

# CBV PROPOSAL



**MOTOROLA**

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

DESIGN VERIFICATION

CBV Proposal/John Havlicek 3/20/02

1

# Roadmap Review

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

- **Core CBV, a.k.a. revised Original CBV**
  - Includes extensions consistent with Original CBV
  - Tidies up
- **Core CBV with Regular Expressions (CRE CBV)**
  - Adds regular expressions
  - More powerful matching tasks
- **Extended CBV (ECBV)**
  - Satisfies all Accellera requirements and many additional requests
- **Extended CBV with Automata (EA CBV)**
  - Adds automaton acceptance conditions for alternating automata



# Core CBV Status

- VHDL expression subset sketched, not in parser yet (R1e).
- Enumerated types are in the parser (R7b).

- **Example:**

```
type fruit =  
  enum  
    apple,  
    banana,  
    cranberry  
  end
```

- “nat” type for natural number constant expressions is in the parser.

- **Example:**

```
task foo(m : nat, n : nat);  
  +(m to n) : bar();  
endtask
```

- Compound type constructors are sketched, some are in the parser.

- **Example:**

```
type pair =  
  record  
    x : bool;  
    y : byte;  
  end  
assign p : pair = pair(.y(b_byte),.x(a_bool)) ;  
always p != pair(1'b0,8'hff);
```



# Core CBV Status (cont.)

- Name binding of parameters is in the parser.
- “random(<type>)”, “random()” for free variables are in the parser.

- **Example:**

```
assign q_dynamic : pair = random(pair) ;
local q_static[0:7] = random();
```

- “group-endgroup” for nested (cbv)module contexts with defaults is in the parser.

- **Example:**

```
module foo ;

    default always;
    default sample @(a_posedge(clk));

    if (a)
        +(1) : b;
    b != x;

    group bar ;
        default sample @(a_posedge(clk2));
        (!c || y);
        initially +(0 to 5) : c != d;
    endgroup

endmodule
```



# Core CBV Status (cont.)

- **General clock expressions are in the parser (R34a).**

- **Example:**

```
assign my_clk : bool = posedge(clk) && enabled;  
sample @(my_clk)  
  if (a)  
    +(1) : b ;
```

- **Macro expansion constructs (R46a) are still to be set.**
- **“always” and strong nexttime, align, and sync added, since they can already be expressed in Core CBV.**
- **“f\_task” and task lights added to Core CBV give it full omega-regular expressive power.**
  - **Any co-Buchi universal (i.e., forall) FA has a direct representation in Core CBV with linear complexity.**



# CRE CBV Status

- Matching Semantics document is submitted (R55a,R56a).
- Regular expression syntax is in the parser.
- Regular expression examples (Properties <= 18) are submitted.

- **Example:**

```
if (snoop & hitm)
  +(1) : if (first_match [1{*};trans_start])
    writeback ;
```

- “regexp” type is in the parser.

- **Example:**

```
function cbv_next_event(e : bool) : regexp ;
  return [first_match [1{*};e]] ;
endfunction
```

- “fail”, “first\_match”, “first\_fail”, “word\_complement” are in the parser.
- Other requested derived regexp operators still under consideration.



# CRE CBV Status (cont.)

- “m\_task” and extended matching statements are in the parser.

- **Example:**

```
m_task find(x : bool);
begin_or
  if (x) match;
  if +(0 to *) : (!x)
    if +(1) : (x)
      match;
end
endtask
```



# ECBV Status

- ECBV Statement Semantics document is submitted (R55a,R56a).
- Proof of full omega-regular expressiveness is done (R27a,R71c,d).
  - Coding a co-Buchi AFA in ECBV is straightforward.
- Static sufficient conditions for equivalence of Buchi and co-Buchi acceptance have been given.
- “dual” directive for specifying Buchi acceptance rather than co-Buchi is in the parser.
- “always”, “eventually”, “until”, “s\_until” are in the parser. They can be applied to any statement (R25a).
- “assume”, “case\_assume” are in the parser (R71a,b,f).

- **Example:**

```
assume
  always
    if (request)
      #(1) : eventually grant;
```



# ECBV Status (cont.)

- “begin\_and-end”, “begin\_or-end” are in the parser.
- “f\_task” and task lights are in the parser.

- **Example:**

```
f_task another_strong_until();
begin_or
  T();
begin_and
  S();
  #(1) : another_strong_until();
end
end
endtask
```

- “not”, “complement” are in the parser (R27a,R65a).
- Dual forms “observe”, “s\_align”, “s\_sync”, “#(n):”, etc. are in the parser (R35b).
- Branching semantics could be added easily with quantified nexttime operators “+A(n):”, “+E(n):”.
  - Branching semantics is not a requirement. We envision adding branching semantics to the standard in the future.



# EA CBV Status

- Tasks with lights facilitate direct encoding of any Buchi or co-Buchi AFA.
- Adding the “automaton” construct is low-priority now; it is not needed to meet the requirements.
  - Would be nice for a revision of the standard.



# CBV Advantage: Tiered Extensions

- **Implementation of Core CBV requires no automaton complementation or determinization.**
  - Any co-Buchi universal FA has a straightforward representation in Core CBV.
  - Any Core CBV module can be compiled efficiently into a co-Buchi universal FA. The compiled form can be represented textually in the language, providing portability.
  - Model checking algorithms for co-Buchi universal FAs are well-known. The complementary language is that of the dual Buchi existential FA.
  - Implementation of Core CBV provides an early product entry point for a vendor.
- **CRE CBV adds the convenience of regular expressions and more powerful matching tasks.**
  - Determinization algorithms for finite word automata are needed to implement CRE CBV.
  - Such algorithms are well-known and can be derived from the subset construction for finite word automata.



# CBV Advantage: Tiered Extensions (cont.)

- **ECBV adds negation, completing the language.**
  - Any co-Buchi AFA has a straightforward representation in ECBV.
  - Any ECBV module with the default acceptance condition can be compiled efficiently into a co-Buchi AFA. The compiled form can be represented textually in the language, providing portability.
  - A large part of the ECBV Statement Semantics document is devoted to sketching the compilation algorithm from ECBV to an AFA.
  - Model checking algorithms for co-Buchi AFAs are well-known. The co-Buchi AFA can be uniformized to a co-Buchi universal FA with at most an exponential blowup (Muller & Schupp, 1995). The Core CBV model checking algorithm can then be applied.



# CBV Advantage: Look and Feel

- **CBV has the “look and feel” of a programming-language rather than a temporal logic.**

- **CBV:**

```
always
  if (snoop & hitm)
    +(1) : if (first_match [1{*};trans_start])
      writeback ;
```

- **SugarFL:**

```
{[*]; snoop & hitm} | => {!trans_start[*];trans_start & writeback}
```

- **CBV language constructs are carefully chosen to be readable and to convey intuition. Non-intuitive symbols are avoided.**
- **CBV promotes a vertical style of code, whose line numbers provide references for simulation code coverage and debugging.**
- **Compact formulas with powerful pre-defined operators hide much of the detail of what is being specified. They are harder to read, harder to debug, and harder to re-use or export as verification IP.**
  - **“I can write that program in one line . . .”**



# CBV Advantage: Flexibility

- **CBV provides flexible basic coding constructs:**
  - **functions**
  - **recursive tasks**
- **Using these constructs, users can define libraries that are appropriate to their needs.**
- **Macros will allow construction of customized temporal operators.**
- **LTL operators are also provided in ECBV for those who want to use them.**



# CBV Advantage: Accessibility

- **CBV is designed to be accessible to logic designers and verification engineers with undergraduate-level engineering education.**
  - Logic designers and verification engineers understand recursion from their programming language experience.
  - Logic designers and verification engineers generally do not have experience with LTL or CTL.
- **CBV has an operational semantics that is intuitive and easy to learn.**
  - An execution of a CBV statement is a computation of the associated AFA. In Core CBV, the AFA is a universal FA, and it is easy to teach the intuitive idea of the multithreading of the various paths in the computation tree.
- **CBV coding paradigms for ongoing, recursive monitors are easy to teach.**
  - At Motorola, logic designers have begun using Core CBV effectively with as little as a 2-hour introductory tutorial.



# CBV Advantage: Multiple Clock Syntax

- **CBV multiple clock syntax flows with the overall coding style, rather than appending clocking directives at the ends of formulas.**

- **CBV:**

```
sync @(a_posedge(clka))
  always
    if (req_in)
      sync @(a_posedge(clkb))
        +(1) : req_out ;
```

- **SugarFL:**

```
(always (req_in -> (next req_out) @ rose(clkb))) @ rose(clka)
```

- **CBV provides before-edge clock constructs.**



# Summary

- **CBV has undergone a substantial revision to meet all the Accellera requirements.**
- **We are working hard to put all the details into place in order for the committee to make an informed decision.**
- **We believe we have the better language because:**
  - **ECBV absorbs SugarFL.**
  - **CBV has tiered extensions, providing multiple implementation points for a vendor and crisp boundaries for users and implementors.**
  - **CBV provides recursive tasks. Additional predefined operators can always be added, but adding access to a fundamental construct such as recursive tasks is not so easy.**
  - **CBV has an intuitive operational semantics corresponding to computations of an AFA.**
  - **CBV has the better look and feel, as well as accessibility.**
  - **ECBV does not have branching semantics now, but adding quantified nexttimes is straightforward for a future revision.**





**MOTOROLA**

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

DESIGN VERIFICATION



**MOTOROLA**

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

DESIGN VERIFICATION

# ENDROADMAP



**MOTOROLA**

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

DESIGN VERIFICATION

CBV Proposal/John Havlicek 3/20/02 20