Cadence and Specman

ACP Summer School, June 2011

Reuven Naveh

Cadence

(rnaveh@cadence.com)

cadence®

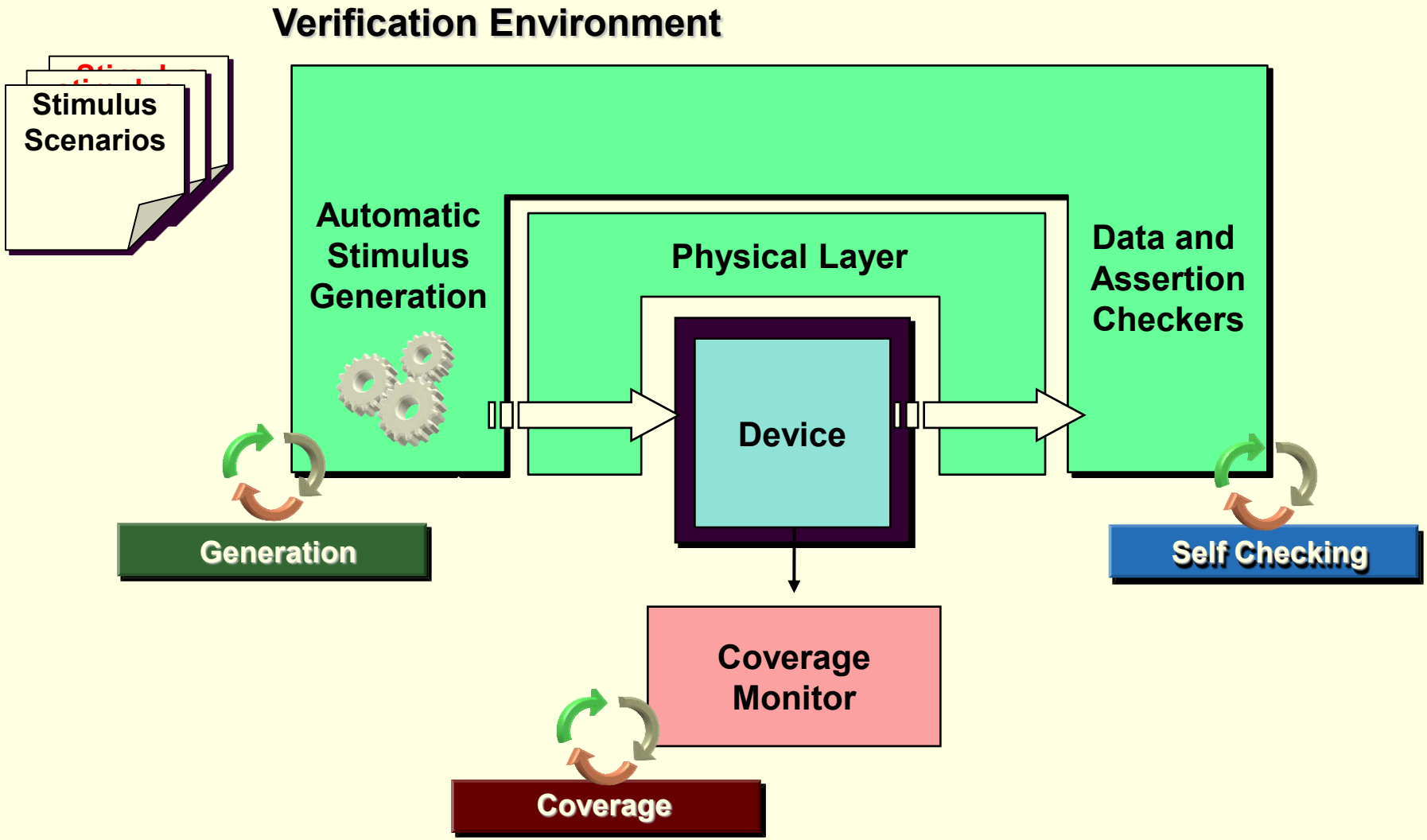
Cadence

- **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



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

cadence®

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 overriden
- **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

cadence®

Constraints in *e* (examples)

- **Arithmetic and logic constraints**
  - *keep soft f() => 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<<size'*
- **Distribution constraints (always soft)**
  - *keep soft x == {80:[1..100];10:[500..5000];10:others}*

cadence®

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 problems 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