BSP User Guide

Xilinx Zynq-7000 SoC ZC702
Xilinx Zynq-7000 SoC ZC706

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: Wednesday, January 21, 2015

Table of Contents

About this guide

In this guide

BSP User Guide: Xilinx Zynq-7000 SoC ZC70x contains installation and start-up instructions for the QNX Board Support Package (BSP) for the Xilinx Zynq-7000 SoC ZC702 and the Xilinx Zynq-7000 SoC ZC706 boards.

For convenience, the Xilinx Zynq-7000 SoC ZC702 and Xilinx Zynq-7000 SoC ZC706 boards may be referred to as simply the Xilinx Zynq-7000. Unless specifically noted otherwise, the information in this guide is for both these boards.

To find out about:	See:
The resources available to you, and what you should know before starting to work with this BSP	*Before you begin* (p. 9)
What's included in the BSP, and supported host OSs and boards	*About this BSP* (p. 11)
Building and installing this BSP	*Building the BSP* (p. 16)
Driver commands	*Driver commands* (p. 21)

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	`NULL`
Data types	`unsigned short`
Environment variables	***PATH***
File and pathnames	`/dev/null`
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

Chapter 1
Before you begin

Before you begin working with this BSP you should become familiar with the resources available to you when building QNX embedded systems.

Essential information

Before you begin building and installing your BSP, you should review the following information about your board's the hardware and firmware, available from Xilinx at:

- Xilinx Zynq-7000 SoC ZC702:

  www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm

  ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide at *www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf*.

- Xilinx Zynq-7000 SoC ZC706:

  www.xilinx.com/products/boards-and-kits/EK-Z7-ZC706-G.htm

  ZC706 Evaluation Board for the Zynq-7000 XC7Z045 All Programmable SoC User Guide at *www.xilinx.com/support/documentation/boards_and_kits/ug954-zc706-eval-board-xc7z045-ap-soc.pdf*.

For general information about QNX BSPs, and instructions for tasks common to all BSPs, see *QNX SDP 6.6.0 BSPs* available on the the QNX Infocentre (*www.qnx.com/developers/docs/index.html*). This documentation includes:

- an overview of QNX BSP
- links to documentation about how to build QNX embedded systems, which you should read before you begin working with this BSP
- what's new in the BSPs for this release
- structure and contents of a BSP
- how to prepare a bootable SD card
- how to modify an older BSP to work with the current release

Pre-requisites

Before you start to work with this BSP, you need:

- a QNX account (myQNX)
- a USB to micro USB cable for debugging

- installed on your host system:
  - the QNX Neutrino RTOS 6.6 SDP
  - a terminal program (e.g., QNX Momentics IDE's Serial Terminal View, or Putty)

Technical Support

To obtain technical support for any QNX product, visit the Support area on our website. You'll find a wide range of support options, including community forums.

Latest version of this BSP

For the most up-to-date version of this user guide, log in to your myQNX account, and download it from the same location as the BSP.

Chapter 2
About this BSP

These notes list what's included in the BSP, and identify the host OSs and boards it supports.

What's in this BSP

This BSP contains the components listed in the table below. For more detailed information about binaries and libraries, see "*Driver commands* (p. 21)".

Component	Format	Comments
IPL	Source	
Startup	Source	
WTD kick	Source	
Real-time clock	Source	
Serial driver	Source	
I2C driver	Source	
SPI driver	Source	
MMCSD	Source	
Network	Source	
SPI NOR flash	Source	
CAN	Source	Only available on the ZC702 board.
FPGA	Source	
GPIO	Source	
OCM	Source	
XADC	Source	
USB host	Binary only	
User Guide	PDF	This document: includes release notes and installation note.

Supported OSs

In order to install and use this BSP, you must have installed the QNX Software Development Platform (SDP) 6.6, on either a Windows or Linux Host PC.

This BSP supports the following target OS:

- QNX Neutrino® RTOS 6.6

Supported boards

This BSP supports the following boards:

- Xilinx ZC702 Evaluation Board for the Xilinx Zynq-7000 SoC7020
- Xilinx ZC706 Evaluation Board for the Xilinx Zynq-7000 SoC7Z045

Required software

To work with the Xilink Zynq-7000 boards and their QNX BSP, in addition to the QNX Neutrino RTOS 6.6, you need:

- the Xilinx Software Development Kit, which you can dowload from Xilinx at *www.xilinx.com/support/download.html*
- a terminal emulation program, such as QNX Momentics IDE Terminal, Qtalk, tip or Hyperterminal, etc.

Required cables

To connect to your board, you need:

- a USB mini-B to USB cable
- an Ethernet cable

Chapter 3
Installation notes

These installation notes describe how to build, install and start this BSP.

Please refer to the *QNX SDP 6.6.0 BSPs* guide, available as part of the QNX Software Development Platform OS Core Components documentation in the QNX Infocentre for detailed instructions how to extract and build a BSP, and how to prepare a bootable, DOS / FAT32 formatted SD card.

Connecting and configuring the hardware

Before you build the BSP, you should configure the board DIP switches and connect the board and peripherals.

Configure the board's switches

The Xilinx Zynq-7000 boards' SW16 (ZC702) or SW11 (ZC706) DIP switches should be set as follows for default booting off the SD card. Note that the black block represents the physical lever inside the switch:

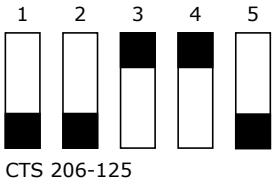

Figure 1: SW16 (ZC702) or SW11 (ZC706) DIP switch settings

Figure 2: SW16 (ZC702) or SW11 (ZC706) DIP switch settings (photo of a ZC702)

Connect the hardware

You will require a USB serial device driver on your host development machine in order to establish a terminal session with the target over USB. You can download the appropriate device driver, for either Linux or Windows, from www.ftdichip.com.

To connect to your board:

1. Connect an RJ-45 Ethernet cable between the Xilinx Zynq-7000's Ethernet port and your local network.
2. Connect the power supply to the Xilinx Zynq-7000's power connector.
3. Connect the USB cable (included with the board) between the board's USB mini-B connector (J17 on the ZC702 or J21 on the ZC706) and your host machine.
4. Use the board's Power on/off slider switch to power up the board.
5. On your host machine, determine the port name of the USB port used for serial communications with the Xilinx Zynq-7000 board, and start your favorite terminal program with these settings:
   - Baudrate: **115200**

- Data: **8 bit**
- Parity:**none**
- Stop: **1 bit**
- Flow control: **none**

Building the BSP

You can use the QNX Momentics IDE or the command line to build the BSP.

After you have connected the hardware and configured the board switches, you can use either the QNX Momentics IDE or the command line to build and install the BSP.

Use the IDE to build the BSP

To work with this BSP using the QNX Momentics IDE on a Windows or Linux system, please refer to the the chapter "Using BSPs in the IDE" in the QNX *Building Embedded Systems* guide (*www.qnx.com/developers/docs/660/topic/com.qnx.doc.neutrino.building/topic/about.html*) and the *QNX BSP wiki* for instructions on how to import and build the BSP available on.

Use the command line to build the BSP

To work with this BSP at the command line on a Windows or Linux machine, simply open a shell, then:

1. Create a new directory for the BSP.
2. Copy the BSP's `.zip` archive into this new directory.

If you plan to work with multiple different BSPs, we recommend creating a top- level BSP directory, then creating subdirectories for each different BSP. The directory you create for each BSP will be that BSP's root directory (***$BSP_ROOT_DIR***).

3. Once the `.zip` archive is in place, extract it using the `unzip` command. The `unzip` utility ships with the QNX SDP, for all supported host platforms.
4. When you're ready to build the BSP, from the BSP's root directory, type `make`. You may need to source the SDP's environment variables, depending on your tool chain.

The `make` command will build the BSP and create directories, as described above.

Preparing boot images

You can modify the boot images that are shipped with the BSP, then rebuild and copy them to the SD card you will use to boot your target system.

Pre-built image

After you have unzipped the BSP, a pre-built QNX IFS image is available in the BSP's `/images` directory.

This pre-built IFS image is configured for the various BSP device drivers already on your system. If you modify and rebuild the IFS, you will overwrite the pre-built image. If you need to revert to this pre-built image, simply run a `make` from the BSP's root directory (`BSP_ROOT_DIR`).

Build the IPL boot image

The Xilinx Zynq-7000 boards require a custom bootloader from Xilinx, which we have incorporated into the QNX Initial Program Loader (IPL) for these boards. Because they require a custom bootloader, booting from the IPL is the the only supported boot method: U-Boot is not supported..

To use the IPL to boot your board, you first need to run the `mkflashimage` shell script in the BSP's `/images` directory.

Before you start, modify the SDK version (2013.*x*) and SDK path in the `mkflashimage` shell file to match with the installed version of the Xilinx tool.

To build the IPL boot image:

1. On your host system, open a command-line terminal.
2. Go to the BSP's `BSP_ROOT_DIR/images` directory.
3. Run the shell script to build the IPL:
   - Linux: `./mkflashimage`
   - Windows: `sh mkflashimage`

These scripts run in the BSP's `/images` directory generate a file called `BOOT.bin`, which contains the QNX IPL and the Xilinx First Stage Boot Loader.

Modifying and rebuilding the IPL and the IFS

The IPL binary: `ipl-xzynq-zc702.elf` or `ipl-xzynq-zc706.elf` comes pre-built in the `BSP_ROOT_DIR/images` directory. If you need to change the IPL source code, you can make your changes, then rebuild only the IPL binary. You don't need to rebuild the whole BSP.

1. To rebuild the IPL:

   ```
   $ cd BSP_ROOT_DIR/src
   $ make
   $ make install
   $ cd BSP_ROOT_DIR/images
   $ make clean
   $ make
   $ ./mkflashimage
   ```

   The `mkflashimage` script generates the binary file `BOOT.bin`.

2. If you need to make changes to the BSP source code (`BSP_ROOT_DIR/src`), you can make the changes, then rebuild the IFS image:

   ```
   $ cd BSP_ROOT_DIR/src
   $ make
   $ make install
   $ cd BSP_ROOT_DIR/images
   $ make clean
   $ make
   $ ./mkflashimage
   ```

3. Copy the newly generated IPL (`BOOT.bin`), then the IFS (`QNX-IFS`) to the SD card you are using to boot your board.

Prepare a bootable micro SD card

To enable booting the system from the SD card, you need create a DOS FAT32 partition (type 12) on the card. The following is a quick, step-by-step method for formatting the SD card from an Ubuntu Linux terminal:

1. Show the SD card's mountpoint. We'll assume for this example that it is `/dev/sda`:

   ```
   $ mount
   ```

2. Unmount the card:

   ```
   $ umount /dev/sda
   ```

3. Create the partition, entering the default start and end cylinder numbers, and the partition type: 12 or `c`, FAT32:

   ```
   $ sudo fdisk /dev/sda
   > u
   > o
   > n
   > p
   > 1 start cylinder end cylinder
   > a
   > 1
   > t
   > c
   > w
   ```

```
$ sudo mkfs.vfat -F 32 /dev/sda1
$ sync
```

4. 

   Copy the IPL *first*.

   Use a standard `cp` command to copy the IPL (`BOOT.bin`), then the IFS (`QNX-IFS`) to the SD card.

When you have finished preparing your SD card, all the boot images should be in the root directory in the card's FAT partition.

Load and run the IFS from the SD card

To load and run the QNX IFS from the SD card:

1. Insert the card into the board's SD/MMC slot.
2. Power up the board.

   Your terminal should start to output QNX boot information.
3. When the IPL boot menu appears, type `m` to load the IFS from the SD card.

Chapter 4
Driver commands

The tables below provide a summary of driver commands.

Some of the QNX drivers may be commented out in the default build file in the startup directory. To use the drivers in the target hardware, you'll need to uncomment them in your build file, rebuild the image, and load the image onto the board. For more information about these QNX drivers and other QNX commands, see the *Neutrino Utilities Reference*.

Startup

Device	Startup
Command	`startup-xzynq-zc702`
Command	`startup-xzynq-zc706`
Required binaries	`startup-xzynq-zc702` or `startup-xzynq-zc706`
Required libraries	
Source location	`src/hardware/startup/boards/xzynq/zc702` or `src/hardware/startup/boards/xzynq/zc706`

Watchdog

Device	Hardware watchdog
Command	`wdtkick`
Required binaries	`wdtkick`
Required libraries	
Source location	`src/hardware/support/xzynq/wdtkick`

Real time clock

Device	RTC
Command	`rtc hw`
Required binaries	`rtc`
Required libraries	

Source location	`src/utils/r/rtc`

Serial

Device	Serial (UART1)
Command	`devc-serxzynq -e -F -S 0xE0001000,82`
Required binaries	`devc-serxzynq`
Required libraries	
Source location	`src/hardware/devc/serxzynq`

I2C

Device	I2C1
Command	`i2c-xzynq -p 0xE0004000 -i 57 --u 1`
Required binaries	`i2c-xzynq`
Required libraries	
Source location	`src/hardware/i2cxzynq`

SPI

Device	SPI1
Command	`spi-master -u 0 -d xzynq base=0xE0006000,irq=58`
Command	`spi-master -u 1 -d xzynq base=0xE0007000,irq=81`
Required binaries	`spi-master`
Required libraries	`spi-xzynq.so`
Source location	`src/hardware/spi/xzynq`, `src/hardware/spi/master`

USB

Device	USB host
Command	`io-usb -d ehci-xzynq ioport=0xe0002100,irq=53`
Required binaries	`io-usb`, `usb`, `devb-umass`
Required libraries	`devu-ehci-zynq.so`, `libusbdi.so`
Source location	Prebuilt only

MMC/SD

Device	SD card interface
Command	`devb-mmcsd-xzynq mmcsd verbose=1`
Required binaries	`devb-sdmmc-xzynq`
Required libraries	
Source location	`src/hardware/devb/mmcsd`

Network

Device	Ethernet
Command	`io-pkt-v4 -dxzynq -p tcpip stacksize=8192`
Required binaries	`io-pkt-v4`, `ifconfig`, `dhcp.client`
Required libraries	`devnp-xzynq.so`, `libsocket.so`
Source location	`src/hardware/devnp/xzynq`

Quad SPI NOR flash

Device	NOR Flash (QSPI)
Command	`devf-qspi-xzynq-zc702`
Required binaries	`devf-qspi-xzynq-zc702`
Required libraries	
Source location	`src/hardware/flash/boards/qspi-xzynq`

CAN

Only available on the ZC702 board.

Device	CAN interface
Command	`dev-can-xzynq –p9 xzynqcan1`
Required binaries	`dev-can-xzynq`, `canctl`
Required libraries	`gpio-xzynq`
Source location	`src/hardware/can/xzynq`

FPGA

Device	Zynq FPGA
Command	`fpga-xzynq`
Required binaries	`fpga-xzynq`
Required libraries	`devn-smsc9500.so`, `devnp-shim.so`
Source location	`src/hardware/support/xzynq/fpga`

The FPGA can be flashed by writing a bitstream to `/dev/fpga`. If the bitstream is valid and successfully written, the DONE LED (DS3) will turn on. You can also check the PROG DONE signal can be checked through a `devctl` message to `/dev/fpga`.

You can use `devctl` messages to toggle secure writes on or off. Below is an example showing how to flash the FPGA from the command line:

```
# cat BITSTREAM_FILE.bit.bin > /dev/fpga
```

Here is some code that flashes the FPGA:

```
int flash_fpga(char *bitstream_buffer, int bitstream_size){
    int fd;
    int bytes_written;
    _Uint8t prog_done;

fd = open("/dev/fpga", O_RDWR);
if (fd == -1) {
    return -1;
}

bytes_written = write(fd, bitstream_buffer, bitstream_size);
if (bytes_written != bitstream_size) {
    return -1;
}

devctl(fd, DCMD_FPGA_IS_PROG_DONE, &prog_done, sizeof(prog_done), NULL);
if (prog_done) {
    //Success
    return 0;
    }
    return -1;
}
```

GPIO

Device	GPIO interface
Command	
Required binaries	
Required libraries	`libgpio-xzynq.so`
Source location	`src/lib/gpio`

A library available through the public header `hw/gpio.h` can be used to control every GPIO signal and every pin's multiplexer (MUX) configuration on the Xilinx Zynq-7000 board.

Below is some code that uses the GPIO library to blink an LED:

```
void led_test() {
/* Get the structure which contains the functions to  control the GPIO */
    gpio_functions_t gpiofuncs;
    get_gpiofuncs(&gpiofuncs, sizeof(gpio_functions_t));

    /* Initialize the subsystem */
    gpiofuncs.init();

    /* Blink the LED DS23 */
    gpiofuncs.gpio_set_direction(10, 1);
    gpiofuncs.gpio_set_output_enable(10, 1);
    gpiofuncs.gpio_set_output(10, 0);
    sleep(1);
    gpiofuncs.gpio_set_output(10, 1);
    sleep(1);
    gpiofuncs.gpio_set_output(10, 0);
    sleep(1);
    gpiofuncs.gpio_set_output(10, 1);

    /* Clean up the subsystem */
    gpiofuncs.fini();
}
```

OCM

Device	On-Chip Memory interface
Command	`ocm-xzynq`
Required binaries	`ocm-xzynq`
Required libraries	
Source location	`src/hardware/support/xzynq/ocm`

The On-Chip Memory can be configured dynamically at either of the following address ranges:

- Low address: `0x00000000` to `0x0003FFFF`
- High address: `0xFFFC0000` to `0xFFFFFFFF`

There are two ways to get the current mapping of an OCM block. You can use the cat command:

```
# cat /dev/ocm/ocm1
# LOW
```

Or, you can write a C program. Here is an example of code that accesses and reads the OCM:

```
int main(int argc, char *argv[]){
        int fd, size;
        char value[MAX_CHAR];
        uintptr_t ocm_lowa_virt_base;
```

```
        uintptr_t ocm_high_virt_base;

        ThreadCtl(_NTO_TCTL_IO, 0);
        fd = open("/dev/ocm/ocm1", O_RDWR);
        if (fd == -1) {
            perror("open");
            return EXIT_FAILURE;
        }

        /* Memory Mapping */
        ocm_lowa_virt_base = mmap_device_io(OCM_SIZE, 0x000C0000);
        ocm_high_virt_base = mmap_device_io(OCM_SIZE, 0xFFFC0000);

        /* Output the first word in the OCM1 */
        write(fd, "HIGH", 5);
        size = read(fd, value, MAX_CHAR);
        /* For future read ... */
        lseek(fd, 0, SEEK_SET);
        /* Make sure the string is terminated by \0 */
        value[min(size-1, MAX_CHAR)] = '\0';

        printf("OCM1 status: %s\n", value);
        printf("0x000C0000 = %08x\n", in32(ocm_lowa_virt_base));
        printf("0xFFFC0000 = %08x\n", in32(ocm_high_virt_base));}
        return EXIT_SUCCESS;
}
```

XADC

Device	XADC interface
Command	`xadc-xzynq`
Required binaries	`xadc-xzynq`
Required libraries	
Source location	`src/hardware/support/xzynq/xadc`

The XADC is accessed through `/dev/xadc` and controlled through a `devctl` API defined in `/hw/xadc.h`. The example below reads the chip temperature, and checks that it is between 1 and 100 degrees Celcius:

```
int check_temp (int *safe_temp){
        int fd;
        xadc_data_read_t adc_data;

        if ((fd = open("/dev/xadc", O_RDONLY)) == -1){
            return -1;
        }

        adc_data.channel = XADC_CH_TEMP;
        if (devctl(fd, DCMD_XADC_GET_ADC_DATA, &adc_data,
        sizeof(xadc_data_read_t), NULL)) {
            return -1;
        }

        *safe_temp = ((adc_data.data > 0x8B40) &&
        (adc_data.data < 0xBD90));
        return 0;
}
```

Index