Freescale Semiconductor
Application Note

Document Number: AN4420
Rev. G, December/2011

Linux Kernel Program Tracing using Nexus

(QorIQ e500mc devices)

by *Software Analysis Team*
Freescale Semiconductor, Inc.
Austin, TX

This application note details the configuration of the Nexus debug facilities in a Freescale QorIQ device allow tracing of the Linux kernel through Nexus Messages. Several scenarios are listed to allow varied collection based on the specific need.

Note

For details on the Nexus implementation and configuration, refer to the “e500mc Reference Manual” Revision J.

Contents

1 Introduction

The Nexus debug facilities of the QorIQ device allows the collected data to be captured to one or more of the following configurable buffer locations:

1) Internally in the Nexus Port Controller (NPC) Trace Buffer
2) A trace buffer defined within the SOC's DDR memory

Host side configuration, capture and decode of the trace data can use a combination of CodeWarrior and/or some expert scripts. These scripts configure the trace logic and perform a simple trace collection by using either of the two trace buffer locations.

There are tradeoffs associated with each method that dictate which is desirable for different trace scenarios.

These tradeoffs will be discussed in detail in each scenario section but the most general statement is comparison is done between speed and volume.

All buffer capabilities offer sufficient bandwidth to allow simple data and event tracing from all cores at full clock rates. However the ability to fully trace the program flow from a single core is limited to the NPC trace buffer.

Tracing to DDR buffers allow larger fixed size traces, but the writing of the buffer may be intrusive to the measured application since the DDR is a shared resource.

Note

This application note defines the tool that will be used in section 2 and details in section 3 some common scenarios for collecting data and the associated specific configuration.

2 General Setup

2.1 CodeWarrior Power Architecture Version 10 Debugger Setup

Note

This application note assumes the use of CodeWarrior Power Architecture (PA) Version 10.1.1 or later. We refer to this as CodeWarrior PA 10 throughout this note.

After using the standard CodeWarrior for Power Architecture 10 installation process, follow the instructions from the next sections to prepare the CodeWarrior debugger for attaching to the P4080 device.

2.1.1 CodeWarrior Project

Create a workspace and create a new project by importing the Linux kernel binary image:

- Choose **File > Import > CodeWarrior > Power Architecture ELF Executable**
- Enter the project name
- Select the path to the Linux kernel image (for example, `vmlinux` file)
- Select **Bareboard Application** toolchain option
- Select the processor type (P4080). In the Debugger Launch Configuration wizard, select the default values

2.1.2 Memory Mapping File

Create a memory mapping file (for example, `linux_mem.txt`) to help the debugger correctly display the assembly code retrieved from the physical memory, and handle software breakpoints correctly. Using a text editor, create a file in your project's folder, using the settings that match your kernel image. For example:

```
range v:0xfe11c500 v:0xfe11c510 1 ReadWrite LogicalData
// An example for specifying address translation blocks:
// Syntax is:
//   translate <v_addr> <ph_addr> <size>
// define an address translation of size 16MB
translate 0xC0000000 0x00000000     0x30000000
// request that elf file downloaded to be subjected to address translation.
AutoEnableTranslations  true
```

2.1.3 Debug Launch Configuration

Create an **Attach** type debug configuration utilizing all P4080 cores which are running Linux:

- Open Run/Debug Configurations
- Right-click on the **CodeWarrior Attach** entry in the left panel and select **New**

- On the Main tab, select the project created above
- On the Main tab, click on **Search project** and select the Linux kernel image
- On the Main tab, click on the **New** button from the Remote System section and follow the wizard steps:
    - Select **CodeWarrior Bareboard Debugging**
    - Select **Hardware** or **Simulator** Connection
    - Provide a connection name (for example, linux)
    - Create a new system type:
        - Use the “CodeWarrior Bareboard Debugging/Hardware or Simulator System/Hardware or Simulator System (default)”
        - Use a system type name such as “linux_system”Select P4080 for the system type
        - On the Initialization tab of the RSE system configurator:
            - uncheck the **Execute system reset** option
            - uncheck all other options
        - On the System tab:
            - check the **Memory Configuration** option for all debugged cores and specify for each core the path to the memory mapping file created above
    - Select the connection type that matches your TAP (for example, Gigabit TAP + Trace, if you have a Gigabit TAP with an Aurora/Nexus daughter card) and complete configuring the TAP settings by following the CodeWarrior online help instructions
- On the Main tab of the debug launch configuration created above, in the Remote System section, select all cores on which the Linux kernel is running (for example, 0 through 7)
- On the Main tab, check the **Ignore target initialization settings** option
- On the Debugger tab, select “None” for the OS Awareness’ Target OS setting
    - Note: setting “OS Awareness” to Linux will only allow core 0 to be controlled unless Linux has already been booted on all cores (for example, before U-boot has completed) and Linux won’t be able to boot successfully.
- On the Trace and Profile tab, uncheck the **Enable Trace and Profile** settings.

2.1.4 Using the Debug Launch Configuration

The Attach debug launch configurations do not reset the P4080 device and therefore, you have two options for using the debug launch configuration created above.

2.1.4.1 Attaching to a Running Linux Kernel

If you have already booted the Linux kernel on the P4080 device, you can start using the Attach debug launch configuration created above.

2.1.4.2 Attaching Before the Linux Kernel Boot Time

In certain situations, you may need to attach the debugger to the P4080 device before the Linux kernel starts running on the cores. For example, this may be necessary if you wish to collect trace data before the kernel starts running (for example, to trace the boot loader) or if you need to trace the Linux kernel’s boot sequence.

The easiest way to achieve this is to reset the P4080 board using the reset button, before using the Attach debug launch configuration you created. However, if this is a remote system and this cannot be done, you may follow these steps:

- Create a **Connect** type debug launch configuration, using a procedure similar to the one used above, for creating an **Attach** type debug launch configuration
- Edit the RSE configuration associated with the **Connect** type debug launch configuration and check the **Execute system reset** and the **Core reset** checkboxes (for all cores) on the Initialization tab
- Launch the **Connect** type debug configuration
- After the debugger successfully connects to all cores, click on the **Multi-core Terminate** button
- Launch the **Attach** type debug configuration

2.2 Linux Console Setup

The Gigabit TAP probe provides a serial port which can be configured to access the serial port of the target system. This is useful if you need to remotely access the serial port of a target system. Refer to the CodeWarrior online help for more instructions regarding connecting the Gigabit TAP probe to the serial port of the P4080 device.

To connect from a host machine to the serial port of the P4080 device, you need to telnet to the IP address of the Gigabit TAP, using port 1082. For example, "`telnet 10.20.30.40 1082`."

2.3 P4080 Boot Process

After launching the **Attach** type debug launch configuration created above, the debugger will connect to all debugged cores and display their current status.

Unless the Linux kernel is already running, a console connection may also need to be open, in order to send commands to the boot loader (U-boot). For example, a boot loader command may need to be entered in order to have the P4080 device boot from an NFS location. The steps are the following:

- Assuming that the cores are halted at the reset vector, they must be resumed by clicking on the **Multicore Resume** button
- The console will start to display the output messages of the boot loader and the boot loader will wait for five seconds for the user to interrupt the default boot sequence, by hitting the Enter key. The boot loader commands, specific to your setup, can be entered at the prompt. For example, type `run nfsboot` in order to boot from an NFS location previously configured.

2.4 Trace Control Python Scripts

Python scripts are provided to allow collection using the supported trace collection methods. The script found at:
`<CodeWarrior>/PA/cdde/sasdk/samples/py/tracebuffers/GetBufferTrace.py`
collects trace data using either the NPC or DDR trace buffers.

The full instructions for running this script are found in file `README_GetBufferTrace.txt` from the directory above. The following is a summary of the instructions from those file.

The Python script is intended to be run from the `<CodeWarrior>/PA/cdde/bin` directory.

The script utilizes a set of default configuration information defined in xml-formatted files found in subdirectories of `<CodeWarrior>/PA/cdde/sasdk/data`

The xml files used by GetBufferTrace are:

`<CodeWarrior>/PA/cdde/sasdk/data/fsl.configs.sa.tracetarget/P4080Expert.xml`

`<CodeWarrior>/PA/cdde/sasdk/data/fsl.configs.sa.traceprobe/TraceProbeP4080DDR.xml`

`<CodeWarrior>/PA/cdde/sasdk/data/fsl.configs.sa.traceprobe/TraceProbeP4080NPC.xml`

While each file contains multiple named configurations, generally we will only use the first configuration that appears in the file. Configurations are named using the `<Configration><Name></Name>` xml block.

The configuration name is important to the scripts and so changing the name is not recommended.

2.4.1 P4080Expert.xml

The `P4080Expert.xml`file defines the Nexus configurations that will be enabled on the target. The default name is Nexus Trace.

Ensure the probe IP address and connection type are entered correctly in the CCS Server configuration block. Only the “Debug Connection Id” needs to be adjusted.

If collecting program trace search for the “Enable Hard Sync” attribute and set its value to true.

All other configurations are scenario specific and will be detailed in Data Collection Scenarios section.

2.4.2 TraceProbeP4080DDR.xml

The `TraceProbeP4080DDR.xml`file defines the Nexus configurations that pertain to the definition of the DDR buffer that will be enabled on the target. The default name is Nexus P4080 DDR.

Ensure the probe IP address and connection type are entered correctly in the CCS Server configuration block. Only the “Debug Connection Id” needs to be adjusted.

The buffer specific configuration blocks will be adjusted when using the DDR as the Nexus Trace buffer.

2.4.3 TraceProbeP4080NPC.xml

This file defines the Nexus configurations that pertain to the definition of the NPC Trace buffer. The default name is Nexus P4080 NPC.

Ensure the probe IP address and connection type are entered correctly in the CCS Server configuration block. Only the “Debug Connection Id” needs to be adjusted.

The buffer specific configuration blocks are read-only since the NPC buffer definition is fixed by the hardware design.

3 Data Collection Scenarios

3.1 Collection from the NPC Trace Buffer

The NPC Trace buffer is small (16k) and can operate in a single fill mode or a wrap mode where only the last 16kB worth of trace messages are retained.

Defining the beginning of the trace collection is important and if the Nexus tracing is enabled, the trace buffer will quickly fill with Timestamp overflow messages.

The following scenario uses the NPC buffer in the non-wrapped mode, triggered from exiting debug mode:

3.1.1 TraceProbeP4080NPC.xml

3.1.1.1 ConfigBlock “Buffer”

Set the **Wrap Enable** attribute to **false** to set the NPC Trace buffer wrap mode off.

3.1.1.2 ConfigBlock “Misc”

Set the **Suppress trace until core 0 resumes** attribute to **true**. Trace collection will begin as soon as the core is released from the debug mode.

Set the **Trigger trace on core 0 IAC1** attribute to **false**.

3.1.2 P4080Expert.xml

3.1.2.1 ConfigBlock “Core 0”

All of the configuration will be defined for core 0 since this is the boot core for Linux. All other “core n” should have the **Enable** attribute set to **false**.

Set **Program Trace Enable** to **true**. Set **Watchpoint Trace**, **Data Acquisition Trace**, and **Ownership Trace** as desired.

3.1.2.2 ConfigBlock “Misc”

Set the **Enable Hard Sync** attribute to **true** in order to cause the trace logic to periodically send program trace synchronization messages. This is recommended when collecting trace with the NPC trace buffer configured in the circular (wrapping) mode. This is also recommended in case the decoded program trace data appears to be invalid.

3.1.3 GetBufferTrace.py

Begin the trace collection by invoking the CodeWarrior Debug configuration. All cores would be held in suspended.

Run the GetBufferTrace script with appropriate values for the options.

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=NPC -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-
p4080/boot/vmlinux -out=trace-npc-out.txt
```

Command line switches used explained below:

`python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py` – Runs the specified python script

`-buffer=NPC` – Defines the buffer location as NPC (also implicitly selects the `TraceProbeP4080NPC.xml` configuration file)

`-target="Nexus Trace"` – Selects the configuration named “Nexus Trace” from all configuration files in the `fsl.configs.sa.tracetarget` directory

`-preftype=expert` – Selects the “Expert” version of the target configuration files (P4080Expert.xml)

`-elf=/srv/nfsroot/sa-p4080/boot/vmlinux` – Selects the ELF file which the trace decoder will use to resolve trace program addresses.

`-out=trace-npc-out.txt` – Defines an output file for the decoded trace

GetBufferTrace will configure the target system and wait for trace output before automatically uploading a single full buffer trace from the NPC and beginning the trace decode.

Program output will be similar to:

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=NPC -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-
p4080/boot/vmlinux -out=trace-npc-out.txt
Configuring target to output trace to NPC buffer...
name = "Nexus Trace"
found requested config
Core 0...
Core 1...
Core 2...
Core 3...
Core 4...
Core 5...
Core 6...
Core 7...
Starting trace collection, ...
Task "Fill NPC Buffer"
NST = 0x90000001
Current buffer count = 0 (0x0)
Fill NPC Buffer progress:   0.0% NST = 0x90000001
Current buffer count = 0 (0x0)
NST = 0xb01c0000
Current buffer count = 16384 (0x4000)
Fill NPC Buffer progress: 100.0%, elapsed -  0 m  0 s, remaining -  0 m  0 s
Task "Upload NPC Buffer"
Upload NPC Buffer: progress 100.0%, elapsed -  0 m  0 s, remaining -  0 m  0 s
```

```
redirecting output to "trace-npc-out.txt"
Done
Trace collection and decode successful.
```

If the NPC buffer is set to wrap mode enabled, a prompt reminder will be printed to hit Ctrl-C key to manually stop the trace collection and to begin upload and decode of the trace data.

Trace output will be similar to:

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=NPC -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-
p4080/boot/vmlinux -out=trace-npc-out.txt
Configuring target to output trace to NPC buffer...
name = "Nexus Trace"
found requested config
NCP Trace buffer will wrap on overflow. Hit Ctrl-C to stop trace.
Core 0...
Core 1...
Core 2...
Core 3...
Core 4...
Core 5...
Core 6...
Core 7...
Starting trace collection, ...

Task "Fill NPC Buffer"
Fill NPC Buffer progress:   0.0%
NST = 0x90000001
Current buffer count = 0 (0x0)
NST = 0xb008015b
Current buffer count = 5536 (0x15a0)
Fill NPC Buffer progress:  33.8%, elapsed -  0 m  0 s, remaining -  0 m  0 s
NST = 0xb000003b
Current buffer count = 928 (0x3a0)
NST = 0xb00400ca
Current buffer count = 3216 (0xc90)
NST = 0xb0080151
Current buffer count = 5376 (0x1500)
^CStopped trace and begin trace upload

Task "Upload NPC Buffer"
Upload NPC Buffer: progress 100.0%, elapsed -  0 m  0 s, remaining -  0 m  0 s
redirecting output to "trace-npc-out.txt"
Done
Trace collection and decode successful.
```

3.2 Collection from the NPC Trace Buffer from Specific Instruction Address

The NPC Trace buffer is small (16k) so particular attention is needed to focus in on the code of interest for tracing. The NPC buffer can also operate in a wrap mode where only the last 16kB worth of trace messages are retained.

The following scenario uses the NPC buffer in the non-wrapped mode, triggered from a specific instruction address.

3.2.1 TraceProbeP4080NPC.xml

3.2.1.1 ConfigBlock “Buffer”

Set the **Wrap Enable** attribute to **false** to set the NPC Trace buffer wrap mode off.

3.2.1.2 ConfigBlock “Misc”

Set the **Trigger trace on core 0 IAC1** attribute to **true** to configure Nexus tracing to begin only when the specified Instruction address is executed. For more details, refer to the `P4080Expert.xml` section.

Set the **Suppress trace until core 0 resumes** attribute to **false**.

3.2.2 P4080Expert.xml

3.2.2.1 ConfigBlock “Core 0”

All of the configuration will be defined for core 0 since this is the boot core for Linux. All other “core n” should have the **Enable** attribute set to **false**.

Set **Program Trace Enable** to false. Program trace will be enabled under nexus control when IAC1 matches.

Set **Start Trigger** to 1. This selects Program Trace enable to trigger on IAC1.

Optionally set **End Trigger** to 2 – Tracing stops on IAC2

Set **Watchpoint Trace**, **Data Acquisition Trace**, and **Ownership Trace** as desired.

Set “Instruction Address Compare” “Enable”; “Address 1 enable” and optionally “Address 2 enable” True.

Set values for **Address 1** and optionally **Address 2** to define the Linux address of interest.

3.2.2.2 ConfigBlock “Misc”

Set the **Enable Hard Sync** attribute to **true** in order to cause the trace logic to periodically send program trace synchronization messages. This is recommended when collecting trace with the NPC

trace buffer configured in the circular (wrapping) mode. This is also recommended in case the decoded program trace data appears to be invalid.

3.2.2.3 ConfigBlock “Event Out Control”

Set **Event Out Control Enable** to **true**.

Set **Event Out 0 Mask** to **0x1**. This asserts EVTO0 when IAC1 address is executed. This trigger is also linked to the NPC enable to allow tracing to the NPC buffer to begin.

3.2.3 GetBufferTrace.py

Begin trace collection by invoking the CodeWarrior Debug configuration. All cores would be held in Suspended.

Run the GetBufferTrace script with appropriate values for the options.

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=NPC -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-
p4080/boot/vmlinux -out=trace-npc-out.txt
```

Command line switches used explained below:

`python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py` – Runs the specified python script

`-buffer=NPC` – Defines the buffer location as NPC (also implicitly selects the `TraceProbeP4080NPC.xml` configuration file)

`-target="Nexus Trace"` – Selects the configuration named “Nexus Trace” from all configuration files in the `fsl.configs.sa.tracetarget` directory

`-preftype=expert` – Selects the “Expert” version of the target configuration files (P4080Expert.xml)

`-elf=/srv/nfsroot/sa-p4080/boot/vmlinux` – Selects the ELF file which the trace decoder will use to resolve trace program addresses.

`-out=trace-npc-out.txt` – Defines an output file for the decoded trace

GetBufferTrace will configure the target system and wait for trace output before automatically uploading a single full buffer trace from the NPC and beginning the trace decode.

Program output will be similar to:

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=NPC -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-
p4080/boot/vmlinux -out=trace-npc-out.txt
Configuring target to output trace to NPC buffer...
name = "Nexus Trace"
found requested config
Core 0...
Core 1...
Core 2...
```

```
Core 3...
Core 4...
Core 5...
Core 6...
Core 7...
Starting trace collection, ...
Task "Fill NPC Buffer"
NST = 0x90000001
Current buffer count = 0 (0x0)
Fill NPC Buffer progress:   0.0% NST = 0x90000001
Current buffer count = 0 (0x0)
NST = 0xb01c0000
Current buffer count = 16384 (0x4000)
Fill NPC Buffer progress: 100.0%, elapsed -  0 m  0 s, remaining -  0 m  0 s
Task "Upload NPC Buffer"
Upload NPC Buffer: progress 100.0%, elapsed -  0 m  0 s, remaining -  0 m  0 s
redirecting output to "trace-npc-out.txt"
Done
Trace collection and decode successful.
```

If the NPC buffer was set to wrap mode enabled, a prompt reminder will be printed to hit Ctrl-C key to manually stop the trace collection and to begin upload and decode of the trace data.

Trace output will be similar to:

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=NPC -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-
p4080/boot/vmlinux -out=trace-npc-out.txt
Configuring target to output trace to NPC buffer...
name = "Nexus Trace"
found requested config
NCP Trace buffer will wrap on overflow. Hit Ctrl-C to stop trace.
Core 0...
Core 1...
Core 2...
Core 3...
Core 4...
Core 5...
Core 6...
Core 7...
Starting trace collection, ...

Task "Fill NPC Buffer"
Fill NPC Buffer progress:   0.0%
NST = 0x90000001
```

```
Current buffer count = 0 (0x0)
NST = 0xb008015b
Current buffer count = 5536 (0x15a0)
Fill NPC Buffer progress:  33.8%, elapsed -  0 m  0 s, remaining -  0 m  0 s
NST = 0xb000003b
Current buffer count = 928 (0x3a0)
NST = 0xb00400ca
Current buffer count = 3216 (0xc90)
NST = 0xb0080151
Current buffer count = 5376 (0x1500)
^CStopped trace and begin trace upload

Task "Upload NPC Buffer"
Upload NPC Buffer: progress 100.0%, elapsed -  0 m  0 s, remaining -  0 m  0 s
redirecting output to "trace-npc-out.txt"
Done
Trace collection and decode successful.
```

3.3 Collection from the DDR memory buffer

The following scenario uses the DDR memory as the trace buffer in the non-wrapped mode, triggered from exiting debug mode.

The main configuration goal is to define a DDR buffer at an address which is not in conflict with any user application or the Operating System.

Linux can be restricted to use less than all available memory thus freeing some memory for Nexus trace use.

3.3.1 U-boot configuration for Linux

Restrict Linux to use less than 2GB of memory. For more details, refer to the ConfigBlock “Buffer” section.

For example, using 1.5GB

```
=> setenv othbootargs mem=1536M
=> saveenv
```

3.3.2 TraceProbeP4080DDR.xml

3.3.2.1 ConfigBlock “Buffer”

Set the **Address** attribute to the beginning of the DDR buffer. For example, 0x6000_0000 (1.5GB).

NOTE

Issue: The CW scripts will not allow an address greater than 0x7fff_ffff (2GB) to be entered.

Set the **Size** attribute to the size of the DDR buffer in bytes. For example, 0x0010_0000 (1MB).

3.3.2.2 ConfigBlock “Misc”

Set the **Suppress trace until core 0 resumes** attribute to **true**. Trace collection will begin as soon as the core is released from the debug mode.

3.3.3 P4080Expert.xml

3.3.3.1 ConfigBlock “Core 0”

All of the configuration will be defined for core 0 since this is the boot core for Linux. All other “core n” should have the **Enable** attribute set to **false**.

Set **Program Trace Enable** to **true**.

Set **Watchpoint Trace**, **Data Acquisition Trace**, and **Ownership Trace** as desired.

3.3.3.2 ConfigBlock “Misc”

Set the **Enable Hard Sync** attribute to **true** in order to cause the trace logic to periodically send program trace synchronization messages. This is recommended when collecting trace with the NPC trace buffer configured in the circular (wrapping) mode. This is also recommended in case the decoded program trace data appears to be invalid.

3.3.4 GetBufferTrace.py

Begin trace collection by invoking the CodeWarrior Debug configuration. All cores would be held in suspended.

Run the GetBufferTrace script with appropriate values for the options.

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=DDR -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-
p4080/boot/vmlinux -out=trace-ddr-out.txt
```

Command line switches used explained below:

`python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py` – Runs the specified python script

`-buffer=DDR` – Defines the buffer location as DDR (also implicitly selects the `TraceProbeP4080DDR.xml` configuration file)

`-target="Nexus Trace"` – Selects the configuration named “Nexus Trace” from all configuration files in the `fsl.configs.sa.tracetarget` directory

`-preftype=expert` – Selects the “Expert” version of the target configuration files (P4080Expert.xml)

`-elf=/srv/nfsroot/sa-p4080/boot/vmlinux` – Selects the ELF file which the trace decoder will use to resolve the trace program addresses.

`-out=trace-ddr-out.txt` – Defines an output file for the decoded trace

GetBufferTrace will configure the target system and wait for trace output before automatically uploading a single full buffer trace from the DDR and beginning the trace decode.

Program output will be similar to:

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=DDR -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-p4080-
2/boot/vmlinux -out=trace-ddr-out.txt
Configuring target to output trace to DDR buffer...
name = "Nexus Trace"
found requested config
Applying Trace Tool Configuration...

Applying Target Configuration...
Starting trace collection, ...
Task "Fill DDR Buffer"
Current buffer count = 32 (0x20)
Fill DDR Buffer progress: 100.0%, elapsed -  0 m  0 s, remaining -  0 m 60 s
Task "Upload DDR Buffer"
Upload DDR Buffer progress:   0.0%
Upload DDR Buffer progress: 100.0%, elapsed -  0 m  2 s, remaining -  0 m  0 s
Upload DDR Buffer: progress 100.0%, elapsed -  0 m  4 s, remaining -  0 m  0 s
redirecting output to "trace-ddr-out.txt"
Done
Trace collection and decode successful.
```

If the DDR buffer was set to wrap mode enabled, a prompt reminder will be printed to hit Ctrl-C key to manually stop the trace collection and to begin upload and decode of the trace data.

Trace output will be similar to:

```
[sgeorge@sa bin]$ python ../sasdk/samples/py/tracebuffers/GetBufferTrace.py -
buffer=DDR -target="Nexus Trace" -preftype=expert -elf=/srv/nfsroot/sa-p4080-
2/boot/vmlinux -out=trace-ddr-out.txt
Configuring target to output trace to DDR buffer...
name = "Nexus Trace"
found requested config
Applying Trace Tool Configuration...
NCP Trace buffer will wrap on overflow. Hit Ctrl-C to stop trace.

Applying Target Configuration...
Starting trace collection, ...
Task "Fill DDR Buffer"
Current buffer count = 32 (0x20)
Fill DDR Buffer progress:   0.0%, elapsed -  0 m  0 s, remaining - 16 m 23 s Current
buffer count = 32 (0x20)
Current buffer count = 1400576 (0x155f00)
```

```
Fill DDR Buffer progress: 133.6%, elapsed -  0 m  0 s, remaining -  0 m 59 s Current
buffer count = 75315808 (0x47d3a60)
Fill DDR Buffer progress: 7182.7%, elapsed -  0 m  0 s, remaining -  0 m 59 s Current
buffer count = 150280768 (0x8f51a40)
Fill DDR Buffer progress: 19769.3%, elapsed -  0 m  0 s, remaining -  0 m 59 s Stopped
trace and begin trace upload

Task "Upload DDR Buffer"
Upload DDR Buffer progress:   0.0%
Upload DDR Buffer progress:  27.3%, elapsed -  0 m  1 s, remaining -  0 m  3 s
Upload DDR Buffer: progress 100.0%, elapsed -  0 m  7 s, remaining -  0 m  0 s
redirecting output to "trace-ddr-out.txt"
Done
Trace collection and decode successful.
```

4 Revision History

Table 1 provides a revision history for this application note.

Table 1. Revision History

Rev. Number	Date	Substantive Change
A	2010 September 13	Initial creation
B	2010 October 15	Added more scenarios
C	2010 October 21	Added DDR scenario trace output
D	2010 December 8	Removed FCP from footers.
E	2010 December 15	Added debug launch configuration setup information. Added instructions for enabling the hard sync option.
F	2011 September 30	Update due to CodeWarrior interface changes.
G	2011 December 12	Fix Typos

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4420
Rev. G
12/2011