# Coresight - HW Assisted Tracing on ARM

**Author:** Mathieu Poirier <mathieu.poirier@linaro.org>
**Date:** September 11th, 2014

## Introduction

Coresight is an umbrella of technologies allowing for the debugging of ARM based SoC. It includes solutions for JTAG and HW assisted tracing. This document is concerned with the latter.

HW assisted tracing is becoming increasingly useful when dealing with systems that have many SoCs and other components like GPU and DMA engines. ARM has developed a HW assisted tracing solution by means of different components, each being added to a design at synthesis time to cater to specific tracing needs. Components are generally categorised as source, link and sinks and are (usually) discovered using the AMBA bus.

"Sources" generate a compressed stream representing the processor instruction path based on tracing scenarios as configured by users. From there the stream flows through the coresight system (via ATB bus) using links that are connecting the emanating source to a sink(s). Sinks serve as endpoints to the coresight implementation, either storing the compressed stream in a memory buffer or creating an interface to the outside world where data can be transferred to a host without fear of filling up the onboard coresight memory buffer.

At typical coresight system would look like this:

```
 *****************************************************************
**************************** AMBA AXI  ****************************===||
 *****************************************************************    ||
       ^                    ^                            |            ||
       |                    |                            *            **
    0000000    :::::     0000000    :::::    :::::    @@@@@@@    ||||||||||||
    0 CPU 0<-->: C :     0 CPU 0<-->: C :    : C :    @ STM @    || System ||
 |->0000000    : T :  |->0000000    : T :    : T :<--->@@@@@     || Memory ||
 |  #######<-->: I :  |  #######<-->: I :    : I :      @@@<-|   ||||||||||||
 |  # ETM #    :::::  |  # PTM #    :::::    :::::       @   |
 |   #####      ^ ^   |   #####      ^ !      ^ !        .   |   |||||||||
 | |->###       | !   | |->###       | !      | !        .   |   || DAP ||
 | |   #        | !   | |   #        | !      | !        .   |   |||||||||
 | |   .        | !   | |   .        | !      | !        .   |      |  |
 | |   .        | !   | |   .        | !      | !        .   |      |  *
 | |   .        | !   | |   .        | !      | !        .   |      | SWD/
 | |   .        | !   | |   .        | !      | !        .   |      | JTAG
 *****************************************************************<-|
*************************** AMBA Debug APB ************************
 *****************************************************************
  |    .          !         .          !        !        .    |
  |    .          *         .          *        *        .    |
 *****************************************************************
******************** Cross Trigger Matrix (CTM) *******************
 *****************************************************************
  |    .     ^              .                            .    |
  |    *     !              *                            *    |
 *****************************************************************
****************** AMBA Advanced Trace Bus (ATB) ******************
 *****************************************************************
  |          !                        ===============         |
  |          *                         ===== F =====<---------|
  |   :::::::::                         ==== U ====
  |-->:: CTI ::<!!                       === N ===
  |   :::::::::  !                        == N ==
  |    ^         *                        == E ==
  |    !  &&&&&&&&&       IIIIIII         == L ==
  |------>&& ETB &&<......II     I        =======
  |    !  &&&&&&&&&       II     I           .
  |    !                    I     I          .
  |    !                    I REP I<..........
  |    !                    I     I
  |    !!>&&&&&&&&&       II     I           *Source: ARM ltd.
  |------>& TPIU  &<......II    I            DAP = Debug Access Port
          &&&&&&&&&       IIIIIII            ETM = Embedded Trace Macrocell
              ;                              PTM = Program Trace Macrocell
              ;                              CTI = Cross Trigger Interface
              *                              ETB = Embedded Trace Buffer
         To trace port                       TPIU= Trace Port Interface Unit
                                             SWD = Serial Wire Debug
```

While on target configuration of the components is done via the APB bus, all trace data are carried out-of-band on the ATB bus. The CTM provides a way to aggregate and distribute signals between CoreSight components.

The coresight framework provides a central point to represent, configure and manage coresight devices on a platform. This first implementation centers on the basic tracing functionality, enabling components such ETM/PTM, funnel, replicator, TMC, TPIU and ETB. Future work will enable more intricate IP blocks such as STM and CTI.

## Acronyms and Classification

Acronyms:

PTM:
: Program Trace Macrocell

ETM:
: Embedded Trace Macrocell

STM:
: System trace Macrocell

ETB:
: Embedded Trace Buffer

ITM:
: Instrumentation Trace Macrocell

TPIU:
: Trace Port Interface Unit

TMC-ETR:
: Trace Memory Controller, configured as Embedded Trace Router

TMC-ETF:
: Trace Memory Controller, configured as Embedded Trace FIFO

CTI:
: Cross Trigger Interface

Classification:

Source:
: ETMv3.x ETMv4, PTMv1.0, PTMv1.1, STM, STM500, ITM

Link:
: Funnel, replicator (intelligent or not), TMC-ETR

Sinks:
: ETBv1.0, ETB1.1, TPIU, TMC-ETF

Misc:
: CTI

## Device Tree Bindings

See Documentation/devicetree/bindings/arm/coresight.txt for details.

As of this writing drivers for ITM, STMs and CTIs are not provided but are expected to be added as the solution matures.

## Framework and implementation

The coresight framework provides a central point to represent, configure and manage coresight devices on a platform. Any coresight compliant device can register with the framework for as long as they use the right APIs:

**System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\trace\coresight\[linux-master][Documentation][trace][coresight]coresight.rst`, line 146)**

Unknown directive type "c:function".

```
.. c:function:: struct coresight_device *coresight_register(struct coresight_desc *desc);
```

**System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\trace\coresight\[linux-master][Documentation][trace][coresight]coresight.rst`, line 147)**

Unknown directive type "c:function".

```
.. c:function:: void coresight_unregister(struct coresight_device *csdev);
```

The registering function is taking a `struct coresight_desc *desc` and register the device with the core framework. The unregister function takes a reference to a `struct coresight_device *csdev` obtained at registration time.

If everything goes well during the registration process the new devices will show up under /sys/bus/coresight/devices, as showns here for a TC2 platform:

```
root:~# ls /sys/bus/coresight/devices/
replicator  20030000.tpiu    2201c000.ptm  2203c000.etm  2203e000.etm
20010000.etb         20040000.funnel  2201d000.ptm  2203d000.etm
root:~#
```

The functions take a `struct coresight_device`, which looks like this:

```
struct coresight_desc {
        enum coresight_dev_type type;
        struct coresight_dev_subtype subtype;
        const struct coresight_ops *ops;
        struct coresight_platform_data *pdata;
        struct device *dev;
        const struct attribute_group **groups;
};
```

The "coresight_dev_type" identifies what the device is, i.e, source link or sink while the "coresight_dev_subtype" will characterise that type further.

The `struct coresight_ops` is mandatory and will tell the framework how to perform base operations related to the components, each component having a different set of requirement. For that `struct coresight_ops_sink`, `struct coresight_ops_link` and `struct coresight_ops_source` have been provided.

The next field `struct coresight_platform_data *pdata` is acquired by calling `of_get_coresight_platform_data()`, as part of the driver's _probe routine and `struct device *dev` gets the device reference embedded in the `amba_device`:

```
static int etm_probe(struct amba_device *adev, const struct amba_id *id)
{
 ...
 ...
 drvdata->dev = &adev->dev;
 ...
}
```

Specific class of device (source, link, or sink) have generic operations that can be performed on them (see `struct coresight_ops`). The `**groups` is a list of sysfs entries pertaining to operations specific to that component only. "Implementation defined" customisations are expected to be accessed and controlled using those entries.

## Device Naming scheme

The devices that appear on the "coresight" bus were named the same as their parent devices, i.e, the real devices that appears on AMBA bus or the platform bus. Thus the names were based on the Linux Open Firmware layer naming convention, which follows the base physical address of the device followed by the device type. e.g:

```
root:~# ls /sys/bus/coresight/devices/
 20010000.etf  20040000.funnel      20100000.stm     22040000.etm
 22140000.etm  230c0000.funnel      23240000.etm     20030000.tpiu
 20070000.etr  20120000.replicator  220c0000.funnel
 23040000.etm  23140000.etm         23340000.etm
```

However, with the introduction of ACPI support, the names of the real devices are a bit cryptic and non-obvious. Thus, a new naming scheme was introduced to use more generic names based on the type of the device. The following rules apply:

```
1) Devices that are bound to CPUs, are named based on the CPU logical
   number.

   e.g, ETM bound to CPU0 is named "etm0"

2) All other devices follow a pattern, "<device_type_prefix>N", where :

      <device_type_prefix>    - A prefix specific to the type of the device
      N                       - a sequential number assigned based on the order
                                of probing.

      e.g, tmc_etf0, tmc_etr0, funnel0, funnel1
```

Thus, with the new scheme the devices could appear as

```
root:~# ls /sys/bus/coresight/devices/
 etm0     etm1     etm2         etm3  etm4      etm5      funnel0
 funnel1  funnel2  replicator0  stm0  tmc_etf0  tmc_etr0  tpiu0
```

Some of the examples below might refer to old naming scheme and some to the newer scheme, to give a confirmation that what you see on your system is not unexpected. One must use the "names" as they appear on the system under specified locations.

## Topology Representation

Each CoreSight component has a `connections` directory which will contain links to other CoreSight components. This allows the user to explore the trace topology and for larger systems, determine the most appropriate sink for a given source. The connection information can also be used to establish which CTI devices are connected to a given component. This directory contains a `nr_links` attribute detailing the number of links in the directory.

For an ETM source, in this case `etm0` on a Juno platform, a typical arrangement will be:

```
linaro-developer:~# ls - l /sys/bus/coresight/devices/etm0/connections
<file details>  cti_cpu0 -> ../../../23020000.cti/cti_cpu0
<file details>  nr_links
<file details>  out:0 -> ../../../230c0000.funnel/funnel2
```

Following the out port to `funnel2`:

```
linaro-developer:~# ls -l /sys/bus/coresight/devices/funnel2/connections
```

```
<file details> in:0 -> ../../../23040000.etm/etm0
<file details> in:1 -> ../../../23140000.etm/etm3
<file details> in:2 -> ../../../23240000.etm/etm4
<file details> in:3 -> ../../../23340000.etm/etm5
<file details> nr_links
<file details> out:0 -> ../../../20040000.funnel/funnel0
```

And again to `funnel0`:

```
linaro-developer:~# ls -l /sys/bus/coresight/devices/funnel0/connections
<file details> in:0 -> ../../../220c0000.funnel/funnel1
<file details> in:1 -> ../../../230c0000.funnel/funnel2
<file details> nr_links
<file details> out:0 -> ../../../20010000.etf/tmc_etf0
```

Finding the first sink `tmc_etf0`. This can be used to collect data as a sink, or as a link to propagate further along the chain:

```
linaro-developer:~# ls -l /sys/bus/coresight/devices/tmc_etf0/connections
<file details> cti_sys0 -> ../../../20020000.cti/cti_sys0
<file details> in:0 -> ../../../20040000.funnel/funnel0
<file details> nr_links
<file details> out:0 -> ../../../20150000.funnel/funnel4
```

via `funnel4`:

```
linaro-developer:~# ls -l /sys/bus/coresight/devices/funnel4/connections
<file details> in:0 -> ../../../20010000.etf/tmc_etf0
<file details> in:1 -> ../../../20140000.etf/tmc_etf1
<file details> nr_links
<file details> out:0 -> ../../../20120000.replicator/replicator0
```

and a `replicator0`:

```
linaro-developer:~# ls -l /sys/bus/coresight/devices/replicator0/connections
<file details> in:0 -> ../../../20150000.funnel/funnel4
<file details> nr_links
<file details> out:0 -> ../../../20030000.tpiu/tpiu0
<file details> out:1 -> ../../../20070000.etr/tmc_etr0
```

Arriving at the final sink in the chain, `tmc_etr0`:

```
linaro-developer:~# ls -l /sys/bus/coresight/devices/tmc_etr0/connections
<file details> cti_sys0 -> ../../../20020000.cti/cti_sys0
<file details> in:0 -> ../../../20120000.replicator/replicator0
<file details> nr_links
```

As described below, when using sysfs it is sufficient to enable a sink and a source for successful trace. The framework will correctly enable all intermediate links as required.

Note: `cti_sys0` appears in two of the connections lists above. CTIs can connect to multiple devices and are arranged in a star topology via the CTM. See (Documentation/trace/coresight/coresight-ect.rst) [4] for further details. Looking at this device we see 4 connections:

```
linaro-developer:~# ls -l /sys/bus/coresight/devices/cti_sys0/connections
<file details> nr_links
<file details> stm0 -> ../../../20100000.stm/stm0
<file details> tmc_etf0 -> ../../../20010000.etf/tmc_etf0
<file details> tmc_etr0 -> ../../../20070000.etr/tmc_etr0
<file details> tpiu0 -> ../../../20030000.tpiu/tpiu0
```

## How to use the tracer modules

There are two ways to use the Coresight framework:

1. using the perf cmd line tools.
2. interacting directly with the Coresight devices using the sysFS interface.

Preference is given to the former as using the sysFS interface requires a deep understanding of the Coresight HW. The following sections provide details on using both methods.

1. Using the sysFS interface:

Before trace collection can start, a coresight sink needs to be identified. There is no limit on the amount of sinks (nor sources) that can be enabled at any given moment. As a generic operation, all device pertaining to the sink class will have an "active" entry in sysfs:

```
root:/sys/bus/coresight/devices# ls
replicator  20030000.tpiu    2201c000.ptm  2203c000.etm  2203e000.etm
20010000.etb         20040000.funnel  2201d000.ptm  2203d000.etm
root:/sys/bus/coresight/devices# ls 20010000.etb
enable_sink  status  trigger_cntr
root:/sys/bus/coresight/devices# echo 1 > 20010000.etb/enable_sink
root:/sys/bus/coresight/devices# cat 20010000.etb/enable_sink
1
root:/sys/bus/coresight/devices#
```

At boot time the current etm3x driver will configure the first address comparator with "_stext" and "_etext", essentially tracing any instruction that falls within that range. As such "enabling" a source will immediately trigger a trace capture:

```
root:/sys/bus/coresight/devices# echo 1 > 2201c000.ptm/enable_source
root:/sys/bus/coresight/devices# cat 2201c000.ptm/enable_source
1
root:/sys/bus/coresight/devices# cat 20010000.etb/status
Depth:          0x2000
Status:         0x1
RAM read ptr:   0x0
RAM wrt ptr:    0x19d3   <----- The write pointer is moving
Trigger cnt:    0x0
Control:        0x1
Flush status:   0x0
Flush ctrl:     0x2001
root:/sys/bus/coresight/devices#
```

Trace collection is stopped the same way:

```
root:/sys/bus/coresight/devices# echo 0 > 2201c000.ptm/enable_source
root:/sys/bus/coresight/devices#
```

The content of the ETB buffer can be harvested directly from /dev:

```
root:/sys/bus/coresight/devices# dd if=/dev/20010000.etb \
of=~/cstrace.bin
64+0 records in
64+0 records out
32768 bytes (33 kB) copied, 0.00125258 s, 26.2 MB/s
root:/sys/bus/coresight/devices#
```

The file cstrace.bin can be decompressed using "ptm2human", DS-5 or Trace32.

Following is a DS-5 output of an experimental loop that increments a variable up to a certain value. The example is simple and yet provides a glimpse of the wealth of possibilities that coresight provides.

```
Info                                    Tracing enabled
Instruction     106378866       0x8026B53C      E52DE004        false   PUSH     {lr}
Instruction     0       0x8026B540      E24DD00C        false   SUB      sp,sp,#0xc
Instruction     0       0x8026B544      E3A03000        false   MOV      r3,#0
Instruction     0       0x8026B548      E58D3004        false   STR      r3,[sp,#4]
Instruction     0       0x8026B54C      E59D3004        false   LDR      r3,[sp,#4]
Instruction     0       0x8026B550      E3530004        false   CMP      r3,#4
Instruction     0       0x8026B554      E2833001        false   ADD      r3,r3,#1
Instruction     0       0x8026B558      E58D3004        false   STR      r3,[sp,#4]
Instruction     0       0x8026B55C      DAFFFFFA        true    BLE      {pc}-0x10 ; 0x8026b54c
Timestamp                                       Timestamp: 17106715833
Instruction     319     0x8026B54C      E59D3004        false   LDR      r3,[sp,#4]
Instruction     0       0x8026B550      E3530004        false   CMP      r3,#4
Instruction     0       0x8026B554      E2833001        false   ADD      r3,r3,#1
Instruction     0       0x8026B558      E58D3004        false   STR      r3,[sp,#4]
Instruction     0       0x8026B55C      DAFFFFFA        true    BLE      {pc}-0x10 ; 0x8026b54c
Instruction     9       0x8026B54C      E59D3004        false   LDR      r3,[sp,#4]
Instruction     0       0x8026B550      E3530004        false   CMP      r3,#4
Instruction     0       0x8026B554      E2833001        false   ADD      r3,r3,#1
Instruction     0       0x8026B558      E58D3004        false   STR      r3,[sp,#4]
Instruction     0       0x8026B55C      DAFFFFFA        true    BLE      {pc}-0x10 ; 0x8026b54c
Instruction     7       0x8026B54C      E59D3004        false   LDR      r3,[sp,#4]
Instruction     0       0x8026B550      E3530004        false   CMP      r3,#4
Instruction     0       0x8026B554      E2833001        false   ADD      r3,r3,#1
Instruction     0       0x8026B558      E58D3004        false   STR      r3,[sp,#4]
Instruction     0       0x8026B55C      DAFFFFFA        true    BLE      {pc}-0x10 ; 0x8026b54c
Instruction     7       0x8026B54C      E59D3004        false   LDR      r3,[sp,#4]
Instruction     0       0x8026B550      E3530004        false   CMP      r3,#4
Instruction     0       0x8026B554      E2833001        false   ADD      r3,r3,#1
Instruction     0       0x8026B558      E58D3004        false   STR      r3,[sp,#4]
Instruction     0       0x8026B55C      DAFFFFFA        true    BLE      {pc}-0x10 ; 0x8026b54c
Instruction     10      0x8026B54C      E59D3004        false   LDR      r3,[sp,#4]
Instruction     0       0x8026B550      E3530004        false   CMP      r3,#4
Instruction     0       0x8026B554      E2833001        false   ADD      r3,r3,#1
Instruction     0       0x8026B558      E58D3004        false   STR      r3,[sp,#4]
Instruction     0       0x8026B55C      DAFFFFFA        true    BLE      {pc}-0x10 ; 0x8026b54c
Instruction     6       0x8026B560      EE1D3F30        false   MRC      p15,#0x0,r3,c13,c0,#1
Instruction     0       0x8026B564      E1A0100D        false   MOV      r1,sp
Instruction     0       0x8026B568      E3C12D7F        false   BIC      r2,r1,#0x1fc0
Instruction     0       0x8026B56C      E3C2203F        false   BIC      r2,r2,#0x3f
Instruction     0       0x8026B570      E59D1004        false   LDR      r1,[sp,#4]
Instruction     0       0x8026B574      E59F0010        false   LDR      r0,[pc,#16] ; [0x8026B58C] = 0x805
Instruction     0       0x8026B578      E592200C        false   LDR      r2,[r2,#0xc]
Instruction     0       0x8026B57C      E59221D0        false   LDR      r2,[r2,#0x1d0]
Instruction     0       0x8026B580      EB07A4CF        true    BL       {pc}+0x1e9344 ; 0x804548c4
Info                                    Tracing enabled
Instruction     13570831        0x8026B584      E28DD00C        false   ADD      sp,sp,#0xc
Instruction     0       0x8026B588      E8BD8000        true    LDM      sp!,{pc}
Timestamp                                       Timestamp: 17107041535
```

2. Using perf framework:

Coresight tracers are represented using the Perf framework's Performance Monitoring Unit (PMU) abstraction. As such the perf framework takes charge of controlling when tracing gets enabled based on when the process of interest is scheduled. When configured in a system, Coresight PMUs will be listed when queried by the perf command line tool:

linaro@linaro-nano:~$ ./perf list pmu

List of pre-defined events (to be used in -e):

cs_etm// [Kernel PMU event]

linaro@linaro-nano:~$

Regardless of the number of tracers available in a system (usually equal to the amount of processor cores), the "cs_etm" PMU will be listed only once.

A Coresight PMU works the same way as any other PMU, i.e the name of the PMU is listed along with configuration options within forward slashes '/'. Since a Coresight system will typically have more than one sink, the name of the sink to work with needs to be specified as an event option. On newer kernels the available sinks are listed in sysFS under ($SYSFS)/bus/event_source/devices/cs_etm/sinks/:

```
root@localhost:/sys/bus/event_source/devices/cs_etm/sinks# ls
tmc_etf0  tmc_etr0  tpiu0
```

On older kernels, this may need to be found from the list of coresight devices, available under ($SYSFS)/bus/coresight/devices/:

```
root:~# ls /sys/bus/coresight/devices/
 etm0     etm1     etm2         etm3  etm4      etm5      funnel0
 funnel1  funnel2  replicator0  stm0  tmc_etf0  tmc_etr0  tpiu0
root@linaro-nano:~# perf record -e cs_etm/@tmc_etr0/u --per-thread program
```

As mentioned above in section "Device Naming scheme", the names of the devices could look different from what is used in the example above. One must use the device names as it appears under the sysFS.

The syntax within the forward slashes '/' is important. The '@' character tells the parser that a sink is about to be specified and that this is the sink to use for the trace session.

More information on the above and other example on how to use Coresight with the perf tools can be found in the "HOWTO.md" file of the openCSD gitHub repository [3].

2.1) AutoFDO analysis using the perf tools:

perf can be used to record and analyze trace of programs.

Execution can be recorded using 'perf record' with the cs_etm event, specifying the name of the sink to record to, e.g:

```
perf record -e cs_etm/@tmc_etr0/u --per-thread
```

The 'perf report' and 'perf script' commands can be used to analyze execution, synthesizing instruction and branch events from the instruction trace. 'perf inject' can be used to replace the trace data with the synthesized events. The --itrace option controls the type and frequency of synthesized events (see perf documentation).

Note that only 64-bit programs are currently supported - further work is required to support instruction decode of 32-bit Arm programs.

2.2) Tracing PID

The kernel can be built to write the PID value into the PE ContextID registers. For a kernel running at EL1, the PID is stored in CONTEXTIDR_EL1. A PE may implement Arm Virtualization Host Extensions (VHE), which the kernel can run at EL2 as a virtualisation host; in this case, the PID value is stored in CONTEXTIDR_EL2.

perf provides PMU formats that program the ETM to insert these values into the trace data; the PMU formats are defined as below:

"contextid1": Available on both EL1 kernel and EL2 kernel. When the
    kernel is running at EL1, "contextid1" enables the PID tracing; when the kernel is running at EL2, this enables tracing the PID of guest applications.

"contextid2": Only usable when the kernel is running at EL2. When
    selected, enables PID tracing on EL2 kernel.

"contextid": Will be an alias for the option that enables PID
    tracing. I.e, contextid == contextid1, on EL1 kernel. contextid == contextid2, on EL2 kernel.

perf will always enable PID tracing at the relevant EL, this is accomplished by automatically enable the "contextid" config - but for EL2 it is possible to make specific adjustments using configs "contextid1" and "contextid2", E.g. if a user wants to trace PIDs for both host and guest, the two configs "contextid1" and "contextid2" can be set at the same time:

perf record -e cs_etm/contextid1,contextid2/u -- vm

## Generating coverage files for Feedback Directed Optimization: AutoFDO

'perf inject' accepts the --itrace option in which case tracing data is removed and replaced with the synthesized events. e.g.

```
perf inject --itrace --strip -i perf.data -o perf.data.new
```

Below is an example of using ARM ETM for autoFDO. It requires autofdo (https://github.com/google/autofdo) and gcc version 5. The bubble sort example is from the AutoFDO tutorial (https://gcc.gnu.org/wiki/AutoFDO/Tutorial).

```
$ gcc-5 -O3 sort.c -o sort
$ taskset -c 2 ./sort
```

```
Bubble sorting array of 30000 elements
5910 ms

$ perf record -e cs_etm/@tmc_etr0/u --per-thread taskset -c 2 ./sort
Bubble sorting array of 30000 elements
12543 ms
[ perf record: Woken up 35 times to write data ]
[ perf record: Captured and wrote 69.640 MB perf.data ]

$ perf inject -i perf.data -o inj.data --itrace=il64 --strip
$ create_gcov --binary=./sort --profile=inj.data --gcov=sort.gcov -gcov_version=1
$ gcc-5 -O3 -fauto-profile=sort.gcov sort.c -o sort_autofdo
$ taskset -c 2 ./sort_autofdo
Bubble sorting array of 30000 elements
5806 ms
```

## How to use the STM module

Using the System Trace Macrocell module is the same as the tracers - the only difference is that clients are driving the trace capture rather than the program flow through the code.

As with any other CoreSight component, specifics about the STM tracer can be found in sysfs with more information on each entry being found in [1]:

```
root@genericarmv8:~# ls /sys/bus/coresight/devices/stm0
enable_source   hwevent_select  port_enable     subsystem       uevent
hwevent_enable  mgmt            port_select     traceid
root@genericarmv8:~#
```

Like any other source a sink needs to be identified and the STM enabled before being used:

```
root@genericarmv8:~# echo 1 > /sys/bus/coresight/devices/tmc_etf0/enable_sink
root@genericarmv8:~# echo 1 > /sys/bus/coresight/devices/stm0/enable_source
```

From there user space applications can request and use channels using the devfs interface provided for that purpose by the generic STM API:

```
root@genericarmv8:~# ls -l /dev/stm0
crw-------    1 root     root       10,  61 Jan  3 18:11 /dev/stm0
root@genericarmv8:~#
```

Details on how to use the generic STM API can be found here: - Documentation/trace/stm.rst [2].

## The CTI & CTM Modules

The CTI (Cross Trigger Interface) provides a set of trigger signals between individual CTIs and components, and can propagate these between all CTIs via channels on the CTM (Cross Trigger Matrix).

A separate documentation file is provided to explain the use of these devices. (Documentation/trace/coresight/coresight-ect.rst) [4].

## CoreSight System Configuration

CoreSight components can be complex devices with many programming options. Furthermore, components can be programmed to interact with each other across the complete system.

A CoreSight System Configuration manager is provided to allow these complex programming configurations to be selected and used easily from perf and sysfs.

See the separate document for further information. (Documentation/trace/coresight/coresight-config.rst) [5].

[1] Documentation/ABI/testing/sysfs-bus-coresight-devices-stm

[2] Documentation/trace/stm.rst

[3] https://github.com/Linaro/perf-opencsd

[4] (*1*,*2*) Documentation/trace/coresight/coresight-ect.rst

[5] Documentation/trace/coresight/coresight-config.rst