ASHLING
EMBEDDED TECHNOLOGY AND INNOVATION

Ashling Application Note APB198

Using the Ashling Opella-XD Debug Probe for ARM™ with the GNU GDB Debugger

Table of Contents

1. Introduction

Ashling's Opella-XD for ARM Debug Probe as shown in Figure 1 is a powerful JTAG Debug Probe for embedded development with ARM cores with a high-speed USB 2.0 connection to the host PC.

Figure 1. The Ashling *Opella-XD for ARM*™ Debug Probe

The Ashling GDB Server allows you to use Opella-XD to connect to your embedded target and debug using the Free Software Foundation's (FSF) GNU GCC GDB Debugger (see the FSF GCC home page at **http://www.gnu.org/software/gcc/gcc.html**).

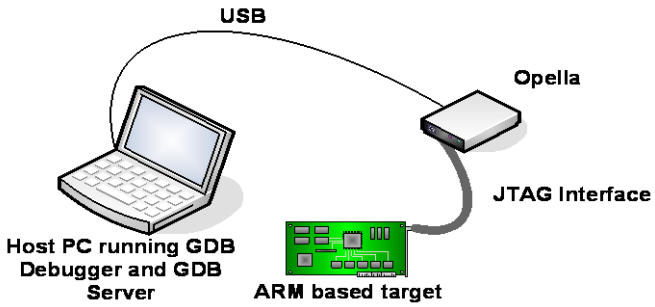

Figure 2. Using the Ashling GDB Server

The Ashling GDB Server is a console type application hosted on either Windows™ or GNU/Linux x86 based PCs. It takes requests from the GDB Debugger and sends them directly to the Opella-XD Debug Probe. In the above figure, the GDB Debugger and Server are shown running on the same PC, however, this is not a requirement and the GDB Server can reside on any machine that has a network connection to the machine running the GDB Debugger (as the Server communicates to the Debugger using the TCP/IP protocol). The Ashling GDB Server supports any debugger that adheres to the GDB Remote Serial Protocol including:

- GDB version 5 or later
- Insight (graphical version of GDB)
- Eclipse CDT

1.1 Installing your System

The Ashling Opella-XD for ARM Development system consists of:

- Ashling Opella-XD Debug Probe with Ashling JTAG Target Probe Assembly (TPA) for connecting between the Opella-XD and your target system's JTAG connector. For the hardware connection to your ARM target, Ashling provide the **TPAOP-ARM20** 20-pin .1" JTAG probe cable, for connection to a 20-pin JTAG pin-strip on your target board.
- Standard USB cable for connecting between Opella-XD and your PC.
- Opella-XD for ARM Software on CD.

Installation and target connection requires the following steps:

1. Installing the Ashling Opella-XD for ARM software. Opella-XD software works under Windows™ and GNU/Linux x86 hosts.
2. Connecting the Opella-XD to your PC's USB port.
3. Connecting the Opella-XD to your target's JTAG connector (e.g. application hardware) using the appropriate Ashling Target Probe Assembly.
4. Configuring your Debugger to use the Opella-XD as it's target connection.

1.1.1 Connecting the Opella-XD to your PC and Target

Opella-XD is designed to connect to your PC via the USB Port and your target via the supplied TPA. Please note the following recommended target connection sequence:

1. Ensure your target is powered off.
2. Connect Opella-XD to your PC using the supplied USB cable and ensure Opella-XD's **Power** LED is on.
3. Windows users may get a **New USB hardware found** message from Windows and will be prompted to install the appropriate USB drivers. The Ashling Opella-XD drivers are supplied on your **Ashling Opella-XD for ARM Software** CD and installed in your installation directory. Direct the Windows **Hardware Installation Wizard** to your installation directory so that it can locate the necessary drivers and complete the installation. Windows only needs to perform this operation the first time you connect your Opella-XD to the PC. The Opella-XD USB driver is called `libusb0.sys`
4. Connect Opella-XD to your target's JTAG connector using the supplied TPA.
5. Power up your target.

Ashling strongly recommend you adhere to this sequence at all times to avoid damage to the Opella-XD or to your target. When disconnecting from your target ensure you:

1. Power off your target.

2. Power off Opella-XD (disconnect USB cable).
3. Disconnect Opella-XD from your target.

An un-powered Opella-XD should never be connected to a powered target.

1.1.2 Software installation (GDB server) under Windows

To install, run the SETUP.EXE program located on your CD. Follow the on screen instructions and the SETUP program will install the software in the directory of your choice. By default, the software is installed in the `C:\Program Files\Ashling\Opella-XDforARM` directory of your hard drive. To verify Opella-XD is installed and working correctly run the supplied diagnostic program `OPXDDIAG` (this is a console application and must be run from a Command prompt). To run all tests enter:

```
> opxddiag --diag 1
```

Running with out any parameters displays all available options

```
> opxddiag
```

Make sure Opella-XD is connected to your PC and that you have installed the USB drivers before running these tests.

1.1.3 Software installation (GDB server) under GNU/Linux

Installation consists of two steps: installing the USB driver and installing the GDB Server as follows.

1.1.3.1.1 USB driver installation

Opella-XD uses the `libusb` driver (http://libusb.sourceforge.net/). By default, the driver is stored in: `/usr/lib/libusb`. Check for this as follows:

```
$ ls /usr/lib/libusb* /usr/include/usb.h
```

If you see `/usr/include/usb.h` and `libusb-0.1.so.4.4.0` or higher then you don't need to install or update libusb and can skip to the next section.

1.1.3.1.1.1 Ubuntu/Debian

Install using the following command:

```
$ sudo apt-get install libusb-dev
```

1.1.3.1.1.2 Fedora or other distributions

Download the latest `libusb` from http://libusb.sourceforge.net/ and install as follows:

```
$ tar zxf libusb-0.1.12.tar.gz (use appropriate version number)

$ ./configure --prefix=/usr
$ make
$ make install
```

1.1.3.1.2 GDB Server installation

1. Ensure that Opella-XD is connected to the PC, connected to the target and that the target is powered
2. Expand the supplied `gdbserverarmvXYZ.tar.gz` (located on the root of the CD where XYZ is the actual version number) file in a directory of you choice, e.g.:
   ```
   $ cd /home/ashling
   $ tar -zxf gdbserverarmv100.tar.gz
   ```
3. This creates the following sub directories:
   ```
   ash-arm-gdb-server
   ```
4. Change to ash-arm-gdb-server directory
   ```
   $ cd ash-arm-gdb-server
   ```
5. To ensure the current `$USER` has access to the Opella-XD device we recommend using the Linux utility `udev` (requires kernel 2.6 or later).
6. Ensure `udev` is installed and running on your system by checking for the `udev` deamon process (`udevd`) eg:
   ```
   $ ps -aef | grep udev
   ```
7. Create a `udev` rules file to uniquely identify the Opella-XD device and set permissions as required by owner/ groups. An example `udev` file is supplied (`60-ashling.rules`) which identifies Opella-XD device (by Ashling's USB product ID and Vendor ID).

8. The rules file must then be copied into the rules directory (requires root permission) e.g.:
```
$ sudo cp ./60-ashling.rules /etc/udev/rules.d
```
9. Finally, you can now run ash-arm-gdb-server within the ash-arm-gdb-server directory, e.g.:
```
$. /ash-arm-gdb-server
```

To verify Opella-XD is installed and working correctly run the supplied diagnostic program `OPXDDIAG` (this is a console application and must be run from a Command prompt). To run all tests enter:

```
$ ./opxddiag --diag 1
```

Running with out any parameters displays all available options

```
$ ./opxddiag
```

Make sure Opella-XD is connected to your PC and that you have installed the USB drivers before running these tests.

2. Configuring the Ashling GDB Server

The server is supplied as a console type application (`ash-arm-gdb-server.exe` for Windows and `ash-arm-gdb-server` for GNU/Linux) and is configured via command line parameters. Supported parameters can be seen by running `ash-arm-gdb-server` without any parameters as follows:

```
C:\Program Files\Ashling\Opella-XDforARM>ash-arm-gdb-server.exe
Ashling GDB Server for ARM (ash-arm-gdb-server).
v1.0.0 13-Jun-2008, (c)Ashling Microsystems Ltd 2008.

ash-arm-gdb-server [options]

If no options are specified then, ash-arm-gdb-server will read options from
the command file ash-arm-gdb-server.ini if it exists.

Valid options are:

--help                          Display usage information.
--version                       Display program version.
--debug-stdout                  Display debug messages to standard output.
--debug-file <file>             Log debug messages to <file>.
--command-file <file>           Read command line options from <file>.
--gdb-port <port>               Listen on <port> for remote connections
                                from GDB. Default is port 2331.
--instance <number>             Select Opella-XD from multiple Opella-XD(s)
                                connected.
                                <number> is Opella-XD serial number.
--jtag-frequency <freq>         Specifies JTAG frequency.
                                <freq> for Opella-XD can be in range
                                from 1kHz to 100MHz (default 1MHz) or RTCK.
[snip]…
```

For further details on the supported parameters, please refer to the user manual.

Connecting GDB to the Ashling GDB Server
Once running and connected to the target, the Ashling GDB Server will listen on the specified port (default:2331) using the GDB Remote Serial Protocol (RSP) for incoming connections. To connect to the Server from GDB (or any other compliant debugger), you must specify:

1. The protocol to use.
2. The IP address of the machine running the GDB Server.
3. The port that GDB Server is listening for connections on.

This may be done from within GDB using the **target remote** command. This command tells GDB to use the RSP protocol, the command parameters specify the IP address and port as follows:

```
target remote <SERVER IP ADDRESS>:<PORT>
```

For example:
```
target remote 192.168.10.27:2331
```

tells GDB to connect using the RSP protocol to the GDB Server running at IP address 192.168.10.27 and using port 2331. If you are running GDB on the same machine as the server then this can be abbreviated to:

```
target remote :2331
```

3. Example Debug Session using the Ashling GDB Server

This section documents an example GDB debug session using the Opella-XD and an ARM7TDMI-S target board (Ashling LPC2000 EVBA7 board).

1. Ensure Opella-XD is connected to the PC and the target board and everything is powered up.

2. Run the GDB Server from a command line prompt as follows:
```
>ash-arm-gdb-server.exe --program-entry-point 0x40000000 --device ARM7TDMI-S --
target-reset 1 5 --jtag-frequency RTCK
```

   Where:

Switch	Action
`--program-entry-point 0x40000000`	specifies the initial PC value to be 0x4000-000
`--device ARM7TDMI-S`	specifies the target device type is ARM7TDMI-S
`-target-reset 1 5`	forces a hard-reset (by asserting the nSRST pin) followed by a 5 second delay
`--jtag-frequency RTCK`	Uses returned TCK (RTCK). When selected, Opella-XD will wait for RTCK before sending a subsequent TCK pulse.

Table 1. Ashling GDB Server Command Line Switches

   the GDB Server should connect to the target as follows:

```
Ashling GDB Server for ARM (ash-arm-gdb-server).
v0.0.3 30-Apr-2008, (c)Ashling Microsystems Ltd 2008.

Setting program entry point (PC) to 0x40000000.

Checking Opella-XD firmware.
Configuring Opella-XD.
Delaying 5 seconds after target hard reset.
Connected to target device configured as: ARM7TDMI-S
(currently in Little Endian mode).
Connected to target via Opella-XD
(diskware:v0.0.3, firmware:v1.0.0) using RTCK.
Waiting for debugger connection on port 2331 for core 1.
Press 'Q' to Quit.
```

The last line indicates that the GDB Server is now ready for a debugger connection (and that the pressing 'Q' will quit the Server).

3. Now run the GDB debugger and specify the program you wish to debug.

```
> arm-elf-gdb obj/simple.elf
```

`arm-elf-gdb` is the executable name of the ARM GDB debugger and `simple.elf` is the program we wish to debug. Windows users may need to run this in a Cygwin Command Shell (as `arm-elf-gdb` requires access to Cygwin DLLs).

Connect to the GDB Server in GDB by entering:

```
(gdb) target remote:2331
```

4. When we connect, we will see an acknowledgement in the GDB Server Console Window as follows:

```
…
Waiting for debugger connection on port 2331 for core 1 .
Press 'Q' to Quit.
Got a debugger connection from 192.168.10.58 on port 2331.
```

5. Now download the program to the target:

```
(gdb) load
Loading section .text, size 0x6e0 lma 0x40000000
Start address 0x40000000, load size 1760
Transfer rate: 14080 bits in <1 sec, 146 bytes/write.
(gdb)
```

6. Set a breakpoint and execute as follows:

```
(gdb) break main
Breakpoint 1 at 0x40000138: file src/simple.c, line 44.
(gdb) continue
Continuing.

Breakpoint 1, main () at src/simple.c:44
44          InitialiseI2C();
(gdb)
```

Our breakpoint was taken at main, lets look at the source-code associated with the current PC value

```
(gdb) list
39          unsigned char i;
40          volatile unsigned long * pulGpioReg;
41          unsigned long x;
42
43          // Initialisation....
44          InitialiseI2C();
45
46          // loop forever,
47          while (1)
48          {
(gdb)
```

7. To view memory, use the X command as follows:

```
(gdb) x /16 0x80000000
0x80000000: 0xe321f0d2 0xe321f0d2 0xe321f0d2 0xe321f0d2
0x80000010: 0xe321f0d2 0xe321f0d2 0xe321f0d2 0xe321f0d2
0x80000020: 0xe321f0d2 0xe321f0d2 0xe321f0d2 0xe321f0d2     0x80000030: 0xe321f0d2
0xe321f0d2 0xe321f0d2 0xe321f0d2     (gdb)
```

8. The SET command allows you to write to memory as follows:

```
(gdb) set {int} 0x40001000 = 0x12345678
 (gdb) x /w 0x40001000
0x40001000 <cI2CAddrVal>:       0x12345678
(gdb)
```

4. GDB Detach, Quit and Kill commands

The Ashling GDB Server supports these commands as follows:

- **detach** disconnects from the Ashling GDB Server and resumes target execution.
- **quit** disconnects from the Ashling GDB Server, resumes target execution and exits GDB.
- **kill** resets the target and disconnects from the Ashling GDB Server (target execution is not resumed).

5. Improving GDB Download Performance

GDB allows you to set the packet size used when transferring information to/from the GDB Server. A larger packet size will improve throughput and thus download performance. The maximum supported packet size (as of writing) is 2048 bytes. The packet size can be set from the GDB command prompt for both reading and writing memory as follows:

```
set remote memory-write-packet-size 2048
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 2048
set remote memory-read-packet-size fixed
```

Alternatively, add the above commands to your `.gdbinit` file to automatically use these settings every time.

GDB's current settings can be seen as follows:

```
show remote memory-write-packet-size
show remote memory-write-packet-size
```

6. Extended GDB functionality

The Ashling GDB server adds several useful additional commands to the GDB debugger that can be accessed via the monitor command. For example, to assert a hardware reset from within GDB, issue the following command:

```
> monitor hwreset
```

This causes Opella-XD to assert the nSRST* pin on the target.

MONITOR command	Explanation

`ashload`	The Ashling GDB Server allows you to directly download ELF files which will give improved performance over GDB's load command. This can be invoked directly from the GDB command line (console) as follows:
	`> monitor ashload <filename> <--verify>`
	for example, the following command will load the elf file `c:\kernels\vmlinux.elf` to target memory:
	
	`> monitor ashload c:\kernels\vmlinux.elf`
	
	the following command will load the elf file `c:\kernels\vmlinux.elf` to target memory and verify it:
	
	`> monitor ashload c:\kernels\vmlinux.elf --verify`
	
	Please note the following:
	o Full path names should be used for files to ensure that the Ashling GDB Server can find them.
	o If you are downloading very large images then GDB may timeout, increase the GDB timeout value (in seconds) using the following command:
	`> set remotetimeout 10000`

`hwreset`	Reset target (hardware reset via nSRST pin).
`hwbreak <on|off>`	Force use of hardware breakpoints (default is off)

`swreset`	Reset target (software reset) which is defined as follows: o Ensure ARM core processor in debug mode o Put core into ARM/supervisor mode with IRQ and FIQ interrupts disabled. o •Sets the PC value to Program Entry Point (as set via `--program-entry-point` switch).
`wpaddr [r|w|d] [b|h|w|d] <addr>`	Set watchpoint when <addr> accessed. Support access qualifiers are: **r** read. **w** write. **d** don't care. Supported size qualifiers are: **b** byte. **h** half-word. **w** word. **d** don't care.
`wpdata [r|w|d] [b|h|w] <data>`	Set watchpoint when <data> value occurs on the bus. Access/size qualifiers as `WPADDR`.
`monitor wpad [r|w|d] [b|h|w|d] <addr> <data>`	Set watchpoint when <addr> accessed with specific <data> value. Access/size qualifiers as `WPADDR`.
`wpclr`	Clear all watchpoints
`semi-hosting <on|off> <addr1> <addr2>`	Enable semi-hosting (default is off). <addr1> Top of memory (no default). <addr2> Vector trap address (default:0x8).

Doc: APB198-ARMGDB.doc, v1.2, 25th June 2008, Hugh O'Keeffe, Ashling Microsystems

Ashling Microsystems Ltd
National Technology Park
Limerick
Ireland
Phone: +353 61 334466
Email: ashling.support@nestgroup.net