CSEE W4823x **Homework 2 *(corrected)*** Handout 11
Prof. Steven Nowick September 27, 2016

This homework is due **at the beginning of class** on Tuesday, October 11. Because it is larger than some of the other homeworks, it will be worth *7%* of your final grade.

Note: A correct answer without adequate explanation or derivation will have points deducted. To get full credit, (a) write legibly/type, and (b) show all work (label relevant items, show deriviations, include explanations).

Solo vs. Group Work: For *Part I,* only solo work is allowed; each student must hand in their own assignment. However, for *Part II,* you can work either solo or in a group of two. If you choose to work in a group, hand in one copy of the solution and clearly indicate on it the names of everyone in your group. Everyone in the group will receive the same grade.

Part I (solo work only)

1. (8 points) **State Minimization: Completely-Specified FSMs.**
   In this problem, you will use a *modified version* of the symbolic state table in B/V (*3rd edition only*), fig. 8.92 (p. 568) as your starting point.

   Make the following 3 modifications to the table: (i) in present state B, with input $w = 0$, the next-state is A; (ii) in present state D, with input $w = 0$, the next-state is A; (ii) in present state D, the output z is 1.

   Do state minimization using the *implication chart method* as presented in Handout #10 on this modified state table: (a) write the initial unminimized symbolic state table; (b) write an initial implication chart; (c) iterate through the chart to form a final implication chart; (d) indicate the final merged states; (e) draw the final minimized symbolic state table.

2. (7 points) **Xilinx XC4000-Series FPGA Structures.**
   (a) Solve problem 9.34 at end of Handout #12 (FPGA handout, released shortly).
   Note: you do not need a formal proof, simply a direct and precise procedure for mapping any 5-bit Boolean function into the CLB of Figure 9-45. Clearly indicate relevant inputs to blocks F, G and H, and any relevant MUX settings. Show all work, and give a clear written explanation.

   (b) Solve problem 9.35 at end of Handout #12 (FPGA handout). Follow the same "Note" above. Show all work, and give a clear written explanation.

Part II (group or solo work)

3. (20 points) **Word Problem: FSM Specification for DNA Sequencing**
   For this problem, you will design an FSM for a *pattern detector for DNA sequencing*. This problem covers developing only the specification (i.e. symbolic state diagram) for the detector. Later, in "CAD Tool Problem #2", you will focus on synthesizing, modelling and simulating your design.

   Setup/Assumptions. The setup is similar to that of problem #5 in HW #1, and is repeated below. However, the problem is different.

   The problem of detecting patterns or "sequencing" is critical in modern genomics. A DNA sequence consist of patterns of 4 distinct symbols or *bases:* A, C, G, and T. These symbols are ordered in DNA into large and complex sequences, which from our viewpoint can simply be regarded as a linear "string" which mixes the 4 symbols in some order.

   For this problem, you are to assume that the 4 symbols have already been translated into a binary 2-bit code, as follows: $A = 00$, $C = 01$, $G = 10$, and $T = 11$. You can also assume that an FSM detector receives 1 symbol per clock cycle, i.e. on 2 input wires $x1$ $x0$. The detector has a single output, z.

In this problem, you will design a pattern detector to detect the presence of *two particular subsequences* embedded in the DNA sequence. Whenever either of these complete subsequences is detected, the FSM will assert its z output as 1 for one clock cycle.

Note: For purposes of this assignment, your designs should allow the possibility not only of an isolated instance of the subsequence occurring in the DNA, but also two or more subsequences back-to-back as well as mutually-overlapping subsequences.

Subsequence #1: DNA Sequence Detector for Starting Transcription: TATAAA.
The subsequence $TATAAA$ is the code for starting transcription by RNA polymerase.

Subsequence #2: DNA Sequence Detector for Starting Protein Translation: ATG.
The subsequence ATG is the code for starting translation of the DNA sequence into a protein.

Problem Summary:
In this problem, you are to write a single Moore specification (i.e. state diagram) that detects both of the above subsequences. *Do not implement this specification, just write the symbolic state diagram!*

Your specifications will have as input 1 symbol, encoded digitally by *2 input wires* ($x1$ $x0$) and *1 output wire* (z). The machine will process each element in serial order of a DNA sequence consisting only of symbols A, C, G, and T, with 1 input symbol received per clock cycle. Each time either of the two subsequences is received, the output is asserted high (i.e. 1) for one clock cycle, on the last symbol of the subsequence. At all other times, the output is deasserted low (i.e. 0). See "Setup/Assumptions" above for further assumptions.

What to Do: Draw the corresponding *Moore* state diagram.

Note: Your specification should be neatly drawn, with all states clearly marked. All 4 possible input transitions in each state must be explicitly labelled: do not omit any input combinations. For good documentation, *a simple text label should indicate the purpose of each state,* either placed next to the state or in a readable separate key (i.e. table).

For full credit, your state diagram should have close to the minimum number of states. A small increase over minimum is fine, you do not *need to perform a state minimization procedure. However, unduly large state diagrams will receive a moderate penalty.*

4. (5 points) **CAD Tool Problem #1: An Introduction to VHDL, Schematic Capture, Functional Simulation, and the Altera Quartus II CAD Tool Environment.**

   In this problem, you will learn to use the Altera Quartus II Design Environment, to specify and simulate a small design. (Note: you don't need to know almost any VHDL to do this basic tutorial!) I suggest you re-read B/V 2.9 on CAD Tools, and B/V 2.10 on an introduction to VHDL. This latter includes the target example of this problem. The other assigned sections, such as 4.12 and 5.5.1-5.5.3, are also very relevant.

   Getting started: Details on access to the Quartus/Altera CAD tool software are provided in **the top-level class web page, "Quartus Usage and Access".** If you do not have the right Windows platform, or want more flexibility, you also can use the 12th floor Mudd embedded systems lab, with direct use of the Linux machines there (the Quartus II software has been installed) or remote access, as well as download to your laptop.

   You should all now have swipe access to the lab, as well as an account on the Embedded Lab machines. If you do not, contact the instructor or CA's.

   How long will this problem take?: Should not be much more than 3-5 hours.

   What does it cover?: It covers how to set up and run the tool environment, and then, to enter a specification in **two ways:** (i) via "schematic capture", and (ii) via VHDL specification.

   Where is the tutorial?: You will be following B/V Appendix B, sections B.1-B.4.

What to do?:

(a) *Getting Started:* Follow B/V Appendix B.1.

(b) *Starting a New Project:* Follow B/V Appendix B.2.

(c) *Design Entry Using Schematic Capture:* Follow B/V Appendix B.3.

For your simulation, follow the directions to use *all 8* possible input vectors, and set the functional simulation timing as specified in the book.

Note: Section B.3.2 is a brief digression on how to *synthesize* a circuit, starting from your schematically entered initial design. This will not be the focus of this course, but is good to go over and understand.

(d) *Design Entry Using VHDL:* Follow B/V Appendix B.4.

For your simulation, follow the directions to use *all 8* possible input vectors, and set the functional simulation timing as specified in the book.

Note: Section B.4.3 is likewise a brief digression on how to *synthesize* a circuit, starting from your VHDL source code. This will not be the focus of this course, but is good to go over and understand.

What to Hand In: For (c), print out and hand in the schematic of your entered circuit, as well as the waveform that results from your functional simulation (similar to Fig. B.25). For (d), print out and hand in your VHDL code, as well as the waveform that results from your functional simulation (similar to Fig. B.25).

5. (60 points) **CAD Tool Problem #2: Designing an Iterative DNA Sequence Detector.** This problem is a followup to problem #3 above, where you specified and designed a Moore version of an FSM to detect two different DNA sequences: TATAAA and ATG.

   In this problem, you will first convert the Moore FSM design to an *iterative circuit cell,* and optimize the leftmost and rightmost instances (i.e. end cells). Then, you will model each of the cells (both regular and optimized) *using dataflow VHDL,* assemble a collection into a 20-symbol iterative pattern detector *using structural VHDL,* and then finally simulate the cells and entire detector using Altera/Quartus.

   Modified Moore Iterative Circuit Structure. In Handout #9 and class lecture, we have covered how to transform a Moore FSM into an iterative circuit. The block structure of the iterative circuit is shown in Fig. 17-4 of Handout #9: it consists of N replicated cells (none of which has an output block, just next-state block) followed by a *single output block* at the right in stage N+1. Such an iterative circuit only gives a single final output, at the right of the network, after all inputs have been processed.

   In contrast, for this problem, you will create a *modified Moore iterative circuit structure,* where outputs are produced by *each cell*. The purpose is to allow results to be indicated at each point in a given input sequence, rather than only at the end of the sequence. The modification to the iterative circuit is simple: While the old Moore-style iterative circuit (Fig. 17-4, Handout #9) only has an output block in cell N+1, your new Moore-style iterative circuit will have output blocks in cells 1 through N+1.

   In summary, the output block of cell 1 appears in cell 2, the output block of cell 2 appears in cell 3, etc. Finally, the output block of cell N-1 appears in cell N, and (similar to Fig. 17-4 of Handout #9) the output block of cell N appears in cell N+1. (Cell N+1 has no next-stage block.) The above modification of the Moore-based iterative circuit allows nearly every individual cell to produce an output, thereby producing intermediate information about the sequence, rather than only a single final answer.

   Note that, because this is a Moore-based iterative circuit, if an identified pattern ends in cell K, the circuit will assert its output bit "off-by-1": *in cell K+1,* i.e. in the following cell after the end of the pattern.

What to do:

(a) *Design the iterative Moore-based pattern detector for the DNA sequence detector.* Refer to problem #3 above, and your Moore state diagram specification, as well as the above important note on how to construct a "modified" Moore-based iterative circuit. Follow the steps presented in class, in the Roth iterative handout and class lectures. The result should be (i) the complete gate-level design of a *regular* iterative cell, as well as (ii) the complete designs of any *optimized* cells (leftmost cells; rightmost cells). *Show all work and clearly label each step you perform.*

State Assignment Guidelines: As much as possible, use a Gray code for state assignment. The start state $S0$ should be assigned the all-0 code. If states form a linear sequence away from the start state, they should follow the Gray code sequence. If they deviate from a linear sequence, try to make your assignment as close as possible to Gray code. *These are very informal guidelines!:* your machine specification is likely to have a complex structure, and not fit neatly into a linear sequence of states.

Karnaugh Map Guidelines: If you need a 5- or 6-input (or higher) Karnaugh map, follow examples in B/V, as well as Roth (on reserve), on how to write and work with these larger K-maps.

(b) *VHDL modelling of basic "regular" iterative cell.*
You will model a regular iterative cell in VHDL. The combinational cell should be modelled using a **dataflow style,** and not in a structural or behavioral style. That is, using dataflow VHDL, model the outputs as a function of the inputs (do not model individual structural gates with the cells!) The regular cell, for the modified Moore-based iterative circuit, has both a next-state block (for current cell) and primary output block (for previous cell). See discussion above.

Hand in a printout of the file containing your VHDL code, describing both the entity and architecture of the regular cell.

(c) *VHDL modelling of "optimized" left cell(s).*
Following the presentation in class and in the Roth handout, you are now to create models for optimized left cells. In part(a) above, you were asked to determine if the leftmost cell could be optimized, and if the second-to-left cell could be optimized, etc. Depending on your conclusions, you may have one or more optimized cells, for the left portion of the iterative circuit.

For each of your optimized cells, follow the same guidelines as in part(b). Your VHDL models should only use **dataflow style**.

Hand in a printout of the file containing your VHDL code, describing both the entity and architecture of each optimized left cell.

(d) *VHDL modelling of "optimized" right cell(s).*
Following the presentation in class and in the Roth handout, you are now to create models for optimized right cells. In part(a) above, you were asked to determine if the rightmost cells could be optimized. Depending on your conclusions, you may have one or more optimized cell designs: optimized rightmost cell, optimized second-to-right cell, etc.

For each of your optimized cells, follow the same guidelines as in parts (b) and (c). Your VHDL models should only use **dataflow style**.

Hand in a printout of the file containing your VHDL code, describing both the entity and architecture of each optimized left cell.

(e) *VHDL modelling of a complete 20-symbol pattern detector.*
Create a model for the entire 20-symbol pattern detector. You should use **structural style**, *not* a dataflow view, for your 20-symbol detector. The structural model should in turn use the models for your basic cells from parts (b) to (d) above. The resulting design should have 20 concatenated cells. For simplicity, make the component declaration for the above cells *local* to the VHDL architecture for the 20-symbol pattern detector (following the style presented in B/V Fig. 5.23).

Hand in a printout of the file containing your VHDL code, describing both the entity and architecture.

(f) *Verifying the basic cells.*

You are now to verify the correctness of the basic VHDL cell models that you created in parts (b)-(d), by providing a limited set of test vectors and running them through simulation using the Altera simulator.

Follow the general flow presented in the tutorial (problem #4(d) and B/V Appendix B.3.3), for Design Entry Using VHDL, followed by simulation. You will apply a new input pattern every 50 ns using the Altera waveform generator. Create the input vectors as detailed below and run each functional simulation.

For each case, exercise **all possible legal input combinations** to each cell. An "input combination" will be a vector which sets both the present state bits as well as the two primary input bits (i.e. x_1, x_0). The simulation will examine the values of the "output bits" of the cell, which include the two next state bits and one primary output bit (i.e. z_i).

Note: remember that several input combinations may be impossible, because not all state codes are used! You don't need to simulate unused combinations.

What to Do: Run all possible legal input combinations for each of your cells of parts (b)-(d). Also, create a text file listing each input vector applied for each cell, and the corresponding expected correct output results. One text file is appropriate for all runs. After creating the above vectors, run a complete simulation for each of your cells. So, for each cell, you should have a complete simulation exercising all possible input combinations. Repeat this process for each of your cells in (b)-(d) For each simulation, observe the outputs in the waveform window and make sure the output waveforms are as you expected (this method becomes impractical for larger designs).

Hand in printouts of the text file and of the output waveforms from the simulations.

(g) *Verifying the 20-symbol pattern detector.*

You are now to verify the correctness of the entire 20-symbol DNA pattern detector that you created in part (e), by providing a limited set of test vectors and running them through simulation using the Altera simulator.

Again, follow the general flow presented in the tutorial (problem #4(d) and B/V Appendix B.3.3), for Design Entry Using VHDL, followed by simulation. You will apply a new input pattern every 10 ns using the Altera waveform generator. Create the input vectors as detailed below and run each functional simulation.

(i) Create a simple input vector which detects the two distinct subsequences, ATG and TATAAA, when separated from each other.

(ii) Create an input vector which contains neither of the two subsequences.

(iii) Create an input vector with a complicated mix of several partial (but incomplete) subsequence patterns.

(iv) Create an input vector with several adjacent ATG patterns, with no separation by other symbols, and with one instance of mutually-overlapping adjacent TATAAA patterns.

(v) Create an input vector of your choice to capture other interesting cases.

What to Do: After creating the above vectors, run the 5 simulations. For each simulation, observe the outputs in the waveform window and make sure the output waveforms are as you expected (this method becomes impractical for larger designs). Also, create a text file listing each input vector (for parts (i)-(v) above) and the corresponding detection results. One text file is appropriate for all runs.

Hand in printouts of the text file and of the output waveforms from the five simulations.