



XAPP1221 (v1.0) January 12, 2015

Using ENEA OSE BSP for the Zynq-7000 AP SoC

Author: Kester Aernoudt

Summary

This application note is intended to be a getting started guide for new users of ENEA OSE BSP on the Zynq®-7000 AP SoC. The document contains the following sections:

- [Building OSE for the Zynq-7000 AP SoC](#) goes over the steps to download, configure and build the OSE BSP, as well as how to boot the device and set up the remote debugging capabilities using ENEA Optima tools.
- [Building and Debugging an OSE Application for the Zynq-7000 AP SoC](#) explains how to create a simple application running within OSE, and then sets up the debugging feature of the ENEA tools, and explains how to perform basic debugging of OSE running on a Zynq-7000 AP SoC.
- [Accessing a Peripheral in the Programmable Logic](#) goes over the configuration needed to access a peripheral in the programmable logic of the Zynq-7000 SoC, showing the ability for ENEA Optima tools to develop applications and drivers targeting custom IP instantiated within the Zynq-7000 AP SoC programmable logic.

You can download the [reference design files](#) for this application note from the Xilinx® website. For detailed information about the design files, see [Reference Design](#).

Hardware and Software Requirements

Software Requirements

- Vivado® Design Suite, 2014.2.
- Xilinx SDK 2014.2
- ENEA OSE5.7
- TFTP server application, such as tftpd64

Hardware Requirements

- Xilinx ZC702 Development Board

Building OSE for the Zynq-7000 AP SoC

Installing OSE

The installation of OSE for the Zynq-7000 AP SoC is split into different components which are installed separately. The required components are:

- The OSE kernel
- The BSP for the Zynq-7000 AP SoC device

Optionally, but used for this application note, the ENEA development environment Optima can be installed as well. Optima is an Eclipse based environment that allows you to create, build, and debug the OSE kernel and modules, and also allows you monitor and analyze running targets, all done over Ethernet.

OSE Kernel

This image is typically named something similar to `ose5_7_arm-BLXXXXXX_BLXXXXXX-image.zip`, meaning a maintenance build of the OSE 5.7 version. The installation wizard walks you through a couple of steps in which you will be asked to enter the license key and installation path. The suggestion is to use the default installation path, and to select **Typical installation** when selecting the packages to install.

All documentation for this BSP can be found in the `<OSE installation path>/doc` folder.

BSP for the Zynq-7000 AP SoC Device

Similar to the OSE kernel itself, the Zynq-7000 BSP has to be installed using an installation wizard. The image itself is provided as an archive, similar to `bsp_zynq7020_refsys_OSE5.7.1_BLXXXXXX_BLXXXXXX.zip`. For this application note, the version BL770298 is used.

The default folder for the installation of the Zynq-7000 BSP is `C:\Enea\BSP_Zynq7020_OSE5.7`, and the Typical Installation can be used, which will install all the necessary packages and documentation.

Optima

As a last step, ENEA Optima IDE is installed, which will allow you to debug a running kernel and its modules. Again, the default location (`C:\Enea\Optima2.8`) and Typical Installation provide you with all you need to target and debug a Zynq-7000 device.

Once installation is complete, a full Start Menu will be populated with the member elements, as shown in [Figure 1](#).



Figure 1: Start Menu

Configuring OSE Build Environment

After installation, the environment to build the kernel and run the tools has to be configured. This is done by modifying some makefiles, and creating a script to modify the environment before launching the tools.

Makefile Modifications

The OSE reference BSP comes with a makefile based build environment. It can also be integrated in an existing build environment, but this application note will use the provided system. To do so, some modifications have to be made for the exact environment and installation parameters used.

The reference BSP is created for the ZC702 board, and can be found in the `<Zynq BSP install path>/refsys` folder. Within this folder, the `environment.mk` file is the global configuration file for building the project. This file contains settings such as, but not limited to, Include Paths and Library Paths, along with Host and Target platform configuration details, etc.

However, two items need to be changed. Therefore, open the `environment.mk` file from the `<Zynq BSP install path>/refsys` folder and make the following modifications:

1. The `OSEROOT` variable needs to be set correctly. This is not needed when installing the Zynq-7000 BSP within the OSE base directory, but when installing multiple platforms BSPs, or when (as in this application note) installing the BSP outside of the OSE base folder, you need to configure the correct path to the sources of the OSE kernel. Look for the `OSEROOT` variable, and modify the path to reflect the installed environment:

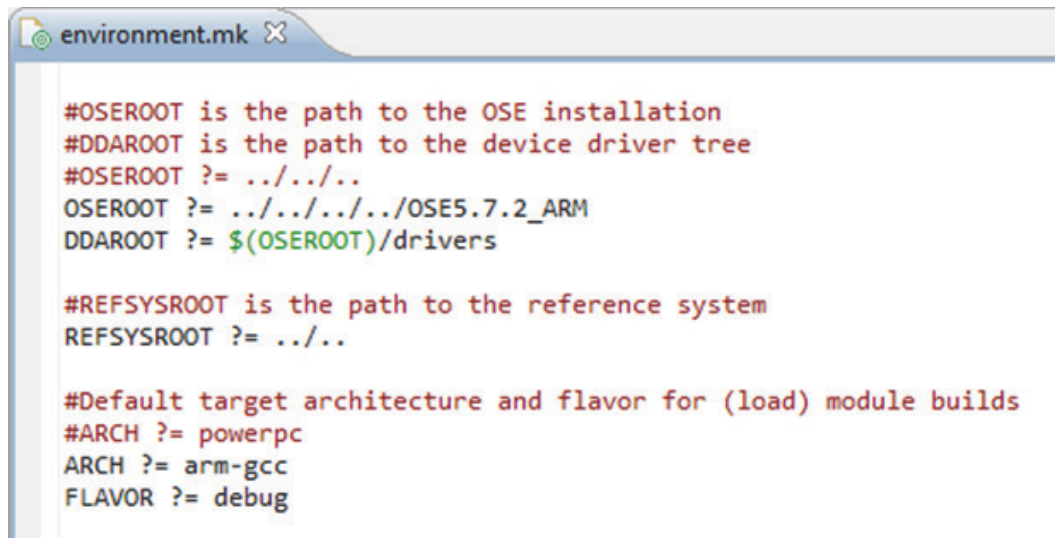
```
OSEROOT ?= ../../../../OSE5.7.2_ARM
```

The `OSEROOT` variable has to point to the installation folder for the OSE kernel itself, relative to the build folder of the current project, which for a default installation would be `<Zynq BSP Install Path>\refsys\rtose\zynq7020`.

- The correct platform needs to be selected. The default platform in the OSE 5.7.2 package was PowerPC® processor, which you need to change to `arm-gcc`. Look for the lines which contain the ARCH specification (see [Figure 2](#)), and modify or add an extra line underneath:

```
ARCH ?= arm-gcc
```

The ARCH variable configures the build system to use the correct cross compilation toolchain. This modification is not strictly needed when using the command line `make` utility, but is required when using the Optima Development Environment.



```
environment.mk
#OSEROOT is the path to the OSE installation
#DDAROOT is the path to the device driver tree
#OSEROOT ?= ../../..
OSEROOT ?= ../../../../OSE5.7.2_ARM
DDAROOT ?= $(OSEROOT)/drivers

#REFSYSROOT is the path to the reference system
REFSYSROOT ?= ../../

#Default target architecture and flavor for (load) module builds
#ARCH ?= powerpc
ARCH ?= arm-gcc
FLAVOR ?= debug
```

x1221_02_093014

Figure 2: `environment.mk` Modifications

Environment Setup

The environment has to be configured for all tools to work correctly. The `LM_LICENSE_FILE` environmental variable has to be set, and point to the correct license file. Also, if not configured globally, the ENEA Cygwin path has to be added to the `PATH` environmental variable:

For convenience, create a batch file on your desktop with the following content:

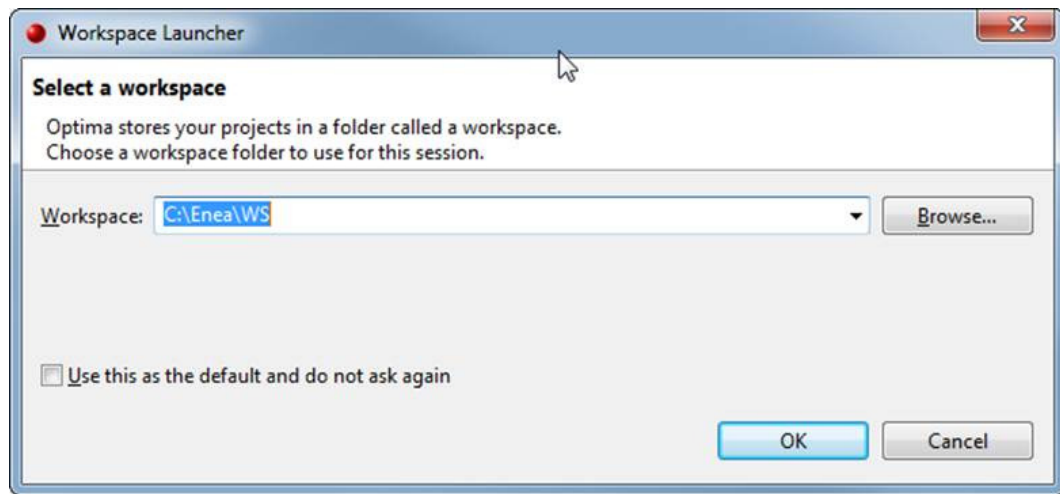
```
set LM_LICENSE_FILE=C:\Enea\License_keys_SFK_TOOLSEVAL.TXT
set PATH=%PATH%;C:\Enea\OSE5.7.2_ARM\cygwin\bin\
C:\Enea\Optima2.8\optima_win32\eclipse.exe
```

This script will launch Optima with the correct path and license file in the environment.

Building the OSE Kernel for Zynq-7000 AP SoC Devices

The installation of ENEA OSE comes with a Cygwin environment which can be used to compile the kernel and modules. Additionally, the Optima IDE can also be used. For this application note, the Optima IDE based approach will be used.

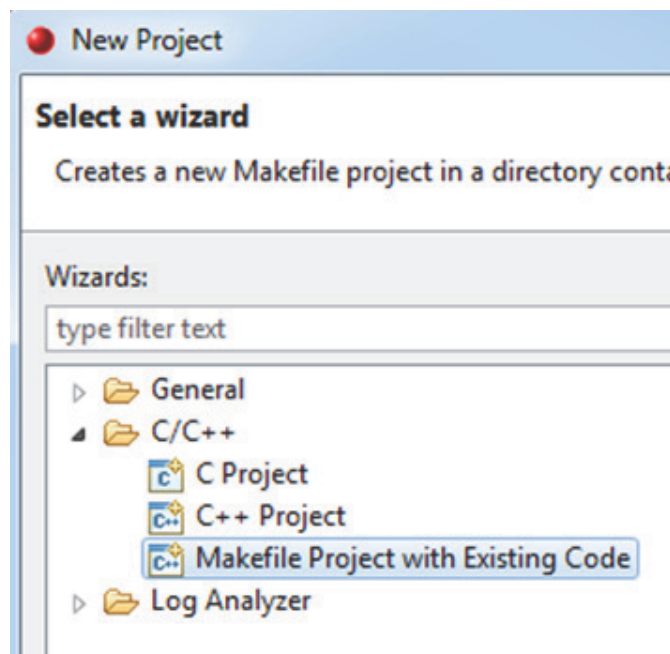
1. As a first step, a new workspace should be created, as shown in Figure 3.



x1221_03_093014

Figure 3: **Workspace Launcher**

2. Now create a new Makefile based project in our Workspace, referencing the OSE kernel sources:
 - a. **Select File -> New -> Project**
 - b. Select the **Makefile Project with Existing Code** template, as shown in Figure 4.

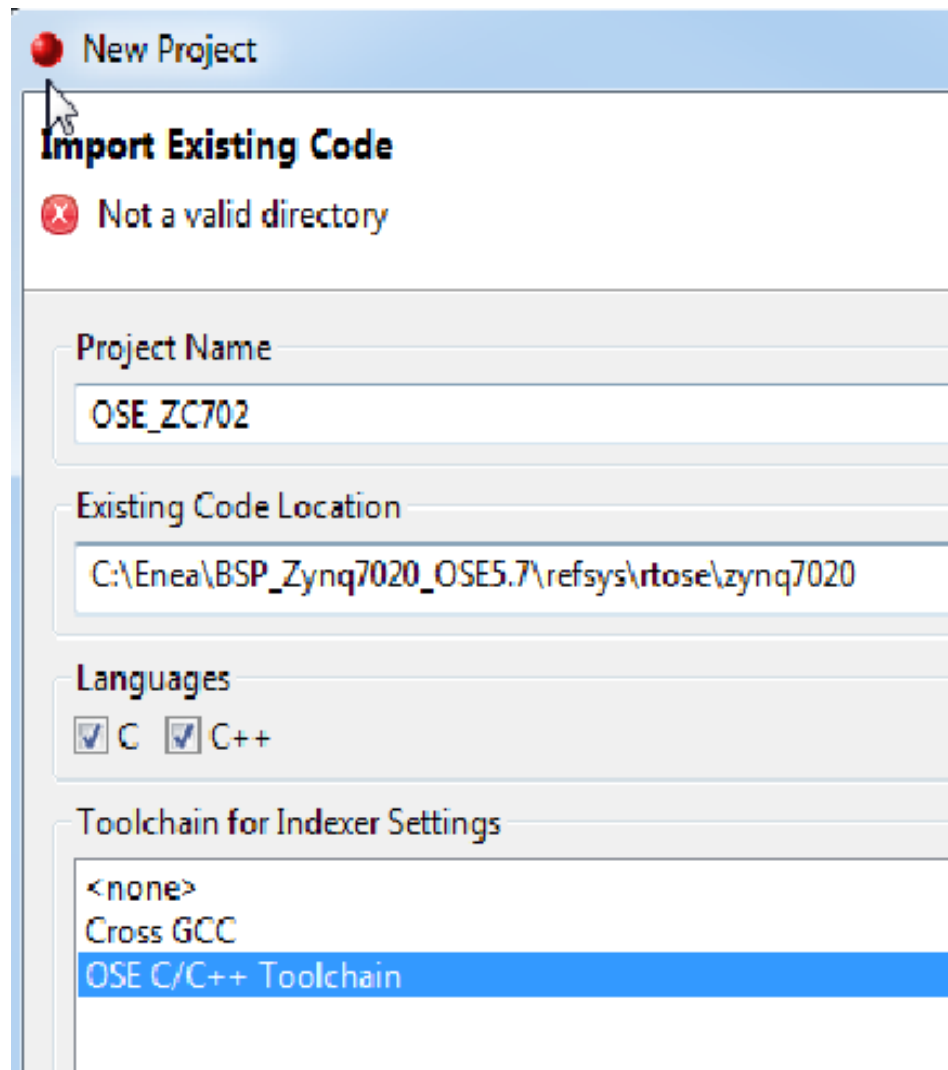


x1221_04_093014

Figure 4: **New Project Wizard**

3. Set the **Project Name** to OSE_ZC702
4. Set the **Existing Code Location** to <Zynq BSP Install Path>/refsys/rtose/zynq7020

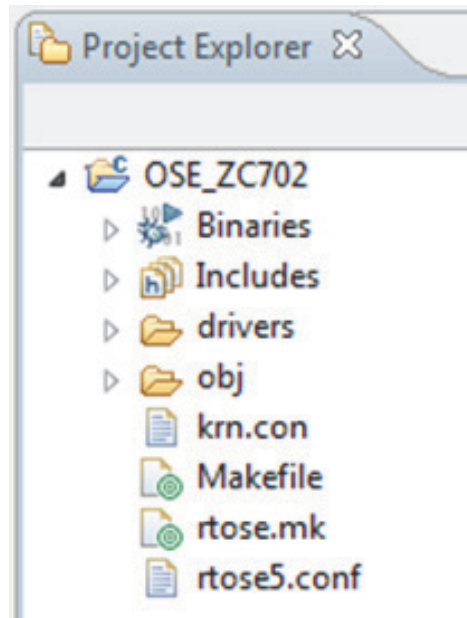
5. Select the OSE C/C++ Toolchain, as shown in Figure 5.



x1221_05_093014

Figure 5: New Project Dialog Box

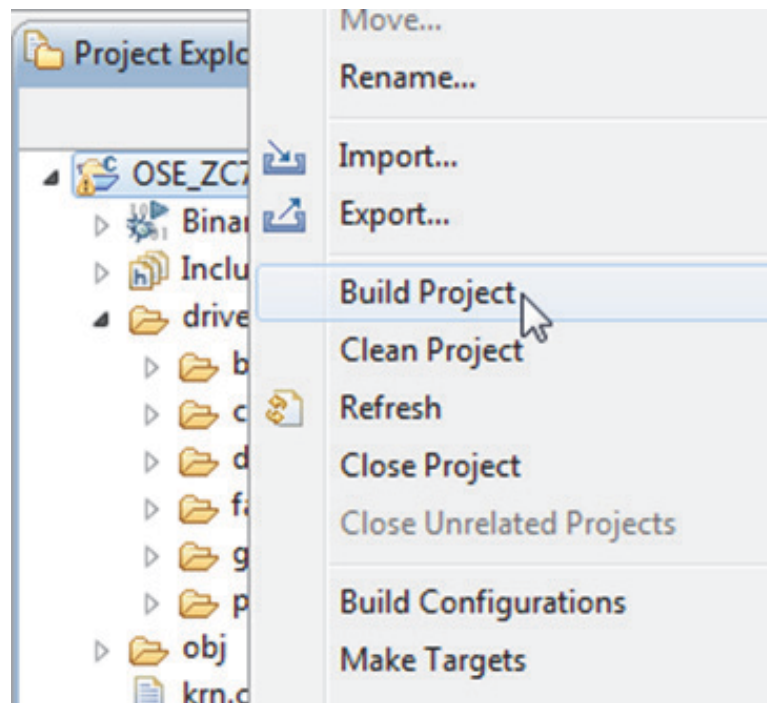
This will create the OSE_ZC702 project, which is visible in the **Project Explorer** tab, as shown in [Figure 6](#).



x1221_06_093014

Figure 6: **Project Explorer**

6. Build this project by right-clicking on the OSE_ZC702 project, and selecting **Build Project**, as shown in [Figure 7](#).



x1221_07_093014

Figure 7: **Build Project**

This will build both release and debug versions of the OSE kernel, which will be made visible in the GUI under the Binaries section in the Project Explorer, and are located in the <Zynq BSP Install Path>/refsys/rtose/zynq7020/obj/rtose_debug and <Zynq BSP Install Path>/refsys/rtose/zynq7020/obj/rtose_release folders. See [Figure 8](#).

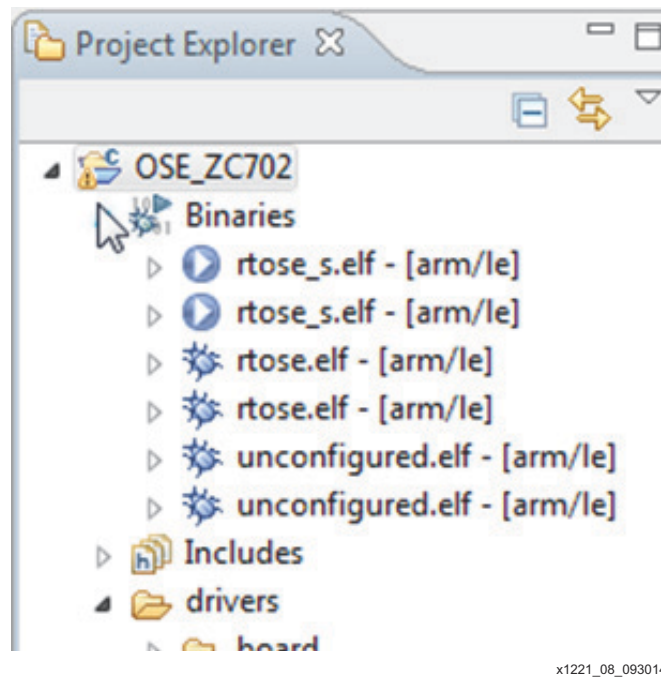


Figure 8: Binaries

Running OSE on Zynq-7000 AP SoC Devices

As described in the *Zynq-7000 All Programmable SoC Technical Reference Manual (UG585)* [Ref 1], Zynq-7000 AP SoC can use a selection of its hardened peripherals as a primary boot interface under initial control of the Zynq-7000 AP SoC BootROM. However, the process remains consistent in each and the need for a boot image containing a Boot header, First Stage Bootloader, Bitstream (optional) and a second stage bootloader/application is required at all times.

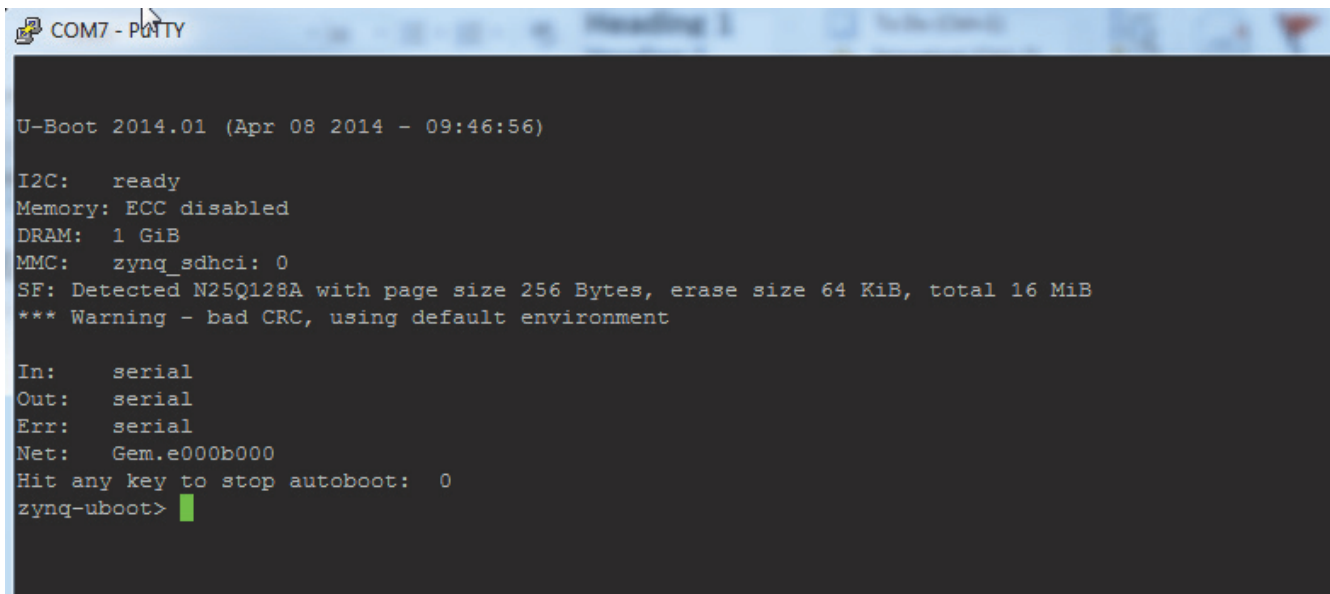
For the purpose of this application note, it is assumed that a prebuilt `boot.bin` image containing FSBL and a precompiled `u-boot.elf` has been downloaded and is available to the developer. The `boot.bin` image found on our Embedded Wiki [Ref 2] provides tftp boot capability, and the Vivado Design Suite 2014.2 version has been shown to work with this application note.

1. Download the `boot.bin` image and place it on a formatted SD Card.
2. Insert the SD card into the board.
3. Boot the platform with all appropriate cables connected.
4. Open a serial console on the correct COM port and configure the terminal settings:
 - a. Set the Baud rate to 115200.

- b. Set Data to 8-bit.
- c. Set Parity to none.
- d. Set Stop to 1 bit.
- e. Set Flow control to none.

During boot, you should see the u-boot boot messages, which will show a download counter to start autoboot.

- f. Select any key to stop this autoboot, as the default configuration is to launch Linux. See [Figure 9](#).



```

COM7 - PTY
U-Boot 2014.01 (Apr 08 2014 - 09:46:56)

I2C:   ready
Memory: ECC disabled
DRAM:  1 GiB
MMC:   zynq_sdhci: 0
SF: Detected N25Q128A with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
zynq-u-boot>
  
```

x1221_09_093014

Figure 9: U-boot Console

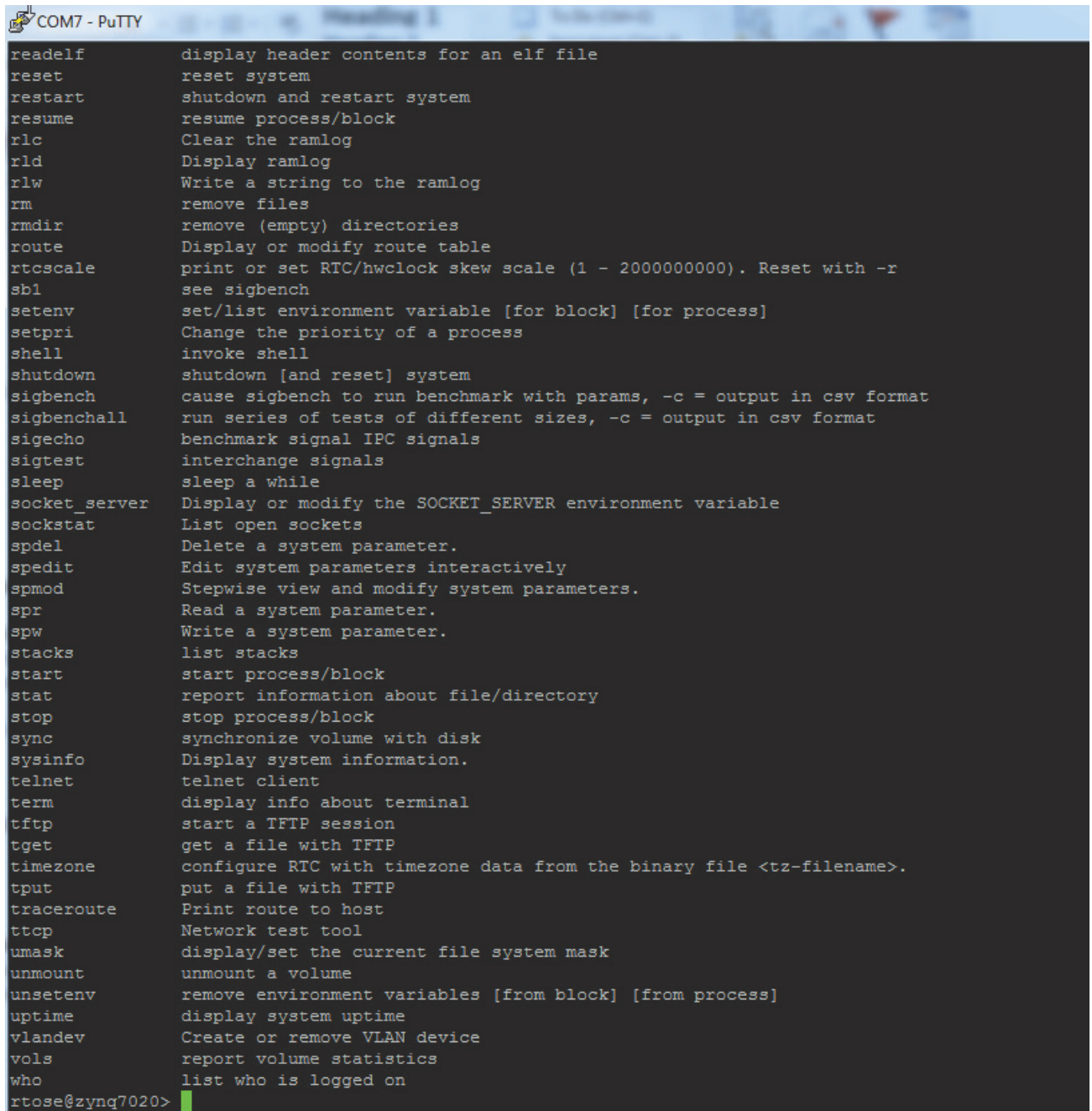
Now that you are in the u-boot prompt, you can use different methods to load the actual OSE kernel image. For this application note, you will use a TFTP server on the host machine, and configure u-boot to fetch the kernel over TFTP.

5. Launch the tftp64 application, and store the `rtose.bin` (<Zynq BSP install path>\refsys\rtose\zynq7020\obj\rtose_debug\rtose.bin) file in the root of the tftp shared folder.
6. Configure the tftp host machine to have a fixed IP address (192.168.1.1, for example), and start the tftp server. Make sure that your local firewall allows tftp to be used on your machine (or disable the firewall temporarily).
7. Now, from the u-boot prompt, issue the following commands to fetch the OSE kernel and boot it on the Zynq-7000 AP SoC:

```

setenv ipaddr 192.168.1.111; setenv serverip 192.168.1.1; setenv tftpblocksize 512;
tftpboot 0x00050000 rtose.bin
go 0x00050000
  
```


8. Type **help** for an overview (see Figure 11) of the available commands that are supported by this kernel.



```

COM7 - PuTTY
readelf      display header contents for an elf file
reset        reset system
restart      shutdown and restart system
resume       resume process/block
rlc          Clear the ramlog
rld          Display ramlog
rlw          Write a string to the ramlog
rm           remove files
rmdir        remove (empty) directories
route        Display or modify route table
rtcscal     print or set RTC/hwclock skew scale (1 - 2000000000). Reset with -r
sb1         see sigbench
setenv       set/list environment variable [for block] [for process]
setpri       Change the priority of a process
shell        invoke shell
shutdown     shutdown [and reset] system
sigbench     cause sigbench to run benchmark with params, -c = output in csv format
sigbenchall  run series of tests of different sizes, -c = output in csv format
sigecho      benchmark signal IPC signals
sigtest      interchange signals
sleep        sleep a while
socket_server Display or modify the SOCKET_SERVER environment variable
sockstat     List open sockets
spdel        Delete a system parameter.
spedit       Edit system parameters interactively
spmmod       Stepwise view and modify system parameters.
spr          Read a system parameter.
spw          Write a system parameter.
stacks       list stacks
start        start process/block
stat         report information about file/directory
stop         stop process/block
sync         synchronize volume with disk
sysinfo      Display system information.
telnet       telnet client
term         display info about terminal
tftp         start a TFTP session
tget         get a file with TFTP
timezone     configure RTC with timezone data from the binary file <tz-filename>.
tput         put a file with TFTP
traceroute   Print route to host
ttcp         Network test tool
umask        display/set the current file system mask
unmount      unmount a volume
unsetenv     remove environment variables [from block] [from process]
uptime       display system uptime
vlandev      Create or remove VLAN device
vols         report volume statistics
who          list who is logged on
rtose@zynq7020>

```

x1221_11_093014

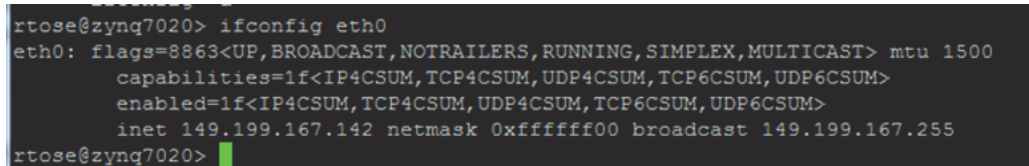
Figure 11: OSE Help

9. If there is a DHCP server in the network, no further configuration is needed. Otherwise, use the following command to set the IP address of the target:

```
ifconfig eth0 192.168.1.111
```

The actual IP address can be shown by entering the following (see [Figure 12](#)):

```
ifconfig eth0
```



```
rtose@zynq7020> ifconfig eth0
eth0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    capabilities=1f<IP4CSUM,TCP4CSUM,UDP4CSUM,TCP6CSUM,UDP6CSUM>
    enabled=1f<IP4CSUM,TCP4CSUM,UDP4CSUM,TCP6CSUM,UDP6CSUM>
    inet 149.199.167.142 netmask 0xffffffff broadcast 149.199.167.255
rtose@zynq7020>
```

x1221_12_093014

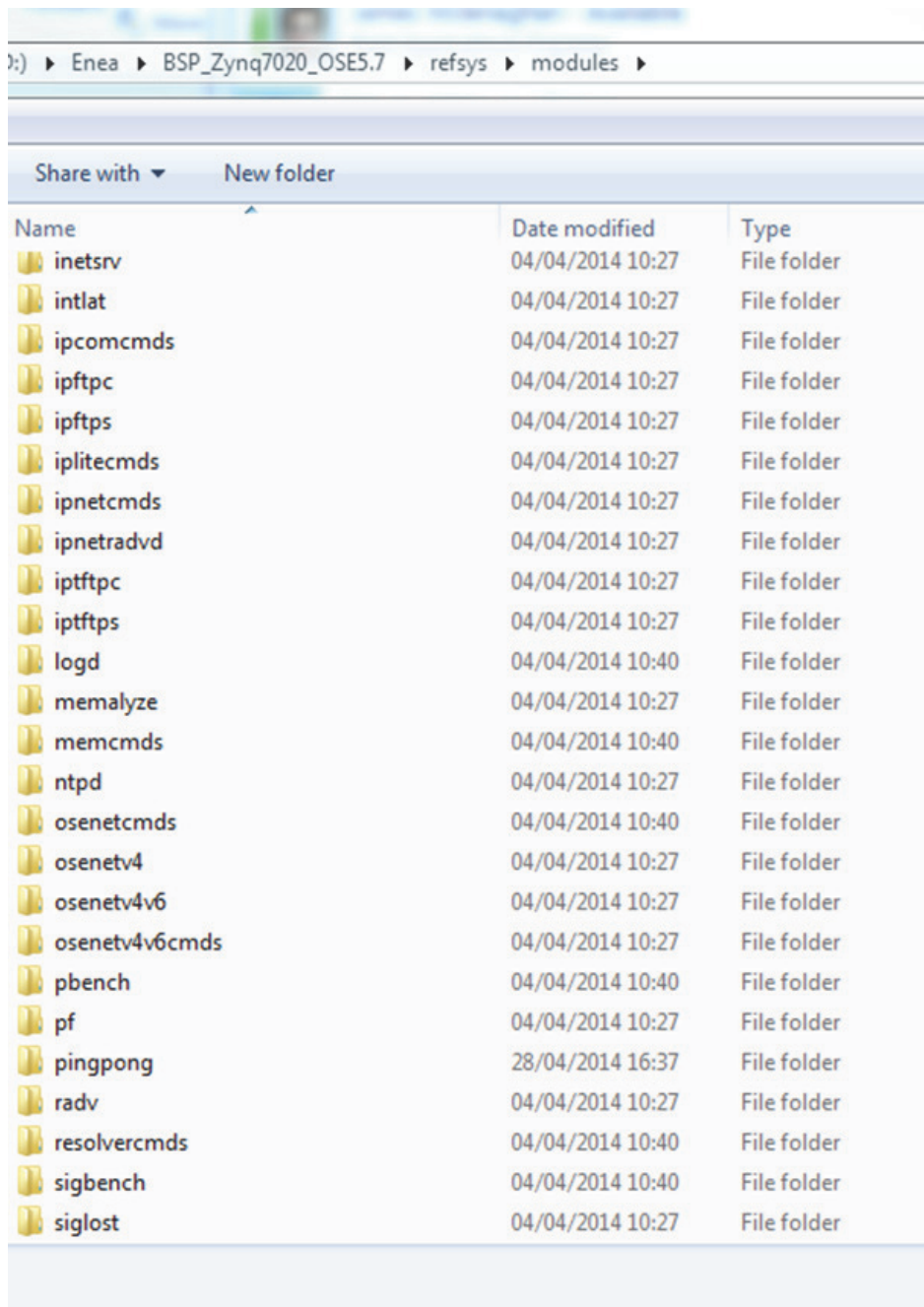
Figure 12: Network Configuration when using DHCP

Building and Debugging an OSE Application for the Zynq-7000 AP SoC

Building an OSE Module

The OSE BSP for Zynq-7000 AP SoC devices comes with several modules that can be added to the OSE kernel image or loaded at run time. See the *ENEA OSE Device Drivers User's Guide* [\[Ref 3\]](#) for more information. They can be used as is, or some are meant as an example of how to create and use modules within OSE. Such modules are located in the BSP install folder at

<Zynq BSP install path>/refsys/modules. See [Figure 13](#).



Name	Date modified	Type
inetsrv	04/04/2014 10:27	File folder
intlat	04/04/2014 10:27	File folder
ipcomcmds	04/04/2014 10:27	File folder
ipftpc	04/04/2014 10:27	File folder
ipftps	04/04/2014 10:27	File folder
iplitecmds	04/04/2014 10:27	File folder
ipnetcmds	04/04/2014 10:27	File folder
ipnetradvd	04/04/2014 10:27	File folder
iptftpc	04/04/2014 10:27	File folder
iptftps	04/04/2014 10:27	File folder
logd	04/04/2014 10:40	File folder
memalyze	04/04/2014 10:27	File folder
memcmds	04/04/2014 10:40	File folder
ntpd	04/04/2014 10:27	File folder
osenetcmds	04/04/2014 10:40	File folder
osenetv4	04/04/2014 10:27	File folder
osenetv4v6	04/04/2014 10:27	File folder
osenetv4v6cmds	04/04/2014 10:27	File folder
pbench	04/04/2014 10:40	File folder
pf	04/04/2014 10:27	File folder
pingpong	28/04/2014 16:37	File folder
radv	04/04/2014 10:27	File folder
resolvercmds	04/04/2014 10:40	File folder
sigbench	04/04/2014 10:40	File folder
siglost	04/04/2014 10:27	File folder

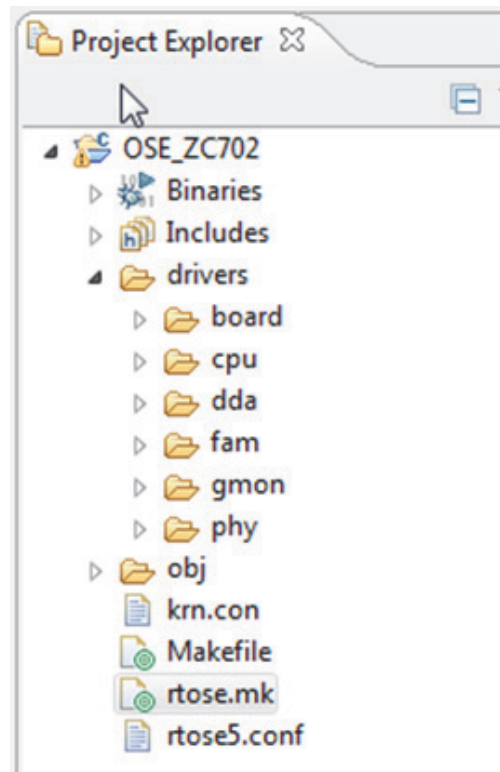
x1221_13_093014

Figure 13: OSE BSP Modules

The example module used in this application note is the `pingpong` module. This is a very simple demonstration module, containing two processes sending signals to each other. A signal in OSE is used to pass messages between processes. It contains an ID, and can optionally contain other information as well. The `pingpong` module sends an integer value back and forth using such a signal. See the *ENEAS OSE Device Drivers User's Guide* [Ref 3] for more information on signals and their use.

The ping process creates a signal and sends it to the pong process, and this pong process receives the signal, increments a counter variable within this signal, and sends it back.

There are several mechanisms to use and launch modules, but for this example, you will compile the module in the kernel so it is launched at startup. One way of accomplishing this is to modify the kernel build system environment to add the module. See *Enea OSE Refsys User's Guide* [Ref 4] for more information on the build system. The file you need to modify to include this module is in the kernel source folder `OSE_ZC702/rtose.mk`. See [Figure 14](#).

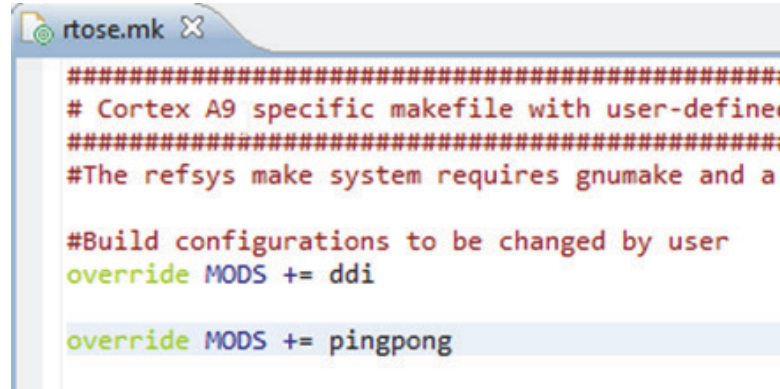


x1221_14_093014

Figure 14: OSE_ZC702 > rtose.mk

1. To include the `pingpong` module, add the following string somewhere in the file (see [Figure 15](#)):

```
override MODS += pingpong
```



```
#####
# Cortex A9 specific makefile with user-defined
#####
#The refsys make system requires gnumake and a

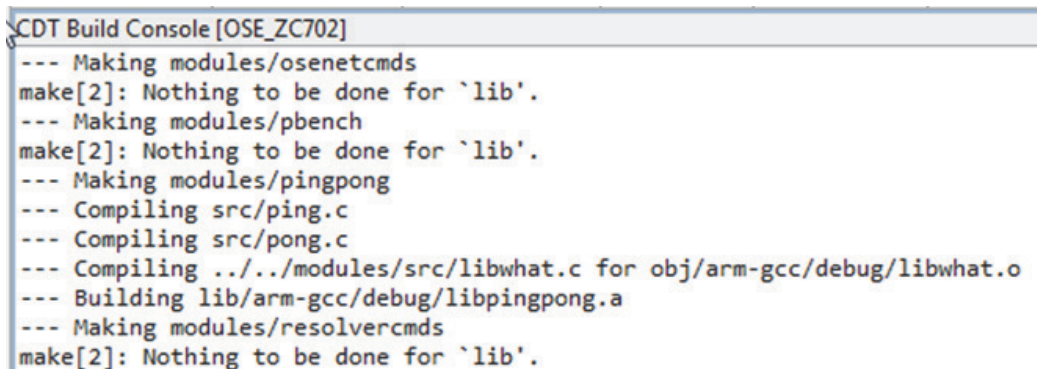
#Build configurations to be changed by user
override MODS += ddi

override MODS += pingpong
```

x1221_15_093014

Figure 15: Adding Pingpong Module to `rtose.mk`

2. Recompile the OSE kernel itself, which will compile the `pingpong` module.
3. Link the `pingpong` module in the OSE kernel. Right-click the **OSE_ZC702** project and select **Build Project** to do so, as shown in [Figure 16](#).



```
CDT Build Console [OSE_ZC702]
--- Making modules/osenetcmds
make[2]: Nothing to be done for `lib'.
--- Making modules/pbench
make[2]: Nothing to be done for `lib'.
--- Making modules/pingpong
--- Compiling src/ping.c
--- Compiling src/pong.c
--- Compiling ../../modules/src/libwhat.c for obj/arm-gcc/debug/libwhat.o
--- Building lib/arm-gcc/debug/libpingpong.a
--- Making modules/resolvercmds
make[2]: Nothing to be done for `lib'.
```

x1221_16_093014

Figure 16: CDT Build Console

4. Following the guidelines for running OSE on Zynq-7000 devices will boot this new kernel containing the `pingpong` module. After booting, you can show the blocks loaded by the

kernel by issuing the `bl` command, as shown in [Figure 17](#).

```

rtose@zynq7020> bl
      bid name                procs    segid
0x00100047 pingpong           2 0x00000000
0x00100033 ose_cfs_blk        1 0x00000000
0x0010000e osemain            69 0x00000000
0x00100008 main                3 0x00000000
0x00100001 OSE                  8 0x00000000
Total 5 blocks
rtose@zynq7020>

```

x1221_17_093014

Figure 17: Blocks Loaded

- Similarly, use the `ps` command to show the running processes, which include our ping and pong processes, as shown in [Figure 18](#).

```

rtose@zynq7020> ps
      pid name                tpr block    own status
00100049 pong                 p30 pingpong  1 delay
00100048 ping                 p30 pingpong  0 rcv
00100034 ose_cfs_jeff         p25 ose_cfs_blk 2 rcv
00400059 ps                    p25 osemain    6 running
00100058 shell                 p25 osemain    8 rcv
00100057 ose_vfs/pop3            p22 osemain    1 rcv
00100056 ose_vfs/http           p22 osemain    1 rcv
00100055 ose_vfs/ftp             p22 osemain    1 rcv
00100054 ose_vfs/tftp            p22 osemain    1 rcv

```

x1221_18_093014

Figure 18: Running Processes

Debugging an Application using Optima

To set up the connection between the ZC702 board running the OSE kernel and the Optima IDE, a new hardware target should be configured in Optima.

1. To add such a target, click the **Add new Gate** in the **System Browser** tab of the System Browsing perspective, as shown in [Figure 19](#).

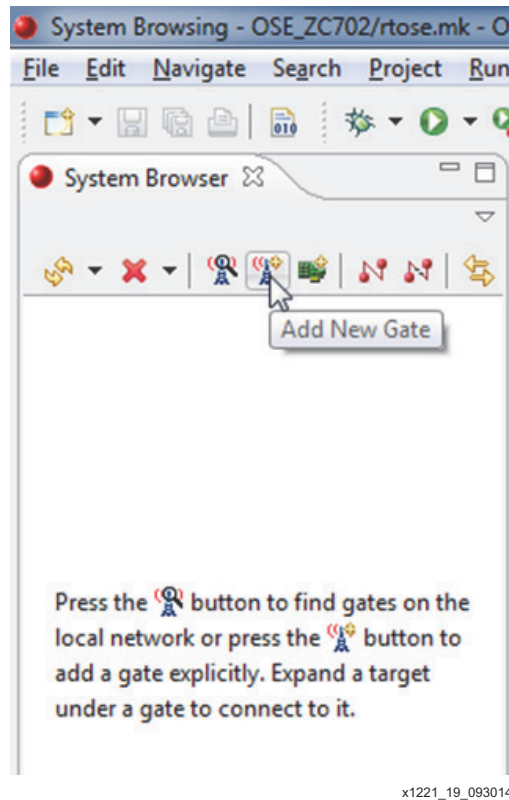


Figure 19: System Browser

2. Enter the IP address of the Zynq-7000 AP SoC target running OSE, which will be 192.168.1.111 if the illustrated defaults were used. See [Figure 20](#).

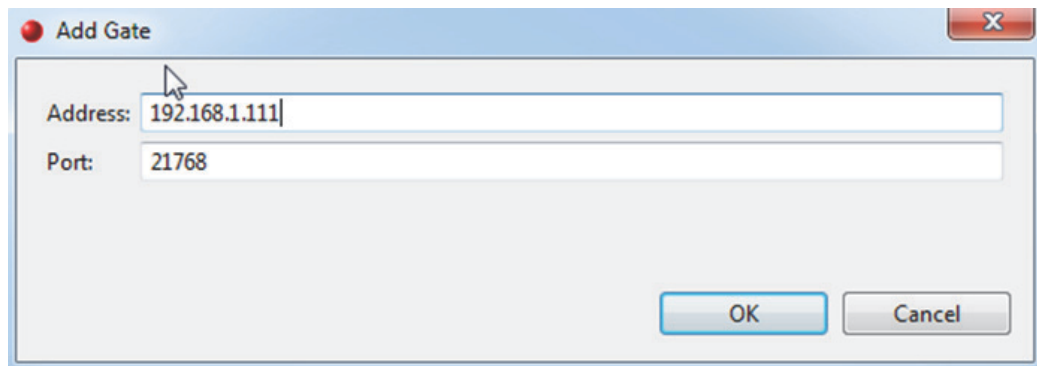
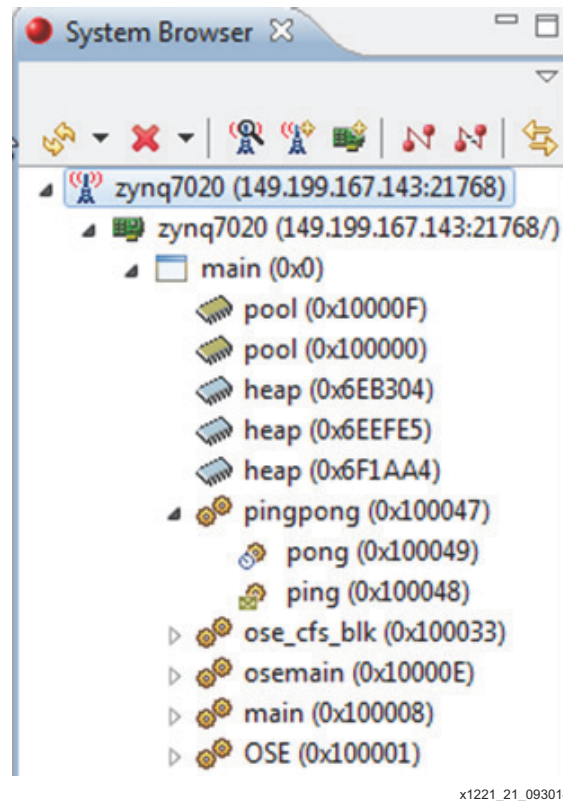


Figure 20: Gate Configuration

Upon successful connection to the target running the OSE kernel, you will be able to interact with it using the **System Browser** tab, as shown in [Figure 21](#).

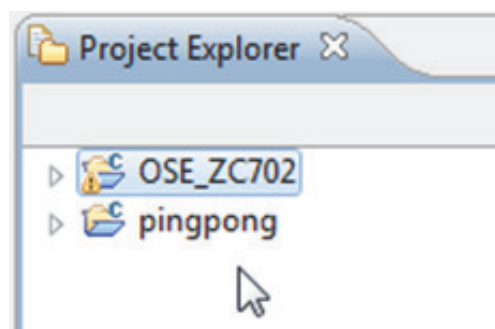


x1221_21_093014

Figure 21: **System Browser**

Further information on what can be done with the **System Browser** tab in Optima can be found in ENEA Optima documentation. In this section, the debugging capabilities will be demonstrated.

- Before debugging an application using Optima, add the application to the current Workspace. Follow the same steps as when you created the kernel project to add the pingpong project (<Zynq BSP Install Path>\refsys\modules\pingpong) to the workspace. This will then also show up in the Project Explorer, as shown in [Figure 22](#).



x1221_22_093014

Figure 22: **Project Explorer**

Both the OSE kernel and the `pingpong` module projects can be built from within the Optima environment as well. For example, **Project > Build All** will build the current project debug and release versions.

4. The next step is to create the debug configuration. Select **Run > Debug Configurations...**, as shown in [Figure 23](#).

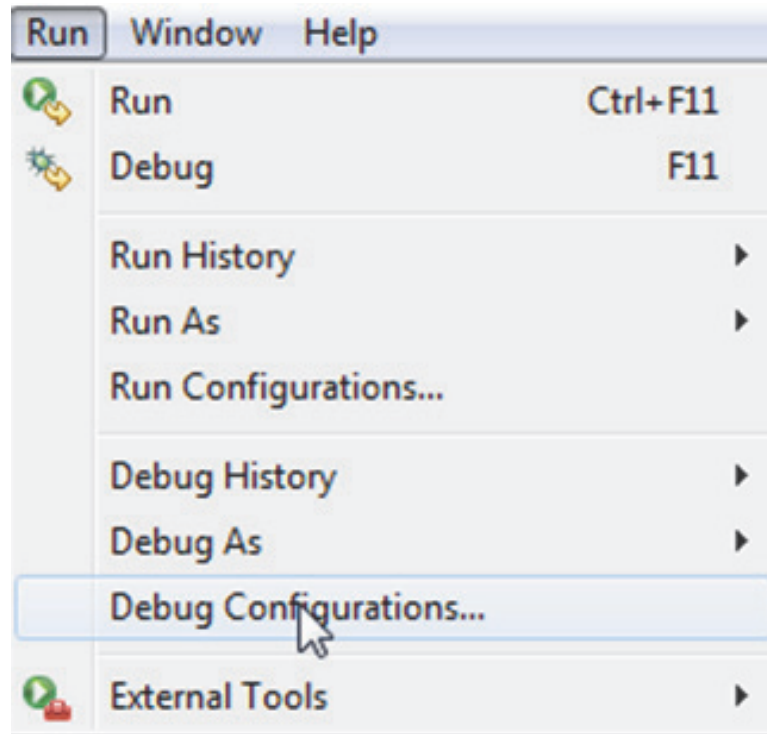
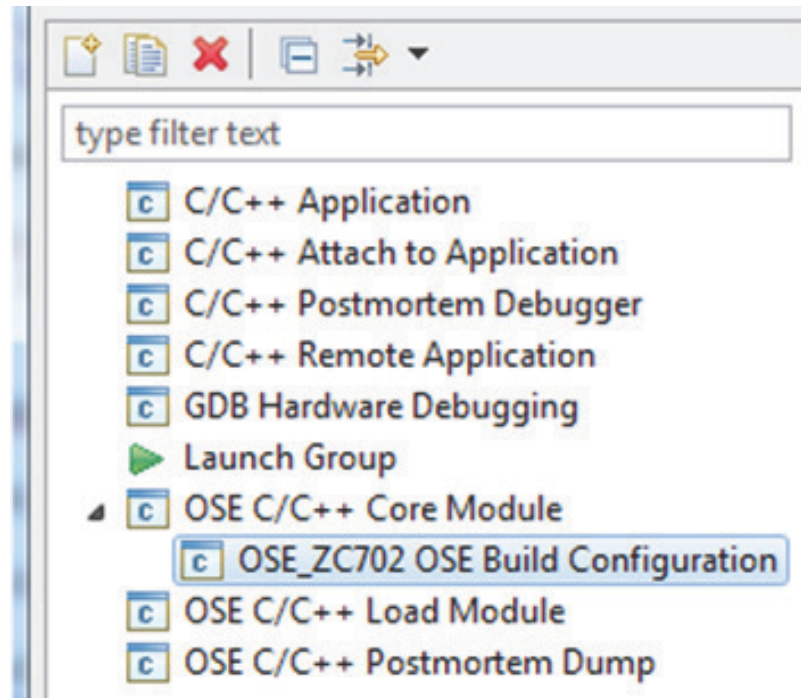


Figure 23: Debug Configurations...

x1221_23_093014

5. In the wizard that appears, double-click the **OSE C/C++ Core Module** entry, which will create a new debug configuration (see [Figure 24](#)).

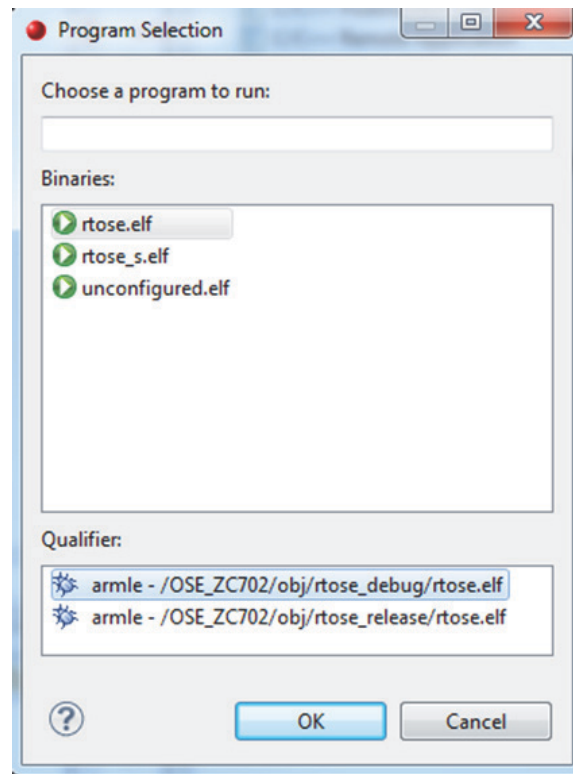


x1221_24_093014

Figure 24: New Debug Configuration

6. Fill in the C/C++ Application by clicking **Search Project...** and selecting the `.elf` that was used to create the U-Boot image (`rtose.bin`) of the OSE kernel: `<Zynq BSP install`

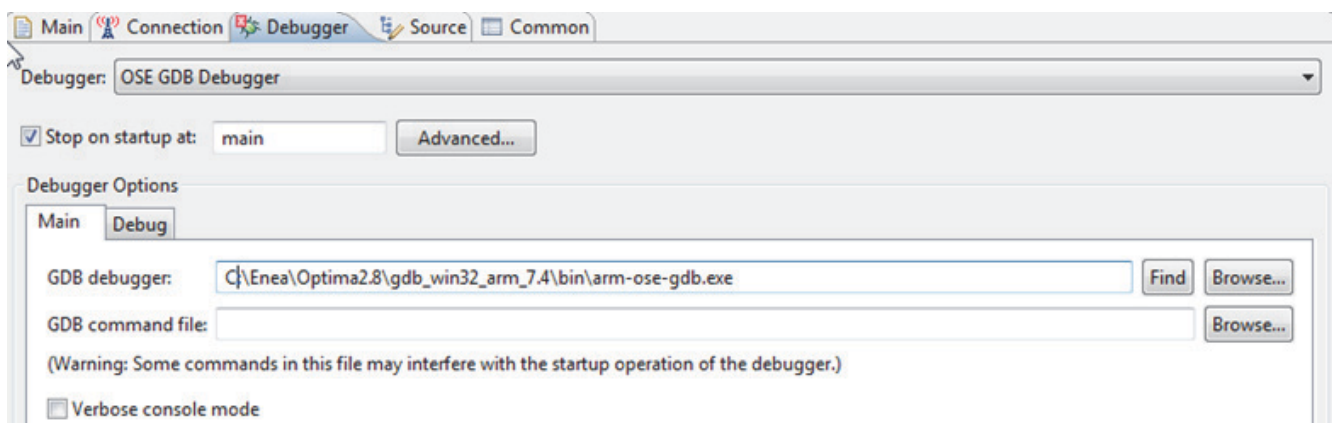
path>\refsys\rtose\zynq7020\obj\rtose_debug\rtose.elf. See [Figure 25](#).



x1221_25_093014

Figure 25: Select ELF File

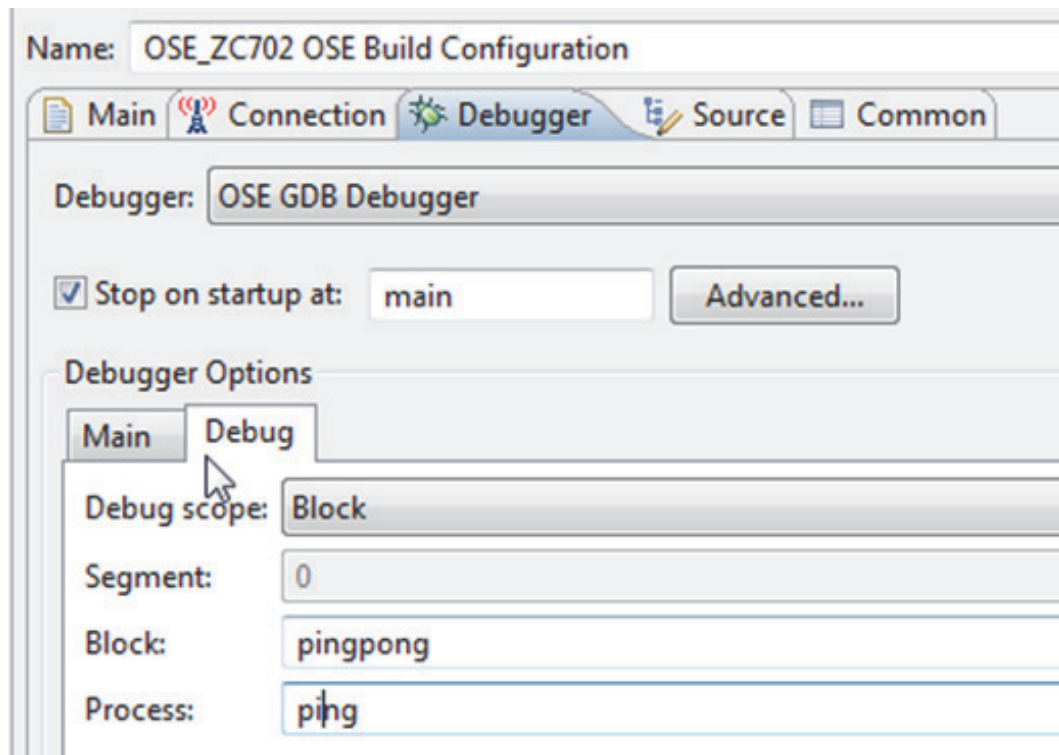
7. In the **Connection** tab, set the IP address of the target and leave the **Gate Port** number to the default (21768).
8. In the **Debugger** tab, set the **GDB debugger** by clicking **Find** (next to the GDB debugger). See [Figure 26](#).



x1221_26_093014

Figure 26: GDB Debugger Configuration

9. And finally, in the **Debug** tab, change the scope of the **Debug scope** to **Block**, and set the **Block** name to `pingpong`, and the **Process** to `ping`, as shown in Figure 27.

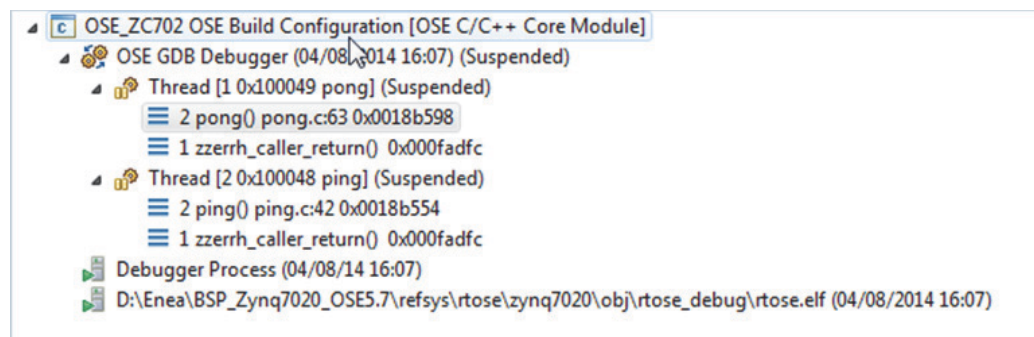


x1221_27_093014

Figure 27: **Debug Tab**

10. To start the debug session, close the wizard by clicking on **Debug**. Eclipse will prompt to change the perspective from the System Browsing to the Debug perspective. This should be accepted.

Once the debug connection has been made, the two threads of the pingpong process will be stopped, which can be seen in the **Debug** tab. See Figure 28.

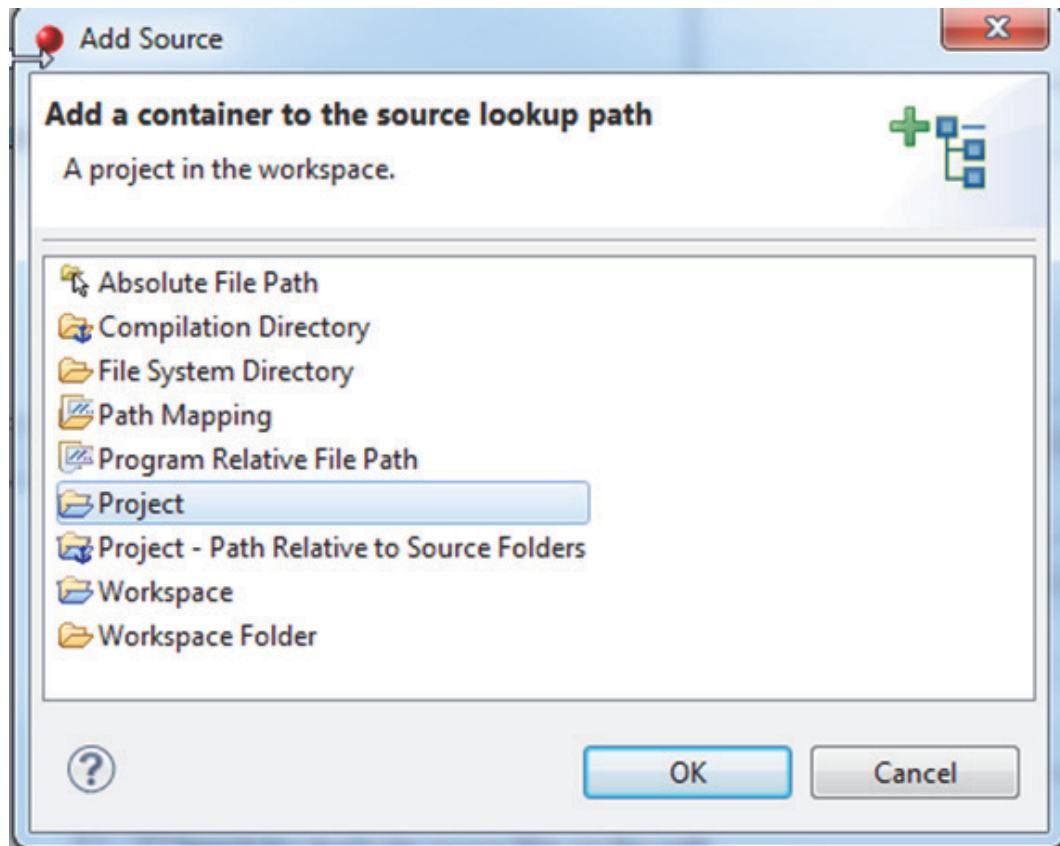


x1221_28_093014

Figure 28: **pingpong Process**

For example, clicking the ping process will attempt to open the `ping.c` source file, but since you are actually debugging the complete OSE kernel project, these sources cannot be found automatically. This can be resolved by clicking **Edit Source Lookup Path...** and selecting the correct source path for these files.

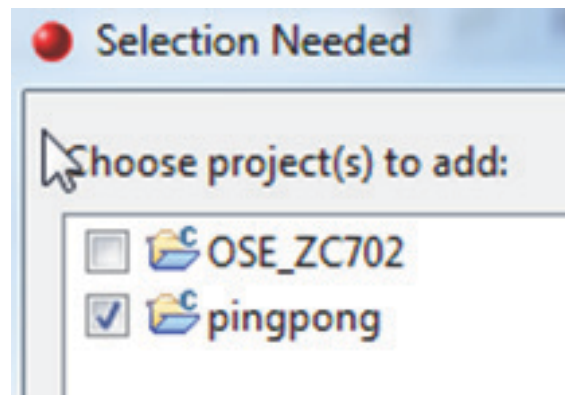
11. Select the **Add Source**, then **Project**. See [Figure 29](#).



x1221_29_093014

Figure 29: **Add Source > Project**

12. Finally, select the pingpong project. See [Figure 30](#).



x1221_30_093014

Figure 30: **Select Project**

This will allow Optima to find the source files used by this module.

With the debugger connected now, the standard eclipse features can be used to debug, break, view variables, etc.

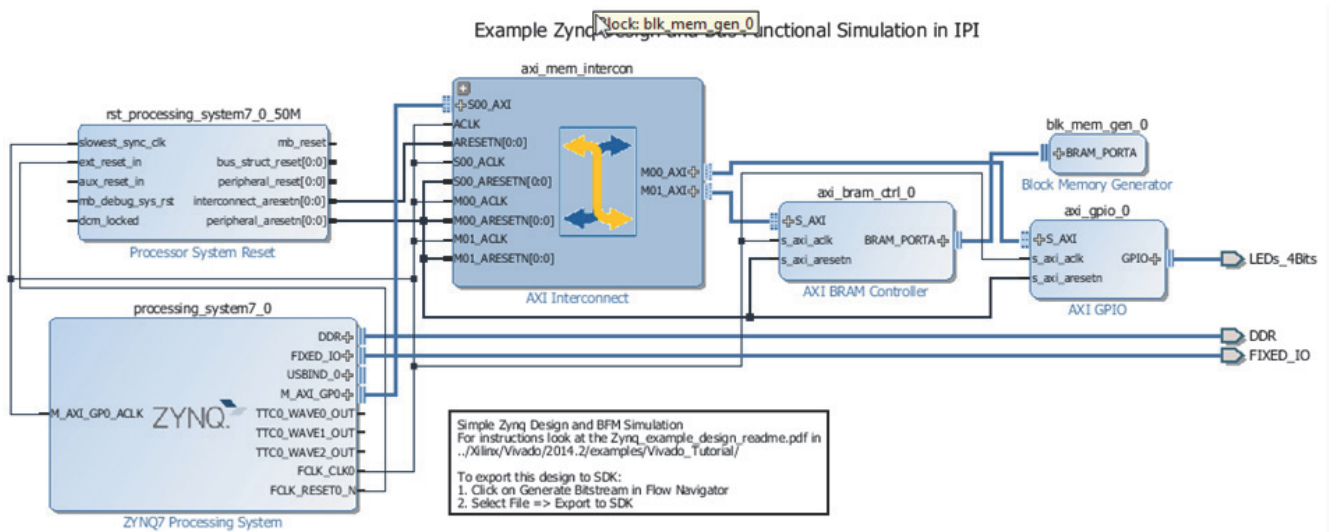
Accessing a Peripheral in the Programmable Logic

Inclusion of two general purpose AXI Master interfaces between the PS and the programmable logic ensures that user IP cores within the programmable logic are memory mapped in the same way that each of the peripherals within the PS are memory mapped. See the *Zynq-7000 All Programmable SoC Technical Reference Manual (UG585)* [Ref 1] for details. This common memory mapped approach ensures that access to peripherals, whether they are within the processing system or programmable logic, is an act of accessing a device-specific memory region.

In this section, you will modify the pingpong example application to show how to access a peripheral in the Programmable Logic.

Vivado IP Integrator Hardware Design

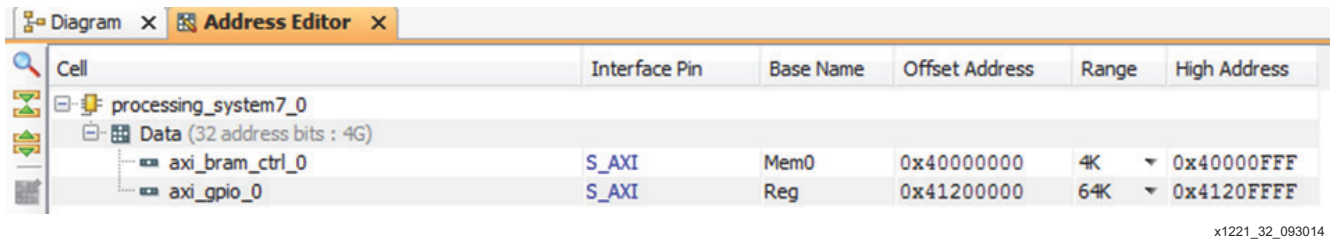
The hardware design used for this section is created using the Vivado IP Integrator tools. The details of how to create such a design is out of scope for this application note, but can be found in *Vivado Design Suite User Guide: Embedded Processor Hardware Design (UG898)* [Ref 5] and *Vivado Design Suite Tutorial: Embedded Processor Hardware Design (UG940)* [Ref 6]. The project you will use is the Zynq example design, available in the 2014.2 Vivado release, which contains only the processing system, a block RAM controller, and an AXI GPIO peripheral in the programmable logic, driving four LEDs on the board. See Figure 31.



x1221_31_093014

Figure 31: Block Diagram of Example Design

The memory map of this design is also configured in the IP Integrator environment, as shown in [Figure 32](#).



Cell	Interface Pin	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0x40000000	4K	0x40000FFF
axi_gpio_0	S_AXI	Reg	0x41200000	64K	0x4120FFFF

x1221_32_093014

Figure 32: Address Map

For your convenience, the implemented bitstream is available for download (see [Reference Design](#)) with this application note.

After the implementation of this design, you also need to generate a First Stage Bootloader to configure the Processing Subsystem as defined by the IP Integrator project. There are multiple methods to create this First Stage bootloader, for example using XSDK (see [\[Ref 7\]](#)). A precompiled FSBL is also included in the attached reference design.

Example Application Modification

For the simple example hardware design, you will modify the `pingpong` example to access our GPIO peripheral in the programmable logic. The modifications are done in the `pong.c` file, which will take the counter value in the signal that is sent between the ping and pong process, and drive the LEDs with this value. The only modification needed for this purpose is to configure the AXI GPIO peripheral to make all GPIO outputs:

```
Xil_Out32(GPIO_TRI_ADDR, 0x0);
```

And then, when a signal is received, you need to drive the LEDs with this value:

```
Xil_Out32(GPIO_DATA_ADDR, new_value);
```

The I/O functions by reading and writing from the correct memory location, and are defined as follows:

```
void Xil_Out32(u32 OutAddress, u32 Value)
{
    *(volatile u32 *) OutAddress = Value;
}

u32 Xil_In32(u32 Addr)
{
    return *(volatile u32 *) Addr;
}
```

The complete source code for the `pong.c` process can also be found in the attached reference design (see [Reference Design](#)).

OSE Kernel Modifications

Because OSE uses the MMU to access and protect memory regions, you also need to modify its configuration to be able to access our peripheral in the Programmable Logic. By default, the MMU is not configured to allow access to the memory regions used by the two Master AXI interfaces. OSE has multiple methods to modify this MMU configuration, both dynamically and statically. See the *ENEA OSE Architecture User's Guide* [Ref 8] for more information. For this example, a static configuration was selected.

The static configuration of the MMU is done in the file `<Zynq BSP install path>/refsys/rtose/zynq7020/rtose5.conf`. To add a memory region to the MMU, add the following line to the file, as shown in [Figure 33](#):

```
krn/region/axi_gpio=base:0x41200000 size:64M perm:su_rw_usr_rw cache:inhibited
mem:ordered_access
```

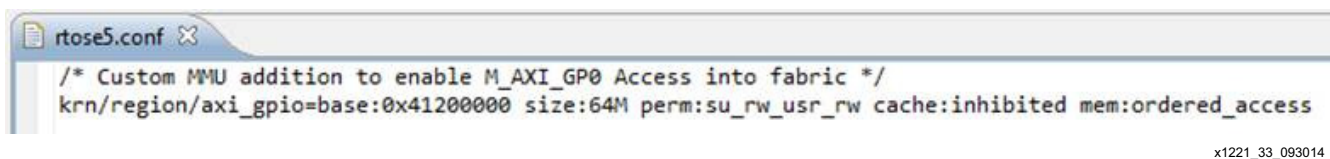


Figure 33: `rtose.conf`

This will allow OSE to access the memory region of our AXI GPIO peripheral located in the programmable logic, without using caches.

Now you can rebuild the OSE kernel, following the steps of [Building the OSE Kernel for Zynq-7000 AP SoC Devices](#).

Booting Zynq-7000 AP SoC Devices using a Bitfile and Updated Kernel

Before you can run the updated kernel image on our ZC702 board, you have to make sure that the programmable logic is configured with our custom hardware design. One method to accomplish this is to update the boot image to include this bitfile, which will then during the boot process be used by the First Stage Bootloader to configure the programmable logic, before U-boot will get loaded and executed. To create a new boot image, use the tool `bootgen`, which is configured by a `.bif` file (`ose.bif`) containing the different partitions in the boot image. For more information on creating a boot image and using the `bootgen` application, see UG821, Appendix A: Using `Bootgen` [Ref 9].

In summary, you need a `.bif` file like this:

```
//ZC702_bif_for_OSE:
{
  [bootloader] fsbl.elf
  zynq_1_wrapper.bit
  u-boot.elf
}
```

Where:

- `fsbl.elf` is the fsbl created in the previous section, [Vivado IP Integrator Hardware Design](#).
- `zynq_1_wrapper.bit` is the bitfile created in the same section.
- `u-boot.elf` is the same U-Boot used in Running OSE on the Zynq-7000 AP SoC.

To generate the new boot image, open an SDK command prompt, and execute the following:

```
bootgen -image ose.bif -o boot.bin
```

This will create a new `boot.bin` file, which needs to be copied on the SD card used to boot the board. For your convenience, the attached reference files contain all sources, as well as generated `boot.bin` file which can be used to boot the board.

Following the instructions from this document in [Running OSE on Zynq-7000 AP SoC Devices](#), you now boot the ZC702 board using this updated image, which configures the programmable logic during the FSBL stage. After the OSE kernel is executed, the updated `pingpong` module will run, which will toggle the LEDs on the board. When connecting using Ethernet, and using Optima to debug the kernel, as explained in [Debugging an Application using Optima](#), the pong process can be debugged.

Conclusion

This application note provides step-by-step instructions for running the OSE BSP on the Zynq-7000 SoC All Programmable device platform, and shows how to build and debug example applications accessing peripherals in the Processing System and Programmable Logic.

Reference Design

Download the [reference design files](#) for this application note from the Xilinx website.

[Table 1](#) shows the reference design matrix.

Table 1: Reference Design Matrix

Parameter	Description
General	
Developer Name	Kester Aernoudt
Target Devices	Zynq-7000 AP SoC
Source code provided?	Y
Source code format (if provided)	Block diagram/C
Design uses code or IP from existing reference design, application note, third-party or Vivado software? If yes, list.	IP cores from Vivado IP Catalog

Table 1: Reference Design Matrix (Cont'd)

Parameter	Description
Hardware Verification	
Hardware verified?	Y
Platform used for verification	ZC702 Development Board

References

1. *Zynq-7000 All Programmable SoC Technical Reference Manual* ([UG585](#))
2. Embedded Wiki <http://www.wiki.xilinx.com/Zynq+Releases>
3. *Enea OSE Device Drivers User's Guide* (installed with BSP)
4. *Enea OSE Refsys User's Guide* (installed with BSP)
5. *Vivado Design Suite User Guide: Embedded Processor Hardware Design* ([UG898](#))
6. *Vivado Design Suite Tutorial: Embedded Processor Hardware Design* ([UG940](#))
7. Chapter 3: Boot and Configuration ([UG821](#))
8. Enea OSE Architecture User's Guide (Installed with OSE)
9. Appendix A: Using Bootgen ([UG821](#))

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/12/2015	1.0	Xilinx initial release

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

Automotive Applications Disclaimer

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.

© Copyright 2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.