Debug and Trace Snapshot File Format

Development Solutions
Blackburn

Document number:	ARM-ECM-0611873	Version:	0.2
Date of Issue:	24/10/2016		
Author:	DSG Engineering		
Authorised by:			

Abstract

This document describes the debug and trace snapshot file formats, used to exchange system state and trace data between debug and trace decode tools.

Keywords

CoreSight, Trace, Debug, Snapshots

Distribution list

Name	Function	Name	Function

Contents

1 ABOUT THIS DOCUMENT

1.1 Change history

- First publication as open standard.

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
CoreSight	ARM Hardware Trace components.
DS-5	ARM Debugger and Trace Decoder based on Eclipse.
OpenCSD	Open source CoreSight trace Decode library – library for decompressing and decoding captured CoreSight trace.
CSAL	CoreSight Access Library – open source library for controlling CoreSight hardware to capture trace data.

2 INTRODUCTION

This document defines a data format that allows debug agents to export and import system state and trace data – this is referred to as a snapshot.

The snapshot can be used to save system state and trace data for later analysis, or can be used to transfer state between environments.

For example the CoreSight Access Library (CSAL) can be used to trace execution on a Linux system and produce a snapshot for analysis within the ARM DS-5 debugger.

2.1 Tools using or generating snapshots

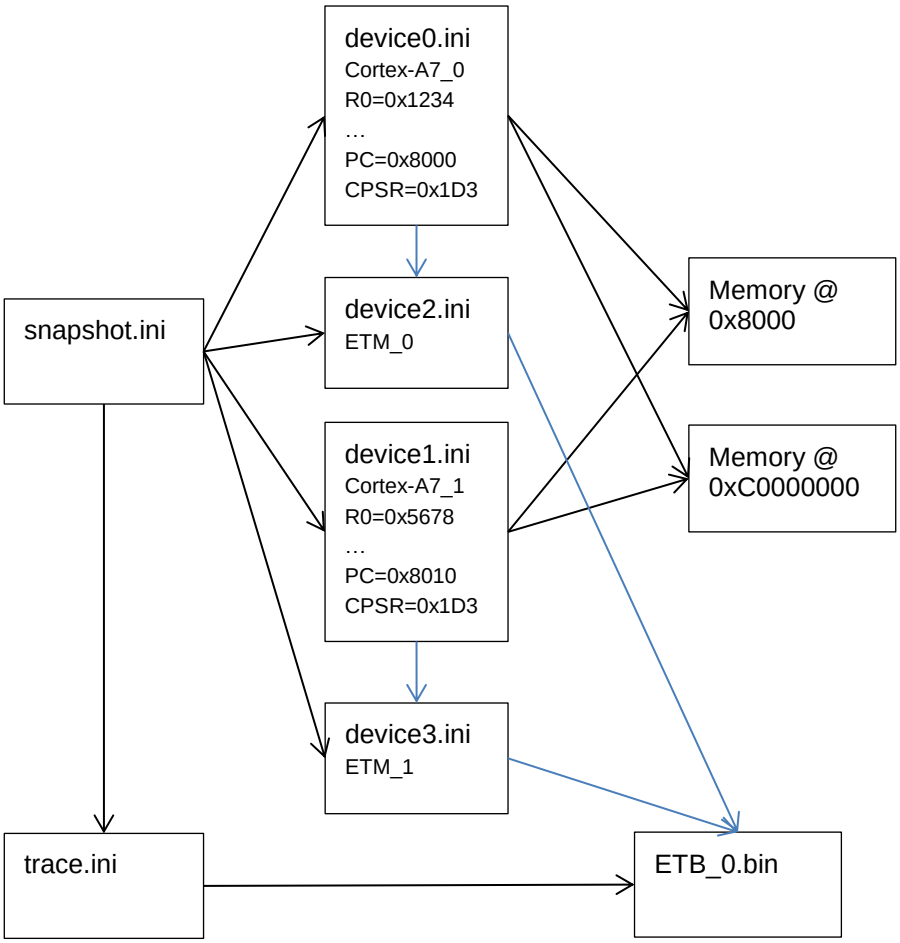
- **ARM DS-5 debugger**: This can generate or consume snapshot data. This will load both a debug view and debug view + trace data snapshot.
- **CSAL**: This generates snapshot data – with both debug view + trace decode metadata present for import into DS-5.
- **OpenCSD**: This can consume snapshot data. This uses the trace decode data only.

3 SNAPSHOT FORMAT

The snapshot consists of a hierarchy of text files containing system data, and associated binary data files. Text files, provided in the `.ini` format, are used to describe the system topology and key register values. The binary files are used for memory and trace data.

A snapshot contains the following files:

- **snapshot.ini**: This mandatory file stands at the head of the hierarchy. It provides the index of available debug devices in the form of device files and the topology of the devices.
- **Device files**: These provide register and memory information for each debug or trace device in the system.
- **trace.ini**: Trace metadata. Provides information on the trace topology and buffers.
- **Trace buffer binary files**: Files containing raw trace data.
- **Memory region binary files**: Files containing device memory data – these may be memory dumped direct from a target system, or offsets for the loadable binary data sections within `elf` files.



The requirements for mandatory data within the various file sections depend on the purpose of the snapshot. There are more detailed requirements for a snapshot that contains trace data and allows that data to be decoded, than for a simple snapshot that represents the halt state of a debug system that is viewable in a debug tool.
The descriptions below will classify keys and data entries as either:-

- Required: required for both debug view snapshots and trace decode snapshots.
- Optional: optional in both debug view snapshots and trace decode snapshots.
- Required for trace: required in trace decode snapshots, optional otherwise.
- Required for debug: required in debug view snapshots, optional otherwise.

3.1 snapshot.ini

This file uses the **`.ini`** file format and consists of the following sections:

- "snapshot" (required)
- "device_list" (required)
- "clusters" (optional)
- "trace" (required for trace)

3.1.1 "snapshot" section

This provides information about the snapshot. This section can contain the following keys:

- "version" (required): The version of the snapshot. The interpretation of the rest of the snapshot depends on the value of this field: readers of the snapshot must check this field first. The only valid value for this is currently "1.0".
- "description" (optional): Description of the contents of this snapshot.

3.1.2 "device_list" section

Each value in this section is a reference to an INI file that defines the state (registers and memory) of a device. Devices may be cores, or trace sources such as ETM or STM. The key names aren't used, but must be unique. It is recommended that they are "device0", "device1", ..."deviceN". The path to the ini file is relative to the snapshot.ini file.

3.1.3 "clusters" section

Each entry in this section defines a cluster. The key is the name of the cluster, the value is a comma separated list of devices in that cluster. If this section is not present, all cores are placed in one cluster.

3.1.4 "trace" section

This section can contain the following keys:

- "metadata" (required): relative path to another ini file that defines the metadata describing the trace buffers and trace source associations

3.1.5 Example

```
[snapshot]
version=1.0
description=Example snapshot

[device_list]
device1=cortex-a15_0.ini
device2=cortex-a15_1.ini
```

```
device3=PTM_0.ini
device4=PTM_1.ini
device5=ITM_0.ini
device6=cortex-a7_0.ini
device7=cortex-a7_1.ini
device8=cortex-a7_2.ini
device9=ETM_0.ini
device10=ETM_1.ini
device11=ETM_2.ini

[clusters]
Cluster 0=cortex-a15_0,cortex-a15_1
Cluster 1=cortex-a7_0,cortex-a7_1,cortex-a7_2

[trace]
metadata=trace.ini
```

3.2 Device files

These are **`.ini`** files with the following sections:

- "**device**": Provides information about the device.
- "**regs**": Provides register values for the device.
- "**dump**": 0 or more sections, each describing a memory region. Memory regions are associated with core type devices, or memory space type devices.

3.2.1 "device" section

This section contains the following keys:

- "**name**" (required): the unique name used to identify this device within the snapshot. Snapshot generators can use any string here – it could be a device name, simple numbering (core_0, core_1, …), functional name or user entered. No meaning should be inferred from the name (for example core type – the "class" and "type" keys provide this). Each device in the snapshot must have a unique name.
- "**class**" (required for trace): the general type of the device. Recognised values for this are:
  - "core": Indicates that this device represents a core
  - "trace_source": Indicates that this device represents a trace source
  - "memory_space": Indicates that this device represents a memory space, e.g. a view of an AHB or AXI bus. A debugger may present the device as an additional address space.
  - Other values are permitted for information purposes
- "**type**" (required for trace): the specific type of the device.
  - For cores this is the name of the core, e.g. "Cortex-A9", "Cortex-A57", "ARM1136JF-S". If the specific core type is not known, the architecture version may be used, e.g. "ARMv7-A", "ARMv8-A". This is not case sensitive. A debugger can use this value to determine which register set to load, what architecture features to support etc.
  - For trace sources this is the name of the trace protocol, e.g. "ETM", "PFT", "ITM", "STM" concatenated with the version of the protocol used, e.g.: "ETM3.3", "ETM4.0", "PFT1.1"
- "**location**" (optional): This describes how the agent that produced the snapshot locates the device. It consists of a comma separated sequence of key:value entries that define a unique path to the device. For an external debugger, this could consist of the DAP number, AP number and AP base address (e.g. dap:2,ap:1,address:0x80010000) of the device. For self-hosted debug, this could consist of the base

address of the device in the core's memory space (e.g. address:0x1200010000). If specified, the location must be unique within the snapshot.

Any other keys in this section are ignored – this allows extra client specific data to be stored.

3.2.2 "regs" section

Each key in this section consists of the register name, with optional extra information in parentheses. The extra information is a comma separated list of keys and values, with ':' separating keys from values. Supported extra information is:

- "size": The size in bits of the register
- "id" or unkeyed: The ID number of the register. May be hex with 0x prefix or decimal

For example:

```
REG_A(77) = 0x1234
REG_B(id:78) = 0x1234
REG_C(id:0x80,size:64) = 0x1234000012340000
REG_D(size:64,0x82) = 0x12340000123400000
```

The values are integers of the appropriate size (32-bits if no size is specified).

For CoreSight components, the register IDs can be converted to address by adding (ID * 4) to the base address of the component.

There are specific requirements for registers that must be present in the core type devices for a debug view snapshot, and in the trace source devices in a trace decode snapshot. These requirements which are core architecture or trace source architecture dependent, are described in detail in section 4 below.

3.2.3 "dump" sections

0 or more dump sections may occur in the device file. The name of the section must start with "dump", but may optionally have a suffix to give each occurrence a unique name. ARM recommends that the unique suffix is used to ensure compatibility between consumers of snapshots. Each defines a region of memory visible from that device. Each section can contain the following keys:

- "file": relative path to the file containing the memory contents
- "space" (optional): address space of the region. Supported values are: "N", "S", "H", "EL1N", "EL1S", "EL2", "EL3", "P", "SP, "NP"
- "address": address of the region
- "length" (optional): length of the region – must be less than or equal to the file size.
- "offset" (optional): offset into the file.

3.2.4 Core Device Example

This example represents a core with the code and data sections of an image visible in the memory

```
[device]
name=cpu_0
class=core
type=Cortex-A7
location=address:0x1200013000

[dump0]
file=ER_RW.bin
space=N
address=0x8000DD90
length=0x00000020
```

```
[dump1]
file=ER_RO.bin
space=N
address=0x80001000
length=0x0000CD90

[dump2]
file=STACK.bin
space=N
address=0x8001D748
length=0x00000800

[dump3]
file=HEAP.bin
space=N
address=0x8000DF48
length=0x0000F7FC

[regs]
R0=0x00000000
R1=0x00000000
R2=0x10001060
R3=0x00000000
R4=0x80011990
R5=0x80011990
R6=0x00000028
R7=0x80010120
R8=0x00000000
R9=0xFFFFFFFF
R10=0x8000D53C
R11=0x00000000
R12=0x00000000
SP=0x8001DF20
LR=0x80001133
PC=0x8000A5F8
CPSR=0x600001D3
D0(id:90,size:64)=0x123456700000000
D1(92,size:64)=0x123456700000000
```

3.2.5 Trace Source Device Example

This example shows a trace source – ETMv4 – with the required registers for decode listed in the regs section.

```
[device]
name=ETM_0
class=trace_source
type=ETM4

[regs]
TRCCONFIGR(0x004)=0x000000C1
TRCTRACEIDR(0x010)=0x00000010
```

```
TRCAUTHSTATUS(0x3EE)=0x000000CC
TRCIDR0(0x078)=0x28000EA1
TRCIDR1(0x079)=0x4100F403
TRCIDR2(0x07A)=0x00000488
TRCIDR8(0x060)=0x00000000
TRCIDR9(0x061)=0x00000000
TRCIDR10(0x062)=0x00000000
TRCIDR11(0x063)=0x00000000
TRCIDR12(0x064)=0x00000000
TRCIDR13(0x065)=0x00000000
```

3.3 Trace metadata

The trace metadata file, typically “trace.ini”, will contain data required to describe the topology of the system and interpret the trace buffers. This file contains the following sections:

- “trace_buffers” (required): Describes the trace buffers present in the snapshot
- Trace buffer metadata: Metadata about each trace buffer
- “core_trace_sources” (required): Defines the associations between cores and trace sources
- “source_buffers” (required): Defines which trace buffer contains data for each trace source

3.3.1 “trace_buffers” section

The “trace_buffers” section describes the trace buffers present in the snapshot. This section has the following keys:

- “buffers” (required): A comma separated list of buffer IDs. Each ID should be unique and must not conflict with other sections in this file (“core_trace_sources”, “source_buffers”). Typically these will be “buffer0”, “buffer1”, .., “bufferN”. The metadata for each buffer will be in a section name with this ID.

3.3.2 Trace buffer metadata sections

Each buffer section can contain the following keys:

- “name” (required): The unique name of this buffer. This may be displayed to the user.
- “file” (required): This entry is a comma separated list of file paths relative to the snapshot.ini that contain the trace data. Multiple files allow the buffer size to exceed the practical file size. Where multiple files are used, the buffer data is the concatenation of the files in the order given, and the buffer size is the sum of the file sizes.
- “format” (required): The format of the data. This will be one of:
    - “coresight”: the data is formatted in 16 byte CoreSight frames
    - “source_data”: the data is unformatted (and only contains data for a single source)
    - Other values for format (e.g. DSTREAM’s raw contents) are supported, but these are not expected to be used.

3.3.3 “core_trace_sources” section

The associations between cores and their trace sources are defined in a section called “core_trace_sources”. Each entry in this section defines the trace source for a core. The key is the name of the core, as defined in the associated core device file, the value is either:

- the name of the trace source, as defined in the associated trace source device file..
- “@” followed by the value of “location” of the trace source, e.g.

```
[core_trace_sources]
cortex-a7_0=@address:0x12308000
```

The association between trace sources and the core attached to the trace source is vital in correctly decoding the trace. As described earlier, “dump” sections describe memory areas associated with a core – these memory areas will normally contain the program executed by that core. Without the appropriate memory area containing the executed opcodes, the trace decode will not be able to proceed.

The core profile – Cortex-A/R or Cortex-M - also has an effect on the correct decoding of trace data, as the appropriate instruction sets and interrupt vectors will be encoded in the trace data.

3.3.4 “source_buffers” section

Where multiple trace buffers are present, the “source_buffers” section defines which buffers the data for each trace source may be found in. Each entry defines the buffers that may contain data for a trace source. The key is the name of the trace source, the value is a comma separated list of buffer names. Where data for a given source may be present in more than one buffer (e.g. a snapshot from system that replicates trace data into an ETB and TPIU), a debugger may present the user with a choice or select one buffer (DS-5 5.21 will select the first).

This section may be omitted if there is only one trace buffer, in which case all trace sources shall use that buffer.

Where a trace source has multiple streams (e.g. ETMv4 has separate instruction and data streams), a snapshot may have the data each stream in different buffers. To specify the buffers, the source may be suffixed with “(stream:N)”, where N is the stream number (i.e. to offset to the base ATB ID). For example with an ETMv4 with name “ETM_0”, this will be “ETM_0(stream:0)” (instruction) and “ETM_0(stream:1)” (data). If a specific buffer is not specified for a stream, then the unqualified source name should be used.

3.3.5 Example

This example shows trace of a two cluster system (2x Cortex-A15 and 3x Cortex-A7) where each cluster traces into a different ETB. There is also an ITM (not associated to any core) that traces into ETB_0

```
[trace_buffers]
buffers=buffer0,buffer1

[buffer0]
name=ETB_0
file=ETB_0.bin
format=coresight

[buffer1]
name=ETB_1
file=ETB_1.bin
format=coresight

[core_trace_sources]
cortex-a7_0=etm_0
cortex-a7_1=etm_1
cortex-a7_2=etm_2
cortex-a15_0=ptm_0
cortex-a15_1=ptm_1

[source_buffers]
etm_0=ETB_0
etm_1=ETB_0
etm_2=ETB_0
ptm_0=ETB_1
ptm_1=ETB_1
itm_0=ETB_0
```

4 REQUIRED CONTENTS OF SNAPSHOTS

A debugger will require certain data to be present in a snapshot in order to load it correctly for a debug view, and other data to correctly decode trace data.

4.1 Cores

When loading a debug view a minimum register set is required for core type devices.

4.1.1 ARMv7-A/R

- The core registers are named R0-R15, CPSR. SP, LR, PC may be used for R13-R15.
- SP, PC and CPSR are required to determine core state.
- If security extensions are supported, SCR is used to determine Secure / Non-secure state. A debugger will assume NS if not SCR is not present.
- If security extensions are supported, valid memory spaces are “S” and “N”. If virtualization is supported, “H” is also valid. If no security or virtualization extensions are supported, no memory space prefix is used.

4.1.2 ARMv8

4.1.2.1 AArch64

- The core registers are named X0-X30, SP, PC. LR may be used for X30.
- CPSR, PC, SP are required to determine core state.
- If security extensions are supported, SCR is used to determine Secure / Non-secure state. A debugger will assume NS if not SCR is not present.
- Valid memory spaces are: “EL3”, “EL2”, “EL1S”, “EL1N”

4.1.2.2 AArch32

- The core registers are named R0-R15, CPSR. SP, LR, PC may be used for R13-R15.
- SP, PC and CPSR are required to determine core state.
- If security extensions are supported, SCR is used to determine Secure / Non-secure state. A debugger will assume NS if not SCR is not present.
- If security extensions supported, valid memory spaces are “S” and “N”. If virtualization is supported, “H” is also valid.

4.1.3 ARMv6-M/ARMv7-M

- The core registers are named R0-R15, xPSR. SP, LR, PC may be used for R13-R15.
- xPSR, PC, SP are required to determine core state.
- Other NVIC registers may be present
- No memory space prefix is used.

4.2 Trace sources

When decoding trace data, a minimum set of registers are required from the CoreSight trace devices, dependent on the trace architecture.

4.2.1 ETMv3

The following registers are required for trace decode

- ETMCR
- ETMCCER
- ETMIDR
- ETMTRACEIDR

4.2.2 PTM

The following registers are required for trace decode

- ETMCR
- ETMCCER
- ETMIDR
- ETMTRACEIDR

4.2.3 ETMv4

The following registers are required for trace decode

- TRCIDR0
- TRCIDR1
- TRCIDR2
- TRCIDR8
- TRCIDR9
- TRCIDR10
- TRCIDR11
- TRCIDR12
- TRCIDR13
- TRCTRACEIDR
- TRCCONFIGR
- TRCAUTHSTATUS

4.2.4 ITM CONTROL_REGISTER

The following registers are required for trace decode

- CONTROL_REGISTER: trace stream ID is taken from bits [22:16]

4.2.5 STM

The following registers are required for trace decode

- STMTCSR: trace stream ID is taken from bits [22:16]