Coresight ETM tracing on dragonboard 410c

Contents:

Compiling linux kernel with Coresight

To compile kernel, perf and OpenCSD we need an aarch64 toolchain. In this case we will use linaro toolchain:

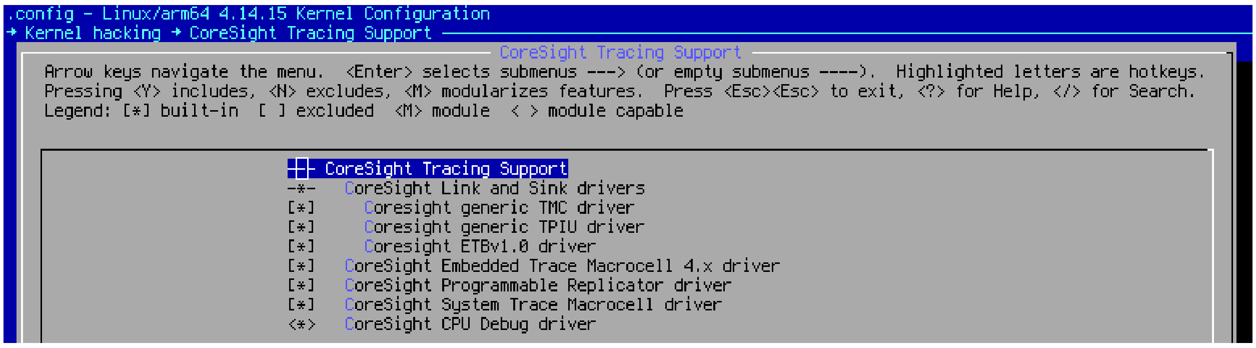
```
mkdir build
cd build
mkdir toolchain
wget releases.linaro.org/components/toolchain/binaries/latest/aarch64-linux-
gnu/gcc-*-x86_64_aarch64-linux-gnu.tar.xz
tar -xvf gcc-*-x86_64_aarch64-linux-gnu.tar.xz -C ./toolchain
```

Download source of linux kernel for dragonboard 410c:

```
git clone http://git.linaro.org/landing-
teams/working/qualcomm/kernel.git
cd kernel
```

Kernel configuration and compilation:

```
export ARCH=arm64
export CROSS_COMPILE=../toolchain/bin/aarch64-linux-gnu-
make defconfig distro.config
make menuconfig
```



```
.config - Linux/arm64 4.14.15 Kernel Configuration
→ Kernel hacking → CoreSight Tracing Support ──────────────────────────
                              CoreSight Tracing Support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.
  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                        --- CoreSight Tracing Support
                        -*-   CoreSight Link and Sink drivers
                        [*]     Coresight generic TMC driver
                        [*]     Coresight generic TPIU driver
                        [*]     Coresight ETBv1.0 driver
                        [*]   CoreSight Embedded Trace Macrocell 4.x driver
                        [*]   CoreSight Programmable Replicator driver
                        [*]   CoreSight System Trace Macrocell driver
                        <*>   CoreSight CPU Debug driver
```

```
make -j$(nproc) Image.gz dtbs
make -j$(nproc) modules
```

As a result, the compressed kernel, dtb and module will be compiled.

Now we need add dtb to the kernel:

```
cat arch/$ARCH/boot/Image.gz arch/$ARCH/boot/dts/qcom/apq8016-sbc.dtb
> Image.gz+dtb
```

Bootable SD card creating, flashing new partition with new kernel.

To create a boot image, ramdisk is required, but in fact the kernel uses a different ramdisk (which is usually written to /dev/***9), so the composition of it doesn't matter:

```
echo "not a ramdisk" > initrd.img
```

Now with new kernel and ramdisk we need to create a boot partition and flash it. I did the following: I downloaded the bootable SD card image and then replaced the boot partition in it:

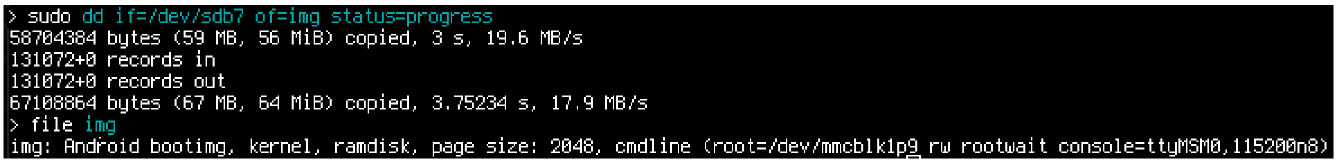
```
wget
https://releases.linaro.org/96boards/dragonboard410c/linaro/debian/lat
est/dragonboard-410c-sdcard-developer-buster-*.zip
unzip dragonboard-410c-sdcard-developer-buster-*.zip
cd dragonboard-410c-sdcard-developer-buster-*
```

Writing image to SD card (where /dev/sdb – is SD card):

```
dd if=dragonboard-410c-sdcard-developer-buster-*.img of=/dev/sdb
status=progress
```

There are nine partitions in this image (/dev/sdb[1-9]), rootfs is usually on /dev/sdb9, boot is on /dev/sdb7. It is best to look at the parameters of the original boot and build a new one with similar parameters:

```
dd if=/dev/sdb7 of=img status=progress
file img
```



```
> sudo dd if=/dev/sdb7 of=img status=progress
58704384 bytes (59 MB, 56 MiB) copied, 3 s, 19.6 MB/s
131072+0 records in
131072+0 records out
67108864 bytes (67 MB, 64 MiB) copied, 3.75234 s, 17.9 MB/s
> file img
img: Android bootimg, kernel, ramdisk, page size: 2048, cmdline (root=/dev/mmcblk1p9 rw rootwait console=ttyMSM0,115200n8)
```

To create an image, use the mkbootimg:

```
mkbootimg --kernel Image.gz+dtb --ramdisk initrd.img  --output boot-
db410c.img --pagesize 2048 --base 0x80000000 --cmdline
"root=/dev/mmcblk1p9 rw rootwait console=ttyMSM0,115200n8"
```

Writing a new boot:

```
dd if=boot-db410c.img of=/dev/sdb7 status=progress
```

To boot from SD card, you need to set the switch S6-2 to ON, and all others switches to OFF.

Compiling perf and OpenCSD

OpenCSD is library Coresight trace decoding.

Compiling OpenCSD:

```
git clone -b master https://github.com/Linaro/OpenCSD.git

cd OpenCSD/decoder/build/linux

make LINUX64=1 DEBUG=0
```

Compiling perf (P.S. For host I used 4.17 perf version, and for the target 4.8 version):

```
git clone -b perf-opencsd-4.8 https://github.com/Linaro/perf-opencsd

export CSTRACE_PATH=path/to/OpenCSD/decoder
```

Perf host compilation:

```
make
```

Also, to be able to decode the trace, you will need to install the libpython / libpython-dev package before compiling (for the host).

Target compilation:

```
make ARCH=arm64 CROSS_COMPILE=path/to/aarch64/toolchain/bin/aarch64-
linux-gnu-
```

For static compilation you need to add line LDFLAGS=-static in Makefile.perf file.

Depending on the version of perf and the toolchain, you may need to compile additional libraries for ARM64 and add them to the toolchain, for example, libelf.

Onboard tracing and displaying results in IDA Pro

If you compile and install the kernel correctly, the following devices will appear:

```
> ls /sys/bus/coresight/devices

820000.tpiu 821000.funnel 824000.replicator 825000.etf 826000.etr
841000.funnel 85c000.etm 85d000.etm 85e000.etm 85f000.etm
```

dragonboard 410c supports ETM tracing. Tracing is performed with the following command:

```
perf record –e cs_etm/@825000.etf/u --per-thread ./elf_name
```

where cs_etm – event type

825000.etf – device name

u (k) – userspace (kernelspace)

It is also possible to set a filter by specifying either the start and end address of the trace or the address range:

```
--filter ‘start 0x4003e0@elf_name, stop 0x4014d3@elf_name’
```

perf output:

```
path: /sys/bus/coresight/devices/825000.etf/enable_sink

[ perf record: Woken up 1 times to write data ]

Warning:

AUX data lost 1 times out of 4!

[ perf record: Captured and wrote 0.015 MB perf.data ]
```

Encoded trace is stored under name perf.data, files necessary for decoding is stored in ~/.debug.

Data collection for subsequent decoding on the host:

```
tar -cvzf cs_trace.tar.gz perf.data ~/.debug
```

Decoding:

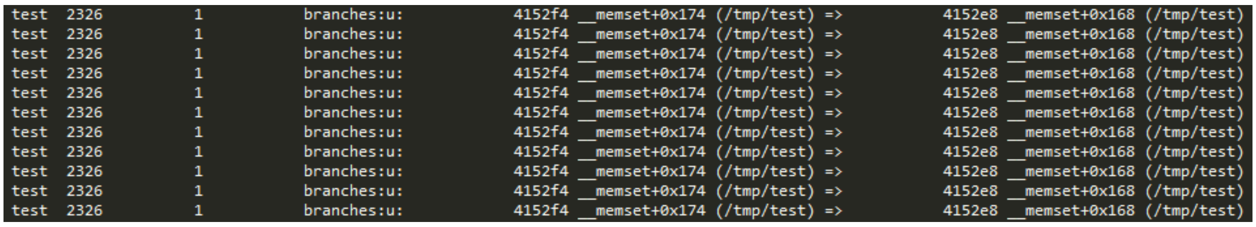
```
tar –xvf cs_trace.tar.gz

mv perf.data path/to/host/perf

mv .debug ~/.debug


cd path/to/host/perf

./perf script > trace.txt
```

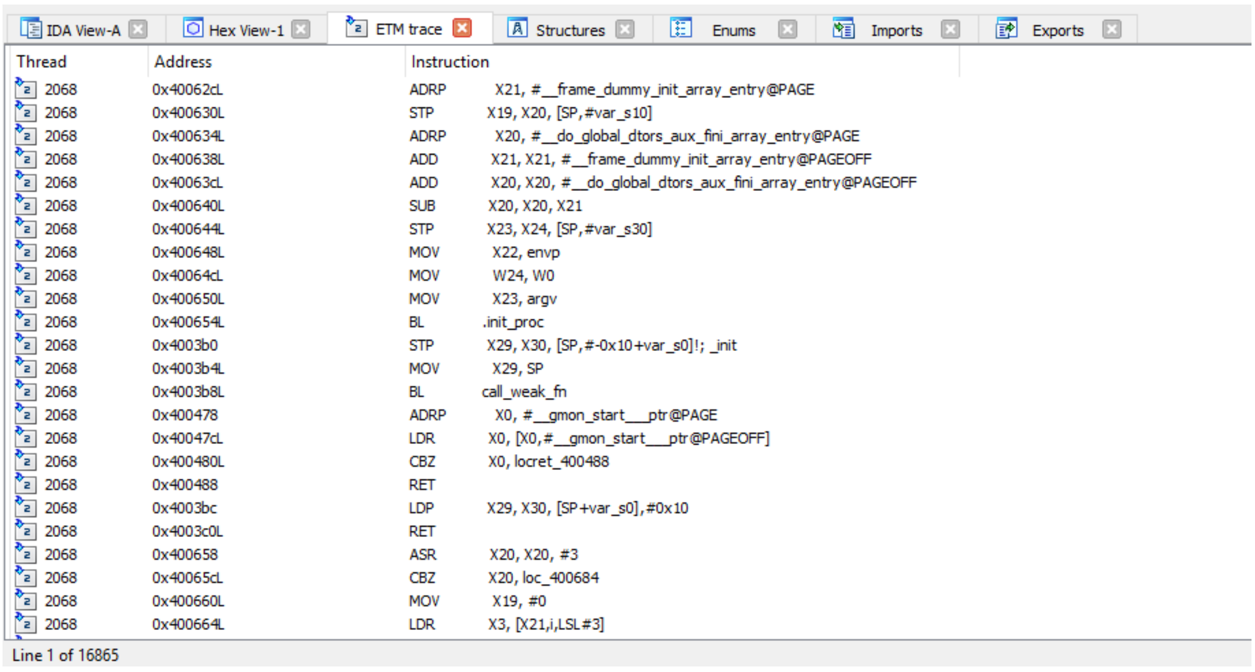


```
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
test  2326         1          branches:u:          4152f4 __memset+0x174 (/tmp/test) =>          4152e8 __memset+0x168 (/tmp/test)
```

The decoded trace contains records of all transitions that violate the sequential execution of the program.

The traced image and decoded trace can be displayed in IDA Pro using the plugin.

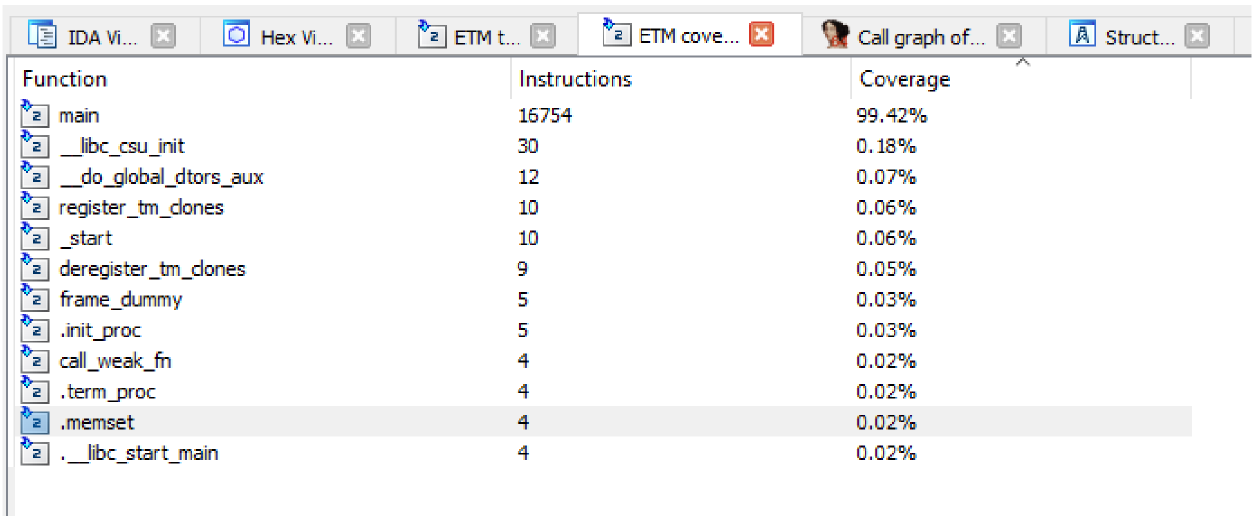
Trace:



Thread	Address	Instruction
2068	0x40062cL	ADRP X21, #__frame_dummy_init_array_entry@PAGE
2068	0x400630L	STP X19, X20, [SP,#var_s10]
2068	0x400634L	ADRP X20, #__do_global_dtors_aux_fini_array_entry@PAGE
2068	0x400638L	ADD X21, X21, #__frame_dummy_init_array_entry@PAGEOFF
2068	0x40063cL	ADD X20, X20, #__do_global_dtors_aux_fini_array_entry@PAGEOFF
2068	0x400640L	SUB X20, X20, X21
2068	0x400644L	STP X23, X24, [SP,#var_s30]
2068	0x400648L	MOV X22, envp
2068	0x40064cL	MOV W24, W0
2068	0x400650L	MOV X23, argv
2068	0x400654L	BL .init_proc
2068	0x4003b0	STP X29, X30, [SP,#-0x10+var_s0]!; _init
2068	0x4003b4L	MOV X29, SP
2068	0x4003b8L	BL call_weak_fn
2068	0x400478	ADRP X0, #__gmon_start___ptr@PAGE
2068	0x40047cL	LDR X0, [X0,#__gmon_start___ptr@PAGEOFF]
2068	0x400480L	CBZ X0, locret_400488
2068	0x400488	RET
2068	0x4003bc	LDP X29, X30, [SP+var_s0],#0x10
2068	0x4003c0L	RET
2068	0x400658	ASR X20, X20, #3
2068	0x40065cL	CBZ X20, loc_400684
2068	0x400660L	MOV X19, #0
2068	0x400664L	LDR X3, [X21,i,LSL#3]

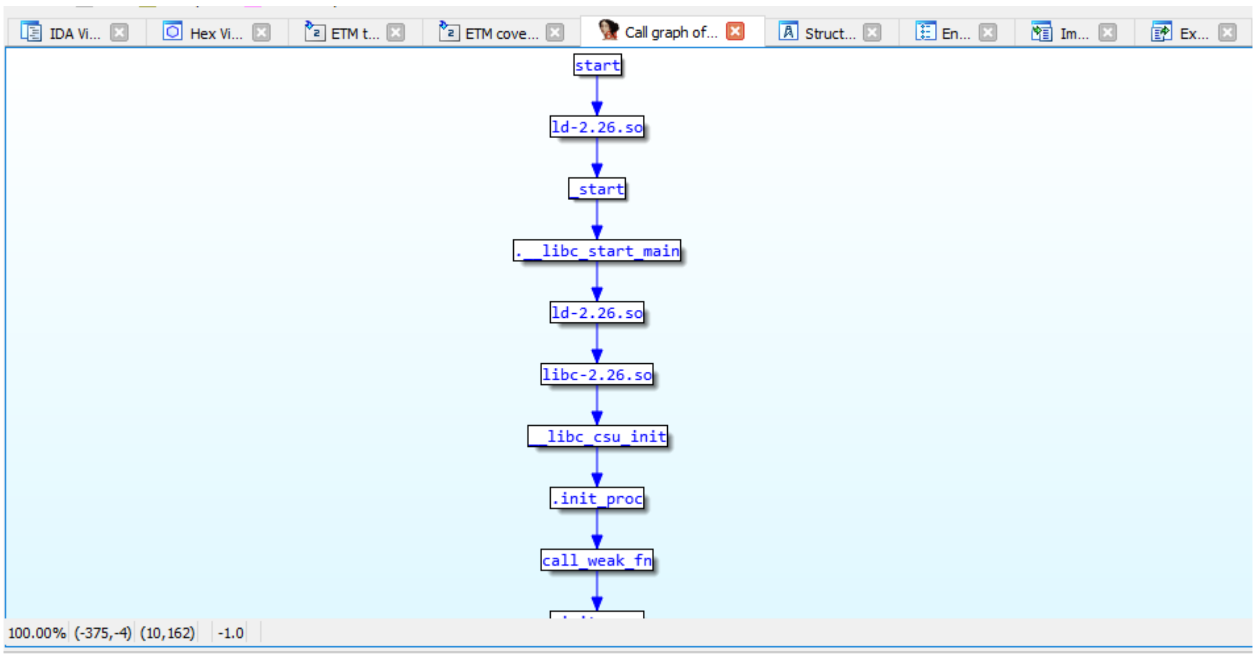
Line 1 of 16865

Code coverage:



Function	Instructions	Coverage
main	16754	99.42%
__libc_csu_init	30	0.18%
__do_global_dtors_aux	12	0.07%
register_tm_clones	10	0.06%
_start	10	0.06%
deregister_tm_clones	9	0.05%
frame_dummy	5	0.03%
.init_proc	5	0.03%
call_weak_fn	4	0.02%
.term_proc	4	0.02%
.memset	4	0.02%
.__libc_start_main	4	0.02%

The sequence of function calls:


IDA Vi...
Hex Vi...
ETM t...
ETM cove...
Call graph of...
Struct...
En...
Im...
Ex...
start
ld-2.26.so
_start
.__libc_start_main
ld-2.26.so
libc-2.26.so
__libc_csu_init
.init_proc
call_weak_fn
100.00% (-375,-4) (10,162) -1.0