Appendix A: Vivado Tutorial

1. Introduction

This tutorial is based on *Vivado HLx 2018.2 WebPACK* (free at xilinx.com). The circuit used in the tutorial is the registered unsigned adder of figure A.1a, synthesized with the VHDL code of figure A.1b. The adder inputs (*a*, *b*) are 3-bit signals, while its output (*sum*) is a 4-bit signal, so overflow never occurs. Both functional and timing simulations are shown using the stimuli of figure A.1c and in the following two situations: using a testbench (figure A.1d) and using a Tcl script (figure A.1e).

The structure of the work library created by Vivado is shown in figure A.2. The *registered_adder.srcs* folder contains all source files (files created by the user), divided into three categories: *sources_1* (design files), *sim_1* (simulation files), and *constrs_1* (constraint files). Note the file called *registered_adder.xpr*, which is the Xilinx project file; clicking on it opens the project.

2. Starting a New Project

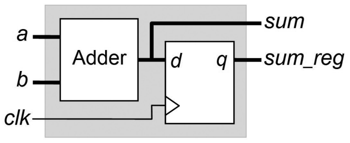
a) Launch Vivado, which opens the screen of figure A.3a.

b) Click **Create Project** and **Next**, which leads to figure A.3b. Enter the project name (*registered_adder*) and the desired location for the project. Mark **Create project subdirectory** and click **Next**.

c) In figure A.3c, mark **RTL Project** (VHDL, in our case) and **Do not specify sources at this time**, then click **Next**.

d) In figure A.3d, select the FPGA device or the FPGA board. In this tutorial, the XC7A35TCPG236 Artix-7 FPGA is employed. Click **Next** and **Finish**, which finally opens the project Flow Navigator (figure A.4).
Note: The FPGA selection can be made or changed later at PROJECT MANAGER > Settings.

e) Observe on the lefthand side of figure A.4 the several sections of the Flow Navigator: PROJECT MANAGER, IP INTEGRATOR, SIMULATION, RTL ANALYSIS, SYNTHESIS,



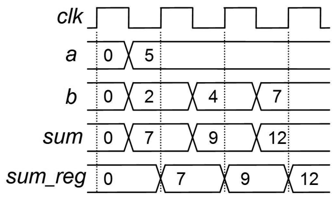
(a) Registered adder.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity registered_adder is
  port (
    clk: in std_logic;
    a, b: in std_logic_vector(2 downto 0);
    sum, sum_reg: out std_logic_vector(3 downto 0));
end entity;

architecture rtl of registered_adder is
begin
  sum <= std_logic_vector(('0' & unsigned(a)) + unsigned(b));
  process(clk)
    begin
      if rising_edge(clk) then
        sum_reg <= sum;
      end if;
  end process;
end architecture;
```

(b) VHDL design file.



(c) Stimuli (and expected responses) employed in the simulations.

(d) VHDL testbench file.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity registered_adder_tb is
end entity;

architecture testbench of registered_adder_tb is
  signal clk: std_logic := '1';
  signal a, b: std_logic_vector(2 downto 0) := "000";
  signal sum, sum_reg: std_logic_vector(3 downto 0);
begin

  duv: entity work.registered_adder
    port map (clk, a, b, sum, sum_reg);

  clk <= not clk after 40 ns;
  a <= "101" after 40 ns;
  b <= "010" after 40 ns, "100" after 120 ns,
       "111" after 200 ns;

end architecture;
```

(e) Tcl script.

```
launch_simulation
restart
#add_wave clk a b sum sum_reg (commented out, not necessary)
add_force clk {1} {0 40} -repeat_every 80 -cancel_after 320
add_force a {0} {5 40} -radix unsigned
add_force b {0} {2 40} {4 120} {7 200} -radix unsigned
run 320
```

Figure A.1

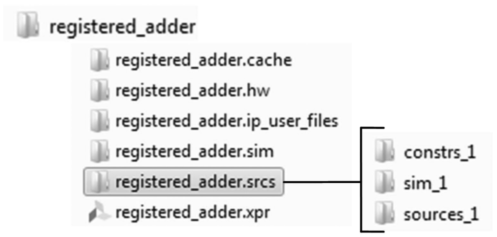


Figure A.2

IMPLEMENTATION, and PROGRAM AND DEBUG (compare to the design flow described in chapter 5).

3. Entering (and Testing) the Design File

Here, we must enter our VHDL design file (*registered_adder.vhd*, figure A.1b) The compiler will check the syntax and compile the code at register transfer level (RTL) level (no synthesis or placement yet), subsequently showing the corresponding elaborated design (i.e., the circuit, as understood from the VHDL code). The resulting schematic is equivalent to RTL View in Quartus Prime.

a) Under PROJECT MANAGER, click **Add Sources**, which opens the window of figure A.5a. Mark **Add or create design sources** and click **Next**.

b) In the next screen, click **Create File** (or click **Add Files** if the file is already available).

c) In figure A.5b, select **VHDL** and enter the file name (*registered_adder*), then click **OK**.

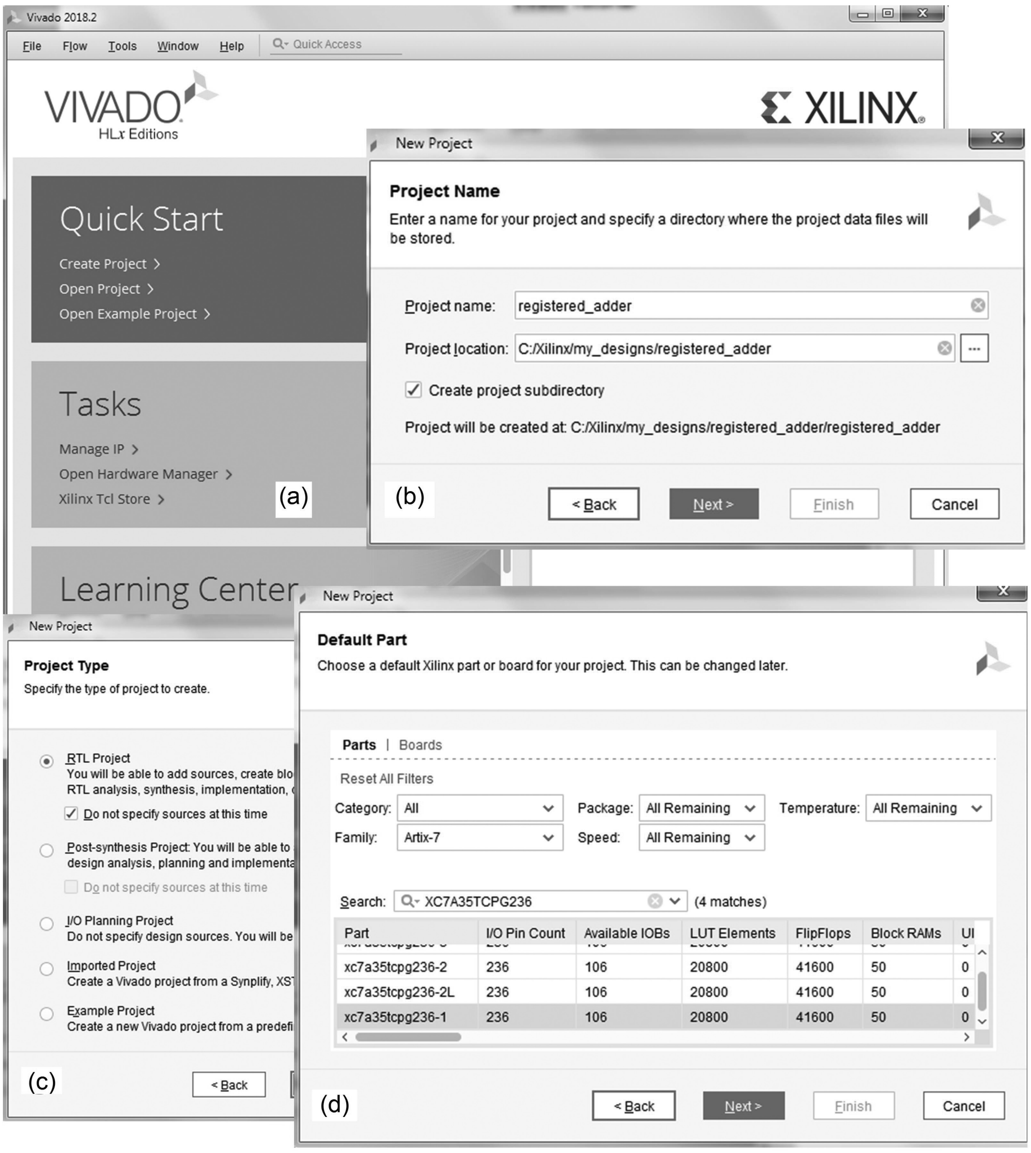
d) In figure A.5c, enter the entity name (*registered_adder*) and the architecture name (*rtl*). Click **OK** and then **Finish**.

e) In figure A.5d, note that **registered_adder…** is included in the Design Sources list and in the Simulation Sources list. Double click the former, which opens the editor (figure A.5f). Type the VHDL file (*registered_adder.vhd*, figure A.1b) and save it by clicking 💾.

f) In figure A.5e, open the General tab and select **Type: VHDL 2008**.

g) A very important feature of Vivado is that errors in the code of nonsupported VHDL constructs are underlined in red. Introduce an intentional error in the code to observe that.

h) RTL Analysis: We can now check how our code was understood by Vivado. Under RTL ANALYSIS, click **Open Elaborated Design**; when done, click **Schematic**. The resulting RTL view is shown in figure A.6, which matches the circuit of figure A.1a.



New Project

Project Name

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

Project name: registered_adder

Project location: C:/Xilinx/my_designs/registered_adder

✓ Create project subdirectory

Project will be created at: C:/Xilinx/my_designs/registered_adder/registered_adder

New Project

Project Type

Specify the type of project to create.

● RTL Project
You will be able to add sources, create blo
RTL analysis, synthesis, implementation, c

✓ Do not specify sources at this time

○ Post-synthesis Project: You will be able to
design analysis, planning and implementa

Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be

○ Imported Project
Create a Vivado project from a Synplify, XST

○ Example Project
Create a new Vivado project from a predefi

New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Parts | Boards

Reset All Filters

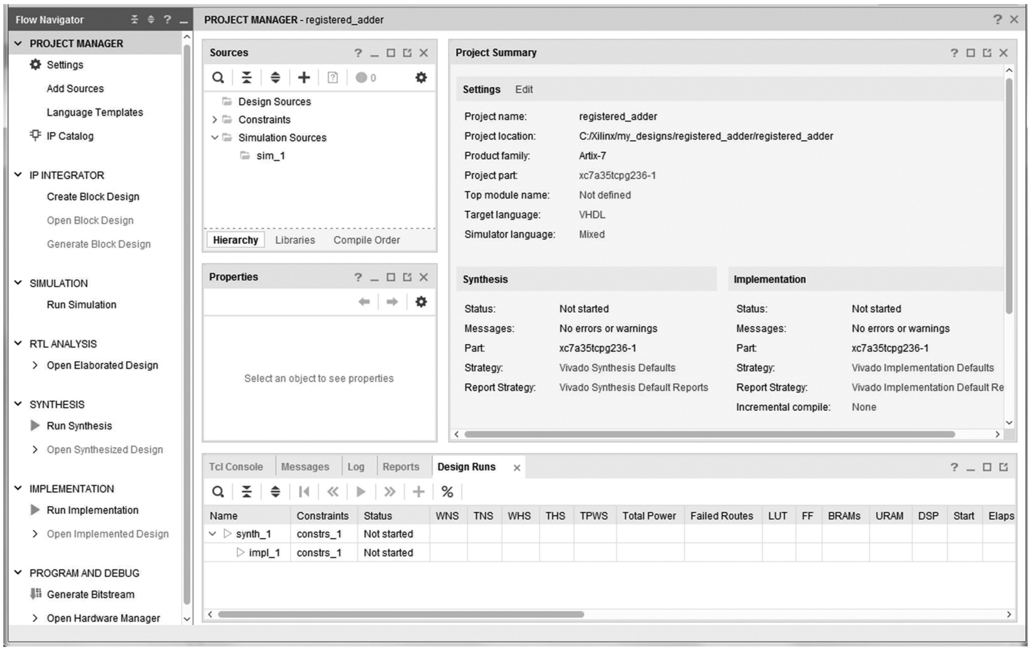
Category: All Package: All Remaining Temperature: All Remaining

Family: Artix-7 Speed: All Remaining

Search: XC7A35TCPG236 (4 matches)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	UI
xc7a35tcpg236-2	236	106	20800	41600	50	0
xc7a35tcpg236-2L	236	106	20800	41600	50	0
xc7a35tcpg236-1	236	106	20800	41600	50	0

Figure A.3



Project Summary — Settings

Project name:	registered_adder
Project location:	C:/Xilinx/my_designs/registered_adder/registered_adder
Product family:	Artix-7
Project part:	xc7a35tcpg236-1
Top module name:	Not defined
Target language:	VHDL
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Re
Incremental compile:	None

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elaps
synth_1	constrs_1	Not started														
impl_1	constrs_1	Not started														

Figure A.4

4. Doing Behavioral Simulation with Testbench

Notes:

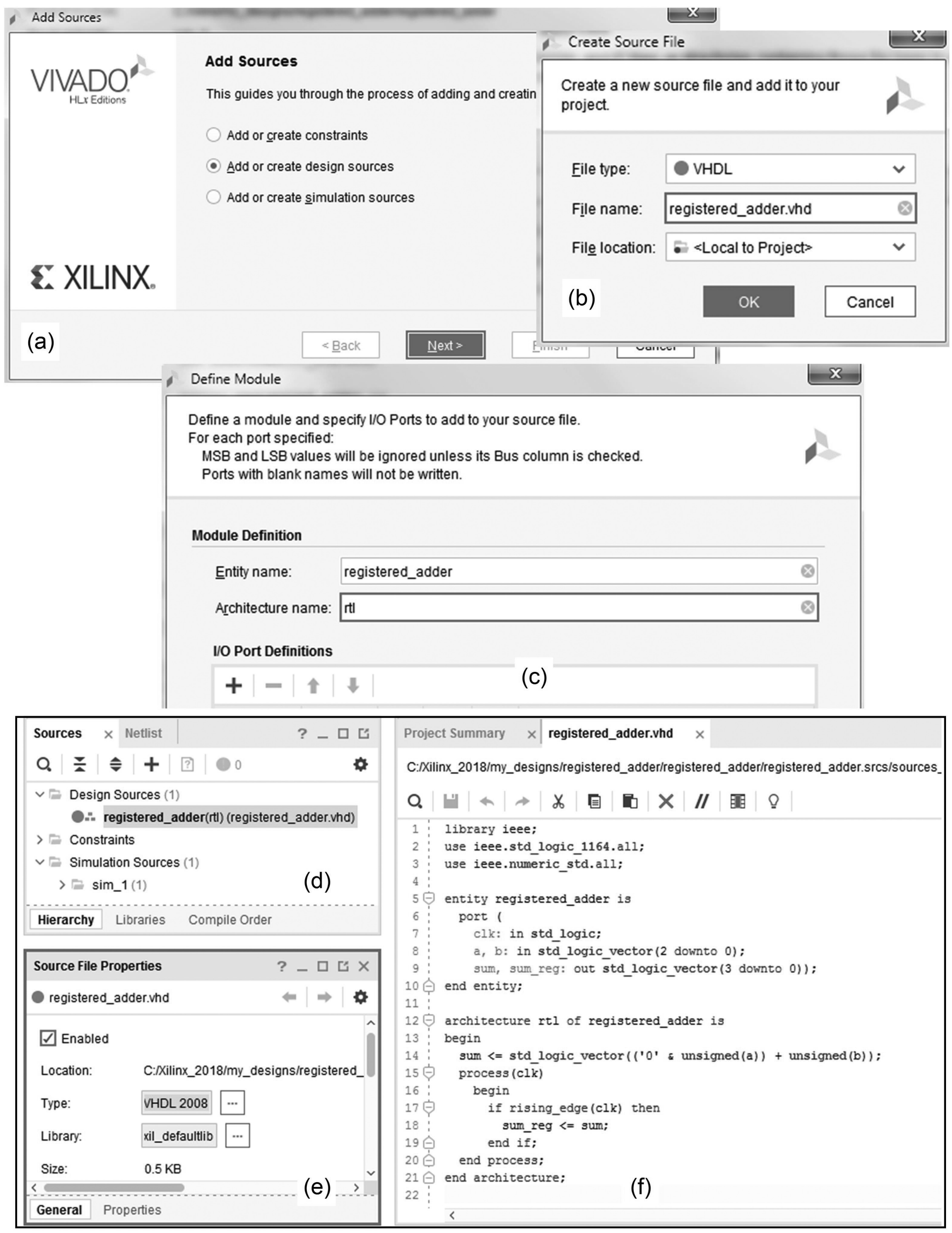
1) Recall that functional simulation, still at the RTL stage, is called *behavioral* simulation. After Synthesis or after Synthesis plus Implementation, its equivalent is called *functional* simulation. *Timing* simulation only exists for the latter two cases.

2) In simple designs, one might opt for skipping behavioral and even functional simulation. Timing simulation is always indispensable.

3) The simulation described here uses a VHDL testbench (see chapter 18; the testbench file is that of figure A.1d). Another option, described in the next section, is to use a Tcl script.

a) The first step is to enter the VHDL testbench file. Under PROJECT MANAGER, click **Add Sources**, which opens the window of figure A.5a. This time, mark **Add or create simulation sources** and click **Next**.

b) In the next screen, click **Create File** (or click **Add Files** if the file is already available).



Add Sources

This guides you through the process of adding and creatin

- Add or create constraints
- [✓ Add or create design sources]
- Add or create simulation sources

Create a new source file and add it to your project.

File type:	VHDL
File name:	registered_adder.vhd
File location:	<Local to Project>

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name:	registered_adder
Architecture name:	rtl

I/O Port Definitions

Design Sources (1)
registered_adder(rtl) (registered_adder.vhd)
Constraints
Simulation Sources (1)
sim_1 (1)

registered_adder.vhd

✓ Enabled	
Location:	C:/Xilinx_2018/my_designs/registered_
Type:	VHDL 2008
Library:	xil_defaultlib
Size:	0.5 KB

C:/Xilinx_2018/my_designs/registered_adder/registered_adder/registered_adder.srcs/sources_

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity registered_adder is
  port (
    clk: in std_logic;
    a, b: in std_logic_vector(2 downto 0);
    sum, sum_reg: out std_logic_vector(3 downto 0));
end entity;

architecture rtl of registered_adder is
begin
  sum <= std_logic_vector(('0' & unsigned(a)) + unsigned(b));
  process(clk)
    begin
      if rising_edge(clk) then
        sum_reg <= sum;
      end if;
  end process;
end architecture;
```

Figure A.5

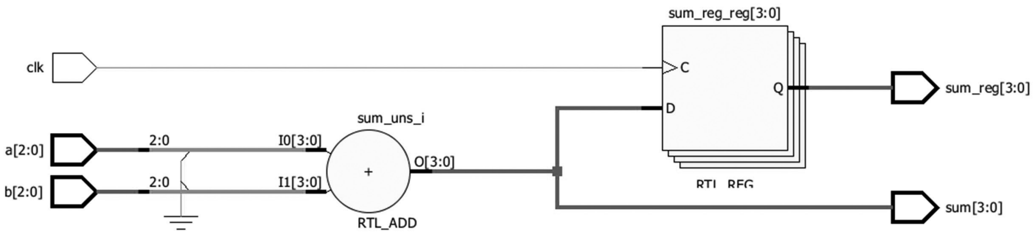


Figure A.6

c) In figure A.5b, select **VHDL** and enter the file name (*registered_adder_tb.vhd*), then click **OK**.

d) In figure A.5c, enter the entity name (*registered_adder_tb*, figure A.1b) and the architecture name (*testbench*). Click **OK** and then **Finish**.

e) In figure A.5d, note in the Sources pane that **registered_adder_tb - ...** is added to the Simulation Sources list. Double click it, which opens the editor (figure A.5f). Type the testbench file (*registered_adder_tb.vhd*, figure A.1d) and save it by clicking 💾.

f) In figure A.5e, open the General tab and select **Type: VHDL 2008**.

g) We can now run the simulation. Under SIMULATION, select **Run Simulation > Run Behavioral Simulation**, which leads to figure A.7a. Move *clk* to the top if not there yet.
Note: To break a simulation, select **Run > Break**.

h) Make the following adjustments:

- Change the time to 320 ns at 320 ns.
- Click the **Restart** icon ⏮.
- Click the **Run for time T** icon ▶(T).
- Click the **Zoom Fit** icon ⛶.
- Select all signals except *clk* of figure A.7a, click the right mouse button, and change the radix to **Radix > Unsigned Decimal**.

The final result is shown in figure A.7b.

i) Finally, inspect the simulation results of figure A.7b and confirm that they comply with figure A.1c.

j) Click ▶(T) and then ⛶ for the simulation to advance another 320 ns.

k) To end a simulation, close the wave pane or type **close_sim –force** in the Tcl console.

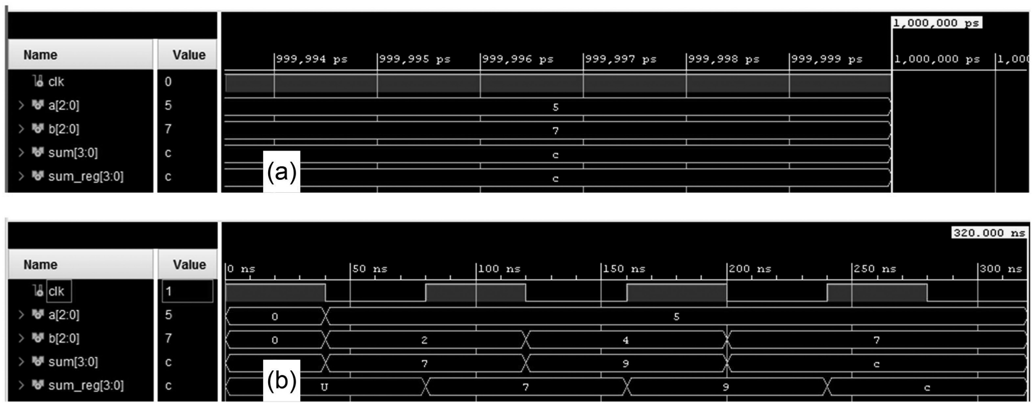


Figure A.7

5. Doing Behavioral Simulation with Tcl Script

This section shows how to run a simulation using tool command language (Tcl, pronounced "tickle") scripts.

Note: If you prefer, you can remove the testbench file from the project by right-clicking on the file name and selecting **Remove File from Project**.

a) Review the Notes at the beginning of part 4 of this appendix.

b) The first step is to prepare the Tcl script (check appendix A1, *Some Important Tcl Commands for Vivado*, at the end of this tutorial). The script of figure A.1e will be used here. Assuming again that the clock period in figure A.1c is 80 ns, the total running time is 320 ns.

c) You can type the script in the Tcl console (**Window > Tcl Console**) one line at a time, or you can save it in a text file and run it all at once. For the former, proceed in (d); for the latter, jump to (g).

d) Under SIMULATION, select **Run Simulation > Run Behavioral Simulation**. This opens the waveforms pane of figure A.7a. Move *clk* to the top if not there yet.

e) Enter the Tcl commands. After entering **run 320** (and clicking , if necessary), the screen of figure A.7b will be displayed.

f) Now that the simulation is done, play with the simulator by doing parts (h)–(k) of this appendix's part 4.

g) The Tcl script of figure A.1e can be typed in a text editor and saved (call it *registered_adder.tcl*) in the same folder the testbench file was saved before (i.e., *registered_adder.srcs/sim_1/new*). Another option is to use Vivado's editor as follows.

h) Under PROJECT MANAGER, click **Add Sources**, which leads to figure A.5a. Mark **Add or create simulation sources** and click **Next**.

i) In the next screen, click **Create File**.

j) In figure A.5b, select **Memory File** and enter the file name (*registered_adder.tcl*), then click **OK**.

k) Note in figure A.5d that this new file name appears under Memory File in the Simulation Sources list. Click on it; this opens the editor. Type the script, and save the file. Finally, in figure A.5e, change Memory File to **TCL**.

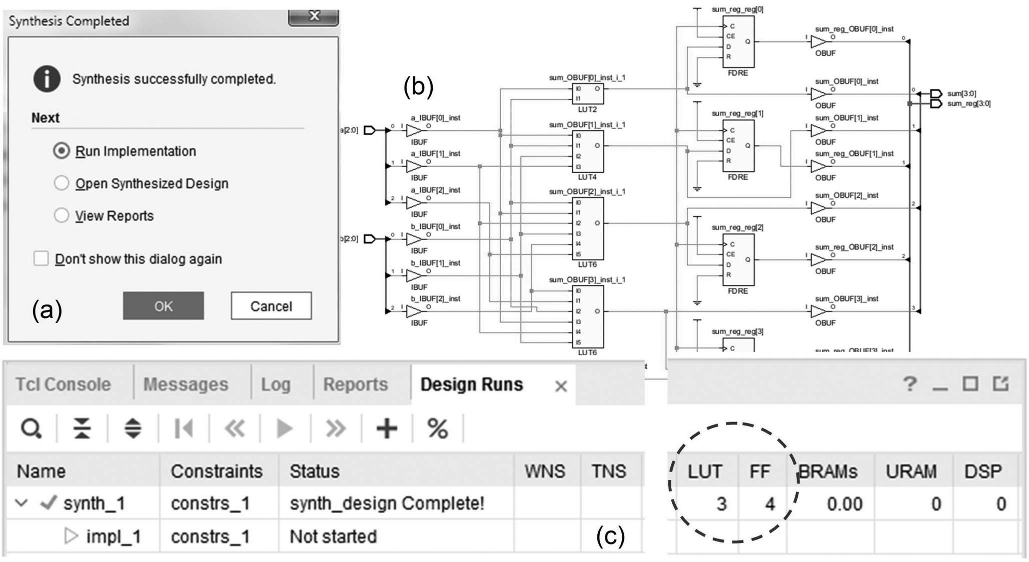
l) Run the Tcl file by selecting **Tools** > **Run Tcl Script** and pointing to the Tcl file. The result is that of figure A.7b.

m) Now that the simulation is done, play with the simulator by doing parts (h)–(k) of part 4 of this appendix.

6. Synthesizing the Design

a) Under SYNTHESIS, click **Run Synthesis**.

b) When finished, the window of figure A.8a opens. Cancel Implementation for now.



Synthesis Completed

Synthesis successfully completed.

Next

[✓ Run Implementation]

Open Synthesized Design

View Reports

Don't show this dialog again

OK Cancel

Name	Constraints	Status	WNS	TNS	LUT	FF	BRAMs	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!			3	4	0.00	0	0
impl_1	constrs_1	Not started							

Figure A.8

c) Under SYNTHESIS, click on **Open Synthesized Design**, then open **Schematic**, which shows the circuit after synthesis (figure A.8b), which is the circuit that will actually be fitted and routed in the final (Implementation) phase.

d) Select **Window > Design Runs** or open the Design Runs tab in the lower part of the main window to see the resources usage. As shown in figure A.8c, three lookup tables (LUTs) and four flip-flops were employed to build this circuit.

7. Implementing the Design

a) Under IMPLEMENTATION, click **Run Implementation**.

b) When finished, observe that information similar to that in figures A.8b and A.8c is given here.

c) Observe also in the Project Summary window (click Σ if it is not open) that now **Synthesis Status = Complete** and **Implementation Status = Complete**. The resources usage is also summarized at the bottom of the screen.

8. Doing Functional Simulation with Testbench

Follow this appendix's part 4, except for part 4(g), in which you must select one of the following:

Simulation > Run Simulation > Run Post-Synthesis Functional Simulation or

Simulation > Run Simulation > Run Post-Implementation Functional Simulation.

9. Doing Functional Simulation with Tcl Script

Follow part 5, except for part 5(d), in which you must select one of the following:

Simulation > Run Simulation > Run Post-Synthesis Functional Simulation or

Simulation > Run Simulation > Run Post-Implementation Functional Simulation.

10. Doing Timing Simulation with Testbench

Follow part 4, except for part 4(g), in which you must select one of the following:

Simulation > Run Simulation > Run Post-Synthesis Timing Simulation or

Simulation > Run Simulation > Run Post-Implementation Timing Simulation.

Timing simulation results are shown in figure A.9a. In figure A.9b, a zoomed-in view is shown, so propagation delays can be clearly observed.

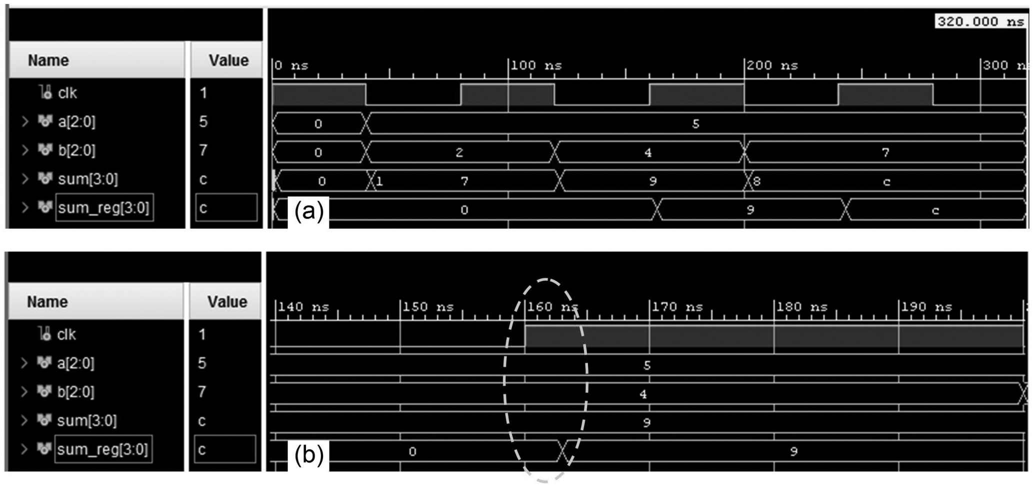


Figure A.9

11. Assigning Pins

a) Select **Flow > Open Implemented Design**.

b) Select **Window > I/O Ports** or click the **I/O Ports** tab at the bottom of the screen, which leads to figure A.10.

c) In the **Package Pin** column, enter the names of the pins to which the circuit ports should be routed. Mark them as **Fixed**.

d) Update the design by selecting **Implementation > Run Implementation** (this will include resynthesis).

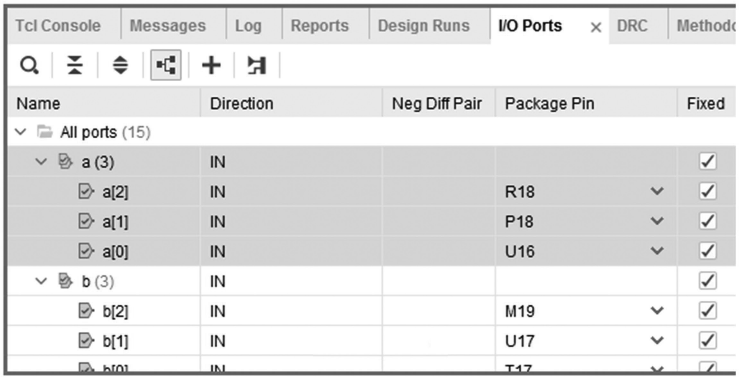
12. Programming the FPGA

a) Select **Flow > Open Implemented Design**.

b) Under PROGRAM AND DEBUG, select **Generate Bit Stream**.

c) Under PROGRAM AND DEBUG, select **Open Hardware Manager** to program the FPGA.

d) Finally, play with the FPGA board to verify whether the implementation works as expected.



Name	Direction	Neg Diff Pair	Package Pin	Fixed
All ports (15)				
a (3)	IN			✓
a[2]	IN		R18	✓
a[1]	IN		P18	✓
a[0]	IN		U16	✓
b (3)	IN			✓
b[2]	IN		M19	✓
b[1]	IN		U17	✓
b[0]	IN		T17	✓

Figure A.10

Appendix A1. Some Important Tcl Commands for Vivado

- Command **add_wave**: Adds waveforms to the wave pane.

  Examples:

  add_wave clk Adds wave *clk* to the wave pane.

  add_wave clk -after_wave rst Adds wave *clk* to the wave pane after the *rst* wave.

  add_wave clk inp outp Adds waves *clk*, *inp*, and *outp* to the wave pane.

  add_wave / Adds all ports in the design to the wave pane.

  add_wave sum –radix dec Adds wave *sum* to pane with *dec* radix.

  Note: Allowed radix values are bin (default), unsigned, dec (signed decimal), hex, oct, ascii.

- Command **add_force**: Defines the shape and radix of the waveforms.

  Examples:

  add_force clk {1 0} {0 40} –repeat_every 80 –cancel_after 2000

  Wave *clk* has value = 1 at time = 0, then 0 at 40 ns, repeats after 80 ns, resulting $T = 80$ ns, and stops after 2 μs.

  add_force clk {1} {0 40} –repeat_every 80 –cancel_after 2000

  Same as above (time = 0 does not need to be specified).

  add_force inp1 {2} {5 40} –radix unsigned

  Wave *inp1* is 2 at time = 0, then 5 for time = 40 ns and higher, with unsigned radix (see radix options above).

  add_force inp2 {–3} {3 40} {–8 250} –radix dec

Wave *inp2* is –3 at time = 0, 3 at 40 ns, then –8 at 250 ns, signed decimal radix.

- Command **launch_simulation**: Opens a simulation.
- Command **run**: Runs a simulation.

  Examples:

  run 700 ns Runs simulation for 700 ns.

  run 700 Same as above (default time unit is ns).
- Command **close_sim –force**. Ends simulation without saving waveforms.
- Command **restart**: Restarts a simulation.
- Command **current_time**: Gets the current time in the simulation.
- Command **open_project**: Opens a project.

  Example: **open_project c:/xilinx/my_designs/registered_adder.xpr**
- Command **open_report**: Displays or copies to an output file the contents of an RPX file.

  Examples:

  open_report –file results1 design1.rpx Copies the contents of *design1.rpx* to file *results1*.

  open_report design1.rpx Shows the contents of *design1.rpx* in the Tcl console.

Tcl script example: See figure A.1e, which contains the stimuli of figure A.1c.