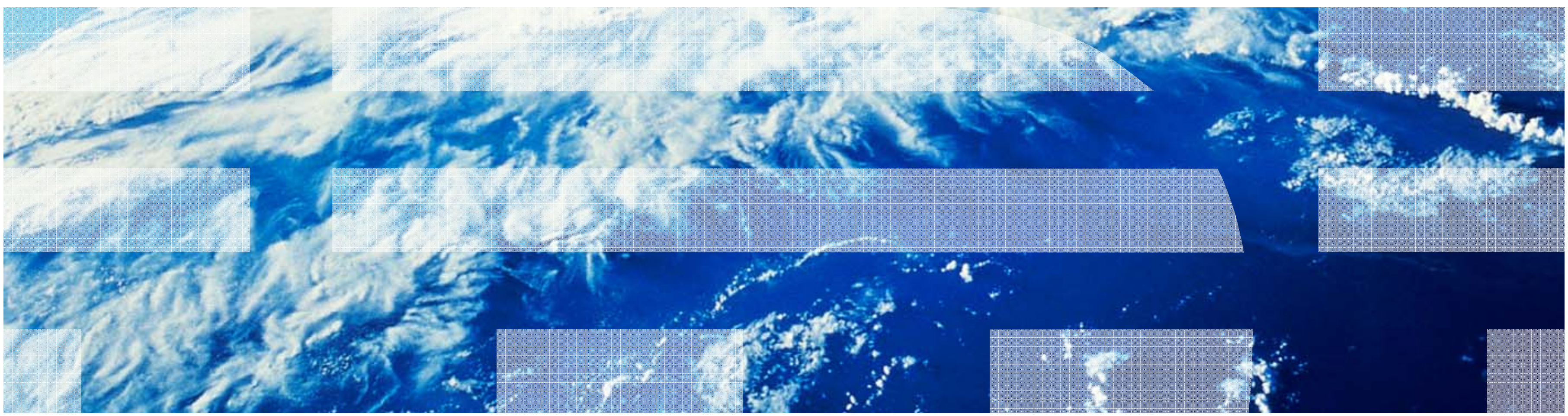
Experiences in Simulation at IBM

Mike Kistler
IBM Research, Austin TX



Overview

- Simulation/modeling is used across a broad range of activities in the system development life-cycle
  - Early design studies to evaluate architectural alternatives
  - Processor and system performance analysis
  - Development of processor verification suites
  - Early firmware/hypervisor/OS development
  - Post-silicon verification test development
  - Cluster / Network simulation

Early Design Studies

- Primary tools: queuing models, high-level perf models
  - In IBM: various tools, Mambo (cycle-mode)
- Key characteristics
  - Models only key structures / features of system
  - Highly parameterized
  - Primarily used for comparative studies
  - Typically trace-driven
  - Typically application-level studies
- Supporting Tools
  - Trace generation/collection tools

Processor performance modeling

- Primary Tool: Processor Performance Model
  - In IBM (Power): M1 -written in special in-house “T” language
  - Developed by a small team of performance experts
- Key simulator characteristics
  - Cycle-accurate
  - Trace-driven (Qtrace – detailed instruction traces)
  - Models all key microarchitectural resources/facilities
  - Collects and reports detailed statistics on performance
- Supporting tools
  - Trace generation tools (e.g. Mambo, hardware instrumentation)
  - Trace sampling tools (similar to Simpoints)
  - Performance visualization tools (ScrollPipe)

Processor Verification Reference Model

- Primary simulation tool: Processor simulator
  - In IBM Power: Mambo
- Key simulator characteristics
  - Models all *functional* features of Processor
  - “Performance” features such as cache instructions are no-ops
  - Computational instructions produce bit-accurate results
    - Including results that are “undefined” in the architecture
  - Allows undo of instructions
  - Tracks registers/architected state effected by a sequence of instructions
- Supporting Tools
  - Test Generation Tool (GPro)

Processor Verification

- Primary simulation tool: VHDL simulator
  - In IBM (BG/Q): Twinstar (FGPA-based)
- Key simulator characteristics
  - Models “full” design of processor/chip
  - Execution-driven
  - Cycle-accurate
  - Cycle-reproducible
  - Support for detailed trace / state inspection
- Supporting Tools
  - VHDL to FPGA tool chain
  - Specialized tools for partitioning design across FPGAs

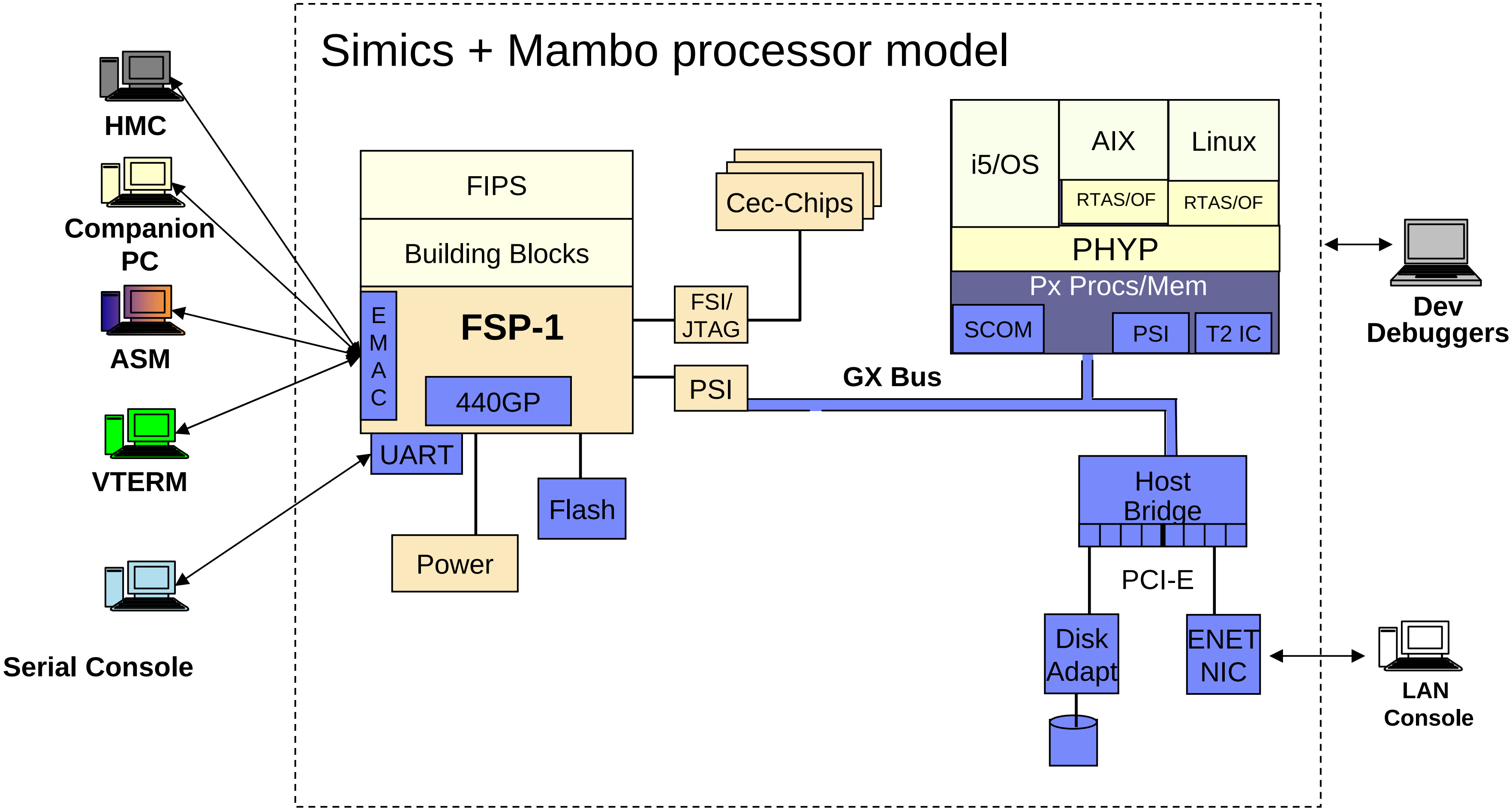
Twinstar BG/Q Node simulator

- Multi-FPGA simulation of one BG/Q node)
  - 16 PowerPC A2 cores, L1 caches, memory prefetch logic, crossbar switch, 32MB L2 cache, 2 memory controllers, and 2GB system memory
- Simulated on 45 Virtex 5 LX330 FPGAs
  - Two systems constructed
- FPGA platform runs at 4 MHz simulated processor clock speed (compared to 10 Hz for S/W RTL simulation)
- Runs complete, unmodified system software

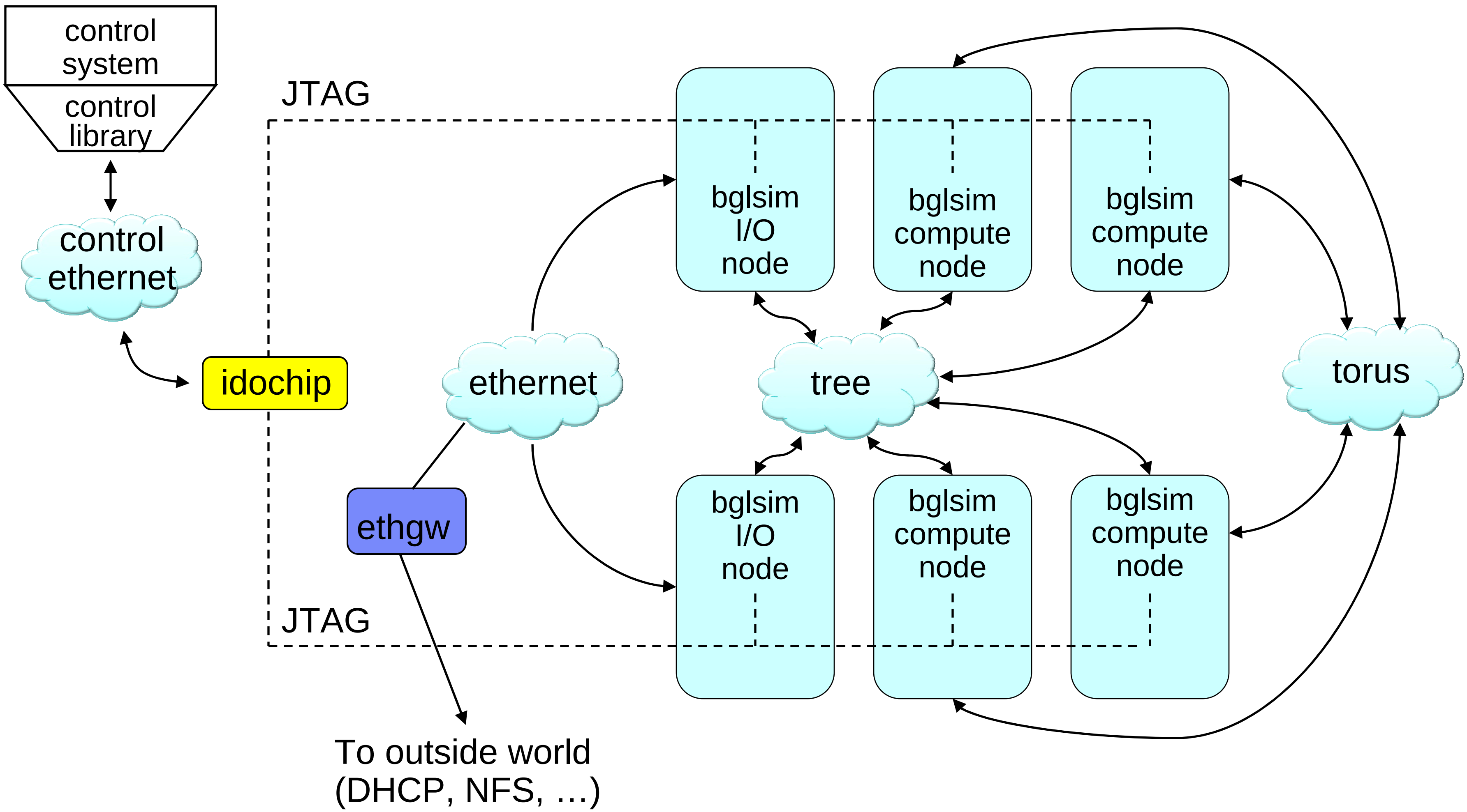
Early firmware/hypervisor/OS development

- Primary simulation tool: Full-system simulator
  - In IBM (Power): Mambo (turbo mode), Simics+Mambo, BGLsim
- Key simulator characteristics
  - Models all *functional* features of the entire system
    - ROMs, memory controllers, I/O buses, devices, bridge chips, clocks, service processors, coprocessors, etc.
  - System configuration defined dynamically (run-time)
  - Execution-driven
  - Fast functional simulation
    - sufficient to boot entire Firmware/OS stack & run applications
- Supporting Tools
  - Complete tool chain for target architecture
    - Compiler, assembler, linker, Source-level debugger
  - Image creation (kernel & filesystem images)
  - Image storage and composition
  - Execution checkpointing

Simics+Mambo Full-System Simulation for Power



BGLsim simulation architecture



Cluster/Network Simulation

- Primary simulation tool: Cluster/Network simulator
  - In IBM: BGLsim-multi (BG), MARS (PERCS)
- Key simulator characteristics
  - Models network features
    - Network interfaces, switches, routing mechanisms
  - Trace-driven or execution-driven
  - Fast-functional simulation
    - sufficient run large scale applications
- Supporting Tools
  - Trace collection (possibly at varying network layers)

MARS – MPI Application Replay Simulation

- Full-system HPCS simulation framework
  - with emphasis on interconnection network,
  - and application modeling via MPI trace replay,
  - based on OMNEST™ (OMNeT++) Simulation Environment
- Supports range of network design and validation activities
  - network topology design
  - network component design and dimensioning (switches, adapters)
  - network functionality design (routing, deadlock prevention)
  - benchmark requirement validation
  - MPI library and application tuning
- Demonstrated scalability
  - 65,536 nodes system running in parallel on 32-way SMP cluster

Taking a Step Back

- Trace-driven or execution-driven simulation ?
  - Yes
- Detailed-functional or Fast-functional or Cycle-accurate ?
  - Yes – with dynamic mode switching
- Software-based or FPGA-based simulation ?
  - Yes – and allowing both within a single simulation
- Dynamically configurable ?
  - Yes
- High Performance ?
  - Of course

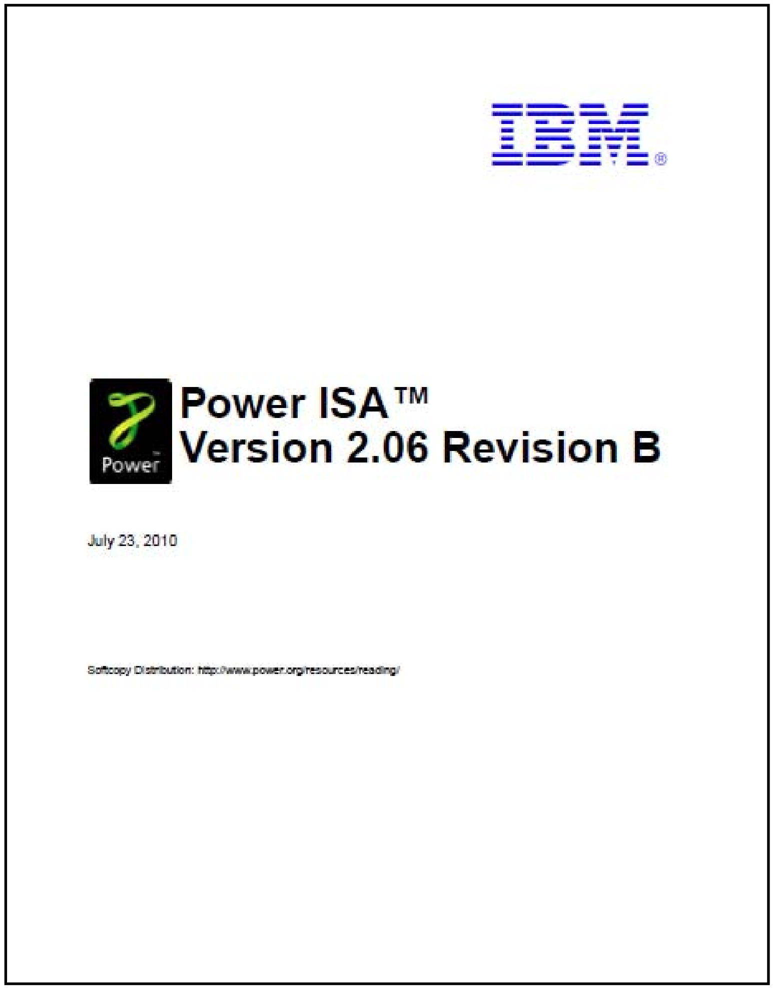
More considerations

- Support parallel execution of simulation components ?
  - Yes
- Ensure deterministic results ?
  - As an option
- Portability ?
  - As much as you can stand
- Formal architecture specifications ?
  - Gosh this would be great

Implications

- Reuse is essential
- The simulation infrastructure must be **modular**, with well designed and carefully constructed interfaces and core services
  - Allow alternative processor / network / component modules to be plugged in
  - Allow caching / reuse of data / prior computations
- The simulation infrastructure must **co-exist and interoperate** with a wide variety of supporting tools
- Simulator construction is primarily a **software engineering** activity
  - Working knowledge of hardware architecture is also valuable but secondary

Reuse is Essential


IBM®

Power ISA™
Version 2.06 Revision B

July 23, 2010

Softcopy Distribution: http://www.power.org/resources/reading/

- 1341 pages
- ~1200 instructions
  - sync, isync, lwsync, ptesync, eieio
  - lwarx, stcwx
- ~150 special purpose registers
- This is the **public** version of the architecture

Reuse is Essential …

VSX Vector Multiply-Add Double-Precision XX3-form

xvmaddadp XT,XA,XB (0xF000_0308)

60	T	A	B	97	AX	BX	TX
0	6	11	16	21	29	30	31

xvmaddmdp XT,XA,XB (0xF000_0348)

60	T	A	B	105	AX	BX	TX
0	6	11	16	21	29	30	31

```
XT        ← TX || T
XA        ← AX || A
XB        ← BX || B
ex_flag   ← 0b0

do i=0 to 127 by 64
   reset_xflags()
   src1 ← VSR[XA]{i:i+63}
   src2 ← "xvmaddadp" ? VSR[XT]{i:i+63} : VSR[XB]{i:i+63}
   src3 ← "xvmaddadp" ? VSR[XB]{i:i+63} : VSR[XT]{i:i+63}
   v{0:inf}        ← MultiplyAddDP(src1,src3,src2)
   result{i:i+63} ← RoundToDP(RN,v)
   if(vxsnan_flag) then SetFX(VXSNAN)
   if(vximz_flag)  then SetFX(VXIMZ)
   if(vxisi_flag)  then SetFX(VXISI)
   if(ox_flag)     then SetFX(OX)
   if(ux_flag)     then SetFX(UX)
   if(xx_flag)     then SetFX(XX)
   ex_flag       ← ex_flag | (VE & vxsnan_flag)
   ex_flag       ← ex_flag | (VE & vximz_flag)
   ex_flag       ← ex_flag | (VE & vxisi_flag)
   ex_flag       ← ex_flag | (OE & ox_flag)
   ex_flag       ← ex_flag | (UE & ux_flag)
   ex_flag       ← ex_flag | (XE & xx_flag)
end

if( ex_flag = 0 ) then VSR[XT] ← result
```

Let XT be the value TX concatenated with T.
Let XA be the value AX concatenated with A.
Let XB be the value BX concatenated with B.

For each vector element i from 0 to 1, do the following.
Let src1 be the double-precision floating-point operand in doubleword element i of VSR[XA].

For *xvmaddadp*, do the following.
- Let src2 be the double-precision floating-point operand in doubleword element i of VSR[XT].
- Let src3 be the double-precision floating-point operand in doubleword element i of VSR[XB].

For *xvmaddmdp*, do the following.
- Let src2 be the double-precision floating-point operand in doubleword element i of VSR[XB].
- Let src3 be the double-precision floating-point operand in doubleword element i of VSR[XT].

src1 is multiplied by src3, producing a product having unbounded range and precision.

See part 1 of Table 80.

src2 is added[2] to the product, producing a sum having unbounded range and precision.

The sum is normalized[3].

See part 2 of Table 80.

The intermediate result is rounded to double-precision using the rounding mode specified by the Floating-Point Rounding Control field RN of the FPSCR.

See Table 46, "Floating-Point Intermediate Result Handling," on page 344.

The result is placed into doubleword element i of VSR[XT] in double-precision format.

See Table 68, "Vector Floating-Point Final Result," on page 400.

If a trap-enabled exception occurs in any element of the vector, no results are written to VSR[XT].

Special Registers Altered:
FX OX UX XX VXSNAN VXISI VXIMZ

1. Floating-point multiplication is based on exponent addition and multiplication of the significands.
2. Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added or subtracted as appropriate, depending on the signs of the operands, to form an intermediate sum. All 53 bits of the significand as well as all three guard bits (G, R, and X) enter into the computation.
3. Floating-point normalization is based on shifting the significand left until the most-significant bit is 1 and decrementing the exponent by the number of bits the significand was shifted.

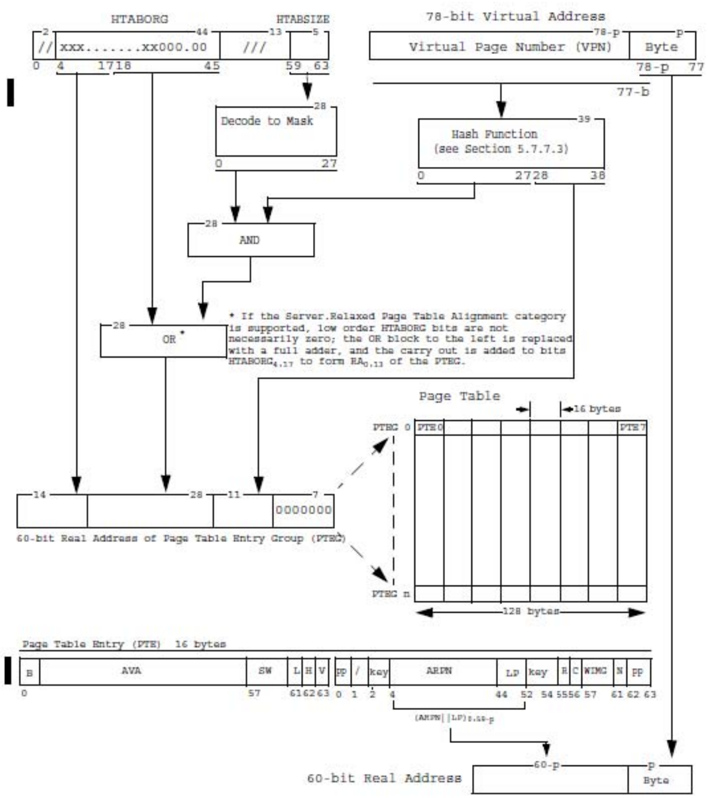


Figure 19. Translation of 78-bit virtual address to 60-bit real address

Modular Infrastructure Design

- Interface Design
  - Allow composition of models with wide variety of performance / functionality characteristics
  - Allow models to be constructed in a variety of languages
    - C, C++, SystemC, Java, Python, etc.
    - Also hardware-based models (FGPA)
  - Allow models to be built as self-contained components
    - Avoid the "build the world" approach
  - Support for "binary-only" components
    - Allows industry to contribute while protecting IP
- Core Services
  - Must provide the "right" abstractions
  - Must be high-performing
  - Must enable parallel execution

The Simulation Ecosystem

toolchain for target platform

source-level debugger

disk image creation

trace generation

trace sampling

design partitioning (for FPGA)

JIT infrastructure

processor simulation

statistics collection

device simulation

display and control

configuration

event dispatch

functional verification suite

performance verification suite

More Software Engineering

- Use a modern source control tool
  - e.g. git, mercurial
- Establish and enforce source code conventions
- Establish a developer community
  - Wikis, forums, mailing lists, etc.
- Encourage “test-driven development”
- Automate regression testing
  - Log and retain test results (Health Monitor)

Final Thoughts

- This is a noble cause
  - but it will be difficult
- There are some challenging technical issues
  - parallelization
  - correlation of models at differing levels of abstraction
- There are equally challenging non-technical issues
  - how to get hardware designers to practice good software engineering
  - how to incent development of infrastructure and sharing of models
  - how to build an ecosystem around a common simulation infrastructure
  - how to maintain expertise in the infrastructure
- This workshop is a great start

References

- Application of full-system simulation in exploratory system design and development, J.L. Peterson et al., IBM Journal of Research and Development, Volume 50, Number 2/3, Spring 2006, pp. 321-332
- Mambo – A Full System Simulator for the PowerPC Architecture, P. Bohrer et al, ACM SIGMETRICS Performance Evaluation Review, Volume 31, Number 4, March 2004, pp. 8-12
- Full Circle: Simulating Linux Clusters on Linux Clusters, Luis Ceze et al, in Proceedings of the Fourth LCI International Conference on Linux Clusters: The HPC Revolution 2003
- A Framework for End-to-end Simulation of High-performance Computing Systems, W. Denzel et al, Simulation 86(5-6) 2010, pp. 331-350
- A Cycle-accurate, Cycle-reproducible multi-FPGA System for Accelerating Multi-core Processor Simulation, Sameh Asaad et al, 20th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Feb 2012