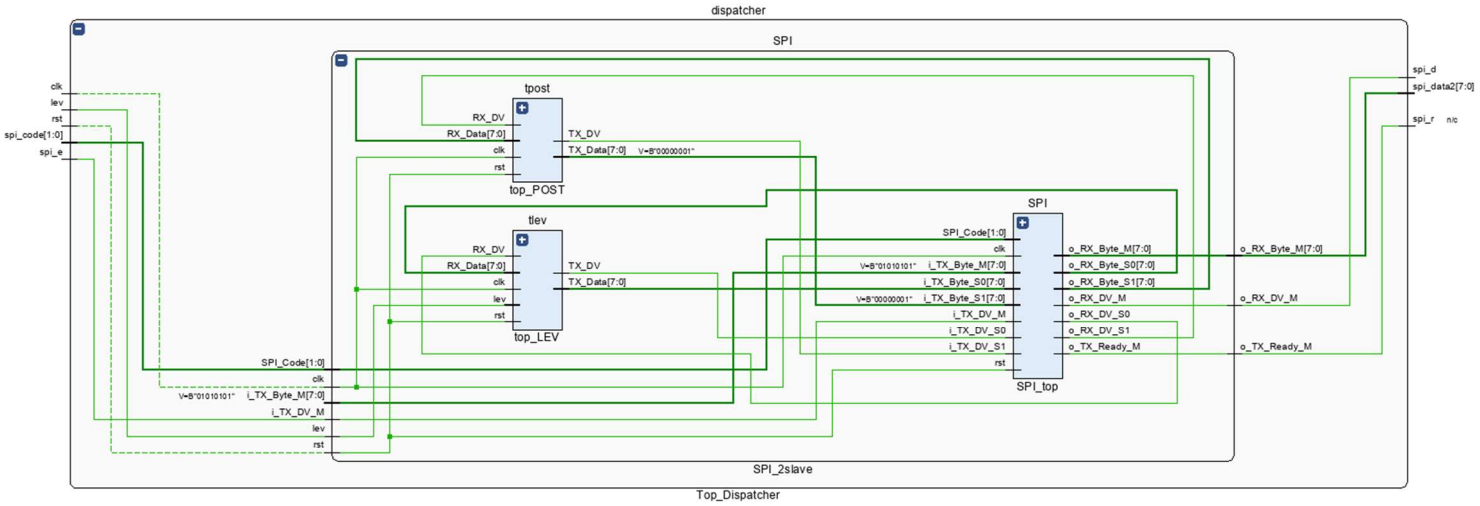




We had problems with the post implementation testbench, failing to provide screenshots

SPI_2slave

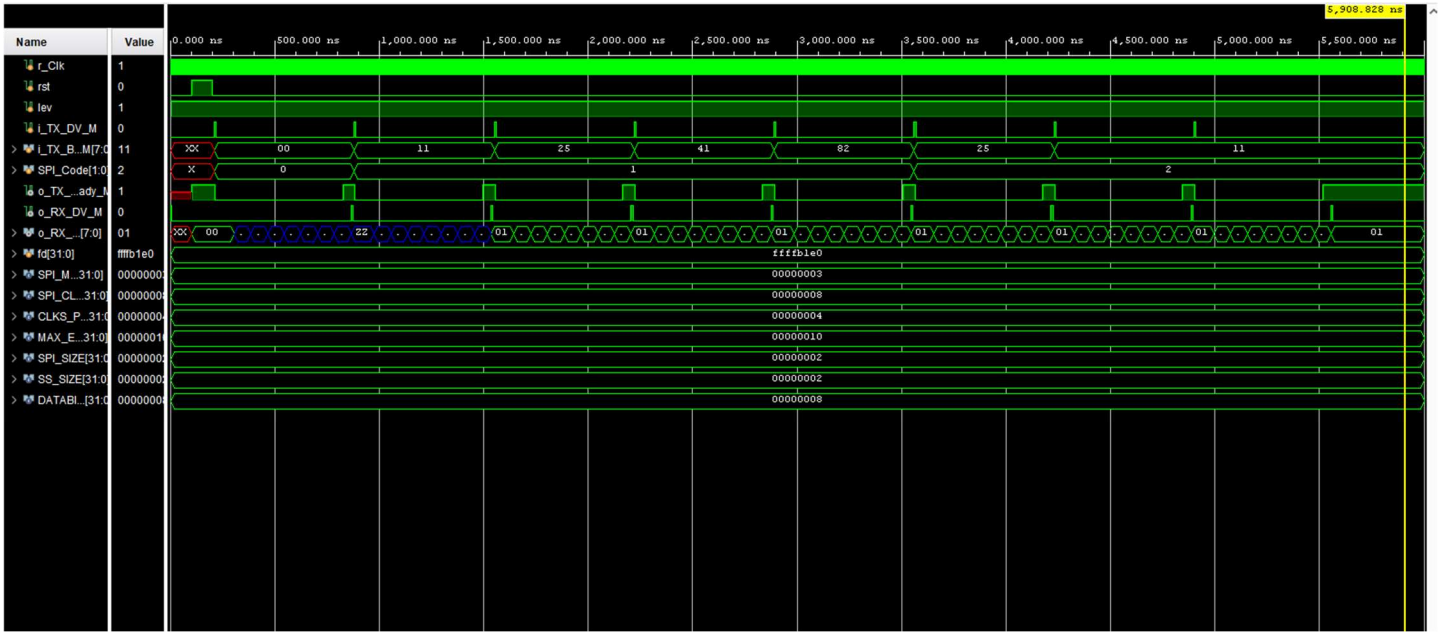
Schematic



Testbench

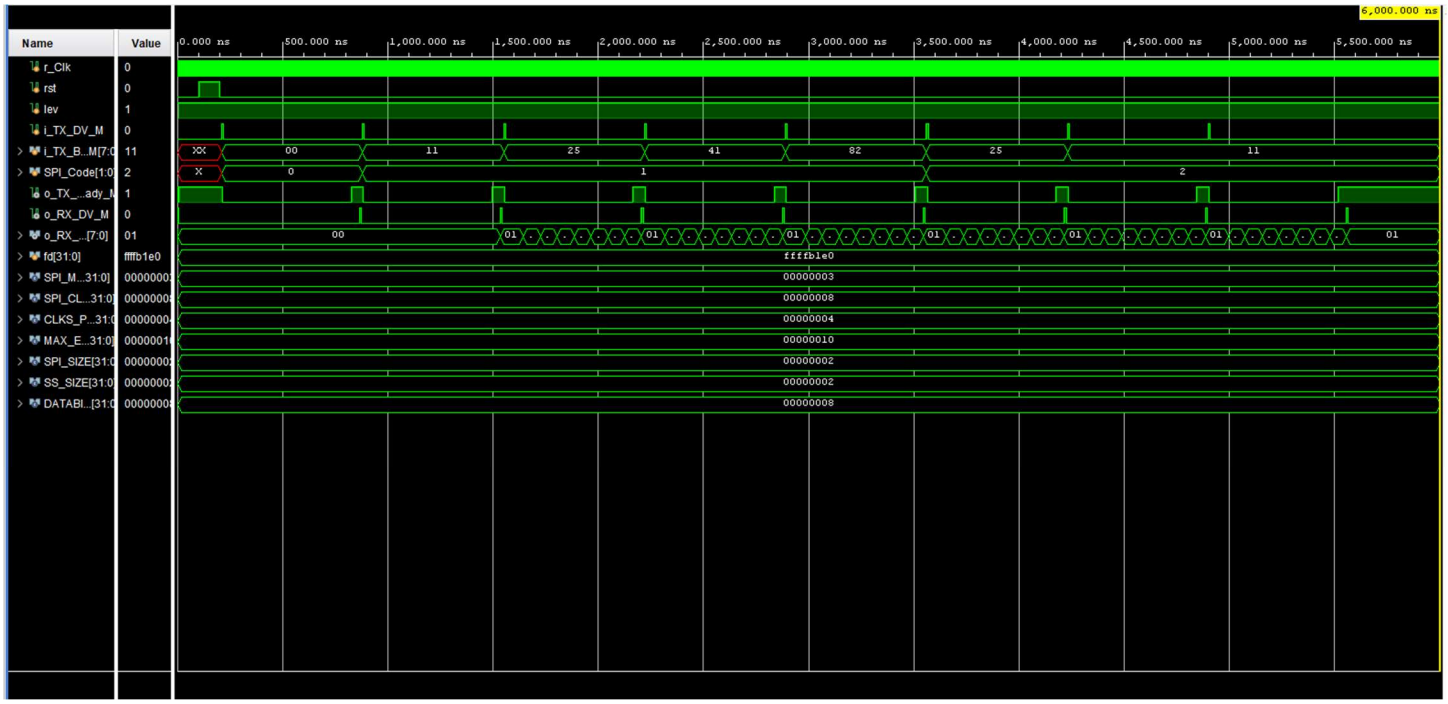
In the testbench we do, we show how the spi samples the data, and in particular we show the case where the clear has priority over the lev since the bag change has to take place automatically

Behavioral

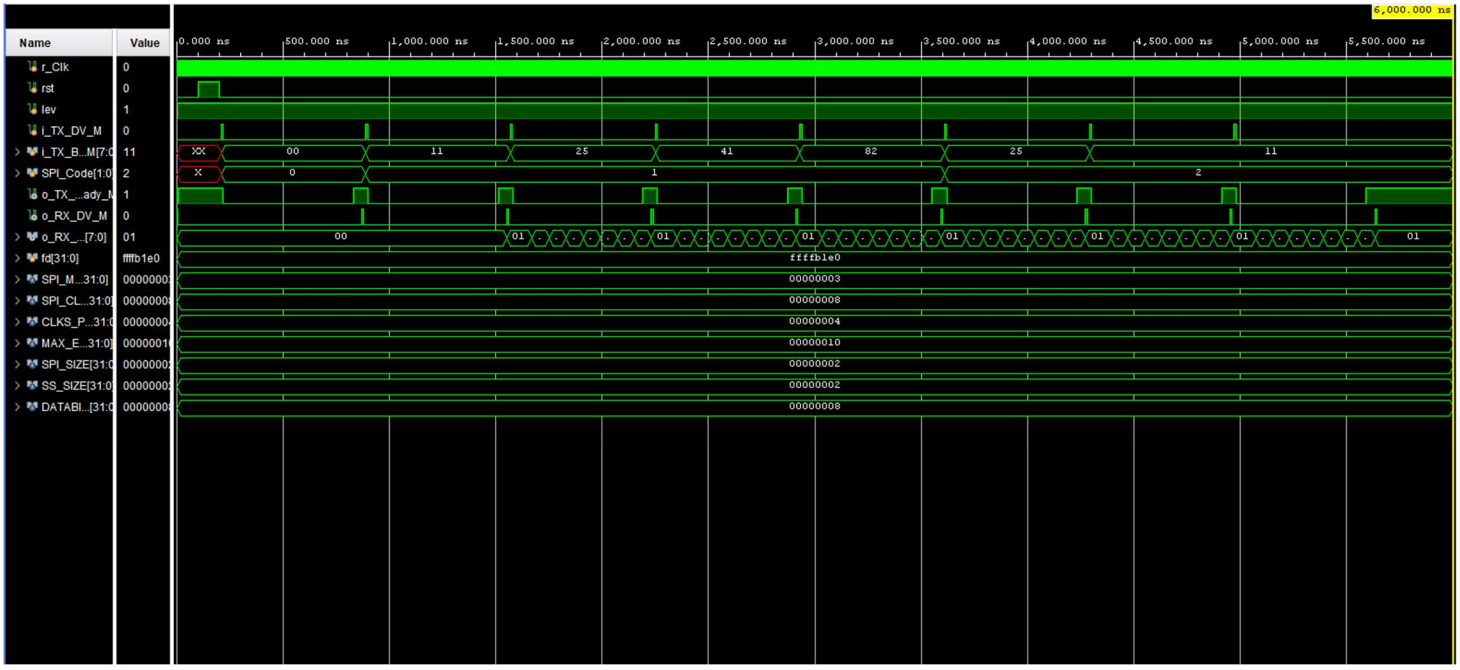


Post Synthesis Functional and Timing:

Functional

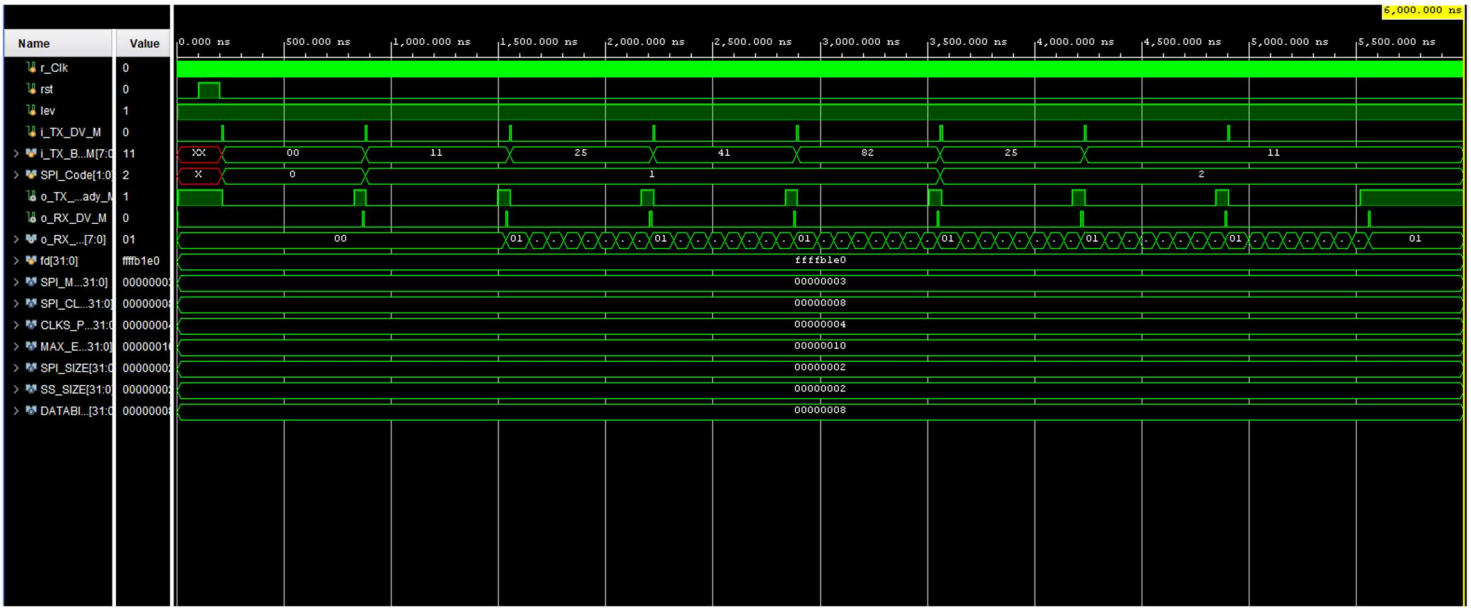


Timing

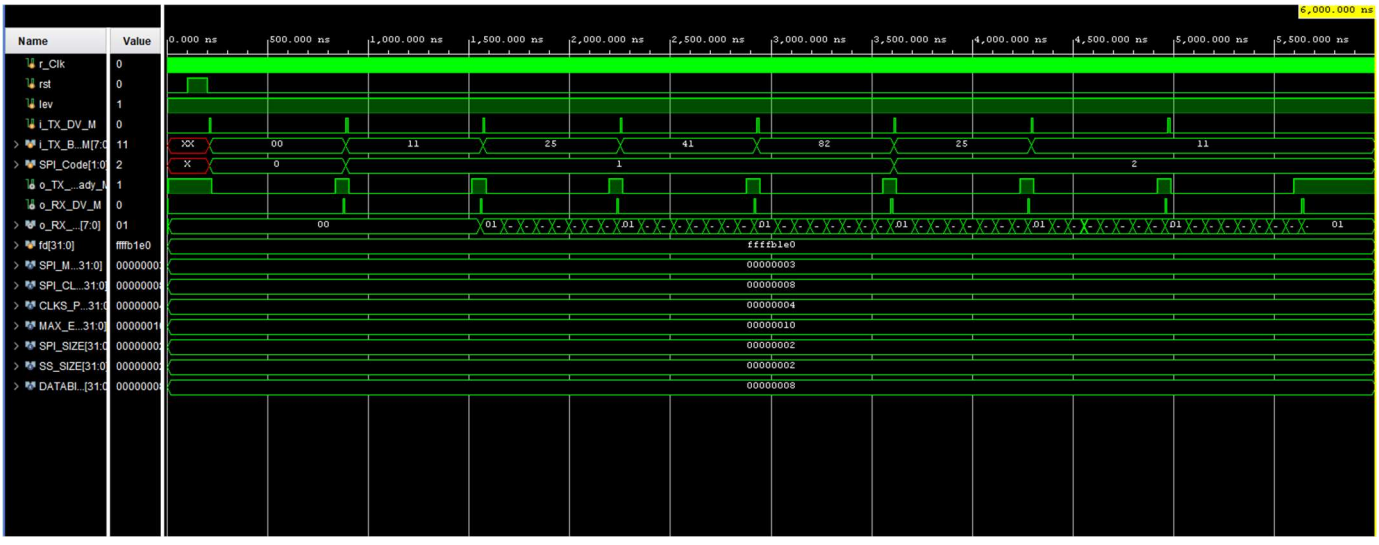


Post Implementation Functional and Timing:

Functional


Name
Value
r_Clk
rst
lev
i_TX_DV_M
i_TX_B...M[7:0]
SPI_Code[1:0]
o_TX_...ady_M
o_RX_DV_M
o_RX_...[7:0]
fd[31:0]
SPI_M...31:0]
SPI_CL...31:0]
CLKS_P...31:0]
MAX_E...31:0]
SPI_SIZE[31:0]
SS_SIZE[31:0]
DATABI...[31:0]
6,000.000 ns
0.000 ns
500.000 ns
1,000.000 ns
1,500.000 ns
2,000.000 ns
2,500.000 ns
3,000.000 ns
3,500.000 ns
4,000.000 ns
4,500.000 ns
5,000.000 ns
5,500.000 ns
ffffb1e0
00000003
00000008
00000004
00000010
00000002
00000002
00000008

Timing


Name
Value
r_Clk
rst
lev
i_TX_DV_M
i_TX_B...M[7:0]
SPI_Code[1:0]
o_TX_...ady_M
o_RX_DV_M
o_RX_...[7:0]
fd[31:0]
SPI_M...31:0]
SPI_CL...31:0]
CLKS_P...31:0]
MAX_E...31:0]
SPI_SIZE[31:0]
SS_SIZE[31:0]
DATABI...[31:0]
6,000.000 ns
0.000 ns
500.000 ns
1,000.000 ns
1,500.000 ns
2,000.000 ns
2,500.000 ns
3,000.000 ns
3,500.000 ns
4,000.000 ns
4,500.000 ns
5,000.000 ns
5,500.000 ns
ffffb1e0
00000003
00000008
00000004
00000010
00000002
00000002
00000008

7) Resource Utilization

Synthesis

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	672	0	20800	3.23
LUT as Logic	672	0	20800	3.23
LUT as Memory	0	0	9600	0.00
Slice Registers	386	0	41600	0.93
Register as Flip Flop	386	0	41600	0.93
Register as Latch	0	0	41600	0.00
F7 Muxes	3	0	16300	0.02
F8 Muxes	0	0	8150	0.00

7. Primitives

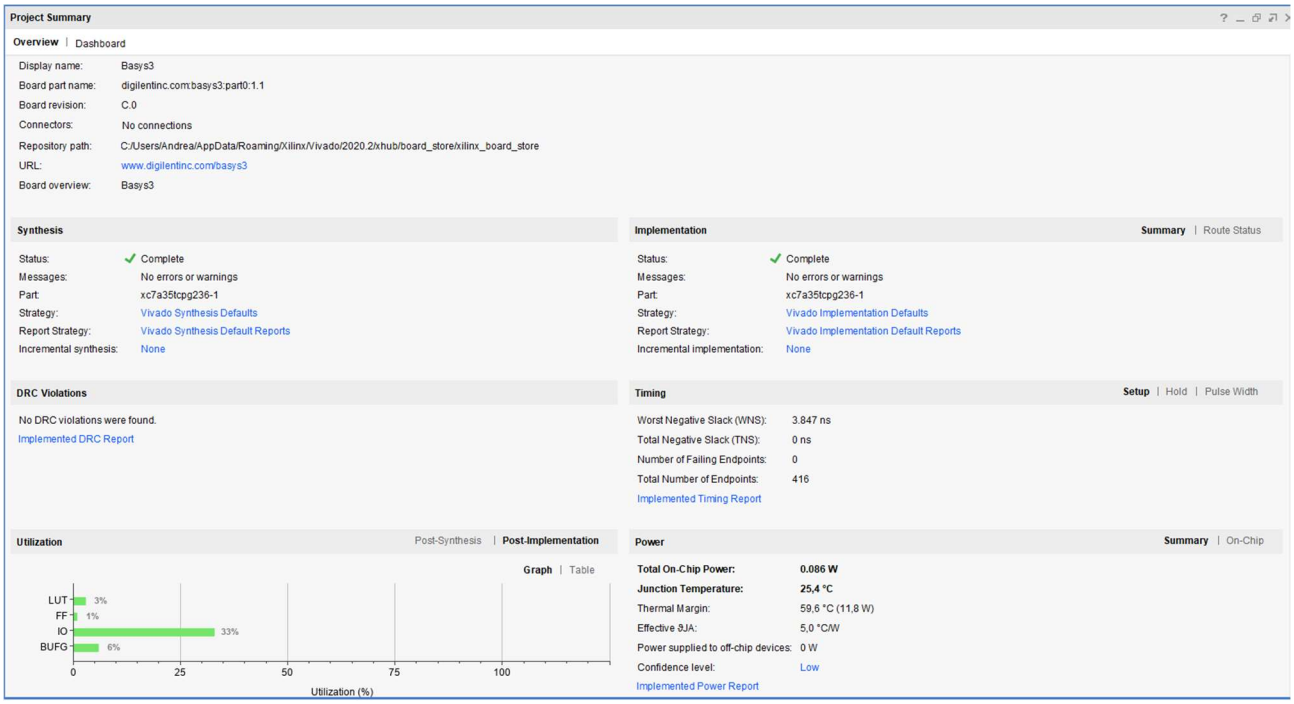
Ref Name	Used	Functional Category
FDCE	356	Flop & Latch
LUT6	304	LUT
LUT3	166	LUT
LUT5	155	LUT
LUT2	149	LUT
LUT4	102	LUT
CARRY4	50	CarryLogic
OBUF	28	IO
FDPE	22	Flop & Latch
FDRE	8	Flop & Latch
IBUF	7	IO
LUT1	6	LUT
MUXF7	3	MuxFx
BUFG	2	Clock

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,823 ns	Worst Hold Slack (WHS):	0,139 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	416	Total Number of Endpoints:	416	Total Number of Endpoints:	327

All user specified timing constraints are met.

Implementation



No warning post implementation

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |  666 |     0 |     20800 |  3.20 |
|   LUT as Logic          |  666 |     0 |     20800 |  3.20 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |  386 |     0 |     41600 |  0.93 |
|   Register as Flip Flop |  386 |     0 |     41600 |  0.93 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    3 |     0 |     16300 |  0.02 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

8. Primitives

Ref Name	Used	Functional Category
FDCE	356	Flop & Latch
LUT6	304	LUT
LUT3	166	LUT
LUT5	155	LUT
LUT2	149	LUT
LUT4	102	LUT
CARRY4	50	CarryLogic
OBUF	28	IO
FDPE	22	Flop & Latch
FDRE	8	Flop & Latch
IBUF	7	IO
LUT1	6	LUT
MUXF7	3	MuxFx
BUFG	2	Clock

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,847 ns	Worst Hold Slack (WHS):	0,153 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	416	Total Number of Endpoints:	416	Total Number of Endpoints:	327

All user specified timing constraints are met.

Optimization Strategies

We have used two implementations to improve the slack: Flow AreaOptimized High and Flow PerfOptimized High

Flow AreaOptimized High Synthesis and Implementation

Synthesis

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	585	0	20800	2.81
LUT as Logic	585	0	20800	2.81
LUT as Memory	0	0	9600	0.00
Slice Registers	394	0	41600	0.95
Register as Flip Flop	394	0	41600	0.95
Register as Latch	0	0	41600	0.00
F7 Muxes	9	0	16300	0.06
F8 Muxes	0	0	8150	0.00

7. Primitives

Ref Name	Used	Functional Category
FDCE	361	Flop & Latch
LUT6	192	LUT
LUT2	183	LUT
LUT3	158	LUT
LUT4	115	LUT
LUT5	112	LUT
CARRY4	50	CarryLogic
OBUF	28	IO
FDPE	25	Flop & Latch
LUT1	16	LUT
MUXF7	9	MuxFx
FDRE	8	Flop & Latch
IBUF	7	IO
BUFG	2	Clock

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,112 ns	Worst Hold Slack (WHS):	0,139 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	428	Total Number of Endpoints:	428	Total Number of Endpoints:	334

All user specified timing constraints are met.

Implementation

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	577	0	20800	2.77
LUT as Logic	577	0	20800	2.77
LUT as Memory	0	0	9600	0.00
Slice Registers	394	0	41600	0.95
Register as Flip Flop	394	0	41600	0.95
Register as Latch	0	0	41600	0.00
F7 Muxes	8	0	16300	0.05
F8 Muxes	0	0	8150	0.00

8. Primitives

Ref Name	Used	Functional Category
FDCE	361	Flop & Latch
LUT6	192	LUT
LUT2	183	LUT
LUT3	159	LUT
LUT4	115	LUT
LUT5	112	LUT
CARRY4	50	CarryLogic
OBUF	28	IO
FDPE	25	Flop & Latch
LUT1	12	LUT
MUXF7	8	MuxFx
FDRE	8	Flop & Latch
IBUF	7	IO
BUFG	2	Clock

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,492 ns	Worst Hold Slack (WHS):	0,151 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	428	Total Number of Endpoints:	428	Total Number of Endpoints:	334

All user specified timing constraints are met.

Flow PerfOptimized High Synthesis and Implementation

Synthesis:

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	918	0	20800	4.41
LUT as Logic	918	0	20800	4.41
LUT as Memory	0	0	9600	0.00
Slice Registers	417	0	41600	1.00
Register as Flip Flop	417	0	41600	1.00
Register as Latch	0	0	41600	0.00
F7 Muxes	5	0	16300	0.03
F8 Muxes	0	0	8150	0.00

7. Primitives

Ref Name	Used	Functional Category
FDCE	374	Flop & Latch
LUT6	324	LUT
LUT2	175	LUT
LUT3	163	LUT
LUT5	151	LUT
LUT4	103	LUT
CARRY4	50	CarryLogic
FDPE	35	Flop & Latch
OBUF	28	IO
LUT1	8	LUT
FDRE	8	Flop & Latch
IBUF	7	IO
MUXF7	5	MuxFx
BUFG	2	Clock

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,112 ns	Worst Hold Slack (WHS):	0,139 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	428	Total Number of Endpoints:	428	Total Number of Endpoints:	334

All user specified timing constraints are met.

Implementation

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	820	0	20800	3.94
LUT as Logic	820	0	20800	3.94
LUT as Memory	0	0	9600	0.00
Slice Registers	417	0	41600	1.00
Register as Flip Flop	417	0	41600	1.00
Register as Latch	0	0	41600	0.00
F7 Muxes	4	0	16300	0.02
F8 Muxes	0	0	8150	0.00

8. Primitives

Ref Name	Used	Functional Category
FDCE	374	Flop & Latch
LUT6	324	LUT
LUT2	175	LUT
LUT3	164	LUT
LUT5	151	LUT
LUT4	103	LUT
CARRY4	50	CarryLogic
FDPE	35	Flop & Latch
OBUF	28	IO
FDRE	8	Flop & Latch
IBUF	7	IO
LUT1	6	LUT
MUXF7	4	MuxFx
BUFG	2	Clock

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,326 ns	Worst Hold Slack (WHS):	0,137 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	444	Total Number of Endpoints:	444	Total Number of Endpoints:	356

All user specified timing constraints are met.

8) Division of tasks

To work on the project, we always worked together, always compared and never divided tasks except in the composition of the report, where we still compared despite having parallelized the work in some parts.

For the realization of the project we started from the system core to generate the operation of the system, then we took care of the communications with spi and later with the uart adding new modules to the system core every time we needed them, and finally we did the encoding of the sentences and the printing on the screen.

9) Warnings

The only warnings that appear are when we create the bitstream, which should indicate that Vivado is unable to find the card definition for "digilentinc.com:basys3:part0:1.1" , we tried searching the internet to see if it was a problem or not , since it appeared out of nowhere the last few times we generated it. We read that it was possible to ignore the warning since it depended on some versions of vivado, since then in the board_path the one related to basys is present in our vivado.