Using macros to mimic VHDL in ACL2

Dominique Borrione & Philippe Georgelin
TIMA, Grenoble, France

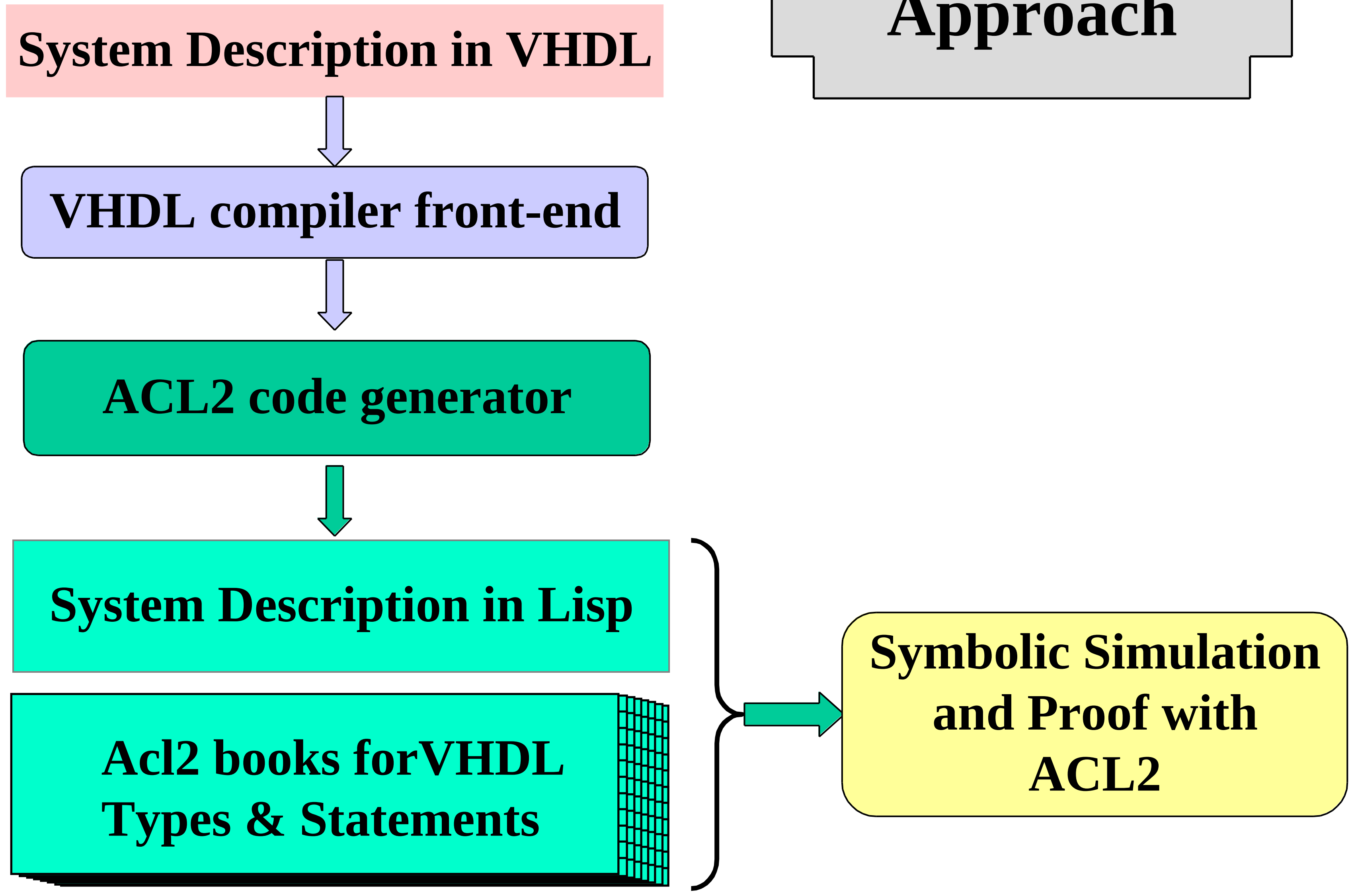
ACL2 Workshop
Austin, Texas, March 1999

Contents

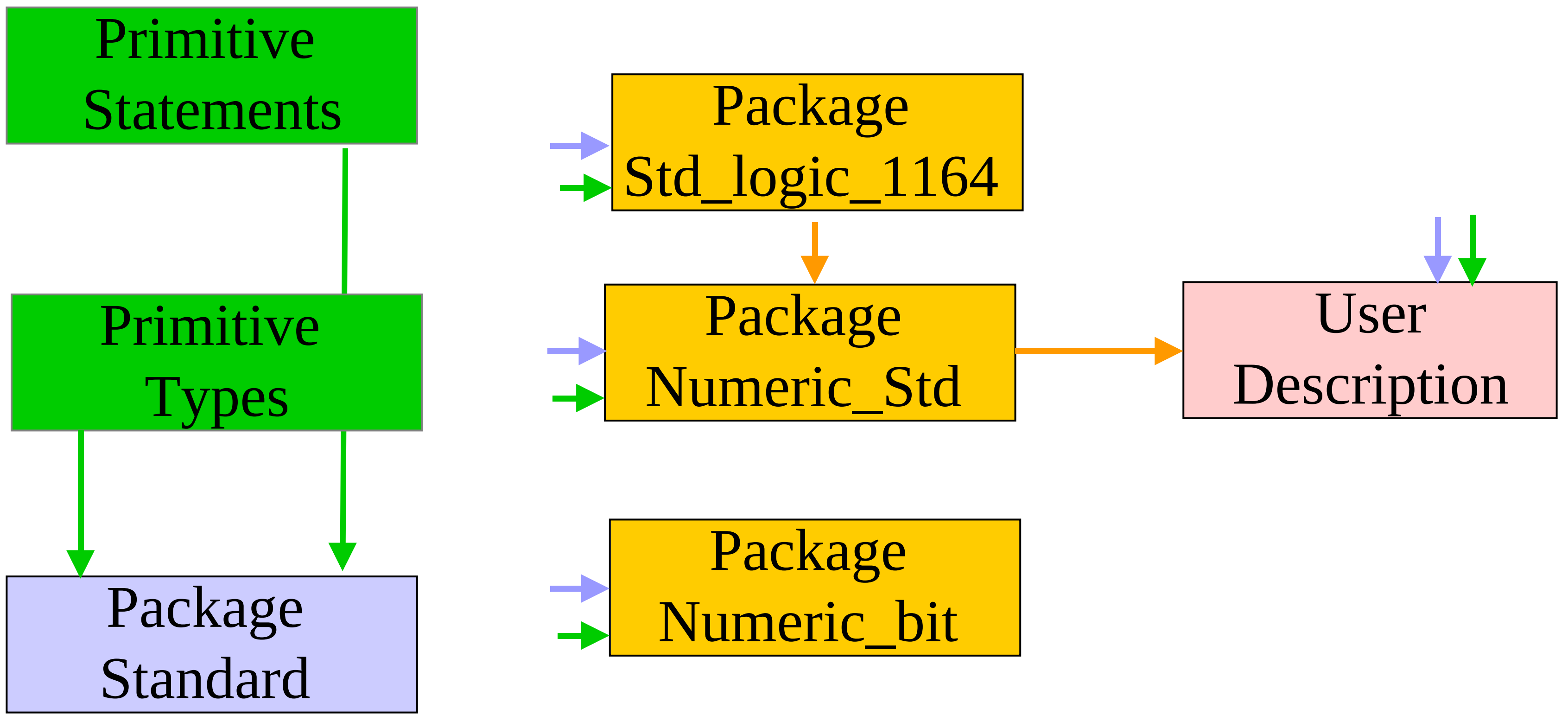
Context and Motivation

- TIMA: electronic design and CAD
- A long experience with HDL's
- Involvement with VHDL
  - Participation in IEEE standardization
  - Previous attempts at formal semantic definition
  - Use of NQTHM
- Objective: automate the formal verification
  - of the initial designer's high-level specifications
  - in a readable model

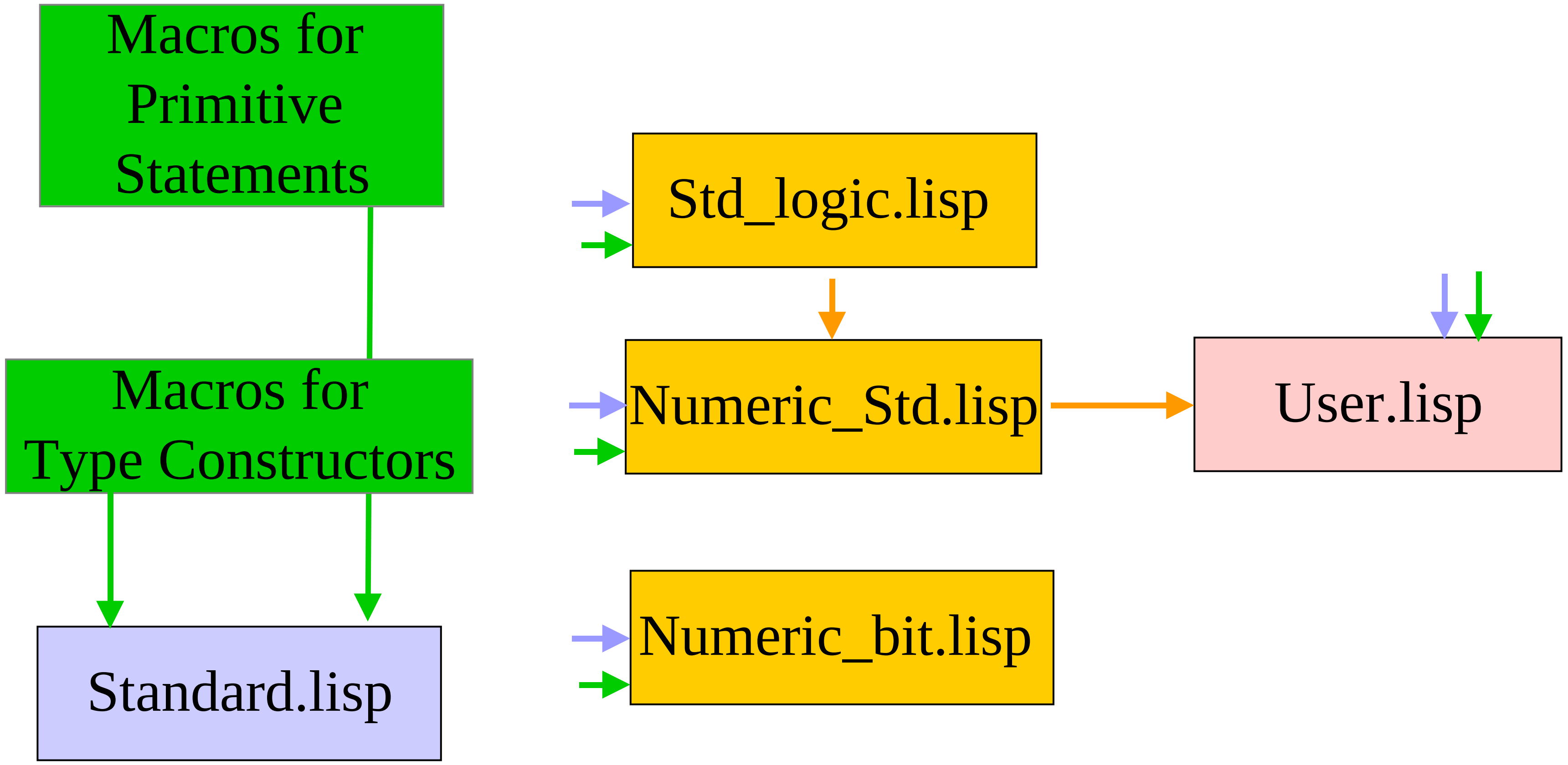
Approach


System Description in VHDL
VHDL compiler front-end
ACL2 code generator
System Description in Lisp
Acl2 books forVHDL Types & Statements
Symbolic Simulation and Proof with ACL2

Standard VHDL language and packages



Corresponding ACL2 Approach



Formalization of a VHDL subset

- Macros are used to define models for uninstanciated statements
- After instanciation, these models automatically generate
  - Definitions
  - Theorems
  - Constants
  - Miscellaneous operations (e.g. state modifications)

Enumerated data type

Type color is (Green, Yellow, Red, Blue, White);

ACL2 MACRO

```
(vhdl_type "color" :is ("Green" "Yellow" "Red" "Blue" "White"))
```

GENERATES DEFINITIONS :

```
COLOR-P (X)
ORDER<_COLOR (X Y)
```

GENERATES THEOREMS:

```
- COLOR_SIGNATURE_LEMMA
-  ORDER<_COLOR_SIGNATURE_LEMMA
-  ORDER<_COLOR_WELL_FOUNDED_RELATION
```

GENERATES CONSTANT:

```
*COLOR*      (8 ("Green" "Yellow" "Red" "Blue" "White"))
```

Ascending Interval

Type Byte is range 0 to 7;

ACL2 MACRO

```
(vhdl_type "byte" :is_range (0 "to" 7))
```

GENERATES DEFINITIONS :

BYTE-P (X)

GENERATES THEOREMS:

BYTE_SIGNATURE_LEMMA

GENERATES CONSTANT:

```
*BYTE*        (7  (0   "to"   7))
```

Descending Interval

Type Word is range 31 downto 0;

ACL2 MACRO

```
(vhdl_type "word" :is_range (31 "downto" 0))
```

GENERATES DEFINITIONS :

WORD-P (X)

GENERATES THEOREMS:

WORD_SIGNATURE_LEMMA

GENERATES CONSTANT:

```
*WORD*      (7  (31   "downto"   0))
```

Unconstrained array type

Type ScratchPad is array (integer range <>) of integer;

ACL2 MACRO

(vhdl_type "scratchpad" :is_array "integer" :range "<>" :of "integer")

GENERATES DEFINITIONS :

SCRATCHPAD-P (X)

GENERATES CONSTANT:

SCRATCHPAD   (5 integer integer)

Skeleton of macro type

```
(DEFMACRO vhdl_type (name &key
                          (is_range 'nil is_rangep)
                          (is_array 'nil is_arrayp)  (range 'nil rangep) (of 'nil))
                          (is 'nil isp)
(cond
 (is_rangep `(progn
   (defun ,(intern (concatenate 'string (string-upcase name) "-P")  "ACL2") (x) ...)
   (defconst ,(intern (concatenate 'string "*" (string-upcase name) "*") "ACL2") ...)
   (defthm ,(intern (concatenate 'string (string-upcase name)
    "_SIGNATURE_LEMMA") "ACL2") ...)))

 (is_arrayp `(progn
   (defconst ,(intern (concatenate 'string "*" (string-upcase name) "*")  "ACL2")  ...)
   (defun ,(intern (concatenate 'string (string-upcase name) "-P")  "ACL2") (x) ...)))

 (isp `(progn
    () ...)))))
```

Skeleton of macro type

```
(isp `(progn
     (defun ,(intern (concatenate 'string (string-upcase name) "-P")  "ACL2") (x) ...)
     (defun ,(intern
          (concatenate 'string "ORDER<_" (string-upcase name)) "ACL2") (x y) ...)
     (defconst ,(intern (concatenate 'string "*" (string-upcase name) "*")  "ACL2") ...)
     (defthm ,(intern
         (concatenate 'string (string-upcase name) "_SIGNATURE_LEMMA")  "ACL2") ...)
     (defthm ,(intern
         (concatenate 'string "ORDER<_" (string-upcase name "_SIGNATURE_LEMMA")
                 "ACL2") ...)
     (defthm ,(intern (concatenate 'string "ORDER<_" (string-upcase name)
                 "_WELL_FOUNDED_RELATION")  "ACL2") ...)))))
```

Standard and Numeric_Bit packages

- Two standard Vhdl packages :
  - standard: all types and operators of the LRM
  - Numeric_Bit: arithmetic interpretation of bit-vectors

- Example :
  - (vhdl_type "BIT" :is (#\0 #\1))

    with and, or, xnor, etc …
  - Unsigned, Signed

    with +, -, *, /, or, and, <, <=, etc ..

First notion of "model state"

- Fields :
  - Memory (for signals and variables) M
  - Processes of the architecture PR
  - The program counters of each process PC
- The state evolves with the interpretation of the statements

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))
({<=} "sig" "var1")
```

Memory

Var1 = 0

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))
({<=} "sig" "var1")
```

Memory

```
var1 = 0

Var_tab = ((1 0)
           (2 0)
           (3 0)
           (4 0)
           (5 0)
           (6 0)
           (7 0)
           (8 0))
```

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3")
({<=} "sig" "var1"))
```

Memory

```
var1 = 0

var_tab = ((1 0)
           (2 0)
           (3 0)
           (4 0)
           (5 0)
           (6 0)
           (7 0)
           (8 0))
var_index = (("ind1" 0)
             ("ind2" 0)
             ("ind3" 0)
             ("ind4" 0))
```

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))
({<=} "sig" "var1")
```

Memory

```
var1 = 0

var_tab = ((1 0)
           (2 0)
           (3 0)
           (4 0)
           (5 0)
           (6 0)
           (7 0)
           (8 0))
var_index = (("ind1" 0)
             ("ind2" 0)
             ("ind3" 0)
             ("ind4" 0))
sig_now =  0
sig_next =  0
```

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))
({<=} "sig" "var1")
```

Memory

```
var1 = "a"

var_tab = ((1 0)
           (2 0)
           (3 0)
           (4 0)
           (5 0)
           (6 0)
           (7 0)
           (8 0))
var_index = (("ind1" 0)
             ("ind2" 0)
             ("ind3" 0)
             ("ind4" 0))
sig_now =  0
sig_next =  0
```

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))      <-
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))
({<=} "sig" "var1")
```

Memory

```
var1 = (+ "b" "a")

var_tab = ((1 0)
           (2 0)
           (3 0)
           (4 0)
           (5 0)
           (6 0)
           (7 0)
           (8 0))
var_index = (("ind1" 0)
             ("ind2" 0)
             ("ind3" 0)
             ("ind4" 0))
sig_now =  0
sig_next =  0
```

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))   ←
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))
({<=} "sig" "var1")
```

Memory

```
var1 = (+ "b" "a")

var_tab = ((1 0)
           (2 1)
           (3 2)
           (4 3)
           (5 4)
           (6 5)
           (7 0)
           (8 0))
var_index = (("ind1" 0)
             ("ind2" 0)
             ("ind3" 0)
             ("ind4" 0))
sig_now =  0
sig_next =  0
```

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))  ←
({<=} "sig" "var1")
```

Memory

```
var1 = (+ "b" "a")

var_tab = ((1 0)
           (2 1)
           (3 2)
           (4 3)
           (5 4)
           (6 5)
           (7 0)
           (8 0))
var_index = (("ind1" 0)
             ("ind2" "a")
             ("ind3""b")
             ("ind4" 0))
sig_now =  0
sig_next =  0
```

Example

```
(vhdl_type "Tab" :is_array (1 "to" 8) :of "integer")
(vhdl_type "index" :is ("ind1" "ind2" "ind3" "ind4"))
(vhdl_type "tab_index" :is_array *index* :of "integer")

(vhdl_variable "var1" :type "integer")
(vhdl_variable "var_tab" :type *Tab*)

(vhdl_variable "var_index" :type *tab_index*)
(vhdl_signal "sig" :type "integer")
({=} "var1" "a")
({=} "var1" '(+ "b" "var1"))
({=} "var_tab" '(1 2 3 4 5) :range '(2 "to" 6))
({=} "var_index" '("a" "b") :range '("ind2" "to" "ind3"))
({<=} "sig" "var1")
```

Memory

```
var1 = (+ "b" "a")

var_tab = ((1 0)
           (2 1)
           (3 2)
           (4 3)
           (5 4)
           (6 5)
           (7 0)
           (8 0))
var_index = (("ind1" 0)
             ("ind2" "a")
             ("ind3""b")
             ("ind4" 0))
sig_now =  0
sig_next =  (+ "b" "a")
```

Conclusion

- Hopes:
  - Increased readability of circuit models in ACL2
  - Improve industrial acceptance of theorem provers in Europe
- Early stage
  - Few VHDL statements in our subset
  - No real circuit has been verified
  - Hand made translation
- Next steps
  - Finite loops (macros perform unrolling), including nested loops
  - Structural architectures with IF and FOR .. GENERATE.
  - Generic parameters.
  - Time