Report Template

Jerry Lemberg, Marko Kosunen, Enrico Roverato

22.01.2016

Aalto University
School of Electrical
Engineering

Contents

1 Workflow

In this project, I was required to build a microntroller model refered to a part of PIC16F84A, with VHDL language. I used synthesizable VHDL to design and describe the circuit. The VHDL description comes from the course exercise 6. According to the datasheet of PIC16F84A microntroller, the execution of an instrution could be devided into 4 phases:

1. Fetch instruction from memory and decode it as arithmetic command and operand(if needed).
2. Catch operand from memory, which is the result from the previous stroage.
3. Do computation according to the arithmetic command.
4. Write the result into memory address and register.

After the VHDL description was done, I used Modelsim HDL simulator to do the simulation and verfication of functionality, meanwhile debugging the errors and warnings.After the simulation and verfication were done, I followed the instructions to do synthesis, palce-and-route as well as final verfication which will be presentated in the follwing sections.

1.1 VHDL description

The VHDL description of PIC16F84A are refered to the exercise 6 which consist of ALU entity for arithmetic operation, RAM entity for memorable storage, MUX for Multiplexer and Finite State Machine(FSM) for finite state control. As the microcontroller's behaviour is stepped by clock, the execution of one instruction will take max four cyles. To make design easier, I used a finite state machine to implement the steps and named then as iFetch, Mread, Execute and Mwrite. The input were assumed as 14-bit logic vecotr, in which from bit 13 downto bit 7 stands for arithmetic instruction, while the part from bit 7 to bit 9 decide bit select and the part from bit 0 to bit 6 are address to write into memory. For each arithmetic instruction, I created the relative functions and collected them in package "Instructionset". Then after one clock cycle, FSM entity tries to decode the instruction. It requires function “string_to_instruction” to get arithmetic instructions and send the signal to ALU entity. Then ALU will do computation of operand from command and thr “operand_array” in RAM and return the result to FSM. Meanwhile, during one

clock cycle, FMS's status has changed from Execure to Mwrite and increase program counter(PC) by 1. FSM enables the writing to the "operand_array" in RAM and return to the iFetch state in the next clock state. In this process, MUX plays role to decide the source of result written to RAM depending on the instruction type which is indicated by message from bit 13th to 12th. If it is byte or literature operation, result shall be wrriten to memory. If the result is bit type, then the result shall be written to register.

1.2 Interface Construction

During this step, I refered to the example code of SPI interface which consist of a chain of Digital Flip-Flop connecting to n-bit register. The VHDL code from exercise originally cannot be utilzed for synthesis because the test bench connectes all components but not synthesizable at all. So I have to modify the test bech file into an interface connecting each components together and split the test part as a new test bench to do simulation and verfication. In this interface architecture, I should keep the port map of entities created and the necessary process. As I realized the importance of this step after browsing the instruction, I took two days to finish it.

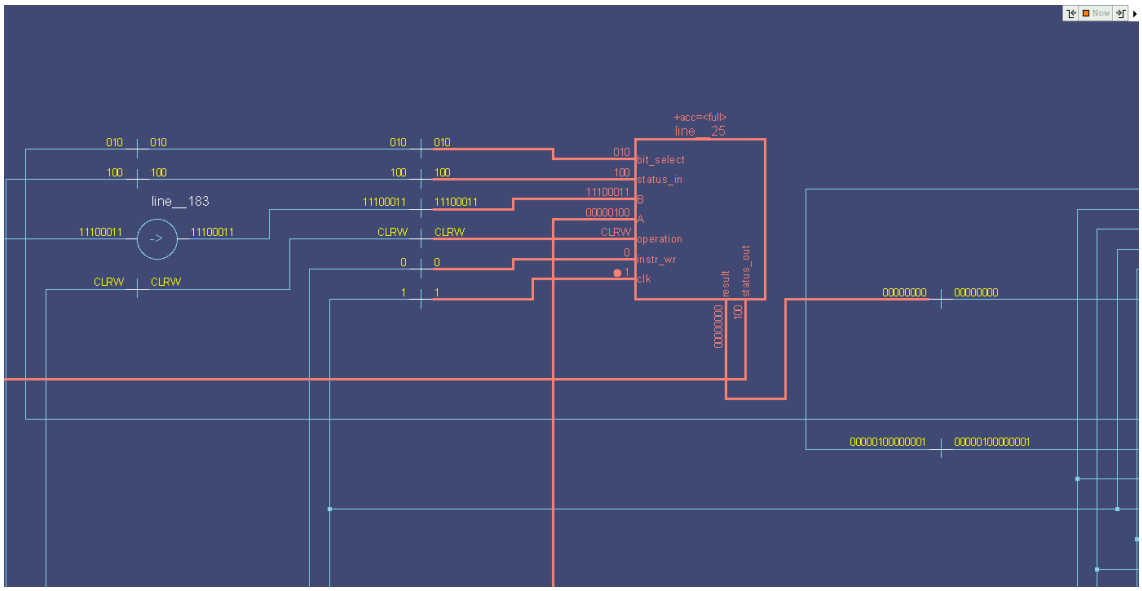

Figure 1.1: microcontroller interface

I declared a microntroller_inteface has:

1. generic
    - depth: memory size
    - length: instruction size
2. port

- clk: clock input
- rst: reset input
- instrwr: input enable writing to instruction
- PC_tot: input total number of instructions
- instrution_in: input instruction, string type
- opcode_status: output FMS state
- statusOutput: output arithmetic status
- resultOutput: output arithmetic result
- addr: output current memory address

1.3 Test Bench

Except the signals for components connection, there are four signals assign to input and output as concurrent process statement. Base on that, I created a new test bench called "ALU_tb" to do the functionality test. There are two process in the test bench: "clk_finite" and "Reading_text". "clk_finite" will provide finite length of clock input and enable reset originally for a while to initalize the signals. "Reading_text" will firstly set instrwr as 1, reading the text lines from input file until the end, storing the instruction into memory via instruction_in and increment PC_tot. After the reading is done, process set instr_wr as 0, and write the output when the FSM status turns to be Mwrite. The output file will show address, arithmetic result, arithmetic status.

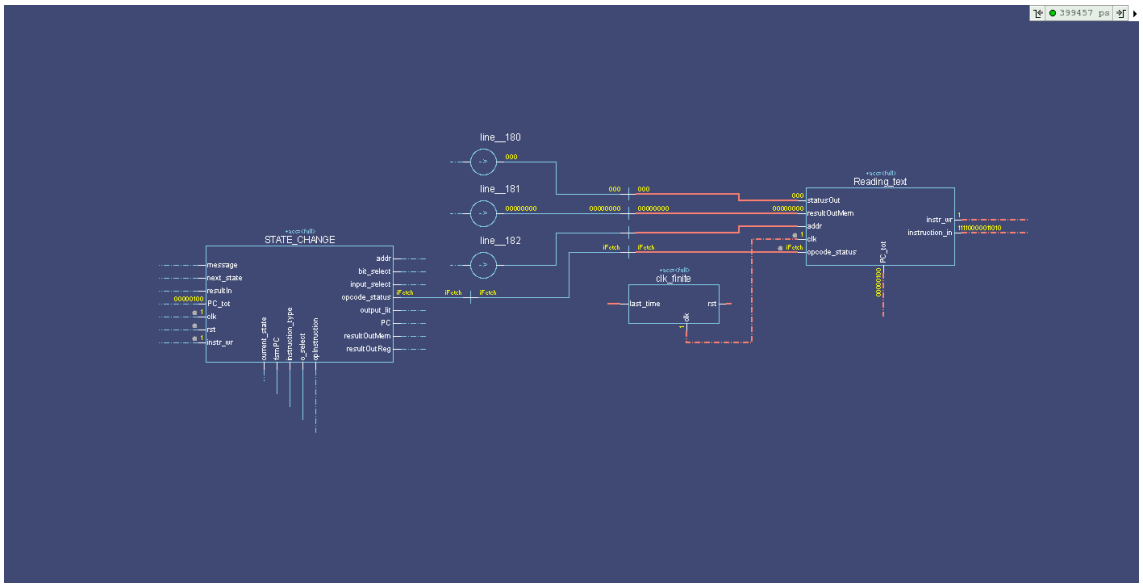

Figure 1.2: Test Bench

1.3.1 21st May

On the original test bench, I merge the signals for address, operation_wr, instruction_wr on both RAM and FSM side because they shall not be separated on the test bech level. As I suppose, RAM accepts to store the instruction line from text file, and the enable port is instr_wr, controlled by external signal. Once the reading process finished, the instr_wr should be disabled and finite state machine started to extract, decode, computing and output result. So the op_wr is controlled by finite state machine itself. Although the instr_wr signal is transferred to both RAM and FSM, as long as it is disabled, the FSM should not work. Similarly since the op_wr is controlled by FSM, RAM won't record the result as long as the op_wr is disabled.

Simulation in Modelsim:

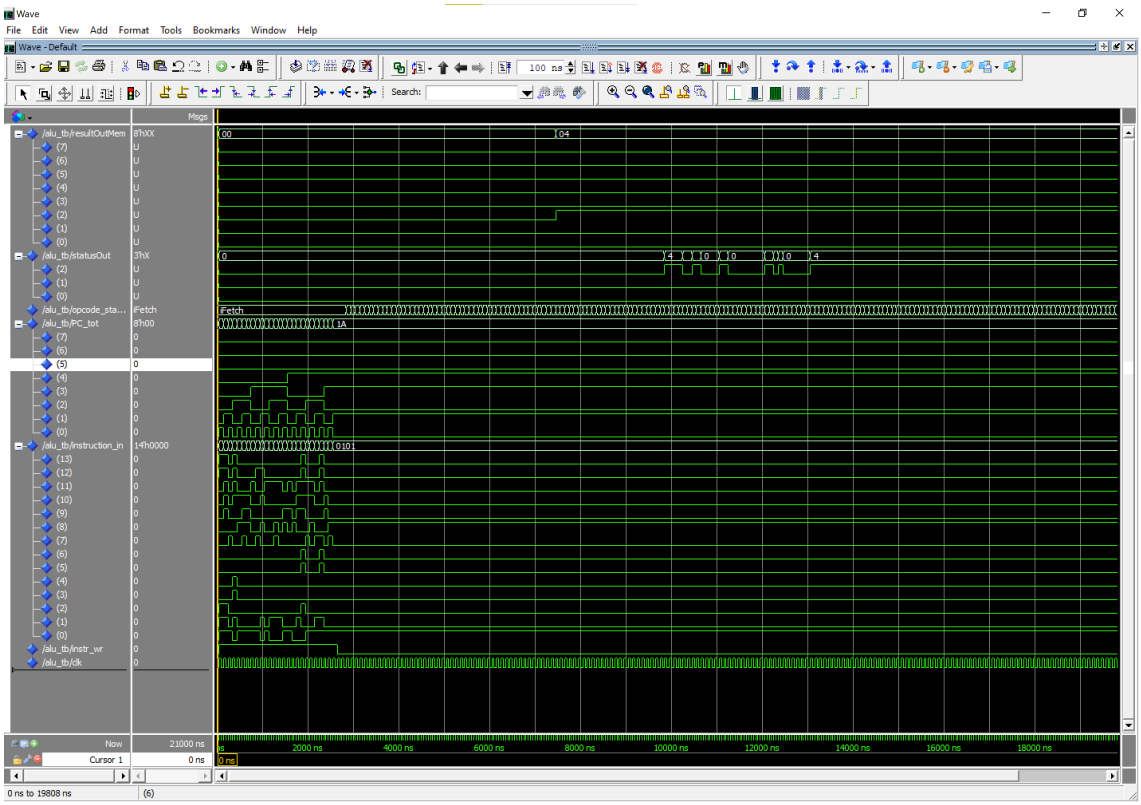

Figure 1.3: Modelsim Simul and verfication

As the picture shows that, the process starts by reading the instructions when the "instruction_in" shows active, and then "instr_wr"was pulled down and "opcode_status" become actvive too because the FSM is working now. So the resultOut and statusOut give the output. Currently I could declare that the computation result is as expected and the VHDL description is ready to do synthesis.

1.4 Synthesis

Synthesis process will convert the VHDL description into a gate-level netlist which describe the interconnection of logic gates. In this process I used Synopsys Design Compiler(DC).

1.4.1 21st May

I review the dicussion history on Slack channel and found that the warnings of latches might happen and I should check it before the synthesis. To avoid latches, I checked all palces with conditional logic such as if-else and case-when. I should declare the assignment for the situation if condition is not met or the other situations in case-when. Besides, if the case-when take enumerate type condition, the "when others=>" are not needed or elaboration will rise alerts. After that synthesiswill add desgin constraints like speed and operating conditions. Designer compiler will return a gate-level list which uses the process vendor's library currently. Then we use Formality to do static verfication of functionality which compares the netlist created and the original VHDL description. The intention is to ensure that the logic should be equivalent on two different language description.

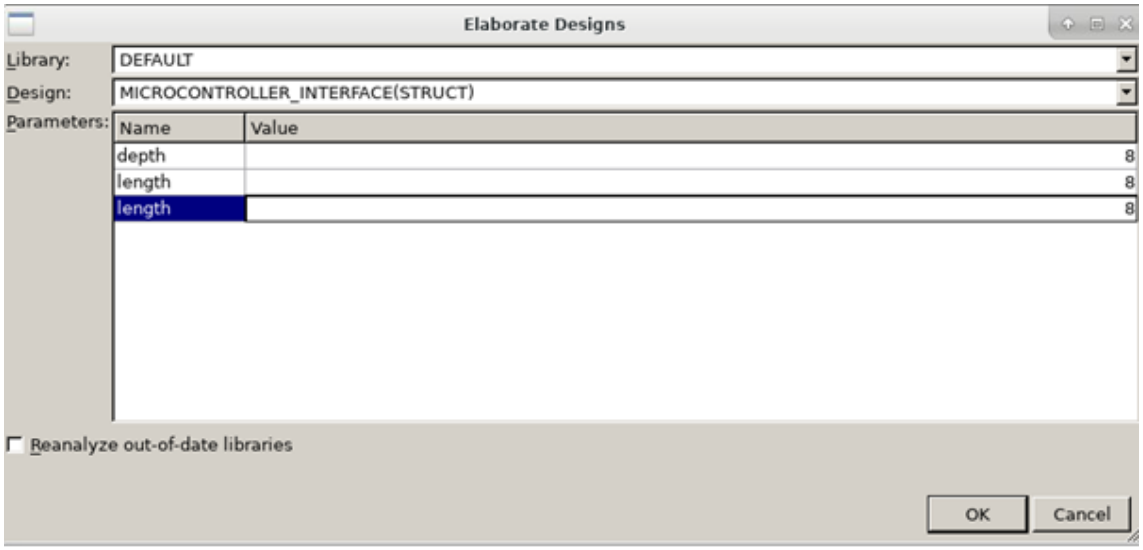

Figure 1.4: Elaboration has redundant parameter

1.4.2 22nd May: Analysis, Elaboration

When I used Design_Vision, I should make sure the "microcontroller_interface.vhd" the top level file is the first to add, in other words, the top 1 in list. The order of other VHDL files are not matter even if they used each other as package(like "Instructionset.vhd"). After I loaded vhdl files, it was time to do elaboration. I found that the value for length delcaration are generic parameters instead of constant

parameters. So I went back to modify the VHDL files to removed the constant number "N" and replace it with "length". I make "length" and "depth" as generic parameters in all VHDL files. Finally when I come back to elaboration, I found that the parameters had one extra "length". I still don't understand it but finally I moved the unecessary generic parameter from ALU and MUX entity, and here is no redundant parameter.

1.4.3 23rd May: Constraint definitions, Compilation

During the compilation, the system alerts that "the inital value for signal 'xxxxx' is not supported for synthesis. Presto ignores it. " So I removed all initial vlaue for signals and variables. It created another issue that circuit behaviour might have mestable status if some signals are not initalized. Then I noticed that I can initalize them all in rest session. To do this, I went back to modify the entities architecture that in their process sensitive with rst, all signals are assigned with initial value. And the test bench could define 1/20 running time in the begining as reset phase.

1.4.4 24th May

Today I am still in Elaboration session, and spent time to reduce the warnings in FSM. I found that in case conditional process for FSM, it would be best to declare the assignment to a signal in all situations, or it will make mestable status too. For signals which requires no operation in other phases, I should move the assignment line to another synchronous process which is triggered by clock and keep checking current stauts of FSM. I also removed the length defiend in package file because the generic parameters "length" and "depth" might be variable. I also noticed that the system sent many notifications about FlipFlop phenomenons in each entities. I asked question in Slack and assistant professor supposed it as out of problem.

```
               '/home/xzhong/E3540/vhdl/microcontroller_interface.vhd'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|  statusOutSPI_reg   | Flip-flop |   3   |  Y  | N  | N  | N  | N  | N  | N  |
| resultOutMemSPI_reg | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|     addrSPI_reg     | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|  resultOutReg_reg   | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
===============================================================================
Presto compilation completed successfully.
Warning: Overwriting design file '/home/xzhong/E3540/synthesis/microcontroller_interface_8_8.db'. (DDB-24)
Elaborated 1 design.
Current design is now 'microcontroller_interface_8_8'.
1
Current design is 'microcontroller_interface_8_8'.
design_vision>
```

Figure 1.5: FlipFlop notification

When I configure the drive strength and load to input and output ports, I found that Design_Vision cannot recognize the ports as long as they are multibi array. And the constraints check will rise the area violoation instead of power violation

mentioned in instruction because we set: "set_max_area 0". Assistan professor supposed it as normal situation.

In max transition session, the violation happens with Net points n835, n8337, n8339 which are related to B0 bit. As I know, B is the input of ALU to accept the result from register and it will be sent to arithmetic functions in Instructionset entity.

In Static verification session, the Synopsys Formality checks the logic of VHDL code and netlist created from DC. When I set top design among the vhdl files loaded, I met another error:

- **Error: unsuppressed RTL interpretation messages**
- **Failed to set top design to microcontroller_interface**
- **Warning: Out of range write possible, may cause simulation and synthesis mismatch. (Signal: instruction_array Block: RAM.rtl/state_reg File: /home/xzhong/E3540/vhdl/RAM.vhd Line: 53) (FMR_ELAB-146)**
- **Warning: Index may take values outside array bound, may cause simulation mismatch .. (Signal: instruction_array Block: RAM.rtl/state_reg File: /home/xzhong/E3540/vhdl/RAM.vhd Line: 55) (FMR_ELAB-147)**

I Gooled the warning content and found that it happens when we use integer as counter to address the memory storage. So the solution is to change the program counter type to std_logic_vector and convert it to integer when needed.

1.5 Palce and route

1.5.1 25th May

In previous process, I saved the TCL script of DC and Formality, and used to automate the process to do synthesis. It saved much time for me to debug the potential issues in VHDL files. Today I continue to do place and route in Encounter. The problem is that Encoutner's pin editor has fixed-size window while the OK button is below the desktop, so I cannot confirm the settings to the ports. So I have to use Innovus. I foudn that the default.view can be loaded in Innovus. Definitly it saved much time and energy. When I did the power planning, the console warnned that the VDD and VSS did not exist. The root cause is in the session to specify the floor plan, the distnace between core and margin was not valid for some reasons. Ater I specified the floor plan again, the issue had gone. In this process I found that console is quite useful to monitor the process while UI window is not such important.

After Tiling, I found that several metal layers had violation. So I inserted OPC metal fill shapes on the Metal1, Meta2 and Metal3. After that I verified the metal density again, the layout still had Metal2 and Metal3 violated.

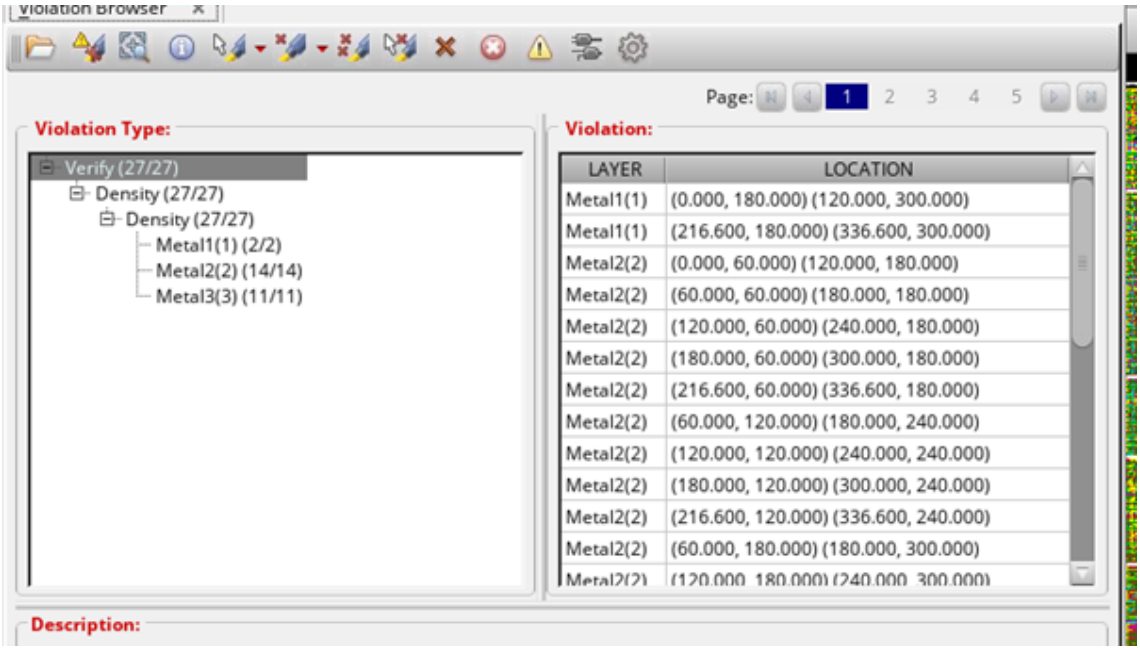

Figure 1.6: Browse the verification result of metal density

1.6 Static verfication and timing analysis

1.6.1 26th May

After I finished the static verfication and timing analysis, the final result is like below:

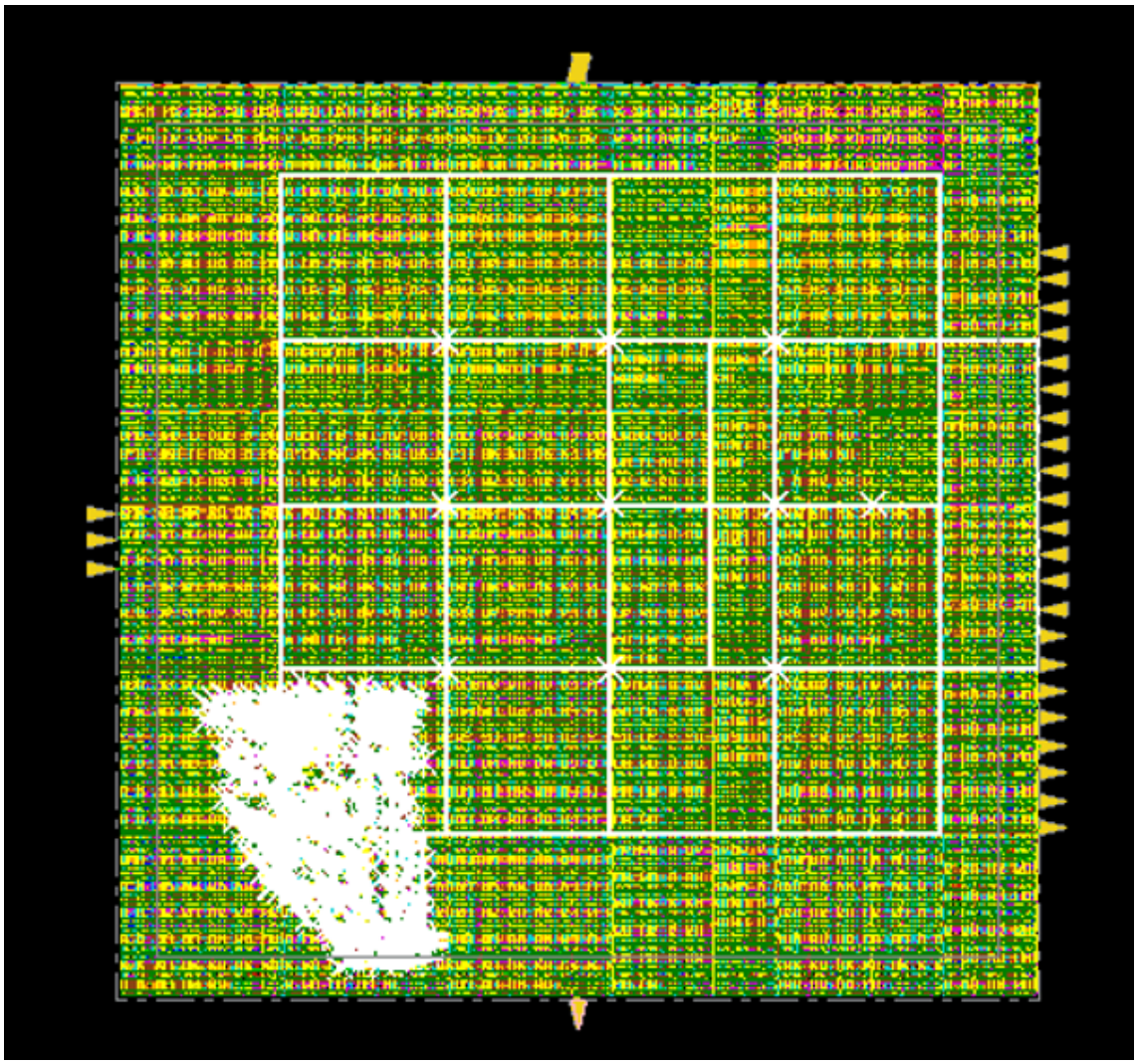

Figure 1.7: Final layout

2 Report

2.1 30th May: Synthesis report

During the syntehsis, I browsed the report about Power, Constraints and Timing path.

Especially in timing report, there are max 10 worst paths in the design which provides direction to optimize the logic behind. 5 out of 10 paths are related to PC_tot[1](configuration) and fsmPC(from bit 7 to bit 3) in FSM, which is the signal to count PC. Other 5 paths starts from instr_wr to resultOutMem in FSM. Definitly it should happen when the test process switch the instr_wr from reading mode to calulcation mode. Besides, I still have two latches on the resultOutMem_reg and resultOutReg_reg. These are output port of FSM to the RAM or register W. The result input will be assigned to one of them depending on the value of output selector which is judged by the instrction type. I havn't find the way avoid this latch without calcaulation errors on result. So I decided to keep them because the correct behaviour of circuit is the first priority instead of reliability.

2.2 30th May: Formality history

The formality alets that some signals are not in the part of process sensitivity list such as "current_state", "Instruction","fsmPC" and so on. These are the internal signals I used in two synchronous process. It is because I want to simply the FSM process strucure by dividing it into two processes. One is for assigning output select and reading_writing mode switch, another one is for executing other assignments in specific status. So I decided to add them only into process sensitivity list.

2.3 30th May: Static timing synthesis report

I browsed the setup_viol report, and found these paths with violations:

1. FE_PHC6961_DUT3_N359 : input_select in FSM, actua transition takes 0.32

2. DUT2_instruction_array_1844 : instruction array in RAM, actual transition takes 0.29

In hold_vio report, the violoated pin and the transition are the same as in setup. And the check report for setup is indicating that the wost paths on timing are the same as in DC analysis. 8 out of 10 are related to the path from PC_tot to FMS's fsmPC signal. Other two are related to the path from instr_wr to resultOutMme. Differently in check report for hold phase, seven critial paths are starting from the RAM's signal "insruction_array" to itself, 2 are starting from RAM's output port "d_out" to itself, which is confusing. One is in FSM, from "insturction_type" to "o_select"(output selector), which is understandable.

The power report for hold and setup session are not visiable becasue the site is not licensed for 'PrimeTime-PX' so it cannot proceed without power anaysis feature. [1].

Bibliography

[1] P. Godoy, S. Chung, T. Barton, D. Perreault, and J. Dawson, "A 2.4-GHz, 27-dBm asymmetric multilevel outphasing power amplifier in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 10, pp. 2372–2384, Oct. 2012.