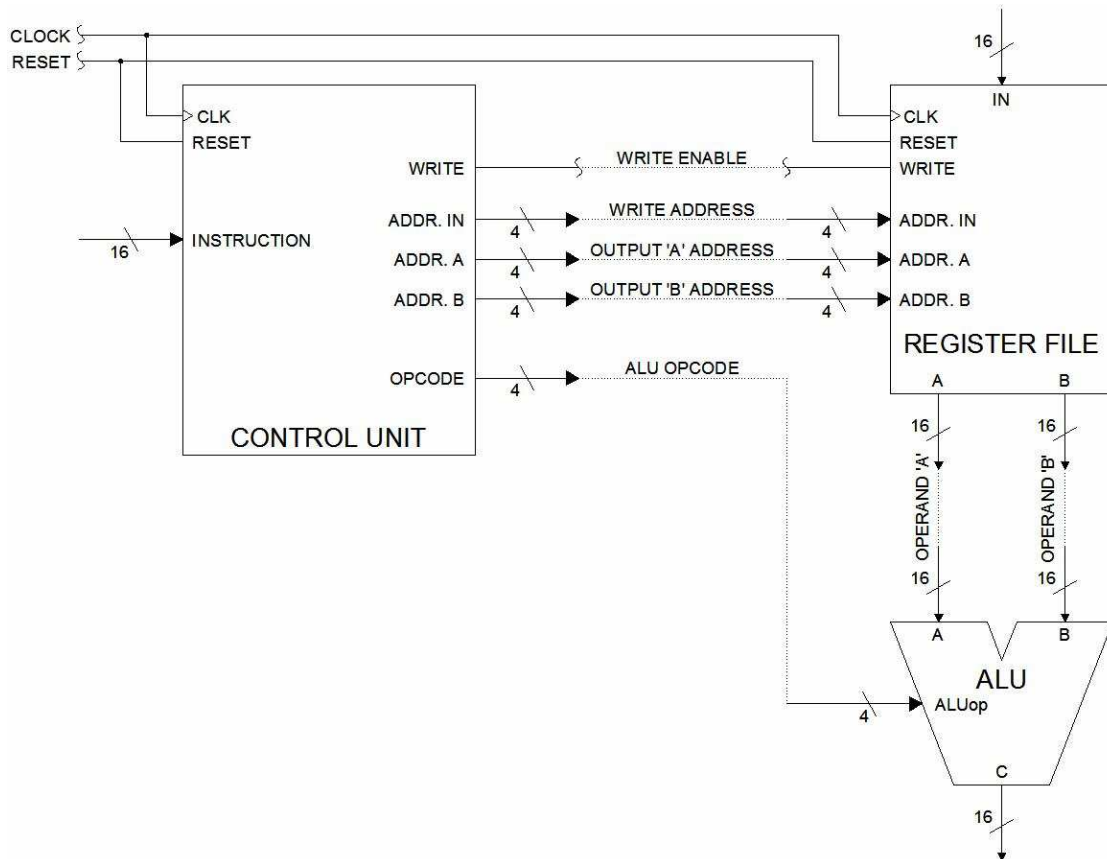


ECE 436 Laboratory 2

INTERNAL DATA MOVEMENT AND CONTROLLER DESIGN

Description:

For this laboratory, you will be designing two new components, a register file and a control block. These two components will be integrated with your 16-bit ALU (increase the size of your ALU from Lab 1 from 4-bit to 16-bit) to form the beginnings of a processor. The register file will consist of (at least) 12 16-bit registers (increase the size of your register from Lab 1). You will need multiplexing and decoding logic so each of these registers can be independently read from or written to using 4-bit address lines. The control block will interface the register file and the ALU, making it possible to perform any one of the first seven ISA instructions. The diagram below illustrates the necessary control lines and modules that will be needed for this assignment. (Note that this schematic does not include the handling of multiple-bit shifts, but your design must. Again, you may use either a barrel-shifter or a single-bit shifter (including a controller that manages the number of shifts) with a register feedback.) Other than that, the Control Unit is really just a bus splitter, directing the appropriate field of the instruction to the appropriate component inputs.



In addition, you will become familiar with the state machine tools that are part of the Xilinx tool set by performing the StateCAD tutorial at the end of this document. These tools will be necessary when designing the finite state machine for your processor's controller.

Process:

As illustrated above, the register file will have two 16-bit outputs, each capable of outputting data from any one of 12 registers that are selected from the "Output 'A' Address" or "Output 'B' Address". The "Write Address" selects which of the 12 registers is being

written to when the "Write Enable" line is high. The control block, designated "Control Unit" in the diagram, breaks apart the instruction and sends the 4-bit destination and source operands to the proper address lines on the register file. It also controls the "Write Enable" line and takes the opcode and drives the appropriate bit-string into the ALU's "4-bit Select" line. (As stated above, there are multiple ways that you can handle the multi-bit shifts.)

You must also figure out how to load values into the register file that you can use to test your design. Hint: use a control signal (controlling a MUX) that puts the system into either a data load mode or a normal operation mode.

It is clear that delay and timing are important in this lab. During your simulations, attempt to identify your implementation's maximum clock frequency by increasing the frequency until proper functionality is no longer maintained for the post place and route circuit. It is important to test using the worst case delay; for blocks that have data-dependent delays you need to choose the right data vectors to exercise the longest delays. This is especially important for the addition and subtraction operations (what are the inputs that will lead to full carry propagation?).

Notes:

- Remember that reset occurs on logic low.
- Each of the components may be implemented using any of the design entry utilities that the tools offer: schematic editor, state machine editor, and VHDL editor.
- You may design your components however you like for this laboratory (e.g. bit-sliced vs. non-bit-sliced, etc.).
- It is OK for you to use components from the tool libraries. But I strongly encourage you to make sure that you understand how the components work, as some of those components can be misleading.
- If you use VHDL code you find (e.g. online, textbook, etc.), you MUST clearly cite your source. As with the library components you should make sure that you understand how the components work before you used them.
- If you synthesize a component that has a large amount of I/O (such as the 16-bit n:1 MUXes that you might use for the register file) by itself, the resource report will tell you that over 100% of the I/O pins are used, and you will not be able to place-and-route the design (but you can still simulate it post-synthesis, just not post-place-and-route). This is NOT a problem. The tool simply doesn't understand that the component is going to eventually be placed into a larger design. (Why would it?) When, for example, the MUX is incorporated into the register file and synthesized all together, the I/O problem will go away.

Deliverables:

We will be expecting the following items in the post-laboratory report. Please note that you do not have to show all of your VHDL or schematics for this write-up; however, any diagrams, schematics, VHDL, etc., that you think are useful in describing your implementation are welcome (i.e. don't waste space, but clearly explain the key points).

- A description of your register file implementation.
- A description of your control block implementation.
- A printout of the FSM you designed during the StateCAD tutorial.
- A description of your test plan and ModelSim simulation samples for verification of your design.
- A description of your synthesis results (optimizations, area values, delay values, etc.).

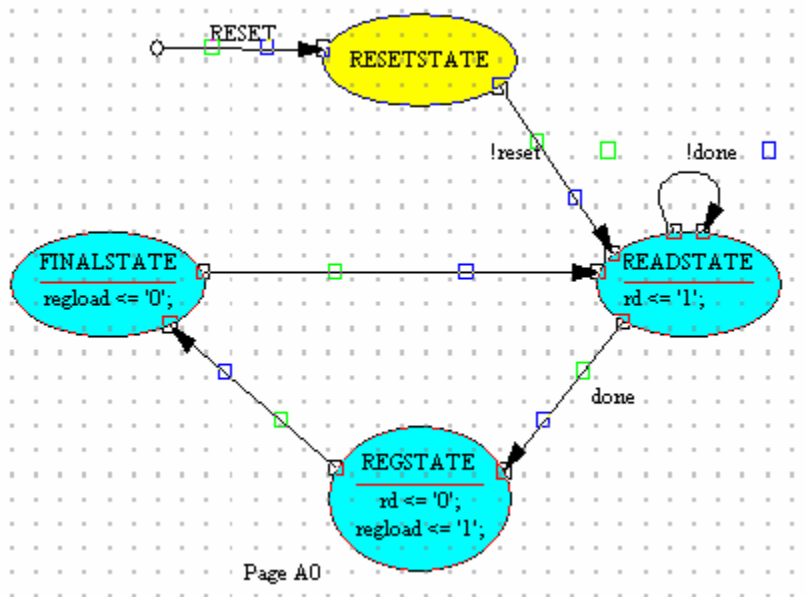
- Compare your maximum clock frequency achieved during simulation to the delay values stated in the timing reports provided by the tool. If they are different, explain why you think that is the case.
- A brief summary of any problems encountered or critical lessons learned from the assignment.

Please include your name, collaborators, and pledge on the report.

StateCAD tutorial:

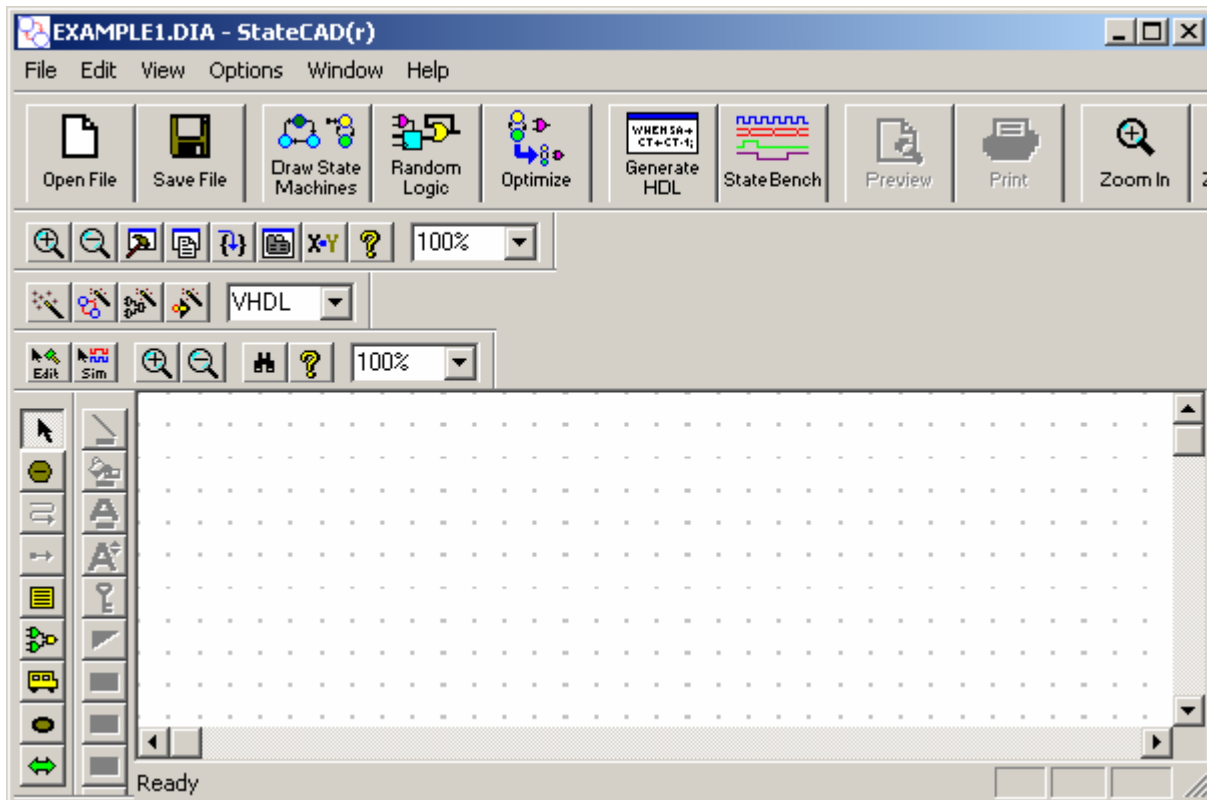
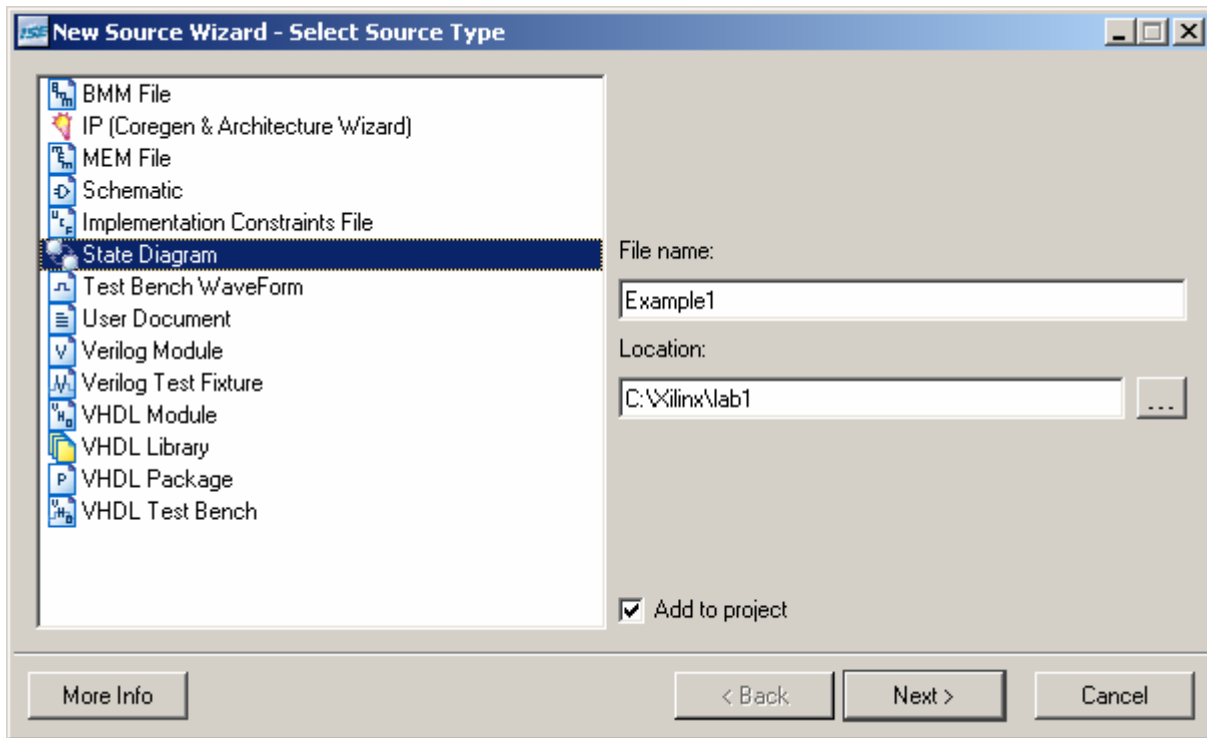
In this tutorial we will create a simple state machine with four states. The first state is the RESETSTATE that is reached when a 'reset' signal is asserted. From the RESETSTATE, the system transitions to the READSTATE on the next clock cycle. In the READSTATE, a read signal 'rd' is asserted high. The system stays in the same state till a 'done' signal is received. The system then moves to state REGSTATE, and 'rd' is asserted low. A 'regload' signal is now asserted. The system then moves to state FINALSTATE on the next clock cycle. The 'regload' signal is asserted low. From FINALSTATE, the system goes back to READSTATE on the next clock cycle.

The figure below shows what the final FSM would look like.



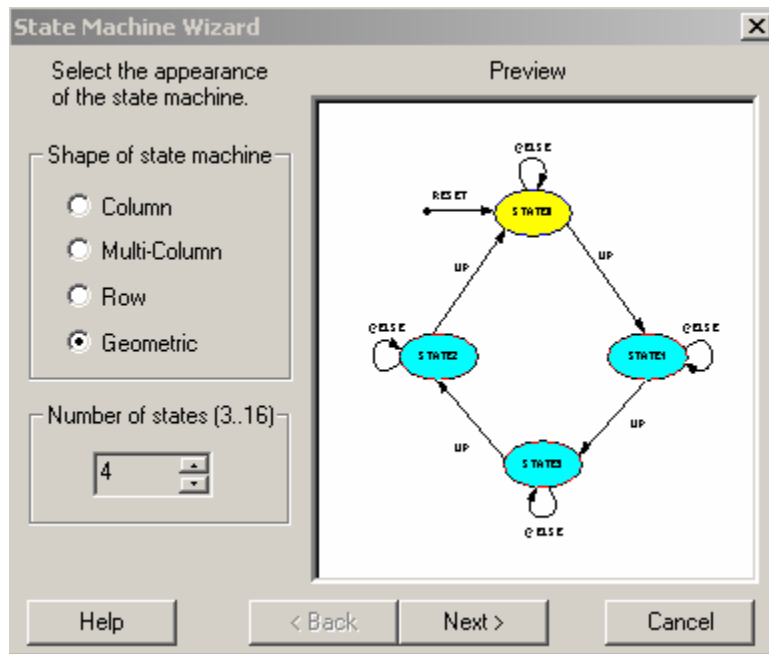
We will now go through a step by step tutorial for building the FSM shown above.

1. From project navigator, select New Source. In the New source menu, select State Diagram and give the file a name. We call the File Example1 in this tutorial.
2. Hit Next, and then Finished. This will open up StateCAD.

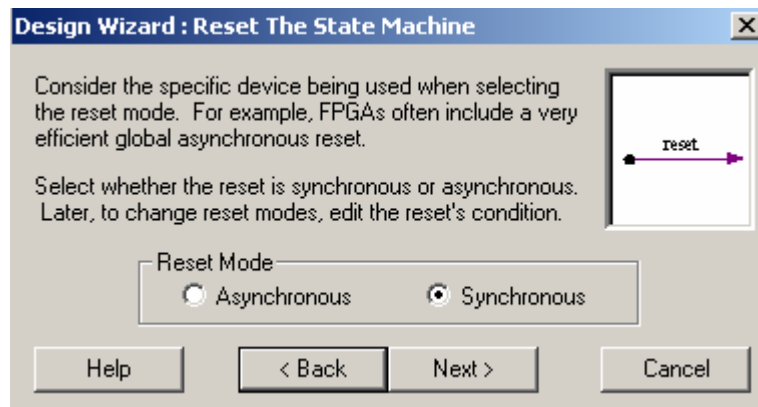


- Pick the  icon to start a new FSM.

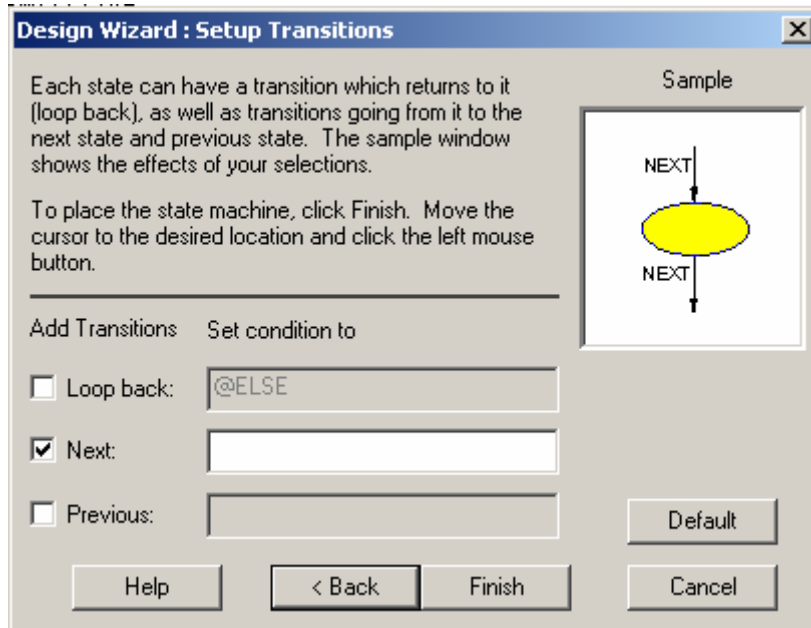
- The state machine wizard now opens up. Select geometric diagram with 4 states (the default is 5). Click next.



- Make the reset asynchronous. Click next.



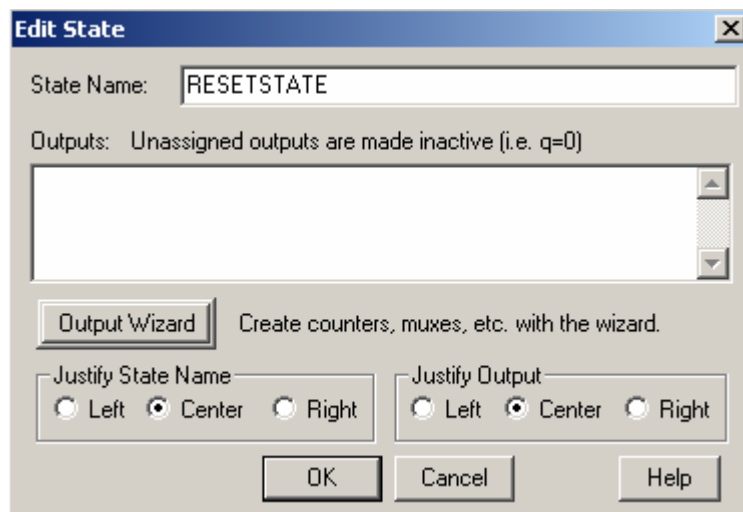
- You can now setup transitions. We will select only the transition to next state option. You can always add transitions back to the original state or to previous states later.



7. Click Finish. The design wizard screen goes away. You can place the FSM anywhere on the screen by left clicking where you want to place it.

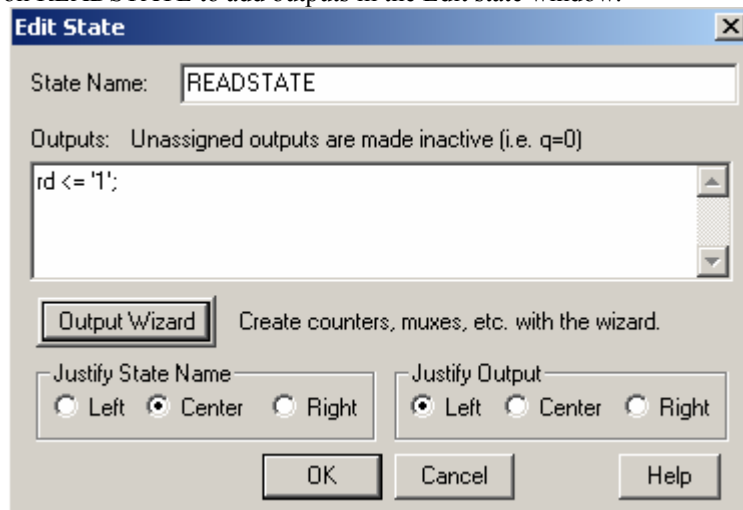
Note: If you are working in E225 or C123 after Step 7 of the tutorial, you might get an error saying: "System Error: Unable to re-open file_import.dmo during import (this is due to permissions settings, you should not get this if you work on your own PC). You will need to re-open the diagram using the file open command."

- a. Click OK.
 - b. Go to File Design Wizard.
 - c. A dialogue will ask you "Use the current diagram(YES) or create a new diagram(NO)?"
 - d. Select "YES".
 - e. Go through the "Design Wizard" process, which is basically same as Steps 4–6 of the tutorial.
 - f. After the diagram is placed in the window, click "cancel" to close the Design Wizard dialogue.
 - g. Continue with the rest of the tutorial.
8. Double click on STATE0 to change the name to RESETSTATE.

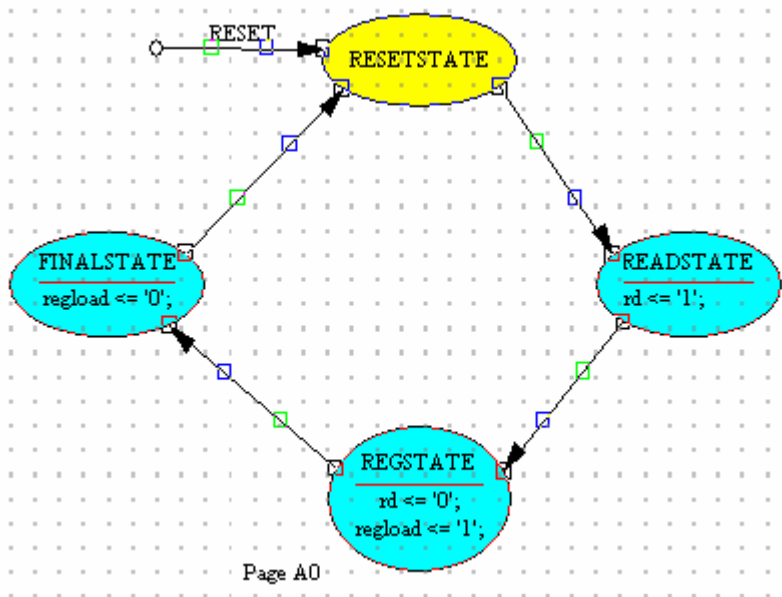



Rename the other states as well.

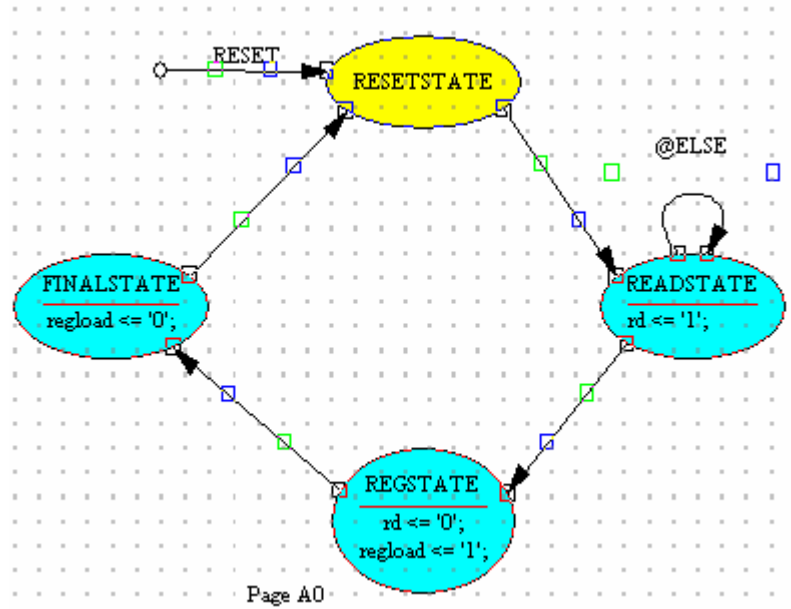
- Now double click on READSTATE to add outputs in the Edit state window.



Add the necessary outputs to the other states as well. Your FSM should now look like this.



- To recap, you have setup the state machine states and the outputs in each state. You are now ready to add transitions.
- The only extra transition to be added is a transition from READSTATE to itself while waiting for the done signal to become '1'. You can do this by using the add transitions  button on the left pane.
- Click inside the state you want to start the transition from. Now when you move the mouse arrow out of the state, a small red square appears at one corner of the state. Click on the state where you want the state to end. This will create a transition edge between the states. Note that the add transition interface is one of the clumsiest parts of the tool. You may have to click a few times to get it right.
For our example, we create a transition from READSTATE back to itself. An ELSE transition is now added to the state.



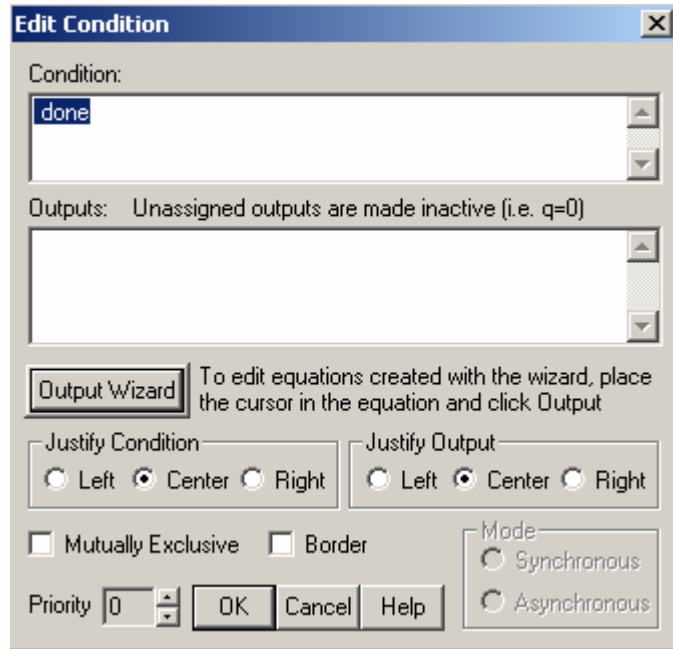
13. You set the conditions on each transition edge by clicking on the line representing the transition. Click on the line joining READSTATE and REGSTATE.

Note: If you are working on PCs in E225 or C123 after Step-13, you might get an error saying: “System Error: Unable to write profile when set_hdl_filename”

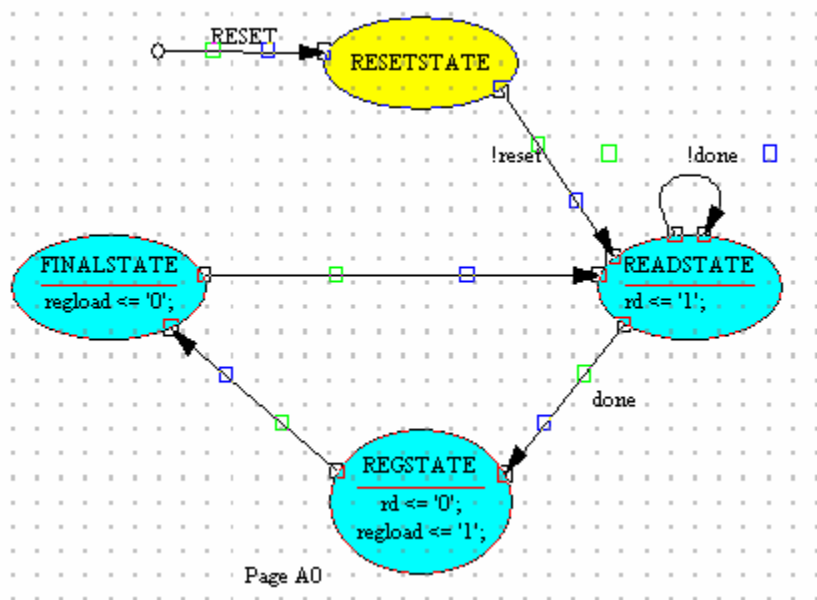
- a. Click OK and ignore the error.
- b. To perform the “Speed Optimization” of step 13, click on the “Optimize” button and follow the wizard. Select all the default values.
- c. Save the state diagram and close the window.
- d. Return to your main ISE window.
- e. Select your state diagram file (.dia file) in the sources window. Make sure “Synthesis/Implementation” is selected.
- f. Go to “Project Add Source”
- g. This should show up your vhdl file that was generated by the StateCad tool.
- h. In case you get an error saying “cannot add to project—source already exist”, then try “Project clean project files”. This will clear the vhdl file. Then generate VHDL again from the state machine and add it to your project.


For more info, see: <http://www.ece.osu.edu/xilinx/faq.html>

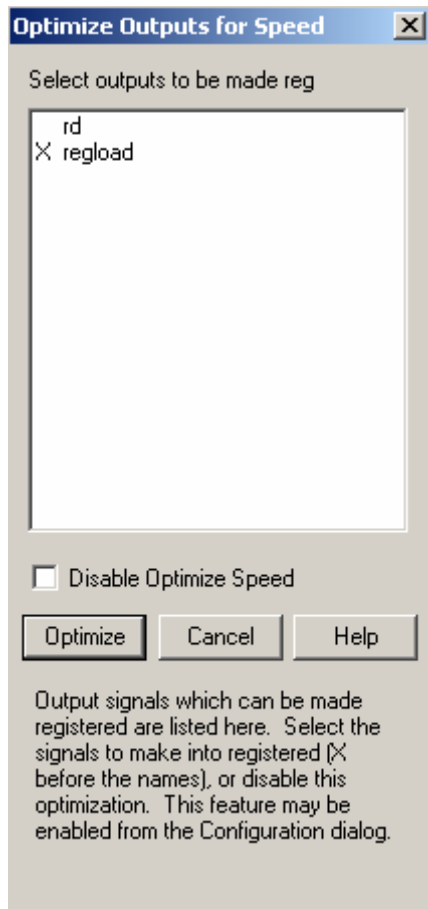
14. Enter the condition ‘done’. Note that the condition is evaluated and the transition is made if the condition evaluates to true. You do not need to set the condition as “if done=1” for example. You can set conditions like “cond1 & cond2 & (!cond3)” and so on. Also note that if you have a vector like control(3 downto 0) coming in, you set conditions differently. To check for control(3 downto 0) = “1110”, use the condition (control3 & control2 & control1 & (!control0)).



12. Add other conditions if needed. Note the (!reset) condition on the transition out of the RESETSTATE. Also move the transition joining FINALSTATE with RESETSTATE and to READSTATE. Your final FSM should look like this.



13. You can now generate HDL using the  button on the top pane. A new window titled “optimize outputs for speed” would open up. The output signals are shown with an X to indicate that the signals will be registered. Click on the signals to remove the X. This will ensure that the outputs are obtained in the same clock cycle as the state producing them. Click optimize.



15. The VHDL file “example1.vhd” representing the state machine is created and opens in a State CAD HDL browser window. You can look at the file to see the state machine created. You can now exit State CAD and go back to the Project Navigator. You need to import the VHDL file using the ADD new source option. Note that once the VHDL file is imported into the navigator and opened within it, the file is active in navigator. You need to close the file in navigator before you make any changes to the State CAD diagram and generated a new VHDL file. Navigator tends to crash if the VHDL file is changed from State CAD. Congratulations! Your controller is ready for use.