KNIGHT RIDER

PRAC 4: KNIGHTRIDER







EEE4084: Digital Systems University of Cape Town

Prepared by M. Bridges & S. Winberg

The aim of this tutorial is to introduce you to Xilinx ISE and the Digilent Nexys2 FPGA platform. You will be taught to create VHDL & Verilog Modules as well as Schematics. You will also be taught how to simulate your designs using both VHDL and Verilog.

The Design that we will implement in this project is based on Knightrider project form the Square Kilometres Array South Africa Organisation. It is implemented on their ROACH 2 Platforms used at the site of KAT-7. You are given a precompiled bit file pulsetrain.bit to try out beforehand.

There is no report hand-in for this assignment. You will need to hand in your VHDL and Verilog code and you will need to demonstrate a working solution to your tutor.

Overview of prac

The breakdown of stages for this prac is outline below. You can use this table of contents as a checklist for having completed the necessary steps involved.

Contents

Overview of prac	2
Introductory Steps	3
STEP 1: Connecting up and initial test	3
STEP 2: Initial testing using the Adept software	4
Prelude to developing a simple FPGA application	4
Creating FUNC1: the first part of the Nexys2 Pulse Train	5
STEP 3: Starting up ISE and configuring a new project	5
STEP 4: Creating a new VHDL code file and VHDL entity	7
STEP 5: Write the VHDL code for func1	8
STEP 6: Simulating func1	9
Step 7: Running func1, where the fun begins1	1
STEP 8: Programming via USB port using Digilent Adept Software1	2
STEP 9: Testing func1 on hardware1	2
Creating FUNC2: the second part of the Nexys2 Pulse Train1	3
STEP 10: Creating a new Verilog code file and Verilog module1	3
STEP 12: Write the VHDL code for func11	4
STEP 13: Simulating Func21	5
STEP 14: Running Func21	6
Creating Pulse train: the final part of the Nexys2 Pulse Train1	7
STEP 15: Creating a new Schematic file: Pulse1	7
STEP 16: Port-mapping to create the pulse train1	9
STEP 17: Running Pulsetrain	0

This prac uses the Creative Common License: Attribution 3.0 Unported



This prac is available free for use and reuse without written consent from the lecturer; see the <u>EEE4084F</u> <u>Resource Use Policy</u> for more details on this use policy.

Introductory Steps

The Nexys2 has several input devices, output devices, and I/O ports (or expansion connectors). There are eight slide switches and four pushbuttons for input experiments, and for experimenting with outputs, there is the 4-digit 7-segment display and eight green LEDs. See the Nexys2 Reference Manual from Digilent for further details (file: Nexys2_rm.pdf).

This tutorial focuses on the pushbuttons, switches and the LEDs.

STEP 1: Connecting up and initial test

You should each have received a Nexys2 kit. Inside the kit should be two items:

- 1) A Nexys2 circuit board itself as shown in Figure 1 below
- 2) A USB extendable connector

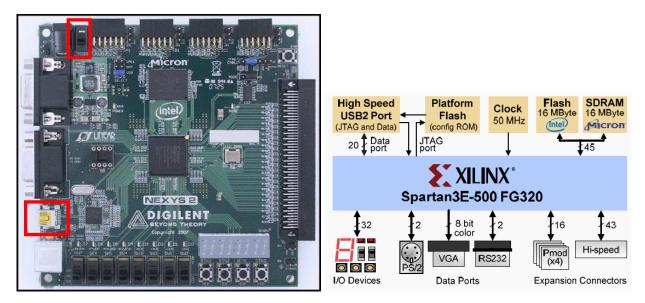


Figure 1: The Nexys2 board (left) and its component composition (right).¹

Connect up the Nexys 2 to your PC (see red boxes in figure 1). There is no need to connect an external power supply as the board is powered via USB. At this stage, a pop-up message will likely occur to indicate you have connected new hardware, and shortly should indicate it is installing device drivers for "Digilent Onboard USB". (If you don't get this message, you'll need to unplug the board, install the Digilent Adept software, reboot and try again).

If all is well, then the red power LED should light up on the board (and possibly other LEDs depending on what software if any is running on the board). If not, **MAKE SURE POWER SLIDE SWITCH IS UP** (see red boxes in figure 1).

Try to familiarise yourself with the platform, at this stage you should be in a position to recognise the ports and make logical assumptions as to what the chips on the board do. The schematics and reference guide for the board is available from the Digilent website as well as many examples that will be useful when you begin your projects. http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2

¹ Images adapted from Nexys2 reference manual, 2010.

STEP 2: Initial testing using the Adept software

On your PC, click start \rightarrow programmes \rightarrow Digilent \rightarrow Adept \rightarrow Adept (i.e., start the Adept program). You should now see a screen as shown below. Select the **Test** tab.

Digilent Adept	Digilent Adept
Connect: Nexys2 V Product: Nexys2 • 500	NEXYSE Connect: Nexys2 - 500
Config Flash Test Register I/O File I/O I/O Ex Settings	Config Flash Test Register VO File VO VO Ex Settings
FPGA Image: Program PROM Image: Program VD704S Image: Program	Test FAM O Test Flach O Simiches Button O Start Test Stop Test
Initialize Chain	Found device ID: #004603 Found device ID: #0422993 Initialization Complete. Device 1: X125500E Device 2: X127945

Press the **Start Test** button in the Adept program. If your board is working correctly, the green LEDs should light up, and the 7-segment display should flash the messages '128' and 'PASS'. Go ahead and press the pushbuttons, and the corresponding circles shown in adept should indicate the buttons pressed. When changing the slide switches, the switches display on the window should change (and the message on the 7-segment display will stop). Once you are satisfied that your board is in good order, close the Adept program and continue to step three.

Comment from lecturer: You probably think this first step is rather trivial and unimportant. But in a real project you may well find otherwise. I'm sure you can think of some development cost risks associated with using a development board without checking that it is working properly.

Prelude to developing a simple FPGA application

In order to get a FPGA to do useful operations it needs to be configured. This process involves generating a 'bit file' using a design tool, and then transferring this bit file into memory that resides within the FPGA, and which defines the logic element interconnects. These logical elements, if you didn't listen properly during the lecture, are the individual lookup tables, gates and other individual processing elements which are connected together to do higher level processing operations. Free ISE Web Pack software is available from Xilinx that allows a developer to describe the hardware design to be implemented on the FPGA. There are different ways in which this can be done, including the use of Hardware Description Languages (HDL) such as VHDL or Verilog, or using graphical schematic diagrams. These can be combined together very effectively, allowing large hierarchical designs with multiple engineers using multiple languages.

In Gateware design, an engineer will often need to make use of all three of these, and it is expected that you will know all of them.

Creating FUNC1: the first part of the Nexys2 Pulse Train

STEP 3: Starting up ISE and configuring a new project

Regardless of what application you are going to create for the Nexys 2, the first few steps are likely to be the same for each new project. Follow the steps below to create the project. In order to work smart in the figure, it is a good idea to create a template project (i.e. save the project under Tut1 and again as Nexys2Template at the end of this step – then return to changing Tut1).

Start by loading Xilinx ISE. You probably already have a Xilinx ISE Design Suite icon on your desktop that you can click to load ISE. This can alternatively be done by Start \rightarrow Programs \rightarrow Xilinx ISE Design Suite 13.4 \rightarrow ISE Design Tools \rightarrow 64-bit Project Navigator (or obviously select the 32-bit version if you are running on a 32-bit architecture). It may take a while if you are on a slow machine; 2GB or more RAM is recommended.

Licensing problems? If there is a license problem, please call one of the tutors to help out.

Create a New Project in ISE by selecting from the menu File \rightarrow New Project...

Create New Pro		
Enter a name, locatio	ns, and comment for the project	
Name:	Prac4	
Location:	C:\Xilinx\Projects\Prac4	
Working Directory: Description:	C:\Xilinx\Projects\Prac4	
Select the type of to Top-level source typ HDL	o-level source for the project e:	

The screen shown on the above should appear. Fill out the entries as illustrated and click next. Choose 'HDL' for you top-level source type².

WARNING: you do not want any spaces, dots, commas or other funny characters in the file names or directory names, as this tends to lead to compile errors.

² In future projects, you may likely want the top-level source type to be 'schematic'. This is what I typically tend to choose as I find it gives a more meaningful overview of the system being built.

The next step is to specify to ISE what system you are targeting. Start by selecting Spartan3E as the family and then make the selections as shown in the figure below. Press next when done.

Project Settings		
Specify device and project properties. Select the device and design flow for the pr	niert	
Property Name	Value	
Evaluation Development Board	None Specified	
Product Category	All	
Family	Spartan3E	-
Device	XC3S500E	
Package	FG320	-
Speed	-4	-
Top-Level Source Type	HDL	-
Synthesis Tool	XST (VHDL/Verilog)	-
Simulator	ISim (VHDL/Verilog)	
Preferred Language	VHDL	-
Property Specification in Project File	Store all values	-
Manual Compile Order		
VHDL Source Analysis Standard	VHDL-93	-
Enable Message Filtering		

The next screen will show a summary of the settings. Click Finish to accept the settings and to create the new project. You should now get the main project screen as shown below. (At this stage, you could make a copy of the project using File \rightarrow Copy Project to make a backup starting point for future project; but for this tutorial, you can skip this step and proceed on).

JSE Project Navigator (0.87xd) - C:\Xilinx\Projects\	prac4\prac4.xise - [ISE Design Suite InfoCenter]	
File Edit View Project Source Process Too		- 5 >
∽ ~ X @ @ X ∽ ~ .	୬୬୫୫୬≧ 🔀 ╚금⊟ 🖻 ୬₭? ► Σ 🛠 💡	
Perior ↔ □ 0 × □ 0 × □ Wex:	Release Overview Design Resources Documentation Project Havigator ISE Design Suite 13 Release Overview ISE Design Suite 13 Release Overview ISE Design Suite Highlights: Introducing the 7 Series Device Support Begin designing in Kintex[™] -7 and Virtex®-7 devices today Includes Intelligent Clock Gating to reduce dynamic power Team Design Flow Improved productivity in embedded design, implementation design flow, and verification. IEEE P1735 encryption flow for simulation inter-operability Cadence@ AXI Bus Function Model (BFM), sold separately, to verity your AXI4 IP IP Updates: Click here for the IP Release Notes library updates and new products. 	E
Start Console	Resources Step Design Suite: Release Notes Guide (Including What's New in Xilinx ISE Design Suite Chapter) ISE Design Suite: Installation and Licensing Guide Known Issues in the Xilinx ISE Design Suite 13 Release ISE Design Suite InfoCenter	•□8
TT TT	es Results	

STEP 4: Creating a new VHDL code file and VHDL entity

Create a new code file by selecting Project \rightarrow New Source and then select VHDL Module, type in the name 'func1' for the module, and then click next.

Select Source Type Select source type, file name and its location. P (CORE Generator & Architecture Wizard) Schematic User Document Verilog Test Fixture Verilog Test Fixture VHDL Module VHDL Module VHDL Package	
VHDL Test Bench Embedded Processor	File name: func1 Location: C:\Xilinx\Projects\prac4

You should now get a screen where you can specify the ports for your VHDL block. (It is optional to fill in port details at this stage, as it can be done later in the code itself, but using this window is probably quicker). Fill out the ports as illustrated below.

Specify ports for							
Entity name							_
Architecture name	Behavioral						
	Port Name	Direct	ion	Bus	MSB	LSB	
clk		in					
ce		in	-				
rst		in	-				
q		out	-		7	0	
		in	-				
		in	-				
		in 📕	-				
		in	-				
		in					
		in	-				
		in	-				
		in	•		1		

STEP 5: Write the VHDL code for func1

Using the func1 module create an 8-bit Up Counter that wraps round to zero after reaching its max count. The counters output should only increment after a fixed number of clock cycles assigned by the value of the generic DELAY. See code below. In a sense, we are dividing the clock such that for a 100Hz clock we only want a counter that changes at 1HZ so we delay for 99 clock cycles and then perform our function.

Sequence is: 00000000, 00000001, 00000010, 00000011, ..., 11111110, 11111111, 00000000, 00000001

Note1: It is very important that you factor in 'ce' and 'rst'.

• Only count **when ce=1** and **if rst=1** then q=00000000.

Note2: The counter should only count after a number of cycles determined by the generic DELAY.

• Only count when pre_count=DELAY.

```
_____
library IEEE;
use IEEE.STD LOGIC 1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity func1 is
      generic (
             --This is used so that our program runs in Human Speed.
             --We will alter this Delay throughout the Prac, so be sure to leave
             --this line as is, until otherwise specified.
             DELAY : unsigned (23 downto 0) := x"000004"
      );
      Port (
            clk : in STD LOGIC;
            ce : in STD_LOGIC;
            rst : in STD LOGIC;
             q : out STD LOGIC VECTOR (7 downto 0)
      );
end func1;
architecture Behavioral of func1 is
-- tmp signal is register to store calculated q value. Initial value is HEX "00"
signal tmp: unsigned(7 downto 0) := x"00";
-- pre count is used to for generating the delay. Initial value is HEX "00...00"
signal pre_count: unsigned(23 downto 0) := (others => '0');
begin
      process(clk, ce, rst)
      begin
             --TODO: This is where your counter goes.
      end process;
      -- Unsigned type is converted back to std logic vector
      q <= std logic vector(tmp(7 downto 0));</pre>
```

STEP 6: Simulating func1

Now that you have designed your func1, it is time to test that it has been implemented correctly. The best practice for FPGA development is to simulate *first* and then, once the simulation looks right, implement on hardware. We will use ISIM, which is Xilinx ISE's built-in HDL simulator.

Change to ISE simulate mode: enable ISIM (i.e., ISE Simulator) by clicking on the option button shown by the red arrow in the figure on the right.

This will cause the Process panel to change.

Now a simulation test bed needs to be created.

Select Project \rightarrow New Source...

Select VHDL Test Bed, and then specify an appropriate file name, such as func1_tb (see below). Click next and then choose func1 as the VHDL file associated with the test. Click next and then click Finish.

> New Source Wizard		New Source Wizard	×
Select Source Type Select source type, file name and its location. Select Source typ	File name: func1_tb Location: C:\Xillinx\Projects\prac4	Associate Source Select a source with which to associate the new source.	
More Info	Next Cancel	More Info Next	Cancel

Open up the automatically generated test bench file, as per the screen shot shown below (if needed, double click 'func1_tb – behaviour' in the Simulation hierarchy panel).

>	ISE	Project N	lavigator	· (0.87xd) -	C:\Xilinx\	Projects\	orac4\pr	ac4.xis	se - [fun	c1_tb.vhd	nd]										
	Fi	le <mark>Edit</mark>	View	Project	Source	Process	Tools	Wi	ndow	Layout	Help	6									
i C)	2	a 5	X C		50	»	Æ	PB	Ø /	2		<u>в</u> в		ъ .	P 12?	▶	21	e	V	
		View: O Behavioral Hierarchy C pr C pr C pr C pr C pr C pr C pr C pr	imple ac4 3s500e funcl im with uu	ementation 4fg320 tb - behat rt - func1 - Running	↔	I P × mulation	© © ★ ★ ★ III2 IIII II2 III IIII III	28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45	LIBH USE U 8 US ENTI END	RARY is isee.s Jncomme arithme SE isee HITY fur func1_ HITECTU Con COMPON PORT(COMPON	Leee; std_l ment th metic second ancl_th L_tb; TURE bo pompone: DNENT	ogic funct aeric b IS behav: ent De func:	_1164 ollow: tions _std.1 ior Ol eclar: 1	.ALL; ing] witP ALL; F fur atior	; h Sig nc1_t n for	ary c gned tb IS	decla or U	aratio Insign	on i: ned 1	f using values Test (UU Impo	rts
		Processes	ISim Si	tb - behav mulator navioral Cl nulate Beh	neck Synta			45 46 47 48 49 50 51 52 53 54 55	5	I	rst : q : O ; COMPON its its iclk icclk	IN DUT IENT; : sto std	std_1 std_1c d_log: _logic	logic ogic_ ic :=	c; _vect = '0';		7 dow	nto O	" }	func1 comp	as a

6

G

0

¢5

ims

38

(A)

ntation 🧿 🐖 Simulation

└─ 🔄 Tut1 └─ 🛄 xc3s500e-4fg320 └─ 强 func1 - Behavioral (func1.vhd)

View: 🔘 🔯 Imple

Behavioral

Hierarchy

-

.

2

The func1_tb.vhd provides a baseline starting point for VHDL code that describes how the func1 entity is to be tested. If you look closely, you will notice that func1 is defined as a COMPONENT in this file (see the marked section in diagram above). This means that func1 is essentially being imported to the funct1_test entity as a 'black box' for which its ports are going to be exercised in various ways.

Have a look through the func1_tb file. You will notice that various signals are initialized which will be connected to the inputs and outputs of func1; specifically 'clk', 'ce' and 'rst' are set to 0 (line 52).

At line 61 the clock speed is specified as 10ns. To make the clock 50 MHz, change this line to:

```
constant clk_period : time := 20 us;
```

At line 66 an instance of func1, called UUT (for 'unit under test') is instantiated. The clock process, i.e. how it changes state, is specified as a process on line 74.

The main testing procedure is described by the 'stim_proc' (for stimulus process) on line 84. This process waits 100ns (to cover any initialization needed) then waits another 10 clock pulses (you can change this if impatient) and then performs any other stimulus processes. For example, let us carry out the following test:

- 1. Enable the chip: Set the line ce high
- 2. Wait 600us
- 3. Reset the chip: Set the line rst high.

To implement this, copy and paste the following VHDL code at line 90 in func1_test.vhd:

```
-- insert stimulus here
ce <= '1';
wait for 600 ns;
rst <= '1';
wait for 100 ns;
Save the file and then...</pre>
```

Double-click 'Behavioral Check Syntax', which is on the left under 'Isim Simulator'. The syntax check should be successful. Then double-click on 'Simulate Behavioral Model'. At this stage, the ISim application is loaded. The ISim application should now be started, as shown in the figure below.

And Fil	le Edit View Simula	ation Window Layout	Help			7				- 5
	🖻 🗟 🕹 🖌 🖻	🕅 🗠 🔁 🔕 🗶 🗊		6809	PR P1	8 / 2	* *	1 0 1		X
les 🔹	(B)							140.000 ns		
Source Files	Name	Value	l0 ns	100 ns	200 ns	300 ns	400 ns	n 2000 -	500 ns	600 n
		0								
-	Ce La ce	1								
and Processes	📀 🛛 🖟 rst	0								
oce	🕥 🕨 式 q[7:0]	0000001	(00000	000	X	0000	0001	00000010	000
ž l	👍 🛛 🕌 clk_period	20000 ps			2	0000 ps				
	*r									

If you do not see the signal changes as on the right hand side of the diagram above, then the signal view is probably at the wrong zoom level. Fix this by right clicking on the black waveform display, and select To Full View (or press F6).

From the waveforms on the right, the clock is going at 50MHz. The value of q should count up after a number of clock cycles set by the value of your delay. You can change the radix of q by right clicking on it.

Step 7: Running func1, where the fun begins

At this stage, we are now ready to implement and run our design. Switch back to implementation mode and open your func1.vhd file. The first thing we need to do is convert our design into human time, since we would otherwise be unable to recognise the result. To do this we change the generic DELAY as shown below:

DELAY : unsigned (23 downto 0) := x"FFFFFF"

The next step is to set up the pin assignment for the FPGA. You are given an edited version of the Master UCF file from Digilent. You can import this as is into your project as shown on the left.

Make sure that func1 is set as your top-level module (i.e. it has three little boxes next to it name) and that the UCF files is associated with it. It should look the same as in the picture above.

Now double click 'Synthesize' in the Process panel and wait for it to complete, then 'Implement' and wait for that to complete. You should be getting green ticks if you done it correctly. The final step is to generate a bit file to upload to the board.

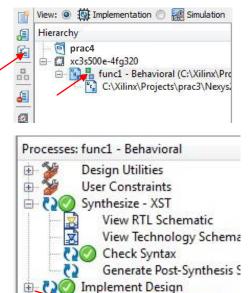
First you need to change the start clock setting for the FPGA since we want it to start once the JTAG programming has completed (rather that a reset that would clear the memory we set).

Change the clock by right clicking on Generate Program File icon in the
Process panel and then select Process Options.

General Options	-g StartUpClk:	FRCA Sheet Up Cheat	Telephone concerns and the
Configuration Options	2	FPGA Start-Up Clock	JTAG Clock
Startup Options	-g DonePipe:	Enable Internal Done Pipe	
Readback Options	-g DONE_cycle:	Done (Output Events)	Default (4)
	-g GTS_cycle:	Enable Outputs (Output Events)	Default (5)
	-g GWE_cycle:	Release Write Enable (Output Events)	Default (6)
	-g LCK_cycle:	Wait for DLL Lock (Output Events)	Default (NoWait)
	-g DriveDone:	Drive Done Pin High	100
	Proper	ty display level: Standard 💌 🗸 Disp	play switch names Defa

Select Startup Options from the diagram that pops up (see diagram on right). Then change the FPGA status clock setting from CCLK to JTAG Clock. Click Apply and then OK.

Now create the bit file by double clicking 'Generate Programming File in the Process' panel. Again, you should get a green tick, if so you have successfully implemented your first design. The next step will show you how to load your design, or more correctly Programme the FPGA, on the Nexys2.



Generate Programming File

Files

10

Configure Target Device

Et Design

(II)

sole

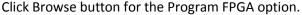
Start

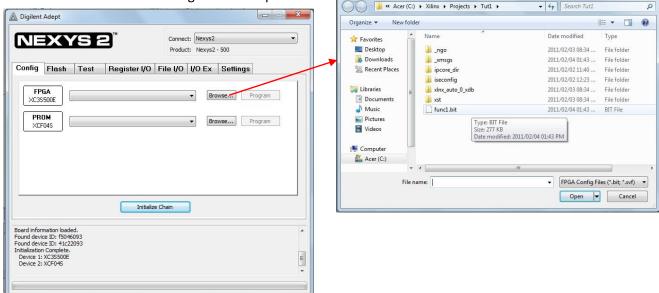
STEP 8: Programming via USB port using Digilent Adept Software

At this stage, you are ready to load the bit file into the FPGA and run it. The Nexys2 should still be plugged into your computer at this point.

Start by loading the Adept program (as used at the start of this tutorial).

Select the Config panel as shown in the diagram on right.





A Open

The bit file will reside in location C:\Xilinx\projects\prac4 if you used the path as suggested at the start of this tutorial.

Once you have selected the bit file, press Program.

If all went well, you should get a success message at the bottom of the Adept window:

Device 2: XCF04S Set Config file for XCS500E: "C:\Xilinx\Projects\Tut1\func1.bit" Preparing to program XC3S500E	*
Programming Verifying programming of device Programming Successful.	u v
8	

Note that there is no need to program the PROM (doing so will change the program that executes on startup – although the Nexys2 design is robust, it's best practice to program the PROM with a 'trusted' bit file that has been tested in memory to avoid chances of corrupting IO, FLASH memory, etc).

STEP 9: Testing func1 on hardware

Since the 'ce' line was assigned to the pin connecting to SWO on the board, you need to set this high to enable your function. The LEDs should reflect the output of your counter remember the output is binary, i.e. 00000000, 00000001, 00000010, 00000011, etc.

To reset your counter you can press BTNO, which is the assignment for 'rst', and it should return to zero.

×

Creating FUNC2: the second part of the Nexys2 Pulse Train

STEP 10: Creating a new Verilog code file and Verilog module

The next section involves creating 'func2', which is a Verilog module similar to the counter, but instead performing a shift operation. All the steps are mostly the same as for the 'func1' just using Verilog instead of VHDL.

Create a new code file by selecting Project \rightarrow New Source and then select Verilog Module, type in the name 'func2' for the module, and then click next.

You should now get a screen where you can specify the ports for your Verilog Module. Fill out the ports as stated below, or the same as in the func1 module.

		-			
					_
New Source Wizard					
P.C. M.LL					
Define Module					
Specify ports for module.					
1odule name func2					
Port Name	Directio	on	Bus	MSB	LSB
clk	input	•			
ce	input	-			
rst	input	-			
q	output	-	V	7	0
	input	-			inte e
	input	-			
	input	-			
	input	-			
	input	•		0	
	input	-			1
	input	-			

Click next and then finish.

STEP 12: Write the VHDL code for func1

Using the func2 module create a bi-directional shifter function that matches the sequence shown below.

What we do is take a register "000000000000000011111111", shift it left 16 places to get

"1111111100000000000000000" and then shift it right 16 places. Then repeat. To get our output we look at the middle 8 bits. You can use the algorithm given below in the code if you would like.

The counters output should only increment after a fixed number of clock cycles assigned by the value of the generic DELAY. See code below.

Note1: It is very important that you factor in 'ce' and 'rst'.

• Only shift when ce=1 and if rst=1 then q=00000000.

Note2: The shifter should only shift after a number of cycles determined by the generic DELAY.

• Only shift **when pre_count=DELAY**.

```
`timescale 1ns / 1ps
_____
module func2(
    input clk,
    input ce,
   input rst,
   output [7:0] q
   );
      // This is used so that our program runs in Human Speed.
      // We will alter this Delay throughout the Prac, so be sure to leave
      // this line as is, until otherwise specified.
      parameter DELAY = 24'h000004;
      // tmp is the register that we will be shifting
      reg [23:0] tmp = 24'h0000FF;
      // pre count is used to for generating the delay. Initial value is HEX "00...00"
      reg [23:0] pre_count = 24'h000000;
      // shift_count is used to keep track of where we are in the shift operation
      reg [4:0] shift count = 5'h00;
      always @(posedge clk) begin
            // TODO: This is where your shifter goes.
             // This code can be used to perform your shift
             /*if(shift count > 5'hOF) begin
                   tmp <= tmp >> 1'h1;
                   shift count <= shift count + 1'h1;</pre>
            end
                  else begin
                   tmp <= tmp << 1'h1;</pre>
                   shift count <= shift count + 1'h1;</pre>
            end*/
      end
      //Output the register value to the port
      assign q = tmp[15:8];
endmodule
```

STEP 13: Simulating Func2

Create a new code file by selecting Project \rightarrow New Source. Select Verilog Test Fixture, type in the name 'func2_tb' for the module, and then click next and finish.

The test bench code for the 'func2' is given below you can just copy and paste this, or better read through it and then add snippets to your file. You should be able to notice the similarities between the VHDL test bench we implemented for 'func1'.

```
`timescale 1ns / 1ps
```

```
module func2 tb;
      // Inputs
      reg clk;
      reg ce;
      reg rst;
      // Outputs
      wire [7:0] q;
       // Instantiate the Unit Under Test (UUT)
      func2 uut (
             .clk(clk),
             .ce(ce),
             .rst(rst),
             .q(q)
      );
       initial begin
             // Initialize Inputs
             clk = 0;
             ce = 0;
             rst = 0;
             // Wait 100 ns for global reset to finish
             #100;
             // Add stimulus here
             ce = 1;
             // Wait 100 ns for global reset to finish
              #800;
             rst = 1;
             // Wait 100 ns for global reset to finish
             #100;
       end
       // oscillate clock every 10 simulation units
      always #20 clk <= ~clk;
```

endmodule

When you are happy with your test bench code, double-click 'Behavioral Check Syntax'. The syntax check should be successful. Then double-click on 'Simulate Behavioral Model'. At this stage, the ISim application is loaded again.

STEP 14: Running Func2

Switch back to implementation mode and open your 'func2.v' file. The first thing we need to do is convert our design into human time, since we would otherwise be unable to recognise the result. To do this we change the **parameter DELAY** as shown below:

parameter DELAY = 24'hFFFFFF;

Make sure that func2 is set as your top-level module (i.e. it has three little boxes next to it name) and that the UCF files is associated with it.

Synthesize.

Implement.

Generate Bit File.

Programme the Board. You can refer to Step 8 if you have forgotten how to do this.

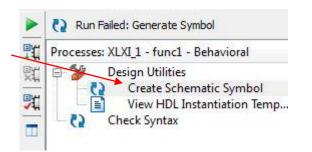
Creating Pulse train: the final part of the Nexys2 Pulse Train

STEP 15: Creating a new Schematic file: Pulse

You are given a VHDL module from the Digilent website called PWM. If you are interested, the Datasheet is also included for this module. Add a copy of this to your project by clicking on the 'Add Copy of Source' button.

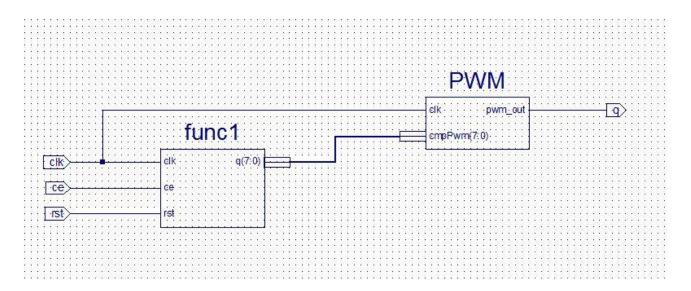
Select the PWM module and then double click on the Create Schematic Symbol. Do the same for the 'func1' module. Again, you should get green ticks.





Now create a new code file by selecting Project \rightarrow New Source. Select Schematic, type in the name 'pulse' for the module and then click next and finish.

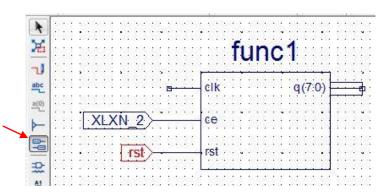
Select Source Type Select source type, file name and its location. BMM File ChipScope Definition and Connection File Implementation Constraints File IP (CORE Generator & Architecture Wizard) MEM File Schematic User Document Verilog Module Verilog Test Fixture VHDL Module VHDL Library VHDL Library VHDL Test Bench Embedded Processor	File name: pulse Location: C:\Xilinx\Projects\prac4
---	---



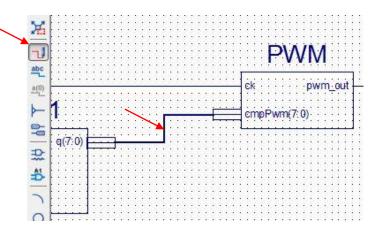
The simple Schematic design is shown above. You need to implement this yourself keeping all the names ports and connections the same. There are three items you will need to add to your design; namely symbols, wires and ports.

Firstly, add the symbols from the panel on the left and place them in roughly the same orientation as shown in the example schematic.

Then add the 4 ports by selecting the port button and clicking on the box. The polarity of the port will be determined automatically but you will have to right click on the port and select rename to give it a logical name. You can move the port around by drag and drop.



Finally, use the wire tool to connect the Symbols together as shown below. Check you schematic against the example above and **SAVE**.



bols	⇔⊡₽×
Categories	
<all symbols=""></all>	
<c:\xilinx\projects\prac3></c:\xilinx\projects\prac3>	122
Arithmetic	H
Buffer	
Carry_Logic	
Comparator	
Counter	-
PWM func1 func2	
mbol Name Filter	
otate 0	•
Symbol Info	1
Symbol 1110	

STEP 16: Port-mapping to create the pulse train.

The Final step of this practical involves putting all the pieces together and then admiring your handiwork.

You are given a VHDL module called 'pulsetrain.vhd'. Add a copy of this to your project by clicking on the 'Add Copy of Source' button.

This module consists of the entity declarations of the other designs we created, a few signals to interconnect and finally the port mapping of the entities. There are eight LEDs on the board so we want to declare eight instantiations of the pulse entity: p0, p1, ..., p7. The first and last of these instantiations have been done for you, but you will need add the ones in between.

);

STEP 17: Running Pulsetrain

And our Design is finally complete...

Change your timings in func1 and func2 to the values below. This will give us cool looking LED pattern. To keep the two function in sync func1 needs to run 32 times faster than func2.

Func1.vhd

DELAY : unsigned (23 downto 0) := x"01FFFF"

Func2.v

parameter DELAY = 24'h3FFFFF;

Make sure that 'pulsetrain' is set as your top-level module (i.e. it has three little boxes next to it name) and that the UCF files is associated with it.

Synthesize.

Implement.

Generate Bit File.

Programme the Board. You can refer to Step 8 if you have forgotten how to do this.

THE END