System-wide visibility in post-silicon to drive meaningful analytics

EDPS Symposium September 2017

Agenda

- Some obvious statements
- Some problems with existing approaches
- Key Requirements
- Some examples of Performance analysis and Debug
- Use cases
- Summary

Some obvious statements

- SoCs have become increasingly complicated and they are not going to get simpler.
  - Contain several processors, from different vendors
    - Verified in isolation and come with test suite
  - Contain 100s of SIP
    - Each verified in isolation
  - Contain complex interconnects
    - Verified for certain, identified conditions
  - Software created by large disparate teams.
    - If lucky, modules and subsystem verified for certain, identified conditions.
  - All this has to successfully work together
- Understanding real world system behaviour is just plain HARD!

Some Problems with existing approaches

- Processor-centric, not system-centric
  - Processors are a very small part of the overall system
- Hard to get a handle on bus behaviour, memory controllers, let alone interactions between blocks etc.
- Where they include analytics it's lip service - very little smarts
  - Knock-on effect of fast data pipe off-chip
- Intrusive
- Ad hoc
- Developing, but still essentially signal-based.
  - Hard to close timing
- In-field monitoring is not easy

Key requirements

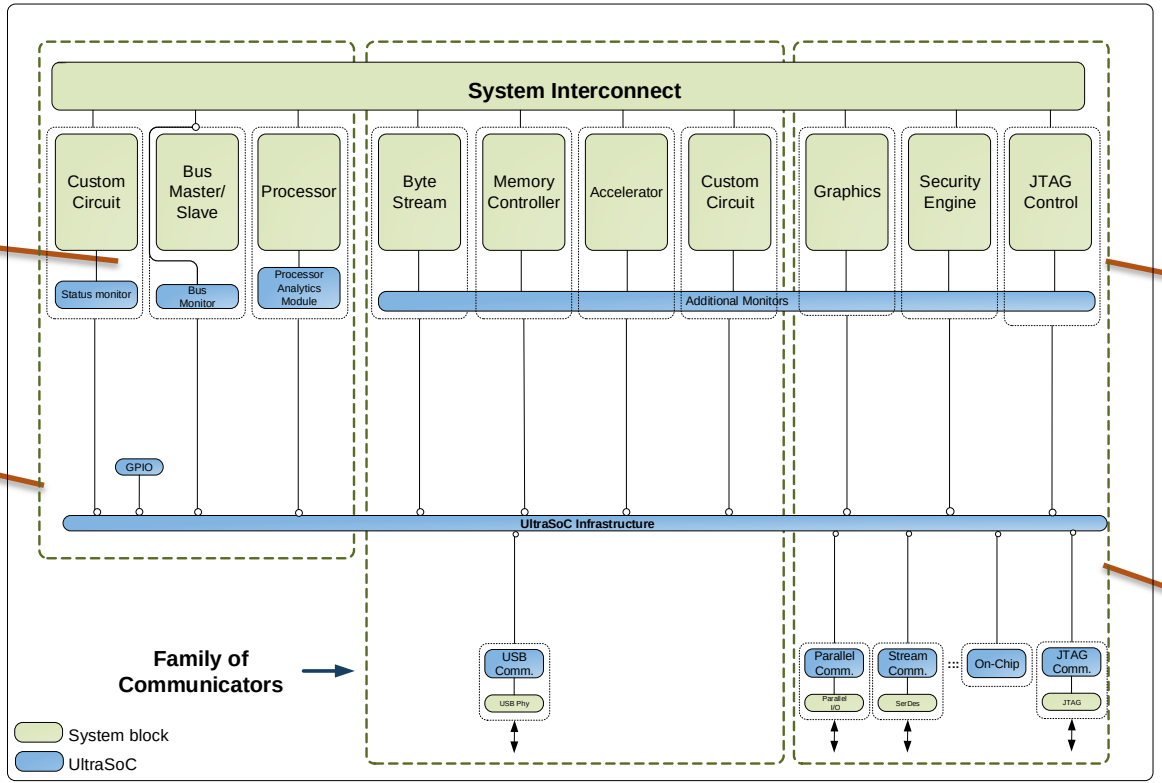
- A System-centric vendor-neutral debug and monitoring infrastructure
  - One that enables access to different proprietary debug schemes used today by various cores
  - Allows for monitors into interconnects, interfaces and custom logic
  - These need to be run-time configurable
    - Re-use the hardware to provide visibility for different scenarios.
    - Run-time configuration of cross-triggering
    - Support 10s if not 100s of cross-triggering events
  - These can be interrogated after a problem to determine actual status
  - Need to be power aware
  - Security built-in
  - Can be used during the whole development flow and more importantly in the field

Advanced Monitoring and Debug for the Whole SoC

Modules are protocol aware and "smart" with filter and trace

Flexible scalable message fabric, easy to route

Debug & trace is transparent: does not impact system bus

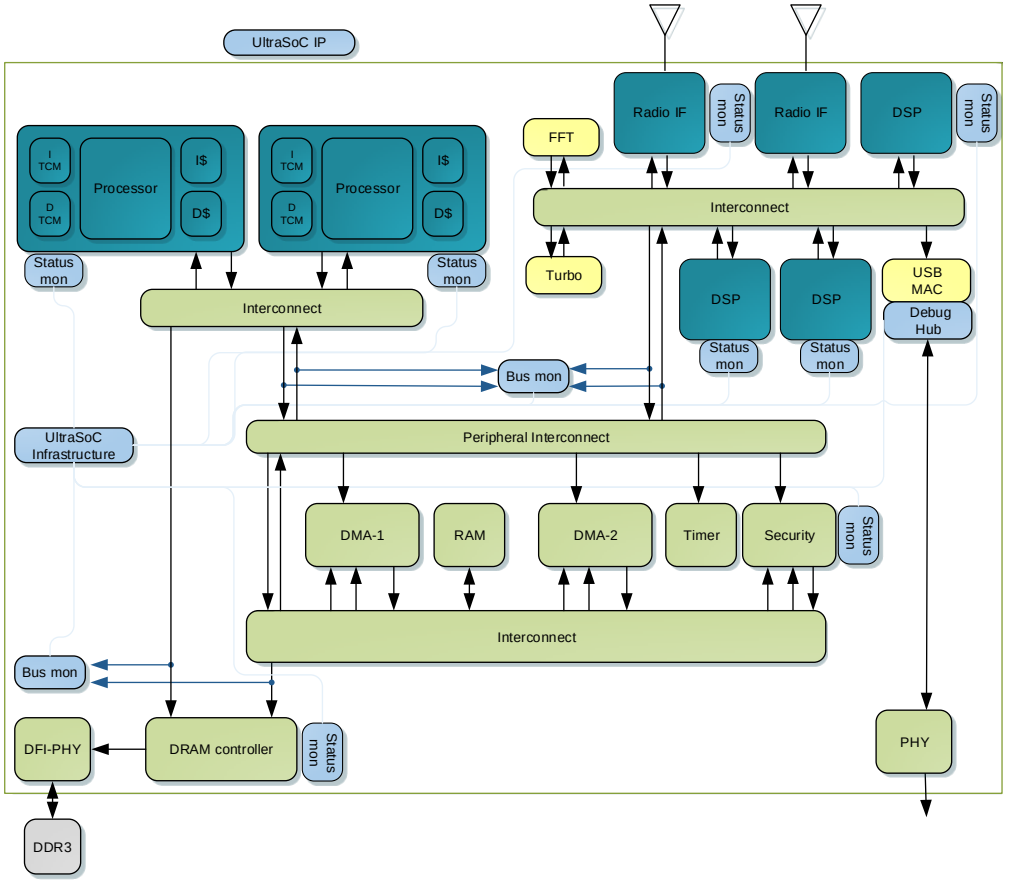


Portfolio of configurable modules, optimized for different system IP blocks

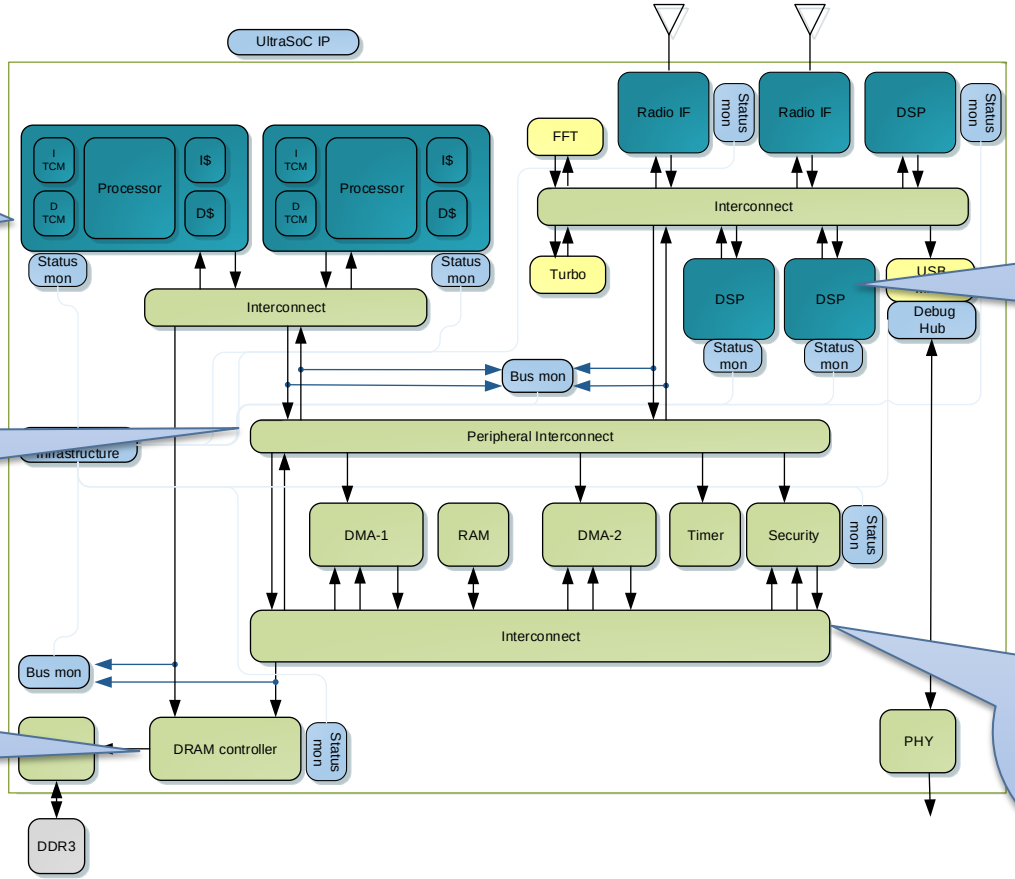
Supports subsystems with different power domains, clock domains

Some examples of Performance analysis and debug

Example of UltraSoC Enabled SoC



Example Problems UltraSoC Solves



Why is the CPU not performing as fast as expected?

What is the mismatch between the host & the DSP?

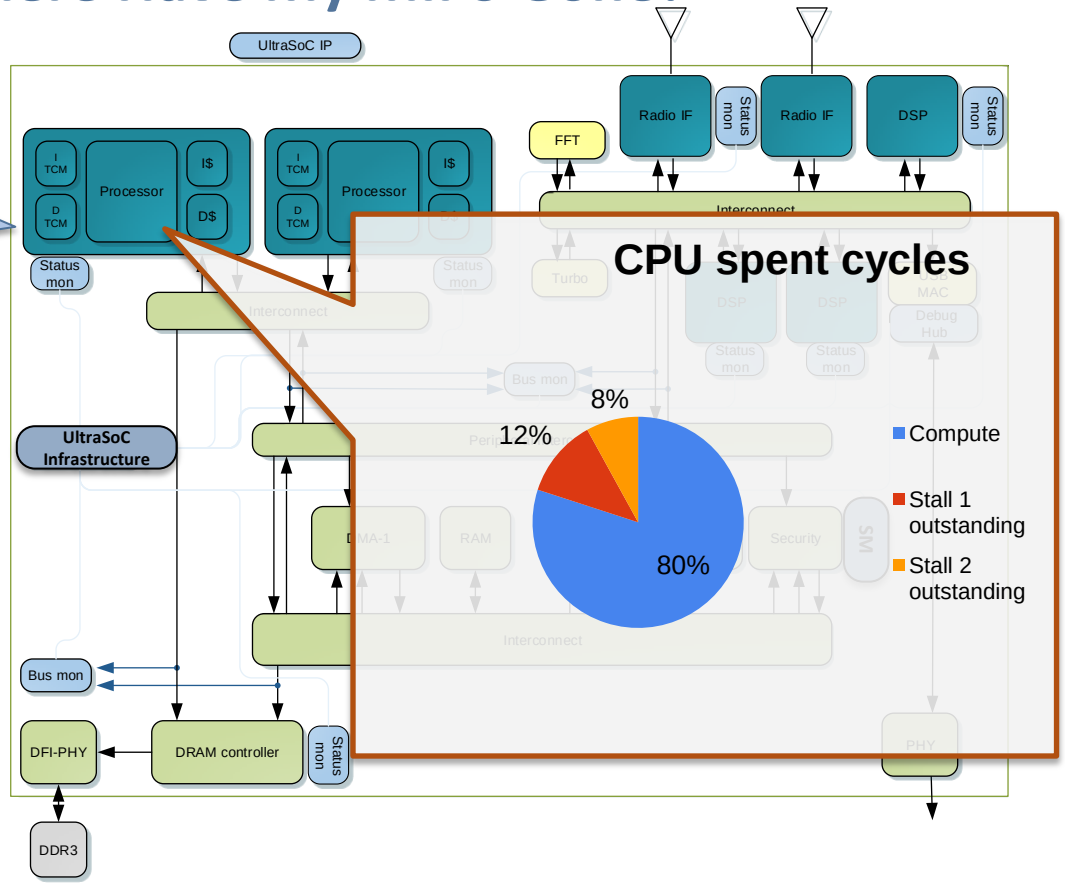
Why do some DMA transfers take too long?

What is going on with my memory controller?

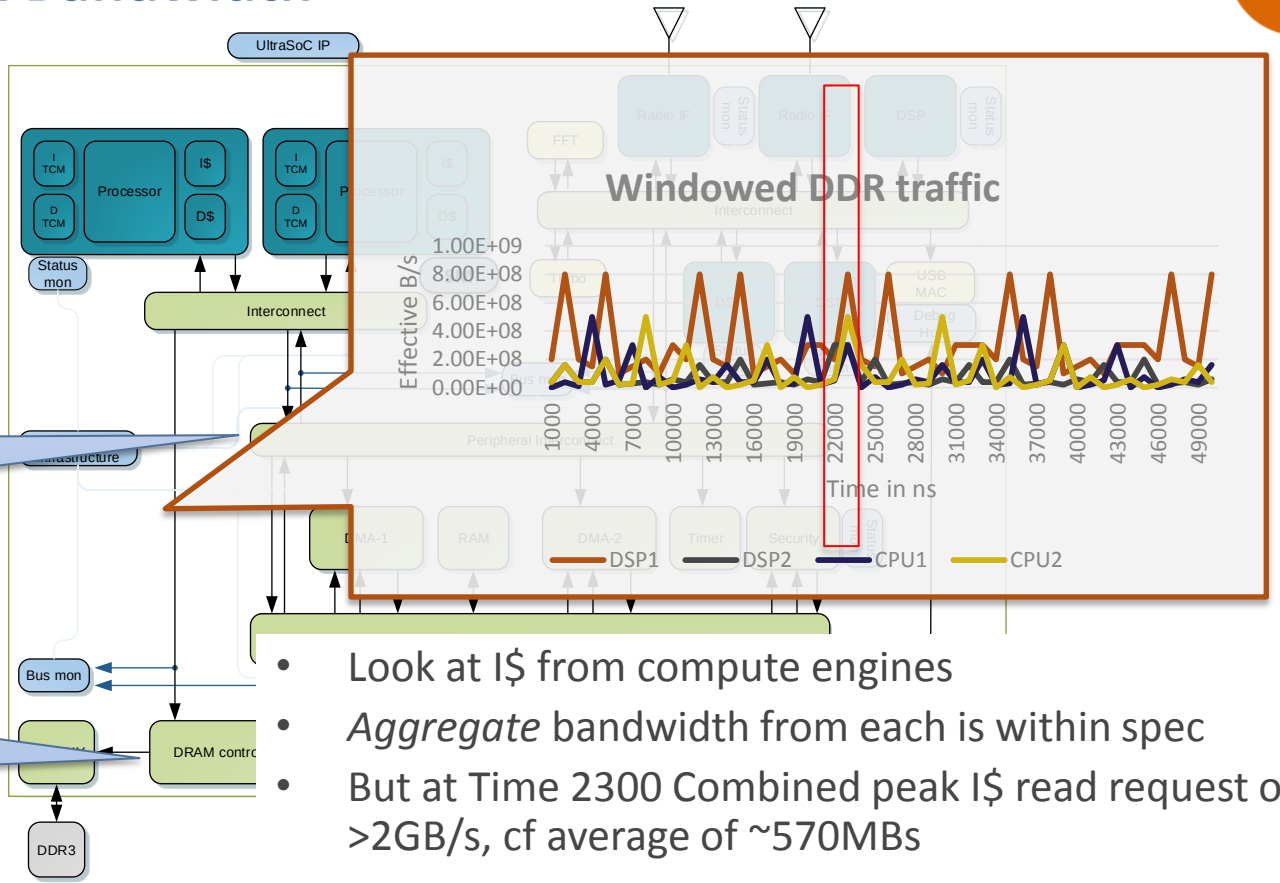
Why does the system hang or deadlock on rare occasions?

Example 1: “Where Have My MIPS Gone?”

Why is the CPU not performing as fast as expected?



Example 2: DDR Bandwidth



Why do some DMA transfers take too long?

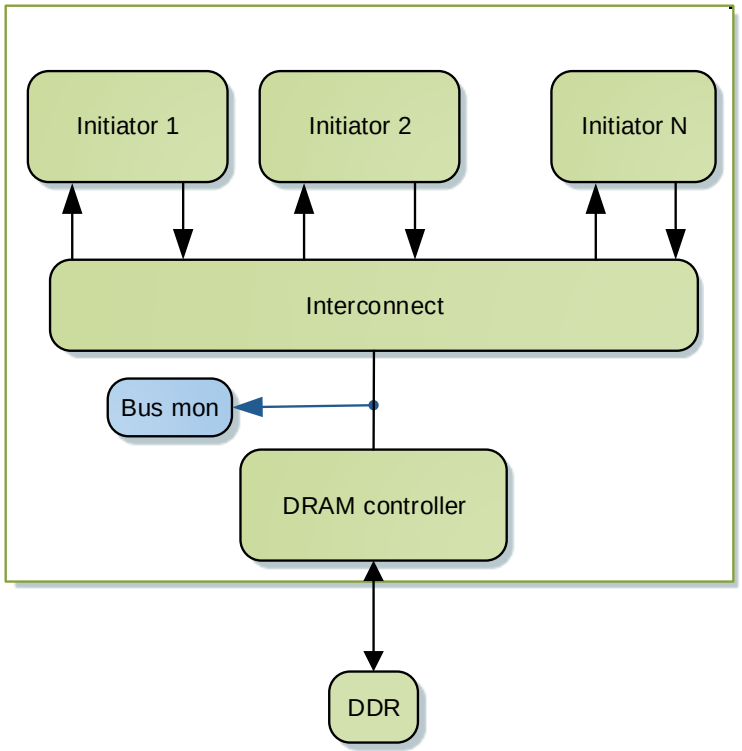
What is going on with my memory controller?

- Look at I$ from compute engines
- *Aggregate* bandwidth from each is within spec
- But at Time 2300 Combined peak I$ read request of >2GB/s, cf average of ~570MBs

Example 3 : Deadlock Detection

- Many different types but consider this as an example
  - CPU (master) asserts arvalid and issues a read address to the Slave
  - Slave asserts rvalid and outputs read data but never sees rready asserted
- Configure bus monitor trace to trigger when transaction duration exceeds threshold (programmable up to 16k cycles)
  - Trace not output until triggered.
  - When triggered by deadlocked transaction, trace will output most recent transactions up to and including the deadlocked transaction
  - Trace identifies transaction ID and address, identifying both master and slave of deadlocked transaction

Example 4 : Data Corruption Detection



To detect the initiators doing write access to a same memory location (or a range) - MemAddress. We can configure our Bus Monitor do something like:

```
if <Address> == MemAddress && <RW> == Write
then if Count > 1
        CaptureTrace()
        SendEventMessage()
     else
        IncrementCount()
fi
```

Where:

- <> are AXI bus fields being observed by the bus monitor.
- CaptureTrace() puts the transaction into the trace buffer
- SendEventMessage() is an instruction to the monitor to send an event out on our message bus
- IncrementCount increments the counter by 1

- NB This is pseudo-code actual filtering is down in hardware and not software

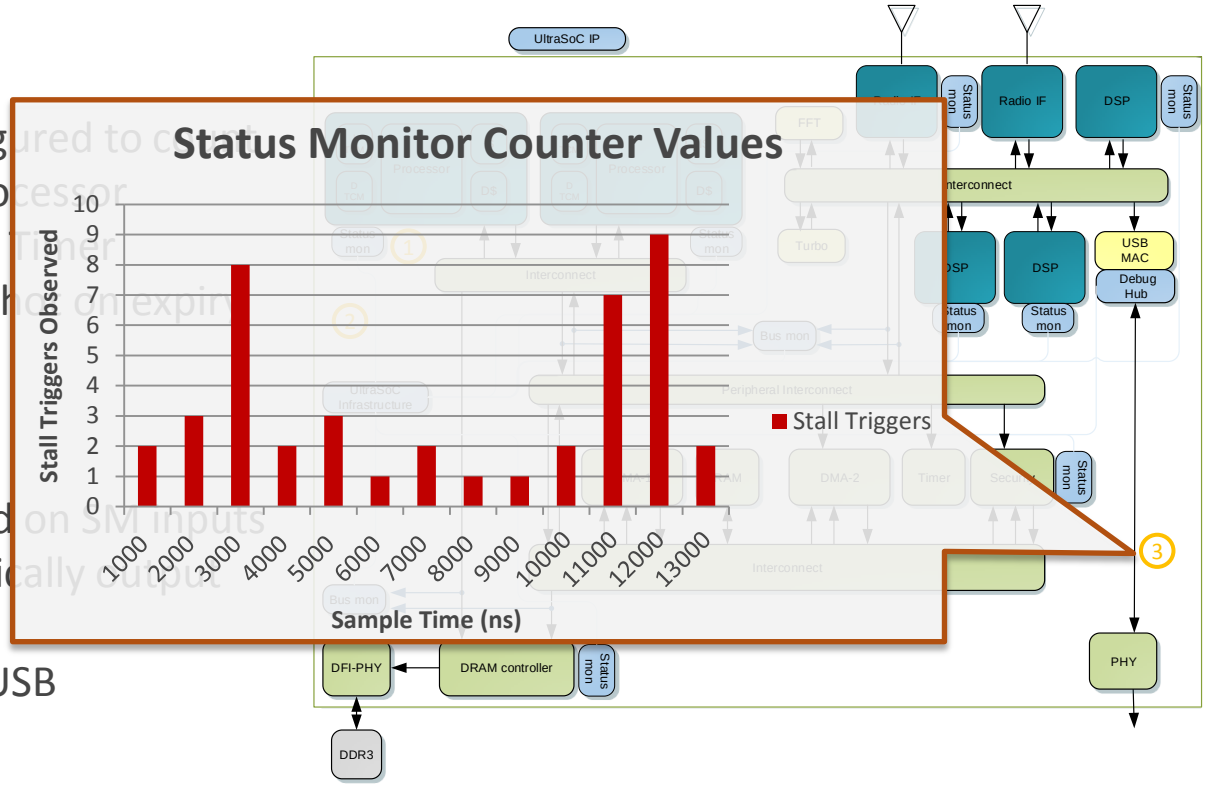
Metrics Generation – Example 1

Runtime Configuration

- Status Monitor configured to count Stall triggers from Processor
- Set period of Interval Timer
- Counter values snapshot on expiry of interval timer.

Data Flow

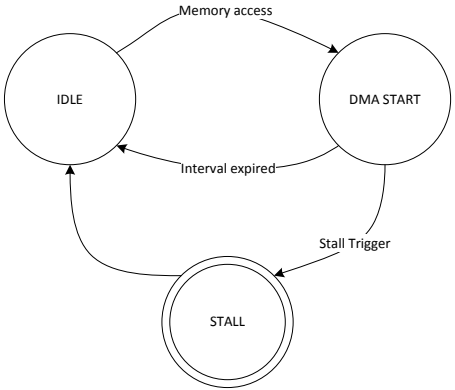
1. Stall trigger observed on SM inputs
2. Counter data periodically output from SM
3. Data traced out via USB



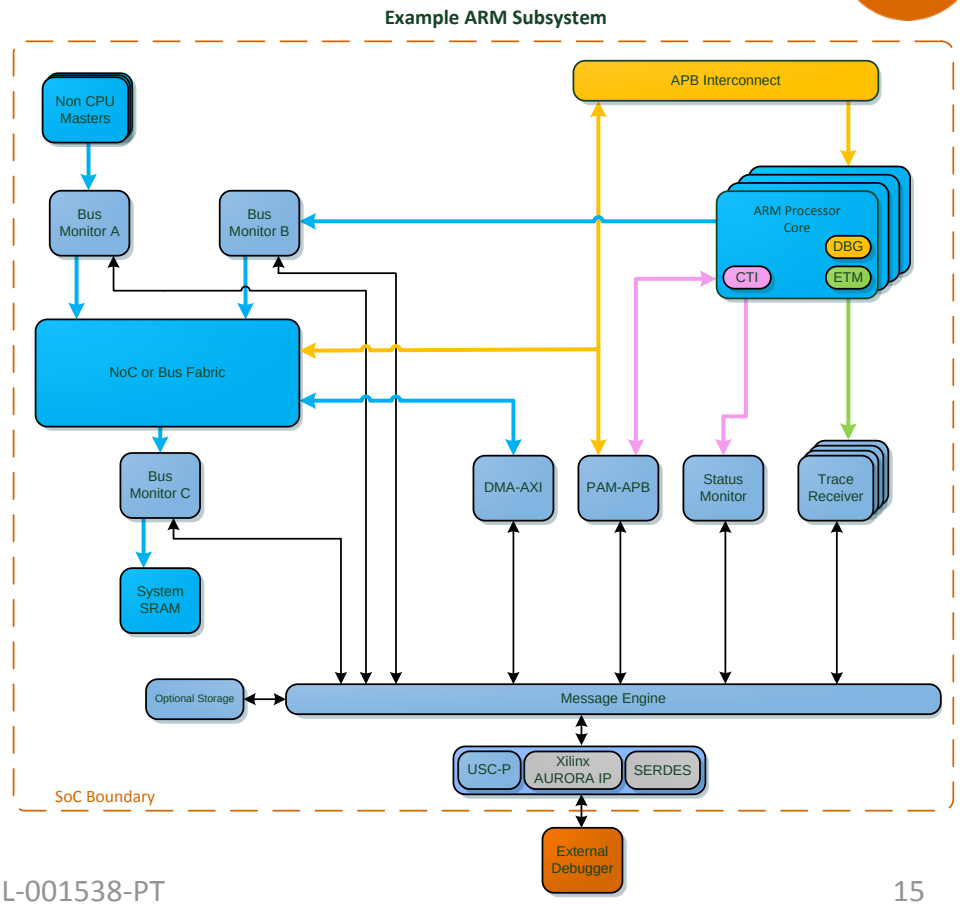
Cross Triggering – Example 1

Runtime Configuration

- Bus Monitor A outputs Event on DMA access
- Set the period of the Status Monitor's Interval Timer
- Configure the Status Monitor to observe the following sequence:



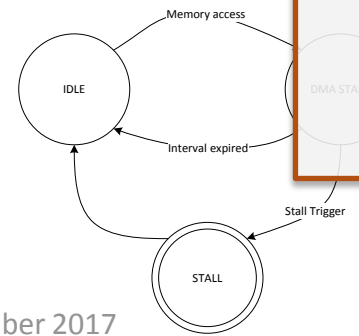
- Output trigger from SM when entering the STALL state
- Configure Trace Receiver(s) to enable tracing on receipt of trigger

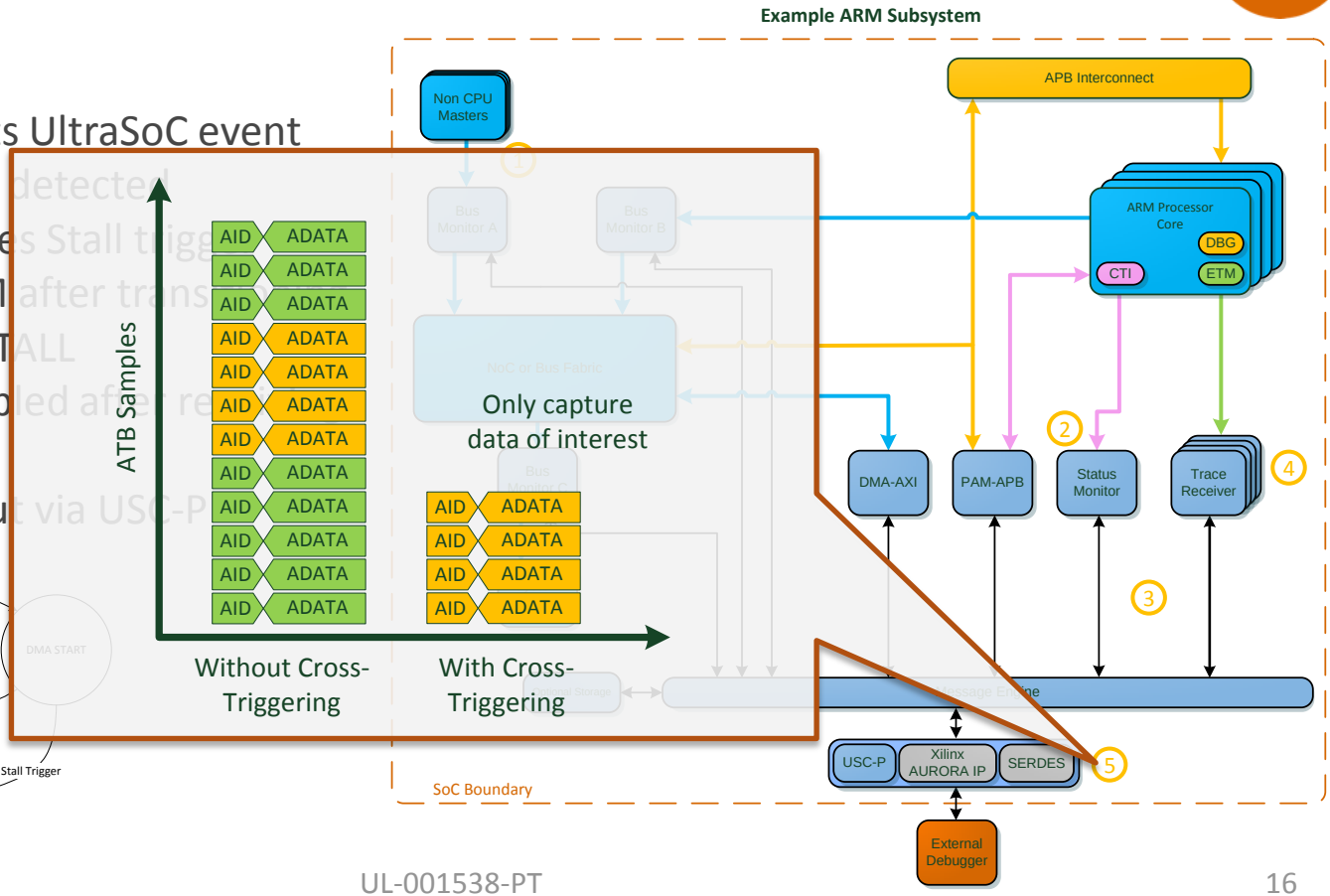


Cross Triggering – Example 1 (cont)

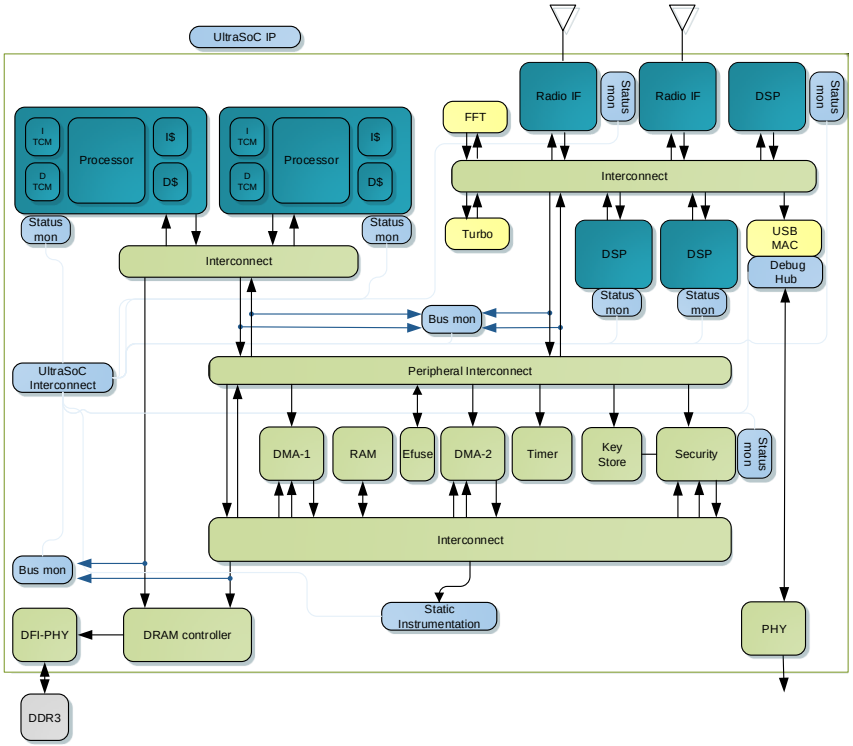
Data Flow

1. Bus Monitor A outputs UltraSoC event when memory access detected
2. Status Monitor receives Stall trigg
3. Event output from SM after trans
   from DMA START -> STALL
4. Trace Receiver(s) enab
   event
5. Processor Trace outpu via USC-P



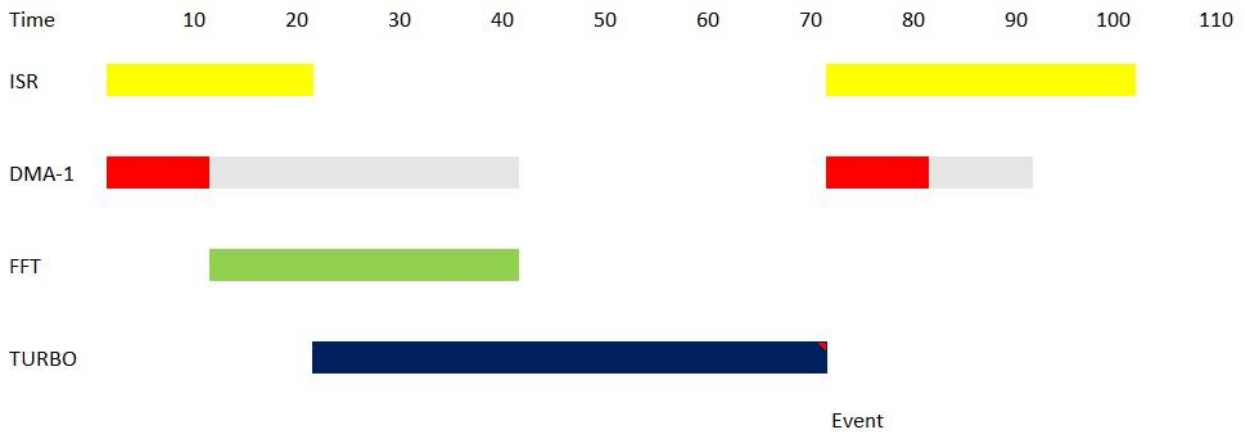


Example of Instrumented SoC

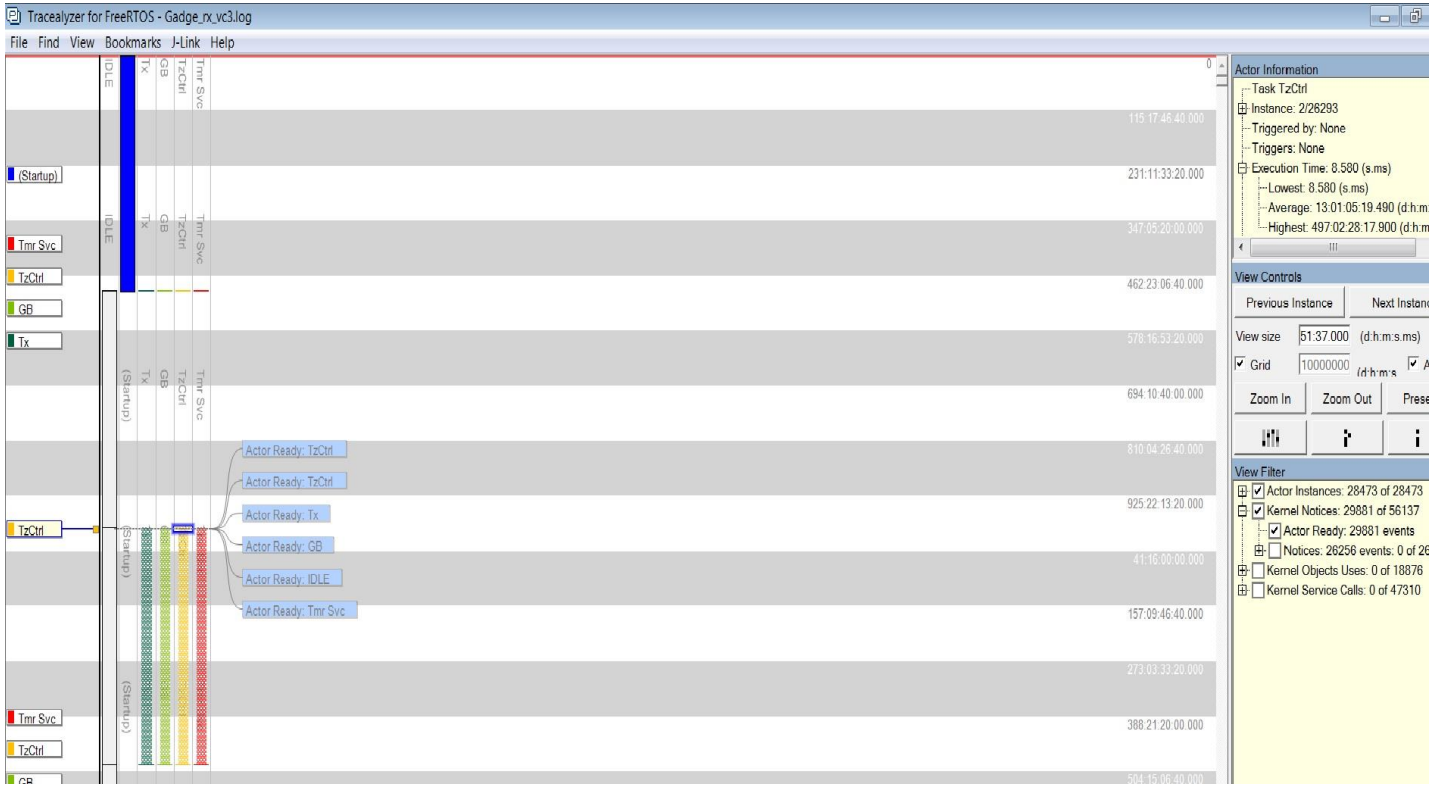


- The SI provides independent memory-mapped channels (mailboxes)
- Software and hardware can post writes to these channels which can be used to understand system wide behaviour
- The data is timestamped
  - Or no data if only timestamp needed.
- The channels can be filtered
- Each channel can be enabled to provide events which can be used for cross-triggering
- The Virtual Console provides bi-directional channels

Simple SI visualization



Integration with external tools



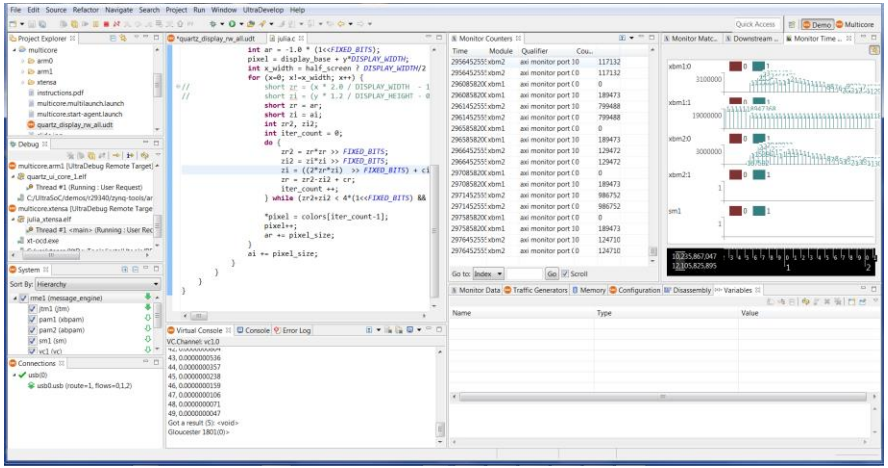
Key Features

Non-intrusive	Monitoring does not impact system performance. Instrumentation (light intrusion) seamlessly incorporated.
“Smart” monitors	Detect items of interest in hardware, at wirespeed. Massively reduce trace bandwidth & memory. Home in on problems efficiently
Protocol-aware bus monitors (AXI, ACE, ACE-lite, OCP, OCP 2.0, CHI etc.)	Identify specific transactions; easily spot problems
Full support for all standard processors (ARM, RISC-V, MIPS, Xtensa, CEVA, etc.)	Easily support heterogeneous architectures; “mix & match” across vendors; fix hardware, software or HW+SW integration problems
Message-based protocol	Easy to place & route; extensible & versatile; allows local processor for “autonomous” control in the field
Powerful status monitor	Configurable smart logic analyzer for custom logic
Secure	Powerful security architecture
Bare Metal Security	Provides for observation of target system in order to raise ‘alarm’

Use Cases

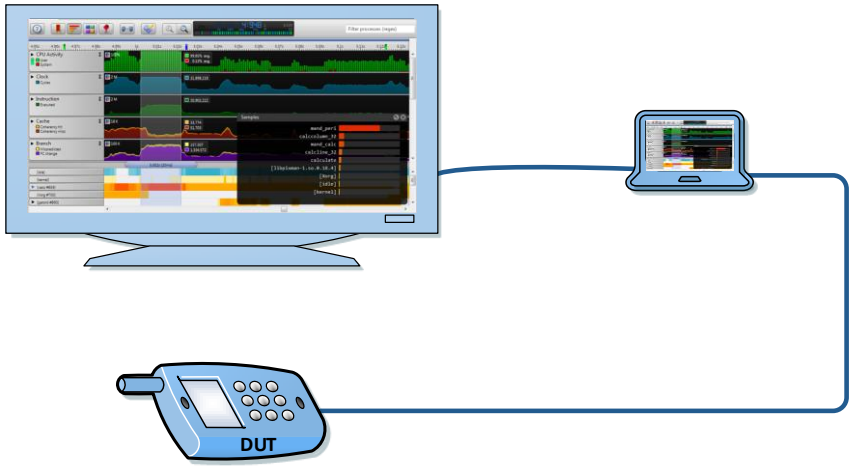
Classic Debug

- In this case the SoC may be on a prototype board or in the final product form.
- This allows for device validation and bring-up.
- Typically done with board attached to work station.
- CPU breakpoints, starting, stopping of software executing on the SoC.
- More and more of the system will be integrated (brought up) and exploration of the whole SoC, under realistic conditions, takes place.



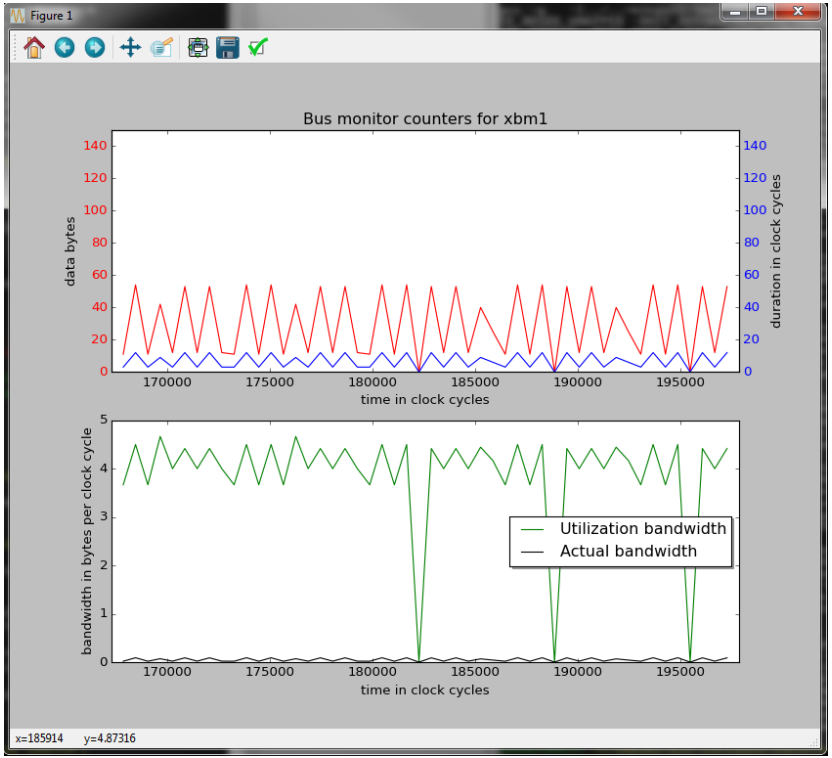
In field debugging and analysis

- In this case the SoC is in the final form and issues such as integration of the software can be debugged.
- The performance of the system can be analysed.
- The software being used could be the IDE as shown previously or specific views of key flows of data through the system.
  - These could be traffic to the memory controller
  - DMA completion times
  - Depth of FIFOs in RF interface
  - Performance of processing engines within the SoC
  - Cache behaviour
  - Etc.
- This can be used to help diagnose why a product has ‘hung-up’ in the field. During operation the device has been continuously capturing trace in circular buffers in the monitors. This effectively gives a system wide core dump.
  - Trace data is extracted from the device and analysed and replayed to give the last N transaction before the failure occurred.
  - The device does not need to be attached as the trace could have been extracted in the field and shipped to the manufacturer



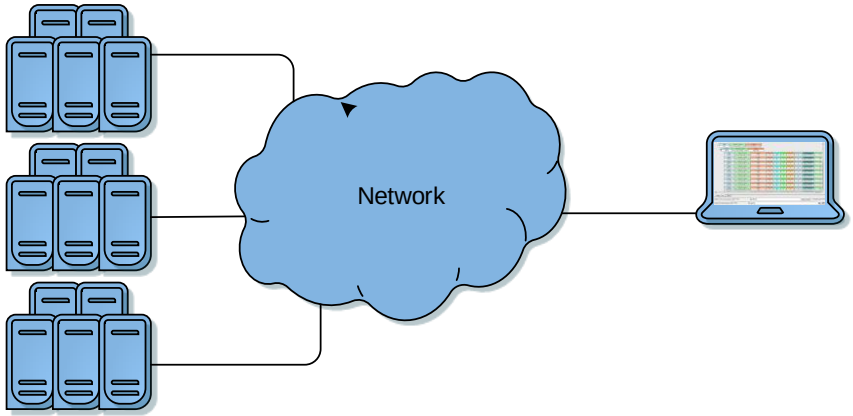
In field analysis

- The areas of interest can be extracted from the system-core dump and specific views created which can be analysed by domain specific engineers
  - These could be memory controller designers, RF interface designers etc.
- Traces extracted from the field can be used for the next generation architecture of the SoC



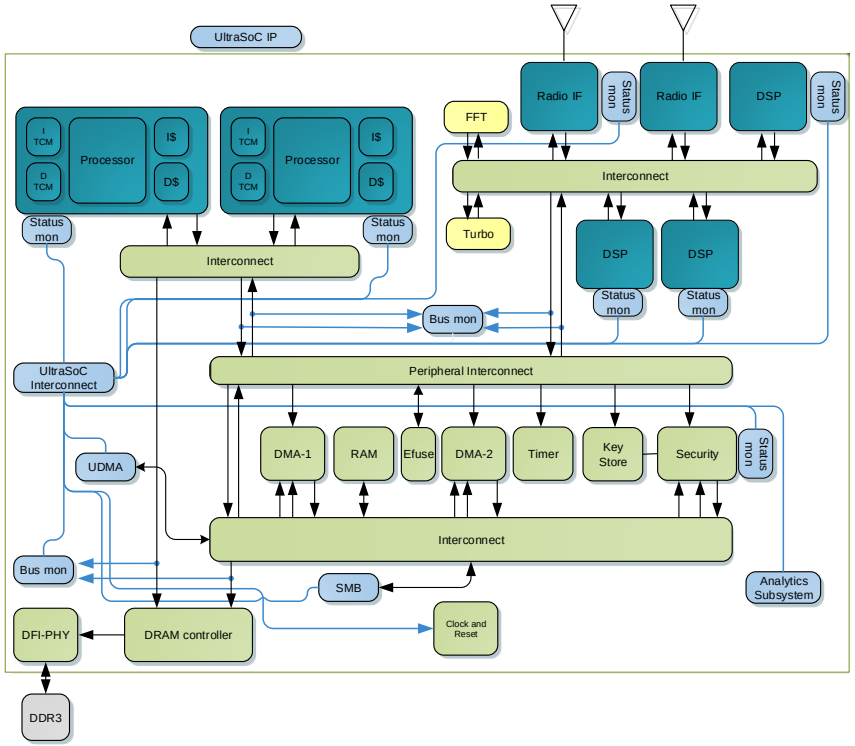
Corporate and IoT use – Performance and Security

- Monitoring of server farms
  - An example would be observing utilisation of the individual servers and the resources such as memory and disks
  - DDoS can be reported back to root/base.
  - Security and safety can be monitored in a similar manner
  - Updates would be maintained by the root/base.
  - Any breaches of security can be reported back to base.



Standalone and unconnected use

- In this there is a self contained Analytics Subsystem.
  - Any communication, if required is done over the air.
  - Many systems will not even have wireless connection
- Detect unauthorized access
  - Eg processors reading from key store
  - Eg Attempt to read decrypted boot code
- Update audit & verification
- Scan internal/external regions
- Detect frequent access / DoS
- Ensure system operates in the ‘bounds of safety’.
  - If any divergence, invoke fail safe state



Summary

- UltraSoC provides a complete advanced universal on-chip analytic and debug platform
  - Full visibility of whole SoC
  - Non-intrusive
  - Independent provider enabling free-selection of IP
  - Multi-vendor and multi-processor in one environment
  - USB connectivity for faster debug or I/O constrained devices
  - Advanced analytics: forensics, optimization, dynamic, power saving
  - Bare metal security
  - Low-power and power-down; power domains & clock domains
  - Full support for large heterogeneous SoC
  - Fully message-based communication
  - Data-flow management and security
  - Silicon proven