

CBV ROADMAP



MOTOROLA

<http://advtools.sps.mot.com/afv/>

DESIGN VERIFICATION

CBV Roadmap/John Havlicek 2/19/02

1

Advantages of Original CBV

- **Programming language “look and feel”**
- **No negation, no disjunction above expression level**
 - Close correspondence between CBV text and forall automaton
 - No automaton determinization or complementation required
- **Constraints restricted to boolean expressions**
 - Allows cycle-by-cycle constrained random stimulus generation
- **Intuitive semantics in terms of leaf-level checks:**
 - Assertions and eventualities of boolean expressions, activated under temporal preconditions
- **Recursive tasks**
 - Powerful automaton-building construct
 - Convenient for coding ongoing monitors

We want to preserve Original CBV, revised, as a clearly-defined subset of the CBV language extensions.



Tiered Extensions

To meet Accellera Requirements, we propose extending CBV in increasing tiers:

- **Core CBV, a.k.a. revised Original CBV**
- **Core CBV with Regular Expressions (CRE CBV)**
 - Adds regular expressions as preconditions and assertions
- **Extended CBV**
 - Satisfies all Accellera requirements
- **Extended CBV with Automata (EA CBV)**
 - Adds automaton acceptance conditions for alternating automata



Core CBV

- **Adds VHDL expression layer bindings (R1e)**
 - Verilog or VHDL mode specified at (cbv)module definition
- **Adds enumeration data types (R7b)**
- **Adds general clock expressions (R34a)**
 - Provide before- and after-edge versions of posedge, negedge, anyedge
 - Allow reference to the next tick (i.e., next point in time at the finest granularity)
- **Adds macro definition and expansion features (R46a)**
 - Perl- or C-style preprocessing; syntax not yet fixed
- **Adds conditional instances of recursive tasks**
- **Cleans up some untidiness**

No automaton determinization or complementation is needed to implement Core CBV.



Core CBV with Regular Expressions

- Adds regular expressions as conditions and assertions

For example,

if (a)

if +(1) : (~a)

if +(0 to *) : (~b)

if +(1) : (b)

c ;

becomes

if (a, ~a & ~b, (~b)*, b) c ;

- **Syntax not fixed yet.**
 - Comma or semicolon for concatenation
 - Other operators not fixed
- **Assertion of regular expressions is strong**



Extended CBV

- **Adds negation operator for statements (R27a, R65a)**
 - “not” keyword
- **Extends the eventually operator to arbitrary statements (R25a)**
- **Adds explicit duals of most constructs**
 - Dual of “if” is “observe”
 - Dual of “+(n):” is “[n]:” in absence of dual keyword
 - Dual of “align” is “s_align”
 - Dual of “sync” is “s_sync”
 - “begin-end” is “begin_and-end”, with dual “begin_or-end”
 - Dual of “task” is “f_task”, finitely recursive task
- **Adds LTL until operators (R25a)**
 - “until”, “s_until”
- **Adds an assume directive (R71f)**
 - Applicable to top-level statements and entire (cbv)modules
- **Adds a case assume directive (R71b)**
 - Applicable to top-level statements and entire (cbv)modules



Extended CBV

- “S until T” is equivalent to a call to the task

```
task A();  
begin_or  
  T  
  begin_and  
    S  
    +(1) : A() ;  
  end  
end  
endtask
```

[Cf. nu Z . T or (S and X Z)]



Extended CBV

- “S s_until T” is equivalent to a call to the task

```
f_task B() ;  
begin_and  
  S  
  begin_or  
  T  
  +[1] : B() ;  
end  
end  
endtask
```

[Cf. mu Z . S and (T or X Z)]



Extended CBV with Automata

- “f_task” construct has finitely recursive acceptance condition
- “begin_or-end” and “begin_and-end” allow (f_)tasks to build alternating automaton transition structure
- Missing acceptance condition specification

- Automaton construct:

```
automaton <name> ( <formal_parameter_list> ) ;  
    <acceptance_condition>  
    <statement>  
endautomaton
```

- Acceptance condition syntax not fixed yet.
 - Specify sets of automaton states using statement labels and (f_)task names
 - Indicate how the sets of states are to be used in forming the acceptance condition (finitely often, infinitely often, etc.)



ENDROADMAP



MOTOROLA

<http://advtools.sps.mot.com/afv/>

DESIGN VERIFICATION