

# **ARM DS-5 Tools and Avnet ZED Series**

**#6**

## **Debug a Linux Kernel using DSTREAM and Avnet ZedBoard or MicroZed**



Sept 2013  
Version 01

## Table of Contents

ARM DS-5 Tools and Avnet ZED Series .....	3
Required Installations .....	4
Technical Support .....	5
Debugging the Linux Kernel .....	6
Debugging the Linux Kernel .....	7
Adding Simultaneous Application Debug .....	13
Appendix I – Creating the vmlinux Symbol File .....	14
Revision History .....	19
Resources .....	19

## ARM DS-5 Tools and Avnet ZED Series

This tutorial is one in a series of step by step instruction manuals. Together they document the procedures necessary to utilize the ARM Development Studio 5 (DS-5™) Software Suite and the DSTREAM Debugging tools with the Avnet Zynq Evaluation and Development (ZED) boards. These tutorials can be used on their own, or in combination with Avnet online videos and OnRamp Technical Session™.

The ARM software and hardware tools provide a powerful debugging suite for processor-based systems built around the dual Cortex-A9 cores present in the Xilinx Zynq SoC, at the heart of the Avnet ZED boards. A Linux software developer can simultaneously debug applications and kernel module code, with separate control over each thread. You can step through Linux boot code, first stage bare metal boot code, and bare metal applications. When used in concert with the Xilinx Vivado tools for FPGA fabric development, the ARM debugger and Internal Logic Analyzer (ILA) IP can be cross-triggered to stop on software and hardware breakpoints, or when a hardware event occurs. For difficult-to-isolate intermittent faults, DS-5 provides access to the Cortex-A9 on-chip Trace facility. Once your embedded system is running correctly, DS-5 uses Streamline, a graphical system profiler, to identify performance bottlenecks in your design to ensure top-shelf operation.

This tutorial series begins with the most basic tool configuration and board connection. It takes you all the way through to the most complex aspects of hardware/software co-debugging to root out design errors that are otherwise apparent only in very complex use cases, or worse, after a product is released. Together the ARM DS-5 tools, Xilinx Vivado and Avnet ZED boards provide an unparalleled combination to compress design timelines, cut project costs and optimize your product for the marketplace.

## Required Installations

### Software

The recommended software for this tutorial series is:

- ARM Development Studio 5 (Exact version used is 5.14, build 1702)
- Xilinx ISE WebPACK 14.5 (Free license and download from Xilinx website)
- Cypress CY7C64225 USB-to-UART Bridge Driver (for ZedBoard serial output)
- Silicon Labs CP2104 USB-to-UART Bridge Driver (for MicroZed serial output)
- Tera Term (Exact version used is V4.75)
- Xilinx Software Development Kit, version 14.5
- For hardware/software co-debugging, Xilinx Vivado 2013.2

### Hardware

The targeted hardware consists of the following:

- PC workstation with at least 5 GB RAM, 30GB free hard disk space, Windows 7 64-bit operating system, and a wired GB Ethernet connection
- Available SD card slot on PC or external USB-based SD card reader
- One of:
  - Avnet ZedBoard Kit (**AES-Z7EV-7Z020-G**)
    - USB cable (Type A to Micro-USB Type B)
    - 4GB SD card
    - 12v Power supply
  - Avnet MicroZed Kit (**AES-Z7MB-7Z010-G**)
    - USB cable (Type A to Micro-USB Type B)
    - 4GB SD card
- Avnet ZedBoard Debug Adapter Kit (**AES-ZBDB-ADPT-G**)
  - 14-pin Xilinx PC4 ribbon cable
- ARM DSTREAM unit and Keil pod with wide cable connector
  - 20-pin JTAG ribbon cable
  - USB cable (Type A to Printer)
  - 5v Power supply
- CAT-5 Ethernet cable

## Technical Support

For technical support with any of the instructions, please contact your local Avnet/Silica FAE or visit the support forums:

<http://www.zedboard.org/forum>

<http://www.microzed.org/forum>

Additional technical support resources are listed below.

*ZedBoard Kit/MicroZed Kit support page with Documentation and Reference Designs*

<http://www.zedboard.org/content/support>

<http://www.microzed.org/content/support>

For Xilinx technical support, you may contact your local Avnet/Silica FAE or Xilinx Online Technical Support at [www.support.xilinx.com](http://www.support.xilinx.com) . On this site you will also find the following resources for assistance:

- Software, IP, and Documentation Updates
- Access to Technical Support Web Tools
- Searchable Answer Database with Over 4,000 Solutions
- User Forums
- Training - Select instructor-led classes and recorded e-learning options

Contact your Avnet/Silica FAE or Avnet Support for any additional questions regarding the reference designs, kit hardware, or if you are interested in designing any of the kit devices into your next design.

<http://www.em.avnet.com/techsupport>

For ARM technical support, you may contact your local Avnet/Silica FAE or ARM Online Technical Support at [www.arm.com/support](http://www.arm.com/support) .

## Debugging the Linux Kernel

In this tutorial we will boot the ZED target with a standard Linux kernel that was generated in the Avnet Linux for ZedBoard Speedway course, and then connect DS-5 to it through the DSTREAM for source level debugging. Once we have a kernel debugging session established, you can use the information in tutorial #5 to establish a simultaneous connection to the target and debug both a Linux application and Linux kernel at the same time. This can be very useful when developing device drivers for Linux.

Set the ZED target Boot Mode to SD boot as shown below:

### For ZedBoard:

To begin the procedure, set the ZedBoard Boot Mode to SD boot using jumpers JP11 to JP7 set to the following:

	<b>JP11</b>	<b>JP10</b>	<b>JP9</b>	<b>JP8</b>	<b>JP7</b>
Position	SIG-GND	3V3-SIG	3V3-SIG	SIG-GND	SIG-GND

### For MicroZed:

To begin the procedure, set the MicroZed Boot Mode to JTAG only using jumpers JP3 to JP1 set to the following:

	<b>JP3</b>	<b>JP2</b>	<b>JP1</b>
Position	2-3	2-3	1-2

You should have your DSTREAM, ZED target, ZedBoard Adapter and host PC connected together as described in the tutorial #1. Also connect an Ethernet cable between the target and host PC, as per the instructions in tutorial #5.

You will need application and SD card files to complete these instructions. The files are part of the download package that included this tutorial. Browse to the location on your host where the download package was decompressed, and look for the following folder:

**<Download package folder>\Support06**

For the purposes of this tutorial we will assume the files exist in folder:

**C:\OnRamps\Support06**

To prepare to boot the target, copy all the files in the following folders to the root folder of your SD card:

**For ZedBoard: C:\OnRamps\Support06\ZedBoard**

**For MicroZed: C:\OnRamps\Support06\MicroZed**

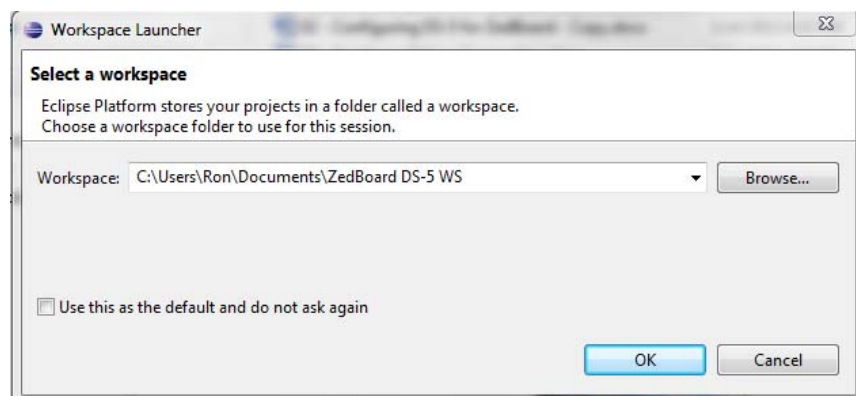
**For Both: C:\OnRamps\Support06\Common**

Insert the card into the SD card slot on the underside of the target.

The only change made to the files from tutorial #5 is that the Linux kernel (ulmage) has been rebuilt with debug symbols. If you would like to do this on your own instead of using the supplied files, please see Appendix I.

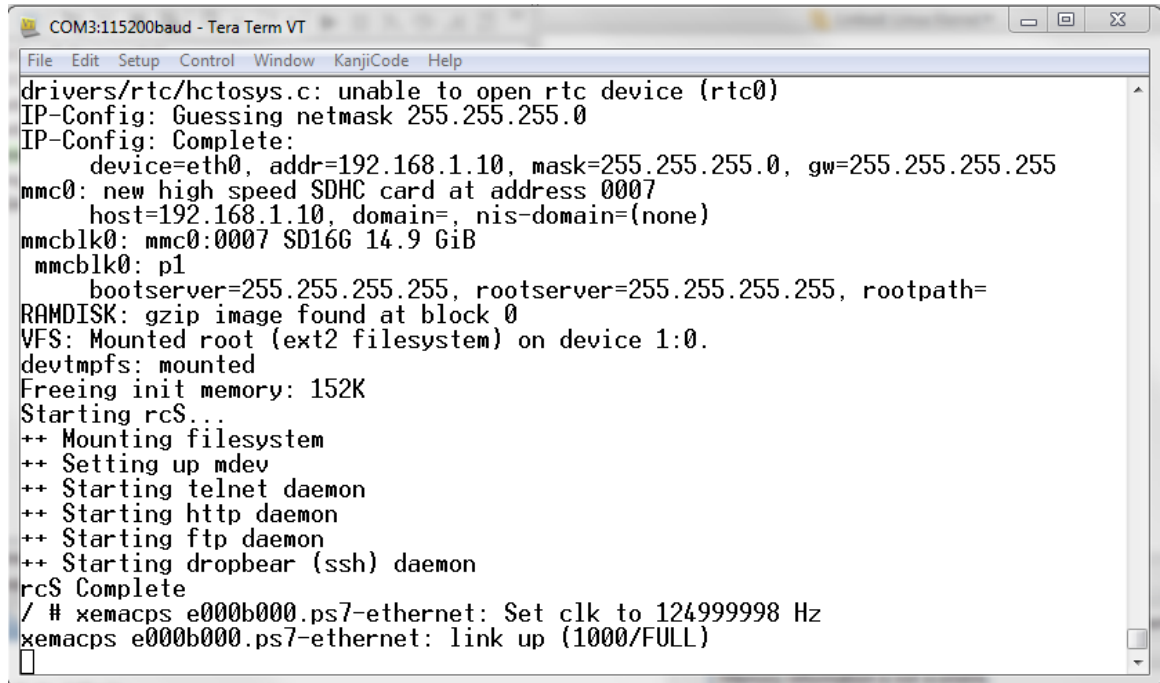
## Debugging the Linux Kernel

1. Open the DS-5 IDE on your host PC. For the purposes of this tutorial series, we will continue to use the same workspace created in an earlier lesson, called **ZedBoard DS-5 WS**.



**Select a DS-5 Workspace**

Power the DSTREAM and ZED target, and watch for the blue configuration LED to illuminate. As soon as this happens, start Tera Term and allow your system to boot to the Linux prompt.

A screenshot of a Tera Term window titled 'COM3:115200baud - Tera Term VT'. The window displays the output of a Linux boot process. The text shows various system messages including driver initialization, IP configuration, SD card detection, and the completion of the rcS (BusyBox) system. The boot process ends with network interface configuration for 'xemacps' and 'xemacps'.

```
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
    device=eth0, addr=192.168.1.10, mask=255.255.255.0, gw=255.255.255.255
mmc0: new high speed SDHC card at address 0007
    host=192.168.1.10, domain=, nis-domain=(none)
mmcblk0: mmc0:0007 SD16G 14.9 GiB
    mmcblk0: p1
    bootserver=255.255.255.255, rootserver=255.255.255.255, rootpath=
RAMDISK: gzip image found at block 0
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing init memory: 152K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
rcS Complete
/ # xemacps e000b000.ps7-ethernet: Set clk to 124999998 Hz
xemacps e000b000.ps7-ethernet: link up (1000/FULL)
```

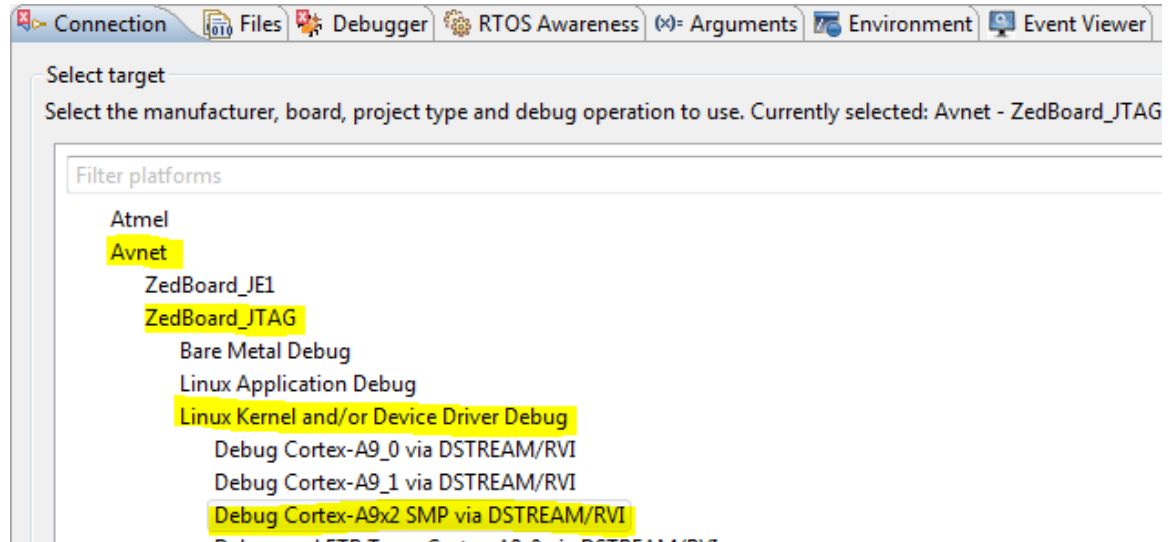
### Linux Boot on ZED target in Tera Term

2. We need to create a new debug configuration for kernel debugging. From the main menu, select **Run | Debug Configurations**. Select the **DS-5 Debugger** entry and create a new configuration.



3. Name the new configuration **Linux Kernel**.

In the **Connection** tab, select **Avnet | ZedBoard\_JTAG | Linux Kernel and/or Device Driver Debug | Debug Cortex-A9x2 SMP via DSTREAM/RVI**.



#### **DS-5 Connection for Linux Kernel Debug on ZED target**

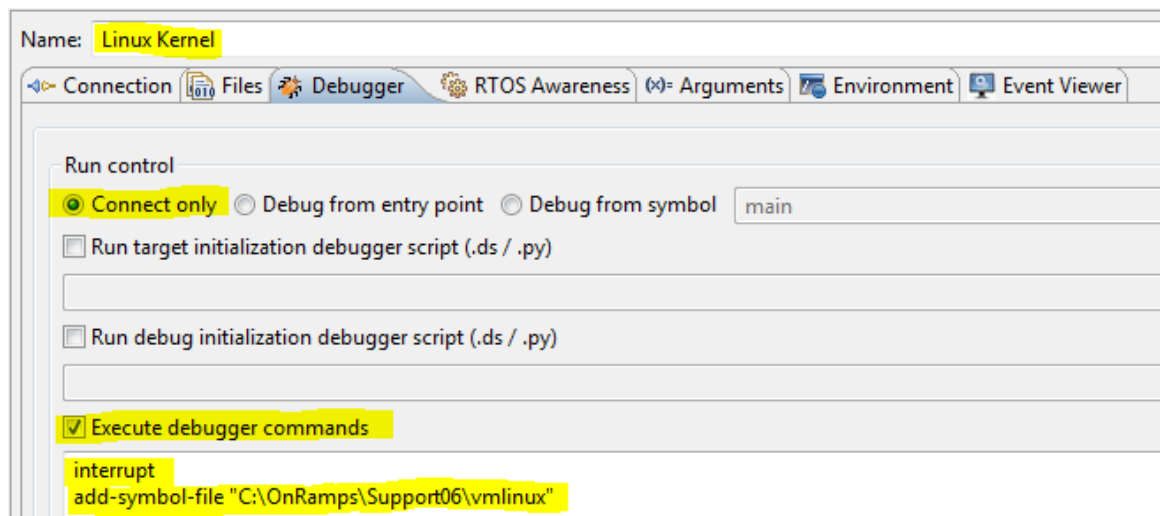
Click the **Browse** button and select the USB connection for your DSTREAM unit.

4. Select the **Debugger** tab. We already have Linux running on the target, so select **Connect only** in the **Run Control** area.

When the debugger connection is established, we want to interrupt the execution and load the symbol file that was created at kernel compilation time. This file allows the debugger to map between the source files used in the compilation, and the kernel image file executing on the board.

Click the checkbox next to **Execute debugger commands**. In the text box beneath, enter the commands:

```
interrupt
add-symbol-file "C:\OnRamps\Support06\vmlinux"1
```



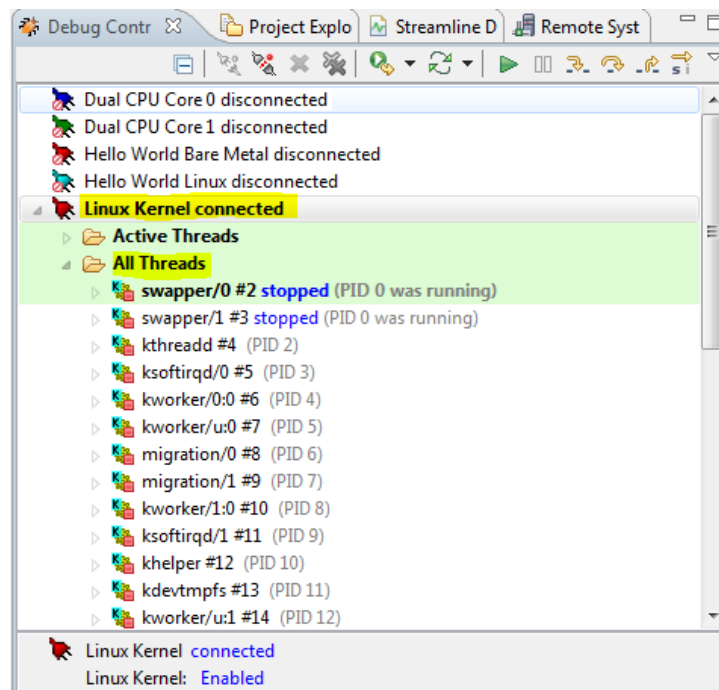
#### Kernel Debug Parameters

5. Click the **Apply** button to save the configuration, and click the **Debug** button to connect to the target. If you are asked to switch to the Debug Perspective, click the **Yes** button.

---

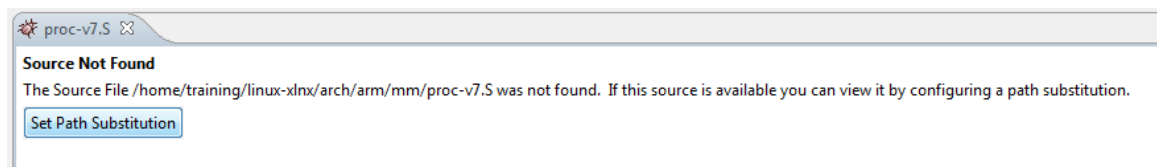
<sup>1</sup> For instructions on creating a vmlinux file for a custom Linux kernel, see Appendix I.

6. In the **Debug Control** tab, select the **Linux Kernel** entry and expand **All Threads**. You can see all the Linux threads and their associated process ids here. At the moment the kernel is stopped, so every thread is suspended.



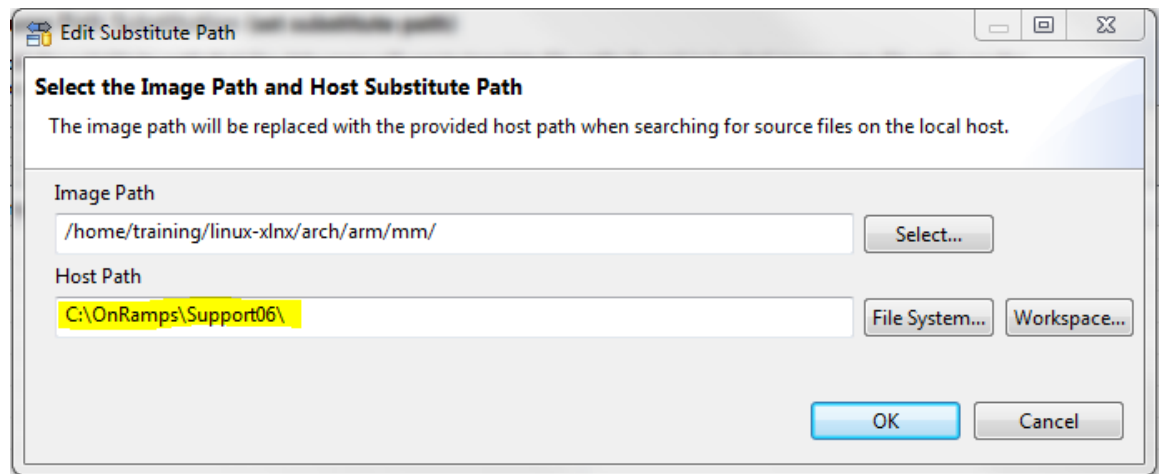
### Kernel Debug Session Stopped

In the source panel below, you will see that the symbol file has directed the debugger to the source file `proc-v7.S`, an assembler routine. The debugger cannot display this file because we have not yet told it where to locate the source. We can do that in the next step.



7. Click the **Set Path Substitution** button. The source files must be located on our local machine to display them in the debugger, but since they were compiled on a separate linux VM we have to copy them over and map the original path to our new location. For the purposes of this tutorial, we only copy a few source files, but in a real kernel debugging session you can copy over the entire kernel source tree.

Click the **File System** button and browse to the location where you decompressed the Support files for this tutorial. Click the **OK** button, and click the **OK** button on the Path Substitution panel as well.



### Map the Source File Location to the DS-5 Host

The source code for the proc-v7.S file is now displayed in the source window. You can step through the assembler as you would any C program using the controls under the Debug Control tab. If you step into an unmapped file, use this same process to add it to the mapping.

```
proc-v7.S
63
64 /*
65  * cpu_v7_do_idle()
66  *
67  * Idle the processor (eg, wait for interrupt).
68  *
69  * IRQs are already disabled.
70  */
71 ENTRY(cpu_v7_do_idle)
72     dsb                @ WFI may enter a low-power mode
73     wfi
74     mov pc, lr
75 ENDPROC(cpu_v7_do_idle)
76
77 ENTRY(cpu_v7_dcache_clean_area)
78
```

### proc-v7.S Source Displayed

8. In the Debug Control tab, start the kernel running again with the play control.



## Adding Simultaneous Application Debug

Now that you have the kernel running in the debugger, you can pause it, set breakpoints, and treat it like any other running program. But you can also set up a second debug connection using an Ethernet cable between your DS-5 host and the ZED target. This process was described in the tutorial #5, so if you completed that you already have a DS-5 configuration to debug a linux application.

1. Connect an Ethernet cable between your host PC and the ZedBoard.
2. In the **Debug Control** tab, right-click on the **Hello World Linux** configuration and select **Connect to Target**. If a password is requested, enter **root**.
3. At this point you will have a second connection for the Linux application, stopped at the entry point. You are now debugging an application and the kernel simultaneously. This is very useful when writing applications that interact with kernel-mode drivers, where a new application may expose a previously undetected bug in the driver code.

You can experiment switching between kernel debugging and application debugging, and when you are finished, disconnect the two debug sessions and power down the hardware.

## Appendix I – Creating the vmlinux Symbol File

The vmlinux symbol file was supplied with the Support Files included in the download, but you can create this file for your own custom Linux kernel using the following procedure.

1. Follow the normal steps you take to build the Linux kernel for your ZED target, but prior to the final make command a use menuconfig to update the build configuration. For complete details on how to build the Linux kernel, consult the Avnet Speedway for Implementing Linux on the Zynq 7000 SOC. Briefly, the procedure consists of the following steps:

- a. Clone the Linux source tree from the Xilinx repository.

**git clone git://git.xilinx.com/linux-xlnx.git <local dir>**

- b. Clean the build directories.

**make distclean**

- c. Configure the default kernel build parameters for the ZED target.

**make ARCH=arm xilinx\_zynq\_defconfig**

- d. Customize the build parameters. This is the extra step you must add to change the configuration to produce the vmlinux symbol file. If you are using the supplied ramdisk image, you will also need to update the configuration to use a larger ramdisk.

**make ARCH=arm menuconfig**

If you are familiar with the operation of menuconfig, or the .config file it creates, you can refer to this summary for all the required changes and skip to step e. If you need step by step details, continue on the following page.

- Update configuration for large ramdisk
  - **Device Drivers →Block Devices**
    - **Default number of RAM disks (change from 16 to 8)**
    - **Default RAM disk size (change from 16384 to 32768)**
- Generate Kernel Debug Information and symbol file
  - **Kernel hacking**
    - **[\*] Compile the kernel with debug info**
    - **[\*] Tracers**
      - **[\*] Trace process context switches and events**

## i. Select Device Drivers

```
Linux/arm 3.6.0 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

^(-)
  Bus support --->
  Kernel Features --->
  Boot options --->
  CPU Power Management --->
  Floating point emulation --->
  Userspace binary formats --->
  Power management options --->
  [*] Networking support --->
  [*] Device Drivers --->
  File systems --->
v(+)

<Select>  < Exit >  < Help >
```

## ii. Select Block Devices

```
Device Drivers
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

^(-)
<*> Memory Technology Device (MTD) support --->
  Device Tree and Open Firmware support --->
  <> Parallel port support --->
  [*] Block devices --->
    Misc devices --->
    <> ATA/ATAPI/MFM/RLL support (DEPRECATED) --->
    SCSI device support --->
    <> Serial ATA and Parallel ATA drivers --->
    [ ] Multiple devices driver support (RAID and LVM) --->
    <> Generic Target Core Mod (TCM) and ConfigFS Infrastructure --
v(+)

<Select>  < Exit >  < Help >
```

## iii. Select Default number of RAM disks

```
Block devices
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

^(-)
<> Cryptoloop Support
<> DRBD Distributed Replicated Block Device support
<> Network block device support
<> NVm Express block device
<> Promise SATA SX8 support
<> Low Performance USB Block driver (deprecated)
<*> RAM block device support
[16] Default number of RAM disks
(16384) Default RAM disk size (kbytes)
[ ] Support XIP filesystems on RAM block device
v(+)

<Select>  < Exit >  < Help >
```

- iv. Enter **8** for the number of RAM disks and click **OK**

Default number of RAM disks

Please enter a decimal value. Fractions will not be accepted. Use the <TAB> key to move from the input field to the buttons below it.

8

< Ok > < Help >

- v. Select **Default RAM disk size (Kbytes)**

Block devices

Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [\*] built-in [ ] excluded <M> module < >

^(-)

- < > Cryptoloop Support
- < > DRBD Distributed Replicated Block Device support
- < > Network block device support
- < > NVM Express block device
- < > Promise SATA SX8 support
- < > Low Performance USB Block driver (deprecated)
- <\*> RAM block device support
- (8) Default number of RAM disks
- (15384) **Default RAM disk size (kbytes)**
- [ ] Support XIP filesystems on RAM block device

v(+)

<Select> < Exit > < Help >

- vi. Enter **32768** for the RAM disk size and click **OK**

Default RAM disk size (kbytes)

Please enter a decimal value. Fractions will not be accepted. Use the <TAB> key to move from the input field to the buttons below it.

32768

< Ok > < Help >



- vii. Click **Exit** twice to return to the main menu. Select **Kernel Hacking**

```
Linux/arm 3.6.0 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

^(-)
  Boot options --->
  CPU Power Management --->
  Floating point emulation --->
  Userspace binary formats --->
  Power management options --->
  [*] Networking support --->
  Device Drivers --->
  File systems --->
  Kernel hacking --->
  Security options --->
v(+)

<Select>  < Exit >  < Help >
```

- viii. Select **Compile the kernel with debug info**

```
Kernel hacking
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

^(-)
[ ] Sleep inside atomic section checking
[ ] Locking API boot-time self-tests
[ ] Stack utilization instrumentation
[ ] kobject debugging
[ ] Highmem debugging
[ ] Verbose BUG() reporting (adds 70K)
[*] Compile the kernel with debug info
[ ] Reduce debugging information (NEW)
[ ] Debug VM
[ ] Debug filesystem writers count
v(+)

<Select>  < Exit >  < Help >
```

(enables CONFIG\_DEBUG\_INFO)

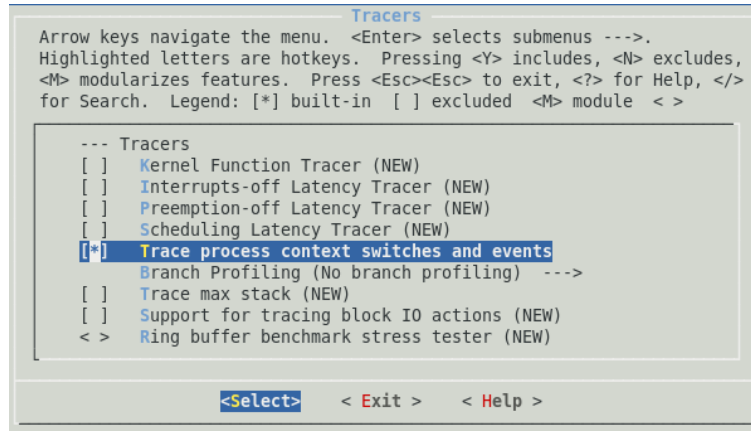
- ix. Select **Tracers**

```
Kernel hacking
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

^(-)
[ ] Force extended block device numbers and spread them
[ ] Force weak per-cpu definitions
[ ] Debug access to per_cpu maps
< > Linux Kernel Dump Test Tool Module
< > Notifier error injection
[ ] Fault-injection framework
[ ] Debug page memory allocations
[*] Tracers --->
[*] Enable dynamic printk() support
[ ] Enable debugging of DMA-API usage
v(+)

<Select>  < Exit >  < Help >
```

x. Select **Trace process context switches and events**



(enables CONFIG\_ENABLE\_DEFAULT\_TRACERS)

e. Exit and save the configuration.

f. Build the Linux kernel for the ZED target.

**e.g.: make ARCH=arm UIMAGE\_LOADADDR=0x8000 uImage**

g. The **uImage** is created in the **arch/arm/boot** directory. The **vmlinux** symbol file is created in the top-level directory where make was run.

## Revision History

Date	Version	Revision
29 May 13	00	Initial Draft
20 Sep 13	01	Release

## Resources

<http://www.zedboard.org>

<http://www.xilinx.com/zyng>

<http://www.arm.com/products/tools/software-tools/ds-5/index.php>