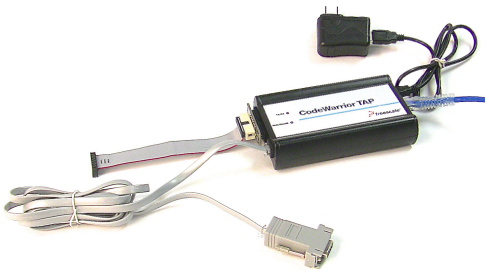
Document Number: GDBBORGS
Rev. 0, 01/2013

GDB for Board Bring-Up Getting Started Guide

If you own or are thinking of purchasing a CodeWarrior TAP (http://freescale.com/CWTAP), then this program "GDB Proxy" might be really useful for you.

What the GDB Proxy program does is allow the GDB debugger to communicate with a Freescale target board to perform board bringup activities. This is different from normal GDB debugging, because normally GDB is for debugging applications in a running operating system. In this example, a working operating system is not required.



Since many people are very familiar with GDB, this is why Freescale has added the ability to have GDB as a debugger for board bring-up. This is done by having GDB talk to the CodeWarrior TAP via a program called GDB Proxy.

IMPORTANT NOTIFICATION REGARDING THE PROGAM (Winter 2012)

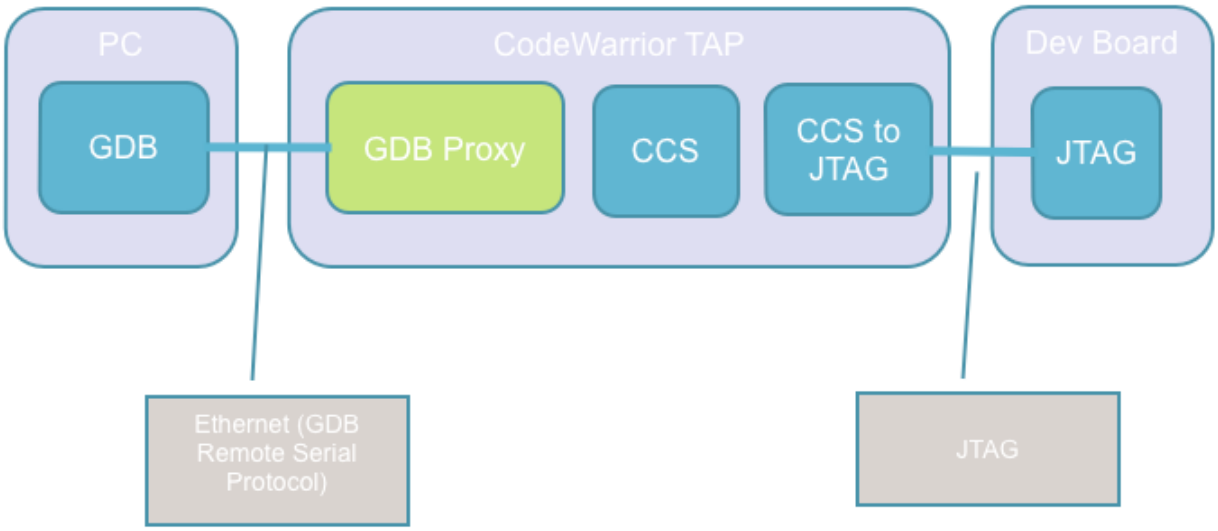
The program you are reading about here is created as a courtesy to our customers. This is based on open source software and the source is available upon request.

We are just now putting in place the ability to get the source freely on an external repository. Along with that we have to work on how to make it easy to build and deploy the program.

It is our sincere hope that you'll join with us and collaborate on the program. Of course we'll work hard to make the program as solid as we can, but rather than delay at all, we thought it would be better to provide the program ASAP to you, even in a BETA-ish form, and ERROR on the side of "availability" over "perfection".

Theory of Operation

The system works like this. Refer to the diagram to help visually see the flow.



- GDB debugger speaks to the CodeWarrior TAP via a protocol called GDB Remote Serial Protocol.
- GDB Remote Serial Protocol is served via a server program called GDB Proxy inside the CodeWarrior TAP.
- The GDB Proxy then takes GDB Remote Serial Protocol commands and then translates them to CCS commands.
- CCS is the native command server inside the CodeWarrior TAP that knows how to communicate with Freescale targets.
- The CCS commands are then sent to the target device.

- This communications process works in both directions with command results and status flowing from the Freescale target to the GDB debugger via the CCS server and the GDB Proxy server.

GDB and other useful basics

Before you get going, there are some basics which many of you might know, but just in case, we thought we would write it down to be helpful.

For the best GDB debugger experience possible, you need some descriptive information about the target. These are stored in XML files, and are what comprise of the the "Target Configurations". We get these from the CodeWarrior XML files that are already completed and then use an XML transformation script to convert into GDB format. As we do more of these, we'll make them available at http://freescale.com/CWTAP - downloads TAB.

Because the boards will not run through a typical 'boot' process when debugging, you'll also have some files, which initialize the target system so you can run your applications. These are typically files that contain GDB commands and are ran as a script from the GDB command prompt. These are also part of the "Target Configurations" that are supported. Again, as we add more, we will make them available. These files are usually specific to a target board, not just the processor.

You could additionally make more XML files to describe the registers that exist on your board, such as peripherals on auxiliary chips.

Host configurations being tested

We believe that the primary development platform for engineers wanting to use GDB for board bringup is Linux. These versions of Linux we have tested with our GDB debugger process for board bring-up. Our primary platform is RedHat but we have spent some time with Ubuntu.

It is likely, but not guaranteed, that what we describe in this paper will work on a different Linux. Please feel free to try it and let us know.

- RedHat version 2.6.18-238.1.1.el5
- Ubuntu version 3.2.0-24-generic

Target configurations currently supported

Below are the target configurations which we have tested our GDB Proxy debugging system as of Nov 2012. You can expect scripts for initializing the board as well as XML that describe the registers in the Freescale processor.

This does not mean that MORE configurations would not work, but it does mean we just have not verified for ourselves OR created the support files to make GDB debugging work OR have not updated this document. Again, the most current information is on http://freescale.com/cwtap.

- P1010 (e500v2)
- P4080 (e500mc)

Where to get a GCC

The GCC we are speaking about is a GCC compiler that will cross compile from a Linux workstation and create Power Architecture based programs. These programs run on the Freescale target system and not on the probe.

You can find GCC in the Freescale in the Freescale Software Development Kit (SDK) for Power Architecture Linux. It is located here (http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=SDKLINUX). You would select the downloads TAB and pick the download marked a "source ISO". This isn't the exact name since the file would have the version name in it as well.

To install and use the SDK, you'll need to follow the instructions in the SDK. Later on in this document we also have some information that will help with the SDK, but we still recommend learning about the SDK from that package and its information directly.

Commands to use the gcc cross compiler:

```
bitbake gcc-cross
```

For p4080 gcc will be in this dir :

<[base directory for your development]/QorIQ-SDK-V1.2-20120614-yocto/build_p4080ds_release/tmp/sysroots/i686-linux/usr/bin/ppce500mc-fsl-linux?

Where to get a GDB

This GDB will run on your HOST Linux system, not your target system. This GDB will communicate with the CodeWarrior TAP via GDB Proxy to perform the unique board bring-up activities.

Typically, you can get this GDB from Freescale in the Freescale Software Development Kit (SDK) for Power Architecture Linux mentioned earlier in this short document.

We made 1 or 2 small modifications to that SDK version 1.2 GDB program to make it work just a little bit better for board bring-up. One change in particular was to have GDB delay its reading of stack values later in its work list.

The procedure for getting GDB going is detailed here :

- GDB could be built with QorIQ SDK 1.2 (yocto). The two "source" commands here should be ran before building GCC or GDB.

```
source ./fsl-setup-poky -m p4080ds
or
source ./fsl-setup-poky -m p1010rdb
```

You can find GDB in the same directory as gcc in QorIQ SDK : <[base directory for your development]/QorIQ-SDK-V1.2-20120614-yocto/build_p4080ds_release/tmp/sysroots/i686-linux/usr/bin/ppce500mc-fsl-linux

- Then apply lr patch

```
bitbake gdb-cross -c clean
cp gdb-7.3-lr-fix.patch  <sdk 1.2 dir>/QorIQ-SDK-V1.2-20120614-yocto/./meta-fsl-ppc/recipes-devtools/gdb/files/gdb-7.3-lr-fix.patch
```

Add this patch in this file ./meta-fsl-ppc/recipes-devtools/gdb/gdb-fsl.inc

```
SRC_URI_append_fsl = " file://gdb-7.4.1-e500mc-disassemble-fix.patch \
            file://gdb-7.3-lr-fix.patch"
bitbake gdb-cross -c patch
bitbake gdb-cross
```

Where to get XML files with register information

- Freescale.com/CWTAP and then click on downloads TAB
- This will either be in a download that has lots of GDB technology OR we might put it in an XML/init file ONLY download specific to the chip and board.
- Just look for your Freescale processors model number and you should find what you need.

Where to get latest CCS

Freescale.com/CWTAP – Then select the "Downloads" TAB. We'll have the latest version available for download and clearly marked as the CCS program for downloading.

How to update CWTAP with latest CCS

The primary method to do this is to run the CCS program. CCS will then make contact with the probe and update the probe with the new program. By updating, the new program is placed into the CodeWarrior TAP's flash memory.

Connect CCS to CWTAP (config cc <cwtap name or ip>; show cc)

For more information on the CodeWarrior TAP, visit Freescale.com/CWTAP. You will find the users manual for download on the “Downloads” TAB.

How to build an elf file

There are numerous examples on how to do this. There is nothing special about building an ELF file for loading into the target due to GDB debugging with the CodeWarrior TAP. The notes here are simply to help point you in the correct direction.

- An example for p1010 and p4080 is provided in the GDB download package found on Freescale.com/CWTAP – on the downloads tab.
- To run the downloadable examples, users will need gcc, and lcf lib files. The heritage of the files is from the CodeWarrior Development Studio.

How to launch the GDB and load the ELF File

This is the key information on how to perform your first GDB bare board debugging session with our example files. We tend to use the “ddd” debugger because we think it makes life a little bit easier.

- Run gdb commands from p1010_demo.gdb or p4080_demo.gdb to load elf file, set breakpoint at main() etc.

Troubleshooting

- The probe reboots very fast – the most effective way to restart GDB Proxy and CCS is to simply reboot the probe by cycling power.
- Pay particular attend to whether or not the “target remote” gdb command is successful. If it is, then gdb is connected to the GDB Proxy program (gdbproxy) and the monitor commands should now be available.
- get_probe_status monitor command prints the status. Here is an example status. All of the lines should indicate a proper connection.

```
(gdb) mon get_probe_status
Connected to gdbserver FSL021352:2345
Connected to CCS fsl021352.am.freescale.net:41475
Connected to probe fpga
Connected to target p1010
```

How to install sdk1.2 and build gcc-cross and gdb-cross

Download QorIQ-SDK-V1.2-SOURCE-20120614-yocto.iso from
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=SDKLINUX

As root

```
$ mount -o loop QorIQ-SDK-V1.2-SOURCE-20120614-yocto.iso
<tmp dir>
$ exit
$ cd <tmp dir>
$ ./install
```

Give path as requested, for example /opt/fsl/sdk12

As root

```
$ umount <tmp dir>
$ exit
```

Binary images (for example SDK V1.2 e500mc Binary ISO) could be installed to the same directory (for example /opt/fsl/sdk12) to save time when building toolchain. Binary images are installed the same way.

```
$ cd /opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto
$ ./scripts/host-prepare.sh
```

Add "root ALL" line if you get this message:

```
--------------
"To configure this, as root using the command
"/usr/sbin/visudo",
and add the following line in the User privilege section:"
root ALL = NOPASSWD: /usr/bin/yum
----------------
```

Python 2.6 is required for sdk 1.2.

Install missing packages if needed and run ./scripts/host-prepare.sh again. Keep doing this till you get the message

```
"Nothing to do Done. You're ready to go with Yocto build
now"
```

Set up the environment for your target, for example for p4080ds

```
$ source ./fsl-setup-poky -m p4080ds -j 8 -t 8
```

Please check your host info (cat /proc/cpuinfo) to specify number of jobs and threads.

If the command above passes, you get a message

```
Configuring for p4080ds board type
```

Run the following commands to start a build:

```
bitbake fsl-image-minimal
  bitbake fsl-image-lsb-sdk
  bitbake fsl-image-kvm
  bitbake fsl-image-core
  bitbake fsl-image-flash
  bitbake fsl-image-full
```

To return to this build environment later please run:

```
source /opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto/fsl-
setup-poky -m p4080ds
```

Otherwise sdk 1.2 will report which other dependencies have to be installed. In most cases the required packages have to be installed with root access.

After the "source ./fsl-setup-poky" command the current directory becomes build_<target>_release, for example /opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto/build_p4080ds_release.

Now you can build gcc-cross

```
$ bitbake gcc-cross
```

On 64bit linux machine gcc-cross for e500mc (powerpc-fsl-linux-gcc) would be in this directory:

/opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto/build_p4080ds_release/tmp/sysroots/x86_64-linux/usr/bin/ppce500mc-fsl-linux

With sdk 1.2 the user has to install gdb-7.3-lr-fix.patch patch for gdb. This patch will be available in future sdk releases.

From /opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto/build_p4080ds_release

```
$ bitbake gdb-cross -c clean
```

Copy gdb-7.3-lr-fix.patch patch (available in example tarball) to recipes directory

```
$ cp <path to patch>/gdb-7.3-lr-fix.patch
/opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto/meta-fsl-
ppc/recipes-devtools/gdb/files/gdb-7.3-lr-fix.patch
```

Edit /opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto/meta-fsl-ppc/recipes-devtools/gdb/gdb-fsl.inc and add "file://gdb-7.3-lr-fix.patch" to SRC_URI_append_fsl variable :

```
SRC_URI_append_fsl = " file://gdb-7.4.1-e500mc-disassemble-
fix.patch \
                       file://gdb-7.3-lr-fix.patch"
```

Patch gdb

```
$ bitbake gdb-cross -c patch
```

Build gdb

```
$ bitbake gdb-cross
```

gdb executable will be in the same directory as cross gcc: /opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-yocto/build_p4080ds_release/tmp/sysroots/x86_64-linux/usr/bin/ppce500mc-fsl-linux

Debugging with GDB

Download the example

```
$ cd <example_dir>/gcc_build_elf/p4080
```

Change GCC_PATH in Makefile to point to gcc directory, for example

GCC_PATH = /opt/sdk12/QorIQ-SDK-V1.2-20120614-yocto/build_p4080ds_release/tmp/sysroots/i686-linux/usr/bin/ppce500mc-fsl-linux

```
$ make clean
$ make
```

Check if the p4080-core0.elf has been built.

```
$ cd ../../bin
```

Write correct CWTAP IP addess in p4080_init.gdb.

"cp4080_1.am.freescale.net" has to be replaced in three places.

Port 2345 is a default for for gdbproxy.

You can run ddd with gcc-cross, for example

```
ddd --debugger /opt/fsl/sdk12/QorIQ-SDK-V1.2-20120614-
yocto/build_p4080ds_release/tmp/sysroots/x86_64-
linux/usr/bin/ppce500mc-fsl-linux/powerpc-fsl-linux-gdb &
```

Enter GDB commands from p4080_demo.gdb at GDB prompt

```
(gdb) source p4080_init.gdb
(gdb) directory ../gcc_build_elf/p4080/Source
(gdb) file ../gcc_build_elf/p4080/p4080-core0.elf
(gdb) load
(gdb) b main
(gdb) c
(gdb) s
(gdb) s
(gdb) n
(gdb) delete
(gdb) file
(gdb) kill
```

Example of GDB log

```
(gdb) source p4080_init.gdb
Loading basic powerpc register info
Connecting to proxy
0x00000000 in ?? ()
Kill the program being debugged? (y or n) [answered Y; input
not from terminal]
Loading e500mc register info
Delete all memory regions? (y or n) [answered Y; input not
from terminal]
Connecting to proxy
0x00000000 in ?? ()
p4080
Connecting to probe and target
Connected to CCS
Connected to probe fpga
Connected to target p4080
p4080
24902
Reset
```

```
Read registers
Print ccsrbar
$1 = 0xfe000000
Create ccsr memory region
Delete all memory regions? (y or n) [answered Y; input not
from terminal]
Using user-defined memory regions.
Num Enb Low Addr   High Addr  Attrs
2   y   0x00000700 0x00000704 rw nocache
1   y   0xfe000000 0xffffffff rw nocache
Set SAP memory access
sap
Set RCPM_CTBHLTCR
BRRL - enable all cores
Creating TLB entry for CCSR space
Creating TLB entry for LB
Creating TLB entry for PEX1/2
Creating TLB entry for PEX3
Creating TLB entry for PEX3
Creating TLB entry for PEX I/0
Creating TLB entry for PEX I/0
Creating TLB entry for DDR 0x00000000 - 0x3FFFFFFF
Creating TLB entry for DDR 0x40000000 - 0x7FFFFFFF
Creating TLB entry for BMAN
Creating TLB entry for QMAN
Creating TLB entry for QMAN
Creating TLB entry for BMAN
Initializing interrupt vectors
Writing branch at program exception
Creating Local Access Windows
DDR1 Controller Setup
Wait for DRAM data initialization
LBC Controller Setup
Serial RapidIO - enable timeouts
Setting ijam memory access
ijam
Delete all memory regions? (y or n) [answered Y; input not
from terminal]
Initialization complete
(gdb) directory ../gcc_build_elf/p4080/Source
Source directories searched:
/home/testfarm/br/gdb_version_4/bin/../gcc_build_elf/p4080/S
ource:$cdir:$cwd
(gdb) file ../gcc_build_elf/p4080/p4080-core0.elf
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
```

```
Reading symbols from
/home/testfarm/br/gdb_version_4/gcc_build_elf/p4080/p4080-
core0.elf...done.
(gdb) load
Loading section .intvec, size 0x15dc lma 0x0
Loading section .text, size 0x71964 lma 0x100000
Loading section __libc_freeres_fn, size 0x1084 lma 0x171970
Loading section .rodata, size 0x17cbc lma 0x1729f8
Loading section __libc_subfreeres, size 0x2c lma 0x18a6b4
Loading section __libc_atexit, size 0x4 lma 0x18a6e0
Loading section .data, size 0x6d0 lma 0x18a6e8
Loading section .tdata, size 0x10 lma 0x18adb8
Loading section .eh_frame, size 0x3174 lma 0x18adc8
Loading section .gcc_except_table, size 0x1c3 lma 0x18df3c
Loading section .got2, size 0x44 lma 0x18e0ff
Loading section .dtors, size 0x4 lma 0x18e144
Loading section .got, size 0x10 lma 0x18e148
Loading section .sdata, size 0x98 lma 0x18e158
Start address 0x100064, load size 587703
Transfer rate: 59 KB/sec, 12504 bytes/write.
(gdb) b main
Breakpoint 1 at 0x1001d8: file Source/main.c, line 72.
(gdb) c
Continuing.

Program received signal SIGINT, Interrupt.
main () at Source/main.c:72
72      int i =0;
(gdb) s
s74      FibonacciRecursion();
(gdb) s

Program received signal SIGINT, Interrupt.
FibonacciRecursion () at Source/main.c:265
265         int i = 0,n = 0;
(gdb) n
270         for(i=0;i<kMaxFibonacciForLoopExecutions;i++)
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) file
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
No executable file now.
```

```
Discard symbol table from
`/home/testfarm/br/gdb_version_4/gcc_build_elf/p4080/p4080-
core0.elf'? (y or n) y
No symbol file now.
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
The program is not being run.
```

5 Revision History

Table 12. summarizes revisions to this document since the release of the previous version.

Table 12. Revision History

Location	Revision
Initial public release	0

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: GDBBORGS
Rev. 0
01/2013