2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Architectures to tradeoff performance vs debug for software development on emulation platforms

Loganath Ramachandran, Ph.D,
Verikwest Systems Inc

Ragavendar Swamisai,
Belmont Computing Inc

Prof. Makato Ikeda, Ph.D
University of Tokyo, Japan

VeriKwest

Belmont Computing

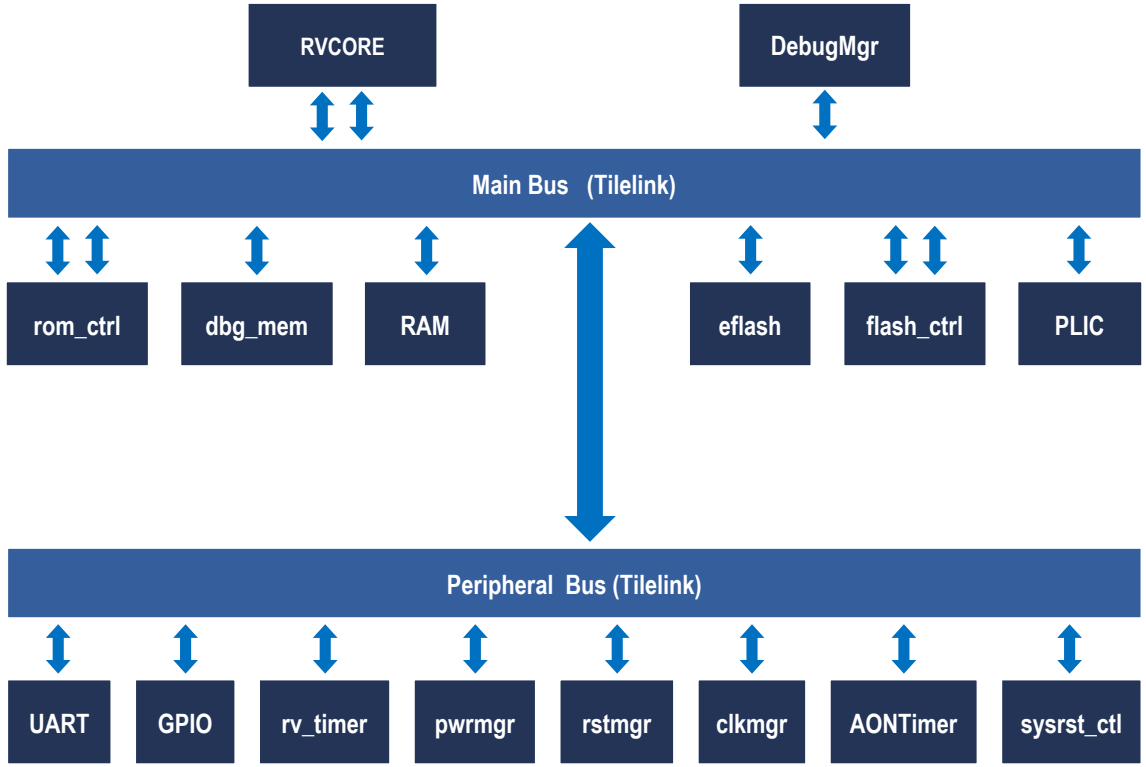
accellera SYSTEMS INITIATIVE

Agenda

- Brief description of SOC
- Early software development requirements
- Emulation use models
- Our solution
  - Debug focused architectures
  - Performance focused architecture
- Implementation ideas on the Palladium engine
- Results

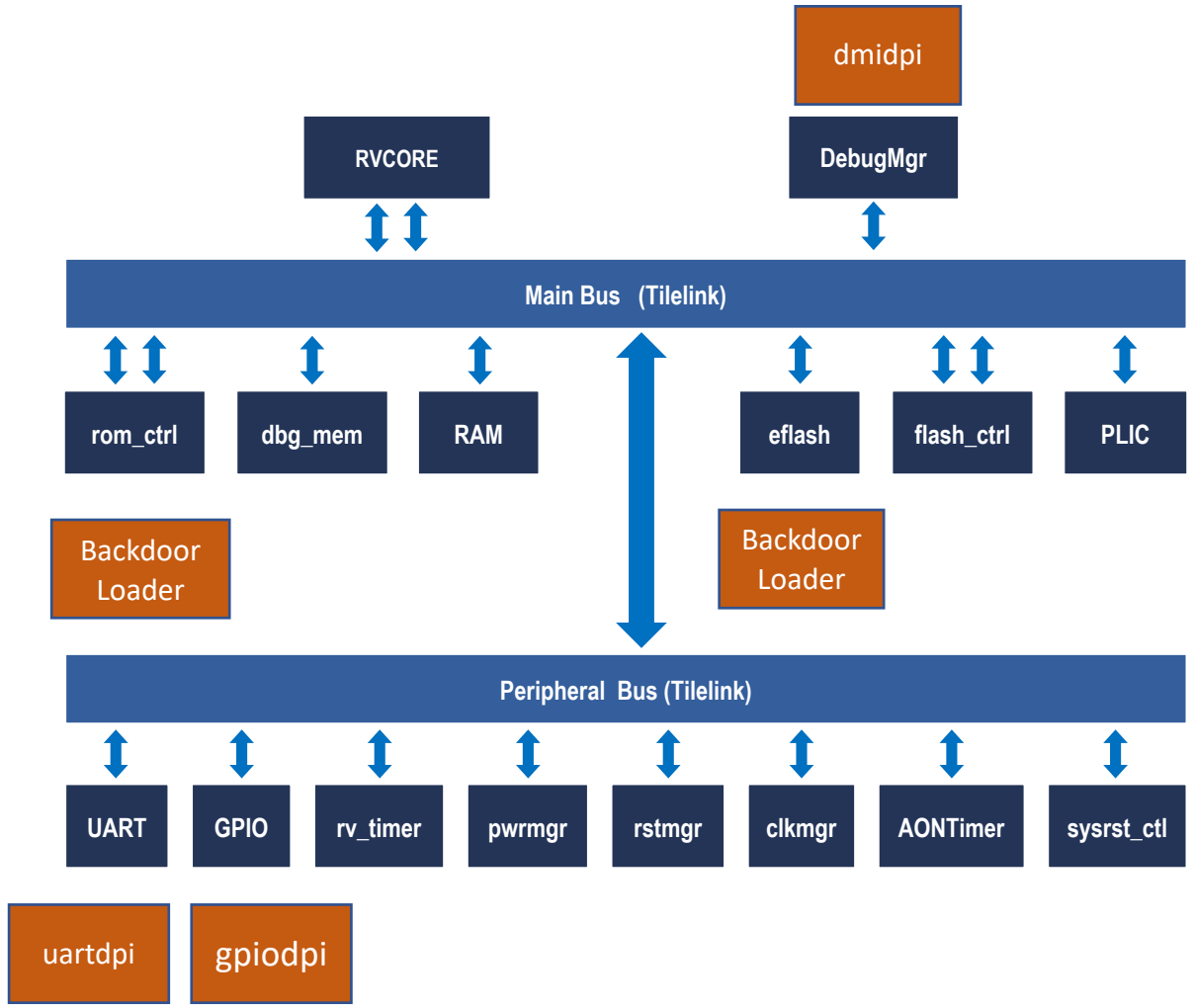
A typical SOC (Athos)

- RISC-V-based SOC platform using the Ibex core
- Easily expandable architecture
- Implemented entirely in System Verilog
- Designed for emulation and rapid prototyping
- Complete software stack (based on C code)
- RISC-V compatible debug components
- Flexible memory architecture
  - 512kB Flash
  - 64kB SRAM for data
  - 8kB ROM for boot



Verification Layer

- Backdoor interfaces for ROM and Flash
- Backdoor interfaces to SW programmable registers
- Virtual I/O blocks for peripherals
- Source code debug interface with OpenOCD.



Verification Layer

- Manages the signals being driven to the SOC
- Uses DPI functions to read/write the SOC primary inputs/outputs
- Drives as per the protocol requirements

uartdpi
gpiodpi
dmidpi

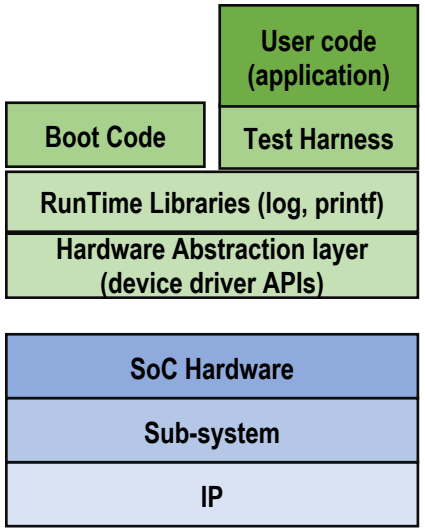
- Captures the signals being driven out of the SOC
- Decodes signals as per protocol requirements
- Outputs data to a unix file or terminal

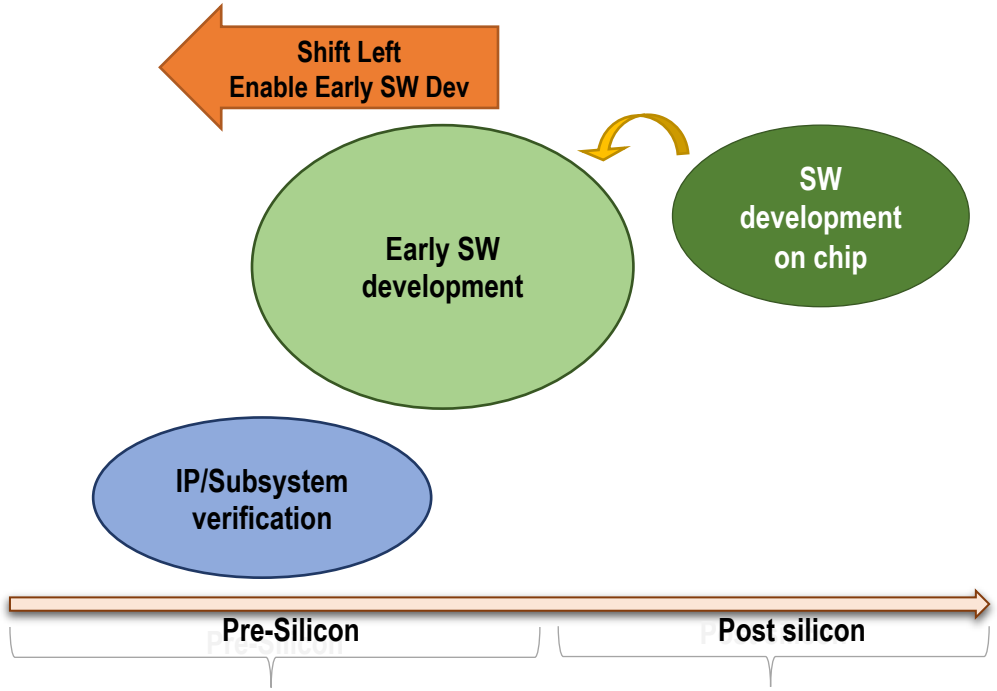
- Manages backdoor loading of ROM
- Manages backdoor loading of FlashRAM

Code is not synthesizeable and is based on DPI.

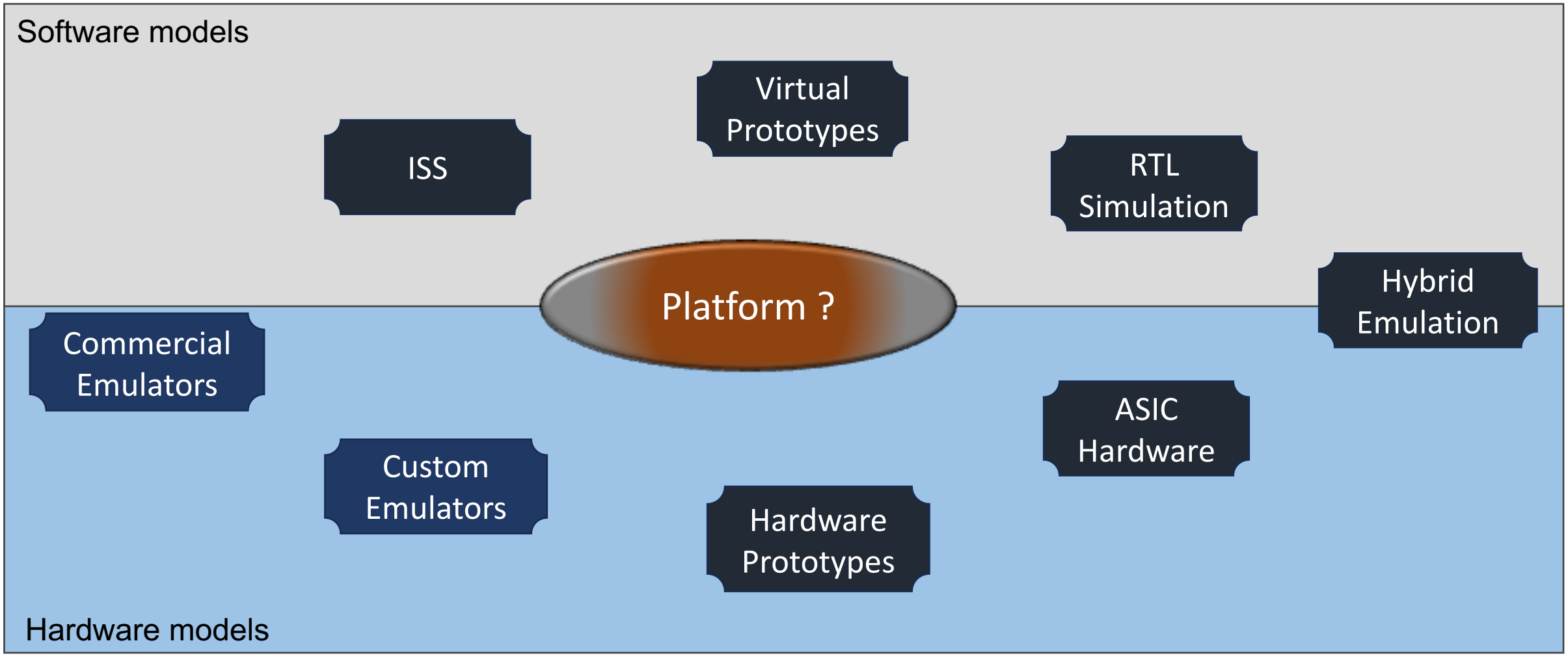
Early software development

- Develop SW concurrently with HW and integrate HW/SW as early as possible.
- Regression testing of multiple complex scenarios involving bare-metal, device interface, firmware and user level software.
- Allows continuous software development before silicon is back.





But… the big question!



Early SW development requirements

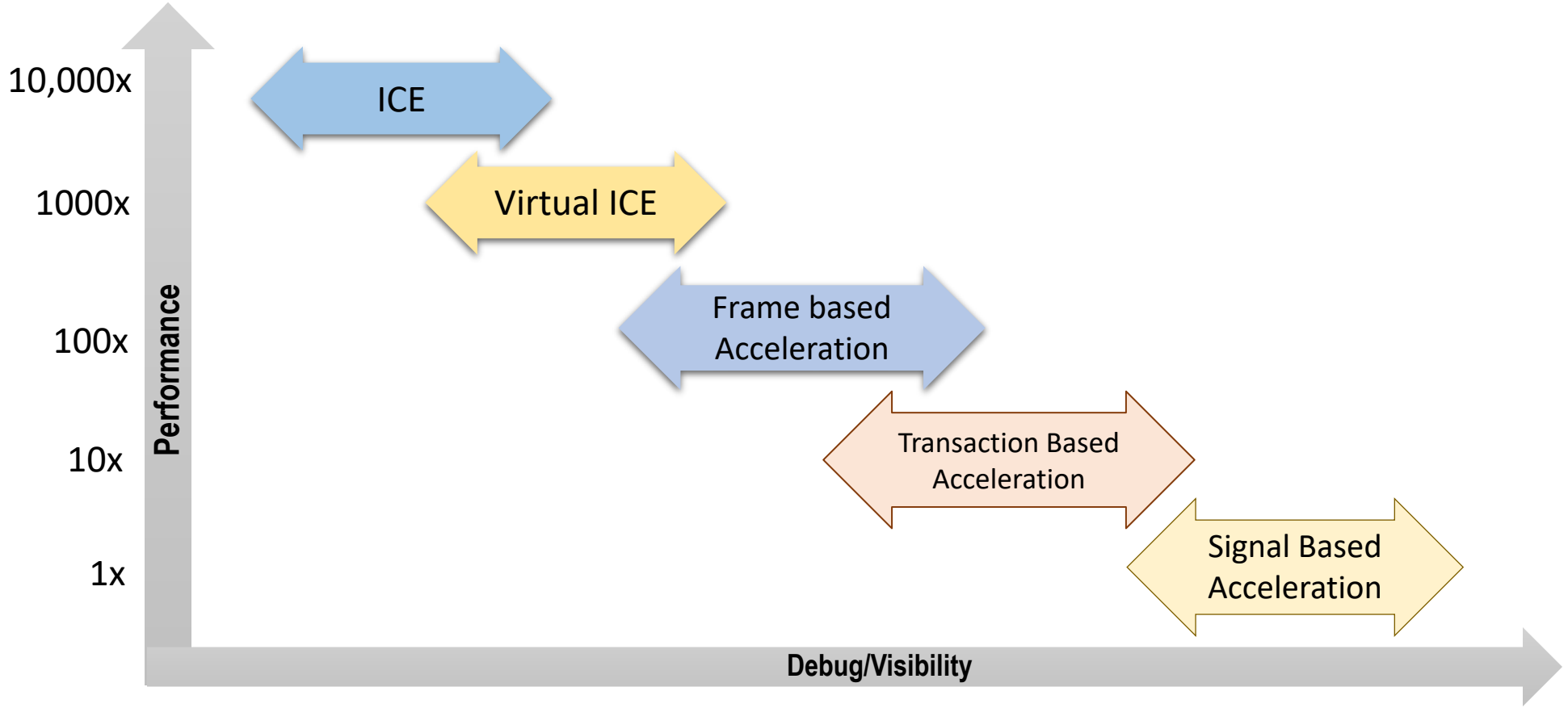
- Support for debug
  - Chase problems through both software and hardware to identify root causes.
  - Reduce turn-around-time for debug and fixing SW/HW
- Very high performance
  - As close to real-world as possible
  - Needed to bringup the complete software stack
- Additional effort to enable the platform (NRE)
  - Ability to quickly bring-up HW on emulators
  - Enable SW users to quickly change their SW and bring-up on the platform
  - Enable easy change of HW in case issues are found and fixed in H/W
- Shift Left strategy feasibility
  - How early can we start?

Emulator is a viable SW dev platform

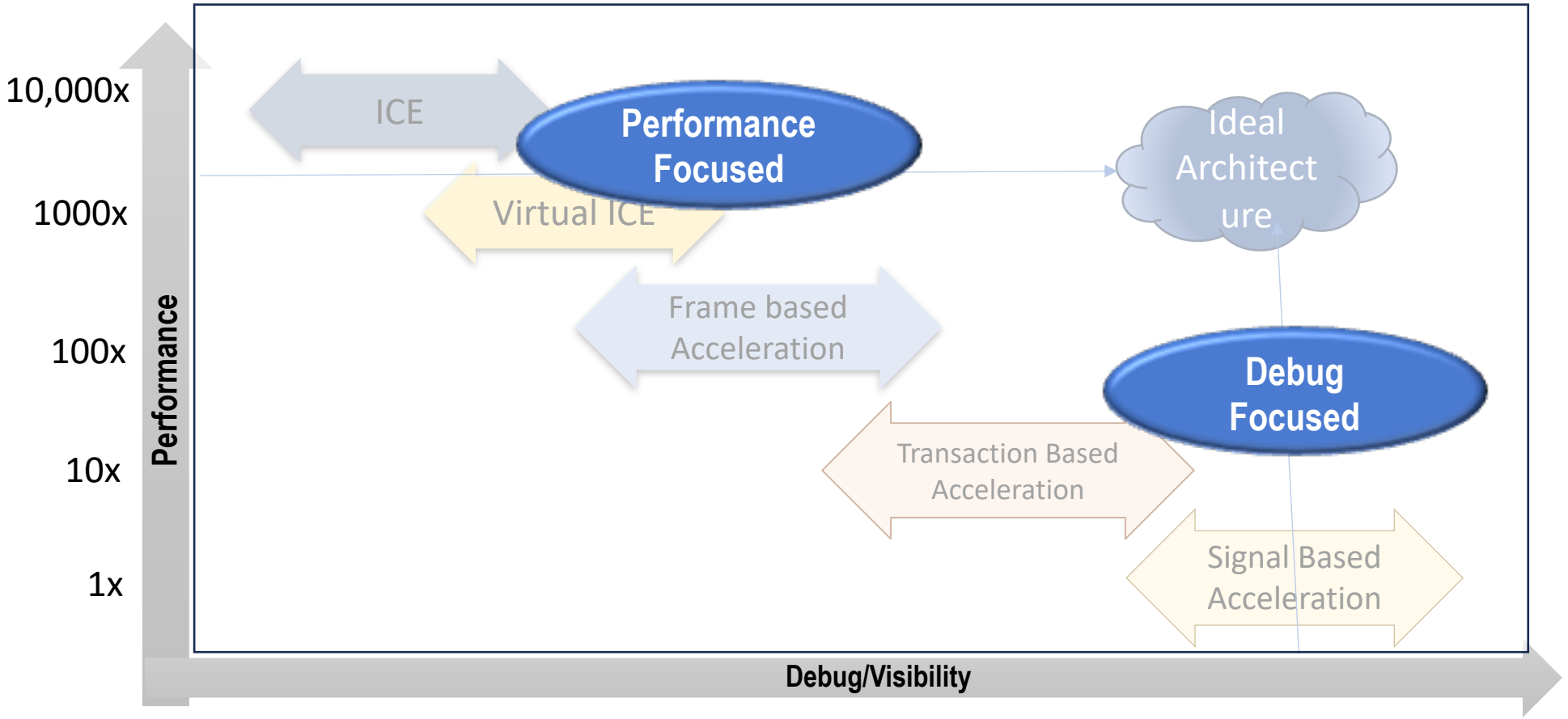
- *Performance* – not the best but fast enough for developing software
- *Shift Left* - can be brought up as soon as RTL reaches a level of maturity
- *Mature build flows* - compiling -> download image -> run -> debug
- *Debuggability* - well developed debug solutions for both HW/SW
- *Deployablility*– remote interactive usage
- *Scalability* – support multiple users on a single emulator
- *IP protection* – HW is synthesized to bitstream, SW is compiled to object code
- *Collaboration* - HW & SW teams working in parallel
- *Productivity* - SW users use their own dev environment to develop SW

Our solution can be implemented on any emulation platform
But we show the implementation on the Palladium

Emulation Use Models



Our solution



Design requirements

- Users should be able to switch between these two architectures seamlessly
- Reduce implementation cost of solution
  - Share as much code as possible
    - Underlying RTL has to be the same in both architectures
  - Build the solution on top of existing emulator capabilities
    - E.g, ixclkgen based on Palladium “cake” technology

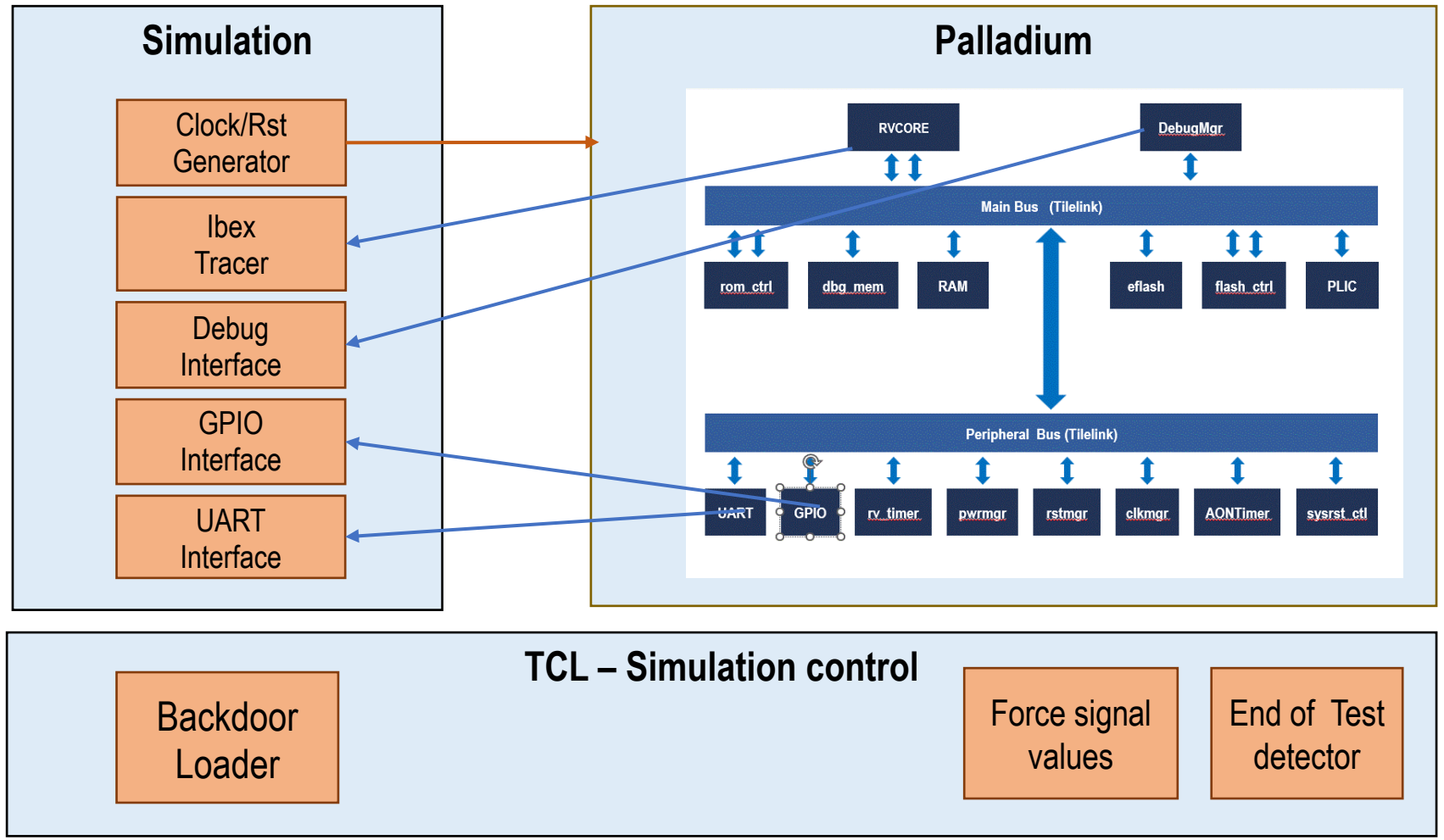
- RTL bugs found in one should be easily reproducible in the other
  - Use the same RTL between both platforms.
- Underlying software code has to be the same for both arch
  - Common compile and run of the SOC software
- “Testbench” architecture will differ but minimize the differences

Features

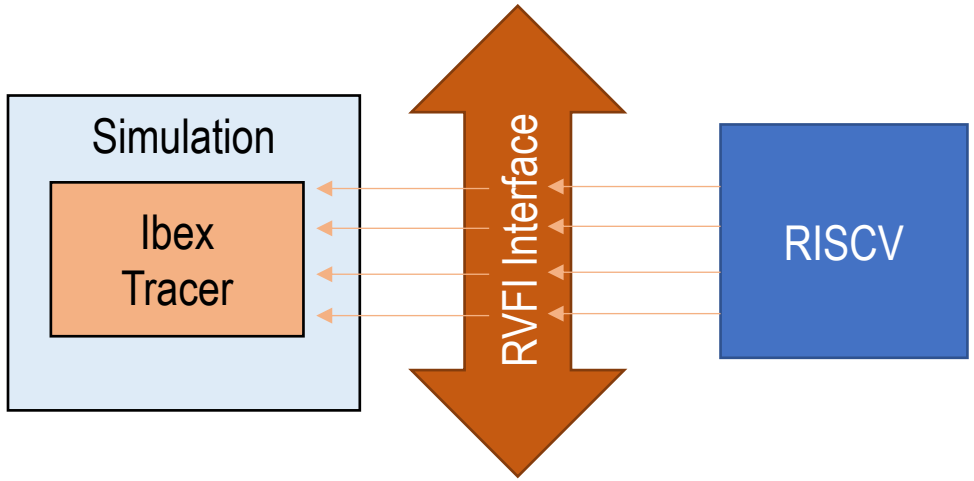
Debug Focused Implementation	Performance Focused Implementation
Cycle Accurate sync between TB and Design	Transaction accurate sync between TB and design
Single common clock design between TB and design	Separate clocks between TB and design
Ability to reuse common debug/monitor components	Implement transactors to support debug/monitors
Performance 10-100 Khz	Performance 100 Khz - 10 Mhz
Tracer to trace each retired instruction	Transaction buffer required to hold symbols
Support for software debugger to set breakpoints	
Signal based Virtual components	Transaction based Virtual components

Common software compilation flow
Common RTL

Debug Focused Architecture



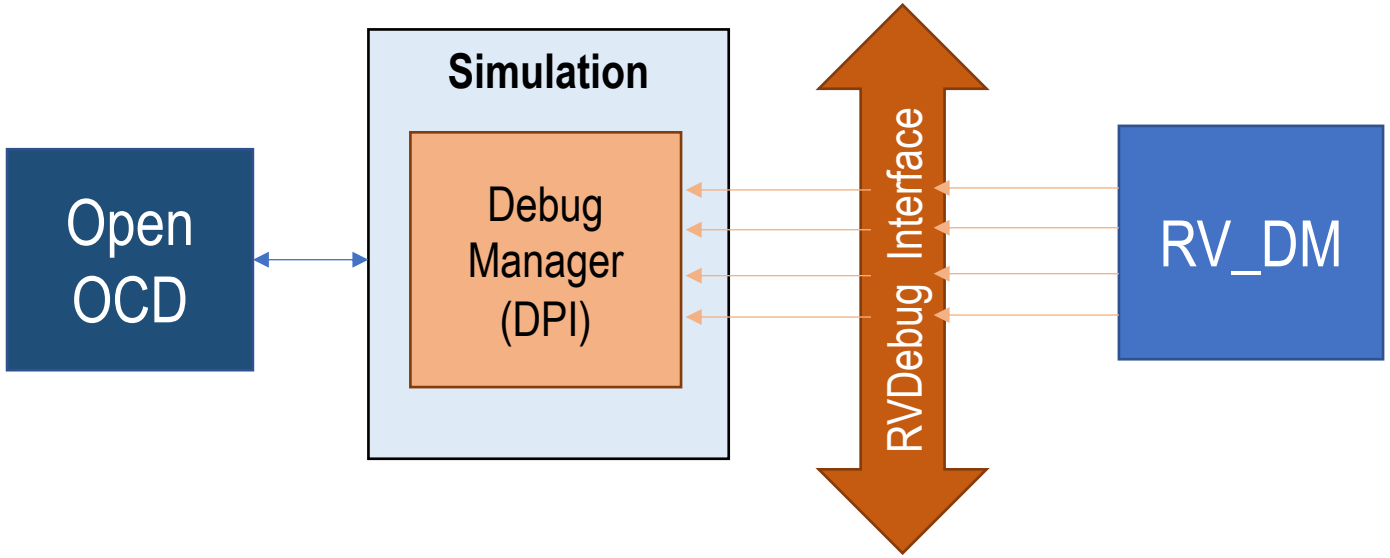
RISC-V Tracer



- Enables easy debug of software execution
- Uses RVFI interface on Ibex
- Signals are transferred using SBA
- Tracer receives retired instructions
- Prints out instruction information

```
554500              525  000080f0    08458593    addi       x11,x11,132  x11:0x00008000 x11=0x00008084
555500              526  000080f4    00052283    lw         x5,0(x10)    x10:0x00008080  x5=0x0040006f PA:0x00008080 store:0x00000000
556500              527  000080f8    0509        c.addi     x10,2        x10:0x00008080 x10=0x00008082
557500              528  000080fa    feb56de3    bltu       x10,x11,80f4             x10:0x00008082 x11:0x00008084  x0=0x00000000
561500              532  000080f4    00052283    lw         x5,0(x10)    x10:0x00008082  x5=0x40810040 PA:0x00008082 store:0x00000000
562500              533  000080f8    0509        c.addi     x10,2        x10:0x00008082 x10=0x00008084
563500              534  000080fa    feb56de3    bltu       x10,x11,80f4             x10:0x00008084 x11:0x00008084  x0=0x00000000
564500              535  000080fe    80818513    addi       x10,x3,-2040              x3:0x10000800 x10=0x10000008
565500              536  00008102    81018593    addi       x11,x3,-2032              x3:0x10000800 x11=0x10000010
```

Debug Interface



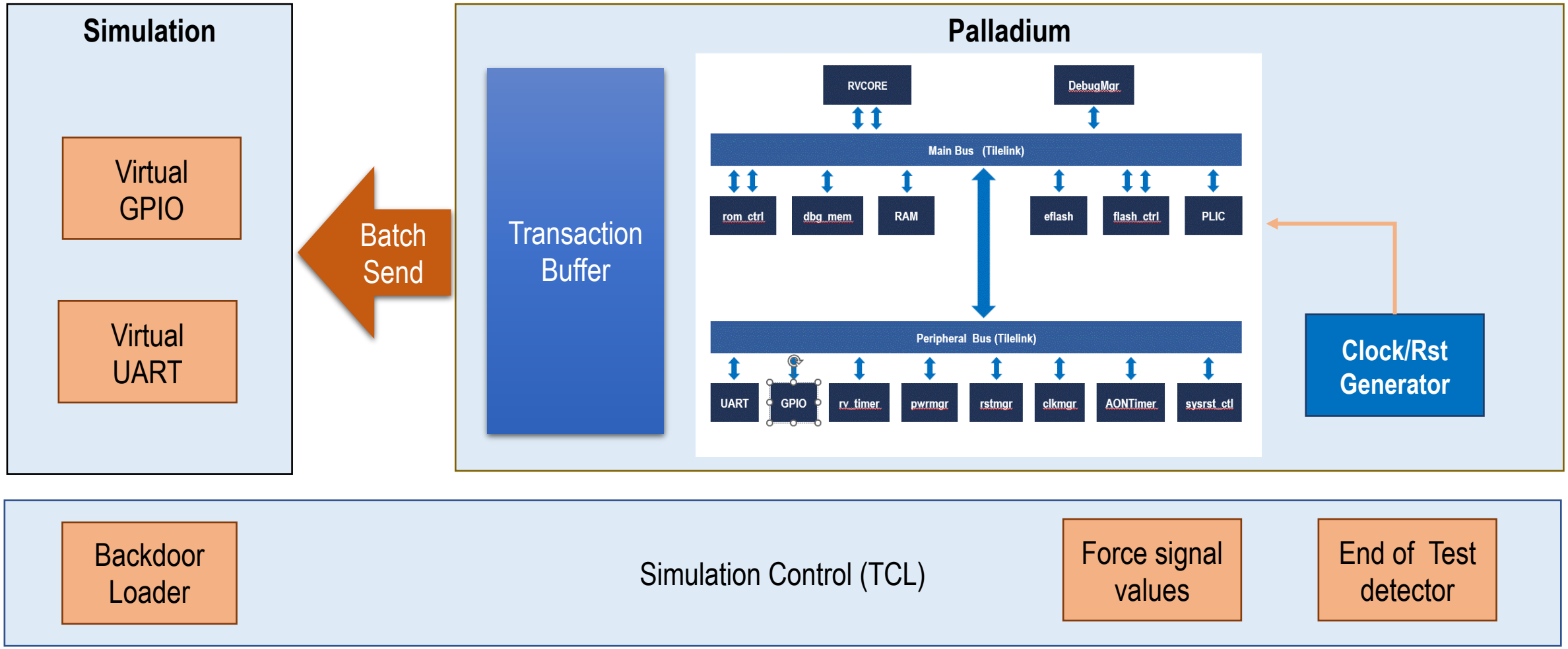
- Interfaces to the RV_DM component
- Interfaces to the debugger using DMIDPI component
- Uses valid-ready interface to RV_DM
- Compatible with RISC-V Debug specification

DFA uses “run” mode

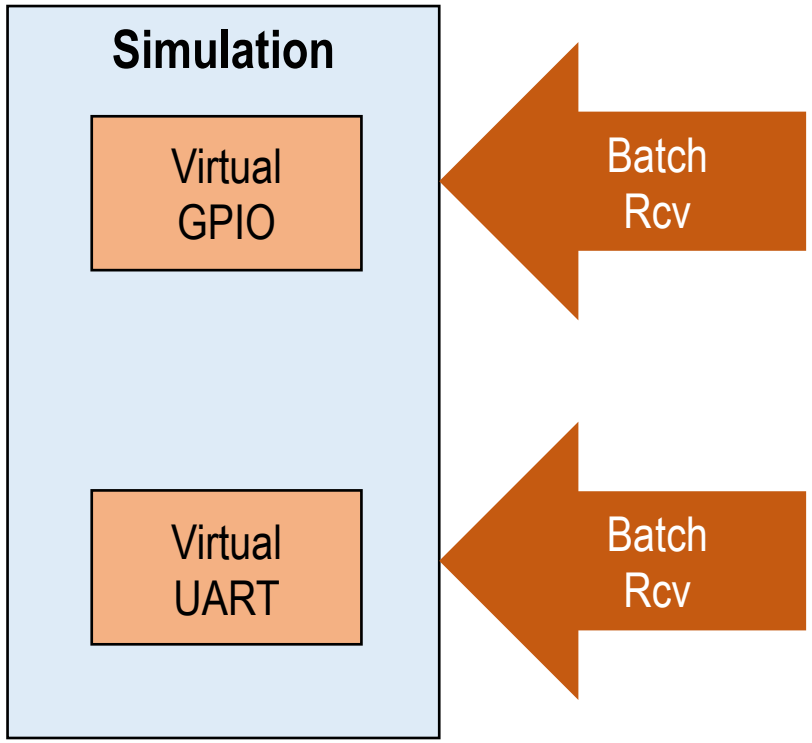
- HW-SW sync happens when
  - TB delay
  - DUT delay
  - All clock edges, even if generated in HW
- Hardware stops at every software delay event for synchronization.
- Number of syncs very high as expected

```
xmsim> xc xt0 zt0 on -run
xmsim> run
```

Performance Focused Architecture



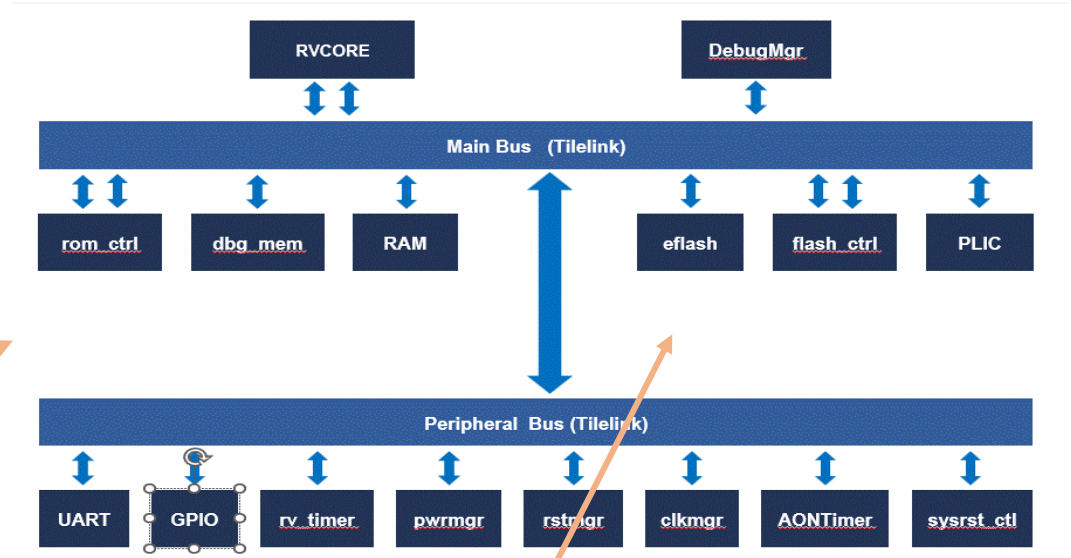
High performance virtual components



- Receives transactions from DUT
  - Batch mode
- Virtual UART (receive only)
  - Decodes UART transactions
  - Uses uartdpi to print log
- Virtual GPIO (receive only)
  - Decodes GPIO transactions
  - Uses gpiodpi to print log

Backdoor Loading

- Using xe_run.tcl
  - memory –load %readmemh
- Works in both modes
  - Path is different




```
memory -load %readmemh <hierarchical_path> -file <filename>
       -start <locn> -end <locn>
```

```
memory -load %readmemh <hierarchical_path – file <filename>
```

Debugging on the Palladium

- **Waveforms can be dumped, but ….**
  - Waves are great for HW engineers … but not suitable for SW debug
  - Waveforms of all nodes for large multi-day workloads is too much data
  - Reduced emulation speed and performance while dumping waves
  - Long process to identify scope of the issue
    - Dumped waves but miss the failure point
    - Schedule more emulation time to run again to narrow the scope of issue

- **SDL mechanism available for monitoring, but ….**
  - Pure TCL functions that run in the background periodically (as fast as 2 ms) and execute commands.
  - Each monitor can optionally log output to a file
  - Possible to miss some events if polling too slow.
  - Performance can be degraded if not used appropriately by users

PFA uses native clock generation


```
clockSource -add clk_i
clockOption -add {technology CAKE 1}
clockFrequency -add {clk_i 1Mhz}
```

```
`timescale 100 ns / 100 ns
`define IXCclkgenTs 100 ns / 100 ns

module clk_gen(output wire clk_i_0);

`ifdef IXCOM_COMPILE
  initial $ixc_ctrl("map_delays");
  initial $ixc_ctrl("hotswap_top");
`endif

  // Generate logic for clock sources
  ixc_master_clock #(5) ixcg_0(clk_i_0  );

  // Bind clock sources to generated clock signal
  ixc_cakebind ixcb_0 (clk_i, clk_i_0);
  ……
```

Optimizing the tbsyncs

Experiment	DUT Speed	Tbsyncs #	TBCalls #	HW-EMU busy %
TBRUN mode with ALL SDLs to dump trace, uart and other debug info	26.92 KHz	900700	0	1.05 sec (1.04%)
TBRUN without SDL	54.97 KHz	900700	0	1.05 sec (2.13%)
NBRun with ALL SDLs to dump trace, uart and other debug info	68.34 KHz	1946	0	2.75 %, 5.93 sec
NBRun with SDL to dump trace info	2236.76 KHz	29	0	86.82 %, 11.44 sec
NBRun without any SDL & waves download	2575.96 KHz	4	0	99.92 %, 113.54 sec

PFA uses asynchronous run mode

- Both software and hardware domains advance the time concurrently instead of alternately.
- Timestamps in the software testbench and hardware DUT are not synchronized.
- Hardware stops only when the hardware needs service or when the software has to send some data to the hardware.

```
xmsim> xc xt0 zt0 on -nbrun 100ns
xmsim> run
```

Results

- Coremark was compiled and run on both architectures

Architecture	DUT speed	#Tbsyncs	HW Busy %
DFA	26.92 kHz	900700	1.04%
PFA (with dumping enabled)	2236.76 kHz	29	86.82

Summary

- Plethora of choices available for early sw development platform for SOC
- Emulators can become a platform of choice
- Implemented two architectures on the Palladium
  - Performance Based
  - Debug Based
- Enabled switchability by maximal reuse of code/components
- Demonstrated our method on a RISC-V SOC

And finally ..