

Ultra96: Hello World

Overview

Once a ZU+ Hardware Platform is created and exported from Vivado, the next step is to create an application targeted at the platform and see it operating in hardware. This tutorial will show how to do that with the simplest of all software applications – Hello World.

Objectives

When this tutorial is complete, you will be able to:

- Import a ZU+ Hardware Platform into SDK
- Create a BSP
- Add a new application based on a Xilinx-provided template in SDK
- Run the application on the Ultra96 hardware

Experiment Setup

Software

The software used to test this reference design is:

- Windows-7 64-bit
- Xilinx SDK 2018.2
- USB-JTAG device driver
- Terminal software, such as the built-in SDK terminal, TeraTerm, or Putty

Hardware

The hardware setup used to test this reference design includes:

- Win-7 PC with the following recommended memory¹
 - 4 GB Typical and 5 GB Peak RAM available for the Xilinx tools to complete a XCZU3EG design
- Ultra96
- 96Boards Power Supply
- USB-JTAG (any compatible Xilinx USB-JTAG)
 - Avnet USB-to-JTAG/UART Pod (available September 2018)
 - Digilent HS3 (<http://avnet.me/jtaghs3>) with flyleads
- USB-UART
 - Avnet USB-to-JTAG/UART Pod (available September 2018)
 - Any other USB-UART dongle

¹ Refer to <https://www.xilinx.com/products/design-tools/vivado/memory.html>

Experiment 1: Import the Hardware Platform

The first step is to launch SDK and then import a hardware platform. An easy way to do this is to use the **File → Launch SDK** feature in Vivado. However, that obscures what is really happening, so these instructions are written explicitly to walk you through those steps.

1. Launch SDK by selecting **Start → All Programs → Xilinx Design Tools → SDK 2018.2 → Xilinx SDK 2018.2**.
2. Select a workspace in any location of your choosing (make sure to avoid a workspace with a space in the pathname though). Click **OK**.

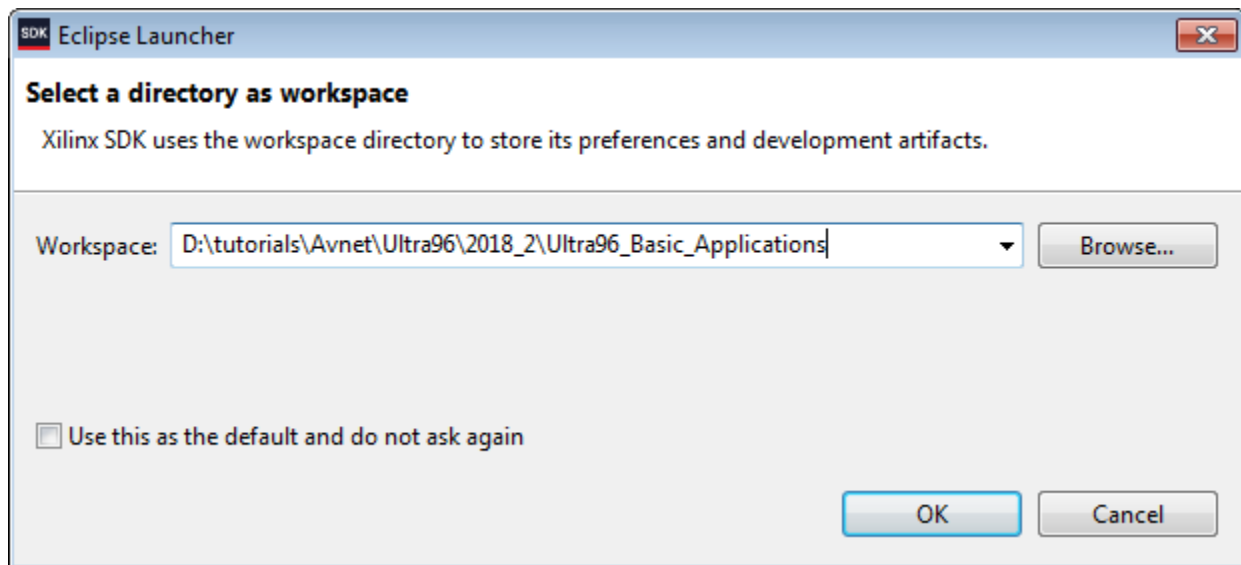


Figure 1 – SDK Workspace

3. If at any time you get a Windows Security Alert, select the first two checkboxes, then click **Allow access**, then click **Yes**.

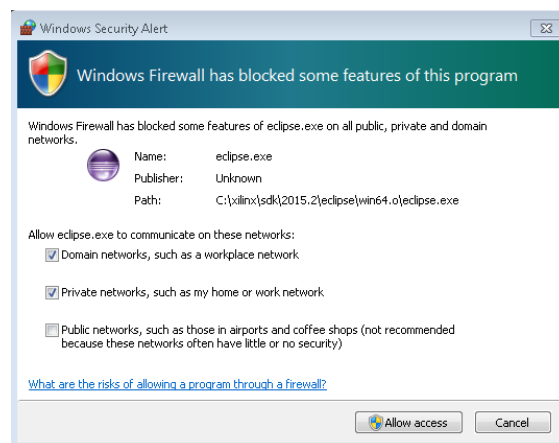


Figure 2 – Windows Security Alert from SDK

4. Close the *Welcome* screen by clicking the  next to *Welcome* on the tab.

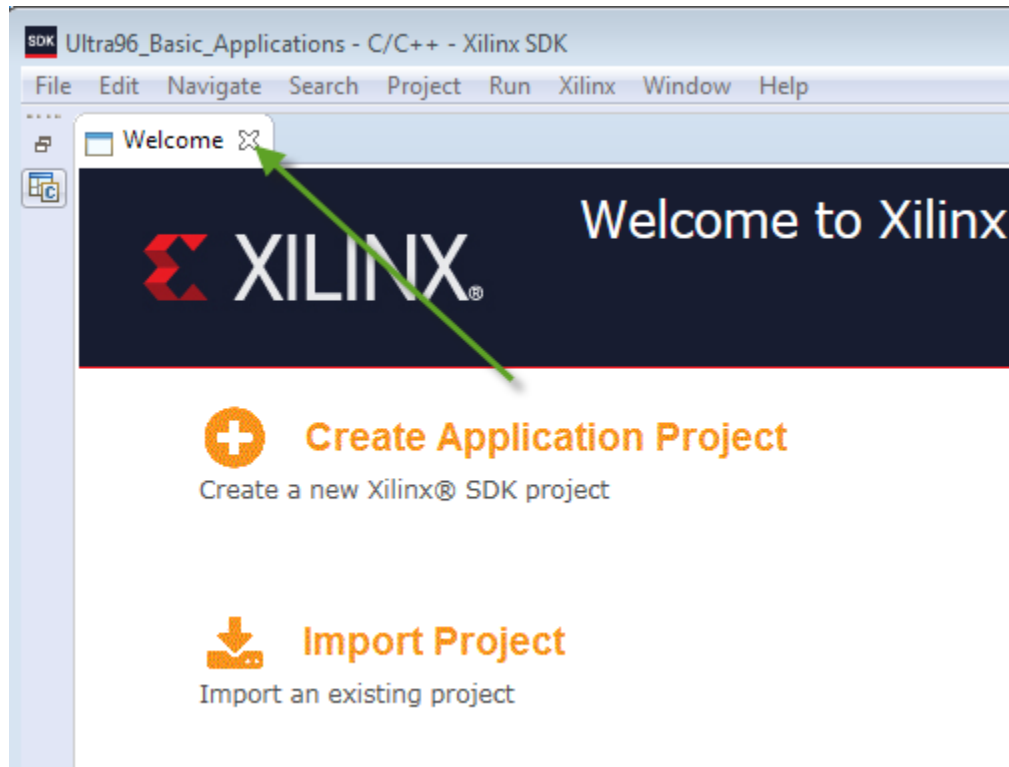


Figure 3 – Close Welcome Screen

Now we will import the ZU+ hardware platform that was designed and built during the first tutorial.

5. Select **File → New → Project**.
6. Expand the *Xilinx* item, and select **Hardware Platform Specification**. Click **Next >**.

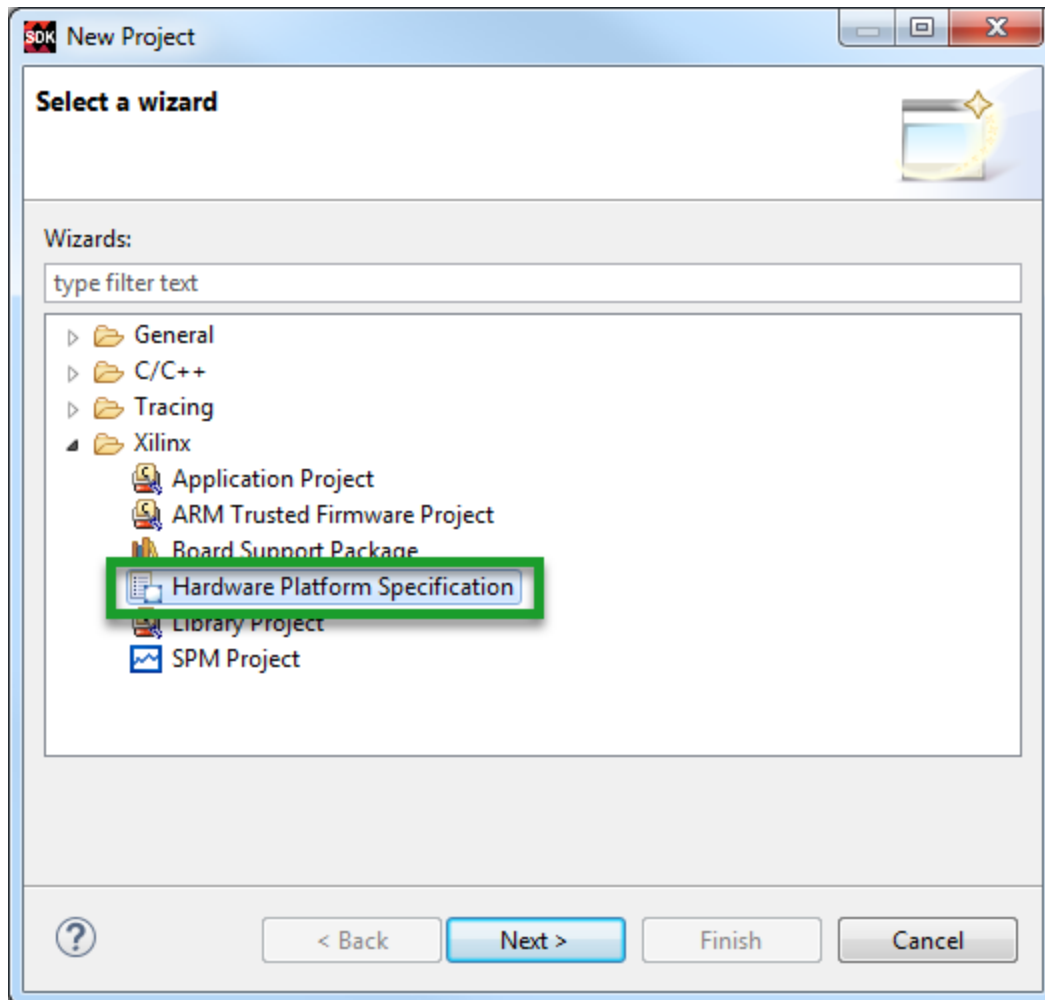


Figure 4 – Creating a New Hardware Platform

If you had simply launched SDK from Vivado, it would have automatically named and imported your hardware platform for you. The disadvantage in this method is that it obscures how the hardware platform gets imported. For consistency, this tutorial will use the same default name that Vivado would have used.

7. Insert **hw_platform_0** for the *Project name*.
8. Click **Browse** and select the `System_wrapper.hdf` file generated during the Export process from Vivado. This will be included in the archive provided by the hardware engineer. Or, if you are continuing from the first tutorial, you will find it in a similar location as here:

```
C:\Avnet\Ultra96\Projects\Ultra96_Basic_System\Ultra96_Basic_System.sdk\
```

9. After selecting `System_wrapper.hdf`, click **Open**.
10. The Bitstream is embedded in the HDF, so it is not separately specified here. Click **Finish**.

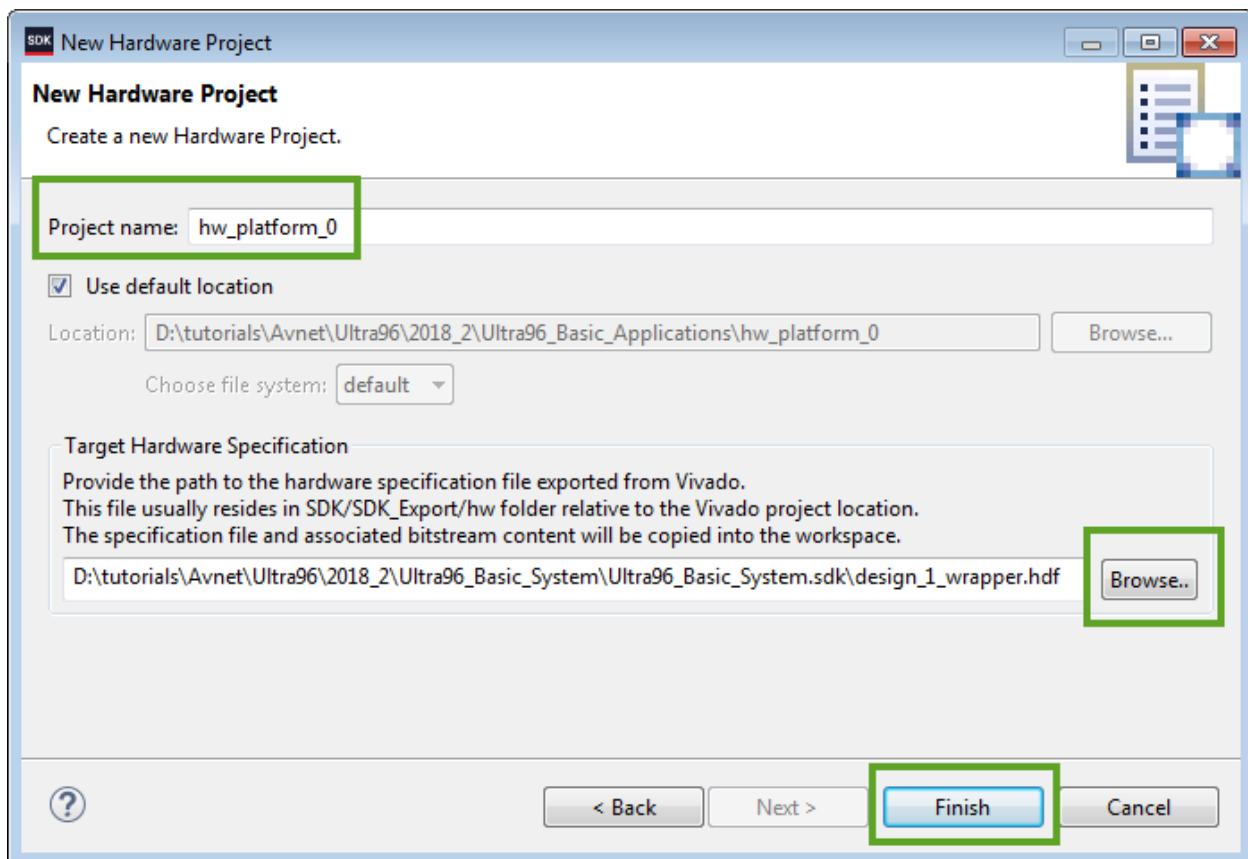


Figure 5 – Import Hardware Platform from Vivado

11. Notice the ZU+ hardware platform is now visible in the *Project Explorer*.

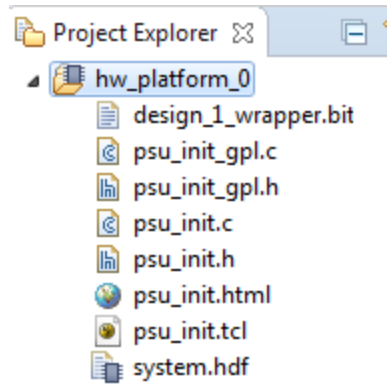


Figure 6 – Hardware Platform Imported and Ready for Use

If you select the HDF file, SDK will show you information about the hardware platform (not the HDF raw code itself).

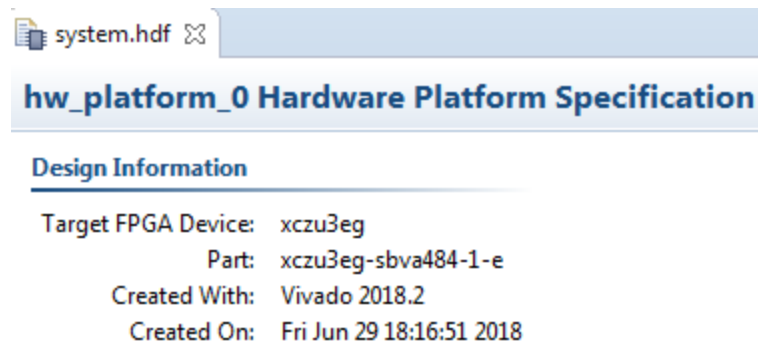


Figure 7 – system.hdf Design Information

Address Map for processor psu_cortexa53_[0-3]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg	Segment	TrustZone	AccessType
psu_gdma_1	0xfd510000	0xfd51ffff		REGISTER		NonSecure	Read/Write
psu_gdma_2	0xfd520000	0xfd52ffff		REGISTER		NonSecure	Read/Write
psu_gdma_3	0xfd530000	0xfd53ffff		REGISTER		NonSecure	Read/Write
psu_crf_apb	0xfd1a0000	0xfd2dffff		REGISTER		NonSecure	Read/Write
psu_gdma_4	0xfd540000	0xfd54ffff		REGISTER		NonSecure	Read/Write
psu_gdma_5	0xfd550000	0xfd55ffff		REGISTER		NonSecure	Read/Write
psu_gdma_6	0xfd560000	0xfd56ffff		REGISTER		NonSecure	Read/Write
psu_gdma_7	0xfd570000	0xfd57ffff		REGISTER		NonSecure	Read/Write
psu_sd_0	0xff160000	0xff16ffff		REGISTER		NonSecure	Read/Write
psu_pmu_global_0	0xffd80000	0xffdbffff		REGISTER		NonSecure	Read/Write
psu_smmu_gpv	0xfd800000	0xfdffffff		REGISTER		NonSecure	Read/Write
psu_siou	0xfd3d0000	0xfd3dffff		REGISTER		NonSecure	Read/Write
psu_smmu_reg	0xfd5f0000	0xfd5fffff		REGISTER		NonSecure	Read/Write
psu_sd_1	0xff170000	0xff17ffff		REGISTER		NonSecure	Read/Write
psu_afi_4	0xfd3a0000	0xfd3affff		REGISTER		NonSecure	Read/Write
psu_afi_5	0xfd3b0000	0xfd3bffff		REGISTER		NonSecure	Read/Write
psu_afi_6	0xff9b0000	0xff9bffff		REGISTER		NonSecure	Read/Write
psu_afi_0	0xfd360000	0xfd36ffff		REGISTER		NonSecure	Read/Write
psu_afi_1	0xfd370000	0xfd37ffff		REGISTER		NonSecure	Read/Write
psu_dpdma	0xfd4c0000	0xfd4cffff		REGISTER		NonSecure	Read/Write
psu_afi_2	0xfd380000	0xfd38ffff		REGISTER		NonSecure	Read/Write
psu_afi_3	0xfd390000	0xfd39ffff		REGISTER		NonSecure	Read/Write
psu_gdma_0	0xfd500000	0xfd50ffff		REGISTER		NonSecure	Read/Write
psu_ttc_0	0xff110000	0xff11ffff		REGISTER		NonSecure	Read/Write
psu_lpd_slcr_secure	0xff4b0000	0xff4dffff		REGISTER		NonSecure	Read/Write

**Figure 8 – Address Maps for Cortex A53 and R5 Processors, and PMU
(A53 Address Map Shown)**

IP blocks present in the design

psu_gdma_1	psu_gdma	1.0
psu_gdma_2	psu_gdma	1.0
psu_gdma_3	psu_gdma	1.0
psu_gdma_4	psu_gdma	1.0
psu_crf_apb	psu_crf_apb	1.0
psu_gdma_5	psu_gdma	1.0
psu_gdma_6	psu_gdma	1.0
psu_gdma_7	psu_gdma	1.0
psu_sd_0	psu_sd	1.0
psu_pmu_global_0	psu_pmu_global_0	1.0
psu_smmu_gpv	psu_smmu_gpv	1.0
psu_siou	psu_siou	1.0
psu_smmu_reg	psu_smmu_reg	1.0
psu_bbram_0	psu_bbram_0	1.0
psu_sd_1	psu_sd	1.0
psu_r5_tcm_ram_global	psu_r5_tcm_ram	1.0
psu_afi_4	psu_afi	1.0
psu_afi_5	psu_afi	1.0
psu_afi_6	psu_afi	1.0
psu_afi_0	psu_afi	1.0
psu_dpdma	psu_dpdma	1.0
psu_afi_1	psu_afi	1.0
psu_afi_2	psu_afi	1.0
psu_gdma_0	psu_gdma	1.0
psu_afi_3	psu_afi	1.0
psu_r5_0_btcn_lockstep	psu_r5_0_btcn_lockstep	1.0
psu_r5_0_atcm_lockstep	psu_r5_0_atcm_lockstep	1.0
psu_ttc_0	psu_ttc	1.0
psu_cortexr5_1	psu_cortexr5	1.0
psu_cortexr5_0	psu_cortexr5	1.0
psu_lpd_slcr_secure	psu_lpd_slcr_secure	1.0
psu_gpu	psu_gpu	1.0
psu_ttc_3	psu_ttc	1.0
psu_ttc_1	psu_ttc	1.0
psu_ttc_2	psu_ttc	1.0
psu_pmu_ram	psu_pmu_ram	1.0
psu_fpd_gpv	psu_fpd_gpv	1.0
psu_cci_reg	psu_cci_reg	1.0
psu_pmu_iomodule	psu_pmu_iomodule	1.0
psu_ocm_ram_0	psu_ocm_ram_0	1.0
psu_spi_0	psu_spi	1.0
psu_spi_1	psu_spi	1.0

Figure 9 – IP Blocks (partial list shown)

Experiment 2: BSP

Next, we will create a bare metal board support package, which Xilinx calls Standalone. This will assemble and compile various drivers that relate to the peripherals in the hardware platform for later use in our applications.

1. In SDK select **File → New → Board Support Package**.
2. Accept the default settings for the standalone BSP OS. Click **Finish**.

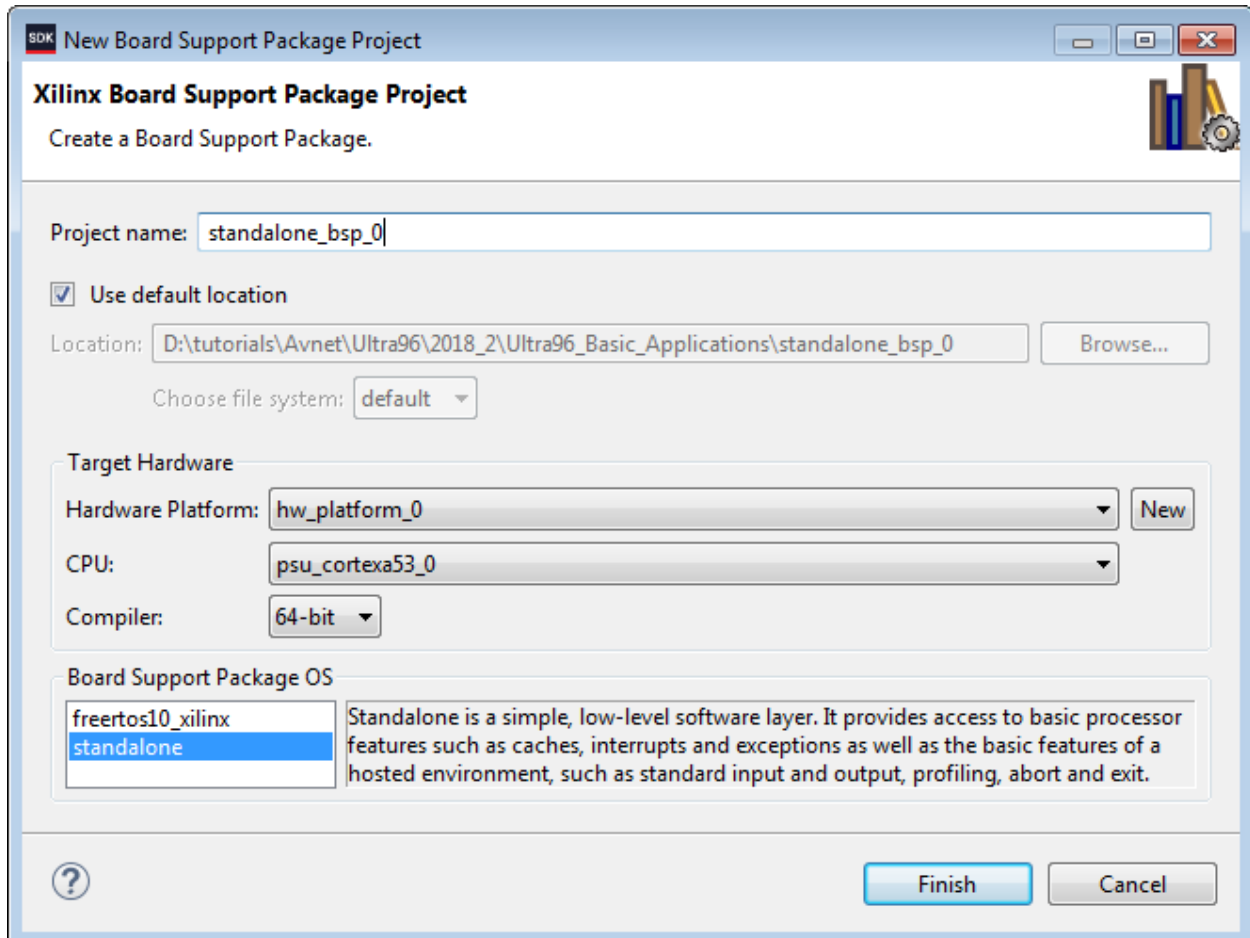


Figure 10 – Standalone BSP

3. In the *Board Support Package Settings*, select standalone. Change the *stdin* and *stdout* Value to **psu_uart_1**. This is done to align with the Ultra96's UART Serial connection.

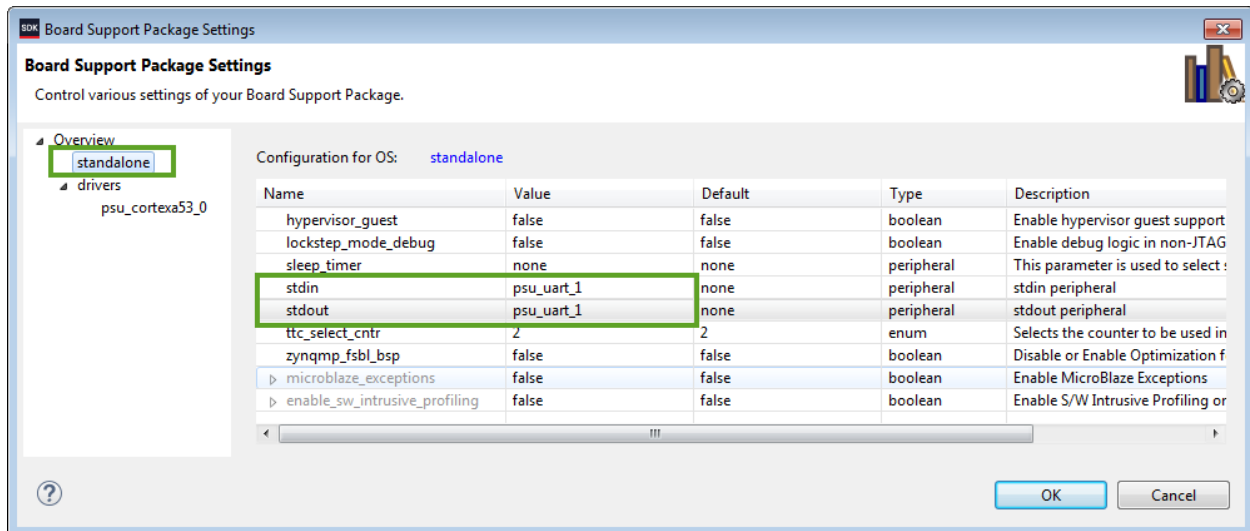


Figure 11 – Board Support Package Settings

Based on the modified settings in SDK, the BSP will automatically be built once it is added to the project. This may take a minute to compile the new BSP. The progress may be seen in the *Console* tab.

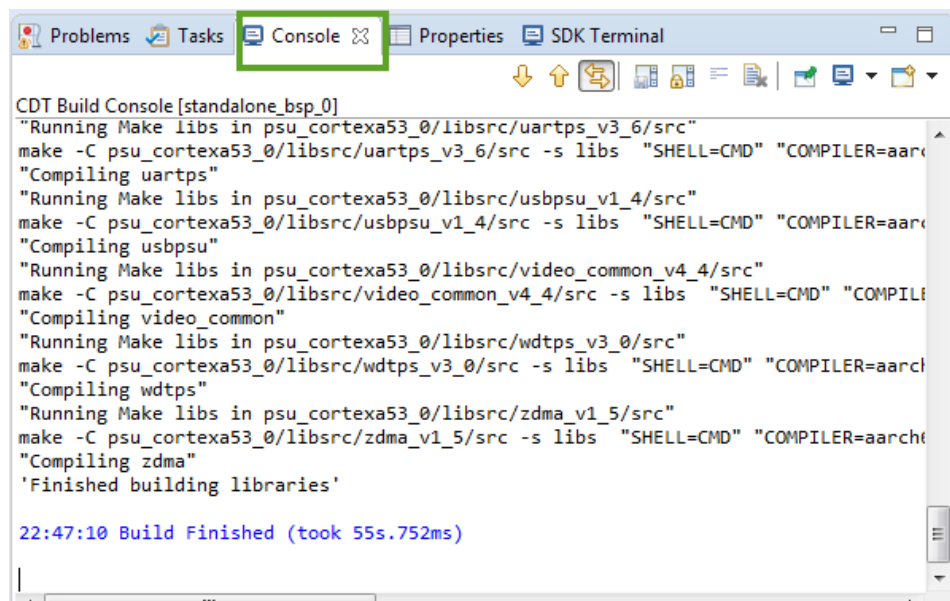


Figure 12 – BSP Built

The standalone_bsp_0 is now visible in the *Project Explorer*.

4. Expand **standalone_bsp_0** under the *Project Explorer*.

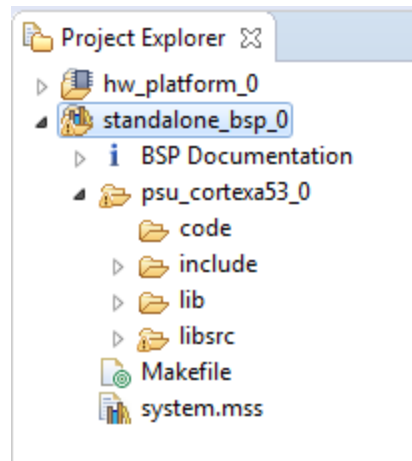


Figure 13 – BSP Added to the Project

Experiment 3: Add Application

With a Hardware Platform and BSP, we are now ready to add an application and run something on the board.

1. In SDK, select **File** → **New** → **Application Project**.
2. In the **Project Name** field type in `Hello_Ultra96`. Change the **BSP** to the existing StandAlone BSP. Click **Next >**.

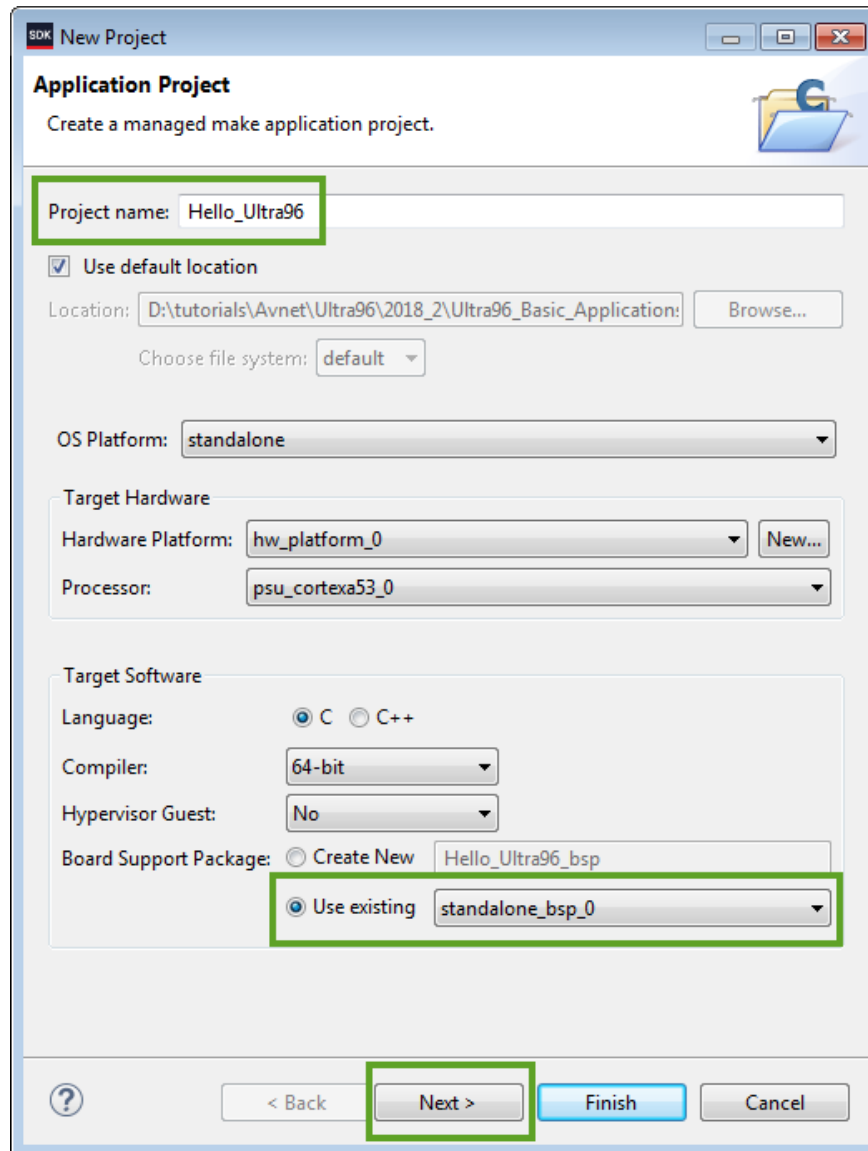


Figure 14 - New Application Wizard

3. Select **Hello World** from the *Available Templates* field. Click **Finish**.

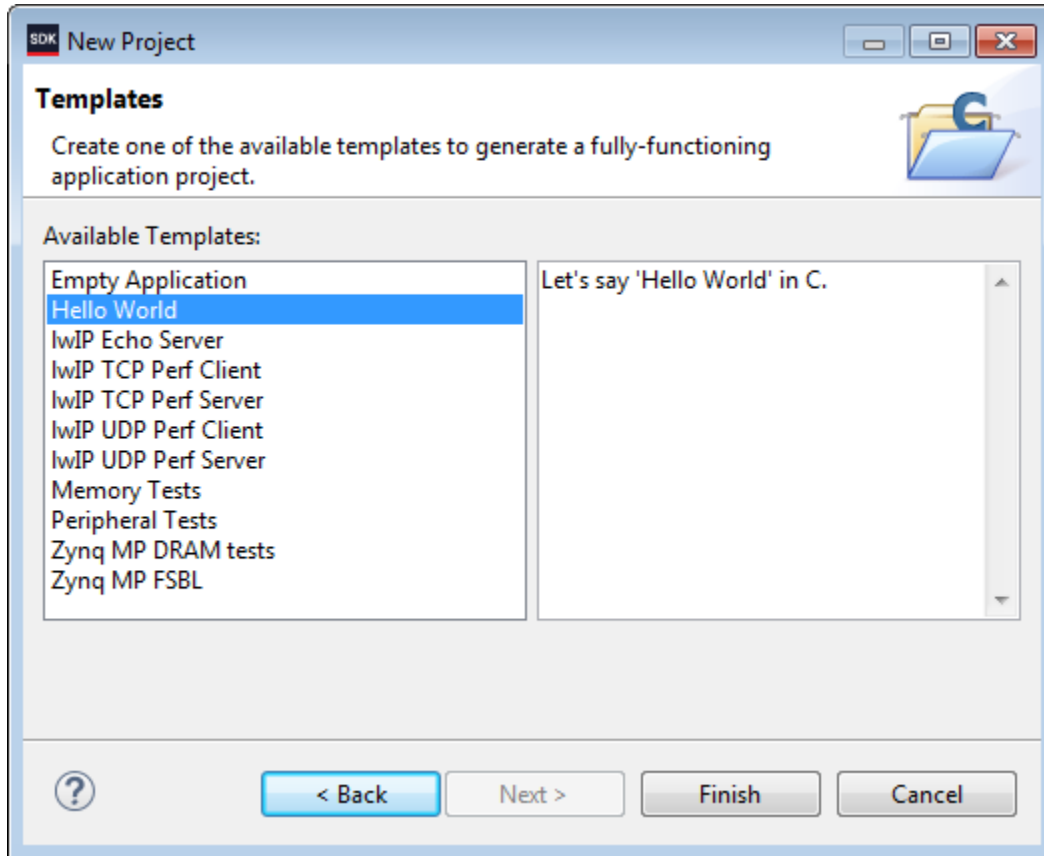


Figure 15 – New Application Project: Hello World

4. Notice that the Hello_Ultra96 application is now visible in *Project Explorer*. By default, SDK will build the application automatically after it is added.

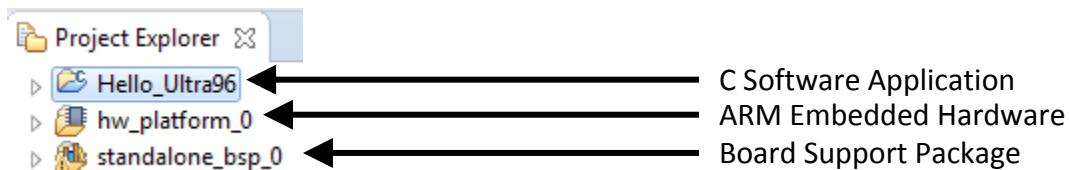
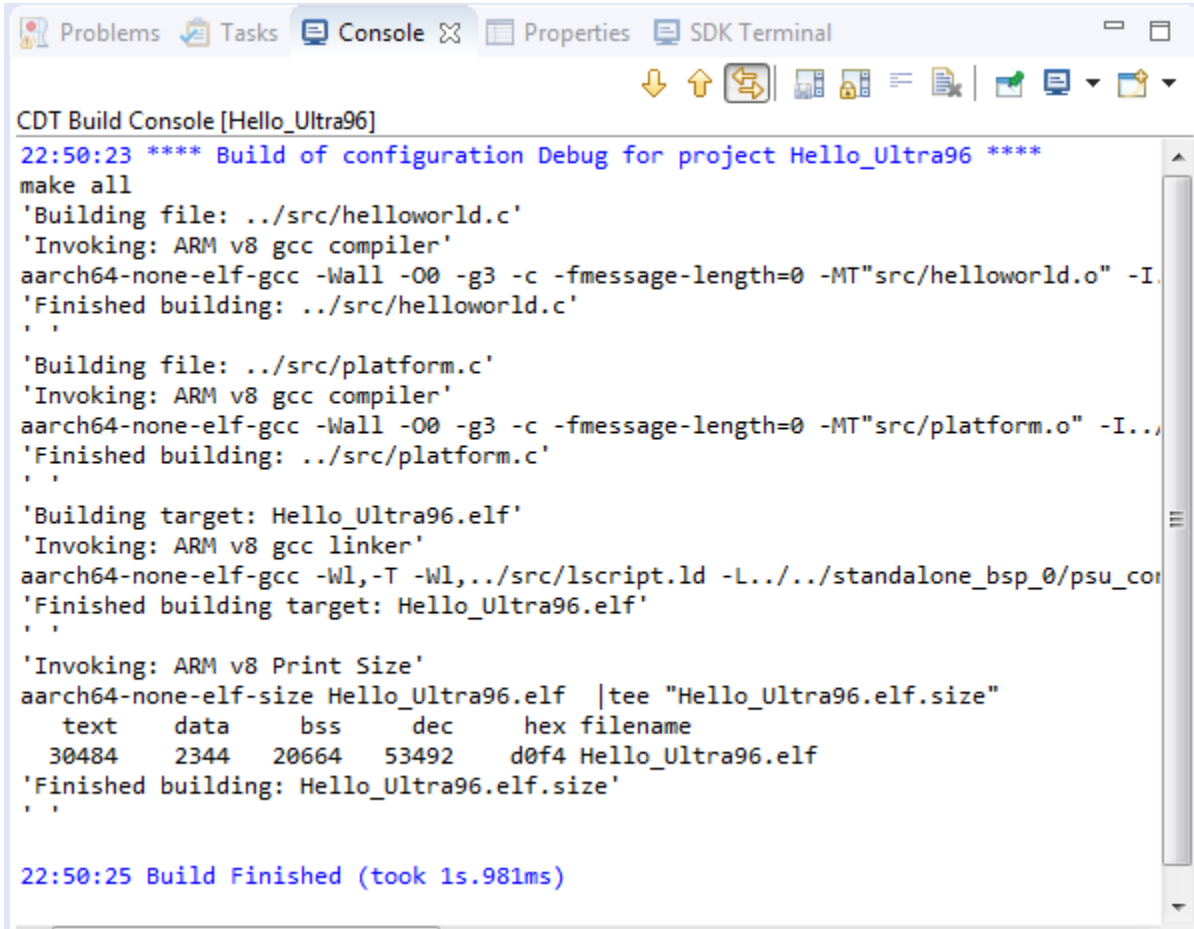


Figure 16 – Project Explorer View with Hello World C Application Added



```
CDT Build Console [Hello_Ultra96]
22:50:23 **** Build of configuration Debug for project Hello_Ultra96 ****
make all
'Building file: ../src/helloworld.c'
'Invoking: ARM v8 gcc compiler'
aarch64-none-elf-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/helloworld.o" -I.
'Finished building: ../src/helloworld.c'
'
'
'Building file: ../src/platform.c'
'Invoking: ARM v8 gcc compiler'
aarch64-none-elf-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/platform.o" -I.
'Finished building: ../src/platform.c'
'
'
'Building target: Hello_Ultra96.elf'
'Invoking: ARM v8 gcc linker'
aarch64-none-elf-gcc -Wl,-T -Wl,../src/lscript.ld -L../standalone_bsp_0/psu_cor
'Finished building target: Hello_Ultra96.elf'
'
'
'Invoking: ARM v8 Print Size'
aarch64-none-elf-size Hello_Ultra96.elf |tee "Hello_Ultra96.elf.size"
  text    data    bss     dec     hex filename
  30484    2344    20664   53492   d0f4 Hello_Ultra96.elf
'Finished building: Hello_Ultra96.elf.size'
'
'
22:50:25 Build Finished (took 1s.981ms)
```

Figure 17 – Hello World Application Automatically Built

Experiment 4: Run on Hardware

1. Connect a USB-UART dongle to the 3-pin J6 and a USB-JTAG dongle to the 7-pin J2 on the Ultra96.

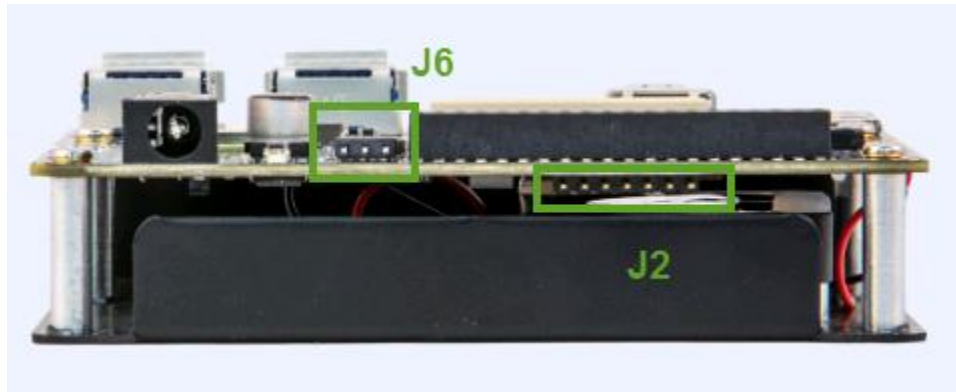


Figure 18 – Locations of UART (J6) and JTAG (J2)

2. The Avnet Ultra96 USB-to-JTAG/UART Pod (available September 2018) is a very handy accessory that allows you to connect both J6 and J2 simultaneously with a single USB port.

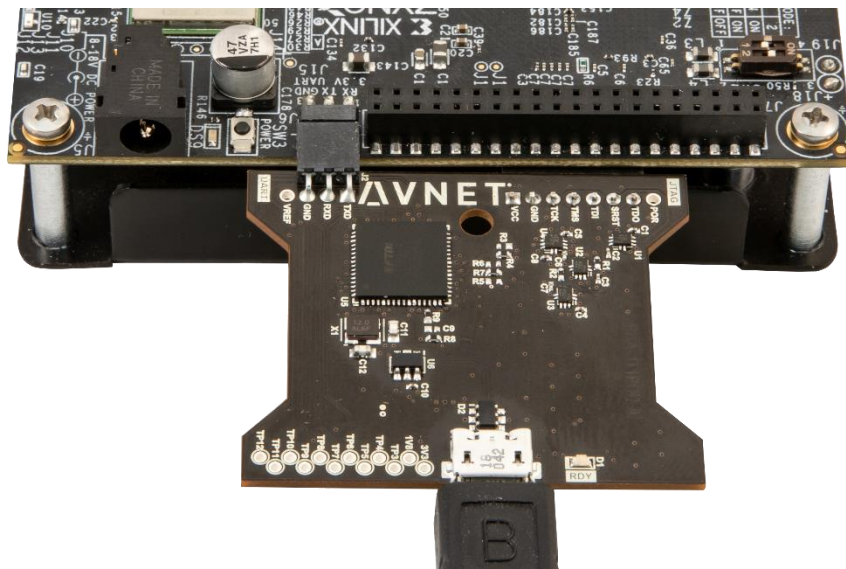


Figure 19 – USB-to-JTAG/UART Pod Attached to Ultra96

3. Make sure all drivers are installed for your JTAG and UART dongles. The drivers for the Avnet USB-to-JTAG/UART Pod will be automatically installed the first time you plug it in.
4. Set the Ultra96 boot mode switch SW2 to JTAG mode ('J' for JTAG) as shown below.

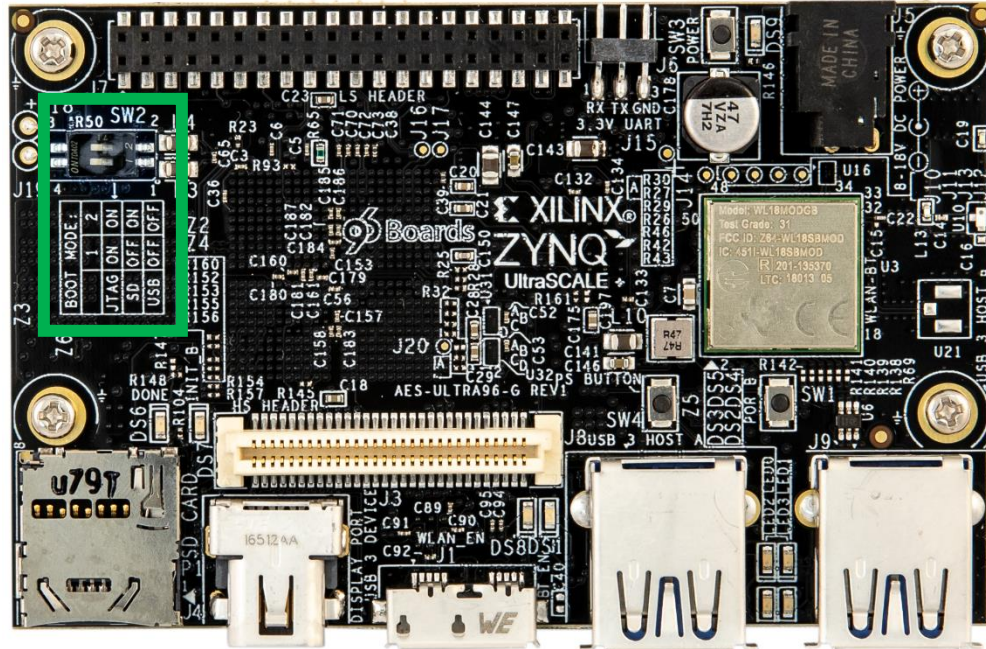


Figure 20 – Ultra96 Boot Mode Switch SW2

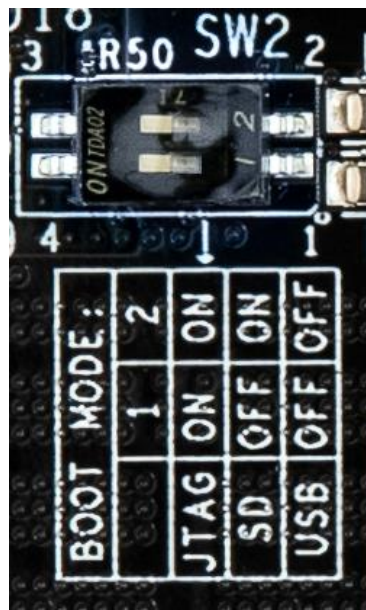


Figure 21 – Ultra96 Boot Mode Switch Set to ON,ON for JTAG

5. Plug in the 96Boards Power Supply to the barrel jack J5 on Ultra96.
6. Press and release the SW3 Power On button. The Green Power On LED (DS9) and the 4 Green User LEDs (between the two Type A USB ports) should illuminate.

7. Use Device Manager to determine the COM port for the USB Serial Port. In Windows 7, click **Start → Control Panel**, and then click **Device Manager**. Click **Yes** to confirm.
8. Expand **Ports**. Note the COM port number for the USB Serial Port device. This example shows COM40.

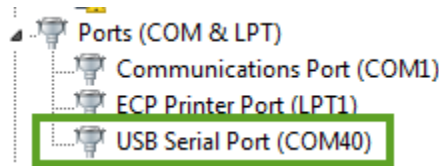


Figure 19 – Find the COM port number for the USB Serial Port device

9. Open your favorite serial communication utility for the COM port assigned on your system with settings shown below. SDK provides a serial terminal utility. Since you may not have used this terminal before, these instructions will show how to use it.

See the SDK Terminal tab in the center bottom window.

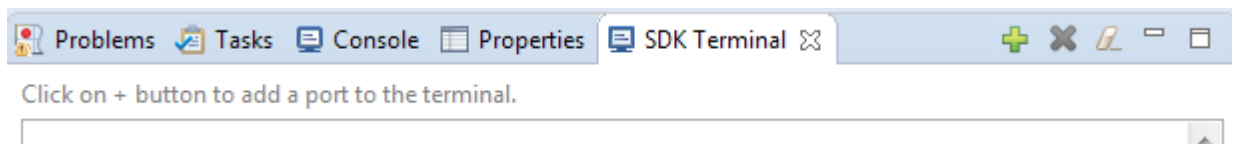



Figure 20 – Terminal Window Header Bar

10. Click the  button to open the Terminal Settings dialog box.

11. Change the settings as shown below. Click **OK**.

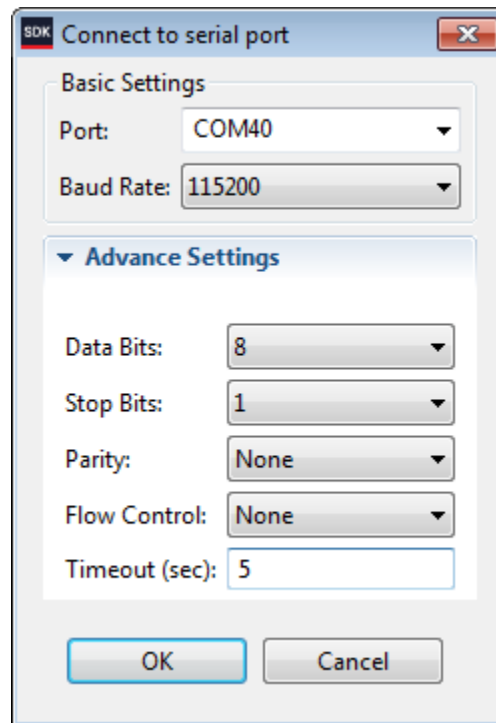


Figure 21 – Terminal Settings Dialog Box

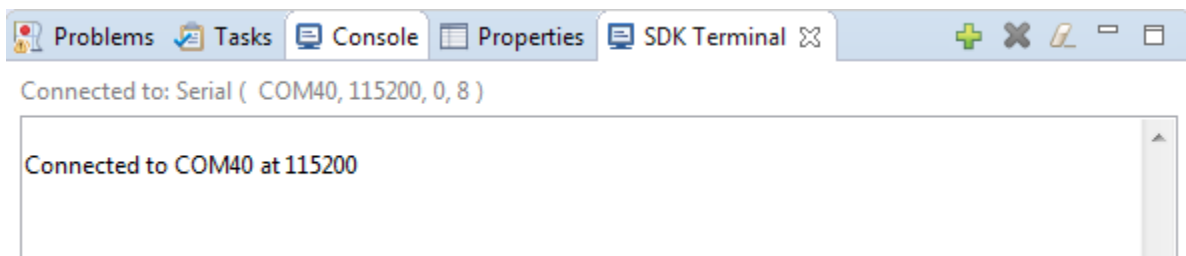



Figure 22 – COM Connection Confirmed

12. Program the PL first by clicking the  icon or selecting **Xilinx Tools** → **Program FPGA**. The default options are acceptable. Click **Program**. When complete, the Green DS6 DONE LED should light.

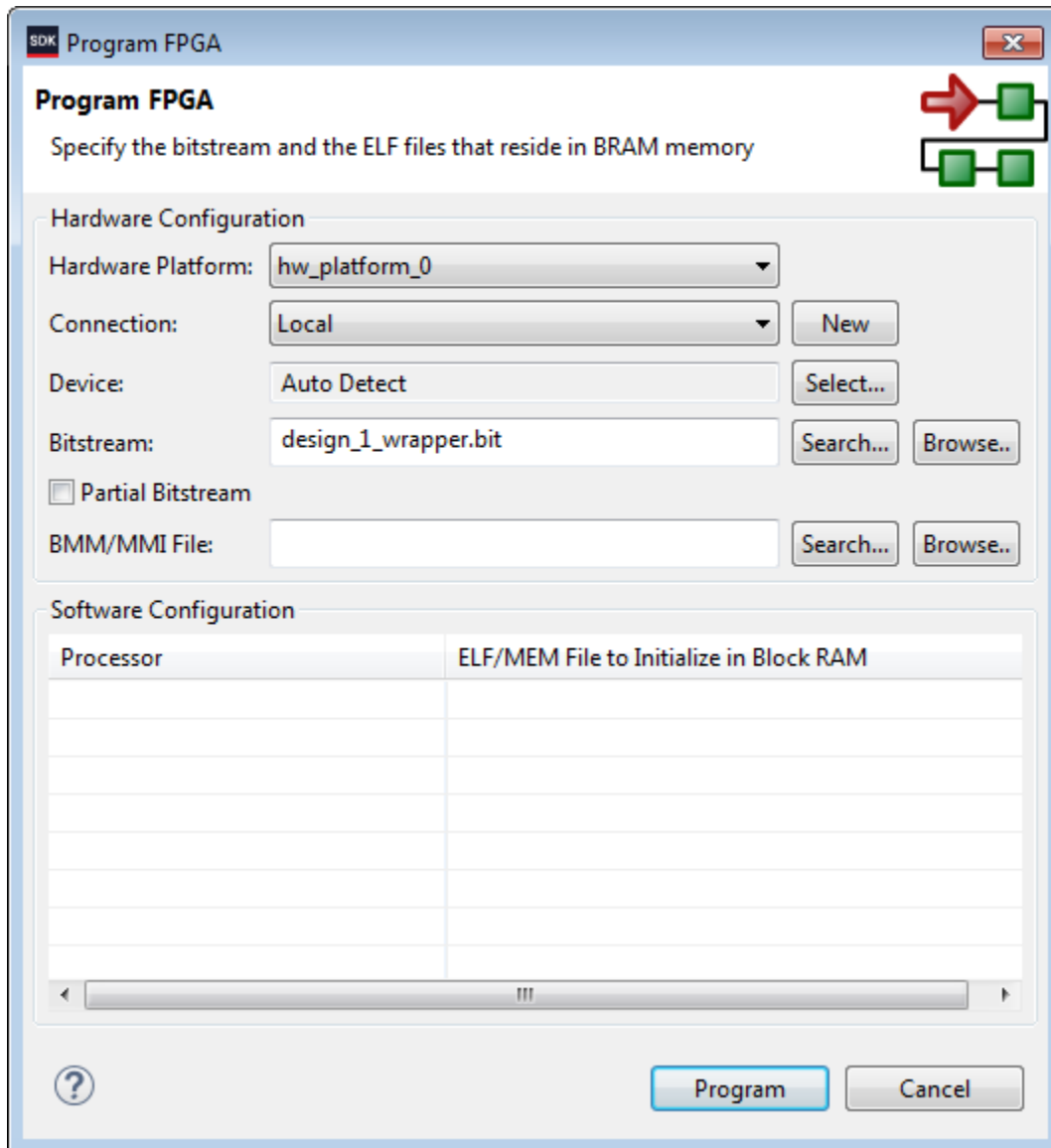


Figure 22 – Program FPGA

13. Right-click on the Hello_Ultra96 application and select **Run As → 1 Launch on Hardware (System Debugger)** (System Debugger).

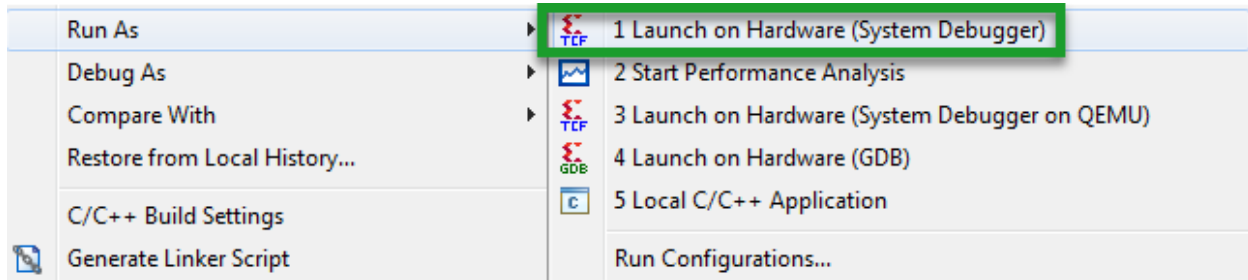


Figure 23 – Launch on Hardware (System Debugger)

14. The tools will now initialize the processor, download the Hello_Ultra96.elf to RAM, and then run Hello_Ultra96. This takes a few seconds to complete, depending on the USB traffic in your system. You can follow the progress in the lower right corner of SDK.

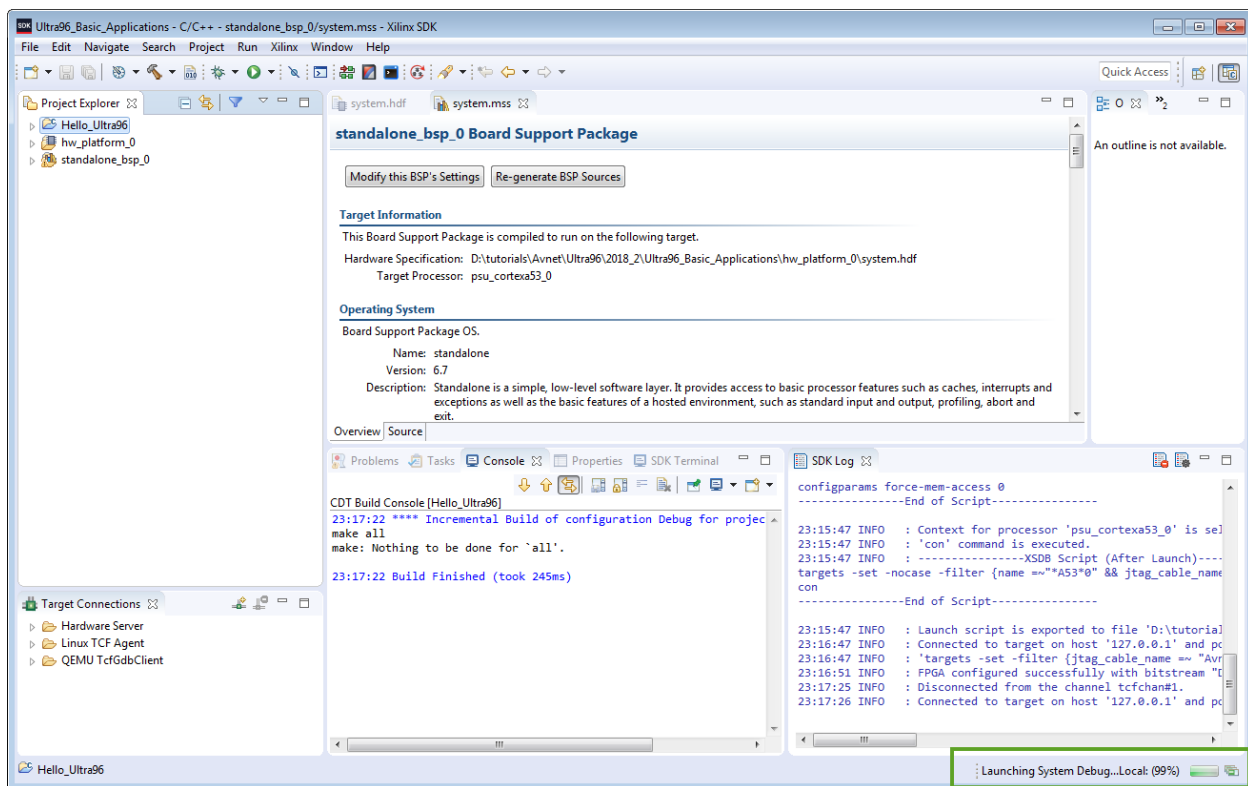


Figure 24 – Launching Hello_Ultra96 Progress

SDK will download the Hello World ELF to the RAM, and the ARM cpu0 begins executing the code. The application standard output is displayed in the SDK Terminal. If SDK automatically switches to the *Console* tab, click on the *SDK Terminal* tab to see the output.

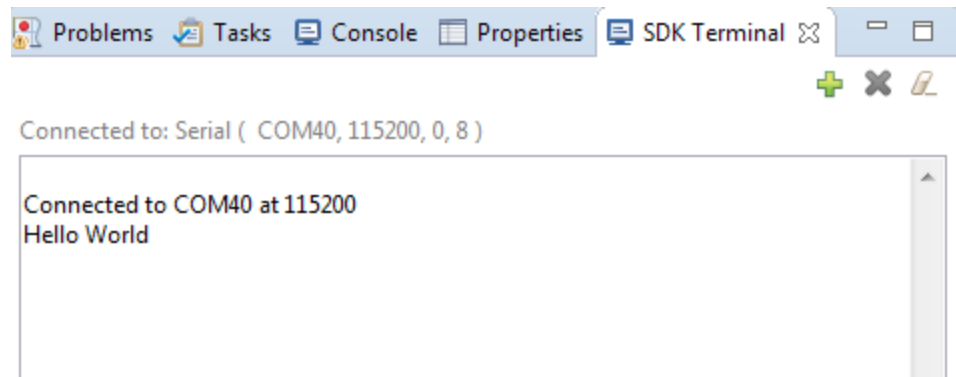



Figure 25 – Hello Ultra96 Complete

You have now booted ZU+ hardware on Ultra96! The Terminal can be disconnected by clicking the  button.

Revision History

Date	Version	Revision
29 Jun 2018	2018_2.01	Initial Avnet release for Vivado 2018.2