LibSoCCA: Non-Intrusive Power & Performance Debugging via JTAG

ELCE 2018 - Alexandre Bailon & Patrick Titiano

Demo

Motivations

Map of the Problematic

- Power and Performance debugging requires
  - Non-intrusive ‘access’ to the target, to avoid altering
    - CPU execution flow
    - CPU/Peripherals/Platform power states
  - Realtime monitoring of the target
    - Dynamic view of a use-case for profiling purposes
  - Multi OS / Arch support
  - Open Source
- Can anyone name one such tool?
  - Most tools available today run on target, target a single OS, and share data with host via UART/USB/Ethernet/…, and are not applicable during low-power transitions
    - Ftrace, perf, powertop, (h)top, DDMS (Android), Snapdragon Profiler (Android, Qualcomm), ...

What could be improved?

- Running on host instead of on target
  - Non-intrusive
  - No code to rebuild/reflash/...
- Use common libraries
  - To enable modular / scalable debug applications
- Define standard way to describe SoC
  - To enable generic (multi-arch) debugging/profiling tools
  - E.g. as Device Tree helped Linux Kernel scales with exploding arch/variants

libSoCCA
(SoC Continuous Analyzer)

Main Features

- Non-intrusive SoC register accesses via JTAG debugger
- Abstracts architectures leveraging SVD files
- OS-agnostic
- Pure python host application
- Sources:
  - https://gitlab.com/socca/lib/libsocca.git
  - https://gitlab.com/socca/apps/pmugraph.git
- Documentation:
  - https://gitlab.com/socca/lib/libsocca/wikis/home

Why JTAG?

- Allow non-intrusive R/W accesses to SoC internal registers
- Support (HW) Breakpoint / Watchpoint
- Supported by most SoC / boards
- Manageable via generic OpenOCD SW library

SVD Files?

- Stands for “System View Description”
- Describes SoC registers (address, bitfields, description)
- XML-based
- Conceptually similar to Linux Kernel Device Tree source files

SVD Files

Sample description of a device (SoC) in SVD format

```
<?xml version="1.0" encoding="UTF-8"?>
<device xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
        schemaVersion="1.1"
        xs:noNamespaceSchemaLocation="CMSIS-SVD_Schema_1_1.xsd">
  <vendor>Amlogic</vendor>
  <name>S905X</name>
  <version>1.0</version>
  <description>S905X SoC</description>
  <addressUnitBits>8</addressUnitBits>
  <width>32</width>
  <size>0x20</size>
  <access>read-write</access>
  <resetValue>0x00000000</resetValue>
  <resetMask>0xFFFFFFFF</resetMask>
  <peripherals>
    ...
  </peripherals>
</device>
```

SVD Files

Sample description of a peripheral in SVD format

```
<peripheral>
  <name>RNG</name>
  <description>Random number generator</description>
  <groupName>RNG</groupName>
  <baseAddress>0x50060800</baseAddress>
  <addressBlock>
    <offset>0x0</offset>
    <size>0x400</size>
    <usage>registers</usage>
  </addressBlock>
  <interrupt>
    <name>FPU</name>
    <description>FPU interrupt</description>
    <value>81</value>
  </interrupt>
  <registers>
    ...
  </registers>
</peripheral>
```

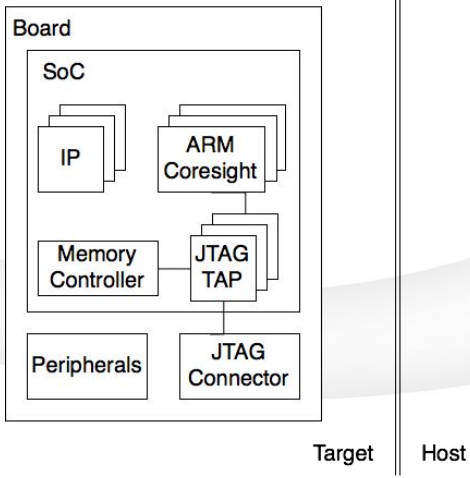
SVD Files

Sample description of a register in SVD format

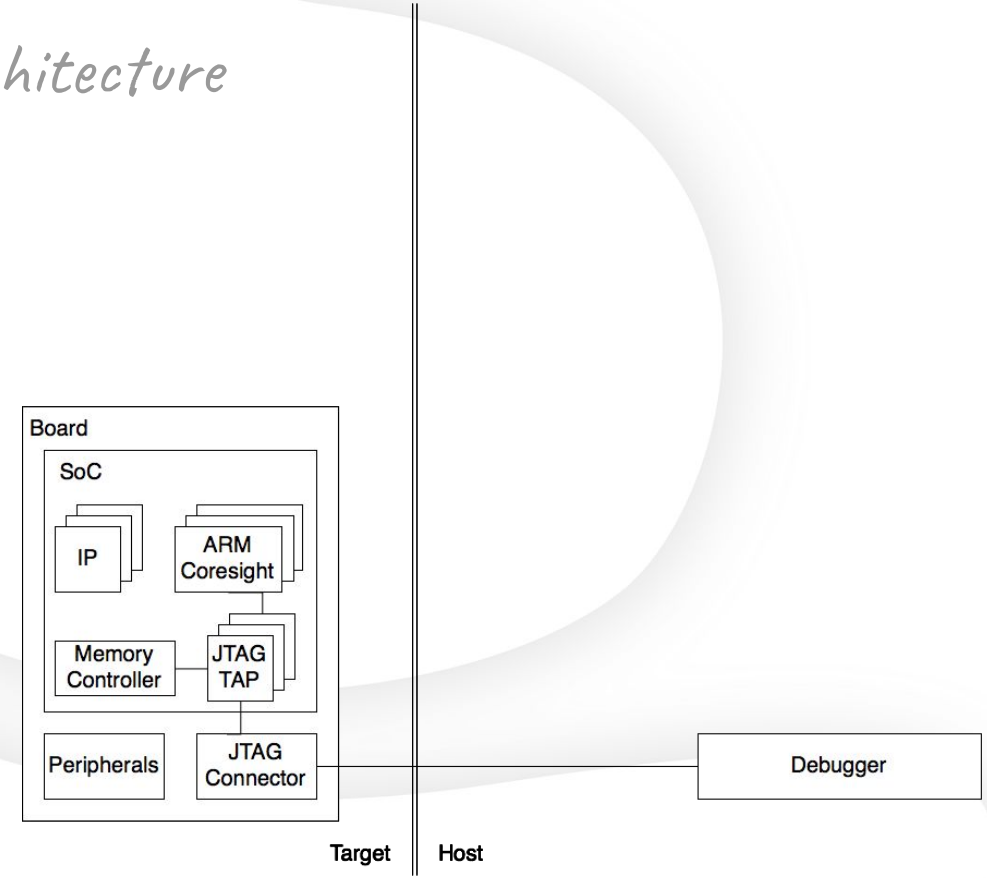
```
<register>
  <name>CR</name>
  <displayName>CR</displayName>
  <description>control register 1</description>
  <addressOffset>0x0</addressOffset>
  <size>0x20</size>
  <access>read-write</access>
  <resetValue>0x0000</resetValue>
  <fields>
    <field>
      <name>ENABLE</name>
      <description>DCMI enable</description>
      <bitOffset>14</bitOffset>
      <bitWidth>1</bitWidth>
    </field>
    ...
  </fields>
</register>
```

libSoCCA Architecture

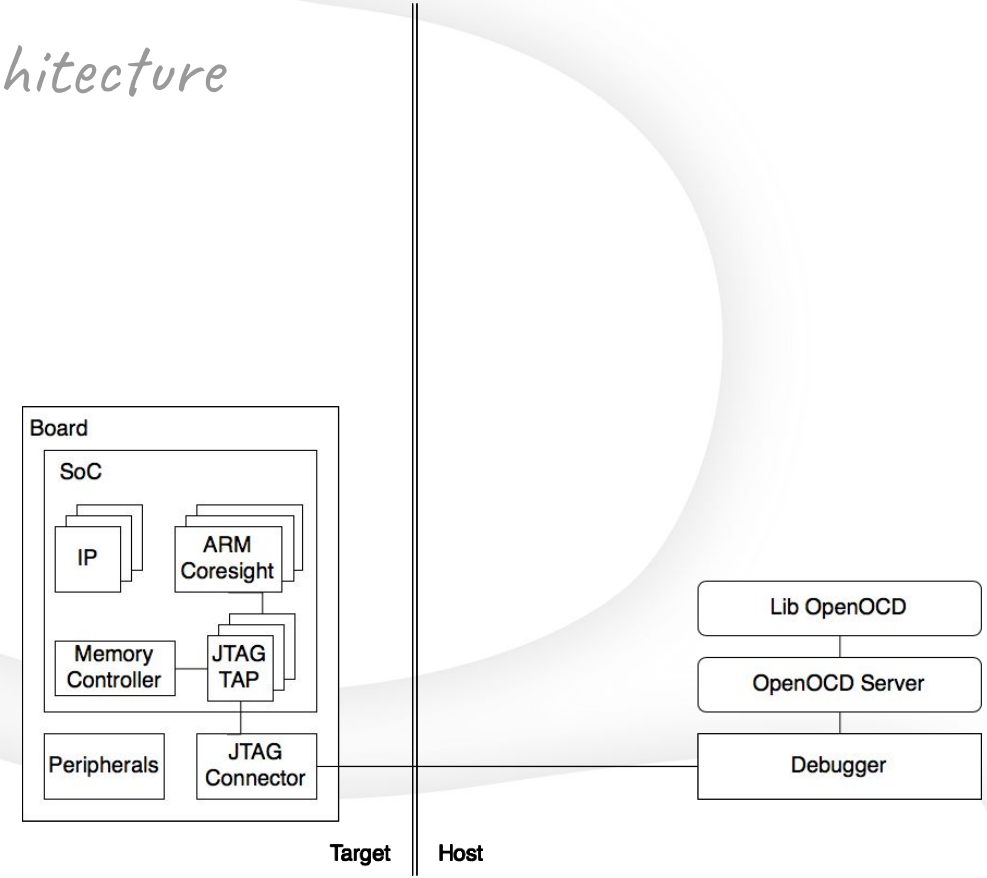
LibSocca Architecture



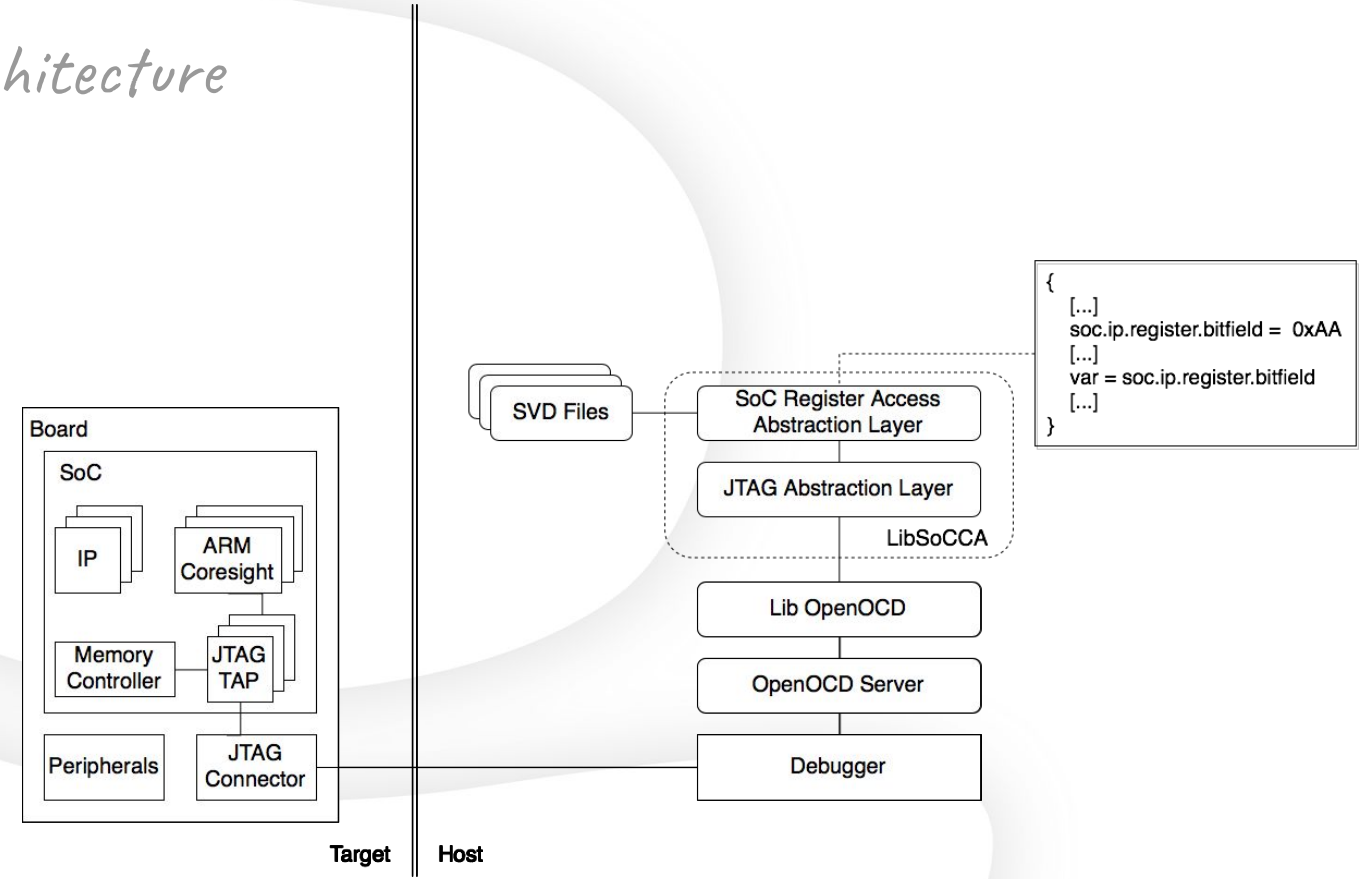
LibSocca Architecture



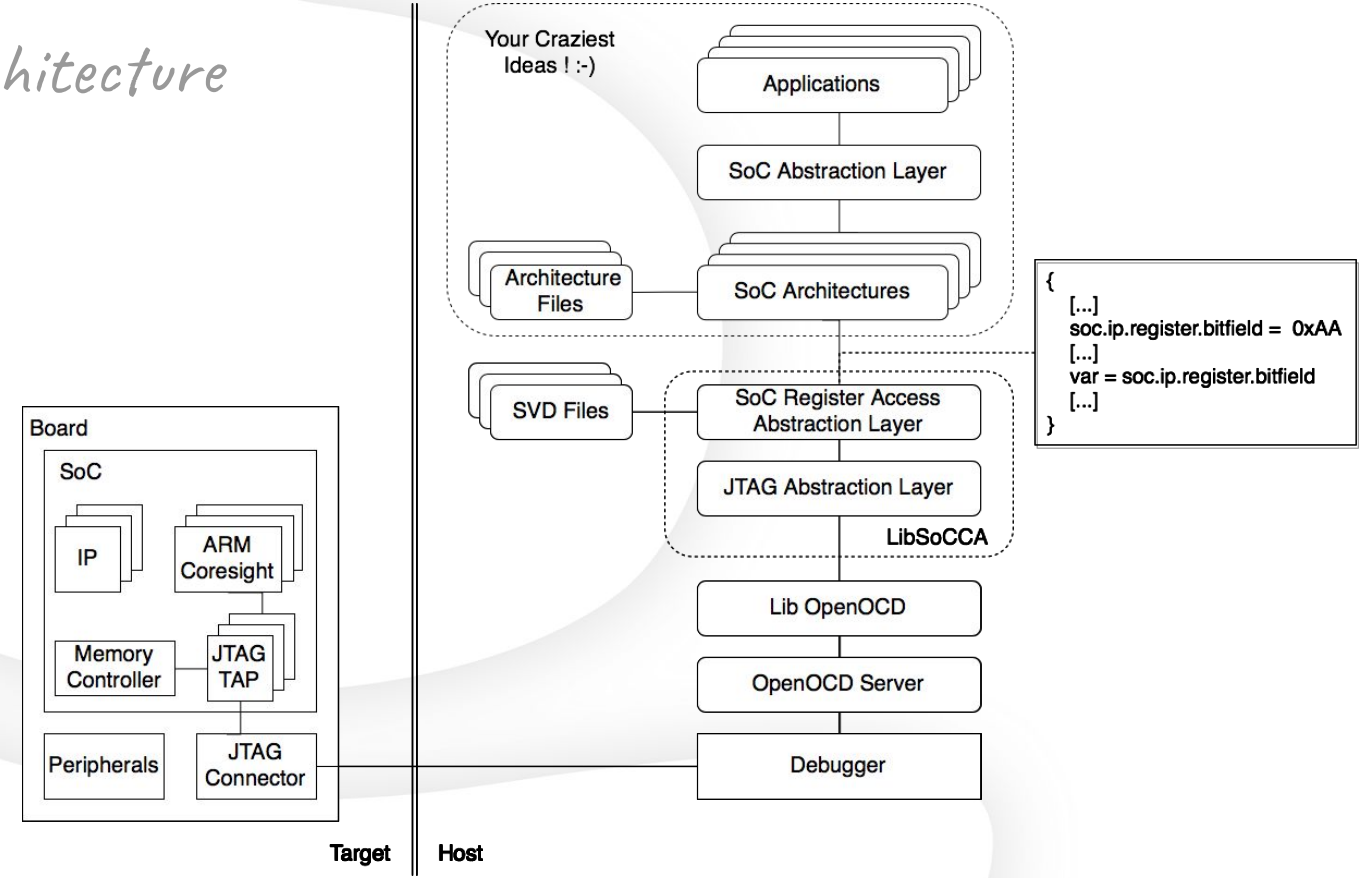
LibSocca Architecture



LibSocca Architecture



LibSocca Architecture



Already Available in LibSoCCA

- Subsystems
  - Clock
  - PMU
- Architecture support
  - ARMv7 & ARMv8
  - AMLogic S905X
  - NXP iMX7ULP
  - STM32F4

Potential Applications Examples (but not limited to!)

Applications

- PMUGraph
  - Realtime plotting of CPU busy cycles (‘CPU Load’), memory access (‘Memory Load’)
  - Collect data from generic ARM Performance Monitoring Unit (PMU)
- Generated overhead on target:
  - CPU: 0 (CPU cores not halted, no code executed) memory
  - Interconnect: 480 Bytes/sec @ 10Hz, 4.8 KBytes/sec @ 100Hz (ARM PMU register reads)
    - Negligible compared to standard interconnect bus speeds (400 MBytes/sec and more)
    - Negligible compared to standard JTAG speed (500K Bytes/sec and more)
- Available on gitlab and demo’ed at ELC-E 2018

Applications

- Memtool
  - Read / write memory
  - Read / Write registers (using theirs name)
  - Support of registers fields
  - Monitor memory / register accesses using watchpoints / breakpoints
- Available on gitlab
  - Development ongoing, only basic features implemented.

Applications

- Clock tool
  - Clock status snapshots (status, speed, statistics, …)
  - Clock changes monitoring / triggering using watchpoints / breakpoints
  - Realtime Clock Tree visualization
  - Runtime clock control (enable / disable clocks on the fly)
- Development not yet started

Applications

- Realtime Power & Performance Profiling Tool
  - Realtime collection of SoC data (clocks, PMU, CPU cores/cluster states, GenPD, …)
  - Realtime SoC power measurement (e.g using BayLibre's ACME)
  - Realtime plotting of all collected data, incl. features like
    - Start/stop/freeze/resume trace collection,
    - Trace zoom in/out,
    - Save/load/export trace,
    - Command-line interface to enable further (CI) integration
- Development not yet started.

How Easy Writing libSoCCA Apps Is

```
def _enable(dev, cnt_id):
    dev.PMU0.PMCNTENSET |= (1 << cnt_id)

def _disable(dev, cnt_id):
    dev.PMU0.PMCNTENCLR |= (1 << cnt_id)

def _enabled(dev, cnt_id):
    return dev.PMU0.PMCNTENSET & (1 << cnt_id)
```

```
def example_pmu(dev):
    pmu = dev.pmus[0]
    counters = pmu.get_counters()
    # Enable ARM CPU cycle counter
    pmccntr = counters['PMCCNTR']
    pmccntr.enable()
    pmu.enable()

    # Read the value of counter
    value = pmccntr.read()
```

How Easy Writing libSoCCA Apps Is

```
def example_cpu_load(dev, cpu_id):
    pmu = dev.pmus[cpu_id]
    cpu_load_event = ARMCPULoad(pmu, cpu_id, 0, 1.5 * GHz)
    cpu_load = cpu_load_event.get_value()

def example2_cpu_load(dev):
    perf = Perf(dev)
    events = perf.get_events(Perf.CPU_LOAD)
    for event in events:
        cpu_load = event.get_value()
```

War Stories

Major Difficulties Faced (1)

- JTAG / ARM Coresight
  - Poor documentation when dealing with connecting to JTAG TAP(s) other than the CPU one
  - Debugging capabilities different from one SoC to another
- SVD Files
  - Limited number of platforms providing SVD files
  - Do not include ARM Clusters description (whereas generic)
    - Had to generate it ourselves
  - Does not support file inclusion ('#include ….')
    - Developed a tool to 'append' SVD files together

Major Difficulties Faced (2)

- OpenOCD (Server part)
  - Tricky to get working
    - ARM v7 / v8 changes poorly documented, leading to many crashes will trying to set it up for S905X
    - Warnings messages mixed up with command responses causing OpenOCD lib unpredictable behaviour
- OpenOCD python library
  - Has a non-friendly way of handling watchpoints or breakpoints
    - Designed for polling on data, not waiting on events
      - Perf./stability issues (e.g. asynchronous events may be lost or cause an error if received while processing another event)
    - Considering writing a new lib using OpenOCD server API instead

What's Next?

What's next?

- Enable watchpoint / breakpoint
  - Avoid data polling for profiling apps
- Integrate CI frameworks
  - Enable regression-testing of use-case KPI / golden settings
- Make LibSoCCA reentrant to enable concurrent use
- Start writing libSoCCA documentation ;-)
- Support more SoC / Subsystems / IPs
- Develop more libSoCCA apps
  - Runtime Clock Tree Visualizer, KPI Checker, Power Profiler, Power Estimation tool based on real data (and not educated guess), ...

Closing

Takeaways

- JTAG offers a unique solution for non-intrusive real-time monitoring tools
- Similarly to device tree for the Linux Kernel, SVD files helps handling multiple architectures and variants
- libSoCCA is an innovative SW framework which helps developing generic debugging/profiling tools combining use of JTAG and SVD files
- PMUGraph is just a first basic illustration of libSoCCA potential
- libSoCCA counts on you to create the smartest apps on top of it!

Thank you!