IP to SoC hardware/software integration and verification flows—Cadence methodology and focus

Applications (Basic to Angry Birds)
Middleware (Graphics, Audio)
OS and Drivers (Linux, Android)
Bare-Metal SW
SoC
Sub-System
IP

Metric-Driven Verification Planning and Management

TLM Design and Verification

Formal Applications

Formal-Assisted Debug
Covered Closure
Connectivity
Sequential Equiv.
Security Path
Register Verification
Power-Aware Verification
Functional Verification
Design Bring-Up

Gate-Level Verification

Software-Driven Verification

SoC Hardware/Software Integration Verification and Architecture Analysis

Coverage-Closure-Driven Verification

UVM e/SV Coverage-Driven Verification

Spec Silicon
IP to SoC pre-silicon verification platforms
A system-centric look at a modern SoC

- Many IPs
  - Standard I/O
    - Wifi, USB, PCI Express® (PCIe®), etc.
  - System infrastructure
    - Interconnect, interrupt control, uart, timer…
  - Differentiators
    - custom accelerators, modem…
- Many cores
  - Both symmetric and asymmetric
  - Both homogeneous and heterogeneous
- Lots of software
  - Part of core functionality
    - Communication stack, DSP software, GPU microcode…
  - User application software infrastructure
    - Android, Linux…
SoC-level verification and validation requirements

- How to communicate/share use cases between users
- How to create and reuse use cases from IP to SoC
- How to use C code to execute natively on many cores and communicate between cores
- How to run use cases across platforms and run more constrained random variants on faster platforms
The solution: Perspec System Verifier

Use-Case Reuse

Vertical Reuse

Scope (Integration)
- Middleware (Graphics, Audio, etc.)
- OS and Drivers
- Bare-Metal Software
- SoC (Hardware + Software)
- Sub-System
- IP

User
- Architect
- Hardware Developer
- Software Developer
- Verification Engineer
- Software Test Engineer
- Post-Silicon Validation Engineer

Abstract Model with Reusable Use Cases

Powerful Solvers

Software
- c test
- c test
- c test
- c test

Scheduling, inter-processor communication, runtime randomization

Perspec™ System Verifier

Platform
- Virtual Platform
- Simulation
- Emulation
- FPGA Prototype
- Silicon Board

Horizontal Reuse
Automated use case verification

Desired Scenario:
Decode video from the DDR and show on the display

Perspec™ System Verifier automatically and exhaustively completes the goals into full legal scenarios

Perspec™ solver checks the feasibility of the goals statically

Coverage model is auto-created and pruned for reachable scenarios

Re-generate the code for derivatives, spec changes, etc…

SLN Models

Abstract Tests

Target C test
**Example use case**
Translating end-user use case to system-level bare-metal actions

**End-user use case:**
Mobile phone requirement:
view a video while uploading it (6 words)

**System-level bare-metal actions:**
Capture a video with camera using graphics processor and save it to a memory buffer in DDR0 in AVI format with medium resolution and MPEG3 audio with 4x3 aspect ratio then transmit the video using the modem and processor 3 while processor 2 shows the video on the built in display being streamed by the graphics processor of the video already saved in DDR0 memory buffer (66 words)
Use-case verification flow with Perspec engine

- SLN Model: Resources, actions, C code templates
- Use Case Composer: Perspec™ Engine
- Perspec Library
- Virtual Platform
- Simulation Platform
- Hardware Emulation Platform
- FPGA Silicon Platform
- Debuggers (C/Design debuggers, etc.)
- Abstract Debug Environment
Step #1: Capture topology and system actions
Step #2: Capture abstract use case >> solve for concrete use case(s) >> analyze gentime coverage

Desired Scenario:
Decode video from the DDR and show on the display

Abstract Use Case

Concrete Use Case

Gentime coverage of use case
Step #3: Generate tests for specific platform(s)
Step #4: Run tests and debug

- Debug from UML activity diagram synchronized with source, waveform and log messages
Perspec Modeling
Modeling elements

- **Component**: Functional unit groups actions and resources

- **Action**: Abstract operation of function

- **Token**: Include information for pre-conditions and outcomes

- **Place**: Defines interaction of tokens and actions (memory, channel, lock)

- **Extend**: Extending functionality of actions, components, and tokens

```plaintext
component config_timer_c {
    timer_setup : memory of timer_t;
    state_fsm : memory of fsm_state_t;
    action timer_interrupt_a { ...; }
};

action timer_interrupt_a {
    timer : to timer_setup;
    next : to state_fsm;
};

type name_t in [A0,A1,A2,A3,B0];
token timer_t {
    name : name_t;
    mode : [stop, Up, Continuous, UpDown];
    interrupt : [ENABLE,DISABLE];
    channel : uint [0 .. 7];
};
extend DVE {
    child mem : memory[1] of power_mode_t;
    child tim : memory[*] of timer_t;
    bind config_timer.timer_setup == tim;
};
extend timer_interrupt_a {
    constraint next.state == enter_lpm;
};
```
Modeling with Perspec System Verifier

System language notation (SLN)
Constraints

extend timer_interrupt_a {
    constraint next.state == enter_lpm;
    constraint timer.mode in [Up, UpDown];
    constraint timer.compare in [0 .. 0xFF];
    constraint timer.channel == 0 ? 
        (timer.ccie_interrupt == ENABLE && timer.interrupt == DISABLE):
        (timer.ccie_interrupt == DISABLE && timer.interrupt == ENABLE);

    table {
        #name | #type | #number | #channel | #wakeup_source;
        A0   | A     | 0       | 5        | TA0     ;
        A1   | A     | 1       | 3        | TA1     ;
        A2   | A     | 2       | 2        | TA2     ;
        A3   | A     | 3       | 5        | TA3     ;
        B0   | B     | 0       | 7        | TB0     ;
    } with {
        constraint timer.name == <#name> => timer.type == <#type> &&
            timer.number == <#number> &&
            timer.channel <= <#channel> &&
            wakeup.wakeup_source == <#wakeup_source>;
    };
}
C-code generation

Input: a code template is always connected to an action.

```c
extend timer_interrupt_a {
  exec body C#:
  _enable_interrupts();
  Timer_start<(timer.mode)>Mode ( 
    TIMER_<((timer.name)>)_BASE,
    TASSEL_<((timer.clock_source)>,
    TIMER_CLOCKSOURCE_DIVIDER_1,
    <(hex((timer.compare))>>, 
    TIMER_TAIE_INTERRUPT_<((timer.interrupt)>,
    TIMER_CCIE_CCR0_INTERRUPT_<((timer.ccie_interrupt)>,
    TIMER_SKIP_CLEAR
  );
  _enable_interrupts();
  Timer_startUpDownMode ( 
    TIMER_B0_BASE, TASSEL_ACLK,
    TIMER_CLOCKSOURCE_DIVIDER_1, 0x29,
    TIMER_TAIE_INTERRUPT_DISABLE,
    TIMER_CCIE_CCR0_INTERRUPT_ENABLE,
    TIMER_SKIP_CLEAR
  );
};
```
Code generation

```c
/* User declaration code */
/* exec declaration: inline code for component DVE [2] : at line 3 in @body */
#include <stdio.h>
#include <test_support.h>

volatile uint16_t int_count;

/* User definition code */
/* exec definition: inline code for action i0:timer_interrupt_a 40 : at line 15 in @body */
#pragma vector=TIMER0_B0_VECTOR
_interrupt void TIMER0_B0_ISR(void)
{
    TQ_success();
}

int_count=1;
Timer_stop [TIMER_B0_BASE ];
__low_power_mode_off_on_exit();

/* Code for scenario: exit_lpcm_a id: 51 processor: */
/* static void sur_exit_lpcm_a_scenario_start_51(void) {
*/
/* Code for i0:timer_interrupt_a 40 declared at line 10 in base_timer */
//_enable_interrupts();

TQ_startUpDownMode (TIMER_B0_BASE,TASSEL_ACU_TIMER_CLOCKSOURCE_DIVIDER_1.0x77,TIMER_TAIE_INTERRUPT_DISABLE,TIMER_CC1E_CC1R0_INTERRUPT_ENABLE,TIMER_SKIP_CLEAR);

TQ_disableCaptureCompareInterrupt (TIMER_B0_BASE,TIMER_CAPTURECOMPARE_REGISTER_2);

/* Code for i1:enter_lpcm_a 44 declared at line 10 in base_lpcm */
int_count=0;
__low_power_mode_1();
if (int_count == 1) {
    TQ_success();
} else {
    TQ_error();
}
```
Public Success Stories
ST TRD: Verify GPU modified for SoC power management

- Higher coverage in less time than manual tests development
  - All 192 generated tests are different and cover all states
  - Covering transition we did not think off
  - Estimated manual effort to reach same coverage: 192 days

<table>
<thead>
<tr>
<th></th>
<th>Nb tests</th>
<th>Lines of code</th>
<th>Development</th>
<th>Maintenance</th>
<th>Nb tests /Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>20</td>
<td>2k (100x20)</td>
<td>20 days</td>
<td>3 to 4 days</td>
<td>1</td>
</tr>
<tr>
<td>Perspec</td>
<td>192</td>
<td>800</td>
<td>10 days</td>
<td>1 day</td>
<td>19.2</td>
</tr>
<tr>
<td>Ratio</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td>19.2</td>
</tr>
</tbody>
</table>

- Simply debug and analyze of failing tests

We deliver:
- More power
- More complex tests
- Specifications

Creating a test:
- New bug types
- Missing isolation
- Control sequence
- Retention scheme
- Memory corruption
- Power sequencing
- Software/Hardware
- Power On Reset
- ....

We have created:
- A test benefit
- IP Power
- IP top states
- Other power states
- An IP API
- Retention
- Clock, ports

Need for automation to create tests for all possible states
### Motivation

- **Complexity:**
  - SoCs become more and more complex with time.
  - Verification can consume up to 80% of the design time.
  - Mainly verification involves multiple parties and tools.
  - To handle this on a typical SoC project a very powerful tool is needed.

- **Structured MSP430 verification flow**
  - *Feature-driven* verification eplan based
  - Several thousand test case written directly
  - Prone to faults in the planning phase
  - Too high manual effort

- **Motivation for a new tool Perspec System Verifier**
  - Simple system model to prove the concept
  - Seamless integration into our existing environment
  - Generation of pipe cleaner test cases (auto generated)
  - Comparison to our today’s existing approach

### Flow comparisons

<table>
<thead>
<tr>
<th>METRICS</th>
<th>Perspec</th>
<th>MSP430 Verification Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>time effort (units)</td>
<td>modeling / update script adoption / update reuse</td>
<td>modeling / update script adoption / update reuse</td>
</tr>
<tr>
<td></td>
<td>10 / 5 / 1</td>
<td>10 / 8 / 7</td>
</tr>
<tr>
<td>manpower</td>
<td>modeler and user</td>
<td>person write and update test cases</td>
</tr>
<tr>
<td>complexity</td>
<td>individual scenarios understandable</td>
<td>difficult</td>
</tr>
<tr>
<td>visibility</td>
<td>scenario understanding/ traceability</td>
<td>plan overview</td>
</tr>
<tr>
<td>readable</td>
<td>same style, one define multi use</td>
<td>depends on writer</td>
</tr>
<tr>
<td>editable</td>
<td>centralized update</td>
<td>change all affected files</td>
</tr>
<tr>
<td>portability</td>
<td>easy, replace one file</td>
<td>high manual effort</td>
</tr>
<tr>
<td>coverage</td>
<td>functional system + execution</td>
<td>execution coverage</td>
</tr>
<tr>
<td>debug</td>
<td>differentiated debugging</td>
<td>traditional debugging</td>
</tr>
<tr>
<td>number</td>
<td>shall be limited by coverage</td>
<td>every test cases written manually</td>
</tr>
</tbody>
</table>
ST CPD: Complex SoC scenarios

Consumer market is pushing towards:
- More IP integrations
- Support most advanced technologies
- High density

Verification methods:
- Bare metal, with all dependencies
- Target tests reuse
  - From IP to SoC
  - Over various SoCs
  - Over CPU model
  - From SoC to SoC
  - A test could require 1000 lines

IO Coherence verification is one of the hot topics addressed in our recent SoCs

Such verification methods:
- Need to configure
  - Configuration of IO
  - Configuration of CPU
- Need to synchronize
  - Constraints on C code
  - Synchronization between operations
- Need to track memory

3 simultaneous memcpy actions

Manually writing and debugging such tests is challenging

Conclusion

- PCIe model
  - 1000 lines: 1/3 model, 2/3 template
  - 4 wk, mostly reverse engineering

- CPU model
  - 1d to fill CSV and provide configuration details

Achievements for PCIe:
- Coverage on test generation
- Easier maintenance
- Multi instances tests

IO Coherence with PCIe:
- Self checking tests generated, with parallel data flows involving multiple CPUs and PCIe instances

Next Opportunities:
- Promote vertical reuse and get IP provider delivering Perspec model
- Build derivative system tests at SOC level, combining with other IPs
- Standardizing through Accellera Portable Stimulus WG
Summary
Connecting it together

Coherency use case

User1

User3

Power shutdown

User2

Power shutdown use case

1. Cache transactions

2. Power down

3. Cache transactions

4. Power up

5. Cache transactions
Perspec System Verifier

- **Productivity**
  10X improvement for complex SoC test creation

- **Abstraction**
  UML-style use-case diagrams

- **Automation**
  System use-case test generation

- **Portability**
  Reuse across all execution platforms

- **Measurement**
  SoC-level hardware/software coverage metrics
Backup
Micro-kernel runtime environment

• Perspec™ System Verifier has ability to manage resources, parallel actions, and test scheduling
  – Before C code is created, automatic planning of the scenario takes place
  – In the C code, sync points are added between cores and testbenches VIP
  – Resources availability is also managed in runtime

• Perspec sync is done via an abstract mailbox
  – Modeled in layers to support multiple communication schemes, e.g., memory, sockets, GPIO, etc…
  – Thread safe to enable multiple cores and same-time communication
  – Small in size and efficient

• Some of the applications of this infrastructure
  – Sync of any activity across languages and platforms
  – Unified analysis and debug
  – Runtime coverage and checking
  – Control external VIP/component
  – Print messages from the embedded cores
Perspec Libraries
Perspec libraries

• Many SoCs have many common characteristics
  – Typically have CPUs, caches, memories, low-power features, etc…
  – Enables capturing a general set of SoC model building blocks

• Cadence provides pre-built libraries for Perspec™
  – Reduces modeling effort and time for customer
  – Models follow good coding style, built for reuse

• Perspec System Methodology Library (SML)
  – Captures system modeling including memories, processors, etc…
  – Customer uses spreadsheet template to configure for specific SoC

• Example: Perspec library for ARM® Architecture
  – Captures configurable cache, MMU, and low-power models
Firstly intra-cluster cache tests are needed to cross-cover MOESI states of L1 and L2 caches.

Coherency verification challenges
Intra-cluster
Coherency verification challenges

Intra-cluster

Firstly intra-cluster cache tests are needed to cross-cover MOESI states of L1 and L2 caches.

Now add inter-cluster cache tests to stress L2 through adjacent snoop traffic.
Coherency verification challenges

Critical coherent I/O

Firstly intra-cluster cache tests are needed to cross-cover MOESI states of L1 and L2 caches.

Now add inter-cluster cache tests to stress L2 through adjacent snoop traffic.

Need to create some software thread to create I/O coherency scenario.
Coherency verification challenges

Firstly intra-cluster cache tests are needed to cross-cover MOESI states of L1 and L2 caches.

Now add inter-cluster cache tests to stress L2 through adjacent snoop traffic.

Need to create some software thread to create I/O coherency scenario.

Extreme stress is now introduced with further threads on both clusters.
Now we need to test I/O coherency for all peripherals that generate sharable transactions.
Power shutoff verification challenges

System Control Processor controls clocks, power, and resets. To be confident the system is robust, we need to exercise all the range of legal power shutoff (PSO) scenarios and traffic that goes with them.

big.LITTLE with Dynamic Voltage Frequency Scaling (DVFS) creates potential hazards through the combinations of clock frequencies. Need to drive the SCP and coherent traffic to cover all the clock combinations.
Example: Library for ARM Architecture
Perspec library for ARM Architecture

• Set of operations to manage the ARM compute cluster
  – Memory management
    – Page table handling, virtual address
  – Predefined actions that user can use in their program
    – Write Data, Read data, Copy data
  – Caching operation
    – True Sharing
    – False Sharing
    – I/O Coherency
  – Low power

• Takes a description of the system in a spreadsheet and creates system scenarios
Perspec library processor configuration tables

• Zero modeling is required since CPU and memory sub-system models are automatically generated from library by reading system configuration tables (shown below)

• Memory blocks
• Processors, names, clusters, coherency
• Pages, virtual address (VA), physical address (PA), size
• Processors to memories accessibility/restrictions

Tables can be extended to add more design specific attributes
Perspec library use model

Step #1: Fill in the standard SML and Coherency tables

Step #2: Load the SML bring-up environment + user boot code

Step #3: Use the composer to write scenarios and use-cases: Memory, coherency, power, and DVM, tune the built-in coverage as needed

Step #4: As needed model SoC-specific subsystems (e.g., PCIe controller), data movers (e.g., DMAs), user defined power states, etc...

User model

SVR bringup env

Step #5: Use the composer to write tests that combine built-in scenarios with user-defined scenarios and actions

Running tests

No modeling effort
Pre-defined basic software operations

- **Basic software operations**
  - **Write:** `write_data`
    - Generates data and writes it into the memory
  - **Copy:** `copy_data`
    - Copy data from one area to another
  - **Read:** `read_check_data`
    - Read data from previously written area
    - Checks against the reference model

- **Main control knobs**
  - Alignment
  - Data size
  - Memory block/address
Advanced software operation: All processors to all memories
Coherency—False Sharing

Processors run in parallel

Address offset

Processors split cache lines

Cache lines
Coherency – True Sharing Scenarios

Processors run in parallel

And synchronize

Address Offset

Processors access same address at different times

Cache Lines
Debug
Perspec debug capabilities

- Abstract debug using UML activity diagram
- Smart log filtering, stepping, and searching
- Lock-step execution of activity diagram, log, C code, and waveform