PARADIGM™ WORKS

Development of a PCI Express Coverage Monitor eVC

Verisity ClubSpring 2003

Stephen D’Onofrio, Ning Guo

Verification Alliance

PARADIGM™ WORKS

Verification Alliance

Overview

- Why develop a PCI Express Functional Coverage Monitor eVC
- Verisity expedites the development process
- Functional Coverage Metrics of the Monitor eVC
- Bootstrapping eVC development
- Metrics
- Conclusion

Why develop a Functional Coverage Monitor eVC

- Smallest reusable subset and reasonable starting point of the eVC
- Easily integrated into verification environments
- Employ the functional coverage features to enhance the verification process
- Springboard to further development of reusable verification components

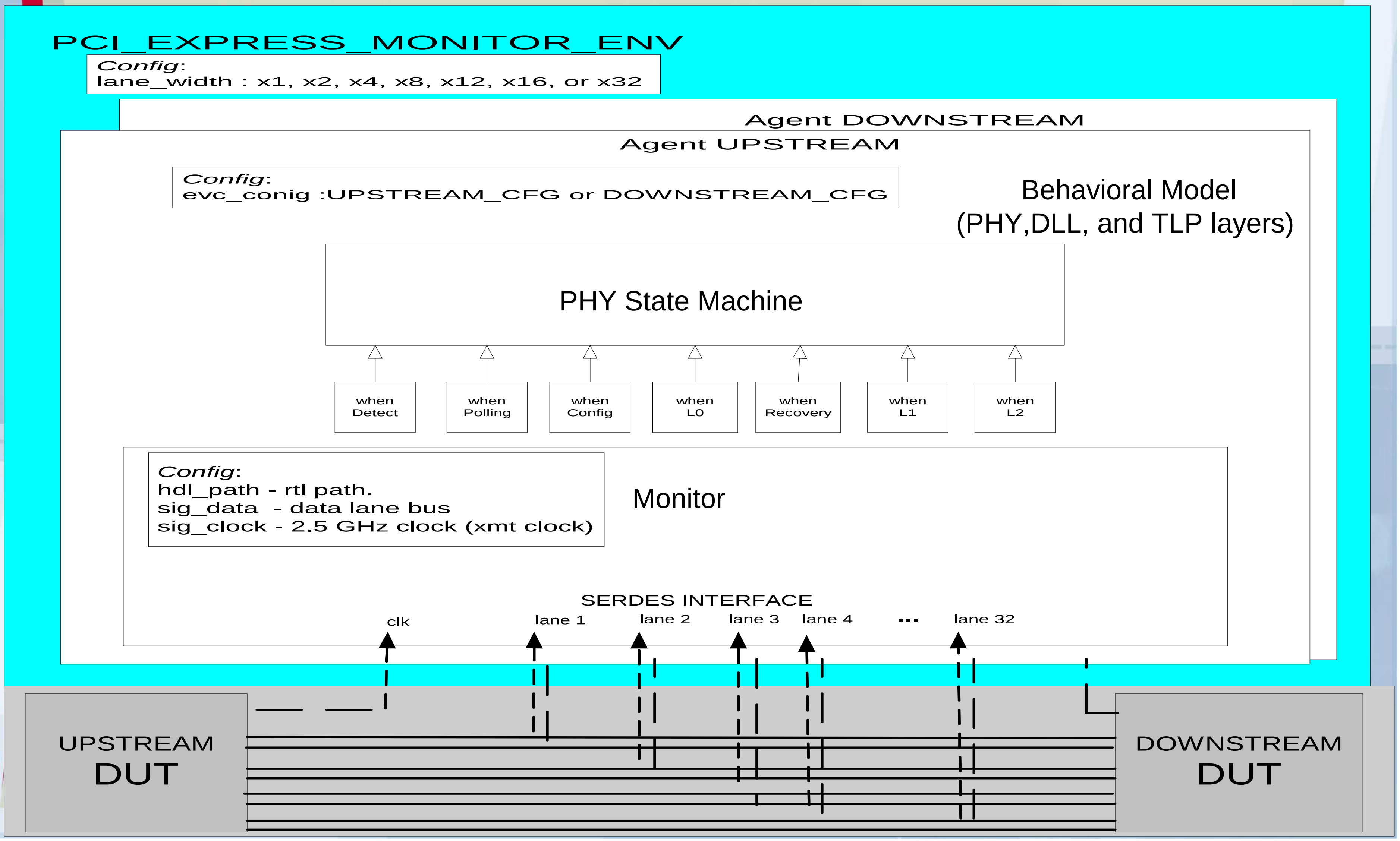
Verisity expedites the development process

- Verification Vault
  - Methodology
  - Tools
- eRM refers to “e Reuse Methodology”
  - Easy Start – packaging
  - Well organized directory/file structure
  - Modularity of the code
  - Messaging/Logging features
  - eRM compliance

Strategy

- Goals:
  - Keep it simple
  - Extensible
  - Reusable
- Process:
  - Define the objects/aspects:
    - From the specification *nouns* become candidates for structures/units or aspects of a structure/unit
    - Verbs often translate into operations – methods within the structure/units
  - Analysis:
    - Map your problem into eRM components
    - Describe operations to be performed by the units/structures
    - Determine what attributes are common to all structures/units
  - Relationship:
    - Determine how the aspets/structures/units will interact

eVC Architecture



Aspect oriented coding techniques

- **File Partitioning**
  - Create files for objects, aspect of objects, types, constants, packets
  - All files are to be imported from one eVC file (called evc_top.e – part of eRM)
  - Avoid import cyclical dependencies
    - Import files using a bottom-top abstraction approach

    i.e. type.e, env.e, agent.e, monitor_receiver.e, …

Aspect oriented Coding

FILE: agent.e

```
unit pw_pci_exp_mon_agent_u  {
        name : string;
        //  back-pointer
        env : pw_pci_exp_mon_env_u;
        … agent code  …
}
extend pw_pci_exp_mon_env_u  {
        agent_names : list of pw_pci_exp_mon_agent_name_t;
        keep agent_names == {UPSTREAM; DOWNSTREAM};
        // agent instantiaition
        agents : list of pw_pci_exp_mon_agent_u is instance;
        keep gen (agent_names) before (agents);
        keep agents.size() == agent_names.size();
        keep for each in agents {
                it.name == agent_names[index].as_a(string);
                // here is the back-pointer assignment
                it.env == me; or get_enclosing_unit(pw_pci_exp_mon_env_u);
}
```

Polymorphism In e

- An object can dynamically change shape at run-time

FILE parent.e

```
type child_object_t: [];

unit parent {
  child_object: child_object_t;
  show() is undefined;
    …
};
```

FILE child_a.e

```
extend child_object_t: [CHILD_A];
extend  CHILD_A parent {
        show() {out("I AM A")};
};
```

FILE child_b.e

```
extend child_object_t: [CHILD_B];
extend  CHILD_B parent {
        show() {out("I AM B")};
};
```

Polymorphism (cont)

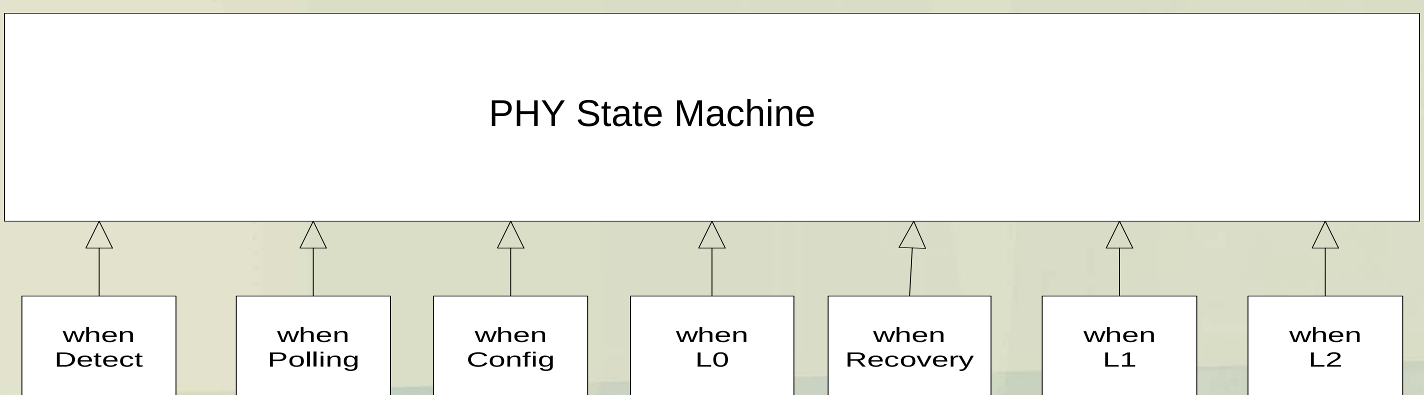
- Dynamic binding

```
FILE top.e
        var my_top : parent is instance;
         my_top.child_object = CHILD_A;
         my_top.show();
         my_top.child_object = CHILD_B;
         my_top.show();
};
```

```
OUTPUT
        I am A
        I am B
```

Polymorphism (cont)

- We implemented the PHY State Machine using polymorphism
- The "PHY State Machine" is the parent and states are the children
- Virtual methods where used for initialization, setting timeouts, processing …



Package Data Hiding

- Version 4.1 adds package data hiding capabilities via the private, protected, and package access modifiers
  - Implicit functional coverage item cannot be private
    - Use package modifier instead
  - Imported packages cannot be inherited!
    - i.e. could NOT use access restrictions in our common package

Functional Coverage Concepts

- Functional Coverage Elements
  - Groups – A set of items which are updated by the same event
  - Basic Items - One or more coverage signals and/or struct fields which represent a point-of-interest
  - Basic Buckets –represent a single value or a range within an item (for integers and enumerators)
- Hit – indicates that an item or bucket coverage definition was met
- Grading – hit/goal – quality of functional coverage
- Hole – indicates that a coverage goal was not met
- Test Ranking
- Extended Functional Coverage Capabilities
  - Transitional Functional Coverage - Item/Bucket changing from one value to another
  - Cross Functional Coverage – Two or more items

Functional Coverage Concepts (cont.)

- The PCI Express Monitor eVC is limited to “Black Box” Functional coverage
  - Visibility into eVC ports to the DUT only
  - Looks at eVC internals too
  - Does NOT look at DUT internals
- Switch functional coverage on and off
- Using functional coverage result
  - Non-reactive
  - reactive
- Automate coverage hole detection

Functional Coverage Implementation

- Place coverage code in a separate file
- Allow users to “extend” functional coverage
- Extend banner
  - Output functional coverage settings and other configuration information
- Could not use the “using per instance” option for transitional coverage definitions
  - Use a macro definition as an alternative

```
define <NAME'struct_member>“NAME [<str'exp>]”as { …}
```

- Writing transitional coverage
  - Use the “using ignore not” option to setup valid states This means “everything is illegal except …”

Coverage Metrics Example

- Functional point-of-interest

PCI Express Specification states that SKP packets can follow TS packets

- Derive coverage items from functional point-of-interest

```
cover rcv_found_upstream_packet_coverage is {
    item pkt_kind;
    transition pkt_kind using ignore =
    not (prev_pkt_kind == TS and pkt_kind == SKP)
};
```

PARADIGM WORKS

Verification Alliance

Bootstrapping Monitor eVC development

- Three tier approach
  - Developed a transactor eVC package
  - Used C++ BFM core from Intel Developer Forum
    - See John Morris' presentation on C++ interface at www.paradigm-works.com
  - Test the monitor at a Beta site
- Developed a regression suite
- Fault insertion
  - Prove that the checkers work
- Functional Coverage Verification
  - Prove that the coverage definitions work

Beta Site

- Applied Monitor eVC into a pre-existing verification environment
  - Monitor eVC package includes examples (part of eRM)
  - Easy integration process
  - Configuration makes eVC robust
- Verification enhancements achieved by the monitor eVC
  - detect coverage holes
    - modify stimulus to fill coverage holes
  - helps verify the functionalities
- Helped reveal bugs in the monitor

Metrics

- Functional coverage
  - 24 functional coverage group
  - 230 functional coverage definitions
- Project Milestones
  - Architecture/Spec/scheduling – 2 weeks
  - Monitor – 2 ½ months
  - BFM Integration – 1 month
  - Xactor – 2 months
  - Regression Environment (tests & scripts) – 1 month
- Profiling Results from Xactor Test (rough estimates)
  - Coverage on/Coverage off =>1.0x to1.25x
  - Interactive/Compiled =>1.2x to 2.85x

Questions/Issues

- Naming space
- Preferred method of turning on/off coverage
- Common packages and data hiding
- C interface issues
  - Access C++ enumerated data type

Conclusion

- eRM augmented the verification design process
- Monitor uncovered flaws in a third party BFM
- Monitor revealed functional coverage holes at beta site
- Next Step – code optimization, new techniques, etc.

The End

Contact Information:

www.paradigm-works.com

Paradigm Works

1 Corporate Drive

Andover, MA 01810

stephen.donofrio@paradigm-works.com

ning.guo@paradigm-works.com