Creating, Packaging and Controlling an IP in Vivado/Vitis - Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq-7000 ARM/FPGA SoC Platform board

Objectives:

After completing this tutorial, you will be able to:

- Create and package an AXI4 IP
- Change the AXI4 interface to read/write from/to the IP via AXI4 bus
- Create an embedded system and add customize user IPs
- Create a constraints file and add IPs' external pins
- Use the IP drivers to control the IP functionality
- Control the IP from the PC via the Zynq UART and AXI4 IP interface

The aim of this tutorial is to familiarize students with the creation and packaging of their own IPs (Intellectual Property). It is a continuation of the two previous tutorials, the *Vivado Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board – a hardware approach*, and the *Vivado/Vitis Zynq Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board – using the Zynq microprocessor*, where a 4-bit up/down counter was developed and implemented in a Xilinx Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform from Digilent, using the Vivado and Vitis design and development tools, first using a hardware approach, and then using a software approach. In this tutorial, we are going to follow a hardware/software approach. We are going to create an AXI4-Lite complaint slave peripheral IP using the Create a new AXI4 Peripheral wizard. This implementation makes the up/down counter controllable by the microprocessor through the AXI4-Lite bus (Figure 1). The initial up/down counter uses the 4 LEDs of the board as outputs, one of the board's slide switches as the up/down selector, and another to enable/disable the counter, plus a push-button to reset the counter. The aim is to keep the same functionality but to add an AXI interface and control the counter from the computer keyboard, via a serial terminal emulator and the Zynq microprocessor UART.

We start by creating the up/down counter and add the AXI4-LITE interface, before generating the IP. After that, we create a design with the Zynq, and connect our IP to it, via the AXI bus. Finally, we write the software that enables to control the operation of the counter through a serial terminal emulator.

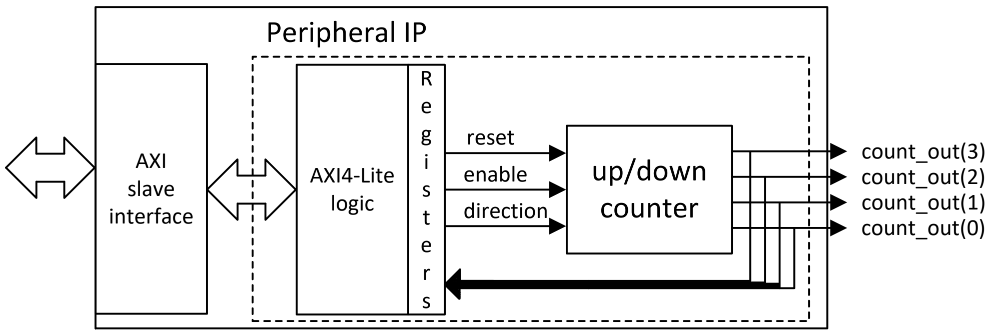


Figure 1

1. Starting the Vivado Software

To start Vivado, double-click the desktop icon,

or start Vivado from the Start menu by selecting:

Start → Xilinx Design Tools → Vivado 20xx.x

Note: Your start-up path is set during the installation process and may differ from the one above.

2. Create a New Project

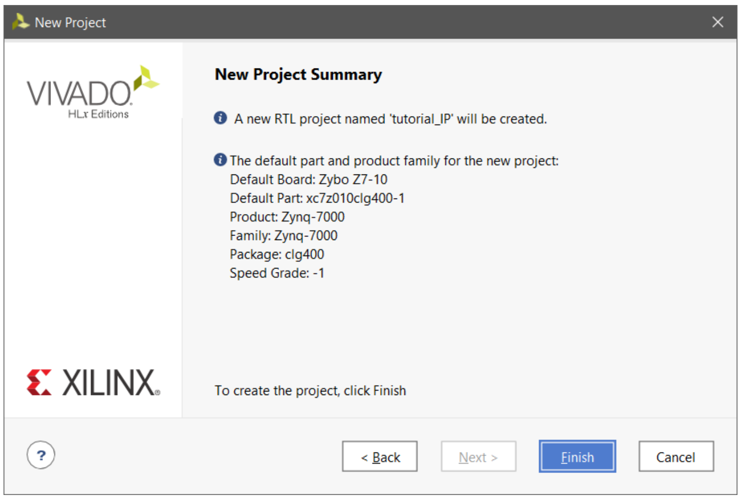
1. Click **Next**
2. Type **tutorial_IP** in the **Project name** field
3. Enter or browse to a location (directory path) for the new project. A tutorial subdirectory is created if the **Create project subdirectory** checkbox is checked

   WARNING: when using any of the Xilinx applications keep your directory path as close as possible to the root, and directory and file names as short as possible! Xilinx applications do not deal well with long directory paths and long directory or file names. Long paths or names usually lead to too hard to detect errors during development. AND IT DOES NOT TOLERATE WHITE SPACES!!!
4. Click **Next**
5. In the **Project Type** window select **RTL Project** and check the **Do not specify sources at this time** checkbox
6. In the next window select **Boards**, choose **Vendor: digilentinc.com** and select **Zybo Z7-10** from the list

Notice: if the Digilent vendor does not appear in the list, go to section 3 of the Vivado Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board – a hardware approach, and check how to proceed.

7. Click **Next** to view the **New Project Summary**
8. Check if everything is correct and then click **Finish** (Figure 2)

   The **Project Manager**, where we manage all our project, opens.



New Project Summary

A new RTL project named 'tutorial_IP' will be created.

The default part and product family for the new project:
Default Board: Zybo Z7-10
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

To create the project, click Finish

Figure 2

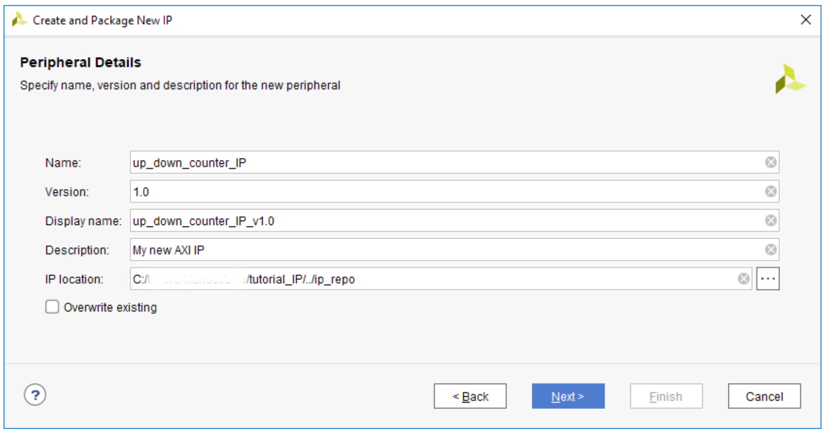
3. Create an AXI4 Peripheral IP

In this section, we learn how to create an IP for our up/down counter with an AXI4-Lite interface.

1. In the **Flow Navigator** window, under the **Project Manager** section, click **Settings**
2. In the new **Settings** window, click **General** in the vertical toolbar
3. Choose **VHDL** as **Target language**, and click **OK**
4. In the Vivado main toolbar select **Tools** ⟶ **Create and Package New IP…**
5. A new **Create and Package New IP** window opens
6. Click **Next** to continue
7. In the **Create Peripheral, Package IP or Package a Block Design** page select **Create a new AXI4 Peripheral** under **Create AXI4 Peripheral** and click **Next**

   Create a new AXI4 peripheral includes HDL, drivers, a test application, and Verification IP (VIP) example template
8. In the **Peripheral Details** page, type **up_down_counter_IP** in the peripheral **Name** field
9. Keep all the remaining default settings and click **Next** to continue (Figure 3)

   The new **Add Interfaces** window enables users to configure some of the characteristics of the IP, namely the AXI interface type - AXI4-Full, AXI4-Lite, and AXI4-Stream, interface mode - Slave or Master, the read and write data width and the memory size, if selecting a AXI4-Full slave interface, and the number of the AXI registers, only supported for AXI4-Lite slave interfaces. For more about the Advanced eXtensible Interface (AXI) check the AXI Reference Guide, Vivado Design Suite, UG1037. It is also possible to enable interrupt support and to add other user-defined pins. At the bottom left corner of the window a ⓘ signal enables the access to a Quick Help with a brief explanation of the configuration options. (Figure 3)

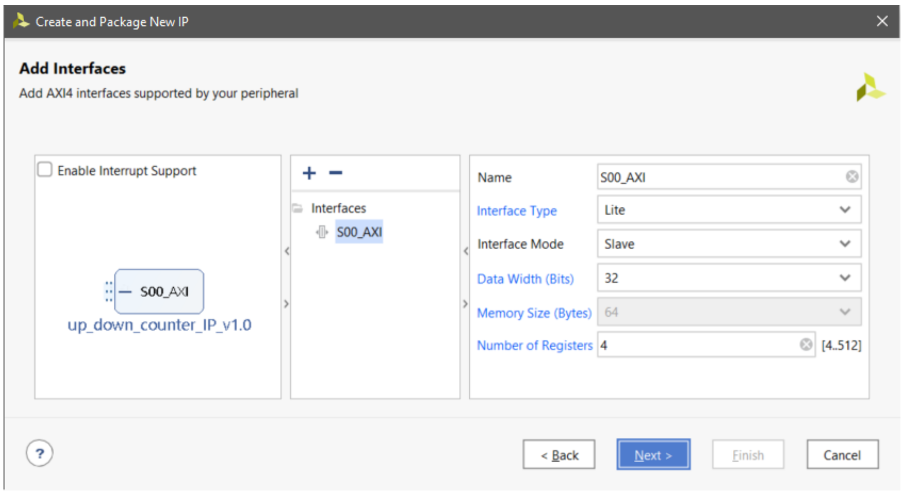

Create and Package New IP
Peripheral Details
Specify name, version and description for the new peripheral
< Back
Next >
Finish
Cancel

Name: up_down_counter_IP
Version: 1.0
Display name: up_down_counter_IP_v1.0
Description: My new AXI IP
IP location: C:/ [illegible] /tutorial_IP/../ip_repo
Overwrite existing

Figure 3

10. In the **Add Interfaces** pages, accept the default settings and click **Next** (Figure 4)

    With the default settings the IP is generated with four 32-bit read/write registers – slv_reg0, slv_reg1, slv_reg2, slv_reg3, we may use to send/receive data from the Zynq microprocessor via the AXI4-Lite interface.

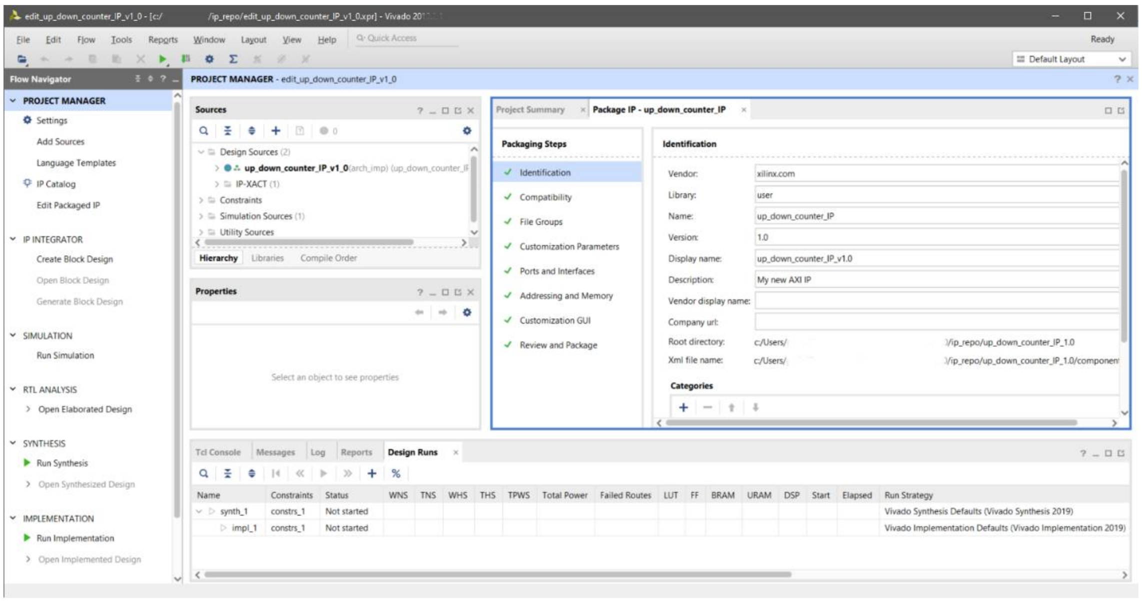


Name: S00_AXI
Interface Type: Lite
Interface Mode: Slave
Data Width (Bits): 32
Memory Size (Bytes): 64
Number of Registers: 4 [4..512]

Figure 4

11. In the **Create Peripheral** page, select **Edit IP** and click **Finish**

    A new project-like window with the Package IP tab opens (Figure 5)



Vendor: xilinx.com
Library: user
Name: up_down_counter_IP
Version: 1.0
Display name: up_down_counter_IP_v1.0
Description: My new AXI IP
Vendor display name:
Company url:
Root directory: c:/Users/ /ip_repo/up_down_counter_IP_1.0
Xml file name: c:/Users/ /ip_repo/up_down_counter_IP_1.0/componen

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)

Figure 5

4. Editing and packaging the IP

The **Create and Package New IP** wizard creates two files: the **up_down_counter_IP_v1_0_S00_AXI.vhd** file, which contains the AXI4-Lite logic description needed to implement the AXI4-Lite protocol, and where we add our up/down counter; and a top file, the **up_down_counter_IP_v1_0.vhd**, that basically contains the AXI4-Lite slave interface, and instantiates the previous file. In this section, we customize both files to accommodate our up/down counter logic and to connect it to the AXI4-Lite registers by which the IP sends and receives data to/from the Zynq microprocessor.

Let us start by editing our top file to add the VHDL code needed to create the external outputs where the board LEDs are going to be connected.

1. Double click **up_down_counter_IP_v1_0 (arch_imp) (up_down_counter_IP_v1_0.vhd)** in the **Sources** window
2. The **up_down_counter_IP_v1_0** VHDL design source opens in the main window
3. Look for the comment `-- Users to add ports here` and add the following VHDL code below this line

   ```
   count_out : out STD_LOGIC_VECTOR (3 downto 0);
   ```

   Next, we need to add this count_out output to the instantiation of the **up_down_counter_IP_v1_0_S00_AXI** to pass these signals from our logic to the IP exterior.
4. Look for the component declaration of the AXI bus interface

   ```
   -- component declaration
      component up_down_counter_IP_v1_0_S00_AXI is
   ```

5. Add to the **port** list the **count_out** output bus

   ```
   port (
   count_out : out STD_LOGIC_VECTOR (3 downto 0);
   ```

6. And to the **port map**

   ```
   S_AXI_RREADY  => s00_axi_rready,
   count_out          => count_out
   );
   ```

   Notice: the above code may be inserted anywhere inside the port list and inside the port map description

7. Save and close the **up_down_counter_IP_v1_0.vhd** file
8. Next, we edit the **up_down_counter_IP_v1_0_S00_AXI.vhd** to add to its interface the output port and to add the up/down counter component. The inputs of the up/down counter are connected to the AXI4 Register 0, enabling its control via the AXI4 bus, and its outputs to AXI4 Register 1, allowing to read the counter output value also via the AXI4 bus.
9. Double click **up_down_counter_IP_v1_0_S00_AXI_inst** (…) in the **Sources** window
10. The **up_down_counter_IP_v1_0_S00_AXI** VHDL design source opens in the main window
11. Look for the comment `-- Users to add ports here` and add the following VHDL code below this line

    ```
    count_out : out STD_LOGIC_VECTOR (3 downto 0);
    ```

12. Look for the architecture declaration

    ```
    architecture arch_imp of up_down_counter_IP_v1_0_S00_AXI is
    ```

13. Add the up/down counter component declaration right after the architecture declaration

    ```
    -- Declare up/down counter "top_counter" component here
       Component top_counter
            Port (
                    clk : in STD_LOGIC;
                    reset : in STD_LOGIC;
                    en : in STD_LOGIC;
                    direction : in STD_LOGIC;
                    count_out : out STD_LOGIC_VECTOR (3 downto 0));
       end component;
    ```

    Our up/down counter will be, at power-up, initialized by a reset signal part of the AXI4 interface, but it may also be, at any moment, reset by the user. Therefore, we need to combine both signals creating a single reset signal to apply to the up/down counter.

14. Add a new signal declaration right after the component declaration for the reset signal of the up/down counter

    ```
    -- Reset signal needed to combine the external reset with the
    -- initial AXI reset that initiates the up/down counter at
    -- power-up
       signal rst : std_logic;
    ```

Now we need to map the inputs of our up/down counter to the AXI4-Lite register through which the microprocessor will control the counter operation, and the output to the corresponding output of the module, and add the logic needed to combine both reset signals in one: S_AXI_ARESETN, coming from the AXI4 logic, and the external reset signal. Notice that S_AXI_ARESETN is active low, while our counter reset is active high. Therefore, we need to invert this signal before applying it to our counter.

15. Go to the bottom of the file, and look for the comment `-- Add user logic here`, and add the following VHDL code below this line

```
-- "top_counter" port map
-- Maps the input reset, en, and direction to bits of the
-- slv_reg0 and the outputs to the outputs of the module

Inst_top_counter: top_counter
PORT MAP(
        clk => S_AXI_ACLK,
        reset => rst,
        en => slv_reg0(1),
        direction => slv_reg0(2),
        count_out => count_out_signal
        );
-- Generate a single reset signal to apply to the up/down counter
-- S_AXI_ARESETN active low
   rst <= not(S_AXI_ARESETN) OR slv_reg0(0);
```

Figure 6 shows the mapping of the input ports of our up/down counter. Basically, what we do is to connect the inputs of our up/down counter to register slv_reg0 of the AXI4-Lite interface and by writing to this register control the operation of the up/down counter. The clock signal for our IP is the clock signal provided by the AXI4-Lite interface, while the reset signal is initially applied by the same interface, at power-up, and can also later be controlled through the slv_reg0 enabling its reset at any moment during operation.

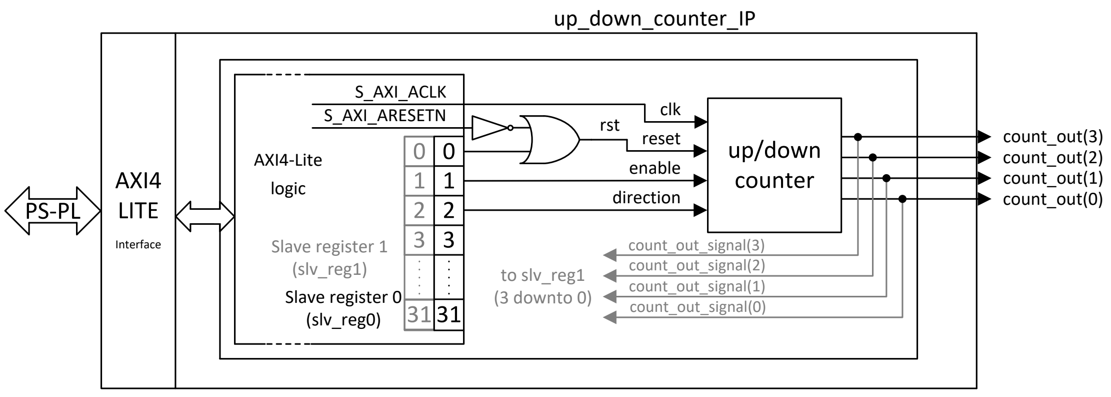


Figure 6

Apart from showing current counter value using the board LEDs, the Zynq microprocessor may read this value through the slv_reg1one of the AXI4-Lite interface.

Since VHDL does not allow to read from a 'out' object, we need to create and add an intermediate signal.

16. Add a new signal declaration right after the component declaration and the reset signal of the up/down counter to enable reading the **count_out** signal before it reachs the output port

```
-- Signal needed to read the current counter value
signal count_out_signal : STD_LOGIC_VECTOR (3 downto 0);
```

This is the reason why, in the port map of the `top_counter` instantiation, `count_out` connects to this new `count_out_signal` instead of connecting directly to `count_out`.

17. Redirect the new signal to the 4 least significant bits of register `slv_reg1` and to the output of the IP, to connect to the external pins, by adding the following VHDL code to the same `-- Add user logic here` section

```
slv_reg1(3 downto 0) <= count_out_signal;
count_out <= count_out_signal;
```

18. Finally, to avoid conflicts due to multiple signals driving the 4 least significant bits of `slv_reg1`, we need to prevent the system from resetting or writing to these bits
19. Look for the process were the registers are reset

```
process (S_AXI_ACLK)
variable loc_addr : std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      slv_reg0 <= (others => '0');
      slv_reg1 <= (others => '0');
      slv_reg2 <= (others => '0');
      slv_reg3 <= (others => '0');
```

and change the reset of `slv_reg1` to

```
-- the 4 LSB are connected to and reset by the counter
slv_reg1(31 downto 4) <= (others => '0');
```

20. To avoid the processor to write there, look and change the following code in the same process

```
when b"01" =>
    for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
        if ( S_AXI_WSTRB(byte_index) = '1' ) then
        -- Respective byte enables are asserted as per write strobes
        -- slave registor 1
        slv_reg1(byte_index*8+7 downto byte_index*8) <=
                           S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
        end if;
end loop;
```

and change the write of `slv_reg1` to

```
            slv_reg1(byte_index*8+7 downto byte_index*8+32) <=
                                 S_AXI_WDATA(byte_index*8+7 downto byte_index*8+32);
```

In this way, we prevent the system from resetting or writing to the 4 least significant bits of `slv_reg1`, now connected to the counter output

21. In the end, save and close the **up_down_counter_IP_v1_0_S00_AXI.vhd** file

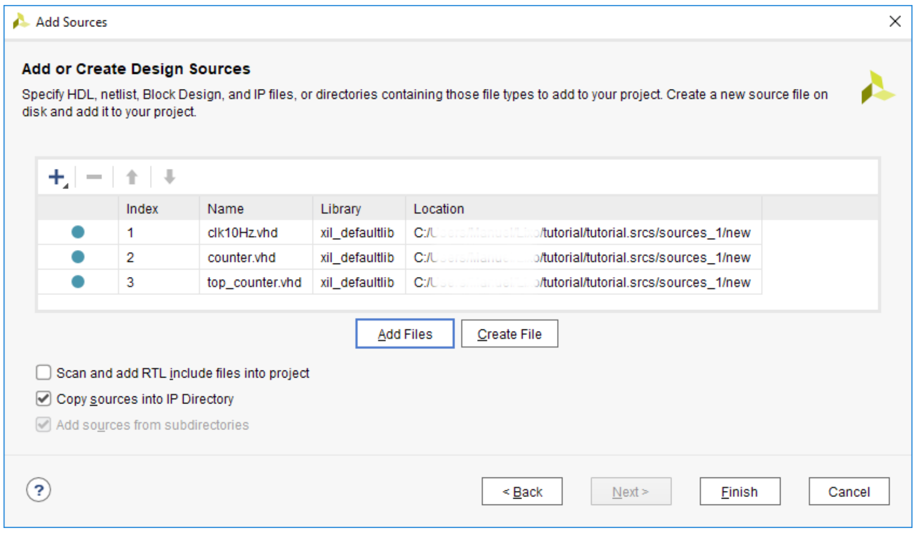
    The only thing missing is the up/down counter VHDL description we need to add to complete our IP logic.

22. In the **Flow Navigator** window, click **Add Sources**, and locate and add to the project the three VHDL files that describe the up/down counter developed in the *Vivado Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board – a hardware approach*:

    - counter.vhd
    - clk10Hz.vhd
    - top_counter.vhd

23. Make sure to check the **Copy sources into IP directory** checkbox is checked, to create copies of the files inside the IP repository (otherwise, any changes in the files will also affect the previous project) (Figure 7)

    WARNING: the IP packager does not support VHDL-2008. In the up/down counter VHDL descriptions no VHDL-2008 types are used



Add Sources

Add or Create Design Sources

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

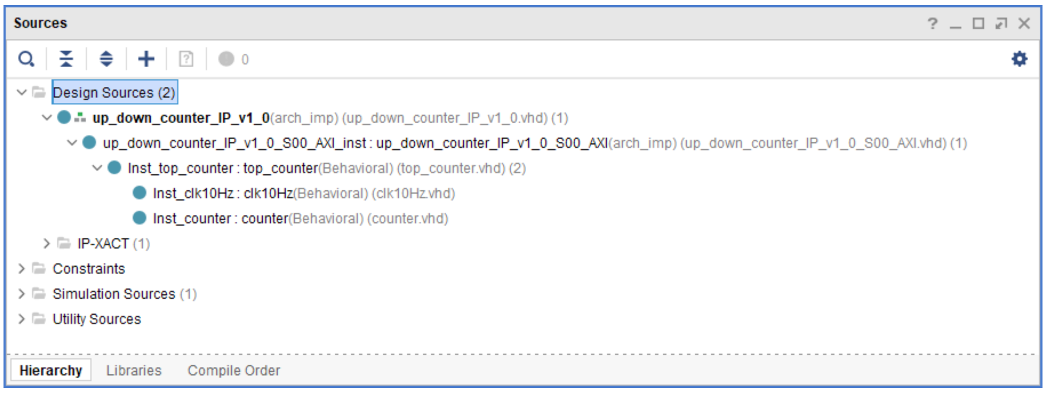
	Index	Name	Library	Location
●	1	clk10Hz.vhd	xil_defaultlib	C:/U[redacted]/tutorial/tutorial.srcs/sources_1/new
●	2	counter.vhd	xil_defaultlib	C:/U[redacted]/tutorial/tutorial.srcs/sources_1/new
●	3	top_counter.vhd	xil_defaultlib	C:/U[redacted]/tutorial/tutorial.srcs/sources_1/new

☐ Scan and add RTL include files into project
☑ Copy sources into IP Directory
☑ Add sources from subdirectories

Figure 7

24. Click **Finish**

25. Check the **Sources** window to see if all the necessary files are there and if the file hierarchy is correct (Figure 8)



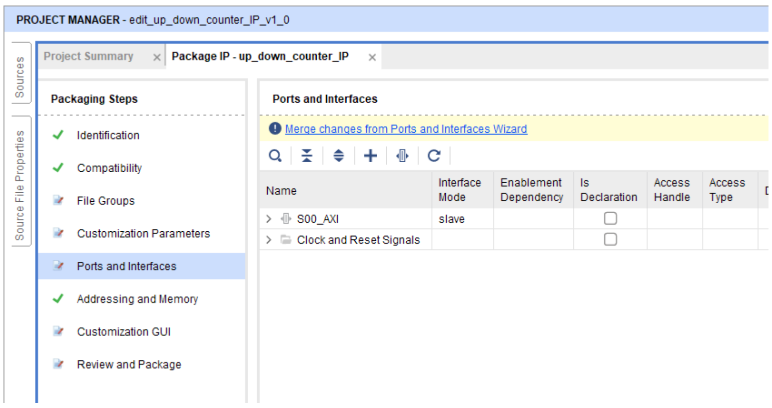
Sources

- Design Sources (2)
  - up_down_counter_IP_v1_0(arch_imp) (up_down_counter_IP_v1_0.vhd) (1)
    - up_down_counter_IP_v1_0_S00_AXI_inst : up_down_counter_IP_v1_0_S00_AXI(arch_imp) (up_down_counter_IP_v1_0_S00_AXI.vhd) (1)
      - Inst_top_counter : top_counter(Behavioral) (top_counter.vhd) (2)
        - Inst_clk10Hz : clk10Hz(Behavioral) (clk10Hz.vhd)
        - Inst_counter : counter(Behavioral) (counter.vhd)
  - IP-XACT (1)
- Constraints
- Simulation Sources (1)
- Utility Sources

Hierarchy Libraries Compile Order

Figure 8

26. Go to the **Package IP – up_down_counter_IP** tab in the main **Project Manager** window
27. Select **Ports and Interfaces** in the left-side list and click **Merge changes from Ports and interfaces Wizard** to add to the IP the external output port of our up/down counter (Figure 9)



PROJECT MANAGER - edit_up_down_counter_IP_v1_0

Project Summary | Package IP - up_down_counter_IP

Packaging Steps
- Identification
- Compatibility
- File Groups
- Customization Parameters
- Ports and Interfaces
- Addressing and Memory
- Customization GUI
- Review and Package

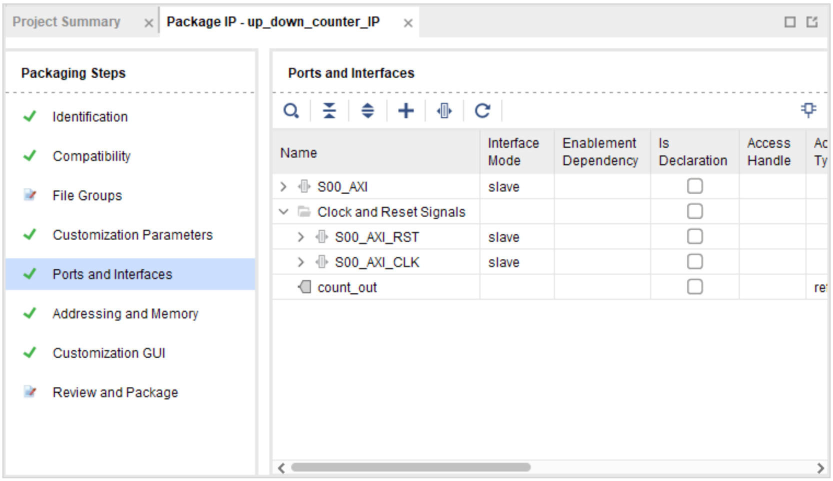
Ports and Interfaces

Merge changes from Ports and Interfaces Wizard

Name	Interface Mode	Enablement Dependency	Is Declaration	Access Handle	Access Type
S00_AXI	slave		☐		
Clock and Reset Signals			☐		

Figure 9

28. Make sure that the window is updated and includes the count_out port (Figure 10)



Project Summary | Package IP - up_down_counter_IP

Packaging Steps
- Identification
- Compatibility
- File Groups
- Customization Parameters
- Ports and Interfaces
- Addressing and Memory
- Customization GUI
- Review and Package

Ports and Interfaces

Name	Interface Mode	Enablement Dependency	Is Declaration	Access Handle	Ac Ty
S00_AXI	slave		☐		
Clock and Reset Signals			☐		
S00_AXI_RST	slave		☐		
S00_AXI_CLK	slave		☐		
count_out			☐		re

Figure 10

29. Select **Review and Package** in the left-side list

30. At the bottom of the **Review and Package** window, click **Re-Package IP**
31. A new window opens stating that the packaging was successfully completed and asking if you would like to close the project
32. Select **Yes** to close it

    The **Create a new AXI4 Peripheral** wizard also creates the necessary software drivers and a test application for the IP.

5. Create a New Block Design to integrate our IP

Now that our AXI4 IP is done, we are going to create a Zynq-based system just like we did in the previous *Vivado/Vitis Zynq Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board – using the Zynq microprocessor* and add our IP to the design.

Create a new block design source for the project using the **IP Integrator** as follows:

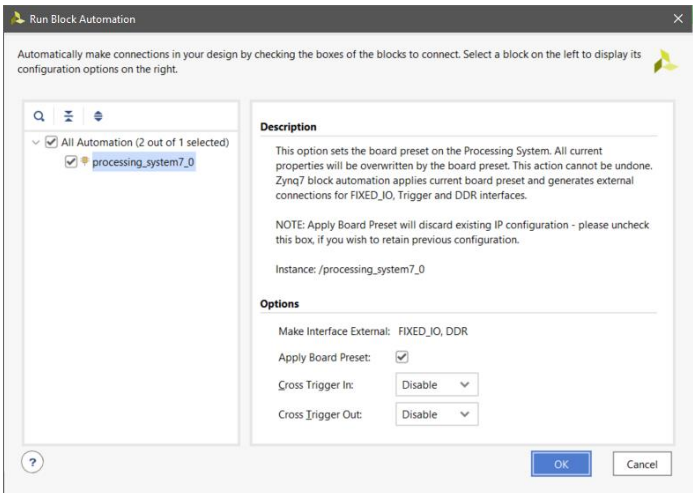
1. In the **Flow Navigator** window, on the left side of the **Project Manager** main window, in the **IP Integrator** section**,** click **Create Block Design**
2. In the new **Create Block Design** window fill in the spaces as follow:
    - Design name: **tutorial_ip**
    - Directory: **<Local to Project>**
    - Specify source set: **Design Sources**
3. Click **OK**

    An empty design workspace is created where we can add IP blocks.

6. Adding and Customizing the ZYNQ7 Processing System

1. In the **Flow Navigator** window, on the left side of the main window, in the **Project Manager** section, click **IP Catalog** to view the list of available IP cores
2. The **IP Catalog** opens on the **Block Design** main window
3. Under **Vivado Repository → Embedded Processing → Processor**, double click **ZYNQ7 Processing System** to add it to our design
4. In the new **Add IP** floating window select **Add IP to Block Design**
5. Click on **Run Block Automation** at the top of the **Diagram** window and a customization assistant window opens with a set of default settings
6. In the new **Run Block Automation** make sure that **processing_system7_0** under **All Automation** in the panel to the left and **Apply Board Preset** in the panel to the right are checked. These options set the board preset on the Processing System. ZYNQ7 block automation applies current board presets and generates external connections for

FIXED_IO, Trigger and DDR interfaces (Figure 11). The preset values are part of the Zybo Z7-10 board file


Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

All Automation (2 out of 1 selected) ✓

processing_system7_0 ✓

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

Options

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ✓

Cross Trigger In: Disable

Cross Trigger Out: Disable

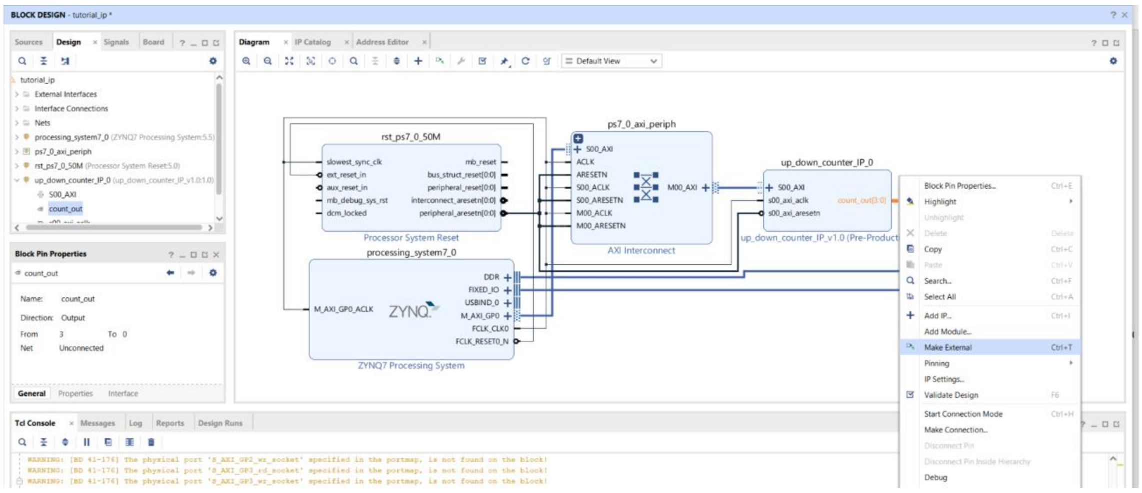
Figure 11

7. Click **OK**

7. Adding the up/down counter IP

The next step is to add our up/down counter IP to the design

1. In the **Block Design** main window choose the **IP Catalog** tab to view again the list of available IP cores
2. Under **User Repository → AXI Peripheral**, double click **up_down_counter_IP_v1.0** to add it to our design
3. In the new **Add IP** floating window select **Add IP to Block Design**
4. Click on **Run Connection Automation** at the top of the **Diagram** window and a customization assistant window opens with a set of default settings
5. Make sure that **All Automation** is checked and just accept the default values by clicking **OK** Since we added a block with an AXI interface, our IP, Vivado automatically adds an **AXI Interconnect** block and a **Processor System Reset** block and connects all the blocks in our system. However, the IP output port is unconnected. We need to make it external to be able to connect the outputs to the board's LEDs.
6. Start by regenerating the layout by selecting the **Regenerate Layout** icon in the horizontal toolbar of the **Diagram** window to rearrange the design
7. Right-click the **count_out[3:0]** output of our IP module and select **Make External** (Figure 12)



```
WARNING: [BD 41-176] The physical port 'S_AXI_GP2_wr_socket' specified in the portmap, is not found on the block!
WARNING: [BD 41-176] The physical port 'S_AXI_GP3_rd_socket' specified in the portmap, is not found on the block!
WARNING: [BD 41-176] The physical port 'S_AXI_GP3_wr_socket' specified in the portmap, is not found on the block!
```

Figure 12

8. An output port is added to our IP
9. Then validate the design by selecting in the menu bar **Tools → Validate Design** or by selecting the **Validate Design** icon ☑ in the horizontal toolbar of the **Diagram** window
10. A new **Validate Design** window appears stating that the validation was successful
11. Click **OK**

If, instead, a **Critical Messages** window opens stating that there may be problems with the PS DDR interface, just ignore it by clicking **OK**. You may rerun **Validate Design** again or just ignore and continue.

8. Making the HDL Wrapper

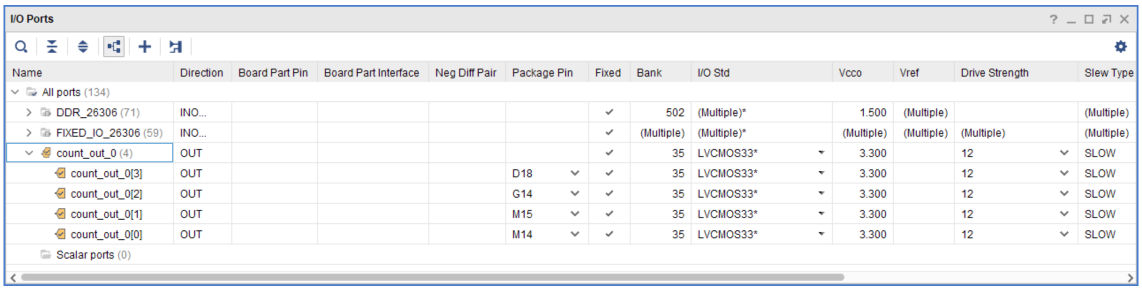
After the design validation step, we proceed with the creation of an **HDL System Wrapper**. The wrapper defines the block design as the top-level design, so we may simulate, synthesize, implement, and generate a bitstream for the block design.

1. In the **Flow Navigator** window, in the left side of the **Block Design** main window, in the **Project Manager** section, click **Settings**
2. In the new **Settings** window click **General** in the vertical toolbar
3. Choose **VHDL** as **Target language** and click **OK**
4. Click on the **Sources** tab located in the left-hand side of the **Block Design** window
5. Right-click on our block design **Design Sources → tutorial_ip (tutorial_ip.bd)** and select **Create HDL Wrapper…**
6. In the new **Create HDL Wrapper** window, check that **Let Vivado manage wrapper and auto-update** is selected and click **OK**

9. Assigning I/O Pin Locations

The next step is to assign the pins where the count_out[3:0] outputs are going to connect to enable the light up of our board's LEDs.

1. In the **Flow Navigator** window, in the **RTL Analysis** section, click **Open Elaborated Design**
2. A new window opens alerting current settings that allow performing I/O planning slow down netlist elaboration. But since we need to make pin assignments to connect the **count_out** port to the board's LEDs, we may not change those settings. So, click **OK** to continue
3. In the end, the schematic representation of our design opens in the **Elaborated Design** main window
4. Open the **I/O Planning** interface by selecting in the main toolbar the **Layout** menu and choosing **I/O Planning**
5. In the **I/O Ports** window, located at the bottom of the main window, we have a list of all I/O signals that are part of our up/down counter design
6. Expand the **count_out_0(4)** output bus to have access to all the individual bus signals
7. The aim is to connect the outputs of our up/down counter to the 4 LEDs of our Zybo Z7-10 board. To do it, attribute to each **count_out_0** signal the respective output pin using the following information from the Board Reference Manual (Figure 13):
    - **count_out_0[3]** (LD3 LED on the board) **Package Pin → D18**, **I/O Std → LVCMOS33**, **Vcco → 3.300**
    - **count_out_0[2]** (LD2 LED on the board) **Package Pin → G14**, **I/O Std → LVCMOS33**, **Vcco → 3.300**
    - **count_out_0[1]** (LD1 LED on the board) **Package Pin → M15**, **I/O Std → LVCMOS33**, **Vcco → 3.300**
    - **count_out_0[0]** (LD0 LED on the board) **Package Pin → M14**, **I/O Std → LVCMOS33**, **Vcco → 3.300**



Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type
All ports (134)												
DDR_26306 (71)	INO...					✓	502	(Multiple)*	1.500	(Multiple)		(Multiple)
FIXED_IO_26306 (59)	INO...					✓	(Multiple)	(Multiple)*	(Multiple)	(Multiple)	(Multiple)	(Multiple)
count_out_0 (4)	OUT					✓	35	LVCMOS33*	3.300		12	SLOW
count_out_0[3]	OUT				D18	✓	35	LVCMOS33*	3.300		12	SLOW
count_out_0[2]	OUT				G14	✓	35	LVCMOS33*	3.300		12	SLOW
count_out_0[1]	OUT				M15	✓	35	LVCMOS33*	3.300		12	SLOW
count_out_0[0]	OUT				M14	✓	35	LVCMOS33*	3.300		12	SLOW
Scalar ports (0)												

Figure 13

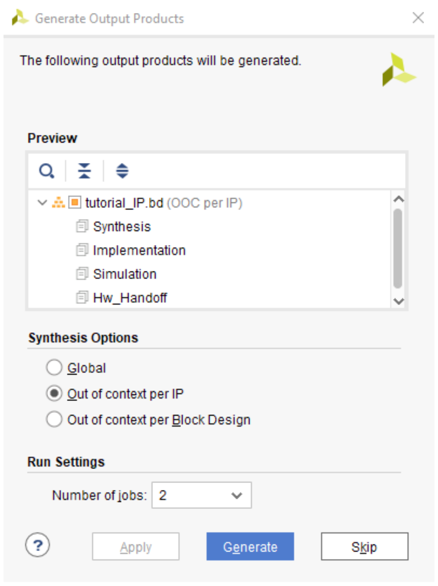
8. Save the newly created **I/O Planning** constraints into the constraints file by selecting in the main toolbar **File → Constraints → Save**
9. A new **Save Constraints** file window opens
10. Since there is no constraints file yet in our design, we need to create one
11. Check the checkbox **Create a new file**
    - File type: **XDC**
    - File name: **tutorial_IP**
    - File location: **<Local to Project>**
12. Click **OK**
13. A new constraint file **tutorial_IP.xdc** is added to the project

Re-editing and repackaging the IP

If during the design process, we found an error, or need to change the functionality of our IP, we may at any time re-edit it, introduce the necessary changes, and repack it.

1. Go to **Block Design → Diagram**, right-click the up_down_counter_IP_0 block and select **Edit in IP Packager**
2. In the new **Edit in IP Packager**, accept the default project name and location for editing by clicking **OK**
3. In case a new **Confirm Overwrite** window appears, confirm if it is **OK** for you to overwrite any previous IP files, or **Cancel** if not, returning to the previous window where you may change the project location
4. A new project-like window with the **Package IP** tab opens
5. Refer to section **Editing and packaging the IP** and make the necessary changes
6. If changing any ports, do not forget to go to **Ports and Interfaces** in the left-side list and click **Merge changes from Ports and interfaces Wizard**
7. In the end, go to **Review and Package** window, click **Re-Package IP**
8. Close the project by clicking **Yes** in the next window
9. Back to the **Block Design** window, click on **Report IP Status** in the top
10. A new **IP Status** tab appears at the bottom of the window
11. Notice that **tutorial_IP** and **up_down_counter_IP_0** checkboxes are checked and that a message – **IP revision change. IP definition 'up_down_counter_IP_v1.0 (1.0) changed on disk** – appears in the **IP Status**, with a **Recommendation** to **Upgrade IP**

12. Click on **Upgrade Selected**
13. Wait for the **IP Upgrade Complete** message and click **OK**
14. A new **Generate Output Products** window appears



In this window, we may synthesize the newly repacked IP using the **Out of context per IP** option. Hierarchical Design flows enable us to partition the design into smaller, more manageable modules to be processed independently. In the Vivado Design Suite, these flows are based on the ability to implement a partitioned module out-of-context (OOC) from the rest of the design. The most common use situation in the context of IP integrator is that you can set an IP or a block design as out-of-context modules which can be synthesized, while creating a design checkpoint (DCP) file. The IP or the block design, if used as a part of a larger Vivado design, does not have to be re-synthesized every time other parts of the design (outside of IP integrator) are modified. This provides considerable run-time improvements. Refer to Designing IP Subsystems Using IP Integrator, UG994, for a deeper explanation of how to set IPs and Block Designs as Out-of-Context Modules.

In the **Generate Output Products** window, the OOC flow is the default flow because it improves synthesis run times since we only synthesize the IP when changes to the IP customization or version require it, rather than re-synthesizing it as part of the top-level design. Refer to Designing with IP, UG896, for a deeper explanation of this subject.

15. Accept the default values and click **Generate** to generate an OOC IP or **Skip** if you do not want to generate an OOC IP

16. In the **IP Status** tab click **Rerun**
17. The **IP Status** tab shows now that the IP is **Up-to-date**
18. Validate the design by selecting in the menu bar **Tools → Validate Design** or by selecting the **Validate Design** icon ☑ in the horizontal toolbar of the **Diagram** window
19. Click on the **Sources** tab located in the left-hand side of the **Block Design** window
20. Right-click on our block design **Design Sources → tutorial_ip (tutorial_ip.bd)** and select **Create HDL Wrapper…**
21. In the new **Create HDL Wrapper** window, check that **Let Vivado manage wrapper and auto-update** is selected and click **OK**
22. Follow section 9 in case you change/add any External Port to the IP
23. Then, proceed to section 10

10.Generating the Bitstream

Now that we completed the hardware part of our embedded system design, we are going to generate the bitstream necessary for the configuration of the Zynq-7010 FPGA of our ZYBO Z7-10 board.

1. In the **Flow Navigator** window, under the **Program and Debug** section, click **Generate Bitstream**
2. In the new **No Implementation Results Available** window, click **Yes** to synthesize and implement the design, and to generate the bitstream
3. If a **Launch Runs** window appear, just click **OK**
4. In the end of the bitstream generation process, in the new floating window click **OK** to open the implemented design

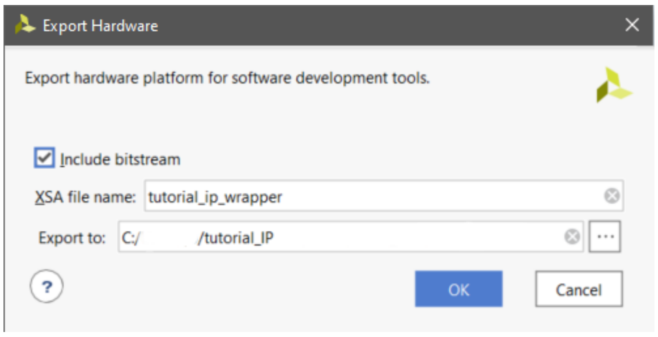
In the main **Implemented Design** window, we have access to a bunch of information related to the implementation of our design. In the main window, check the **Device** tab to have an idea of the space occupied by our design inside the FPGA, and of the connections between PS and PL.

At the bottom, several tabs give access to the I/O ports map, the estimated power consumed by our design, a summary of the timing analysis, and to a long list of reports generated during the synthesis, implementation, place and route, and bitstream generation. Take time to look at this information. It gives an idea of the complexity of the process, and of all variables that must be considered to implement our design inside a ZYNQ7 Processing System.

11.Exporting Hardware Design to Vitis

Now that the hardware is done, it is time to export it to the Vitis Unified Software Platform to start creating the software that will run in our ZYNQ7 Processing System. Refer to section 10 of the *Vivado/Vitis Zynq Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board – using the Zynq microprocessor* for an introduction to the Vitis Unified Software Platform and Vitis Integrated Development Environment (IDE).

1. In the main toolbar, select **File → Export → Export Hardware…**
2. Check the box to **Include Bitstream** and click **OK** (Figure 14)


Export Hardware

Export hardware platform for software development tools.

☑ Include bitstream

XSA file name: tutorial_ip_wrapper

Export to: C:/ /tutorial_IP

Figure 14

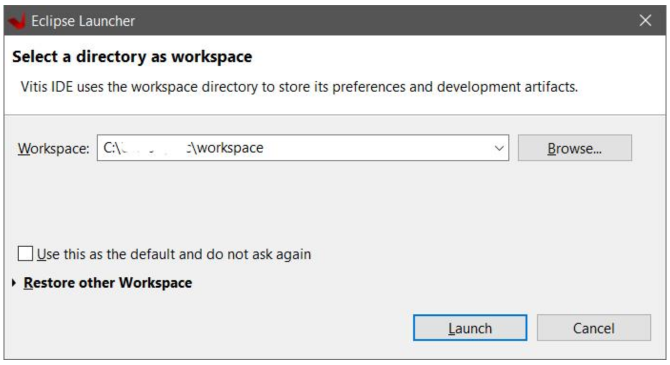
An XSA file is created in the root folder of the project. XSA is a container that contains:

- One or more .hwh (Hardware Handoff File) files:
  - Vivado tool version, part, and board tag information;
  - IP - instance, name, VLNV (vendor, library, name, and version), and parameters;
  - Memory Map information of the processors;
  - Internal Connectivity information (including interrupts, clocks, etc.) and external ports information.
- BMM (Block Memory Map)/MMI (Memory Map Information) and BIT (bitstream) files;
- User and HLS (High-Level Synthesis) driver files;
- Other meta-data files.

The handoff files make the hardware appear to software as an Application-Specific Standard Product (ASSP), commercially available blocks that perform specific functions required by different applications.

12.Create a Platform Project in Vitis

1. In the main toolbar, select **Tools → Launch Vitis**
2. Choose the folder where you want to create the Vitis workspace and click **Launch** (Figure 15)



Eclipse Launcher

Select a directory as workspace

Vitis IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\ :\workspace

Use this as the default and do not ask again

Restore other Workspace

Figure 15

13.The Vitis environment

The software development cycle begins with the handoff files created during the hardware design flow with the Vivado tools. Vitis uses these files in driver and application development. They contain the initialization code for the Zynq SoC Processing System and initialization settings for DDR, clocks, phase-locked loops (PLLs), and MIOs. For more information, refer to section 13 of the *Vivado/Vitis Zynq Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board – using the Zynq microprocessor*.

The **Vitis IDE** window (Figure 16) enables the access to different tools and sources of information about Vitis. Explore them at will. We may always return to this welcome window by choosing **Help → Welcome** in the main toolbar.

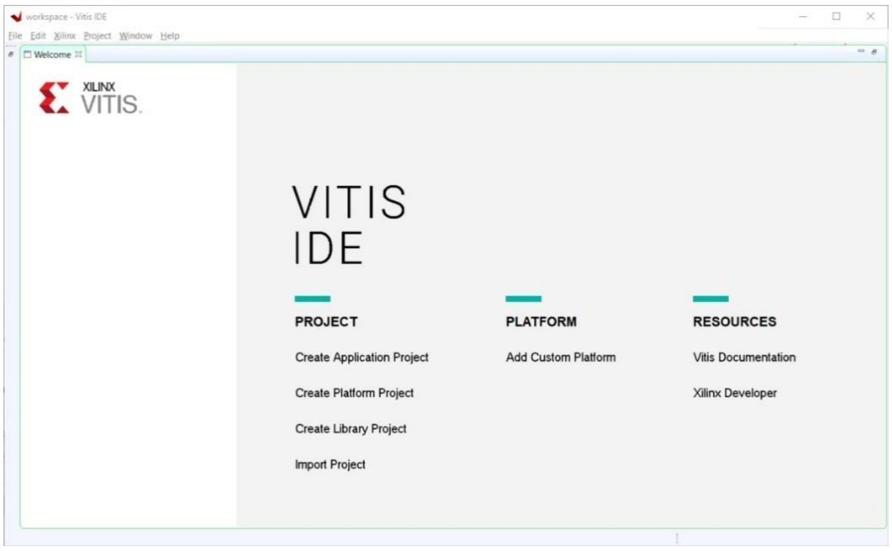


Figure 16

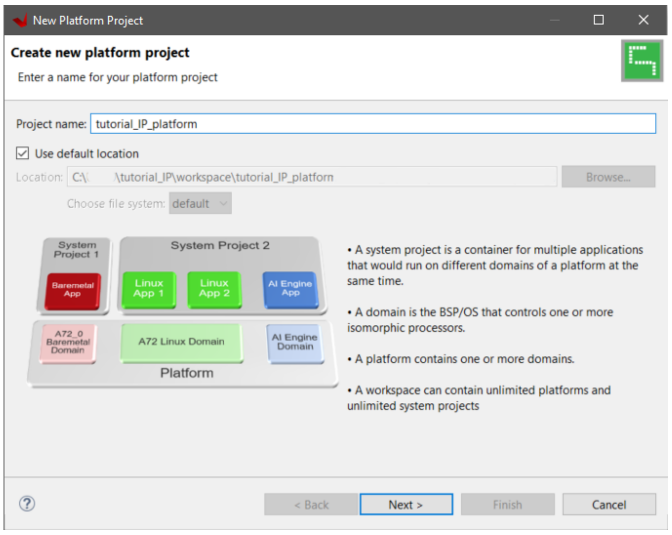
The **Help → Xilinx OS and Libraries Help** link gives access to information about a variety of Xilinx software packages, including drivers, libraries, board support packages, and complete operating systems to help us develop a software platform. Device drivers are documented along with the corresponding peripheral documentation. The embedded drivers and libraries

are hosted on the Xilinx wiki, which can be accessed at the “Embedded Driver and embedded Library Documentation”:
https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841745/Baremetal+Drivers+and+Libraries

14.Creating a New Platform Project in Vitis

As usual, the first application we are going to create is the famous “Hello World”, just to check if our project is OK, before creating an application to control our up/down counter.
Despite being possible to merge the creation of a new platform and a new application project in a single step, we are going to separate it for clarity purposes. So, we start by creating the platform project based on the hardware specification developed in Vivado, and after the applications to run on the Zynq processor, starting with the “Hello World” application, before creating an application to control our up/down counter.

1. Close the **Vitis IDE Welcome** window
2. In the Vitis workspace main toolbar select **File → New → Platform Project…**
3. The **Create new platform project** dialog box opens in the **New Platform Project** window
4. Type **tutorial_IP_platform** in the **Project Name** field (Figure 17)
5. Leave all the other fields with their default values and click **Next**


New Platform Project

Create new platform project
Enter a name for your platform project

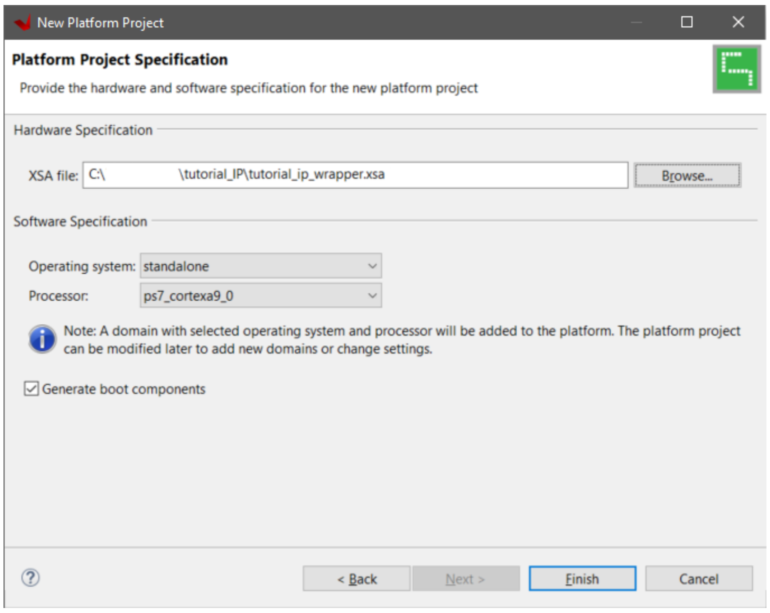
Project name: tutorial_IP_platform
☑ Use default location
Location: C:\ \tutorial_IP\workspace\tutorial_IP_platform
Choose file system: default

• A system project is a container for multiple applications that would run on different domains of a platform at the same time.
• A domain is the BSP/OS that controls one or more isomorphic processors.
• A platform contains one or more domains.
• A workspace can contain unlimited platforms and unlimited system projects

Figure 17

6. In the **Platform Project** page, select the **Create from hardware specification (XSA)** option
7. In the new **Platform Project Specification** page, navigate to the folder where the XSA file **tutorial_ip_wrapper.xsa** was created in Vivado to add it (Figure 18)
   Automatically, some **Software Specifications** are chosen by default: the **Operating system**, **standalone**, a single-threaded, simple operating system (OS) platform that provides the lowest layer of software modules used to access processor-specific functions (check the

Standalone v7.1 document, UG647), and the **Processor**, from the two ARM Cortex-A9 available in our Zynq xc7z010, were our applications are going to run - **ps7_cortexa9_0**. Other options for the operating system are **Linux** or **freertos10_xilinx**, a real-time operating system from Xilinx.


New Platform Project

Platform Project Specification

Provide the hardware and software specification for the new platform project

Hardware Specification

XSA file: C:\ \tutorial_IP\tutorial_ip_wrapper.xsa

Software Specification

Operating system: standalone

Processor: ps7_cortexa9_0

Note: A domain with selected operating system and processor will be added to the platform. The platform project can be modified later to add new domains or change settings.

☑ Generate boot components

Figure 18

The **Generate boot components** checkbox is selected by default, adding the First Stage Bootloader (FSBL) for Zynq to our project, which configures the FPGA with hardware bitstream (if it exists) and loads the Standalone (SA) Image or Operating System (OS) Image and takes the Cortex-A9 out of reset.

8. Leave all the other fields with their default values and click **Finish**

The hardware platform project is created within the Vitis workspace. It is generated with multiple domains, one we specified, and the ones required for the boot components. It can later be modified to add new domains.

In the **Main** tab of the **tutorial_IP_Platform** window of the **Design** perspective, we have access to the **Board Support Package** (BSP) settings and to the list of **Peripheral Drives** used by our platform by selecting **Board Support Package** under **tutorial_IP_Platform → ps7_cortexa9_0 → zynq_fsbl**. The address map of the system is accessible selecting the **Hardware Specification** tab at the bottom of the **Main** window, just above the Vitis **Console**. Notice that our **up_down_counter_IP_0** peripheral is also part of those lists.

When building the BSP project, files **ps7_init.c** and **ps7_init.h** contain the initialization code for the Zynq SoC Processing System and initialization settings in the form of **#define** constants for DDR, clocks, phase-locked loops (PLLs), and MIOs. Vitis uses these settings when initializing the processing system so that applications can be run on top of it.

Software drivers and applications can reference these constants in the code. These files are accessible through the **Explorer** window, on the left side of the Vitis workspace, at **tutorial_IP_platform → zynq_fsbl**

Also in the **Explorer** window, under **tutorial_IP_platform → hw**, there is a folder called **drivers**. Open this folder and go to **up_down_counter_IP_v1_0 → scr → up_down_counter_IP.h** and double click to open this file. This header file contains the C function declarations that enable us to read from and write to the four slave registers created when we added the AXI4-Lite interface. These basic functions are automatically created and added to our project when we package the IP. This header file must be included in our C files if we want to receive or send data from/to the IP. Explore it before proceeding.

9. To generate the platform, right-click the platform project **tutorial_IP_platform (Out-of-date)** and select **Build Project** (Figure 19)

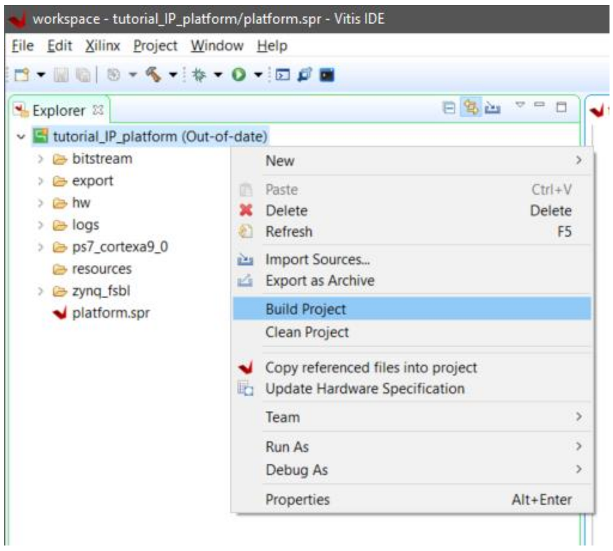


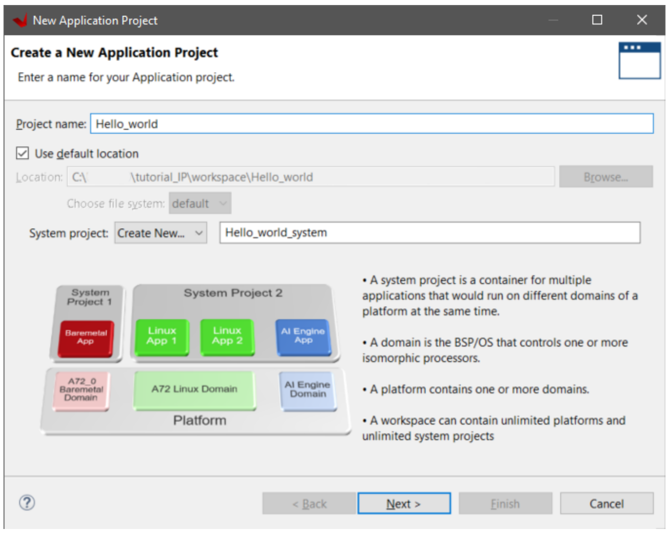
Figure 19

15.Creating a New Application Project in Vitis

Now that we have our platform, the next step is to create our first software application project, the famous "Hello World". Software application projects are the final application containers. The project directory that is created contains (or links to) your C/C++ source files, executable output file, and associated utility files, such as the Makefiles used to build the project.

1. In the Vitis workspace main toolbar select **File → New → Application Project...**
2. The **Create a New Application Project** dialog box opens in the **New Application Project** window

3. Type **Hello_world** in the **Project Name** field (Figure 20)
4. Leave all the other fields with their default values and click **Next**


New Application Project

Create a New Application Project

Enter a name for your Application project.

Project name: Hello_world

☑ Use default location

Location: C:\ \tutorial_IP\workspace\Hello_world

Choose file system: default

System project: Create New... Hello_world_system

• A system project is a container for multiple applications that would run on different domains of a platform at the same time.

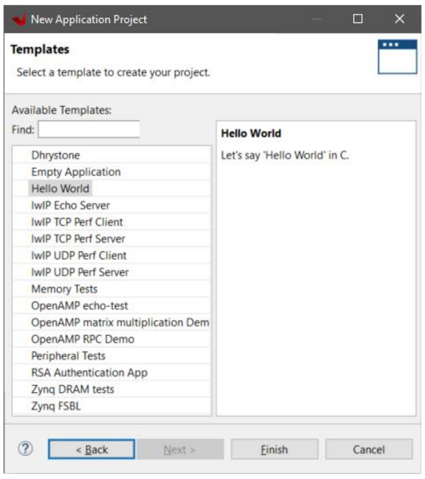
• A domain is the BSP/OS that controls one or more isomorphic processors.

• A platform contains one or more domains.

• A workspace can contain unlimited platforms and unlimited system projects

Figure 20

5. In the **Select a platform from repository** tab of the **Platform** page, select the newly created **tutorial_IP_platform**
6. In the **Domain** page leave all the fields at their default values and click **Next**
7. In the **Templates** page, select **Hello World** from the list of **Available Templates:** and click **Finish** (Figure 21)


New Application Project

Templates

Select a template to create your project.

Available Templates:

Find:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- lwIP TCP Perf Client
- lwIP TCP Perf Server
- lwIP UDP Perf Client
- lwIP UDP Perf Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Dem
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

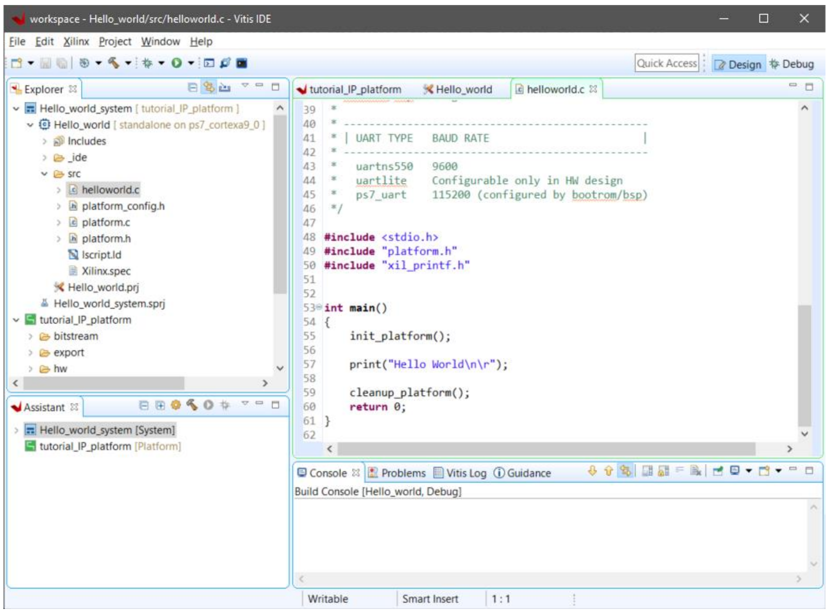
Hello World

Let's say 'Hello World' in C.

Figure 21

The application project is created in the Vitis workspace.

A new directory tree appears in the **Explorer** window on the left side of the workspace. Under the **Hello_world_system**, the **Hello_world** folder is our main working source folder. It contains all the binaries, .C, and .H files. Inside the **src** folder is our application file, **helloworld.c** To open the file in the main Vitis workspace window, double-click on it. You may modify its content if you want (Figure 22). This folder also contains an important file, the **lscript.ld**. This is a Xilinx auto-generated linker script file. It is used to control where different sections of an executable are placed in memory. Double-click to open it and to check its content.



```
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");

    cleanup_platform();
    return 0;
}

```

Figure 22

Two other files present in the **Hello_world** folder are the **Hello_world.prj**, which contains the **Application Project Settings,** and the **hello_world_system.sprj**, which contains the **System Project Settings** and the list of **Application projects**. Double-click each one to open them and check their content.

16.Compiling the Application Project

1. Right-click on the project folder and select **Build Project** (Figure 23)

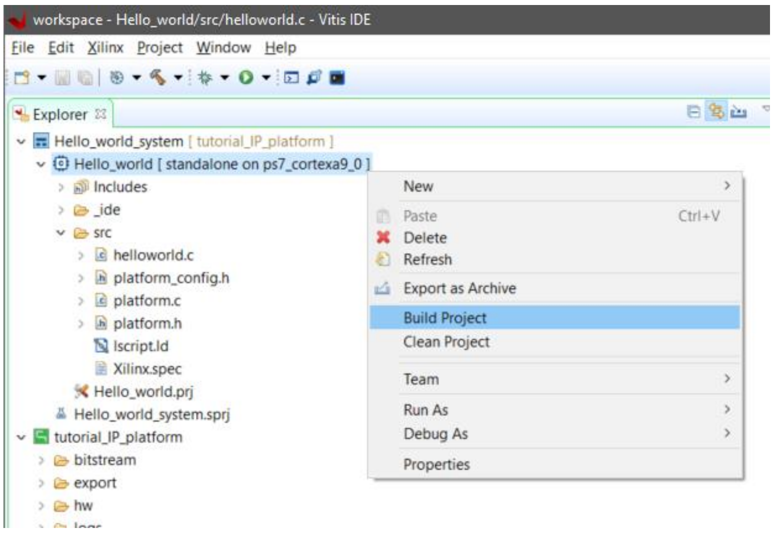


Figure 23

2. Check the **Console** window at the bottom of the Vitis workspace to see when the application project finishes compiling.

The build is the process of converting the source code files into a standalone software file that can be run on the Zynq processor.

17.Setting up the UART Terminal and Running the Application Project

Make sure that the ZYBO Z7-10 is turned on and connected to the host PC via the USB-JTAG port – **PROG UART** – of the board. This port serves a dual purpose as the ZYBO Z7-10 Programming port and as the USB-UART connection to the ZYNQ7 Processing System.

The Zybo Z7 supports three different boot modes: microSD, Quad SPI Flash, and JTAG. The current Zynq configuration boot mode is selected using the **Mode** jumper (JP5) in the Zybo-Z7 board. Since we are configuring the board using the JTAG interface, check that jumper JP5 on the board is selecting **JTAG** (Figure 24).

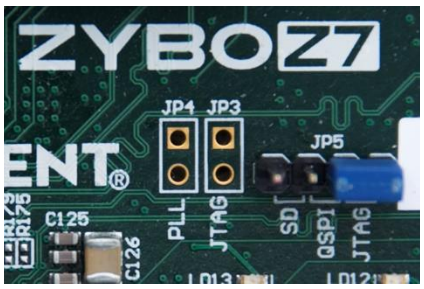


Figure 24

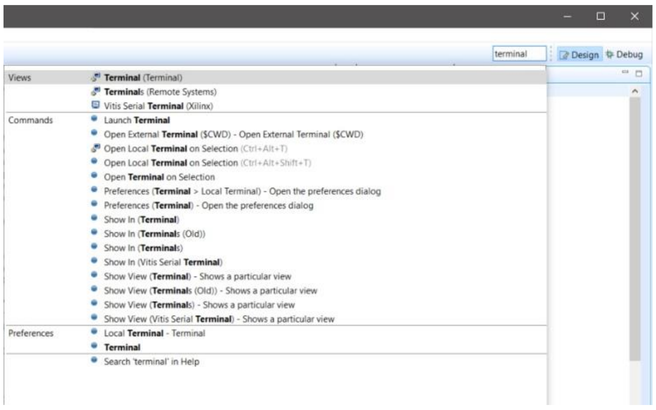
The Zybo Z7 has an onboard 16MB Quad-SPI (Serial Peripheral Interface) Flash memory. Quad-SPI-based interface memories are faster than traditional SPI-based as Quad-SPI uses 4 data lines (I0, I1, I2, and I3) as opposed to just 2 data lines (MOSI and MISO) on the traditional SPI-based interfaces. By default, the board comes with JP5 selecting the Quad-SPI Flash memory (QSPI). This enables the configuration of the FPGA with a manufacture demonstration example stored in the Quad-SPI Flash memory, when the board is powered-up.

The Vitis platform has its own incorporated serial terminal application, the **Vitis Serial Terminal**.

1. To open the **Vitis Serial Terminal** click on **Quick Access** near the top right corner of the Vitis workspace and type **terminal** (Figure 25)
2. Select **Vitis Serial Terminal (Xilinx)** from the list
3. In the new **Vitis Serial Terminal** click on the ➕ icon to add a port to the terminal

   Configure the ZYBO Z7-10 UART port to **Port:** COMx (the serial port number depends on your computer) with a **Baud Rate:** 115200 bit/s, the baud rate defined by default for

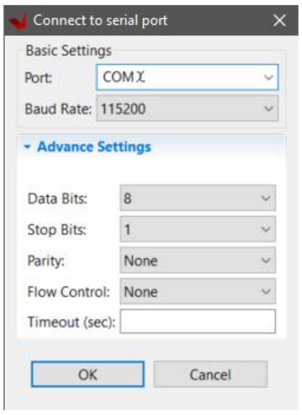
UART1, the one connected to the **PROG UART** connector in the ZYBO Z7-10, in the **PS-PL Configuration** of the **ZYNQ7 Processing System Re-customize IP** window.



Views	**Terminal** (Terminal)
	Terminals (Remote Systems)
	Vitis Serial **Terminal** (Xilinx)
Commands	Launch **Terminal**
	Open External **Terminal** ($CWD) - Open External Terminal ($CWD)
	Open Local **Terminal** on Selection (Ctrl+Alt+T)
	Open Local **Terminal** on Selection (Ctrl+Alt+Shift+T)
	Open **Terminal** on Selection
	Preferences (**Terminal** > Local Terminal) - Open the preferences dialog
	Preferences (**Terminal**) - Open the preferences dialog
	Show In (**Terminal**)
	Show In (**Terminal**s (Old))
	Show In (**Terminal**s)
	Show In (Vitis Serial **Terminal**)
	Show View (**Terminal**) - Shows a particular view
	Show View (**Terminal**s (Old)) - Shows a particular view
	Show View (**Terminal**s) - Shows a particular view
	Show View (Vitis Serial **Terminal**) - Shows a particular view
Preferences	Local **Terminal** - Terminal
	Terminal
	Search 'terminal' in Help

Figure 25

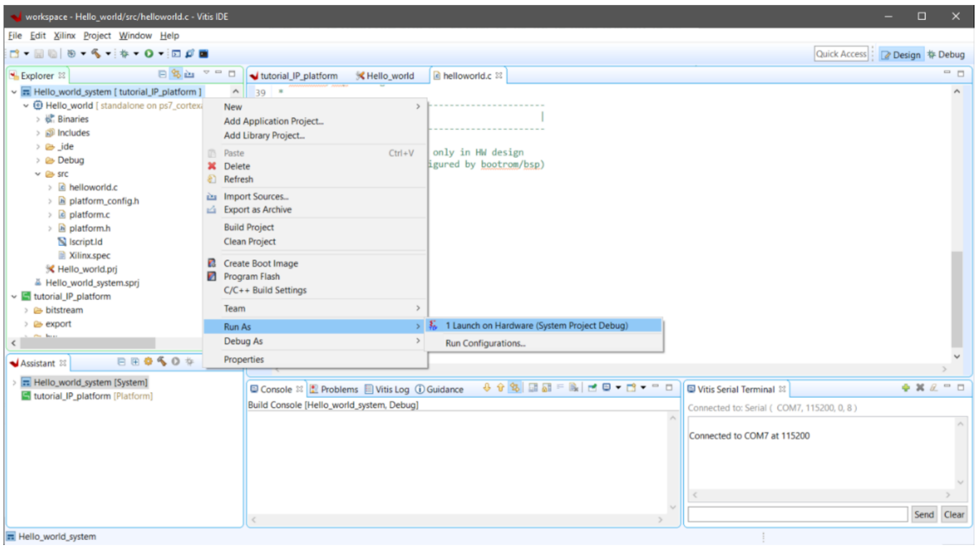
4. Configure the other **Advance Settings** to:
   - **Data Bits:** 8
   - **Stop Bits:** 1
   - **Parity:** None
   - **Flow Control:** None
5. Leave **Timeout (sec):** blank (Figure 26)



Port:	COM X
Baud Rate:	115200
Data Bits:	8
Stop Bits:	1
Parity:	None
Flow Control:	None
Timeout (sec):	

Figure 26

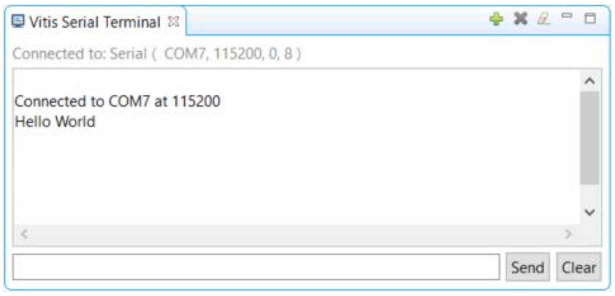
6. Click **OK** to close the configuration window
7. The message **Connected to COMx at 115200** appears in the **Vitis Serial Terminal** window
8. In the **Explorer** window on the left side of the Vitis workspace, select the **Hello_world_system [tutorial_IP_platform]**
9. Right click and select **Run as → 1 Launch on Hardware (System Project Debug)** (Figure 27)
10. Check the **Vitis Serial Terminal** (Figure 28)



```
Connected to COM7 at 115200
```

Figure 27

11. Everything seems to be working as expected! However, if nothing appears in your **Vitis Serial Terminal**, maybe you picked up the wrong port. Go back and change your serial port and repeat and run the application again.



```
Connected to COM7 at 115200
Hello World
```

Figure 28

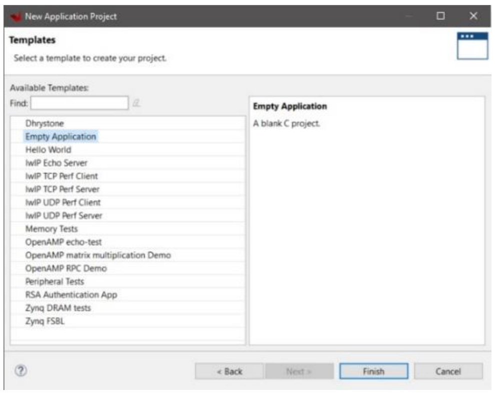
18.Implementing the up/down counter on the ZYNQ7 Processing System Processor

The next step is to test the up/down counter IP. The hardware platform we created in Vivado contains the up/down counter IP block with an AXI interface through which it is possible to control its functionality. The software to run on the ZYNQ7 Processing System processor establish the communication between the Vitis Serial Terminal or other Terminal and the up/down counter IP, via the serial port and the AXI bus that connect it of the Zynq processor.

In the Vitis environment, the same system project can contain multiple application projects (or applications). Thus, we just need to create the software to run on it.

1. In the main toolbar, select **File → New → Application Project…**
2. In the next window, type **up_down_counter** in the **Project Name** field
3. Leave all the other fields with their default values and click **Next**

4. In the **Select a platform from repository** tab of the **Platform** page, select **tutorial_IP_platform,** and click **Next**
5. In the **Domain** page leave all the fields at their default values and click **Next**
6. In the **Templates** page, select **Empty Application** from the list of **Available Templates:** and click **Finish** (Figure 29)



Select a template to create your project.

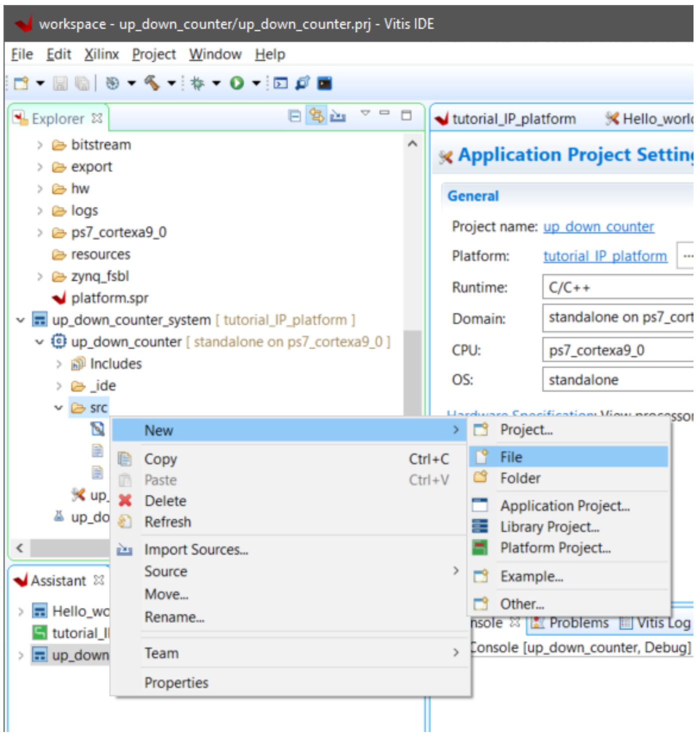
A blank C project.

Figure 29

The new application project is created in the Vitis workspace.

7. In the **Explorer** window, open the directory tree **up_down_counter_system → up_down_counter → src**

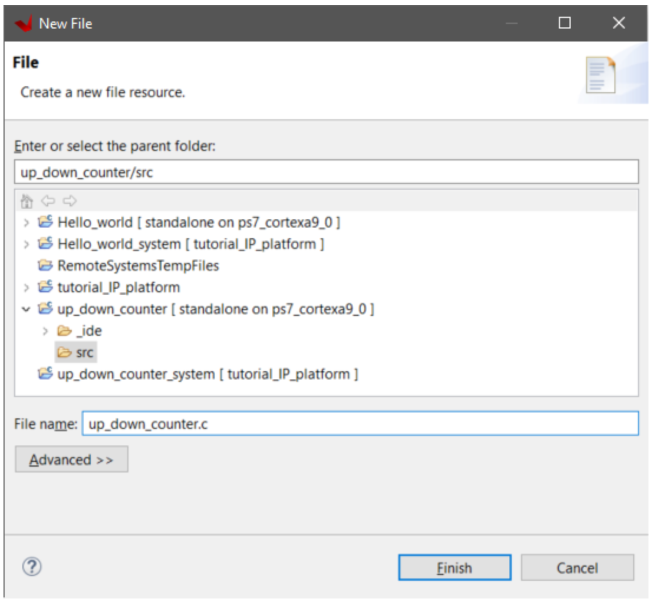
   Obviously, there are no source files yet. We must create our own .c source code file for the application.
8. Right click on **src → New** and select **File** (Figure 30)



Project name: up_down_counter
Platform: tutorial_IP_platform
Runtime: C/C++
Domain: standalone on ps7_cort
CPU: ps7_cortexa9_0
OS: standalone

Figure 30

9. In the new window, type **up_down_counter.c** in the **File name:** field, leaving the other fields unchanged, and click **Finish** (Figure 31)



New File
File
Create a new file resource.
Enter or select the parent folder:
up_down_counter/src
Hello_world [standalone on ps7_cortexa9_0]
Hello_world_system [tutorial_IP_platform]
RemoteSystemsTempFiles
tutorial_IP_platform
up_down_counter [standalone on ps7_cortexa9_0]
_ide
src
up_down_counter_system [tutorial_IP_platform]
File name: up_down_counter.c
Advanced >>
Finish
Cancel

Figure 31

The **up_down_counter.c** file opens in the Vitis workspace main window. Now, it is time to add the C code that will implement our up/down counter IP control

10. Copy and paste the following code into the file:

```
#include "xparameters.h"
#include "xuartps_hw.h"
#include "up_down_counter_IP.h"

u32 CMD = 0x00000001; //32-bit data write to IP register 0 – controls
                      // up/down counter - starts resetting the counter
u32 CONF = 0x00000000; //32-bit data read from IP register 0 – controls
                       // up/down counter
u32 COUNTER = 0x00000000; //32-bit data read from IP register 1 – current
                          // counter value
u8  RecByte =0x00;    //8-bit received from keyboard to control up/down
                      // counter


int main(){

    xil_printf("Start Test Counter\n\n\r");

    // Counter starts counting down due to CMD initial value 0010

    xil_printf("Counter control value = "); //prints initial value of
    xil_printf("%x", CMD);                  // counter control
    xil_printf("\n\n\r");

    // sends initial value to start counter
    Xil_Out32(XPAR_UP_DOWN_COUNTER_IP_0_S00_AXI_BASEADDR, CMD);

    //reads value sent to confirm good reception
    CONF = Xil_In32(XPAR_UP_DOWN_COUNTER_IP_0_S00_AXI_BASEADDR);

    xil_printf("Confirm counter control value = ");  //prints value read to
    xil_printf("%x", CONF);                           // confirm good reception
    xil_printf("\n\n\r");
    //reads current counter value
    COUNTER = Xil_In32(XPAR_UP_DOWN_COUNTER_IP_0_S00_AXI_BASEADDR+4);

    xil_printf("Current counter value = ");  //prints current counter value
    xil_printf("%x", COUNTER);
    xil_printf("\n\n\r");

    xil_printf("Counter control values\n\r");  //prints possible values for
    xil_printf("* direction - enable - reset\n\r");      //counter control
    xil_printf("* enable and reset active high\n\r");
    xil_printf("Counter reset and stop = 1 (bx001)\n\r");
    xil_printf("Counting up = 6 (bx110)\n\r");
    xil_printf("Counting down = 2 (bx010)\n\r");
    xil_printf("Stop counter = 0 (bx000)\n\n\r");

    while(1) {

            xil_printf("New counter control value ? =  ");

            // waits for new counter control value to be received
            RecByte = XUartPs_RecvByte (XPAR_PS7_UART_1_BASEADDR);

            RecByte = RecByte - 0x30; //keyboard value received in ASCII -
                                      // transforms to decimal value
            xil_printf("%x", RecByte);  //prints received value
            xil_printf("\n\n\r");

            CMD = 0x00000000 + RecByte; //adjust to 32-bit

            //sends new control value to IP
            Xil_Out32(XPAR_UP_DOWN_COUNTER_IP_0_S00_AXI_BASEADDR, CMD);

            //reads value sent to confirm good reception
            CONF = Xil_In32(XPAR_UP_DOWN_COUNTER_IP_0_S00_AXI_BASEADDR);

            xil_printf("Confirm counter control value = ");  //prints value
            xil_printf("%x", CONF);        // read to confirm good reception
            xil_printf("\n\n\r");

            //reads current counter value
            COUNTER = Xil_In32(XPAR_UP_DOWN_COUNTER_IP_0_S00_AXI_BASEADDR+4);

            xil_printf("Current counter value = ");  //prints current counter
            xil_printf("%x", COUNTER);                // value
            xil_printf("\n\n\r");

            //SDK Terminal - retrieves one more byte because the SDK
            //Terminal adds a Carriage Return character (0D)
            //Comment this instruction line if using Tera Term
            RecByte = XUartPs_RecvByte(XPAR_PS7_UART_1_BASEADDR);

            }

return 0;

}
```

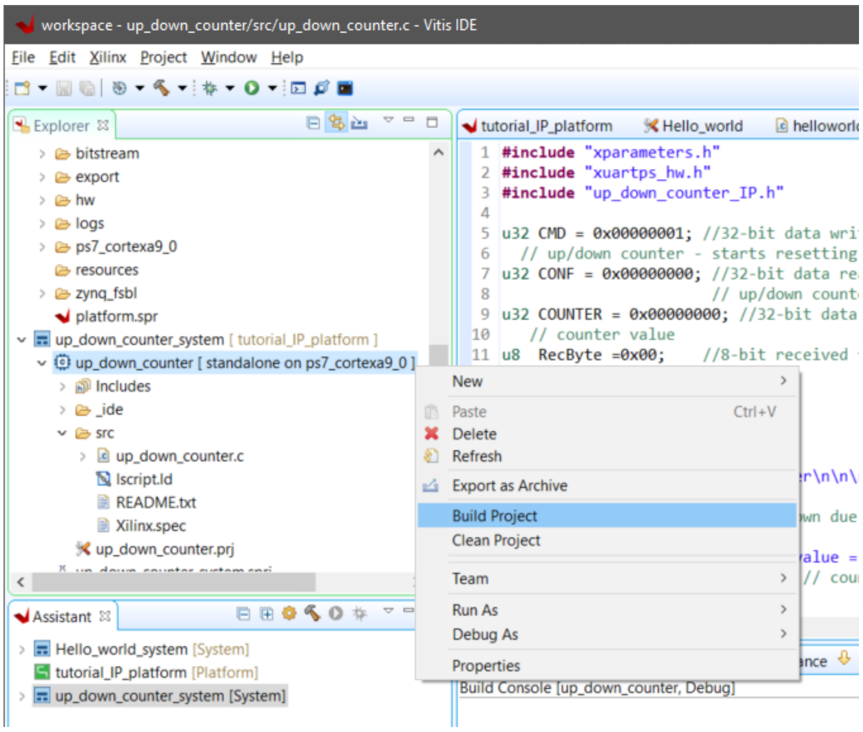
11. Check for any typos or errors and read the comments to understand which each one of the instructions and declarations is intended to do and how the program is expected to work The function `u8 XUartPs_RecvByte ( u32 BaseAddress )` operates in polled mode and blocks until a byte has received. Any time the program reaches this instruction, it stops and waits for a new byte to be sent through the Serial Terminal. The Vitis Serial

Terminal automatically adds a Carriage Return (CR) character after the user sends a new command value. The last instruction

```
RecByte = XUartPs_RecvByte(XPAR_PS7_UART_1_BASEADDR);
```

is used to read this character before waiting for a new command value, avoiding the up/down counter to consider the CR character as a new command value. If using a different terminal, for example, the Tera Term (Tera Term Terminal Emulator Installation Guide, UG1036) that does not automatically send a CR character, just comment this instruction line.

12. Save the **up_down_counter_IP_test.c** by selecting **File → Save** or clicking in the **Save** icon in the main toolbar
13. To compile the new application project, right-click on the project folder and select **Build Project** (Figure 32)

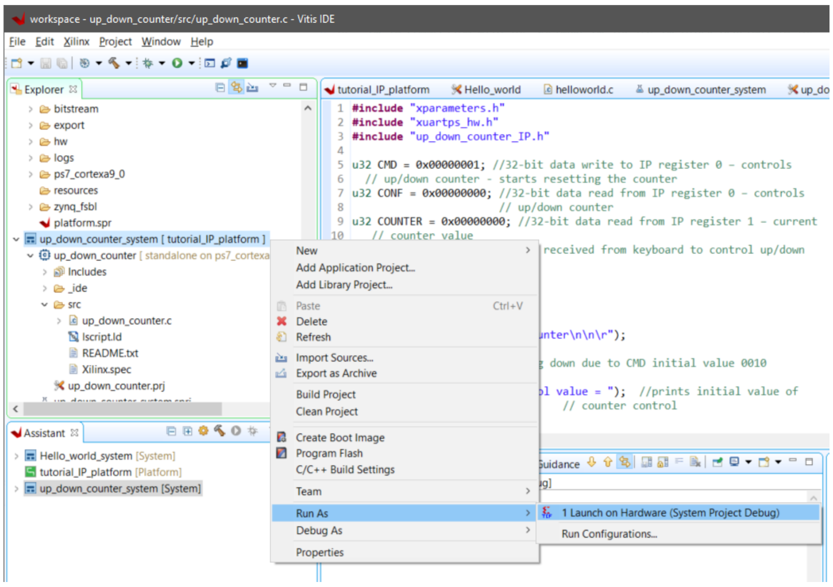


```
#include "xparameters.h"
#include "xuartps_hw.h"
#include "up_down_counter_IP.h"

u32 CMD = 0x00000001; //32-bit data wri
  // up/down counter - starts resetting
u32 CONF = 0x00000000; //32-bit data re
                        // up/down count
u32 COUNTER = 0x00000000; //32-bit data
   // counter value
u8  RecByte =0x00;     //8-bit received
```

Figure 32

14. Make sure that the ZYBO Z7-10 is turned on and connected to the host PC via the USB-JTAG port
15. In the **Explorer** window on the left, select the **up_down_counter_system [tutorial_IP_platform]**
16. Right click and select **Run as → 1 Launch on Hardware (System Project Debug)** (Figure 33)

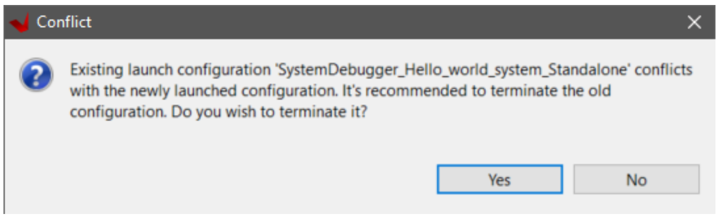


```
#include "xparameters.h"
#include "xuartps_hw.h"
#include "up_down_counter_IP.h"

u32 CMD = 0x00000001; //32-bit data write to IP register 0 - controls
  // up/down counter - starts resetting the counter
u32 CONF = 0x00000000; //32-bit data read from IP register 0 - controls
                       // up/down counter
u32 COUNTER = 0x00000000; //32-bit data read from IP register 1 - current
   // counter value
```

Figure 33

17. If the **Hello_world** application is still running in the ZYNQ7 Processing System, a **Conflict** window appears, asking if we wish to terminate the previous launched configuration (Figure 34)
18. Click **Yes**


Conflict

Existing launch configuration 'SystemDebugger_Hello_world_system_Standalone' conflicts with the newly launched configuration. It's recommended to terminate the old configuration. Do you wish to terminate it?

Yes No

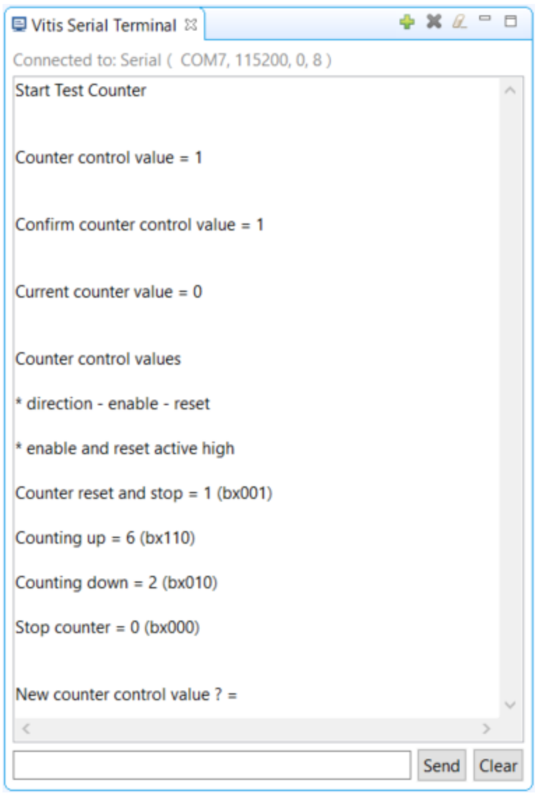
Figure 34

19. Check the **Serial Terminal** and the board LEDs

    In the **Serial Terminal** we may see the first command to be sent to the board is '**1**', meaning that the counter is reset and stopped. This is the reason why all board LEDs remain off. A list of commands is also displayed (see Figure 35). The last line shows the system is waiting for a **New counter control value ? =**
20. In the command line at the bottom of the **Serial Terminal** write '**6**' and click the **Send** button on the right
21. The LEDs on the board show now that the counter is counting up, as expected
22. Try with other command values and check the up/down counter operation

Everything seems to be working as expected!

You have completed the *Creating, Packaging and Controlling an IP in Vivado/Vitis - Quick Start Tutorial for the Digilent Zybo Z7-10 Zynq 7000 ARM/FPGA SoC Platform board*.



```
Connected to: Serial ( COM7, 115200, 0, 8 )
Start Test Counter


Counter control value = 1


Confirm counter control value = 1


Current counter value = 0


Counter control values

* direction - enable - reset

* enable and reset active high

Counter reset and stop = 1 (bx001)

Counting up = 6 (bx110)

Counting down = 2 (bx010)

Stop counter = 0 (bx000)


New counter control value ? =
```

Figure 35

References:

AXI Reference Guide, *Vivado Design Suite User Guide*, UG1037 (v4.0) July 15, 2017

AXI Verification IP v1.1, *LogiCORE IP Product Guide*, Vivado Design Suite, PG267 October 30, 2019

AXI4-Stream Verification IP v1.1, *LogiCORE IP Product Guide*, Vivado Design Suite, PG277 October 30, 2019

Creating and Packaging Custom IP, *Vivado Design Suite User Guide*, UG1118 (v2019.2) March 2, 2020

Creating, Packaging Custom IP Tutorial, *Vivado Design Suite Tutorial*, UG1119 (v2019.2) March 2, 2020

Design Analysis and Closure Techniques, *Vivado Design Suite User Guide*, UG906 (v2019.2) October 30, 2019

Design Flows Overview, *Vivado Design Suite User Guide*, UG892 (v2018.3) Dec. 5, 2018

Designing IP Subsystems Using IP Integrator, *Vivado Design Suite User Guide*, UG994 (v2019.2) November 20, 2019

Designing IP Subsystems Using IP Integrator, *Vivado Design Suite User Guide*, UG994 (v2019.2) October 30, 2019

Designing with IP, *Vivado Design Suite User Guide*, UG896 (v2019.2) March 3, 2020

Embedded Software Development, *Vitis Unified Software Platform Documentation*, UG1400 (v2019.2) March 18, 2020

Getting Started, *Vivado Design Suite User Guide*, UG910 (v2019.2) October 30, 2019

I/O and Clock Planning, *Vivado Design Suite User Guide*, UG899 (v2019.2) October 30, 2019

Implementation, *Vivado Design Suite User Guide*, UG904 (v2019.2) December 18, 2019

Logic Simulation, *Vivado Design Suite Tutorial*, UG937 (v2019.2) October 30, 2019

Logic Simulation, *Vivado Design Suite User Guide*, UG900 (v2019.2) October 30, 2019

OS and Libraries Document Collection, *Xilinx Standalone Library Documentation*, UG643 (2019.2) December 9, 2019

Synthesis, *Vivado Design Suite User Guide*, UG901 (v2019.2) January 27, 2020

System-Level Design Entry, *Vivado Design Suite User Guide*, UG895 (v2019.2) October 30, 2019

Tera Term Terminal Emulator Installation Guide, UG1036 (v1.0.1) January 16, 2019

UltraFast Design Methodology, *Vivado Design Suite User Guide*, UG949 (v2019.2) December 6, 2019

Using Constraints, *Vivado Design Suite Tutorial*, UG945 December 6, 2019

Using Constraints, *Vivado Design Suite User Guide*, UG903 (v2019.2) December 12, 2019

Using the Vivado IDE, *Vivado Design Suite User Guide*, UG893 (v2019.2) October 30, 2019

Zynq-7000 All Programmable SoC Software Developers Guide, UG821 (v12.0) September 30, 2015

Zynq-7000 All Programmable SoC Verification IP v1.0, *Data Sheet*, DS940 (v1.0) April 28, 2017

Zynq-7000 SoC: Embedded Design Tutorial, *A Hands-On Guide to Effective Embedded System Design*, UG1165 (v2019.2) October 30, 2019

Version 1

Manuel Gericota – April 2020

Revision table

v. 0	Initial release	March 2019
v. 1	Updated version for Vivado 2019.2 with Vitis Updated reference list	April 2020