BSP: Xilinx Zynq7000 ZC702

14 May 2013

This "readme" file contains Release Notes and an Installation Note for the Board Support Package (BSP) it accompanies. Read the entire file before you start installing the BSP.

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.

Reference numbers

Throughout this document, you may see reference numbers associated with particular issues, changes, etc. When corresponding with our Technical Support staff about a given issue, please quote the relevant reference number.

You might also find the reference numbers useful for tracking issues as they become fixed.

Latest version of this BSP

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

Contents

Release Notes

Read these notes before you start. The few minutes it takes will save you time and grief later on.

©[201Y] QNX Software Systems Ltd.

Before you begin

Before you begin building and installing your BSP, you should review the documentation about your board's hardware and firmware available from the board vendor, and the QNX documentation that explains how to build an embedded system.

QNX documentation can be found in the QNX Infocentre. You should be familiar with the following documentation:

Manual	Chapter	Learn how to ...
Building Embedded Systems	Working with a BSP	Build the BSP.
Building Embedded Systems	Making an OS image	Use the command-line to customize the BSP.
IDE User's Guide	Building OS and Flash Images	Use the IDE to customize the BSP.

What's in this BSP?

This BSP contains the following components:

Component	Format	Comments
IPL	Source	
Startup	Source	
Serial driver	Source	
i2c driver	Source	
network	Source	
SD card	Source	
USB Host	Binary	
Readme file	HTML	Includes release notes and installation note.

Supported boards and OSs

This BSP supports QNX Neutrino 6.5.0 SP1 on the Xilinx Zynq7000 SoC, using the Xilinx ZC 702 Evaluation Kit.

Host OS

In order to use this BSP, you must have:

- QNX Momentics 6.5.0 SP1, installed on one of the following environments:
  - Microsoft Windows (XP, Vista, or Windows 7)
  - Linux

(See the QNX Momentics 6.5.0 Release Notes, for a full list of the supported Host Development environments.)

- The Xilinx ISE Design Tools (available from xilinx.com, here)
- a terminal emulation program (Qtalk, QNX Momentics IDE Terminal, tip, Hyperterminal, etc.)
- an RS-232 serial port and serial cable, or a USB-to-serial cable
- an Ethernet link

© 2013 QNX Software Systems Ltd.

Note: although this BSP can be built in a self-hosted QNX Neutrino environment, the final step of the BSP build process (mkflashimage), which orients the QNX IPL loader into a format recognized by the Zynq's on board boot ROM, requires using a Xilinx-provided tool (bootgen) that ships with the Xilinx ISE Design Suite as a binary only, and is available only for Linux and Windows hosts. Therefore, a self-hosted QNX BSP build environment is not recommended for this BSP.

Target OS

- QNX® Neutrino® RTOS
- Bootloaders:
  - QNX IPL

Boards Supported

Xilinx ZC702 Evaluation Kit for Zynq7000 SoC

Installation Note

This "Installation Note" provides instructions for building and installing this BSP. It includes:

- Location of the source code
- Binaries, buildfiles, IPLs, and other files
- System layout
- Building and installing the BSP
- Replacing the U-Boot bootloader with the QNX IPL (optional)

Location of the source code

The QNX-packaged BSP is independent of the installation and build paths, if the QNX SDP (Software Development Platform) is installed.

To determine the base directory, open a shell and type `qconfig`.

Binaries, buildfiles, IPLs, and other files

Typically, extracting a BSP .zip archive creates the following directories: `images, install, prebuilt and svc.`

The table below shows typical locations of common BSP components, assuming that `${BSP_ROOT_DIR}` is the name of the directory where you extract the BSP archive, and that `${CPU_VARIANT}` is the specific CPU architecture that this BSP is targeted for: ARMLE, PPCBE, x86, etc.

File	Location
Prebuilt OS image	`${BSP_ROOT_DIR}/images`
IPL and/or startup	`${BSP_ROOT_DIR}/install/${CPU_VARIANT}/boot/sys`
Source code for different drivers	
(serial, flash, block, PCI, PCMCIA, USB)	`${BSP_ROOT_DIR/install/${CPU_VARIANT}/sbin`

© 2013 QNX Software Systems Ltd.

Prebuilt libraries	
(audio, graphics, network)	`${BSP_ROOT_DIR/install/${CPU_VARIANT}/lib/dll}`
Generic header files (not architecture-specific)	`${BSP_ROOT_DIR}/install/usr/include`

For more information about these directories, see the QNX BSP documentation in *Building Embedded Systems*.

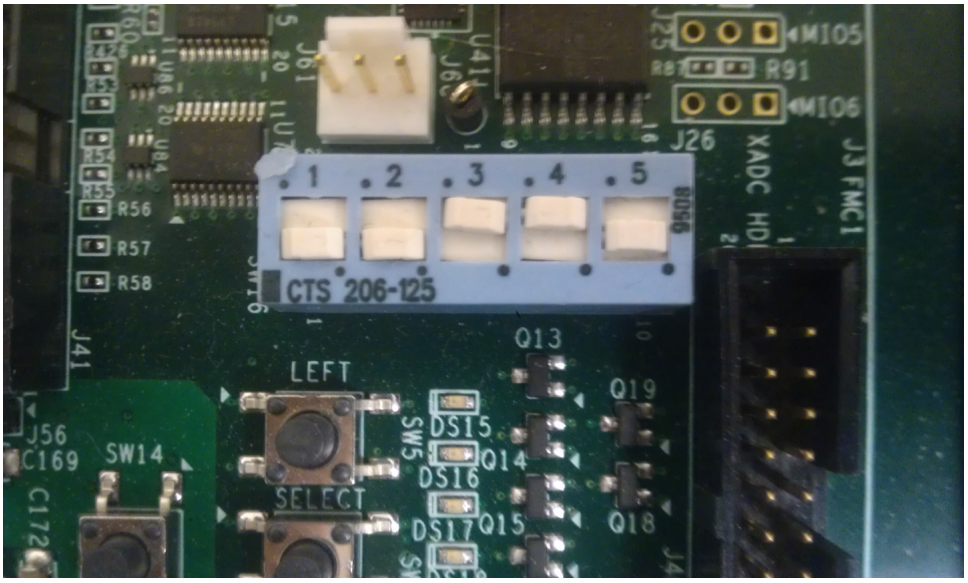
Building and installing the BSP

Building and installing the BSP requires the followings tasks, completed in order.

Task 1. Configure the board's switches and jumpers

The ZC702 board switches should be set as follows for default booting from SD:

Set the SW16 DIP switch bank as follows:



Task 2. Connect the hardware

To connect your board:

1. Connect an RJ-45 Ethernet cable between the Ethernet port of the ZC702 EVM and your local network.
2. Connect the power supply to the ZC702's power connector.
3. Connect the included USB cable between the USB mini-B connector (J16) on the board, and your host machine.
4. Apply power to the board, using the Power on/off slider switch.

© 2013 QNX Software Systems Ltd.

5. On your host machine, determine the port name of the USB port being used for serial communications with the board, and start your favorite terminal program with these settings:
   - baudrate: 115200
   - data: 8 bit
   - parity: none
   - stop: 1 bit
   - flow control: none

Task 3. Build the BSP

You can use the QNX Momentics IDE or the command line to work with this BSP. See below for detailed instructions.

Use the IDE

To work with this BSP using the QNX Momentics IDE on a Windows or Linux system, please refer to the instructions for importing and building the BSP available on the QNX BSP wiki.

Use the command-line

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 zip archive into this new directory.

Note	If you plan to work with multiple different BSPs, we recommend creating a top-level BSP directory, and 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 BSP will build, and the various directories will be created, as described above.

PREBUILT IMAGE

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

The prebuilt IFS image generated with the BSP make file is configured for the various BSP device drivers already running.

When you build the BSP, this prebuilt image will be overwritten with a new IFS that gets generated by the BSP build process, so you may want to make a copy of this prebuilt image for future reference.

However, if you forget to make a copy of the prebuilt image, you can still recover the original prebuilt IFS: simply extract the BSP from the .zip archive into a new directory.

© 2013 QNX Software Systems Ltd.

Build the IPL boot image

To boot the board the board using the QNX IPL, you will need to run the **mkflashimage** shell file in the /images directory. In windows, this is done by invoking a command prompt and executing the following steps:

NOTE: please modify the SDK version (2013.x) in the mkflashimage shell file to match with the correct version of the Xilinx tool.

1. Go to the /images directory of the BSP
2. Run: sh mkflashimage (windows) or . ./mkflashimage (Linux)

This will generate a file called BOOT.bin, which contains the QNX IPL.

Task 4. Prepare a bootable Micro SD Card

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

```
$ mount   (this will show your card's mountpoint - we'll assume for this
           example that it is /dev/sda)
$ umount /dev/sda
$ sudo fdisk /dev/sda
> u
> o
> n
> p
> 1 <ENTER><ENTER>    (default start and end cylinders)
> a
> 1
> t
c          (type is 12 or c, FAT32)
> w

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

Copy the IPL module (BOOT.bin) and the IFS image (QNX-IFS) to the SD card, using a standard ‘cp’ command, in Windows, QNX, or Linux. NOTE – you must copy the ‘BOOT.bin’ file first, followed by the ‘QNX-IFS’ file.

Task 5. Load and run the QNX IFS from the SD card

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

1. Insert the card into the SD/MCC slot on the board.
2. Press the "Power" switch.
3. The terminal should start to output QNX boot info.

© 2013 QNX Software Systems Ltd.

Driver Commands

The table below provides a summary of driver commands, with links to sections in this document with detailed information about each command.

Note	Some of the drivers are 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.

Startup

command: `startup-xzync-zc702`
required binaries: `startup-xzync-zc702`
required libraries:
source location: `src/hardware/startup/boards/xzynq/zc702`

WDT kick

device: Hardware Watchdog
command: `wdtkick`
required binaries: `wdtkick`
required libraries:
source location: `src/hardware/support/xzynq/wdtkick`

RTC utility

device: Real Time Clock
command: `rtc hw`
required binaries: `rtc`
required libraries:
source location: `src/utils/r/rtc`

Serial driver

device: UART1
command: `devc-serxzynq -e -F -S 0xE0001000,82`
required binaries: `devc-serxzynq`
required libraries:
source location: `src/hardware/devc/serxzynq`

© 2013 QNX Software Systems Ltd.

I2C driver

device: I2C1
command:
```
i2c-xzynq -p 0xE0004000 -i 57 --u 1
i2c-xzynq -p 0xE0005000 -i 80 --u 2
```
required binaries: `i2c-xzynq`
required libraries:
source location: `src/hardware/i2c/xzynq`

SPI driver

device: SPI1
command:
```
spi-master -u 0 -d xzynq base=0xE0006000,irq=58
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 driver

device: USB Host
command: `io-usb -d ehci-xzynq ioport=0xe0002100,irq=53`
required binaries: `io-usb, usb, devb-umass`
required libraries: `devu-ehci-xzynq.so, libusbdi.so`
src location: prebuilt binary only

MMCSD driver

device: MicroSD interface
command: `devb-sdmmc-xzynq verbose=5`
required binaries: `devb-sdmmc-xzynq`
required libraries:
source location: `src/hardware/devb/sdmmc`

NETWORK driver

device: Ethernet
command: `io-pkt-v4 -dxzynq -p tcpip`

required binaries: `io-pkt-v4, ifconfig, dhcp.client`
required libraries: `devnp-xzynq.so, libsocket.so`

src location: `src/hardware/devnp/xzynq`

QUAD SPI NOR flash driver

device: NOR flash (QSPI)
command: `devf-norqspi-zc702-xzynq`

© 2013 QNX Software Systems Ltd.

required binaries: `devf-norqspi-zc702-xzynq`
required libraries: N/A
src location: `binary only, source coming soon`

CAN driver

device: CAN interface
command: `dev-can-xzynq -p9 xzynqcan1`
required binaries: `dev-can-xzynq, canctl`
required libraries:
source location: `src/hardware/can/xzynq`

FPGA driver

device: Zynq FPGA
command: `fpga-xzynq`
required binaries: `fpga-xzynq`
required libraries:
source location: `src/hardware/support/xzynq/fpga`

The FPGA can be flashed by writing a bitsteam to /dev/fpga. If the bitstream is valid and successfully written, the DONE LED (DS3) will turn on. Furthermore, the PROG DONE signal can be checked through a devctl message to /dev/fpga. Secure writes can be toggled on/off through devctl messages. Below is an example of flashing the FPGA from the command line.

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

Here is some example code for flashing 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;
}
```

© 2013 QNX Software Systems Ltd.

GPIO driver

device:	GPIO interface
command:	`dev-can-xzynq -t xzynqcan1`
required binaries:	`dev-can-xzynq, canctl`
required libraries:	
source location:	`src/hardware/support/xzynq/gpio`

A library is available through the public header hw/gpio.h, to control every GPIO signal and every pins mux configuration on the board.

Here is some sample code that demonstrates how to blink an LED, via the GPIO library:

```
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();
}
```

© 2013 QNX Software Systems Ltd.

OCM driver

device:	On Chip Memory (OCM) 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. First, you can use the cat command:

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

Or, you can write a C program. Here is some example code for accessing and reading 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);
     lseek(fd, 0, SEEK_SET); /* For future read ... */
     value[min(size-1, MAX_CHAR)] = '\0'; /* Make sure the
     string is terminated by \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;
}
```

© 2013 QNX Software Systems Ltd.

XADC driver

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;
}
```

© 2013 QNX Software Systems Ltd.