QEMU Debug

Generate the Petalinux Image

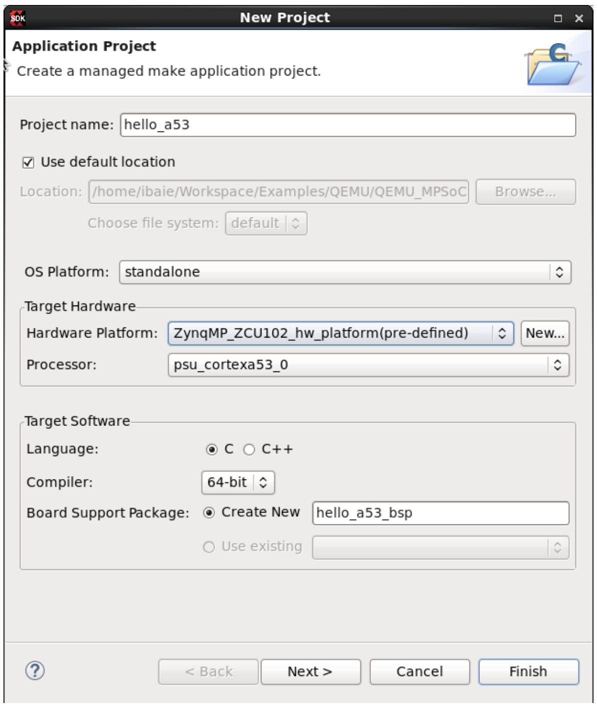
We create the Petalinux image based on a BSP for ZCU102 using the following commands:

```
petalinux-create –t project –s <ZCU-BSP-file> -n <project_name>

petalinux-build
```

Generate demo applications

In order to test the system create two applications one for each type of MPSoC core (A53 & R5) using the Xilinx SDK. Modify the printed message to point the processor where it is run.


New Project

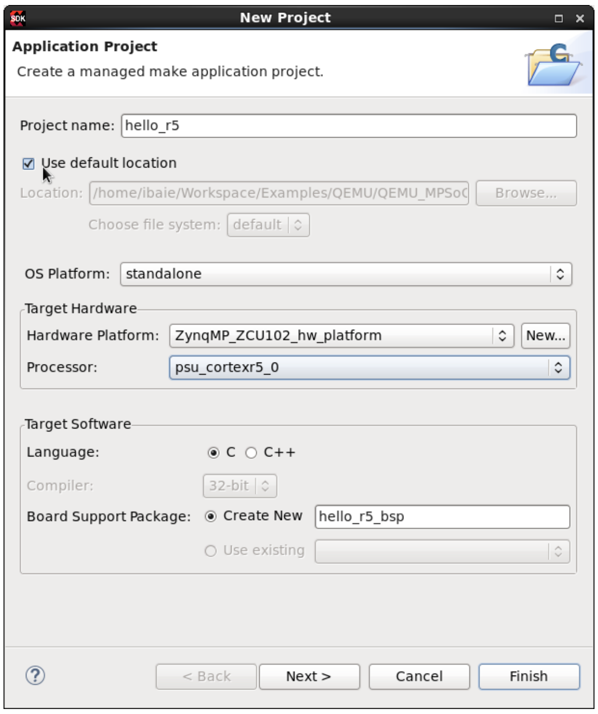
Application Project
Create a managed make application project.

Project name: hello_a53
☑ Use default location
Location: /home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC
Choose file system: default

OS Platform: standalone

Target Hardware
Hardware Platform: ZynqMP_ZCU102_hw_platform(pre-defined)
Processor: psu_cortexa53_0

Target Software
Language: [✓ C] C++
Compiler: 64-bit
Board Support Package: [✓ Create New] hello_a53_bsp
Use existing


New Project

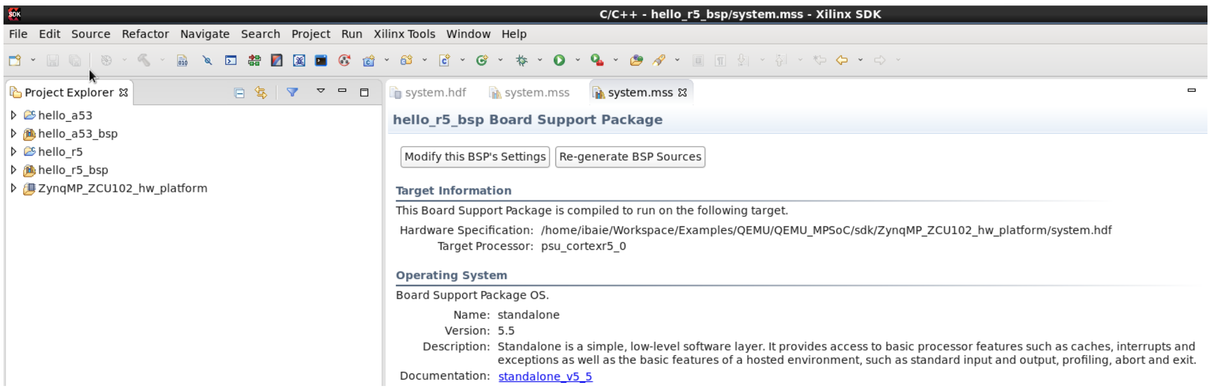
Application Project
Create a managed make application project.

Project name: hello_r5
☑ Use default location
Location: /home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC
Choose file system: default

OS Platform: standalone

Target Hardware
Hardware Platform: ZynqMP_ZCU102_hw_platform
Processor: psu_cortexr5_0

Target Software
Language: [✓ C] C++
Compiler: 32-bit
Board Support Package: [✓ Create New] hello_r5_bsp
Use existing


C/C++ - hello_r5_bsp/system.mss - Xilinx SDK

File Edit Source Refactor Navigate Search Project Run Xilinx Tools Window Help
Project Explorer
system.hdf
system.mss
system.mss

Project Explorer:
- hello_a53
- hello_a53_bsp
- hello_r5
- hello_r5_bsp
- ZynqMP_ZCU102_hw_platform

hello_r5_bsp Board Support Package

Target Information
This Board Support Package is compiled to run on the following target.
Hardware Specification: /home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/sdk/ZynqMP_ZCU102_hw_platform/system.hdf
Target Processor: psu_cortexr5_0

Operating System
Board Support Package OS.
Name: standalone
Version: 5.5
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_v5_5

Run Application

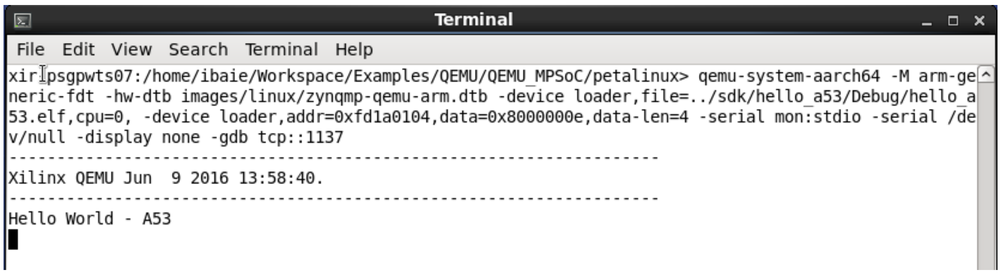
The demo applications can be run using the following commands. Take into account that all the cores start-up in a reset state, so in order to be able to execute the core have to be un-locked (Ref UG1169).

A53 Standalone Application

```
qemu-system-aarch64 -M arm-generic-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device
loader,file=../sdk/hello_a53/Debug/hello_a53.elf,cpu=0, -device
loader,addr=0xfd1a0104,data=0x8000000e,data-len=4 -serial mon:stdio -display none -gdb
tcp::1137
```

Table 3-4: **Single Transaction Unlock Arguments**

Argument	Command
A53-0	`-device loader addr=0xfd1a0104,data=0x8000000e,data-len=4`
A53-1	`-device loader addr=0xfd1a0104,data=0x8000000d,data-len=4`
A53-2	`-device loader addr=0xfd1a0104,data=0x8000000b,data-len=4`
A53-3	`-device loader addr=0xfd1a0104,data=0x80000007,data-len=4`
All A53	`-device loader addr=0xfd1a0104,data=0x80000000,data-len=4`



```
xir-psgpwts07:/home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/petalinux> qemu-system-aarch64 -M arm-ge
neric-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device loader,file=../sdk/hello_a53/Debug/hello_a
53.elf,cpu=0, -device loader,addr=0xfd1a0104,data=0x8000000e,data-len=4 -serial mon:stdio -serial /de
v/null -display none -gdb tcp::1137
--------------------------------------------------------------------
Xilinx QEMU Jun  9 2016 13:58:40.
--------------------------------------------------------------------
Hello World - A53
```

R5 Application

```
qemu-system-aarch64 -M arm-generic-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device
loader,file=../sdk/hello_r5/Debug/hello_r5.elf,cpu=4, -device
loader,addr=0xff5e023c,data=0x80008fde,data-len=4 -device
loader,addr=0xff9a0000,data=0x80000218,data-len=4 -serial mon:stdio -display none -gdb
tcp::1137
```

Table 3-5: **Cortex-R5 Registers and Commands**

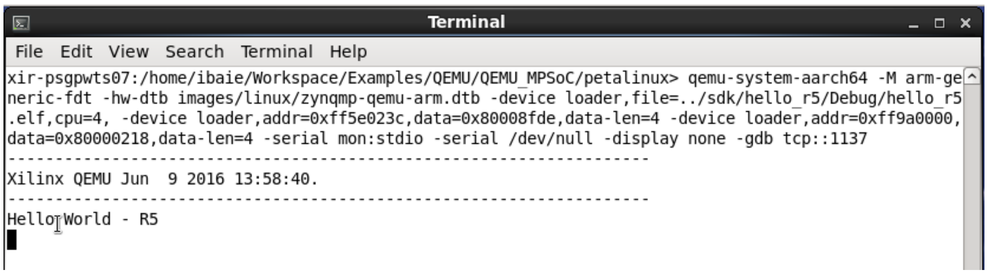
Register	Command

R5-0 (split mode)	`-device loader,addr=0xff5e023c,data=0x80008fde,data-len=4`
	`-device loader,addr=0xff9a0000,data=0x80000218,data-len=4`

R5-1 (split mode)	`-device loader,addr=0xff5e023c,data=0x80008fdd,data-len=4`
	`-device loader,addr=0xff9a0000,data=0x80000218,data-len=4`

Both R5 (split mode)	`-device loader,addr=0xff5e023c,data=0x80008fdc,data-len=4`
	`-device loader,addr=0xff9a0000,data=0x80000218,data-len=4`

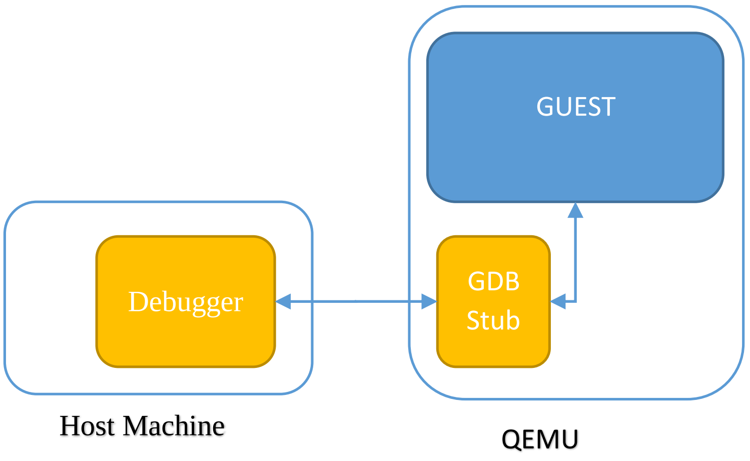
Lockstep Mode	`-device loader,addr=0xff5e023c,data=0x80008fde,data-len=4`



```
xir-psgpwts07:/home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/petalinux> qemu-system-aarch64 -M arm-ge
neric-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device loader,file=../sdk/hello_r5/Debug/hello_r5
.elf,cpu=4, -device loader,addr=0xff5e023c,data=0x80008fde,data-len=4 -device loader,addr=0xff9a0000,
data=0x80000218,data-len=4 -serial mon:stdio -serial /dev/null -display none -gdb tcp::1137
--------------------------------------------------------------------
Xilinx QEMU Jun  9 2016 13:58:40.
--------------------------------------------------------------------
Hello World - R5
```

Debug from XSDB

QEMU includes a built-in Debug server or GDB stub that can be used by a host to debug with a debugger (GDB client). In this case we connected the XSDB to QEMU stub and then use XSDB to load the application and run the example.

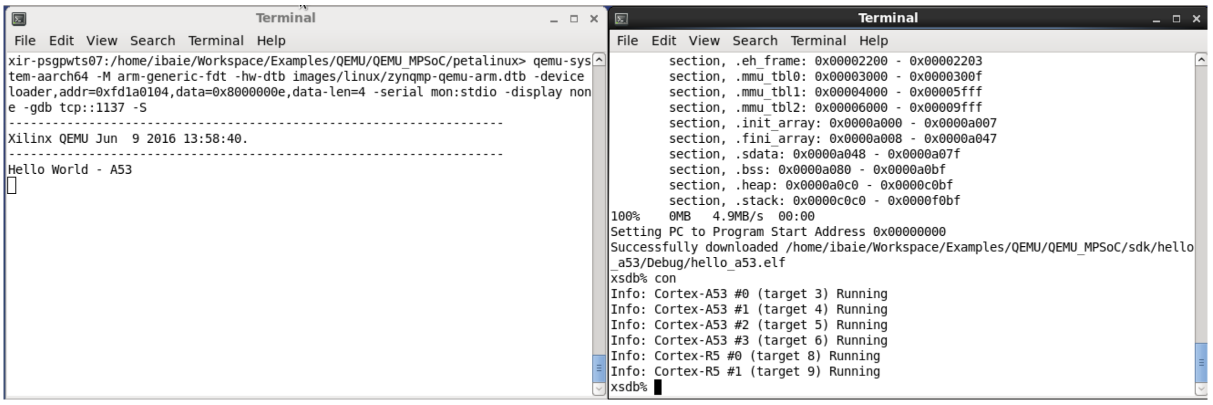


- Launch QEMU image:

```
qemu-system-aarch64 -M arm-generic-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device
loader,addr=0xfd1a0104,data=0x8000000e,data-len=4 -serial mon:stdio -display none -gdb
tcp::1137 -S
```

- Download the demo app and run in the target:

```
gdbremote connect localhost:1137
targets
targets 3
dow hello_a53.elf
con
```


Terminal
File Edit View Search Terminal Help
Terminal
File Edit View Search Terminal Help

```
xir-psgpwts07:/home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/petalinux> qemu-sys
tem-aarch64 -M arm-generic-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device
loader,addr=0xfd1a0104,data=0x8000000e,data-len=4 -serial mon:stdio -display non
e -gdb tcp::1137 -S
--------------------------------------------------------------------
Xilinx QEMU Jun  9 2016 13:58:40.
--------------------------------------------------------------------
Hello World - A53
```

```
        section, .eh_frame: 0x00002200 - 0x00002203
        section, .mmu_tbl0: 0x00003000 - 0x0000300f
        section, .mmu_tbl1: 0x00004000 - 0x00005fff
        section, .mmu_tbl2: 0x00006000 - 0x00009fff
        section, .init_array: 0x0000a000 - 0x0000a007
        section, .fini_array: 0x0000a008 - 0x0000a047
        section, .sdata: 0x0000a048 - 0x0000a07f
        section, .bss: 0x0000a080 - 0x0000a0bf
        section, .heap: 0x0000a0c0 - 0x0000c0bf
        section, .stack: 0x0000c0c0 - 0x0000f0bf
100%    0MB   4.9MB/s  00:00
Setting PC to Program Start Address 0x00000000
Successfully downloaded /home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/sdk/hello
_a53/Debug/hello_a53.elf
xsdb% con
Info: Cortex-A53 #0 (target 3) Running
Info: Cortex-A53 #1 (target 4) Running
Info: Cortex-A53 #2 (target 5) Running
Info: Cortex-A53 #3 (target 6) Running
Info: Cortex-R5 #0 (target 8) Running
Info: Cortex-R5 #1 (target 9) Running
xsdb%
```

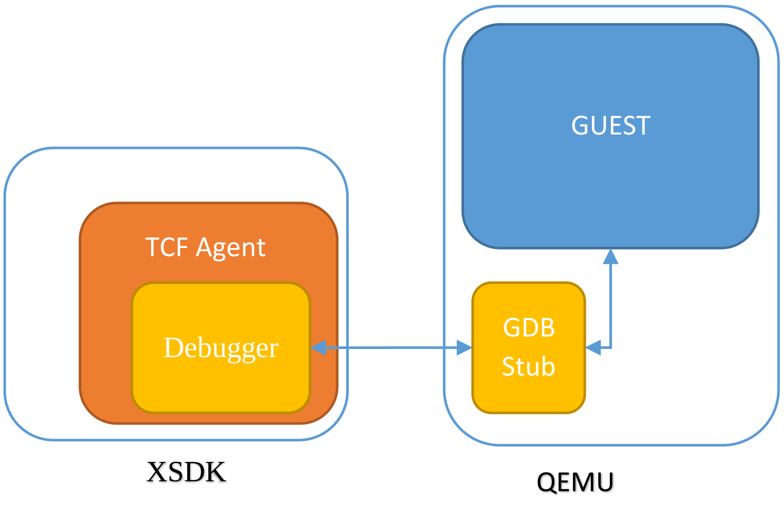
Using the XSDB we can do the following actions:

- Download and run a bare-metal application ELF
- Suspend and resume a processor
- Debug using breakpoints
- Read and write registers

Ref. UG1169 Chapter 4

Debug from SDK

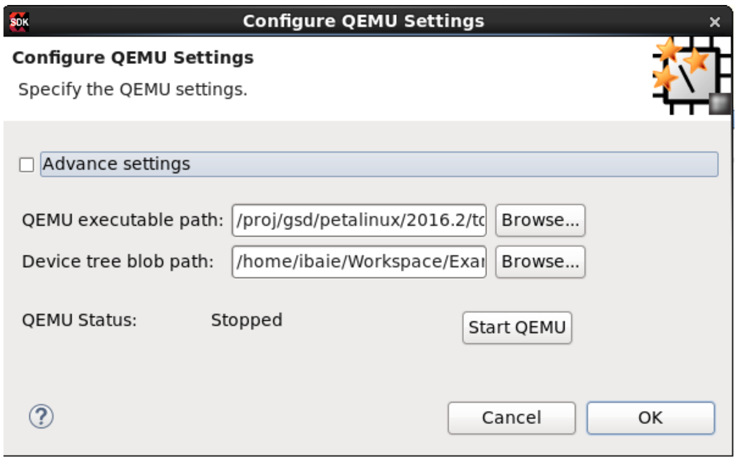
Debugging from the SDK debug session and tool is done by a TCF agent implemented by Xilinx and included in the system debugger. This TCF agent is connected to the GDB stub within QEMU in order to debug.



There are two ways to ways to run QEMU session in SDK, one is with the build in QEMU launcher and the second one is launching from a separate terminal and then connect to the QEMU guest from the SDK session.

Integrated Workflow

In the first approach we have to run the Configure QEMU Settings (Under Xilinx Tools tab). In the pop up box place the path to the QEMU executable (you can this path running which qemu-system-aarch64 command), and the device tree blob path from the Petalinux image. Then press Start QEMU to start the guest session.



Configure QEMU Settings

Specify the QEMU settings.

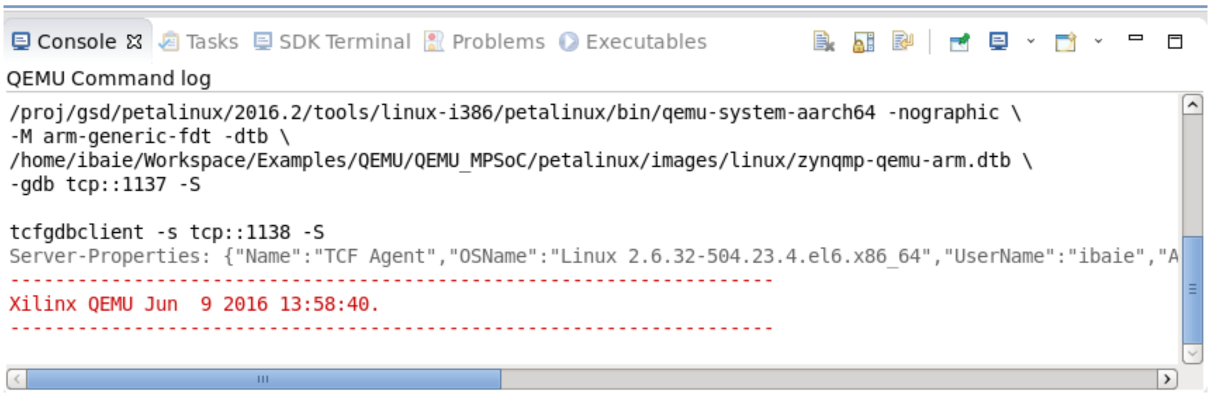
☐ Advance settings

QEMU executable path: /proj/gsd/petalinux/2016.2/tc

Device tree blob path: /home/ibaie/Workspace/Exai

QEMU Status: Stopped

This configuration has limited configurability and will run QEMU session with a fixed default options. You can check the configuration in the SDK console and also see how the TCF agent is run.

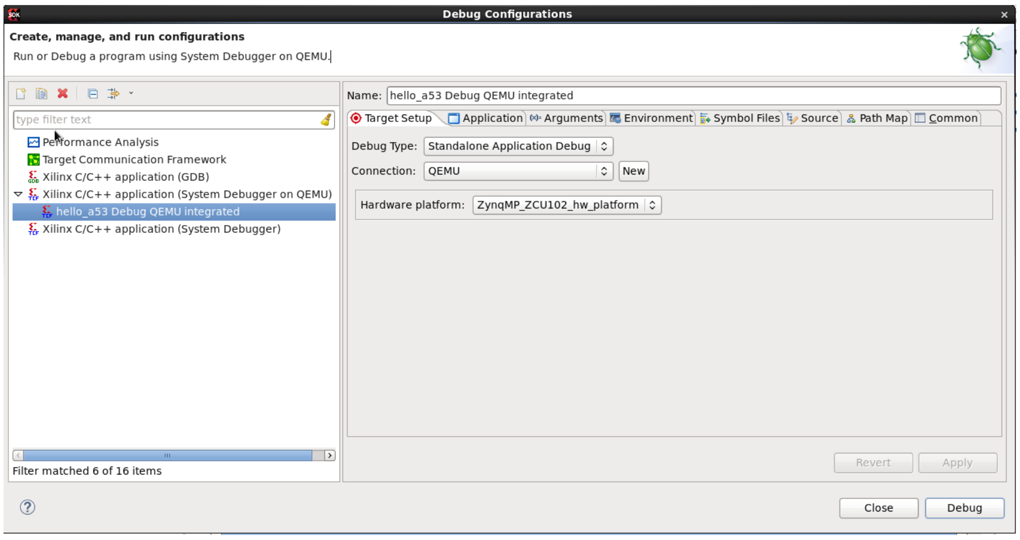


QEMU Command log

```
/proj/gsd/petalinux/2016.2/tools/linux-i386/petalinux/bin/qemu-system-aarch64 -nographic \
-M arm-generic-fdt -dtb \
/home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/petalinux/images/linux/zynqmp-qemu-arm.dtb \
-gdb tcp::1137 -S

tcfgdbclient -s tcp::1138 -S
Server-Properties: {"Name":"TCF Agent","OSName":"Linux 2.6.32-504.23.4.el6.x86_64","UserName":"ibaie","A
-----------------------------------------------------------------
Xilinx QEMU Jun  9 2016 13:58:40.
-----------------------------------------------------------------
```

Once QEMU session is running we can debug the applications created in the SDK using the System Debugger on QEMU pre-defined debug configuration.



Create, manage, and run configurations

Run or Debug a program using System Debugger on QEMU.

type filter text

- Performance Analysis
- Target Communication Framework
- Xilinx C/C++ application (GDB)
- Xilinx C/C++ application (System Debugger on QEMU)
  - hello_a53 Debug QEMU integrated
- Xilinx C/C++ application (System Debugger)

Filter matched 6 of 16 items

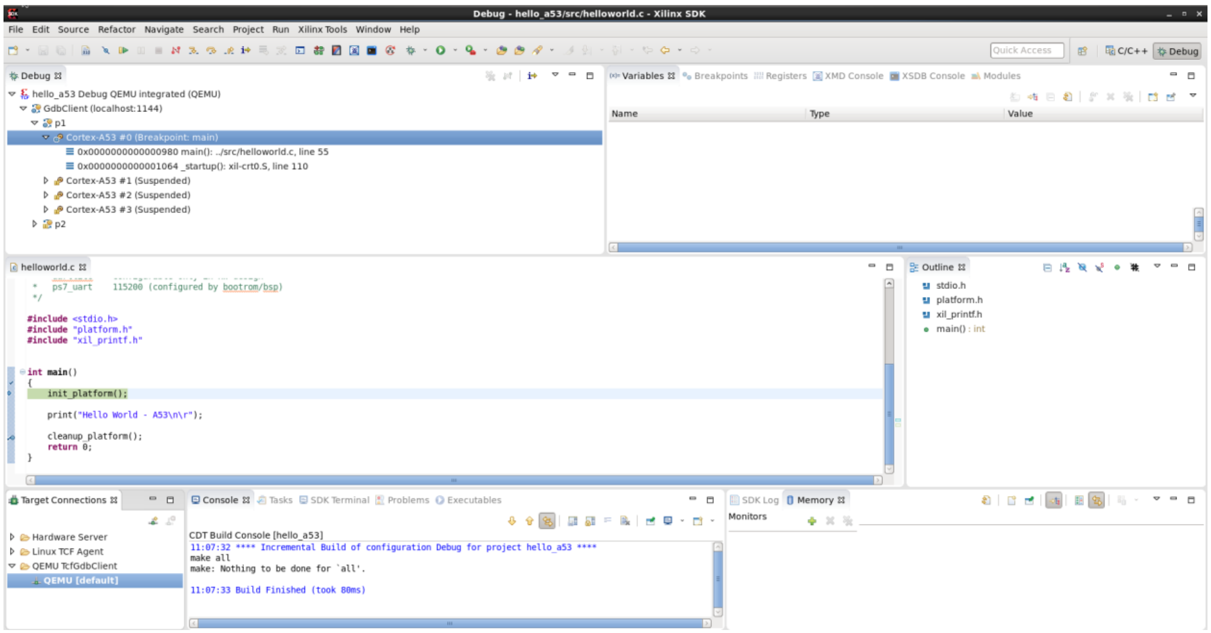
Name: hello_a53 Debug QEMU integrated

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

Debug Type: Standalone Application Debug

Connection: QEMU [New]

Hardware platform: ZynqMP_ZCU102_hw_platform



Debug - hello_a53/src/helloworld.c - Xilinx SDK

Debug

```
hello_a53 Debug QEMU integrated (QEMU)
  GdbClient (localhost:1144)
    p1
      Cortex-A53 #0 (Breakpoint: main)
          0x0000000000000980 main(): ../src/helloworld.c, line 55
          0x0000000000001064 _startup(): xil-crt0.S, line 110
      Cortex-A53 #1 (Suspended)
      Cortex-A53 #2 (Suspended)
      Cortex-A53 #3 (Suspended)
    p2
```

helloworld.c

```
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

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


int main()
{
    init_platform();

    print("Hello World - A53\n\r");

    cleanup_platform();
    return 0;
}
```

Target Connections

- Hardware Server
- Linux TCF Agent
- QEMU TcfGdbClient
  - QEMU [default]

Console

CDT Build Console [hello_a53]

```
11:07:32 **** Incremental Build of configuration Debug for project hello_a53 ****
make all
make: Nothing to be done for `all'.

11:07:33 Build Finished (took 80ms)
```

Custom Workflow

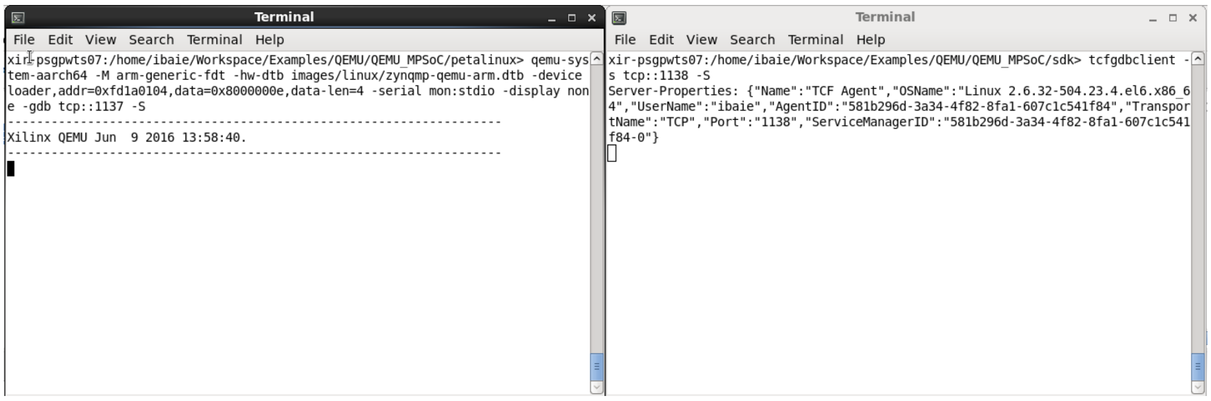
The second approach is launch the QEMU session in a different command line terminal and attach the SDK debug session to the running guest image. First step is to run both the QEMU guest image and TCF agent in different terminals.

- Launch QEMU image:

```
qemu-system-aarch64 -M arm-generic-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device
loader,addr=0xfd1a0104,data=0x8000000e,data-len=4 -serial mon:stdio -display none -gdb
tcp::1137 -S
```

- Launch TCF agent:

```
tcfbclient –s tcp::1138 -S
```



Terminal
File Edit View Search Terminal Help

```
xir-psgpwts07:/home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/petalinux> qemu-sys
tem-aarch64 -M arm-generic-fdt -hw-dtb images/linux/zynqmp-qemu-arm.dtb -device
loader,addr=0xfd1a0104,data=0x8000000e,data-len=4 -serial mon:stdio -display non
e -gdb tcp::1137 -S
-----------------------------------------------------------------
Xilinx QEMU Jun  9 2016 13:58:40.
-----------------------------------------------------------------
```

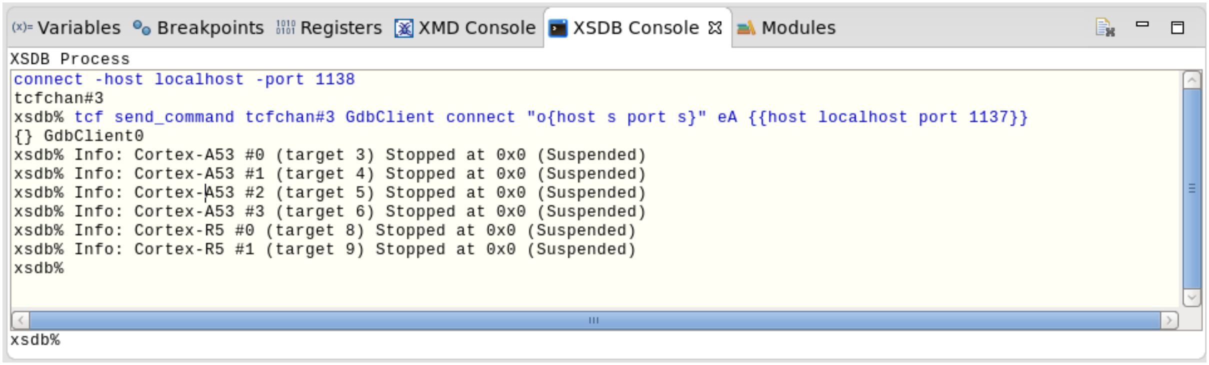
Terminal
File Edit View Search Terminal Help

```
xir-psgpwts07:/home/ibaie/Workspace/Examples/QEMU/QEMU_MPSoC/sdk> tcfgdbclient -
s tcp::1138 -S
Server-Properties: {"Name":"TCF Agent","OSName":"Linux 2.6.32-504.23.4.el6.x86_6
4","UserName":"ibaie","AgentID":"581b296d-3a34-4f82-8fa1-607c1c541f84","Transpor
tName":"TCP","Port":"1138","ServiceManagerID":"581b296d-3a34-4f82-8fa1-607c1c541
f84-0"}
```

Once both applications are running within SDK's XSDB terminal we connect the TCF client to the QEMU debugger, typing the following commands.

```
connect -host localhost -port 1138

tcf send_command tcfchan#3 GdbClient connect "o{host s port s}" eA {{host localhost port
1137}}
```

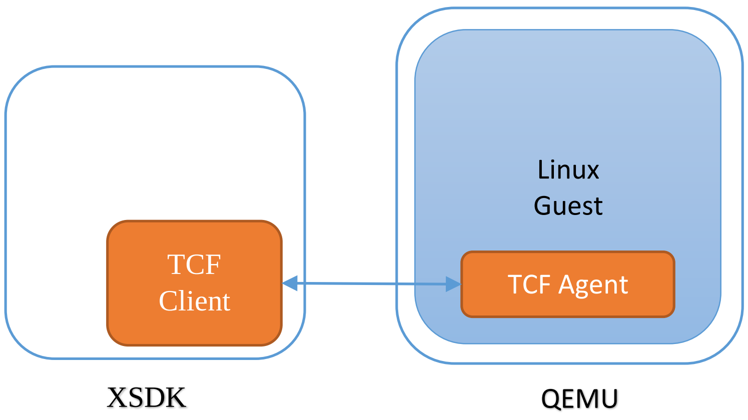


```
XSDB Process
connect -host localhost -port 1138
tcfchan#3
xsdb% tcf send_command tcfchan#3 GdbClient connect "o{host s port s}" eA {{host localhost port 1137}}
{} GdbClient0
xsdb% Info: Cortex-A53 #0 (target 3) Stopped at 0x0 (Suspended)
xsdb% Info: Cortex-A53 #1 (target 4) Stopped at 0x0 (Suspended)
xsdb% Info: Cortex-A53 #2 (target 5) Stopped at 0x0 (Suspended)
xsdb% Info: Cortex-A53 #3 (target 6) Stopped at 0x0 (Suspended)
xsdb% Info: Cortex-R5 #0 (target 8) Stopped at 0x0 (Suspended)
xsdb% Info: Cortex-R5 #1 (target 9) Stopped at 0x0 (Suspended)
xsdb%
xsdb%
```

Once this steps are done, we can use the XSDK debugger as done in the integrated demo.

Debug Linux Application

Debugging Linux applications with SDK is done by TCF (Target Communication Framework) without using GDB at all. The TCF agent within the Linux image provides the services accessed by XSDK debug tool peers.



The first step is ensure that our petalinux image includes the tcf-agent.

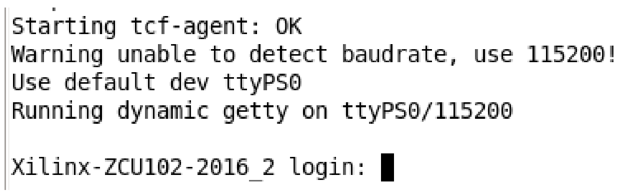
```
petalinux-config -c rootfs
```

- Filesystem Package -> base -> tcf-agent -> [*] tcf-agent

```
petalinux-build
```

Once the new image is build, we run the linux guest image. As the guest image network ports are not accessible by the host machine, the default TCF tcp port have to be redirected to any host available port.

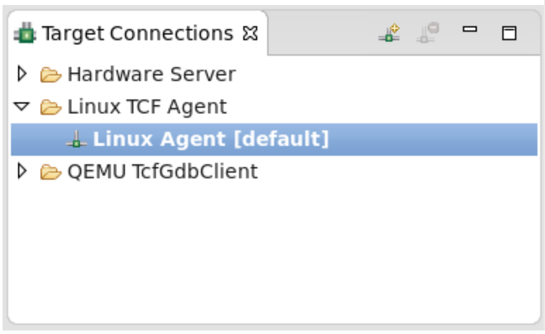
```
petalinux-boot --qemu --kernel --qemu-args "-redir tcp:1540:10.0.2.15:1534"
```

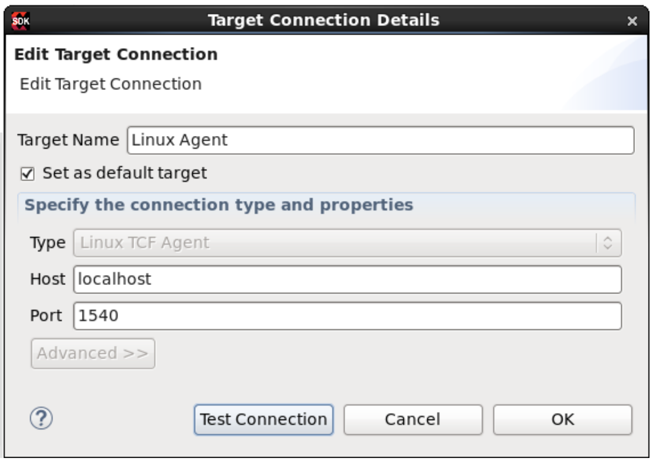


```
Starting tcf-agent: OK
Warning unable to detect baudrate, use 115200!
Use default dev ttyPS0
Running dynamic getty on ttyPS0/115200

Xilinx-ZCU102-2016_2 login:
```

Before creating the application we test the connection with the target TCF-agent from the XSDK. Under Target connections tab, select the default Linux Agent and modify the parameters to ensure that the connection test works.





Target Connection Details

Edit Target Connection

Edit Target Connection

Target Name: Linux Agent

☑ Set as default target

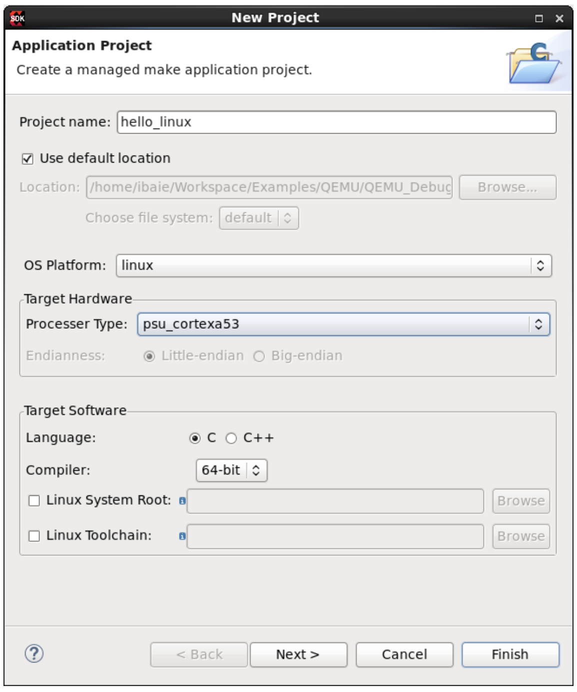
Specify the connection type and properties

Type: Linux TCF Agent

Host: localhost

Port: 1540

Create a new application within SDK targeting Linux OS Platform and psu_cortexa53 processor.


New Project

Application Project

Create a managed make application project.

Project name: hello_linux

☑ Use default location

Location: /home/ibaie/Workspace/Examples/QEMU/QEMU_Debug Browse...

Choose file system: default

OS Platform: linux

Target Hardware

Processer Type: psu_cortexa53

Endianness: (●) Little-endian ( ) Big-endian

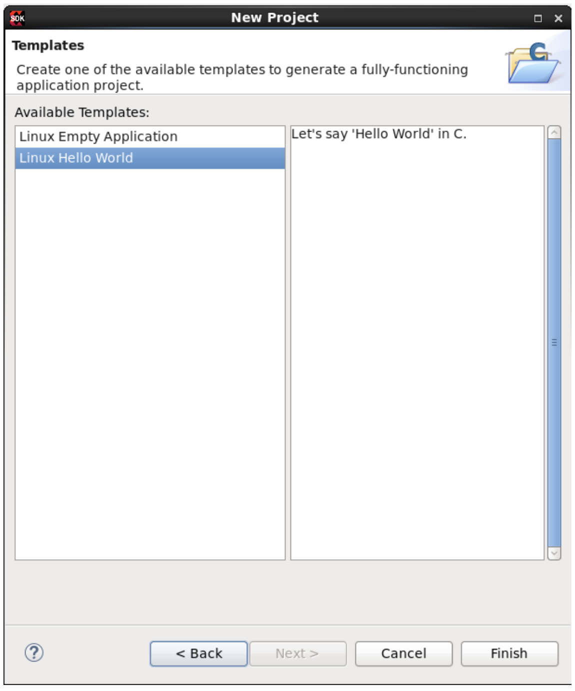
Target Software

Language: (●) C ( ) C++

Compiler: 64-bit

☐ Linux System Root: Browse

☐ Linux Toolchain: Browse


New Project

Templates

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

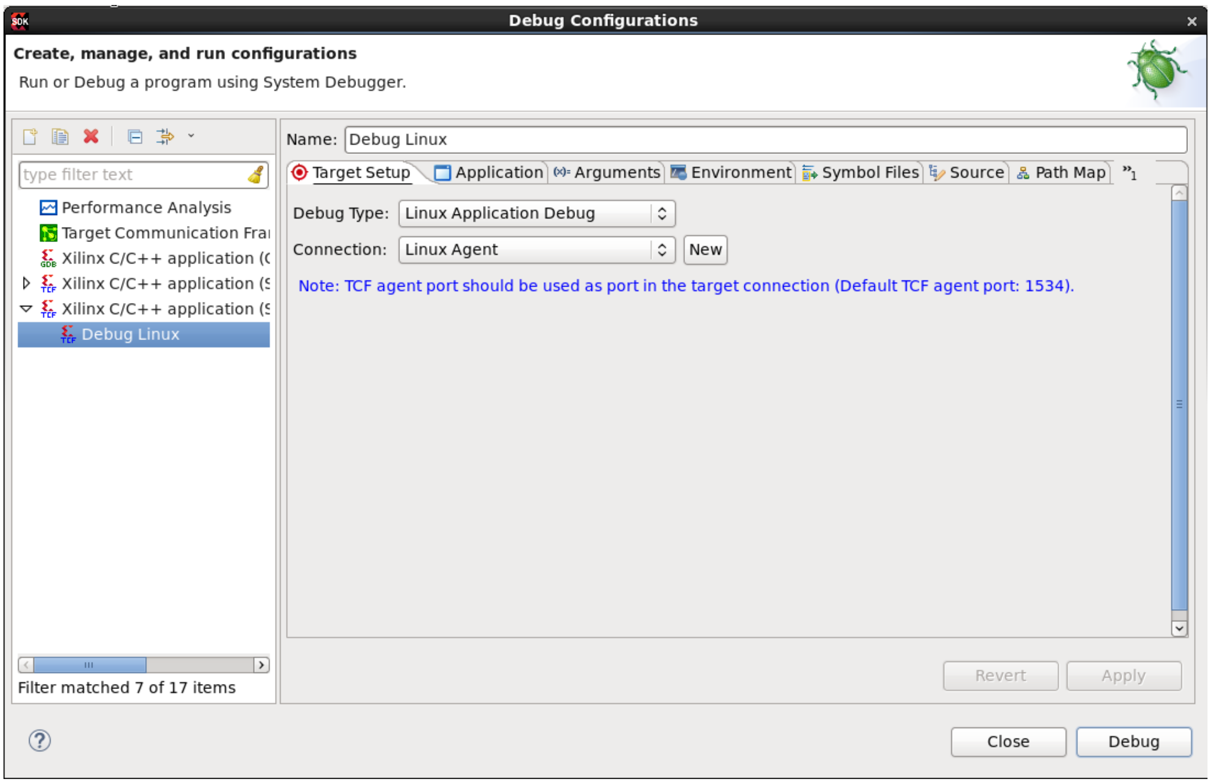
Available Templates:

Linux Empty Application

Linux Hello World

Let's say 'Hello World' in C.

Create a new debug configuration for the demo application, right click on the Linux application and select Debug As -> Debug Configurations. Create a new Xilinx C/C++ application (System Debugger) and select the following options and debug.


Debug Configurations

Create, manage, and run configurations

Run or Debug a program using System Debugger.

type filter text

- Performance Analysis
- Target Communication Frai
- Xilinx C/C++ application (
- Xilinx C/C++ application (S
- Xilinx C/C++ application (S
  - Debug Linux

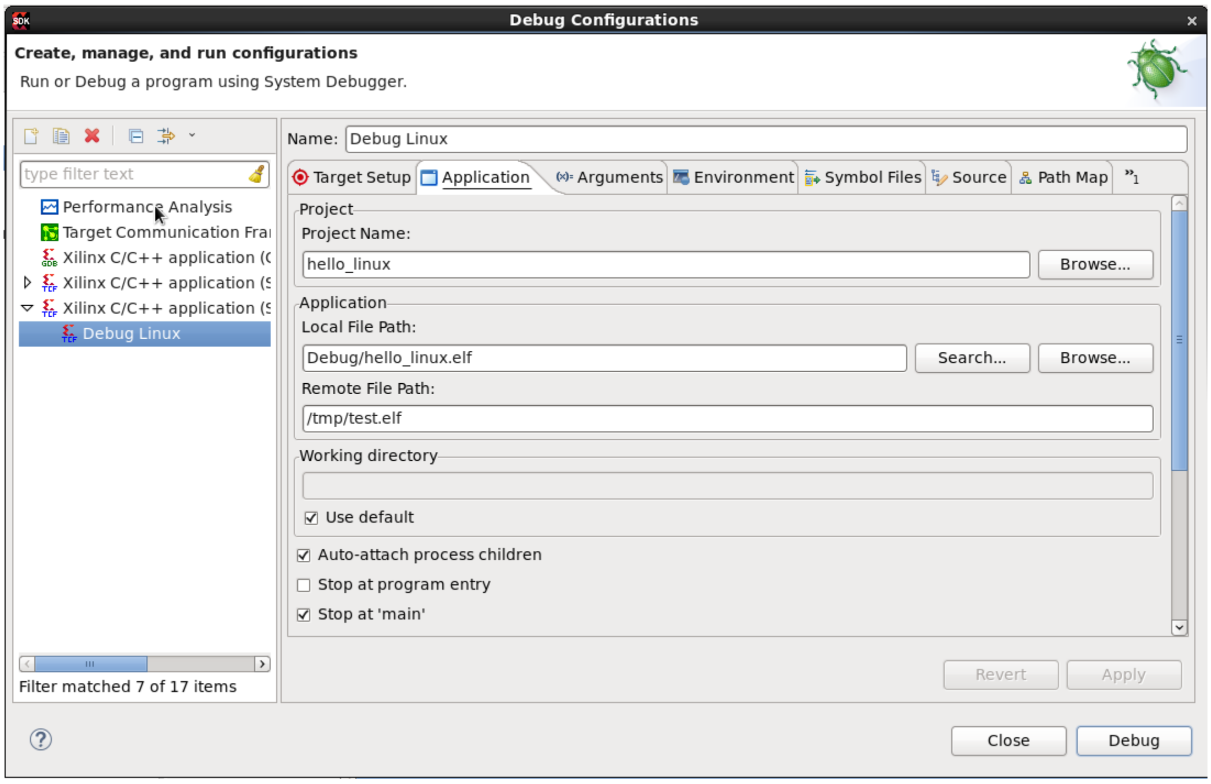
Filter matched 7 of 17 items

Name: Debug Linux

Debug Type: Linux Application Debug

Connection: Linux Agent New

Note: TCF agent port should be used as port in the target connection (Default TCF agent port: 1534).



Debug Configurations

Create, manage, and run configurations

Run or Debug a program using System Debugger.

type filter text

- Performance Analysis
- Target Communication Frai
- Xilinx C/C++ application (C
- Xilinx C/C++ application (S
- Xilinx C/C++ application (S
  - Debug Linux

Filter matched 7 of 17 items

Name: Debug Linux

Target Setup | Application | Arguments | Environment | Symbol Files | Source | Path Map | »1

Project

Project Name: hello_linux

Application

Local File Path: Debug/hello_linux.elf

Remote File Path: /tmp/test.elf

Working directory

[✓] Use default

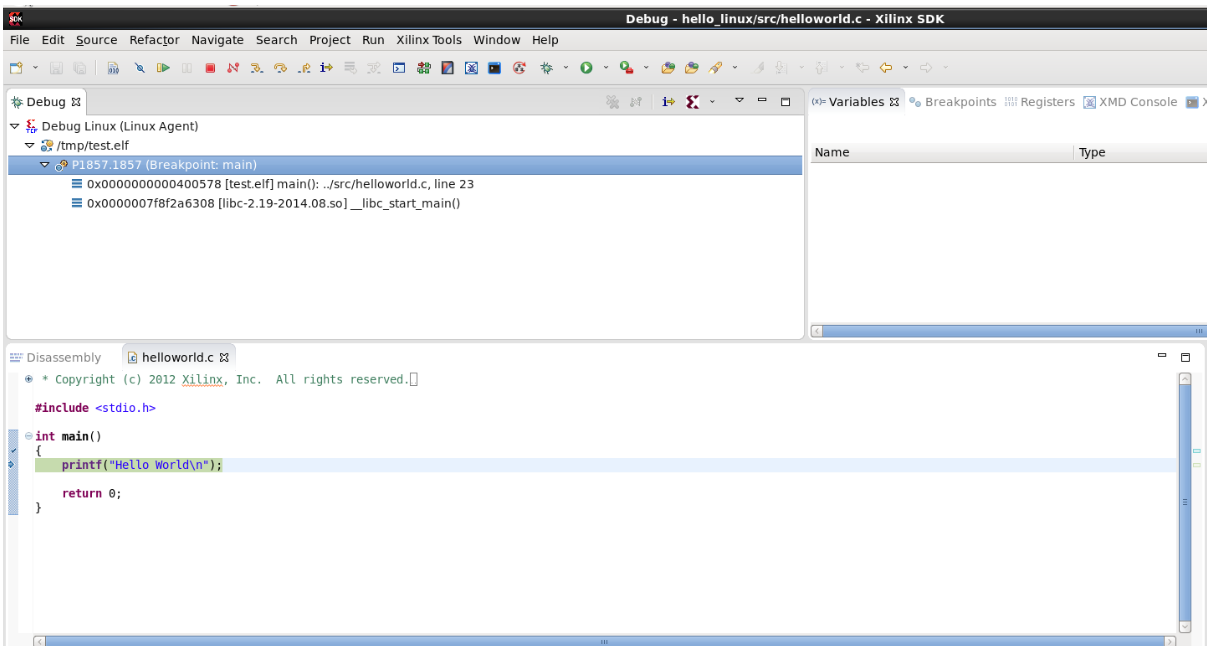
[✓] Auto-attach process children

[ ] Stop at program entry

[✓] Stop at 'main'

Revert | Apply

Close | Debug



Debug - hello_linux/src/helloworld.c - Xilinx SDK

File Edit Source Refactor Navigate Search Project Run Xilinx Tools Window Help
Debug
Variables
Breakpoints
Registers
XMD Console
Name
Type
Disassembly
helloworld.c

```
Debug Linux (Linux Agent)
  /tmp/test.elf
    P1857.1857 (Breakpoint: main)
      0x0000000000400578 [test.elf] main(): ../src/helloworld.c, line 23
      0x0000007f8f2a6308 [libc-2.19-2014.08.so] __libc_start_main()
```

```
 * Copyright (c) 2012 Xilinx, Inc.  All rights reserved.

#include <stdio.h>

int main()
{
    printf("Hello World\n");

    return 0;
}
```