ENABLING A NEW PARADIGM OF SYSTEM-LEVEL DEBUG PRODUCTIVITY WHILE MAINTAINING FULL IN-CIRCUIT EMULATION PERFORMANCE

CDNLive! Silicon Valley 2012

Alex Starr

March 13, 2012

INTRODUCTION

- About the author
  - Alex is AMD's Hardware Emulation Architect, driving the emulation research and development initiatives for CPU, APU, and GPU products.
  - Send questions to alex.starr@amd.com
  - Acknowledgements: Brian Fisk & Eric White
- Emulation use at AMD
  - More than 100 users across design engineering, platform engineering, product engineering, and software engineering.
- Emulation is being used to reduce product time to market
  - Improved design health.
  - Post-silicon infrastructure readiness.
  - Software enablement.
- Been doing emulation for many years.
  - Cadence® Palladium® technology introduced over six years ago.

PROBLEM STATEMENT

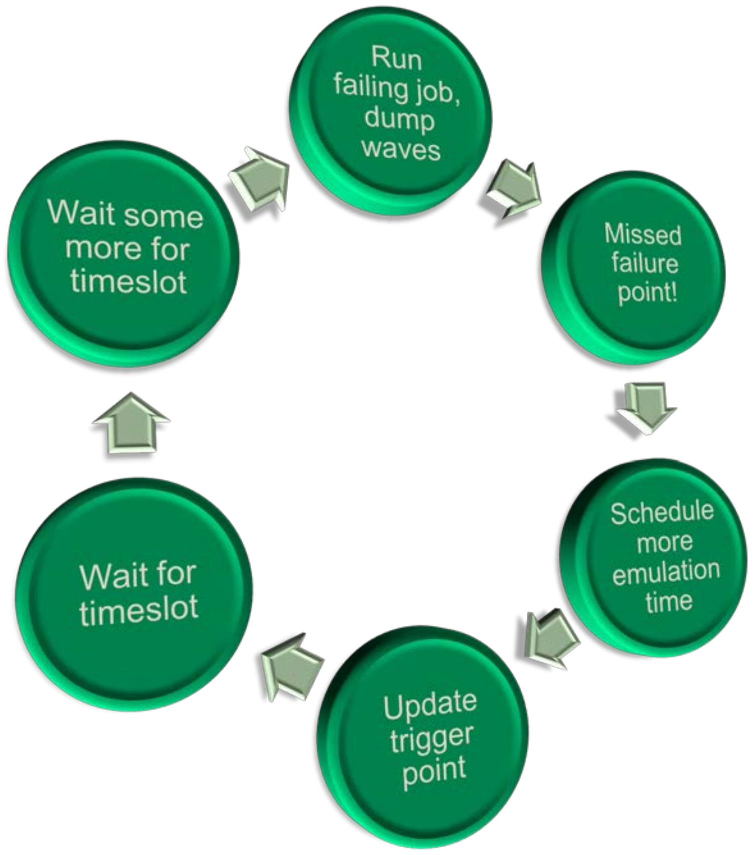
- Debugging designs on hardware emulators today is a challenge when compared with software simulation capabilities.
- Debugging becomes more and more challenging as design complexity increases.
- Large workloads, such as OS boots, driver testing, and application code make the problem even harder.

- Needle in a haystack: Where do I dump waves?
- The faster the emulation technology becomes, the worse the problem becomes.
  - Bigger haystack!
- Users demanding more debug visibility.

THE TRADITIONAL EMULATION DEBUG CYCLE

Large workload debug might go something like this…

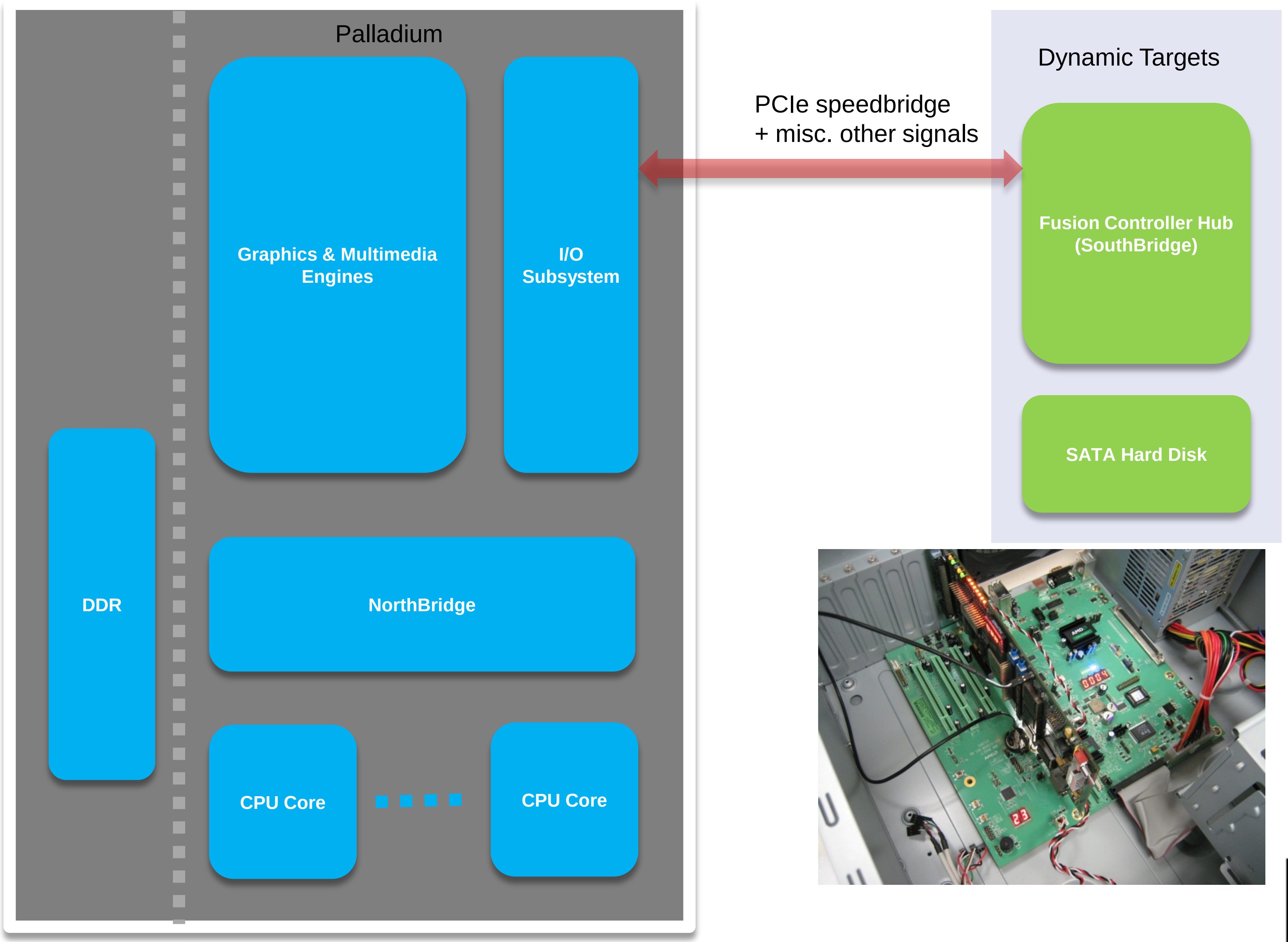
1. "I dumped waves, but missed the failure point."
2. "I'll have to schedule my ten-hour job and try again."
3. "I'll have to figure out how to trigger in the right place."
4. "The next available time on the emulator is next week."
5. Next week … "What was I doing again?"
6. "I dumped waves and missed the failure again!"

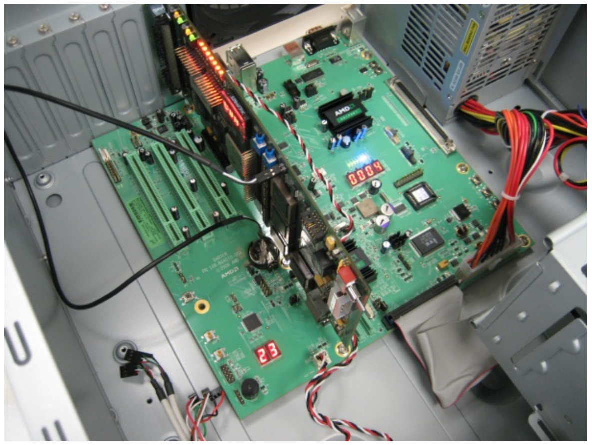

Run failing job, dump waves
Missed failure point!
Schedule more emulation time
Update trigger point
Wait for timeslot
Wait some more for timeslot

WHAT'S WRONG WITH WAVES?

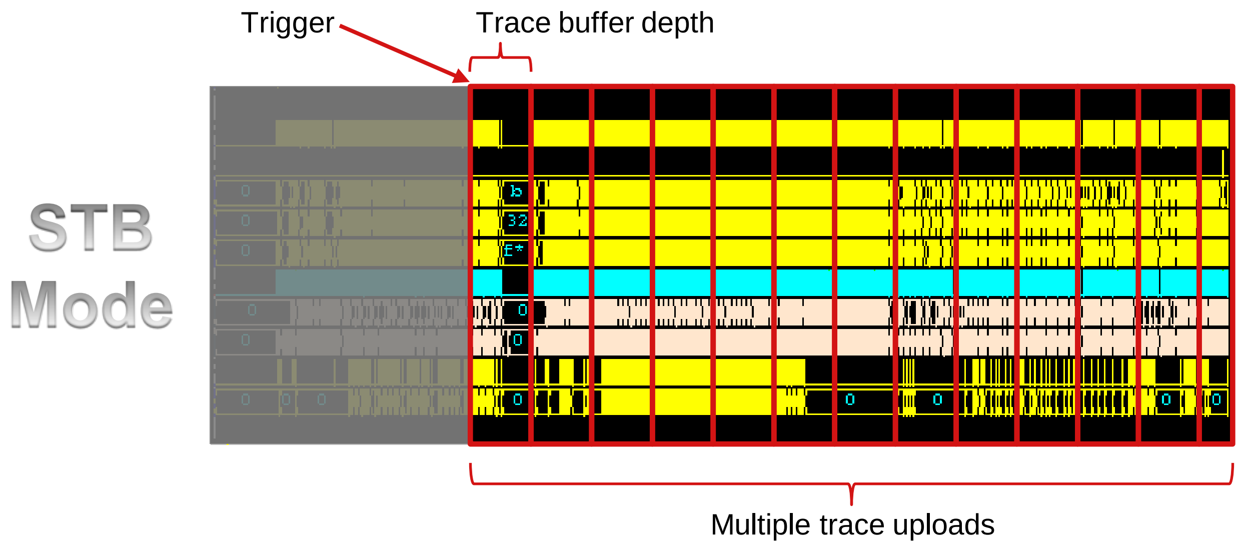
- Industry solutions can generate waves with visibility of all nodes, which is great.
- Wave capture limitations:
  - Wave capture depth typically small
  - Speed of emulation while dumping waves
    - Can be slower on some systems
- When using timing-sensitive dynamic targets, even greater restrictions exist:
  - Design clocks must continue to run unhindered at full speed
  - Possible to miss events for back-to-back wave dumps
- Users need more trace ….
  - However, having waveforms of all nodes for large multi-day workloads is too much data (for humans and tools/disks)

DYNAMIC TARGET EMULATION MODEL OVERVIEW

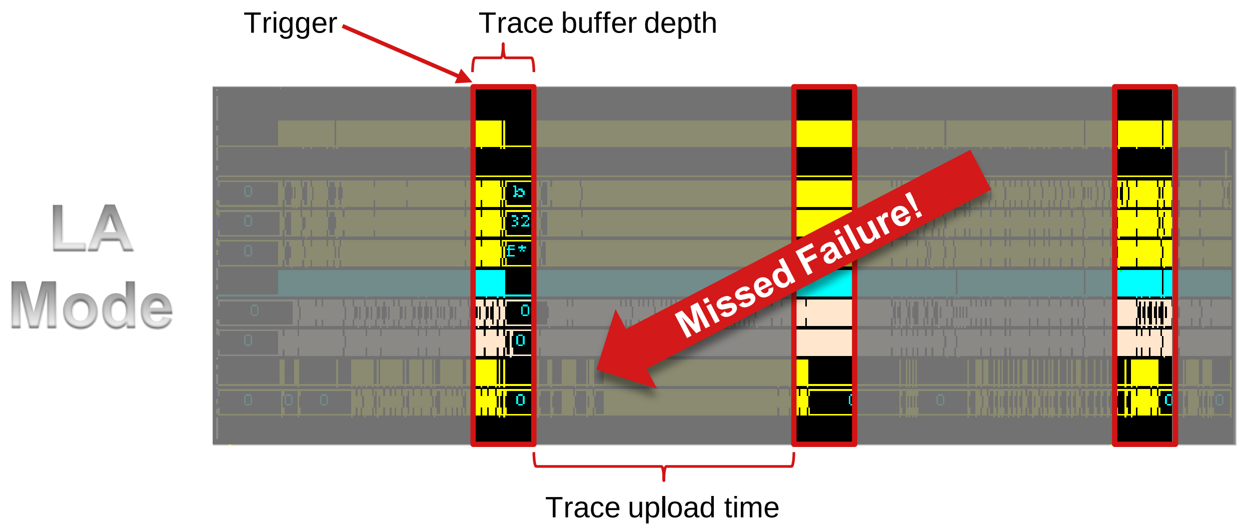




WAVE DUMPING STRATEGIES



- No dynamic targets
- Clocks stopped between uploads
- Takes a long time
- Lots of disk needed



- Timing-sensitive dynamic targets connected
- Clocks cannot be stopped
- Easy to miss point of failure

REQUIREMENTS FOR A NEW CAPABILITY

Requirement
Need to abstract data to a level relevant for end-user debug
Need solution to function with dynamic targets Design clock cannot stop
Need the ability to identify the location for a waveform dump
Should not impact the emulation capacity requirements of designs significantly
Should maintain existing in-circuit emulation speeds
Save and restore continues to function as normal
Capture all design nodes of significance needed for system-level debug
We want to see all data associated with element being monitored Lossless capture, no missed events
Architected such that it is portable across emulation platforms with minimal modification
Easy automation of user actions based on monitored data
No rebuild of databases required when adding monitors

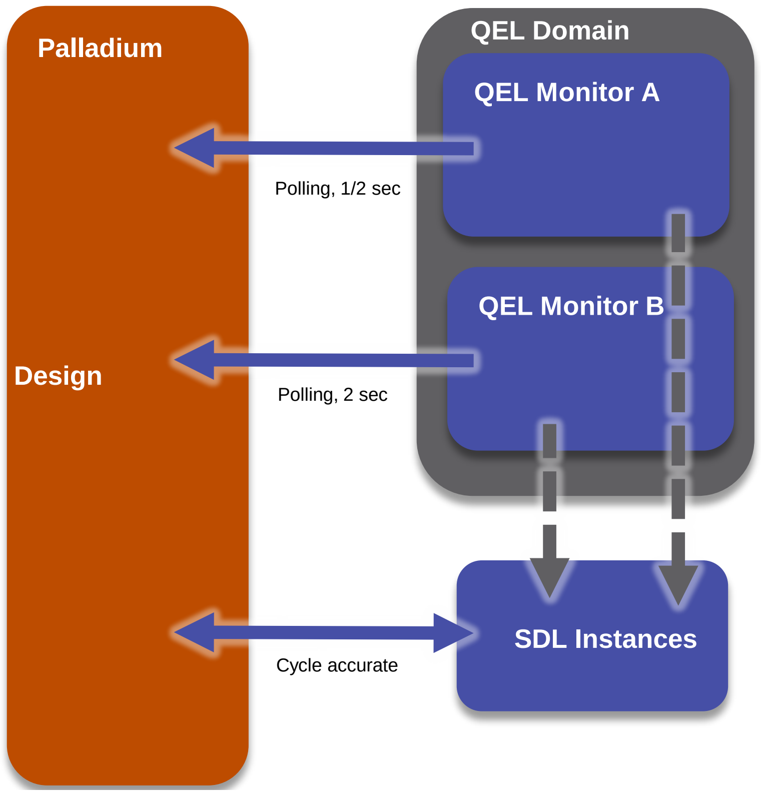
MONITOR COMPARISONS

	Synthesizable RTL monitor	QEL monitor	Transactional streaming monitor
Language	HDL	QEL (tcl) SDL (tdf)	HDL C++
Description	Decoding internal DUT signals into human-consumable formats. Used as foundation for improved monitoring by QEL and transactional streaming monitors.	Pure TCL functions that run in the background periodically (as fast as 2 ms) and execute arbitrary QEL commands. Each monitor can optionally log output to a file or the QEL GUI window without blocking user input.	Using in-circuit acceleration mode to stream data from RTL nodes to the SA workstation for decoding with a C++ process. Targeted to function in situations in which we do not want to stop the clock – with dynamic targets attached.
Output Format	Waves.	Logfiles.	Logfiles, Transaction visualizer.
Pros	Good if you want additional decoded data captured in waveforms.	No hardware or build dependencies. Can iterate quickly on fixes, updates without recompile.	Lossless monitoring. High-bandwidth cycle-accurate trace of activity without wave dumping or interruption. Visualization possible.
Cons	Increases design size. Careful of constructs.	Possible to miss some events if polling too slow. Monitors themselves are not executed in an atomic nature – design still running as commands execute.	Requires simulation acceleration card. Need to deal with buffer flushing management.

DEBUG CAPABILITY: QEL MONITORS

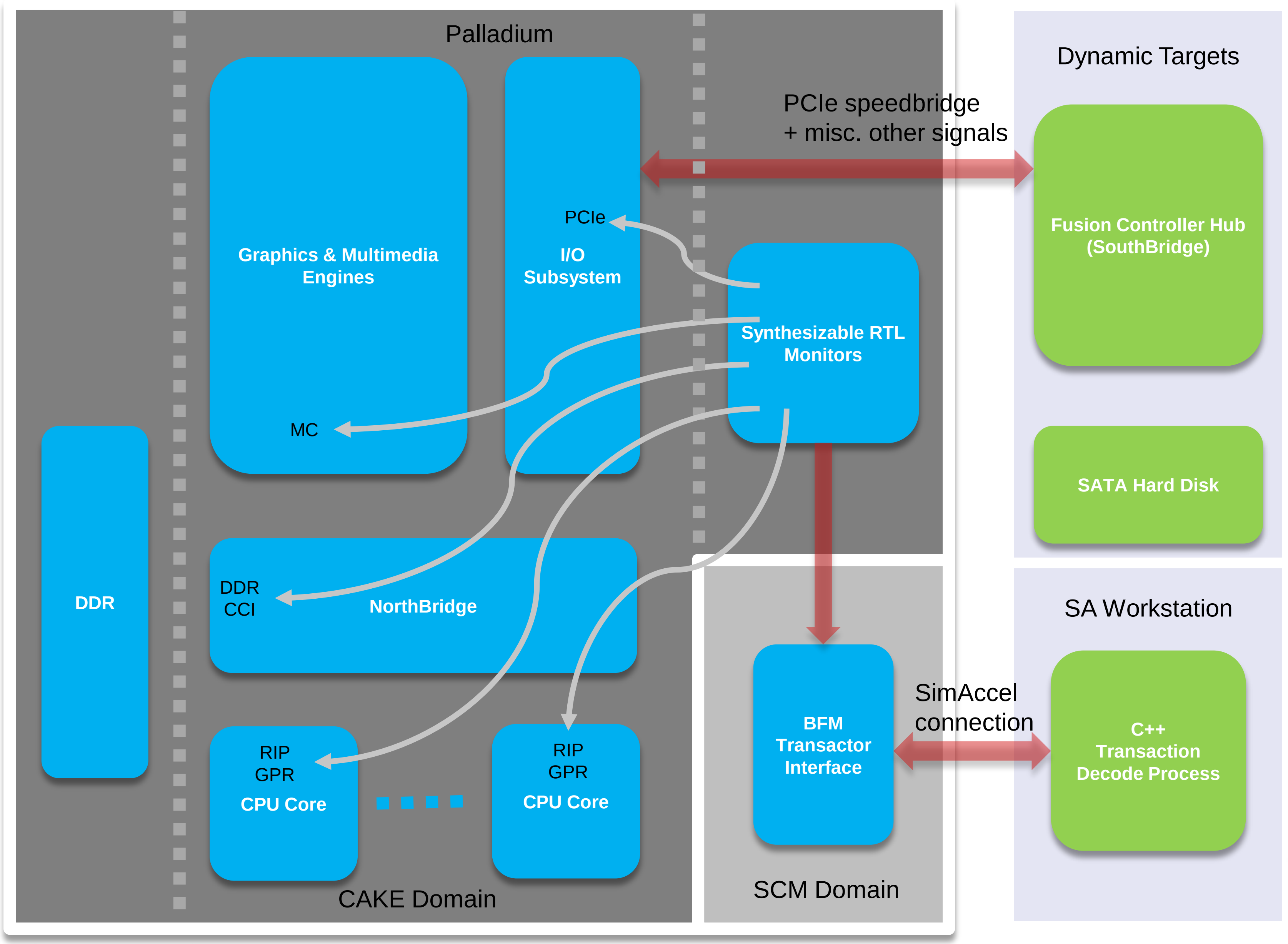
- What are they?
  - Pure TCL functions run in the background periodically (as fast as 2 ms) and execute arbitrary QEL commands
  - Monitor can optionally log output to a file or the QEL GUI window without blocking user input
  - Can be created, enabled, disabled, or changed on the fly & can essentially have any number of them running concurrently
- Value:
  - Provides medium-bandwidth 'periodic' trace of activity without wave dumping or interruption
  - Enables coarse-grained monitoring without any hardware or build dependencies
  - Automation of sequence of user input events over time based on emulator conditions or SDL state possible
  - Lightweight and very flexible, lots of potential uses

QEL Monitor Architecture



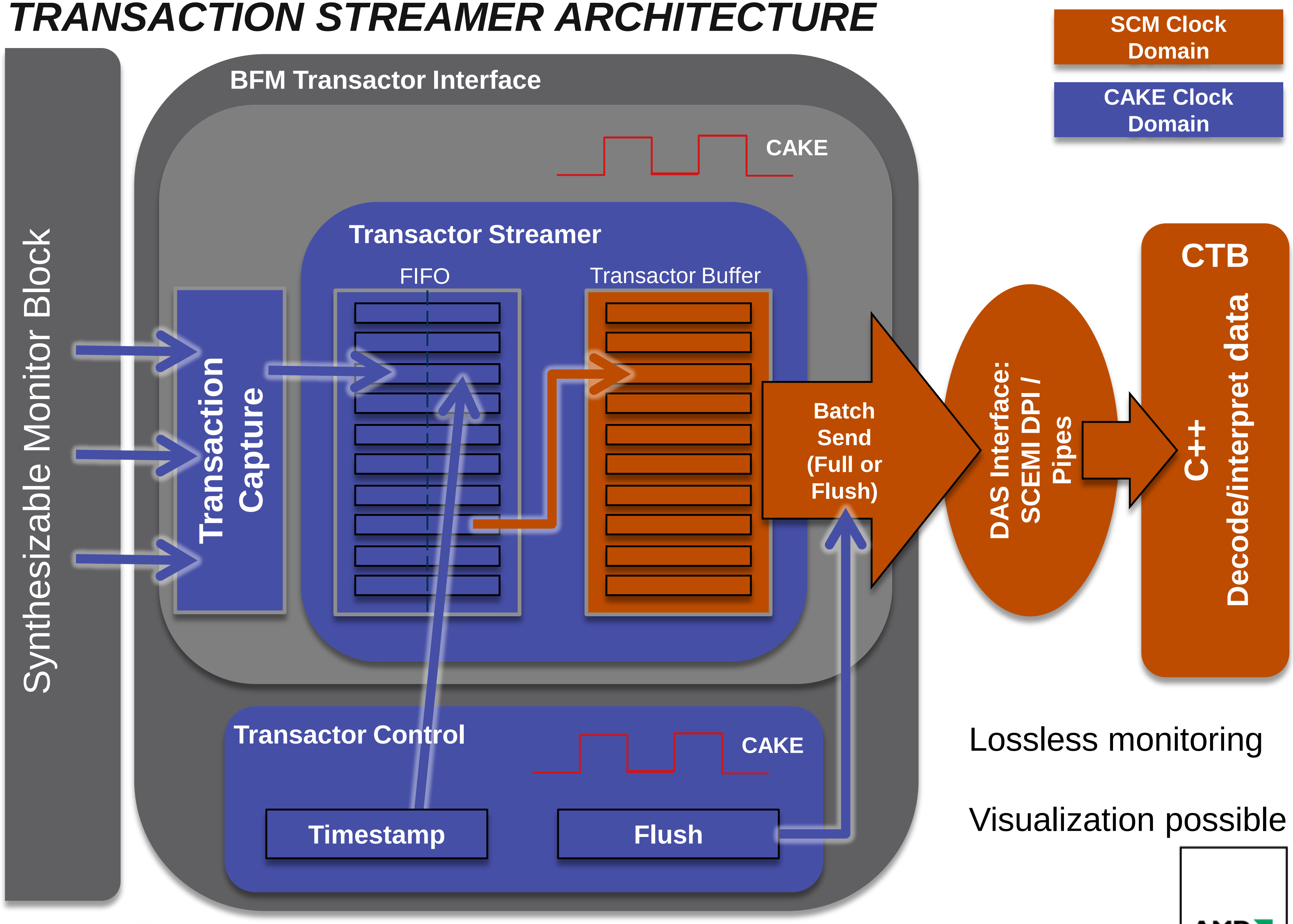
- Notes:
  - Polling-based, not event-based
  - Monitors themselves are not executed in an atomic nature – design still running as commands execute
  - Dependency on TCL event loop – will not run during download or when GUI is otherwise frozen

TRANSACTIONAL STREAMING MONITOR SYSTEM



AMD

TRANSACTION STREAMER ARCHITECTURE



Lossless monitoring

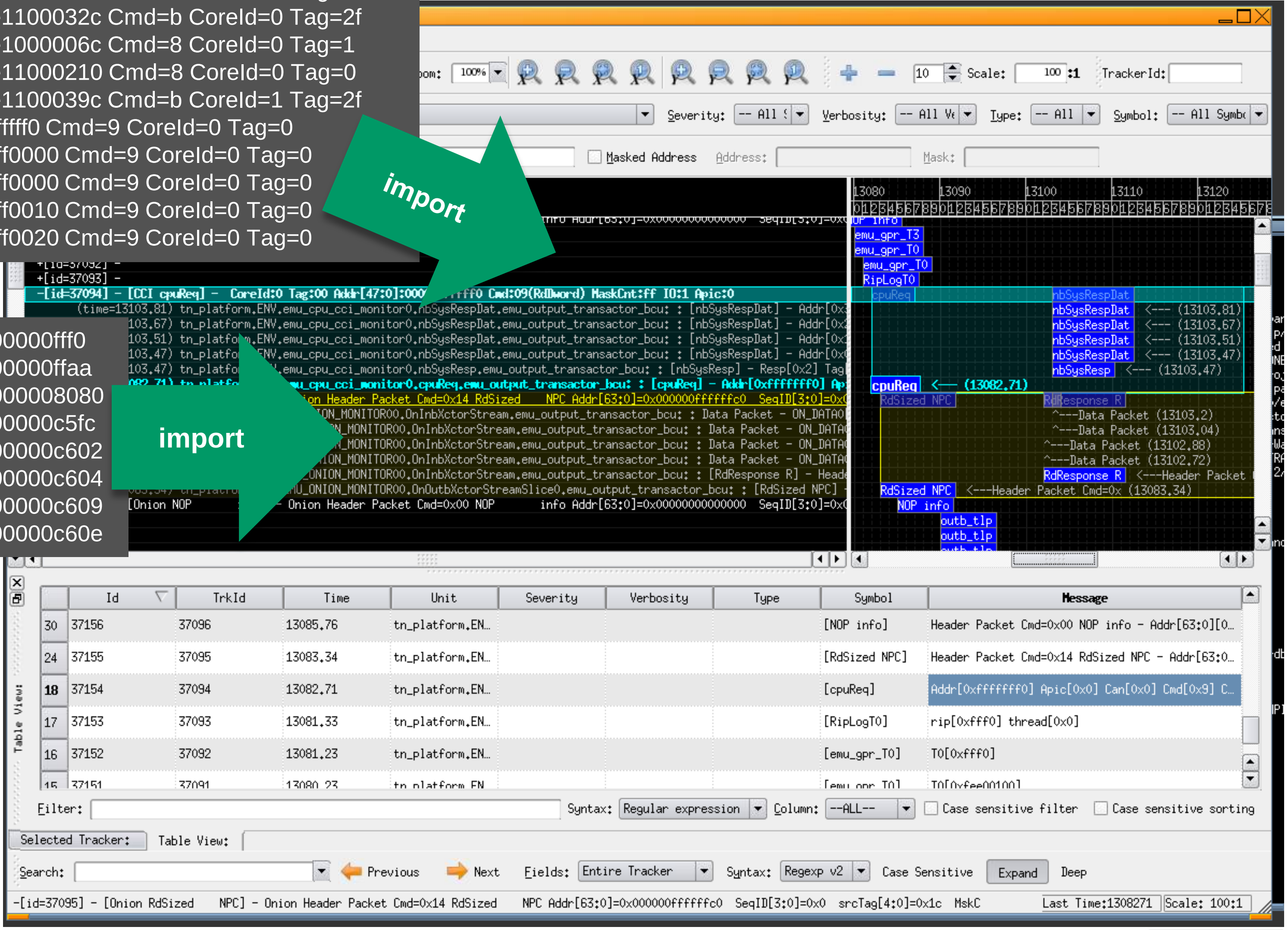
Visualization possible

TRANSACTION VIEWER

Data from all monitors is imported into an AMD internal transaction database

```
3669967: CpuReq Addr=fe1100032c Cmd=8 CoreId=0 Tag=0
3669967: CpuReq Addr=fe1100032c Cmd=b CoreId=0 Tag=2f
3669967: CpuReq Addr=fe1000006c Cmd=8 CoreId=0 Tag=1
3669967: CpuReq Addr=fe11000210 Cmd=8 CoreId=0 Tag=0
3669967: CpuReq Addr=fe1100039c Cmd=b CoreId=1 Tag=2f
3669967: CpuReq Addr=fffffff0 Cmd=9 CoreId=0 Tag=0
3669967: CpuReq Addr=ffff0000 Cmd=9 CoreId=0 Tag=0
3669967: CpuReq Addr=ffff0000 Cmd=9 CoreId=0 Tag=0
3669967: CpuReq Addr=ffff0010 Cmd=9 CoreId=0 Tag=0
3669967: CpuReq Addr=ffff0020 Cmd=9 CoreId=0 Tag=0
```

```
1308137: thread:0 rip:00000000fff0
1311009: thread:0 rip:00000000ffaa
1313417: thread:0 rip:000000008080
1315801: thread:0 rip:00000000c5fc
1320473: thread:0 rip:00000000c602
1321275: thread:0 rip:00000000c604
1323593: thread:0 rip:00000000c609
1323597: thread:0 rip:00000000c60e
```



```
+[id=37092] -
+[id=37093] -
-[id=37094] - [CCI cpuReq] -  CoreId:0 Tag:00 Addr[47:0]:000   fff0 Cmd:09(RdDword) MaskCnt:ff IO:1 Apic:0
    (time=13103.81) tn_platform.ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
              103.67) tn_platform.ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
              103.51) tn_platfo  .ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
              103.47) tn_platfo   ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
              103.47) tn_platfo   .emu_cpu_cci_monitor0.nbSysResp.emu_output_transactor_bcu: : [nbSysResp] - Resp[0x2] Tag
              082.71) tn_platfo   mu_cpu_cci_monitor0.cpuReq.emu_output_transactor_bcu: : [cpuReq] - Addr[0xfffffff0] Ap
                            ion Header Packet Cmd=0x14 RdSized    NPC Addr[63:0]=0x000000ffffffc0  SeqID[3:0]=0x
                             ON_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA0
                             N_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA0
                             _MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA0
                             ION_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA0
                             ONION_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : [RdResponse R] - Heade
              083.34) tn_platfo  U_ONION_MONITOR00.OnOutbXctorStreamSlice0.emu_output_transactor_bcu: : [RdSized NPC] -
              [Onion NOP          - Onion Header Packet Cmd=0x00 NOP      info Addr[63:0]=0x0000000000000  SeqID[3:0]=0x
```

```
13080     13090     13100     13110     13120
0123456789012345678901234567890123456789012345678
NOP info
emu_gpr_T3
emu_gpr_T0
 emu_gpr_T0
 RipLogT0
  cpuReq                nbSysRespDat
                        nbSysRespDat  <--- (13103.81)
                        nbSysRespDat  <--- (13103.67)
                        nbSysRespDat  <--- (13103.51)
                        nbSysRespDat  <--- (13103.47)
                        nbSysResp  <--- (13103.47)
  cpuReq  <— (13082.71)
   RdSized NPC         RdResponse R
                         ^---Data Packet (13103.2)
                         ^---Data Packet (13103.04)
                        ^---Data Packet (13102.88)
                        ^---Data Packet (13102.72)
                        RdResponse R  <---Header Packet
   RdSized NPC  <---Header Packet Cmd=0x (13083.34)
     NOP info
          outb_tlp
          outb_tlp
```

Table View:

	Id	TrkId	Time	Unit	Severity	Verbosity	Type	Symbol	Message
30	37156	37096	13085.76	tn_platform.EN...				[NOP info]	Header Packet Cmd=0x00 NOP info - Addr[63:0][0...
24	37155	37095	13083.34	tn_platform.EN...				[RdSized NPC]	Header Packet Cmd=0x14 RdSized NPC - Addr[63:0...
18	37154	37094	13082.71	tn_platform.EN...				[cpuReq]	Addr[0xfffffff0] Apic[0x0] Can[0x0] Cmd[0x9] C...
17	37153	37093	13081.33	tn_platform.EN...				[RipLogT0]	rip[0xfff0] thread[0x0]
16	37152	37092	13081.23	tn_platform.EN...				[emu_gpr_T0]	T0[0xfff0]
15	37151	37091	13080.23	tn_platform.EN				[emu_gpr_T0]	T0[0xfee00100]

Filter: | Syntax: Regular expression | Column: --ALL-- | ☐ Case sensitive filter | ☐ Case sensitive sorting

Selected Tracker: | Table View:

Search: | Previous | Next | Fields: Entire Tracker | Syntax: Regexp v2 | Case Sensitive | Expand | Deep

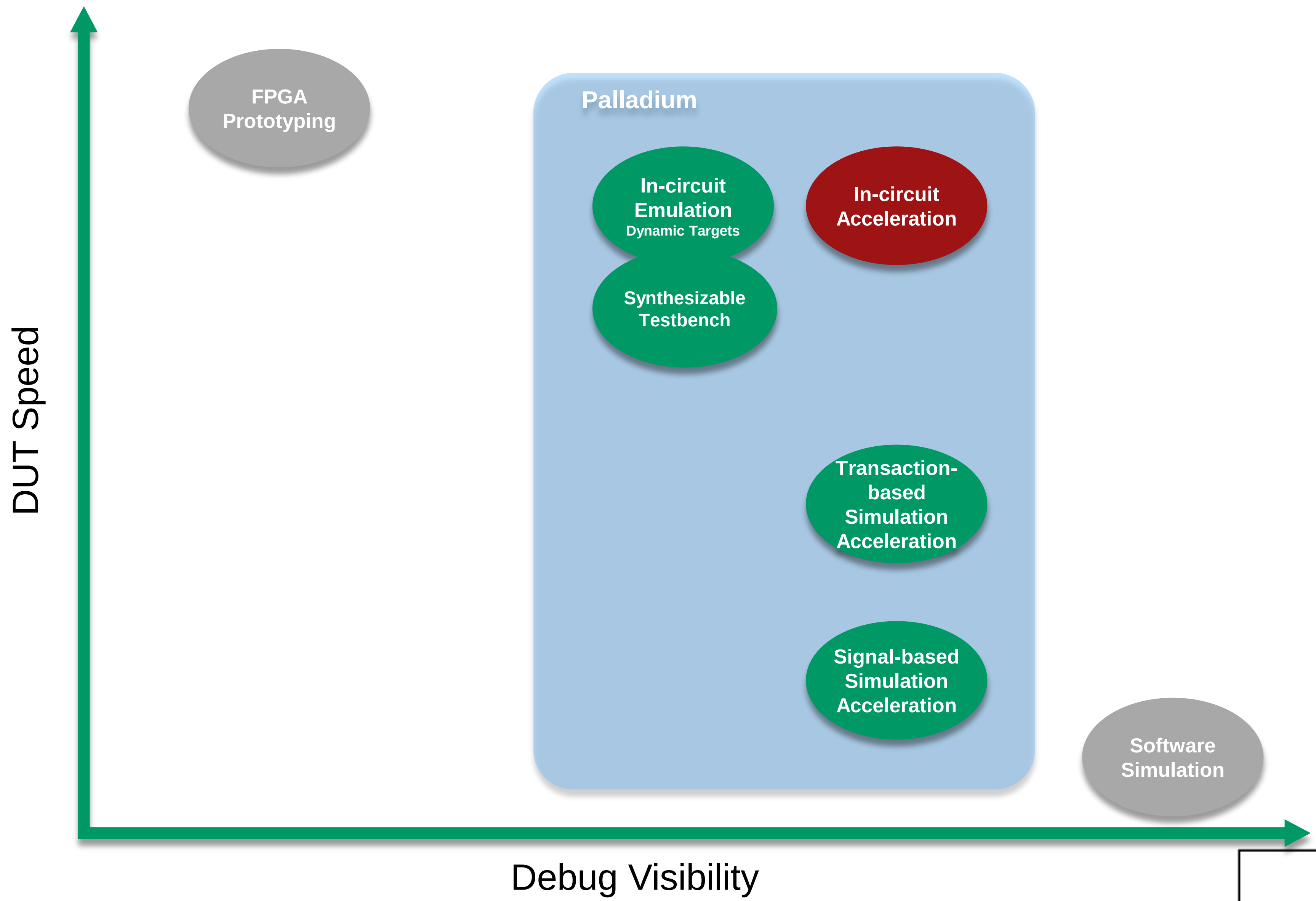
```
-[id=37095] - [Onion RdSized    NPC] - Onion Header Packet Cmd=0x14 RdSized    NPC Addr[63:0]=0x000000ffffffc0  SeqID[3:0]=0x0  srcTag[4:0]=0x1c  MskC    Last Time:1308271  Scale: 100:1
```

Linked transactions

Searchable

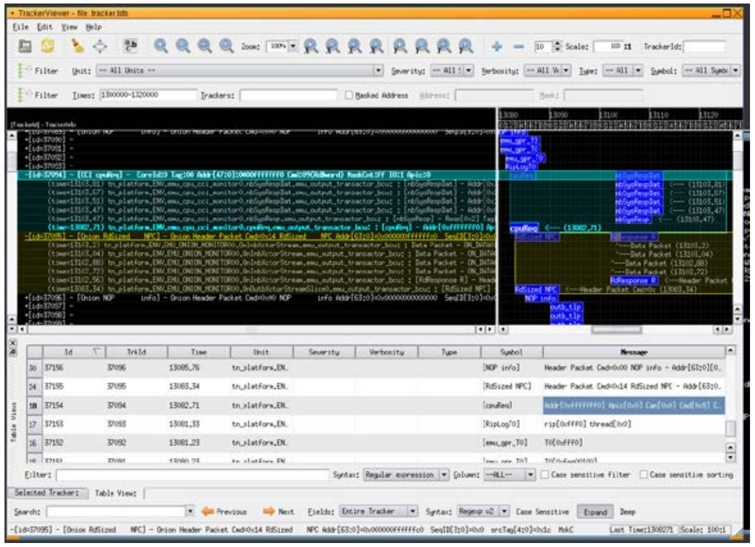
Viewer enables an improved method of interpreting data

INTRODUCTION OF IN-CIRCUIT ACCELERATION (ICA) MODE



AMD

NEW DEBUG METHODOLOGY



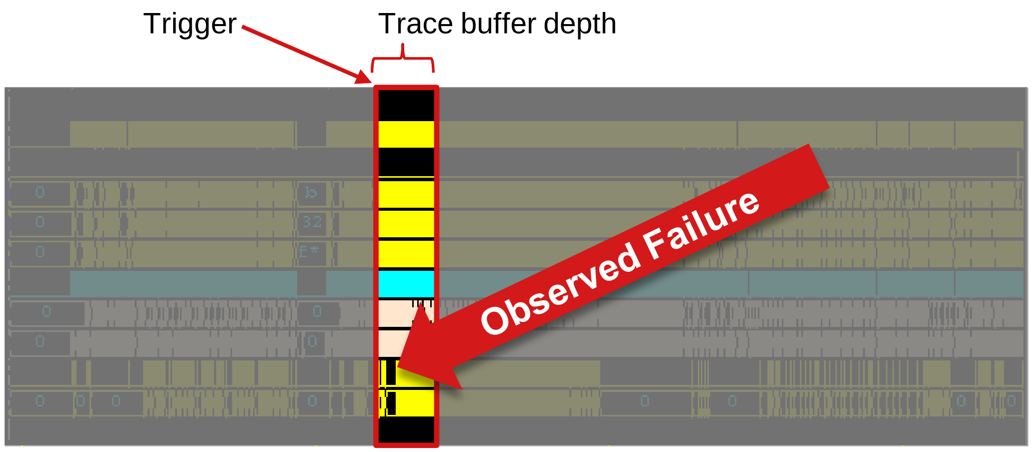
```
-[id=37094] - [CCI cpuReq] - CoreId:0 Tag:00 Addr[47:0]:0000FFFFFFF0 Cmd:09(RdDword) MaskCnt:FF IO:1 Apic:0
    (time=13103.81) tn_platform.ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
    (time=13103.67) tn_platform.ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
    (time=13103.51) tn_platform.ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
    (time=13103.47) tn_platform.ENV.emu_cpu_cci_monitor0.nbSysRespDat.emu_output_transactor_bcu: : [nbSysRespDat] - Addr[0x
    (time=13103.47) tn_platform.ENV.emu_cpu_cci_monitor0.nbSysResp.emu_output_transactor_bcu: : [nbSysResp] - Resp[0x2] Tag
    (time=13082.71) tn_platform.ENV.emu_cpu_cci_monitor0.cpuReq.emu_output_transactor_bcu: : [cpuReq] - Addr[0xffffffff0] Ap
-[id=37095] - [Onion RdSized   NPC] - Onion Header Packet Cmd=0x14 RdSized   NPC Addr[63:0]=0x000000ffffff0  SeqID[3:0]=0x
    (time=13103.2) tn_platform.ENV.EMU_ONION_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA
    (time=13103.04) tn_platform.ENV.EMU_ONION_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA
    (time=13102.88) tn_platform.ENV.EMU_ONION_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA
    (time=13102.72) tn_platform.ENV.EMU_ONION_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : Data Packet - ON_DATA
    (time=13102.56) tn_platform.ENV.EMU_ONION_MONITOR00.OnInbXctorStream.emu_output_transactor_bcu: : [RdResponse R] - Head
    (time=13083.34) tn_platform.ENV.EMU_ONION_MONITOR00.OnOutbXctorStreamSlice0.emu_output_transactor_bcu: : [RdSized NPC]
+[id=37096] - [Onion NOP      info] - Onion Header Packet Cmd=0x00 NOP      info Addr[63:0]=0x0000000000000  SeqID[3:0]=0x
```

	Id	TrkId	Time	Unit	Severity	Verbosity	Type	Symbol	Message
30	37156	37096	13085.76	tn_platform.EN.				[NOP info]	Header Packet Cmd=0x00 NOP info - Addr[63:0][0.
24	37155	37095	13083.34	tn_platform.EN.				[RdSized NPC]	Header Packet Cmd=0x14 RdSized NPC - Addr[63:0.
18	37154	37094	13082.71	tn_platform.EN.				[cpuReq]	Addr[0xffffffff0] Apic[0x0] Can[0x0] Cmd[0x0] C.
17	37153	37093	13081.33	tn_platform.EN.				[RipLog'0]	rip[0xfff0] thread[0x0]
16	37152	37092	13081.23	tn_platform.EN.				[emu_gpr_T0]	T0[0xfff0]

```
-[id=37095] - [Onion RdSized   NPC] - Onion Header Packet Cmd=0x14 RdSized   NPC Addr[63:0]=0x000000ffffffc0  SeqID[3:0]=0x0  srcTag[4:0]=0x1c  MskC    Last Time:1308271  Scale: 100:1
```

Use more abstract, user-digestible transaction data to pinpoint exact waveform trigger.

Sometimes able to debug problem without any waves, especially if a workload problem (rather than design).



More efficiently catch failure point in the narrow waveform trace window.

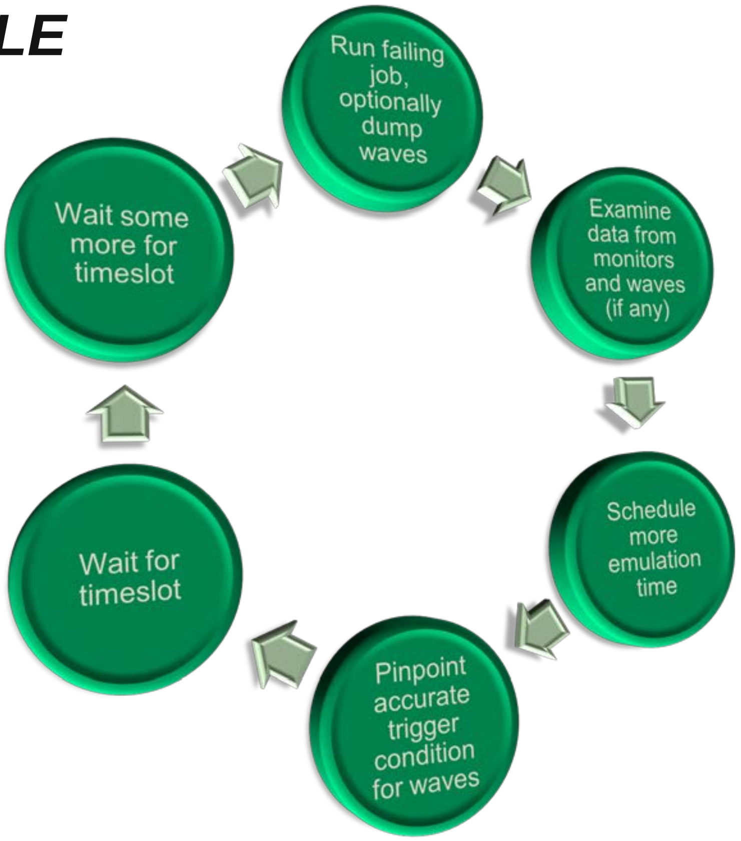
First-time wave dump success!

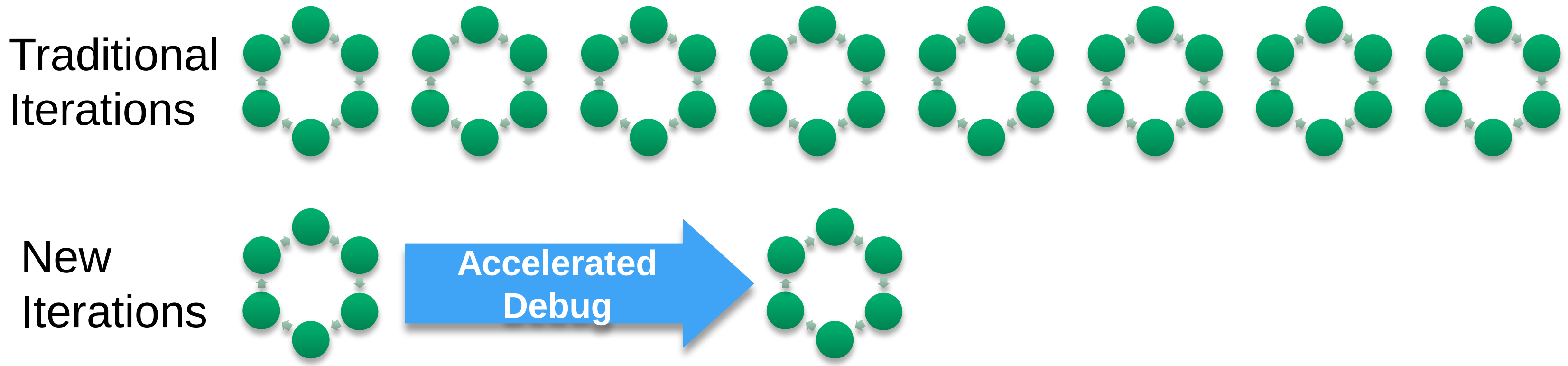
SCORECARD

Requirement	QEL monitor	Streaming transactional monitor
Need to abstract data to a level relevant for end-user debug	✓	✓
Need solution to function with dynamic targets Design clock cannot stop	✓	✓
Need the ability to identify the location for a waveform dump	✓	✓
Should not impact the emulation capacity requirements of designs significantly	✓	✓
Should maintain existing in-circuit emulation speeds	✓	✓
Save and restore continues to function as normal	✗	✓
Capture all design nodes of significance needed for system-level debug	✗	✓
We want to see all data associated with element being monitored Lossless capture, no missed events	✗	✓
Architected such that it is portable across emulation platforms with minimal modification	✗	✓
Easy automation of user actions based on monitored data	✓	✗
No rebuild of databases required when adding monitors	✓	✗

THE NEW EMULATION DEBUG CYCLE

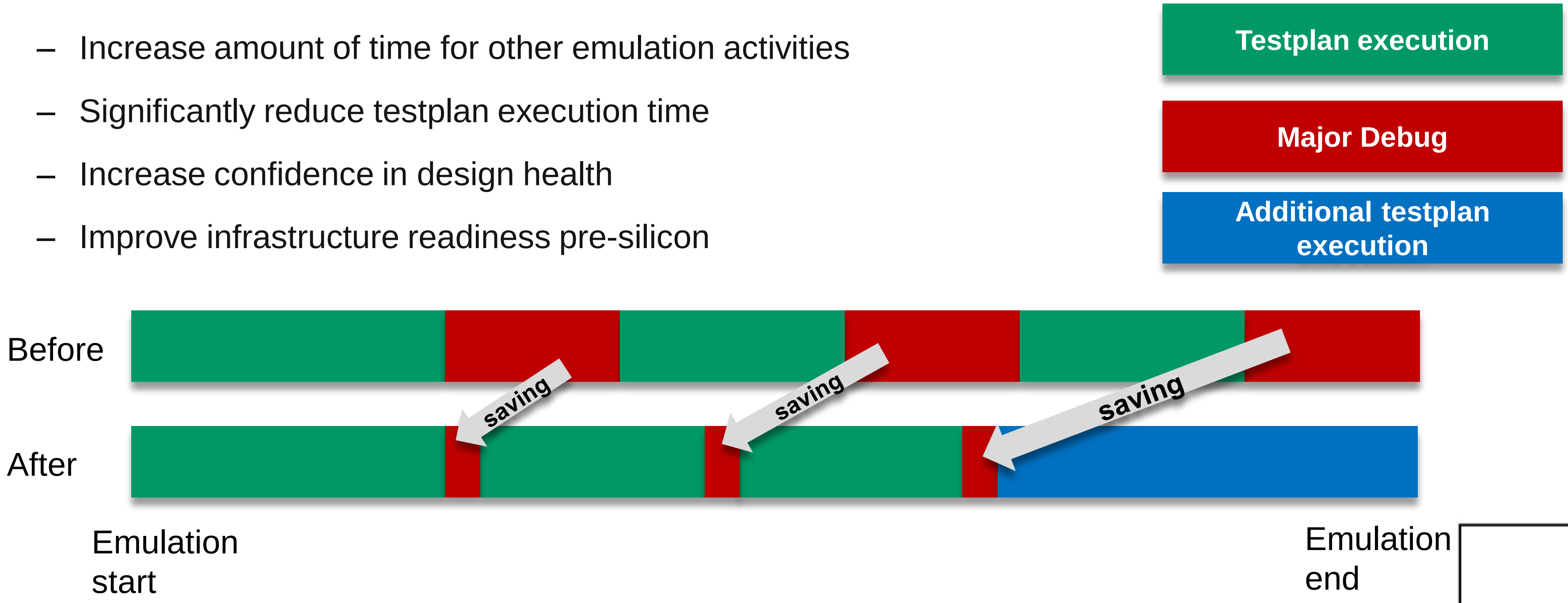
- Single debug iteration becomes more efficient: Debuggers have more data to go on at improved abstraction level.
- Able to jump to failure area with much higher degree of accuracy – usually in one iteration.
- Large overall reduction in number of debug iterations.


Run failing job, optionally dump waves
Examine data from monitors and waves (if any)
Schedule more emulation time
Pinpoint accurate trigger condition for waves
Wait for timeslot
Wait some more for timeslot



RESULTS

- Internal customer testimonials
  - "An issue we were debugging would have taken weeks to root cause without VLA."
  - "Just wanted to highlight another success story with VLA. Debug that would've been drawn out took under two hours to narrow down and form a theory, which took us to root cause in another hour. And all we used was the GPR monitor!!"
- Typically see multiple "difficult to debug", critical path issues per project.
- Users are requesting this enhanced debug capability for all emulation builds.
- Benefits:
  - Increase amount of time for other emulation activities
  - Significantly reduce testplan execution time
  - Increase confidence in design health
  - Improve infrastructure readiness pre-silicon



FUTURE DIRECTION

- Roadmap of monitors:
  - New designs have new interfaces that we plan to add into the system.

- Automatic analysis of monitor data streams:
  - Interface to complex checkers/monitors that cannot be synthesized into emulators.
  - Re-use of verification infrastructure, while optimizing for performance.

- Apply techniques to other areas:
  - Performance and power event monitoring.
  - Design throughput measurements.