



Lab 1

Setting up a Vivado Project for MIPSfpga



These materials produced in association with Imagination.
Join our University community for more resources.

community.imgtec.com/university

Lab 1: Setting up a Vivado Project

1. Introduction

This is the first in a series of laboratory exercises acquainting you with system-on-chip design using Imagination's MIPSfpga platform. In this lab you will learn to set up a Vivado project for simulating, synthesizing, and downloading the MIPSfpga system onto Digilent's Nexys4 DDR FPGA board. As you make changes to the MIPSfpga system in the future, you can follow these steps to compile, simulate, synthesize, download, and test your changes.

MIPSfpga can also be ported to other FPGA boards, for example Digilent's Basys3 board. Section 3 shows how to port MIPSfpga to that board, and those instructions can be used as a guide to port to other boards.

The instructions in this lab use Vivado 2016.2. Instructions for later versions of Vivado are similar, if not exactly the same.

2. Setting up a Vivado Project

In this section we walk through the steps of (1) creating a project for the MIPSfpga system, (2) simulating the project, (3) compiling the project, and (4) downloading the MIPSfpga system onto the Nexys4 DDR board.

Before setting up the Vivado project, make a copy of the MIPSfpga system by copying the **rtl_up** folder in the MIPSfpga_GSG directory (provided with the MIPSfpga Getting Started materials) to **MIPSfpga_Labs\rtl_up**.

The Verilog files in the **MIPSfpga_GSG\rtl_up** directory describe the MIPSfpga system and are the design source files for the Vivado project you are about to create. In later MIPSfpga labs, you will extend the functionality of the MIPSfpga system by both modifying and adding Verilog files to the MIPSfpga_Labs\rtl_up folder.

Step 1. Create Vivado project

Start Vivado. Open a new project by choosing **File → New Project** (see [Figure 1](#)).

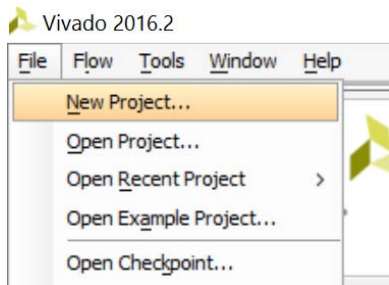


Figure 1. Create new Vivado project

Click **Next**. Browse to the MIPSfpga_Labs\Labs\Xilinx\Part1_Intro\Lab01_Vivado folder and place the new project, **Project1**, in that folder, as shown in [Figure 2](#). Click the Create Project subdirectory box and click Next, as shown below.

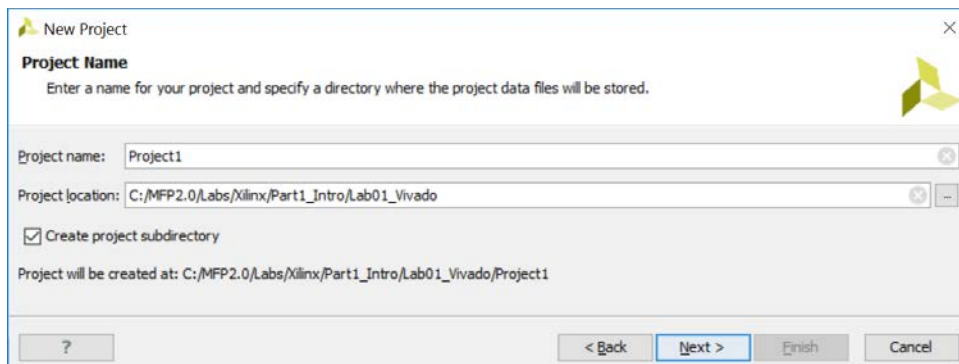


Figure 2. Create Vivado project directory

In the next window, leave **RTL Project** selected and click **Next**. Now in the **Add Sources** window, click on **Add Directories** ([Figure 3](#)).

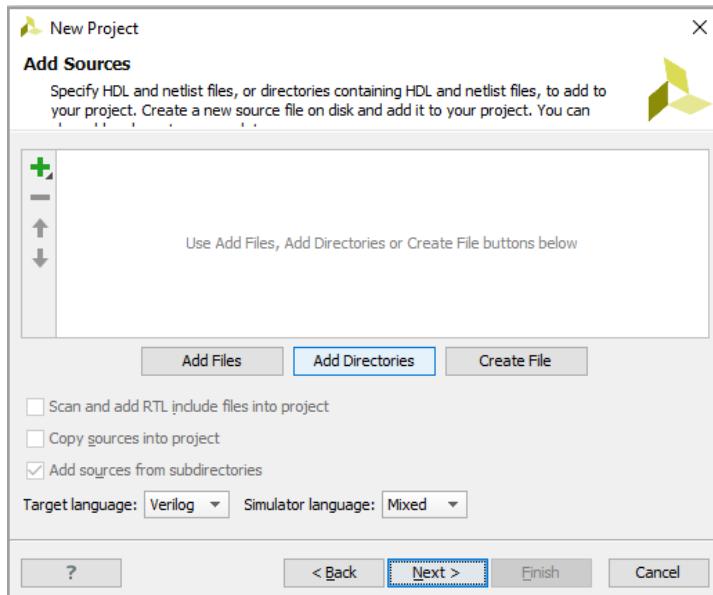


Figure 3. Add Directories of Verilog files

Browse to the **MIPSfpga_Labs\rtl_up** directory. Select the **core, system, boards\nexys4_ddr, and testbench** directories, as shown in [Figure 4](#).

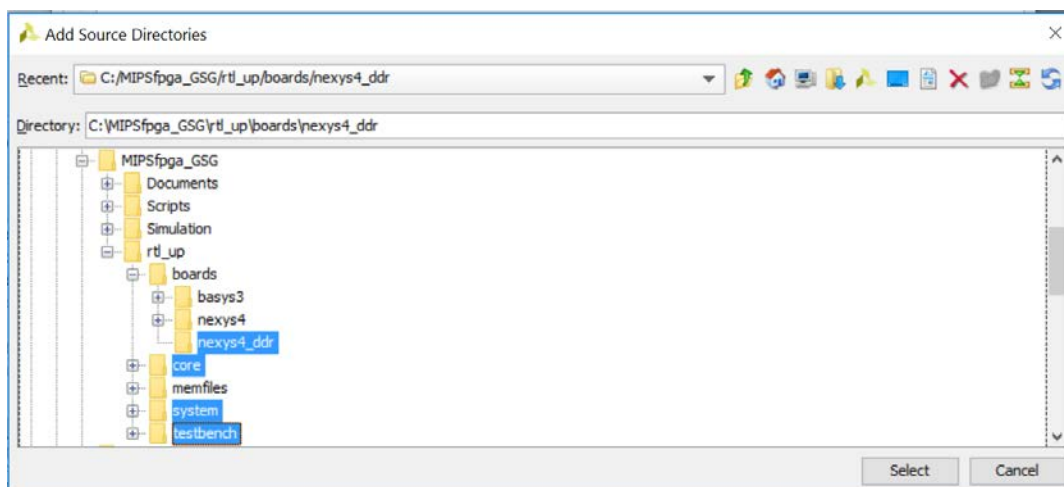


Figure 4. Adding Verilog files to project

Now click on **Add Directories** again and add the **MIPSfpga_Labs\rtl_up\boards\nexys4_ddr** directory.

In the Add Sources window, make sure the **Copy the sources into Project** box is **not** selected (see [Figure 5](#)). The project should refer to the Verilog (.v) and Verilog header (.vh) files located in the MIPSfpga_Labs\rtl_up directory – do **not** make a local copy of the files in the Vivado project. Also make sure the Add sources from subdirectories box is selected, and click **Next**.

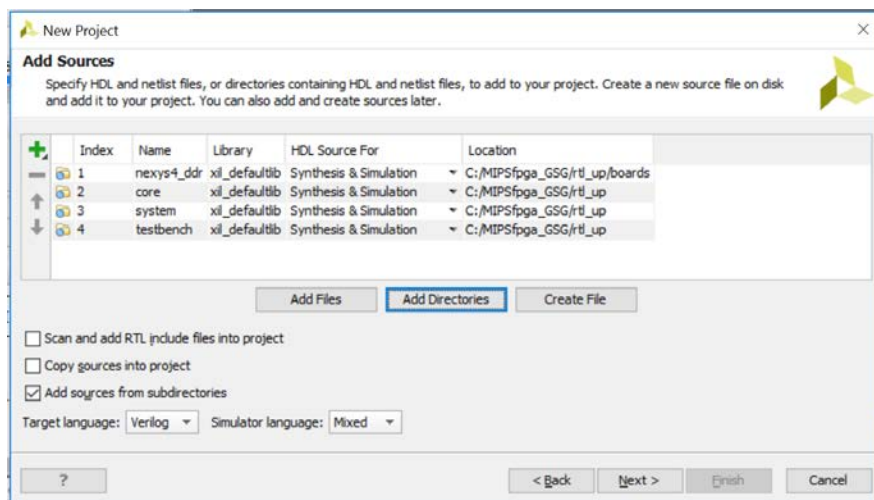


Figure 5. Adding Sources – do not copy sources into project

You will not add any IP, so click **Next** in the **Add Existing IP (optional)** window.

In the **Add Constraints (optional)** window, click on **Add Files**. Browse to the MIPSfpga_Labs\rtl_up\boards\nexys4_dds directory, and select the **mfp_nexys4_dds.xdc** file and click **OK** (see Figure 6). This constraints file maps the Verilog signals to pins on the FPGA and describes timing constraints.

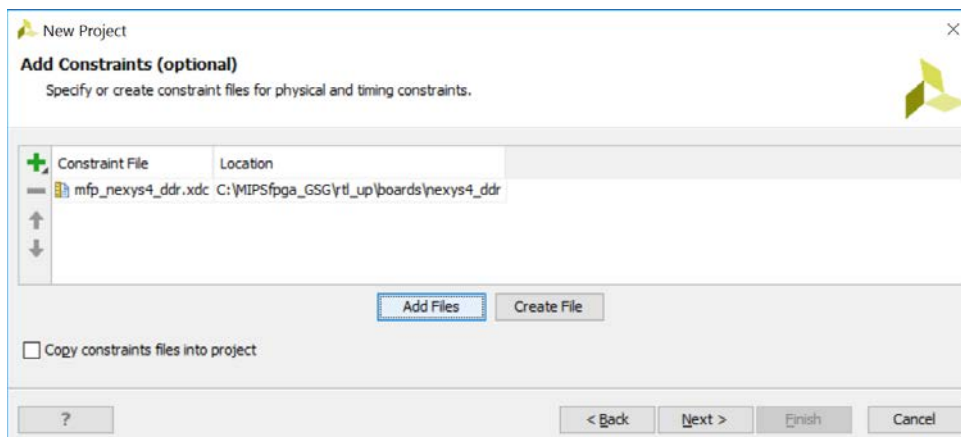


Figure 6. Add Xilinx Design Constraints (.xdc) file to Vivado project

Click on the **Copy constraints files into project** box (see Figure 7) and click **Next**.

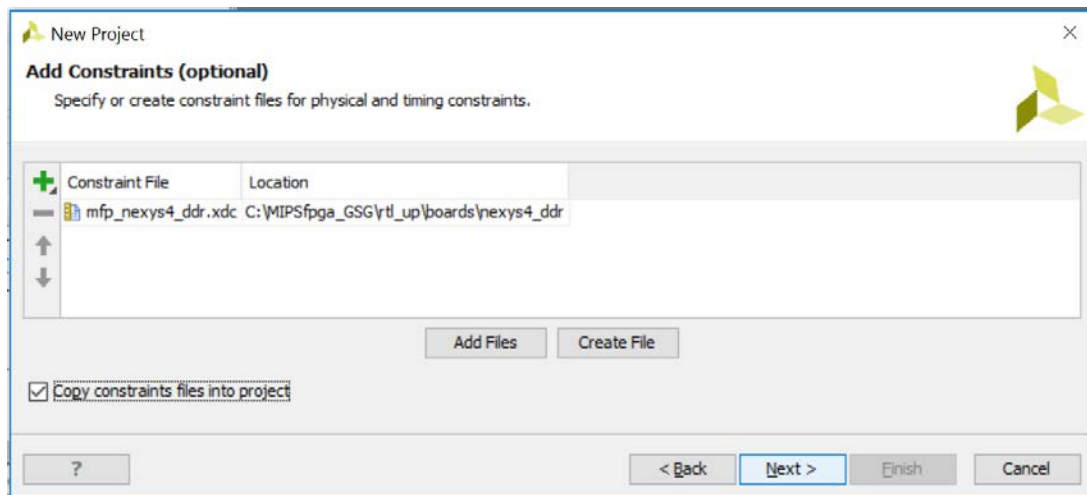


Figure 7. Copy .xdc file into Vivado project

Now you will choose the Artix-7 FPGA that is on the Nexys4 DDR board as the target. Type (or copy-paste) the following into the search box: **xc7a100tcsg324-1**, as shown in [Figure 8](#). Select the part, as shown, and click **Next**.

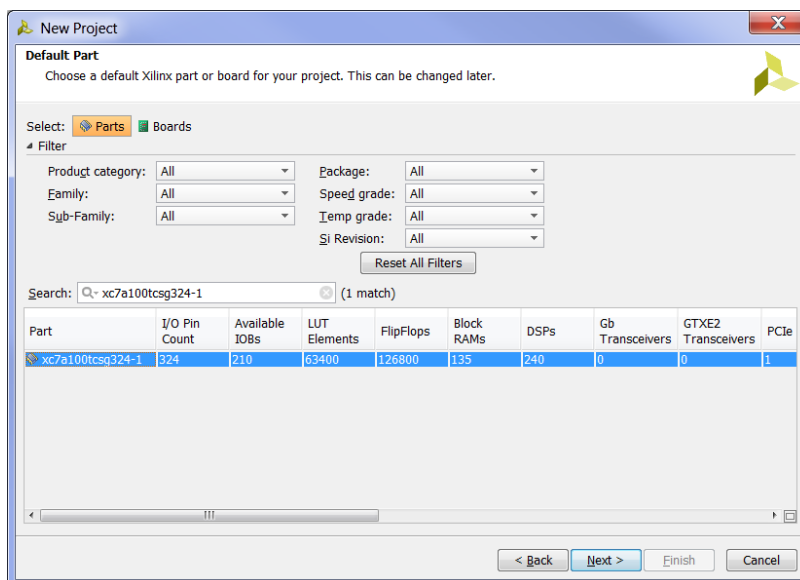


Figure 8. Selecting the Artix-7 FPGA

"xc7a" indicates that it is an Artix-7 FPGA. "100t" says that it has about 100k Logic Cells. "csg324" indicates a "chip scale ball grid array (BGA)" package with 324 pins, and "-1" is the speed grade.

Now click **Finish** in the New Project Summary window (see [Figure 9](#)).

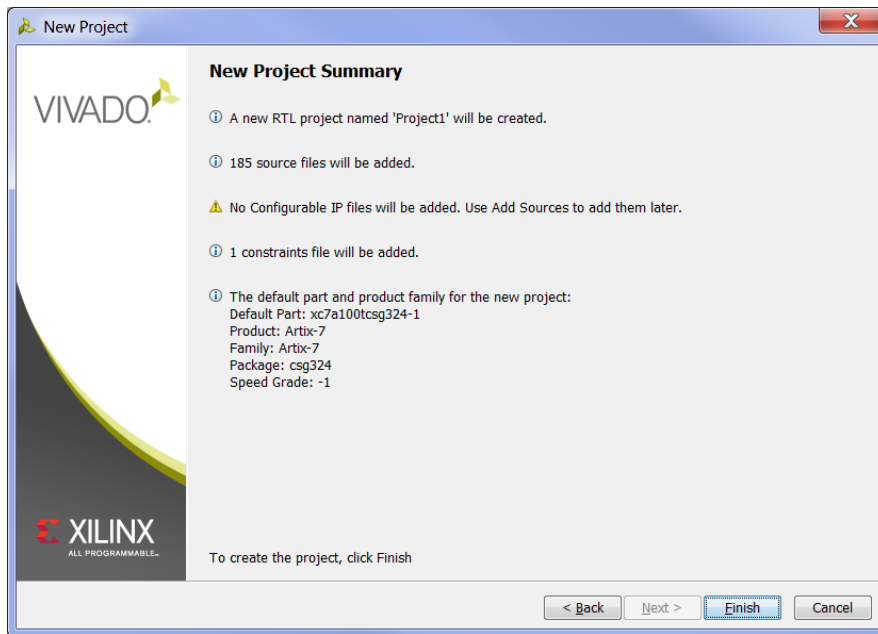


Figure 9. New Project Summary window

After the project initializes, notice the modules listed in the Project Manager Sources window. This lists the hierarchy (modules and sub-modules) of the Project. The **mfp_nexys4_ddr** module should be bold, which indicates that it is the top-level module (see Figure 10). If it is not highlighted, you can right-click on that module and select **Set as Top** in the pull-down menu. This will set that module as the top-level module to synthesize, compile, and download to the Nexys4 DDR board.

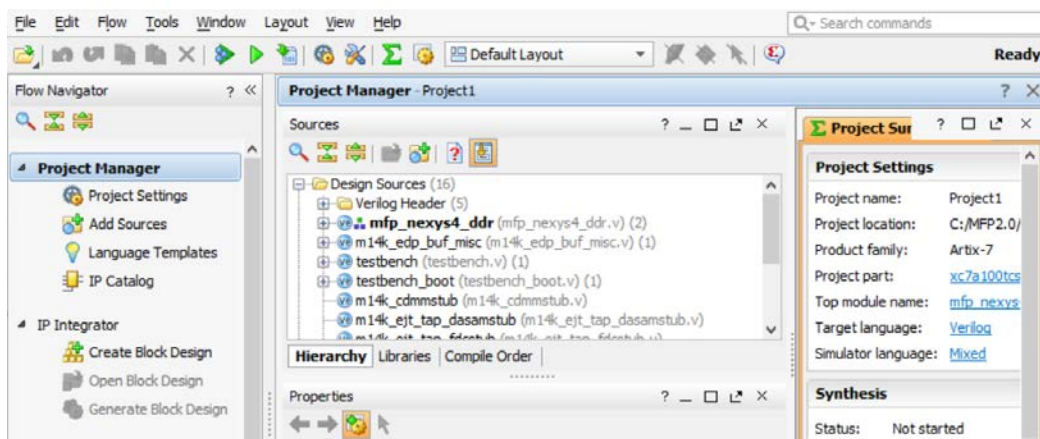


Figure 10. mfp_nexys4_ddr as top-level module for synthesis, implementation, and bitstream generation

The last step of creating the project is to add a PLL (or mixed-mode clock manager – MMCM) that reduces the on-board 100 MHz clock to 50 MHz to meet timing constraints. To create the PLL, click on **IP Catalog** under Project Manager in the Flow Navigator, as shown in Figure 11.

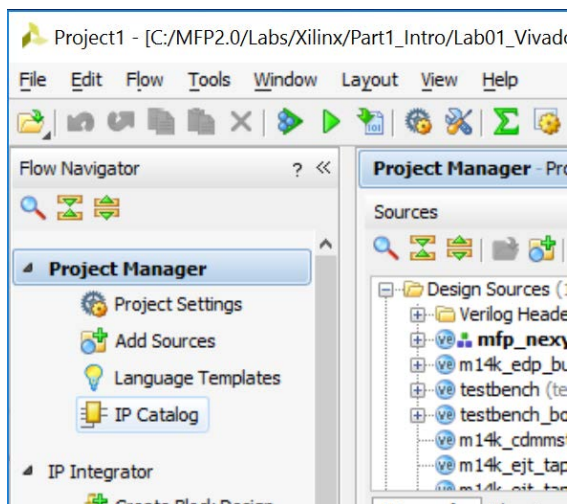


Figure 11. IP Catalog

Now, in the IP Catalog tab of the Project Manager pane, expand **FPGA Features and Design**, and then expand **Clocking**. Double-click on **Clocking Wizard**, as shown in Figure 12.

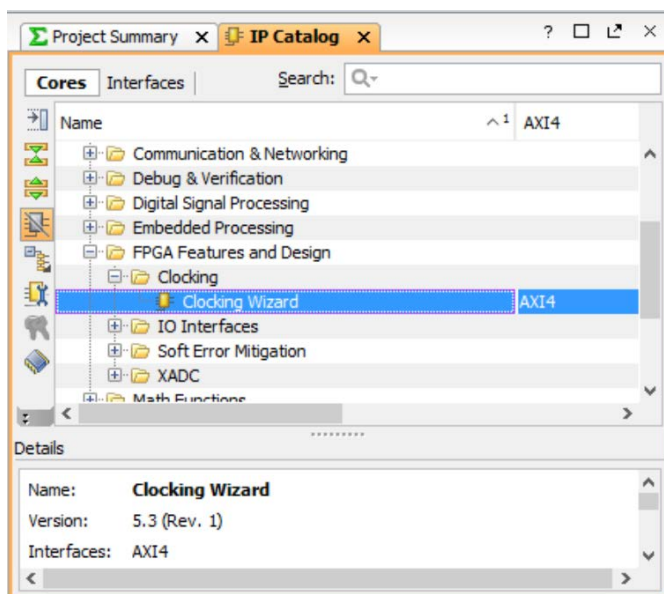


Figure 12. Clocking Wizard

The Clocking Wizard window will pop up, as shown in Figure 13. You can select either MMCM or PLL (shown). Leave the Input Clock information as the default (100 MHz), as shown in Figure 13.

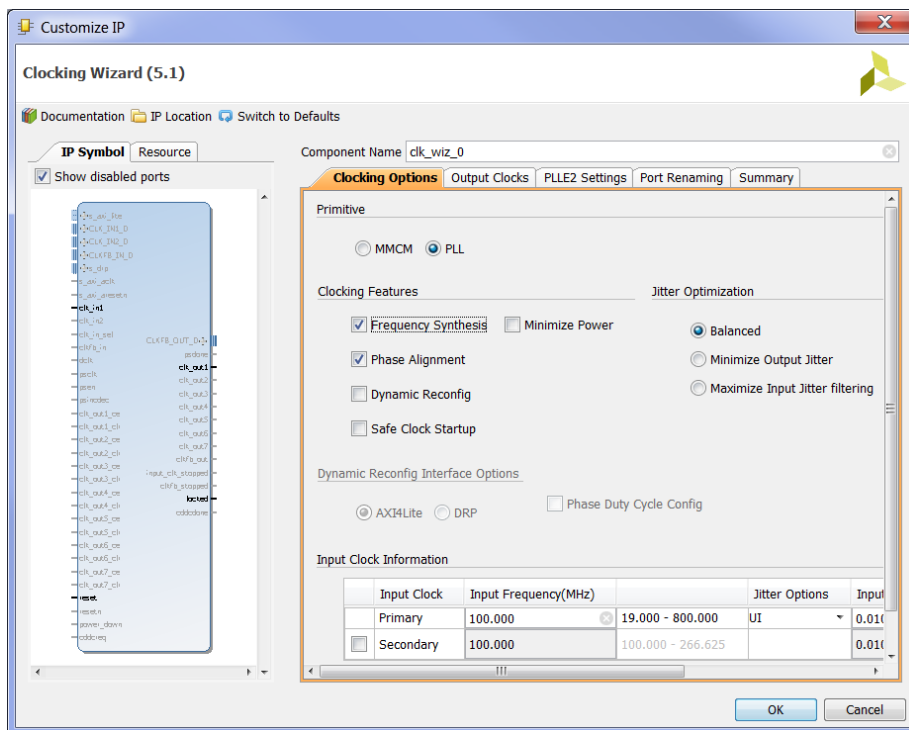


Figure 13. Clocking Wizard window

Now click on the **Output Clocks** tab, and type in **50** as the output frequency in the Output Freq (MHz) Requested box for clk_out1, as shown in Figure 14.

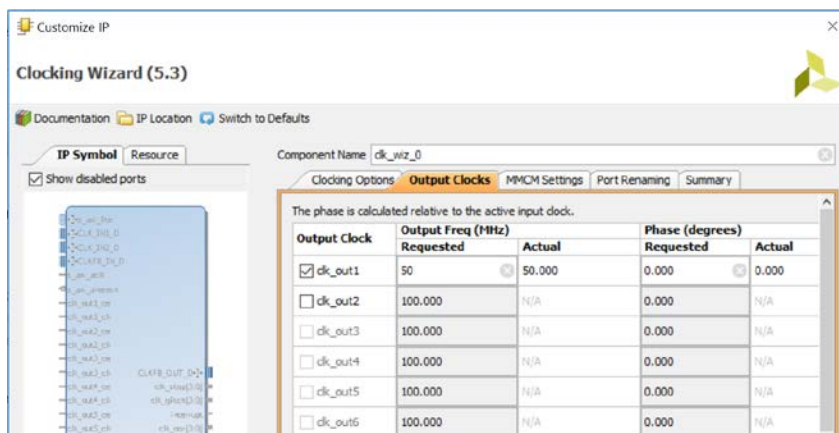


Figure 14. Select output clock frequency

Scroll down in the same tab (Output Clocks) and **deselect** reset and locked, as shown in Figure 15. Then click **OK** to complete the creation of the PLL (or MMCM).

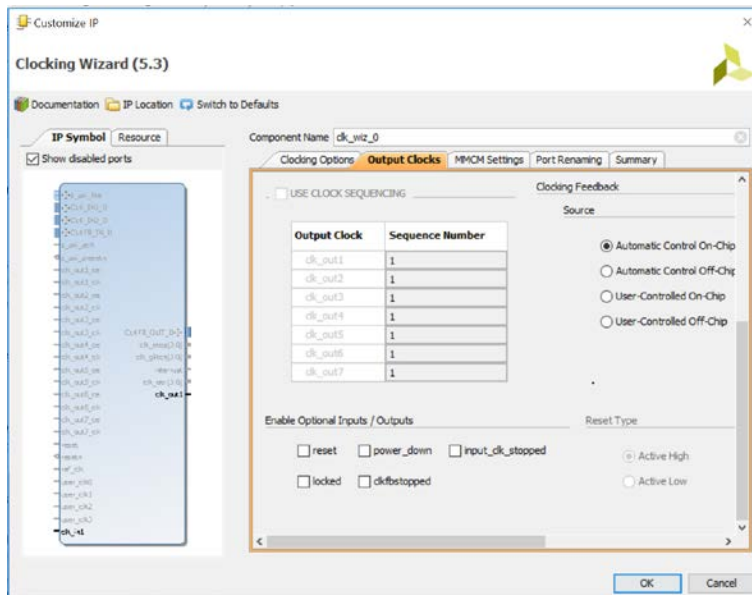


Figure 15. Deselect reset and locked

A pop-up window will prompt you to "Generate Output Products", as shown in Figure 16. Click **Generate**.

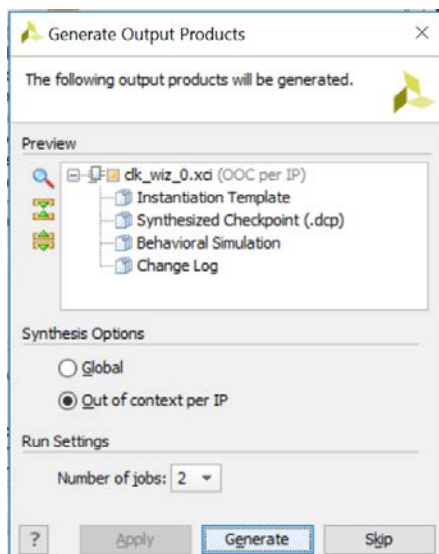


Figure 16. Generate PLL

A window will pop up that says "Out-of-context module run was launched for generating output products," as shown in Figure 17. Click **OK**.



Figure 17. Out-of-context generation of PLL

Step 2. Simulating MIPSfpga

Now you are ready to simulate the MIPSfpga system. You will use Vivado's built-in simulator called XSIM. We already added the testbench.v file when we created the project, and now we will make it the top-level module for simulation. In the Project Manager panel, scroll down to **Simulation Sources** and expand it and the **sim_1** folder. Right-click on testbench.v and **set it as the top-level module** for simulation, as shown in [Figure 18](#). Notice that the testbench entry is now bold.

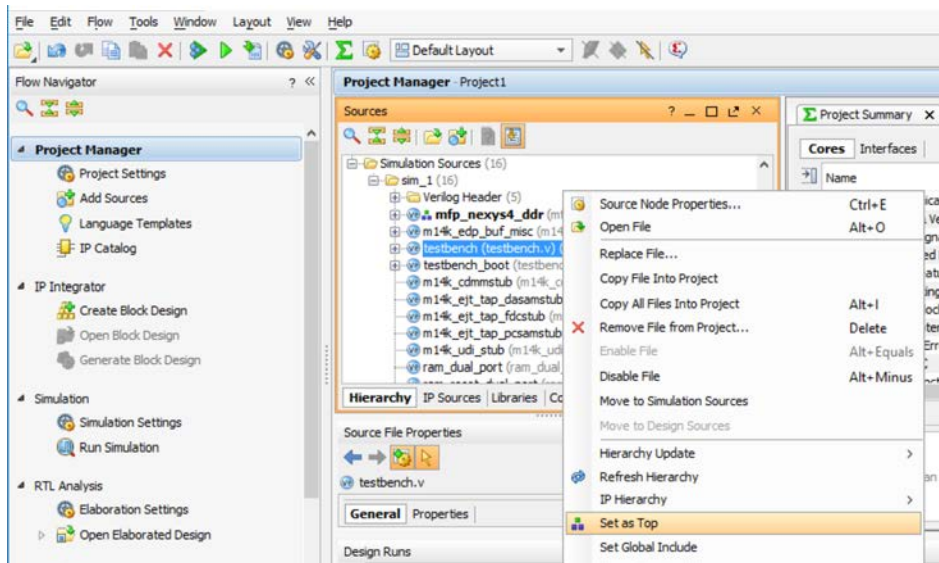


Figure 18. Setting the top-level module for simulation

Now add the memory files that define the program. Click on **Add Sources** in the Flow Navigator window on the left, select **Add or create simulation sources** option (see [Figure 19](#)), and then click **Next**.

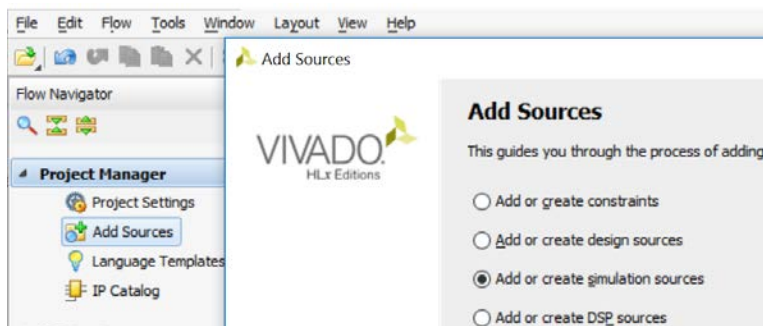


Figure 19. Add simulation sources

Click on **Add Files**, select **All Files** in the *Files of type* filter, browse to MIPSfpga_Labs\rtl_up\memfiles\1_IncrementLEDs, and select (shift-click) all of the .txt files: ram_b0.txt, ram_b1.txt, ram_b2.txt, and ram_b3.txt, and click **OK** (see [Figure 20](#)).

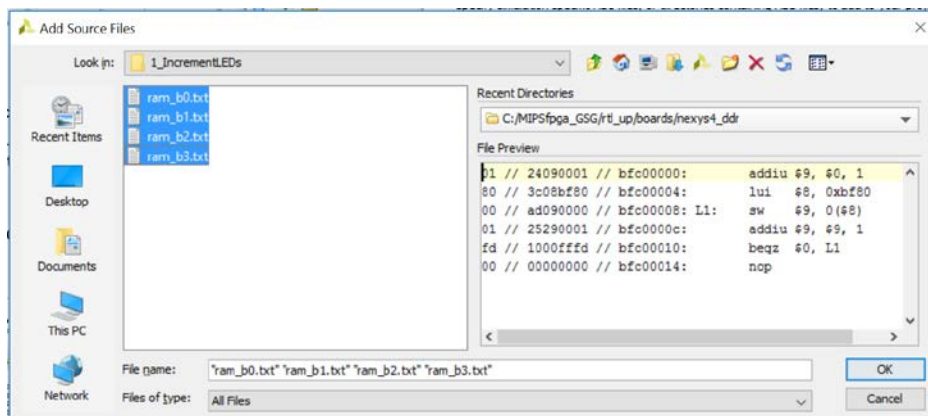


Figure 20. Selecting simulation files that define the memory contents

Leave the Copy sources into project box unselected, but leave the Include all design sources for simulation box checked, as shown in [Figure 21](#). Then click **Finish**.

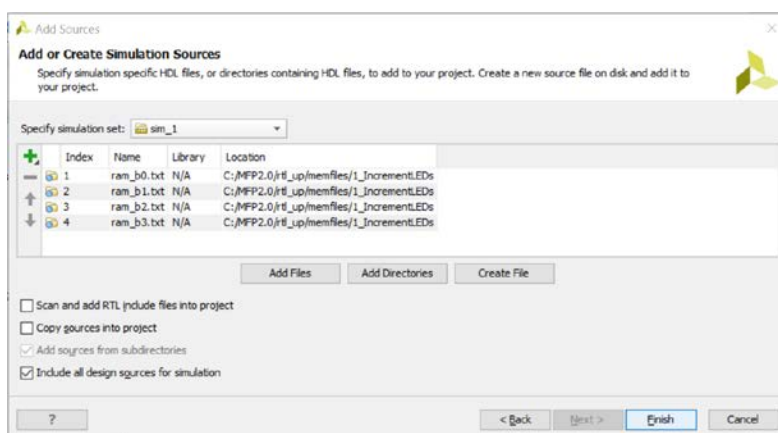


Figure 21. Adding simulation source

These text files contain the instructions that will be loaded into MIPSfpga's memory. ram_b0.txt – ram_b3.txt show the byte-wide versions of the memories. ram_b0.txt holds the least significant byte (LSB) of each instruction, ram_b1.txt the next most significant byte, and so on. As you might recall from the MIPSfpga Getting Started Guide, this program, as shown again in [Figure 22](#) for your convenience, writes incremented values to memory address 0xbf800000. Also recall from the MIPSfpga Getting Started Guide that the LEDs on the Nexys4 DDR board are mapped to memory address 0xbf800000. So the program writes incremented values to the LEDs.

// C code <pre> unsigned int val = 1; volatile unsigned int* dest; dest = 0xbf800000; while (1) { *dest = val; val = val + 1; } </pre>	# MIPS assembly code <pre> # \$9 = val, \$8 = mem address 0xbf800000 addiu \$9, \$0, 1 # val = 1 lui \$8, 0xbf80 # \$8=0xbf800000 L1: sw \$9, 0(\$8) # mem[0xbf800000] = val addiu \$9, \$9, 1 # val = val+1 beqz \$0, L1 # branch to L1 nop # branch delay slot </pre>
---	---

Figure 22. IncrementLEDs program

The equivalent machine code for the IncrementLEDs program is given in [Figure 23](#).

Machine Code	Instruction Address	Assembly Code
24090001	// bfc00000:	addiu \$9, \$0, 1 # val = 1
3c08bf80	// bfc00004:	lui \$8, 0xbf80 # \$8=0xbf800000
ad090000	// bfc00008:	L1: sw \$9, 0(\$8) # mem[0xbf800000] = val
25290001	// bfc0000c:	addiu \$9, \$9, 1 # val = val+1
1000fffd	// bfc00010:	beqz \$0, L1 # branch to L1
00000000	// bfc00014:	nop # branch delay slot

Figure 23. MIPS machine code

Expand the hierarchy under *Simulation Sources* and observe that these four text files are added in a separate sub-folder called Text and it contains the machine code (see [Figure 24](#)). Now you are ready to run the simulation of the MIPSfpga system running that program.

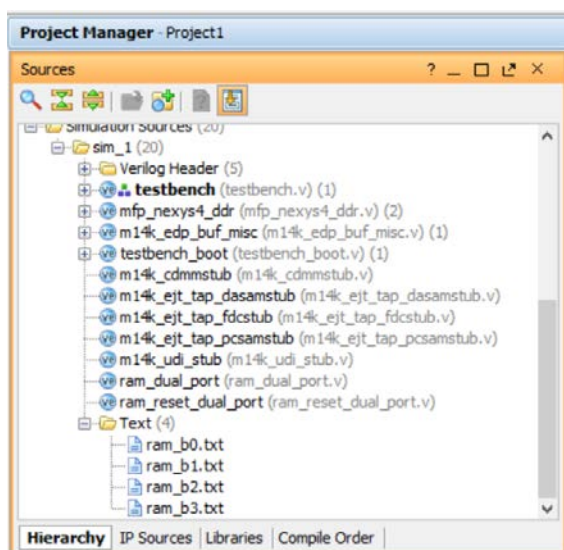


Figure 24. Text file as simulation source

Click on **Simulation Settings** in the Flow Navigator window on the left, as shown in [Figure 25](#).

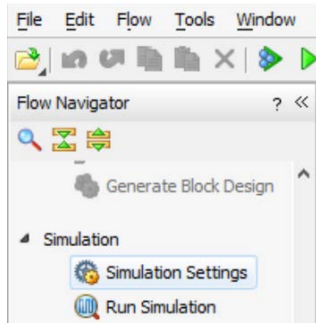


Figure 25. Simulation Settings

The simulation settings window will show up, as shown in [Figure 26](#). Click on the Simulation tab and set **the simulation run time to 2000 ns**. Click **OK**.

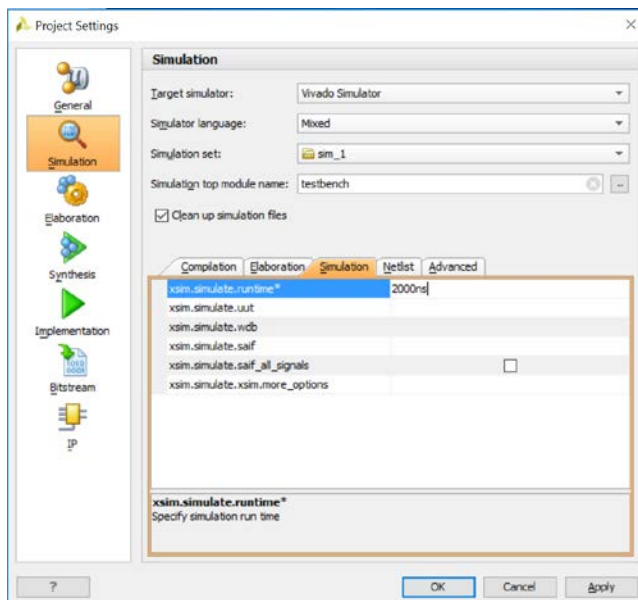


Figure 26. Change simulation run time

Click on **Run Simulation** → **Run behavioral simulation** in the Flow Navigator window, as shown in [Figure 27](#).

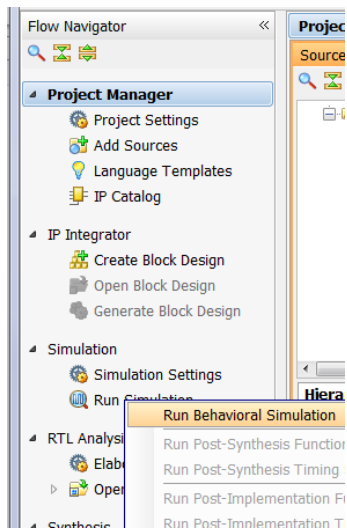



Figure 27. Run simulation

The testbench and lower-level modules will compile, the simulation window will open, and the simulation results will be displayed, as shown in Figure 28. The simulation waveform shows the top-level signals of the top-level module, in this case the testbench module. Click on the Zoom Fit button ().

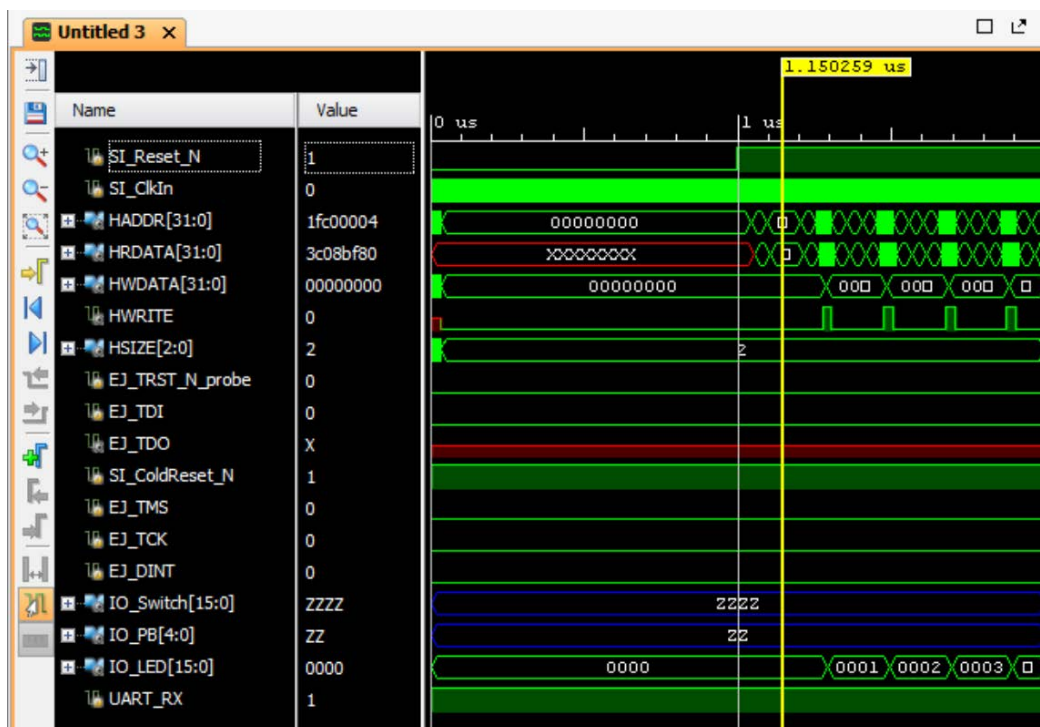


Figure 28. Simulation results showing top-level signals

The radix of the signals defaults to hexadecimal, but you could change the radix of any multi-bit

signal by selecting it in the waveform window and then right-clicking and selecting **Radix** → **Binary**, or whichever representation you prefer. Use shift-click and ctrl-click to select multiple signals at a time.

Delete all of the EJTAG signals and some of the I/O signals, specifically delete:

EJ_TRST_N_probe, EJ_TDI, EJ_TDO, SI_ColdReset_N, EJ_TMS, EJ_TCK, EJ_DINT, IO_Switch, IO_PB, and UART_RX. Do **not** delete IO_LED. Right-click on a signal (or group of signals) and select Delete. (Or simply select the signals and press the Delete key.) Again, you can also select multiple signals using shift-click and ctrl-click.

The waveform window will now resemble Figure 29.

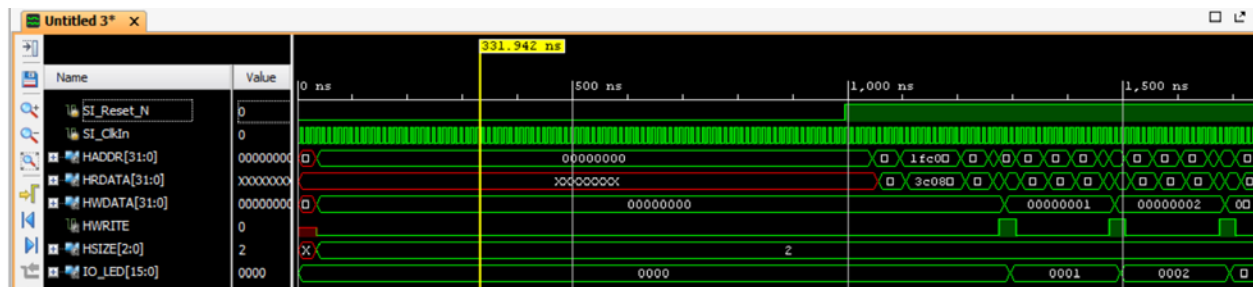







Figure 29. Keeping only the desired top-level signals

You can float the waveform window by clicking on the float button () and then maximize it by clicking on the full size button (left one ). Click on the Zoom Fit button to see the waveform completely. You can also use Zoom In (), Zoom Out (), and Zoom to Cursor () buttons to view a desired section of the waveform.

You can view signals from lower-level modules by adding them to the waveform. For example, as shown in [Figure 30](#), expand the **testbench** hierarchy to **testbench** → **sys** → **mfp_ahb_withloader** → **mfp_ahb** → **mfp_ahb_b_ram** → **ram_b0** to see the **ram_b0** module in the **Scopes** window. Click on the **ram_b0** module to see the corresponding signals in the **Objects** window. This module holds the least significant byte of the instructions in the boot RAM.

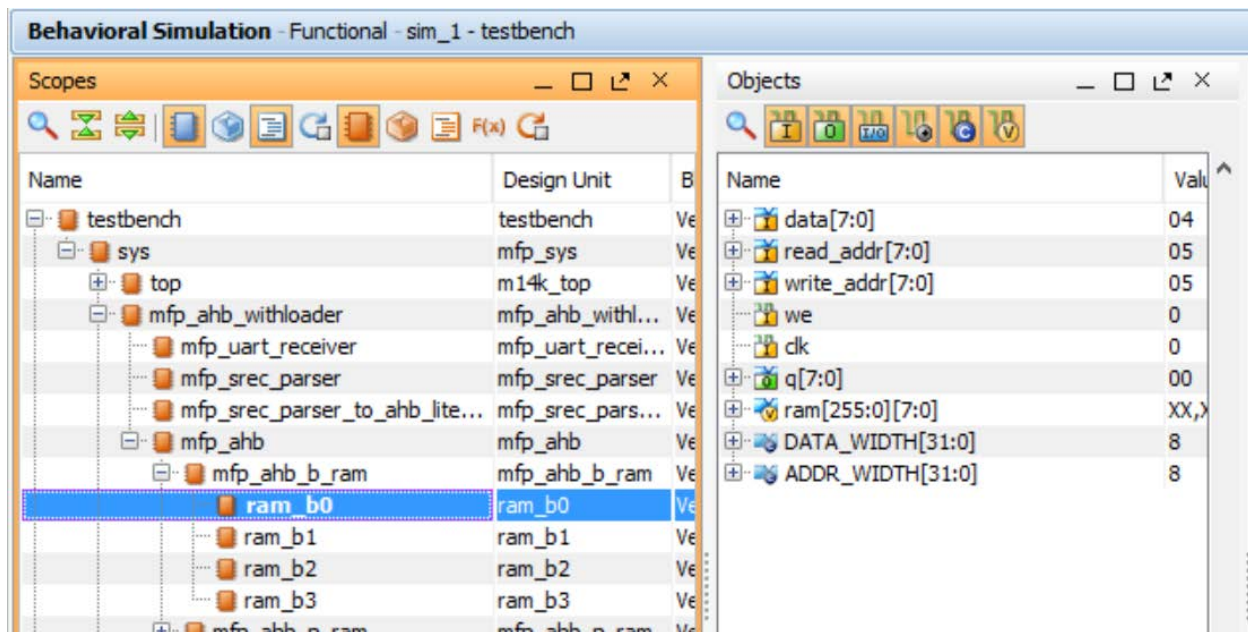




Figure 30. Accessing the signals of lower-level modules

In the waveform window, right-click in the signals area below the last signal, and select **New Divider**. The New Divider dialog box will appear. Type **Boot RAM0** in the field and press **Return**.

Select all of the objects in the **Objects** window (Ctrl-a), then right-click and select **Add to Wave Window** and observe that the signals are added to the Waveform window. In the tool buttons

bar, change the run time to 4 us . Now click on the Restart button .

and then click on the Run for <time> button  to reset and run the simulation for 4 us. You will see the output as shown in Figure 31.

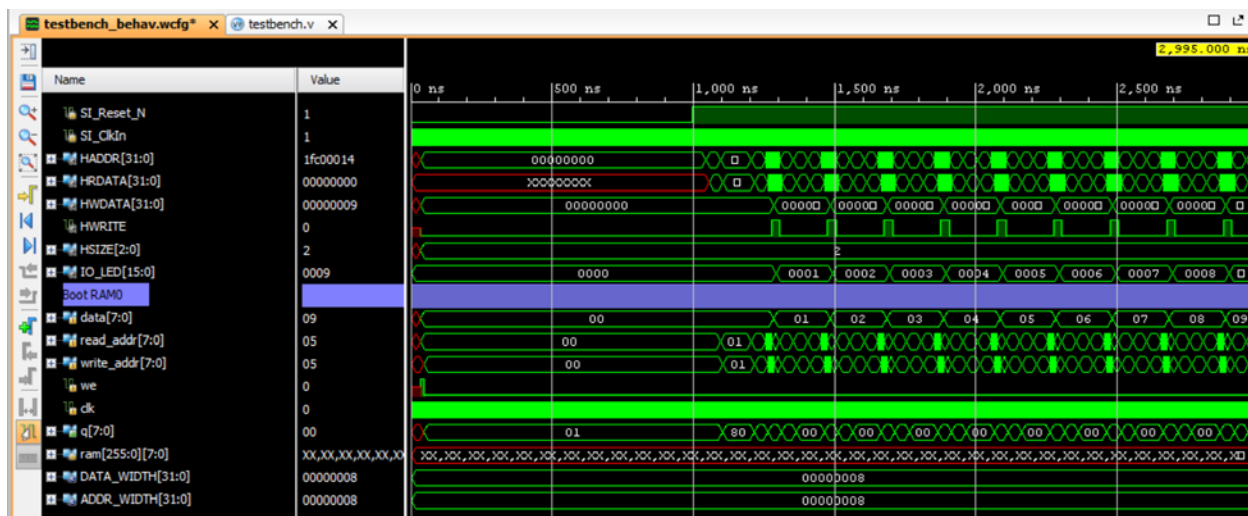



Figure 31. Simulation result showing lower-level module (ram_b0) signals

At first the processor is reset: the low-asserted reset signal `SI_Reset_N` is low. Just after reset (when `SI_Reset_N` transitions from 0 to 1), you can view the waveform as it fetches each instruction starting with instruction (physical) address `0x1fc00000`. This address shows up on `HADDR` and the instruction read from memory appears on the `HRDATA` bus one cycle later. Recall that virtual address `0xbfc00000` translates to physical address `0x1fc00000`. The instructions are executed in sequence until the branch is taken at `0xbfc00010`. The code then continuously repeats from `0xbfc00008` – `0xbfc00014`. Recall from the MIPSfpga Getting Started Guide, that because this is a simple program and contains no boot code, the caches are not initialized, so each instruction takes 5 cycles. Also view how incremented values are written on the `HWDATA` signal as the code executes. `HWDATA` is the data being written to memory or, in this case, memory-mapped I/O. `IO_LED` displays the incremented value because it is memory-mapped to address `0xbf800000`. The `IO_LED` signals are connected to the pins that drive the Nexys4 DDR's LEDs.

After you are finished viewing the waveform, you can close the simulator by selecting **File** → **Close Simulation**. A pop-up window will appear asking if you want to save the waveform. You could select Save but for now, click **Discard**.

Step 3. Compiling MIPSfpga

Now you are ready to compile the MIPSfpga system and create a bitfile that you can download onto the Artix-7 FPGA on the Nexys4 DDR board. Click on the **Generate Bitstream** button  at the top of the window. The bitstream is a file that configures the FPGA to be the MIPSfpga system, as defined by the Verilog files. This file is also referred to as the *bitfile*. Now wait for synthesis, placement, routing, and bitstream generation to complete. This typically takes about 10-20 minutes, depending on your computer speed.

After the bitstream has been generated, you will see the Bitstream Generation Completed pop-up window, as shown in [Figure 32](#).

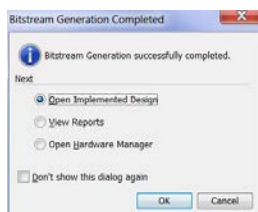


Figure 32. Bitstream Generation Completed pop-up window

Viewing the implemented design is optional, but gives some insight into the timing and layout of the MIPSfpga system. (If you don't want to view it, select Open Hardware Manager and click

OK. Then continue with Step 4 below.) To view the implemented design, leave **Open Implemented Design** selected, and click **OK**. This will take a few minutes. Then the Implemented Design window will open, as shown in Figure 33.

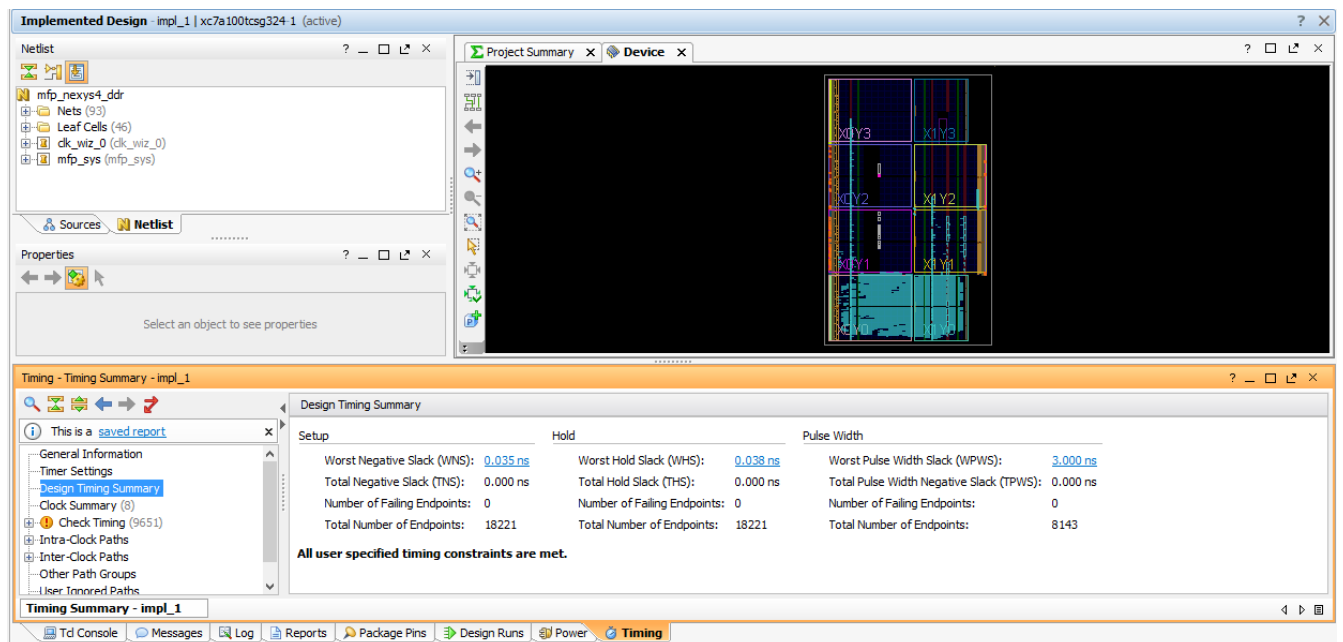


Figure 33. Implemented Design

Notice the **Design Timing Summary** pane at the bottom. Most importantly, it says that **All user specified timing constraints are met**. The Worst Negative Slack (**WNS**) is 0.035 ns and the Worst Hold Slack (**WHS**) is 0.038 ns, so there are no timing violations. The slack values for your project are likely slightly different. Each time Vivado places and routes the design, a different configuration results.

As you add to or modify the MIPSfpga system in the future, check that the timing constraints are met, and if not, reduce the frequency of the PLL and/or change the timing constraints in the Xilinx Design Constraints (.xdc) file until they are. The WNS will indicate how much the cycle time needs to be increased.

For example, Figure 34 shows a Project Summary page for a design that does **not** meet timing. Notice the negative values of WNS (shown in red). In this case the cycle time is too short by 4.91 ns, so add that amount (or slightly more) to the cycle time to meet timing. If the frequency of the failing design were 100 MHz (cycle time = 1/100 MHz = 10 ns), the cycle time would need to increase to at least (10 + 4.91) ns ≈ 15 ns (frequency = 1/15 ns ≈ 66 MHz).

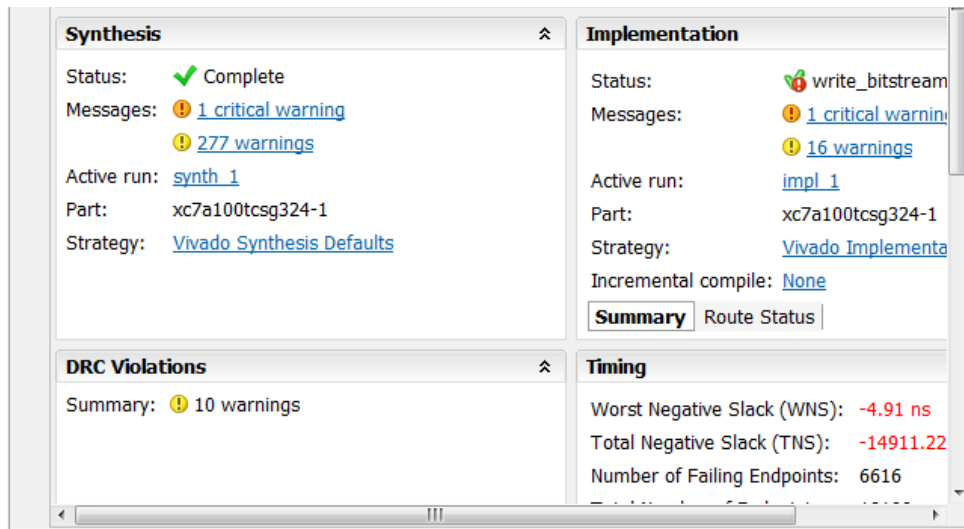


Figure 34. Design that does not meeting timing constraints

Step 4. Downloading MIPSfpga onto the Nexys4 DDR FPGA Board

Now you are ready to download the compiled design onto the Nexys4 DDR FPGA board. First, connect and turn on the Nexys4 DDR board. Figure 35 shows the board and highlights the power switch and the USB port. Plug the standard end of the programming cable into your computer and the micro-USB end of the programming cable into the board, at the location labeled "USB Programmer Port" in Figure 35. Now turn the board's power switch ON. If the board is factory configured, the board will run a pre-loaded program that writes to the 7-segment displays with a snake-like pattern that repeats indefinitely. To program the board, it can be in QSPI mode (as shown in Figure 35 with the left-most Mode pins connected by a jumper) or in JTAG mode (with the two middle pins of the Mode selector connected by a jumper).

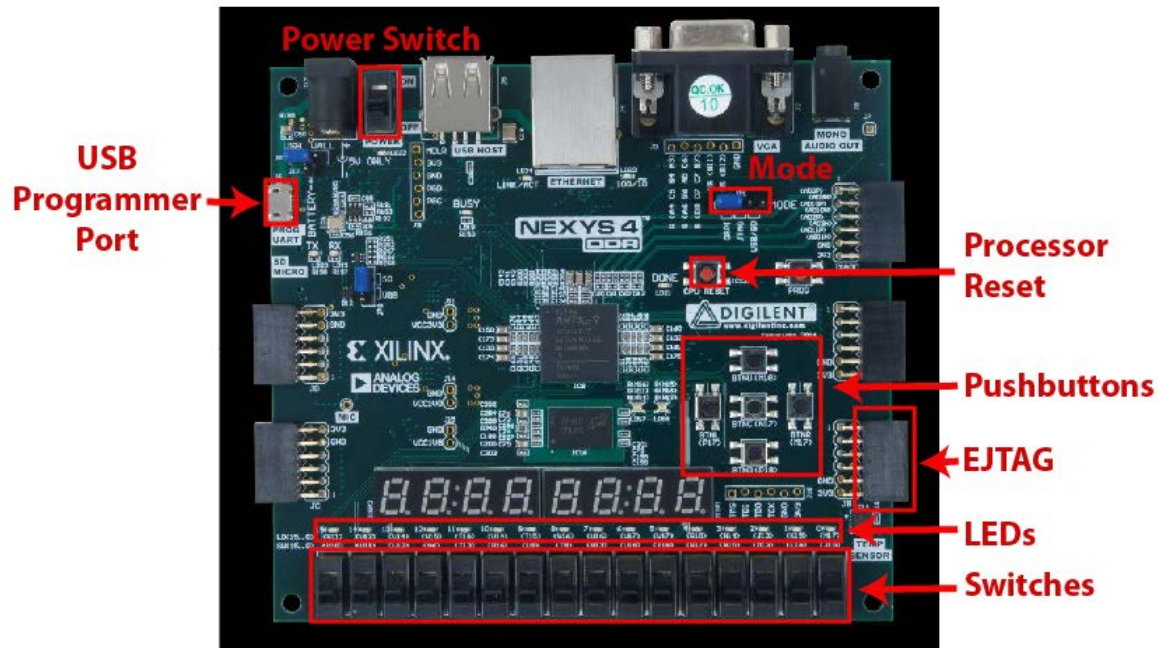


Figure 35. Nexys4 DDR board (photograph © Digilent Inc., 2015)

In the Vivado window, select **Flow** → **Open Hardware Manager**, as shown in Figure 36.

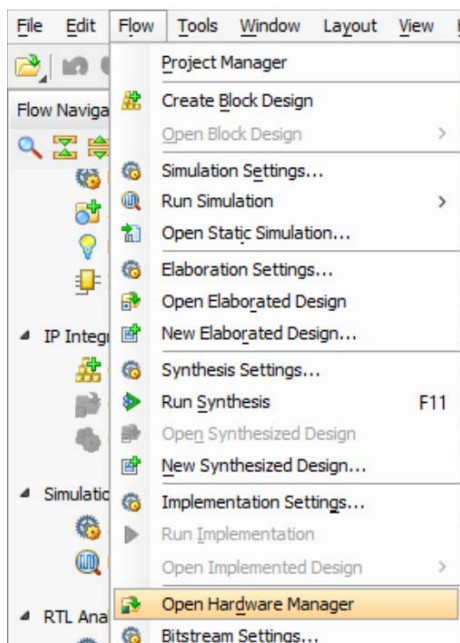


Figure 36. Open Hardware Manager

The Hardware Manager window will open. Now click on **Open Target** and choose **Auto Connect**, as shown in Figure 37.

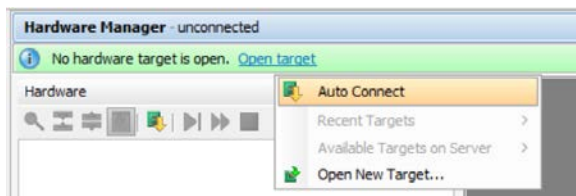


Figure 37. Hardware Manager window, Auto Connect

After you click on *Auto Connect*, Vivado takes several seconds to connect to the target FPGA on the Nexys4 DDR board. You will see the following warning, that you can ignore:

WARNING: [Labtools 27-3123] The debug hub core was not detected at User Scan Chain 1 or 3. ...

If you see the message "No hardware target is open," two causes are most likely:

1. You forgot to plug in the Nexys4 DDR board into your computer and/or turn it on.
or
2. You need to install/reinstall the driver for the Nexys4 DDR board's USB programmer cable. Refer to instructions in the MIPSfpga Getting Started Guide if you need help reinstalling the driver.

Now click on **Program device** and select **xc7a100t_0**, as shown in Figure 38.

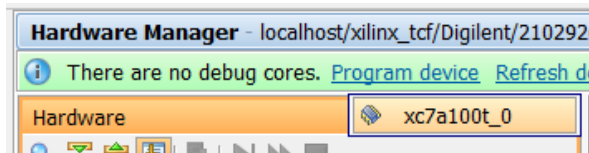


Figure 38. Selecting Program device

The Program Device window will open, as shown in Figure 39. The Bitstream file box should autopopulate, but if it does not, choose:

MIPSfpga_Labs\Labs\Xilinx\Part1_Intro\Lab01_Vivado\Project1\Project1.runs\impl_1\mfp_nexys4_ddr.bit

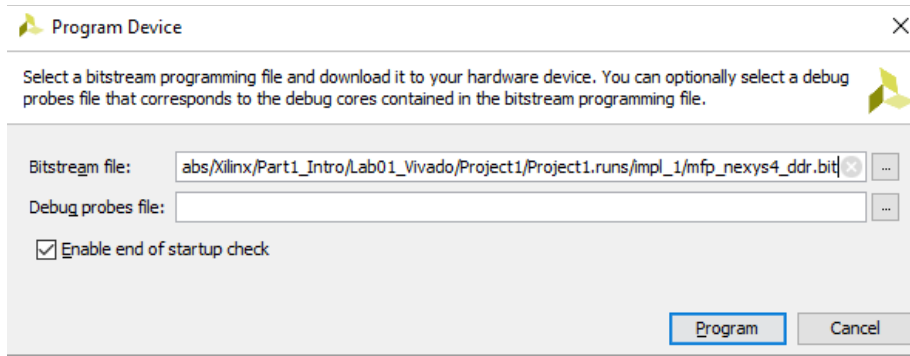


Figure 39. Program device with bitstream file

Leave the **Enable end of startup** box selected and click **Program**.

A window will pop up showing the programming progress, as shown in Figure 40. Programming the Artix-7 FPGA on the Nexys4 DDR board will take several seconds. Once it is complete, the progress window will close.

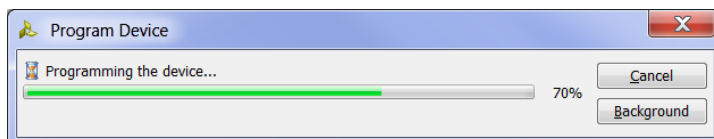


Figure 40. Program Device progress window

The MIPSfpga system is now downloaded onto the Nexys4 DDR board. Push the red reset pushbutton (labeled CPU RESET on the board, see [Figure 35](#)) to reset and start the MIPSfpga core. You will now see the LEDs display increasingly incremented values.

The MIPSfpga system is loaded with the IncrementLEDsDelay program. The machine code for this program is in the ram_rest_init.txt file located in the same directory as the Verilog files (i.e., in MIPSfpga_Labs\rtl_up). The IncrementLEDsDelay program is similar to the IncrementLEDs program that you simulated earlier in this lab, but it adds a delay so that our eyes can detect the results on the LEDs. It would have been tedious to simulate thousands of cycles of delay, so we took the delay out of the program code we used for simulation (see [Figure 22](#)). The C, MIPS assembly, and machine code for IncrementLEDsDelay is shown in [Figure 41](#) and [Figure 42](#).

<pre>// C code unsigned int val = 1; volatile unsigned int* ledr_ptr; ledr_ptr = 0xbf800000; while (1) { *ledr_ptr = val; val = val + 1; // delay }</pre>	<pre># MIPS assembly code # \$9 = val, \$8 = memory address 0xbf800000 addiu \$9, \$0, 1 # val = 1 lui \$8, 0xbf80 # \$8=0xbf800000 L1: sw \$9, 0(\$8) # mem[0xbf800000] = val addiu \$9, \$9, 1 # val = val+1 delay: # loop 2,500,000x lui \$5, 0x026 # \$5 = 2,500,000 ori \$5, \$5, 0x25a0 add \$6, \$0, \$0 # \$6 = 0 L2: sub \$7, \$5, \$6 # \$7 = 2,500,000 - \$6 addi \$6, \$6, 1 # increment \$6 bgtz \$7, L2 # finished? nop # branch delay slot beqz \$0, L1 # branch to L1 nop # branch delay slot</pre>
--	---

Figure 41. IncrementLEDsDelay program

```
24090001 // bfc00000:      addiu $9, $0, 1
3c08bf80 // bfc00004:      lui    $8, 0xbf80
ad090000 // bfc00008: L1:  sw     $9, 0($8)
25290001 // bfc0000c:      addiu $9, $9, 1
3c050026 // bfc00010: delay: lui   $5, 0x026
34a525a0 // bfc00014:      ori    $5, $5, 0x25a0
00003020 // bfc00018:      add    $6, $0, $0
00a63822 // bfc0001c: L2:  sub    $7, $5, $6
20c60001 // bfc00020:      addi   $6, $6, 1
1ce0ffff // bfc00024:      bgtz   $7, L2
00000000 // bfc00028:      nop
1000fff6 // bfc0002c:      beq    $0, $0, L1
00000000 // bfc00030:      nop
```

Figure 42. Machine code for IncrementLEDsDelay (files actually used are ram_b0.txt, ram_b1.txt, ram_b2.txt, and ram_b3.txt – byte-wide memories)

The next labs show how to write, compile, download, and run C and MIPS assembly programs on the MIPSfpga system.

3. Porting MIPSfpga to other FPGA Boards

This section describes how to port the MIPSfpga system onto other FPGA boards. You may choose to use a board other than the Nexys4 DDR board because of, for example, wanting a lower-cost option or having existing availability of other boards.

We describe how to port the MIPSfpga system to Digilent's **Basys3** board and Digilent's **Nexys4** board, which is the predecessor to the Nexys4 DDR board. You may follow similar steps to port the MIPSfpga system to other boards based on Xilinx FPGAs. Table 1 gives an overview of the features of the Nexys4 and Basys3 FPGA boards, as well as the Nexys4 DDR board that we've been using in the prior labs. If desired, the prior labs could have been completed on either the Nexys4 or Basys3 FPGA boards – or on other FPGA boards as well.

Table 1. FPGA Boards

Board	Overall specifications	Web Link	Cost
Nexys4 DDR	<ul style="list-style-type: none">• FPGA: Artix-7 (XC7A100T-CSG324)• Amount of block RAM: 607 KB• # of Logic Cells: 101k• 7-segment displays: 8• Switches: 16• Pushbuttons: 5• LEDs: 16• PMOD Connectors: 5	http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS4DDR	\$159 (academic), \$320 (non-academic)
Nexys4	Similar to Nexys4 DDR (for example, all of the above specifications are the same)	http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS4	\$179 (academic), \$320 (non-academic)
Basys3	<ul style="list-style-type: none">• FPGA: Artix-7 (XC7A35T-CPG236)• Amount of block RAM: 225 KB• # of Logic Cells: 33k• 7-segment displays: 4• Switches: 16• Pushbuttons: 5• LEDs: 16• PMOD Connectors: 4	http://www.digilentinc.com/Products/Detail.cfm?Prod=BASYS3	\$79 (academic), \$149 (non-academic)

Porting MIPSfpga to Digilent's Basys3 Board

Figure 43 shows Digilent's Basys3 board, the target of the MIPSfpga system.

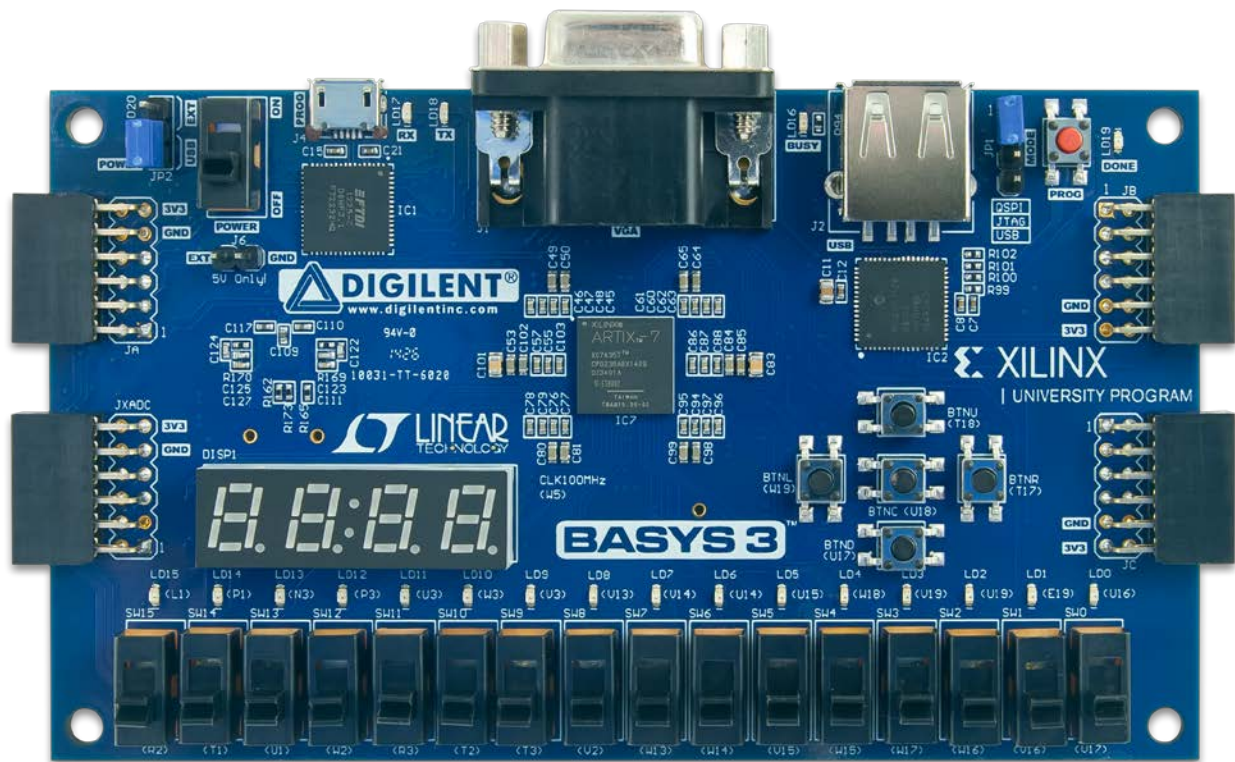


Figure 43. Digilent's Basys3 Board

As listed in Table 1, the Basys3, like the Nexys4 DDR board, is also built around Xilinx's Artix7 FPGA, however Basys3 uses a smaller version. The Basys3's Artix7 FPGA has about a third of the logic and memory as the one on the Nexys4 DDR board (225 KB of block RAM and 33K configurable logic blocks (CLBs) on the Basys3 vs. 607 KB of block RAM and 101K CLBs on the Nexys4 DDR). The MIPSfpga hardware fits easily on either board, however, the amount of block RAM needs to be reduced in order to fit on the Basys3 board.

MIPSfpga Modifications for the Basys3 Board

To port the MIPSfpga system to the Basys3 board, follow these steps:

- Step 1.** Write a *wrapper module* that maps the MIPSfpga I/O to the Basys3 board I/O
- Step 2.** Decrease the memory size of the MIPSfpga system to fit on the Basys3 board
- Step 3.** Add a constraints file that maps the board I/O to the correct FPGA pins

Each step is described in detail below. We then show how to build a Vivado project and compile the MIPSfpga system for the Basys3 board.

Step 1. Basys3 Wrapper Module

First, write a wrapper module that maps the MIPSfpga I/O to the I/O on the Basys3 board. Browse to the MIPSfpga_GSG\rtl_up\boards\basys3 folder and open the mfp_basys3.v file.

First, observe the mfp_basys3 module inputs and outputs. These signals are the interfaces to the Basys3 board.

```
module mipsfpga_basys3( input      clk,
                      input      btnU, btnD, btnL, btnR, btnC,
                      input  [15:0] sw,
                      output [15:0] led,
                      inout  [ 5:0] JB
                    );
```

clk is the onboard 100 MHz clock. btnU, btnD, btnL, btnR, and btnC are the names of the up, down, left, right, and center push buttons on the Basys3 board. We use btnC to reset the processor.

Input signal sw[15:0] are the 16 switches, led[15:0] are the 16 LEDs, and so on. The JB signals are the PMODB connector pins that connect to the EJTAG signals needed by the Bus Blaster. Notice that the JB connector is on the upper right of the Basys3 board (see [Figure 43](#)) and on the lower right of the Nexys4 DDR board.

Next, the phase-lock-loop, clk_wiz_0, is instantiated to create the 50 MHz MIPSfpga system clock from the onboard 100 MHz clock.

```
clk_wiz_0 clk_wiz_0(.clk_in1(clk), .clk_out1(clk_out));
```

The following lines manually connect the tck pin (JB[3]) to an input buffer (IBUF) and then to a clock buffer (BUFG) to address the problem that the JB[3] pin carries a clock signal but is not connected to a clock buffer.

```
IBUF IBUF1(.O(tck_in), .I(JB[3]));
BUFG BUFG1(.O(tck), .I(tck_in));
```

Finally, the heart of the module is to instantiate the MIPSfpga system (mfp_sys) and connect it to the Basys3 I/O.

```
mfp_sys mfp_sys(
    .SI_Reset_N(~btnC),
    .SI_ClkIn(clk_out),
    .HADDR(),
    .HRDATA(),
    .HWDATA(),
```

```

        .HWRITE(),
        .HSIZE(),
        .EJ_TRST_N_probe(JB[4]),
        .EJ_TDI(JB[1]),
        .EJ_TDO(JB[2]),
        .EJ_TMS(JB[0]),
        .EJ_TCK(tck),
        .SI_ColdReset_N(JB[5]),
        .EJ_DINT(1'b0),
        .IO_Switch({2'b0,sw}),
        .IO_PB({btnU, btnD, btnL, 1'b0, btnR}),
        .IO_LEDR(led),
        .IO_LEDG(),
        .UART_RX(RsRx)
    );

```

Step 2. Decrease Memory

The Basys3 board only has 225 KB of block RAM instead of the 607 KB of the Nexys4 DDR board. Thus, the two memory blocks (1 KB for boot RAM and 256 KB for program RAM) will not fit on the Basys3 board. We can limit our program needs to 128 KB. Thus, the total memory need (1 KB boot code + 128 KB program code = 129 KB) fits on the Basys3 board. The remaining 225-129 = 96 KB of block RAM can be used for other memory needs of the MIPSfpga system, such as caches.

We reduce the amount of memory by modifying the memory sizes declared in the Verilog header file. Again, browse to the MIPSfpga_GSG\rtl_up\boards\basys3 directory and open the mfp_ahb_const.vh file.

The updated size of the program RAM address is 14 bits. So the program RAM has 2^{15} 32-bit words = 2^{17} bytes = 128 KB.

```
`define H_RAM_ADDR_WIDTH          (15)
```

Step 3. Basys3 Constraints File

As the last step, we add a constraints file that maps the wrapper module's (mipsfpga_basys3) I/O signal names to the correct FPGA pins on the Basys3 FPGA board. Again, browse to the MIPSfpga_GSG\rtl_up\boards\basys3 directory and open the mfp_basys3.xdc Xilinx Design Constraints file. This file maps the FPGA pins to the inputs and outputs of the mfp_basys3 wrapper module. For example, the following line maps the input `clk` to FPGA package pin W5, which is fed by the 100 MHz clock (period = 10 ns) on the Basys3 board.

```

set_property PACKAGE_PIN W5 [get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk]

```

The following line addresses the issue that the tck pin of the EJTAG interface is not connected to an FPGA pin with a clock buffer.

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets tck_in]
```

The lines below map the switch inputs (sw[15 : 0]) to the FPGA pins that are physically wired to the switches on the Basys3 board. For example sw[0] is connected to the Artix7 package pin V17, sw[1] to pin W16, and so forth. They all use LVCMOS 3.3V signal levels.

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
...

```

The timing constraints for FPGA I/O are specified at the bottom of the file.

Setting up a Vivado Project for MIPSfpga on the Basys3 Board

Use similar steps to those described earlier in this lab to set up a Vivado project targeted to the Artix7 on the Basys3 board. The target device for the Basys3 board is the **xc7a35tcpg236-1** Artix7 FPGA. Be sure to add the following files to the project:

- mfp_basys3.v
- mfp_ahb_cont.vh
- mfp_basys3.xdc (constraints file)

After the project is set up, compile the design and generate a bitfile. Then download the bitfile onto the Basys3 board.

Loading the MIPSfpga System onto the Basys3 Board

After you have downloaded the bitfile onto the Basys3 board, press the center pushbutton (BTNC) on the Basys3 to reset the processor. You should now see the LEDs display incremented values. To use Codescape and the Bus Blaster probe to program the MIPSfpga core, connect the Bus Blaster probe to the PMOD B connector at the top right of the board (labeled JB). Follow the same instructions as provided in the MIPSfpga Getting Started Guide (and also in Labs 2-4) to download and debug programs on the MIPSfpga system running on the Basys3 board.

MIPSfpga on the Nexys4 Board

Some universities or laboratories may have legacy Nexys4 boards, the predecessors to the

Nexys4 DDR board. We provide the wrapper file and Xilinx Design Constraint file for the Nexys4 board for your convenience. They are located in this directory:

```
MIPSfpga_GSG\rtl_up\boards\nexys4
```

The target Artix-7 FPGA is the same as the one on the Nexys4 DDR board: **xc7a100tcsg324-1**.

Other Xilinx FPGAs and FPGA Boards

You may follow the same methods described here to target other FPGA boards and Xilinx FPGAs. Specifically, you will need to write a wrapper module, a board-specific Xilinx Design Constraints (.xdc) file, and possibly modify the amount of memory used by the MIPSfpga system. Some FPGAs may be too small for the MIPSfpga system to fit.