Experiment 8
Greatest Common Divisor using the Krypton CPLD

Dhruv Ilesh Shah — 150070016

April 8, 2017

Overview

In multiple cases, constructing a pure FSM-based sequential machine can be redundant with respect to the number of states, and hence complicated to analyse. An alternate approach can be RTL design. In this report, I present the complete implementation of a circuit that computes the *Greatest Common Divisor* or sixteen 16-bit numbers.

The code was compiled on Quartus Prime, and simulated using ModelSim. GHDL was also used for simulation purposes, at a low level. This was then uploaded to the *Krypton v1.1* 5M1270ZT144C5 CPLD-based board.

The codes and setup have been covered in section 1. We build the string recognizer piece-wise, by implementing each module independently. The VHDL codes have been kept modular and as generic as possible, for reusability and code clarity. Section 2 presents the simulation observations and miscellaneous results. Section 3 presents the observations after running the scan-chain test on the board.

1 Setup

The concept of RTL design is governed, as the name suggests, by implementing *registers* & *transfers*. A simple way to look at it would be that we cut down the number of states, and compensate the same by storing data in registers, which themselves store "state" information.

2 Observations

3 Scan-Chain Tests

We have tested the logic using the RTL simulations, emulated the CPLD performance using the gate-level simulation and uploaded the code on the Krypton board. Next, we need to check that the code is actually running as it is expected to, on the board. We could do so manually but that is not feasible due to the following reasons.

- Our current circuit requires 16 inputs and an additional 4 control switches, including a clock. In any given setup, it *may* not be possible to allocate as many I/O pins. As the complexity increases, it will indeed not be possible to allocate so many pins.
- Even if the above is possible, the total number of test cases is *exponential* in the size of the input and it is impractical to perform each of this manually.[1]

[1] It is not wise to skip any case because, say, we do miss out a failed case it can cascade into unimaginable consequences, which can become difficult to debug.

Hence, we test the uploaded code on the hardware using the scan-chain setup, as suggested in the manual. This setup was run on a set of two collections of text which has occurrences of the concerned string.

Results

Conclusion

Starting from the very scratch, in this report, I have presented the logic and code for a sequential implementation of a string recognizer. The logic was tested using RTL simulation, followed by the gate-level simulation for delay analysis and emulating the CPLD. This was followed by an actual rigorous test on the CPLD board after burning the code on it, using the *TIVA-C* microcontroller.

All the cases passed successfully at all stages and hence the complete string recognizer can be used in hardware, as required.