Lab 4: PRNG with LFSR and TRNG with Ring Oscillators

University of Massachusetts Amherst

- Verilog is a hardware description language(HDL)
- The goal is to describe hardware functionality
- Verilog is similar to programming languages such as C in terms of its syntax but its goal is to describe the hardware and its basic building blocks such as and/nor gates and flip flops etc..

Introduction to Verilog

Data Types in Verilog

- For this lab, we will cover just some of the basics including data types:

- Two main types: Nets and Variables

  - Net: does not store value but meant to connect hardware entities, ex: wire [3:0] n; 4 bit wire acting as bus
  - Variable” Meant to store value, most common is reg but also real, integer and others. Ex: reg [2:0] var; var = 2’b01

Modules

- Similar to classes in Python, modules are meant for functionality.
- Has input and output port lists to connect with other modules and feed input, output
- Many simple examples including D flip flop:

```
1  // Module called "dff" has 3 inputs and 1 output port
2  module dff (    input           d,
3                                  input           clk,
4                                  input           rstn,
5                                  output reg  q);
6
7      // Contents of the module
8      always @ (posedge clk) begin
9          if (!rstn)
10             q <= 0;
11         else
12             q <= d;
13     end
14 endmodule
```

Procedural Blocks

- In this lab, we will use the always statement
- Always @ (posedge clk or posedge rst)
- This statement says that at every positive edge of the clock or rst, certain operations will occur simultaneously.

Example:

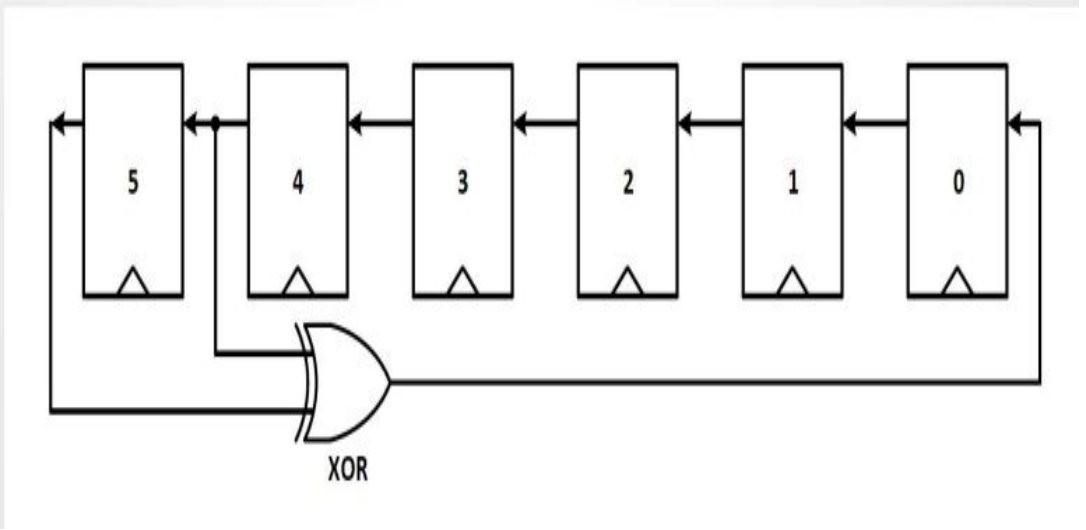
```
always @ (posedge clk or negedge rstn) begin
    if (!rstn)
        q <= 0;
    else
        if (d)
            q <= ~q;
        else
            q <= q;
end
```

- Now that we covered some basics of verilog, let's put it to use.
- Random numbers can be approximated by pseudorandom numbers, which are not truly random since they can be predicted
- A broadly used PRNG method is based on Linear Feedback Shift Register
- The initial value of the LFSR is called a seed and can be any value except 0x0
- The module with n registers will generate a pattern with (2^n)-1 different numbers, after which the pattern is repeated.

Generating Random Numbers with LFSR

LFSR Continued

- In this lab you are given a 6 bit LFSR module and only bits 2, 4 and 6 will be linked with the output for the PRNGs.
- For the chosen parameters, the corresponding feedback polynomial is (x^6) + (x^5) + 1
- With a period of (2^6)-1 = 63



- Create a new project called lfsr_prng
- Add a file called LFSR_PRNG.v
- Your goal is to write verilog to describe the functionality of the LFSR described above: here is some starter code setting the module and the declaration of the shift register:

```
module LFSR_PRNG (input clk, rst, output reg [2:0] out);
    reg [6:0] D123456;
```

Step 1 - Create a Project

- We have to set “LFSR_PRNG.v” as top-level entity and compile with start Analysis and Synthesis.
- Note: If at this point LFSR_PRNG.v is the only file in your project then it is automatically the top-level entity)
- Make sure you have no errors
- We will do a simulation next

Step 2: Analysis And Synthesis

- We now want to simulate the LFSR.
- We are going to use Altera ModelSim, which is a powerful verilog-HDL simulation tool developed by Mentor Graphics.
- This should have downloaded with the Quartus download

Simulation with Modelsim

- Open ModelSim from the start menu
- Go to File->New->Project and save the project location in the same place as your quartus project, and set the project name as “LFSR_sim”
- Leave the Default library Name as ‘work’ and then press OK
- In the next dialog window, select ‘Add Existing File’
- Browse to open the VHDL file “LFSR_PRNG.v”. Click ok and close

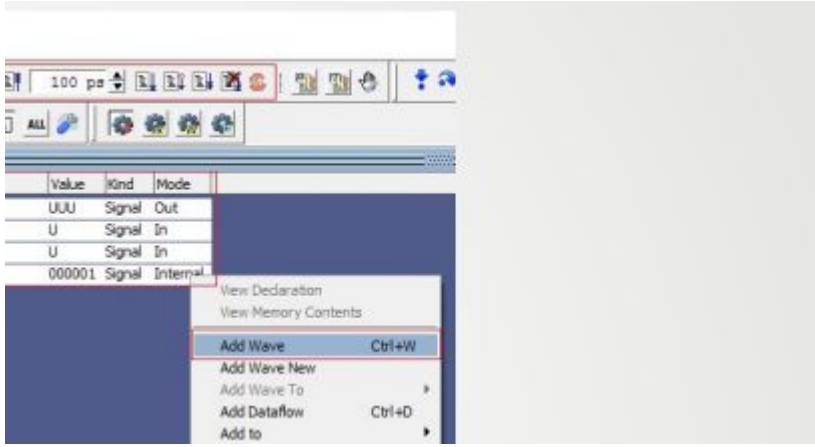
Step 4: Creating a Project with ModelSim

- Familiarize yourself with ModelSim interface
- The project Files can be seen in the tab ‘Project’
- The ‘Library’ tab displays your current library “work” and other libraries
- The ‘Transcript’ is located below the tabs, together with ModelSim shell
- Compile your project by right clicking on the file or by clicking on the icon ‘Compile’ or Compile All’

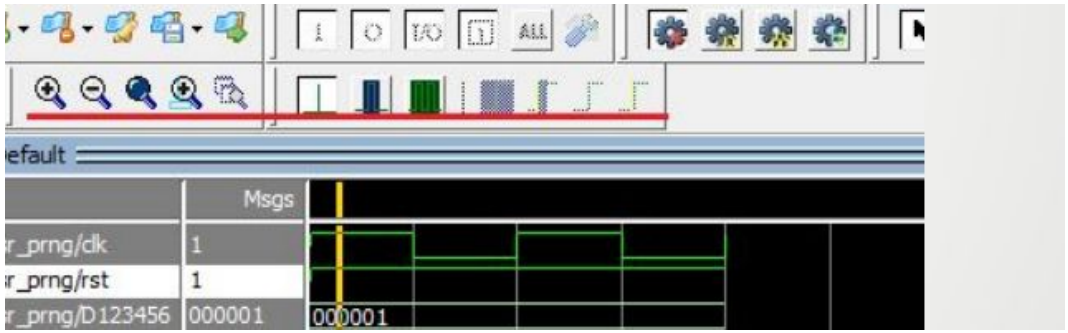
Step 5: Working with ModelSim

- Click on ‘Library’, open the library “work”
- Right click on “lfsr_prng” and choose ‘Simulate’
- A new tab, labelled ‘sim’ will appear next to ‘libraries’ and ‘project’ and will display relevant signals from your design
- Resize the sections for a comfortable view
- Take a moment to inspect the details displayed
- In the section ‘Objects’ select all the signals, right click and select ‘Add Wave’ (as seen in the next slide)
- The wave - default section will open

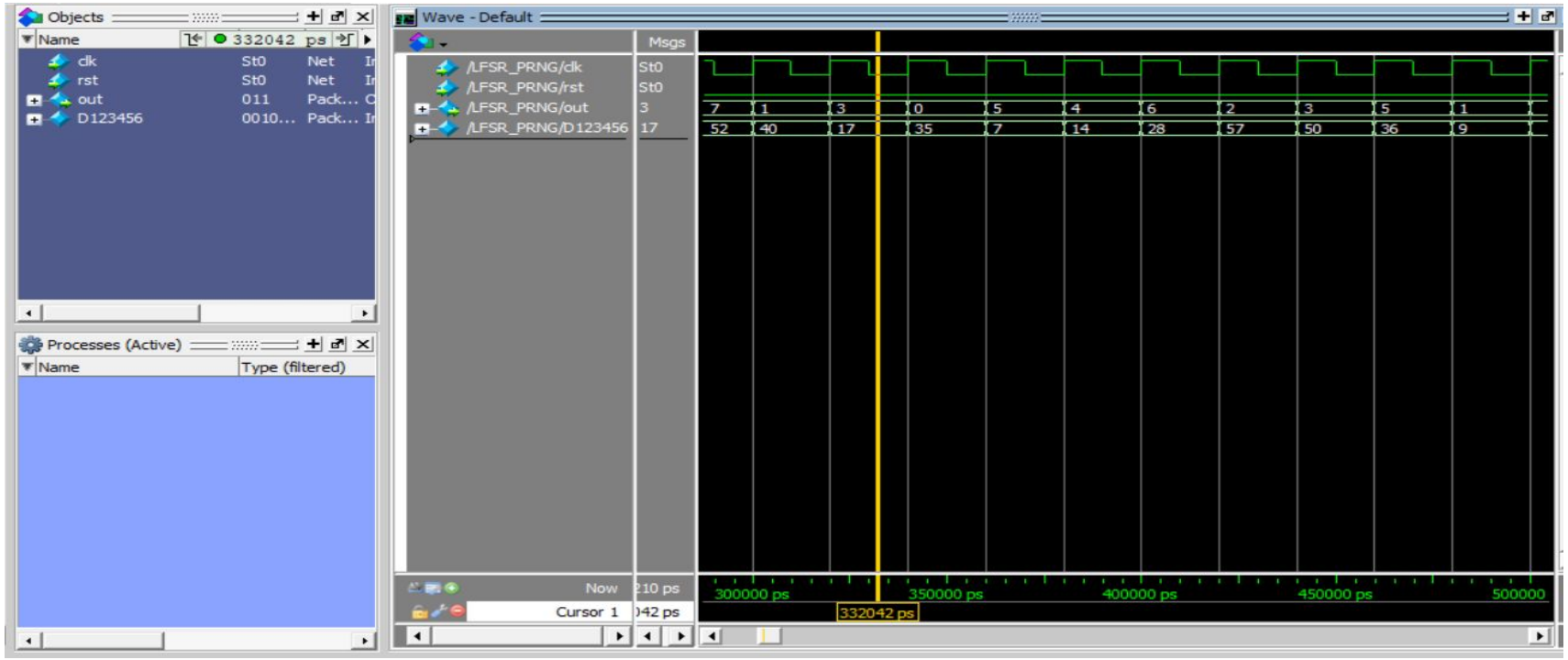
Step 6: Adding a Wave



- **Right click on signal '/lfsr_prng/rst' and select force**
- **Set the value to "1" (in reset)**
- **Set the runtime to 40ns and click run(F9).**
- **Zoom out to obtain a view as shown below**



- Now, force the reset signal to value “0” and run the simulation for 1300ns (or longer, ex 2600ns)
- You should see how the values in both ‘D123456’ and ‘prn’ changes (sample image on next slide)
- Note: If you do not see any values, make sure you are zoomed out
- Notice how the signal ‘prn’ is just a sampled version of ‘D123456’
- For the first clock cycle the LFSR has value “000001”, this value repeats again only in the very last part of the simulation

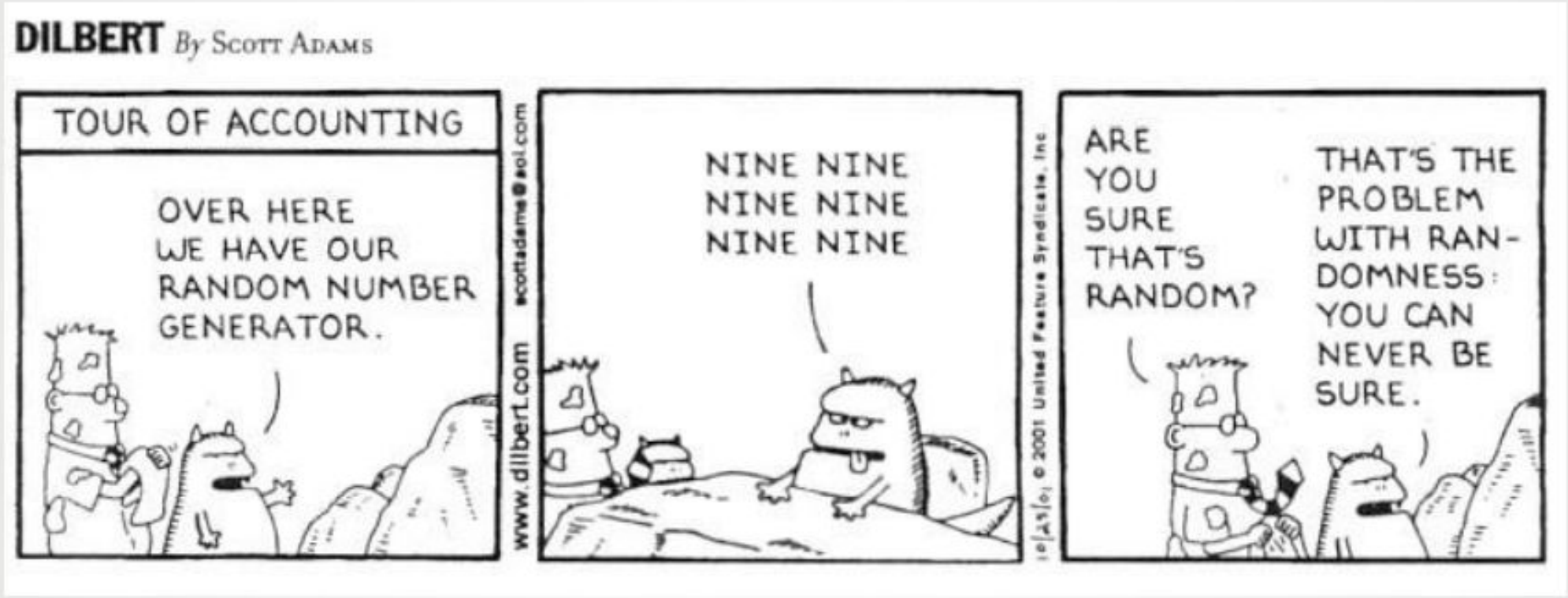


- In addition you can change the signal values from binary to unsigned decimal form
- To do this right click on each ‘prn’ and ‘D123456’
- Go to Radix->Unsigned and view the values in unsigned decimal form

Take a screenshot of your code AND the simulation with the waveforms.

Include both in your report.

END OF PART 1


DILBERT By Scott Adams
TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
www.dilbert.com scottadams@aol.com
NINE NINE NINE NINE NINE NINE
10/15/01 © 2001 United Feature Syndicate, Inc
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.

- **We will use VHDL(very high speed integration circuit hardware description) here.**
- **Similar to Verilog but main differences are:**
  - allows the user to define data types
  - allows concurrent procedure calls
  - harder to learn
- **More commonly used by government organizations and contractors**
- **Note: Please reference VHDL Tutorial - javatpoint for documentation**

VHDL Tutorial

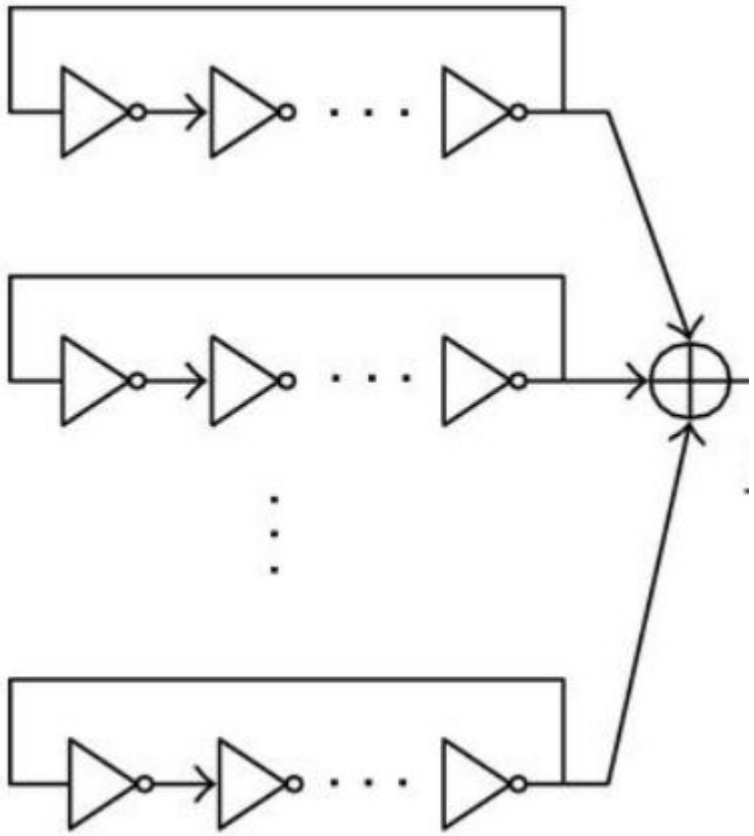
Now that you have had some exposure to Hardware Design Language, we will now dive deeper and implement a ring oscillator

- **We will use a ring oscillator to generate True random numbers**
- **We will program the board using quartus**
- **You will be able to use the sample button to generate a 10 bit TRNG, which should display the bits on the LEDs**

Part 2: TRNG with Ring Oscillator

Why Ring Oscillators?

- Ring Oscillators are just composed of multiple inverters sequenced together
- if you have multiple ring oscillators running at different lengths, physical “glitches” occur from something called metastability
- Read here to learn more about metastability: What is metastability and what are its effect? | vlsi4freshers
- You can then XOR the outputs to obtain the random number



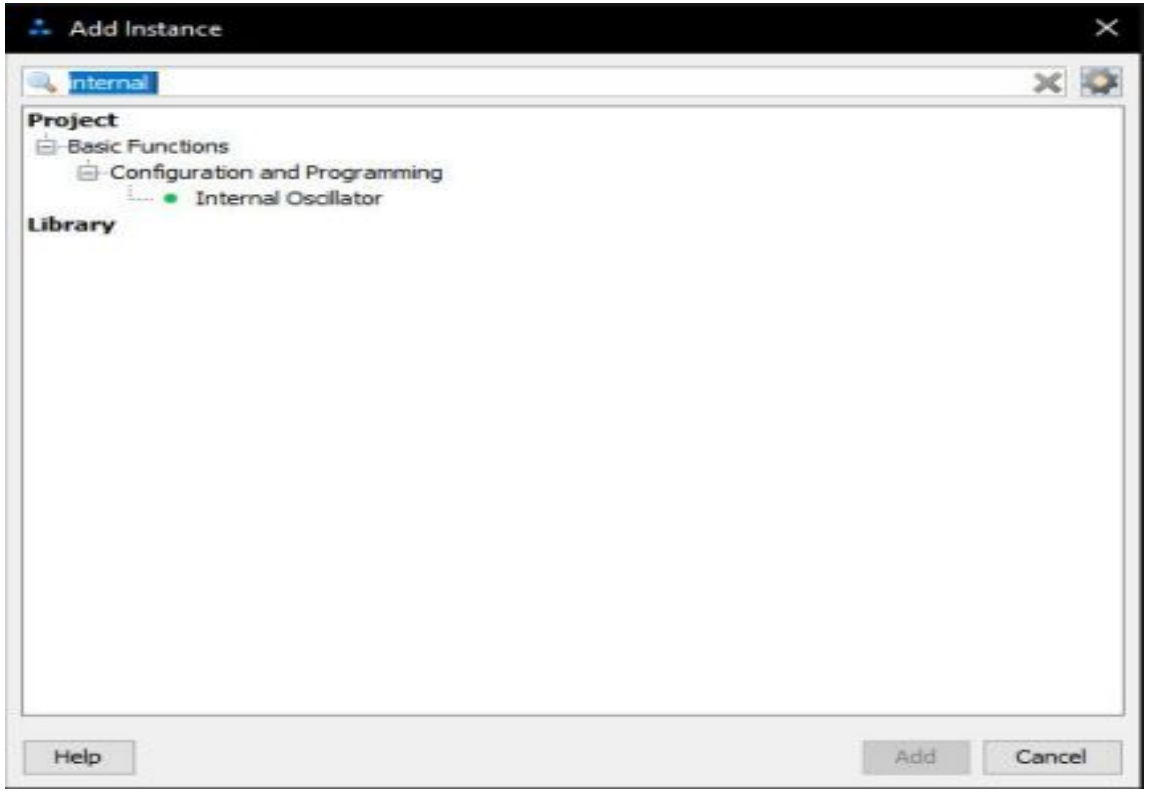
- Download all files for lab 4 from moodle
- Create a new project called RO_trng
- Add the verilog file provided to the project when prompted to add files (The file is called trng in the lab 4 folder)
- Alternatively, you can create a new VHDL file file-> New and select VHDL file under design files and name it TRNG.vhd
- Copy the contents of the verilog file provided into the new VHDL file you created (use trng text file for copy/paste)

Step 1: Create a Project

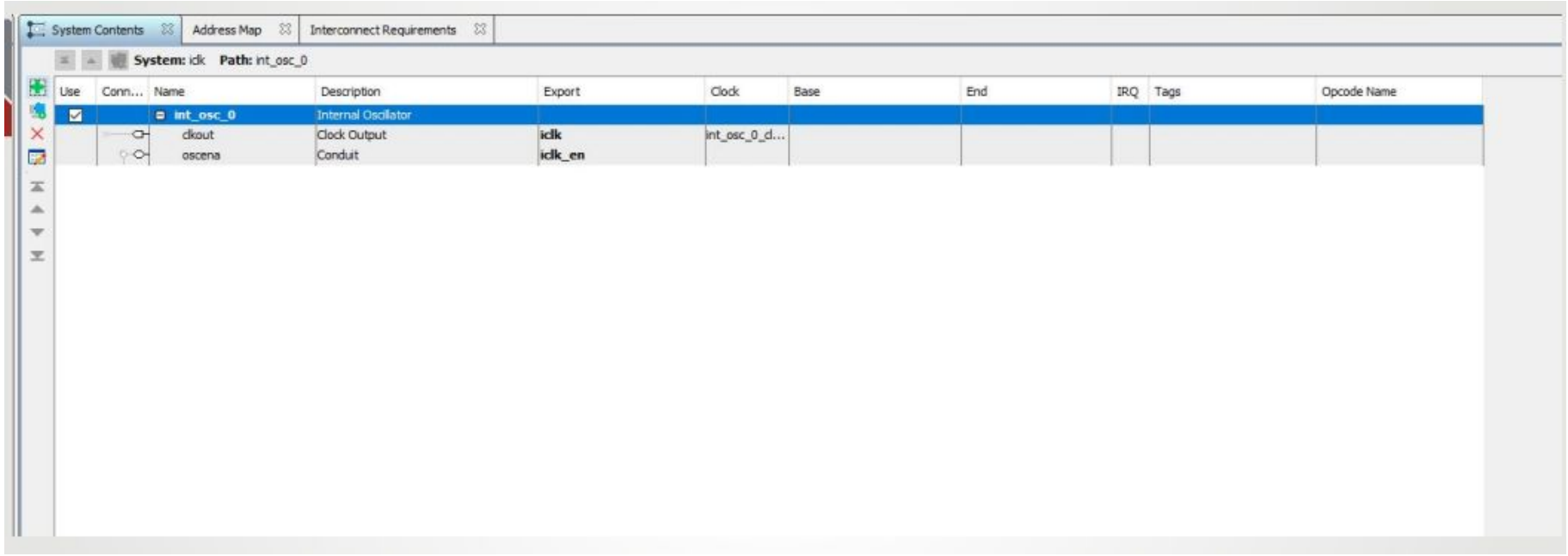
Step 2: Using the Internal Clock

You will need to setup the internal clock so that you can use it the program

- **To do this, go to Tools-> qsys**
- **Delete all components in the default qsys file**
- **We will need a single component in your qsys file**
- **Add a new component called ‘internal Oscillator’**
- **Like the image on the next slide**



- Once you add the component, you need to double click for export
- Note: you can use the default name, but I have changed it to make it a little more convenient



- **Your Qsys window should look something like this.**
- **Once you are done, you are ready to generate the HDL file**
- **Once the HDL file is generated click finish**

- Right click on the files window (it should currently be on hierarchy- the window on the left)
- Find the place where the qsys file was stored and add the file • Note: This should be similar to lab 1
- You will also need to open the iclk_inst.vhdl file in notepad and copy paste the code in the TRNG.vhdl file in quartus
- Note: iclk is the name I defined in the export (in Qsys), if you have named it something else, the vhdl file will be called that

Step 3: add qsys file to the project

- Just to clarify where the code from your iclk_inst.vhdl should be placed. (refer previous slide)
- Place the ‘component’ code under architecture (as seen below)
- Also place the ‘u0: component’ after the first begin

```
  component iclk is
   port (
      iclk_clk        : out std_logic;         -- clk
      iclk_en_oscena : in  std_logic := 'X');  -- oscena
end component iclk;
```

```
begin
   u0 : component iclk
      port map (
         iclk_clk       => clk_int,
         iclk_en_oscena => '1');

  assert ring'length mod 2 = 1 report "Length of ring must be an odd number!" severity failure;
```

Step 4: Adding the Components

- The signal clk_int is already declared for you.
- In your ‘u0 : component” you need to assign iclk_clk to clk_int and assign iclk_en_oscena to ‘1’ as seen in the previous slide • Make sure you understand the code
- Look very carefully at how the ring oscillator is implemented
- Note: the ring oscillator uses a single inverter gate
- You should be ready to compile the design

Step 5: Assigning Pins

- Assign the first 10 bits of the trng to the LEDs (trn[0] to trn[9])
- You will also need to assign two push buttons for the sample and reset
- This should be similar to the instructions on the LFSR part of the lab
- The pins can be found in the manual, I have provided a link below https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1004282204-de1-soc-user-manual.pdf
- Note: You will need to recompile the program for the pins to be assigned

- Again programming the board is similar to the LFSR part of the lab
- You will need to go to Tools-> programmer
- Auto detect the device and change the sof file to the TRNG.sof file
- Look back at the LFSR part of the lab if you are confused
- Click start to program the device
- You can now hit the sample button to see a 10 bit random number displayed on the LEDs

Step 6: Programming the Board

- Now you have successfully built a single Ring oscillator that outputs a 21 bit TRNG, out of which you are displaying the first 10 bits on the LEDs
- Note: The ring oscillator is running continuously in the background and the sample button simply takes the first 10 bits of the ring oscillator when the push button is clicked and displays it on the LEDs

- Incorporate another ring oscillator using the external clock
- The clk is already added for you within the ‘trng’ entity , it just needs to be called in the process and needs to be assigned to the respective pin for a 50 mhz clock similar to the clock we use in the LFSR part of the lab
- You then need to xor the outputs of the 1st ring oscillator with the 2nd ring oscillator and store it in the ‘trn’ vector

Objective

Deliverable

- We will only conduct a demo of your code with two ring oscillators

- Make sure to understand how the randomness is created and why

Good Luck!

University of Massachusetts Amherst