MiniZed: Hello World

Overview

Once a Zynq 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 Zynq Hardware Platform into SDK
- Create a BSP
- Add a new application based on a Xilinx-provided template in SDK
- Run the application on the MiniZed hardware

Experiment Setup

Software

The software used to test this reference design is:

- Windows-7 64-bit
- Xilinx SDK 2018.1
- FTDI FT2232H device driver

Hardware

The hardware setup used to test this reference design includes:

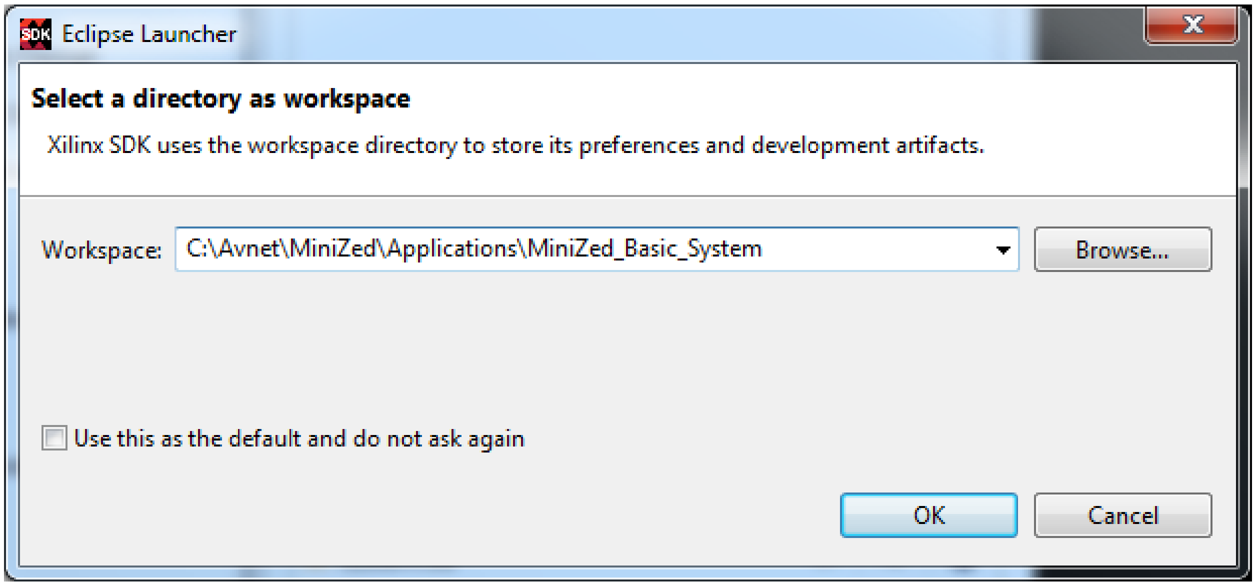
- Win-7 PC with the following recommended memory[1]:
    - 1.6 GB RAM available for the Xilinx tools to complete a XC7Z010 design
    - 2.3 GB RAM available for the Xilinx tools to complete a XC7Z015 design
    - 1.9 GB RAM available for the Xilinx tools to complete a XC7Z020 design
    - 2.7 GB RAM available for the Xilinx tools to complete a XC7Z030 design
- MiniZed
- USB cable (Type A to Micro-USB Type B)

[1] Refer to www.xilinx.com/design-tools/vivado/memory.htm

Experiment 1: Import the Hardware Platform

The first requirement within SDK is to import a hardware platform.

1. Launch SDK by selecting **Start → All Programs → Xilinx Design Tools → SDK 2018.1 → Xilinx SDK 2018.1**.
2. Select a workspace. Click **OK**.



Eclipse Launcher

Select a directory as workspace

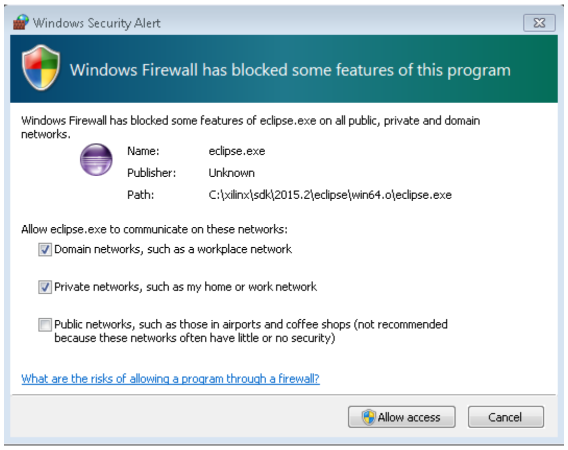
Xilinx SDK uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Avnet\MiniZed\Applications\MiniZed_Basic_System

☐ Use this as the default and do not ask again

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**.



Windows Security Alert

Windows Firewall has blocked some features of this program

Windows Firewall has blocked some features of eclipse.exe on all public, private and domain networks.

Name:	eclipse.exe
Publisher:	Unknown
Path:	C:\xilinx\sdk\2015.2\eclipse\win64.o\eclipse.exe

Allow eclipse.exe to communicate on these networks:

☑ Domain networks, such as a workplace network

☑ Private networks, such as my home or work network

☐ Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing a program through a firewall?

Figure 2 – Windows Security Alert from SDK

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

Now we will import the Zynq 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 >**.

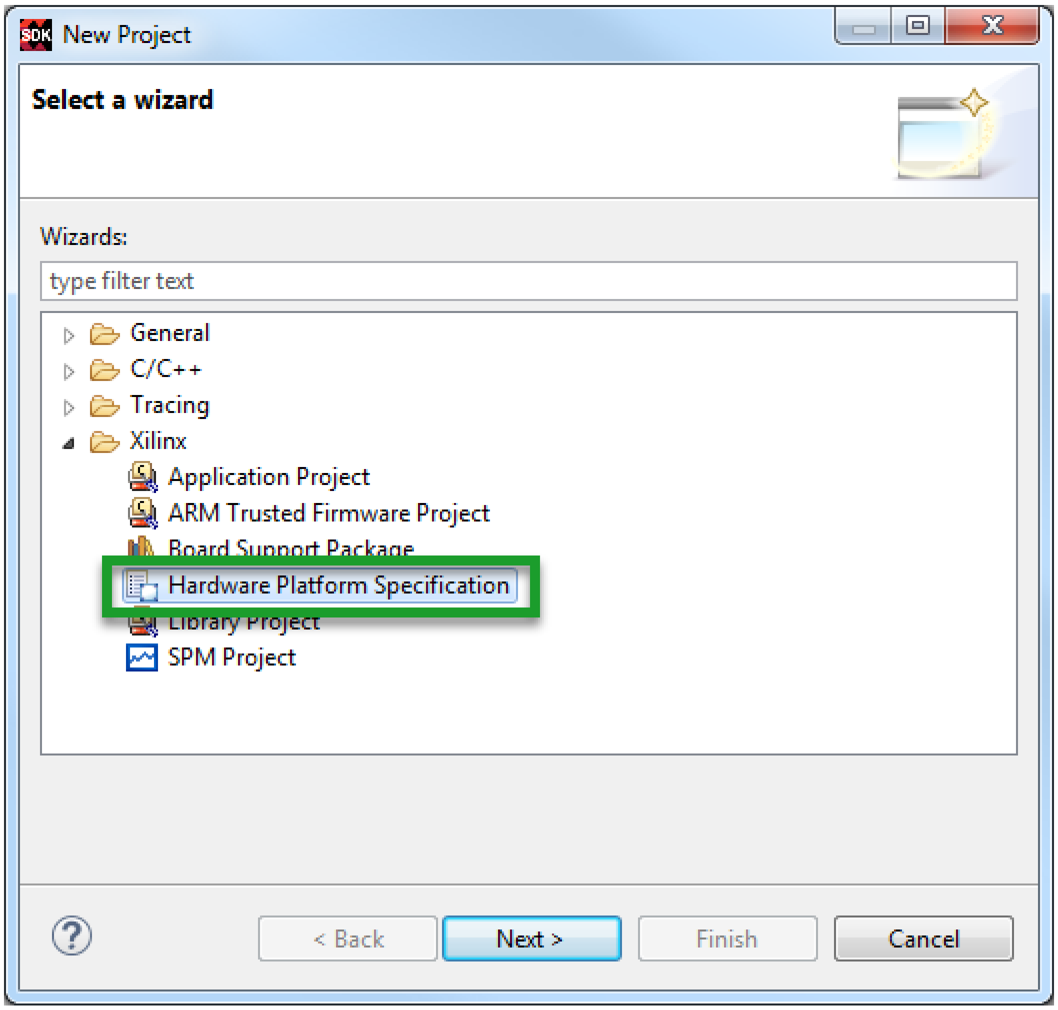


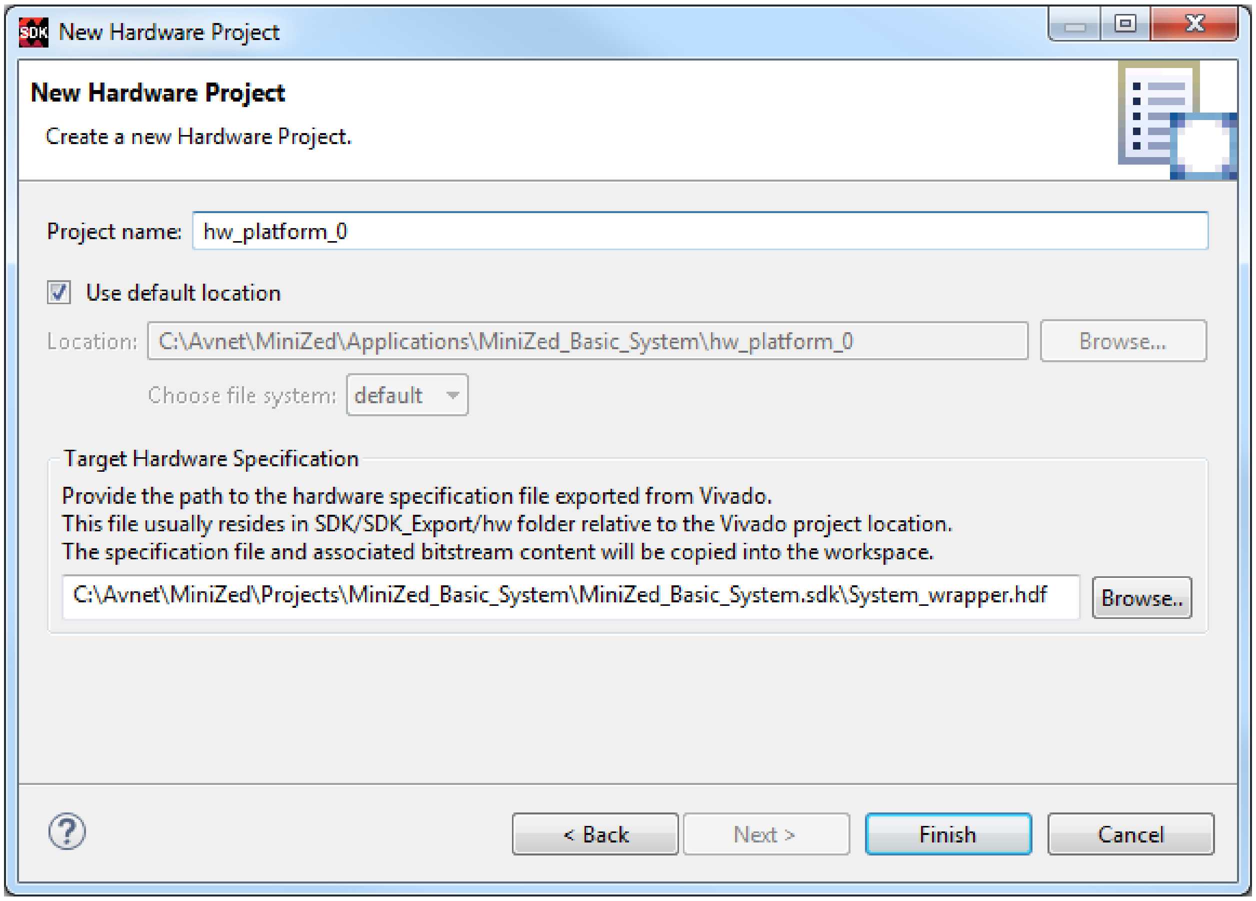
Figure 3 – Creating a New Hardware Platform

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\MiniZed\Projects\MiniZed_Basic_System\MiniZed_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**.



New Hardware Project

Create a new Hardware Project.

Project name: hw_platform_0

☑ Use default location

Location: C:\Avnet\MiniZed\Applications\MiniZed_Basic_System\hw_platform_0

Choose file system: default

Target Hardware Specification

Provide the path to the hardware specification file exported from Vivado.
This file usually resides in SDK/SDK_Export/hw folder relative to the Vivado project location.
The specification file and associated bitstream content will be copied into the workspace.

C:\Avnet\MiniZed\Projects\MiniZed_Basic_System\MiniZed_Basic_System.sdk\System_wrapper.hdf

Figure 4 – Import Hardware Platform from Vivado

11. Notice the PS7 Zynq hardware platform is now visible in the *Project Explorer*.

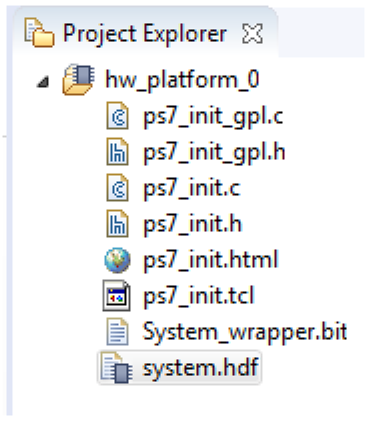
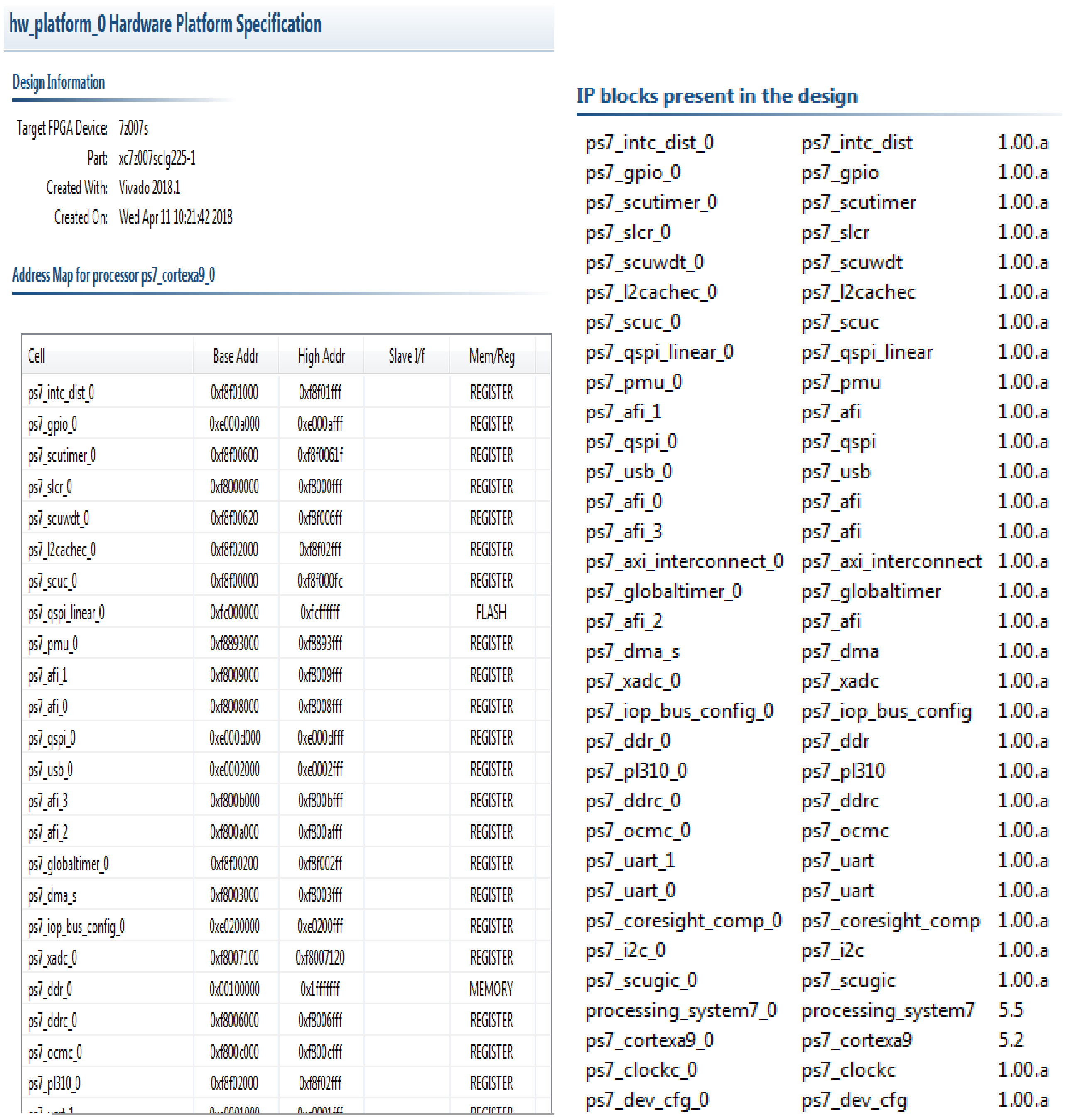


Figure 5 – Hardware Platform Imported and Ready for Use

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



hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device:	7z007s
Part:	xc7z007sclg225-1
Created With:	Vivado 2018.1
Created On:	Wed Apr 11 10:21:42 2018

Address Map for processor ps7_cortexa9_0

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
ps7_qspi_linear_0	0xfc000000	0xfcffffff		FLASH
ps7_pmu_0	0xf8893000	0xf8893fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_qspi_0	0xe000d000	0xe000dfff		REGISTER
ps7_usb_0	0xe0002000	0xe0002fff		REGISTER
ps7_afi_3	0xf800b000	0xf800bfff		REGISTER
ps7_afi_2	0xf800a000	0xf800afff		REGISTER
ps7_globaltimer_0	0xf8f00200	0xf8f002ff		REGISTER
ps7_dma_s	0xf8003000	0xf8003fff		REGISTER
ps7_iop_bus_config_0	0xe0200000	0xe0200fff		REGISTER
ps7_xadc_0	0xf8007100	0xf8007120		REGISTER
ps7_ddr_0	0x00100000	0x1fffffff		MEMORY
ps7_ddrc_0	0xf8006000	0xf8006fff		REGISTER
ps7_ocmc_0	0xf800c000	0xf800cfff		REGISTER
ps7_pl310_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_uart_1	0xe0001000	0xe0001fff		REGISTER

IP blocks present in the design

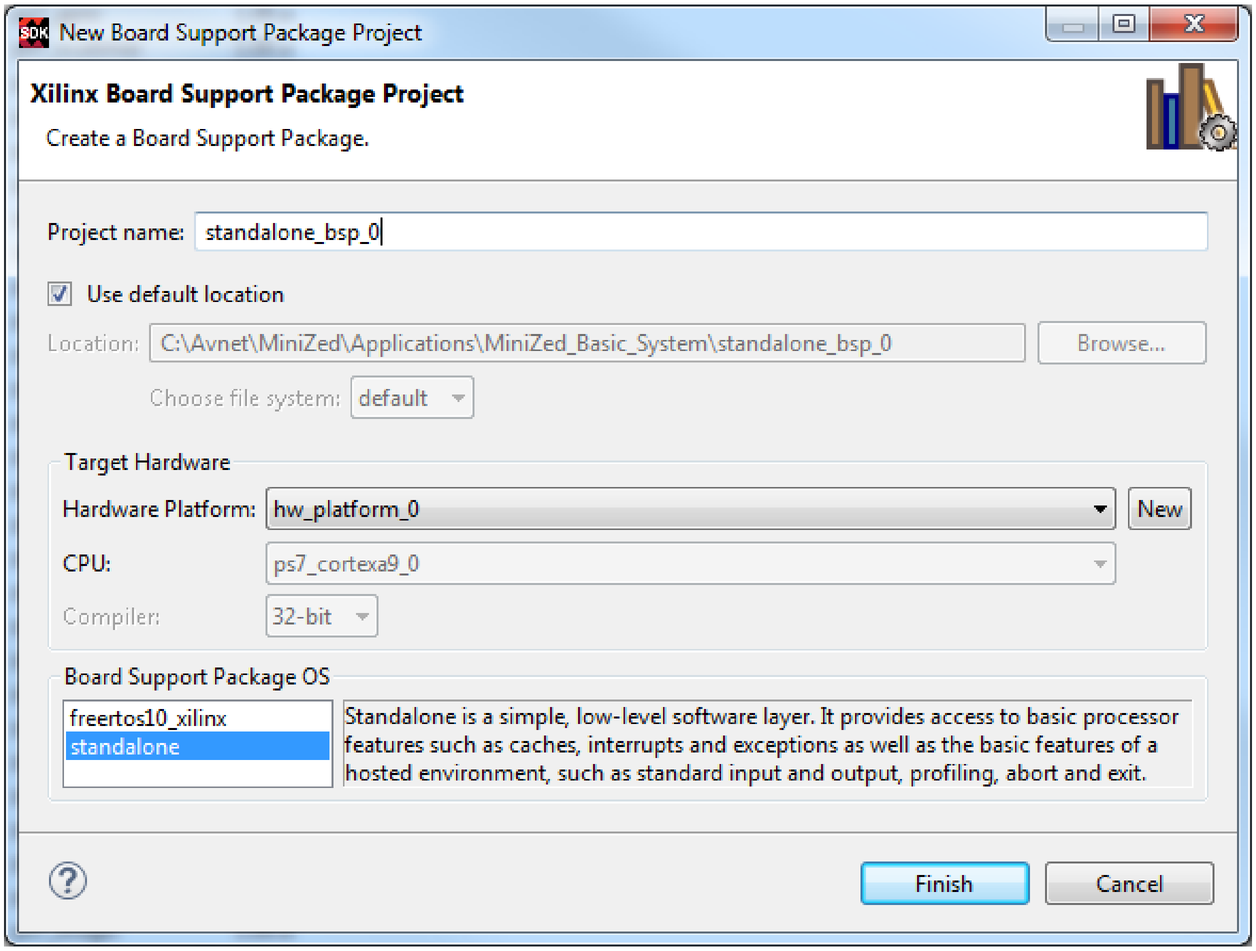
ps7_intc_dist_0	ps7_intc_dist	1.00.a
ps7_gpio_0	ps7_gpio	1.00.a
ps7_scutimer_0	ps7_scutimer	1.00.a
ps7_slcr_0	ps7_slcr	1.00.a
ps7_scuwdt_0	ps7_scuwdt	1.00.a
ps7_l2cachec_0	ps7_l2cachec	1.00.a
ps7_scuc_0	ps7_scuc	1.00.a
ps7_qspi_linear_0	ps7_qspi_linear	1.00.a
ps7_pmu_0	ps7_pmu	1.00.a
ps7_afi_1	ps7_afi	1.00.a
ps7_qspi_0	ps7_qspi	1.00.a
ps7_usb_0	ps7_usb	1.00.a
ps7_afi_0	ps7_afi	1.00.a
ps7_afi_3	ps7_afi	1.00.a
ps7_axi_interconnect_0	ps7_axi_interconnect	1.00.a
ps7_globaltimer_0	ps7_globaltimer	1.00.a
ps7_afi_2	ps7_afi	1.00.a
ps7_dma_s	ps7_dma	1.00.a
ps7_xadc_0	ps7_xadc	1.00.a
ps7_iop_bus_config_0	ps7_iop_bus_config	1.00.a
ps7_ddr_0	ps7_ddr	1.00.a
ps7_pl310_0	ps7_pl310	1.00.a
ps7_ddrc_0	ps7_ddrc	1.00.a
ps7_ocmc_0	ps7_ocmc	1.00.a
ps7_uart_1	ps7_uart	1.00.a
ps7_uart_0	ps7_uart	1.00.a
ps7_coresight_comp_0	ps7_coresight_comp	1.00.a
ps7_i2c_0	ps7_i2c	1.00.a
ps7_scugic_0	ps7_scugic	1.00.a
processing_system7_0	processing_system7	5.5
ps7_cortexa9_0	ps7_cortexa9	5.2
ps7_clockc_0	ps7_clockc	1.00.a
ps7_dev_cfg_0	ps7_dev_cfg	1.00.a

Figure 6 – system.hdf Report on Hardware Specification

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**.



New Board Support Package Project

Xilinx Board Support Package Project

Create a Board Support Package.

Project name: standalone_bsp_0

☑ Use default location

Location: C:\Avnet\MiniZed\Applications\MiniZed_Basic_System\standalone_bsp_0

Choose file system: default

Target Hardware

Hardware Platform: hw_platform_0

CPU: ps7_cortexa9_0

Compiler: 32-bit

Board Support Package OS

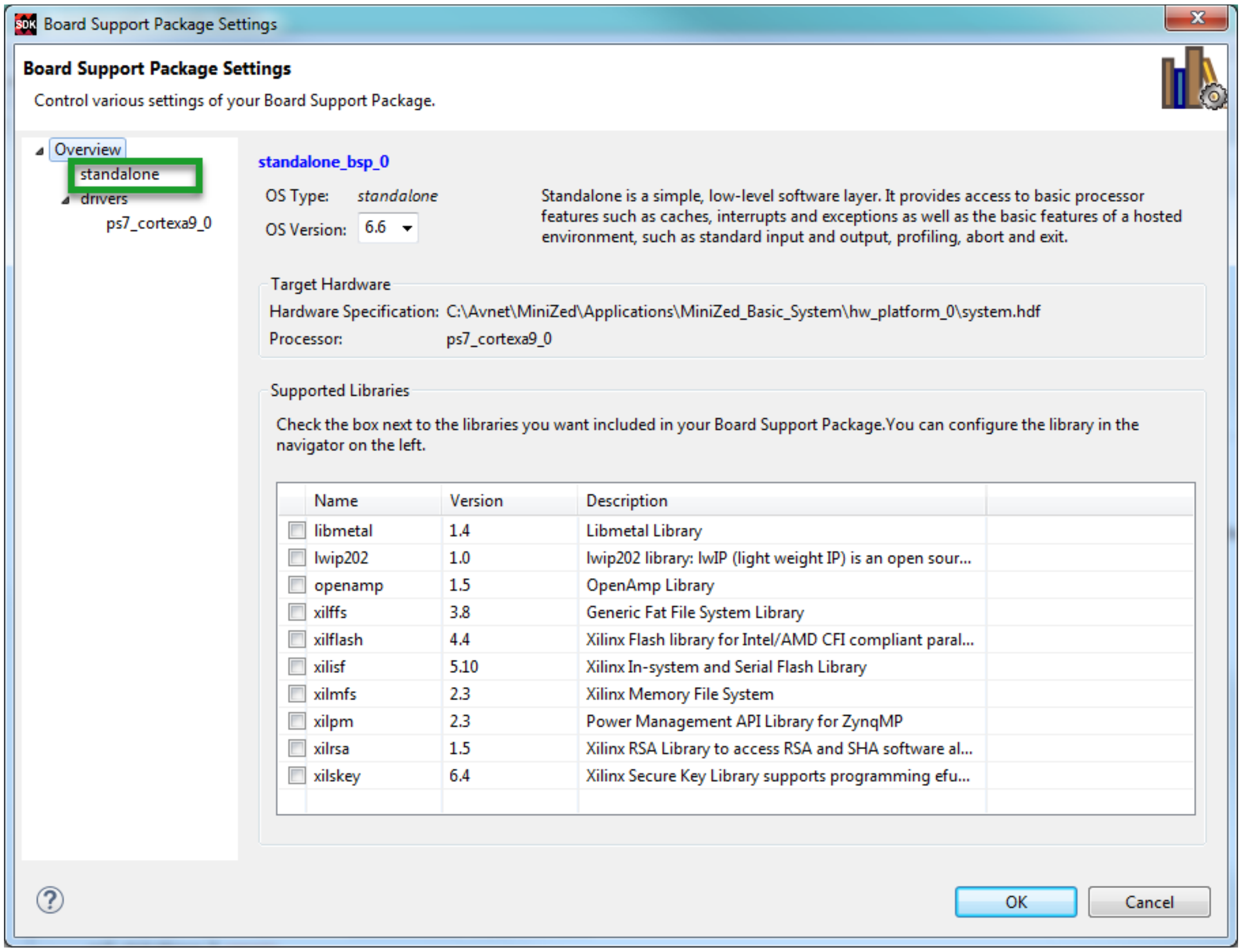
freertos10_xilinx

standalone

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Figure 7 – Standalone BSP

3. In the *Board Support Package Settings*, select standalone.



Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

- Overview
  - standalone
  - drivers
    - ps7_cortexa9_0

standalone_bsp_0

OS Type: *standalone*

OS Version: 6.6

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Target Hardware

Hardware Specification: C:\Avnet\MiniZed\Applications\MiniZed_Basic_System\hw_platform_0\system.hdf

Processor: ps7_cortexa9_0

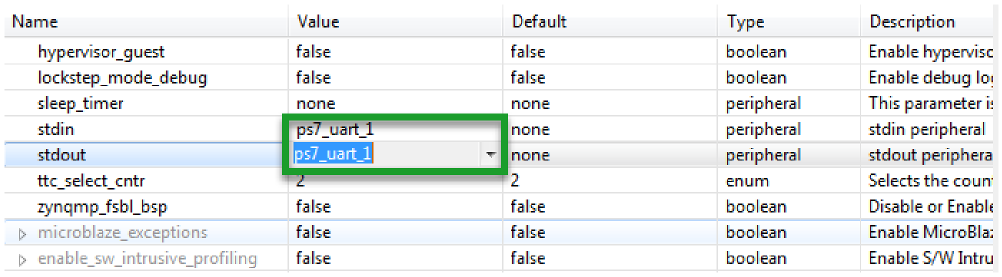
Supported Libraries

Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.

	Name	Version	Description
☐	libmetal	1.4	Libmetal Library
☐	lwip202	1.0	lwip202 library: lwIP (light weight IP) is an open sour...
☐	openamp	1.5	OpenAmp Library
☐	xilffs	3.8	Generic Fat File System Library
☐	xilflash	4.4	Xilinx Flash library for Intel/AMD CFI compliant paral...
☐	xilisf	5.10	Xilinx In-system and Serial Flash Library
☐	xilmfs	2.3	Xilinx Memory File System
☐	xilpm	2.3	Power Management API Library for ZynqMP
☐	xilrsa	1.5	Xilinx RSA Library to access RSA and SHA software al...
☐	xilskey	6.4	Xilinx Secure Key Library supports programming efu...

Figure 8 – Board Support Package Settings

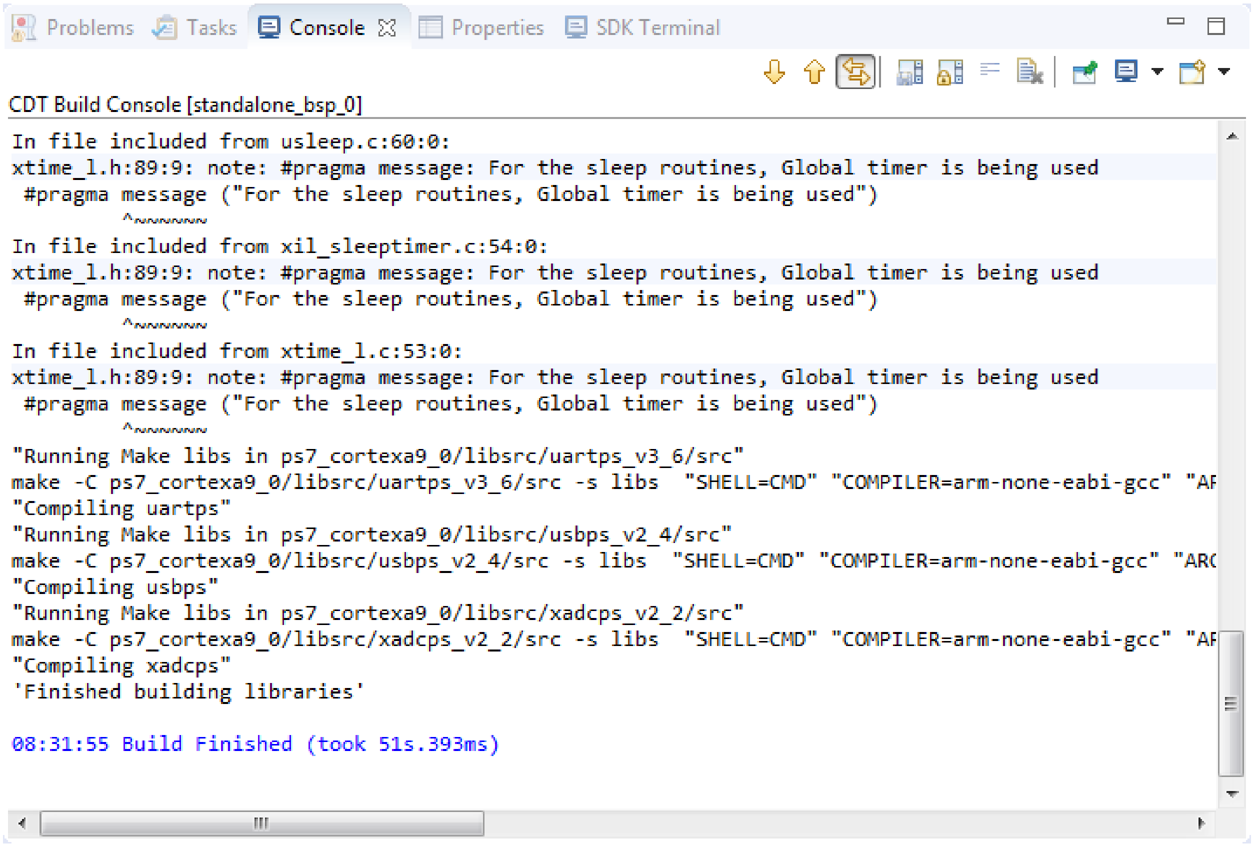
4. Change the stdin and stdout Value to ps7_uart_1. This is done to align with the MiniZed's UART Serial connection.



Name	Value	Default	Type	Description
hypervisor_guest	false	false	boolean	Enable hypervisc
lockstep_mode_debug	false	false	boolean	Enable debug lo
sleep_timer	none	none	peripheral	This parameter i:
stdin	ps7_uart_1	none	peripheral	stdin peripheral
stdout	ps7_uart_1	none	peripheral	stdout periphera
ttc_select_cntr	2	2	enum	Selects the coun
zynqmp_fsbl_bsp	false	false	boolean	Disable or Enable
microblaze_exceptions	false	false	boolean	Enable MicroBla:
enable_sw_intrusive_profiling	false	false	boolean	Enable S/W Intru

Figure 9 -- Modify BSP

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.



```
In file included from usleep.c:60:0:
xtime_l.h:89:9: note: #pragma message: For the sleep routines, Global timer is being used
 #pragma message ("For the sleep routines, Global timer is being used")
         ^~~~~~~
In file included from xil_sleeptimer.c:54:0:
xtime_l.h:89:9: note: #pragma message: For the sleep routines, Global timer is being used
 #pragma message ("For the sleep routines, Global timer is being used")
         ^~~~~~~
In file included from xtime_l.c:53:0:
xtime_l.h:89:9: note: #pragma message: For the sleep routines, Global timer is being used
 #pragma message ("For the sleep routines, Global timer is being used")
         ^~~~~~~
"Running Make libs in ps7_cortexa9_0/libsrc/uartps_v3_6/src"
make -C ps7_cortexa9_0/libsrc/uartps_v3_6/src -s libs  "SHELL=CMD" "COMPILER=arm-none-eabi-gcc" "AF
"Compiling uartps"
"Running Make libs in ps7_cortexa9_0/libsrc/usbps_v2_4/src"
make -C ps7_cortexa9_0/libsrc/usbps_v2_4/src -s libs  "SHELL=CMD" "COMPILER=arm-none-eabi-gcc" "ARC
"Compiling usbps"
"Running Make libs in ps7_cortexa9_0/libsrc/xadcps_v2_2/src"
make -C ps7_cortexa9_0/libsrc/xadcps_v2_2/src -s libs  "SHELL=CMD" "COMPILER=arm-none-eabi-gcc" "AF
"Compiling xadcps"
'Finished building libraries'

08:31:55 Build Finished (took 51s.393ms)
```

Figure 10 – BSP Building

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

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

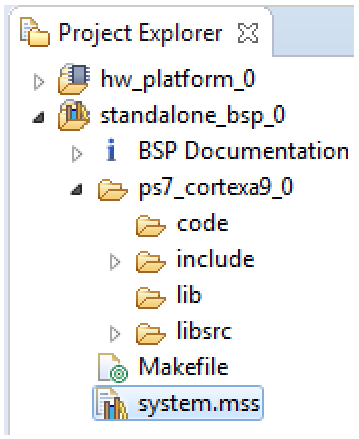


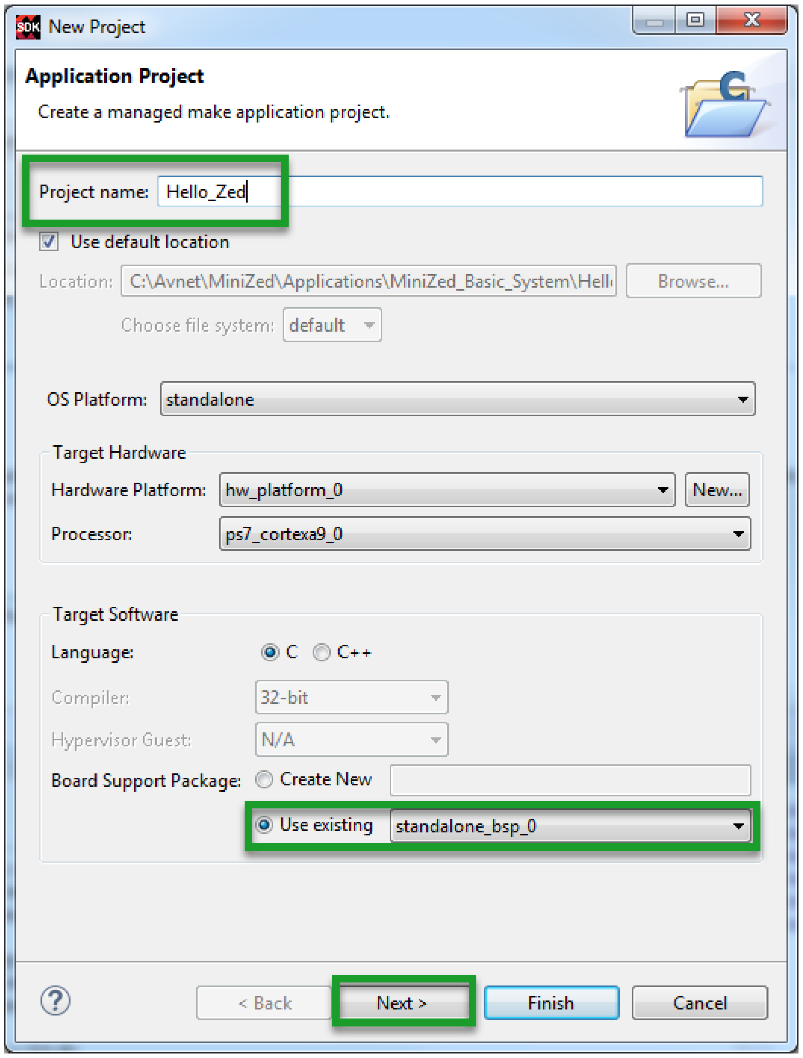
Figure 11 – BSP Added to the Project

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_Zed`. Change the **BSP** to the existing StandAlone BSP. Click **Next >**.



New Project

Application Project

Create a managed make application project.

Project name: Hello_Zed

[✓] Use default location

Location: C:\Avnet\MiniZed\Applications\MiniZed_Basic_System\Hell

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [● C] C++

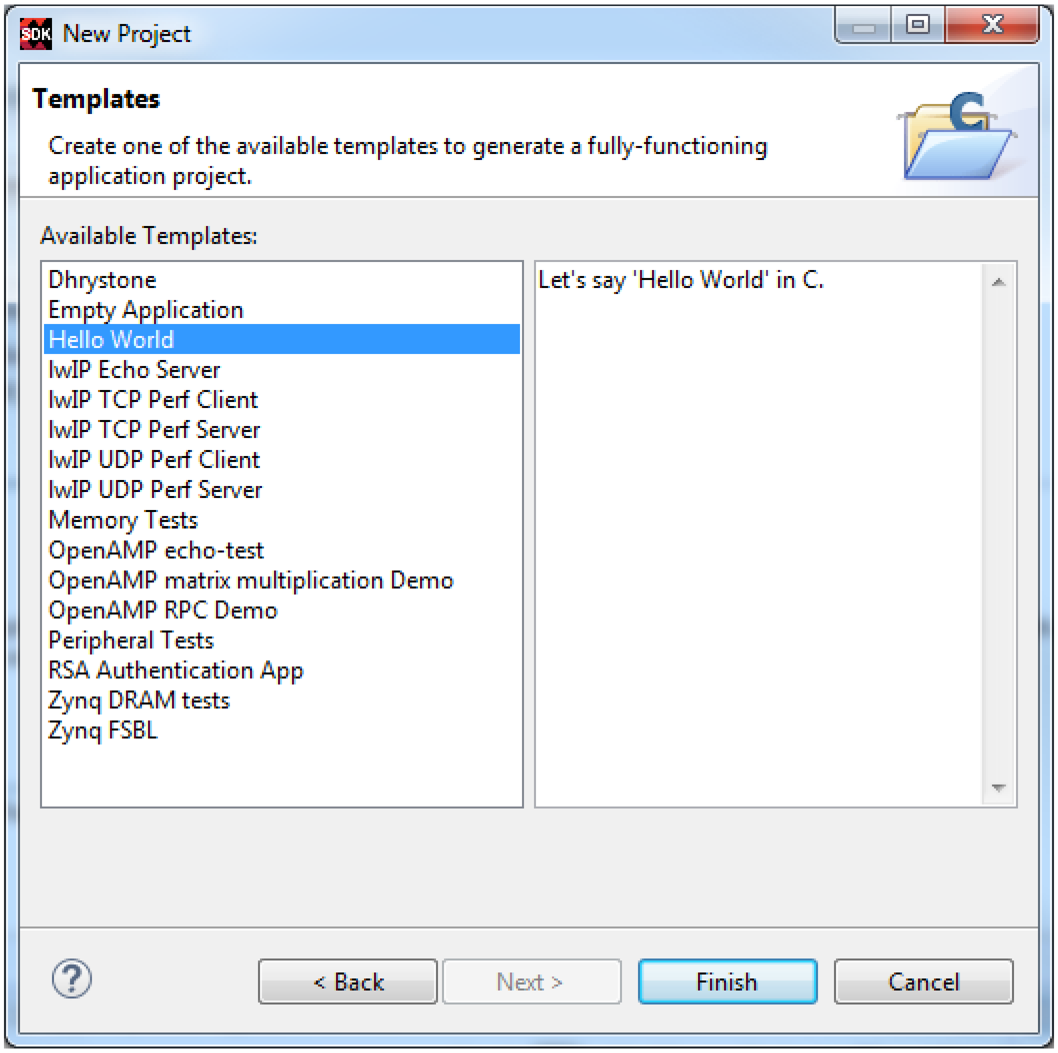
Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: Create New [● Use existing] standalone_bsp_0

Figure 12 - New Application Wizard

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



Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- 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 Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

Let's say 'Hello World' in C.

Figure 13 – New Application Project: Hello World

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

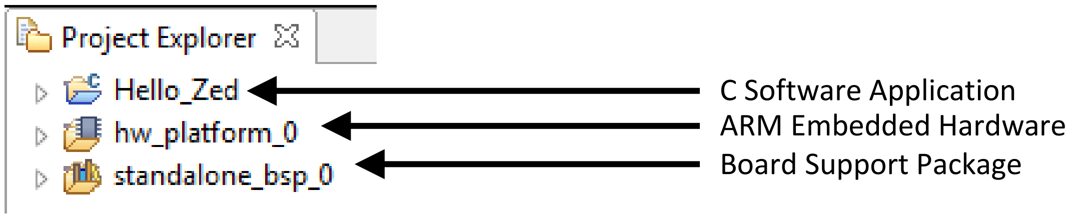
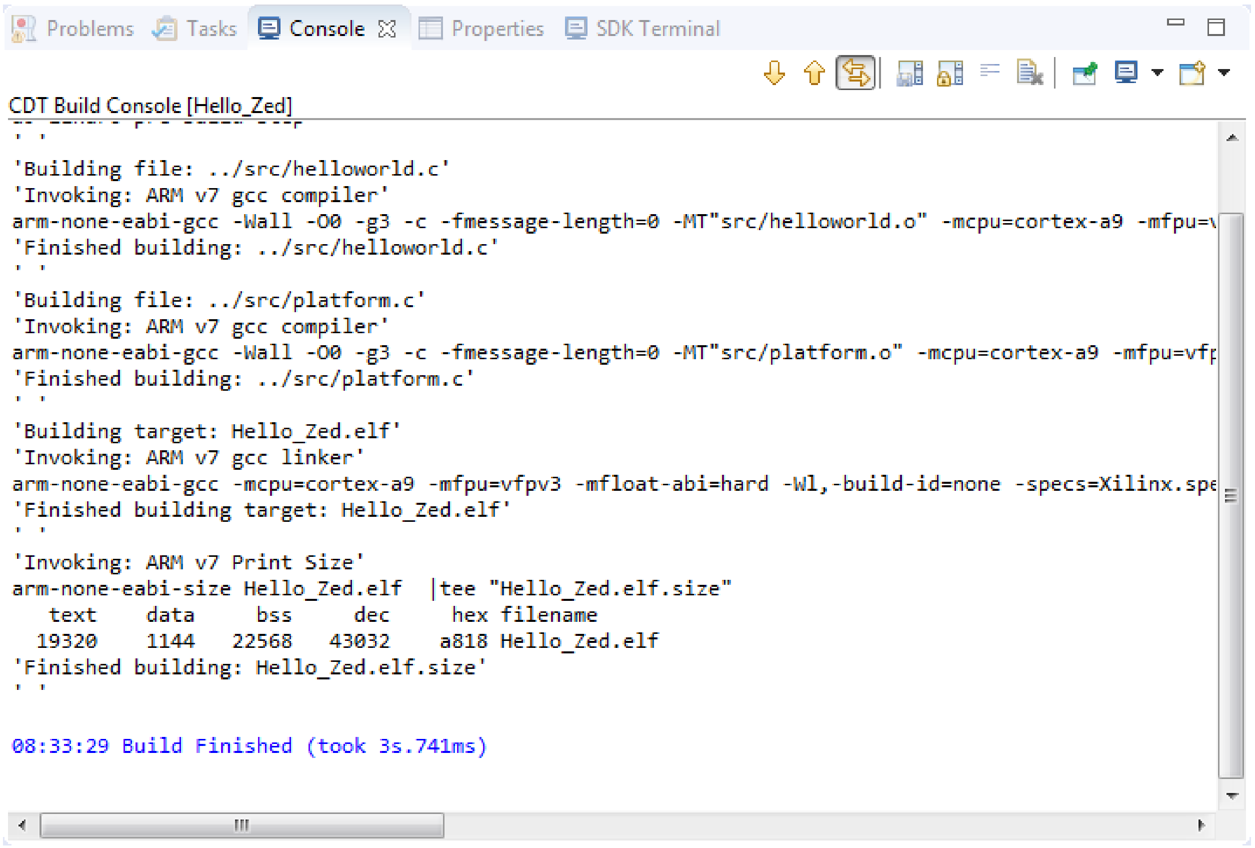


Figure 14 – Project Explorer View with Hello World C Application Added



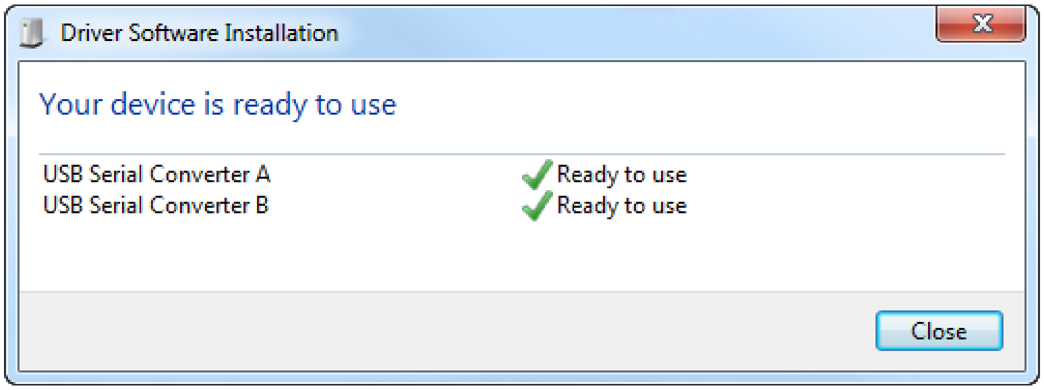
```
' '
'Building file: ../src/helloworld.c'
'Invoking: ARM v7 gcc compiler'
arm-none-eabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/helloworld.o" -mcpu=cortex-a9 -mfpu=v
'Finished building: ../src/helloworld.c'
' '
'Building file: ../src/platform.c'
'Invoking: ARM v7 gcc compiler'
arm-none-eabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/platform.o" -mcpu=cortex-a9 -mfpu=vfp
'Finished building: ../src/platform.c'
' '
'Building target: Hello_Zed.elf'
'Invoking: ARM v7 gcc linker'
arm-none-eabi-gcc -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -Wl,-build-id=none -specs=Xilinx.spe
'Finished building target: Hello_Zed.elf'
' '
'Invoking: ARM v7 Print Size'
arm-none-eabi-size Hello_Zed.elf  |tee "Hello_Zed.elf.size"
   text    data     bss     dec     hex filename
  19320    1144   22568   43032    a818 Hello_Zed.elf
'Finished building: Hello_Zed.elf.size'
' '

08:33:29 Build Finished (took 3s.741ms)
```

Figure 15 – Hello World Application Automatically Built

Experiment 4: Run on Hardware

1. A terminal program is required. Tera Term was used in this example which can be downloaded from the Tera Term project on the SourceForge Japan page: ttssh2.sourceforge.jp Install Tera Term or another terminal program of your choice.

2. Connect the MiniZed USB-JTAG/UART port J2 to your Windows PC. It should automatically install the proper drivers, giving you a confirmation as shown below:



Your device is ready to use

USB Serial Converter A	✓ Ready to use
USB Serial Converter B	✓ Ready to use

Figure 16 – MiniZed USB-JTAG/UART Installed Correctly

3. In the rare circumstance that the drivers are not auto-installed, then you must manually install the driver for the FTDI FT2232H device. Visit the FTDI website and download the appropriate driver for your operating system. http://www.ftdichip.com/Drivers/VCP.htm

4. Make sure the MiniZed is unplugged from the PC. Unzip and install the driver.

5. Reboot your PC then plug in the MiniZed.

6. Set the MiniZed boot mode switch SW1 to JTAG mode ('J' for JTAG) as shown below.

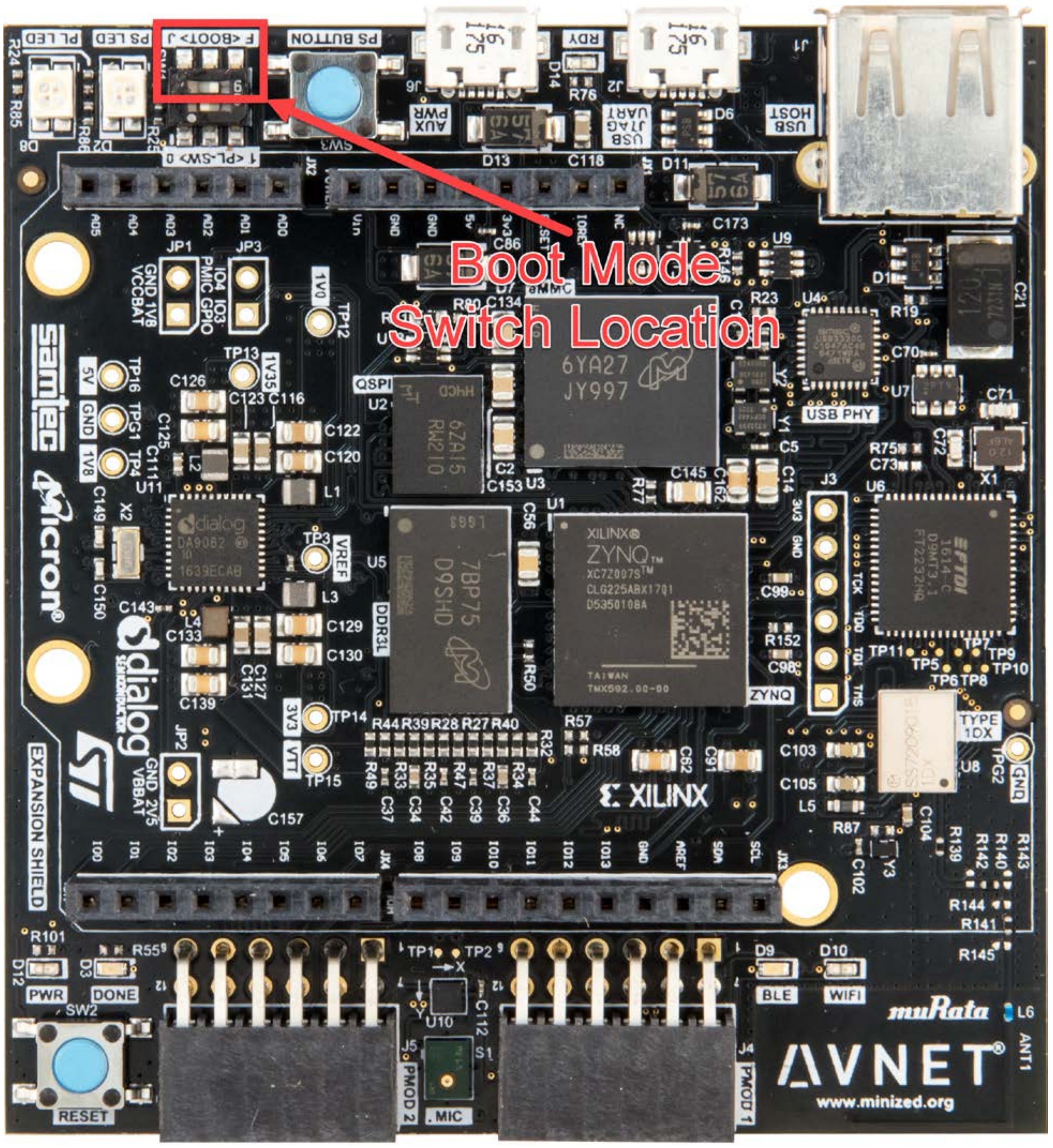


Figure 17 – MiniZed Switch Location

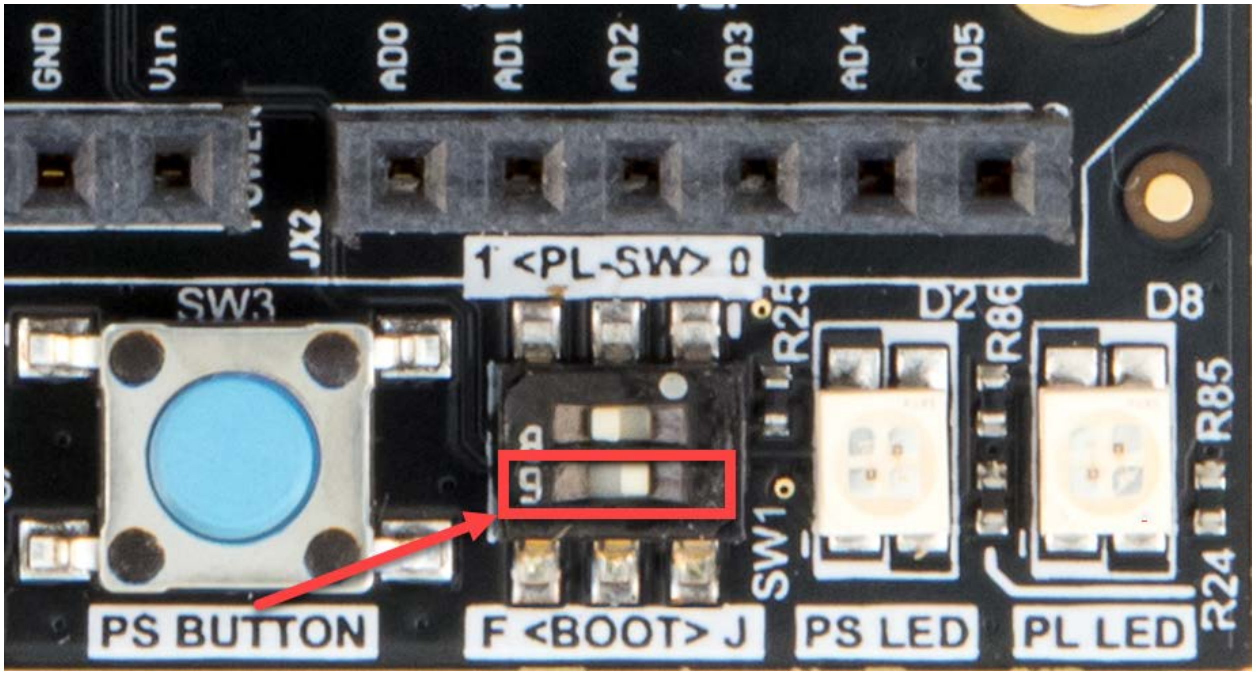


Figure 18 – JTAG Boot Mode

7. Use Device Manager to determine the COM port for the Silicon 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 COM25.

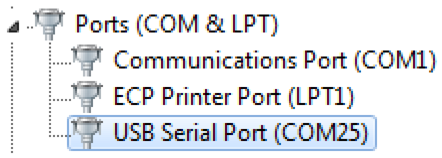
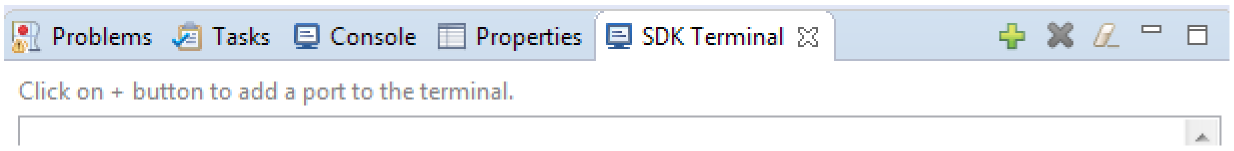

Ports (COM & LPT)
Communications Port (COM1)
ECP Printer Port (LPT1)
USB Serial Port (COM25)

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

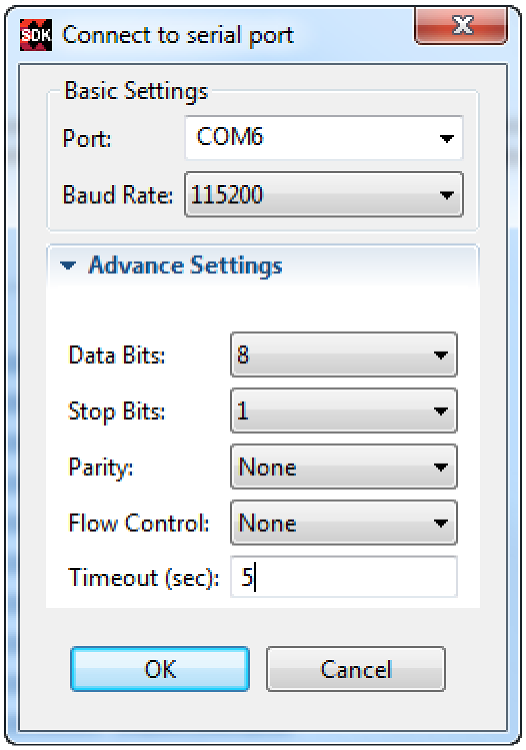
9. Open a serial communication utility for the COM port assigned on your system. SDK provides a serial terminal utility. See the SDK Terminal tab in the center bottom window.


Problems Tasks Console Properties SDK Terminal

Click on + button to add a port to the terminal.

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**.


Connect to serial port

Basic Settings

Port: COM6

Baud Rate: 115200

Advance Settings

Data Bits: 8

Stop Bits: 1

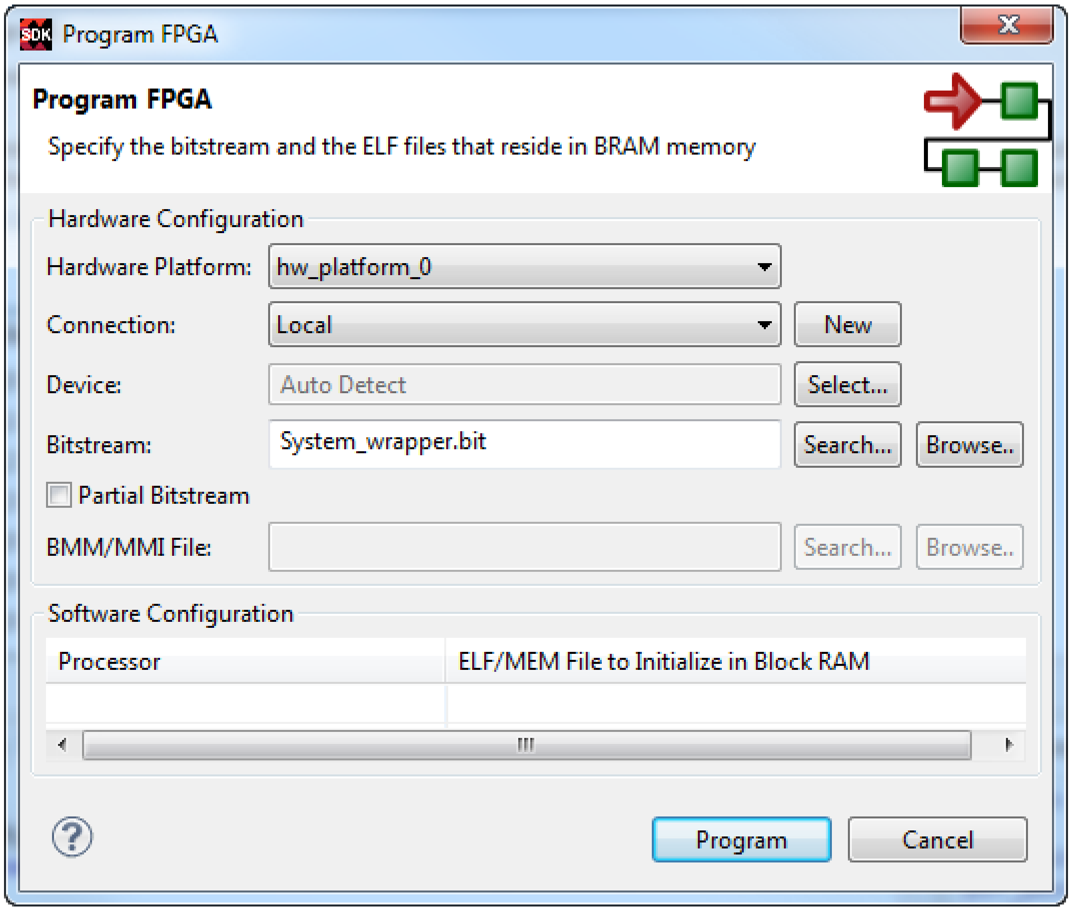
Parity: None

Flow Control: None

Timeout (sec): 5

Figure 21 – Terminal Settings Dialog Box

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 Blue DONE LED should light.



Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Platform: hw_platform_0

Connection: Local

Device: Auto Detect

Bitstream: System_wrapper.bit

Partial Bitstream ☐

BMM/MMI File:

Software Configuration

Processor	ELF/MEM File to Initialize in Block RAM

Figure 22 – Program FPGA

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

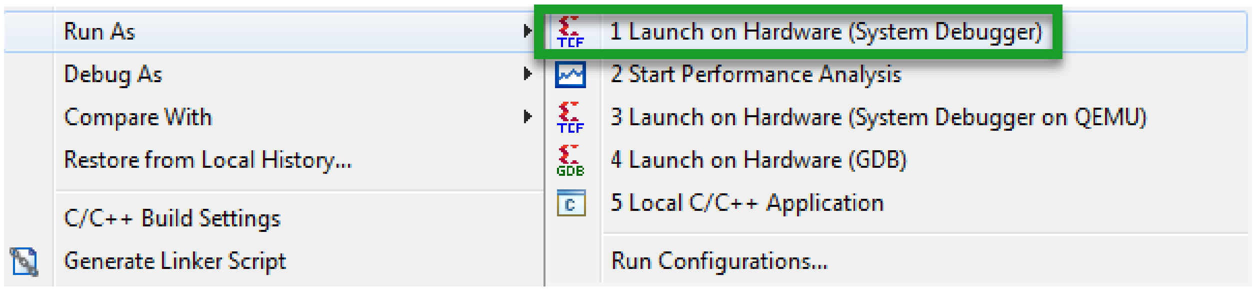
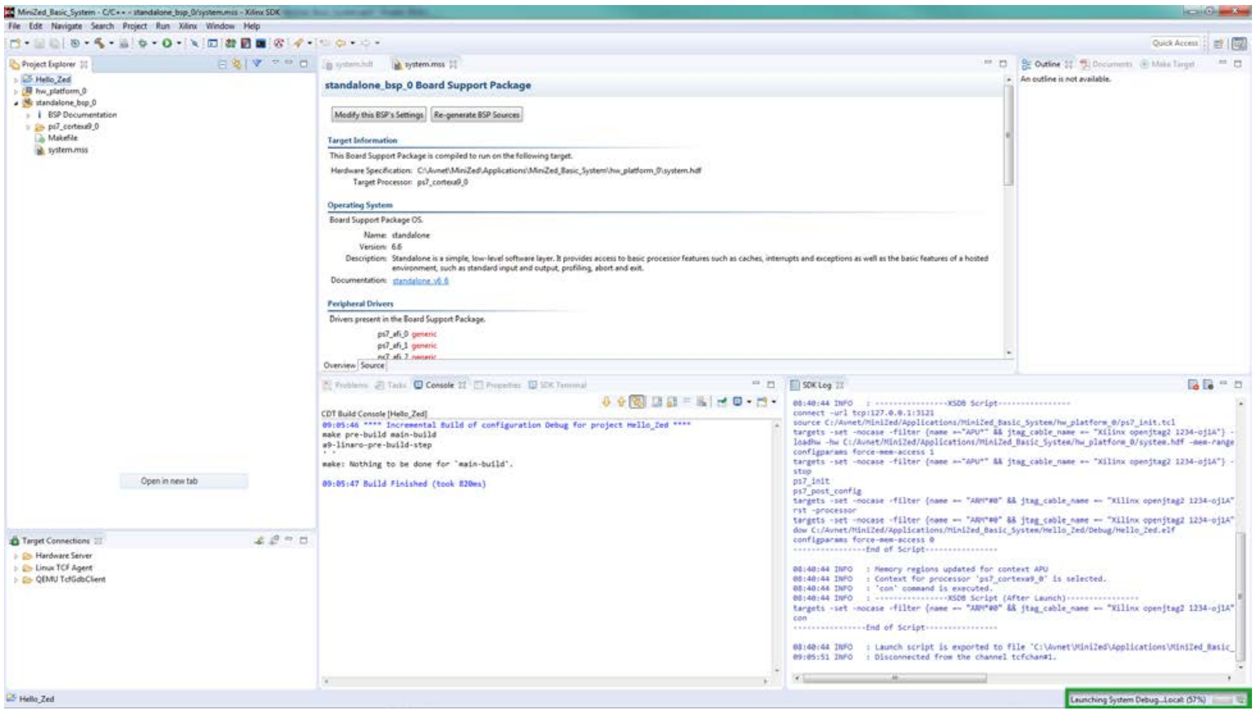

Run As
Debug As
Compare With
Restore from Local History...
C/C++ Build Settings
Generate Linker Script
1 Launch on Hardware (System Debugger)
2 Start Performance Analysis
3 Launch on Hardware (System Debugger on QEMU)
4 Launch on Hardware (GDB)
5 Local C/C++ Application
Run Configurations...

Figure 23 – Launch on Hardware (GDB)

14. The tools will now initialize the processor, download the Hello_Zed.elf to DDR, and then run Hello_Zed. 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.



standalone_bsp_0 Board Support Package

Target Information

This Board Support Package is compiled to run on the following target.

Hardware Specification: C:\Avnet\MiniZed\Applications\MiniZed_Basic_System\hw_platform_0\system.hdf
Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone
Version: 6.6
Description: Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.
Documentation: standalone_v6_6

Peripheral Drivers

Drivers present in the Board Support Package.

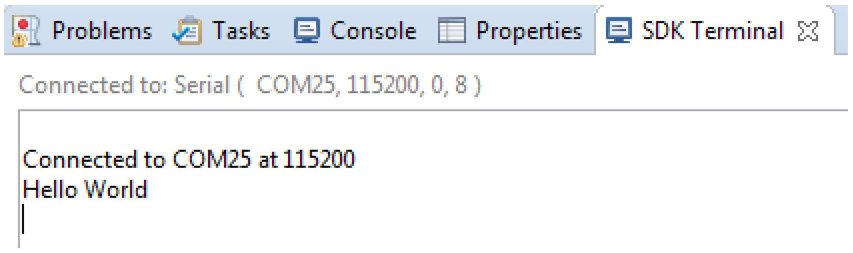
ps7_afi_0 generic
ps7_afi_1 generic
ps7_afi_2 generic

```
CDT Build Console [Hello_Zed]
09:05:46 **** Incremental Build of configuration Debug for project Hello_Zed ****
make pre-build main-build
a9-linaro-pre-build-step
' '
make: Nothing to be done for 'main-build'.

09:05:47 Build Finished (took 820ms)
```

```
08:40:44 INFO   : ----------------XSDB Script----------------
connect -url tcp:127.0.0.1:3121
source C:/Avnet/MiniZed/Applications/MiniZed_Basic_System/hw_platform_0/ps7_init.tcl
targets -set -nocase -filter {name =~"APU*" && jtag_cable_name =~ "Xilinx openjtag2 1234-oj1A"} -
loadhw -hw C:/Avnet/MiniZed/Applications/MiniZed_Basic_System/hw_platform_0/system.hdf -mem-range
configparams force-mem-access 1
targets -set -nocase -filter {name =~"APU*" && jtag_cable_name =~ "Xilinx openjtag2 1234-oj1A"} -
stop
ps7_init
ps7_post_config
targets -set -nocase -filter {name =~ "ARM*#0" && jtag_cable_name =~ "Xilinx openjtag2 1234-oj1A"
rst -processor
targets -set -nocase -filter {name =~ "ARM*#0" && jtag_cable_name =~ "Xilinx openjtag2 1234-oj1A"
dow C:/Avnet/MiniZed/Applications/MiniZed_Basic_System/Hello_Zed/Debug/Hello_Zed.elf
configparams force-mem-access 0
----------------End of Script----------------

08:40:44 INFO   : Memory regions updated for context APU
08:40:44 INFO   : Context for processor 'ps7_cortexa9_0' is selected.
08:40:44 INFO   : 'con' command is executed.
08:40:44 INFO   : ----------------XSDB Script (After Launch)----------------
targets -set -nocase -filter {name =~ "ARM*#0" && jtag_cable_name =~ "Xilinx openjtag2 1234-oj1A"
con
----------------End of Script----------------

08:40:44 INFO   : Launch script is exported to file 'C:\Avnet\MiniZed\Applications\MiniZed_Basic_
09:05:51 INFO   : Disconnected from the channel tcfchan#1.
```

Figure 24 – Launching Hello_Zed Progress

SDK will download the Hello World ELF to the DDR3, 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 *Terminal* tab to see the output.



```
Connected to: Serial ( COM25, 115200, 0, 8 )

Connected to COM25 at 115200
Hello World
```

Figure 25 – Hello Zed Complete

You have now booted Zynq hardware on MiniZed! The Terminal can be disconnected by clicking the ✖ button.

Revision History

Date	Version	Revision
23 Aug 2013	2013_2.01	Initial Avnet release for Vivado 2013.2
09 Jun 2014	2014_1.01	Update to 2014.1
11 Jun 2014	2014_2.01	Update to 2014.2
29 Jun 2015	2015_1.01	Update to 2015.1. Add support for PicoZed.
15 Jul 2015	2015_2.01	Update to 2015.2.
06 Apr 2016	2015_4.01	Update to 2015.4. Add support for PZCC-FMC-V2.
01 Jun 2016	2015_4.02	Update to 2015.4. Add picoZed JTAG Boot picture.
29 Aug 2016	2015_4.03	Clarified JTAG port for PZ FMC Carrier Card V2
09 Sept 2016	2016_2.01	Updated to 2016.2
20 Jan 2017	2016_4.01	Updated to 2016.4
06 Jun 2017	2017_1.01	Updated to 2017.1 for MiniZed
10 Apr 2018	2018_1.01	Updated to 2018.1 for MiniZed