

# Cadence and Specman

ACP Summer School, June 2011

Reuven Naveh

Cadence

([rnaveh@cadence.com](mailto:rnaveh@cadence.com))

# Cadence

---

- **One of three major EDA companies**
- **Established in 1988, over 4000 employees**
- **Wide variety of areas and products in hardware design**
  - Virtuoso, Encounter, Allegro...
- **We will focus on functional verification**
  - Incisive platform
  - Mainly „Incisive Enterprise Specman Elite“ (AKA „Specman“)

# Functional Verification

## ■ Why so important?

- Prevents much costly bugs at the post-silicon level
- Takes more than half of the effort of hardware projects

## ■ Formal verification

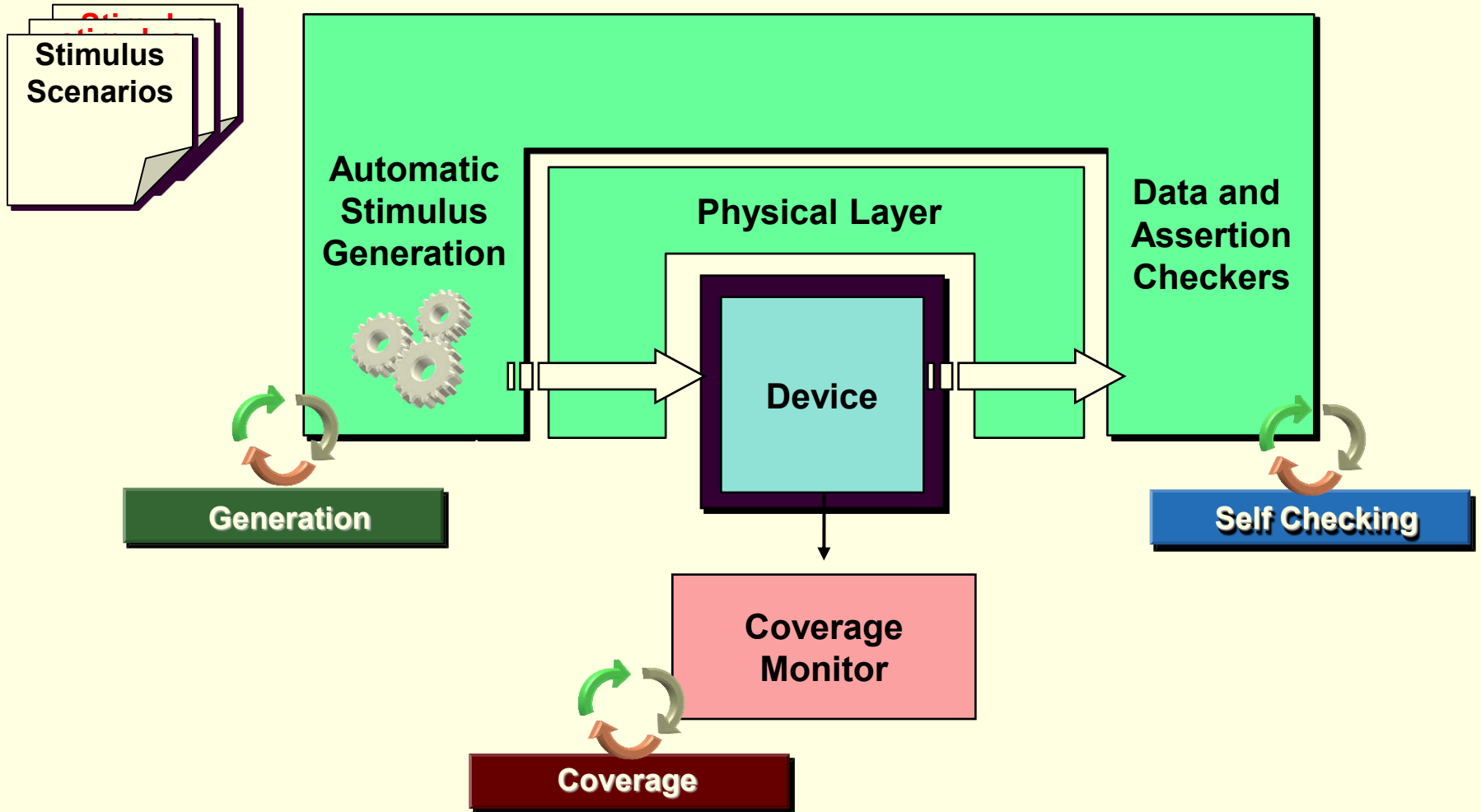
- Attempts to prove the correctness of a given specification
- Solvers are used to find a solution
- Cadence tool – IFV (Incisive Formal Verifier)

## ■ Simulation based verification

- Tests the hardware through simulation
- Constraints are used to create legitimate random stimuli
- Verification languages: SystemVerilog (IEEE 1800), e (IEEE 1647)

# Coverage Driven Verification Environment

## Verification Environment



# Specman

---

- **Cadence's major test bench automation tool**
- **Developed by Verisity in the nineties**
- **Being used in the biggest and most advanced verification environments**
- **Supports all aspects of coverage-driven-verification**
- **Can be attached to any simulator**
- **Uses e verification language**

# *e* Language (some examples)

## ■ “AOP”

```
//environment code
struct data {
  len:uint;
  kind:[SMALL,BIG];

  keep kind == SMALL
    => len in [10..20];

  keep kind == BIG
    => len in [50..70];
};
```

```
//test code
extend data {
  keep kind == SMALL;
};
```

# ■ Predicate Classes (Chambers 1993)

```
type packet_protocol: [Ethernet, IEEE];
struct packet {
  protocol: packet_protocol;
  data: list of byte;

  show() is {
    out("Packet length is ",
      data.size(). " bytes.");
  };
};

extend Ethernet packet {
  header: Ethernet header;
  show() is first {
    out("I am an Ethernet packet.");
  };
};
```

```
extend sys {
  test_packets: list of packet;

  keep test_packets.size() == 1000;
  keep test_packets.count(
    it.protocol == Ethernet) in [100..200];
};
```

- **See: Hollander, Y., Morley, M., Noy, A.: The e Language: A Fresh Separation of Concerns. In TOOLS (38)(2001) 41-50**

# Constraints in *e*

## ■ Declarative entities, struct members

- *keep my\_field < 100;*
- *keep for each in my\_list {it.field1 < it.field2}*

## ■ Can be:

- Hard or soft
- Reset or overridden

## ■ Apply to:

- Scalar and non-scalar fields
  - Lists, pointers
- > constraints determine the structure and not just the generated values!

## ■ Used for:

- Random stimuli generation
- Environment configuration
- Pointer binding



# Constraints in $e$ (examples)

## ■ Arithmetic and logic constraints

- *keep soft  $f() \Rightarrow x + y == z$ , keep  $a > 0$  and  $x \% a != 0$*

## ■ Global constraints

- *keep  $my\_list.sum(it.x) == 100$ , keep  $my\_list.all\_different(it)$*

## ■ Bit constraints

- *'keep  $addr[1:0] == 0$ ', 'keep soft  $num == 1 \ll size$ '*

## ■ Distribution constraints (always soft)

- *keep soft  $x == \{80:[1..100];10:[500..5000];10:others\}$*

# CSP Challenges

---

- **Problems are not (always) hard**
  - Extensive search is usually not required
  - Backtracks typically indicate bad modeling
- **Scalability of simple problems**
  - Solving a single problem many times
  - Solving many different problems
  - Problems are influenced by the state of the environment
- **Bit level mixed with word level**
- **Distribution!**
  - Random solution and not just a solution
  - It is hard to define and achieve the „optimal“ distribution
  - Random stability is required

# Solutions

---

## ■ BDD solver

- Bit level, predictable (“uniform”) distribution, very fast
- Capacity problems

## ■ SAT solver

- Bit level (loses high level dependencies), bad distribution
- Translation to CNF is expensive

## ■ Finite Domain Solver

- Word level
- Non-uniform and hard-to-control distribution
- Cheap on easy problems

# Specman solver (IntelliGen)

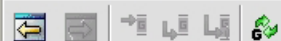
---

- **Collects and analyzes all the environment's constraints**
  - The environment may be huge (tens of thousands of variables and constraints).
- **Partitions to separate solving problems**
  - Each problem is relatively small
  - Orders the partitions properly and automatically according to dependencies
- **Creates matching solving technology for each partition**

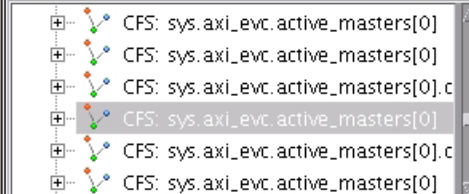
# Gen Debugger

---

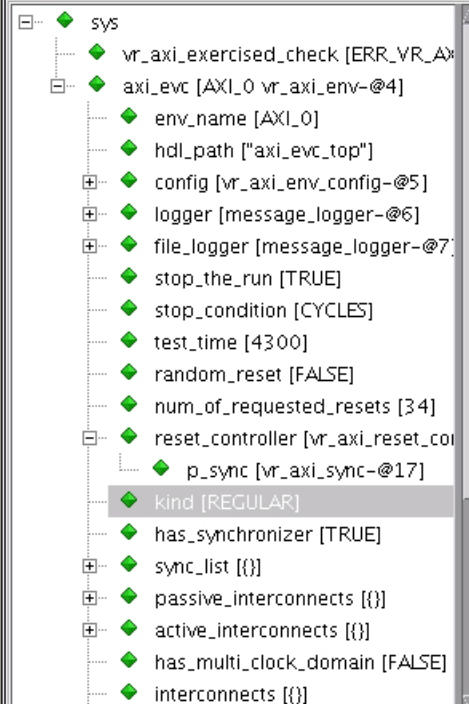
- **Interactive GUI tool to debug constraints**
- **Information is presented in user terms**
  - Each problem details the relevant constraints and variables
  - Reductions show “constraints” and not “propagators”
  - Includes range information and bit information
- **Works in online and offline modes**
  - Rich set of breakpoints
  - Step-by-step debugging and debugging in retrospect
  - Easy navigation between solving steps



## Generation Process Tree



## Generated Tree



## Connected Field Set #2242 : sys.axi\_evc.active\_masters[0]

## Variables

	Type	Name	Initial	Current
<input type="checkbox"/>	Input	value(module_or_port)		MODULE
<input type="checkbox"/>	Input	config.active_passive		ACTIVE
<input type="checkbox"/>	Input	config.interface.env_name		AXI_0
◆	Gen	config.interface.write_da...	[0..42949672...]	5
◆	Gen	config.module_or_port	[MODULE.POR...]	MODULE
◆	Gen	config.ACTIVE'ordering_...	[NORMAL_OR...]	NORMAL_O...

## Constraints

```

keep config.module_or_port == value(module_or_port) at line
keep soft ordering_algorithm == NORMAL_ORDER at line 392 in @
keep ordering_algorithm != DATA_PHASES_END_TIMES at line 393
keep module_or_port != PORT => ordering_algorithm != INTERC
keep interface.write_data_interleaving_depth == 1 => ordering_algc
keep module_or_port == PORT and interface.write_data_interlea
keep write_data_interleaving_depth >= 1 at line 461 in @vr_ax
keep soft write_data_interleaving_depth == 1 at line 462 in @vr_ax
keep write_data_interleaving_depth == 5 at line 118 in @vr_ax

```

## Variable config.module\_or\_port: vr\_axi\_module\_or\_port\_t

General Info    Source    Past Steps    Constraints

Source File: vr\_axi\_config.e

```

178 -- Base unit for all agents configs.
179 -----
180 unit vr_axi_agent_config like vr_axi_config {
181     -- Active or passive agent
182     active_passive: erm_active_passive_t;
183
184     -- Module or port, one of:
185     -- MODULE - the agent is a real one.
186     -- PORT - the agent is connected to an interconnect port.
187     module_or_port: vr_axi_module_or_port_t;
188
189     -- Name of the containing agent
190     name: vr_axi_intf_name_t;
191
192     keep gen (name) before (interface);
193
194     -- Interface unit, containing common fields to the master and slave connected
195     -- through it.
196

```