**Project 2**

Alex Beaulier

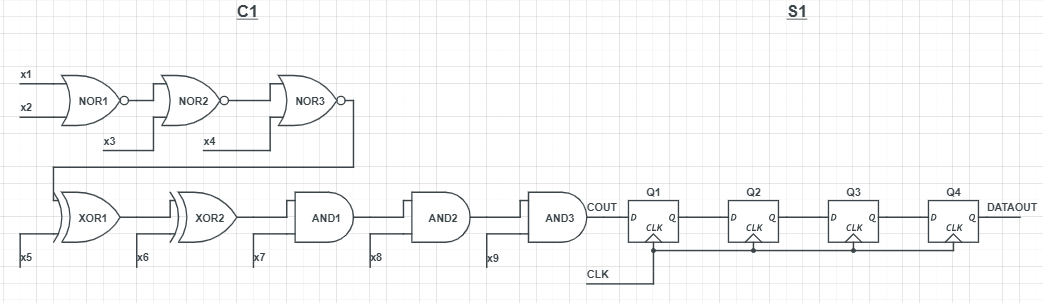
**Problem 1:**

Design a sequential circuit S1 with the following requirements

1. The combinational logic part C1 for the next state functions should contain at least 3 NOR gates, 2 XOR gates and 2 AND gates.
2. The number of the flip-flops should be at least 4.
3. The sequential circuit should have at least an output.
4. Make sure that your design be different from those used by other students. If your design is identical to a design used by other students, you should revise your design and redo the work.

**Tasks:**

1. Draw your sequential circuit S1. Figures should be plotted by using a professional software tool. Scanned or cut/paste hand drawn figures and writing are not acceptable.



1. Derive the next state functions for S1.

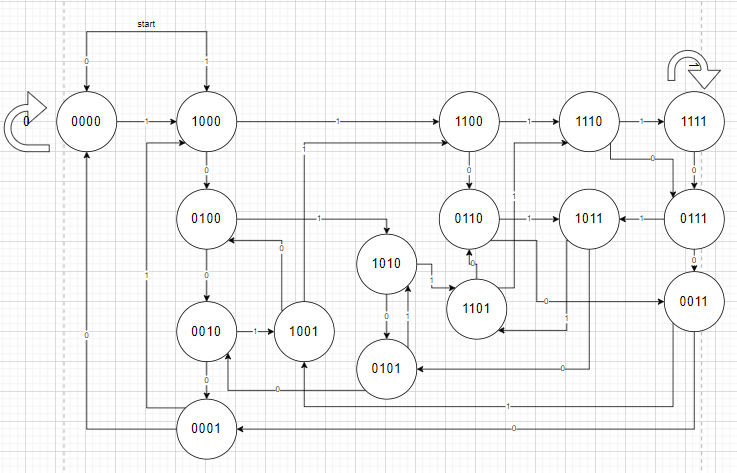
Q1out = f(Cout)

Q2out = f(Q1out)

Q3out = f(Q2out)

Q4out(DATAOUT) = f(Q3out)

1. Draw the state transition graph of S1 including all the POSSIBLE states.



1. Write an SV code for S1 using its next state functions. The solution should Not be the implementation using the FF instantiation in a structural model. Simulate the code using an HDL simulator.

**See Attached**

**Problem 2:**

Design a sequencer detector. It compares the input sequence with its own built-in sequence bit by bit. If the input sequence is identical to the built-in sequence, it will display a message "matched", otherwise it will display a message "not-matched". When the mismatched bit occurs, the detector will return to the initial state and process the next input bit as the beginning of a new sequence. Your built-in sequence is an 8-bit BCD code created by converting the last two digits of the PSU ID number of one member of your group. Write an SV code for implementing the detector. Simulate the code using an HDL simulator. 20 = [0010\_0000]

**Attached**

**Problem 3. (Design Compiler)**

Study the **DC\_tutorial** package (next to this project at D2L) and follow the examples step-by-step to generate the netlist file and the timing constraint spec file. Tcl scripts are suggested to use in DC and Tcl templates are provided in the package.

* You will need to connect to the RedHat system remotely: **mo.ece.pdx.edu**. Please check:

https://cat.pdx.edu/services/network/vpn-services/ to install PSU VPN on your system. Then install a remote access software support x-server such as MobaXterm.

* Remote Login to mo.ece.pdx.edu with your ECE username and password.
* Read the tutorial in dc\_task1 to set up DC. Please read the tutorial first and install DC in your UNIX account. If you have any questions about the installation process, please email your problems to the CAT team: support@cat.pdx.edu.

**Attached output and tcl file.**

**Problem 4: (Future of HDLs?)**

Have you heard of writing a hardware model in software programming languages? Please follow the links on page 50 of the ASIC lecture and study the related documents. Write a report of at least 3 pages to address the following points.

**ATTACHED BELOW**

1. Find at least 3 hardware coding examples in both Chisel and a Python-base language.
2. What are the motives and advantages of such an effort? Why important?
3. If such an effort continues with the successful tool supports, what is the future role of the Systemverilog RTL modeling
4. (Optional) Why can hardware be modeled in a software language? What construct is essential for hardware modeling?

Hardware modeling is a complicated tool in an effort to generate modern electronics. Hardware has typically been generated by software that includes a synthesizer and simulator. Hardware can be modeled due to software being able to emulate real constructs such as a time domain and topics such as sequential and parallel or combinational logic. Concurrency and other elements can be simulated and therefore the hardware can be emulated through the language. The role of the synthesizer is to transform high level constructs to low level logical constructs to literally be modeled in the form of transistors or other hardware components of FPGA and ASICS. The role of the simulator is to run this synthesized logic for verification and testing. A new modern developed approach for hardware development has been through support in Python and Chisel.

Chisel is a hardware design language for generating and reusing ASIC and FPGA logic designs. It adds toolsets to Scala and creates complex, parametrized circuits. The output of Chisel is synthesizable Verilog. This tool provides abstraction while designing digital logic and is referred to as flexible intermediate representation for RTL (FIRRTL). Four notable Chisel projects are Rocket Chip Generator, Berkeley Out-of-order Machine and RISC-V Mini.

The Rocket Chip Generator can instantiate RISC-V Rocket core which encompasses tools to generate and test SOC designs. The code can also generate RTL. Hardware generation uses Chisel to utilize Scala and produce RTL for a fully complete SOC. Other noteable libraries within this module include hardfloat for generating floating point units with precision to IEEE standard 754-2008. Rocket tools is also a version of the RISC-V software tools to be compatible with RTL. Torture library generates and executes random instruction streams for stress-testing the core and uncore. This package optimizes Verilog for FPGA and VLSI. The emulator is also called a Verilator and Rocket Chip is high-performance and cycle accurate. This code also allows the user to use multiple cores specific to their host machine.

RISC-V Mini is a 3-stage pipeline that was written in Chisel. It uses RV32 ISA and contains simple instructions and data caches. The goal of RISCV-Mini is an intermediate step before Rocket-Chip.

OpenSoC Fabric is an on-chip generator for network infrastructure which can be parameterizable. This is to develop powerful, high performance computing architectures based on SOC technology. Utilizing Chisel, Scala and DSL, it generates C++ and Verilog. OpenSoC utilizes collections of configurations, is object-oriented and has various functionality as extensions.

Another project is a Fast Fourier Transform accelerator generator in Chisel. FFT’s are useful across mobile communications and radio astronomy so the Berkeley design team along with DARPA developed this FFT also for hardware acceleration. The group implemented divide and conquer algorithms for developing large FFTs from smaller FFTs. The team used Cooley-Tukey decomposition to produce FFT accelerators for Rocket Chip systems. The advantage of Chisel expressed by the paper is that seamlessly the group could have high level programming features for concise and easier to express hardware. Chisel does encourage conciseness at the risk of safety and the tool hasn’t developed specific error messages. This leads to difficult debugging. Chisel doesn’t have support for test coverage analysis. Fixed floating point numbers were also difficult to implement properly. Bundles will also yield no errors at compilation and can make debugging extremely difficult such as for a simple error of a wire connecting to a whole packet rather than a containing packet.

Python is an interpreted software language that encompasses the idea of being open source and flexible code. Python is abstract but simple to use and combine for modularity. This allows Python to be used as a simple tool for a variety of projects and tasks.

The first example of using Python is PyRTL. This has allowed Python to also develop RTL design, simulation, tracing and testing. The goal of Python within hardware design is simplicity, usability, clarity and extensibility than performance or optimization.

The second example of using Python is PyLSE. PyLSE is a pulse transfer level design, simulation and verification for superconductor electronics. In the examples, a full adder is developed and executed in a simulation.

The third example of using Python is OpenTPU. OpenTPU uses Google’s version of tensor processing units for neural network computations. It’s a custom ASIC for acceleration. This project is still lacking in Convolution, pooling and programmable normalization. This also uses additional Python libraries that contain wrappers in C for faster computation such as numpy.

The reason for developing these tools are to simplify design and reduce cost as well as to generate complex tasks faster. The motives are cost and business. Eliminating manpower and simplifying complicated tools for ease of use are essential for business. This eliminates headcount, time to market and saves cost for production. Businesses want to have the best developed product first ahead of their competitors. The reuse of a modular design is also essential as many companies save on cost by reusing existing designs with minor tweaks for the customer. Modularity by the tool eliminates having large teams to build projects from the ground up.

The future role of SystemVerilog would only be the translated output from the modular programming. Potentially it could also be eliminated if a tool could develop hardware instead of using Verilog. Phd Specialists may be the only field left who would look over modular code and verify the translation was correct for minor issues.

Current discussions with Chisel and this development are that Verilog is dominant in industry and has simulation support. Chisel also doesn’t support constructs that Verilog does such as analog elements. Chisel does however enable machine learning libraries with Python whereas trying to develop tensorflow in C would be extremely complicated due to abstraction that isn’t readily available by this hard typed language rather than a scripting language like Python. The comparison to Verilog can’t easily be assessed by small coding examples. Chisel is a powerful language for describing circuits. Hardware engineers have typically been proficient with C but have not had the depth of object oriented programming or functional programming that software engineering principles rely on which is where Chisel’s introduction is a reason it’s value is enabling more than Verilog traditionally supported.