10 Xilinx Vivado Quick Start Guide

This appendix will walk you through the process of synthesis and mapping of HDL code (such as Verilog HDL) to an FPGA using Xilinx Vivado.

A Introduction

Xilinx Vivado is a software tool from Xilinx which is used to synthesize and analyze HDL designs. The designers/developers can synthesize the design, check the RTL schematic, perform timing analysis, simulate the design with different test fixtures, and program the target FPGA.

B Steps

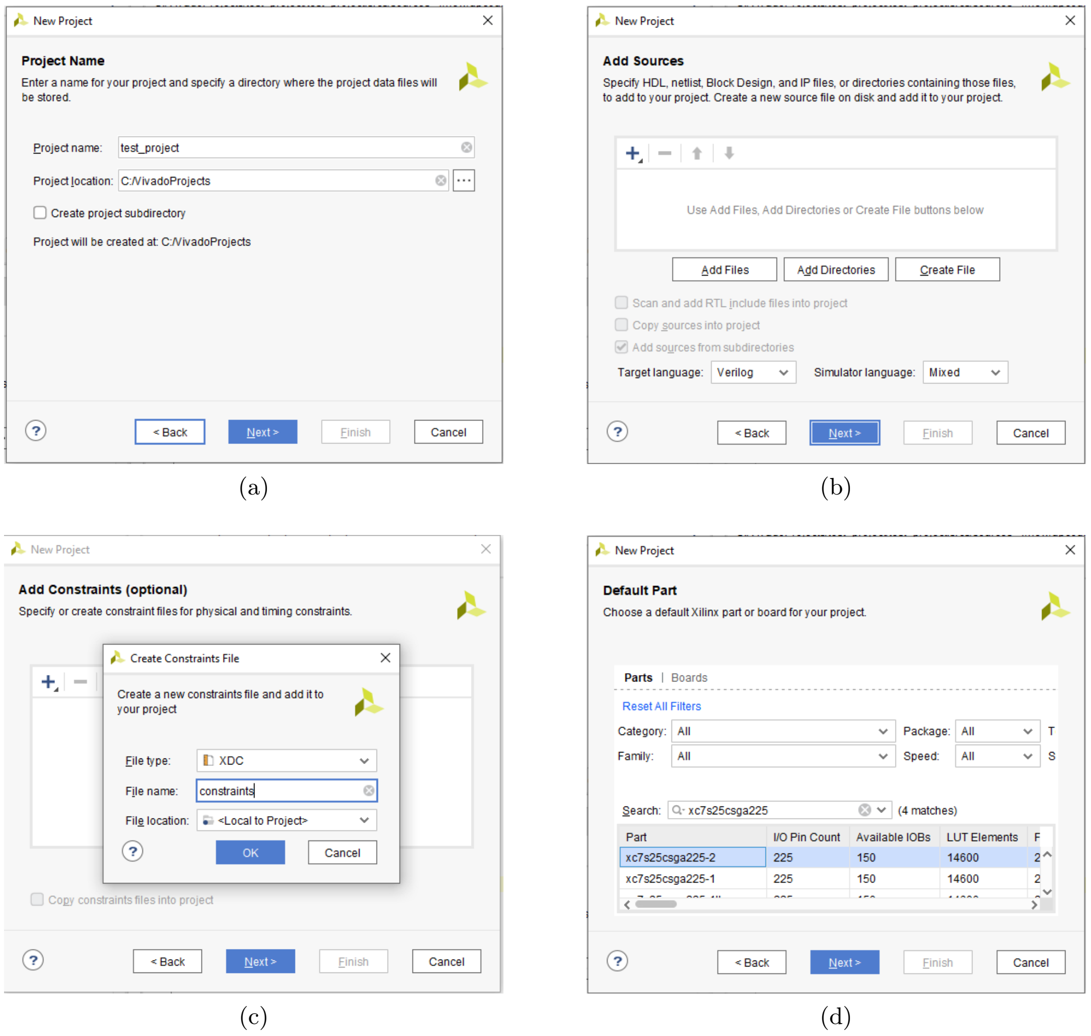
Install the latest edition of Vivado ML Standard Edition (AMD Xilinx Vivado v2021.1-2022.2). After a successful installation, follow these steps to create a Vivado project, create the source files, synthesize the design, implement the design and finally verify the functionality in FPGA. For more details on Xilinx Vivado, refer to the in-depth guide by Xilinx [1].

1. Open the Vivado project wizard and create a new project as shown in Figure 1(a-d). Set the project name and location (a); then, you'll proceed through multiple project settings prompts as you hit "next" through each. Set the "Project Type" to "RTL Project". Next you have the option to add or create sources to the design (b). Click "Create File" and specify the file type as Verilog, with the name "counter.v" (for this tutorial). When you proceed, you will be given a prompt to declare input/output ports, this step is optional.

   Listing 1: Verilog Code for Up/Down Counter

```
module counter(clock, reset, dir, cval);
    input clock, reset, dir;
    output reg [7:0] cval;
    always @ (posedge clock)  begin
        if (reset == 1'b1) begin
            cval = 8'b0;
        end
        else begin
            if (dir == 1'b1) begin
                cval = cval + 1;
            end
            else begin
                cval = cval - 1;
            end
        end
  end
endmodule
```

2. Open your constraints file, and add mappings for the top module's I/O to different ports on the FPGA. The base constraints file is provided by Digilent here ↗ . A copy of this file is also



(a)

New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: test_project

Project location: C:/VivadoProjects

Create project subdirectory

Project will be created at: C:/VivadoProjects

(b)

New Project

Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project.

Use Add Files, Add Directories or Create File buttons below

Scan and add RTL include files into project

Copy sources into project

[✓] Add sources from subdirectories

Target language: Verilog

Simulator language: Mixed

(c)

New Project

Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Create Constraints File

Create a new constraints file and add it to your project

File type: XDC

File name: constraints

File location: <Local to Project>

Copy constraints files into project

(d)

New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | Boards

Reset All Filters

Category: All

Package: All

Family: All

Speed: All

Search: xc7s25csga225 (4 matches)

Part	I/O Pin Count	Available IOBs	LUT Elements	F
xc7s25csga225-2	225	150	14600	2
xc7s25csga225-1	225	150	14600	2

Figure 1: Create new project (a), add existing or new design sources to project (b), add existing or new constraint file(s) to project (c), and select FPGA part to target (d).

included in the lab directory. Copy the relevant lines into your constrants.xdc file, making sure to uncomment the lines you need (in this case, the clock, two pushbuttons, and output pins). A minimum example is shown here:

Listing 2: Example Constraints (XDC) File

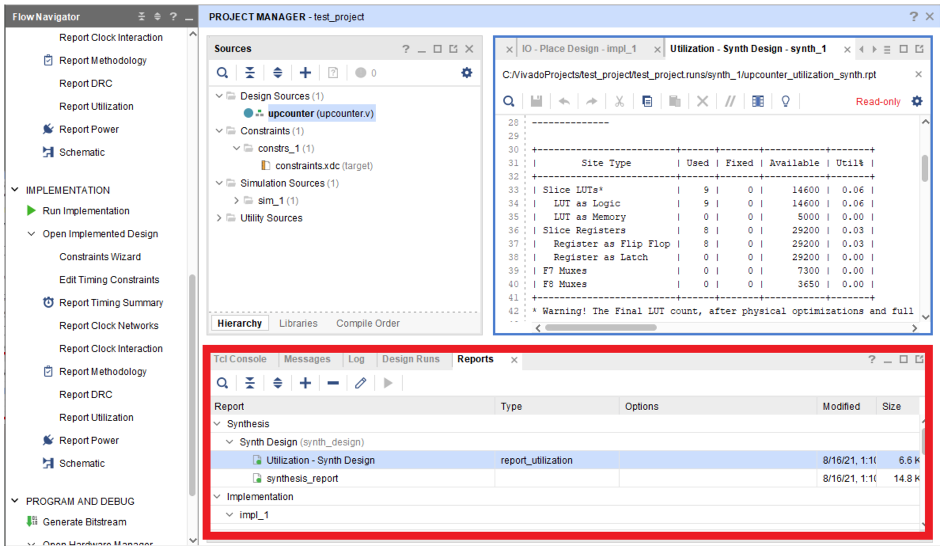
```
## 12 MHz System Clock
set_property -dict { PACKAGE_PIN M9 IOSTANDARD LVCMOS33 }
    [get_ports { clock }];
create_clock -add -name sys_clk_pin -period 83.33
    -waveform {0 41.66} [get_ports { clock }];

## Push Buttons
set_property -dict { PACKAGE_PIN D2 IOSTANDARD LVCMOS33 }
    [get_ports { reset }];
set_property -dict { PACKAGE_PIN D1 IOSTANDARD LVCMOS33 }
    [get_ports { dir }];

## 4 LEDs
#set_property -dict { PACKAGE_PIN E2 IOSTANDARD LVCMOS33 }
#   [get_ports { led[0] }];
#set_property -dict { PACKAGE_PIN K1 IOSTANDARD LVCMOS33 }
#   [get_ports { led[1] }];
#set_property -dict { PACKAGE_PIN J1 IOSTANDARD LVCMOS33 }
#   [get_ports { led[2] }];
#set_property -dict { PACKAGE_PIN E1 IOSTANDARD LVCMOS33 }
#   [get_ports { led[3] }];

## Dedicated Digital I/O on the PIO Headers
set_property -dict { PACKAGE_PIN L1 IOSTANDARD LVCMOS33 }
    [get_ports { cval[0] }];
set_property -dict { PACKAGE_PIN M4 IOSTANDARD LVCMOS33 }
    [get_ports { cval[1] }];
set_property -dict { PACKAGE_PIN M3 IOSTANDARD LVCMOS33 }
    [get_ports { cval[2] }];
set_property -dict { PACKAGE_PIN N2 IOSTANDARD LVCMOS33 }
    [get_ports { cval[3] }];
set_property -dict { PACKAGE_PIN M2 IOSTANDARD LVCMOS33 }
    [get_ports { cval[4] }];
set_property -dict { PACKAGE_PIN P3 IOSTANDARD LVCMOS33 }
    [get_ports { cval[5] }];
set_property -dict { PACKAGE_PIN N3 IOSTANDARD LVCMOS33 }
    [get_ports { cval[6] }];
set_property -dict { PACKAGE_PIN P1 IOSTANDARD LVCMOS33 }
    [get_ports { cval[7] }];
```

You may use the two input buttons for reset/dir, and the dedicated on-board clock for the clock signal. Because we are not using a communication protocol like UART in this project, we can just assign the 8-bit output to the dedicated digital I/O on the PIO headers. Alternatively, for a visual output, you can use the on-board LEDs to display the current counter value. Note however that the Cmod S7 board has a 12 MHz clock; you would need a *much* slower clock signal to display the counter value, otherwise the LEDs will turn on and off too fast to see! In this case, a 23-bit (or so) counter would slow the 12 MHz clock down to about 1 second.



C:/VivadoProjects/test_project/test_project.runs/synth_1/upcounter_utilization_synth.rpt

```
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |    9 |     0 |     14600 |  0.06 |
|   LUT as Logic          |    9 |     0 |     14600 |  0.06 |
|   LUT as Memory         |    0 |     0 |      5000 |  0.00 |
| Slice Registers         |    8 |     0 |     29200 |  0.03 |
|   Register as Flip Flop |    8 |     0 |     29200 |  0.03 |
|   Register as Latch     |    0 |     0 |     29200 |  0.00 |
| F7 Muxes                |    0 |     0 |      7300 |  0.00 |
| F8 Muxes                |    0 |     0 |      3650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full
```

Report	Type	Options	Modified	Size
Synthesis				
Synth Design (synth_design)				
Utilization - Synth Design	report_utilization		8/16/21, 1:1(	6.6 K
synthesis_report			8/16/21, 1:1(	14.8 K
Implementation				
impl_1				

Figure 2: View synthesis (and other procedures) output reports in the bottom window of Vivado.

3. Synthesize the design by double-clicking "Run Synthesis" from the left panel. Once synthesis is completed, a prompt will appear which will allow you to view the synthesized design and reports. At the bottom of the screen, you can see tabs for several items such as Messages, Log, and Reports – as shown in Figure 2. Down there you can double click on each report and it will be opened in the above editor. The messages and log tabs can provide useful information such as errors and warnings.

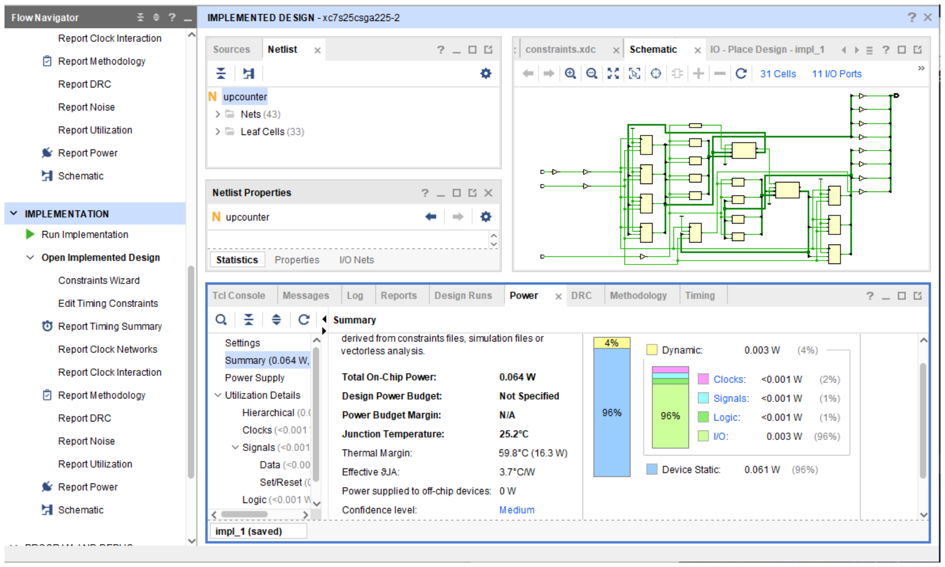
3. Run the implementation process by clicking 'Run Implementation', which includes translate, map, place and route. After that, check the IO report under "place design" from the reports to ensure the pins are correctly assigned.

4. Click on "Open Implemented Design" to view more details on the implementation. Here, we can view a diagram of where the design has been place on the FPGA. We can also view a schematic of the design by right-clicking the upcounter netlist and selecting "Schematic". In the "Power" tab where the reports are, we can view a nice breakdown of the module's power consumption – as shown in Figure 3

5. To generate the bitstream, which is the configuration file used to program the FPGA, generate the bitstream by clicking on "Generate Bitstream".

6. Program the target device using the Hardware Manager in Vivado.

   If you encounter an error such as "No top module specified", right click on upcounter.v and select "Set as top module".



Summary

derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.064 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.2°C**
Thermal Margin:	59.8°C (16.3 W)
Effective ϑJA:	3.7°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

🟨	Dynamic:	0.003 W	(4%)
🟪	Clocks:	<0.001 W	(2%)
🟦	Signals:	<0.001 W	(1%)
🟩	Logic:	<0.001 W	(1%)
🟩	I/O:	0.003 W	(96%)
🟦	Device Static:	0.061 W	(96%)

Figure 3: Open implemented design, view schematic and a detailed view of how the design has been placed on the FPGA.

References Cited

[1] Xilinx, "Vivado design suite tutorial," 2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug888-vivado-design-flows-overview-tutorial.pdf