

Using OpenOCD and Source Level Debug on Intel[®] Quark SoC X1000

Application Note

November 2013



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

Any software source code reprinted in this document is furnished for informational purposes only and may only be used or copied and no license, express or implied, by estoppel or otherwise, to any of the reprinted source code is granted by this document.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. ("products") in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped. They are never to be used as "commercial" names for products. Also, they are not intended to function as trademarks.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2013, Intel Corporation. All rights reserved.



Contents

1	Introduction	4
	1.1 Terminology	4
2	Prerequisites	5
3	Debugging	6
	3.1 OpenOCD	6
	3.2 GDB	7
	3.3 Eclipse	8

Tables

Table 1.	Terminology	4
----------	-------------------	---

Revision History

Date	Revision	Description
November 2013	0.8.0	Updated text to replace code name with official product name.
July 2013	0.6	Initial release.



1 Introduction

This document explains briefly how to use OpenOCD with Eclipse* or GDB for source level debugging of the Intel® Quark SoC X1000.

You may see references in the code to product codenames:

- Intel® Quark SoC X1000 (formerly codenamed Clanton)
- Intel® Quark Core (formerly codenamed Lakemont Core)

Note: This document is not a complete guide to source level debugging. Its purpose is to enable you to begin debugging the Linux* kernel on the Intel® Quark SoC X1000 at source level using OpenOCD with GDB or Eclipse.

For a complete set of supporting documentation, please visit the website for your specific JTAG hardware. The board has been tested with the following:

- Olimex* ARM-USB-OCD-H
<https://www.olimex.com/Products/ARM/JTAG/ARM-USB-OCD-H/>
- TinCanTools* FLYSWATTER2
http://www.tincantools.com/wiki/Compiling_OpenOCD

1.1 Terminology

Table 1. Terminology

Term	Description
Eclipse	An integrated development environment (IDE) comprising a base workspace and an extensible plug-in system for customizing the environment.
GDB	GNU* Debugger is the standard debugger for the GNU operating system.
JTAG	Joint Test Action Group (JTAG) is the common name for the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. Debuggers communicate on chips with JTAG to perform operations like single stepping and breakpointing.
OpenOCD	Free and Open On-Chip Debugger.
vmlinux	A statically linked executable file that contains the Linux kernel in one of the object file formats supported by Linux (such as ELF, COFF and a.out).



2 Prerequisites

Please refer to the OpenOCD section of the Intel® Quark SoC X1000 Board Support Package (BSP) Build Guide and complete the instructions before attempting the steps outlined in this document.

Required software:

- Linux* host system (running Eclipse/GDB/OpenOCD)
- Quark-patched OpenOCD
- GDB
- Eclipse (Juno tested) with CDT Plugin Installed (Main + Optional Features)
- Quark Kernel compiled with debug symbols

Required hardware:

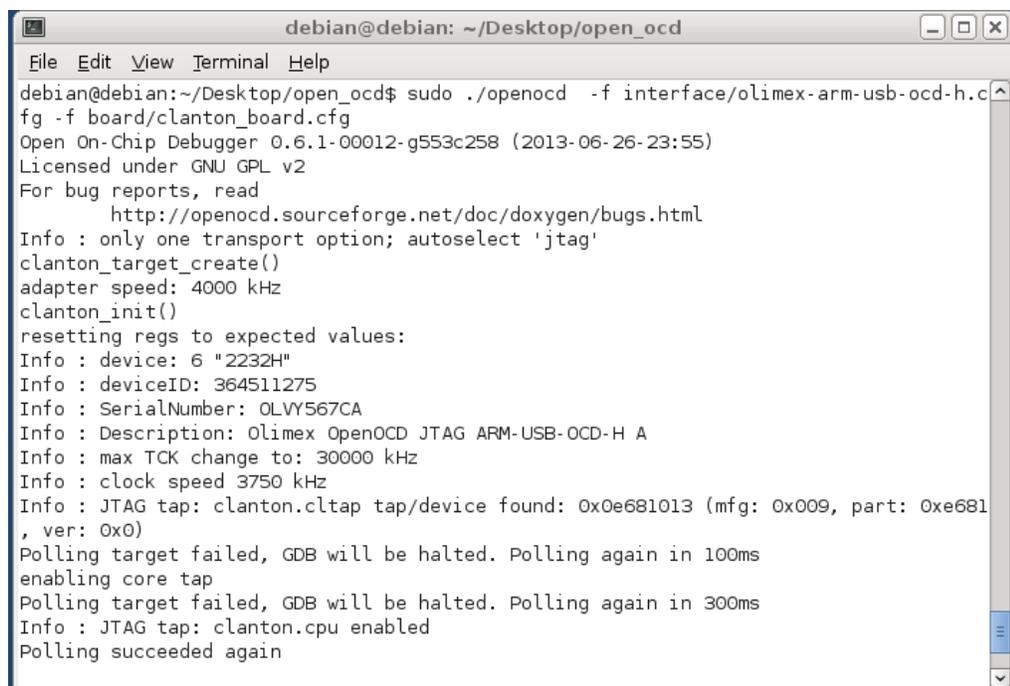
- OpenOCD supported JTAG debugger.
For example:
 - Olimex* ARM-USB-OCD-H
 - TinCanTools* FLYSWATTER2

3 Debugging

3.1 OpenOCD

The first step to enable source level debug is to connect your JTAG debugger to the board and run OpenOCD with the correct interface configuration file for your JTAG debugger. The example below uses an "olimex-arm-usb-ocd-h" JTAG debugger.

```
sudo ./openocd -f interface/olimex-arm-usb-ocd-h.cfg -f board/clanton_board.cfg
```



```
debian@debian: ~/Desktop/open_ocd
File Edit View Terminal Help
debian@debian:~/Desktop/open_ocd$ sudo ./openocd -f interface/olimex-arm-usb-ocd-h.c
fg -f board/clanton_board.cfg
Open On-Chip Debugger 0.6.1-00012-g553c258 (2013-06-26-23:55)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
clanton_target_create()
adapter speed: 4000 kHz
clanton_init()
resetting regs to expected values:
Info : device: 6 "2232H"
Info : deviceID: 364511275
Info : SerialNumber: OLVY567CA
Info : Description: Olimex OpenOCD JTAG ARM-USB-OCD-H A
Info : max TCK change to: 30000 kHz
Info : clock speed 3750 kHz
Info : JTAG tap: clanton.cltap tap/device found: 0x0e681013 (mfg: 0x009, part: 0xe681
, ver: 0x0)
Polling target failed, GDB will be halted. Polling again in 100ms
enabling core tap
Polling target failed, GDB will be halted. Polling again in 300ms
Info : JTAG tap: clanton.cpu enabled
Polling succeeded again
```

It is possible to use OpenOCD as a standalone tool for basic debugging. You can connect to the OpenOCD session using telnet and issue commands (this step is not required for source level debug). This can be seen in the following screenshot.



```
debian@debian: ~/Desktop/open_ocd
File Edit View Terminal Help
debian@debian:~/Desktop/open_ocd$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> halt
halted at 0xc100945c in protected mode
target state: halted
target halted due to debug-request, EIP=c100945c
> reg
===== lakemont registers
(0) EAX (/32): 0xC140F3C0
(1) ECX (/32): 0x00000000
(2) EDX (/32): 0x00000004
(3) EBX (/32): 0x00000002
(4) ESP (/32): 0xC1405F98
(5) EBP (/32): 0xC1405F98
```

3.2 GDB

It is possible to perform source level debug using GDB by connecting to OpenOCD's internal GDB server. OpenOCD must be running as shown in the previous section.

Run GDB pointing to a debug symbol compiled Quark Kernel vmlinux file:

```
gdb /path/to/vmlinux
```

Connect to the OpenOCD internal GDB server and halt the board:

```
(gdb) target remote localhost:3333
```

```
monitor halt
```

```
continue
```

```
ctrl + c
```

The screenshot below shows these steps in operation. After they are completed, the board is ready to be source level debugged using GDB.



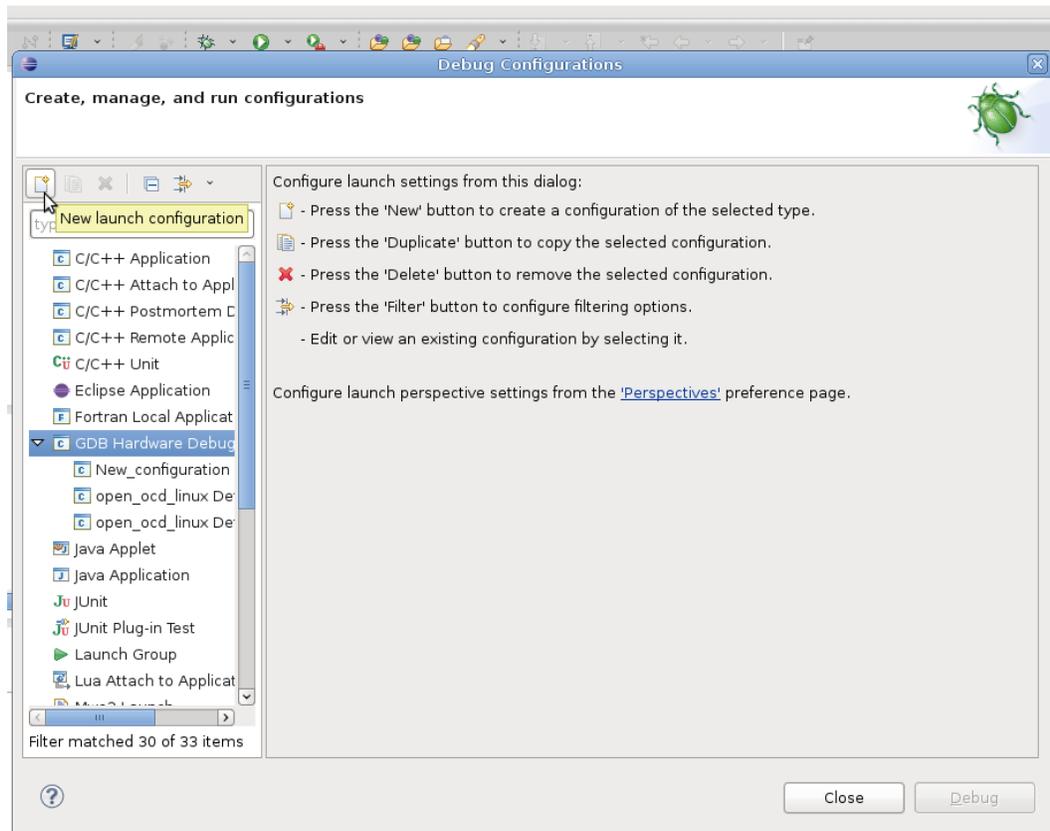
```
debian@debian: ~/Desktop/linux-yocto-dev
File Edit View Terminal Help
debian@debian:~/Desktop/linux-yocto-dev$ gdb vmlinux
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/debian/Desktop/linux-yocto-dev/vmlinux...done.
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
0x00000000 in ?? ()
(gdb) monitor halt
^[[Ahalted at 0xc100945c in protected mode
target state: halted
target halted due to debug-request, EIP=c100945c
(gdb) continue
Continuing.
target running
^Chalted at 0xc100945c in protected mode

Program received signal SIGINT, Interrupt.
default_idle () at arch/x86/kernel/process.c:391
391          current_thread_info()->status |= TS_POLLING;
(gdb) █
```

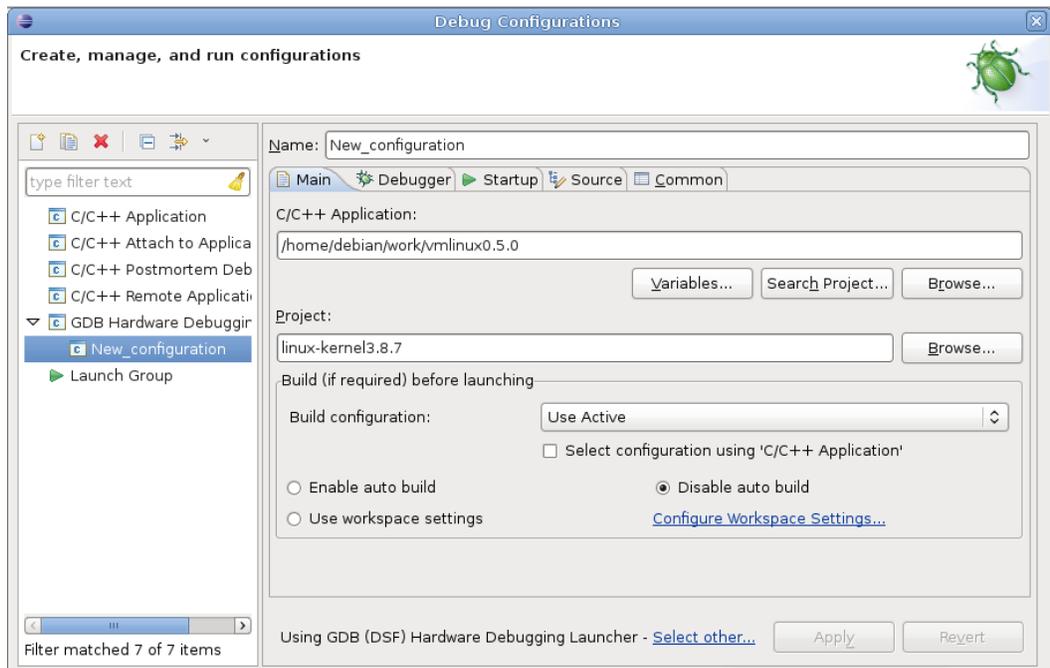
3.3 Eclipse

It is also possible to perform source level debug using Eclipse with the CDT GDB Hardware Debugger plug-in. The following configuration is required to enable source level debugging of the board in the Eclipse environment.

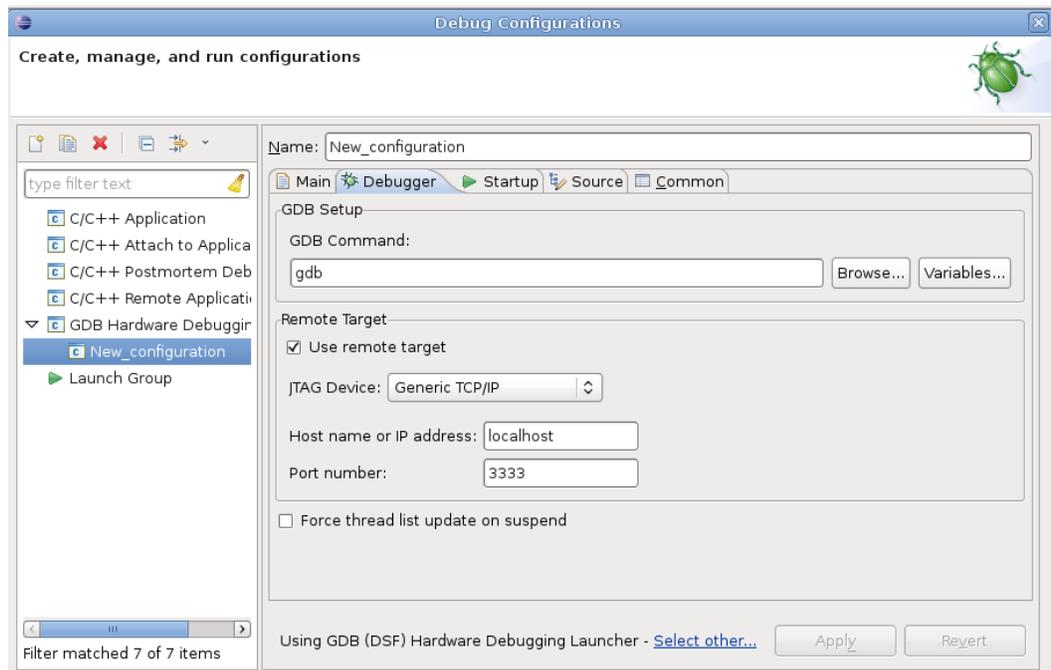
Go to the debug configurations menu, and add a new launch configuration under GDB hardware debugging, as shown below.



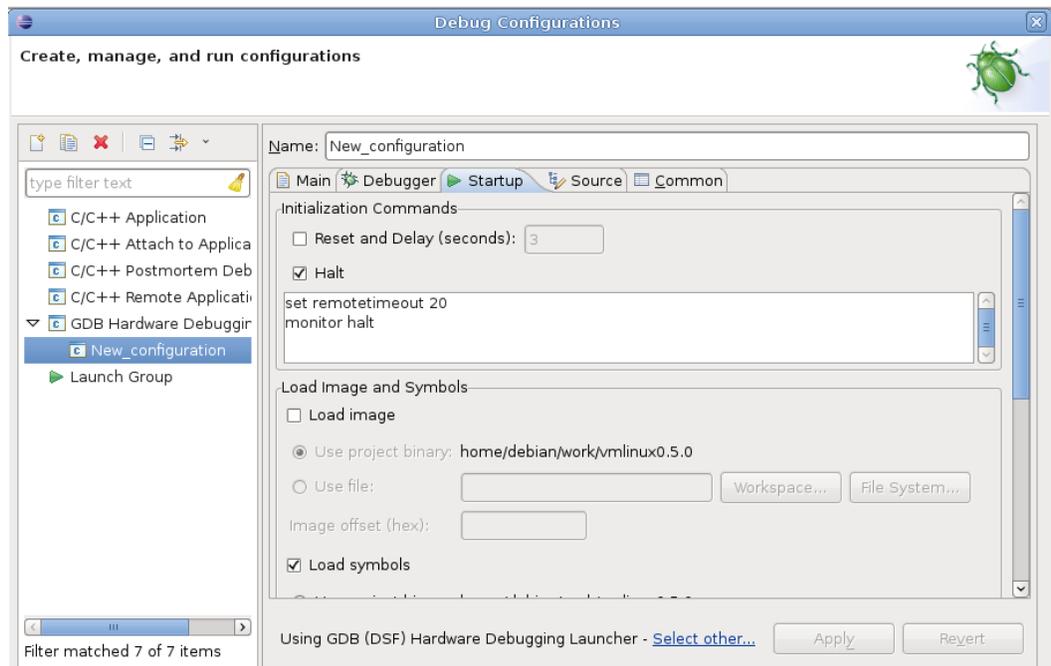
Set application to the debug symbol enabled vmlinux kernel file.



Enable **Use remote target** and set the host name and port number.



Select **Halt** and add the commands: **set remotetimeout 20** and **monitor halt**.



Eclipse is now set up to perform source level debug on the board as shown below.



The screenshot displays the Eclipse IDE interface for debugging a kernel process. The main window shows the source code for `process.c` with the following code snippet:

```

else
    local_irq_enable();
    current_thread_info()->status |= TS_POLLING;
    trace_power_and_rcuidle(smp_processor_id());
    trace_cpu_idle_rcuidle(PWR_EVENT_EXIT, smp_processor_id());
}
#ifdef CONFIG_APM_MODULE
EXPORT_SYMBOL(default_idle);
#endif

bool set_pm_idle_to_default(void)
{
    bool ret = !pm_idle;
    pm_idle = default_idle;
    return ret;
}

void stop_this_cpu(void *dummy)

```

The Disassembly window shows the following assembly code:

```

c100945c: mov %esp,%eax
c100945e: and $0xffff000,%eax
391     current_thread_info()->status |= TS_POLLING;
c1009463: orl $0x4,0xc(%eax)
394     }
c1009467: pop %ebp
c1009468: ret
300     local_irq_enable();
c1009469: call 0xc1062ae0 <trace_hardirqs_on>
c100946f: jmp 0xc100945c <default_idle+28>
c1009471: jmp 0xc1009480 <end_e400_idle>
c1009473: nop
c1009474: nop
c1009475: nop
c1009476: nop
c1009477: nop
c1009478: nop

```

The Memory monitor window shows the following data:

Address	0 - 3	4 - 7	8 - B	C - F
C1009450	08A80875	14E8895A	05009B4	88E02500
C1009460	E0FFFF83	4800A45D	C3E97294	0500FBEB
C1009470	EED00090	90909090	90909090	90909090
C1009480	5086E553	83EC0C89	E02500E0	FFFF8840
C1009490	08A80875	17803078	67A8C100	75528955

S