Freescale Semiconductor
Application Note

Document Number: AN5001

Collecting Linux Trace without using CodeWarrior

1. Introduction

This document guides you how to collect Linux trace directly from QDS or TWR board without using CodeWarrior. The tool encapsulates the trace configurator and probe into one small and cross-compiled component that is uploaded on the target machine. Its main use is to collect the trace of a program and also it is used for profiling the application.

Contents

2. Overview

The Linux trace mechanism is independent of CodeWarrior. The trace data is collected using a QorIQ LS1021A or LS1024A Linux board.

The advantages of the ARMv7 standalone tracing tool are:

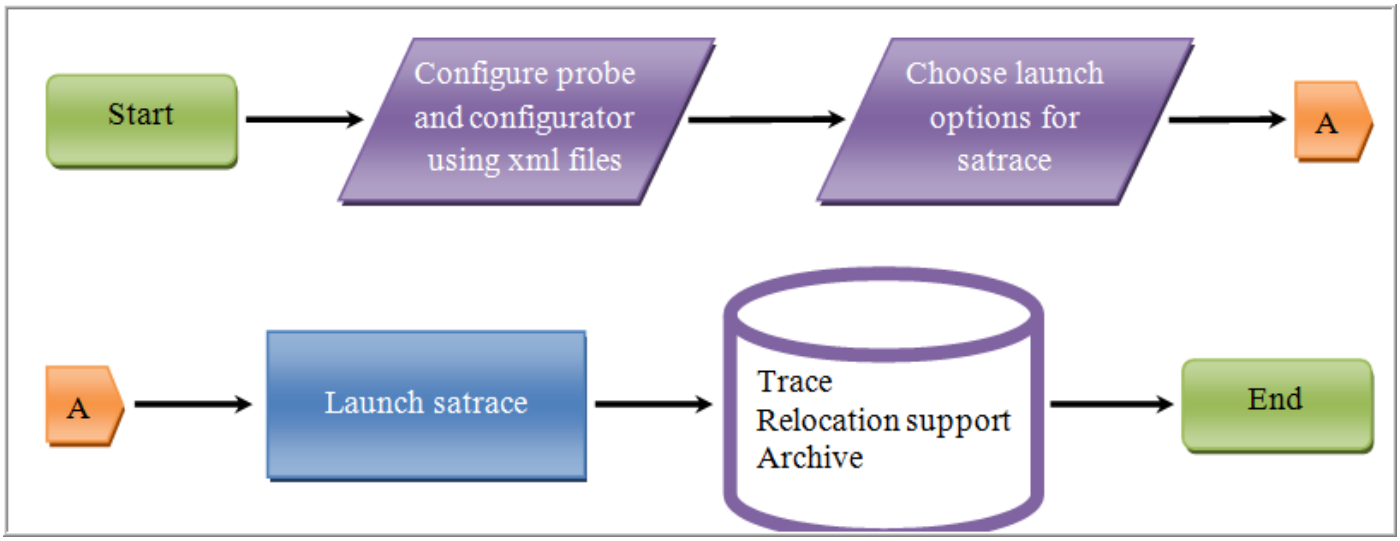
- Size: Contains only what is needed
- Speed: All services are hosted on target machine and there are no delays caused by communication between multiple workstations or languages

- Nonintrusive: No need to instrument the target application
- Easy to use: Collects all required/available information for decoding
- Simple API: Can be easily integrated into any testing framework
- Data-driven: The platform and probe configuration can be easily tuned up and scaled to user needs

3. Execution flow

The execution flow is described below:

Figure 1. Flowchart for ARMv7 standalone Linux trace



You can configure the Layerscape platform and probe in `PlatformConfig.xml` file, and then specify the launch configuration options in `ls.linux.satrace` executable. Run the executable with user space commands, it will generate the relocation support file and the trace file for the application that is being traced.

The root folder of the package will have the following file structure:

- `bin` folder: Binary files are needed for trace session; it will have `ls.linux.satrace` executable
- `config` folder: Platform configuration files; it will have `PlatformConfig.xml` file
- `lib` folder: Library files

Below are the listed options with a short description.

Usage: *./ls.linux.satrace [Options] app [app_args]*

Table 1. General command description - Options

Command	Description
-v [--verbose]	Verbose mode
-V [--version]	Product version
-h [--help]	Displays the help message

Table 2. User space command description - Options

Command	Description
-A [--archive-file]	Path of the generated archive
-b [--backtrace]	Shows backtrace on SEGFAULT
-p [--pid]	Attach to a process giving a PID

Table 3. Kernel space command description - Options

Command	Description
-K [--kernel]	Path of the generated kernel archive
-i [--kernel-image]	vmlinux image path
-m [--module-name]	Name of the traced module

Table 4. System trace command description - Options

Command	Description
-S [--system] arg	Path of the generated archive
-i [--kernel-image]	vmlinux image path
-p [--pid] PID	Attach to a process giving a PID
-b [--backtrace]	Shows backtrace on SEGFAULT

4. User space trace

The relocation file contains a list of libraries linked with the traced application with their load addresses. This list also contains libraries injected through *LD_PRELOAD* variable.

The trace file incorporates the raw trace collected by Embedded Trace Buffer (ETB) and Trace Memory Controller (TMC) probes from the location specified in the probe configuration file.

The *-A* option is the most verbose. It archives the applications, all its dependencies (shared libraries), trace file, the configuration file, and relocation support. This is the default option. Its use increases the time and file-system space required for archiving. The main advantage is the generated **.cwzsa* file. It is an archive file that can be imported and fully decoded using ARMv7 decoder or ARMv7 CodeWarrior.

The *-v* option will generate a more detailed output at standard output. The *SEGFAULT* is a signal triggered by the kernel to a user space application when a memory access violation is made. Usually, the *SEGFAULT* signal is the main reason for the crash of C/C++ applications. Thus, a backtrace on *SEGFAULT* is important where each byte of file system matters. The *-b* option will dump all known stack frames without having support from a debugger. Before using this option, you must ensure that the traced application has been compiled with debug information (*-g* for GCC) and extra code for exception propagation (*-funwind-tables* for GCC) and all symbols are added to the dynamic symbol table (*-rdynamic* for GCC).

Before running any examples, make sure that your kernel is already compiled with enabled *PID_IN_CONTEXTIDR* configuration option. All the steps mentioned below are done on the target machine.

Create a small program that computes the sum of elements from *0* to *num* and crashes due to a segmentation fault.

```
#include <iostream>

class SegFaultTest

{

      public:

      SegFaultTest()

      {

            sum(5);

            function1();

      }

      private:

      void function1() { function2(); }

      void function2() { function3(); }

      void function3() { function4(); }
```

```
        void function4() { crash(); }
        void crash()
        {
                char * p = NULL;
                *p = 0;
        }
        int sum(int n)
        {
                if (n <= 0)
                {
                        return n;
                }
                return n + sum(n - 1);
        }
};

int main(int argc, char ** argv)
{
        SegFaultTest * f = new SegFaultTest();
        return 0;
}
```

After saving the above program in a file, `segfault.cpp`, you should compile it with debugging symbols as shown below:

```
g++ -g3 -funwind-tables -rdynamic segfault.cpp -o segfault
```

Now, try to figure out which line caused the crash. Launch the `segfault` executable using `ls.linux.satrace`.

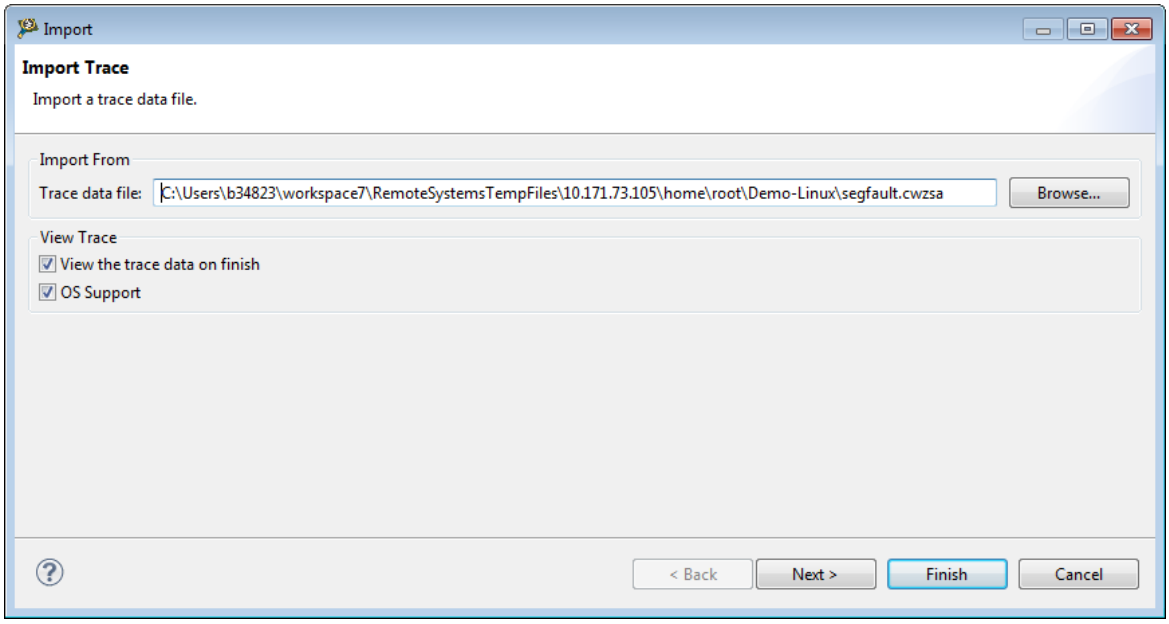
```
root@ls1021aqds:~# ./linux.armv7.satrace/bin/ls.linux.satrace -b -v ./segfault
```

```
User space trace

Application: `./segfault`

Arguments:

Relocation file: `/home/root/segfault.rlog`

Trace file: `/home/root/segfault.dat`

Starting `./segfault`

Signal 11 (Segmentation fault), address is 0

     (1) ./segfault: SegFaultTest::crash()+0xf [0x8bc4]

    (2) ./segfault: SegFaultTest::function4()+0xd [0x8bae]

    (3) ./segfault: SegFaultTest::function3()+0xd [0x8b9a]

    (4) ./segfault: SegFaultTest::function2()+0xd [0x8b86]

    (5) ./segfault: SegFaultTest::function1()+0xd [0x8b72]

    (6) ./segfault: SegFaultTest::SegFaultTest()+0x15 [0x8b5a]

    (7) ./segfault: main+0x19 [0x8ad2]

    (8) /lib/libc.so.6: __libc_start_main+0x110 [0x76c912b8]

User application terminated because it didn't catch signal number: 11 (Segmentation
 fault)

Master process

Collecting trace ...

Archive file: `/home/root/segfault.cwzsa`

Creating archive....

Archiving /home/root/segfault.rlog

Archiving /home/root/segfault

Archiving /lib/librt-2.18-2013.10.so

Archiving /lib/libdl-2.18-2013.10.so

Archiving /lib/libpthread-2.18-2013.10.so

Archiving /lib/libc-2.18-2013.10.so
```

```
Archiving /lib/libm-2.18-2013.10.so

Archiving /lib/ld-2.18-2013.10.so

Archiving ./linux.armv7.satrace/config/PlatformConfig.xml

Archiving /home/root/segfault.dat
```

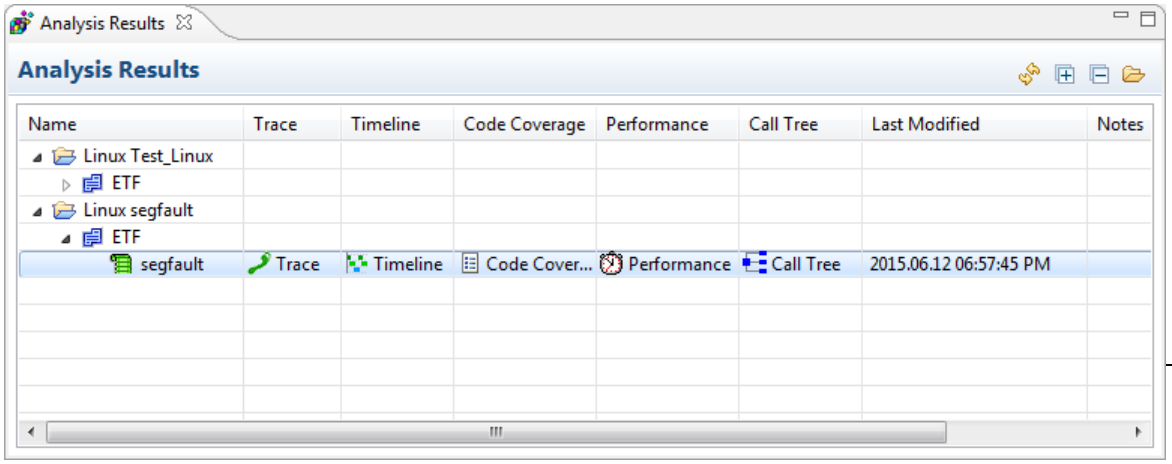
The executable collects trace and archives all dependencies into `/home/root/segfault.cwzsa` archive. You can view the generated archive in CW ARMv7 with a drag-and-drop action. As a result, the **Import** wizard starts, as shown in the figure below.

Figure 2. Import wizard – user space



Click **Finish** to end the **Import** wizard. The file is imported and it is displayed in the **Analysis Results** view.
Click the **Trace** link under the **Trace** column in the **Analysis Results** view to view the trace data, as shown in the figure below.

Figure 3. Analysis Results view – user space



The trace data file opens in the **Trace** viewer showing the trace results, as shown in the figure below.

Figure 4. Trace viewer – user space

segfault.csv

Index	Source	Type	Description	Address	Destination	Timestamp
97	Core 0	Linear	Function SegFaultTest::sum(int)	0x87e0		0
98	Core 0	Branch	Branch from SegFaultTest::sum(int) to SegFaultTest::sum(int)	0x87ec	0x87e0	0
99	Core 0	Linear	Function SegFaultTest::sum(int)	0x87e0		0
100	Core 0	Branch	Branch from SegFaultTest::sum(int) to SegFaultTest::sum(int)	0x87ec	0x87e0	0
101	Core 0	Linear	Function SegFaultTest::sum(int)	0x87e0		0
102	Core 0	Branch	Branch from SegFaultTest::sum(int) to SegFaultTest::sum(int)	0x87ec	0x87e0	0
103	Core 0	Linear	Function SegFaultTest::sum(int)	0x87e0		0
104	Core 0	Branch	Branch from SegFaultTest::sum(int) to SegFaultTest::SegFaultTest()	0x87ec	0x8744	0
105	Core 0	Linear	Function SegFaultTest::SegFaultTest()	0x8744		0
106	Core 0	Branch	Branch from SegFaultTest::SegFaultTest() to SegFaultTest::function1()	0x8746	0x8754	0
107	Core 0	Linear	Function SegFaultTest::function1()	0x8754		0
108	Core 0	Branch	Branch from SegFaultTest::function1() to SegFaultTest::function2()	0x875e	0x8768	0
109	Core 0	Linear	Function SegFaultTest::function2()	0x8768		0
110	Core 0	Branch	Branch from SegFaultTest::function2() to SegFaultTest::function3()	0x8772	0x877c	0
111	Core 0	Linear	Function SegFaultTest::function3()	0x877c		0
112	Core 0	Branch	Branch from SegFaultTest::function3() to SegFaultTest::function4()	0x8786	0x8790	0
113	Core 0	Linear	Function SegFaultTest::function4()	0x8790		0
114	Core 0	Branch	Branch from SegFaultTest::function4() to SegFaultTest::crash()	0x879a	0x87a4	0
115	Core 0	Linear	Function SegFaultTest::crash()	0x87a4		0
116	Core 0	Info	Exception packet - ETM - last instruction traced was canceled			0

5. Kernel space trace

The same executable can be used for kernel space tracing without using a dedicated hardware probe. For this type of trace, the following three kernel space options are used:

- `-K`: Starts a kernel space trace session and also specifies the name of the generated archive
- `-i`: It is optional. It points to the vmlinux image of the system. This option is useful only when the kernel image contains debug information; otherwise, `-K` option is more convenient to use.
- `-m`: Traces the code generated from a kernel module

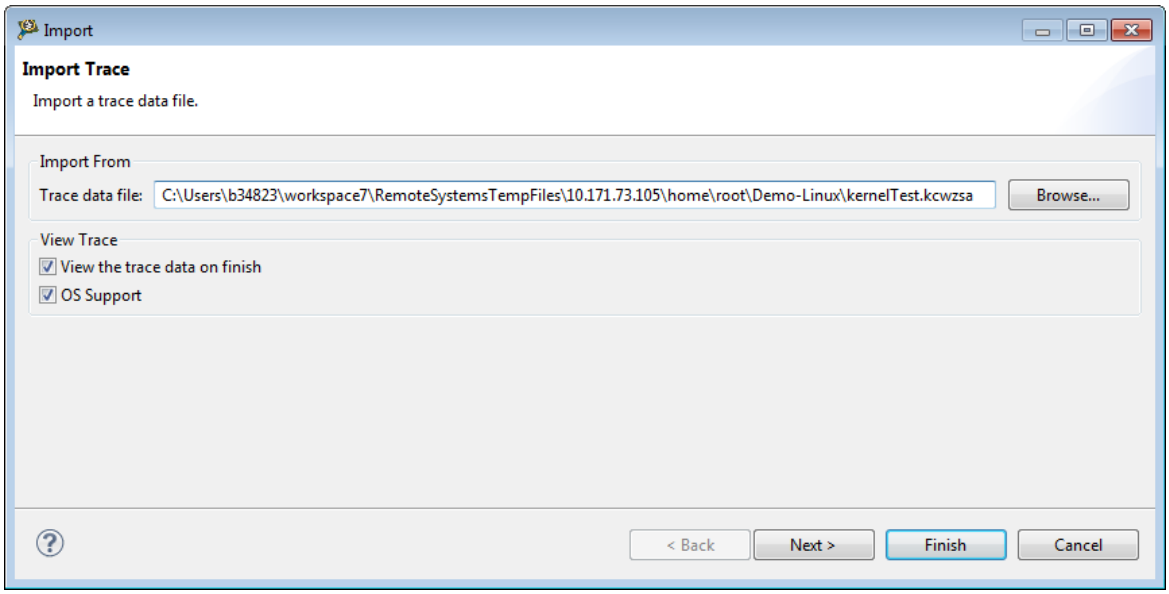
Run the `satrace` with `-K` and `-i` options. After few seconds, send a `SEGINT` signal by pressing `CTRL+C` on your keyboard.

```
root@ls1021aqds:~# ./linux.armv7.satrace/bin/ls.linux.satrace -v -K kernelTest -
i ~/vmlinux

Kernel space trace

Archive: `kernelTest.kcwzsa`

Hit CTRL+C to stop trace.

Collecting trace ...

Kernel image: `/home/root/vmlinux`
```

```
Archive file: `kernelTest.kcwzsa`

Creating archive ....

Archiving /home/root/vmlinux

Archiving ./linux.armv7.satrace/config/PlatformConfig.xml

Archiving kernelTest.dat
```

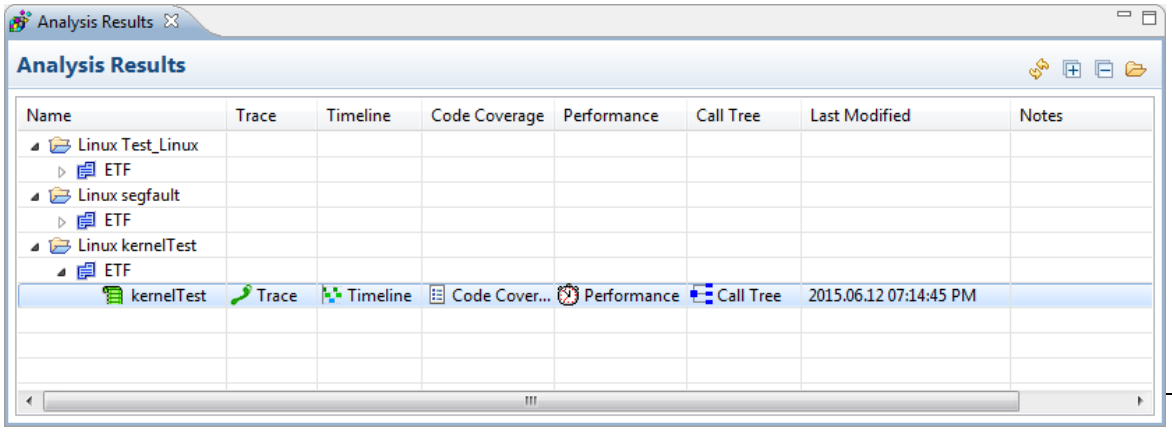
The generated archive can be opened in CW ARMv7 with a drag-and-drop action. As a result, the **Import** wizard starts, as shown in the figure below.

Figure 5. Import wizard – kernel space



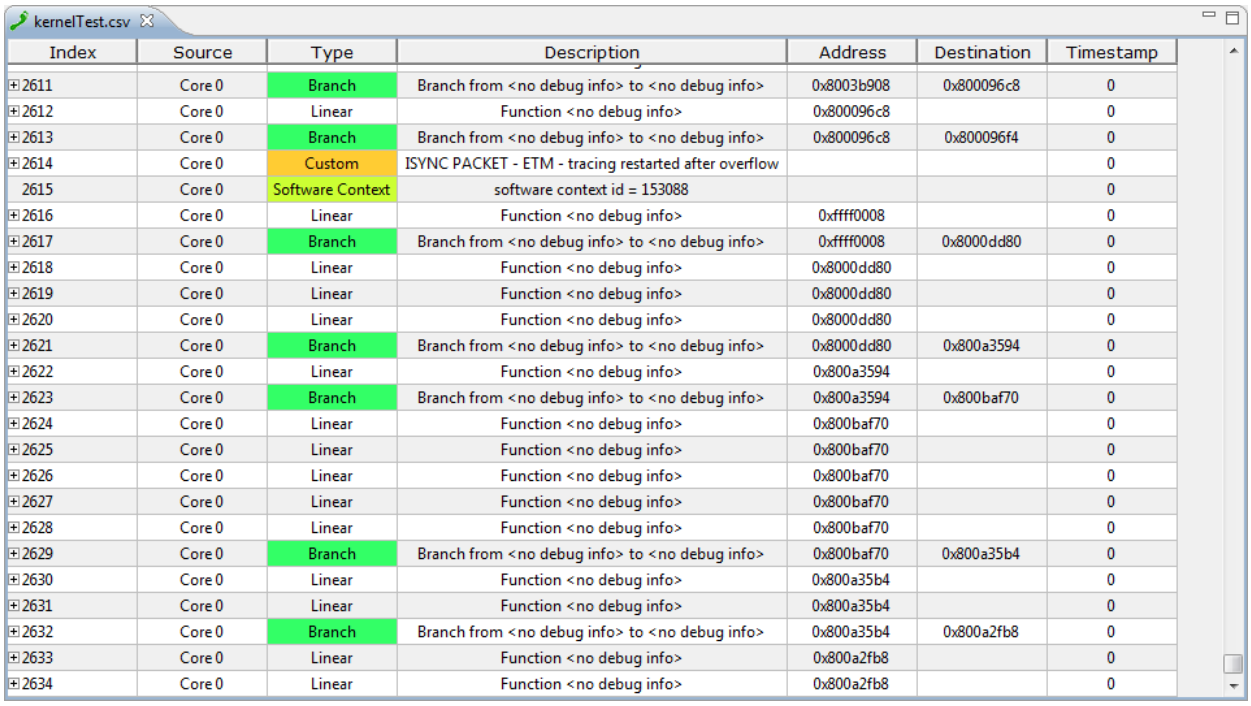
Click **Finish** to end the **Import** wizard. The file is imported and it is displayed in the **Analysis Results** view.
Click the **Trace** link under the **Trace** column in the **Analysis Results** view to view the trace data, as shown in the figure below.

Figure 6. Analysis Results view – kernel space



The trace data file opens in the **Trace** viewer showing the trace results, as shown in the figure below.

Figure 7. Trace viewer – kernel space



kernelTest.csv

Index	Source	Type	Description	Address	Destination	Timestamp
2611	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x8003b908	0x800096c8	0
2612	Core 0	Linear	Function <no debug info>	0x800096c8		0
2613	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x800096c8	0x800096f4	0
2614	Core 0	Custom	ISYNC PACKET - ETM - tracing restarted after overflow			0
2615	Core 0	Software Context	software context id = 153088			0
2616	Core 0	Linear	Function <no debug info>	0xffff0008		0
2617	Core 0	Branch	Branch from <no debug info> to <no debug info>	0xffff0008	0x8000dd80	0
2618	Core 0	Linear	Function <no debug info>	0x8000dd80		0
2619	Core 0	Linear	Function <no debug info>	0x8000dd80		0
2620	Core 0	Linear	Function <no debug info>	0x8000dd80		0
2621	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x8000dd80	0x800a3594	0
2622	Core 0	Linear	Function <no debug info>	0x800a3594		0
2623	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x800a3594	0x800baf70	0
2624	Core 0	Linear	Function <no debug info>	0x800baf70		0
2625	Core 0	Linear	Function <no debug info>	0x800baf70		0
2626	Core 0	Linear	Function <no debug info>	0x800baf70		0
2627	Core 0	Linear	Function <no debug info>	0x800baf70		0
2628	Core 0	Linear	Function <no debug info>	0x800baf70		0
2629	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x800baf70	0x800a35b4	0
2630	Core 0	Linear	Function <no debug info>	0x800a35b4		0
2631	Core 0	Linear	Function <no debug info>	0x800a35b4		0
2632	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x800a35b4	0x800a2fb8	0
2633	Core 0	Linear	Function <no debug info>	0x800a2fb8		0
2634	Core 0	Linear	Function <no debug info>	0x800a2fb8		0

The `satrace` offers the possibility to trace a kernel module using *–m* option. The trace will be started after loading the module in kernel using `insmod` or `modprobe`.
For example, to start a kernel session for a module, `demoModule`, you should run the following command:

```
./linux.armv7.satrace/bin/ls.linux.satrace -K test -m demoModule
```

Use a kernel space/user space application that calls functions defined into the loaded module (`demoModule`), otherwise the trace will be empty. The trace session ends after hitting *CTRL+C*. The collected trace will be stored into an archive placed in the current working directory. It can be decoded and analyzed using CodeWarrior or Trace Complex 1 (TC1) command line utility.

6. Conclusion

The `ls.linux.satrace` executable can be used by Linux user who wants to know the reason for crash or wants to follow the function calls or needs to evaluate the software without any hardware probe. After saving the trace file into an archive that contains all required files for a full decoding, can be viewed in CodeWarrior. The user is benefited from all advantages offered by CW ARMv7. You can have the profiling data code coverage, call tree, performance analysis as well.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

Document Number: AN5001

27 August 2015