ECE559 Report

cl344

December 2018

1 Static Timing Analysis

A comprehensive static timing analysis includes analysis of register-to-register, I/O, and asynchronous reset paths. It verifies circuit performance and detect possible timing violations by investigating data required times, data arrival times, and clock arrival times. The Timing Analyzer determines the timing relationships that must be met for the design to correctly function, and checks arrival times against required times to verify timing. In our design, we focus on the timing constraints on internal register-to-register paths and are not worrying about the I/O constraints. The Timing Analysis is operated with the Base Clock at 50 MHz. We used `Post-fit` and `Fast-corner` for a full timing analysis.

1.1 Identifications

1.1.1 Worst Case Slack

The Timing Analyzer reports the result of clock setup checks as slack values. Slack is the margin by which a timing requirement is met or not met. Positive slack indicates the margin by which a requirement is met, and negative slack indicates the margin by which a requirement is not met. The Timing Analyzer determines clock setup slack as shown in the following equation for internal register-to-register paths, where *slack* is Clock Setup Slack Time, $t_{require}$ is Data Required Time and t_{arrive} is Data Arrival Time.

$$slack = t_{require} – t_{arrive} \tag{1}$$

Since slack indicates that the clock meets the requirement with some margin, larger slack means better performance. According to the `Report All Core Time` page in our design, The Worst Case Slack against Setup Violations is 13.307 ns. From this result, we are confident that we don't have any failing path since there's no negative slack for the clock domain.

1.1.2 Maximum Frequency

According to `Report FMax Summary`, we find that the maximum frequency at which our design can be clocked is 149.41 MHz.

1.1.3 Critical Path

By using `Report Worst Case Path` function, we find that the worst data delay is 6.528 ns and the path of Worst Case Slack is:

```
shiftreg_buf:input_shiftreg_inst1|mem[1464] :  (0.232 ns)
input_shiftreg_inst1|mem[1464]|q :  (0 ns)
reg_inst|Q[1464]|asdata :  (5.89 ns)
reg6144:comb_6149|Q[1464] :  (0.406 ns)
```

According to the Path above, `reg_inst|Q[1464]|asdata` is the Critical Path since it takes the longest delay which is 5.89 ns. According to the path name, we locate this path in the second buffer of our design.

The plot of Worst Case Path is shown in the plot below. The green region represents the slack area. We can see that with the clock period of 20 ns, our design is fairly fast since the worst slack period is longer than half of the clock cycle.

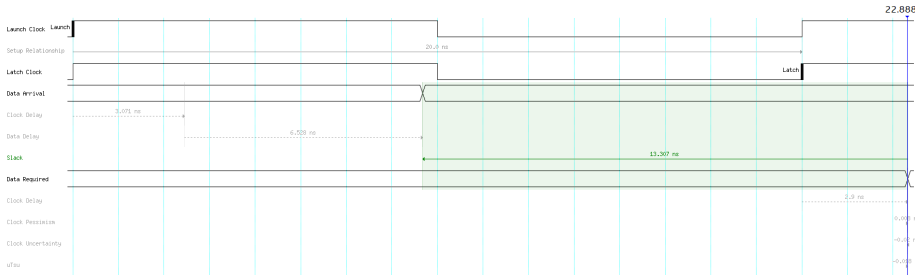

Figure 1: Worst Case Path

1.2 Max Throughput

We are using the following equation to calculate Throughput, where $data_{period}$ represents Data Per Clock Period and $freq$ represents Clock Frequency.

$$throughput = data_{period} * freq \tag{2}$$

According to the equation above, with Max Frequency as 149.41 MHz and data per clock period as 1 bits, we have

$Throughput = 1bits * 149.41MHz = 0.15bits/ns = 150MB/sec.$

According to the calculation above, the Max Throughput is estimated to be 150 MB/sec.

1.3 Modification

The initial design includes redundant nodes, which yields a Worst Case Slack as 8.079 ns and Max Frequency as 83.89 MHz. An example of the redundant nodes is show in the picture below where the declaration of `remap_in` is not necessary.

```
wire [6143:0] remap_in,
assign remap_in = reg_buf_out;
coder_interleaver ci_inst(
                        .cin(remap_in),
                        .K_eq_6144(k_size_6144),
                        .cout(remap_out),
                        .ciout(ci_array)
                        );
```

Figure 2: Redundant Node

After removing the redundant wires, the Worst Case Slack is 13.307 ns and the Max Frequency increases to 149.41 MHz. We find huge time improvement after removing the redundant nodes which is very surprising that wiring between two nodes may take such a huge amount of extra time. The trade-off here is between the easiness in understanding the code and the time performance of the design.

1.3.1 Worst Path Comparison

The worst slowest path before node reduction is 11.216 ns, and after the node reduction is 5.89 ns. We see huge timing improvements.

- Before Wire Reduction:
  `shiftreg_buf:input_shiftreg_inst1|mem[4653] : (0.232 ns)`
  `input_shiftreg_inst1|mem[4653]|q : (0 ns)`
  `comb_6149|Q[4653]|asdata : (11.216 ns)`
  `reg6144:comb_6149|Q[4653] : (0.406 ns)`

- After Wire Reduction:
  `shiftreg_buf:input_shiftreg_inst1|mem[1464] : (0.232 ns)`
  `input_shiftreg_inst1|mem[1464]|q : (0 ns)`
  `reg_inst|Q[1464]|asdata : (5.89 ns)`
  `reg6144:comb_6149|Q[1464] : (0.406 ns)`

1.3.2 All Clock Histogram Comparison

According to the All Clock Histogram shown below, most of the slack are between 16 ns and 19 ns which doesn't change a lot before and after node reduction. However a samll amount of slack less than 13 ns are improved because of the

node reduction. We can see that the node reduction greatly improved the worst slacks while doesn't have big influence on other slacks.

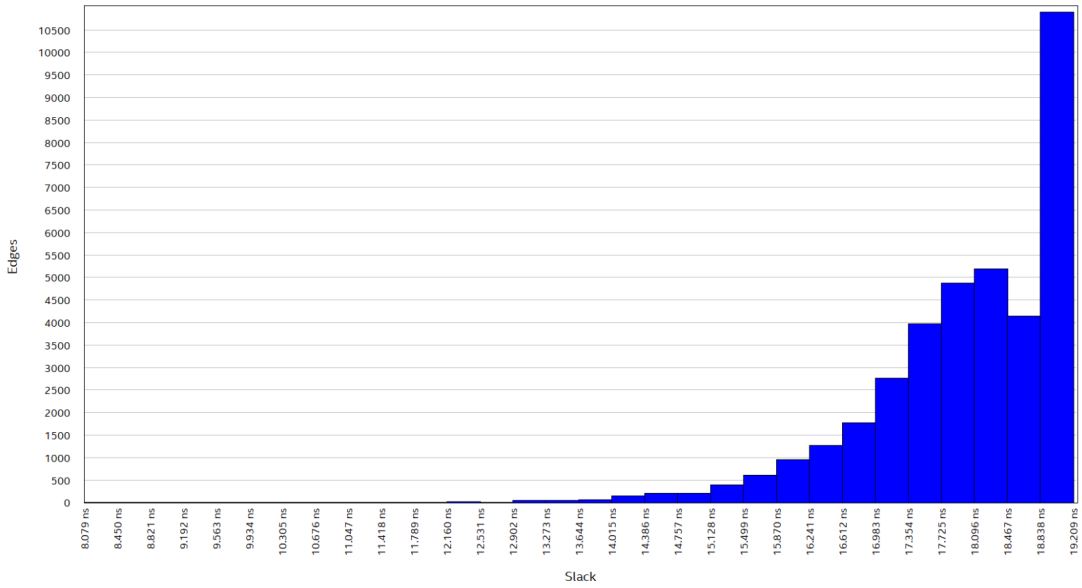

Figure 3: All Clock Histogram before node reduction

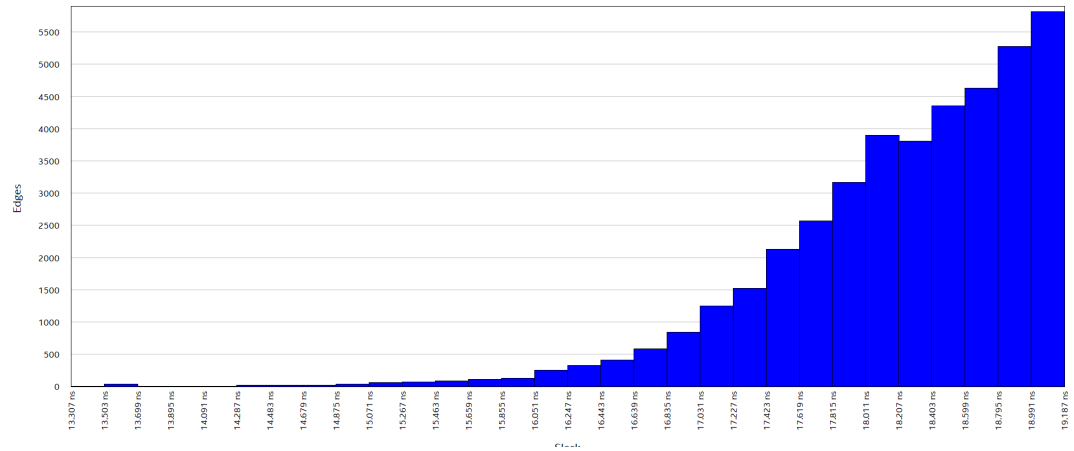

Figure 4: All Clock Histogram after node reduction

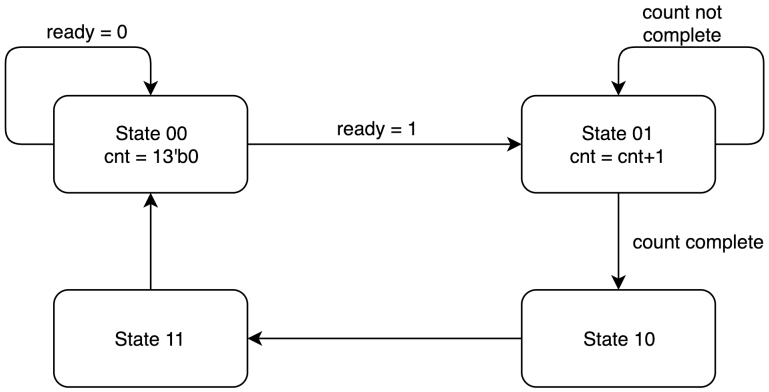

Figure 5: All Clock Histogram after node reduction

2 FSM

The FSM in our design is implemented in the index generating module. This FSM contains four states. The state in this FSM doesn't make output, but manipulate the 13-bit register `cnt`, which relates to the output of the module. The function of each state is listed below.

- State00 is the waiting state. It waits for a `ready` signal to be pulled high to go to State01. If the `ready` signal stays low, State00 keeps going back to itself. At State00, `cnt` is set to 0.
- State01 is the counting state. This state goes to State10 is the counting is complete, otherwise it goes to itself. The counting is considered complete if either the `kout` signal is 0 and `cnt` is 1055 or the `kout` signal is 1 and `cnt` is 6143. At State01, `cnt` is incremented by 1.
- State10 is a reserved state to accommodate with any signal indicating data readiness required in the future.
- State11 is a reserved state to accommodate with any signal indicating data readiness required in the future.