SAN19-410 CoreSight

New Features and Ongoing Development

Linaro connect
San Diego 2019

New Features and Ongoing Work

CoreSight Power Management

ACPI support and new naming convention

Support for CPU wide trace scenarios

Enhancement to the perf tools

CTI Drivers

Complex Configuration

CoreSight Power Management

Architecturally done in firmware (PMIC)

Also requires HW support

Problem is dependent on how Perf events are scheduled

`$ perf record -e cs_etm/etr0/ --per-thread $MYAPP` → no problem

`$ perf record -e cs_etm/etr0/ --all-cpus $TIMEVAL` → will hang your board

CoreSight Power Management

Problem currently addressed by Andrew Murray

Only minor cosmetic problems to his last posting (tiny.cc/i917bz)

I tested his work on the Dragonboard 410c platform

Flexible solution → DT binding and new kernel command line option

ACPI and New Naming Convention

The CoreSight framework will now work from ACPI firmware

Arm has published ACPI binding for CoreSight (tiny.cc/wm27bz)

Suzuki Poulose added support that complies with the specification

New naming convention as an interesting side effect...

ACPI and New Naming Convention

Before:

```
linaro@juno:~$ 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
```

After:

```
linaro@juno:~$ ls /sys/bus/coresight/devices/
etm0  etm1  etm2  etm3  etm4  etm5  funnel0  funnel1  funnel2  replicator0  stm0
tmc_etf0  tmc_etr0  tpiu0
```

ACPI and New Naming Convention

```
linaro@juno:~$ ls -l /sys/bus/coresight/devices/

lrwxrwxrwx 1 root root 0 Sep  4 20:50 etm0 -> ../../../devices/platform/3040000.etm/etm0
lrwxrwxrwx 1 root root 0 Sep  4 20:50 etm1 -> ../../../devices/platform/2040000.etm/etm1
lrwxrwxrwx 1 root root 0 Sep  4 20:50 etm2 -> ../../../devices/platform/2140000.etm/etm2
lrwxrwxrwx 1 root root 0 Sep  4 20:50 etm3 -> ../../../devices/platform/3140000.etm/etm3
lrwxrwxrwx 1 root root 0 Sep  4 20:50 etm4 -> ../../../devices/platform/3240000.etm/etm4
lrwxrwxrwx 1 root root 0 Sep  4 20:50 etm5 -> ../../../devices/platform/3340000.etm/etm5
…
…
```

Support for CPU wide trace scenarios

It is now possible to do CPU wide tracing

```
$ perf record -e cs_etm/etr0/ -C 1,3 $MYAPP
$ perf report -e cs_etm/etr0/ -a $MYAPP
```

Traces will be recorded for as long as $MYAPP is running

Perf Tools Enhancement

```
root@debian:~# perf record -e cs_etm/@tmc_etr0/ --per-thread -- uname

root@debian:~# perf script -F,+flags,+insn,+insnlen

[...]

   uname   845          1          branches:   call          ffffa9473748 _dl_sysdep_start+0x278 (/usr/lib/aarch64-linux-gnu/ld-2.28.so) ilen: 4 insn: 0c 04 00 94
   uname   845          1          branches:   syscall       ffffa9474780 brk+0x8 (/usr/lib/aarch64-linux-gnu/ld-2.28.so) ilen: 4 insn: 01 00 00 d4
   uname   845          1          branches:   jmp           ffff000010082420 vectors+0x420 ([kernel.kallsyms]) ilen: 4 insn: 58 04 00 14
   uname   845          1          branches:   jcc           ffff00001008364c el0_sync+0xcc ([kernel.kallsyms]) ilen: 4 insn: b3 00 a8 36
   uname   845          1          branches:   jmp           ffff000010083660 el0_sync+0xe0 ([kernel.kallsyms]) ilen: 4 insn: 0b 00 00 14
   uname   845          1          branches:   jcc           ffff0000100836c4 el0_sync+0x144 ([kernel.kallsyms]) ilen: 4 insn: e0 57 00 54
   uname   845          1          branches:   call          ffff0000100841c4 el0_svc+0x4 ([kernel.kallsyms]) ilen: 4 insn: d5 4a 00 94
   uname   845          1          branches:   call          ffff000010096d3c el0_svc_handler+0x24 ([kernel.kallsyms]) ilen: 4 insn: 9d ff ff 97
   uname   845          1          branches:   jcc           ffff000010096bec el0_svc_common.constprop.0+0x3c ([kernel.kallsyms]) ilen: 4 insn: a9 02 00 54
   uname   845          1          branches:   call          ffff000010096c5c el0_svc_common.constprop.0+0xac ([kernel.kallsyms]) ilen: 4 insn: 20 00 3f d6
   uname   845          1          branches:   call          ffff00001026b45c __arm64_sys_brk+0x4c ([kernel.kallsyms]) ilen: 4 insn: 8f 37 29 94
   uname   845          1          branches:   call          ffff000010cb92ac down_write_killable+0x14 ([kernel.kallsyms]) ilen: 4 insn: 9d 89 ff 97
   uname   845          1          branches:   return        ffff000010c9b93c __ll_sc___cmpxchg_case_acq_64+0x1c ([kernel.kallsyms]) ilen: 4 insn: c0 03 5f d6
   uname   845          1          branches:   return        ffff000010cb92cc down_write_killable+0x34 ([kernel.kallsyms]) ilen: 4 insn: c0 03 5f d6
   uname   845          1          branches:   jcc           ffff00001026b470 __arm64_sys_brk+0x60 ([kernel.kallsyms]) ilen: 4 insn: e3 05 00 54
   uname   845          1          branches:   call          ffff00001026b534 __arm64_sys_brk+0x124 ([kernel.kallsyms]) ilen: 4 insn: 5b 32 fb 97
   uname   845          1          branches:   call          ffff000010137eb8 up_write+0x18 ([kernel.kallsyms]) ilen: 4 insn: 38 8d 2d 94
   uname   845          1          branches:   return        ffff000010c9b3b8 __ll_sc_arch_atomic64_fetch_add_release+0x20 ([kernel.kallsyms]) ilen: 4 insn: c0 03 5f d6
```

“+flags” display the branch types

“+insn” and “+insnlen” for length and bytecode

This slide was taken directly from SAN19-223

More Perf Tools Enhancement

More cool features to come from Leo Yan...

Initial foundation for CoreSight testing

Support for callchain in perf report

Recap of CTM and CTI hardware

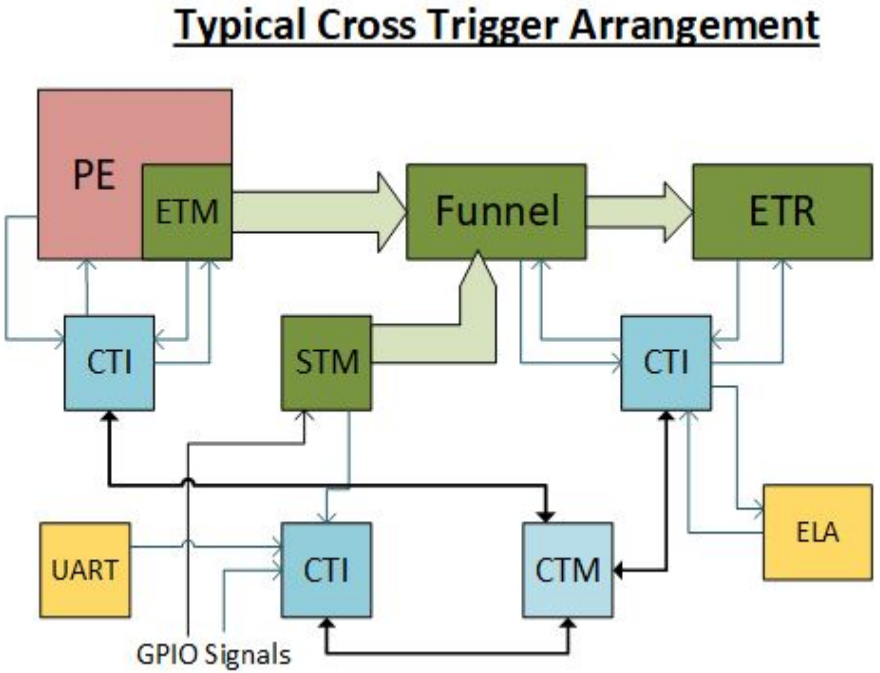
CTI devices connect to cores, ETM, CoreSight ETF, ETR, ETB, TPIU, STM.

Also connect to any hardware capable of sourcing or sinking a hardware signal.

Connections to v8 architecture PE & ETM are architecturally defined.

All other connections are implementation defined at hardware design time.

Signals (triggers) connect between devices and CTI, channels connect between CTIs via the CTM



CTI Driver Status

Initial CTI driver has had a few iterations on the CoreSight mailing list.
(coresight@lists.linaro.org)

Will soon be presented to wider audience for upstreaming. (provisionally aimed at 5.4 kernel revision)

Device tree representations of the CTI will form part of the first driver release. ACPI representations to follow.

CTI driver - Device tree representation (1).

Trigger interconnections between the CTI and other devices can be complex.

The device tree information encapsulates all that information within the CTI node definition.

This allows the trigger definitions to be encapsulated at the point they are needed - the device that is programmed to used them, and avoids large re-writing of other component definitions.

A minimal definition is allowed, that will use the hardware ID register to declare channels and triggers according to maximum values.

```
/* Minimum CTI - no explicit triggers defined */
cti@810000 {
        compatible = "arm,coresight-cti", "arm,primecell";
        reg = <0x810000 0x1000>;

        clocks = <&rpmcc RPM_QDSS_CLK>;
        clock-names = "apb_pclk";
};
```

Device tree - v8 cores and CTI

Where the CTI is connected to a v8 core, and an optional ETM, then the V8 architecture defines the layout of the triggers and interconnections between the CTI and devices.

In this case a short form in the device tree can fully define the CTI.

In this case the “arm,cti-v8-arch” declares that this is a v8 attached CTI. The cpu definition is required, and if an ETM is present then this is declared as the “arm,cs-dev-assoc” device.

```
/* CTI - CPU-0 */
cti@858000 {
        compatible = "arm,coresight-cti", "arm,primecell";
        reg = <0x858000 0x1000>;

        clocks = <&rpmcc RPM_QDSS_CLK>;
        clock-names = "apb_pclk";

        arm,cti-v8-arch;
        cpu = <&CPU0>;
        arm,cs-dev-assoc = <&etm0>;
};
```

Device tree - custom CTI declarations (1)

All other CTI declarations will completely define the connections between the CTI and any other device.

For CoreSight devices the “arm,cs-dev-assoc” parameter is used. This establishes that the association is between CoreSight devices - the driver framework will then ensure that the CTI is enabled if the associated device is enabled.

Sub-nodes declare the ‘in’ and ‘out’ triggers that exist between the CTI and a given device.

Here we see the trigger interconnect between ETR, ETF, STM and one of the system CTIs on the Juno platform.

Type values for commonly used CoreSight component signals are used. Generic types can also be declared.

```
cti@20020000 { /* sys_cti_0 */
[ removed some declarations ]
          trig-conns@0 {
                    arm,trig-in-sigs=<2 3>;
                    arm,trig-in-types=<SNK_FULL SNK_ACQCOMP>;
                    arm,trig-out-sigs=<0 1>;
                    arm,trig-out-types=<SNK_FLUSHIN SNK_TRIGIN>;
                    arm,cs-dev-assoc = <&etr_sys>;
          };
          trig-conns@1 {
                    arm,trig-in-sigs=<0 1>;
                    arm,trig-in-types=<SNK_FULL SNK_ACQCOMP>;
                    arm,trig-out-sigs=<7 6>;
                    arm,trig-out-types=<SNK_FLUSHIN SNK_TRIGIN>;
                    arm,cs-dev-assoc = <&etf_sys0>;
          };
          trig-conns@2 {
                    arm,trig-in-sigs=<4 5 6 7>;
                    arm,trig-in-types=<STM_TOUT_SPTE STM_TOUT_SW STM_TOUT_HETE
STM_ASYNCOUT>;
                    arm,trig-out-sigs=<4 5>;
                    arm,trig-out-types=<STM_HWEVENT STM_HWEVENT>;
                    arm,cs-dev-assoc = <&stm_sys>;
          };
}
```

Device tree - custom CTI declarations (2)

Here we see a CTI on the Juno platform connected to a number of non-CoreSight devices.

In this case we supply the connection with a name “arm,trig-conn-name” rather than a CoreSight association.

```
cti@20110000 { /* sys_cti_1 */
[ removed some declarations ]
        trig-conns@0 {
                arm,trig-in-sigs=<0>;
                arm,trig-in-types=<GEN_INTREQ>;
                arm,trig-out-sigs=<0>;
                arm,trig-out-types=<GEN_HALTREQ>;
                arm,trig-conn-name = "sys_profiler";
        };
        trig-conns@1 {
                arm,trig-out-sigs=<2 3>;
                arm,trig-out-types=<GEN_HALTREQ GEN_RESTARTREQ>;
                arm,trig-conn-name = "watchdog";
        };
        trig-conns@2 {
                arm,trig-out-sigs=<1 6>;
                arm,trig-out-types=<GEN_HALTREQ GEN_RESTARTREQ>;
                arm,trig-conn-name = "g_counter";
        };
};
```

CTI driver - connection representation in sysfs

The previously described device tree connections are displayed to the user as sub-directories in sysfs.

Individual trigger connection signal information appears in the **`./triggers<N>`** sub-directory. Common connection info in the **`./connections`** sub-directory.

The **`./triggers<N>`** directory contains information about the trigger signals and types for a connection.

Contains the number of connections along with sysfs links to any connected component that is a CoreSight component.

```
>$ ls /sys/bus/coresight/devices
cti_cpu0  cti_cpu2  cti_sys0  etm0  etm2  funnel0  replicator0  tmc_etr0
cti_cpu1  cti_cpu3  cti_sys1  etm1  etm3  funnel1  tmc_etf0     tpiu0

>$ ls /sys/bus/coresight/devices/cti_cpu0
channels  connections  ctmid  enable  mgmt regs triggers0 triggers1

>$ ls ./cti_cpu0/triggers0/
in_signals  in_types  name  out_signals  out_types
>$ cat ./cti_cpu0/triggers0/name
cpu0
>$ cat ./cti_cpu0/triggers0/out_signals
0-2
>$ cat ./cti_cpu0/triggers0/out_types
pe_edbgreq pe_dbgrestart pe_ctiirq
>$ cat ./cti_cpu0/triggers0/in_signals
0-1
>$ cat ./cti_cpu0/triggers0/in_types
pe_dbgtrigger pe_pmuirq

>$ ls -l ./cti_cpu0/connections/
nr_cons  trigout_filtered etm0 -> ../../../85c000.etm/etm0
```

CTI driver - programming in sysfs

The **`./channels`** sub-directory provides a programming API for enabling triggers to be connected to channels, and a way for channel to be activated or cleared.

Commands use a “channel_index trigger_index” format. e.g. attaches trigout(1) to channel(0).

e.g. then sets channel(0) - which will activate any signal attached to channel(0), including the trigout(1) we attached above.

Using this API, only the trigger signals that are declared in the device tree can be programmed.

Some trigger signals - such as PE_DBGREQ can be filtered to prevent accidental activation.

```
>$ ls ./cti_sys0/channels/
chan_clear  gate_disable  list_gate_enable  show_chan_sel       trigin_attach
chan_pulse  gate_enable   list_inuse        show_chan_xtrigs    trigin_detach
chan_set    list_free     reset_xtrigs      trig_filter_enable  trigout_attach
trigout_detach
```

```
>$ echo 0 1 > ./cti_sys0/channels/trigout_attach
```

```
echo 0 > ./cti_sys0/channels/chan_set
```

Complex Configuration - the problem

ETM strobing for AutoFDO and frequent timestamp programming for cpu wide trace both require multiple registers / resources used in ETM.

These are currently hardwired in the upstream driver for multi-cpu trace, or as an out of tree patch for autoFDO.

Complex config addresses these issues - and creates a methodology to define and implement new configuration at run time.

Equivalent sysfs commands for frequent timestamps (uses a counter to set the rate at which timestamps appear in the trace stream).

```
echo 0 > cntr_idx      ; select counter 0
echo 1 > cntr_val      ; set initial value
echo 1 > cntrldvr      ; set the reload value
echo 0x10001 > cntr_ctrl      ; set control value to reload
echo 2 > res_idx       ; select a resource selector 2
echo 0x20000 > res_ctrl ; res selector 2 picks counter 0
echo 0x2 > ts_ctrl     ; set ts control event to res selector 2.
echo 0x800 > mode      ; set enable timestamp bit.
```

Complex Configuration - Device Features

Features are a class of configuration specific to a device - such as an ETM or CTI.

This can be a simple intrinsic hardware feature - such as enabling cycle counts, or something more complex such as the frequent timestamps.

Features will consume resources - the definition of a feature will declare the resources needed. Each driver for a device type will convert feature definitions into register programming.

Some features will be built in to the driver - but additional features can be defined and loaded at runtime (described later).

Features are additive on a device, so multiple features can be enabled subject to resource availability.

Complex Configuration - Board configuration

Board configuration is specific to a particular system.

A board configuration will define the features used on each device in the system. This can be done individually or by device type - e.g. set freq_timestamps on all ETMs.

Board configuration will be loaded at runtime, and only a single board configuration can be active at any given time.

A board configuration can define new features for a device type.

An example of a board configuration could be the programming of CTIs and ETMs to allow a CTI signal to halt trace collection - for example on a kernel panic or other event.

Complex Configuration - CoreSight config driver

A driver will be provided to manage the runtime update of features and board configurations, and the selection of features and board configurations prior to starting trace sessions.

Runtime updates of features will use the **`configfs`** interface. The driver will then take the configuration and update any features defined with the relevant device types.

The config driver will allow selection of features for all instances of a given device type - e.g. set freq_timestamps on all ETMs, and selection of the active board configuration.

Selection APIs provided at the sysfs level, and as programmatic API for tools such as perf.

Complex Configuration - adding new features.

Text files will be used to define new features and board configurations for use at runtime.

A tool will convert these to the binary format needed by the config driver and load them using the configfs interface.

Resources are device type specific - ETM is the most complex, CTI is simpler.

Feature definitions can contain parameterised values to allow adjustment at runtime.

Example syntax for the freq_timestamps feature:

```
[feature]
Name = freq_timestamps
Type = ETMV4
P(0) = 1    ; input parameter 0, default 1
P(1) = 1    ; input parameter 1, default 1
Res(1) = timestamp      ; declare timestamp hw
Res(2) = res_selector   ; need a res selector
Res(3) = counter        ; need a counter
Set Res(3) cntr_val = P(0)
Set Res(3) cntrldvr = P(1)
Set Res(3) cntr_ctrl = 0x10001
Set Res(2) res_ctrl = Res(3)
Set Res(1) ts_ctrl = Res(2)
```

Complex Configuration - development roadmap

Phase 1 : Update ETMv4 to define implicit features and selection APIs. Add builtins for freq_timestamps, etm_strobe and trace_all (the current default setting for the driver)

This phase will provide initial support required to get AutoFDO working and required features for perf cpu wide trace.

Phase 2: Add in config driver to manage feature selection by device type.

Phase 3: Create tooling and update config driver for runtime feature updates and board config control.

Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org

Linaro connect
San Diego 2019