*Swinburne University of Technology*

***Faculty of Engineering Computing and Science***

***Xilinx ISE V14.7 Development System Introduction (CMOD hardware)***

**Aim**

To introduce the student to the Xilinx ISE tool for the implementation of hardware in Xilinx CPLDs. This tool will be used in later experiments and project work to develop and debug VHDL designs. The student will also be introduced to the hardware platform being used.

**Assessment - This exercise has no assessment.**

# Introduction

The Xilinx ISE tool supports the development of schematic and hardware design language (HDL) based designs intended for downloading into Xilinx FPGAs and CPLDs. This tool will be used to develop a design for a module that incorporates a Xilinx XC2C64A CPLD. The hardware will implement an electronic die that will produce a “random” roll when a button is pressed. The design consists of a counter, decoder and a top-level structural VHDL description that is used to “wire together” these modules.



# Design Flow

A simplified design flow for the Xilinx tool is shown in the diagram below:



In the case of our designs we will be using VHDL as the design entry method. As shown in the diagram above, the VHDL description is passed to the internal tool *Xilinx Synthesis Tool* (XST), to do the actual VHDL synthesis. The result of this synthesis is then passed to the Xilinx place & route tool. This means that it is also possible to open the intermediate design to examine the synthesis results. The transfer of design information between the synthesis and place & route tools is handled transparently by ISE.

# Experiment Procedure

Please completely read each step before attempting it.

|  |  |
| --- | --- |
| * *Start the software* from the Start Menu or Desktop Shortcut * ***Start🡪All Programs🡪Xilinx Design Tools 14.7🡪ISE Design Tools🡪 64/32 Bit Project Navigator (depending on your CPU Architecture)*** | |
| * *Create a new project* by choosing ***File->New Project*** from the menu. Or click new project in the Welcome menu.   The resulting dialogs allow us to select the CPLD device family and particular device within the family as well as the tool-set to be used. | |
| The first dialog locates the project directory where all the project files will be located. The directory name will be the same as the project – In this case ISEIntro.  The second dialog determines the device characteristics.  The CMOD board contains a Coolrunner2 CPLD (XC2C64A) in a VQ44 package with a speed grade of 7.  This device has 64 macrocells as described in lectures (not bad for about AU$10).  We will be using Xilinx’s synthesis engine XST as the VHDL synthesis tool. | Simulator and Preferred Language  Set for hardware being used. |
| Skip the following dialogues by clicking ***Next>*** or ***Finish*** since we will not be adding existing files or creating new files until later. | |

* *Copy the required lab files* to the **ISEIntro** directory that was created by the above step (C:\FPGA Projects\ISEIntro). These files are available from the Blackboard site. It will be necessary to download and unzip the files into the **ISEIntro** directory. This will create three files:
  + - ***TopLevel.ucf*** – a pin constraint file used to relate the physical pins of the CPLD to top-level ports of the VHDL hierarchy. This file also has comments that then allow you to identify which pins of the CPLD board are connected to the CPLD pins and hence the VHDL ports.
    - ***Toplevel.vhd*** – top-level VHDL file. This is the top of the VHDL hierarchy.
    - ***DieCounter.vhd*** – a counter with sequence “001” to “110” (1 to 6) for the die roll.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * *Add the source files to the project*. Three of the files for our project already exist in the appropriate location so it is only necessary to add them to the project.   Choose ***Project->Add Source…*** to start the process of adding these files. | | | | | | | | | |  | |
| Navigate to the appropriate directory and select the three files to be added. (Use shift-click to select each file without unselecting the earlier ones.)  Click on Open to confirm the selection and add the files. |  | | | | | | | | | | |
| A dialogue will open which allows you to set the Design View in which each of the files just added will appear. The two VHDL files may be used for both synthesis to hardware or simulation while the UCF file is only used with physical hardware implementation.  The defaults that appear in the dialogue may be accepted.  The constraints file (***TopLevel.ucf***) controls how the ports of the top-level VHDL entity (in this case **Toplevel)** are mapped to the pins of the CPLD**.**  The required pin locations are determined by the physical wiring on the CPLD board (see the schematic at the end of the lab). | | | | | | | |  | | | |
| * The ***Module View*** window tab should now appear as shown. The contents of the two VHDL files have been analysed and the entities they contain now appear in the appropriate position in the hierarchy. There is also a *dieDecoder* entity referenced by the *toplevel* entity. This has not been created yet and the question mark icon indicates its absence.   Double-clicking on an entity’s name will open the source file for editing *if it exists*.  Open each of the two existing VHDL files and confirm that their contents agree with the listings attached. | | | | | | | Double-click to open. + Expand the list | | | | |
| * *Create the missing VHDL file*. We will create the missing DieDecoder.vhd file from scratch. This module has a 3-bit wide input from the current counter value and produces a 4-bit output to control the LED display that represents the die roll. This is a combinational circuit that is best represented as a lookup table. A behavioural implementation using a case statement is the usual approach adopted to represent this in VHDL. The VHDL code listing attached shows this implementation. | | | | | | | | | | |  |
| Right click in the Module View window and select ***New Source…*** or use the menu ***Project->New Source*** | |  | | | | | | | | | |
| The next dialog allows the ports of the new entity to be defined. Fill in the dialog as shown and click ***Next*** twice to complete the process.  The new file should now appear in the hierarchy and be opened in an edit window.  The file created contains a basic VHDL skeleton that needs to be fleshed out. | |  | | | | | | | | | |
| **Type in the rest of the file in accordance with the listing attached. You may omit the comments to save typing. (You may cut and paste from the PDF to save time but fix the indenting!)**  Save the file when complete. | | | | | | | | | | | |
| * *Check the syntax of the new file* by selecting it in the ***Sources*** window tab and double-clicking on the ***Check Syntax*** process in the process window below.   A report will appear in the bottom window with any errors or warning flagged on the left border.  Double-clicking on any syntax error message will open the appropriate file in the edit window and position the edit cursor at the offending line if possible.  **Correct any errors before proceeding to the next step.** | | | | 2. Double-click  1. Select | | | | | | | |
| * We are almost ready to download to the hardware but first it is necessary to check which pins are being used for the inputs and outputs of the CPLD. The constraints file that was added to the project earlier (***Toplevel.ucf***) controls this. Open this file in the editor before proceeding – see the necessary steps shown at right. | | | | | 1. Select  3. Double Click to edit  2. Click ‘+’ to expand | | | | | | |
| The supplied file has a line for each pin of the CPLD and has comments indicating which pins they are connected to on the module. For example, locate the line:  NET "clock" LOC = "p1|IOSTANDARD=LVCMOS33"; # CMOD-P35 - GCLK2 Global Clock  This line indicates that the VHDL port named “clock” on the top level VHDL entity is to be assigned to pin 1 of the CPLD. The comment at the end of the line documents that this CPLD pin (p1) has been wired to the P35 of the module by the fixed wiring on the module PCB. This cannot be changed. An external clock source from the clock board needs to be manually connected to this connection.  Locate the line:  NET "dieLeds<0>" LOC = "p18|IOSTANDARD=LVCMOS33"; # CMOD-P9 – I/O  This indicates that the VHDL port named “dieLeds(0)” is to be assigned to pin 18 of the CPLD. The comment shows that this pin is permanently wired to pin 9 of the module. Note that this is a single bit of a *std\_logic\_vector* hence the use of the angle brackets.  Descriptions for unused pins of the CPLD have been left in the file for later use. These have been commented out until needed.  CMOD Module  CPLD  VHDL  clock dieLeds(0)  Clock  Module  LED  Module  p1  p18  P35  P9  Controlled by the UCF File  Fixed wiring on CMOD module  Wire links to modules  Examine the ***Toplevel.ucf*** file to determine which pins of the CPLD board are connected to the Clock, switches, LEDs and the Die display boards. Make the appropriate connections using **neat** links before connecting the power supply (plug pack).   * Connect the cables in the following order:  1. **Connect the JTAG Cable to the PC then the CMOD board.** Be very careful when connecting the JTAG cable plug - Do not connect it backwards as it may damage the board. 2. **Connect the power supply (plug-pack) last.**   **Reverse** this procedure when **disconnecting**. | | | | | | | | | | | |
| * At this point we can run the synthesis and place-and-route tools to create a bit file to download to the board.   Ensure that the top-level entity is still selected and then double-click on the Process ***Generate Programming File*.**  This will cause the execution of all the intermediate steps necessary to produce the bit file.  The bottom console window should show the files being synthesized and the place-and-route tool running.  The final outcome is the bit-file ***toplevel.jed***.   * The actual downloading of the bitfile is done by a separate program called ***impact***. This may be launched from ISE by double-clicking on ***Configure Target Device*** from the process view window tab. | | | | | | 3. Double-click  2. Double-click  1. Select | | | | | |
| * A dialogue will appear. If not just select new project from the file menu.   Make sure the first radio-button (***Configure … JTAG***) is selected and then select the correct option from the drop-down box as shown. This identifies the type of cable we are using to download and asks the software to automatically identify and connect to the single device connected – in this case the CPLD. The other unused drop-down option allows the selection of a particular device in a “chain” of devices all connected to the same download cable. This is not needed since there is only one device. | | |  | | | | | | | | |
| * A file dialogue will now open. It is necessary to select the bitfile to be downloaded to the hardware. This is the file ***Toplevel.jed*** which was generated by the synthesis done above.   Just click OK on the next dialogue to confirm the default options. | | |  | | | | | | | | |
|  | | | | | | | | | | | |
| * Right-click on the symbol representing the CPLD and select ***Program***. * An option dialog will open. Just select ***OK*** to accept the default options. This will download the bitfile to the CPLD and the circuit should be ready for operation. | | | | | | | | |  | | |

* Confirm that the operation of the CPLD board is as expected with the counter output counting from 1 to 6 and the Die LEDs displaying the appropriate pattern.
* It takes quite a while for the IMPACT program to start up. It is not necessary to re-launch it every time you wish to re-program the CPLD. Simply leave it running and return to the ISE software. After modifying your design, double-click on ***Generate Programming File*** as describe above to re-generate the bit file. When this process has completed, swap back to the still running IMPACT program and just download as above. A dialogue will pop-up flagging that the bit file has been changed. Just accept this and proceed. This is much faster than re-launching IMPACT!

|  |  |
| --- | --- |
| * A quick check that the download has been successful is that the debug LED should illuminate when the reset button is pressed. This is due to the VHDL code that connects the reset signal (from the push button) to the LED output. It is useful to include this hardware in the design as a simple download check.   -- for debug - show reset on LED  debugLed <= reset;   * *View the synthesis results*. Select the top-level entity and then double-click on *View RTL Schematic*. This should open the synthesis schematic in a viewer.   The schematic will have a hierarchy that reflects the hierarchy of the VHDL files used to create the circuit.  Initially only the top-level entity will be visible. You may double-click on a block to *drill-down* into the lower levels in the schematic. You may return up through the hierarchy by using the up arrow on the tool bar.  Alternatively you can navigate on the panel that appears at the left of the schematic. |  |
| * Drill down once to view the structure inside the toplevel entity. You should be able to identify the flip-flop used to synchronise the button input (FDC), the counter module and the decoder module. | |
|  | |
| * Drill down into the counter module. You will notice that the circuit consists of a counter block and comparator logic that reflects the logic implied by the VHDL code. | |
|  | |

**VHDL Source File – toplevel.vhd**

--===================================================================

-- Example VHDL for HET202 - Electronic Die

--

-- Revision History

--===================================================================

-- 6/ 5/06 - Created - pgo

--===================================================================

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all; -- Need arithmetic

entity Toplevel is

Port ( Clock : in std\_logic; -- system clock

Reset : in std\_logic; -- reset button on FPGA board

debugLed : out std\_logic; -- debug LED on FPGA board

leds : out std\_logic\_vector(2 downto 0); -- counter outputs

button : in std\_logic; -- run button

dieLeds : out std\_logic\_vector(3 downto 0) -- die pattern LEDs

);

end Toplevel;

architecture behaviour of Toplevel is

-- Synchronized button inputs

signal rollButton : std\_logic;

-- Die roll

signal dieCount : std\_logic\_vector(2 downto 0);

begin

-- For debug - show reset on debug LED

debugLed <= reset;

-- For debug - show dieCount on LEDs

leds(2 downto 0) <= dieCount;

--=====================================

-- Input button synchronization

-- Required to prevent metastability, and input uncertainty due to skew

buttonSync:

process( Reset, Clock, button )

begin

if (reset = '1') then

rollButton <= '0';

elsif (clock'event and (clock = '1')) then

rollButton <= button;

end if;

end process buttonSync;

--=====================================

-- Instantiate decoder

decoder:

entity work.DieDecoder

Port Map ( count => dieCount,

LEDs => dieLeds );

--=====================================

-- Instantiate counter

counter:

entity work.dieCounter

Port Map ( reset => Reset,

clock => Clock,

roll => rollButton,

count => dieCount );

end architecture behaviour;

**VHDL Source File – DieDecoder.vhd**

--===================================================================

-- DieDecoder.vhdl

--

-- Decoder to convert a die roll (1..6) into the correct drive for

-- 7 LEDs arranged in a die pattern (driven by 4 lines).

--

-- +-------+

-- | D C | Die pattern used.

-- | B A B | Each letter A..D represents a drive signal.

-- | C D |

-- +-------+

-- +-------+ +-------+ +-------+ +-------+ +-------+ +-------+

-- | | | | | C | | D C | | D C | | D C |

-- | A | | B B | | A | | | | A | | B B |

-- | | | | | C | | C D | | C D | | C D |

-- +-------+ +-------+ +-------+ +-------+ +-------+ +-------+

-- 001 010 011 100 101 110

--

-- Count => drive

-- 210 DCBA

-- ---------------

-- 000 => 0001 centre LED

-- 010 => 0010 centre pair

-- 011 => 0101 two corners & centre

-- 100 => 1100 2 by two corners

-- 101 => 1101 2 by two corners and centre

-- 110 => 1110 2 by two corners and centre pair

--===================================================================

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity DieDecoder is

Port ( count : in std\_logic\_vector(2 downto 0);

LEDs : out std\_logic\_vector(3 downto 0)

);

end DieDecoder;

architecture Behavioral of DieDecoder is

begin

-- Decodes die count value to Die pattern

decoder:

process( count )

begin

case count is -- DCBA

when "001" => LEDs <= "0001"; -- centre

when "010" => LEDs <= "0010"; -- centre pair

when "011" => LEDs <= "0101"; -- two corners & centre

when "100" => LEDs <= "1100"; -- 2 by two corners

when "101" => LEDs <= "1101"; -- 2 by two corners and centre

when "110" => LEDs <= "1110"; -- 2 by two corners and centre pair

when others => LEDs <= "XXXX"; -- other counts shouldn't occur

end case;

end process;

end Behavioral;

**VHDL Source File – DieCounter.vhd**

--===================================================================

-- DieCounter.vhdl

--

-- A counter to produce a roll in the range 1 .. 6 (like a die)

--===================================================================

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity dieCounter is

Port ( reset : in std\_logic;

clock : in std\_logic;

roll : in std\_logic;

count : out std\_logic\_vector(2 downto 0));

end entity dieCounter;

architecture Behavioural of dieCounter is

signal dieCount : std\_logic\_vector(2 downto 0);

begin

count <= dieCount;

-- Produce dieCount sequence from 1 ... 6

counter:

process( reset, clock )

begin

if (reset = '1') then

dieCount <= "001"; -- start count from 1

elsif (clock'event and (clock = '1')) then

if (roll = '1') then -- rolling

if (dieCount >= "110") then

dieCount <= "001"; -- wrap count

else

dieCount <= dieCount + "001";

end if;

end if;

end if;

end process;

end architecture Behavioural;

Constraints File (abbreviated) – Toplevel.ucf

##################################################

# #

# Pin Assignments for Digilent C-Mod C2 Module #

# #

##################################################

# The C-MOD Pin numbers are shown in the COMMENTS

# e.g. CMOD-P35 is pin 35 of the module and is (usually) used for the Clock input

# The following 3 CPLD pins have special on-chip Clock buffers

# Usually uncomment one Clock line as required

# Make sure use doesn't conflict with I/O use!

NET "Clock" LOC = "p1"|IOSTANDARD=LVCMOS33; # CMOD-P35 - GCK2, Global Clock input

#NET "clock" LOC = "p43"|IOSTANDARD=LVCMOS33; # CMOD-P33 - GCK0, GCI, May be used for I/O

#NET "clock" LOC = "p44"|IOSTANDARD=LVCMOS33; # CMOD-P34 - GCK1, GCI, May be used for I/O

# System level constraints

Net Clock TNM\_NET = Clock;

Timespec TS\_Clock = PERIOD Clock 10 us; # adjust to suit Clock, 10 us -> 100kHz

# Convenient to allocate a Reset pin

NET "Reset" LOC = "p30"|IOSTANDARD=LVCMOS33; # CMOD-P18 - GSR, Global Set/Reset

Net "Reset" TIG; # No timing constraint on system reset

# Pins in pin number order

#

#NET "" LOC = "p12"|IOSTANDARD=LVCMOS33; # CMOD-P1 - I/O

#NET "" LOC = "p13"|IOSTANDARD=LVCMOS33; # CMOD-P2 - I/O

#NET "" LOC = "p14"|IOSTANDARD=LVCMOS33; # CMOD-P3 - I/O

NET "debugLed" LOC = "p16"|IOSTANDARD=LVCMOS33; # CMOD-P4 - I/O

# # CMOD-P5 - N/C

# # CMOD-P6 - N/C

# # CMOD-P7 - N/C

# # CMOD-P8 - N/C

NET "dieLeds<0>" LOC = "p18"|IOSTANDARD=LVCMOS33; # CMOD-P9 - I/O

NET "dieLeds<1>" LOC = "p19"|IOSTANDARD=LVCMOS33; # CMOD-P10 - I/O

NET "dieLeds<2>" LOC = "p20"|IOSTANDARD=LVCMOS33; # CMOD-P11 - I/O

NET "dieLeds<3>" LOC = "p21"|IOSTANDARD=LVCMOS33; # CMOD-P12 - I/O

#NET "" LOC = "p22"|IOSTANDARD=LVCMOS33; # CMOD-P13 - I/O

#NET "" LOC = "p23"|IOSTANDARD=LVCMOS33; # CMOD-P14 - I/O

NET "leds<0>" LOC = "p27"|IOSTANDARD=LVCMOS33; # CMOD-P15 - I/O

NET "leds<1>" LOC = "p28"|IOSTANDARD=LVCMOS33; # CMOD-P16 - I/O

NET "leds<2>" LOC = "p29"|IOSTANDARD=LVCMOS33; # CMOD-P17 - I/O

# # CMOD-P18 - GSR, Global Set/Reset - see above

# # CMOD-P19 - N/C

# # CMOD-P20 - Vcc

# # CMOD-P21 - Gnd

NET "button" LOC = "p31"|IOSTANDARD=LVCMOS33; # CMOD-P22 - GTS2, Global Tristate control

#NET "" LOC = "p32"|IOSTANDARD=LVCMOS33; # CMOD-P23 - GTS3, Global Tristate control

#NET "" LOC = "p33"|IOSTANDARD=LVCMOS33; # CMOD-P24 - GTS0, Global Tristate control

#NET "" LOC = "p34"|IOSTANDARD=LVCMOS33; # CMOD-P25 - GTS1, Global Tristate control

#NET "" LOC = "p36"|IOSTANDARD=LVCMOS33; # CMOD-P26 - I/O

#NET "" LOC = "p37"|IOSTANDARD=LVCMOS33; # CMOD-P27 - I/O

#NET "" LOC = "p38"|IOSTANDARD=LVCMOS33; # CMOD-P28 - I/O

#NET "" LOC = "p39"|IOSTANDARD=LVCMOS33; # CMOD-P29 - I/O

#NET "" LOC = "p40"|IOSTANDARD=LVCMOS33; # CMOD-P30 - I/O

#NET "" LOC = "p41"|IOSTANDARD=LVCMOS33; # CMOD-P31 - I/O

#NET "" LOC = "p42"|IOSTANDARD=LVCMOS33; # CMOD-P32 - I/O

#NET "" LOC = "p43"|IOSTANDARD=LVCMOS33; # CMOD-P33 - I/O, GCI, May be used for I/O

# # CMOD-P34 - I/O, GCI, May be used for I/O

# # CMOD-P35 - I/O, GCI, May be used for I/O

#NET "" LOC = "p2"|IOSTANDARD=LVCMOS33; # CMOD-P35 - I/O

#NET "" LOC = "p3"|IOSTANDARD=LVCMOS33; # CMOD-P36 - I/O

#NET "" LOC = "p4"|IOSTANDARD=LVCMOS33; # CMOD-P37 - I/O

#NET "" LOC = "p5"|IOSTANDARD=LVCMOS33; # CMOD-P38 - I/O

#NET "" LOC = "p6"|IOSTANDARD=LVCMOS33; # CMOD-P39 - I/O

#NET "" LOC = "p8"|IOSTANDARD=LVCMOS33; # CMOD-P40 - I/O