Digital Systems

EEE4084F

2017-05-10

FPGA Introduction – Verilog and Xilinx ISE [30 Marks]

Background

This practical is divided into two parts. The first is a tutorial that shows you how to set up a new FPGA project in Xilinx ISE. The second is a practical in which you are expected to implement a real-time clock on the 7-segment display.

Although this practical has been designed with the Nexys3 in mind, some of you have a Nexys2 or Nexys4 instead. These boards are largely compatible. The only things to bear in mind are the FPGA device and pin numbers, which differ. There are also small differences in the compiler for the various target devices, but you shouldn't need to worry about those.

During the practical you might experience some problems. The Appendix (page 10) lists the most common ones, along with the solution.

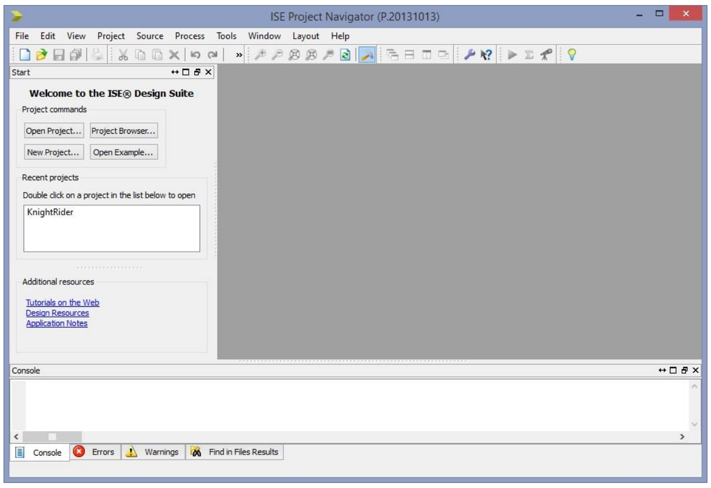
Submission Information

You may be delighted to know that you do not need to submit a formal prac report for this prac. However, you need to demo your project to the tutor or lecture to get it approved and also submit your project files, you can do this as a zip file attachment to the Vula assignment. But do try to have well-structured and suitably commented code (and UCF) as these count for marks.

Setting Up an FPGA Project

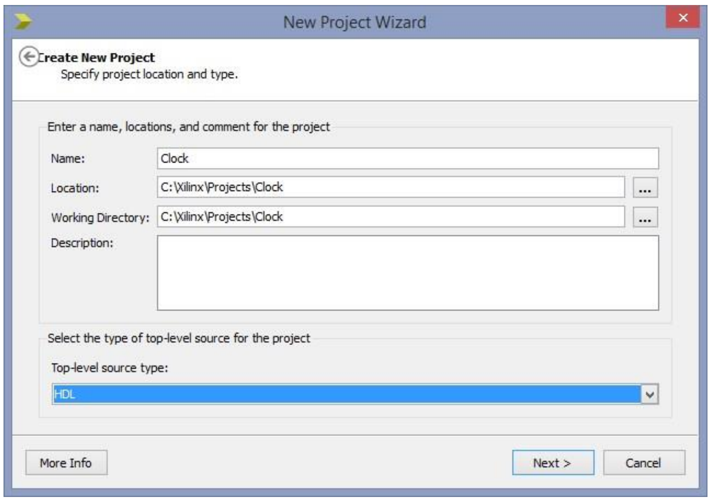
#1 First, start Xilinx ISE Project Navigator. Starting ISE Design Suite does the same thing. You should then see the image on the right.

In the Blue lab you can open a terminal and run ise (or impact). On a Windows installation you can use the desktop shortcut.



#2 Click on the “New Project...” button, which opens up the screen on the right.

Fill in your new project details and click “Next”.



Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

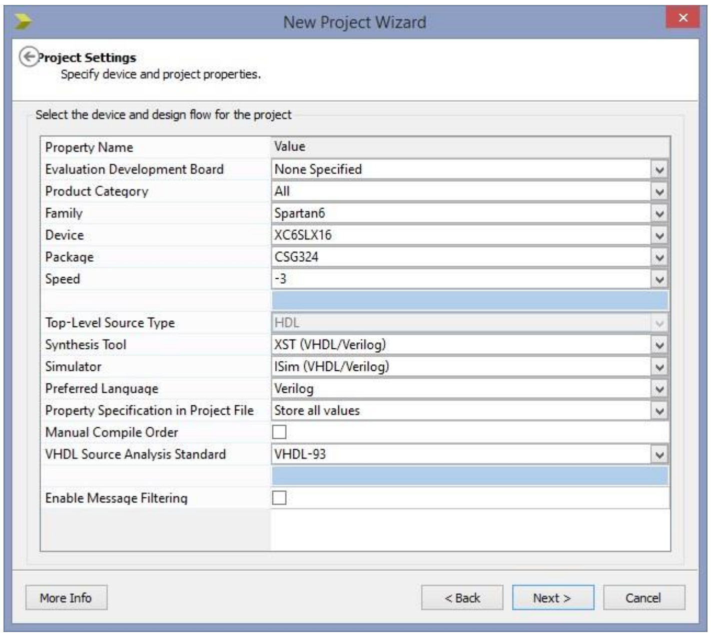
Name:	Clock
Location:	C:\Xilinx\Projects\Clock
Working Directory:	C:\Xilinx\Projects\Clock
Description:	

Select the type of top-level source for the project

Top-level source type: HDL

#3 If you’re using the Nexys 3, specify that you’re using a Spartan6 FPGA: XC6SLX16CSG324 (it is printed on the IC). Follow a similar method for the Nexys 2 or Nexys 4 board.

Choose “Verilog” as your Preferred Language and click “Next”.

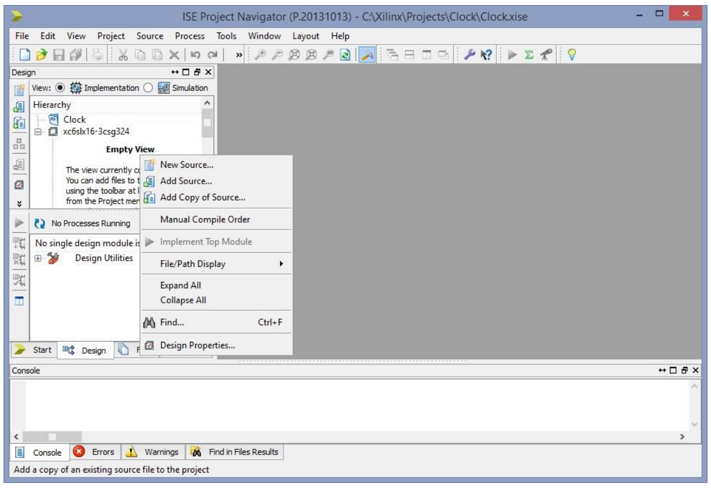


Project Settings
Specify device and project properties.

Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

#4 You can either right-click on the Hierarchy display, or click the “New Source” icon on the left. This will open the “New Source Wizard”.

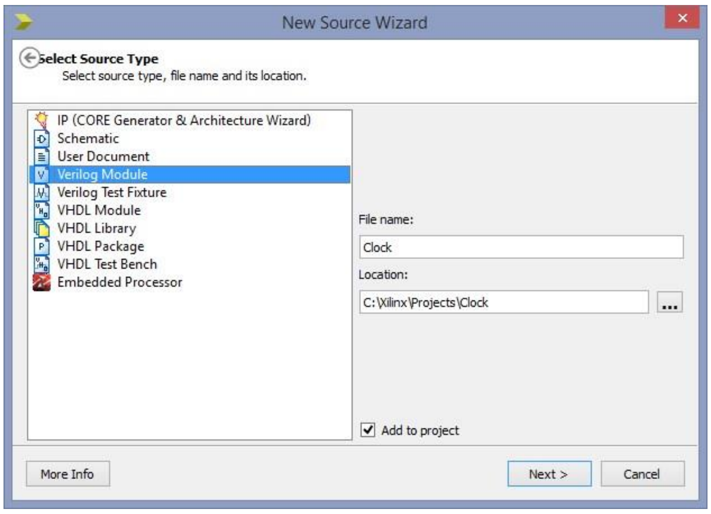


The view currently c
You can add files to t
using the toolbar at l
from the Project mer

No single design module is

Add a copy of an existing source file to the project

#5 Select "Verilog Module", fill in an appropriate name and click "Next". It will ask you what ports and things you'd like to add. Leave those blank.

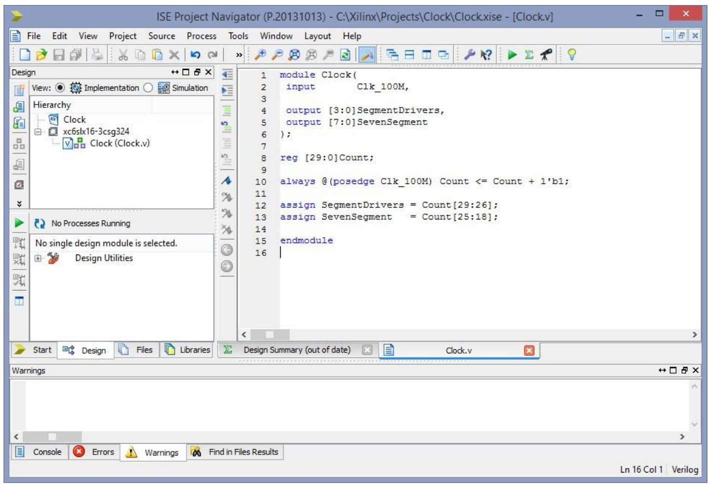

New Source Wizard

Select Source Type
Select source type, file name and its location.

- IP (CORE Generator & Architecture Wizard)
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: Clock
Location: C:\Xilinx\Projects\Clock
[✓] Add to project

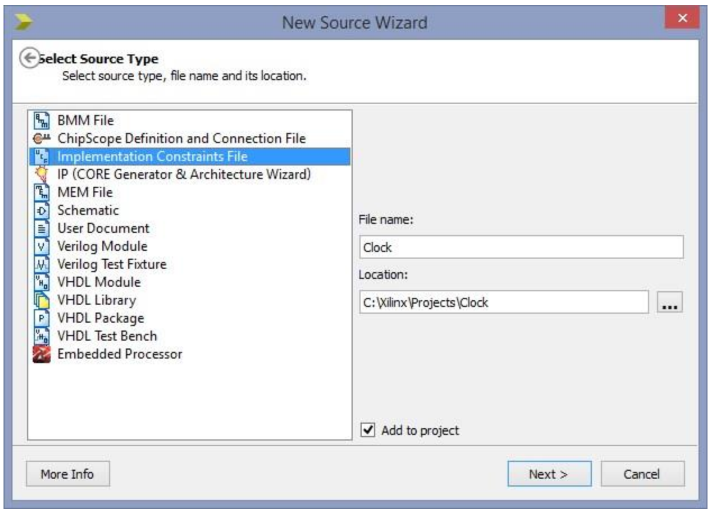
#6 Delete the default module contents and type in the module definition as depicted on the right.


ISE Project Navigator (P.20131013) - C:\Xilinx\Projects\Clock\Clock.xise - [Clock.v]

```
module Clock(
 input       Clk_100M,

 output [3:0]SegmentDrivers,
 output [7:0]SevenSegment
);

reg [29:0]Count;

always @(posedge Clk_100M) Count <= Count + 1'b1;

assign SegmentDrivers = Count[29:26];
assign SevenSegment   = Count[25:18];

endmodule

```

#7 Add a new source file: this time an Implementation Constraints File.

You can read up all about design constrains in the Xilinx Constraints Guide: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/cgd.pdf
You want to look at info about UCF files starting on pg. 34.


New Source Wizard

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 Package
- VHDL Test Bench
- Embedded Processor

File name: Clock
Location: C:\Xilinx\Projects\Clock
[✓] Add to project

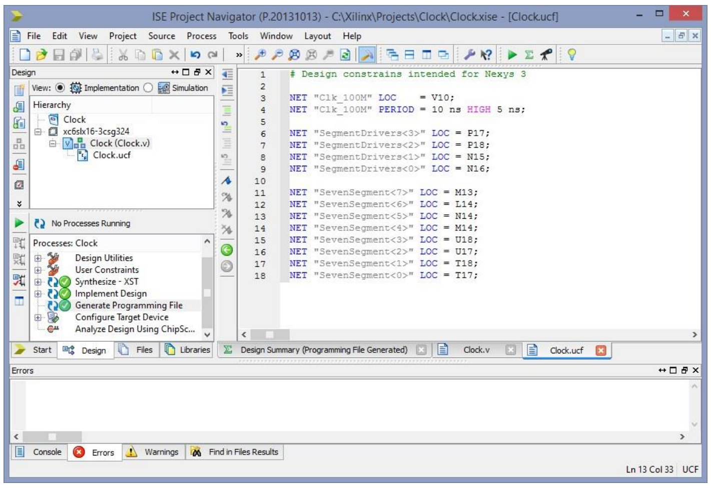
#8 There is a GUI to add pins and assignments (i.e., PlanAhead[1]); but it is often easier, especially with large designs, to text-edit the constraints file directly.

Add the pin assignments depicted on the right. The pin numbers are available from the Nexys 3 reference manual:

http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm_V2.pdf

(if you're not using a Nexys 3, change the numbers accordingly).

Remember to specify the clock period (see line 4).


ISE Project Navigator (P.20131013) - C:\Xilinx\Projects\Clock\Clock.xise - [Clock.ucf]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
Clock
xc6slx16-3csg324
Clock (Clock.v)
Clock.ucf
No Processes Running
Processes: Clock
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipSc...
Start Design Files Libraries
Design Summary (Programming File Generated)
Clock.v
Clock.ucf
Errors
Console Errors Warnings Find in Files Results
Ln 13 Col 33 UCF

```
1   # Design constrains intended for Nexys 3
2
3   NET "Clk_100M" LOC    = V10;
4   NET "Clk_100M" PERIOD = 10 ns HIGH 5 ns;
5
6   NET "SegmentDrivers<3>" LOC = P17;
7   NET "SegmentDrivers<2>" LOC = P18;
8   NET "SegmentDrivers<1>" LOC = N15;
9   NET "SegmentDrivers<0>" LOC = N16;
10
11  NET "SevenSegment<7>" LOC = M13;
12  NET "SevenSegment<6>" LOC = L14;
13  NET "SevenSegment<5>" LOC = N14;
14  NET "SevenSegment<4>" LOC = M14;
15  NET "SevenSegment<3>" LOC = U18;
16  NET "SevenSegment<2>" LOC = U17;
17  NET "SevenSegment<1>" LOC = T18;
18  NET "SevenSegment<0>" LOC = T17;
```

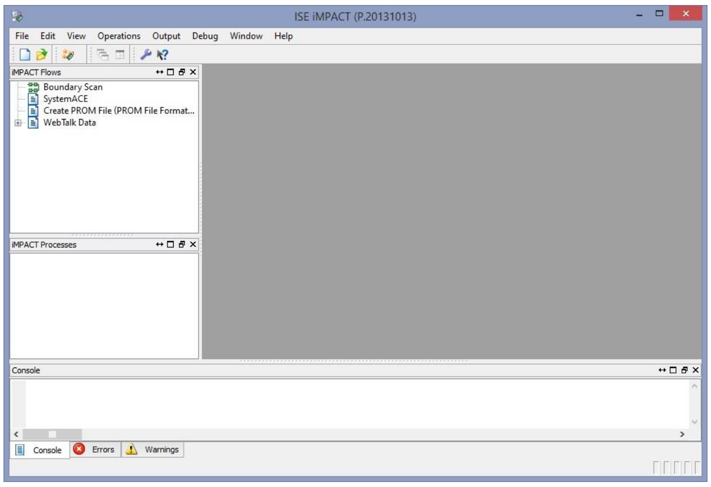
If you have a Nexys4, you also have to specify the I/O standard. You can do this by appending "|IOSTANDARD = LVCMOS33" to every pin assignment. Even if you do not have a Nexys4, it remains a good idea to do so.

It is also a good idea to specify "timing ignore" for every asynchronous pin (which, in your case, is all of them except the clock). Do this by appending "|TIG" to the line. You might, for instance, have a line that reads:

```
NET "SevenSegment<0>" LOC = L3 | IOSTANDARD = LVCMOS33 | TIG;
```

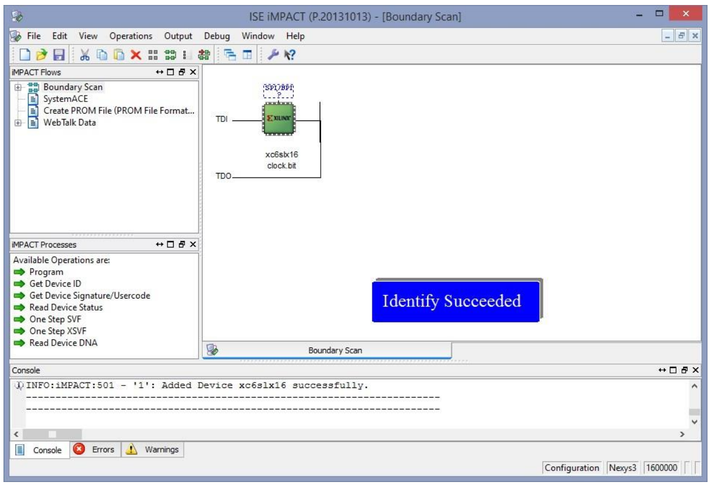
Remember to add pin specifications for every pin you add to your design.

#9 Double-click on the "Configure Target Device" item in the process tree. This should compile your project and then open iMPACT.



[1] Useful YouTube to view if you want to try PlanAhead: https://www.youtube.com/watch?v=KOTLVQS3XGM (but swap his mention of Vertex5 for Spartan6 or whatever your Nexys board has)

#10 Plug in your FPGA board and double-click on the "Boundary Scan" item in the Flows tree. Right-click on the canvas and select "Initialize Chain". iMPACT should detect the board, which FGPA is on it, and then ask you to specify a bit-stream for it. If it does not, there is something wrong with the board, cable or driver.



```
INFO:iMPACT:501 - '1': Added Device xc6slx16 successfully.
------------------------------------------------------------------
------------------------------------------------------------------
```

Tell it to use your newly generated bit-stream (typically Clock.bit). You can now double-click on "Program" in the Processes tree to program your FPGA.

Time-Delayed Reset

Many external peripherals require some set-up time, so it is useful to have a time-delayed reset. Add one of the buttons to the top-level module (and remember to map a pin to it).

#1 Create a new module called Delay_Reset.v and add it to your project. Fill in the code below:

```
module Delay_Reset (
 input  Clk,  // Input clock signal
  input  Button,   // Input reset signal (external button)
  output reg Reset   // Output reset signal (delayed)
);
//------------------------------------------------------------------------------

reg   LocalReset;
reg [22:0]Count;          // Assume Count is null on FPGA configuration
//------------------------------------------------------------------------------
always @(posedge Clk) begin            // Activates every clock edge
  LocalReset <= Button;                // Localise the reset signal

  if (LocalReset) begin                // Reset block
    Count <= 0;                        // Reset Count to 0
    Reset <= 1'b1;                     // Reset the output to 1

  end else if(&Count) begin            // When Count is all ones...
    Reset <= 1'b0;                     // Release the output signal
    // Count remains unchanged         // And do nothing else

  end else begin                       // Otherwise...
    Reset <= 1'b1;                     // Make sure the output signal is high
    Count <= Count + 1'b1;             // And increment Count
  end
 end // always
//------------------------------------------------------------------------------

endmodule
//------------------------------------------------------------------------------
```

#2 Within your top-level entity, instantiate this new module. Here’s how to do it:

```
wire Reset; // Define a local wire to carry the Reset signal

Delay_Reset Delay_Reset1 (Clk_100M, BTNS, Reset);
//-----------------------------------------------------------------------------
```

Seven-Segment Driver

#1 In order to make the seven-segment display easier to use, we make use of a driver module. This module takes 4 BCD values and displays them on the seven segment display. Create a new module, call it SS Driver.v and add it to your project. Add the following code to it:

```
module SS_Driver(
  input Clk, Reset,
  input [3:0] BCD3, BCD2, BCD1, BCD0,   // Binary-coded decimal input
  output reg [3:0] SegmentDrivers,      // Digit drivers (active low)
  output reg [7:0] SevenSegment         // Segments (active low)
);
//-----------------------------------------------------------------------------

// Make use of a subcircuit to decode the BCD to seven-segment (SS)
wire [6:0]SS[3:0];
BCD_Decoder BCD_Decoder0 (BCD0, SS[0]);
BCD_Decoder BCD_Decoder1 (BCD1, SS[1]);
BCD_Decoder BCD_Decoder2 (BCD2, SS[2]);
BCD_Decoder BCD_Decoder3 (BCD3, SS[3]);
//-----------------------------------------------------------------------------
// Counter to reduce the 100 MHz clock to 762.939 Hz (100 MHz / 2^17)
reg [16:0]Count;
// Scroll through the digits, switching one on at a time
 always @(posedge Clk) begin
    Count <= Count + 1’b1;
    if ( Reset) SegmentDrivers <= 4’hE;
    else if(&Count) SegmentDrivers <= {SegmentDrivers[2:0], SegmentDrivers[3]};
end
//-----------------------------------------------------------------------------
always @(*) begin // This describes a purely combinational circuit
  SevenSegment[7] <= 1’b1; // Decimal point always off
  if (Reset) begin
    SevenSegment[6:0] <= 7’h7F; // All off during Reset
  end else begin
      case(~SegmentDrivers)                              // Connect the correct signals,
     4’h1 : SevenSegment[6:0] <= ~SS[0]; // depending on which digit is on at
     4’h2 : SevenSegment[6:0] <= ~SS[1]; // this point
     4’h4 : SevenSegment[6:0] <= ~SS[2];
     4’h8 : SevenSegment[6:0] <= ~SS[3];
     default: SevenSegment[6:0] <= 7’h7F;
    endcase
   end
  end
  endmodule
//-----------------------------------------------------------------------------
```

You need to instantiate it in your top-level entity:

```
SS_Driver SS_Driver1(
  Clk_100M, Reset,
  4'd1, 4'd2, 4'd3, 4'd4, // Use temporary test values
  SegmentDrivers, SevenSegment
);
```

You'll require a module to decode the BCD to seven-segment, so implement BCD_
Decoder.v as follows:

```
module BCD_Decoder(
 input [3:0]BCD,
 output reg [6:0] SevenSegment
);
//------------------------------------------------------------------------------
// Combinational circuit to convert BCD input to seven-segment output
always @(BCD) begin
   case(BCD)                        // gfedcba
    4'd0    : SevenSegment <= 7'b0111111;  //     a
    4'd1    : SevenSegment <= 7'b0000110;  //    ----
    4'd2    : SevenSegment <= 7'b1011011;  //   |    |
    4'd3    : SevenSegment <= 7'b1001111;  //  f| g |b
    4'd4    : SevenSegment <= 7'b1100110;  //    ----
    4'd5    : SevenSegment <= 7'b1101101;  //   |  |
    4'd6    : SevenSegment <= 7'b1111101;  //  e|  |c
    4'd7    : SevenSegment <= 7'b0000111;  //    ----
    4'd8    : SevenSegment <= 7'b1111111;  //     d
    4'd9    : SevenSegment <= 7'b1101111;
    default: SevenSegment <= 7'b0000000;
   endcase
 end
//------------------------------------------------------------------------------
endmodule
//------------------------------------------------------------------------------
```

Digital Clock

#1 Implement a simple state-machine to display the real time (hours and minutes) on the 7-segments display. You can start your clock at 00:00 upon reset. Make your clock faster in order to test that the time overflows correctly. Use a 24-hour time format.

The easiest way to do this is with deeply nested if statements. Run a counter that overflows every second and increment the seconds counter on every overflow. Every time this seconds counter equals 59 (i.e. it will overflow on this clock-cycle), increment a minutes _units counter. Every time this minutes units counter is about to overflow, increment a minutes _tens counter, etc.

Only use non-blocking assignments (<=). Blocking assignments (=) inside clocked structures are much more difficult to debug. Remember that the entire always block is evaluated at once: the statements are not evaluated sequencially.

#2 Display the seconds on the LEDs, in binary format. This is done with a simple assignment outside the always block.

#3 Use one of the buttons, properly debounced, to set the minutes. It must increment time by one minute every time it is pressed. Make sure that your time overflows correctly. You do not need to increment the hours when changing the minutes.

It is recommended to write a Debounce module for this. On every clock cycle of the system clock (the fast one), check the state of the button. If it is not the same as the current module output, change the output and start a dead time counter. While this counter is counting, do not change the output of the module, no matter what the input is doing. Use a dead-time of between 20 ms and 40 ms. To prevent unstable states, register the button before use.

In the clock state machine, you can use a register to store the button's 'previous' state. If the current state is high, and the previous state is low, the button signal went through a rising-edge. Do not use always @(posedge Button) – keep everything in the same clock domain.

#4 Use another one of the buttons, properly debounced, to set the hours. It must increment time by one hour every time it is pressed. Make sure that your time overflows correctly.

#5 Use the slide-switches to represent a binary "brightness" word. Make use of pulse-width modulation (PWM) to dim the brightness of the LED display.

Ensure that the phasing between the driver signals and the PWM signals is correct. The easiest way to do this is to select a PWM frequency such that the PWM signal goes through exactly one period between driver signal state changes. You can implement this within the SS _Driver module.

Appendix

Common Problems

#1 **Xilinx ISE crashes every time it wants to browse to a file**

Xilinx 64-bit is incompatible with Windows 8. All the short-cuts point to

- C:\Xilinx\14.7\ISEDS\ISE\bin\nt64\ and uses
- C:\Xilinx\14.7\ISEDS\settings64.bat. Change them to point to
- C:\Xilinx\14.7\ISEDS\ISE\bin\nt\ and use
- C:\Xilinx\14.7\ISEDS\settings32.bat instead.

#2 **The compilation button is greyed-out**

You have to have your top-level module selected (in the Hierarchy tree) for the compilation button to be available.

#3 **The compilation button does not actually do anything**

The compiler is confused sometimes and does not notice that files have changed. Right-click on the "Create Programming File" and then click on "Rerun All".
Sometimes it only runs half-way, in which case you have to do this again.

#4 **iMPACT does not display a program button**

You have to have the target device selected (on the canvas) for the program button to be available. Alternatively, you can right-click on the target device and then click on "Program".

#5 **A module has been added to the hierarchy, but the compiler complains that it does not exist**

This has something to do with the compilation order. Remove the module from the hierarchy, re-add it and try again.

#6 **iMPACT complains that the download cable is in use**

Close iMPACT, plug out the USB cable, restart iMPACT, plug the cable back in and try again. If this doesn't work, restart Windows.

#7 **The compilation is stuck somewhere and never ends**

You probably have a syntax error in your .UCF file. This causes the compiler to crash, without messages. Forcibly stop the processing, fix the syntax error, and try again.

This also sometimes happens even when there are no syntax errors. In that case, close ISE Project Navigator, reopen it and try again.