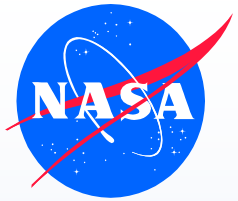
National Aeronautics
and Space Administration

2019 NEPP ETW



Radiation Effects on ARM Devices

Steven M. Guertin

Jet Propulsion Laboratory / California Institute of Technology

Pasadena, CA

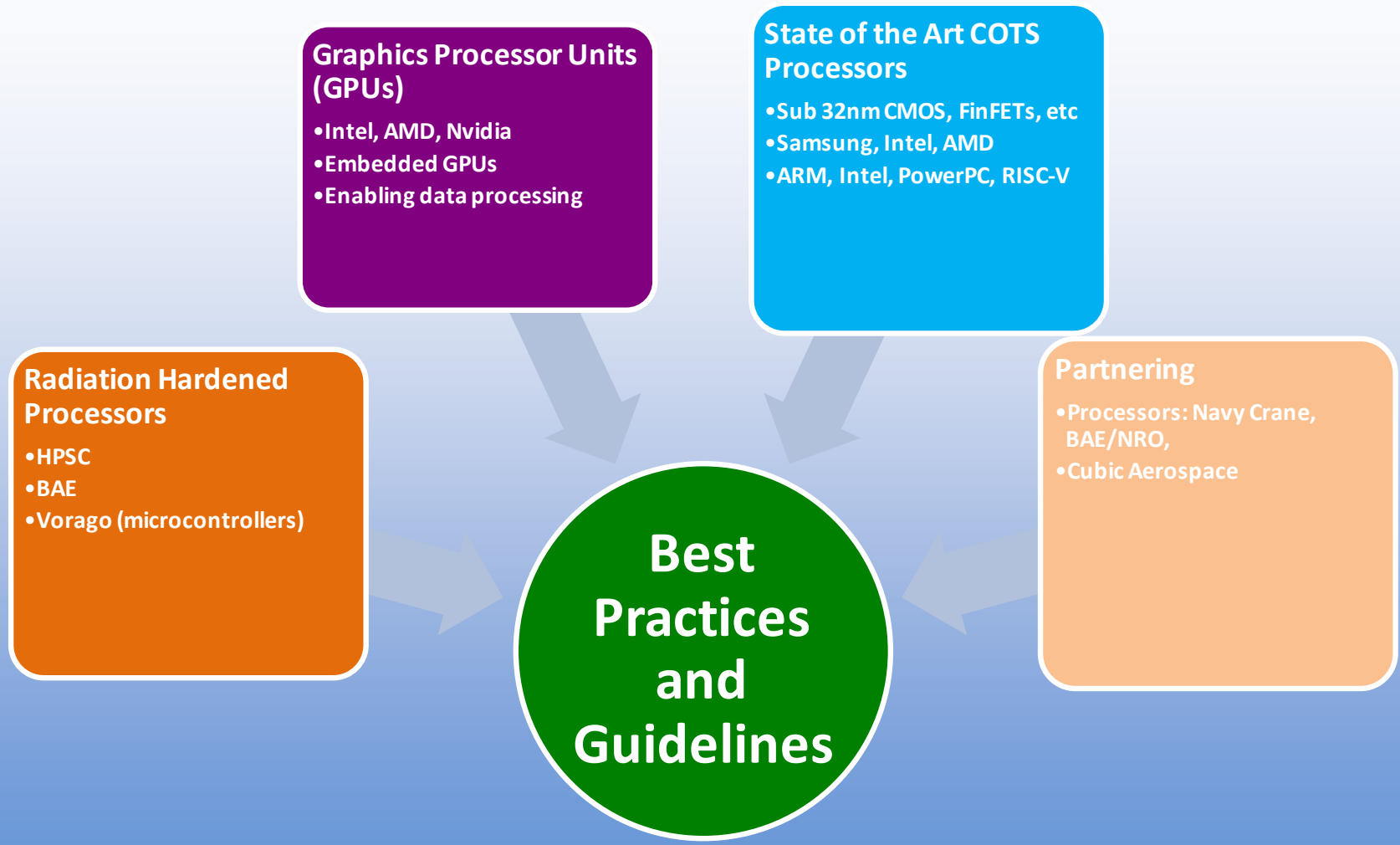
This work was performed at the Jet Propulsion Laboratory, California Institute of Technology,
Under contract with the National Aeronautics and Space Administration (NASA)
This work was funded by the NASA Electronic Parts and Packaging Program (NEPP)

Outline

- SOCs/Program Perspective
- Goals
- Background on ARM and Target Devices
- Radiation Effects Evaluation Methods
- Results – Snapdragon 835, 845
- Looking to the Future?
- Conclusion

National Aeronautics and Space Administration

NEPP – ARM Processors and Context



Potential future task areas:
artificial intelligence (AI) hardware, Intel Stratix 10

Task Partnering

- Engaging in collaborative efforts:
  - NSWC Crane
  - Carl Szabo, Ed Wyrwas, Ted Wilcox, and Ken LaBel, GSFC
  - Jeff George, Aerospace Corporation
  - Larry Clark, ASU
  - Heather Quinn, LANL, and other members of the Microprocessor and FPGA Mitigation Working Group
  - Sergeh Vartanian, Andrew Daniel, and Greg Allen, JPL
  - Vorago Technologies – collaborating on hardware/plans
  - Paolo Rech – GPU/Applications, UFRGS
  - Intel – informally
  - BAE Systems
  - AFRL

- Looking for additional collaborators
  - Tester side – are you testing processors?
  - Manufacturer side – knowledge or hardware support
  - Application side – specific applications…

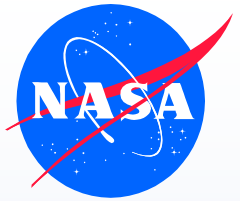
Focus Categories

- **Architecture – to support evaluation and use of processor architectures throughout NASA, including processor types and FPGA/Soft processors**
  - **ARM-specific effort – Can we identify appropriate methods for ARM devices – identify architecture-specific and implementation-specific issues?**
- Implementations – to support evaluation and use of primary form-factors
- **Fabrication Facilities/Technology – to obtain information on fabrication facilities and related technology (e.g. Samsung 7nm, 3D, etc.)**
- Application/Use Case – to support ways of using devices for different NASA needs
- Develop data on specific devices/Methods for evaluation – support actual flight use, and understand that in many cases the project will have to evaluate their own part (but we can provide guidance)
- Manufacturers – to have an up-to-date tool set for understanding devices from various manufacturers.
- Collaborations – to engage manufacturers when they are available or can work with us, we want to harness this
- **Test Reports, Test Methods, Guidelines, BOKs**

National Aeronautics and Space Administration

Advanced Processors - Commercial

- collaborative with NSWC Crane, others



14nm CMOS Processors (w/Navy Crane)
- Intel 14nm FinFET commercial
- Samsung 14nm LPP Snapdragon 820

Closing Out on processor side – see GPUs later today.

10nm CMOS Processors
- Samsung 10nm Snapdragon 835, 845

Radiation Testing

- Intel 10nm

Radiation Testing

7nm CMOS Processes
- Samsung 7nm (Qualcomm)

Radiation Testing

GPUs (See Ed's talk)
- High Performance GPUs
- (12-20 nm)

Radiation Testing

- SOC GPUs
- (7-20 nm)

Radiation Testing

FY18 FY19 FY20 FY21

National Aeronautics and Space Administration

Advanced Processors – ARM & Flight/RHBD - collaborative with BAE Systems, HSPC, others

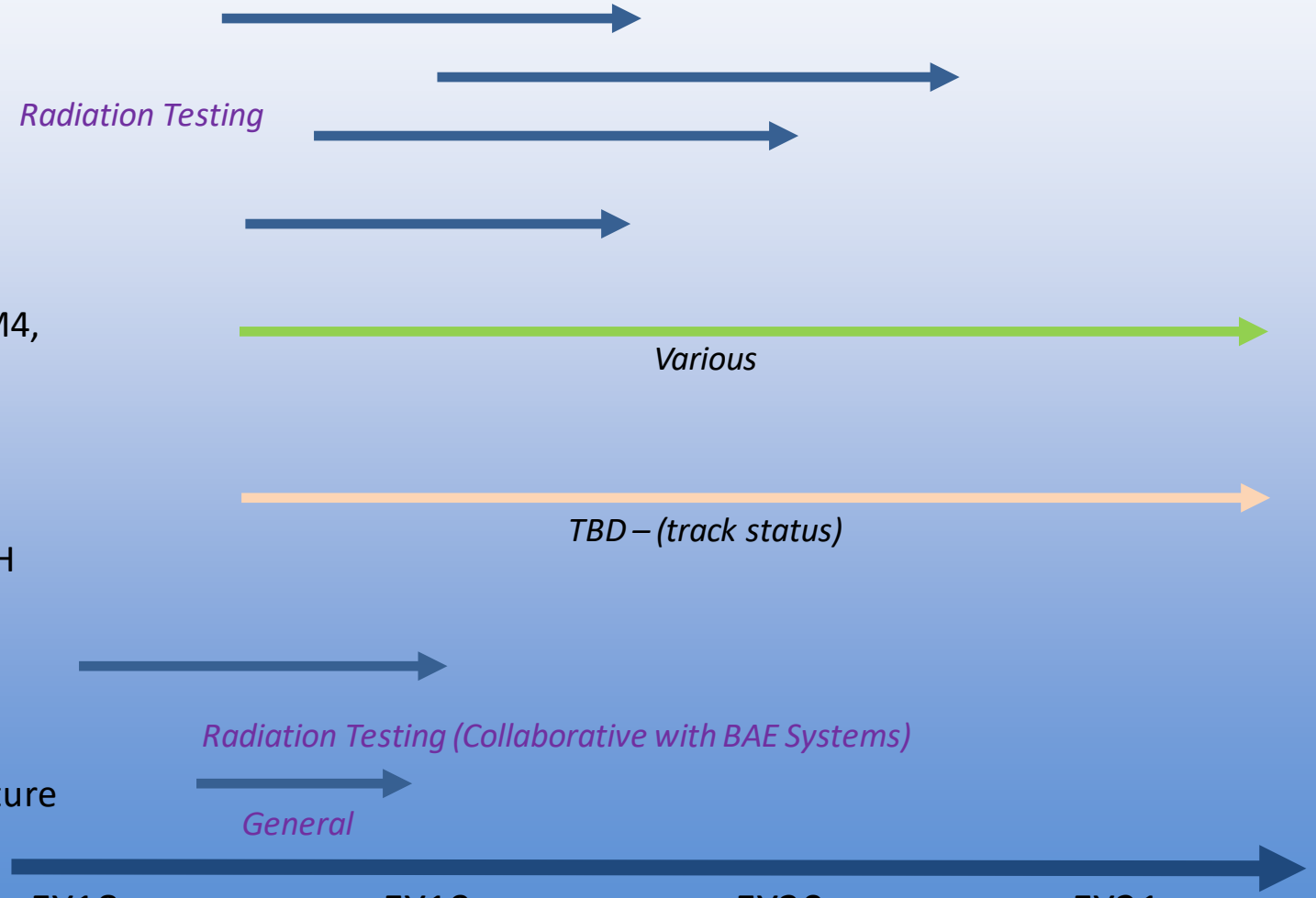
Compare soft- vs. hard-ip ARM devices
- A5 Microcontroller SAMA5D3
- A5 FPGA hard-IP
- A5 soft IP

ARM Fault Tolerance
- Dual R4 or R5

Additional Vorago/Cobham M0, M4, R5, etc….

High Performance Spaceflight Processor (HPSC)
- Joint NASA-AFRL Program for RH multi-core processor

RH Processor
- BAE Systems RAD5510/5545
  - Leverages P5040 architecture

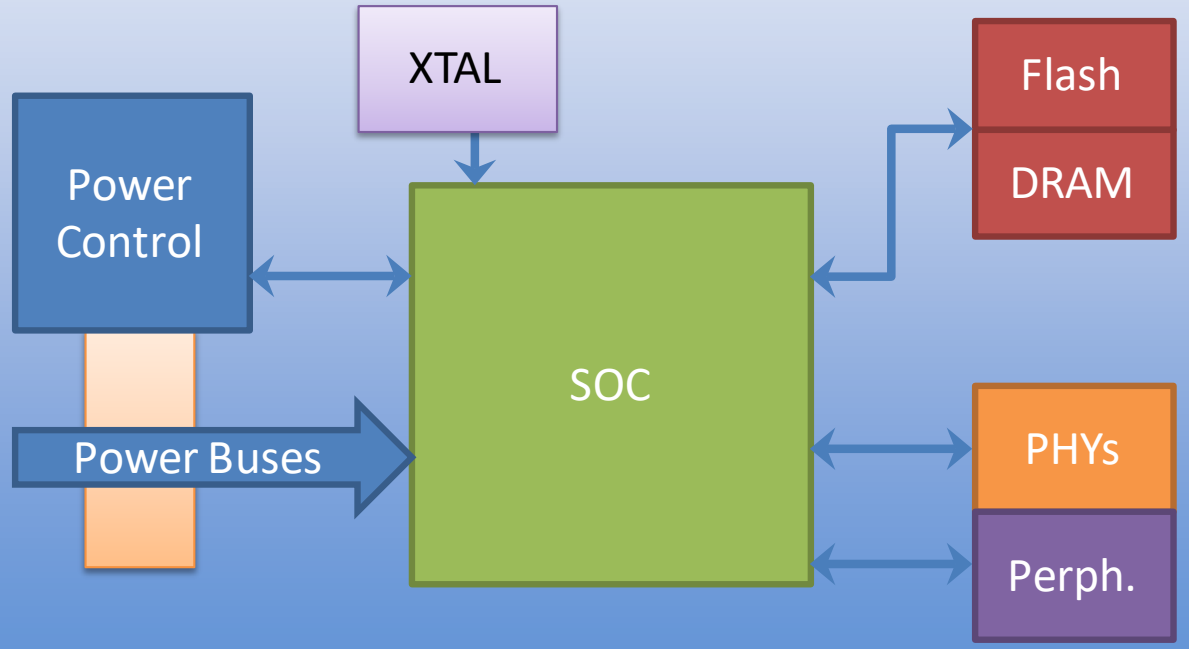


Primary Goals

- We are working to get a handle on the architectural implications of SEE on the ARM platform
- ARM devices (even newer high performance cores) are implemented in many different silicon environments
  - Cell phone processors at 7nm, microcontroller devices at 100s of nm,
  - Provides opportunities to directly compare process nodes, and to get data on the newest process nodes
  - Not all implementations easily explored (e.g. “custom” ARM cores in phones)
- We would like to develop an approach that intelligently identifies what issues are inherent to the architecture, and what issues are primarily due to the fabrication
- Further, ARM cores (not just processors) have a significant number of configuration parameters that can affect SEE
  - Includes FT settings
  - In some cases devices are very well documented
  - Indication that ARM is interested in knowing how well their devices work in order to improve reliability options

What is an SOC

- Anything that combines multiple functions to create a system
  - Does not need to be a single device that is a system by itself – SOCs may need support devices



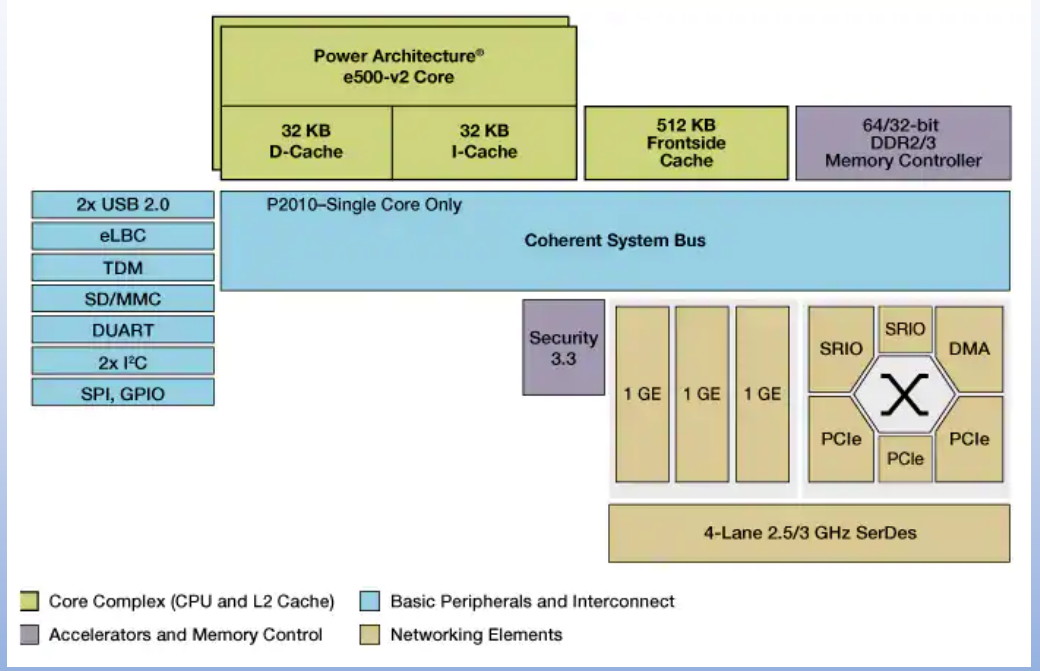
- most microcontrollers
- cell phone processors
- modern commercial processors
- BAE RAD5545
- HPSC Chiplet
- GR712, UT699/700
- nVidia Tegra series
- drone processors
- FPGAs (different topic)

Why ARM?

- Modern SOCs can't be considered stand-alone processors
  - Support devices must be considered, especially buses
- ARM or ARM support devices are used in many modern SOCs
  - Allows compare & contrast test approaches across similar-but-different devices
- In many cases, the architecture is well-documented
  - Especially in FPGAs how to properly use resources

What do we know about our SOCs?

- 45 nm we had things like Freescale/NXP P2020



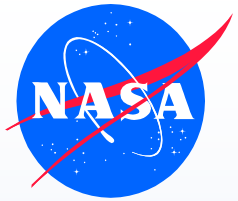
- We knew what peripherals we had on chip, capabilities, memory controllers, etc.
- Cell phone processors (with custom ARM corse) and high performance SOCs generally have very little documentation.
  - Undocumented SOCs are like undocumented boards that you can't look at.
- But ARM cores are very well documented, as are any IP instantiations into FPGAs. Further, microcontrollers also have good documentation.

Devices of Interest

- Drone processors – like Snapdragon 801
- Cell phone processors – like
  Snapdragon 820, 835, 845, 855 (7nm!)
- Microchip/Atmel SAMA5D3
  - (ARM A5 devices)
- Xilinx Ultrascale+ MPSoC, similar
- TI TMS570 or similar – for fault tolerant ARM architecture
- (Huawei, Rockchips – esp. RK1808 with neural CPUs… depending…)

National Aeronautics and Space Administration

ARM-Based Microcontrollers w/Fault Tolerance



- Primarily focusing on similarities and differences from other ARM devices, and getting actual ARM-architecture fault-tolerance data
- Devices like: TI TMS570 “Hercules” , NXP S32S / MPC57, ST Micro SPC5, Cypress “Traveo” are being produced for automotive applications.
  - For Safety Applications
- Processing Cores (ARM-R series, PPC e200z0)
- Requirements for reliable operation -40 – +165 C
- Features included are delayed lockstep CPU, ECC on internal memories (end-end EDAC), advanced BIST (M-BIST / L-BIST / A-BIST), memory protection units, fault collection, & monitoring peripherals / supervisors (clocks, buses, voltages)
- FT-task seeks to evaluate protection features for use in natural space environment.

Other Architectures

- SOCs have their own ecosystems, though many buses are common
  - AXI, AMBA, etc.
  - Memory coherency, memory mapped peripherals common to most devices
  - Multiple microprocessors and support devices
- Freescale utilizes corenet/coherency bus, standard peripherals, etc.
- ARM has a set of buses, memory controllers, etc.

- Through other efforts under NEPP, there is data on Intel architecture

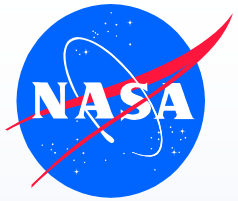
- Also of interest is RISC-V
  - ARM's SOC architecture is present in a lot of newer CPU ideas, including RISC-V

Fundamental SEE Approaches

- Ideal:
  - Obtain SEE data on individual structures
    - By direct observation of N structures
    - In the same operating conditions as normal use
    - Utilizing debuggers or specialized test code
  - Divide out (normalize) any observations to the number of targets available
  - Maximize targets being tested
- Non-Ideal:
  - Run an operating system (OS) with a specified workload
    - Count events – beware normalization
    - Count crashes…
  - Run test software under an OS
    - Count events & crashes
  - Biggest issue is normalization
- Flight Like:(???)
  - This is something of a myth, because test conditions are not flight conditions… and you can't get flight code
  - Accelerated tests are not inherently "flight-like" (e.g. latent errors)

Fundamental SEE Approaches

- Ideal:
  - Obtain SEE data on individual structures
    - By direct observation of N structures
    - In the same operating conditions as normal use
    - Utilizing debuggers or specialized test code
  - Divide out (normalize) any observations to the number of targets available
  - Maximize targets being tested
- **Also looking into ways to resolve test issues, lack of visibility,** application of data, **and limited documentation**
  - **New approaches for low level data**
  - Hybrid methods to get "flight-like" information
  - Biggest issue is normalization
- Flight Like:(???)
  - This is something of a myth, because test conditions are not flight conditions… and you can't get flight code
  - Accelerated tests are not inherently "flight-like" (e.g. latent errors)



Snapdragon Test Development/Methods

- Video playback test
- 3DMark benchmarking software
- Also porting some specific benchmark codes
  - Don’t expect standard benchmarks to be of significant benefit
    - This is an SOC, not a processor – need SOC benchmarks…
    - SOC benchmarks are inherently not well defined…
  - But exploring altering the benchmarks for modern devices – e.g. Mean Work To Failure (MWTF) based on which resources are used – DSP, multiple cores, etc

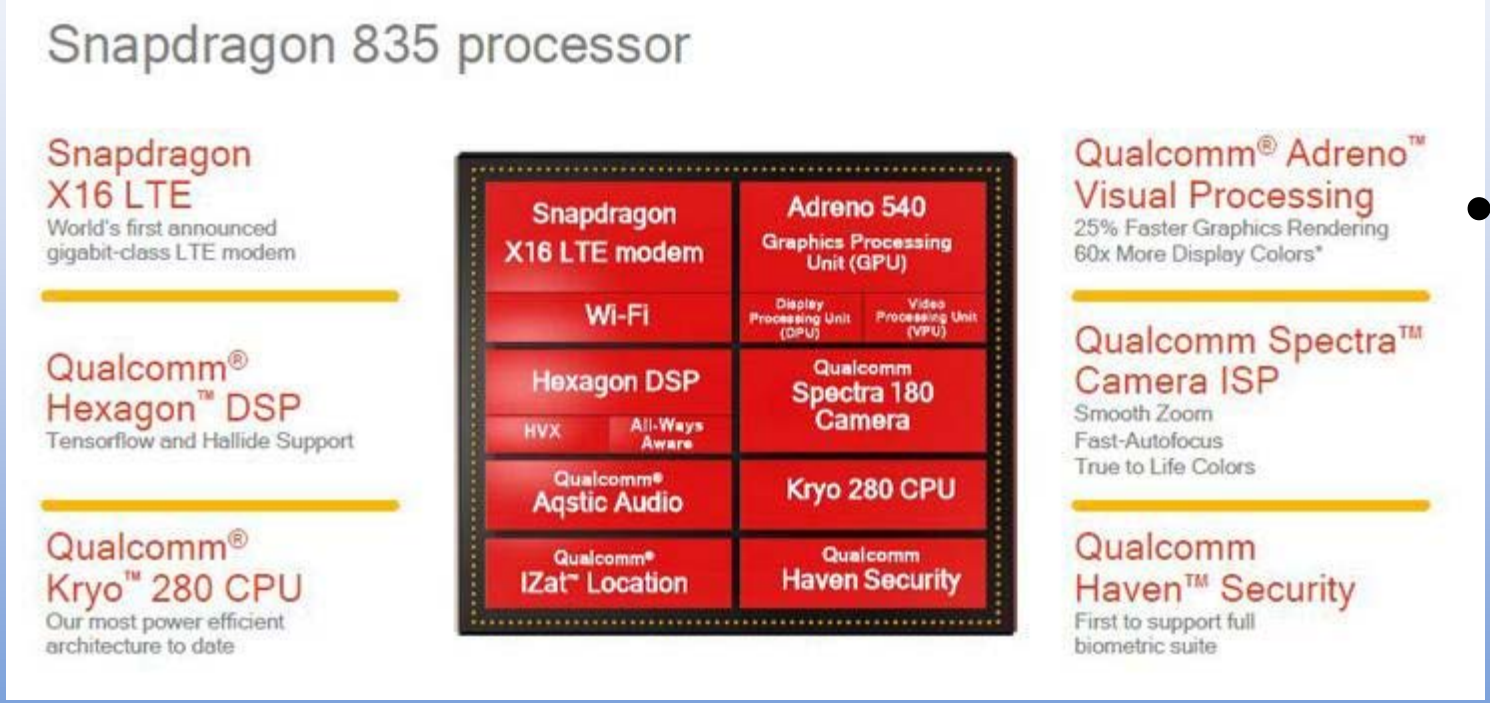
Snapdragon 835

- Samsung 10 nm
  - 8 Kyro 280 CPUs
  - Adreno 540 GPU
  - Hexagon DSP

- Using Intrinsyc's 835 Mobile Hardware Development Kit – Android only, which is not desired…
- This board uses package-on-package (they essentially all do)
- No avenue to put Linux on the board.
  - Linux gives us more ability to understand how the it actually runs…

What do we know about our SOCs?

Typical Datasheet



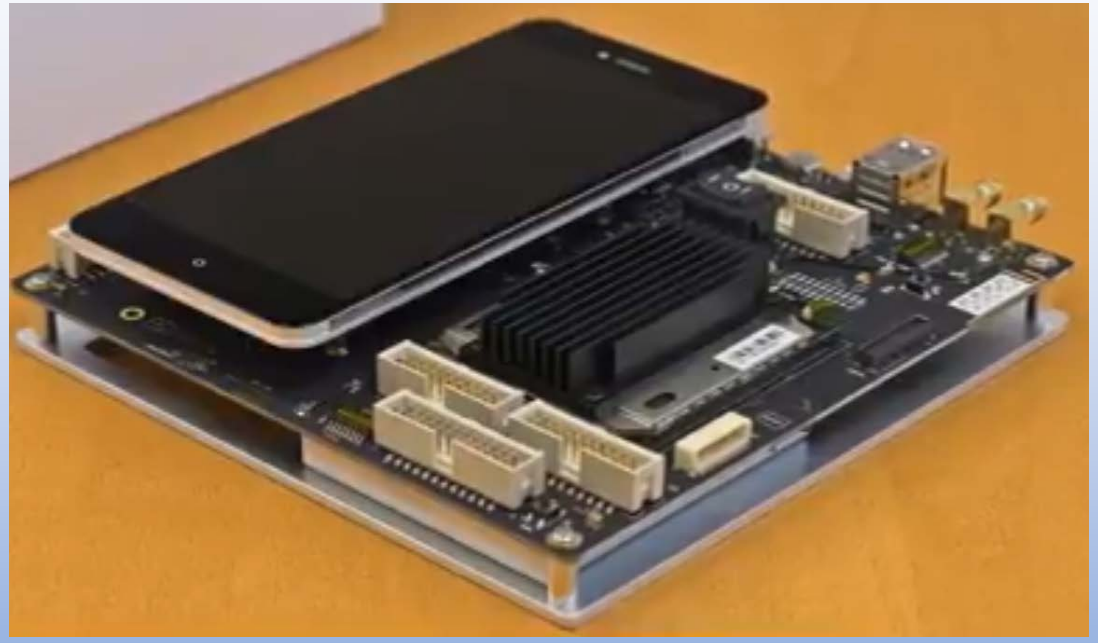
- Even the details of what chip capabilities are available on the eval boards require NDAs
  - This is not “by itself” a problem – but programs often have trouble with not sharing test data

Snapdragon 835 Results

- SEFI behavior with a cross section of about 1×10^{-4} cm^2 for LETs under 2 MeV-cm^2/mg
- Independent of the test program used (video playback or graphics benchmark program)
  - All tests operated under the Android OS
  - Maybe slightly different mix of L2 correctable bit errors (statistics)
- Up to an LET of about 2 MeV-cm^2/mg the cross section for L2 bit errors was on the order of 1×10^{-11} cm^2/bit.

Snapdragon 845 - TBD

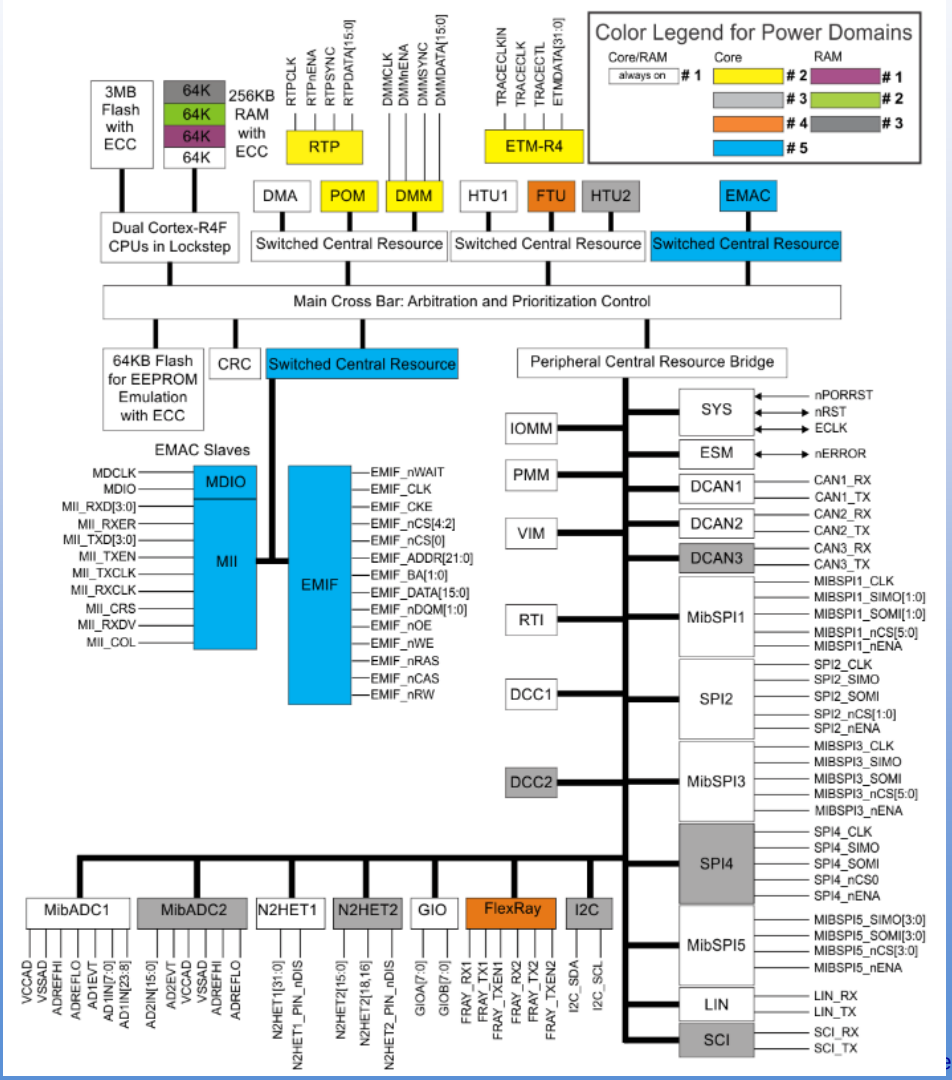
- Samsung 10 nm
  - 8 Kyro 385 CPUs
  - Adreno 630 GPU
  - Hexagon DSP

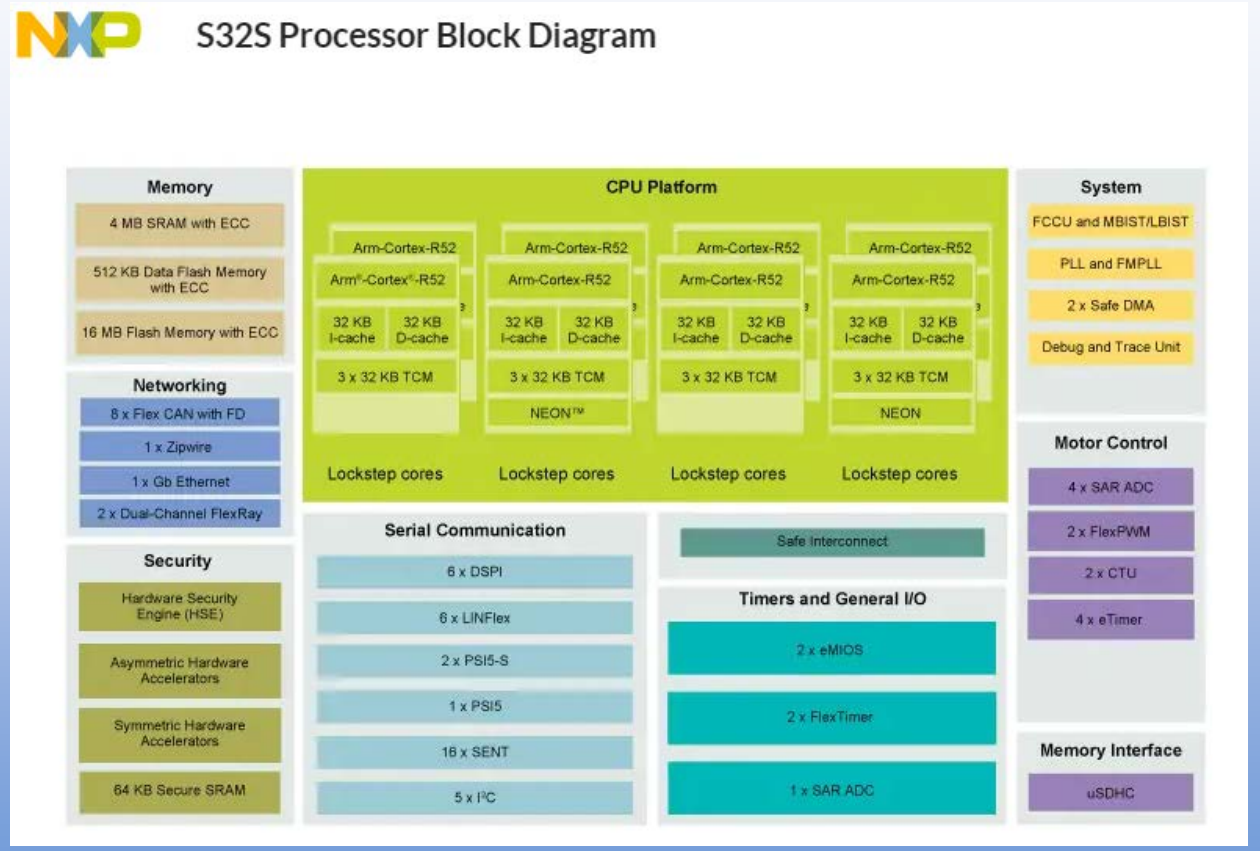


- Using Intrinsyc's 845 Mobile Hardware Development Kit – Android only, which is not desired…
- No Linux.

National Aeronautics
and Space Administration

Microcontroller Block Diagrams



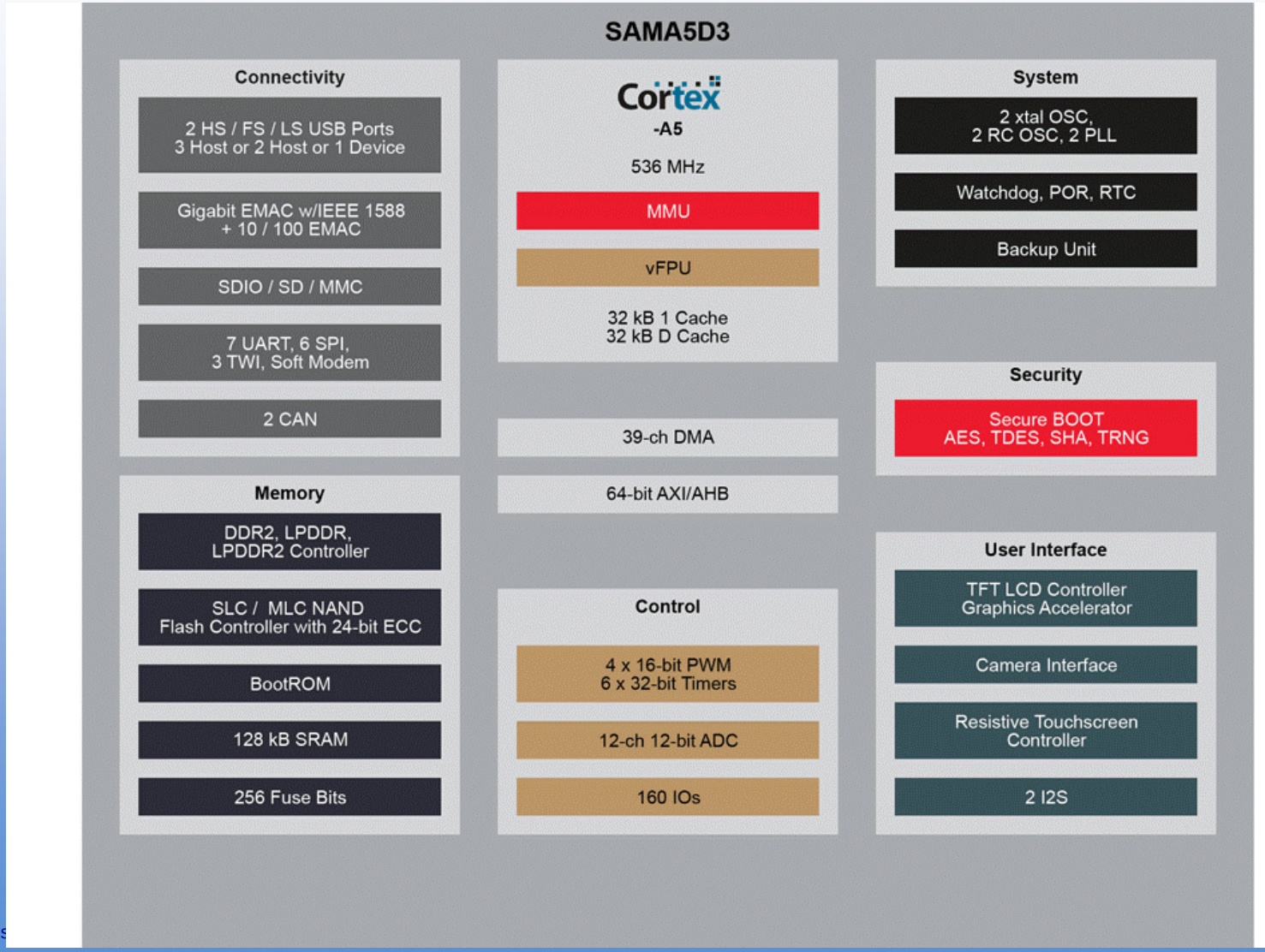


Proposed Testing – FT/Mitigation

- Crash cross section single core mode vs lockstep mode
- Bus fault protection during external memory SEFI
- Internal flash memory error modes
- Error / fault checking peripheral crash coverage
- Cortex R4 vs R5 vs R52 crash rate
- Internal Volatile memory error handling (SBU vs MBU)
- Parity protected peripheral memory improvement
- (Actual test method will depend on development issues and observed in-beam behavior…)

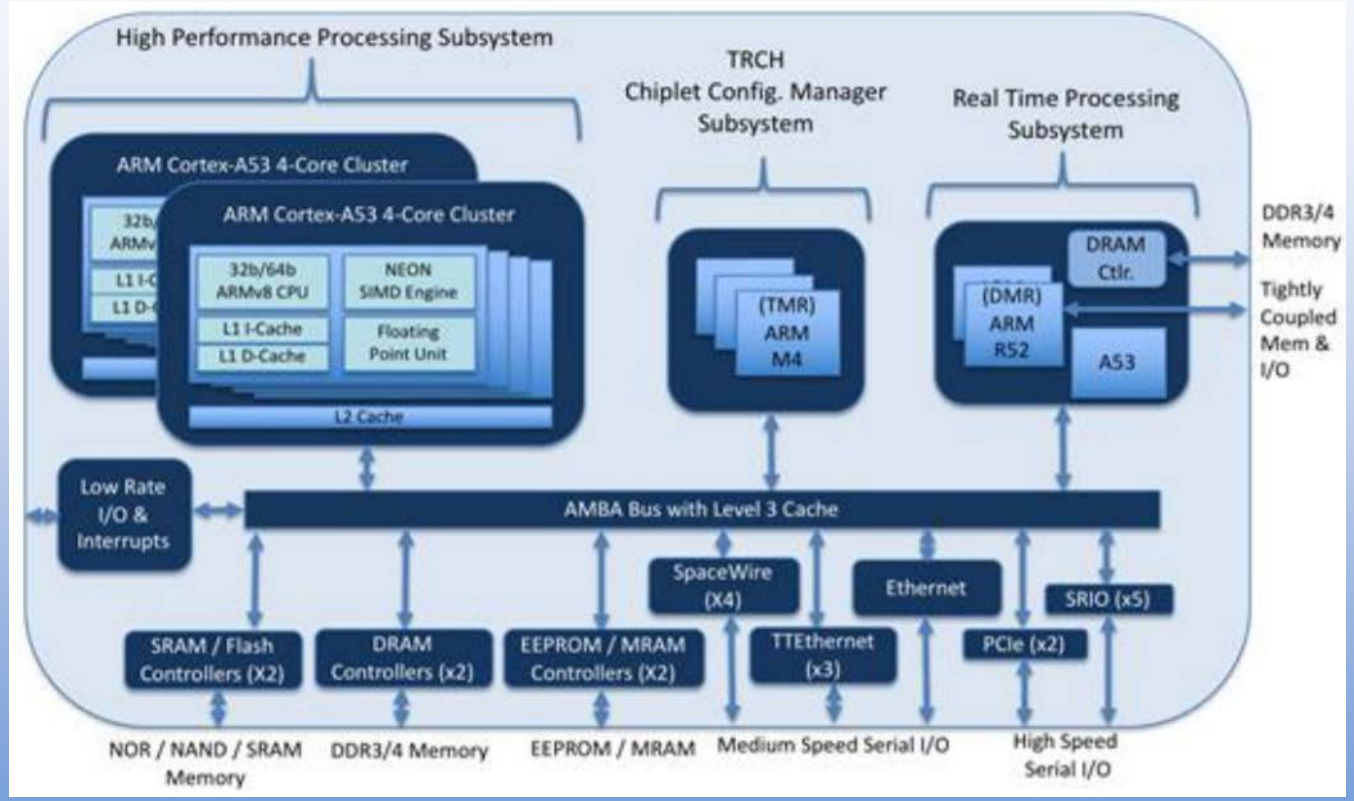
SAMA5D3

- A5-Based Microcontroller
- Xilinx UltraScale+ MPSoC has dual-core A5s for comparison
  - (And comparison to on-chip quad-core A53)
- Also working on getting A5-IP via collaboration



Heterogenous Error Handling

- SEE performance of different subsystems does not need to be the same.
- SOCs already utilize this – think L2 cache error correction vs. L1 cache error trapping.
- But this is a hardware capability vs. a software implementation problem. The hope is that as the industry develops, it will become more common to have systems and software that can support subsystem isolation in the event of an error.
- This type of approach is taken in the HPSC program https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180007636.pdf

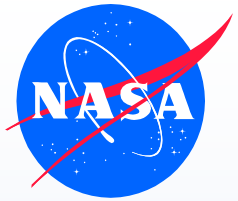


Future

- Tests planned:
  - Testing on Snapdragon 835, 845
  - TI TMS570 – specifically to evaluate with and without fault mitigation enabled
  - ARM Cortex A5 device, for comparison across multiple fabrication situations and soft/hard core IP
- Extract critical info on what fault tolerance works and what doesn't, in order to help specify fault tolerance settings in ARM devices.

Conclusions

- SOC/Complex Processor Efforts focusing on ARM architecture
- Using multiple hard- and soft-IP approaches
- Testing on Snapdragon 835 and 845, recently, has indicated expected, but high, error rates. But ability to investigate is limited.
- Multi-angle approach allows isolation of architecture-specific and fabrication-specific behavior.
  - Additional manufacturing platforms allow us to better handle some limited documentation issues.
- Fault-Tolerant, Mitigated ARM device data gives good feedback on effectiveness of mitigation options offered.



END

Intel CE3100

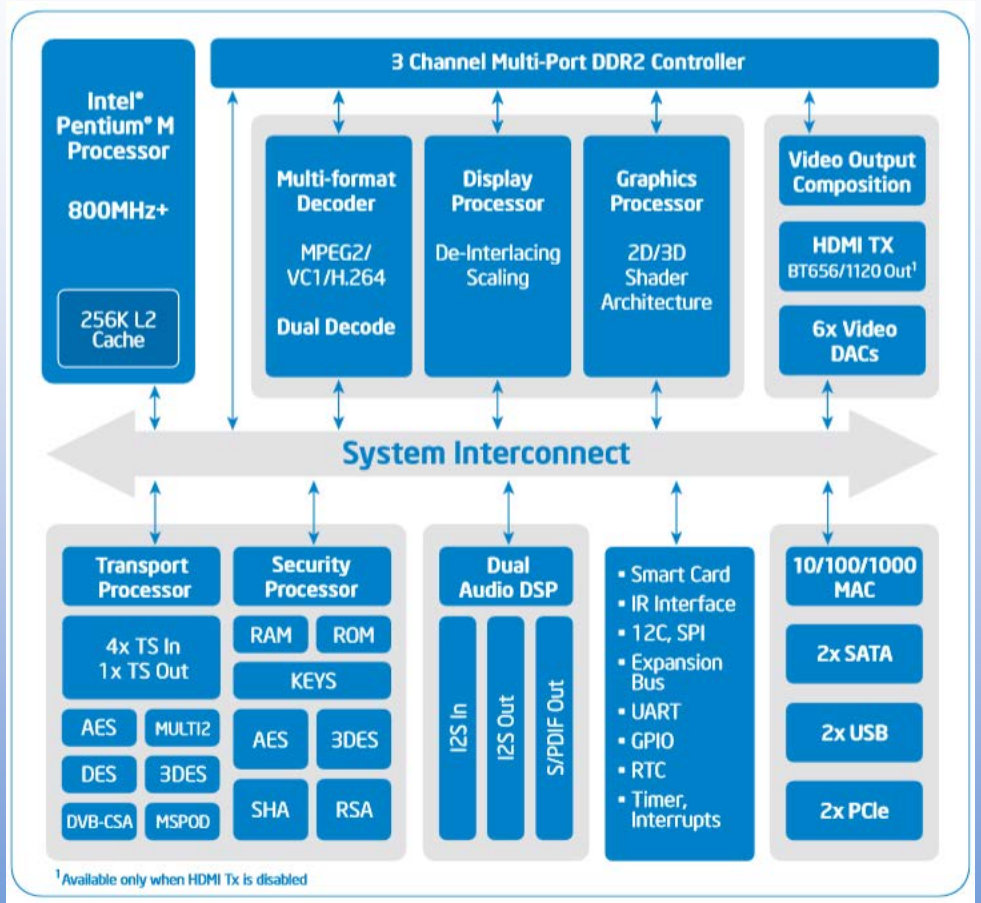


Figure 1 - Intel® Media Processor CE 3100 - Block Diagram

The Intel® Media Processor CE 3100 is a highly integrated system-on-a-chip that combines a high-performance Intel® architecture processor core with leading-edge video decoding and

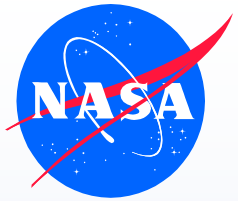
New Processors are Here - SOCs!

- Lots of new and upcoming missions are using new devices (SOC architectures)
  - Cell phone processors/drone processors on Cubesats and Mars Helicopter
  - (see https://rotorcraft.arc.nasa.gov/Publications/files/Balaram_AIAA2018_0023.pdf)
  - RAD5545 – quad core 64-bit PowerPC being considered for several programs
  - Interest in GPUs (nVidia & AMD) for machine learning, Intel devices, Qualcomm cellphone processors, etc.
- SEE testing of these devices, especially commercial, is running into problems

Anecdotes – Conspiracies?

- Apple A12 “bionic” processor has
  - 6 CPU cores
  - 4 GPU cores
  - 8 Neural Processor Cores
  - This is just what we’re told about
- Typical older cell phone processor with 4 cores actually had 3 other processor cores running it… (boot, housekeeping, etc).

- **SOCs are actually multi-processor computers**
  - Without NDA, a lot of functionality/capability is unknown
  - With NDA, you aren’t likely to get direct access to the 100s+ of designers involved.
- Significant risk that seeing an SEE gives a very limited signature
- SOC events are like tracking an error in a full computer, with limited documentation



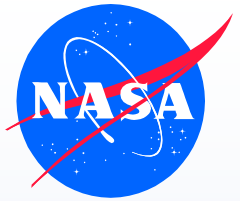
SOME INFORMATION

Config, Config, Config

- A Hardware Designer, a Software Engineer, and a Program Manager walk into a review…
  - Program Manager: Our system utilizes equipment with a low rate of errors in our flight environment
  - Hardware Designer: Our system utilizes equipment that can be configured to have a low rate of errors
  - Software Engineer: We disable all error correction to meet our data processing requirements
- SOCs have a huge number of configuration settings – and active recovery methods…

Parity, EDAC, and Cache Mode

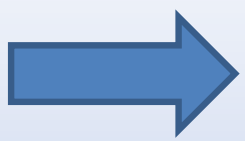
- Processors have included error detection and correction on L2 caches for over 20 years.
- L1 caches have principally had only parity protection.
  - EDAC on L1 has become available in recent years
- Both L1 and L2 have “tag” information that is usually only parity protected.
- Tag information can also be EDAC protected – but this has a risk
  - The tag information may be corrected when read, but not updated
  - Some tags are kept in the cache indefinitely…
- If an EDAC-protected tag gets two bit errors it cannot be corrected and the system crashes
  - Likely with locked cache lines

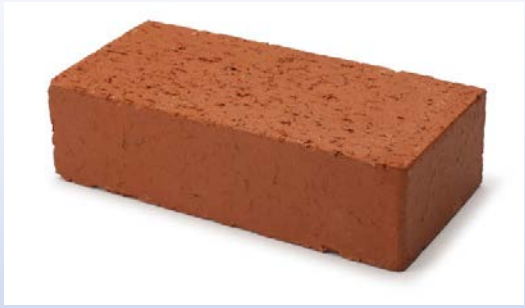


TYPES OF RADIATION FAILURES

Global Categories

- Permanent Failures
  - Device is out of specification
  - System cannot reliably boot
- Temporary Failures
  - Crash – more later
  - Bit or calculation errors
    - Bad data may get into the system somewhere
  - Exceptions
    - Usually cause thread termination
    - Or, if they occur while the operating system is doing something critical, can result in a crash





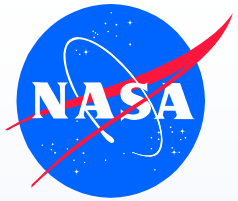
Crashes

- Crash, Hang, Unhandled Exception, Error Mode, Trap in Trap, Inverted Beer Mug, Runaway Loop, Mouse in Computer (observed once on a test)
  - Often not possible to identify what actually causes the event
  - Always recovered by full power-cycle (turn the mug back over, remove the mouse), and sometimes recovered by just hitting reset

- Unless clearly defined in a test approach/report, you should assume the full set of synonyms (up to beverages and rodents).

Radiation Failures for Test

- In a perfect world, you test everything. In reality you have to pick things and build detection.
  - With limited info, you have to infer what you don't measure directly…
  - Try to understand sensitivity of enough basic structures and subsystems
- Testing basic structures and subsystems
  - SRAM-based targets – such as caches and some types of registers
  - Flip-Flop-based targets – generally the other registers and execution unit pipelines (if possible)
  - Exception/Trap/Interrupt systems (especially catching unexpected events)
  - Data flow through memory controllers, communications systems, etc.
- Testing complex devices with multiple subsystems and fault handling is not deterministic.
  - Cycle-by-Cycle comparison to expectation is problematic
  - Simple targets can be easily tested with simple hardware
  - Most everything else seems to require in-situ detection or using debugging hardware to repeat tests during exposure
- Performance benchmarks and manufacturer's hardware test codes must be modified for SEE testing (poor target control and duty cycle)



RADIATION EFFECTS EVALUATION METHODS

Testing Ideas Seen & Issues

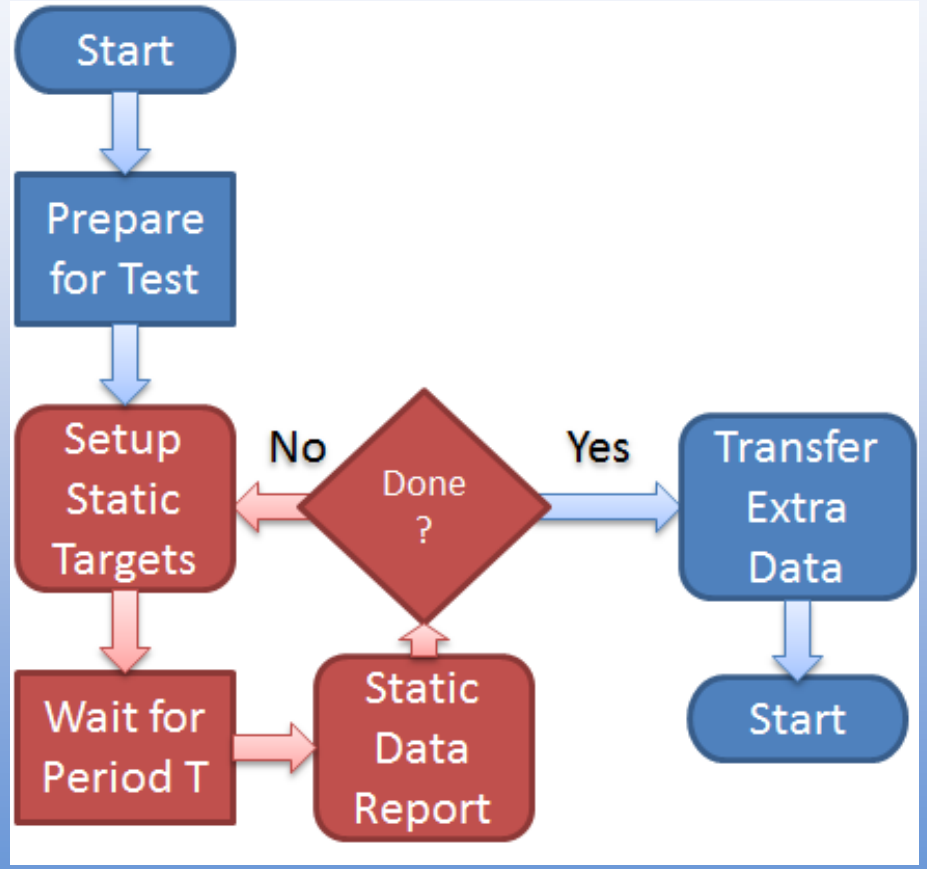
- Test using code running on the DUT
  - E.g. Benchmarks, FFTs, etc…
- Use hardware verification tools
- Use a debugger to access registers and cache
- Use a debugger to run custom code
- Test registers – load values, check
- L1/L2 cache bits – load, check
- Test with operating system
  - Application/flight code?

- Results depend on code robustness
- Poor time-structure
- Quasi-static
- Very few bits…
- Error correction may obscure, code execution may depend on caches

National Aeronautics
and Space Administration

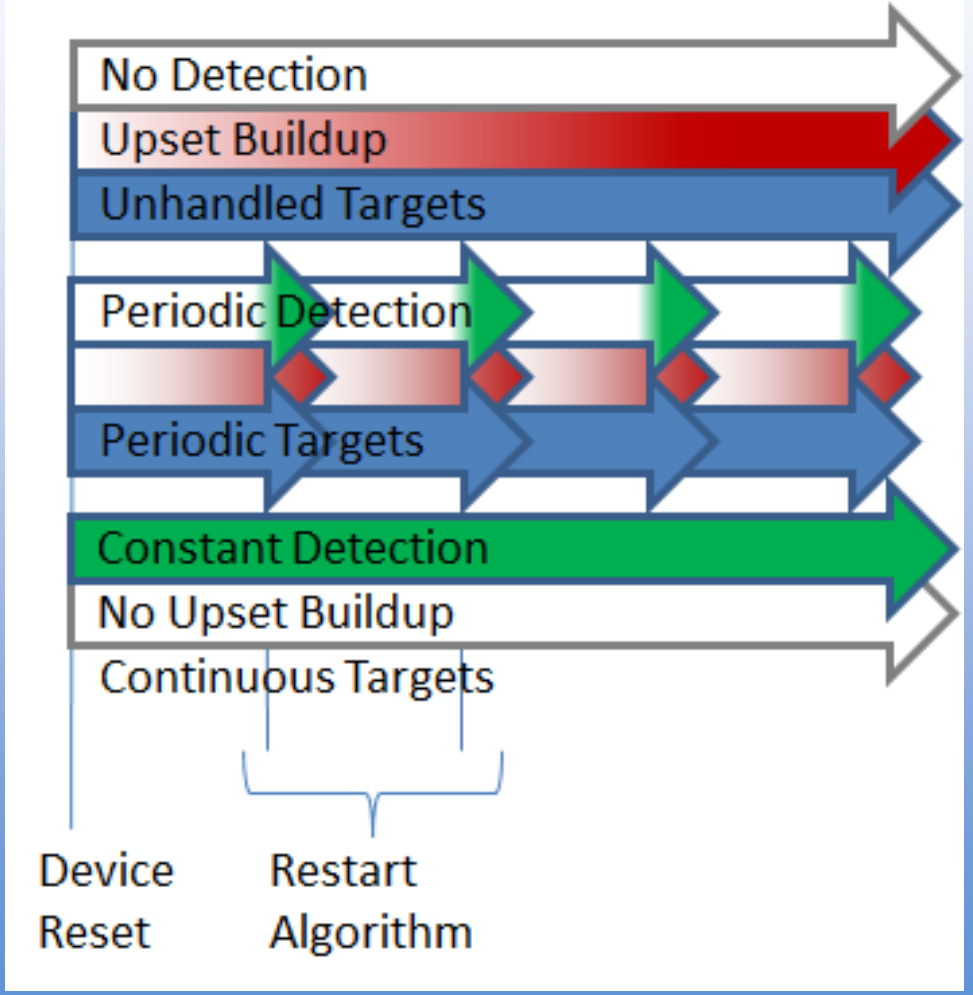
Standard Static Soak Test

- Test algorithms are generally based on repeated sub-tests that may report on each loop iteration…
- Test algorithms should periodically report results to enable immediate detection of loss of operation.
- Periodic reporting enables use of partial runs.
  - This is a periodic test
- If only one readout and/or targets are only initialized once, it is a “no detection” test
  - Alternately, if targets are continually monitored, it is a “constant detection” test



Test Algorithm Time Models

- Unhandled targets build upsets during exposure.
- Periodic targets build upsets during an integration phase
- Constant detection



Keys to Test Algorithms - Anomalies

- It is common to want to explore test anomalies because they may be “rare SEEs”. Avoid this want as it leads to much lost time.
- Rely on beam and event statistics. If it doesn’t repeat, put a limit on it and move on. If it does repeat, figure out what it is if possible.
  - If you can’t figure out what it is, that’s ok. Report the rate, but don’t claim a mechanism.
  - Be aware you may be exploring a bug in your code, or even upsets in support or test equipment.
- When possible, use debugging tools to examine these as they enable fast and detailed examination

National Aeronautics and Space Administration

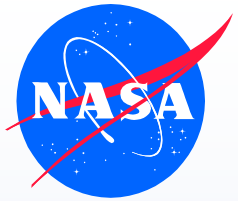
- Capture of errors during boot
- LET=7, flux ~5e3 #/cm^2
- Using Android boot

L2 Error

Unspecified Exception

Unspecified Exception

```
[    42.074530] Bluetooth: Tx timer expired
[    42.074530]
[    43.814375] L2 Error detected!
[    43.816419]  L2ESR    = 0x00010010
[    43.819807]  L2ESYNR0 = 0x6200080a
[    43.823195]  L2ESYNR1 = 0x04a4977a
[    43.826583]  L2EAR0   = 0x0002c160
[    43.829971]  L2EAR1   = 0x00000000
[    43.833358]  CPU bitmap = 0x1
[    43.836319] L2 data soft error, single-bit
[    43.840378] Kernel panic - not syncing: L2 single-bit error detected
[    43.846787] [<c0014918>] (unwind_backtrace+0x0/0x11c) from [<c07ba978>] (pani
c+0x84/0x1d4)
[    43.855028] [<c07ba978>] (panic+0x84/0x1d4) from [<c007d7f4>] (msm_l2_erp_irq
+0x1fc/0x260)
[    43.863268] [<c007d7f4>] (msm_l2_erp_irq+0x1fc/0x260) from [<c00d2844>] (hand
le_irq_event_percpu+0xb0/0x290)
[    43.873065] [<c00d2844>] (handle_irq_event_percpu+0xb0/0x290) from [<c00d2a60
>] (handle_irq_event+0x3c/0x5c)
[    43.882862] [<c00d2a60>] (handle_irq_event+0x3c/0x5c) from [<c00d554c>] (hand
le_fasteoi_irq+0xdc/0x148)
[    43.892232] [<c00d554c>] (handle_fasteoi_irq+0xdc/0x148) from [<c00d20a8>] (g
eneric_handle_irq+0x30/0x44)
[    43.901785] [<c00d20a8>] (generic_handle_irq+0x30/0x44) from [<c000ee40>] (ha
ndle_IRQ+0x7c/0xc0)
[    43.910544] [<c000ee40>] (handle_IRQ+0x7c/0xc0) from [<c0008620>] (gic_handle
_irq+0x94/0x110)
[    43.919059] [<c0008620>] (gic_handle_irq+0x94/0x110) from [<c07c8cc0>] (__irq
_svc+0x40/0x70)
[    43.927453] Exception stack(0xc0d05e70 to 0xc0d05eb8)
[    43.932488] 5e60:                                     00000000 c0f3cd00 00000
000 00000000
[    43.940668] 5e80: c0d04000 00000000 00000202 c0d05f28 00000012 fa00300c c0d43
75c 0000000a
[    43.948817] 5ea0: c0d4f080 c0d05eb8 c0087d3c c0087710 20000113 ffffffff
[    43.955440] [<c07c8cc0>] (__irq_svc+0x40/0x70) from [<c0087710>] (__do_softir
q+0x4c/0x248)
[    43.963680] [<c0087710>] (__do_softirq+0x4c/0x248) from [<c0087d3c>] (irq_exi
t+0x48/0xa0)
[    43.971860] [<c0087d3c>] (irq_exit+0x48/0xa0) from [<c000ee44>] (handle_IRQ+0
x80/0xc0)
[    43.979734] [<c000ee44>] (handle_IRQ+0x80/0xc0) from [<c0008620>] (gic_handle
_irq+0x94/0x110)
[    43.988249] [<c0008620>] (gic_handle_irq+0x94/0x110) from [<c07c8cc0>] (__irq
_svc+0x40/0x70)
[    43.996642] Exception stack(0xc0d05f28 to 0xc0d05f70)
[    44.001678] 5f20:                   00000000 00000000 00000001 00000003 00000
003 00000003
[    44.009858] 5f40: c0d65924 00000003 00000000 c0d65924 00000000 00000000 00000
002 c0d05f70
[    44.018007] 5f60: c07c873c c005bf7c 60000013 ffffffff
[    44.023073] [<c07c8cc0>] (__irq_svc+0x40/0x70) from [<c005bf7c>] (msm_cpuidle
_enter+0x70/0x78)
[    44.031680] [<c005bf7c>] (msm_cpuidle_enter+0x70/0x78) from [<c056911c>] (cpu
idle_enter+0x14/0x18)
[    44.040592] [<c056911c>] (cpuidle_enter+0x14/0x18) from [<c0569690>] (cpuidle
_idle_call+0x1e0/0x3c0)
```



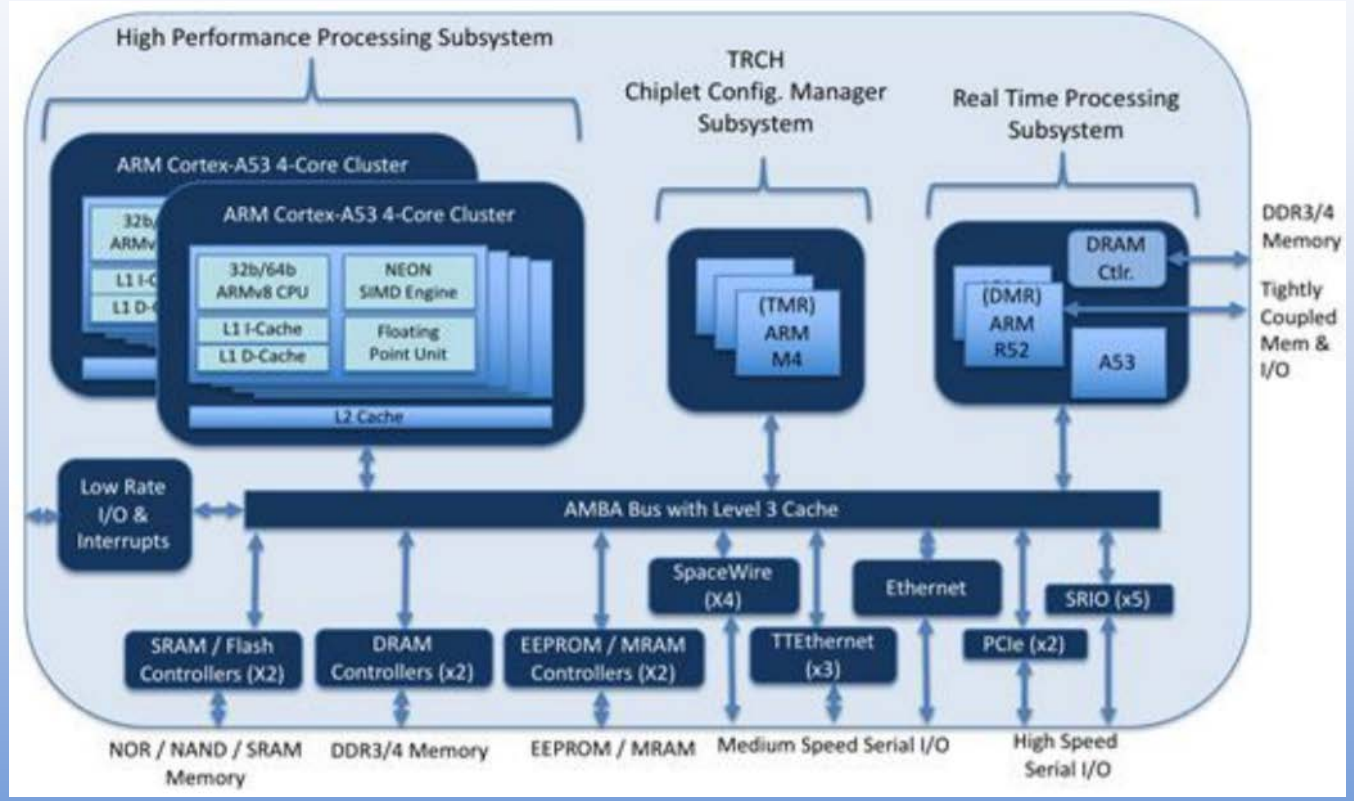
LOOKING TO THE FUTURE

General Trends

- Availability of documentation will get worse
- We are looking to engage commercial manufacturers to see if it is possible to get enough info to assess radiation sensitivity without compromising internal IP
- Multi-chip devices are already a problem, this will get worse. A lot of today's SOCs almost meet the definition of a hybrid or multi-chip module
- Test results will likely become more "high-level" requiring significant conservatism in how test results are applied to flight systems.

Heterogenous Error Handling

- SEE performance of different subsystems does not need to be the same.
- SOCs already utilize this – think L2 cache error protection vs. L1 cache error trapping.
- But this is a hardware capability vs. a software implementation problem. The hope is that as the industry develops, it will become more common to have systems and software that can support subsystem isolation in the event of an error.
- This type of approach is taken in the HPSC program https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180007636.pdf



More Devices/Uses

- Efforts are underway to explore performance of SOCs with GPUs - cell phone processors, nVidia TX1
- As well as with dedicated GPU chips, can errors in the GPUs be contained? Can software& hardware handle the errors and continue running?
- Examples of GPU testing:
  - Memory transfer
  - Typical image generation/manipulation
  - AI/Machine learning algorithms