Debug U-boot

Xilinx Wiki

Exported on 07/02/2021

Table of Contents

Xilinx SDK System Debugger supports source level debugging of self-relocating programs, like u-boot as described in the AR66591[1] or in the Help[2] documentation. This wiki page is an step-by-step guideline of this documents using U-Boot as an example of self relocated application.

Table of Contents

1 https://www.xilinx.com/support/answers/66591.html
2 https://www.xilinx.com/html_docs/xilinx2017_1/SDK_Doc/index.html

1 Build U-Boot

U-Boot build process, including source fetching and tool configuration, is well documented on the wiki page[3].
Note: *MPSoC U-Boot build process generates two ELF files,* ***u-boot*** *(before applying relocation) and* ***u-boot.elf*** *(after applying relocation to the* bin[4] *file and get back to ELF file). The final U-Boot (u-boot.elf) file does not include the symbols present in the code so for debugging purposes u-boot file have to be used when remapping.*

1.1 Optimizations

By default U-Boot makefiles are set to optimize the code so the debug process might not follow the C source file and local variables might not be visible for the debugger. Unfortunatelly removing optmization globally is broken in the current U-Boot source code, however more specific changes can be done to get better debugging experience. The proposed workaround is to modify the required driver/feature Makefile to modify the compilation flags. i.e. / drivers/mmc/Makefile

```
ccflags-y = -O0 -g3 -fno-inline -DDEBUG
```

3 https://xilinx-wiki.atlassian.net/wiki/display/A/Build+U-Boot
4 https://github.com/theopolis/u-boot/blob/master/doc/README.arm64

2 Debug U-Boot

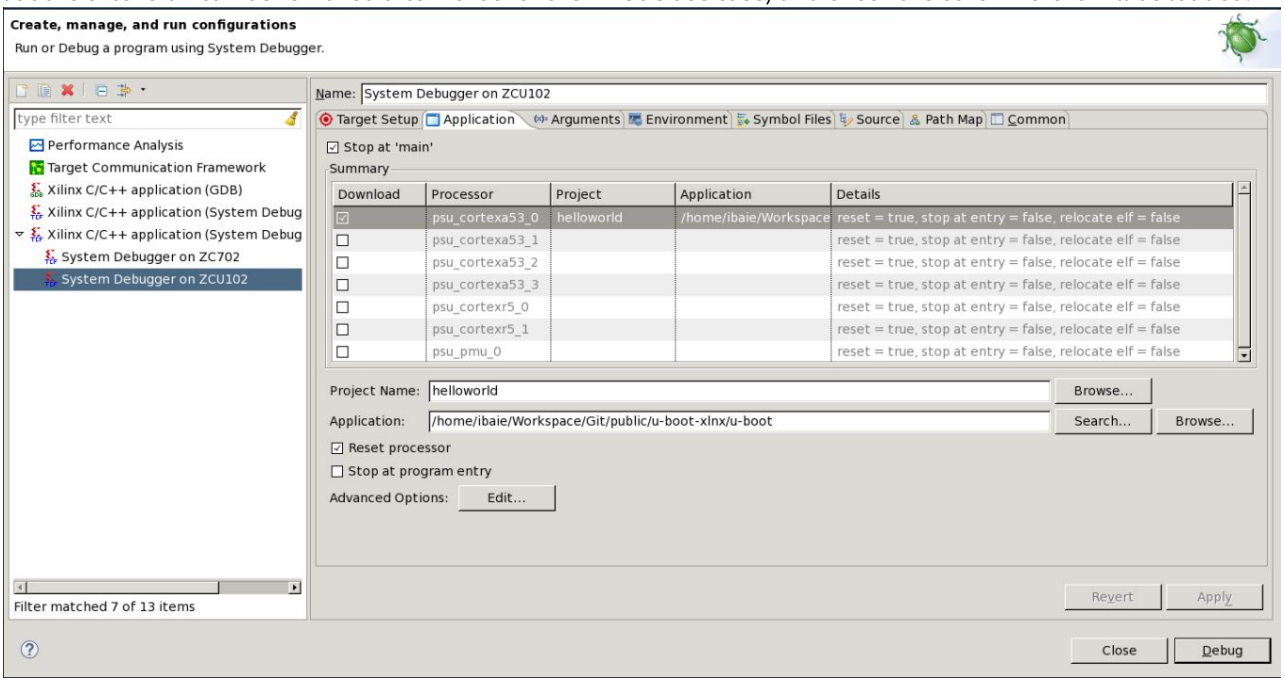
2.1 Debug as standalone application

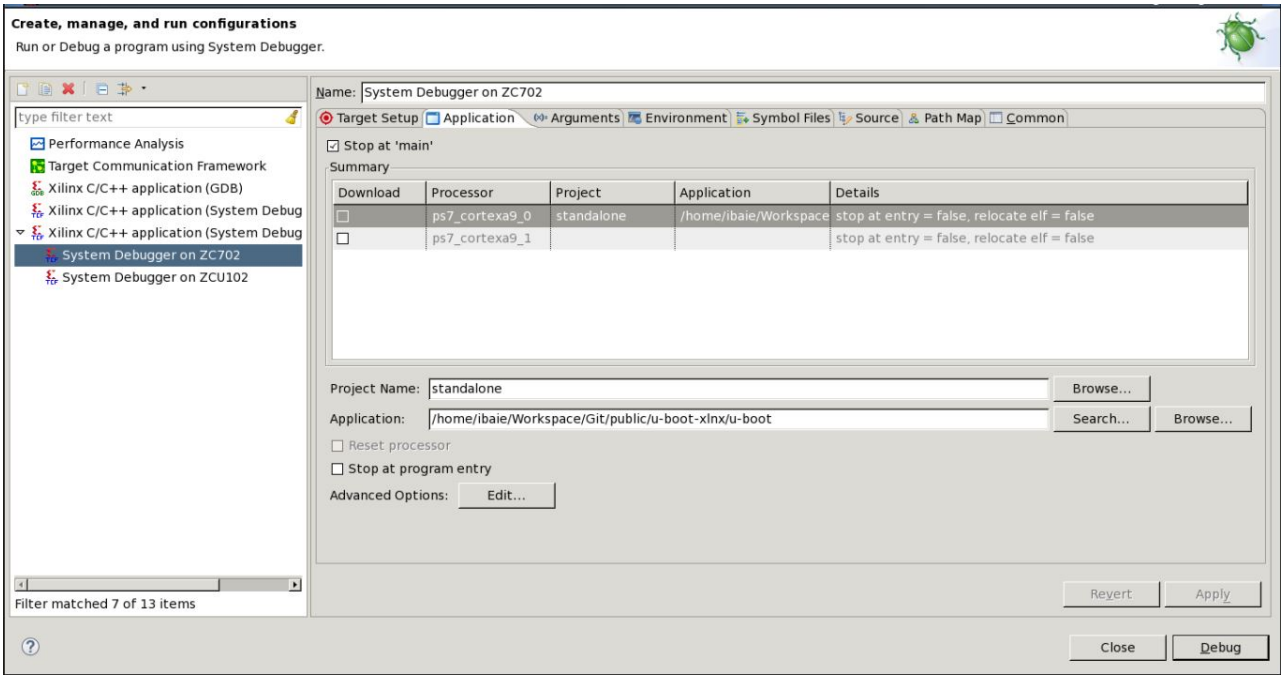
As U-Boot commonly is build out of the SDK IDE, "Standalone Application Debug" requires to at least to have a hardware platform project in the workspace in order to run the initialization script. This use case maps the symbol file to the debug session so it does not require any command be ussed from the XSCT console.

> **ZynqUS+ Users**
>
> Debugging U-boot as a standalone application is not fully possible on ZynqUS+ (ZynqMP) devices due to dependencies on other software components (e.g. ATF and PMU-FW). For ZynqUS+ you should attach to a running target as described in the "Debug running target" section.

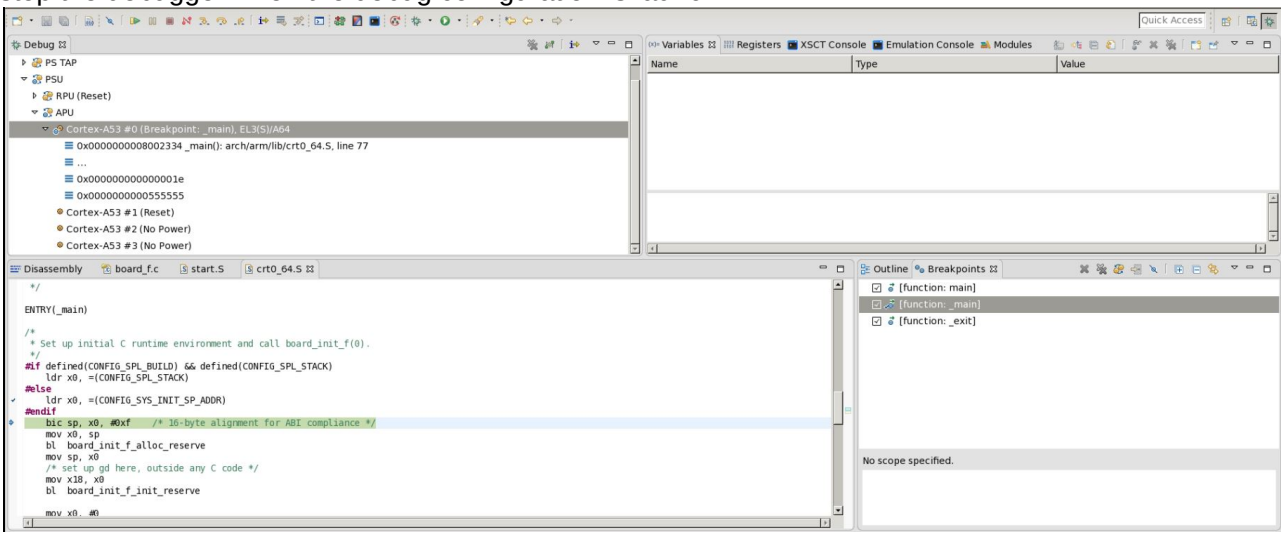
2.1.1 Debug before Relocation

In the Application tab select the U-Boot executable file using the Browse feature (file browser force to use .elf files, but the extension can be removed afterwards for the MPSoC use case) and check the core where it will be loaded.



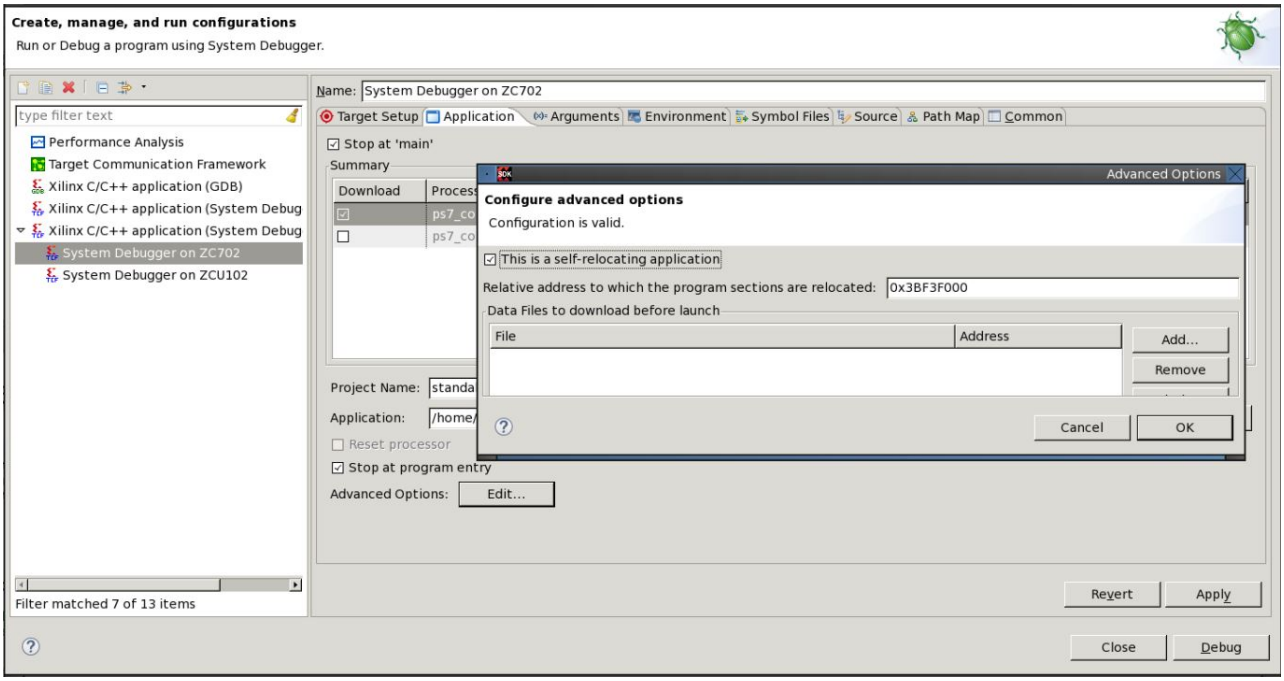


By default the Debug configuration adds two breakpoints at main and _exit functions, however the main entry function for U-Boot is _main rather than main, which means that the debugger will not stop by default ot it. Add a function breakpoint in _main or use the "Stop at program entry" option in order to stop the debugger when the debug configuration is launch.

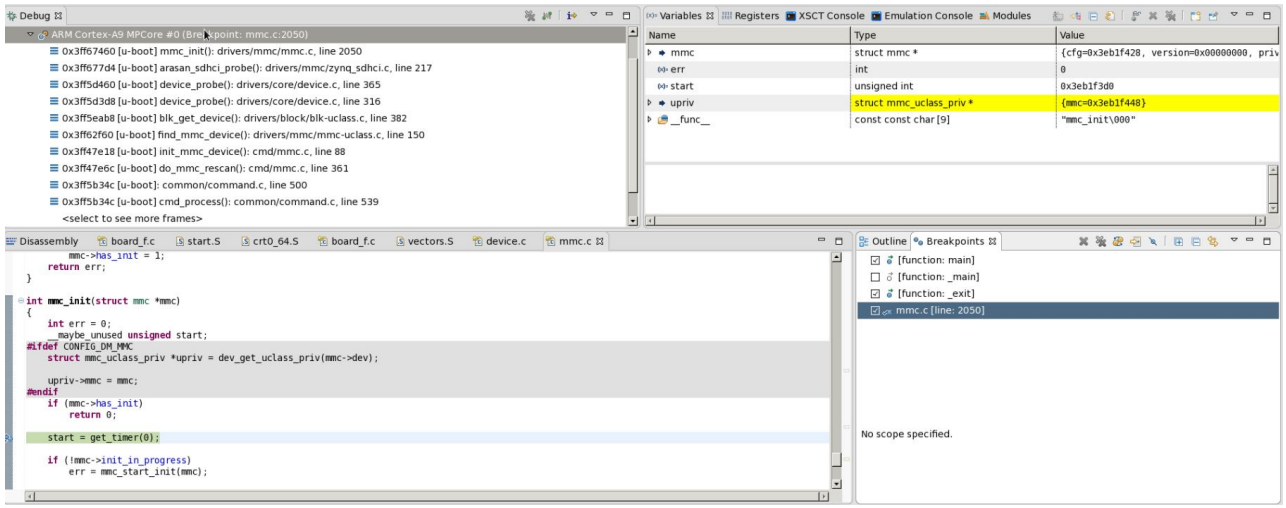


2.1.2 Debug after relocation

For debugging the U-Boot code after the relocation, the debug configuration provides an advanced options menu where relocation information can be set. Get the relocation offset from the running U-Boot command line using bdinfo tool, and set the offset in the advanced tabs according to that number.




```
Zynq> bdinfo
arch_number = 0x00000000
boot_params = 0x00000000
DRAM bank   = 0x00000000
-> start    = 0x00000000
-> size     = 0x40000000
baudrate    = 115200 bps
TLB addr    = 0x3FFF0000
relocaddr   = 0x3FF3F000
reloc off   = 0x3BF3F000
irq_sp      = 0x3EB1EED0
sp start    = 0x3EB1EEC0
ARM frequency = 666 MHz
DSP frequency = 0 MHz
DDR frequency = 533 MHz
Early malloc usage: 49c / 800
```



2.2 Debug running target

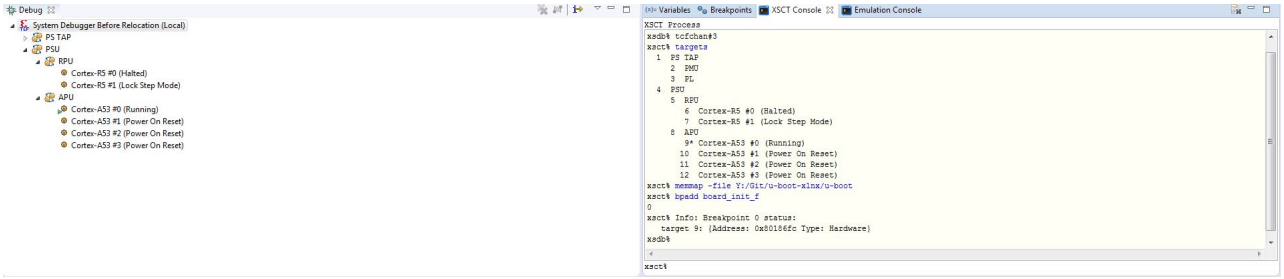
This Debug configuration type can be used either for running targets where U-Boot has been loaded using a QSPI or SD boot modes, or for JTAG bood mode where a script is used for the loading process (i.e. MPSoC where PMUFW and ATF are being loaded before U-Boot).

> **ZynqUS+ Users**
>
> An example Tcl script for JTAG booting a ZynqUS+ (ZynqMP) device is provided in Ch. 10 of UG1137 (Zynq UltraScale+ MPSoC: Software Developers Guide). As of v11.0 of this document this was under the "Loading PMU firmware in JTAG Boot Mode" section and although the focus of that chapter is the PMU-FW the provided script demonstrated loading all boot firmware (i.e. PMU-FW, FSBL, ATF, U-boot).

2.2.1 Debug before relocation

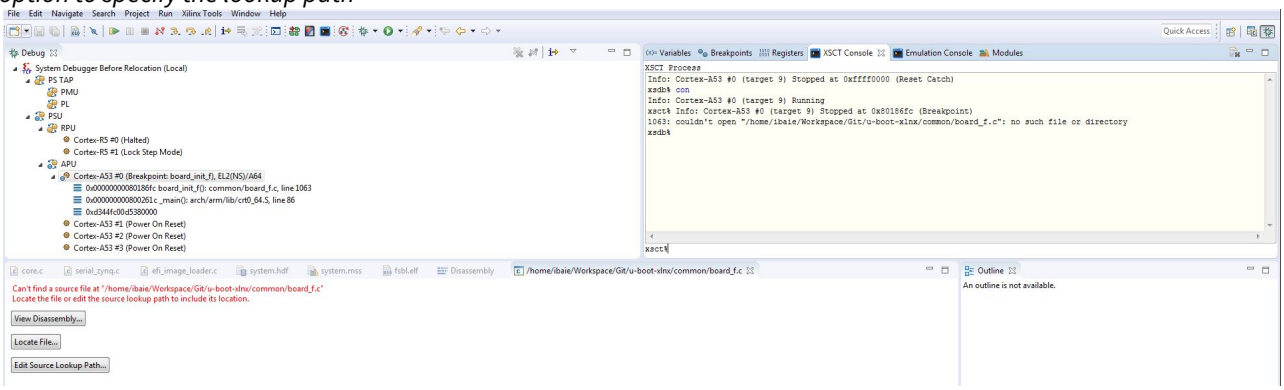
The symbols present on the ELF file match with the with the runtime state before the self-relocation is performed, so this means that there is no remmapping requirement for this use case and the information on the ELF file can be used straightforward for debugging purposes. Nevertheless the main issue for this use case, is stopping the processor before the relocation is done, as there is a small window of time to connect the debugger. The easiest way is just set the a breakpoint once U-Boot is relocated and then drive a warm reset to make the processor stop in the breakpoint.

- Boot the board with the BOOT binary using a SD card
- Generate a new Debug configuration for running target and launch it
- Connect to the target using XSCT console
- mmap the u-boot ELF file and add a breakpoint in the board_init_f function (Function prior the relocation)



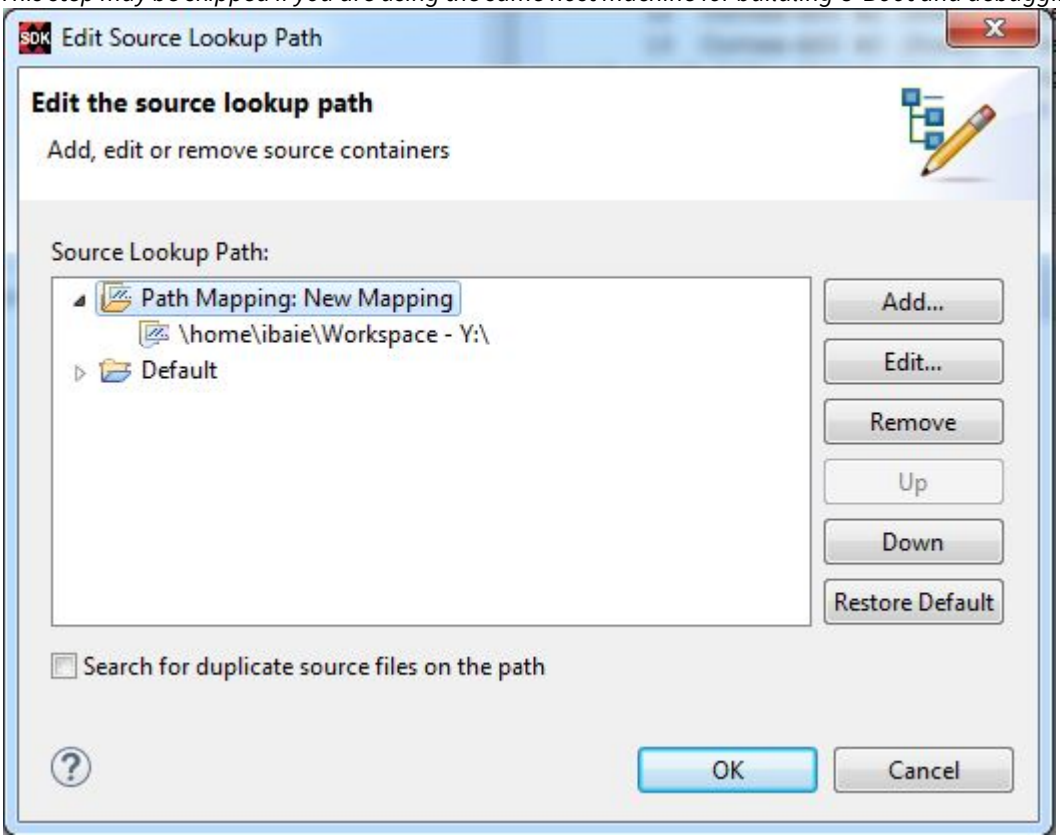
- Perform a reset typing "reset" command in the U-Boot command line console

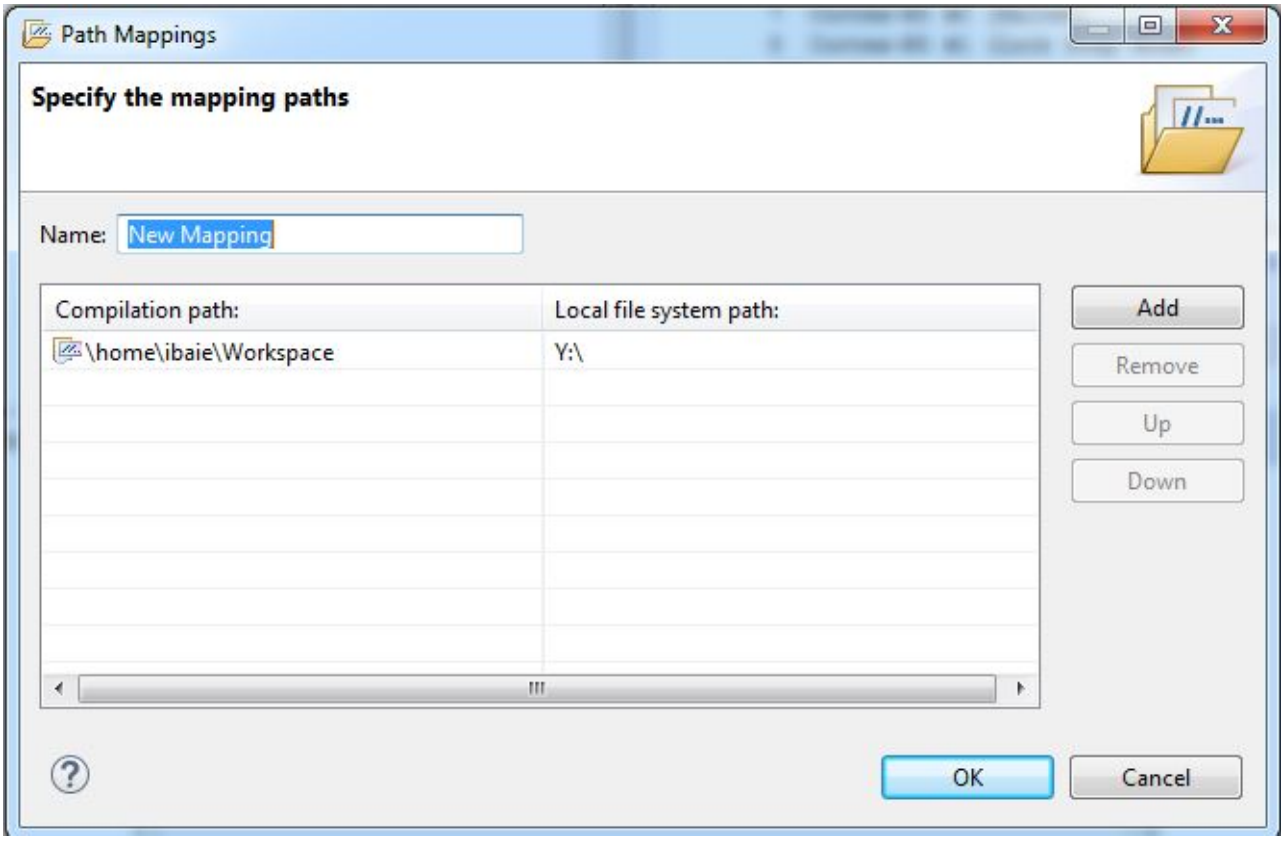
Once the symbol file is associated with the execution, the debugger will identify the current execution point and also will open the source file where the core is stop. In case of cannot find the source file it will point a error message and option to specify the lookup path



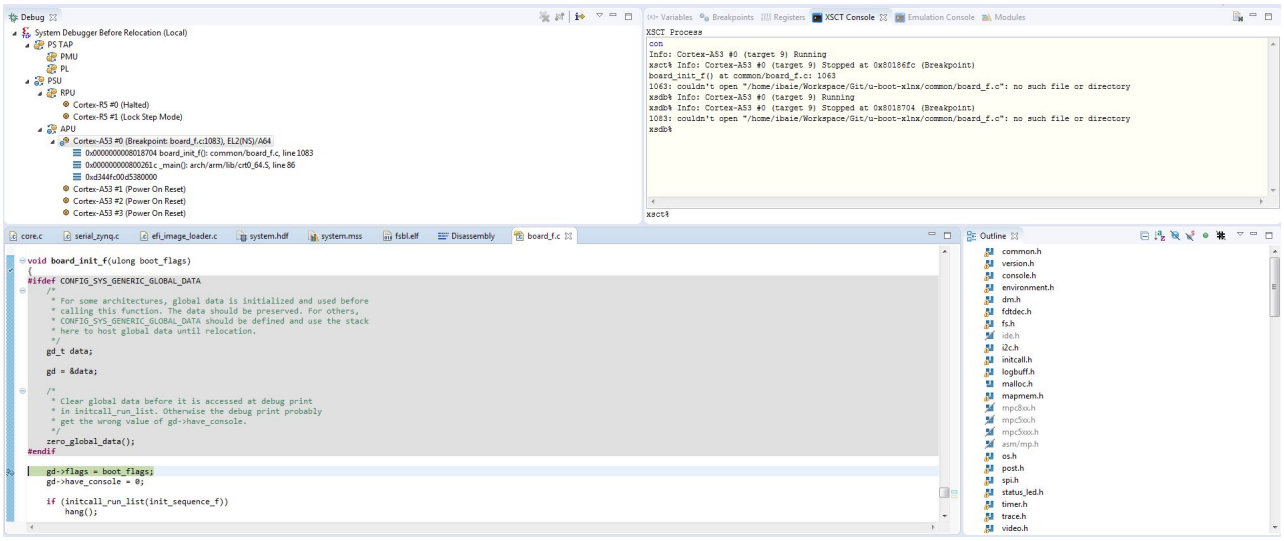
- Edit the source lookup Path to find the source files from where U-Boot was compiled

This step may be skipped if you are using the same host machine for buildling U-Boot and debugging the application





- Debug the application



2.2.2 Debug after relocation

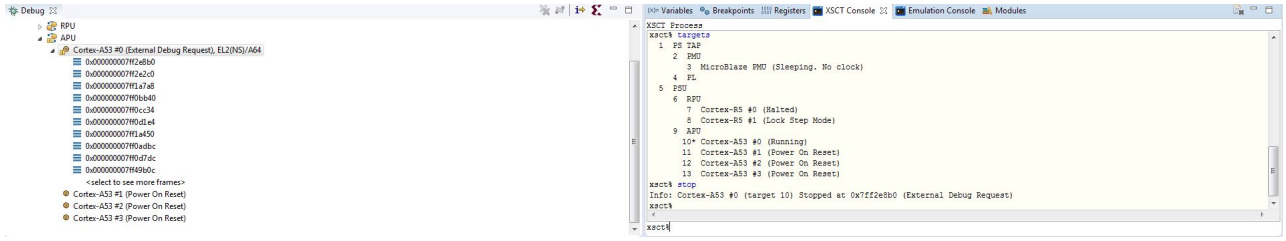
After the relocation is done, the symbols present on the ELF file does not match with the with the runtime state, so the mapping have to be relocated for debugging purposes.

- Boot the board with the BOOT binary using a SD card
- Check the relocation offset using bdinfo command in the U-boot console

```
ZynqMP> bdinfo
arch_number = 0x00000000
boot_params = 0x00000000
DRAM bank   = 0x00000000
-> start    = 0x00000000
-> size     = 0x80000000
baudrate    = 115200 bps
TLB addr    = 0x7FFE0000
relocaddr   = 0x7FF25000
reloc off   = 0x77F25000
irq_sp      = 0x7DEE4DD0
sp start    = 0x7DEE4DD0
ARM frequency = 33 MHz
DSP frequency = 0 MHz
DDR frequency = 0 MHz
Early malloc usage: ba8 / 8000
```

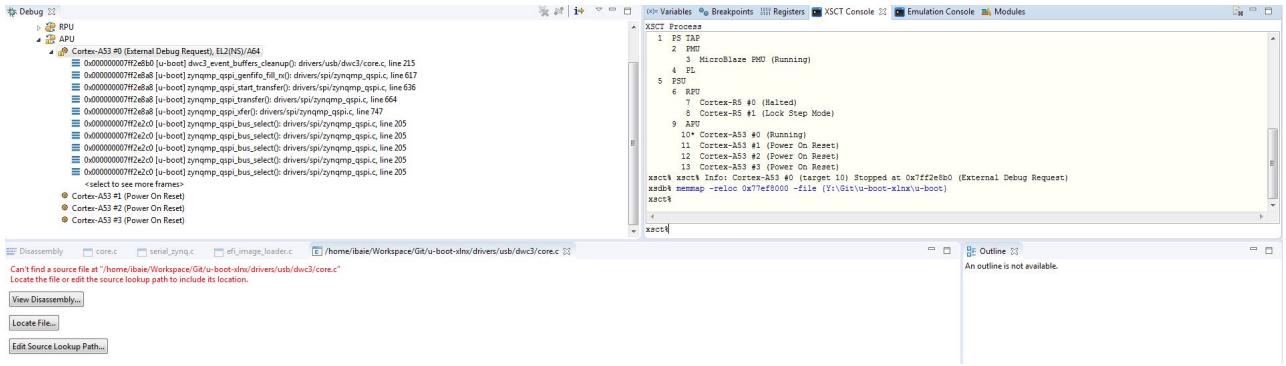
- Open a SDK workspace and generate a Debug configurtion for running target
- Generate a new Debug configuration for running target and launch it
- Connect to the target using the XSCT Console and stop the running processor

By default the debugger is not able to identify the code by itself, so there is no symbols or source files associated to the execution



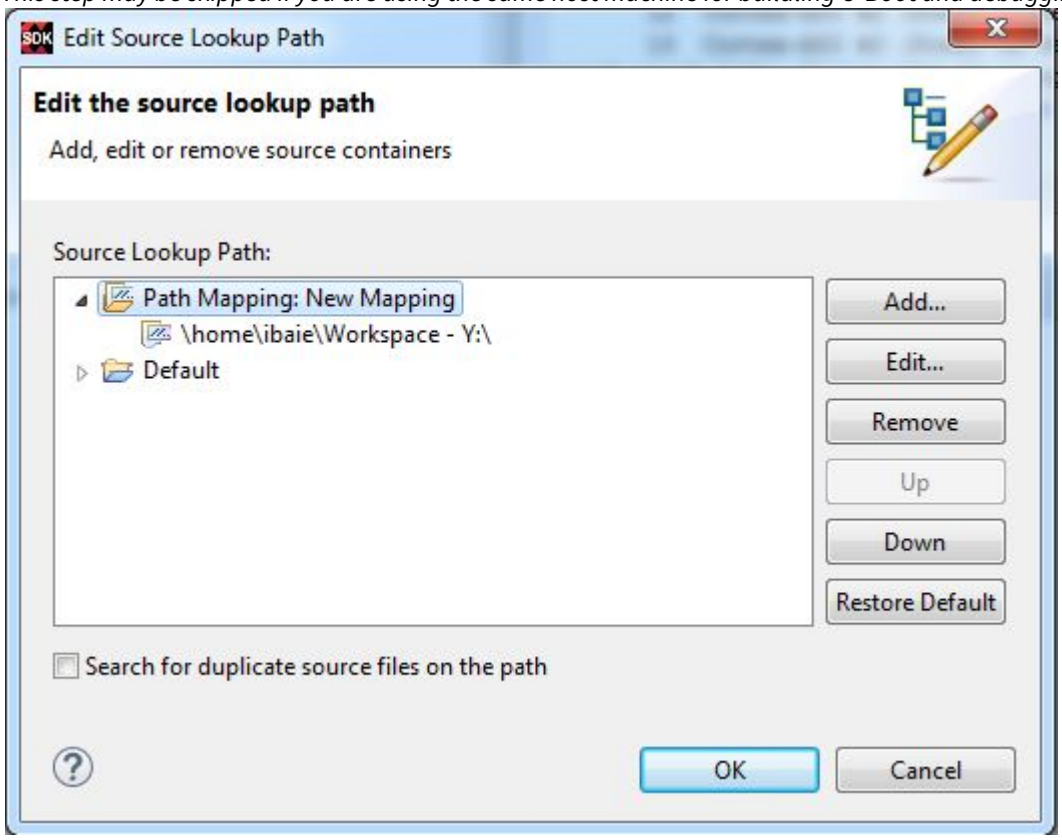
- Add the symbol file with the relocation offset

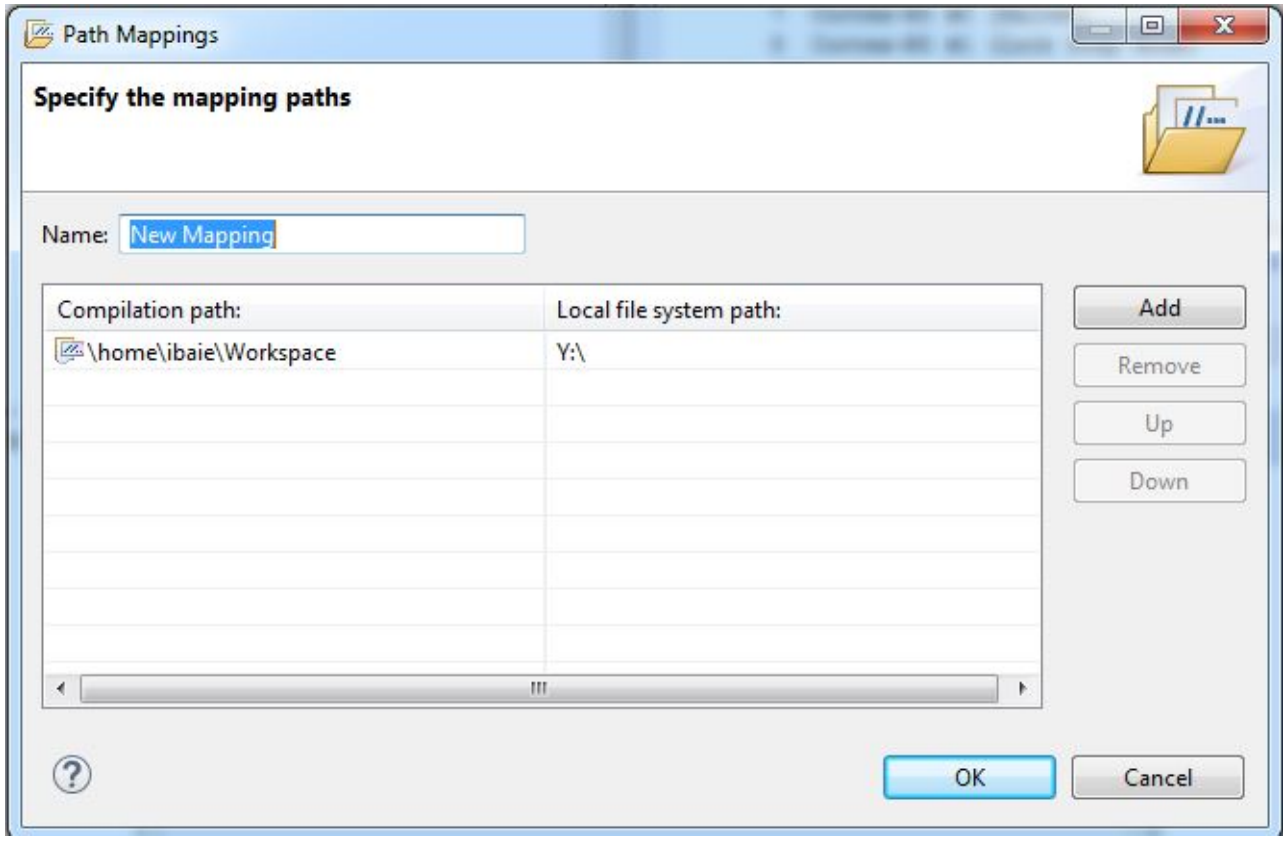
Once the symbol file is associated with the execution, the debugger will identify the current execution point and also will open the source file where the core is stop. In case of cannot find the source file it will point a error message and option to specify the lookup path



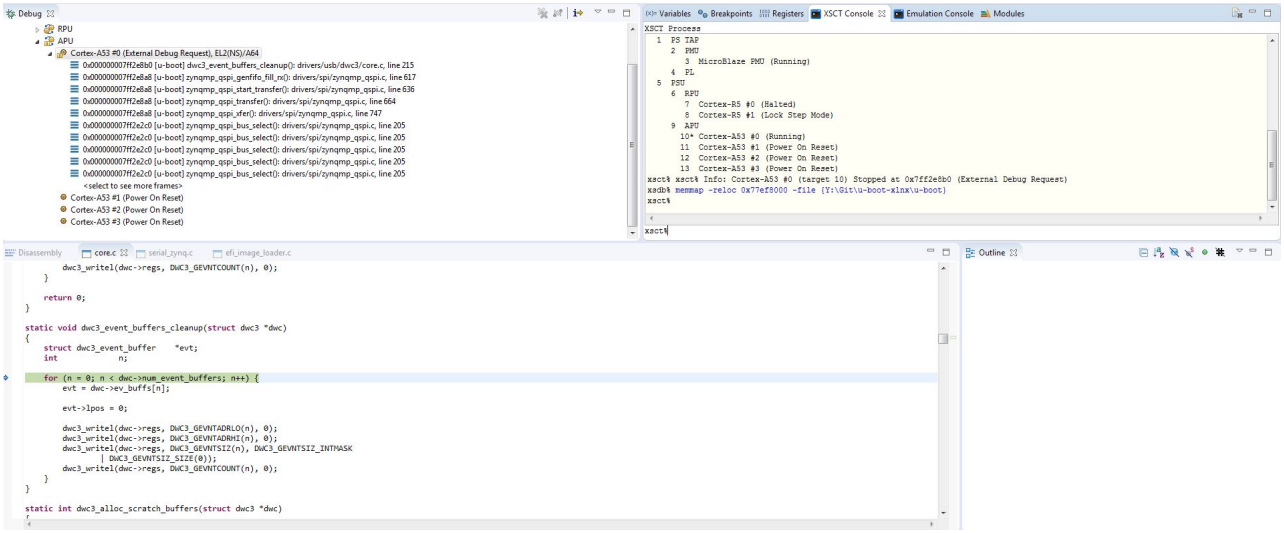
- Edit the source lookup Path to find the source files from where U-Boot was compiled

This step may be skipped if you are using the same host machine for buildling U-Boot and debugging the application





- Debug the application

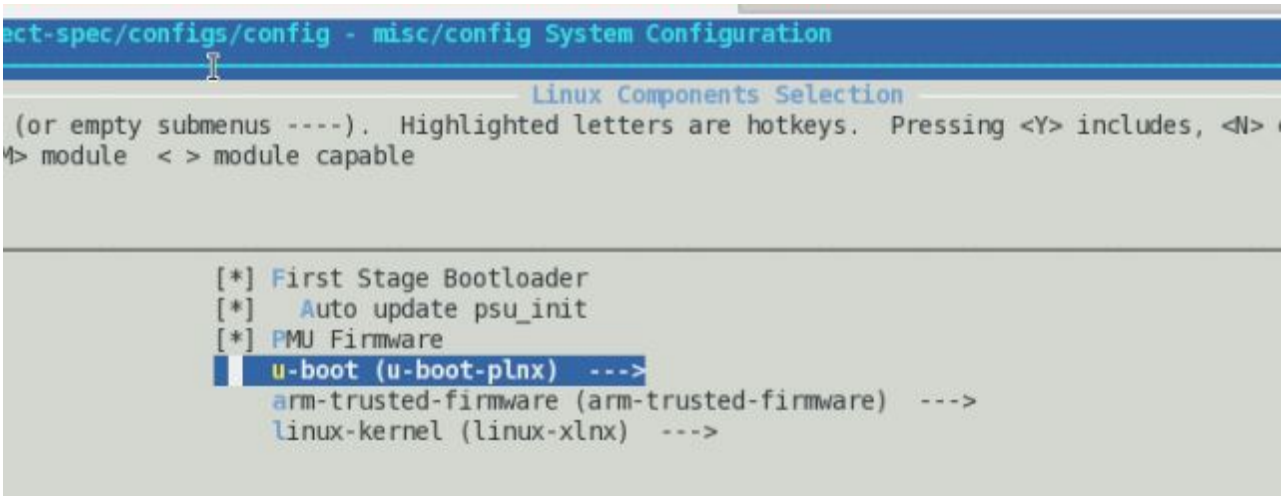


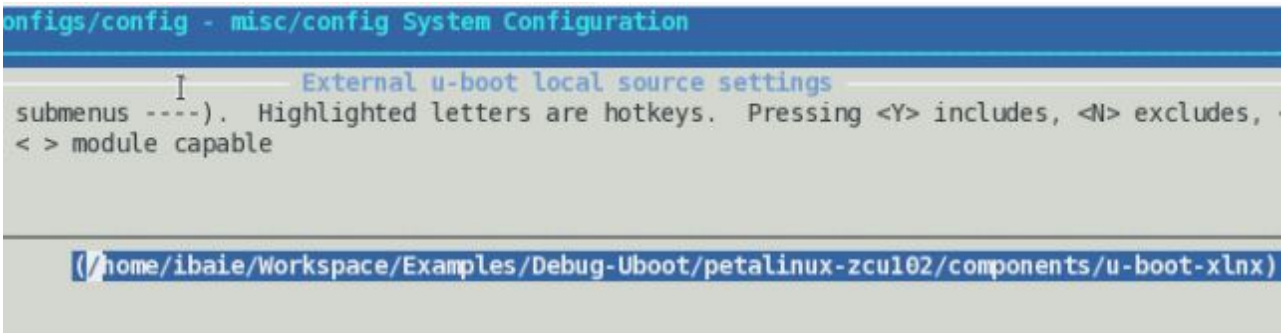
3 Petalinux Notes

When Petalinux is used to build U-Boot (as well as all the images required to run a Linux image in the target), the default configuration does not provide neither the source files or the intermediate ELF file (the one with the debugging information) within the project folders, so some changes may be applied to have a debugging features.

There are different way to get access to the U-Boot sources when using Petalinux as build system, in this example the external source feature is used to point to a cloned repository. This method provides an easy way to modify the sources as well as to have access to the code within the petalinux project folder.

```
$ petalinux-create -t project -s <Path-to-BSP-File> -n petalinux
$ cd petalinux/components
$ git clone https://github.com/Xilinx/u-boot-xlnx.git
$ cd u-boot-xlnx
$ git checkout tags/xilinx-v2017.1 -b v2017.1
$ petalinux-config
```





Once the project is build, the u-boot ELF file with the compleate symbol table is located in the build directory, so it can be copied to the deployment folder for easy usage.

```
xir-psgpwts06:/home/ibaie/Workspace/Examples/Debug-Uboot/petalinux-zcu102 $ cat build/conf/plnxtool.conf | grep TMP
TMPDIR = "/tmp/petalinux-zcu102-2017.05.23-09.09.16"
xir-psgpwts06:/home/ibaie/Workspace/Examples/Debug-Uboot/petalinux-zcu102 $ find /tmp/petalinux-zcu102-2017.05.23-09.09.16/work -name "u-boot"
/tmp/petalinux-zcu102-2017.05.23-09.09.16/work/plnx_aarch64-xilinx-linux/u-boot-xlnx/v2017.01-xilinx-v2017.1+git999-r0/u-boot-xlnx-v2017.01-xilinx-v2017.1+git999/u-boot
xir-psgpwts06:/home/ibaie/Workspace/Examples/Debug-Uboot/petalinux-zcu102 $ cp /tmp/petalinux-zcu102-2017.05.23-09.09.16/work/plnx_aarch64-xilinx-linux/u-boot-xlnx/v2017.01-xilinx-v2017.1+git999-r0/u-boot-xlnx-v2017.01-xilinx-v2017
.1+git999/u-boot images/linux/
xir-psgpwts06:/home/ibaie/Workspace/Examples/Debug-Uboot/petalinux-zcu102 $
```

The same steps described in the Debug section can be used once the source files and u-boot ELF location is well defined.

4 Related Links

- U-Boot[5]
- Debugging U-Boot with SDK[6]
- SDK - How to debug applications that self relocate[7]
- Using Xilinx SDK[8]

5 https://xilinx-wiki.atlassian.net/wiki/display/A/U-Boot
6 https://www.xilinx.com/video/hardware/debugging-u-boot-with-sdk.html
7 https://www.xilinx.com/support/answers/66591.html
8 https://www.xilinx.com/html_docs/xilinx2017_1/SDK_Doc/index.html