WINTECHCON

Validation methodology for Nest Memory Management Unit (NMMU)

September 27, 2019

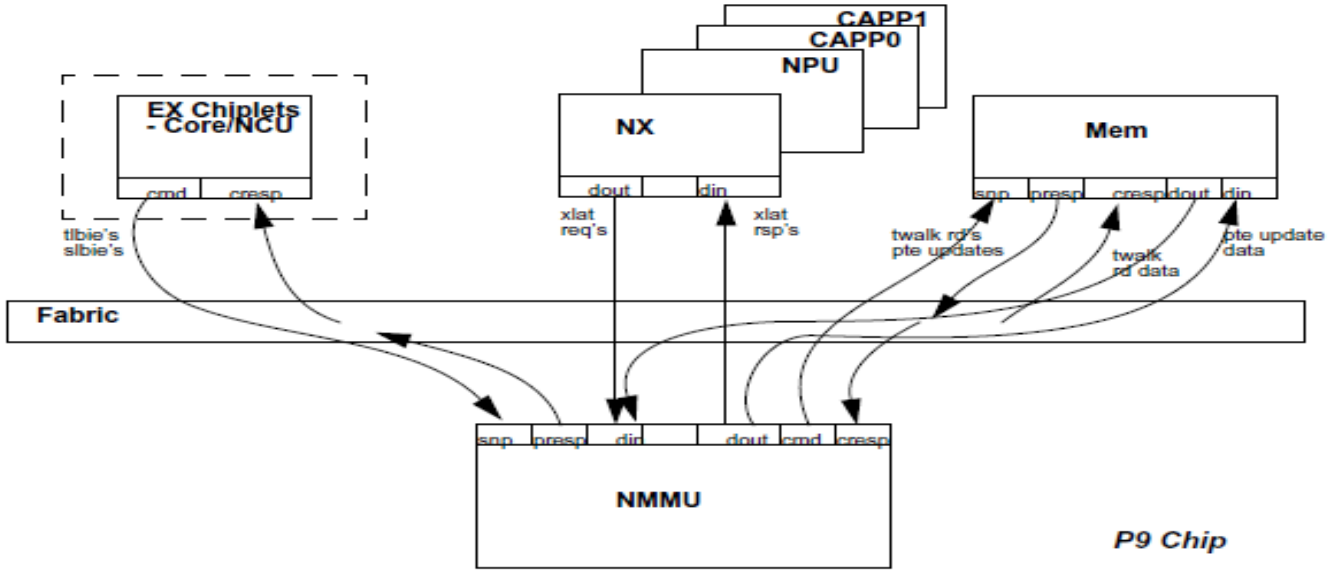
Nandhini R/Jayakumar NS/Larry SL
IBM

Background

- Nest Memory Management Unit (NMMU)
  - Memory management unit for I/O devices
- Complex integrated circuitry in each processor chip
  - provides address translation support for
    - on-chip nest accelerator (NX)
    - off-chip Nvidia Processing Unit (NPU) and
    - Coherent Accelerator Processor Proxy (CAPP0/1) units.
- Primary goal of NMMU –
  - provide effective address (EA) to physical address (PA) translation
  - Improves the response time of accelerator agents
  - Access protection for memory regions

NMMU - Interfaces

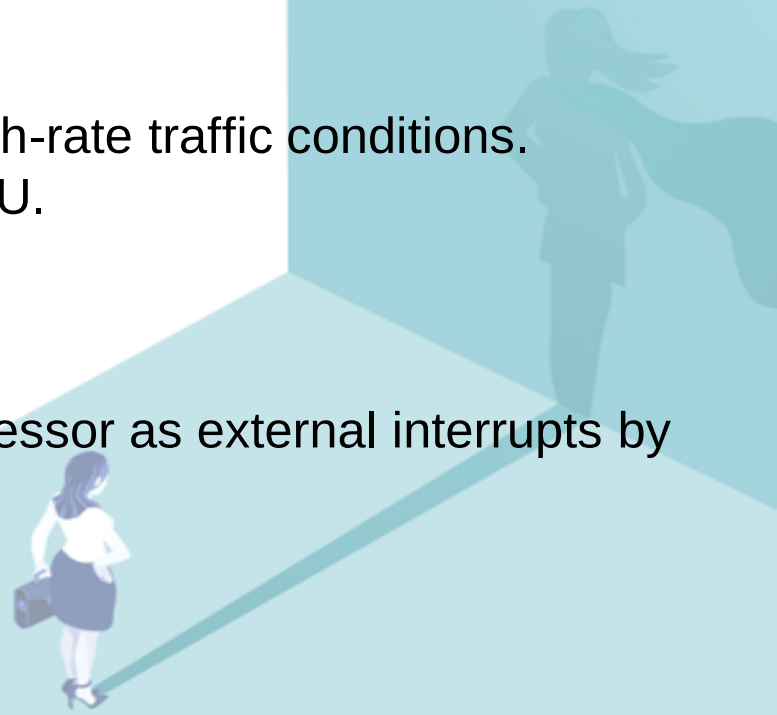
- Power 9 Chip



Problem Statement

- Validation of NMMU
  - design complexity
  - validation schedule

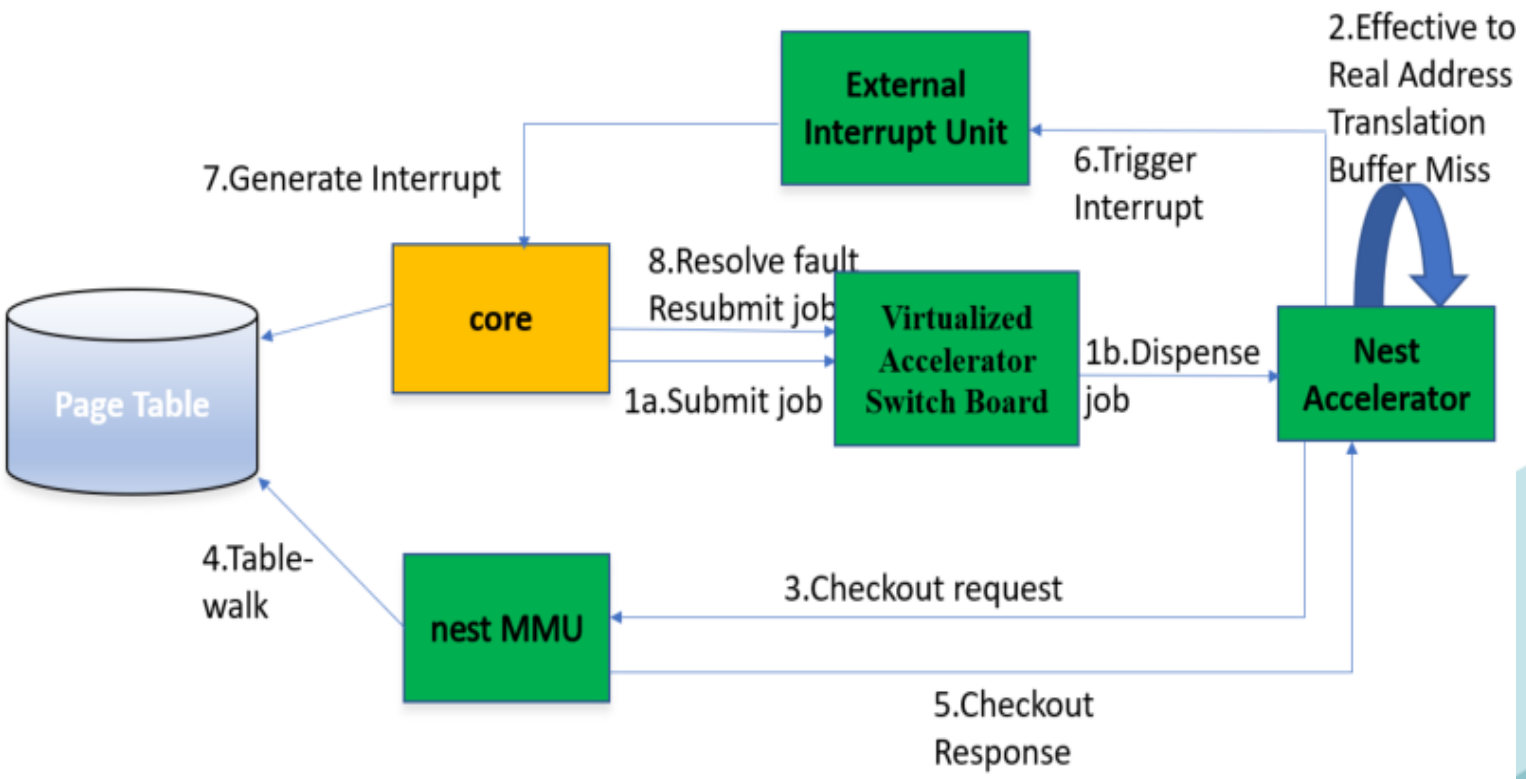
- Challenges:-
  - Need to have special accelerator agents to generate high-rate traffic conditions.
  - Fewer agents to trigger translation requests to the NMMU.
  - Accelerator operates on blocks of data
  - Accelerator job turn around time - significant
    - Need an asynchronous job submission mechanism
  - Translation faults from agents are presented to the processor as external interrupts by a virtualized external interrupt hardware unit.
    - Should not swarm a specific processor with interrupts



Solution approach

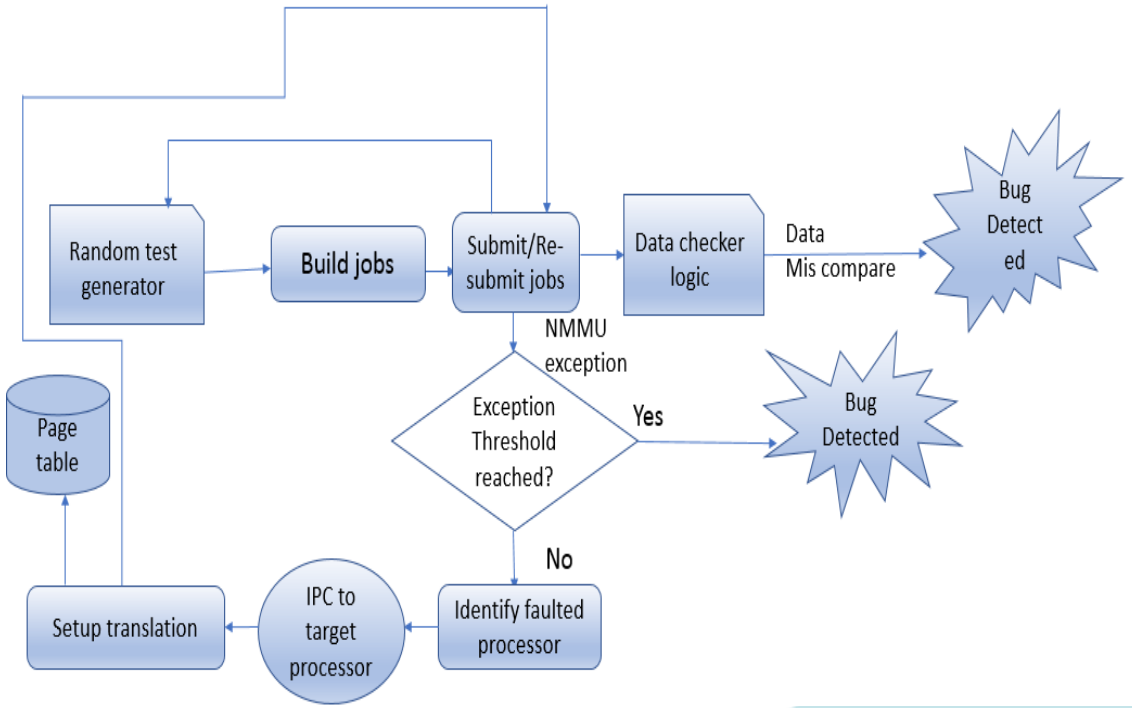
- Core MMU translation used as reference model to validate NMMU
  - Translation table shared between Nest MMU and core MMU
  - Existing processor core storage exception handlers are leveraged
    - minimize the validation tool software development effort
- Asynchronous Job submission model to drive translation traffic
  - create a swarm of translations requests without waiting for each request to complete.
- Optimized threshold based checker to detect hardware issues
- Interrupt hardware unit configured in such a way that the interrupts were fairly distributed among different processors running on the system.
- Nest Accelerator Unit (NX) agent to induce translation traffic to NMMU.
  - cryptographic engine
  - memory compression/decompression engines (coprocessors).

Solution approach



Validation methodology - phases

- Testcase generation
- Translation Generation
- Checker logic



Testcase Generation

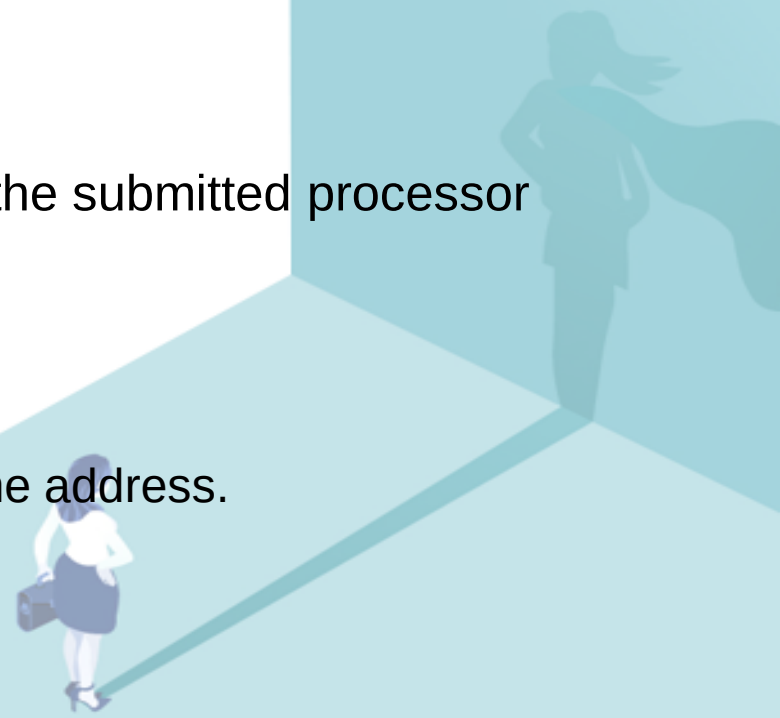
- Build job
  - job defined by an agent-specific control block in memory
    - NX control block - Coprocessor Request Block (CRB).
    - pointers to input parameters, input data
    - pointers to store the output data
    - Status block.
    - pointer to address translation context.
    - control information
  - Translation context
    - control fields including privilege levels, translation mode, partition-id and process-id
- Dispatch job - asynchronous
  - Status of job
    - Accelerator status block – upon completion
    - Per processor queue – fault information

Translation Generation

- Processor submits a job to Accelerator
- Accelerator interrupts any processor on Translation miss

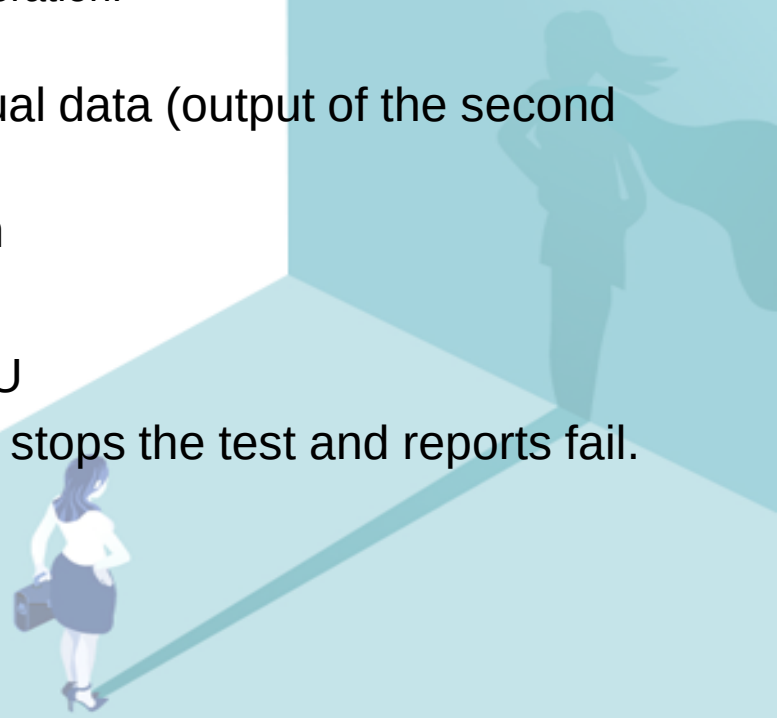
- The Interrupted processor
  - determines the submitted processor number
  - sends the faulted address and other fault information to the submitted processor

- The submitted processor
  - access the data from the faulted address.
  - generates a storage interrupt to the core
  - core storage interrupt handler sets up translation entry for the address.
  - Resubmit the job to Accelerator



Checker logic

- EOT Mismatch checker is used to check the correctness of target data.
  - Each job has two operations
    - output of the first operation is used as an input to the second operation.
    - Second operation is the reverse of the first one.
  - compares the expected data (original source) with the actual data (output of the second operation).
  - Any incorrect translation is manifested as a data mismatch
- Threshold based checker code
  - increments a per-fault counter value upon fault from NMMU
  - When checker has reached a maximum threshold value, it stops the test and reports fail.



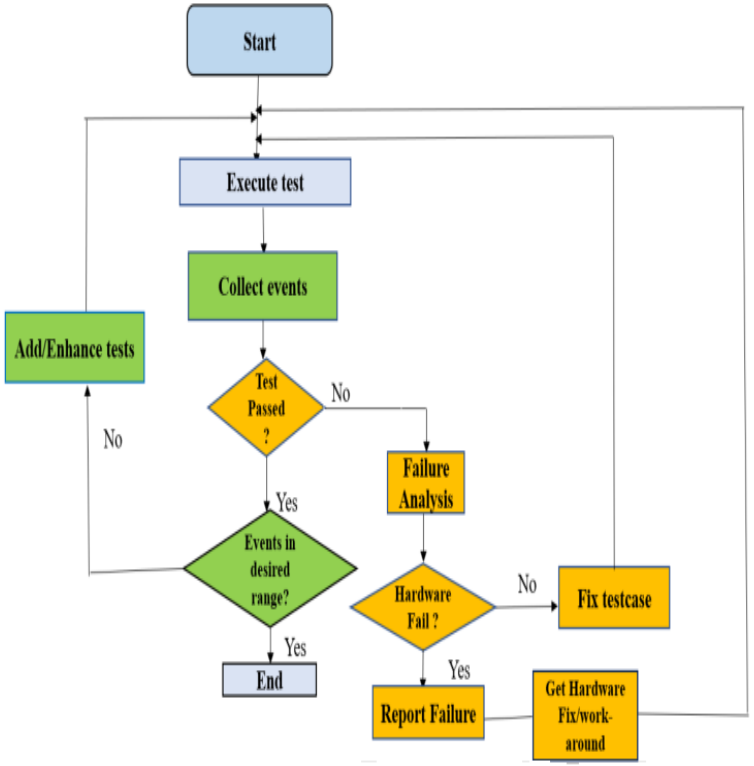
Pre-silicon verification test bench

- Internal ASIC-based simulation acceleration platform - AWAN
  - Exercisers on Accelerators (EoA)
  - to provide additional functional coverage to pre-silicon testing.
  - to use the pre-silicon coverage data to further enhance the test-cases.
  - tool development and testing are done in the simulation environment well in advance
  - Pre and post silicon - same test without any modification.

- BugSpray - an extension of VHDL used for functional coverage instrumentation.
  - efficiently annotate the RTL with assertion and coverage events.
  - to detect low coverage areas.

Post silicon validation procedure

- System reset and initialization
- Loading Kernel and testcase generator into system memory.
- Cores triggered.
- Scheduling of test-case on different cores by the kernel.
- Special (Scratch) register polled for pass or fail criteria.



Experimental Results

Table I
Statistics collected by coverage tool with VTS1

Coverage Parameters	Values
Total translations	5440
Clean checkouts	3452
Total number of cycles	469341426
Cycles per translation	86275.99743
Translations per second	23181.418

Table II
Statistics collected by coverage tool with VTS2

Coverage Parameters	Values
Total translations	4481
Clean checkouts	3812
Total number of cycles	237279016
Cycles per translation	52952.24
Translations per second	37769.88

VTS1 – Validation Test Suite 1
VTS2 – Validation Test Suite 2

Conclusions

- Logic verification and validation:
  - 80-90% of the bugs (performance, functional ,design document specification) were uncovered
  - fixes re-checked at the early verification stage
- Coverage and throughput enhancement:
  - Pre and post silicon level
    - coverage metrics were used to quantify which design functions have been reached.
  - Tests were tuned to improve the coverage.
  - Achieved 80% of the throughput supported by the unit
- Foster rapid software development:
  - Kernel software for core interrupt handlers were reused in this approach.
  - Avoided duplicate development work for interrupt handlers.
  - Achieved the goal of reducing the overall validation tool development time.

References

- Accelerate the Future of Computing with Power Acceleration,2018,https://www.linkedin.com/pulse/accelerate-future-computing-power-acceleration-manoj-dusanapudi/?published=t
- George Papadimitriou; Athanasios Chatzidimitriou; Dimitris Gizopoulos; Ronny Morad,“An Agile Post-Silicon Validation Methodology for the Address Translation Mechanisms of Modern Microprocessors”, *IEEE Transactions on Device and Materials Reliability*,2016.
- Power9 Processor’s User Manual, 2018, https://openpowerfoundation.org/?resource_lib=power9-processor-users-manual
- K.-D. Schubert ; S. S. Abrar ; D. Averill ; E. Bauman ; A. C. Brown ; R. Cash ; D. Chatterjee ; J. Gullickson ; M. Nelson ; K. A. Pasnik ; K. Sugavanam ,”Addressing Verification challenges of heterogeneous systems based on IBM POWER9”,*IBM Journal of Research and Development,*2018
- Viresh Paruthi, *“Large-scale application of formal verification:from fiction to fact”,* Formal Methods in Computer Aided Design,2010.