Joined up debugging and analysis in the RISC-V world

RISC-V Workshop November 29-30 2016

Agenda

- Some obvious statements
- Key Requirements
- Some examples of Performance analysis and Debug
- Use cases
- Demos

Some obvious statements

- SoCs have become increasingly complicated and they are not going to get simpler.
  - Contain several processors, from different vendors
  - Contain 100s of SIP
  - Contain complex interconnects
  - Software created by large disparate teams.
  - All this has to successfully work together
- Debugging is more that just Run-control
- It is more than just CPU centric information such as instructions trace
- These are important but are only parts of the problem
- In order for RISCV to be successful it must be useable in systems constructed as above.

Key requirements

- A vendor-neutral debug 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

UltraSoC Technologies

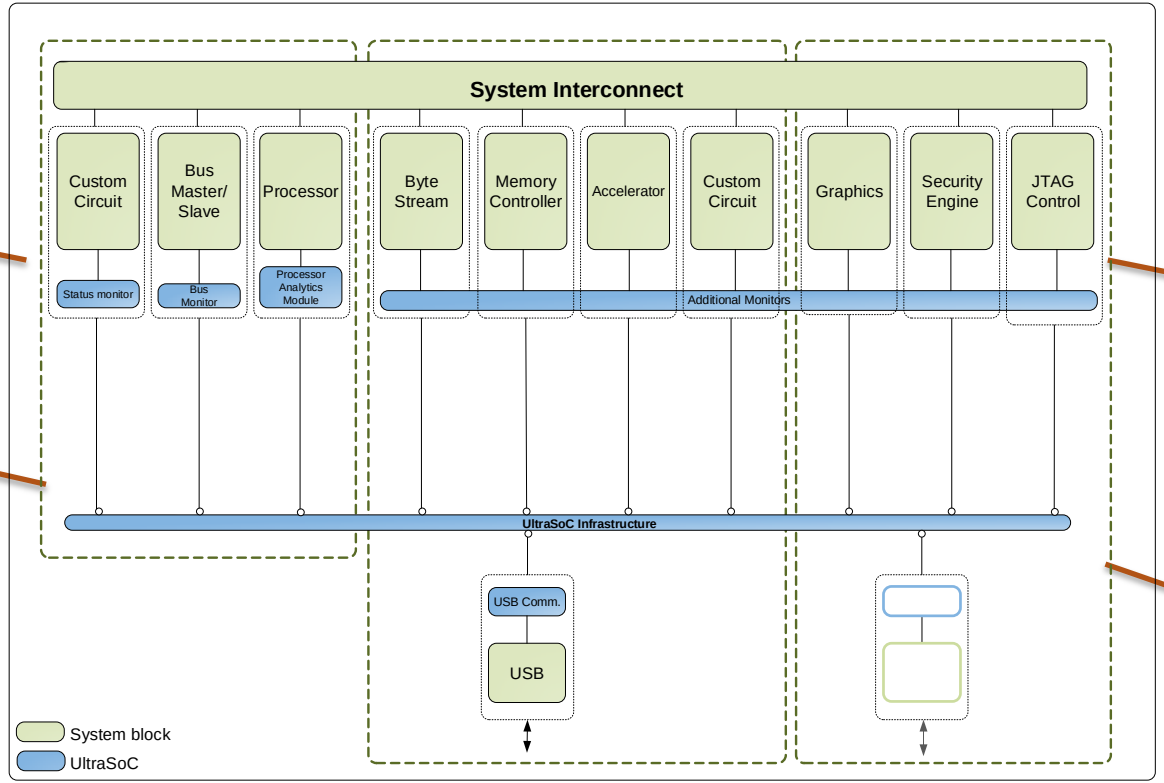
- VC funded start-up based in Cambridge UK
- Members of the foundation.
- Part of the Debug Task Group
- Provides Silicon IP + tools
  - System wide On-chip debug, optimization, analytics, forensics, bare metal security
- Partnership with other IP vendors.
  - e.g. IMG, Ceva, Cadence, Codasip, Baysand
- Partnership with leading tool vendors
  - e.g. Lauterbach, Ceva
- Mature, silicon proven product

Advanced 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

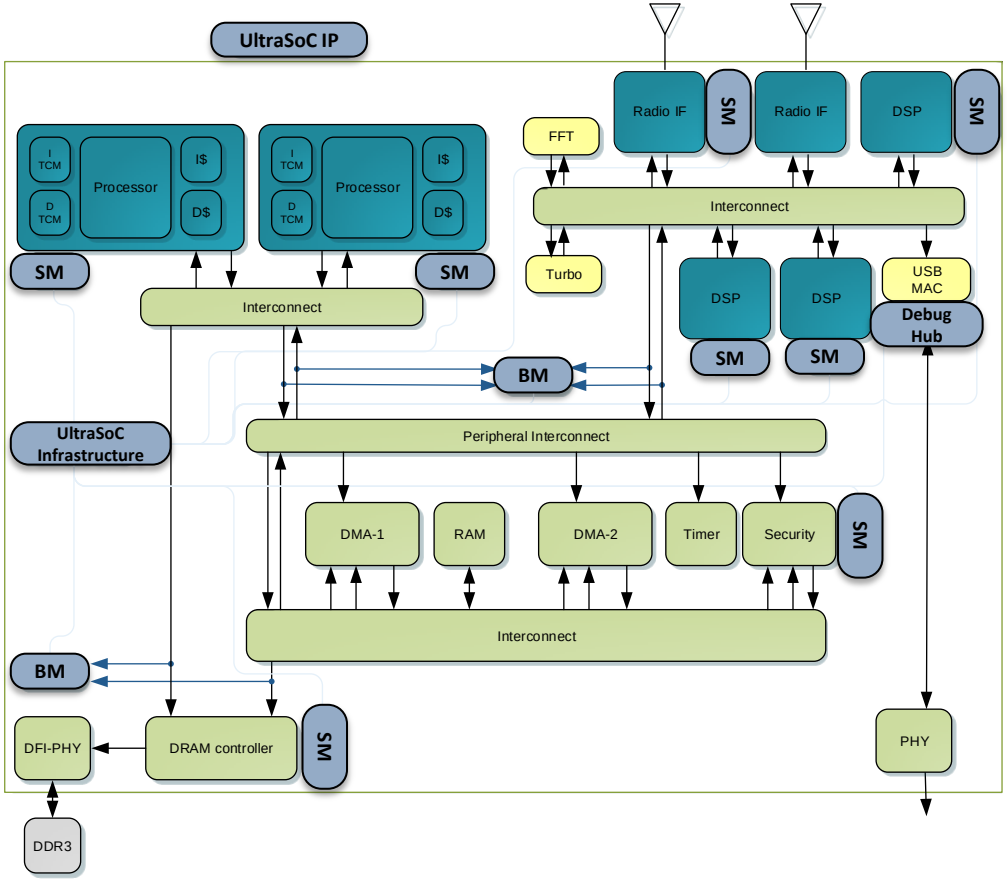


Portfolio of configurable modules, optimized for different system IP blocks

Supports subsystems with different power domains, clock domains

Some examples of Performance analysis and debug

Example of UltraSoC Enabled SoC

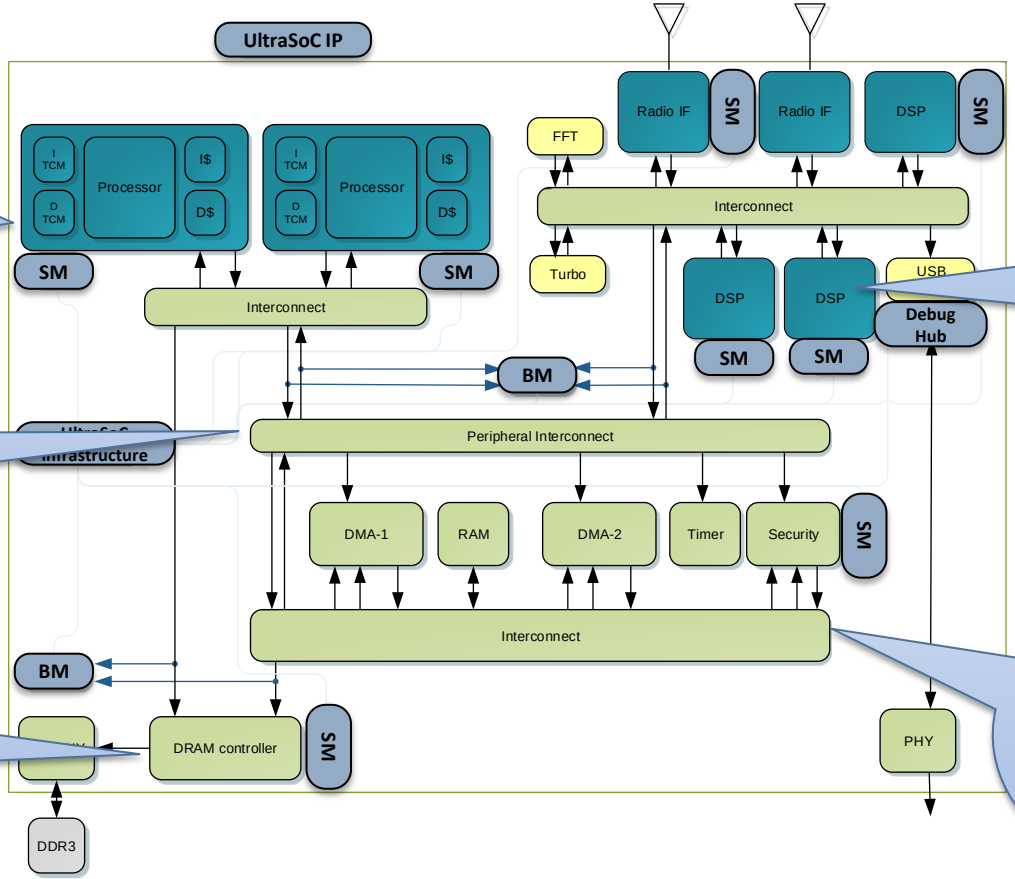


Example Problems UltraSoC Solves

Why is the CPU not performing as fast as expected?

Why do some DMA transfers take too long?

What is going on with my memory controller?

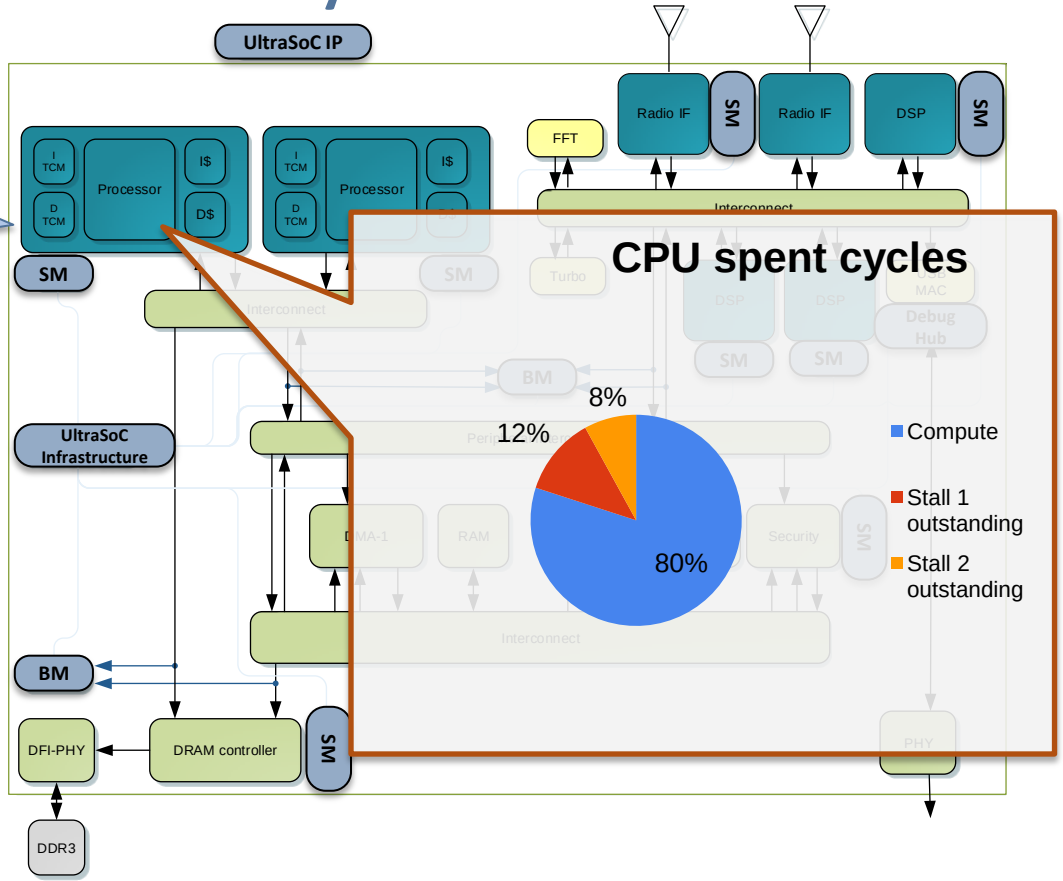


What is the mismatch between the host & the DSP?

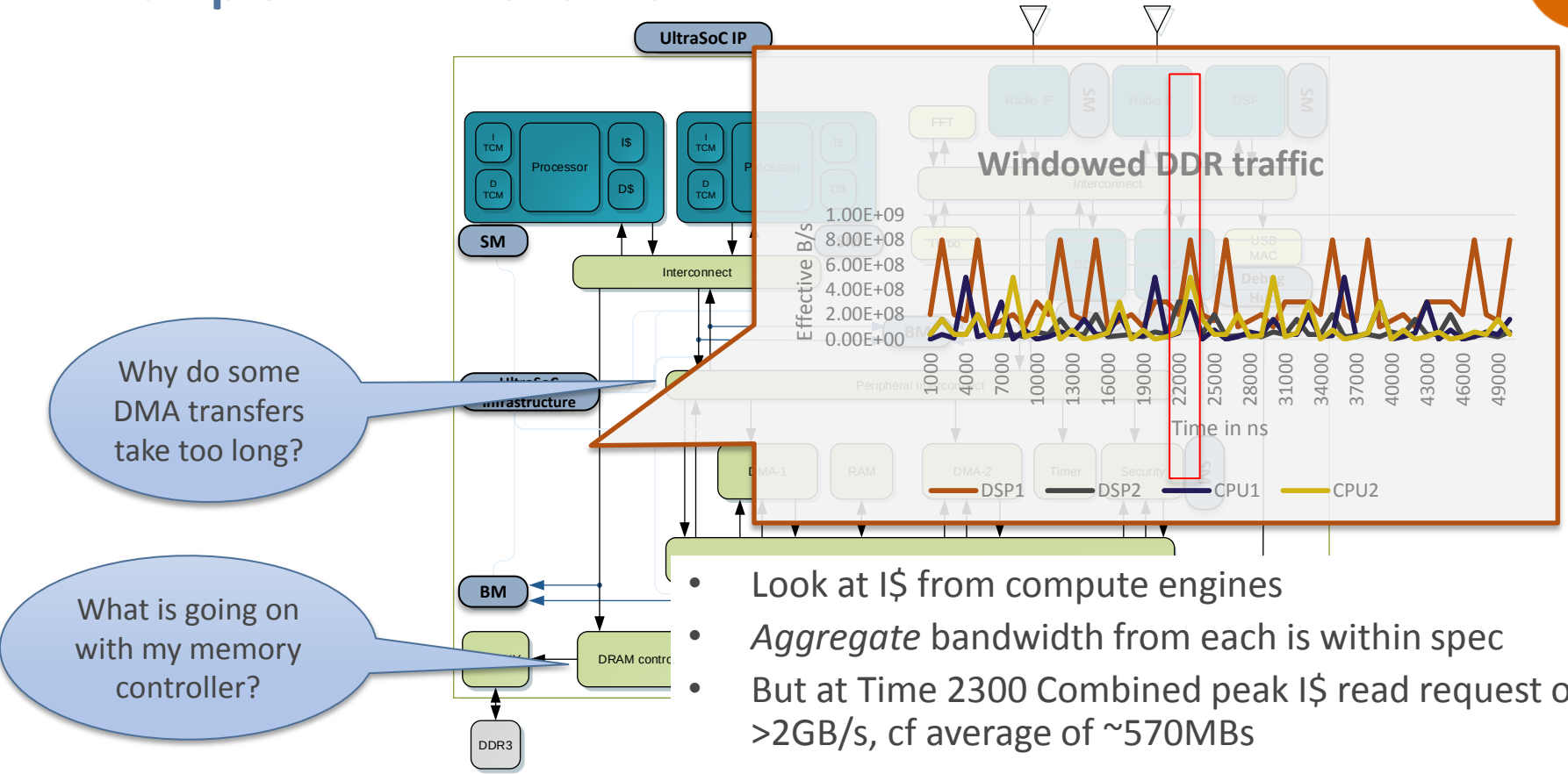
Why does the system hang or deadlock on rare occasions?

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?

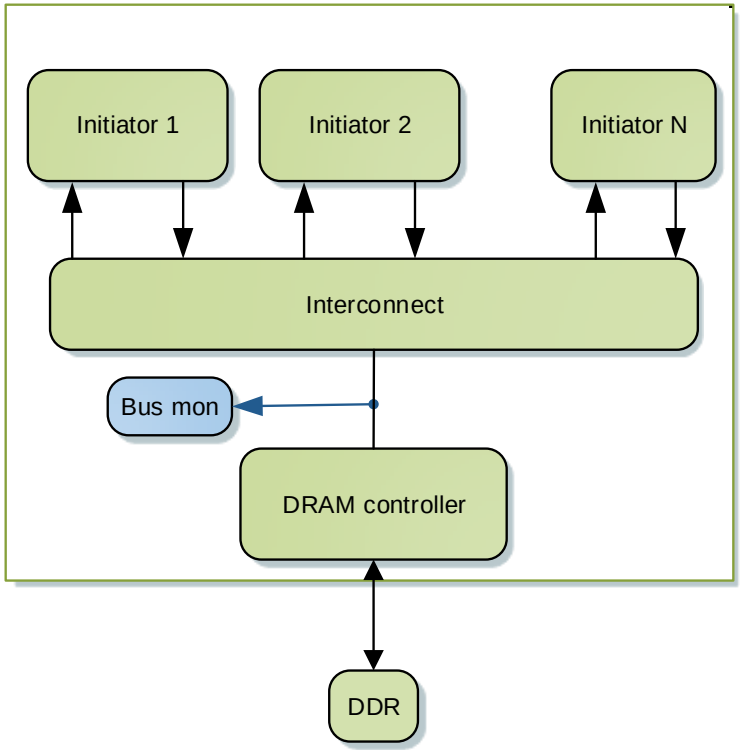
What is going on with my memory controller?

- 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

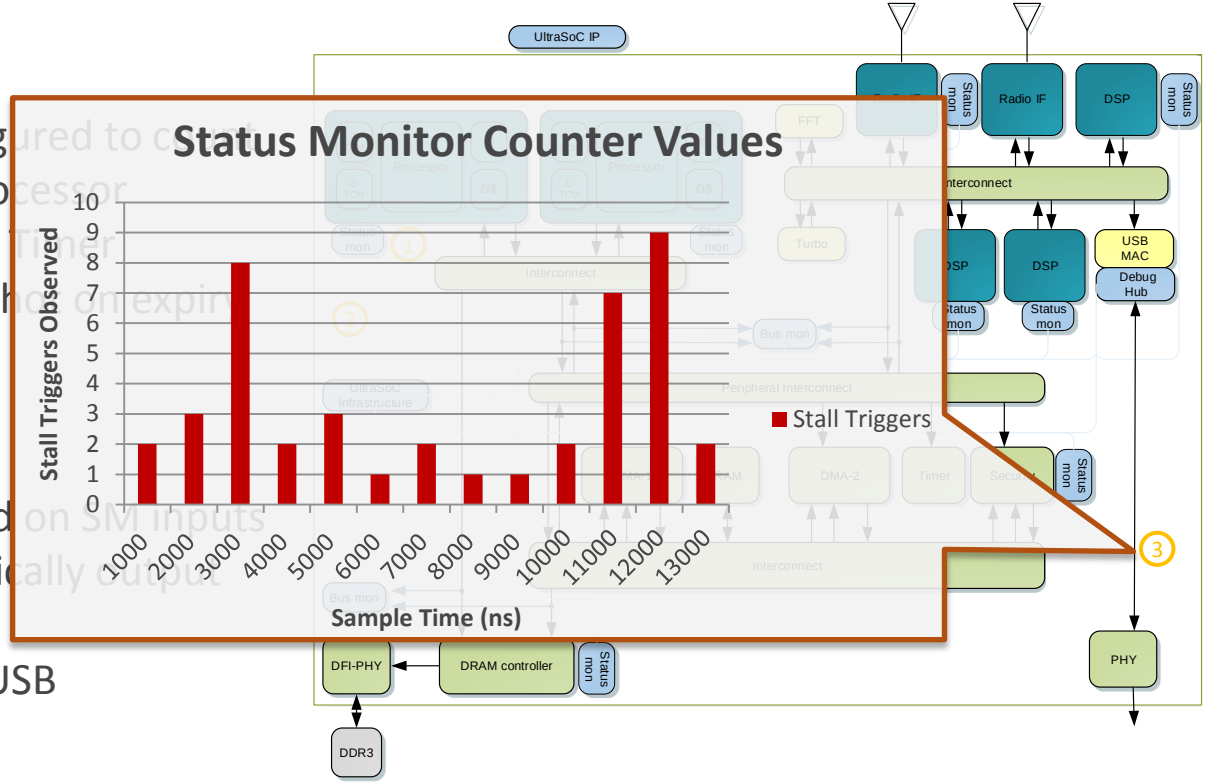
Metrics Generation – Example 1

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

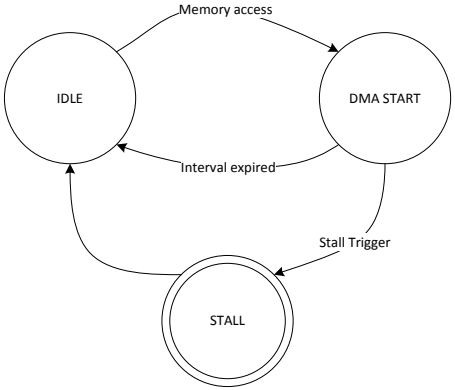
1. Stall trigger observed on SM inputs
2. Counter data periodically output from SM
3. Data traced out via USB



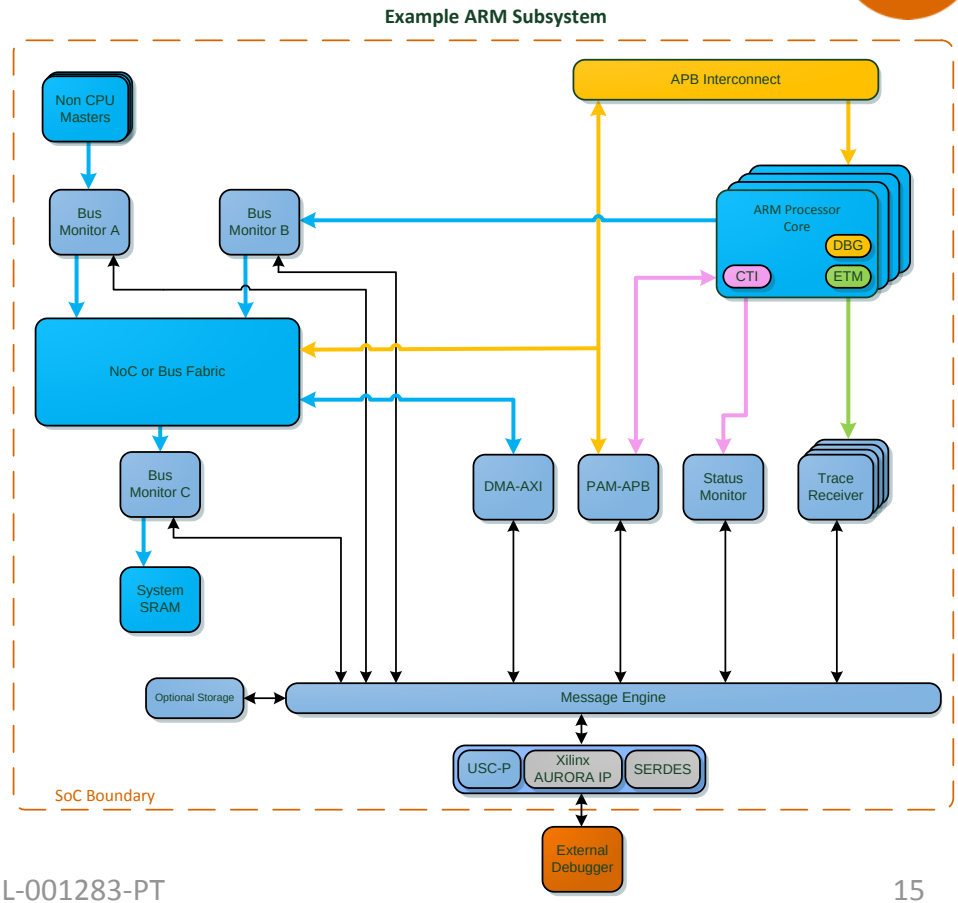
Cross Triggering – Example 1

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:



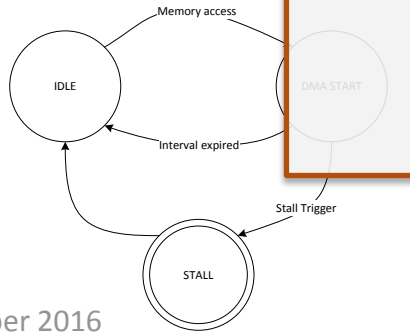
- Output trigger from SM when entering the STALL state
- Configure Trace Receiver(s) to enable tracing on receipt of trigger



Cross Triggering – Example 1 (cont)

Data Flow

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



Example ARM Subsystem

Non CPU Masters

APB Interconnect

ARM Processor Core
DBG
CTI
ETM

Bus Monitor A

Bus Monitor B

NoC or Bus Fabric

Only capture data of interest

ATB Samples

AID ADATA

Without Cross-Triggering

With Cross-Triggering

DMA-AXI

PAM-APB

Status Monitor

Trace Receiver

Message Engine

USC-P

Xilinx AURORA IP

SERDES

SoC Boundary

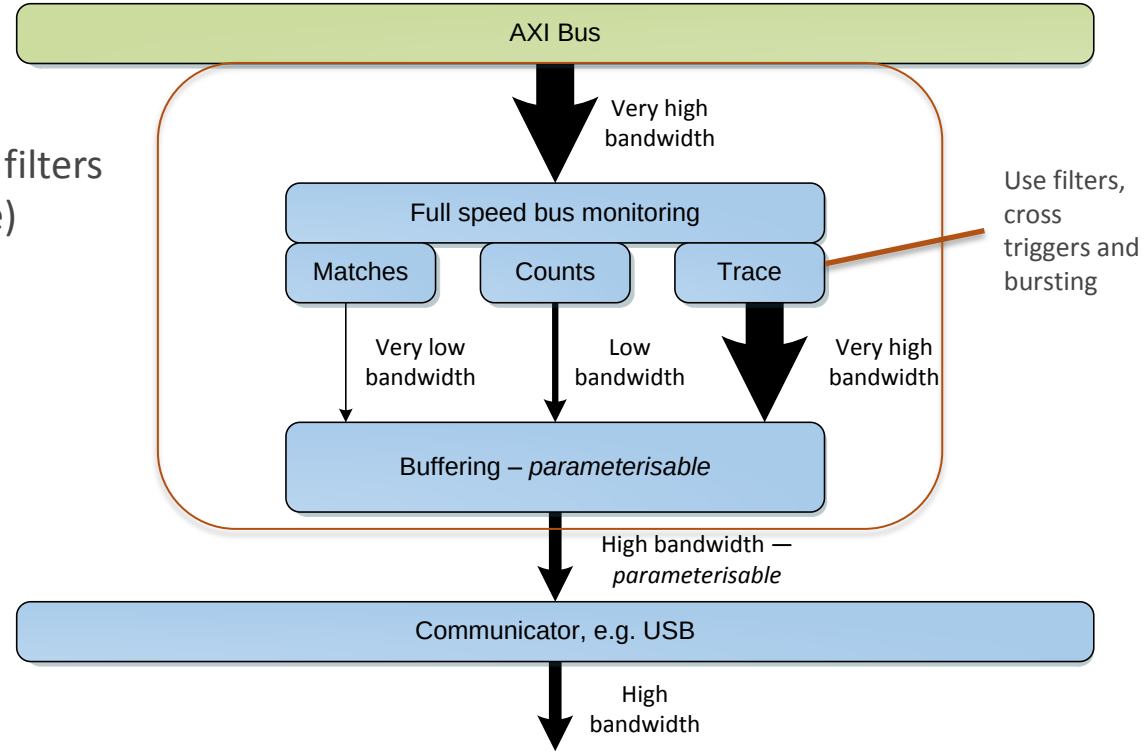
External Debugger

Key Features

Non-intrusive	Debug does not impact system performance
“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’

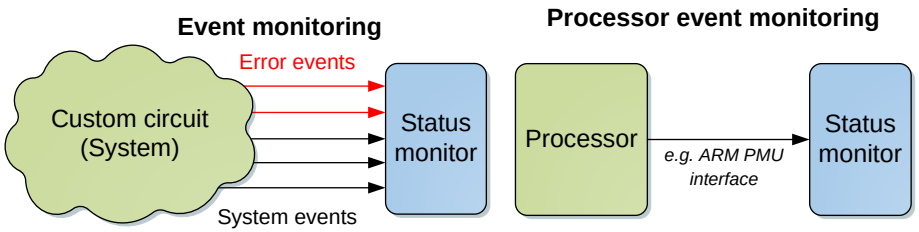
“Smart” Modules Optimize Bandwidth

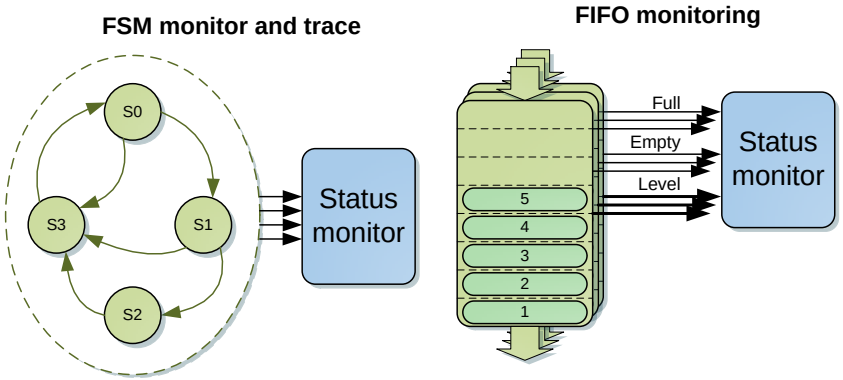
- Configured with filter, match logic for triggers
- Configurable buffer size & number of filters
- Gather statistics (best, worst, average) in hardware at wirespeed
- Only meaningful information is sent
- Reduces trace and performance data bandwidth
- Show relevant information: focus on issues and find problems faster



Status Monitoring: Custom Logic

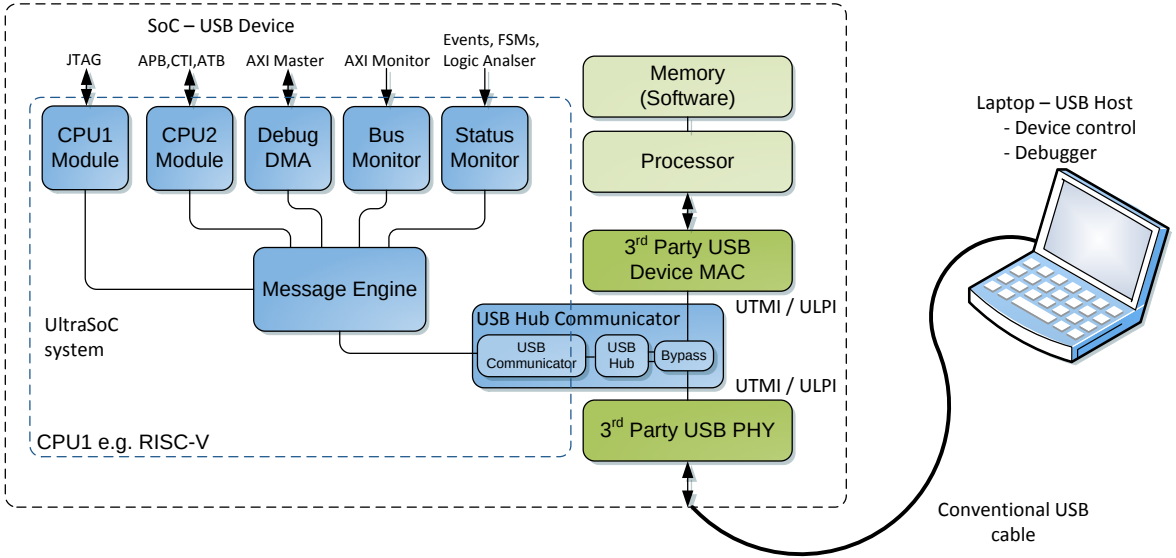
- Rich support for custom logic; arbitrary functions or common building blocks
- Event counter and trigger
  - System events
  - Processor events
  - Error events
- Accumulator
  - Capture time-averaged values
  - Average FIFO depths
  - Performance counter
- Analyser
  - State-machine trace
  - Data trace
  - Logic analyser





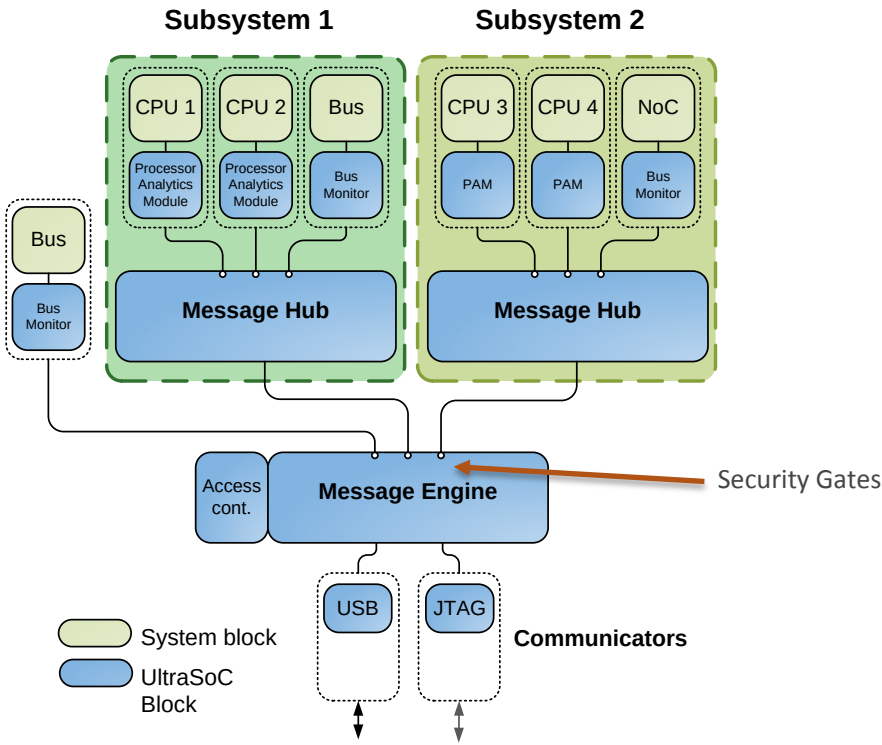
USB Debug: Share Interfaces

- Fast access to debug
- No need for dedicated debug port
- (“closed chassis debug”)
- Hub bypassed in functional mode
- Hub active in debug mode
- No extra processor needed
- No changes to software
- No interactions with system
- Security Support



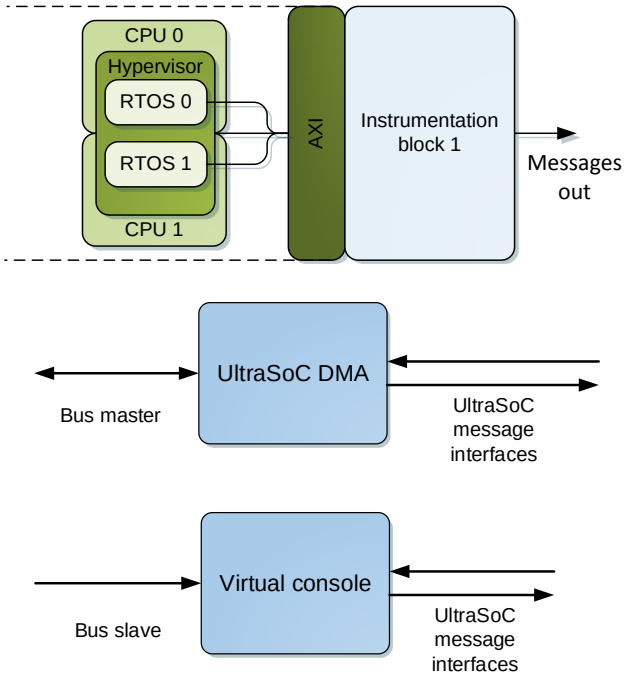
Security in the UltraSoC domain

- All messages fully encrypted
- All modules individual access control
- Secure debug interfaces
  - Challenge response authentication
  - Encrypted debug traffic
  - Enables in-field debug
- Layered security for globalised development
  - Different access rights for each user group
  - Hierarchical security gates
  - Uses proven techniques



Portfolio of 30 Modules

- Most modules are non-intrusive and “listen” e.g.
  - Bus monitor AXI, AXI4, OCP2.1, ACE, CHI, OCP etc.
  - Status monitor
  - Static Instrumentation
  - Instruction Trace
- But some are active e.g.
  - UltraSoC DMA
  - Virtual Console
- Communicators e.g.
  - Universal Streaming Communicator
  - System memory Buffer
  - Serial
  - JTAG
  - USB
  - AXI
- Message infrastructure



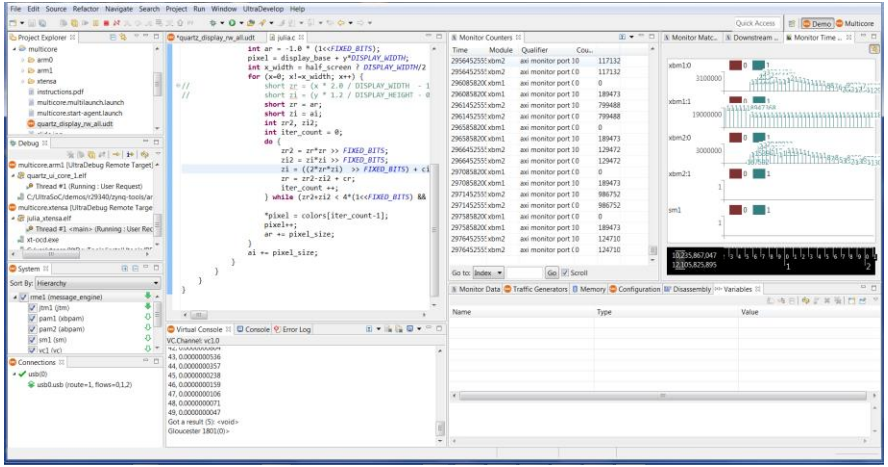
Summary

- 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
- RISC-V foundation needs
  - Standardise on Debug Interface
  - Standardise on Instruction trace

Use Cases

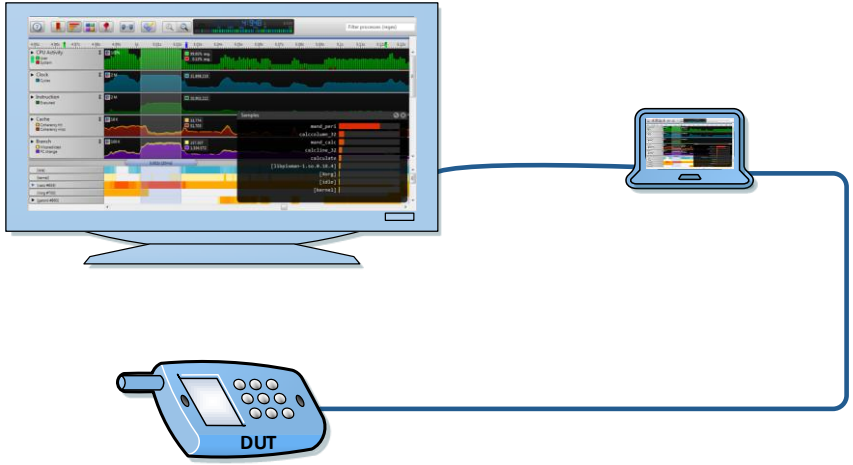
Classic Debug

- 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.



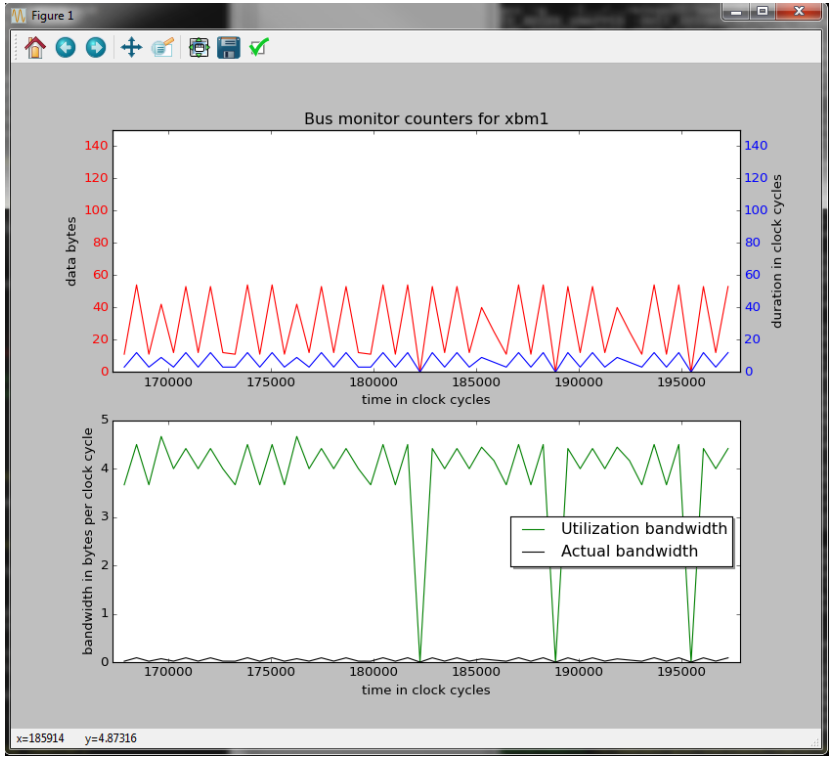
In field debugging and analysis

- 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



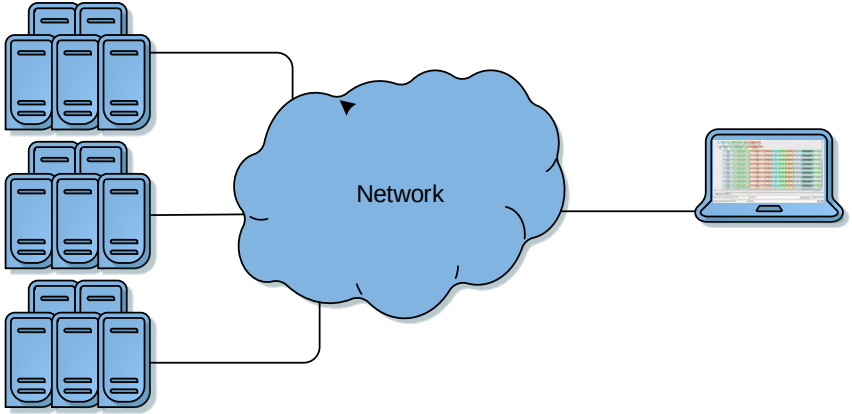
In field analysis

- 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



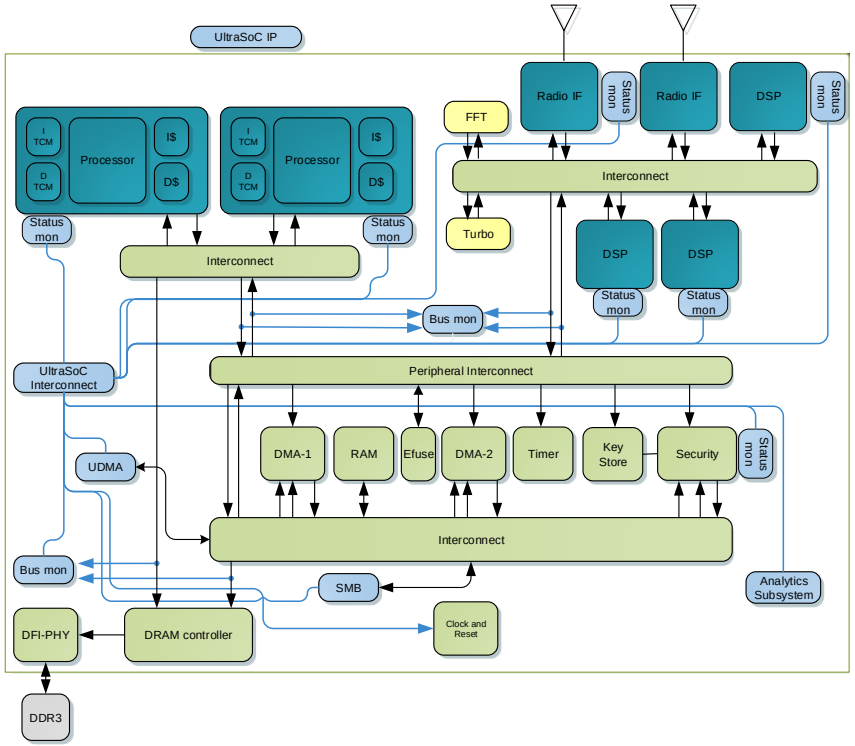
Corporate and IoT use – Performance and Security

- 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

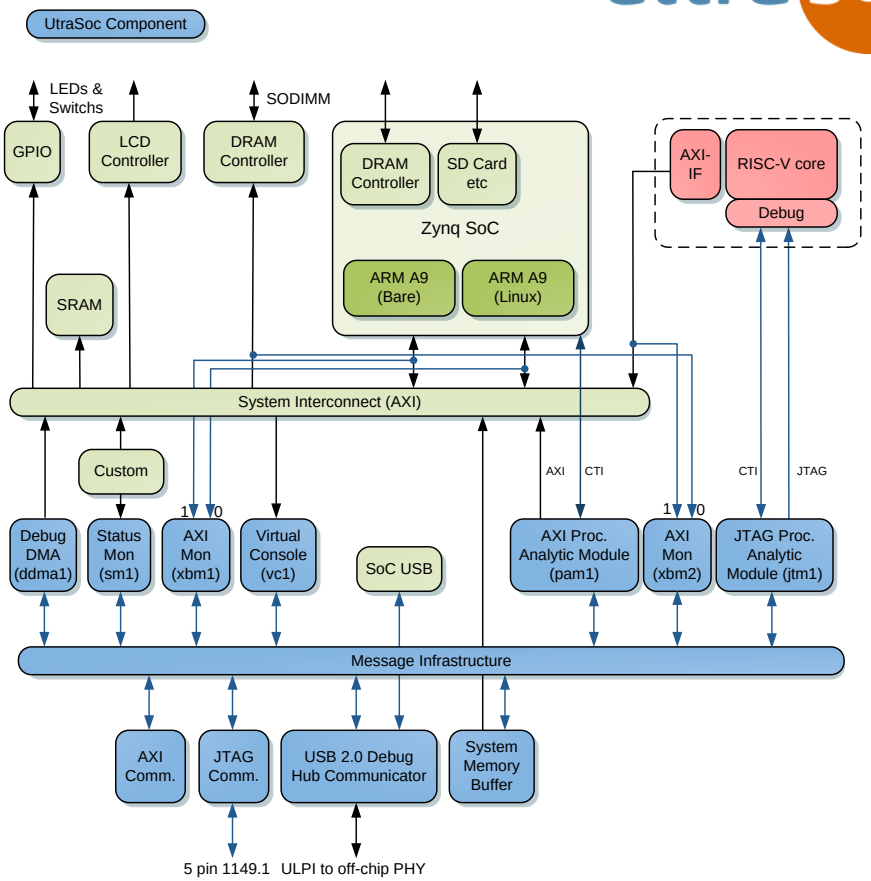
- In this there is a self contained Analytics Subsystem.
  - Any communication, if required is done over the air.
  - Many systems will not even have wireless connection
- Detect unauthorized access
  - e.g. processors reading from key store
  - e.g. Attempt to read decrypted boot code
- Update audit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



Demonstration

Demo System Architecture

- Zynq FPGA platform
  - ARM
    + Rocket RV32 RISCV
    + custom logic
- Demo shows:
  - Bus state
  - Traffic
  - Performance histogram
  - Memory
  - Processor control



Demonstration Features

- **Heterogeneous multi-core + Custom logic**
- 2 x ARM A9 + Rocket RISC-V
- Processor status & code visibility (three processors)
- AXI Bus Monitor counters (read/write bytes) & trace
- Status Monitor for custom logic (FIFO, traffic source + sink)
- UltraSoC DMA access (read/write to system memory)
- Heterogeneous CPU run and halt
- USB 2.0 Debug Hub connectivity
- Deadlock detection

UltraSoC IDE

Visibility of software

Bus activity

Control configuration

File Edit Source Refactor Navigate Search Project Run Window UltraDevelop Help

Project Explorer: multicore, arm0, arm1, xtensa, instructions.pdf, multicore.multilaunch.launch, multicore.start-agent.launch, quartz_display_rw_all.udt

Debug: multicore.arm1 [UltraDebug Remote Target], quartz_ui_core_1.elf, Thread #1 (Running : User Request), C:/UltraSoC/demos/r29340/zynq-tools/ar, multicore.xtensa [UltraDebug Remote Targe, julia_xtensa.elf, Thread #1 <main> (Suspended : Signa, main() at julia.c:38 0x6000047a

System: Sort By: Hierarchy; rme1 (message_engine), jtm1 (jtm), pam1 (xbpam), pam2 (xbpam), sm1 (sm), vc1 (vc)

Connections: usb(0), usb0.usb (route=1, flows=0,1,2)

*quartz_display_rw_all.udt / julia.c

```
int ar = -1.0 * (1<<FIXED_BITS);
pixel = display_base + y*DISPLAY_WIDTH;
int x_width = half_screen ? DISPLAY_WIDTH/2
for (x=0; x!=x_width; x++) {
//      short zr = (x * 2.0 / DISPLAY_WIDTH  - 1
//      short zi = (y * 1.2 / DISPLAY_HEIGHT - 0
        short zr = ar;
        short zi = ai;
        int zr2, zi2;
        int iter_count = 0;
        do {
            zr2 = zr*zr >> FIXED_BITS;
            zi2 = zi*zi >> FIXED_BITS;
            zi = ((2*zr*zi)  >> FIXED_BITS) + ci
            zr = zr2-zi2 + cr;
            iter_count ++;
        } while (zr2+zi2 < 4*(1<<FIXED_BITS) &&

        *pixel = colors[iter_count-1];
        pixel++;
        ar += pixel_size;
    }
    ai += pixe
  }
 }
}
```

Expression	Type	Value
pixel_size	int	20

Name : pixel_size
Details:20
Default:20
Decimal:20
Hex:0x14
Binary:10100

Virtual Console / Console

```
multicore.start-agent [UltraDebug Agen
filter: modules enumerated: 0
reusing index
done
filter_compile(time,(null))
filter: modules enumerated: 0
reusing index
done
```

Monitor Counters

Time	Module	Qualifier	Cou...	
171645255	xbm2	axi monitor port 10		0
171645255	xbm2	axi monitor port (0		180544
172085820	xbm1	axi monitor port (0		186044
172085820	xbm1	axi monitor port 10		118421
172145255	xbm2	axi monitor port 10		0
172145255	xbm2	axi monitor port (0		185936
172585820	xbm1	axi monitor port (0		155544
172585820	xbm1	axi monitor port 10		118421
172645255	xbm2	axi monitor port 10		0
172645255	xbm2	axi monitor port (0		145736
173085820	xbm1	axi monitor port (0		98748
173085820	xbm1	axi monitor port 10		118421
173145255	xbm2	axi monitor port 10		0
173145255	xbm2	axi monitor port (0		98460
173585820	xbm1	axi monitor port (0		98344
173585820	xbm1	axi monitor port 10		118421
173645255	xbm2	axi monitor port 10		0

Monitor Matc... / Downstream ... / Monitor Time ...

xbm1:0 (3100000), xbm1:1 (19000000), xbm2:0 (3000000), xbm2:1, sm1

10,235,867,047
12,105,825,895

Disassembly / Variables

		Value
		-10957
		19574
		2
		-12663
		14655
ar	int	-7032
x_width	int	400