Altera SoC Embedded Design Suite

Version 2.0

March 18, 2014

Corporate HQ & Design Center
380 Stevens Ave. Suite 206
Solana Beach, CA 92075
http://www.macnica-na.com

About Macnica Americas

Macnica Americas is a franchised semiconductor distributor for multiple, high-tech suppliers within North America. Our business model emphasizes unsurpassed technical support and knowledge versus other distribution options at no cost premium. Macnica Americas is the North American based division of Macnica Inc., a $2.4B global leader in semiconductor distribution. We maintain a field support staff as well as centralized design & applications teams.

Optional design services are headquartered in San Diego, CA., USA and offer partial or full turnkey design of FPGAs, power distribution networks, and full PCB design. Our expertise includes all aspects of high speed communications protocols and networking, video broadcast, signal processing, and storage applications. Macnica's specialty is high density, high speed complex FPGA designs utilizing multiple IP cores with fast time to market requirements.

Macnica can help you deliver a winning project with the unique combination of technical support, custom IP, and design services. Setup a meeting today!

http://www.macnica-na.com

License and Terms of Use

Table of Contents

1 Lab Overview

1.1 Introduction and Goals

The Altera® system on a chip (SoC) Embedded Design Suite (EDS) is a comprehensive tool suite for embedded software development on Altera SoC devices. The Altera SoC EDS contains development tools, utility programs, run-time software, and application examples that enable firmware and application software development on the Altera SoC hardware platform.

The Altera SoC EDS provides the tools you need to work more productively, improve your software quality, and quickly bring your product to market.

This lab is designed as a self-paced-learning tool for understanding the fundamentals of using the tools and reference designs included in the Altera SoC EDS installation. It is highly recommended persons attend additional training, such as that offered by Altera directly, for more detailed education on this rather complex flow and device family.

The lab is broken into a series of major sections or milestones representing the common uses of the EDS, ARM DS-5 and reference designs. Unlike other trainings you may have had, this lab does not explicitly indicate every button to push or value to enter. Instead, your goal is described with the necessary information given. If you are having problems, each section concludes with a series of hints related to the tasks proposed.

1.2 Hardware and Software Requirements

- Macnica Helio SoC Evaluation board with 2 micro-USB cables and 1 Ethernet cable
- This lab uses a Windows host computer, as can be seen from the screenshots and the issued commands. However, the scenario can also be run on a Linux machine which is the more typical development environment for embedded Linux.
- Quartus II v13.1 (recommended, web or subscription edition) or stand-alone device programmer and SignalTap II Logic Analyzer
- Altera SoC EDS v13.1 installed
- The ARM DS-5 AE will be installed with the SoC EDS and a license will be required.
- microSD card loaded with Helio Linux kernel v3.9: helio_sdimage_v3.9.tar.gz
- Downloaded Linux kernel v3.9 source for debug: socfpga-3.9-rel.tar.gz

1.3 Assistance

A dedicated e-mail account has been setup to receive support requests for the vWorkshop series. Please identify the course (in this case SoC EDS) in addition to details on the question.

workshophelp@macnica.com

1.4 Lab Agenda and Milestones

Set-up Host & Target and Launch DS-5

A few items will need to be set-up on both the host PC as well as the target SoC device running Linux. This includes extracting the Linux kernel source for debugging, setting the IP address of the target if a DHCP server is not available and setting up a root password.

Connect to Remote Target

The ARM DS-5 AE can run and debug programs directly on the target with the help of the Remote System Explorer (RSE). Before this feature can be used, the RSE needs to be configured to connect to the target board running Linux.

Import Linux Application

There are several example software applications included with the SoC EDS install. This lab will be using the simple "Hello World" Linux application.

Debug Linux Application

Once the "Hello World" application is compiled and an executable is produced, you will use the remote system connection to download and debug the application.

Debug Running Linux Kernel

It is required for this lab that the microSD card has a working Linux kernel based on the Helio golden reference design (GHRD). The ARM DS-5 offers powerful tools for Linux kernel and driver developers that are accessed via the USB Blaster-II debug interface. A licensed DS-5 in required.

View Peripheral Registers

The ARM DS-5 Altera Edition allows you to specify the peripheral IP register descriptions using SVD files. The SVD files are a result of the hardware project compilation using Quartus-II. The SVD files contain the description of both HPS peripheral registers (UART, EMAC, timers etc.) and soft IP peripheral registers residing in the FPGA.

Trace Linux Kernel execution

ARM DS-5 provides powerful tracing features, allowing PTM and STM tracing and also allows for different tracing buffer memories.

Enable Cross-Triggering

The Altera SoC FPGA offers powerful cross-triggering capability between the HPS and the FPGA fabric. The HPS can trigger the FPGA and the FPGA can also trigger the HPS. ARM has updated the DS-5 tool specifically for Altera to enable this SoC FPGA capability to be easily used.

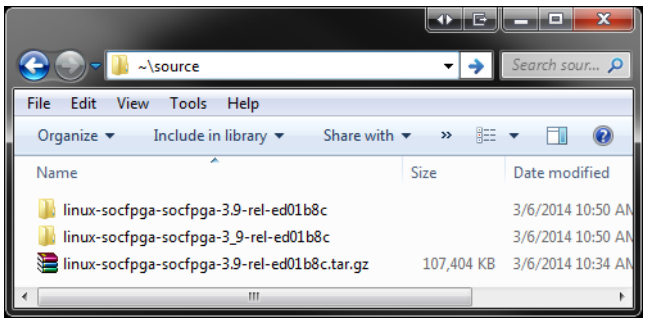
2 Lab Instructions

2.1 Set-up Host & Target and Launch DS-5

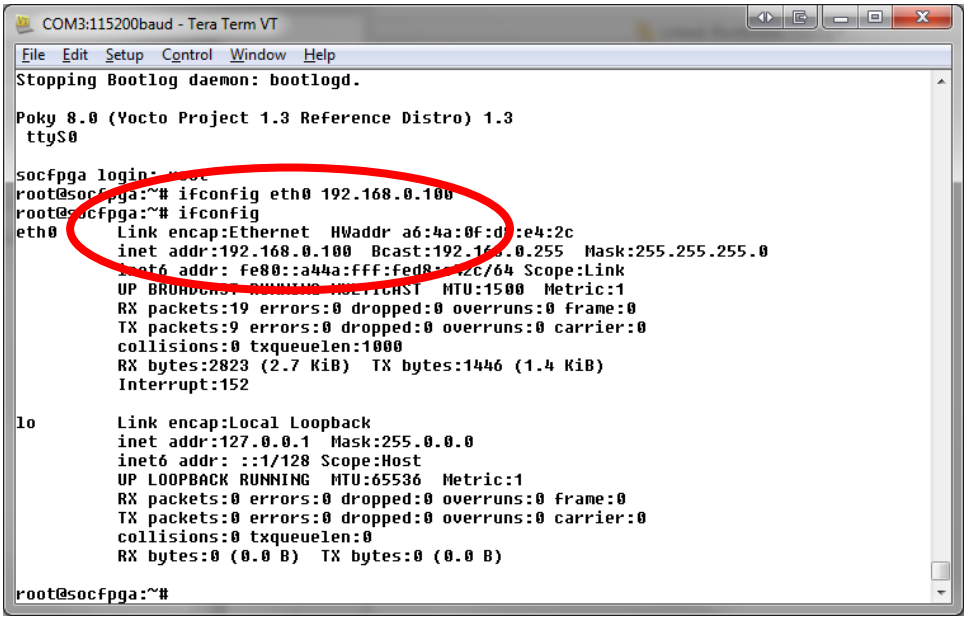
In the scenario presented here the Linux kernel needs to be running on the board, but it can also be downloaded through the debugger. (We will not do this in this lab.) This scenario uses the pre-built Helio GHRD Linux image available on RocketBoards and Linux code available from the GIT source. It is assumed there is a serial connection to the target board and it is verified that the Linux boot process is complete.

- ☐ The Linux kernel executable file needs to be accessible on the host computer. Verify the he kernel executable for the pre-built Linux image is located at **~/altera/13.1/embedded/embeddedsw/socfpga/prebuilt_images/vmlinux**
- ☐ The source code corresponding to the kernel running on the board needs to be accessible on the host computer. The sources for the Linux image can be obtained by extracting the file downloaded from http://git.rocketboards.org/?p=linux-socfpga.git;a=snapshot;h=socfpga-3.9-rel;sf=tgz
- ☐ In order for the ARM DS-5 to communicate with the target over Ethernet, the target's IP address must be known. On the target use the **ifconfig** utility to either set or verify the target's IP address.
- ☐ The DS-5 Remote System Explorer uses secure communications with the target via SSH. The default Linux kernel does not have a password set for the root user. Use the **passwd** utility on the host to set a root password of your choice.
- ☐ Included in the SoC EDS install is an Embedded Command Shell that sets up the environment so that the DS-5 can access utilities directly without hard-coded paths. Launch the **Embedded_Command_shell.bat** batch file (or **embedded_command_shell.sh** in Linux) from **~/altera/13.1/embedded**
- ☐ Launch the ARM DS-5 Altera Edition from within the shell, **eclipse &**, and select a workspace of your choosing.

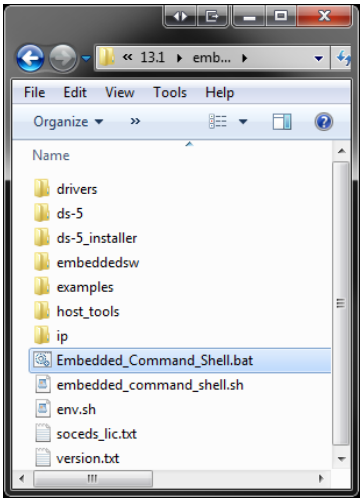
- *Hints:*
  - *Extracted Linux source*

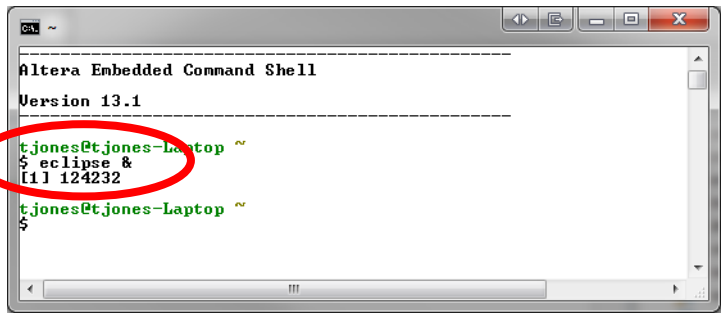


- *IP address set on target Linux kernel*

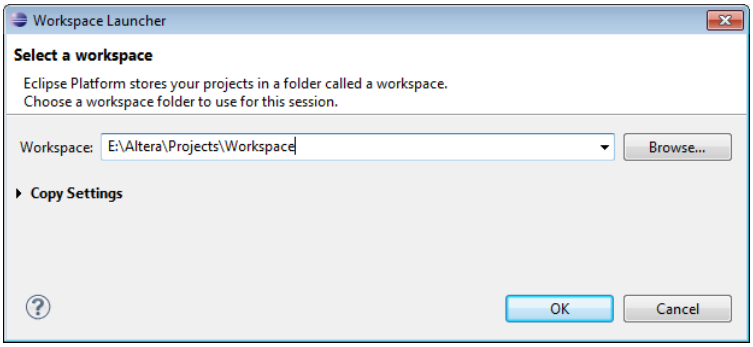


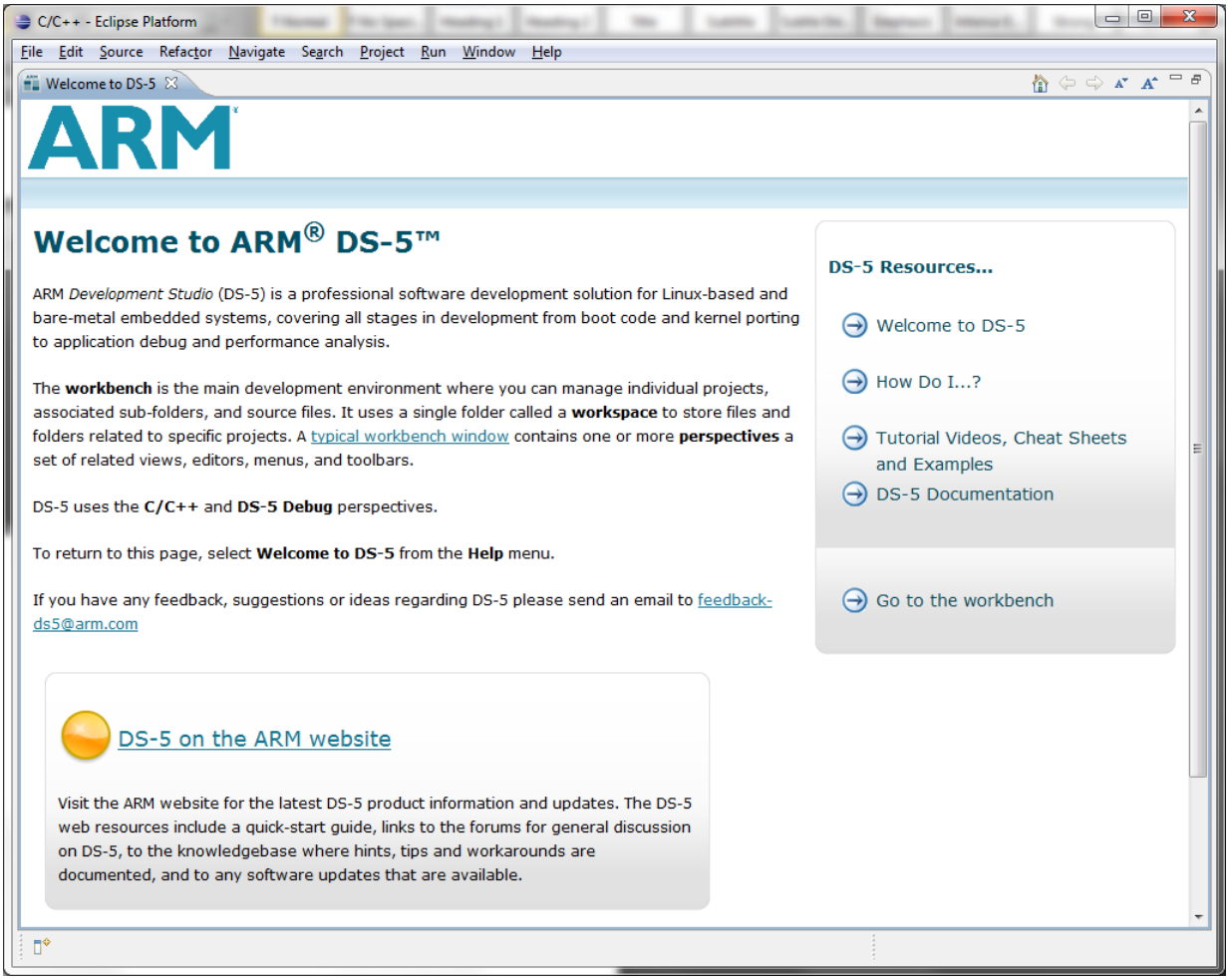
- *Launch Embedded Command Shell and ARM DS-5*





- *Select a DS-5 Workspace*



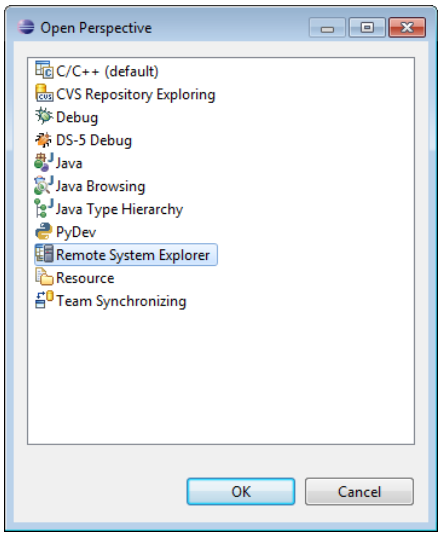


2.2 Connect to Remote Target

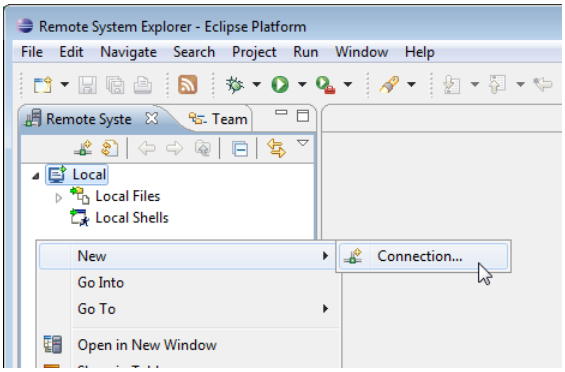
- ☐ Each Eclipse Workspace is comprised of many window Views and Perspectives. Open the **Remote System Explorer** perspective and create a new **Connection**.
- ☐ The RSE connection will be to the running **Linux** system at the **Host Name** (i.e. IP address from previous step). Give the connection a name of your choosing.

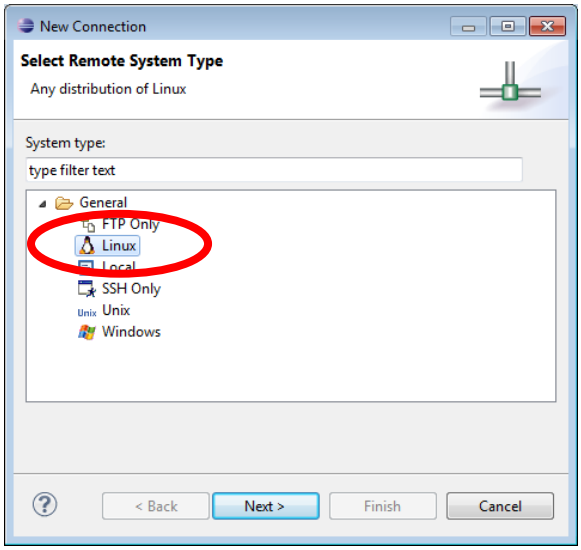
- ☐ As mentioned earlier, the DS-5 will connect to the target using SSH protocols and therefore you will need to modify the default Linux target type of file transfer from **ftp.files** to **ssh.files**.
- ☐ **Connect** to the remote system and enter the root username and password you define in the previous step. The RSA security warning can be safely ignored.
- ☐ Once a connection has been established, expand the **Root** file system tree view to explore the root file system that is on the active target.
- ☐ The DS-5 also has the capability to initiate an **Ssh Terminal** session to the remote target. Once the terminal session is open, feel free to explore the running Linux kernel.

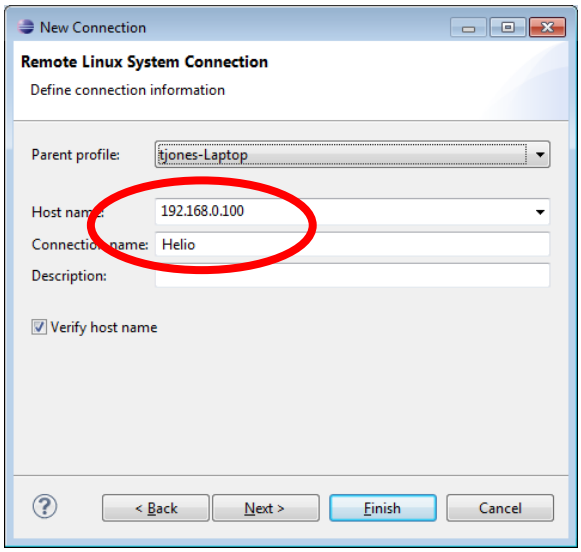
- *Hints:*
  - *Open Remote System Explorer Perspective*

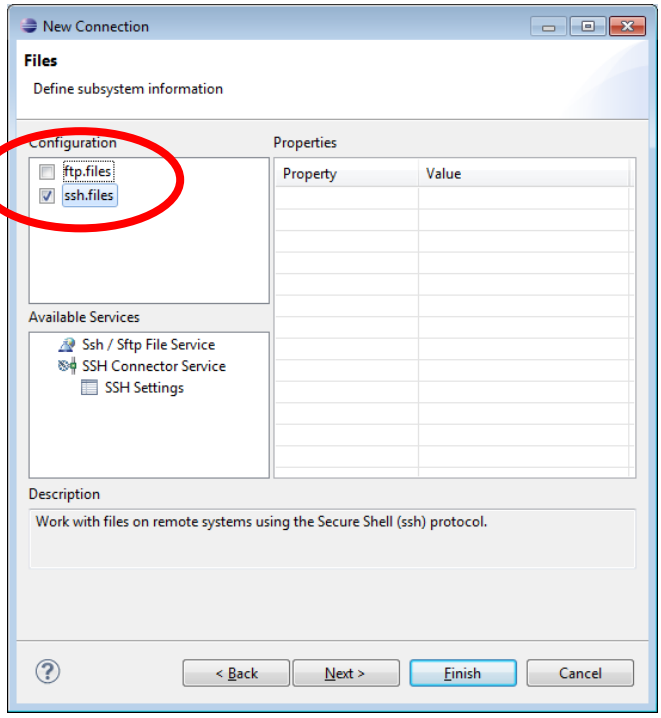


  - *Establish Ethernet connection to remote target using IP address*

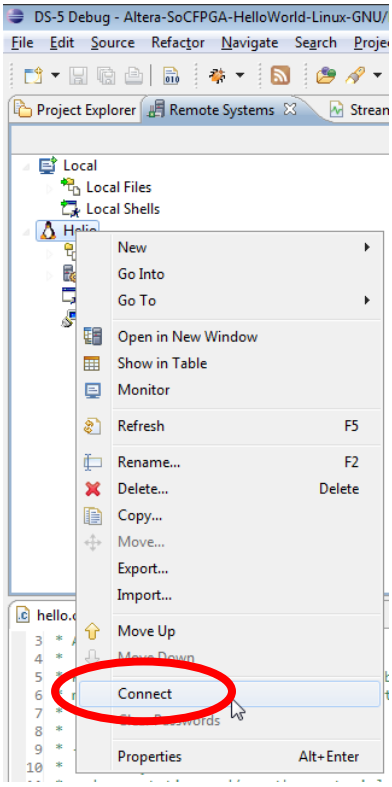


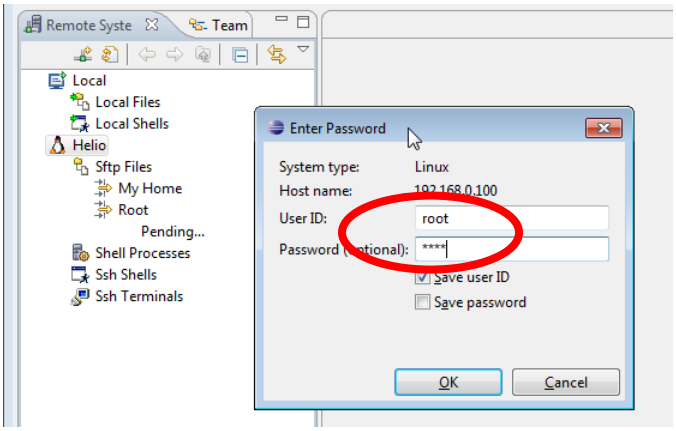

New Connection
Select Remote System Type
Any distribution of Linux
System type:
type filter text
General
FTP Only
Linux
Local
SSH Only
Unix
Windows
< Back
Next >
Finish
Cancel

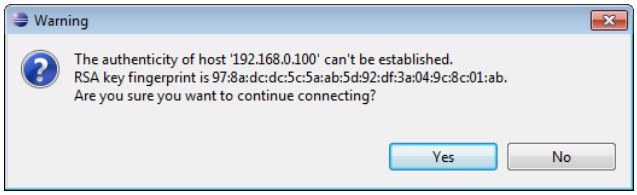

New Connection
Remote Linux System Connection
Define connection information
Parent profile:
tjones-Laptop
192.168.0.100
Helio
Description:
Verify host name
< Back
Next >
Finish
Cancel

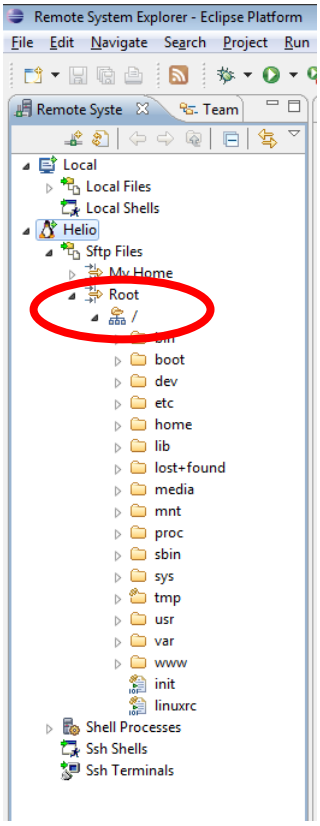


- *Connect to remote target and explore root file system.*

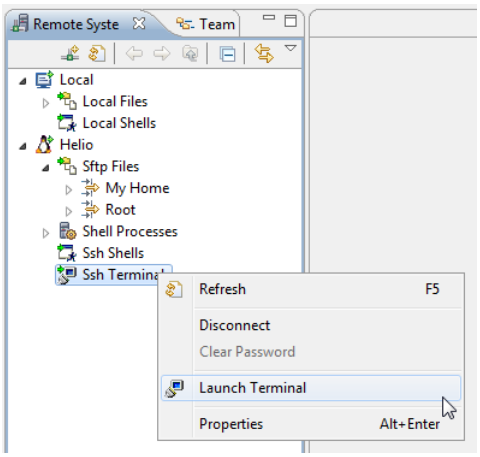


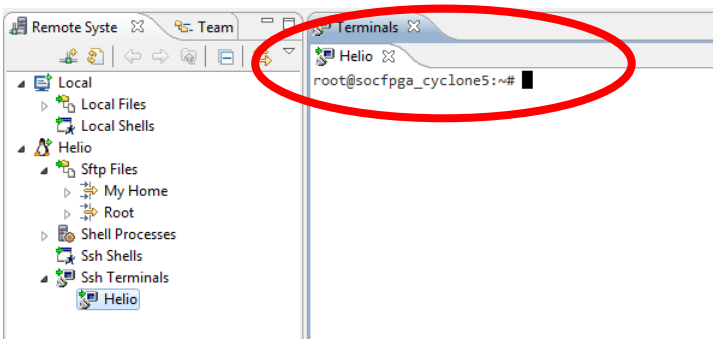

Remote Syste
Team
Local
Local Files
Local Shells
Helio
Sftp Files
My Home
Root
Pending...
Shell Processes
Ssh Shells
Ssh Terminals
Enter Password
System type:
Linux
Host name:
192.168.0.100
User ID:
root
Password (optional):
****
Save user ID
Save password
OK
Cancel


Warning
The authenticity of host '192.168.0.100' can't be established.
RSA key fingerprint is 97:8a:dc:dc:5c:5a:ab:5d:92:df:3a:04:9c:8c:01:ab.
Are you sure you want to continue connecting?
Yes
No


Remote System Explorer - Eclipse Platform
File
Edit
Navigate
Search
Project
Run
Remote Syste
Team
Local
Local Files
Local Shells
Helio
Sftp Files
My Home
Root
/
boot
dev
etc
home
lib
lost+found
media
mnt
proc
sbin
sys
tmp
usr
var
www
init
linuxrc
Shell Processes
Ssh Shells
Ssh Terminals

- *Launch Terminal session from within DS-5*



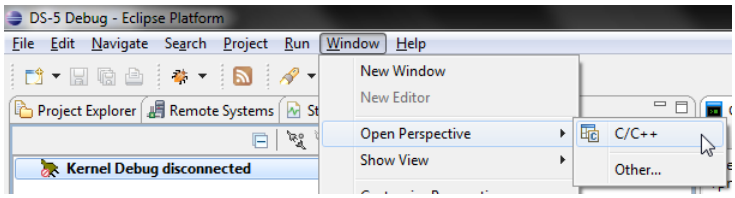


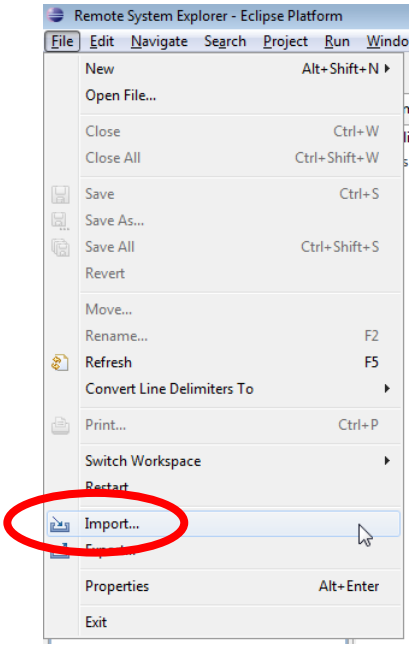
2.3 Import Linux Application

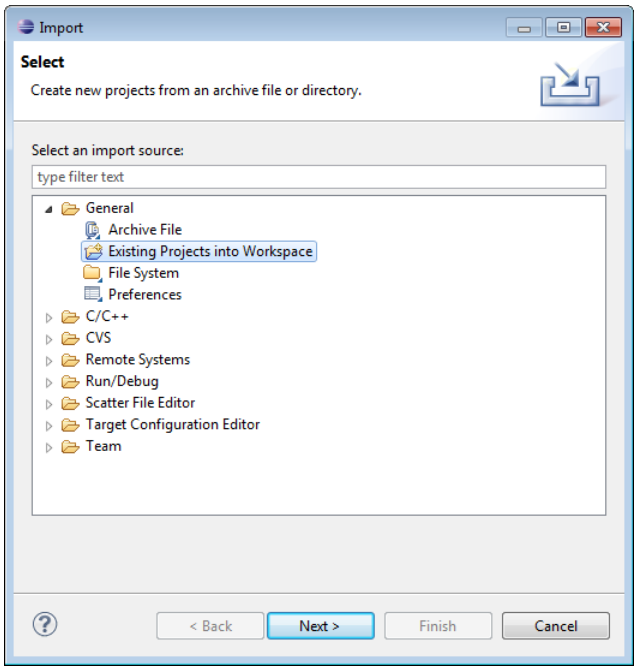
When the SoC EDS was installed, a group of example DS-5 software projects were also included. The DS-5 Eclipse environment has the capability of directly importing existing projects that have been archived.

- ☐ Import the **~\altera\13.1\embedded\examples\software\Altera-SoCFPGA-HelloWorld-Linux-GNU.tar.gz** into your workspace.
- ☐ The imported project is complete and only needs to be built. **Build Project** and make note of the newly generated executable application.

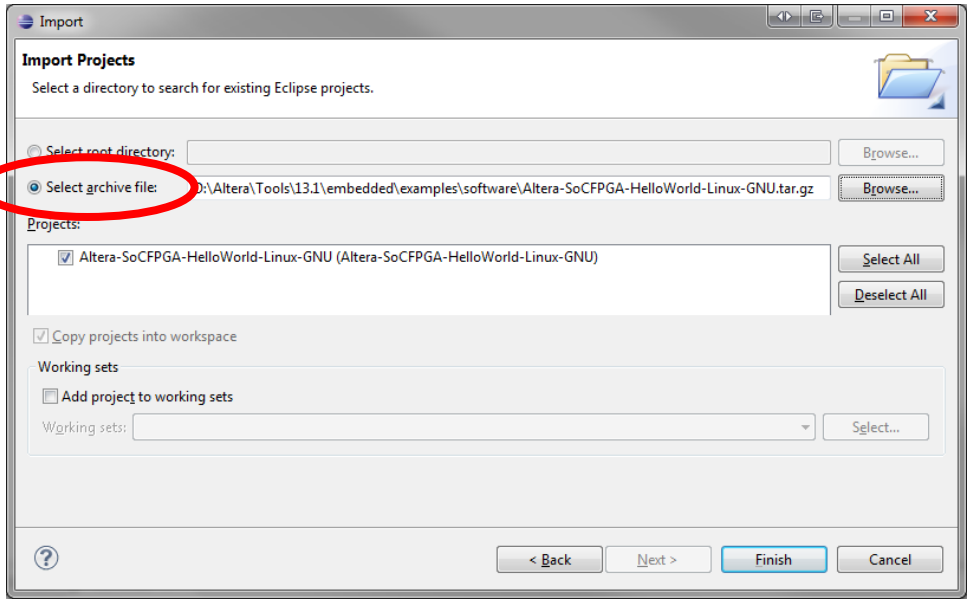
- *Hints:*
  - *Import existing project into Workspace*

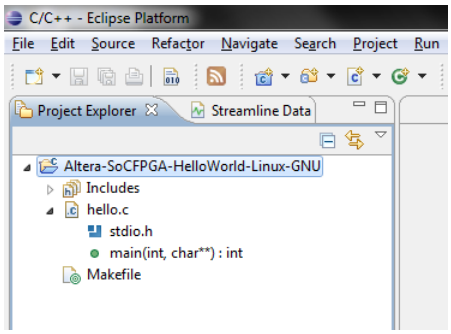


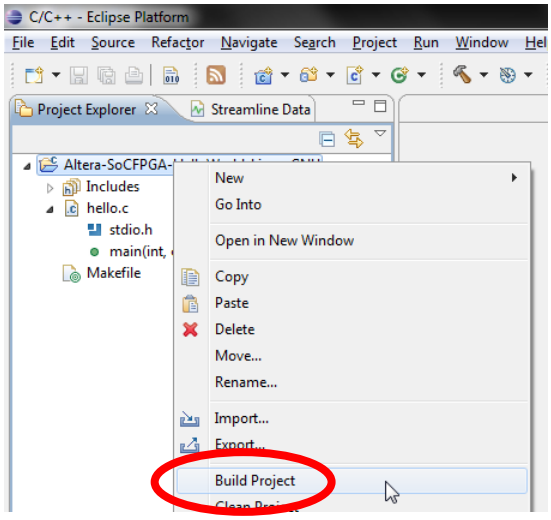

Remote System Explorer - Eclipse Platform
File Edit Navigate Search Project Run Windo
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Properties Alt+Enter
Exit

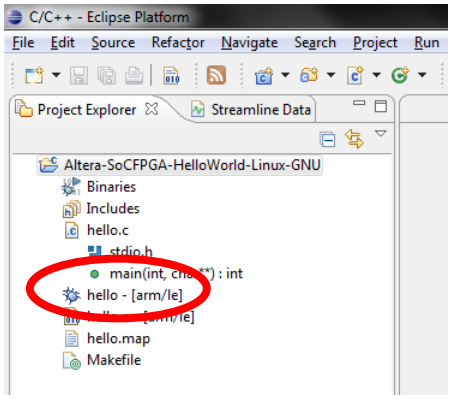

Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
C/C++
CVS
Remote Systems
Run/Debug
Scatter File Editor
Target Configuration Editor
Team
< Back
Next >
Finish
Cancel

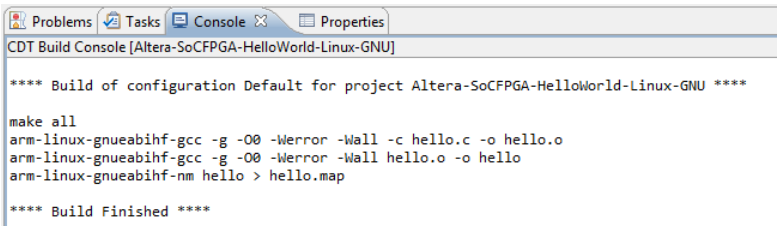
- *Be sure to select the **Select archive file** option*









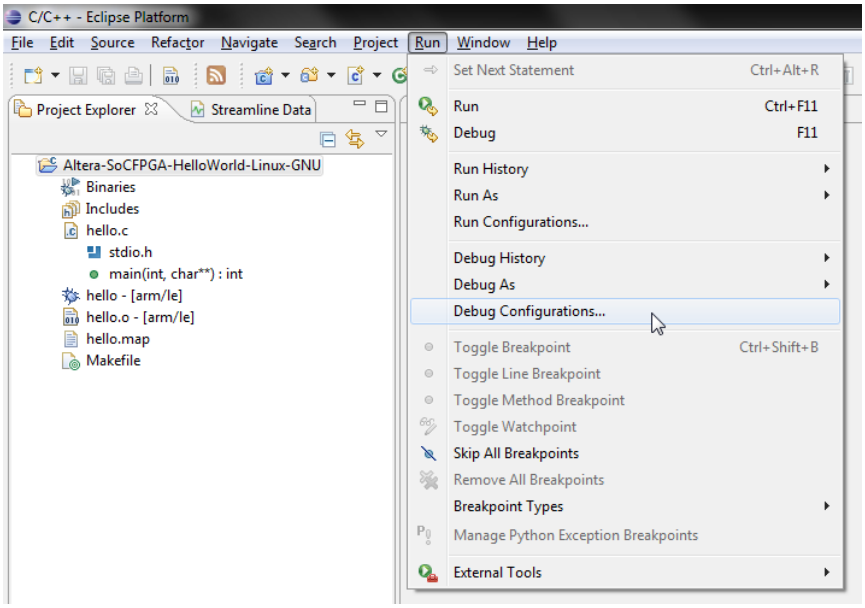


2.4 Debug Linux Application

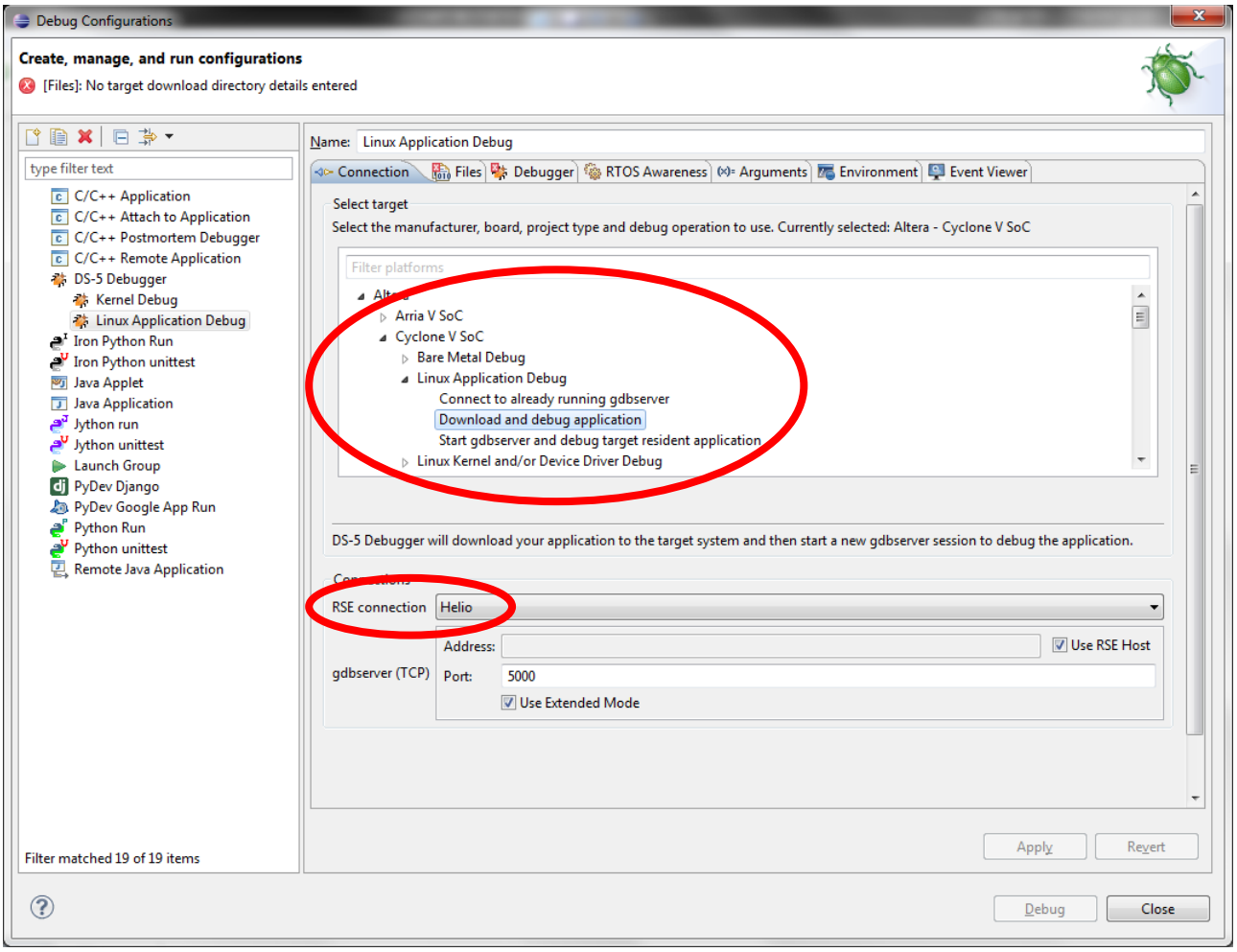
Now that the application is built providing an executable, a configuration needs to be created that will download the application over Ethernet, launch the debug session and turn execution control over to the user.

- ☐ Create a **Debug Configuration** and give the debug configuration a **Name** of your choosing.
- ☐ In the **Connection** options window, select the target SoC and configure the DS-5 to **Download and debug application** and ensure that the **RSE connection** selects the remote system as set up in the first section.
- ☐ The debug configuration will also need to know the **Files** necessary to download. This will be the executable file that was built in the previous step that can be found in your **Workspace**.
- ☐ In order for the download to happen, DS-5 needs to be told where the **Target download directory** and **Target working directory** are. Set these to the target's root home directory.
- ☐ Establish the **Debug** connection to the target to initiate the application download and halt the processor at main(). You now have control to step the processor, look at registers, variables, etc. The STDIO will be presented in the **App Console** window.
- ☐ When finished debugging the application, be sure to **Disconnect from target** so that additional debug sessions can be initiated.

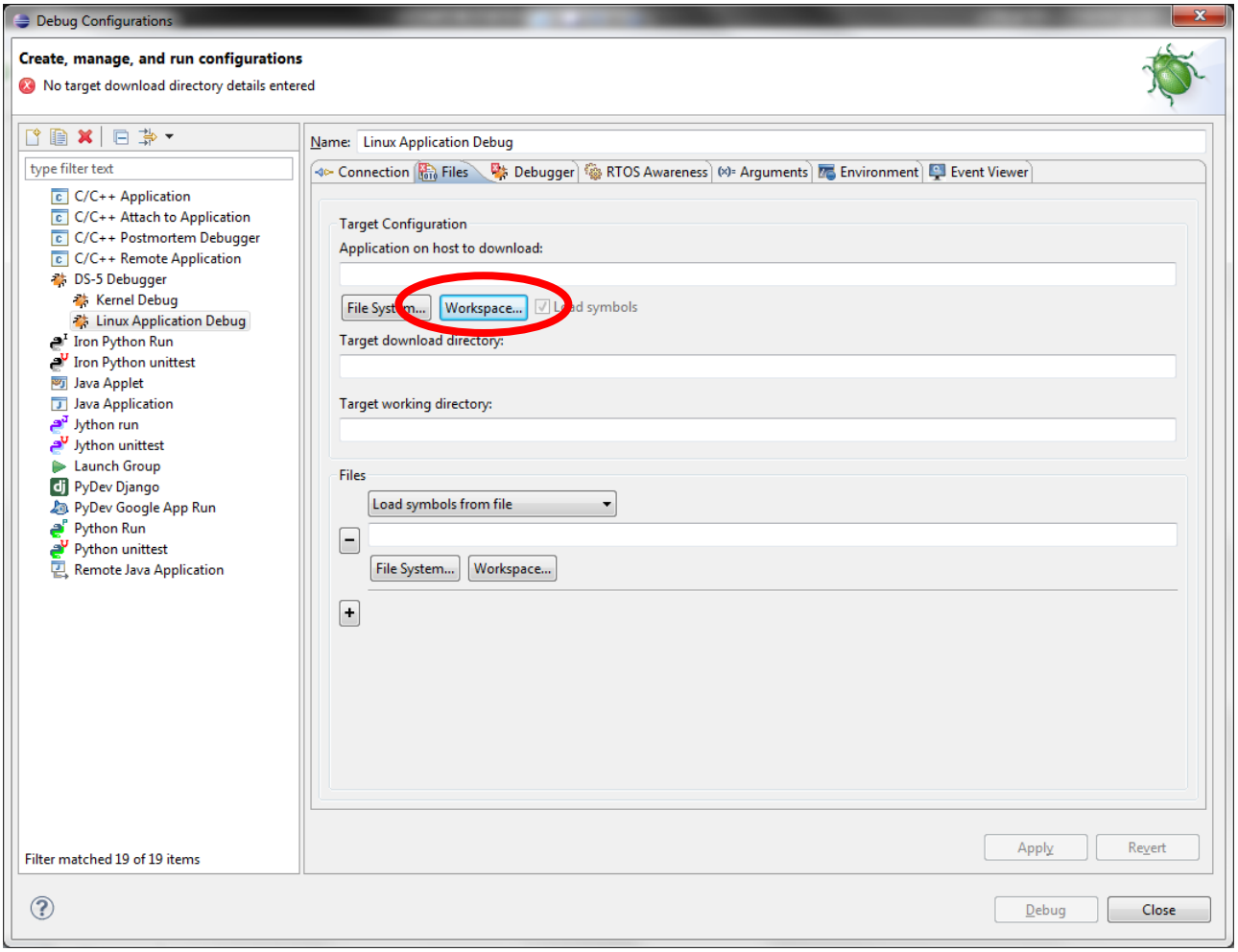
- *Hints:*
  - *Create a new Debug Configuration*

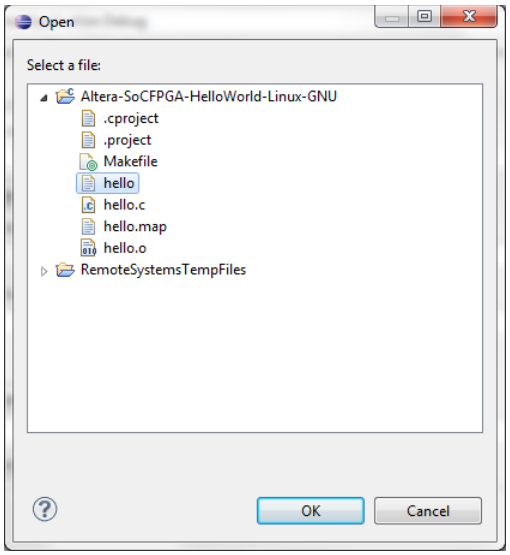


- *Select correct SoC device and target connection*

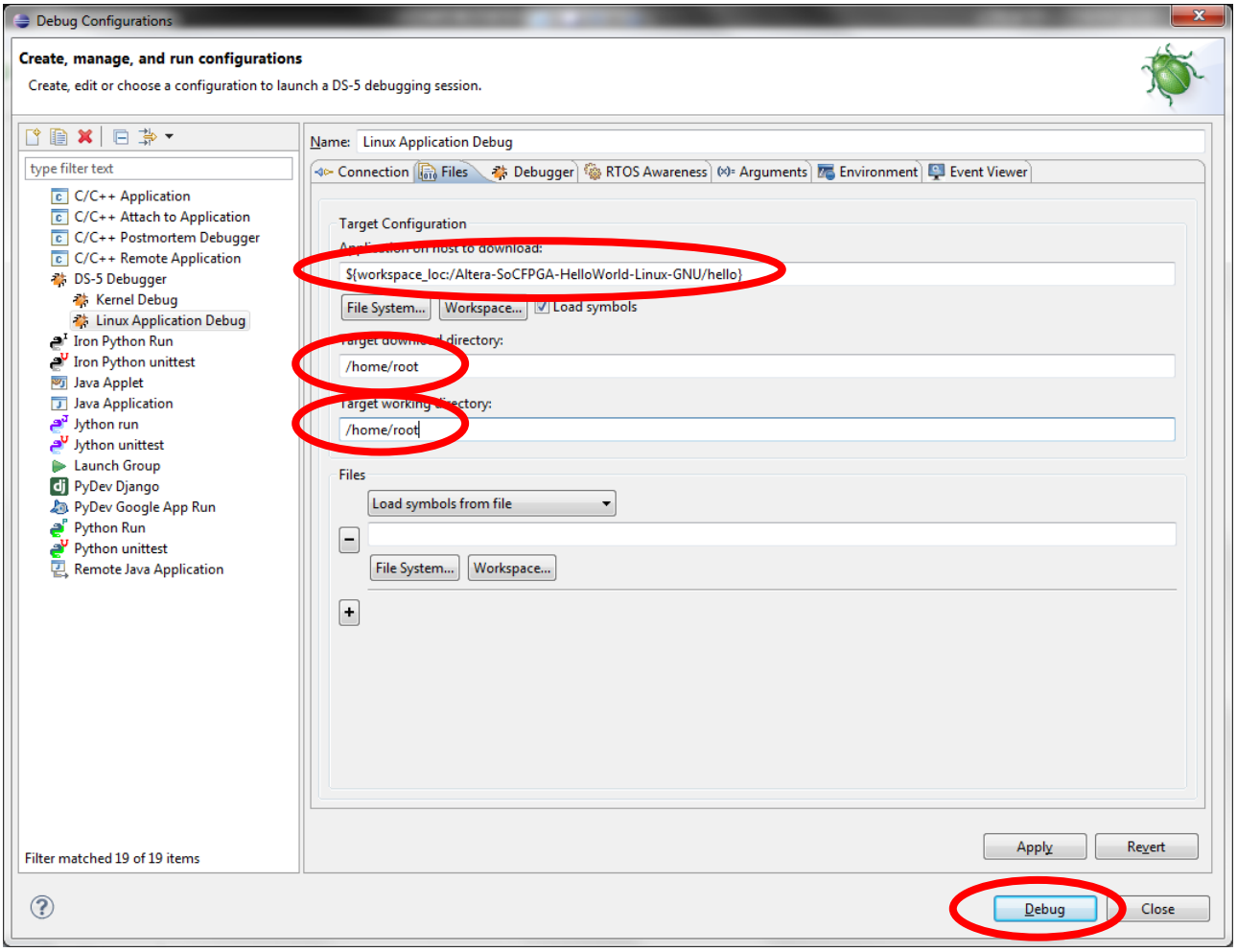


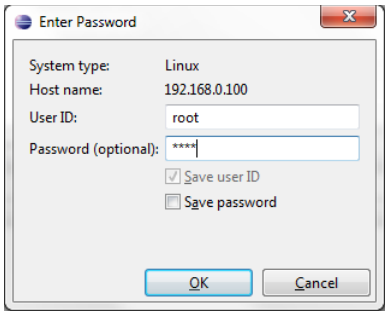
- *Specify the application to download and target directories*



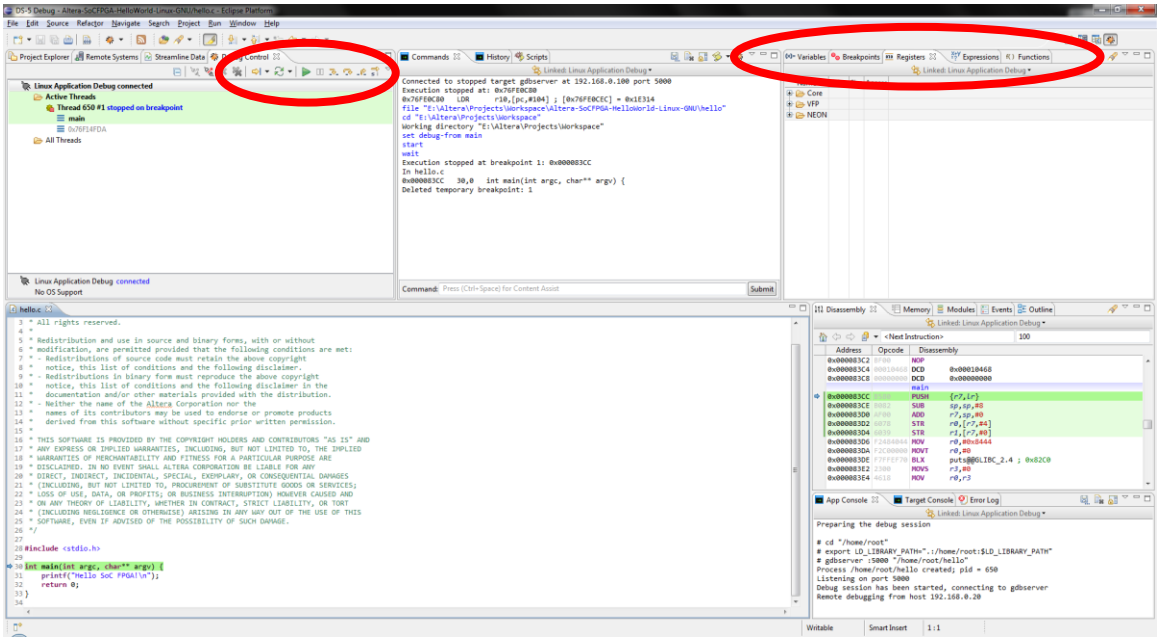


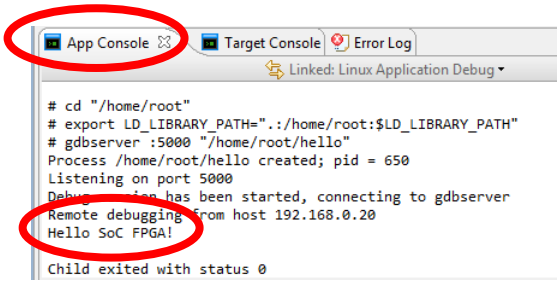
- *Start debug session to download to target and halt processor*



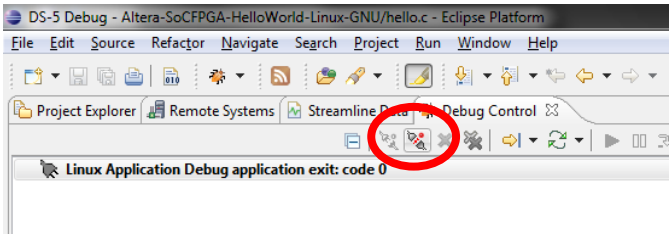


- *Control execution through step and next buttons. Explore the system environment.*





- *Be sure to Disconnect from target when done*



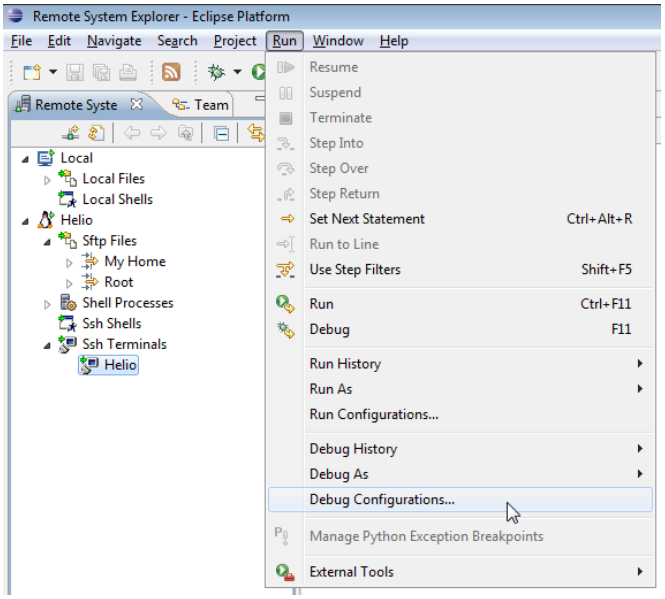
2.5 Debug Running Linux Kernel

The pre-built Linux kernel that is running on the target has the ability to be debugged. DS-5 for Altera has been designed such that a debug connection can be established using the USB Blaster-II that is built into the Helio evaluation board.

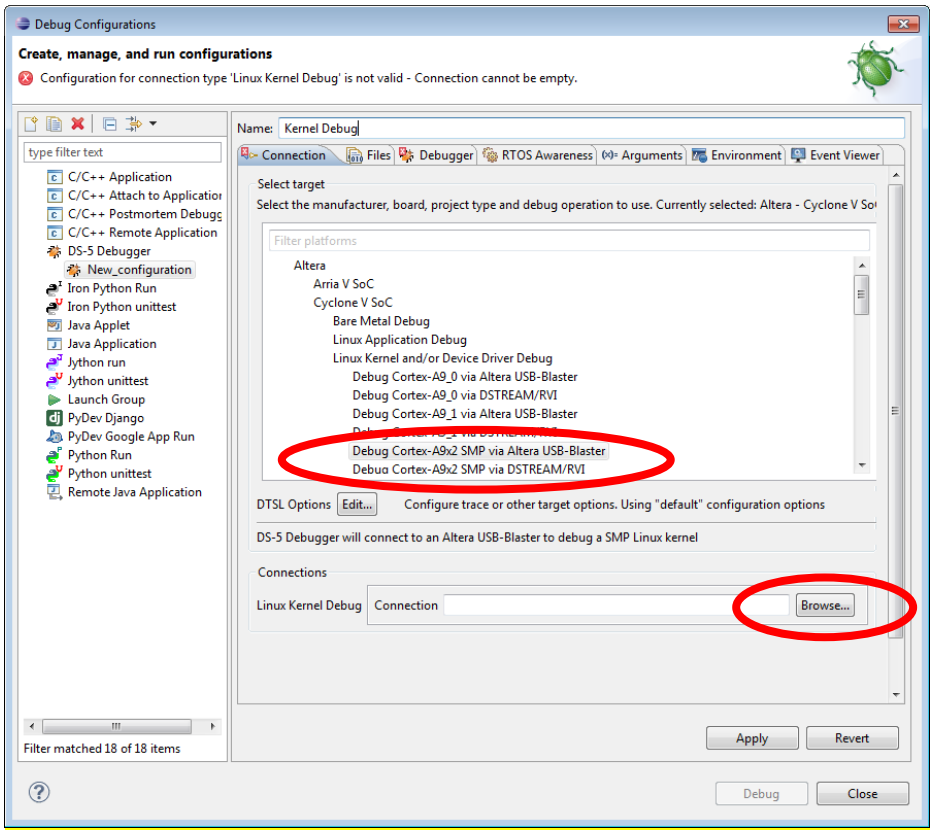
☐ Create a new **Debug Configuration** and select a target connection to debug the **Linux Kernel and/or Device Driver** using the **USB-Blaster** option. It may be necessary to **Browse** for the connection to find the **Helio on localhost [USB-1]** debug hardware.

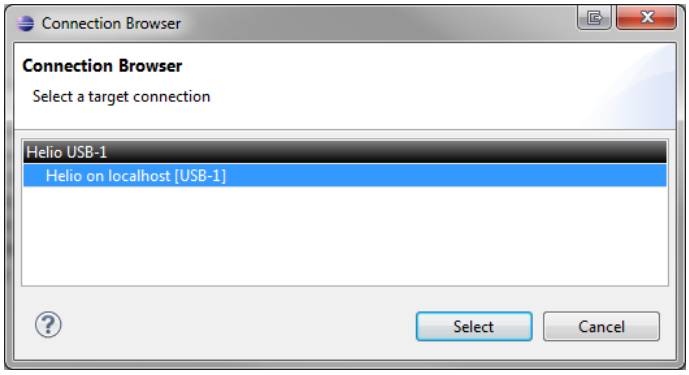
- ☐ Since the kernel is already running there is no need to tell the debugger which executable files to download to the target. However, the debugger does need access to the symbols that are in the running kernel. It will be necessary to **interrupt** the processor with **debugger commands** once a connection is established and
  add-symbol-file ~\13.1\embedded\embeddedsw\socfpga\prebuilt_images\vmlinux
- ☐ It is also necessary to tell the debugger where the **Source search directory** is located. This is the location of the extracted source in the first step of this lab.
- ☐ It may take several seconds for the debugger to connect to the target, download the symbol file and gain access to the environment of the running kernel. (If there are red error messages in the console window referring to System ID mismatches, they can be safely ignored.) Once DS-5 has kernel awareness, feel free to poke around and look at the **Active Threads** for each A9 processor. Look at the **Variables, Registers, Functions,** etc.
- ☐ When finished debugging the kernel, be sure to **Disconnect from target** so that additional debug sessions can be initiated.

- *Hints:*
  - *Create a new Debug Configuration to debug the kernel.*



- *Select the Debug Cortex-A9x2 SMP via Altera USB-Blaster under the Linux Kernel and/or Device Driver Debug tree.*
- *Locate and select the "Helio on localhost [USB-1]"*

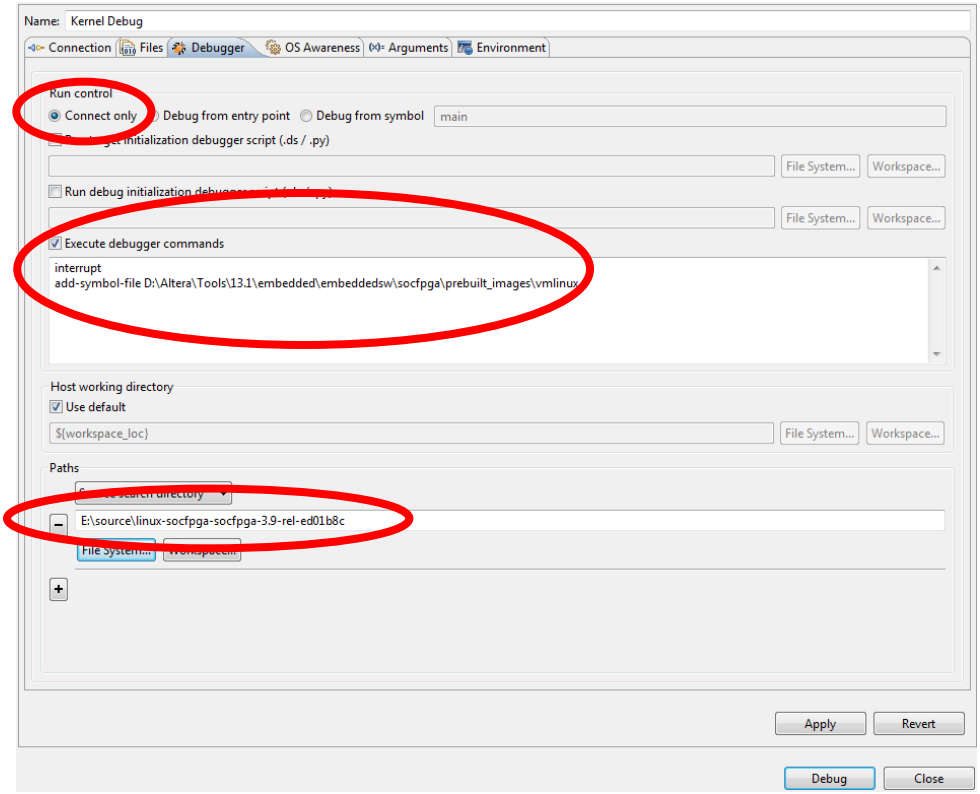




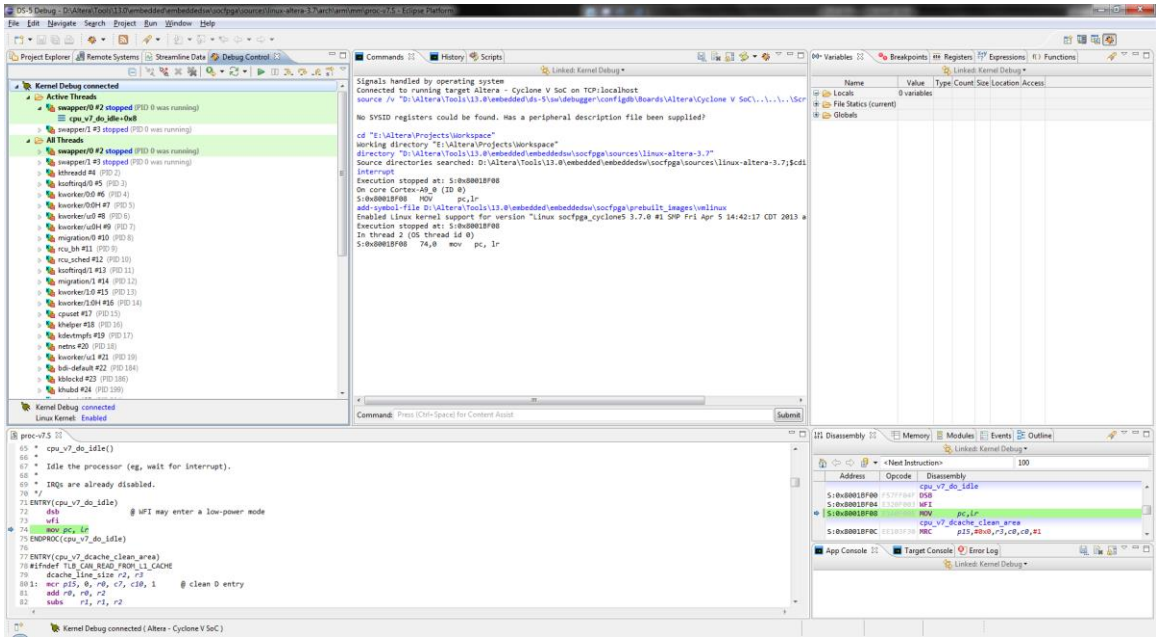
- *On the Debugger tab, select the option to Connect only*
- *Select the option to Execute Debugger commands and enter the following*
  - *interrupt*
  - *add-symbol-file D:\Altera\Tools\13.1\embedded\embeddedsw \socfpga\prebuilt_images\vmlinux (Your path may be different)*
- *Use the pull-down menu to select the Source search directory for the pre-built kernel*

- *E:\source\linux-socfpga-socfpga-3.9-rel-ed01b8c*
  (Your path may be different)

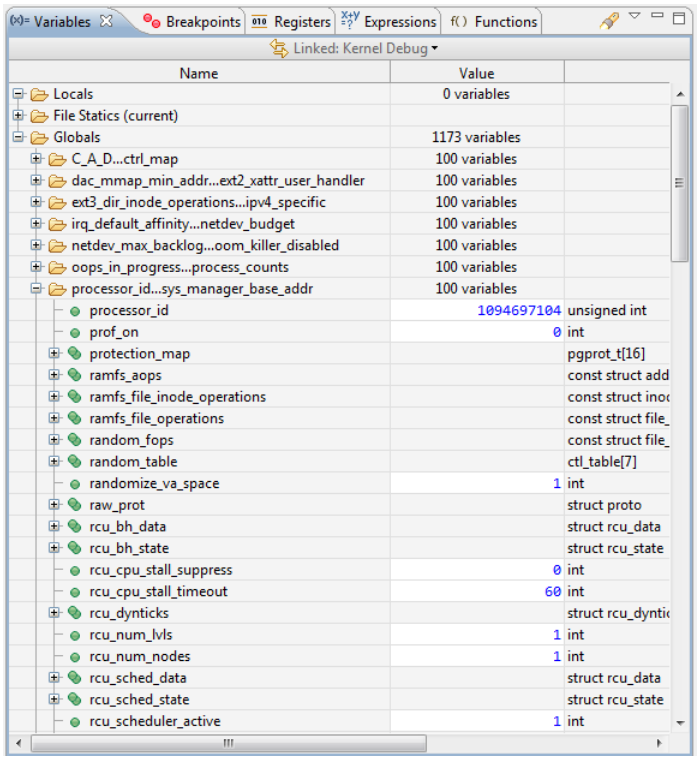
- *Click Debug to connect to target and gain access to the kernel*



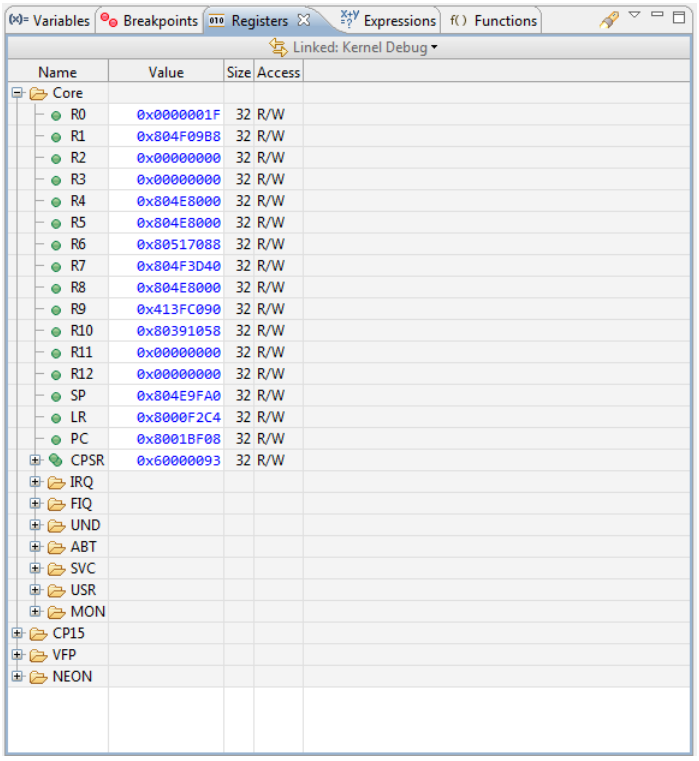
- *Explore the Debug Perspective*



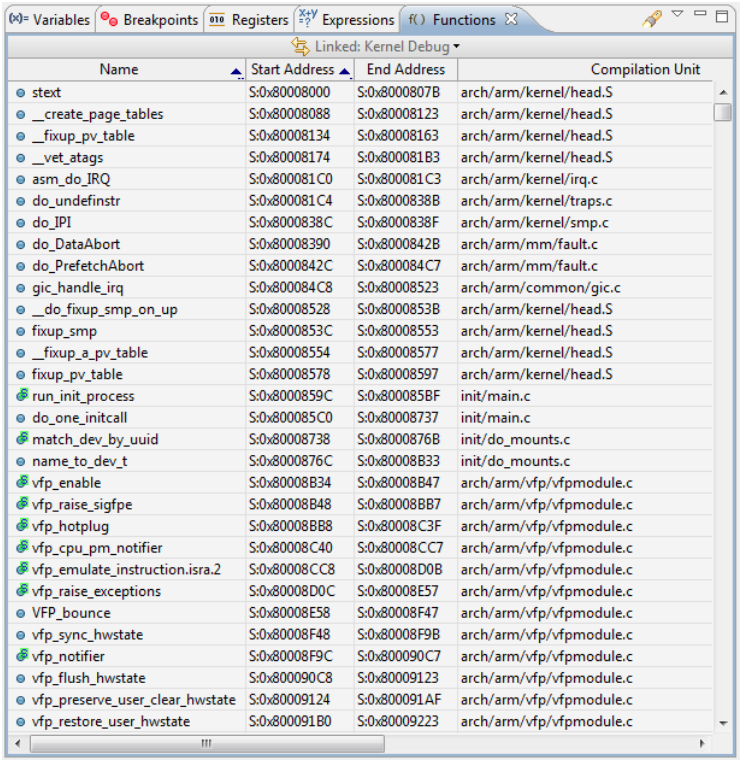
- *Variables window*



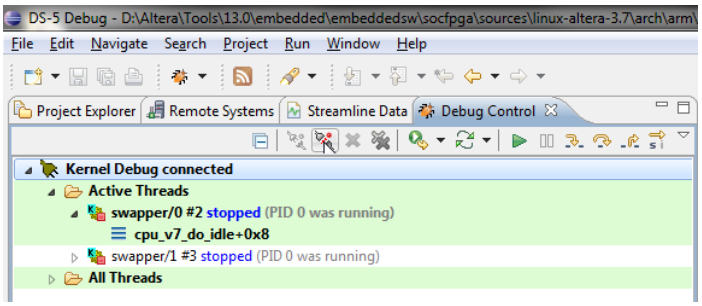
- *Registers window*



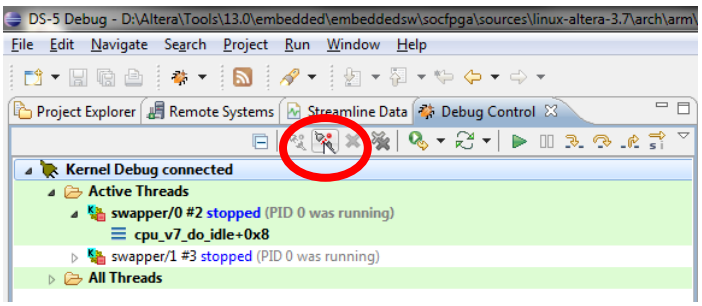
- *Functions window*



- *Active Threads*



- *Be sure to Disconnect from target when done*

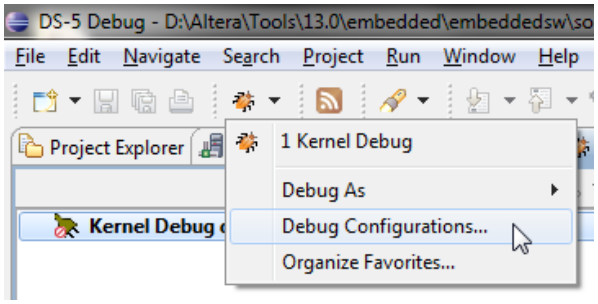


2.6 View Peripheral Registers

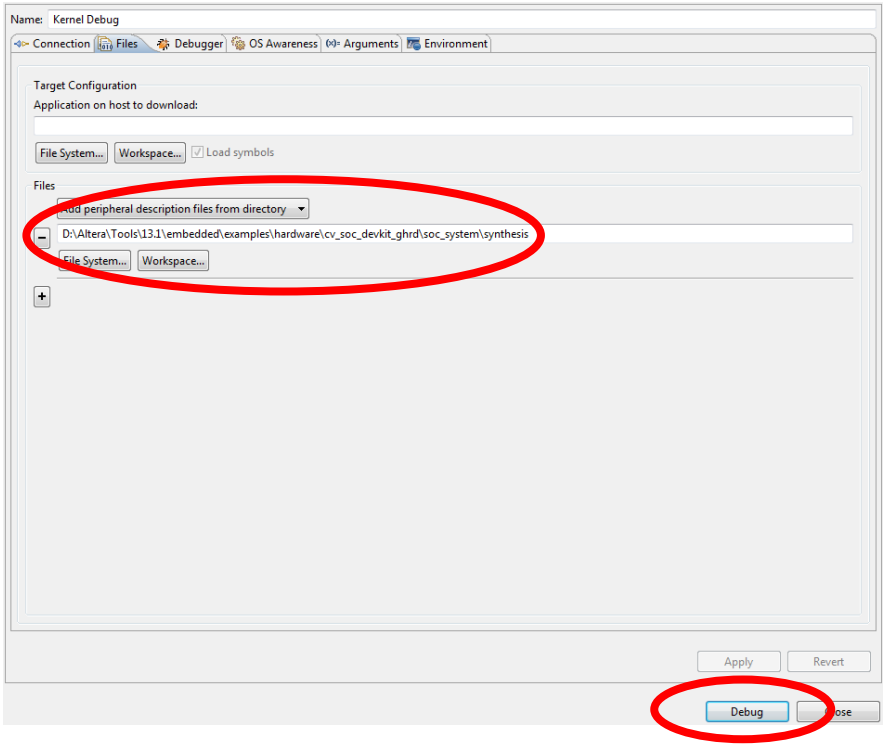
DS-5 Altera Edition has the ability to see all peripheral registers; HPS AND FPGA based user defined IP registers. After a complete hardware build is finished via Quartus-II, a soc_system folder is generated that contains the IP register descriptions, system.svd.

- ☐ Modify the kernel debug configuration from the previous step to **Add peripheral description files from directory** as stated above.
- ☐ Launch the **Debug** session again and notice the new **Peripheral** tree in the **Registers** window. Explore the tree to see not only the HPS peripheral registers, but also the FPGA based peripheral registers at the bottom of the list.
- ☐ Locate the **LED PIO** FPGA based peripheral and change the **DATA** value to observe the effect on the LEDs on the Helio board.
- ☐ When finished debugging the kernel, be sure to **Disconnect from target** so that additional debug sessions can be initiated.

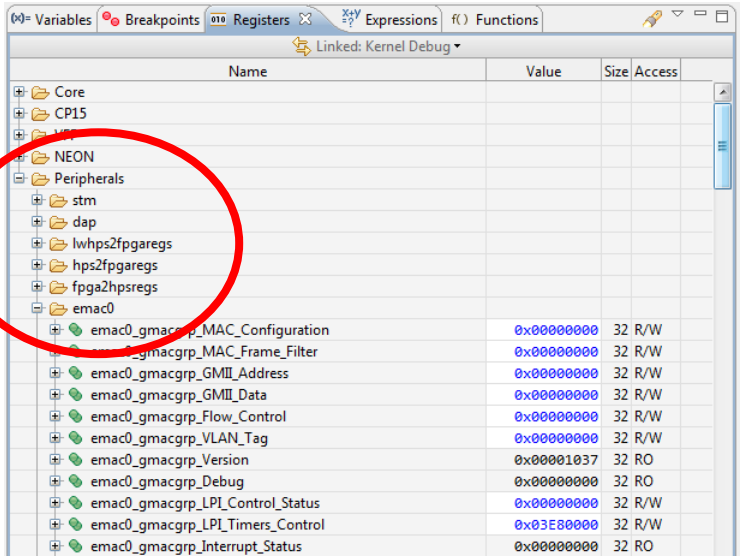
- *Hints:*
  - *Modify existing Kernel Debug Configuration*



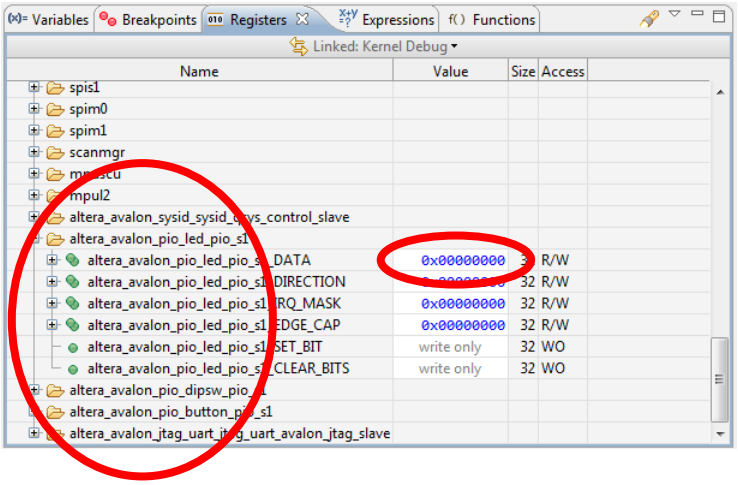
- *Add the location of the svd file:*
  D:\Altera\Tools\13.1\embedded\examples\hardware\cv_soc_devkit_ghrd\soc_system\synthesis
  (Your path may be different)
- *Initiate debug session*



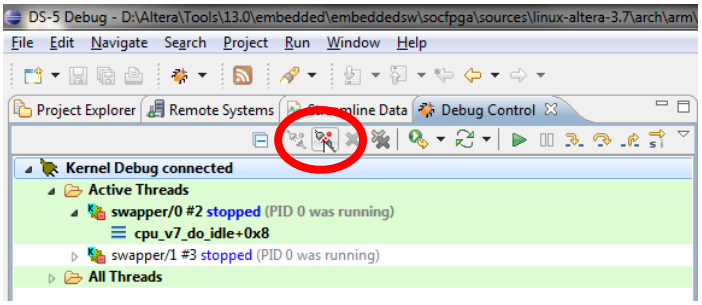
- *HPS peripheral registers*



- *FPGA peripheral registers*



- *Disconnect from target*



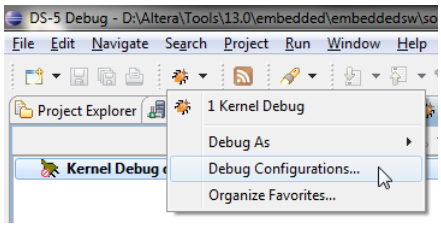
2.7 Trace Linux Kernel execution

This section presents an example of program trace using the Program Trace Macrocell (PTM) and storing the trace information in memory using ETR. The trace scenario presented here uses Linux kernel debugging as an example, but any application can be traced in the same way. As shown, the tracing can be selected to show current core, a particular core, or follow the currently executing thread.

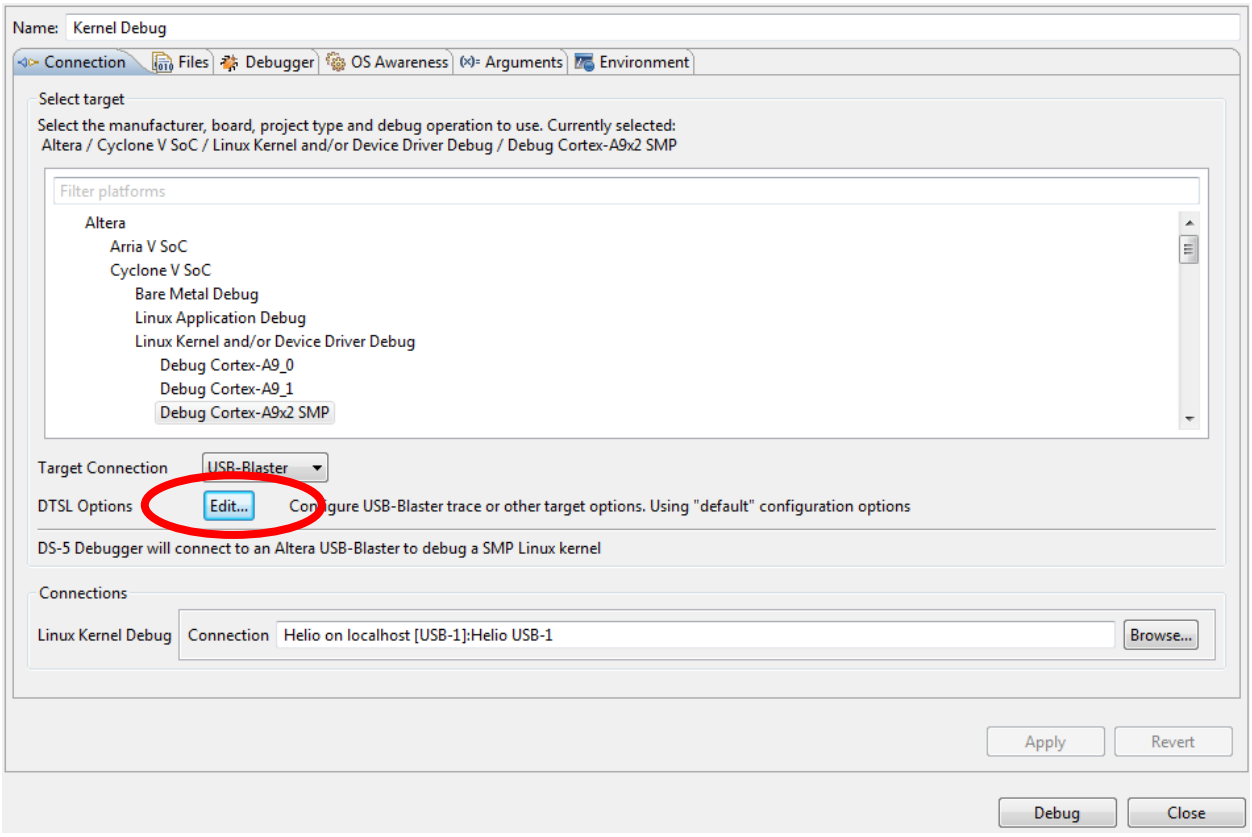
- ☐ Modify the kernel debug configuration from the previous step and **Edit** the Debug and Trace Service Layer (**DTSL**).
- ☐ Select the **System Memory Trace Buffer (ETR)** to capture the trace information in a section of the system memory on the target. Notice that FPGA On-Chip Memory may also be used to store trace data if accessible by the HPS.
- ☐ Enable the **Cortex-A9** to use **core trace**. We will be allowing tracing of both processors and the trace will have timestamps associated with it.
- ☐ Configure the **ETR** such that it will not interfere with other data in the system memory. (Defaults are fine here.)
- ☐ Before launching a new debug session, be sure to open a new **Trace** window view.

- ☐ Once the debugger is accessing the kernel, allow the debugger to **Continue** for a few seconds to populate the trace buffer. **Pause** the debugger and the trace window will populate.
- ☐ Use the mouse to move around in the trace window and observe the correlation to the disassembly code in the window below.
- ☐ Explore selecting different threads and the effect on the trace output.

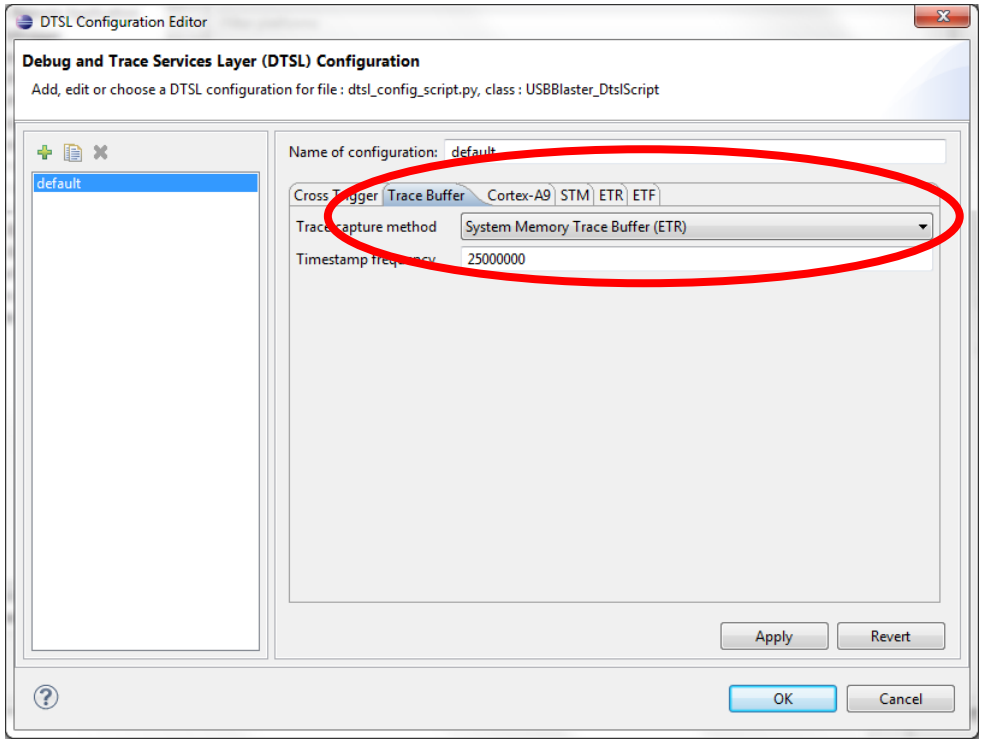
- *Hints:*
  - *Modify existing Kernel Debug Configuration*



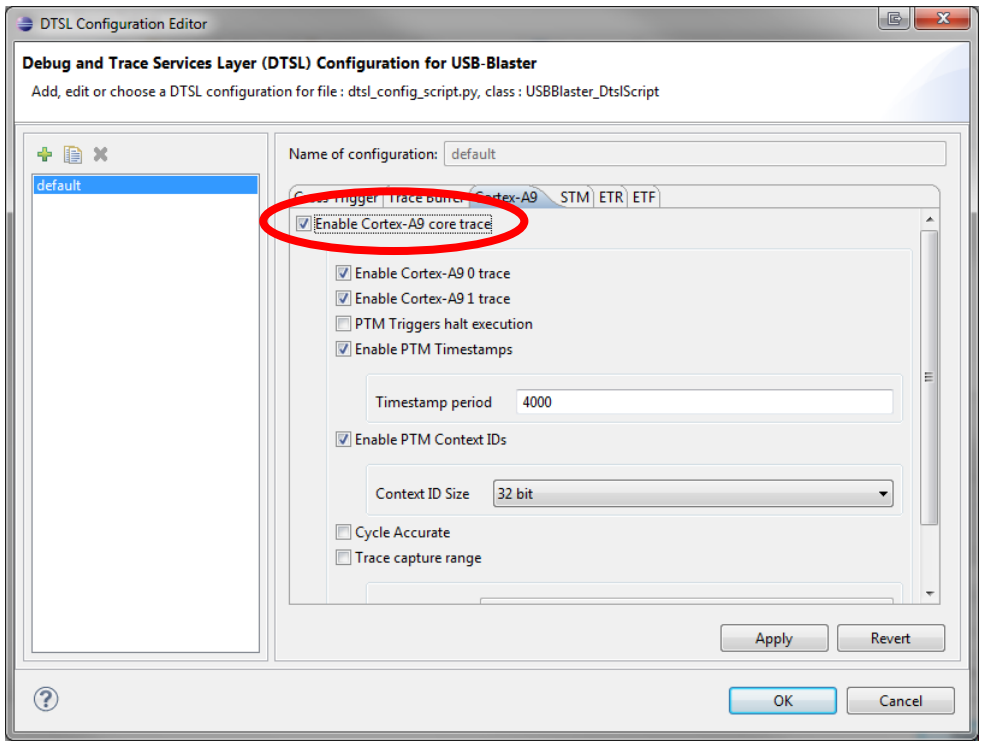
  - *Click Edit on the Connection tab to open the DTSL Configuration Editor*



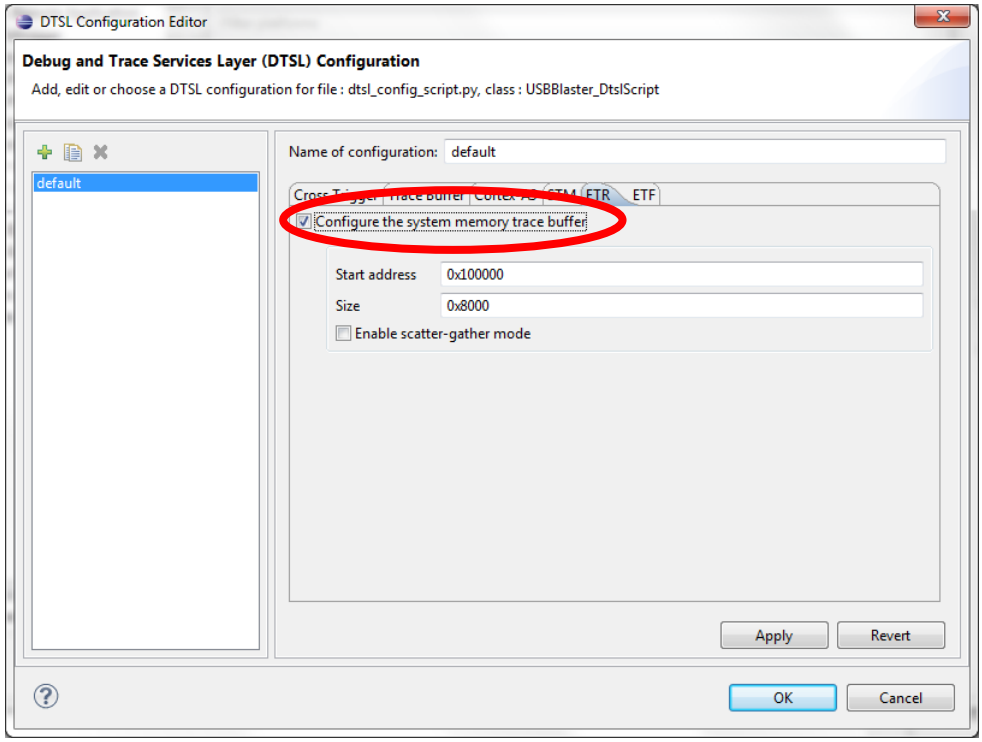
- *On the Trace Buffer tab, select the System Memory Trace Buffer (ETR)'*



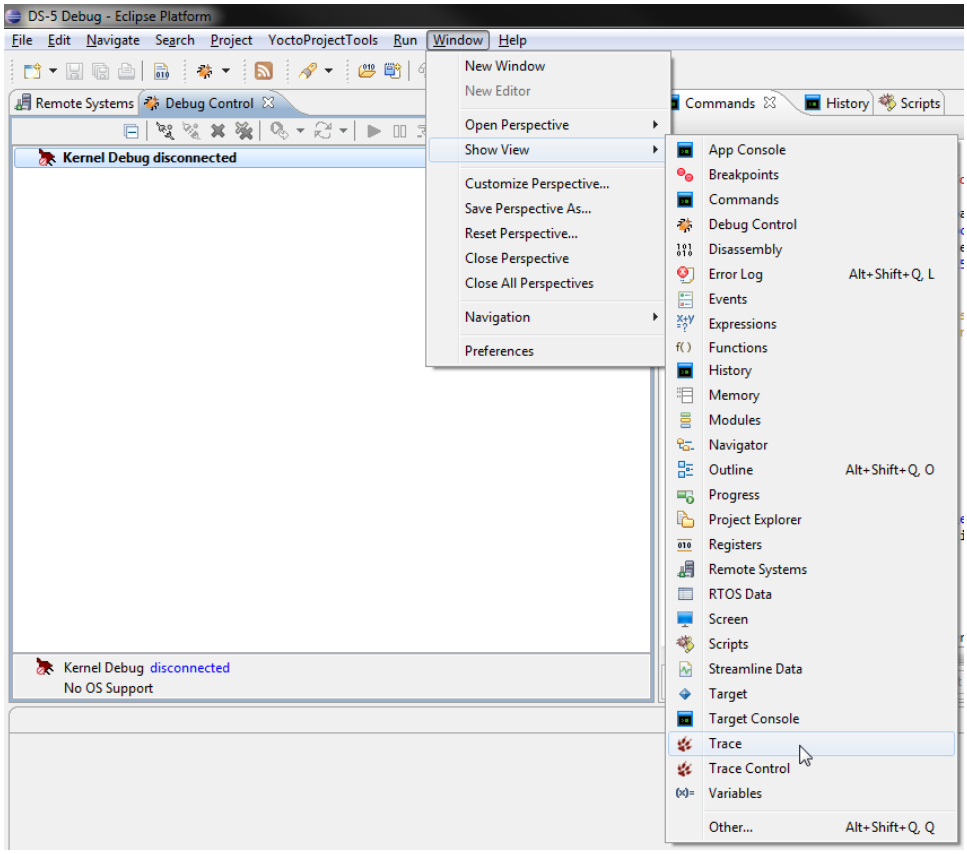
- *On the Cortex-A9 tab, enable the trace capability by checking the Enable ortex-A9 core trace box.*



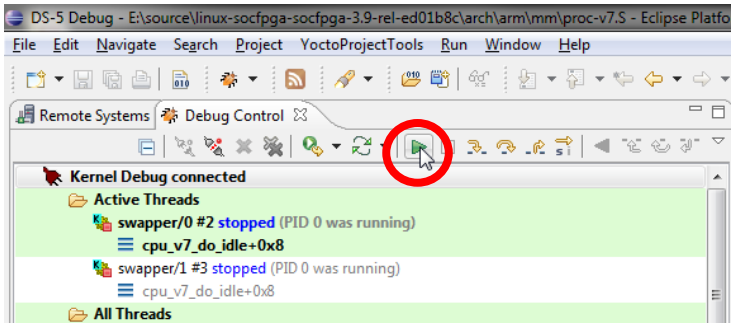
- *On the ETR tab, check the Configure the system memory trace buffer box. The default values here are a safe location and size that will not interfere with the running kernel.*



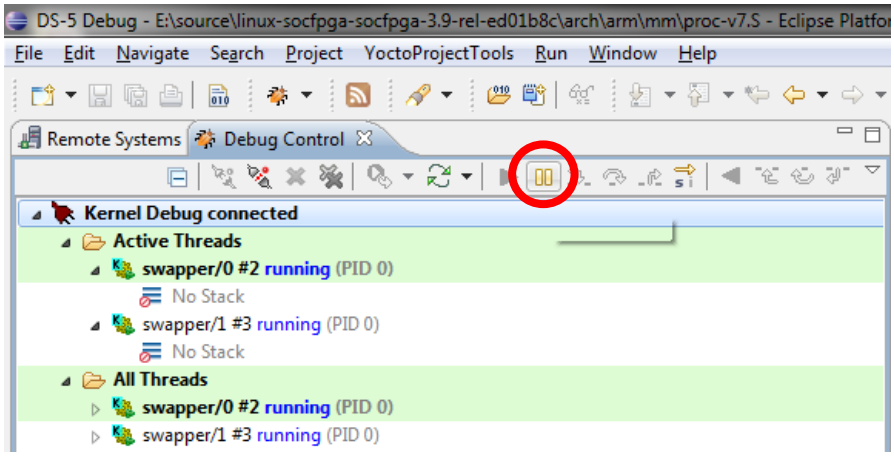
- *Open a new Trace window*

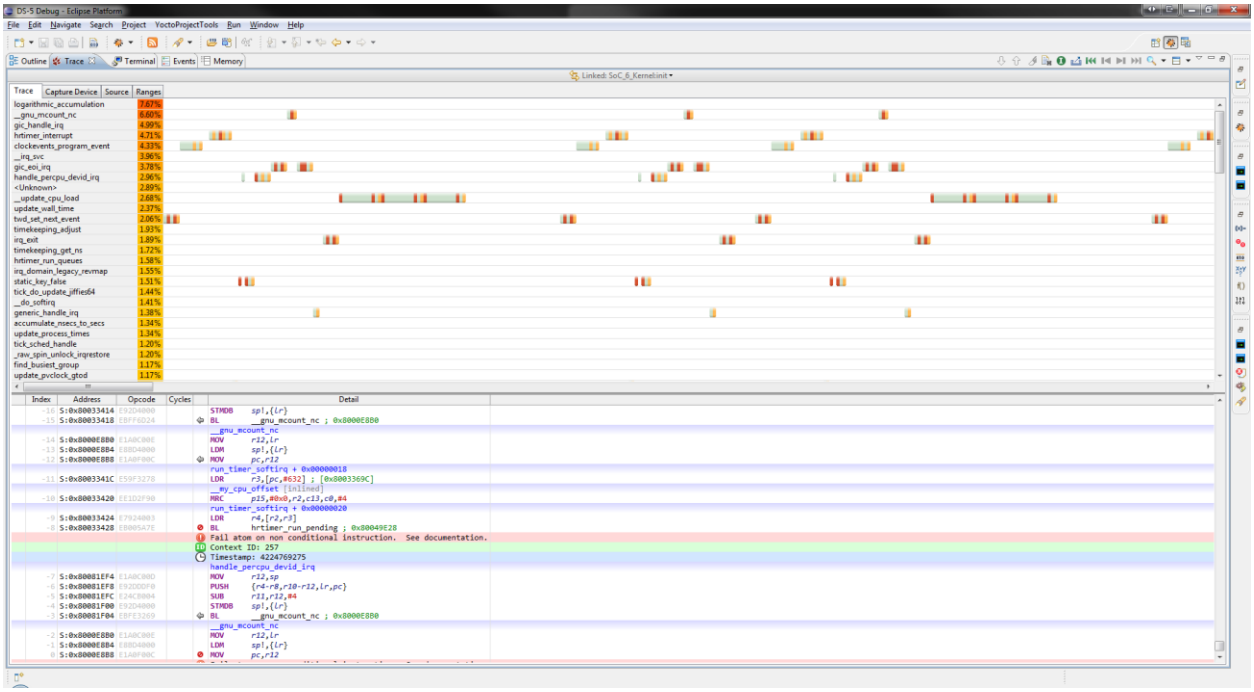


- *Before starting the debug session, press the warm-reset button, SW6, to reload the kernel into a known good state Allow to boot to login prompt. It may be in the weeds depending on how you disconnected one of the previous debug sessions.*
- *Start the debug session and once connected, select Continue and allow the system to run for a few seconds to populate the trace buffer.*



- *Pause the debugger and allow the trace buffers to be uploaded and populate the trace window.*





2.8 Enable Cross-Triggering

Finding out whether a complex problem is caused by a hardware or software bug is normally the first step towards a fix. The main methodologies for debugging across the hardware and software worlds consist of:

- Triggering on an error condition in the software, and analyzing the state of the hardware around that point in time.
- Triggering on an error condition in the hardware, and exploring what the software was doing around that point in time.

- Visualizing a history of software instructions and hardware RTL waveforms, and aligning them on events of interest in order to explore the relationship between the two.

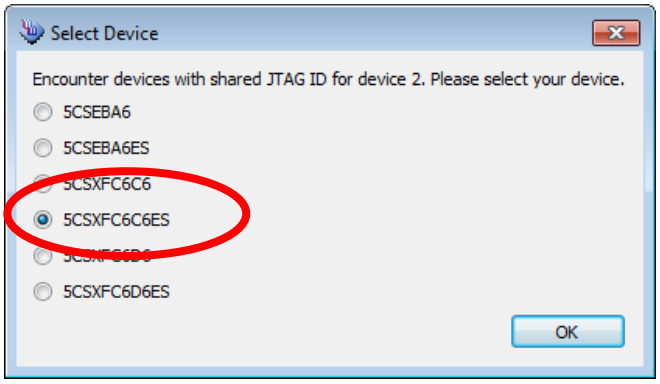
Each of these methodologies requires dedicated debug hardware on the boundary between the hardened processor system and the FPGA fabric. Altera SoCs include a comprehensive implementation of ARM CoreSight on-chip debug and trace logic, including a cross-trigger matrix that connects input and output triggers from all the processors and trace macrocells in the processor sub-system and hardware triggers to and from the FPGA fabric. The cross-trigger matrix can be easily programmed from the DS-5 Debugger to configure which hardware blocks generate triggers and which components are affected by them.

It is necessary to have the FPGA programmed with the example design for the Helio board which includes the FPGA LED design instrumented with SignalTap. You will need to program the FPGA configuration PROM that is on the bottom side of the Helio board such that on board power-up, the FPGA will be configured. (Alternatively, if you are familiar with programming Altera devices, you can program the FPGA directly with the helio_ct.sof file and skip programming the EPCQ as directed below.)

2.8.1 Program the FPGA or EPCQ PROM

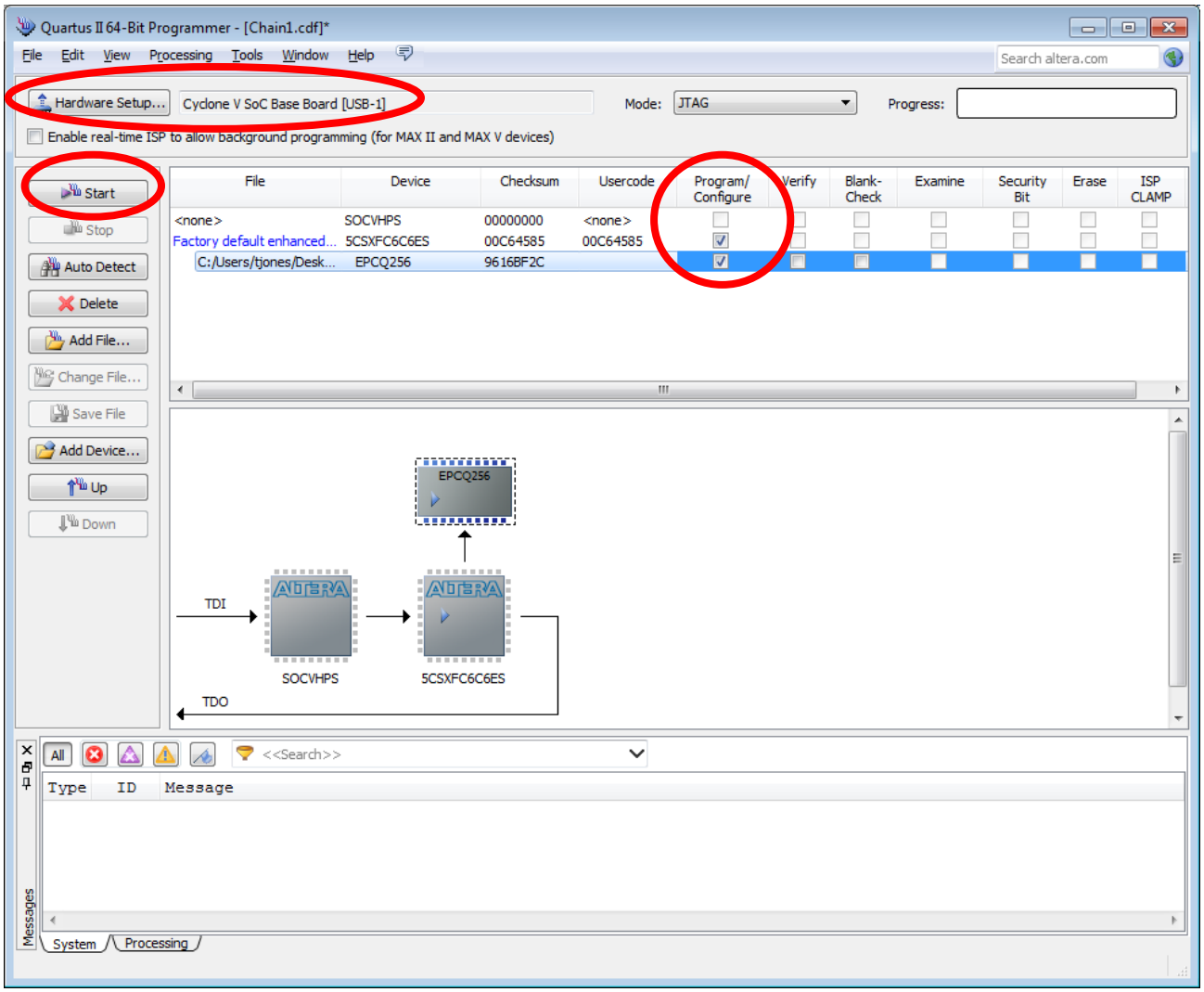
It is necessary to have the FPGA programmed with the hardware design that includes the SignalTap II modules used for cross triggering. The helio_ct.jic file can be used to program the EPCQ PROM and can be found in the resources folder included with this lab manual. Following are the steps necessary to program the EPCQ. Alternatively, if you are familiar with programming just the FPGA, use the helio_ct.sof included the resources folder.

- ☐ Connect a USB cable to the **BLASTER USB** port on the Helio board.
- ☐ Launch the Quartus II Programmer
  - Linux: **~\quartus\bin\quartus_pgmw**
  - Windows: **~\altera\13.1\quartus\bin64\quartus_pgmw.exe**
- ☐ **Auto Detect** the chain and select the **5CSXFC6C6ES** device.



- ☐ Right click on the FPGA device and chose **Change File** and select the **helio_ct.jic** file.
- ☐ Check the box to **Program/Configure** the EPCQ256 device. The FPGA will automatically be selected as well.

- ☐ Ensure the **Cyclone V SoC Base Board [USB-1]** connection is visible in the **Hardware Setup** dialog box. If not, click the button and select the cable. (Drivers must be previously installed. See the Altera web site for instructions.)



- ☐ Click **Start** to initiate the programming of the EPCQ PROM. Programming the EPCQ will take several minutes.

- ☐ Upon successful programming, power-cycle the board and observe the 4 LEDs cycling near the power input jack.

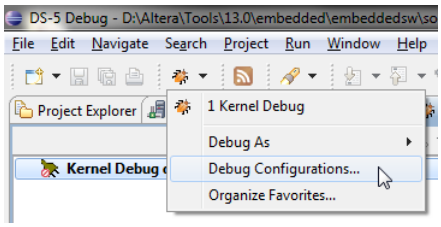
2.8.2 Enable cross-triggering

In order for the ETM to generate HPS to FPGA trigger out conditions as well as receive FPGA to HPS triggers in, you must enable the DTSL option as such.

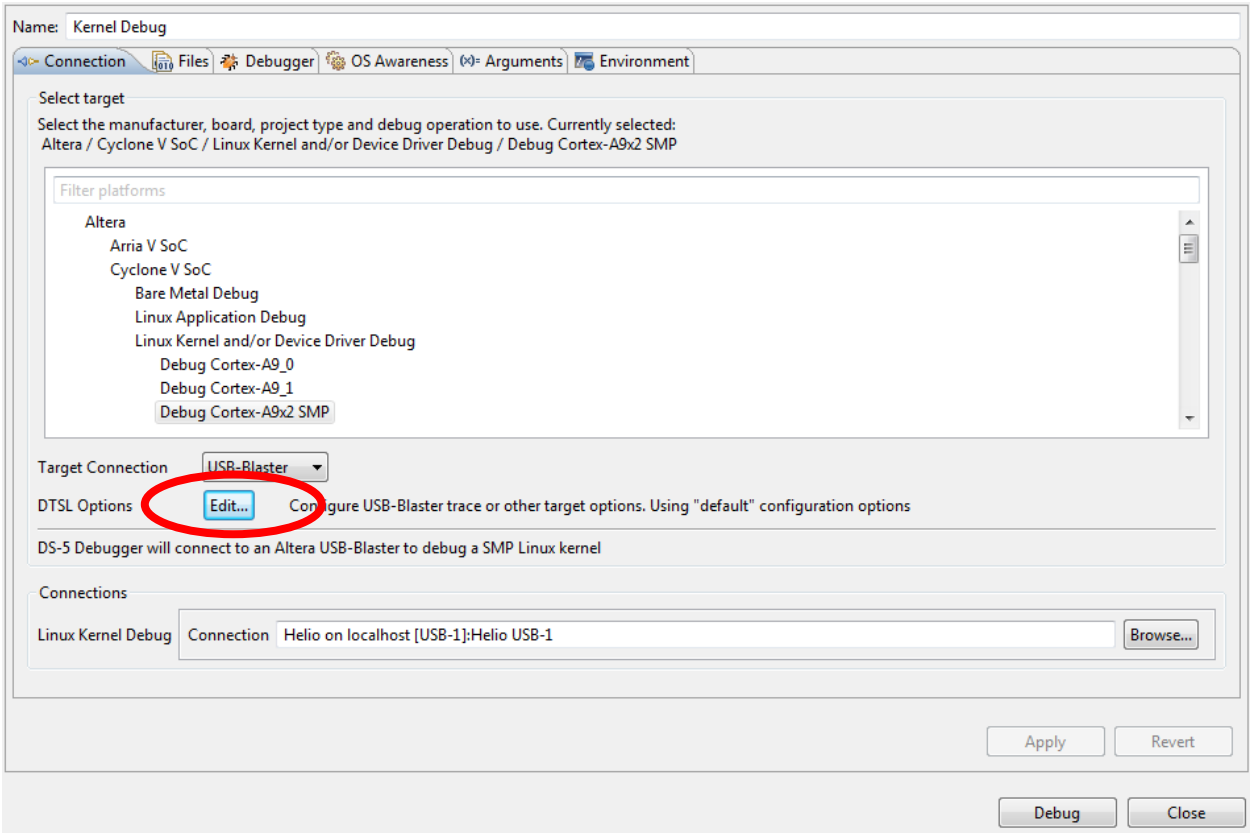
- ☐ Modify the kernel debug configuration from the previous step and **Edit** the Debug and Trace Service Layer (**DTSL**).

- ☐ Enable cross triggering in both directions and assume the triggers are accessible.

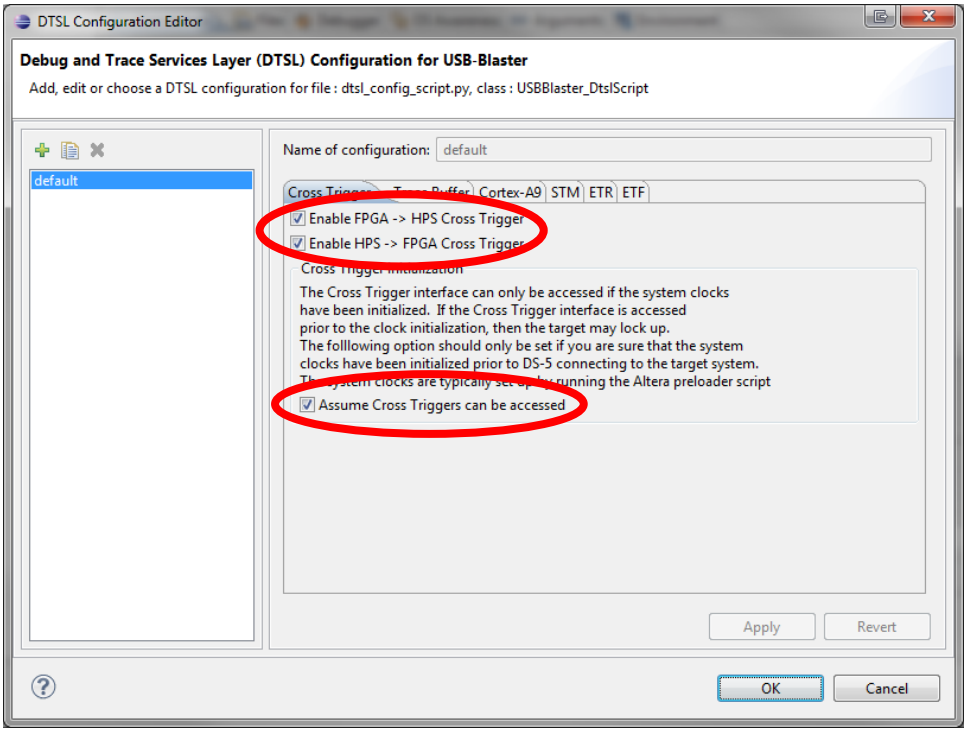
- *Hints:*
  - *Modify existing Kernel Debug Configuration*



  - *Click Edit on the Connection tab to open the DTSL Configuration Editor*



- *On the Cross Trigger tab, select Enable FPGA -> HPS Cross Trigger, select Enable HPS -> FPGA Cross Trigger and select Assume Cross Triggers can be accessed*

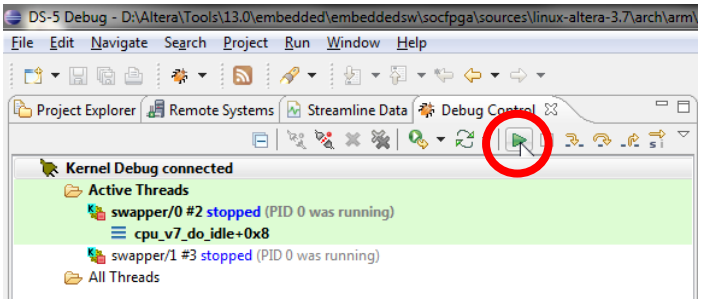


2.8.3 HPS to FPGA cross-triggering

For the HPS to trigger the FPGA based SignalTap system, you will need to allow the software to free run, enable the SignalTap system to trigger off the HPS generated trigger, then "interrupt" the free running kernel as if it hit a breakpoint.

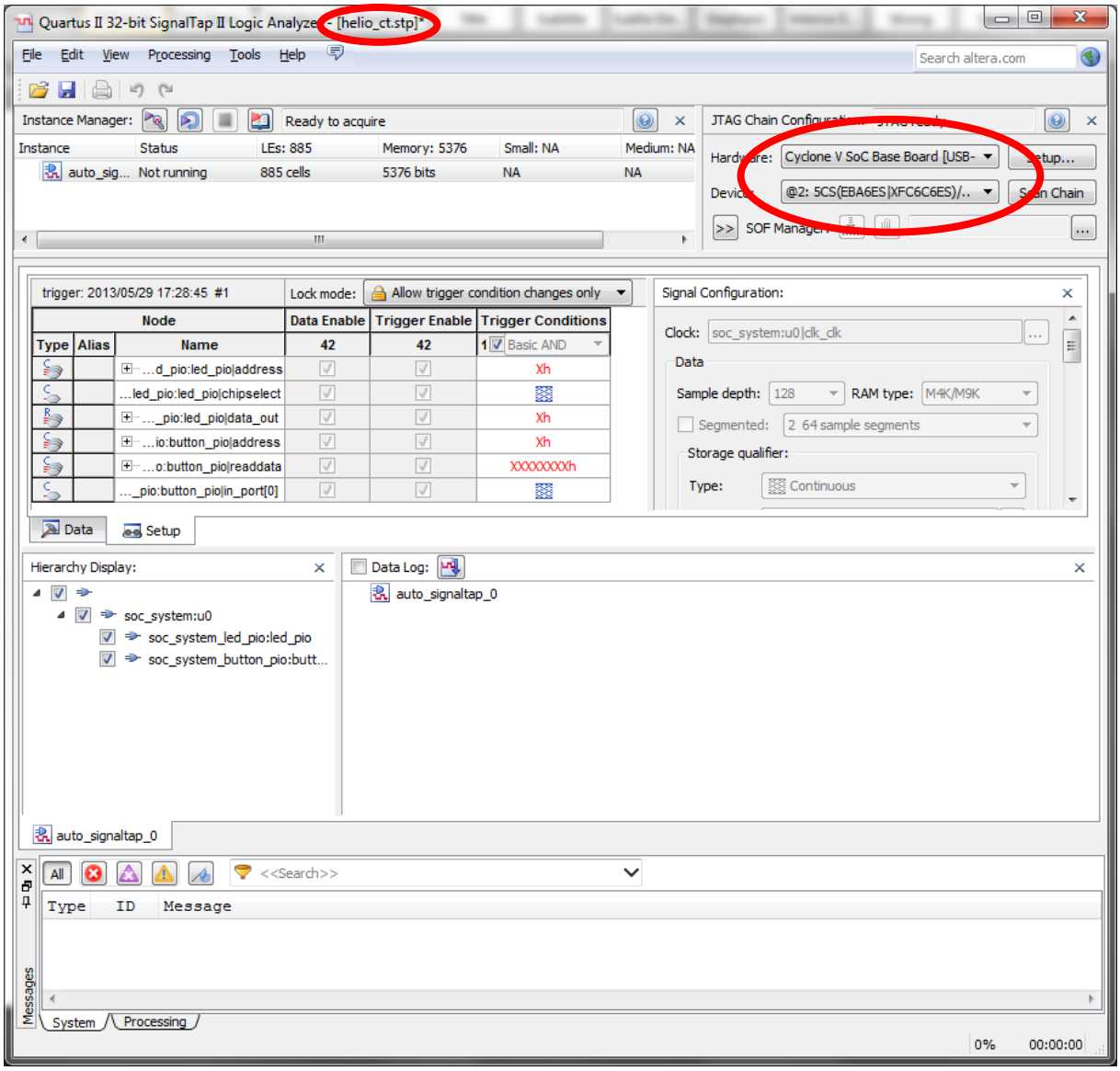
- ☐ Allow the kernel debug session to free run.
- ☐ **Launch** SignalTap and **Open** the pre-defined instrumented system with the **helio_ct.stp** file.
- ☐ SignalTap communicates over the same JTAG connection as the DS-5 debugger. Once SignalTap is up and running make sure you are communicating with the correct JTAG **device**.
- ☐ The pre-defined SignalTap system is setup such that it will trigger on the incoming trigger signal from the HPS CTI. Activate SignalTap system **analysis**.
- ☐ **Pause** the actively running kernel debug session to mimic a software breakpoint being hit.
- ☐ Both the DS-5 AND SignalTap are triggered and have captured data at the exact point in time where the break occurred.

- *Hints:*
  - *Start the kernel debug session once again and when connected, select **Continue** and allow the system to run.*

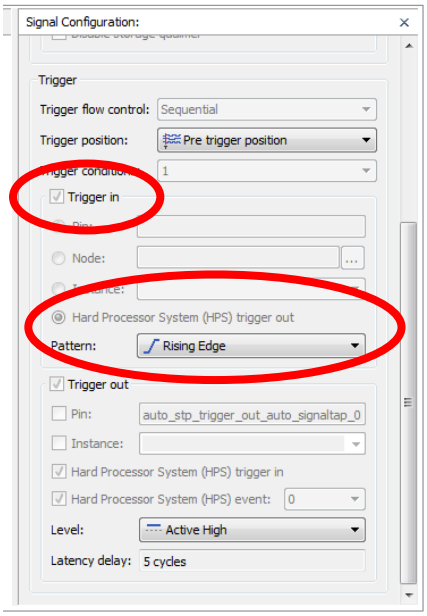


  - *Launch **SignalTap** (Your path may be different)*
    - *Windows: ~\altera\13.0\qprogrammer\bin\quartus_stpw.exe*
    - *Linux: ~/altera-tools/13.0/quartus/bin/quartus_stpw*
  - ***Open helio_ct.stp** (Located in resources folder for this lab)*

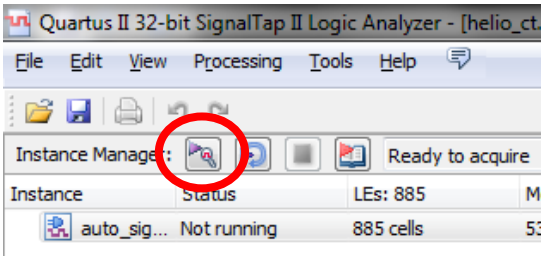
- *Select correct **Hardware (USB Blaster)** and **Device***



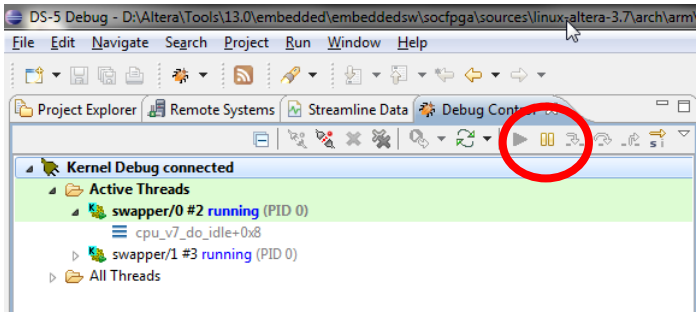
- *Take note of HPS trigger condition (from the SignalTap system viewpoint)*



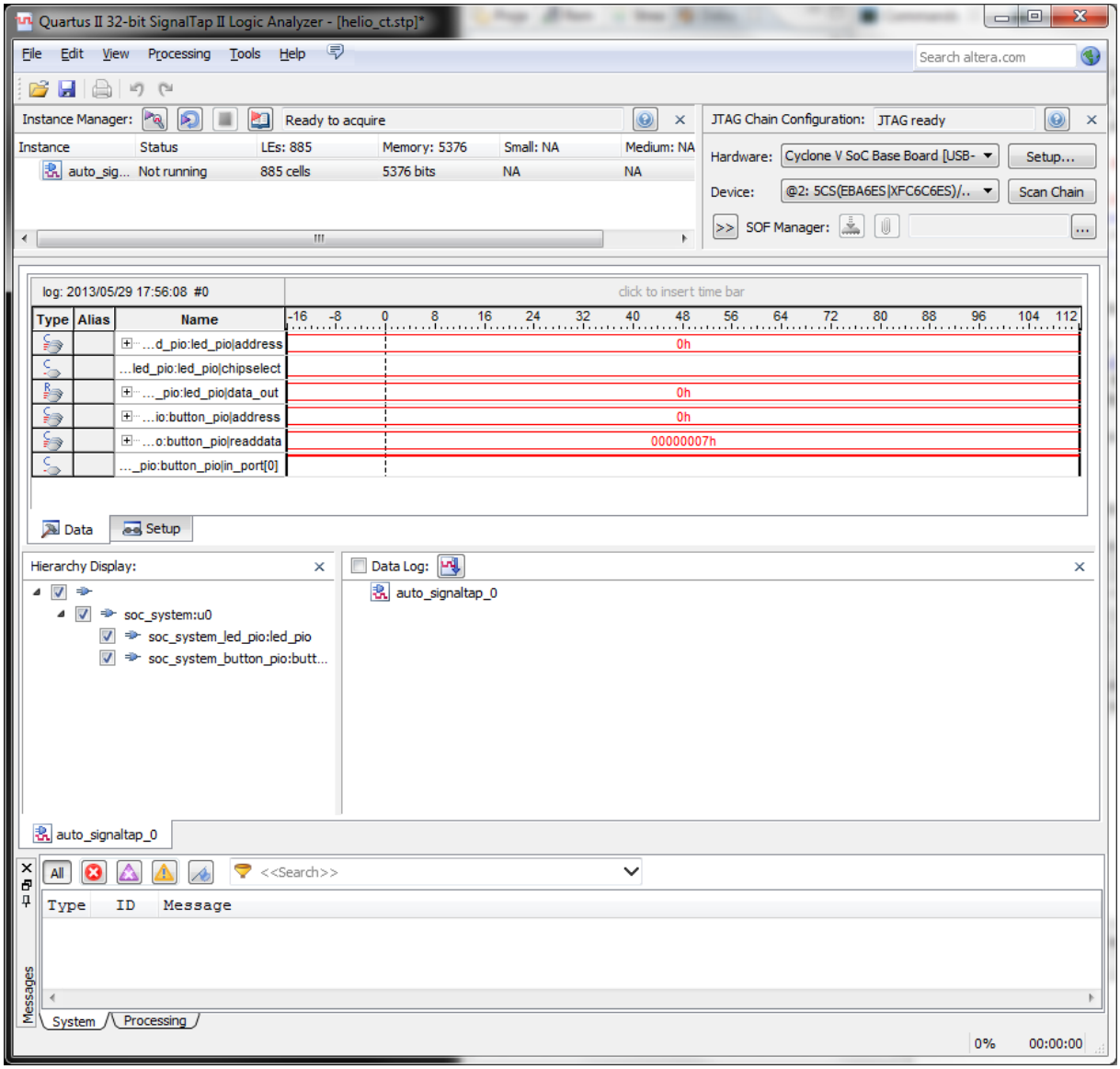
- ***Run Analysis***



- ***Pause*** *the kernel debug session to mimic breakpoint being hit*



- *SignalTap is now triggered and waveforms are captured (nothing really interesting here other than the fact that the software break triggered SignalTap)*



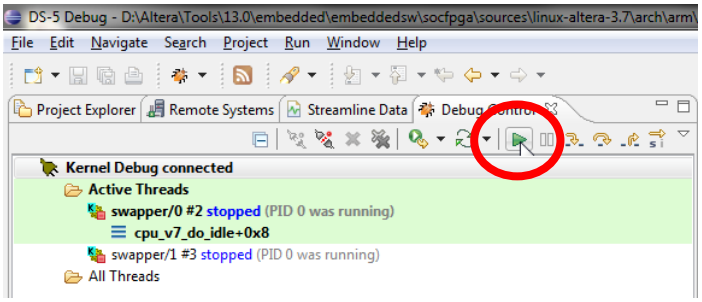
2.8.4 FPGA to HPS cross-triggering

For the FPGA to trigger the HPS you will need to allow the software to free run, enable the SignalTap system to trigger off an event, a push-button on the Helio board in this case, then "interrupt" the free running kernel.

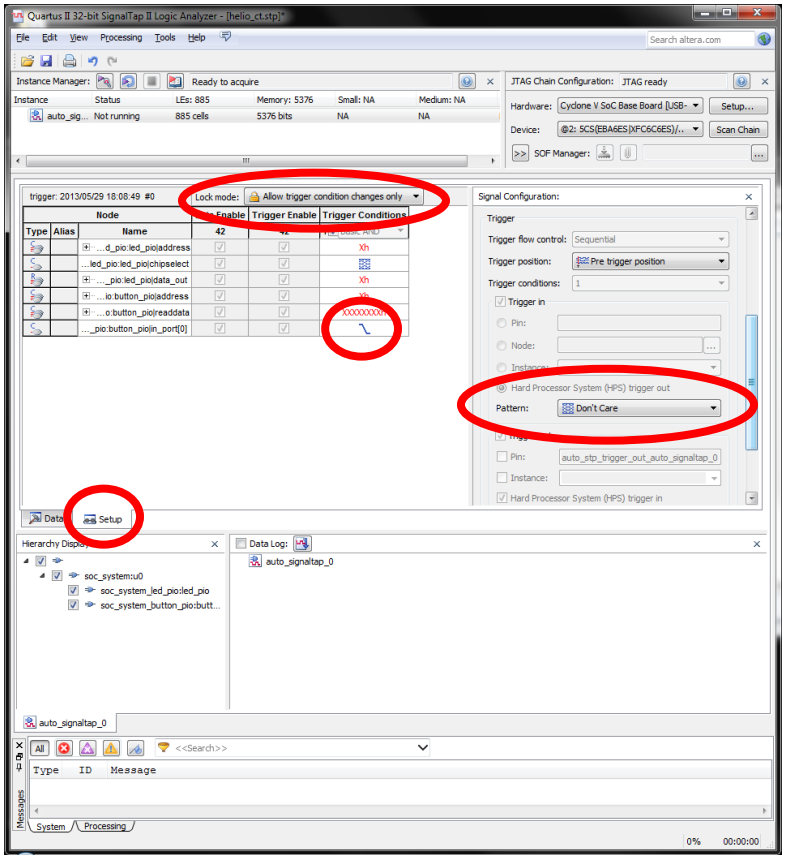
- ☐ Allow the kernel debug session to free run.
- ☐ Modify the SignalTap **setup** such that there is a **Don't Care** for the HPS incoming trigger and the system will trigger on the **falling edge** of the **push-button input**.

- ☐ Activate SignalTap system **analysis.**
- ☐ Press the **SW11** push-button on the Helio board to trigger the SignalTap system.
- ☐ Both DS-5 AND SignalTap are triggered and have captured data at the where the hardware trigger occurred.

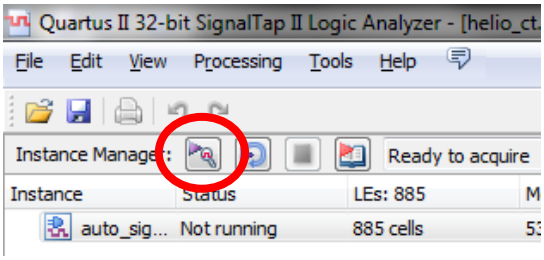
- *Hints:*
  - ***Continue** the kernel debug session again in DS-5.*



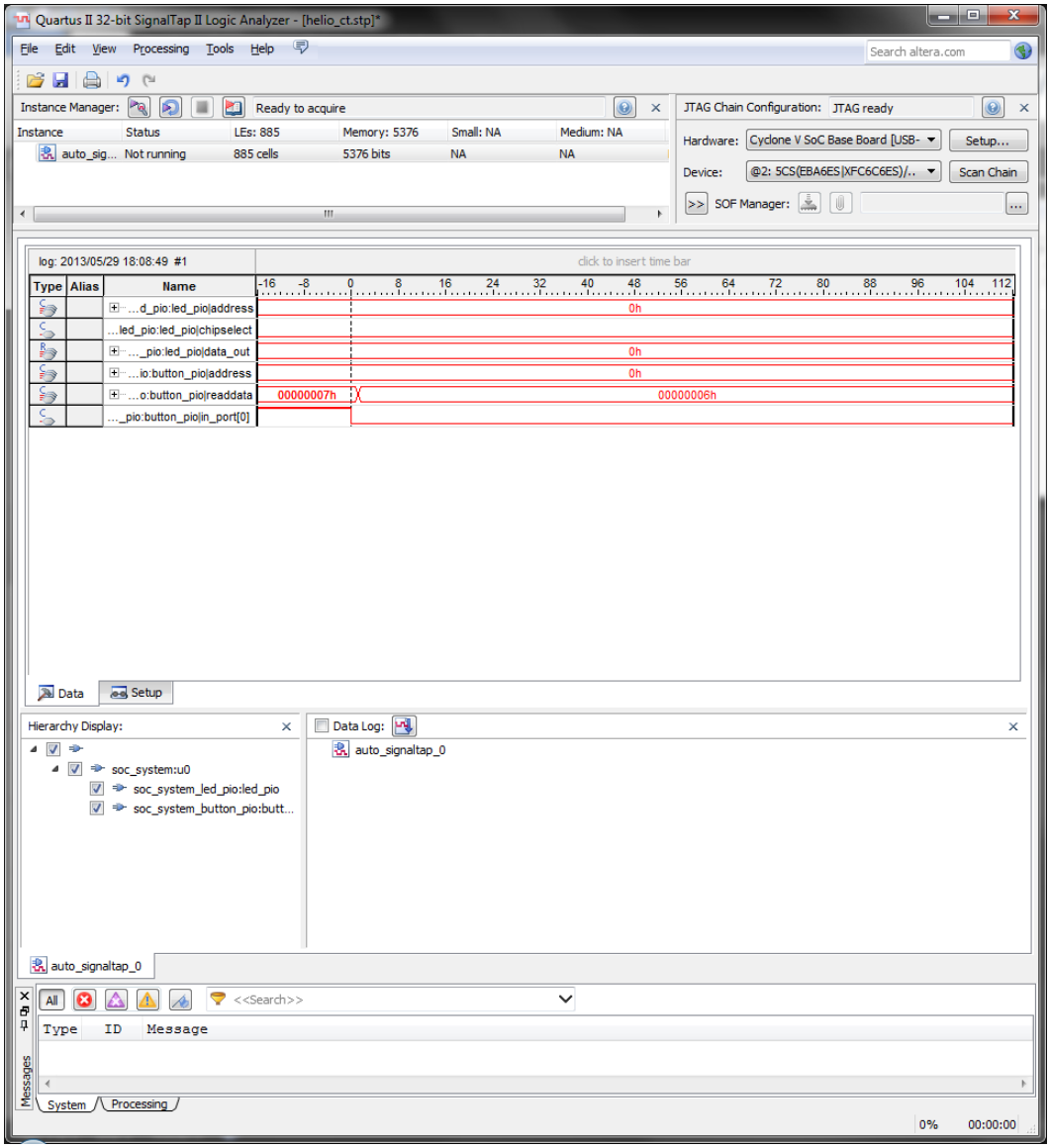
  - *In SignalTap, verify the **Lock mode** to **Allow trigger condition changes only.***
  - *On the **Setup** tab change the HPS **Trigger In Pattern** to a **Don't Care.***
  - *Select **Falling Edge** for **...button_pio\|in_port[0]** (Right-click in **Trigger Condition** column)*



- ***Run Analysis***



- *Press the **SW11** push-button (just below 4 illuminated LEDs) to trigger SignalTap*
- *Waveforms are captured and the debug session in DS-5 is now triggered near the point the push-button was pressed.*



3 Notes

Document Revision History

Revision	Date	Comments
0.1	May 8, 2013	Initial Draft
0.3	May 24, 2013	Internal Review
1.0	May 29, 2013	Initial Release
2.0	March 18, 2014	Updated to 13.1 tools