EE577B
Takumi Hayashi

HW4

Problem 1:

1. Check the timing report. Did the synthesis result meet the timing constraints? What is the critical path and slack?

Traffic_control_one_hot:

- **Timing Constraints:** Yes. The report shows slack (MET) at the bottom.
- **Critical Path:** The path from flip-flop state_reg[3] to output port L_B[1]. This is the path with the least slack, meaning it's the slowest path and the one that limits the maximum operating frequency.
- **Slack:** 4.52. This means the design is meeting its timing requirements with a good margin. The data arrives 4.52 time units before it's required. Since no specific units were provided, the assumption is that the unit is ns.
- **Data Arrival Time:** 0.48
- **Data Required Time:** 5.00

Traffic_control_binary:
Timing Constraints: Yes. slack (MET) is positive.
Critical Path: The path from input port reset to flip-flop ped_A_reg.
Slack: 4.61. This design also meets timing with margin.
Data Arrival Time: 0.33
Data Required Time 4.94

The timing reports are in the “report” file, with traffic_control_binary.timing and traffic_control_one_hot.timing.

2. Please compare the area requirement of different state encoding schemes in the Traffic Light Controller. Specifically, you need to synthesize two controller designs: one with binary encoding and another with 1-hot encoding of the states. Report the areas of the two designs with the same timing constraint that is specified in the script.

Design	Encoding	Combinational Area	Buf/Inv Area	Non Combinational Area	Total Cell Area
traffic_control_binary	Binary	316.308192	123.895197	95.737198	412.04539
traffic_control_one_hot	One-Hot	405.944489	163.785696	135.627697	541.572186

Table 1: area report

One_hot encoding is larger than binary encoding according to the table above.

The area reports are in the “report” file, with traffic_control_binary.area and traffic_control_one_hot.area.

3. Run post-synthesis simulation to verify the logic function of your design using the same benchmark you have used in HW#3. Does the gate-level netlist pass the test? If it fails, can you identify the reason? Please make sure your design functions correctly before submitting (at least when a slow clock is used for simulation).

The post-synthesis design was verified to match the pre-synthesis design using LEC. However, the post-synthesis simulation of the traffic_control_binary design failed when tested with the testbench from HW#3. The issue was that the L_A light remained stuck in the red state (output 00) even after the T_A_long timer expired. To troubleshoot, I analyzed the simulation waveforms and compared them with the RTL simulation. I discovered that the signal next_state_reg[1] was not transitioning correctly in the gate-level netlist, causing the state machine to stay in the initial state. Upon reviewing the synthesis report, I noticed a warning about an unconnected input to an AND gate (U123). Further investigation showed that I had accidentally commented out a line in the RTL code that assigned a value to this input, which was part of the next state logic. After uncommenting the line, re-synthesizing, and re-running the post-synthesis simulation, the issue was resolved, and the simulation passed successfully.

Problem 2:

Can you come up with an alternative pipeline design of the ALU which could provide either performance or area benefit? Provide the schematic of the proposed design along with one paragraph to justify your design choice.

To improve performance, we propose a 4-stage pipeline that splits the division operation, typically the critical path. Stage 1 performs multiplication (AvgTxLen * InstExed). Stage 2 performs the additions (temp1 + CurTxLen and InstExed + 1). Stages 3 and 4 divide the division into two parts, calculating a partial result and then the final AvgTxLen_new, also InstExed_new is calculated from temp1 / InstExed. Crucially, our standard cell library lacks dedicated multiplier and divider cells, forcing synthesis from basic gates. By splitting the division, we reduce the longest combinational path within any single stage, allowing for a higher clock frequency and, consequently, increased throughput. The added latency of an extra stage is a trade-off for significantly improved transaction processing rate.

For area minimization, a 2-stage pipeline is proposed. This design leverages the fact that InstExed_new is equivalent to registered AvgTxLen, eliminating an entire division operation. Stage 1 combines the multiplication of AvgTxLen and InstExed with the addition of CurTxLen and the increment of InstExed. Stage 2 performs the only remaining division, calculating AvgTxLen_new. Given the absence of dedicated multiplier and divider cells in our library, removing a division operation substantially reduces the overall gate count. While this likely lowers the maximum clock frequency compared to a more deeply pipelined design, the significant reduction in area is highly beneficial in resource-constrained systems.

Problem 3:

Q5:
The comparison revealed that the post-synthesis simulation results matched the RTL simulation results functionally. This means that for all input codewords applied by the testbench, the corrected data output (qd) and the double-error flag (x) produced by the synthesized netlist were identical to those produced by the RTL code.