Homework 4 Report

李京叡, 0616329

***Abstract*—This electronic document is the homework 4 report for the course—Microprocessor Systems: Principles and Implementation at NCTU in 2020 fall.**

I. INTRODUCTION

For sequential circuits, flip flops are the major blocks in which clock plays an important role. The relation between clock and data can modify the whole circuit as reliable or unreliable. As a result, the timing of data with respect to time should be precise and compatible or else the circuit will give out unpredictable results. To analyze the timing of a synchronous design, there are three timing parameters, which are t_{cq}, t_{sestup}, t_{hold}. T_{cq} stands for clock to output, which means essentially propagation delay. t_{setup} stands for setup time, which means the time the data needs to arrive befor the clock. t_{hold} stands for hold time, which means the time the data has to be stable after the clock. There are two main problems that can arise in synchronous logic, max delay and min delay. In this assignment we focus on fix the max delay. Max delay means that the data doesn't have enough time to pass from one register to the next before the next clock edge. And this is the problem which may most commonly found in the critical path. As a consequence, I focus on finding and cutting the critical paths by using timing analysis tools provided by Vivado and one of the solutions to the problem is to insert a flip flop into the long path to cut down the length of the path.

II. FINDING CRITICAL PATH

In Vivado, the EDA tool provides a very detailed timing summary report after it has done the implementation. In the report, there are two critical parameters we should pay more attention to, which are WNS and TNS. WNS stands for worst negative slack time in nanosecond, and TNS stands for total negative slack time in nanosecond. If these parameter are negative, it means that the design fails to meet timing constraint. An example of the timing report is shown in figure 1.In the report, the tool also indicates the paths that violate the timing constraint. It describes the paths by pointing out the register which the paths start and the register of the end of the paths. As a result, it is quite easy to locate the critical path among modules.

Name	Slack ^1	Levels	High Fanout	From	To
Path 41	-0.424	12	94	Aquila_SoC/RISCV_C.../pc_r_reg[4]_rep/C	Aquila_SoC/..._r_reg[2]/D
Path 42	-0.221	15	33	Aquila_SoC/RISCV_C...d_addr_o_reg[0]/C	Aquila_SoC/..._o_reg[9]/D
Path 43	-0.187	18	33	Aquila_SoC/RISCV_C...d_addr_o_reg[0]/C	Aquila_SoC/..._r_reg[0]/D
Path 44	-0.142	12	94	Aquila_SoC/RISCV_C.../pc_r_reg[4]_rep/C	Aquila_SoC/...replica_1/D
Path 45	-0.133	16	33	Aquila_SoC/RISCV_C...d_addr_o_reg[0]/C	Aquila_SoC/..._0_5/RAMA/I
Path 46	-0.132	10	123	Aquila_SoC/RISCV_C...r_reg[5]_rep__35/C	Aquila_SoC/I..._o_reg[23]/D
Path 47	-0.083	12	36	Aquila_SoC/RISCV_C...nter/pc_r_reg[1]/C	Aquila_SoC/...]_rep__1/D
Path 48	-0.081	10	150	Aquila_SoC/D_Cac...AM_reg/CLKARDCLK	Aquila_SoC/D...i_reg[83]/D
Path 49	-0.073	12	36	Aquila_SoC/RISCV_C...nter/pc_r_reg[1]/C	Aquila_SoC/...]_rep__2/D
Path 50	-0.073	12	69	Aquila_SoC/RISCV_C...ode/pc_o_reg[0]/C	Aquila_SoC/R...reg[4][1]/D

Figure 1

III. FIXING TIMING VIOLATION

The first critical path I try to fix is the path from the pc_r of the program counter to the likelihood_reg of the branch prediction unit. In FPGA, a register is implemented by a flip flop, so I create an extra register and put it into the path. Since the path crosses multiple modules, I concern where to insert a flip flop into is the best scheme. I've tried inserting a flip flop into the path in each of the modules passed by for different locations of the path for many times. I find that no matter where to put the register in the path can fix the timing violation of the path, or have the WNS and TNS changed significantly. However, when I send the Dhrystone program to test if the processor works normally, many of my attempts fails to have the program run normally. In fact, to decrease the amount of violation of WNS and TNS is very simple, inserting a flip flop in the path can decrease the value of them simply. The most difficult part is to fix the violation path and also have the processor work normally. Finally I add two registers, pc_i_r and dec_pc_i_r for the signals pc_i and dec_pc_i in bpu module respectively. The input signal pc_i and dec_pc_i should store in the registers for one clock cycle before they are passed down. However, adding register in a path causes signals to be asynchronous with other signals, since those paths with extra registers would be delayed for another clock cycle. To solve this problem, I add another register predicted_pc_r to delay the signal predicted_pc for one cycle as well. In this way, all the output signals are delayed for one clock cycle synchronously. As a result, the revised bpu module would be late for one clock cycle compared to the original one. With the revision, the result of WNS and TNS are -0.868 and -189.622 respectively and the result of the performance of the Dhrystone program is decreased to about 0.44 DMIPS/MHz. The result is reasonable, since every instruction has to run for one more clock cycle in the branch prediction unit and DRAM is a relatively slow device.

The second path I try to fix is another path from program counter to branch prediction unit. For this path I revise the core_top.v and insert a register in the wire which connects the pc output from program counter and the pc input of the branch prediction unit. Again, in order to make all signals synchronous, I also insert many registers for each signal going into the branch prediction unit. That is, each one of the signals coming from the program counter is directed to a register, and the register output serves as the input of the branch prediction unit. With these revisions, the WNS and TNS are improved to -0.187 and -1.091 respectively, which are much better than those of the original design.

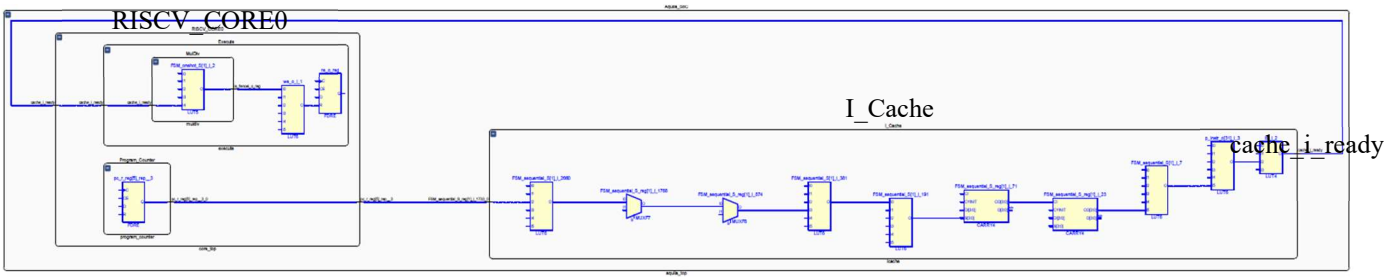

Figure 2

Before I started to fix other paths, I found that there was a function provided by the timing report tool that I had not discovered. That is, the tool can generate the schematic of each path it reports as a timing-violating path. It's very useful and convenient since before I found this function, I traced the paths from the starting registers to the ending registers of the paths by myself. The process was very complicated because I had to check each .v file one by one and check the relation of the input and output of them. Also, generally, I think that the most efficient way to insert a register to cut down the length of a long path is to put a register at the location about middle of the path. Using the schematic, it's easier to find a suitable place where to put a register into. With the help of the schematic, I observe those timing-violating paths and found that almost all of them pass by the I-cache module and the key output signal of the I-cache is p_ready_o, of which the output port name is cache_i_ready. An example of schematic of one of the timing-violating paths is shown in figure 2. Since the related signal is cache_i_ready. I think a suitable location to put a register into is before cache_i_ready. That is, storing the signal for an extra clock cycle. Also, it is necessary to delay p_instr_o for one cycle. After these implementations, the TNS and WNS are improved to both -0.107, which means there is only one timing-violating path left. However, the Dhrystone program cannot run on Aquila. I use ILA to debug, but the relation between p_ready_o, p_instr_o, and p_strobe_i are same as before, which is p_strobe_i coming first and then p_ready_o rises and at the next clock cycle, p_instr_o gives out the requested instruction. I think the key is to make sure p_instr_o is after p_ready_o but somehow there is some problem I haven't figure out.

IV. REVIEW

In this assignment, I've tried many times in inserting a register into a path. The most of time is spending on compensate for the time delay caused by the extra register. Sometimes I decrease the amount of timing violation significantly, but when I send the Dhrystone to Aquila, it cannot work normally. Also, I found that when fix the critical path, the violating amount is not convergent. That means, if I successfully fix one path, it may contribute to generate other longer paths in routing process and sometimes increase the total amount of timing violation amount. As a result. It's hard to follow a regular direction to fix the violation. In other words, the result of my revisions are not predictable.

V. APPENDIX

Name	Constraints	WNS	TNS
✓ **synth_1** (active)	**constrs_1**		
✓ impl_1	constrs_1	-0.187	-1.091

Figure3

```
It tooks 26.06 seconds.
Microseconds for one run through Dhrystone: 26.060061
Dhrystones per Second:                      38372.898438
VAX MIPS:                                   21.840010
DMIPS/Mhz:                                   0.436800


------------------------------------------------------------
Aquila execution finished.


Program exit with a status code 0

------------------------------------------------------------
Aquila execution finished.
Press <reset> on the FPGA board to reboot the cpu ...
```

Figure 4

Figure 3 is the best implementation result I have done. As the figure shows, WNS and TNS I could improve for the best is -0.187 and -1.091.

Figure 4 is the final result of running the Dhrystone program on Aquila with my best implementation. As the figure shows, Aquila works normally and the performance of Dhrystone drops to about 0.44 DMIPS/Mhz. By the way, I also try running the program without bpu, and the result is about 0.44DMIPS/Mhz, too, but the WNS and TNS are much bigger than mine. I think my revision is not very appropriate, since I sacrifices too much on performance to fix timing violation.