ECSE-323 Digital System Design

Lab #4 – ***Finite State Machines with VHDL*** Winter 2016

Prof. W. Gross and Prof. J. Clark

Introduction

In this lab you describe Finite State Machines in VHDL.

You will also learn how to use the SignalTap II on-chip logic analyzer.

Learning Outcomes .

After completing this lab you should know how to:

- Design Finite State Machines using VHDL.
- Use the SignalTap II logic analyzer.

Table of Contents .

This lab consists of the following stages:

1. On-Chip Testing using the SignalTap II Logic Analyzer
2. Design of a Permutation Circuit
3. Simulation of the Permutation Circuit
4. Design of the Rotor Stepper Finite State Machine
5. Simulation of the Rotor Stepper FSM
6. Design of the Rotor Stepper FSM Testbed
7. Testing of the Rotor Stepper FSM using the Testbed
8. Writeup of the Lab Report

1. On-Chip Testing using the SignalTap II Logic Analyzer .

The LEDs on the DE1 board are limited in number, and a human observer of these can only detect changes at a relatively low speed. Thus the LEDs provide only a limited means for probing the circuit behaviour when debugging a design. One could use an oscilloscope to measure signals brought out to the expansion connectors (the connectors on the right hand edge of the DE1 board), but there are no oscilloscopes in the lab, and even if there were, the signals brought out to the connectors can have only a relatively low maximum data rate due to the inductance of the output pins and connector.

It would be advantageous to be able to measure activity on the high speed data lines inside the chip.

One could use simulation to get an idea of the behaviour of the circuits inside the chip, but simulations do not always match the physical reality due to imperfect circuit modeling assumptions.

Fortunately, the Quartus II software provides a way to get at and measure the signals inside the chip. It does this with a tool called ***SignalTap II***.

SignalTap II is a logic analyzer whose circuitry is embedded into the FPGA fabric along with the circuit under test, and uses the FPGA's on-chip memory blocks to store time samples of the data lines being analyzed.

The SignalTap II logic analyzer can only be used if there are enough free logic block and memory block resources in the FPGA to support it. For the projects being constructed in this course, there will be plenty of free resources left over so that you should always be able to fit in a SignalTap II analyzer.

SignalTap II can be used to check timing issues which are not revealed by the functional simulation or the TimeQuest static timing analysis.

The functioning of the SignalTap II logic analyzer is patterned after hardware ***logic analyzer*** systems that have long been employed in digital system design and maintenance.

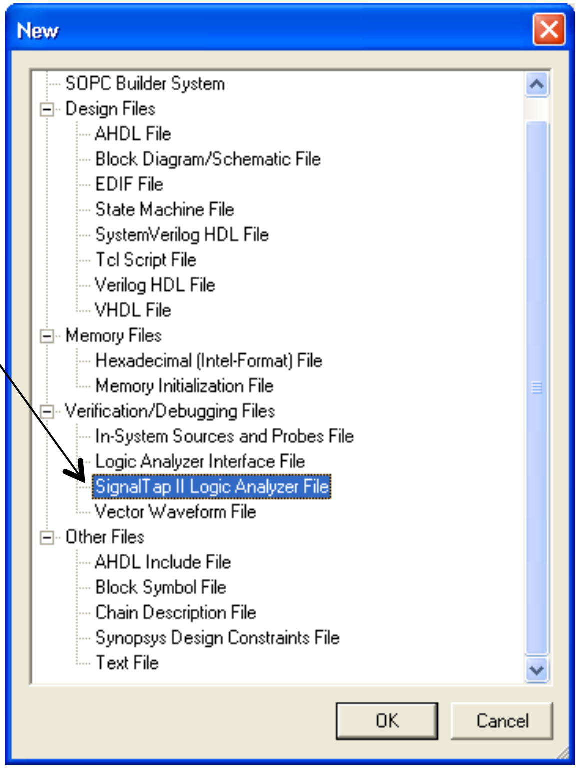
These systems include a set of high-speed probes (usually between 8 and 256) which connect to digital wires to be analyzed.

Software inside the logic analyzer samples the data on the probes at regular intervals, set by an internal or external clock. Typically the sampling is started, or triggered, by some logical condition on the lines being probed, and runs for a fixed time, or until a stop condition is sensed on the inputs.

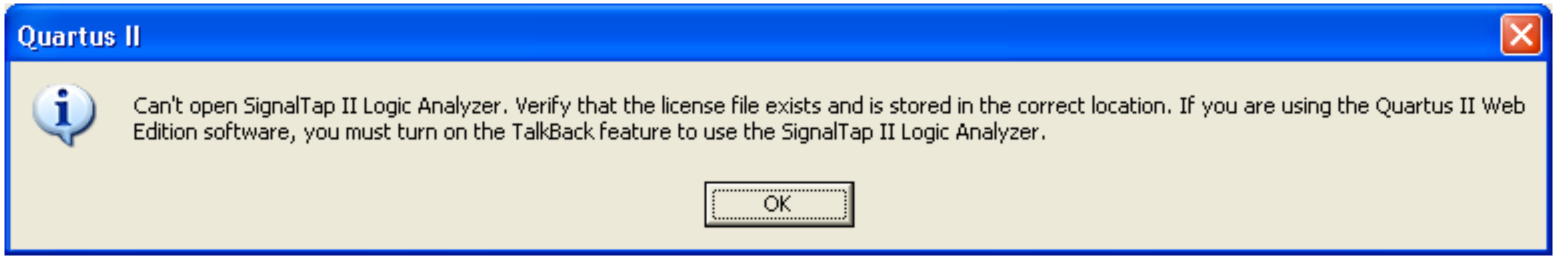
Image from http://www.elexp.com/tst_p100.htm

Likewise, the SignalTap II logic analyzer captures data on a set of data lines (nodes in the circuit that you specify), starting on a user-defined condition, and continuing for a set time period or number of samples. The data thus sampled is stored in on-chip memory, which is then read by the Quartus II software after the sampling period has finished.

To setup the Quartus software so that it can use the SignalTap II tool, first create a new SignalTap II Logic Analyzer File by choosing the corresponding item from the File/New menu.

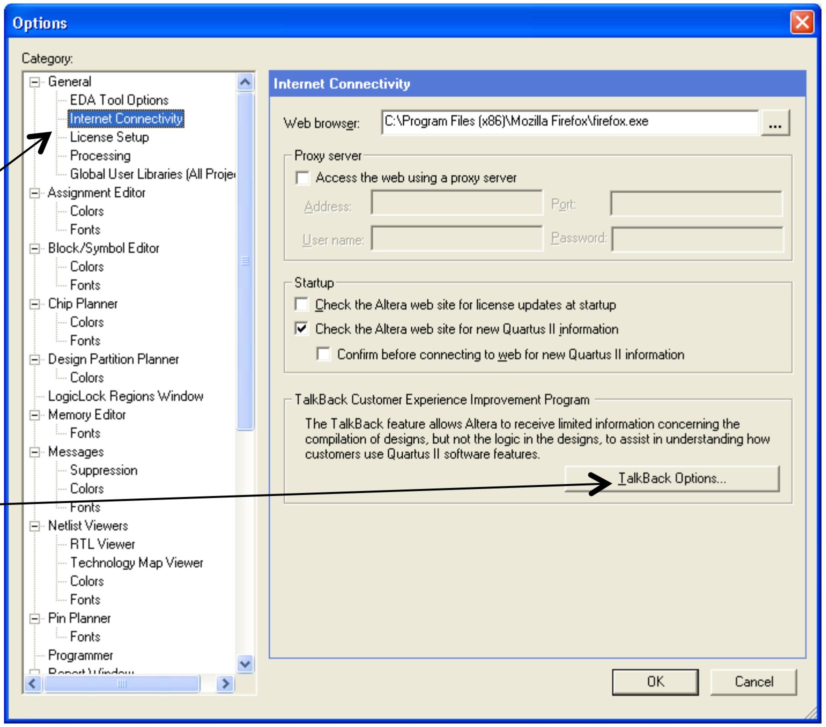


You may get the following error message popup (especially if you are using the web version of Quartus on your own computer:


Quartus II

Can't open SignalTap II Logic Analyzer. Verify that the license file exists and is stored in the correct location. If you are using the Quartus II Web Edition software, you must turn on the TalkBack feature to use the SignalTap II Logic Analyzer.

If so, turn on TalkBack by going to Tools/Options and selecting the Internet Connectivity tab. Then click on TalkBack Options and turn on the TalkBack Feature in the window that pops up.
You will have to restart Quartus.


Options

Internet Connectivity

Web browser: C:\Program Files (x86)\Mozilla Firefox\firefox.exe

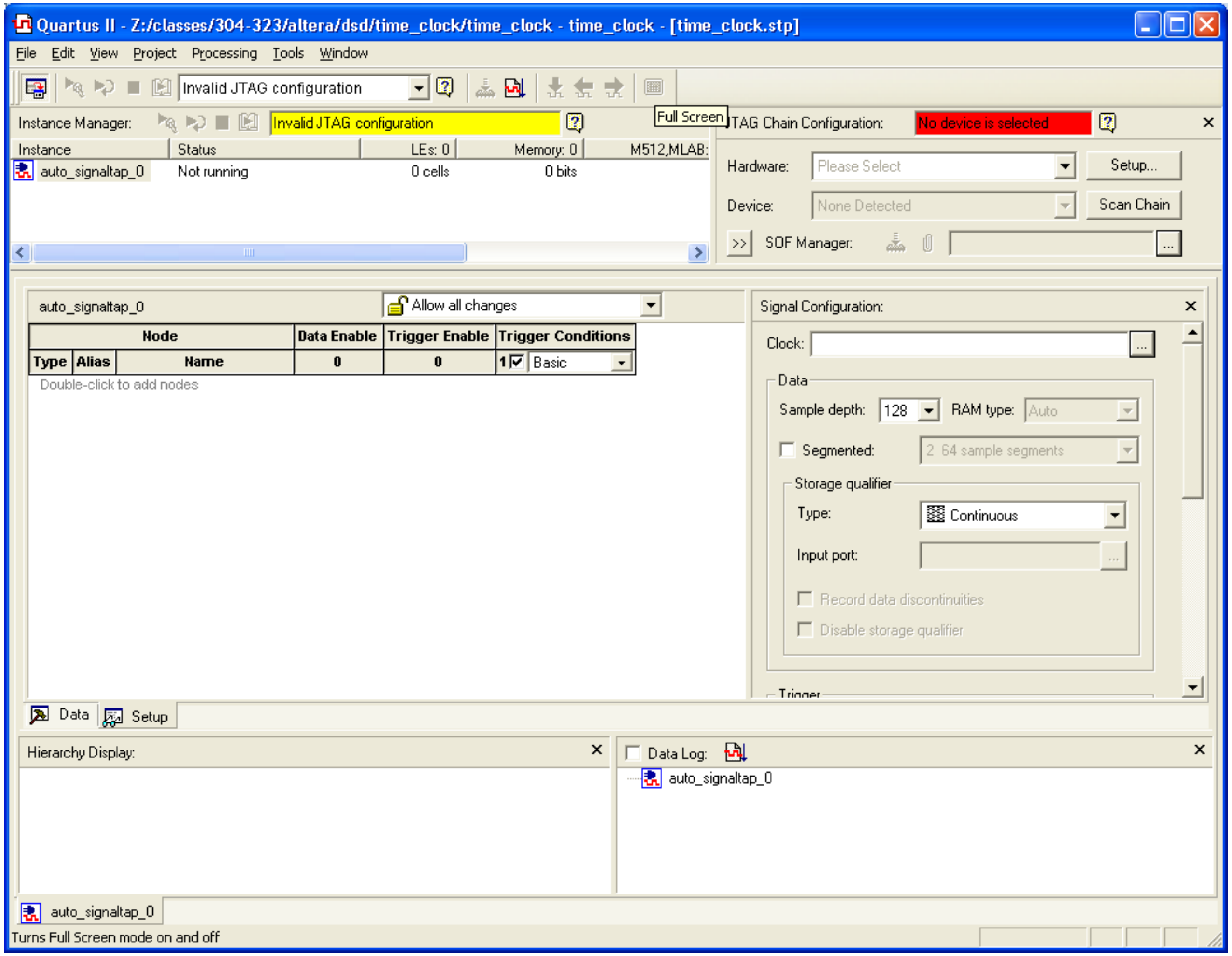
Proxy server
- [ ] Access the web using a proxy server

Startup
- [ ] Check the Altera web site for license updates at startup
- [✓] Check the Altera web site for new Quartus II information
- [ ] Confirm before connecting to web for new Quartus II information

TalkBack Customer Experience Improvement Program

The TalkBack feature allows Altera to receive limited information concerning the compilation of designs, but not the logic in the designs, to assist in understanding how customers use Quartus II software features.

The SignalTap II window should now appear, and look like the screenshot below



Quartus II - Z:/classes/304-323/altera/dsd/time_clock/time_clock - time_clock - [time_clock.stp]

Instance Manager: Invalid JTAG configuration

JTAG Chain Configuration: No device is selected

Instance	Status	LEs: 0	Memory: 0	M512,MLAB:
auto_signaltap_0	Not running	0 cells	0 bits	

Hardware: Please Select
Device: None Detected
SOF Manager:

auto_signaltap_0 — Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	0	0	1 ✓ Basic

Double-click to add nodes

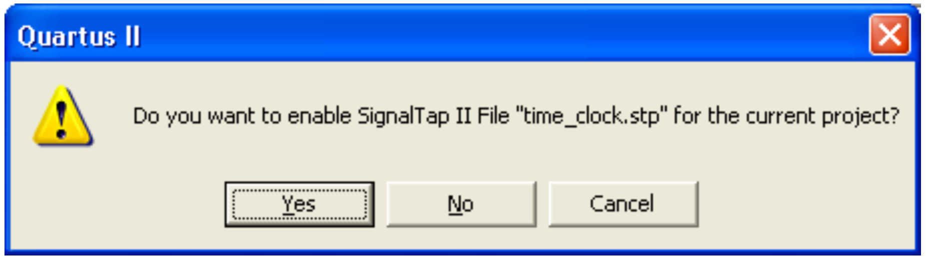
Signal Configuration:
Clock:
Data
Sample depth: 128 RAM type: Auto
Segmented: 2 64 sample segments
Storage qualifier
Type: Continuous
Input port:
Record data discontinuities
Disable storage qualifier

Hierarchy Display:

Data Log: auto_signaltap_0

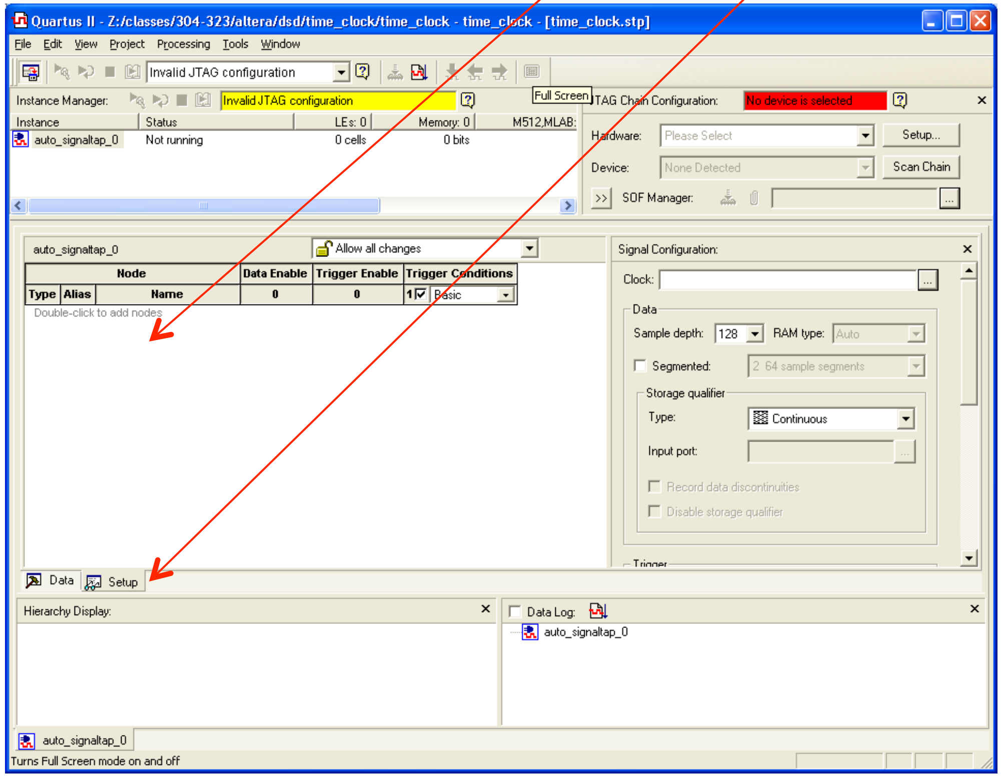
Turns Full Screen mode on and off

Before doing anything else, save the file (File/Save) giving the name "*gNN_testbed.stp*". In the window that pops up, say "Yes". This enables the SignalTap II file to be active for the current project.


Do you want to enable SignalTap II File "time_clock.stp" for the current project?

The next thing to do is to specify the circuit nodes to be analyzed. For this tutorial, you will look at the output signals of the 0-to-25 Counter module from the testbed.

To specify the signals to be analyzed, make sure that the Setup tab is selected, and double-click in the area labeled “Double-click to add nodes”. This will bring up the node finder window, which should be familiar to you.


Quartus II - Z:/classes/304-323/altera/dsd/time_clock/time_clock - time_clock - [time_clock.stp]

Instance Manager: Invalid JTAG configuration

Instance	Status	LEs: 0	Memory: 0	M512,MLAB:
auto_signaltap_0	Not running	0 cells	0 bits	

JTAG Chain Configuration: No device is selected

Hardware: Please Select

Device: None Detected

SOF Manager:

auto_signaltap_0 — Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	0	0	1 ✓ Basic

Double-click to add nodes

Signal Configuration:

Clock:

Data

Sample depth: 128 RAM type: Auto

Segmented: 2 64 sample segments

Storage qualifier

Type: Continuous

Input port:

Record data discontinuities

Disable storage qualifier

Trigger

Data Setup

Hierarchy Display:

Data Log: auto_signaltap_0

Turns Full Screen mode on and off

Add the nodes corresponding to the inputs of the 0-to-25 counter module. Also add the nodes corresponding to outputs.

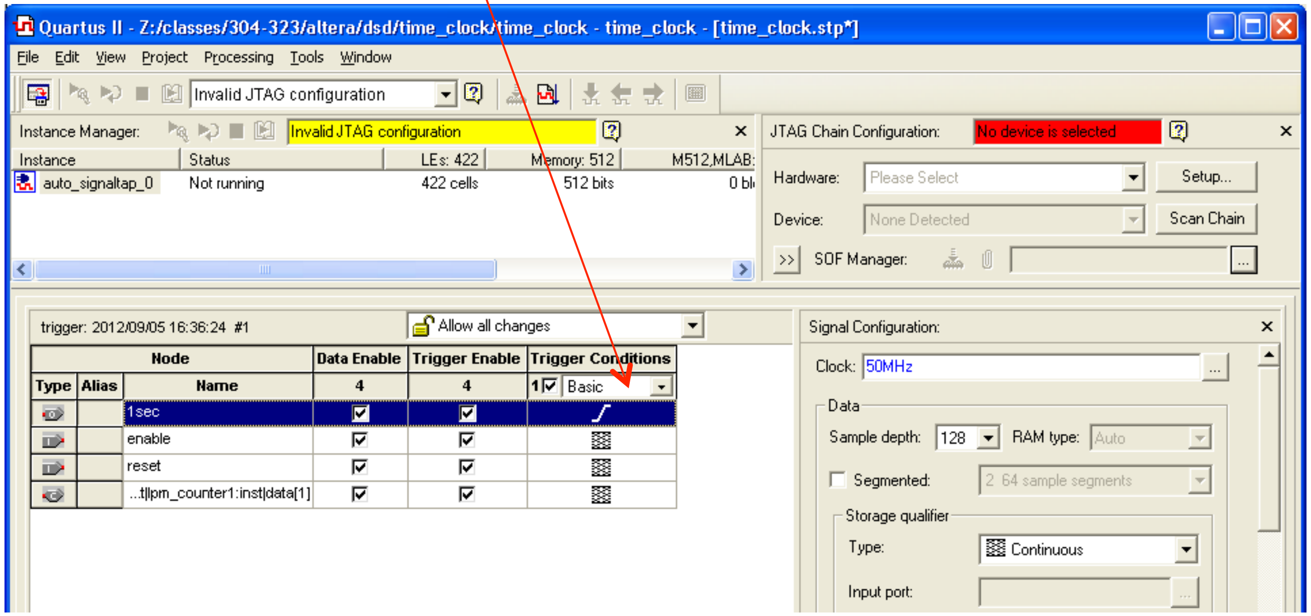
Make sure that the “Filter:” pane at the top has “SignalTap II: pre-synthesis” selected.

Do not add the clock input, as this should be used as the sampling clock (specified by entering it into the “Clock” item in the “Signal Configuration” pane on the right hand side).

Once the nodes have been added, recompile the circuit. This will add in the logic analyzer circuitry to the design, alongside the timer circuit. You will have to recompile every time you make a change to the SignalTap II settings (e.g. to add in another signal to measure). Once compiled, download the design to the board using the Programmer.

Once the design has been recompiled you can setup the triggering conditions, which will tell SignalTap II when it should start collecting measurements. This is done with the Setup tab selected.

Under "Trigger Conditions" select "Basic". We will want to trigger the data capture when the ***clock_enable*** signal goes high. Right-click on the trigger condition box in the row corresponding to count_enable and select "rising edge":



Quartus II - Z:/classes/304-323/altera/dsd/time_clock/time_clock - time_clock - [time_clock.stp*]

Invalid JTAG configuration

Instance Manager: Invalid JTAG configuration

Instance	Status	LEs: 422	Memory: 512	M512,MLAB:
auto_signaltap_0	Not running	422 cells	512 bits	0 bl

JTAG Chain Configuration: No device is selected

Hardware: Please Select

Device: None Detected

trigger: 2012/09/05 16:36:24 #1 Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	4	4	1 ☑ Basic
		1sec	☑	☑	/
		enable	☑	☑	▨
		reset	☑	☑	▨
		...t|lpm_counter1:inst|data[1]	☑	☑	▨

Signal Configuration:

Clock: 50MHz

Data

Sample depth: 128 RAM type: Auto

☐ Segmented: 2 64 sample segments

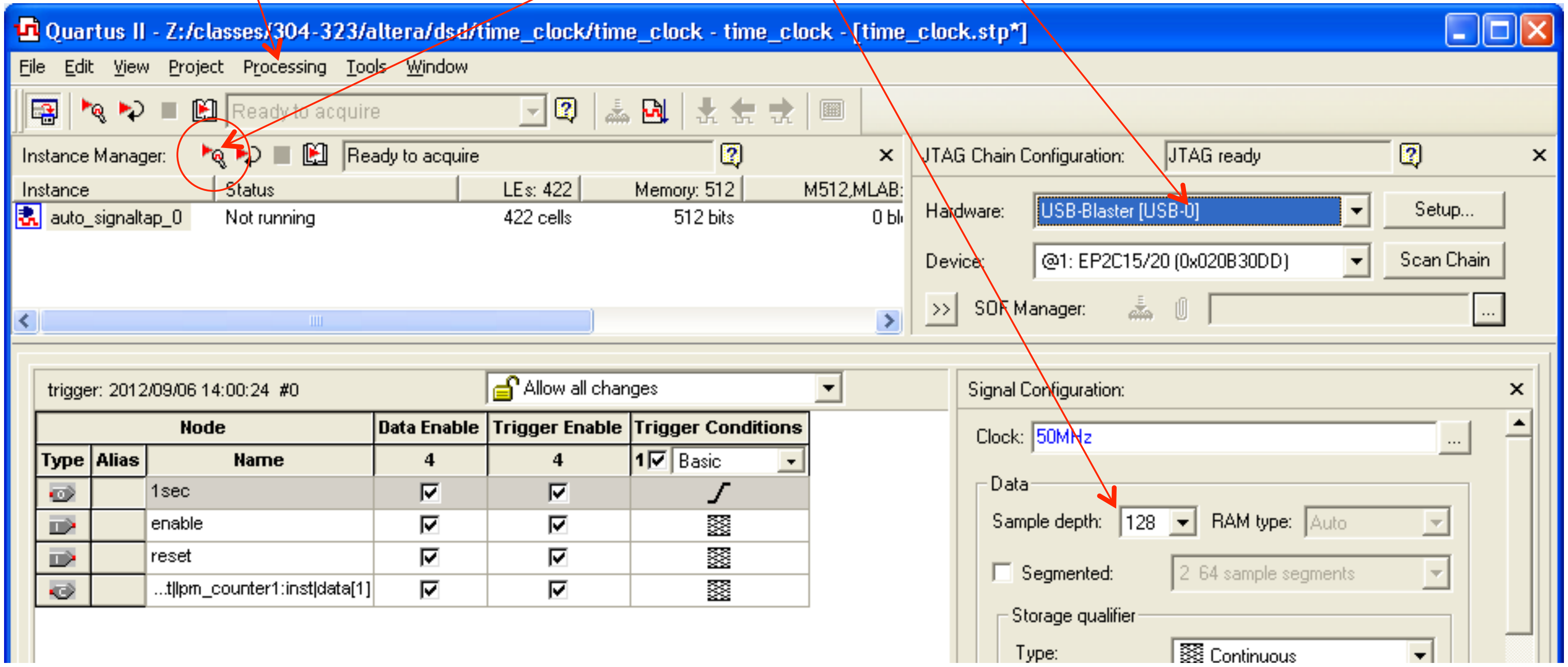
Storage qualifier

Type: Continuous

Input port:

Next, set the Hardware to USB-Blaster. A connection will be made between the Quartus SignalTap II window and the on-chip logic analyzer circuitry and the analyzer will be ready to acquire data.

To actually acquire the data, press on the Run Analysis button, or select Processing/Run Analysis from the menu bar. The analyzer will capture 128 samples, set by the value in the Sample Depth box.


Quartus II - Z:/classes/304-323/altera/dsd/time_clock/time_clock - time_clock - [time_clock.stp*]

File Edit View Project Processing Tools Window

Instance Manager: Ready to acquire

Instance	Status	LEs: 422	Memory: 512	M512,MLAB:
auto_signaltap_0	Not running	422 cells	512 bits	0 bl

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: EP2C15/20 (0x020B30DD)

SOF Manager:

trigger: 2012/09/06 14:00:24 #0 Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	4	4	1 ✓ Basic
		1sec	✓	✓	⌠
		enable	✓	✓	▨
		reset	✓	✓	▨
		...t|lpm_counter1:inst|data[1]	✓	✓	▨

Signal Configuration:

Clock: 50MHz

Data

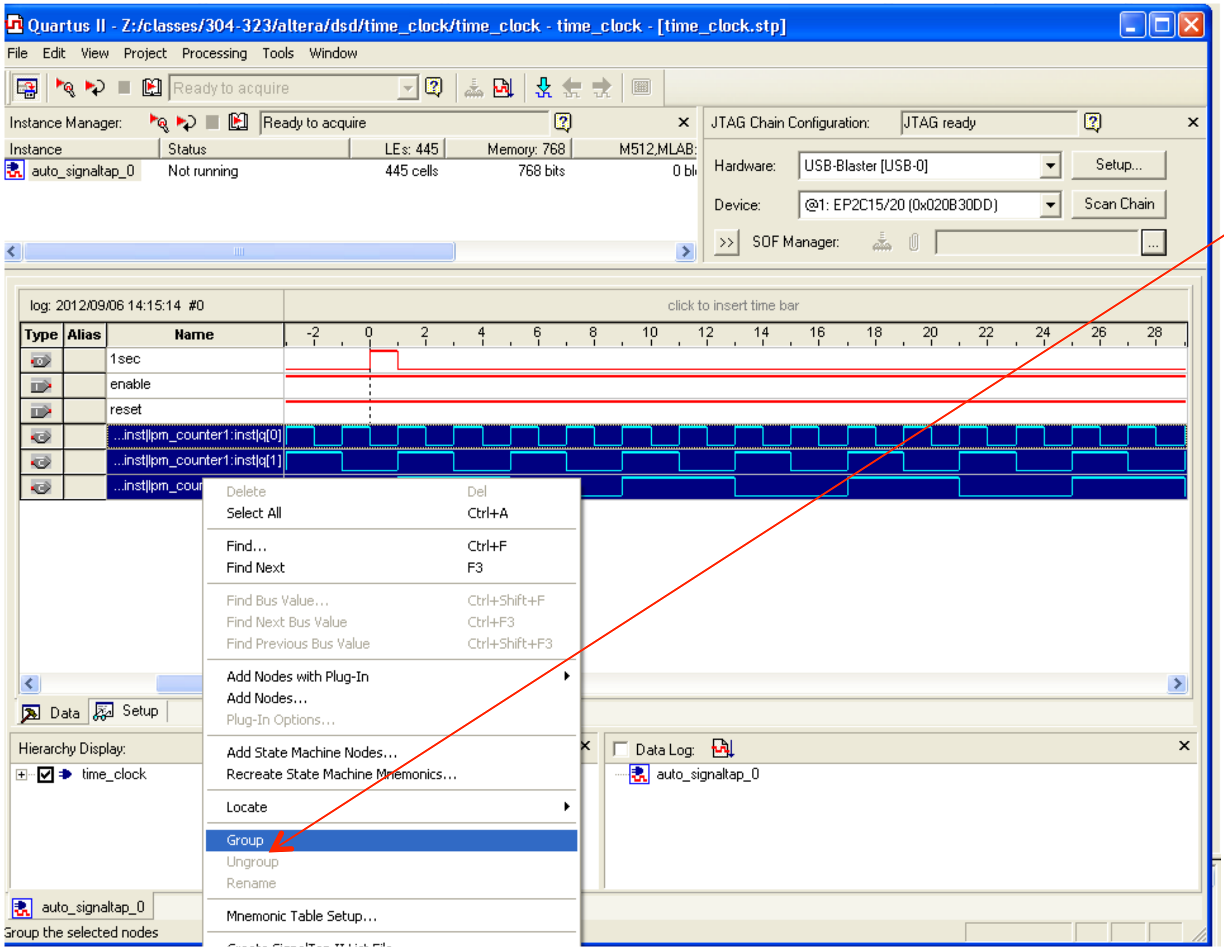
Sample depth: 128 RAM type: Auto

Segmented: 2 64 sample segments

Storage qualifier

Type: Continuous

Look at the **count** signals and check that the count sequence is correct. You can zoom in and out by placing the cursor over the signal traces and left- or right-clicking (left zooms in, right zooms out).



Quartus II - Z:/classes/304-323/altera/dsd/time_clock/time_clock - time_clock - [time_clock.stp]

File Edit View Project Processing Tools Window

Instance	Status	LEs: 445	Memory: 768	M512,MLAB:
auto_signaltap_0	Not running	445 cells	768 bits	0 bl

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: EP2C15/20 (0x020B30DD)

log: 2012/09/06 14:15:14 #0

If you have a set of signals that constitute a bus (such as count), you can group them together for convenience. To do this, select all the signals and then right-click. Select “Group” from the menu that pops up. Use this to view the counter output.

You can set the format of the displayed values by right-clicking on the bus name and selecting the very last entry (“Bus Display Format”).

Show the resulting waveforms of your analyses to the TA

TIME CHECK

You should be this far at the end of your *first* 2-hour lab period!

2. Design of a Permutation Circuit .

The Enigma machine encrypts messages by passing each letter through a scrambling process consisting of 3 successive code wheels. Each of these code wheels maps a letter onto another letter in a one-to-one fashion (and hence the mapping is a *permutation*).

For example, the so-called type I Rotor has the following mapping:

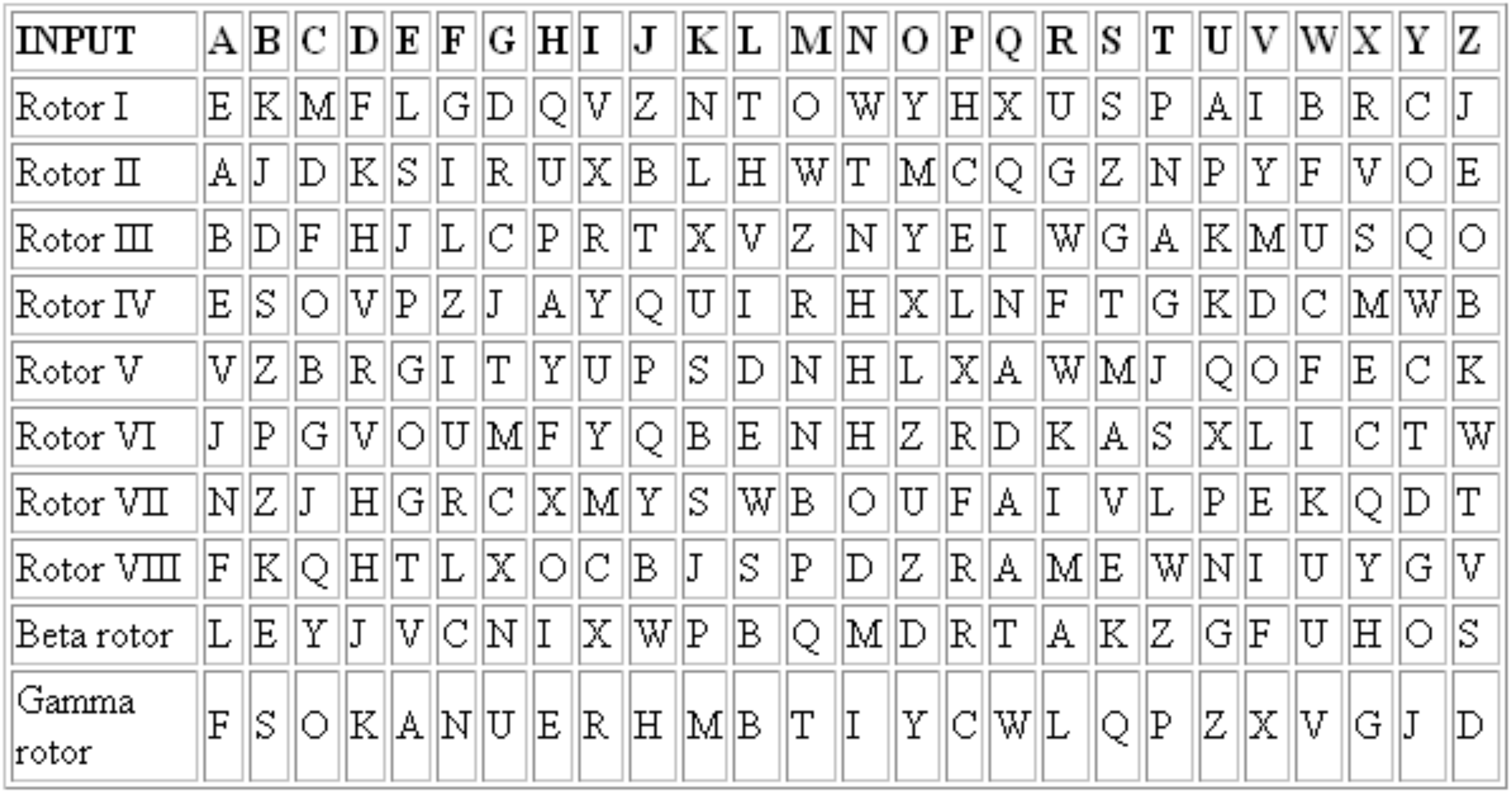
ABCDEFGHIJKLMNOPQRSTUVWXYZ

$\Downarrow$

EKMFLGDQVZNTOWYHXUSPAIBRCJ

(e.g. *A* maps to E, *F* maps to *G*, *Z* maps to *J*, and so forth…).

These are the permutations performed by the rotors:



INPUT	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Rotor I	E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J
Rotor II	A	J	D	K	S	I	R	U	X	B	L	H	W	T	M	C	Q	G	Z	N	P	Y	F	V	O	E
Rotor III	B	D	F	H	J	L	C	P	R	T	X	V	Z	N	Y	E	I	W	G	A	K	M	U	S	Q	O
Rotor IV	E	S	O	V	P	Z	J	A	Y	Q	U	I	R	H	X	L	N	F	T	G	K	D	C	M	W	B
Rotor V	V	Z	B	R	G	I	T	Y	U	P	S	D	N	H	L	X	A	W	M	J	Q	O	F	E	C	K
Rotor VI	J	P	G	V	O	U	M	F	Y	Q	B	E	N	H	Z	R	D	K	A	S	X	L	I	C	T	W
Rotor VII	N	Z	J	H	G	R	C	X	M	Y	S	W	B	O	U	F	A	I	V	L	P	E	K	Q	D	T
Rotor VIII	F	K	Q	H	T	L	X	O	C	B	J	S	P	D	Z	R	A	M	E	W	N	I	U	Y	G	V
Beta rotor	L	E	Y	J	V	C	N	I	X	W	P	B	Q	M	D	R	T	A	K	Z	G	F	U	H	O	S
Gamma rotor	F	S	O	K	A	N	U	E	R	H	M	B	T	I	Y	C	W	L	Q	P	Z	X	V	G	J	D

This table was taken from Tony Sale's web site: http://www.codesandciphers.org.uk/enigma/rotorspec.htm

Write a VHDL description for a circuit that implements the permutation operation. It should be able to specify the action of the ***first four*** rotor types (types I,II,III,IV in the table on the previous page).

The circuit should provide not only the permutations shown in the previous page, but also the inverse permutation. For example, for Rotor type I, the forward permutation is **A->E, B->K, C->M**, etc., while the inverse permutation is **A->U, B->W, C->Y**, etc. **A** corresponds to code 0, **B** to 1, etc.

Use the following entity declaration, and employ a process block, with nested case statements (the outer for the rotor_type, the inner for the input_code).

```
entity gNN_permutation is
 port ( input_code        : in std_logic_vector(4 downto 0);
        rotor_type        : in std_logic_vector(1 downto 0);
        output_code       : out std_logic_vector(4 downto 0);
        inv_output_code   : out std_logic_vector(4 downto 0));
end gNN_permutation;
```

3. Simulation of the Permutation Circuit .

Once you have finished your VHDL description of the permutation circuit, show it to the TA and explain its design.

Perform a functional simulation of the circuit using Modelsim.

Test all 26 valid input codes, for each of the 4 rotor types.
Check that both the forward and inverse permutations are correct for all cases.
Show your simulations to the TA.

TIME CHECK

You should be this far at the end of your *second* 2-hour lab period!

4. Design of the Rotor Stepper Finite State Machine.

The standard Enigma machine has 3 rotors, which sequentially scramble the code corresponding to the typed character. If the position of the rotors were fixed, then having 3 rotors would be no better than having 1, in terms of the scrambling power. But the rotors can rotate in the Enigma machine, increasing the number of different scramblings that can be obtained.

The *right-hand* rotor steps by one place *every time* a key is pressed. The *middle* rotor steps by one place when the right-hand rotor steps ***from*** a certain point, which we will call the *knock-on,* or *notch point*. The notch points for the first 4 rotor types are:

- Rotor I - **Q**
- Rotor II - **E**
- Rotor III - **V**
- Rotor IV - **J**

Thus, when a rotor of type I steps from Q to R, the next rotor will also be stepped.

The same behaviour holds true for the left rotor - it steps when the middle rotor steps from its notch position.

There is one added complication to this behaviour - the middle rotor also steps whenever the left rotor steps. Thus the middle rotor can sometimes step twice in a row.

Here is an example sequence, assuming that the right rotor has type I, the middle rotor of type II, and the left rotor has type III:

ADP ---> ADQ (no rotors are in their notch positions, so just the right rotor steps)

ADQ ---> AER (the right rotor is in its notch position, so both the middle and right rotors step)

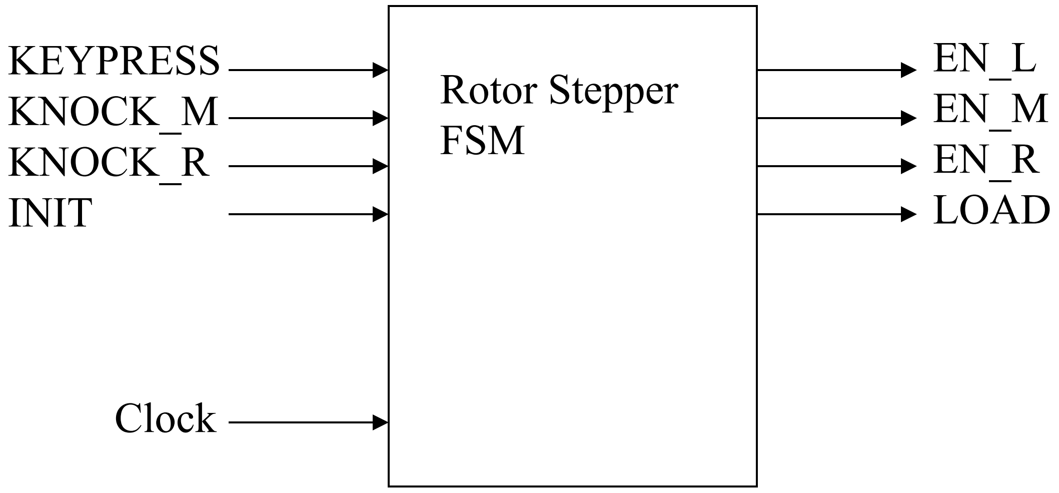
AER ---> BFS (the middle rotor is in its notch position, so all rotors step, even though the right rotor is not in its notch position)

BFS ---> BFT (no rotors are in their notch positions, so just the right rotor steps)

Note that the notch position of the left rotor is irrelevant.

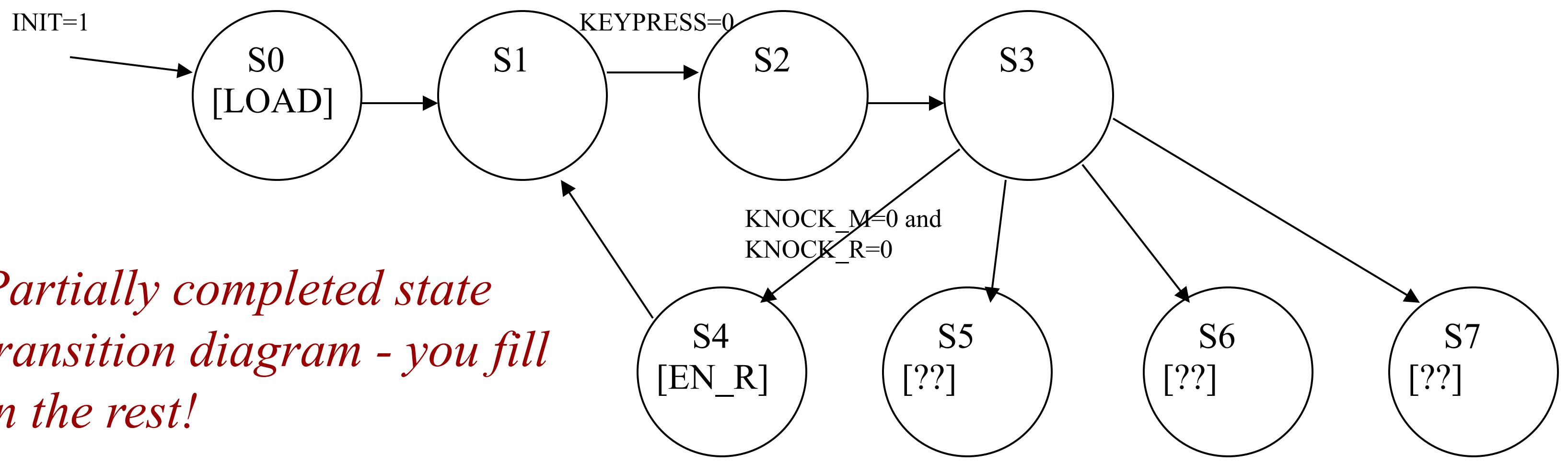
Using VHDL describe a ***Moore***-style finite state machine that will generate the ENABLE signals for 3 instances (one for each rotor) of the 0-25 counter developed in the previous lab. These control signals will be such as to cause the 3 counters to mimic the behaviour of the 3 rotors in the Enigma machine.

The controller FSM should have 4 *control input signals* - **KEYPRESS, INIT**, **KNOCK_M**, and **KNOCK_R**, and should have 4 *control output signals* - **EN_L**, **EN_M**, **EN_R**, and **LOAD**. The **EN_** signals will be used to enable the respective rotor counters, while the **LOAD** signal will go to all 3 of the rotor counters and enable their initialization.



The FSM should check for a rising edge of the KEYPRESS signal using a 3-state process where the first state waits for KEYPRESS to go low, then goes to a state where it waits for KEYPRESS to go high, then proceeds to the state where the values of KNOCK_M and KNOCK_R are checked. If both KNOCK_M and KNOCK_R are low, then only EN_R should be asserted. If KNOCK_R is high then both EN_R and EN_M should be asserted. If KNOCK_M is high then EN_R, EN_M, and EN_L should all be asserted.

If INIT is high in any state, the FSM should go to a state where the LOAD signal is asserted (to initialize the rotor counters) and then go to the KEYPRESS wait states.



Partially completed state transition diagram - you fill in the rest!

5. Simulation of the Rotor Stepper FSM.

Show the completed state diagram for the FSM and your VHDL description of it to the TA and explain to him or her how it works.

Then perform a functional simulation of the FSM using Modelsim. Demonstrate the proper detection of the rising edge of the KEYPRESS signal, and the effect of the INIT signal. Demonstrate that the proper sequence of rotor counter ENABLE signals is generated according to the value of the KNOCK_M and KNOCK_R inputs (note: KNOCK_M and KNOCK_R will not both be high at the same time, unless they are initialized to that state).

Show the results of the simulation to the TA.

TIME CHECK

You should be this far at the end of the *third* 2-hour lab period!

6. Design of the Rotor Stepper FSM Testbed.

In this part of the lab you will make a testbed to demonstrate the functioning in hardware of the Rotor Stepper FSM in controlling the rotor counters.

The testbed should include three instances of the 0-25 counters designed in lab 3, one instance of the rotor stepper FSM developed in the previous section, and two 5-bit comparators used to determine whether the Right and Middle rotors are at their notch positions (i.e. the outputs of these comparators will be KNOCK_R and KNOCK_M). The outputs of the three counters (i.e. the counts from 0 to 25) should be passed to three 7 segment displays.

NOTE: You will have to modify the counter circuits slightly to allow synchronous setting of the count to a given value.

Use the 2Hz Pulse Generator developed in Lab 3 to provide the KEYPRESS input of the FSM. Connect one of the Push Buttons on the Altera board to the INIT input of the FSM. Set the rotor counter initialization values (the values connected to the counters' data inputs) to constant values of your choosing.

Set one of the inputs to the KNOCK comparators to constant values set to the notch positions according to the choice of rotor types that you make (I,II,III, or IV). (The other inputs should be connected to the count output of the rotor counters).

7. Testing of the Rotor Stepper FSM using the Testbed.

Use VHDL to connect all the parts of the testbed together. Show your VHDL to your TA.

Download the compiled testbed circuit to the Altera board.

Test the function of the FSM using the testbed. Demonstrate the functioning to the TA.

TIME CHECK

You should be this far (i.e. have completed the lab) at the end of your *fourth* 2-hour lab period!

8. Writeup of the Lab Report .

Write up ***three*** short reports, one for the permutation circuit, one for the rotor stepper FSM circuit, and one for the complete testbed.

The reports must include the following items:

- A header listing the group number, the names and student numbers of each group member.
- A title, giving the name (e.g. ***gNN_permutation***) and function of the circuit.
- A description of the circuit's function, listing the inputs and outputs. Provide a pinout or symbol diagram.
- The VHDL description of the circuit.
- A discussion of how the circuit was tested, giving details of the testbed and showing representative simulation plots, as well as any physical measurements you may have made.
- A summary of the timing performance of the circuit, giving the timing analysis and the simulated propagation delays.
- A summary of the FPGA resource utilization (from the Compilation Report's Flow Summary).

The lab report, and all associated design files must be submitted, as an assignment to the myCourses site. Only one submission need be made per group (both students will receive the same grade!).

Combine all of the files that you are submitting into one *zip* file, and name the zip file gNN_LAB_3.zip (where NN is your group number).

The reports are due one week after the last day of the lab period, or Friday, April 1, at 11:59 PM.

Grade Sheet for Lab #4 Winter 2016.

Group Number:______________.

Group Member Name:	______________.	Student Number:	______________.
Group Member Name:	______________.	Student Number:	______________.

Marks			TA Signatures
	1.	SignalTap II waveforms	.
	2.	VHDL description of the permutation circuit	.
	3.	Simulation of the permutation circuit	.
	4.	VHDL description of the rotor stepper FSM	.
	5.	Simulation of the rotor stepper FSM	.
	6.	VHDL of the rotor stepper FSM test-bed	.
	7.	Testing of the rotor stepper FSM test-bed	.

Each part should be demonstrated to one of the TAs who will then give a grade and sign the grade sheet. Grades for each part will be either 0, 1, or 2. A mark of 2 will be given if everything is done correctly. A grade of 1 will be given if there are significant problems, but an attempt was made. A grade of 0 will be given for parts that were not done at all, or for which there is no TA signature.