Laboratory 1 – HelloWorld and Application Debugging

I. Hello World

Objectives

- To familiarize with the hardware and software environments
- To execute a software program to print "Hello World" on the screen

Activity Summary

1. Open hardware platform
2. Export hardware platform
3. Launch Xilinx SDK
4. Create HelloWorld application
5. Connect and Program FPGA
6. Run application

Note: You can use the numbers in blue circles to keep track of your activities

Activities

1

First, there is need to get familiar with the Xilinx Software Development Kit (Xilinx SDK), which is the software you will be using in this laboratory to develop your software applications. It is based on Eclipse 4.3.2 and CDT 8.3.0 (as of 2015.1 release). However, before we get to the SDK part, we need to prepare the hardware design required to run our software applications. Download the **Lab_1** folder from Learn and unzip it. On a Linux computer, click **Start** then **SEE Lab**; select **Xilinx** and click **Vivado**. On a Windows machine you can do something similar by going to All Programs from the Start Menu. The window in Figure 1 should show up. Click **Open Project** and navigate to the file named **es3_hw_lab_1** in the folder you downloaded. (Hint: This is the file that has the icon:). Once you do this, wait for the Vivado IDE to open. Eventually, a window that looks like the one in Figure 2 should appear.

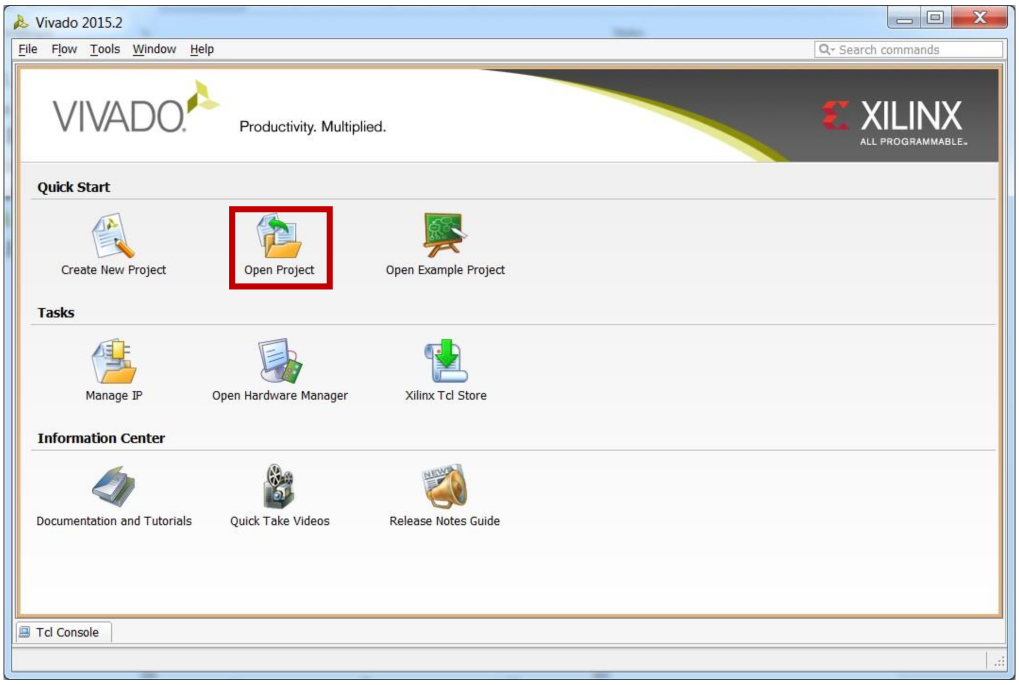
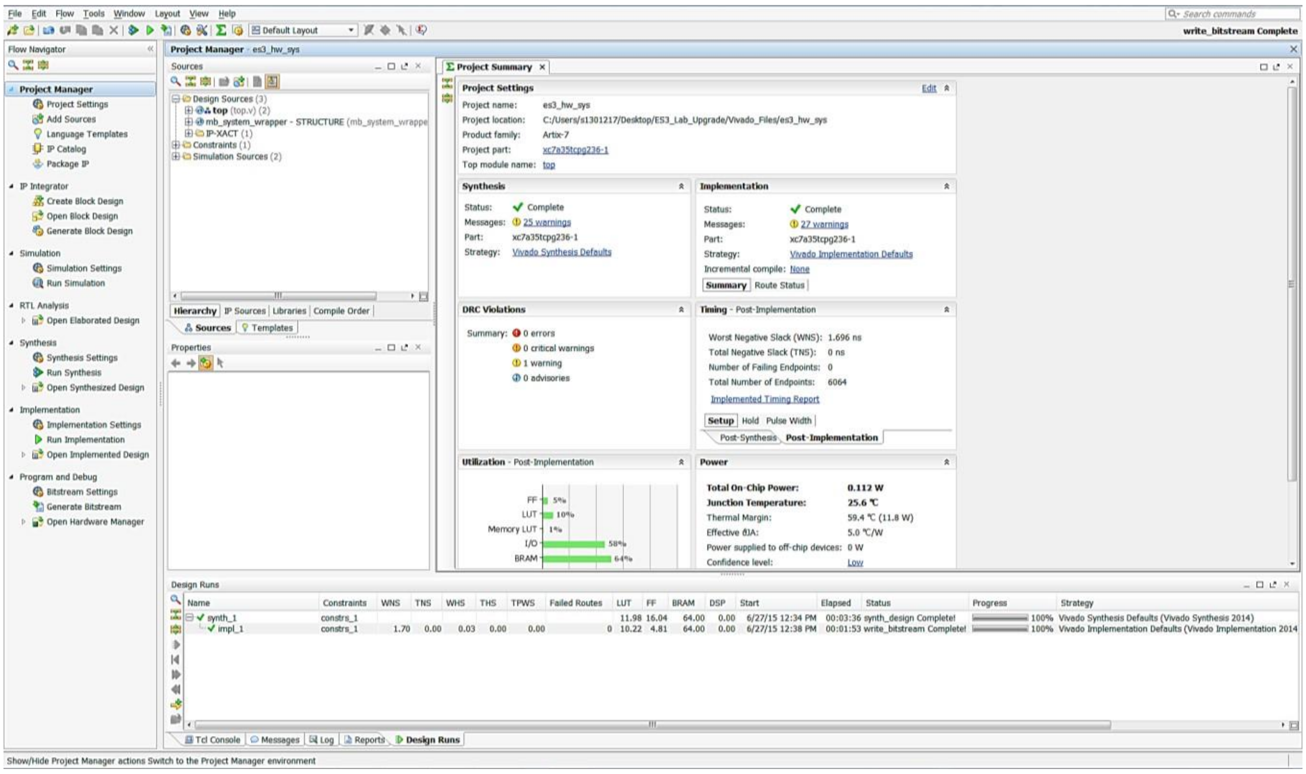


Figure 1



Design Sources (3)
- top (top.v) (2)
- mb_system_wrapper - STRUCTURE (mb_system_wrappe
- IP-XACT (1)

Constraints (1)
Simulation Sources (2)

Project Settings

Project name: es3_hw_sys
Project location: C:/Users/s1301217/Desktop/ES3_Lab_Upgrade/Vivado_Files/es3_hw_sys
Product family: Artix-7
Project part: xc7a35tcpg236-1
Top module name: top

Synthesis

Status: Complete
Messages: 25 warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults

Implementation

Status: Complete
Messages: 27 warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations

Summary: 0 errors
0 critical warnings
1 warning
0 advisories

Timing - Post-Implementation

Worst Negative Slack (WNS): 1.696 ns
Total Negative Slack (TNS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 6064
Implemented Timing Report

Utilization - Post-Implementation

FF 5%
LUT 10%
Memory LUT 1%
I/O 58%
BRAM 64%

Power

Total On-Chip Power: 0.112 W
Junction Temperature: 25.6 °C
Thermal Margin: 59.4 °C (11.8 W)
Effective θJA: 5.0 °C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low

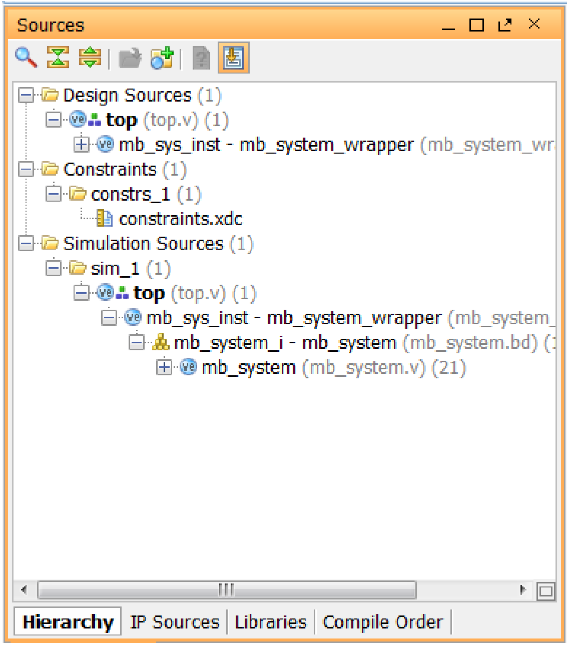
Design Runs

Name	Constraints	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	DSP	Start	Elapsed	Status	Progress	Strategy
synth_1	constrs_1							11.98	16.04	64.00	0.00	6/27/15 12:34 PM	00:03:36	synth_design Complete!	100%	Vivado Synthesis Defaults (Vivado Synthesis 2014)
impl_1	constrs_1	1.70	0.00	0.03	0.00	0.00	0	10.22	4.81	64.00	0.00	6/27/15 12:38 PM	00:01:53	write_bitstream Complete!	100%	Vivado Implementation Defaults (Vivado Implementation 2014

Show/Hide Project Manager actions Switch to the Project Manager environment

Figure 2

Now, you have just opened the project file that contains the underlying hardware for the laboratory. If you have time you can open the hardware code files to have a feel of the hardware design. The files can be located by expanding the file tree in the **Project Manager** window (Figure 3). You can open the files by double clicking, but be careful not to modify anything. You really do not have anything to do here; the hardware design has already been done and the necessary hardware platform needed by the software development tool (Xilinx SDK) has also been generated. You are only required to export this platform.



Design Sources (1)
- top (top.v) (1)
  - mb_sys_inst - mb_system_wrapper (mb_system_wr

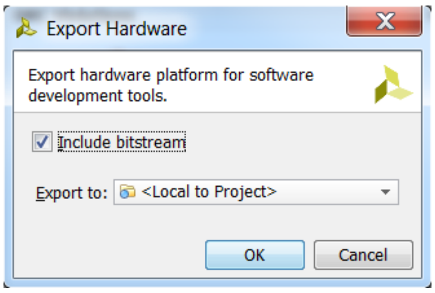
Constraints (1)
- constrs_1 (1)
  - constraints.xdc

Simulation Sources (1)
- sim_1 (1)
  - top (top.v) (1)
    - mb_sys_inst - mb_system_wrapper (mb_system_
      - mb_system_i - mb_system (mb_system.bd) (
        - mb_system (mb_system.v) (21)

Figure 3

2

To export the hardware platform click **File**, click **Export**, then click **Export Hardware**. In the window that appears (Figure 4a), make sure **Include bitstream** is selected and then click **OK**. If an exported file already exists in the project, the window in Figure 4b will appear. Then, click **Yes**. If not, the hardware platform will already be exported.

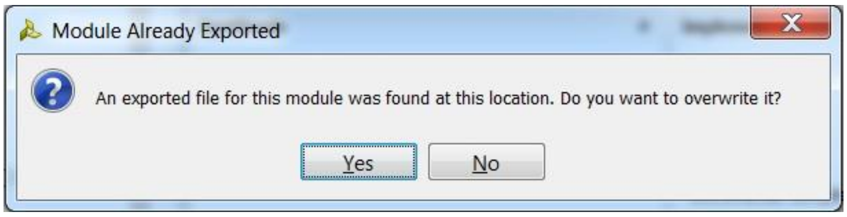


Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

Figure 4(a)

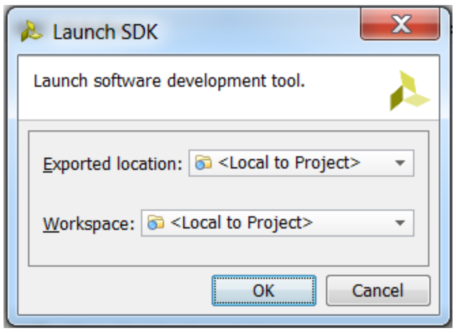


An exported file for this module was found at this location. Do you want to overwrite it?

Figure 4(b)

3

It is now time to launch the Xilinx SDK. Click **File**, then click **Launch SDK**. A window (Figure 5a) appears asking for the location of the exported hardware and the workspace to be used for the software applications. If you have not modified the *Export to* field while exporting the hardware in the previous step, you should just click **OK**. Otherwise, you will have to navigate to the location you selected. It is also recommended to allow the *Workspace* to be "Local to Project". After you have clicked OK, the SDK splash screen will appear (Figure 5b).



Launch SDK

Launch software development tool.

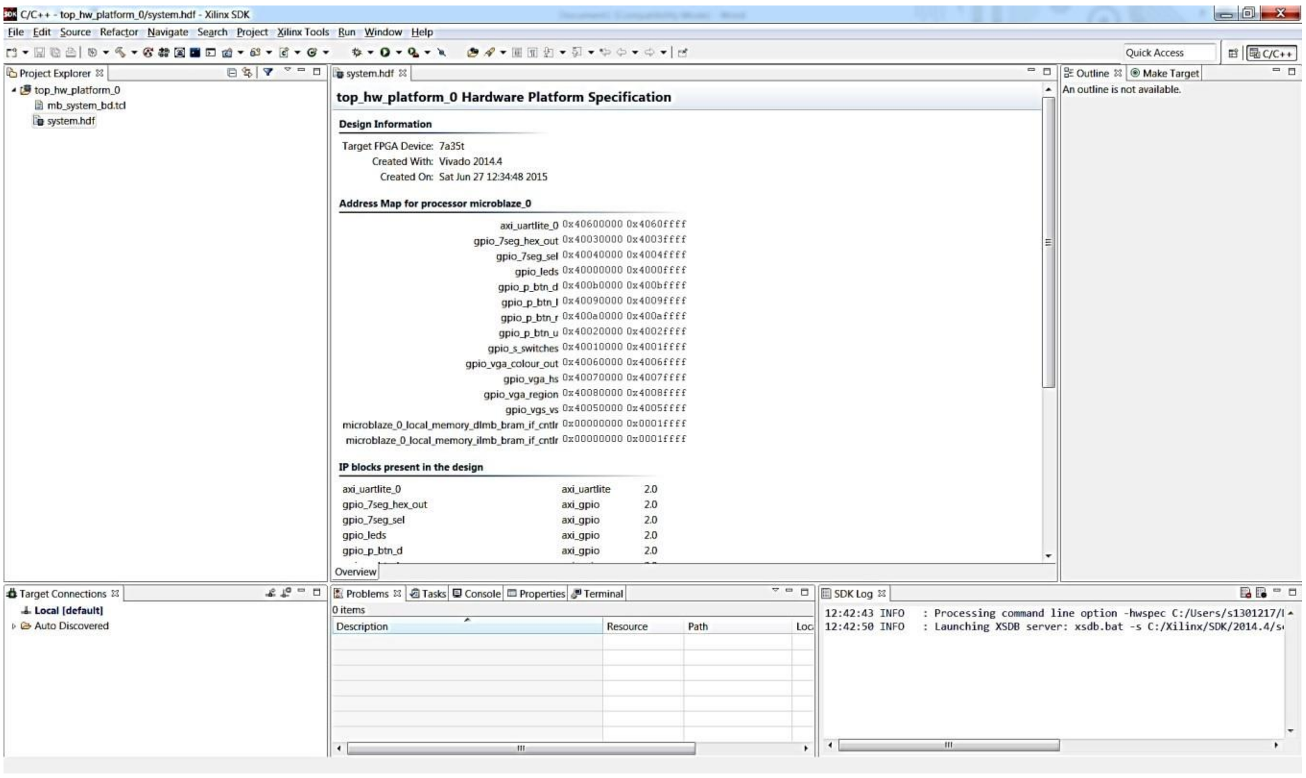
Exported location: <Local to Project>

Workspace: <Local to Project>

Figure 5(a)

Figure 5(b)

The Xilinx SDK window opens and looks like what we have in Figure 6. At this point we no longer need the Vivado IDE and you might as well close it, but it is recommended you leave it open. In case the SDK is closed for any reason, the Vivado IDE is a quick means of relaunching it.



top_hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device: 7a35t
Created With: Vivado 2014.4
Created On: Sat Jun 27 12:34:48 2015

Address Map for processor microblaze_0

axi_uartlite_0	0x40600000	0x4060ffff
gpio_7seg_hex_out	0x40030000	0x4003ffff
gpio_7seg_sel	0x40040000	0x4004ffff
gpio_leds	0x40000000	0x4000ffff
gpio_p_btn_d	0x400b0000	0x400bffff
gpio_p_btn_l	0x40090000	0x4009ffff
gpio_p_btn_r	0x400a0000	0x400affff
gpio_p_btn_u	0x40020000	0x4002ffff
gpio_s_switches	0x40010000	0x4001ffff
gpio_vga_colour_out	0x40060000	0x4006ffff
gpio_vga_hs	0x40070000	0x4007ffff
gpio_vga_region	0x40080000	0x4008ffff
gpio_vgs_vs	0x40050000	0x4005ffff
microblaze_0_local_memory_dlmb_bram_if_cntlr	0x00000000	0x0001ffff
microblaze_0_local_memory_ilmb_bram_if_cntlr	0x00000000	0x0001ffff

IP blocks present in the design

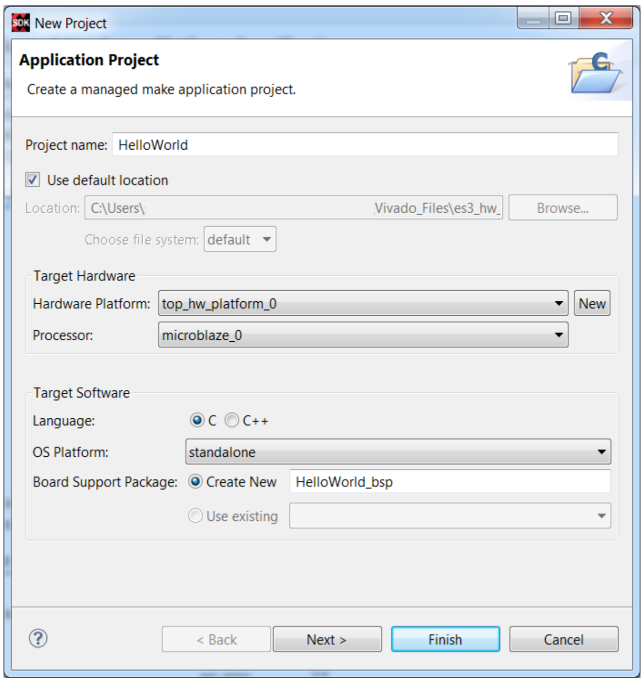
axi_uartlite_0	axi_uartlite	2.0
gpio_7seg_hex_out	axi_gpio	2.0
gpio_7seg_sel	axi_gpio	2.0
gpio_leds	axi_gpio	2.0
gpio_p_btn_d	axi_gpio	2.0

```
12:42:43 INFO  : Processing command line option -hwspec C:/Users/s1301217/l
12:42:50 INFO  : Launching XSDB server: xsdb.bat -s C:/Xilinx/SDK/2014.4/s
```

Figure 6

4

In this lab, you are required to create a HelloWorld application that displays "Hello World" on the SDK console. Follow the following steps to create the HelloWorld application. Click **File**, click **New**, and click **Application Project**. In the window (Figure 7a) that appears, set the *Project name* to "HelloWorld" and make sure *Use default location* is selected. Leave the other settings with their default values, then click **Next**. The New Project Templates window (Figure 7b) is then displayed. Select "Hello World" and click **Finish**.



Project name: HelloWorld

[✓] Use default location

Location: C:\Users\ Vivado_Files\es3_hw_

Choose file system: default

Target Hardware

Hardware Platform: top_hw_platform_0

Processor: microblaze_0

Target Software

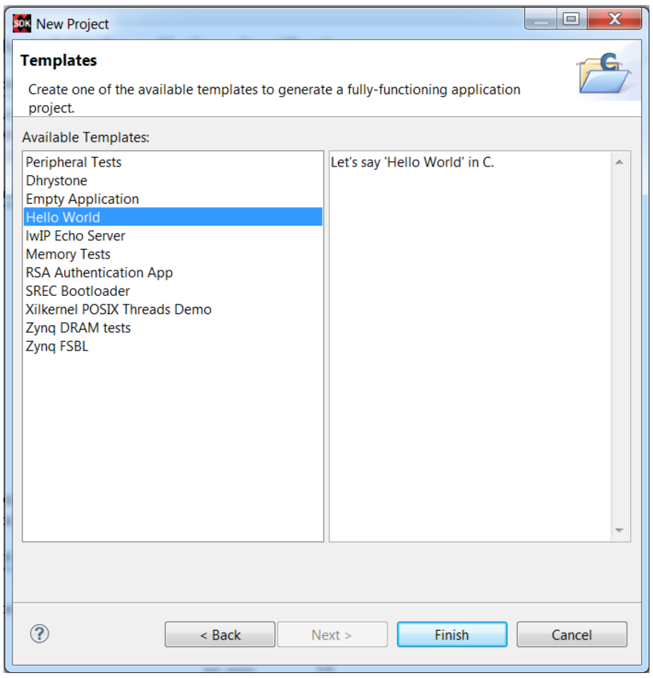
Language: [✓ C] C++

OS Platform: standalone

Board Support Package: [✓ Create New] HelloWorld_bsp

Use existing

Figure 7(a)



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

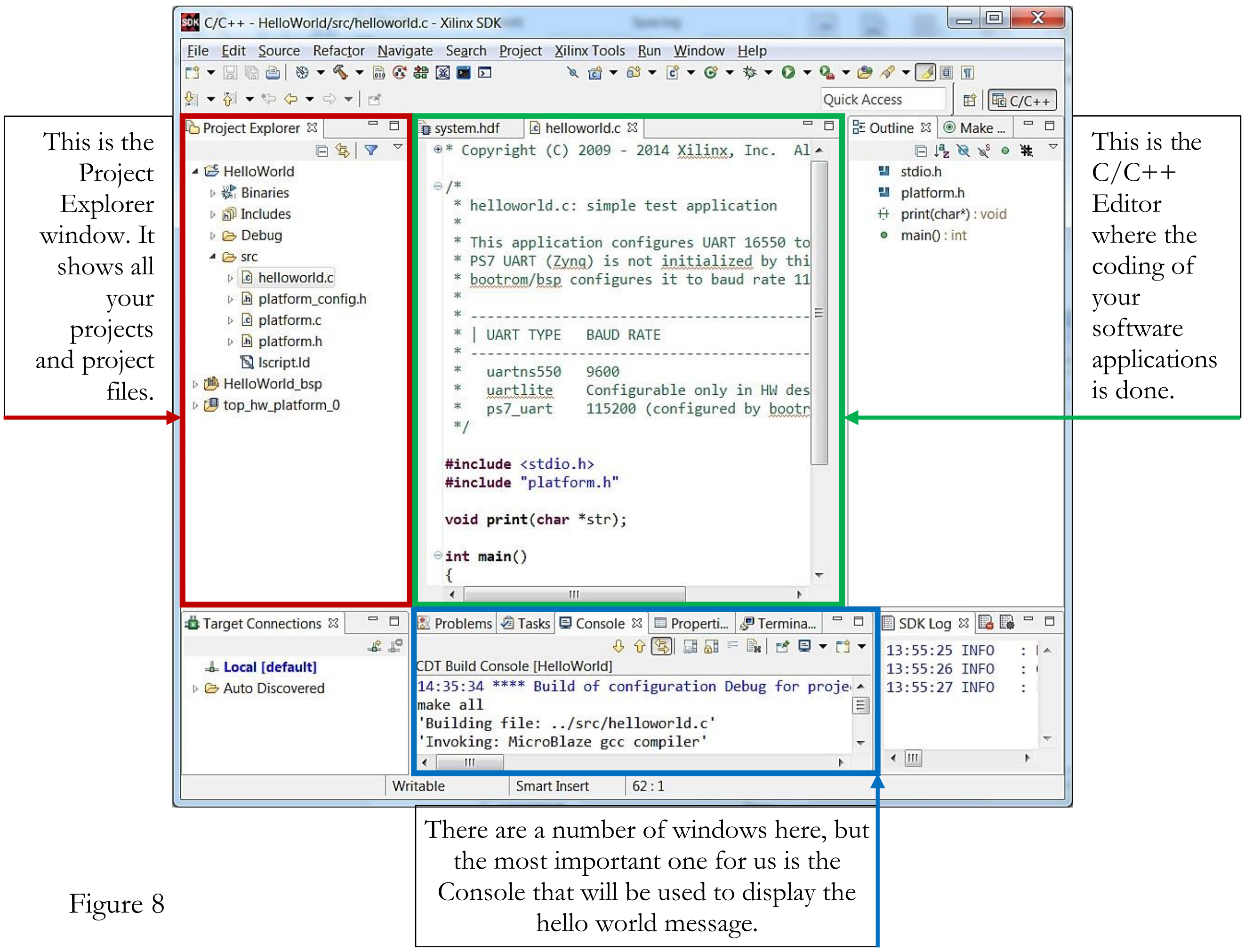
Available Templates:

- Peripheral Tests
- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- RSA Authentication App
- SREC Bootloader
- Xilkernel POSIX Threads Demo
- Zynq DRAM tests
- Zynq FSBL

Let's say 'Hello World' in C.

Figure 7(b)

Then, expand the HelloWorld project in the Project Explorer window (see Figure 8). Expand the **src** folder and open the **helloworld.c** file. This file contains the helloword application C source code and it has been added automatically by the SDK tool because we selected the "Hello World" template in the previous step. Figure 8 shows other useful windows that you need to be familiar with. Once you double click the helloworld.c file, it opens in the working area.



```
* Copyright (C) 2009 - 2014 Xilinx, Inc.  Al

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to
 * PS7 UART (Zynq) is not initialized by thi
 * bootrom/bsp configures it to baud rate 11
 *
 * ------------------------------------------
 * | UART TYPE    BAUD RATE
 * ------------------------------------------
 *   uartns550    9600
 *   uartlite     Configurable only in HW des
 *   ps7_uart     115200 (configured by bootr
 */

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

void print(char *str);

int main()
{
```

```
CDT Build Console [HelloWorld]
14:35:34 **** Build of configuration Debug for proje
make all
'Building file: ../src/helloworld.c'
'Invoking: MicroBlaze gcc compiler'
```

```
13:55:25 INFO   :
13:55:26 INFO   :
13:55:27 INFO   :
```

This is the Project Explorer window. It shows all your projects and project files.

This is the C/C++ Editor where the coding of your software applications is done.

There are a number of windows here, but the most important one for us is the Console that will be used to display the hello world message.

Figure 8

A brief explanation of the Hello World code is now given:

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

These are the inclusions of the header files that contain information needed for the proper execution of `print()` and `_platform()`

```
void print(char *str);
```

Declaration of the `print()` function.

```
int main()
{
    init_platform();

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

    cleanup_platform();
    return 0;
}
```

This is function `main()`, which is the entrance to the application. Every C program must have only one of this. The `init_platform()` is used to enable the MicroBlaze caches and to initialize the uart. Once the application is about to exit, the cleanup_platform() function must be called to disable the caches. The `print()` function enables the writing of the string "`Hello World`" to the console. Note that "`\n\r`" are new line and return characters that enable the console to move to a new line and return the cursor to the beginning of the line. The `return 0` is a standard C return code that signals the end of the application. Since the `main()` function is usually of type integer, it must necessarily return an integer value. Consult the reading materials on Learn for more information.

5

Connect the FPGA board to your PC using USB cable, ensure the status LED is green as shown in Figure 9.

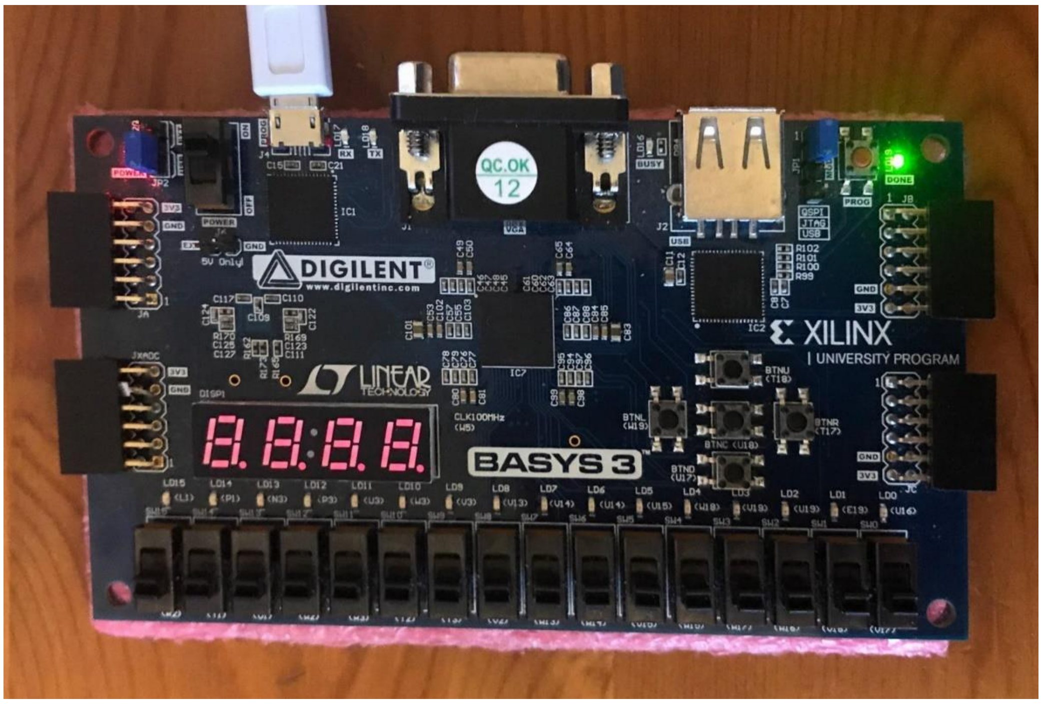


Figure 9

In order to programme the FPGA, click **Xilinx Tools** in the menu bar, then click Program FPGA. A quick method is to use the icon: in the toolbar (see Figure 10). Either way, the window in Figure 11 results. Usually, you do not have to change anything. Simply click **Program** to configure the FPGA. A window then appears to show the progress of the configuration. Once configuration is complete, this window disappears and if everything goes fine, there should be no error message.

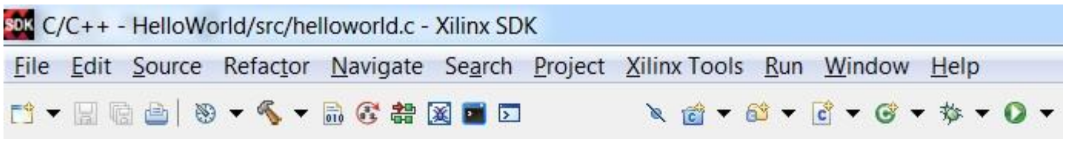
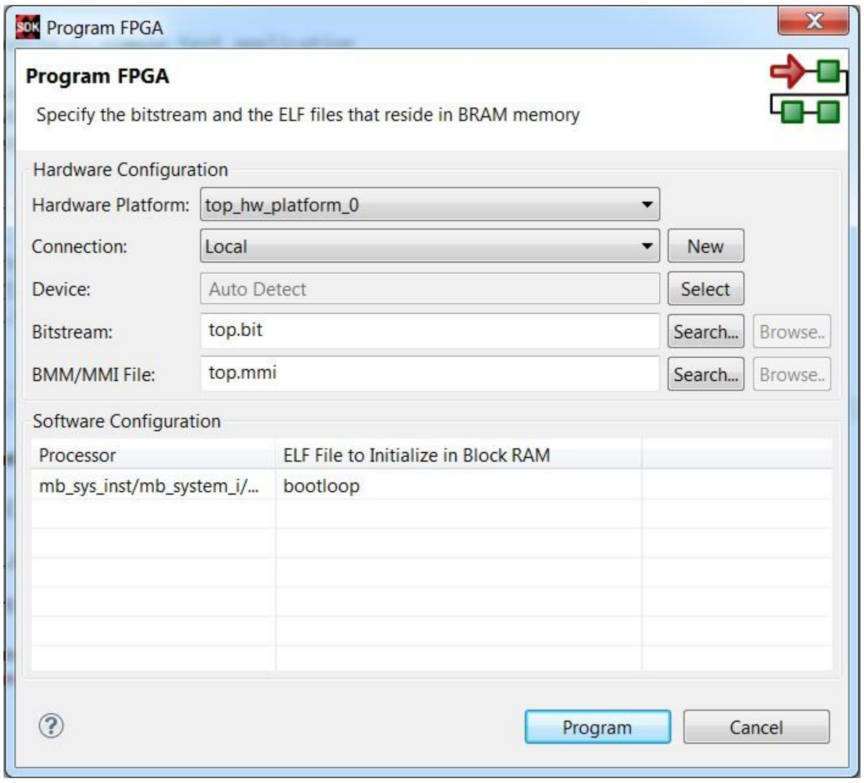

C/C++ - HelloWorld/src/helloworld.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help

Figure 10



Program FPGA

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

Hardware Configuration

Hardware Platform: top_hw_platform_0

Connection: Local

Device: Auto Detect

Bitstream: top.bit

BMM/MMI File: top.mmi

Software Configuration

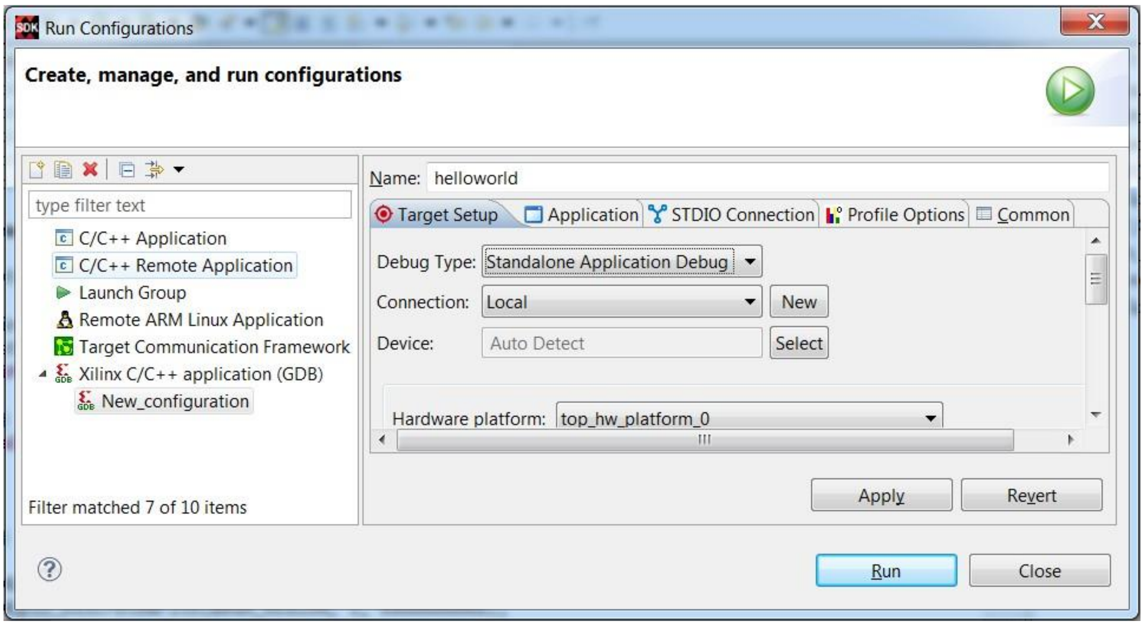
Processor	ELF File to Initialize in Block RAM
mb_sys_inst/mb_system_i/...	bootloop

Figure 11

6

Now we need to run the HelloWorld application. To do this we first need to setup the run configuration. Click **Run** on the menu bar, then click **Run Configurations**. You can also get to the Run Configurations by clicking the down arrow on the icon ▶ ▾. However, be careful not to click the run button ▶ itself since the run configuration has not yet been set up. However, if this happens, simply go back to click Run Configurations. In the window that shows up, double click **Xilinx C/C++ application (GDB)**. Once you do this, a new configuration will be formed. You need to rename this to helloworld (see Figure 12). There are a few other things to do here. Click on the **Application** tab. For the **Project Name**, click **Browse** and select the **HelloWorld** application (see Figure 12-b). Next, select the **STDIO Connection** tab and select **Connect STDIO to Console**. Select the applicable **Port** (other than JTAG UART) and leave the **BAUD rate** at the default 9600. We are now set and done; click on **Apply**, then on **Run**. If everything goes well, the Console and the status bar should show the progress as the SDK downloads the executable file to the Microblaze processor. Once the download is complete, the application immediately starts executing and the "Hello World" string is displayed on the Console (see Figure 13).

You can play around with the print function by writing strings of your own to display on the Console. Simply **modify** the code, and **save** it. Since the run configuration has already been set up, you can run the application. Click **Run**, then click **Run As**. Next, click **Launch on Hardware (GDB)**. Alternatively, right click in the helloworld.c window and click **Run As**, then click **Launch on Hardware (GDB)**. Also, to get to the Run As menu, you can click the down arrow on the icon ▶ ▾. The simplest however, is to click the run button ▶ on the toolbar, but only if the application has been run recently using the other methods.



Name: helloworld

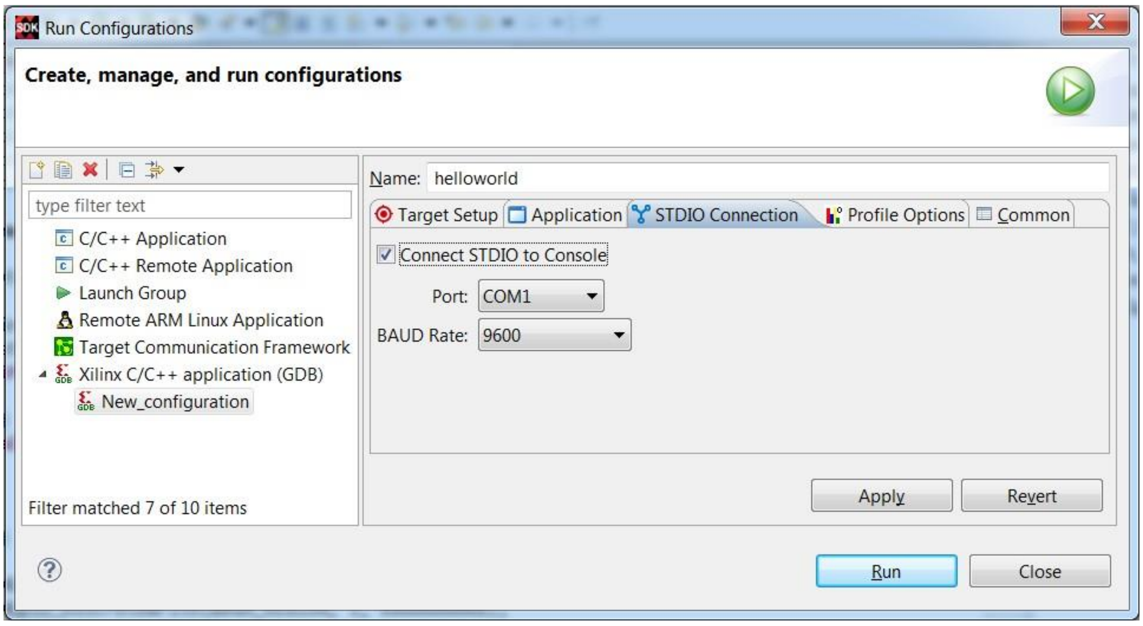
Debug Type: Standalone Application Debug

Connection: Local

Device: Auto Detect

Hardware platform: top_hw_platform_0

Figure 12



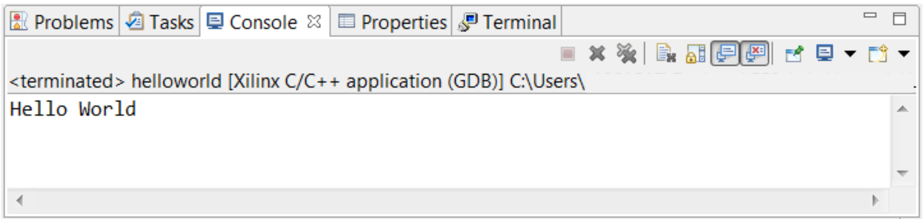
Name: helloworld

☑ Connect STDIO to Console

Port: COM1

BAUD Rate: 9600

Figure 12-b



```
<terminated> helloworld [Xilinx C/C++ application (GDB)] C:\Users\
Hello World
```

Figure 13

II. Application Debugging

Objectives

- To learn how to debug applications in the Xilinx SDK environment

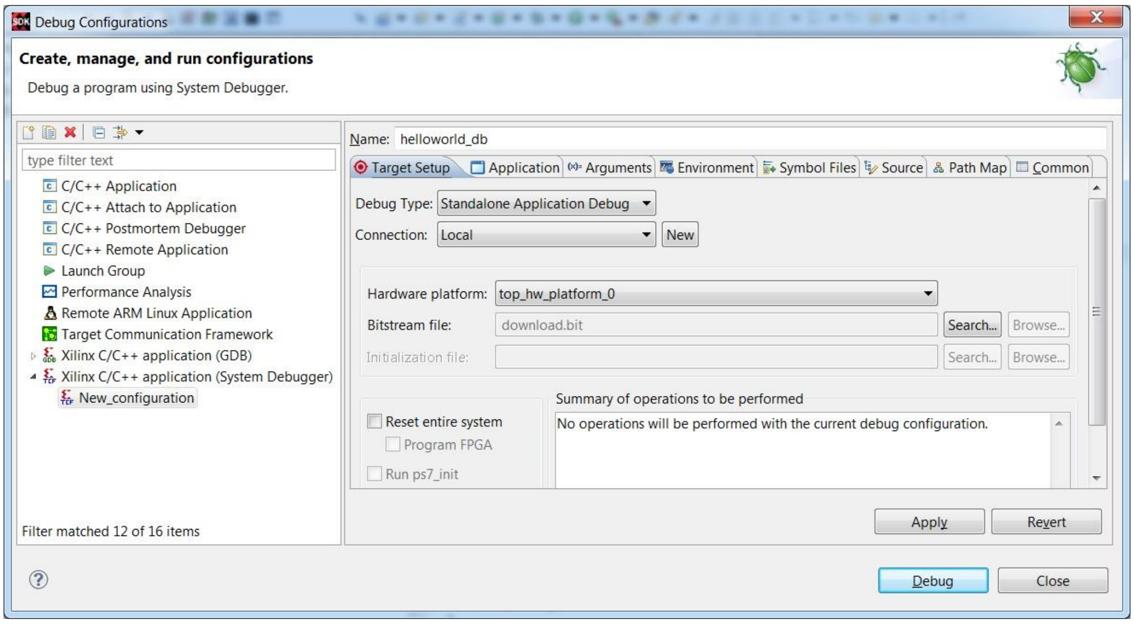
Activity Summary

1. Open the HelloWorld application if you have closed it
2. Setup Debug Configurations
3. Debug the application

Activities

In this section you will learn how to debug software applications in the Xilinx SDK. While the chance of the HelloWorld application not working if you have followed the steps properly is low, it however necessary to understand the usage of debuggers; there are applications with thousands of code lines that cannot be debugged unless a debugger is used. To check the application when a code behaviour seems to be out a place, a debugger is often used. The debugger allows a line-by-line stepping through the code to check for code behaviour. Alternatively, you can put breakpoints or watchpoints in specific locations in the code so that during execution, the processor will stop at that point for you to observe the behaviour of the code. The debugger is a very powerful and necessary tool for practical software application development.

In order to debug the HelloWorld application, we need to setup Debug Configuration. Click **Run** on the menu bar, then click **Debug Configurations**. You can also get to the Debug Configurations by clicking the down arrow on the icon . The window that appears looks very similar to the Run Configurations window. However, here, double click **Xilinx C/C++ application (System Debugger)** and rename the new configuration to helloworld_db (see Figure 14). There are a few other things to do here. In the **Target Setup** tab, change the **Debug Type** to **Standalone Application Debug**. Now, click on the **Application** tab and make sure microblaze_0 is selected. Also, select the **Download application** check box and make sure that the Project Name selected is HelloWorld (in case you have more than one applications in opened) (see Figure 15). Click **Apply**, and then click **Close**.



Debug Configurations

Create, manage, and run configurations
Debug a program using System Debugger.

type filter text
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- Launch Group
- Performance Analysis
- Remote ARM Linux Application
- Target Communication Framework
- Xilinx C/C++ application (GDB)
- Xilinx C/C++ application (System Debugger)
  - New_configuration

Filter matched 12 of 16 items

Name: helloworld_db

Target Setup | Application | Arguments | Environment | Symbol Files | Source | Path Map | Common

Debug Type: Standalone Application Debug
Connection: Local New

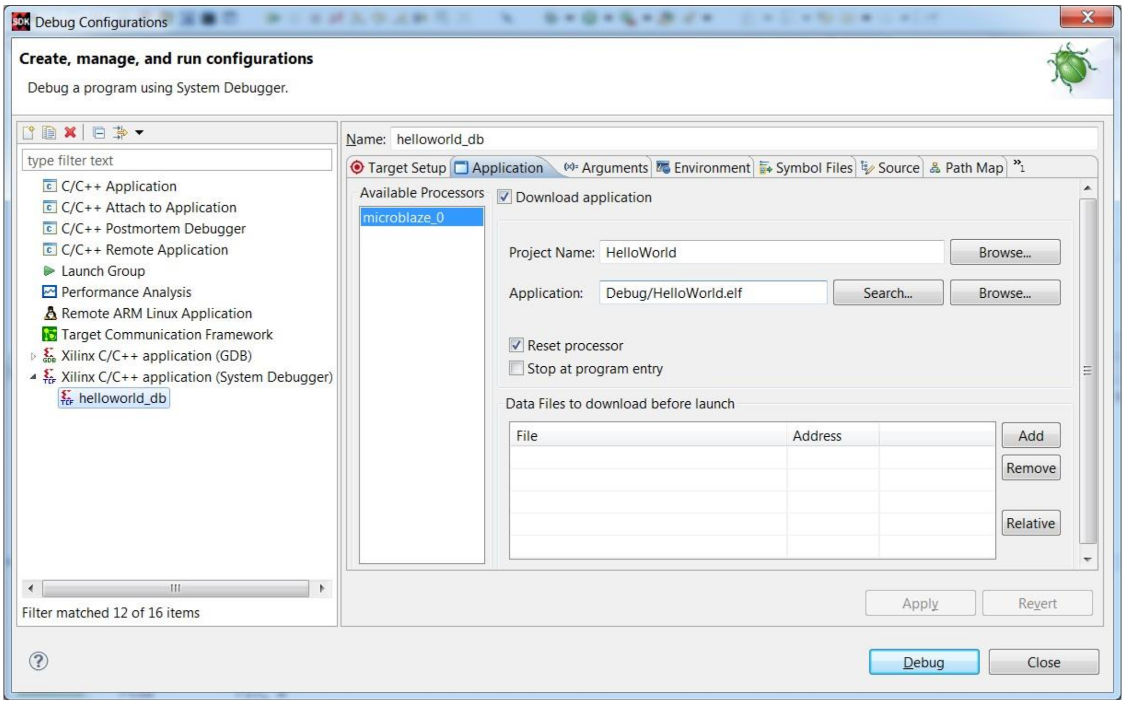
Hardware platform: top_hw_platform_0
Bitstream file: download.bit Search... Browse...
Initialization file: Search... Browse...

Reset entire system
Program FPGA
Run ps7_init

Summary of operations to be performed
No operations will be performed with the current debug configuration.

Apply Revert
Debug Close

Figure 14


Debug Configurations

Create, manage, and run configurations

Debug a program using System Debugger.

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- Launch Group
- Performance Analysis
- Remote ARM Linux Application
- Target Communication Framework
- Xilinx C/C++ application (GDB)
- Xilinx C/C++ application (System Debugger)
  - helloworld_db

Filter matched 12 of 16 items

Name: helloworld_db

Tabs: Target Setup | Application | Arguments | Environment | Symbol Files | Source | Path Map

Available Processors: microblaze_0

☑ Download application

Project Name: HelloWorld

Application: Debug/HelloWorld.elf

☑ Reset processor

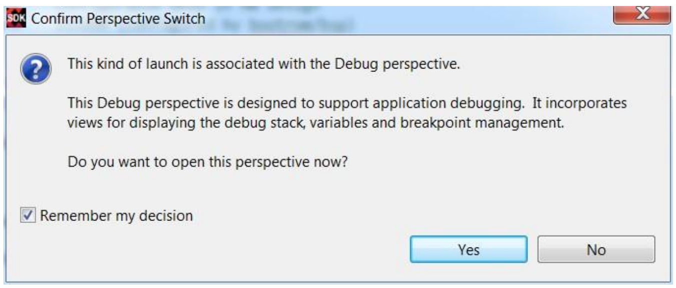
☐ Stop at program entry

Data Files to download before launch

File	Address

Figure 15

There are various ways of debugging an application in Xilinx SDK. By right-clicking inside the helloworld.c code and selecting **Debug As**, you should see five options. The option we will use is **Launch on Hardware (GDB)**. This debugs the application while it is running inside the FPGA. Once you click **Launch on Hardware (GDB)**, the window in Figure 16 shows up. Check **Remember my decision** and click **Yes**. This window is informing you that the Xilinx SDK will switch the perspective to the debug mode and this is what we want. The Debug window of Figure 17 then comes up.


Confirm Perspective Switch

This kind of launch is associated with the Debug perspective.

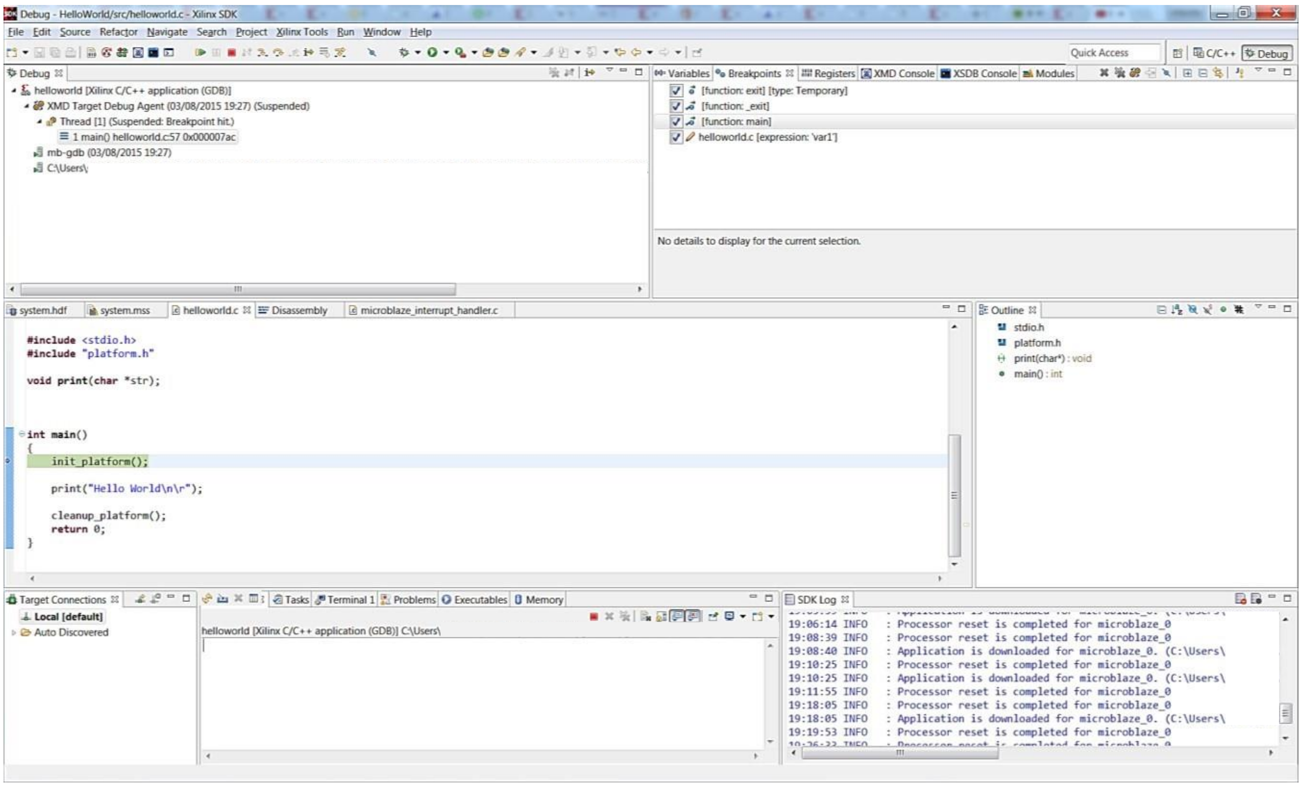
This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☑ Remember my decision

Figure 16

In the debug perspective of Figure 17 we can still see the helloworld.c tab. The Console also shows up in its default bottom-middle position. There are however, additional windows that come as the application is debugged. The **Disassembly tab**, which usually appears automatically shows the assembly language code to which the application has been converted. In the top-right child window, we have the **Variables**, **Breakpoints**, **Registers**, and other tabs. The Variables tab shows the behaviours of the variables in the application while the Breakpoints tab shows the breakpoints that have been added to the application code for debugging. The Registers tab shows the present content of the registers of the Microblaze processor and one should be able to see how the variables in the application are mapped to registers in the processor. To switch at any time between the C/C++ perspective and the Debug perspective, click on the C/C++ button (C/C++) or the Debug button (Debug) respectively at the top-right corner.



```
helloworld [Xilinx C/C++ application (GDB)]
  XMD Target Debug Agent (03/08/2015 19:27) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() helloworld.c:57 0x000007ac
  mb-gdb (03/08/2015 19:27)
  C:\Users\
```

```
[function: exit] [type: Temporary]
[function: _exit]
[function: main]
helloworld.c [expression: 'var1']
```

No details to display for the current selection.

```c
#include <stdio.h>
#include "platform.h"

void print(char *str);


int main()
{
    init_platform();

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

    cleanup_platform();
    return 0;
}
```

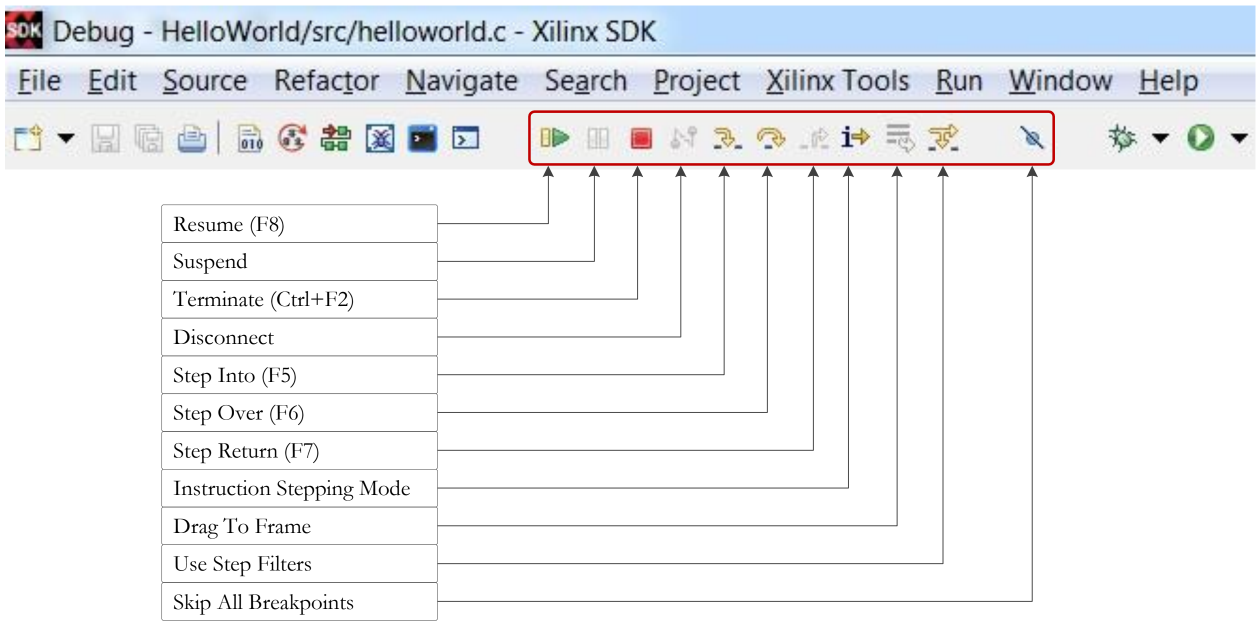
```
stdio.h
platform.h
print(char*) : void
main() : int
```

helloworld [Xilinx C/C++ application (GDB)] C:\Users\

```
19:06:14 INFO   : Processor reset is completed for microblaze_0
19:08:39 INFO   : Processor reset is completed for microblaze_0
19:08:40 INFO   : Application is downloaded for microblaze_0. (C:\Users\
19:10:25 INFO   : Processor reset is completed for microblaze_0
19:10:25 INFO   : Application is downloaded for microblaze_0. (C:\Users\
19:11:55 INFO   : Processor reset is completed for microblaze_0
19:18:05 INFO   : Processor reset is completed for microblaze_0
19:18:05 INFO   : Application is downloaded for microblaze_0. (C:\Users\
19:19:53 INFO   : Processor reset is completed for microblaze_0
```

Figure 17

In order to move around the code the buttons in the status bar can be used. See Figure 18 for a brief description of the buttons. Note that these icon can sometimes be rearranged by the tool, but there functionalities always remain the same. The program can be suspended or terminated by pressing the **Suspend** or **Terminate** buttons respectively. Checking **Instruction Stepping Mode** allows us to examine the program as it steps into disassembled code. Clicking **Resume** will start the application and keep it running unless there are breakpoints in the code. Since we have not added breakpoints at this point, you can press **Step Into** repeatedly to step into the block of codes in a method or press **Step Over** repeatedly to move through lines of code without stepping into method calls. You should notice that the "Hello World" message does not display until after you have pressed Step Into or Step Over a few times. If you run into some errors, simply terminate the program and restart debugging.



Debug - HelloWorld/src/helloworld.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help
Resume (F8)
Suspend
Terminate (Ctrl+F2)
Disconnect
Step Into (F5)
Step Over (F6)
Step Return (F7)
Instruction Stepping Mode
Drag To Frame
Use Step Filters
Skip All Breakpoints

Figure 18

While you are stepping through the helloworld code, it is important to note that once the cursor gets to the last curly bracket you have to stop and restart debugging if you intend to observe the behaviours again. This is because at that point the `return 0` instruction which terminates the application has been run and as such asking the debugger to step into or over will lead to an error message. To avoid this, practical application codes are often written not to reach the last `return` instruction in the `main()`. A loop that executes forever is often used to achieve this. You will get to use this in subsequent laboratory sessions. For more information on Xilinx SDK debugger check the Debugging section of the Xilinx Software Development Kit Help Contents[1]. Note, to add breakpoints to the code, you can right click in the left margin of the C/C++ Editor (see Figure 19) and select **Add Breakpoint…**.

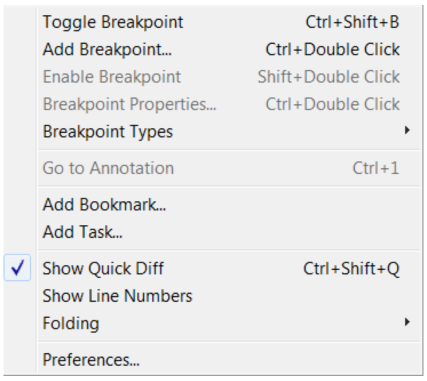

Toggle Breakpoint Ctrl+Shift+B
Add Breakpoint... Ctrl+Double Click
Enable Breakpoint Shift+Double Click
Breakpoint Properties... Ctrl+Double Click
Breakpoint Types
Go to Annotation Ctrl+1
Add Bookmark...
Add Task...
Show Quick Diff Ctrl+Shift+Q
Show Line Numbers
Folding
Preferences...

Figure 19

[1] http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/SDK_doc/index.html