# How-To

All of the following steps were performed on a Linux Ubuntu 3.13.0-24 build. Xilinx’s SDK and Petalinux v2015.2 tool suite were previously installed; see that company’s website for documentation how to do so. All necessary drivers for USB to UART and USB to serial cables are likewise assumed to be installed.

## Prepping SD card

Format the SD card as FAT with 1024 byte block sizes. See following links for more detail:

* <http://www.wiki.xilinx.com/Prepare+Boot+Medium>
* <http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.bsp_update.guide%2Ftopic%2Fbsp_660_sd_card.html>

## Create New BOOT.BIN File

To enable debugging with seL4, a means of communicating with the GDB on the host is necessary. The simplest communication method is serial over a UART. Luckily, the ZC702 board comes with two UARTS. Unluckily, only UART1 is enabled by default. However, it is possible to enable UART0 and route the RX and TX signals to externally available pins with a new BOOT.BIN file. Once that is done, UART0 can be used with a serial cable to communicate with the host.

1. Prior to performing any of the following steps issue the following commands to setup the environment:

source <path to petalinux installation>/settings.sh

source /opt/Xilinx/SDK/2015.3/settings64.sh

1. Change directory to the ZC702 project directory, Xilinx-ZC702-2015.2.
2. Copy the FPGA bitstream created by following the instructions in “FGPA how-to” ./fpga.bit.
3. Modify ./components/bootloader/zynq\_fsbl/ps7\_init.c as follows:

Replace all (2) occurrences of:

EMIT\_MASKWRITE(0XF8000154, 0x00003F33U ,0x00001402U)  
with   
EMIT\_MASKWRITE(0XF8000154, 0x00003F33U ,0x00001403U)

Replace all (3) occurrences of:

EMIT\_MASKWRITE(0XF800012C, 0x01FFCCCDU ,0x01ED044DU)  
with   
EMIT\_MASKWRITE(0XF800012C, 0x01FFCCCDU ,0x01FD044DU)

1. Rebuild the First Stage Bootloader (FSBL).

petalinux-build -c bootloader

1. Copy recently built FSBL to current working directory (cwd).

cp ./images/linux/zynq\_fsbl.elf ./fsbl.elf

1. Copy the pre-built u-boot.elf file to cwd.

cp ./pre-built/linux/images/u-boot.elf ./u-boot.elf

1. Delete any old boot files.

rm -rf BOOT.BIN

1. Run command to combine various components into the boot file.

petalinux-package --boot --fsbl fsbl.elf --u-boot=./u-boot.elf --fpga fpga.bit

1. Copy BOOT.BIN to the SD card.

## Host Setup

1. Change directory to the seL4 project.
2. Setup TTY devices for standard I/O and for debug.
   1. “dmesg | grep ttyUSB” to determine which device to connect to. The standard I/O should be cp210x device.
   2. Note this step may have to wait until the ZC702 board is powered up.
   3. The rest of these instructions assume ttyUSB0 is used for debug and ttyUSB1 is used for standard communication.
   4. Enter the following commands to set permissions on the serial devices:

sudo chmod 666 /dev/ttyUSB0

sudo chmod 666 /dev/ttyUSB1

1. Run GDB on the host :

arm-none-eabi-gdb

1. Enter the following at the GDB command prompt:

set serial baud 115200

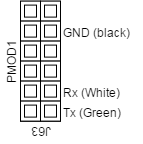
symbol-file stage/arm/zynq7000/bin/hello-2

1. In another host terminal window enter the following command to establish a serial connection to ZC702’s standard I/O:

cu -l /dev/ttyUSB1 -s 115200

After the Host is set up, follow the instructions for booting the ZC702.

## Connecting Target to Host

1. Power off the ZC702 if not already powered off.
2. Connect USB to UART cable for standard I/O communication.
   1. Standard-A to PC (Standard-A).
   2. Mini-B to J17 on the ZC702 board.
3. Connect the USB to serial cable for debug communication.
   1. Black (ground) to J63.9 (target ground).
   2. Green (PC’s TX) to J63.1 (PMOD1\_0, target’s RX).
   3. White (PC’s RX) to J63.3 (PMOD1\_1, target’s TX).

## Booting ZC702

The resulting BOOT.BIN file can be placed on a formatted SD card to boot the ZC702. The following steps assume that a TFTP server with access to the application to load is running on the host at 192.168.1.2. Modify the instructions accordingly if that is not the case.

1. Power off the ZC702 if not already powered off.
2. Set switch bank as follows:  
   
3. Insert SD card with new BOOT.BIN into SD slot.
4. Power on ZC702.
5. Halt u-boot by entering CTRL-C followed by any key.
6. At the u-boot prompt, enter the following commands:

setenv ethaddr aa:bb:cc:dd:ee:ff

setenv ipaddr 192.168.1.3

setenv serverip 192.168.1.2

tftp 20000000 sel4\_image.bin

At this point the target is loaded with the application executable but is still in u-boot.

## Debugging the Application

1. At the GDB command prompt on the host, enter following:

target remote /dev/ttyUSB0

1. At the target terminal (currently at u-boot prompt) enter:

go 20000000

1. At GDB, you should see the following:

(gdb) target remote /dev/ttyUSB2

Remote debugging using /dev/ttyUSB2

breakpoint () at /home/jarvis/sel4DWRC/apps/hello-2/src/gdb-stub.c:572

572 asm("breakinst:");

At target terminal there should be several rows and columns of hexadecimal and decimal numbers, after which should be seen “main: hello world”.

.

.

.

50 0xf8947000 12 0x000001e1 0x000001e2

51 0xf8948000 12 0x000001e2 0x000001e3

52 0xfc000000 20 0x000001e3 0x00000203

53 0xfffc0000 12 0x00000203 0x00000243

54 0x40000000 20 0x00000243 0x00000643

55 0x80000000 20 0x00000643 0x00000a43

main: hello world

1. Use GDB to set a breakpoint at the first printf.
   1. Issue command to display source from GDB.

(gdb) list main.c:400

395 breakpoint();

396

397 printf("\n\tthe 1st key is: %x\n\n",key);

398

399 key = 0xDEADBEEFU;

400 printf("\n\tthe 2nd key is: %x\n\n",key);

401

402 key = 0x1234;

403 printf("\n\tthe 3rd key is: %x\n\n",key);

404

* 1. Set breakpoint, break <filename>:<lineno>.

(gdb) break main.c:397

Breakpoint 1 at 0x9eb8: file /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c, line 397.

* 1. Give the command for the target to resume execution.

(gdb) c

Continuing.

Breakpoint 8, main () at /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c:397

397 printf("\n\tthe 1st key is: %x\n\n",key);

1. View and modify memory.
   1. Get the address of the global variable ‘key’.

(gdb) info addr key  
Symbol "key" is static storage at address 0x373d0.

* 1. View value in memory.

(gdb) x 0x373d0

0x373d0 <key>: 0x00000000

* 1. Change value in memory.

(gdb) set \*0x373d0 = 0x333

* 1. View updated value in memory.

(gdb) x 0x373d0

0x373d0 <key>: 0x00000333

* 1. Use var keyword to view ‘key’.

(gdb) p key

$6 = 819

1. Step and confirm target print’s pit the new key value.

(gdb) step

399 key = 0xDEADBEEFU;

The target’s UART connection should display:

the 1st key is: 333

1. Step again.

(gdb) step

400 printf("\n\tthe 2nd key is: %x\n\n",key);

1. Use ‘var’ keyword to change key value.
   1. Show current key value.

(gdb) p key

$7 = 3735928559

* 1. Update key value.

(gdb) set var key = 0xb0bafe11

* 1. Show updated key value .

(gdb) x 0x373d0

0x373d0 <key>: 0xb0bafe11

1. Step and confirm target print’s pit the new key value.

(gdb) step

402 key = 0x1234;

The target’s UART connection should display:

the 2nd key is: b0bafe11

1. List the last 10 lines of main().

(gdb) list main.c:410

405

406 /\* never exit \*/

407 while(1)

408 {

409 key++;

410 printf("\n\tthe nth key is: %x\n\n",key);

411 while(key > 100);

412 }

413

414 return 0;

1. Set a breakpoint at the inner while loop.

(gdb) break main.c:411

Breakpoint 2 at 0x9f2c: file /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c, line 411.

1. Disassemble the source code around the inner while loop.

(gdb) disassemble 0x9f2c,+32

Dump of assembler code from 0x9f2c to 0x9f4c:

0x00009f2c <main+2084>: ldr r3, [r4, #-4000] ; 0xfa0

0x00009f30 <main+2088>: cmp r3, #100 ; 0x64

0x00009f34 <main+2092>: bhi 0x9f2c <main+2084>

0x00009f38 <main+2096>: b 0x9f14 <main+2060>

0x00009f3c <main+2100>: muleq r1, r4, r9

0x00009f40 <main+2104>: ; <UNDEFINED> instruction: 0x0002c3b0

0x00009f44 <main+2108>: andeq lr, r1, r4, lsr #19

0x00009f48 <main+2112>: andeq lr, r1, r12, lsr #19

End of assembler dump.

1. Set breakpoint at the ‘bhi’ instruction:

(gdb) break \*0x9f34

Breakpoint 3 at 0x9f34: file /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c, line 411.

1. Allow the target to run freely until it hits the next breakpoint.

(gdb) c

Continuing.

Breakpoint 2, main () at /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c:411

411 while(key > 100);

The target’s UART connection should display:

the 3rd key is: 1234

the nth key is: 1235

1. Allow the target to free run until it hits the breakpoint on the ‘bhi’ instruction, the last point to change key’s value before getting stuck in the inner while loop.

(gdb) c

Continuing.

Breakpoint 3, 0x00009f34 in main () at /home/jarvis/sel4/s el4DWRC/apps/hello-2/src/main.c:411

411 while(key > 100);

1. Change key to allow from some additional printf’s.

(gdb) set var key = 95

1. Delete the last breakpoint and display breakpoint information.

(gdb) del 3

(gdb) info break

Num Type Disp Enb Address What

1 breakpoint keep y 0x00009eb8 in main at /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c:397

breakpoint already hit 1 time

2 breakpoint keep y 0x00009f2c in main at /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c:411

breakpoint already hit 1 time

1. Allow the target to run freely.

(gdb) c

Continuing.

Breakpoint 2, main () at /home/jarvis/sel4/sel4DWRC/apps/hello-2/src/main.c:411

411 while(key > 100);

The target’s UART connection should display:

the nth key is: 60

1. Delete this breakpoint.

(gdb) del 2

1. Allow the target to run freely.

(gdb) c

Continuing.

The target’s UART connection should display:

the nth key is: 61

the nth key is: 62

the nth key is: 63

the nth key is: 64

the nth key is: 65

At this point the target is free running, albeit in a never ending while loop, but the current version of the gdb-stub no longer has a means to halt the target for further interactions, this concluding this example.