A methodology for SoC top-level validation

Electronic Design Process ’02
Sylvan Dissoubray
Monterey

Corporate Profile

- Founded in Nov. 1999
- 15+ M€ raised in 2000 and 2001
- 100 employees in 5 countries
- 30 consultants
- 30+ large corporate customers
- 200+ seats deployed

Major shareholders
- Management & staff
- GALILEO Partners
- CDC Innovation
- INNOTECH
- Advanced Capital Europe
- Intel Capital
- THALES Corporate Ventures
- INRIA-Transfert
Esterel Studio Applications

- System Architecture Specifications
  - Wireless core platforms
  - Consumer electronics core platforms
  - Microprocessor chipsets

- Top-level Validation of SoCs
  - Wireless platforms
  - Set top boxes
  - DVD chipsets
  - MPEG decoders

- Telecom & Security Protocol Development (SW / UML)
  - UMTS: RRC, RLC layers
  - Bluetooth
  - Secured military communication
  - Smart-card security
  - HiperLAN2

SoC top-level validation challenges

- SoC’s maximized reuse but design costs did not drop down
  - Because cost moved to SoC’s functional verification
    consumes up to 70% overall design cycle budget (ITRS)

- Why is it so difficult to verify SoC’s?
  - Multi-core, multi-service, multi-traffic SoC’s => extreme concurrency
  - More reuse of blocks and IPs => less and less knowledge of components internal behavior

- The SoC integrator dilemma
  - Functional coverage
    requires a lot of test cases,
    it is difficult to reach corner cases
  - Time
    writing test cases is time consuming,
    running integration level tests is slow
SoC Top Level Validation: objectives

- Generate automatically a Test Suite to validate IP interoperability and concurrency
- High level transactional approach
- Obtain the best coverage in the most effective manner

SoC top-level validation method

1. Write Transactional Models (TM)
2. Integrate all TM in parallel
3. Automatic Generation
4. Run validation testbench
Example: top-level validation of a MP3 SoC

- The MP3 system
  - The DMA block can transfer
  - The WRITER can write into RAM
  - The DECODER can:
    - Copy the stream
    - Decode the stream into another format (wav, pcm, etc.)
- WRITER and DECODER are concurrent and synchronized: the WRITER is responsible to start the DECODER

1. Write Transactional Models (TM): the DECODER

- Internals of block behavior need not be described in Esterel Studio
- not Bus Cycle Accurate level, not Cycle Accurate level
2. Put All TM Models in Parallel

3. Automatically Generate TM coverage tests
Generated tests cover 100% states of the TM

---

TM Inputs sequences

---

Coverage Report

---

Collected Outputs

---

Script to C or other languages

---

C Test Case

```c
void run_test()
{
    transfer();
    write();
    wait_transfer_done();
    decode();
    wait_write_done();
    wait_decode_done();
}
```
From Transactional Level to Simulation Platform

C Top Level Validation Testbench

```c
void run_test() {
  transfer();
  write();
  wait_transfer_done();
  decode();
  wait_write_done();
  wait_decode_done();
}
```

Simulation Platform

TRANSACTIONS

Transactor Libraries

```c
void transfer() {
  *(int)DMA_CHANNEL_REG = (int) REG_INIT;
  initPort(0x00000001);
}
```

```c
void write() {
  ...
}
```

Transaction execution profile on a CPU or simulation platform

- The generated testbench cover interoperability and concurrent behaviors of SoC internal blocks

```plaintext
<table>
<thead>
<tr>
<th>transfer()</th>
<th>write()</th>
<th>wait_transfer_done()</th>
<th>decode()</th>
<th>wait_write_done()</th>
<th>wait_decode_done()</th>
</tr>
</thead>
</table>
```

The generated testbench cover interoperability and concurrent behaviors of SoC internal blocks.
Refinement 1: Adding constraints to direct test generation

- Fine Tune concurrency
- Take into account system constraint and restriction
- Match the test plan
- Use of existing libraries

```
StartX = transfer()

EndX = wait_transfer_done()

StartY = write()

EndY = wait_write_done()
```

Productivity Gain

- Schedules
  - 2 weeks
  - 3 weeks
  - 2 weeks
  - ~1200 tests

Using ES

- Specification understanding
- Transactional Model and constraints
- Test generation and integration

Hand written

- Specification understanding
- Tests written by hand

Generated Test Cases

<table>
<thead>
<tr>
<th>Files</th>
<th>Configuration</th>
<th>Model</th>
<th>Generated Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pre</td>
<td>wait</td>
<td>valid</td>
</tr>
<tr>
<td>dec7_conf1_seqdc_S_out</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dec7_conf2_seqdc_S_out</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Generated test cases are compact and efficient**

<table>
<thead>
<tr>
<th>Test generation strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esterel Studio</td>
</tr>
<tr>
<td>Random or Directed Random</td>
</tr>
<tr>
<td>Manual coding</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>% Coverage (state and transition)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 %</td>
</tr>
<tr>
<td>? %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test case size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production effort (available)</td>
</tr>
</tbody>
</table>

**Top Level Validation Coverage**

- Productivity Gain: 40 %
- Quality gain in corner cases
- Test case size
- Production effort (available)

**Esterel Studio SoC top-level validation: benefits summary**

- Applicable to actual CPUs as well as any simulation platform
- Requires minimal blocks and IPs documentation
- The generated testbench is extremely efficient to cover concurrent behaviors of SoC internal blocks and interoperability
- Applicable to multi-core designs
- Generated testbench shows excellent coverage vs. size efficiency and is good at reaching corner cases
Esterel Studio

```
DWORD WINAPI Thread_A(LPVOID lpstart){
    for (;;){
        if (R) { R = 0; ExitThread(1); }
        if (A) { A = 0; ExitThread(0); }
    }
}
DWORD WINAPI Thread_B(LPVOID lpstart){
    for (;;){
        if (R) { R = 0; ExitThread(1); }
        if (B) { B = 0; ExitThread(0); }
    }
}

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow) {
    DWORD x=0;
    DWORD ExitCode[2];
    HINST = hInstance;
    hthread[2] = CreateThread(NULL,0,
        (LPTHREAD_START_ROUTINE)
            ControlPanel,
        (LPVOID) &x,0,&dwThreadId[2]);
    for (;;){
        hthread[0] = CreateThread
            (NULL,0,(LPTHREAD_START_ROUTINE)
                Thread_A,
            (LPVOID) &x,0,&dwThreadId[0]);
        hthread[1] = CreateThread
            (NULL,0,(LPTHREAD_START_ROUTINE)
                Thread_B,
            (LPVOID) &x,0,&dwThreadId[1]);
        WaitForMultipleObjects(2,hthread,TRUE,INFINITE);
        GetExitCodeThread(hthread[0], &ExitCode[0]);
        GetExitCodeThread(hthread[1], &ExitCode[1]);
        if (!ExitCode[0] && !ExitCode[1]) {
            O();
            for (;;) if (R) break; }
    } }
```

Esterel style

```
loop [ await A || await B ];
emit O
each R
```

State Notion

```
1
Cancel/
wait data or address
Cancel/

2
Data/

3
Address and Data/Write
Cancel/

4
Address/Write
Cancel/

5
Data/Write
```

State Notion
A methodology for SoC top-level validation

Electronic Design Process ’02
Sylvan Dissoubray
Monterey