sirius
powered by phact

Debugging Octeon with Eclipse

Debugging Octeon with Eclipse

In our tutorial “Using Eclipse with Cavium Octeon SDK”, we introduced a way to develop C/C++-code for Octeon from the graphical IDE Eclipse. In this tutorial we take it one step further and introduce a way to debug your code from Eclipse.

Using real hardware

We will need some real hardware on which we can run our application (we talk about debugging with the simulator later). We use the VSN IP Engine as an Octeon development platform, but most of the Octeon production and evaluation boards will do. Plug it in, connect it to your Linux PC using a serial cable or USB, switch it on and you are up and running. Below is a picture that shows the hardware setup for debugging. The IP Engine is connected to our Linux PC via a null-modem cable and connected to our LAN. After Linux has started on the IP Engine, we have assigned it a valid IP address.

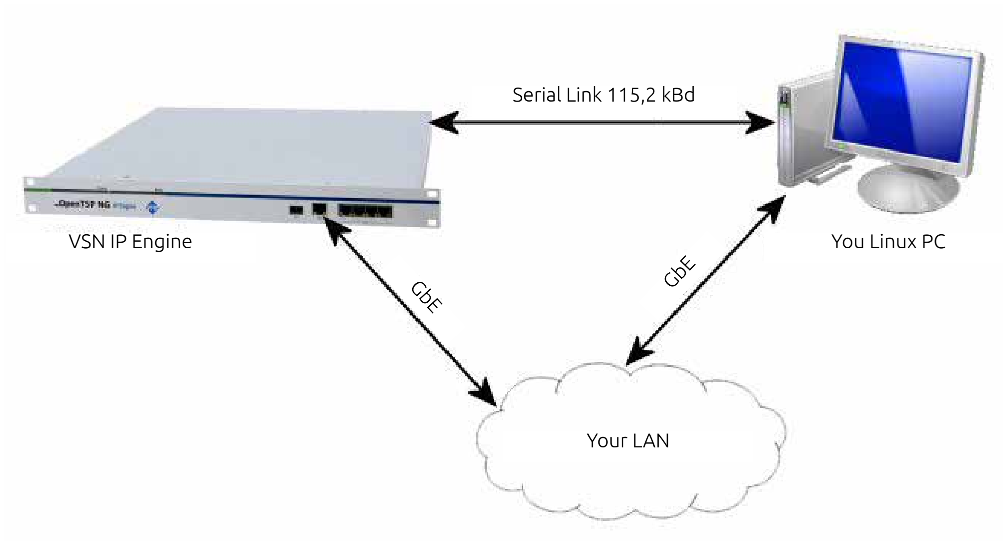


Figure 1: Using VSN IP Engine as a Octeon development platform

As done before in the tutorial “Using Eclipse with Cavium Octeon SDK”, we use the passthrough project. Make sure that you disable compiler optimization (using the –O0 compiler switch) and enable insertion of debugging information (using the –g compiler switch). Open the Makefile of the passthrough project with your favorite editor, search for *CFLAGS_LOCAL* and change it from

CFLAGS_LOCAL = -g -O2 -W -Wall -Wno-unused-parameter

to

CFLAGS_LOCAL = -g –O0 -W -Wall -Wno-unused-parameter

After this, do clean rebuild of the passthrough project.

Next, we have to prepare the VSN IP Engine for debugging. Open a terminal session on your Linux PC using minicom (115200,N,8,1, no hardware handshake) to monitor the serial output of the IP Engine. Switch on the IP Engine and wait for the message *Hit any key to stop autoboot* to appear in the minicom window. Hit a key on the keyboard of your Linux PC. This will stop the boot process in UBoot on the VSN IP Engine.

Type *run linux_cf* to start Linux on the VSN IP Engine. When Linux has booted, check the IP address of the port you connected to your LAN, in our example this is GBE, so we type

```
ifconfig gbe
```

This should reveal a valid IP address for your LAN. If not, use the *ifconfig* command to bring up the port and change its IP address. Our output looks like this

```
gbe  Link encap:Ethernet HWaddr 00:09:25:0B:00:24
   inet addr:10.6.193.225 Bcast:10.6.193.255 Mask:255.255.254.0
   inet6 addr: fe80::209:25ff:fe0b:24/64 Scope:Link
   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
   RX packets:95 errors:0 dropped:0 overruns:0 frame:0
   TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
   collisions:0 txqueuelen:0
   RX bytes:9257 (9.0 Kb) TX bytes:492 (492.0 b)
```

Copy the passthrough executable from the Linux host PC to the VSN IP Engine using a terminal window as follows

```
scp passthrough 10.6.193.225:/mnt/cf
```

Next, you need to restart the IP Engine by rebooting Linux (just type **reboot** in the minicom window) and again stop the boot process in UBoot. Now modify the environment variable **bootcmd** by typing:

```
setenv bootcmd ‘fatload ide 0 $(loadaddr) passthrough;bootoct $(loadaddr) coremask=1 debug=0’
```

followed by

```
saveenv
```

to save the new value of *bootcmd* to flash.

To start the passthrough debug example, type

```
run bootcmd
```

Wait until the bootloader has loaded the application and is waiting for the debugger to connect, then disconnect the serial connection, so no program is using the serial port. We are going to need this for the debugger.

On your Linux host system, generate a debugger command file named *debug-vsnipengine.arg* in the passthrough project directory that contains the following lines

```
set spawn-sim off
file passthrough
target octeon /dev/ttyS0
b __cvmx_interrupt_default_exception_handler
```

This file is loaded by the debugger at startup and it executes the commands defined in the file. The first line makes sure the simulator is not started when the debugger is started. Next, the file *passthrough* is loaded into the debugger, and a connection is made with the VSN IP Engine using serial port 0. If you use another port, change the number accordingly. The last line sets a breakpoint on the execution handler. This comes in very handy when a software crash occurs, which is not unimaginable in the early stages of development. After a crash, simply type

```
bt
```

in the command prompt and the debugger will show the call stack just before the crash. This will probably give you a good hint as to what caused the crash. More on debugging commands can be found in the Octeon SDK documentation, see the *Simple Executive Debugger* section.
When you want to debug the VSN IP Engine at a remote site, a terminal server can be a great device. A terminal server is a device that converts serial data to IP and vice versa. When debugging via a terminal server, you change the second line of the file *debug-vsnipengine.arg* of the line to

```
target octeon tcp:<ipaddress>:<port>
```

where <ipaddress> is the IP address of your terminal server and <port> is the IP port the terminal server listens to (See figure 2 below). Again, see the SDK documentation section *Simple Executive Debugger* for more information on debugging using terminal servers.

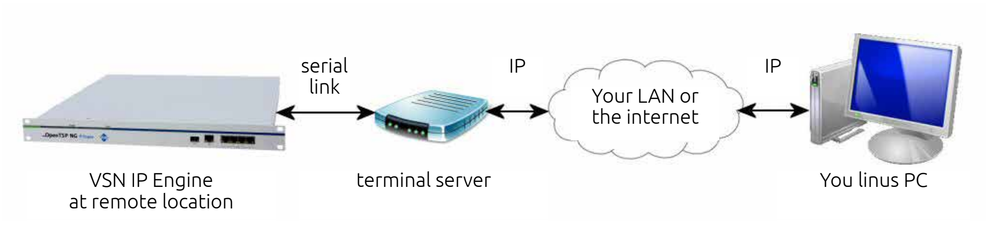


Figure 2: Using a terminal server for debugging

The final step to enable remote debugging with Eclipse is making the Eclipse environment understand the Octeon debug environment. So fire up Eclipse (if not already up and running) and load the passthrough project which we used in the tutorial “Using C++ on Cavium Octeon SDK” to enable C++ programming on the Octeon. From to top menu, click *Run->Debug Configurations* and double click on *C/C++ application* in the left window. A new debug configuration named *passthrough Default* will be made. Rename it to *passthrough remote*. The right window contains the tabs *Main, Arguments, Environment, Debugger, Source and Common*. We are going to make some changes to the settings in the *Main* and *Debugger* tabs.

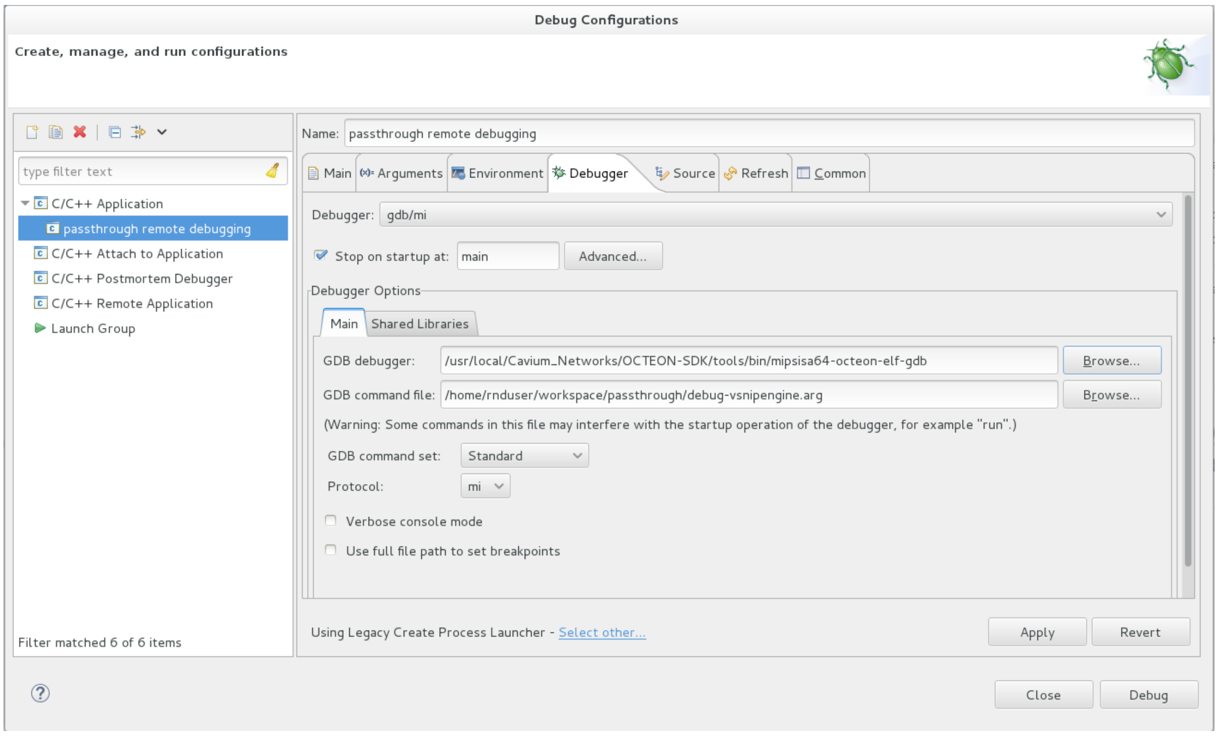
The *Main* tab:

- Fill in the C/C++ application by browsing to the passthrough executable in the passthrough project directory.
- At the bottom, click on *select other* to change the launcher. In the popup dialog, tick the *use configuration specific settings* selection box and select *Legacy Create Process Launcher*. Press *OK* to leave the Launcher selector.

The *Debugger* tab:

- To select the correct GDB debugger, browse to the OCTEON_ROOT/tools/bin directory and select mipsisa64-octeon-elf-gdb. Note: In SDK3.1 we found a bug in the GDB debugger. A patched version of mipsisa64-octeon-elf-gdb is available on our website.
- To select the GDB command file, browse to the passthrough project directory and select the file we created a few minutes ago, debug-vsnipengine.arg.
- This sections is about graphical debugging but if you also want to use the console input of the Eclipse to enter a command manually now and then, tick the Verbose console mode selection box.

Your Debugger tab of the Debug Configuration box should look something like figure 3



Name: passthrough remote debugging

Debugger: gdb/mi

[✓] Stop on startup at: main

GDB debugger: /usr/local/Cavium_Networks/OCTEON-SDK/tools/bin/mipsisa64-octeon-elf-gdb

GDB command file: /home/rnduser/workspace/passthrough/debug-vsnipengine.arg

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

GDB command set: Standard

Protocol: mi

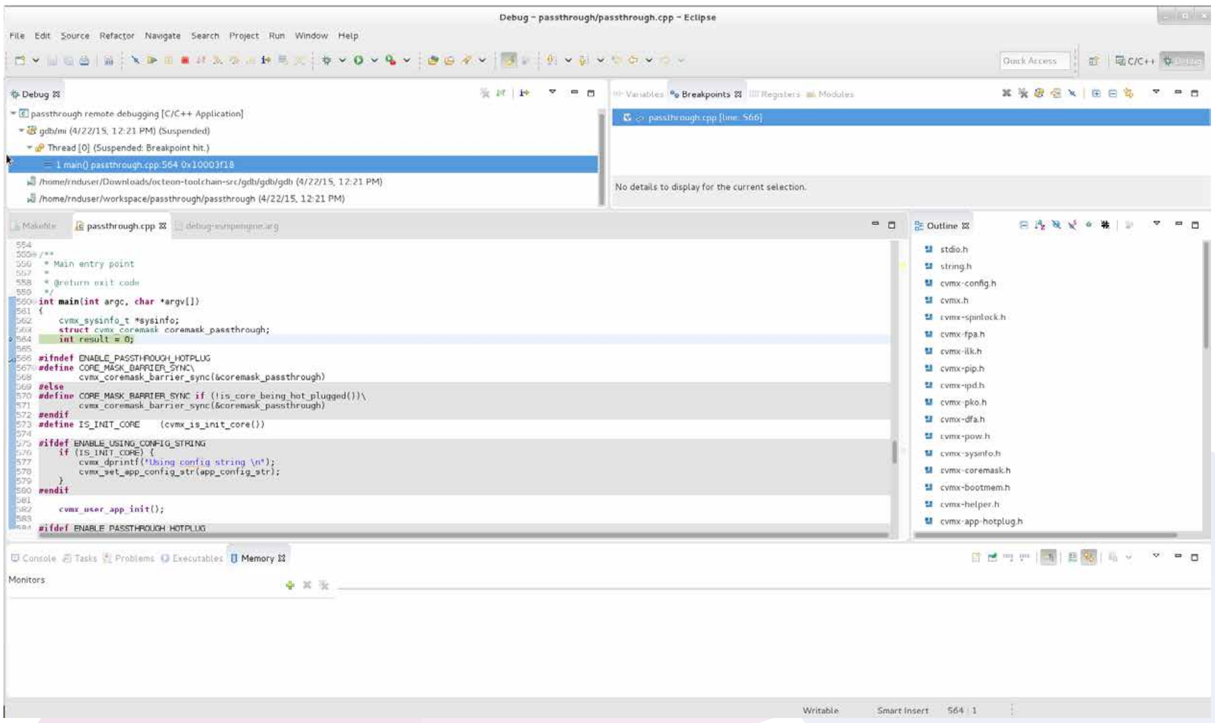
[ ] Verbose console mode

[ ] Use full file path to set breakpoints

Using Legacy Create Process Launcher - Select other...

Figure 3: Debugger tab settings

That's it! Click the *Debug* button on the lower right side to start debugging process. After a short while you should see an Eclipse workspace as depicted in figure 4.



```
passthrough remote debugging [C/C++ Application]
  gdb/mi (4/22/15, 12:21 PM) (Suspended)
    Thread [0] (Suspended: Breakpoint hit.)
      1 main() passthrough.cpp:564 0x10003f18
  /home/rnduser/Downloads/octeon-toolchain-src/gdb/gdb/gdb (4/22/15, 12:21 PM)
  /home/rnduser/workspace/passthrough/passthrough (4/22/15, 12:21 PM)
```

```
passthrough.cpp [line: 566]
```

No details to display for the current selection.

```
/**
 * Main entry point
 *
 * @return exit code
 */
int main(int argc, char *argv[])
{
    cvmx_sysinfo_t *sysinfo;
    struct cvmx_coremask coremask_passthrough;
    int result = 0;

#ifndef ENABLE_PASSTHROUGH_HOTPLUG
#define CORE_MASK_BARRIER_SYNC\
        cvmx_coremask_barrier_sync(&coremask_passthrough)
#else
#define CORE_MASK_BARRIER_SYNC if (!is_core_being_hot_plugged())\
        cvmx_coremask_barrier_sync(&coremask_passthrough)
#endif
#define IS_INIT_CORE    (cvmx_is_init_core())

#ifdef ENABLE_USING_CONFIG_STRING
    if (IS_INIT_CORE) {
        cvmx_dprintf("Using config string \n");
        cvmx_set_app_config_str(app_config_str);
    }
#endif

    cvmx_user_app_init();

#ifdef ENABLE_PASSTHROUGH_HOTPLUG
```

```
stdio.h
string.h
cvmx-config.h
cvmx.h
cvmx-spinlock.h
cvmx-fpa.h
cvmx-ilk.h
cvmx-pip.h
cvmx-ipd.h
cvmx-pko.h
cvmx-dfa.h
cvmx-pow.h
cvmx-sysinfo.h
cvmx-coremask.h
cvmx-bootmem.h
cvmx-helper.h
cvmx-app-hotplug.h
```

Figure 4: Finally, graphical remote debugging of a Simple Executive using Eclipse on an Octeon processor

Now you can single step, set breakpoints, examine variables etcetera from the Eclipse environment. Nice, isn't it? Notice that the layout of Eclipse has changed to the debug perspective. To change it back to the developers perspective, click on the >> symbol at the top right side and select *C/C++*.

Using the simulator

The advantages of using the simulator are numerous. You don't need real hardware to test your program, you can profile the performance via Viewzilla and use the Cavium *oct-packet-io* utility to get packets in and out of the simulator. See the SDK documentation on the simulator and the Fundamentals Volume 1 on the Cavium support site for information about the simulator and debugging via the simulator.

To debug via the simulator and Eclipse, generate a debugger command file named *debug-simulator.arg* in the passthrough project directory that contains the following lines

```
set spawn-sim on
file passthrough
target octeon tcp::2021 –quiet –noperf –serve=1010
b __cvmx_interrupt_default_exception_handler
```

Notice that the first line enables the start of the simulator when the debugger is started. The third line connects the debugger to the simulator on port 2021 and sets several options which are passed to the simulator. The simulator will listen to TCP port 1010 for connections for packet I/O. For example to route all packets from your eth1 port on your Linux host to and from the simulator, type

```
oct-packet-io –p 1010 –b 0:eth1
```

In this way, the simulator appears as a node on your network.

To update the Eclipse environment, create a new debug configuration exactly the same way as we did for debugging real hardware. Rename the new debug configuration to *passthrough simulator* and select *debug-simulator.arg* as the GDB command file. When you click the Debug button, the simulator should start automatically in a separate window and you can start debugging the passthrough example from Eclipse.

Remote debugging of User Mode applications

As a last example, we will show you how to remotely debug user mode applications. User mode applications are applications that run under Linux on the Cavium. For this, we will use tooling that can be downloaded for free! So if you want to play around with the Octeon processor but are not yet willing to buy the SDK, this is a good starting point. Or maybe this free kit is all you need, the choice is up to you. Using these free tools, you can use the Octeon as a 16 core Linux system. Cavium will also provide some free API's to get access to the potential of the Octeon.

First, we download the free Octeon tool chain from www.cnusers.org. Create an account if you not already have one, login and go to the download section. Click on the section for registered members and select the OCTEON cnusers-Software Development Kit. Select the OCTEON cnusers-SDK 3.1. Download *the cnusers SDK v3.1.0* and open it with the Archive Manager.

Install the toolkit in a directory so it will not interfere with your current SDK. We chose */usr/local/bin*. To make sure that the newly installed tool chain is used instead of the official Cavium SDK, add the following line at the end of your startup script (in our case .bash_profile)

```
PATH=/usr/local/bin/OCTEON-SDK/tools/bin:$PATH
```

To let these changes take effect, logout and login again. After logging in, verify your search path by typing

```
echo $PATH
```

It should show something similar like: /usr/local/bin/OCTEON-SDK/tools/bin:……

Next we will create a new C++ project in Eclipse. Enter *File->New->Project*, click on *C/C++ Project* and *select C++ Project*. In the dialog box that pops up, select Executable project type and *Hello World C++ Project*. For the tool chain, select *Linux GCC*. As a project name choose *HelloWorld*. Click *Finish*.

Go to *Project->Properties* and expand the C/C++ Build items by clicking on the triangle in front of it. Go to the Settings item and select the tool settings for the GCC C++ compiler. Change the command from g++ to *mips64-octeon-linux-gnu-g++*. Select *Miscellaneous* under *GCC C++ Compiler* and add the option *–mabi=64* at the *Other flags* section. Repeat this for the *GCC C Compiler settings* but instead of the changing the command to a C++ compiler, change it to *mips64-octeon-linux-gnu-gcc*.

Finally, select the tool settings for the GCC C++ Linker and change the command to *mips64-octeon-linux-gnu-g++*.

Click *OK*.

Do a clean rebuild: *Project->Clean*.

At this point, we assume you have booted Linux on the VSN IP Engine and it has been assigned a valid IP address. In our example it has 1.6.193.225. Also, you have opened a terminal session to the VSN IP Engine using either a serial link or SSH. Using a terminal, copy the executable to the VSN IP Engine with the following command

```
scp Debug/HelloWorld 10.6.193.225:/
```

On the VSN IP Engine, start gdbserver by typing the following in your session window:

```
gdbserver :2021 HelloWorld
```

This will start gdbserver on the VSN IP Engine, debugging the HelloWorld example via a TCP connection on port 2021. Gdbserver should respond with a message that it is listening on port 2021.

In Eclipse, create a new debug configuration by clicking *Run->Debug Configuration*. A new dialog box will open, containing the tabs *Main, Arguments, Environment, Debugger, Source, Refresh, Common*. We will have to make changes in the *Main* and *Debugger* tabs.

Main tab: Double click on *C/C++ Application*. At the bottom, click on *Select Other* to change the process launcher to *Standard Create Process Launcher.*

Debugger tab:
- Change Debugger from gdb/mi to gdbserver
- *Main* tab: Change the GDB debugger to *mips-octeon-linux-gnu-gdb*
- *Main* tab: Clear the GDB command file box
- *Connection* tab: Change connection type to TCP, change the ip address to 10.6.193.225 and change the port number to 2021.

Press *Debug*. Your VSN IP Engine should respond with *Remote debugging from host x.x.x.x*. The Eclipse environment should change to the debug perspective. Happy debugging!

Powered by Phact

Sirius is a TCP/UDP/IP protocol stack and is Phact's solution of daily handling millions of TCP and SSL connections on OCTEON hardware. Sirius is available as source code or as a library build for your OCTEON platform. It runs as Simple Exec or on OCTEON-Linux.

Phact is a one stop shop for software and hardware solutions, support and consultancy and was founded in 2015 with 25 years of experience in the business.

How? Phact originated in VSN Systemen. VSN was founded in 1991. The last couple of years, the company grew and evolved tremendously. It was time to adjust the look and feel of the company to what it had become. That's why Phact was born.

We started developing telecom hardware and software for VoIP- and SS7-carriers in 1991. For them, our engineers developed stable and professional hardware and software solutions. Additionally, we provided our customers with support and management of their solutions.

Knowledge is a key part of our business. Therefore, we are engaged in close partnerships with universities and our professional suppliers.

Phact might still be very young, but we carry years of experience in software and hardware development, support, management and consulting.

phact

Phact BV
Keizersveld 83, 5803AP, Venray, The Netherlands
T +31 (0) 478555000 F +31 (0) 478589563
Email: info@phact.nl
www.phact.nl
www.sirius-networks.com