Verification of a cryptographic circuit: SHA-1 using ACL2

Diana Toma and Dominique Borrione
TIMA - VDS Group , Grenoble, France

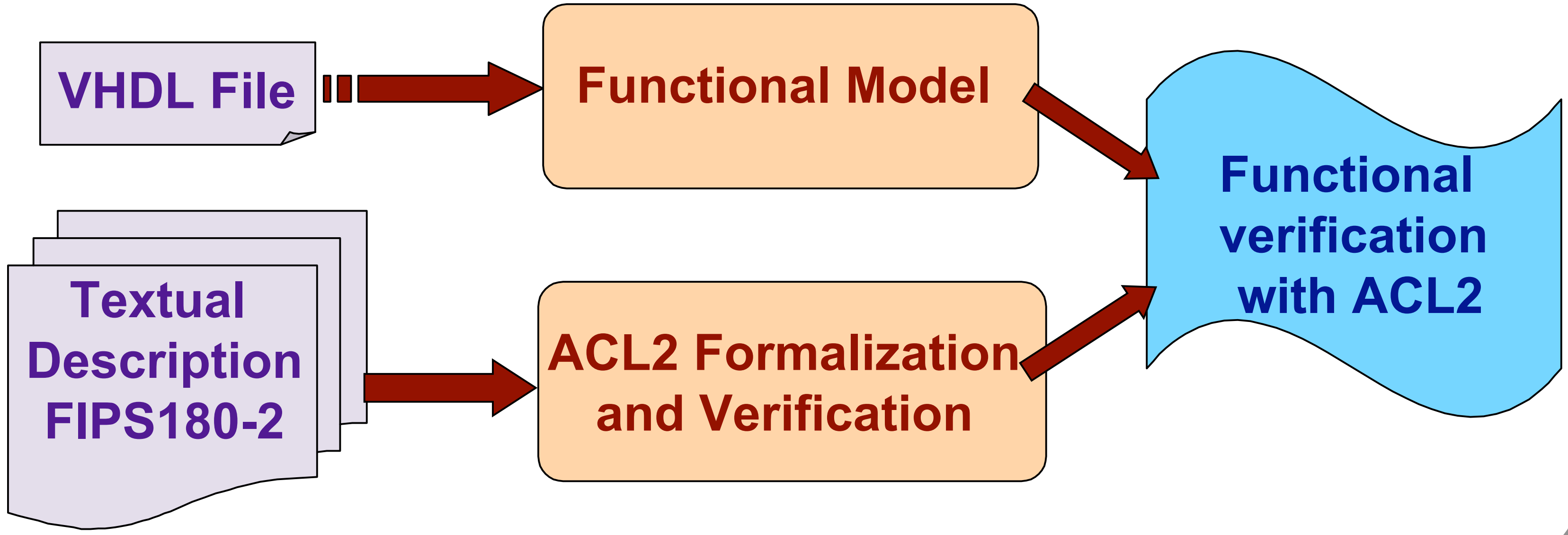
ACL2 Workshop 2004, Austin, Texas

Motivations

- Context: digital systems on a chip
  - Modules with data part
  - Early design steps
- Characteristics
  - Lack of sound semantics, lack of synthesis tools
  - Ad hoc verification, essentially by simulation (Matlab, SystemC)
  - Compliance of the synthesizable level (Verilog/VHDL) not proven
- Objective
  - Specification validation, before reaching the RTL level
  - Implementation verification

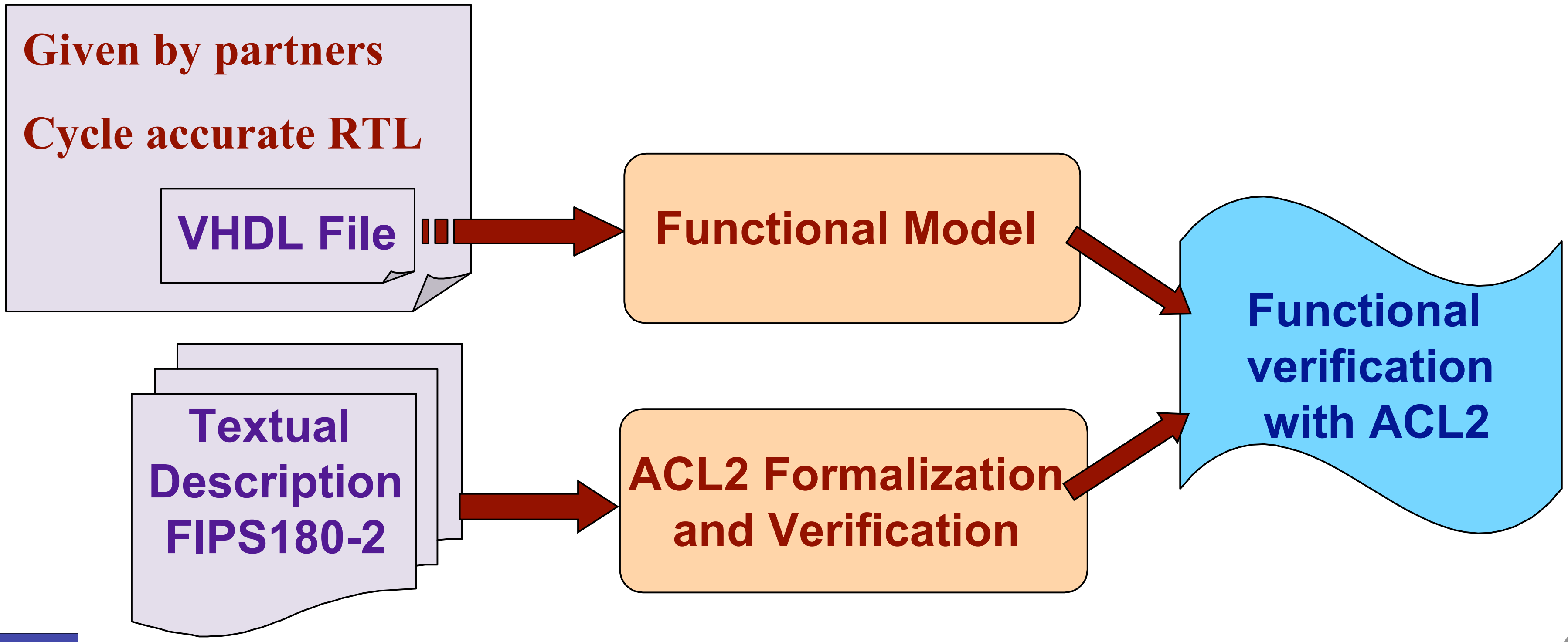
Industrial Project : ISIA2

- Design of a chip for secure transmissions
- Our participation:
  - Verification of the hash block


VHDL File
Functional Model
Textual Description FIPS180-2
ACL2 Formalization and Verification
Functional verification with ACL2

Industrial Project : ISIA2

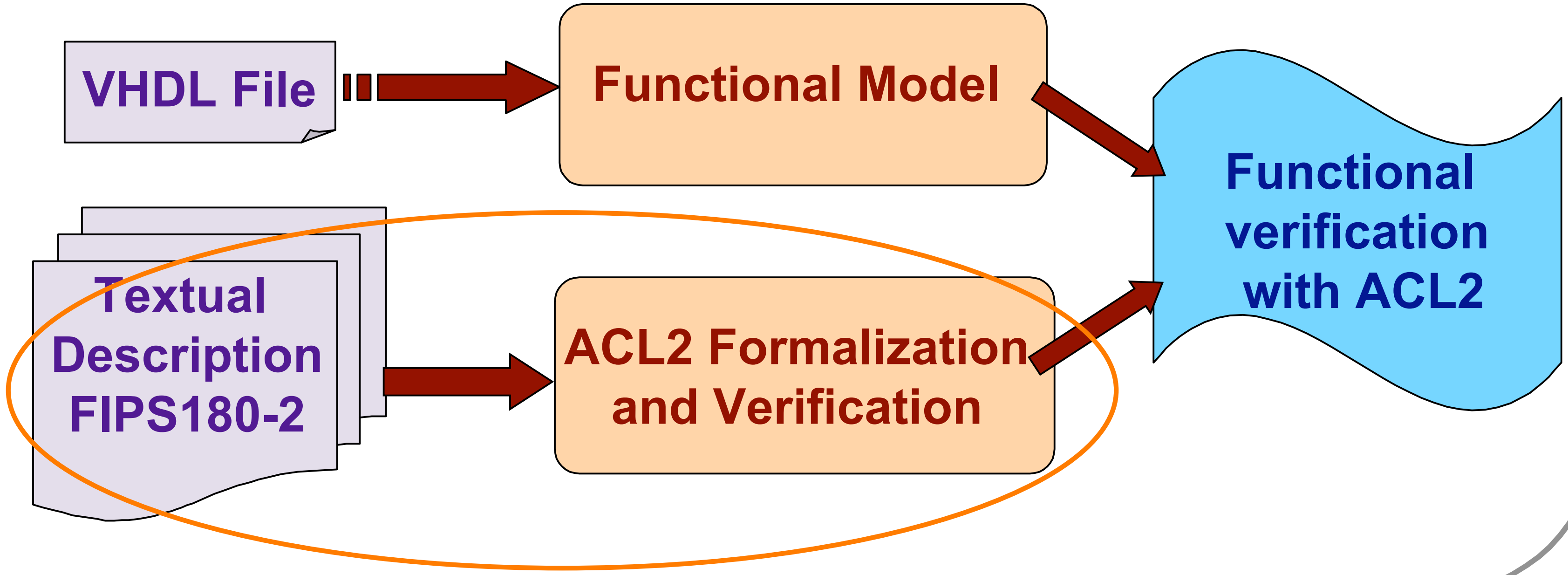
- Design of a chip for secure transmissions
- Our participation:
  - Verification of the hash block



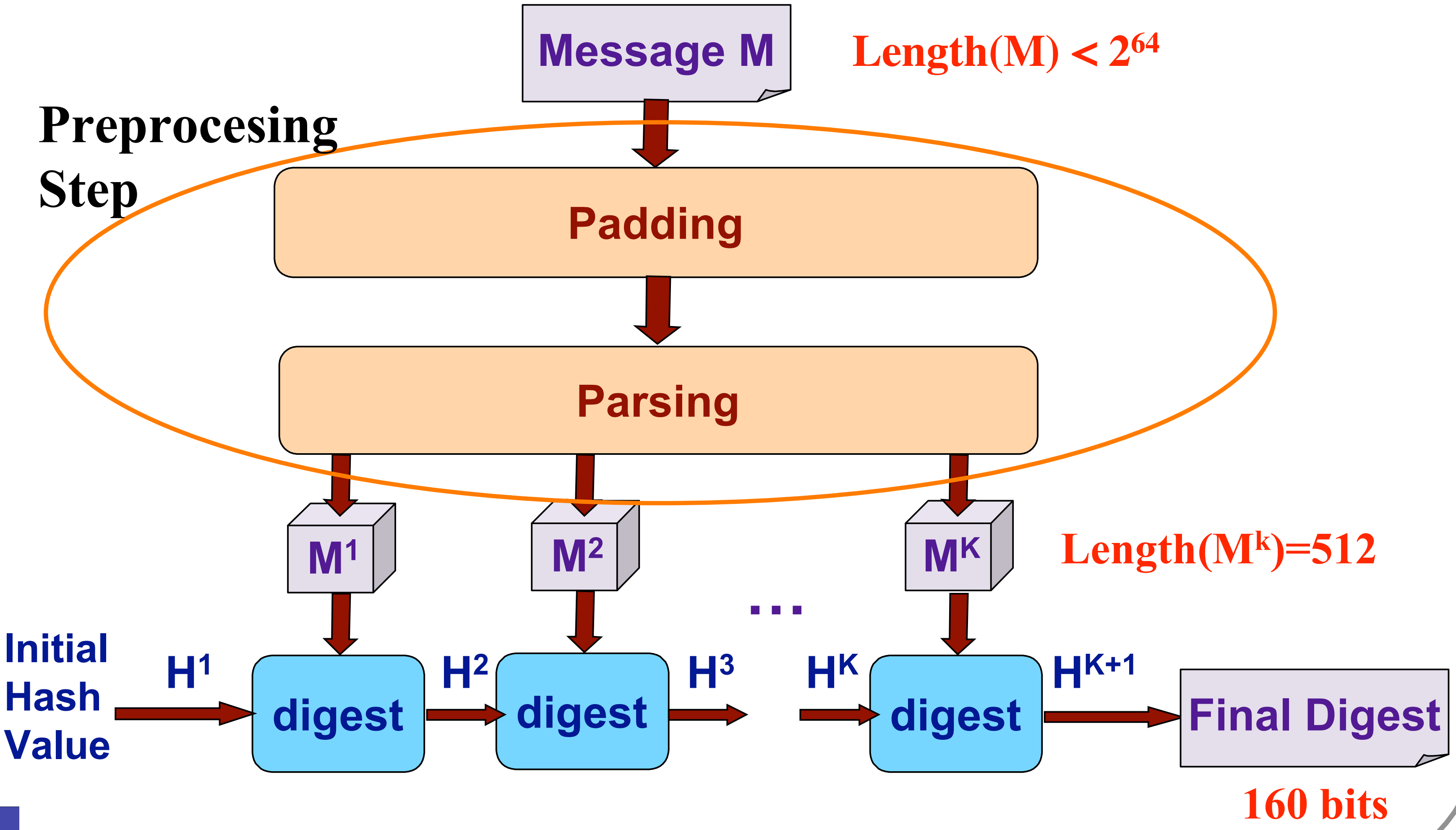
Given by partners
Cycle accurate RTL
VHDL File
Functional Model
Textual Description FIPS180-2
ACL2 Formalization and Verification
Functional verification with ACL2

Industrial Project : ISIA2

- Design of a chip for secure transmissions
- Our participation:
  - Verification of the hash block
  - Specification: standardized Secure Hash Algorithm

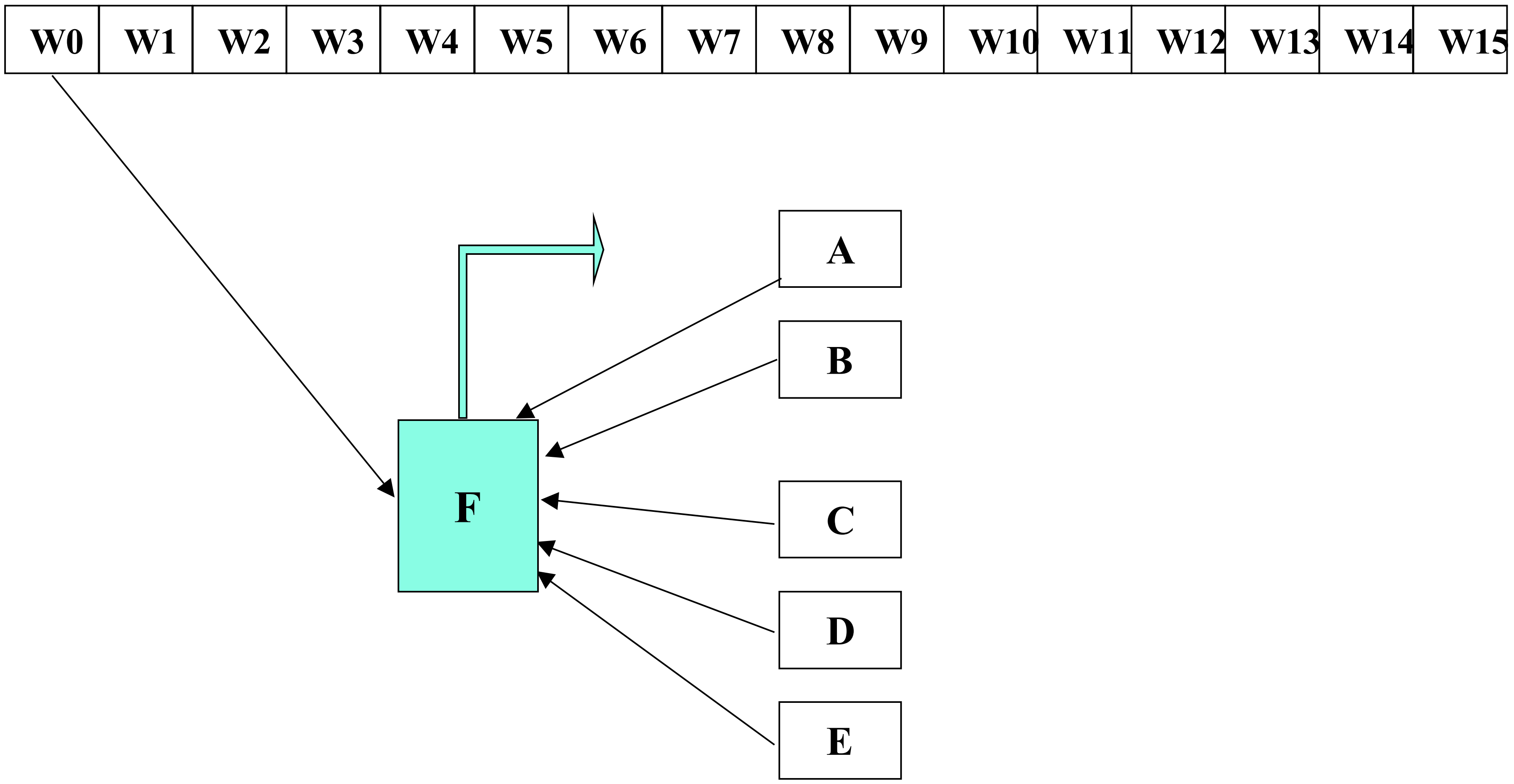

VHDL File
Functional Model
Textual Description FIPS180-2
ACL2 Formalization and Verification
Functional verification with ACL2

SHA Algorithm



Computation of one block digest

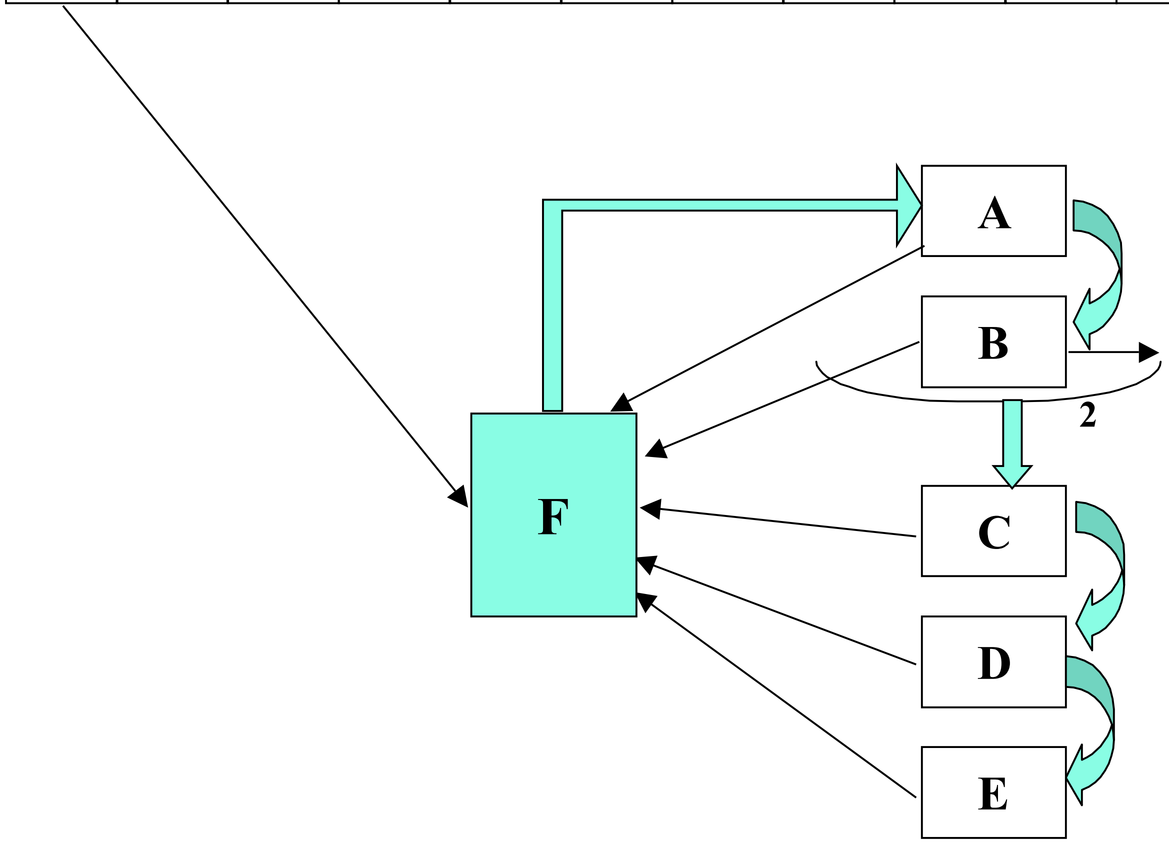
First 16 iterations



Computation of one block digest

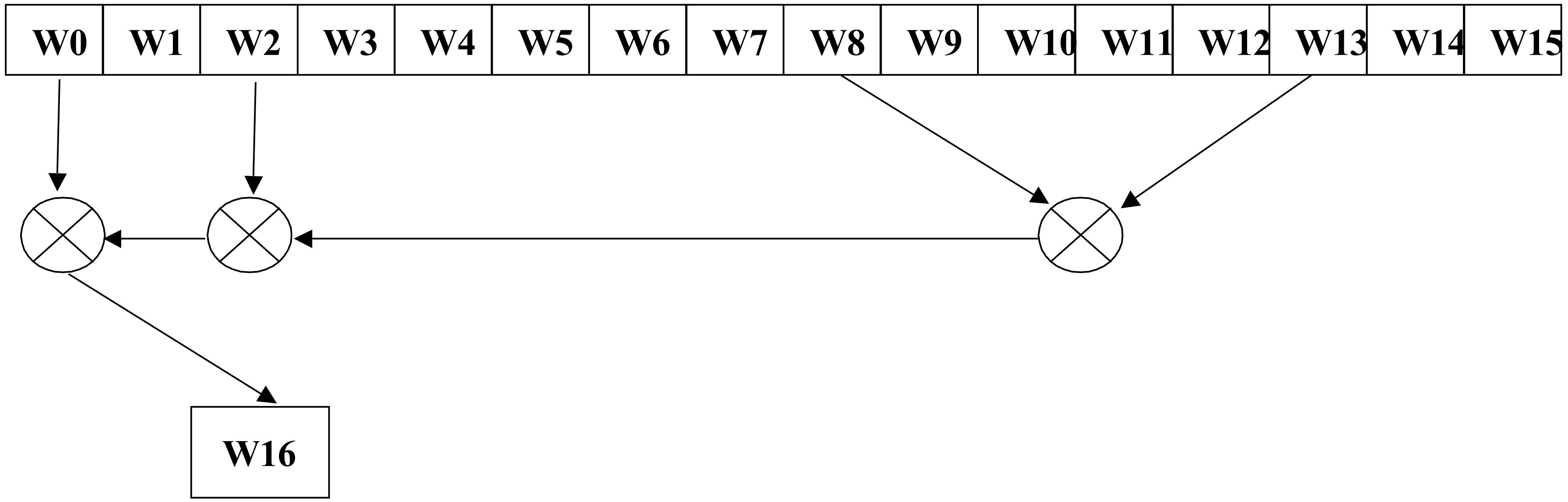
First 16 iterations

W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15



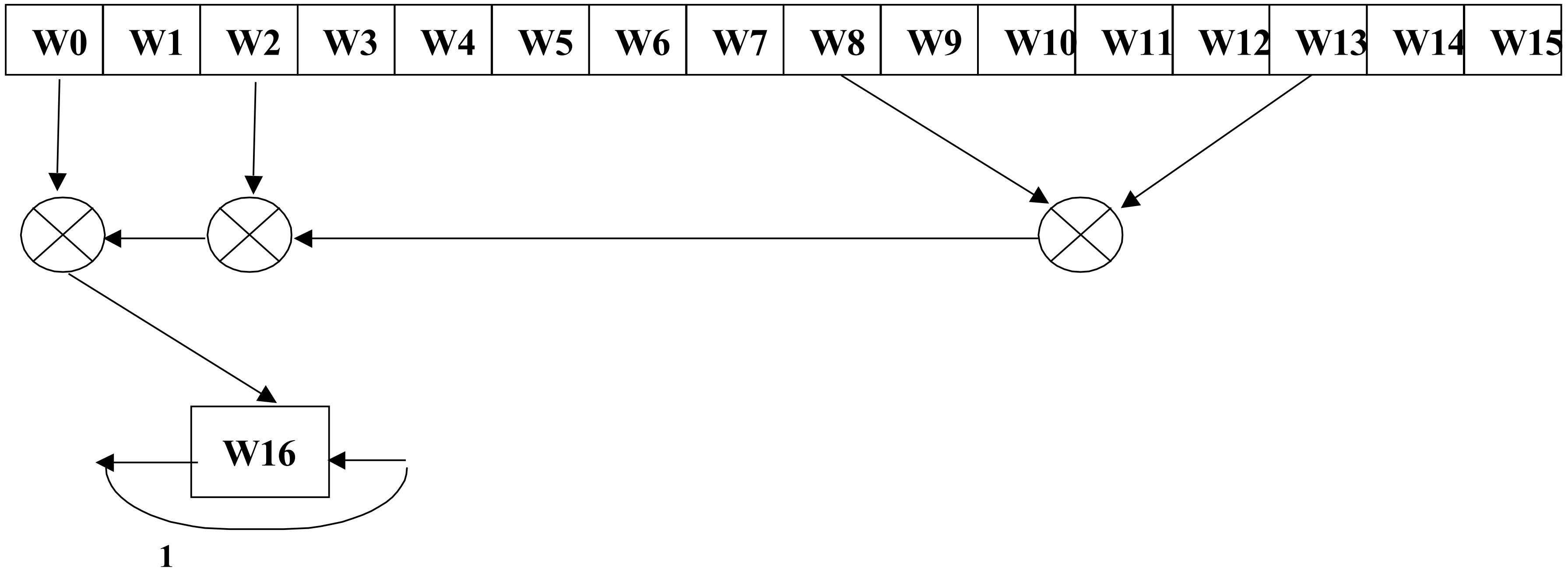
Computation of one block digest

Next 64 iterations



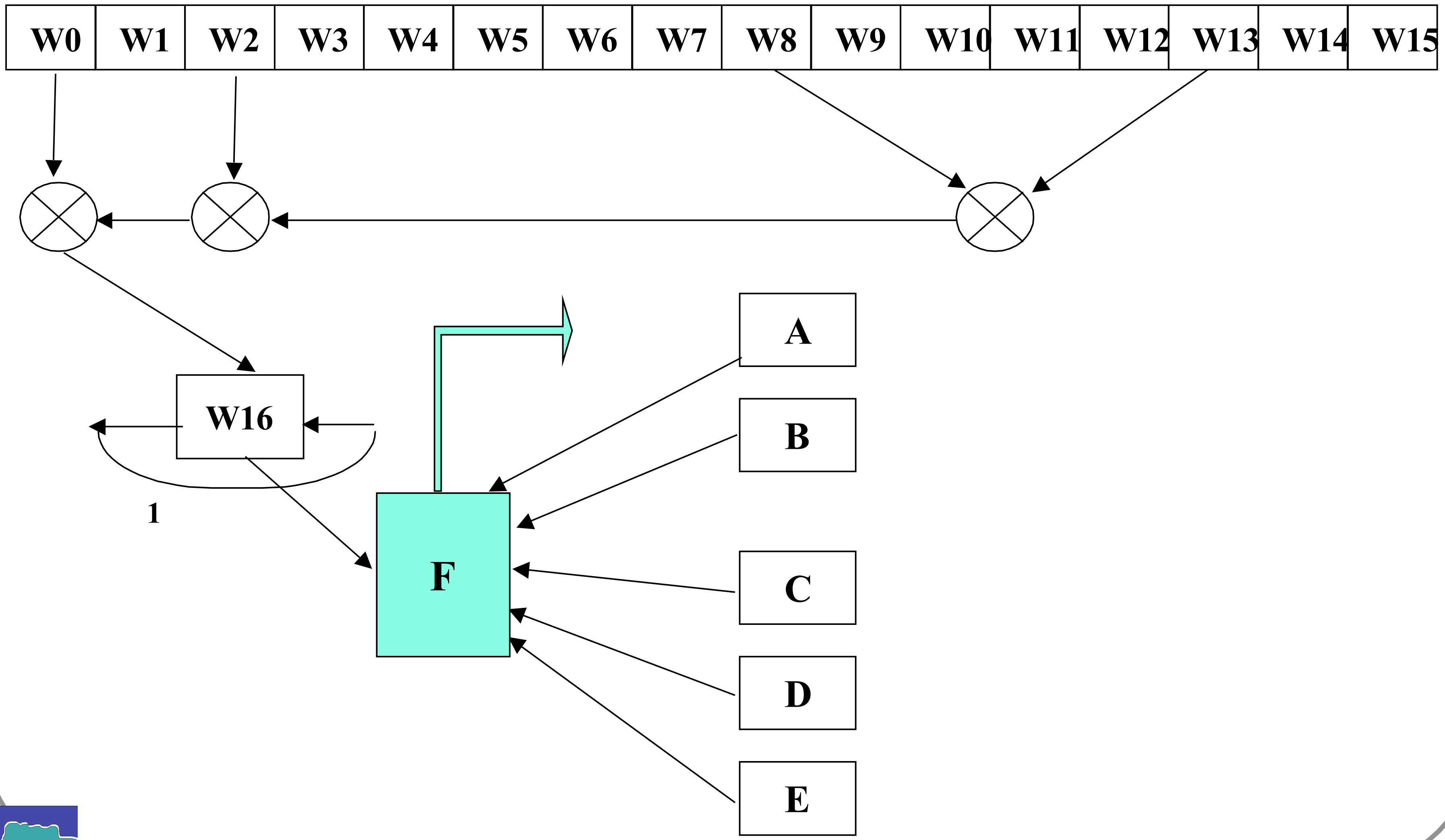
Computation of one block digest

Next 64 iterations



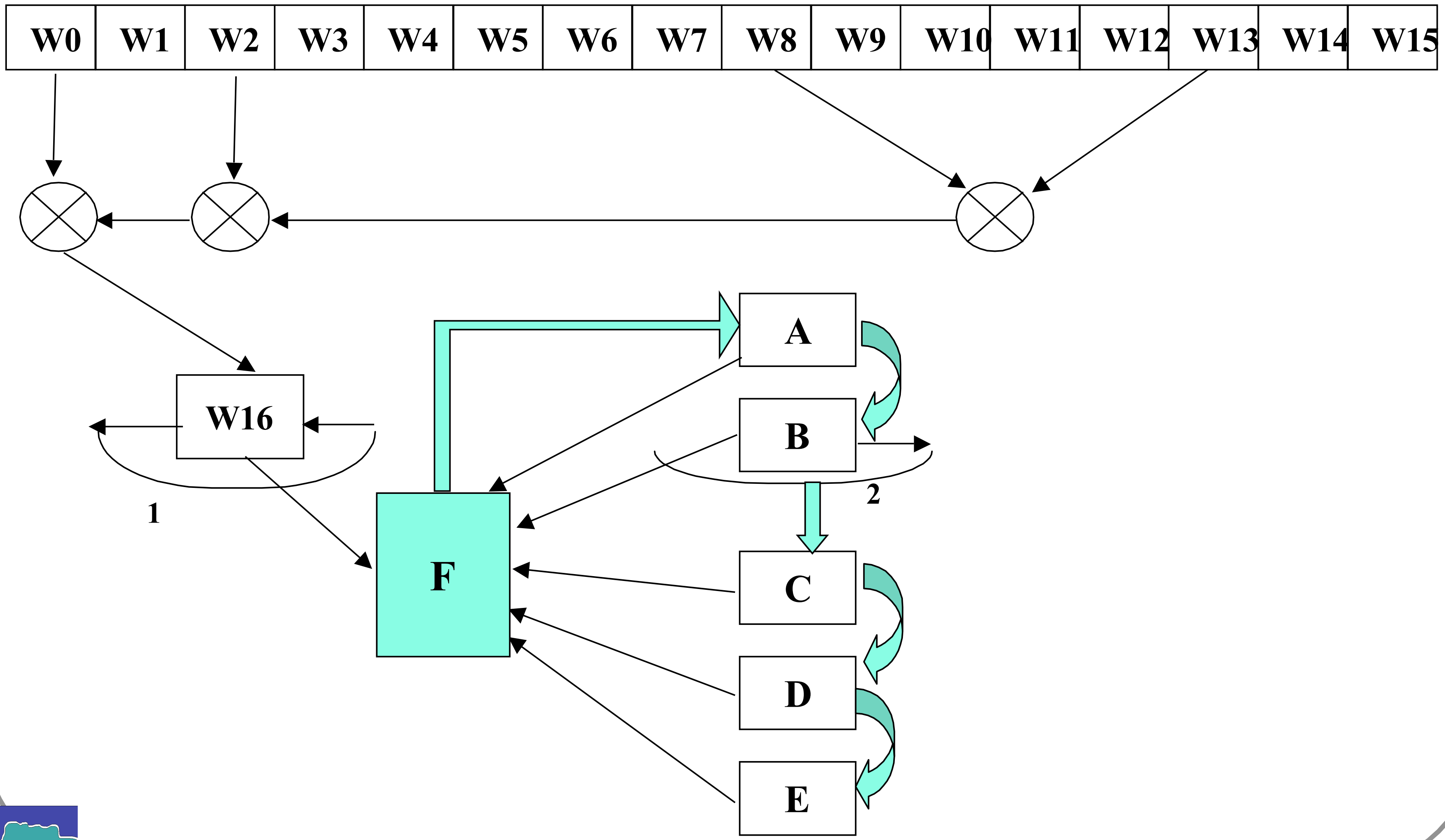
Computation of one block digest

Next 64 iterations



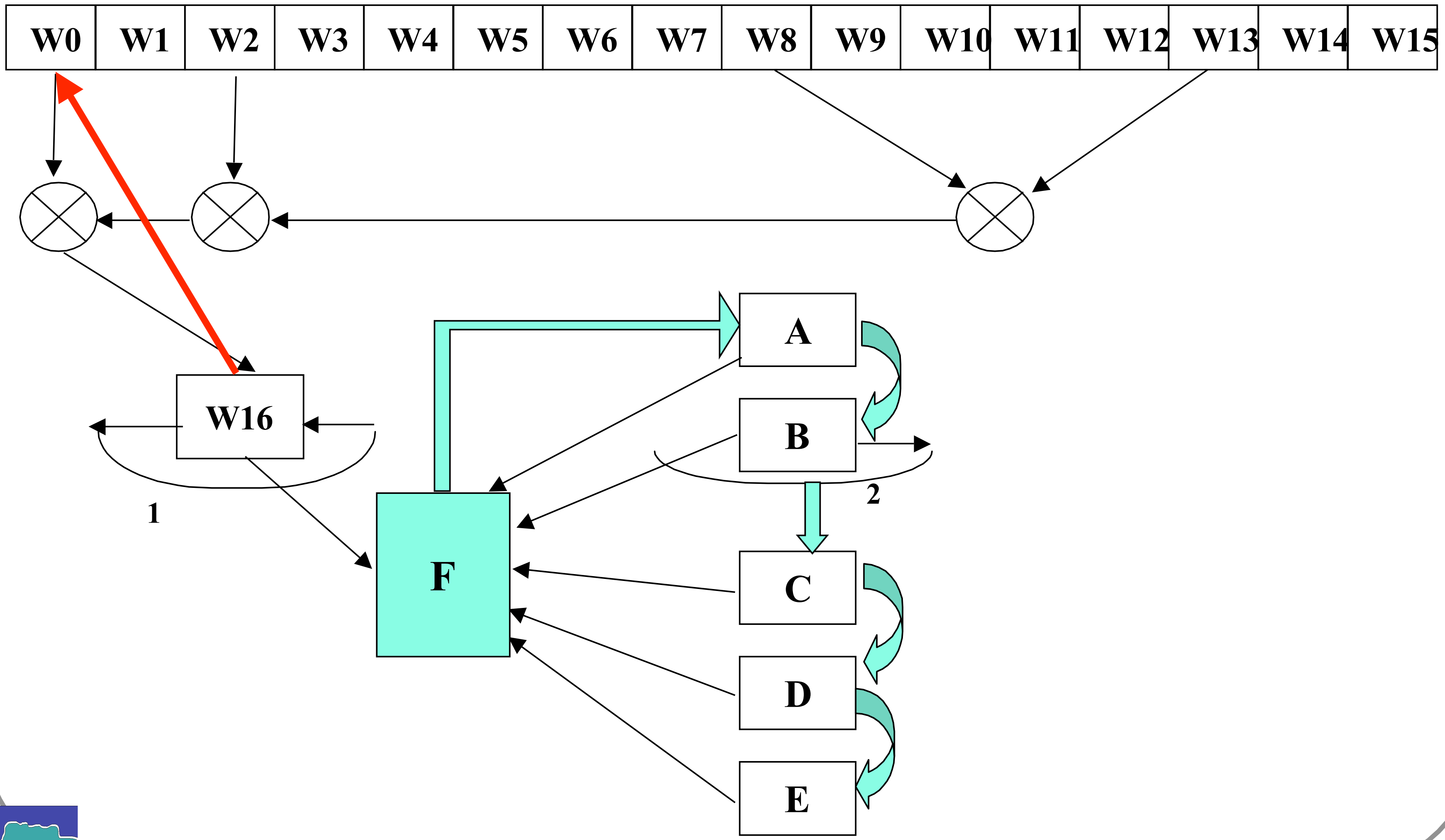
Computation of one block digest

Next 64 iterations



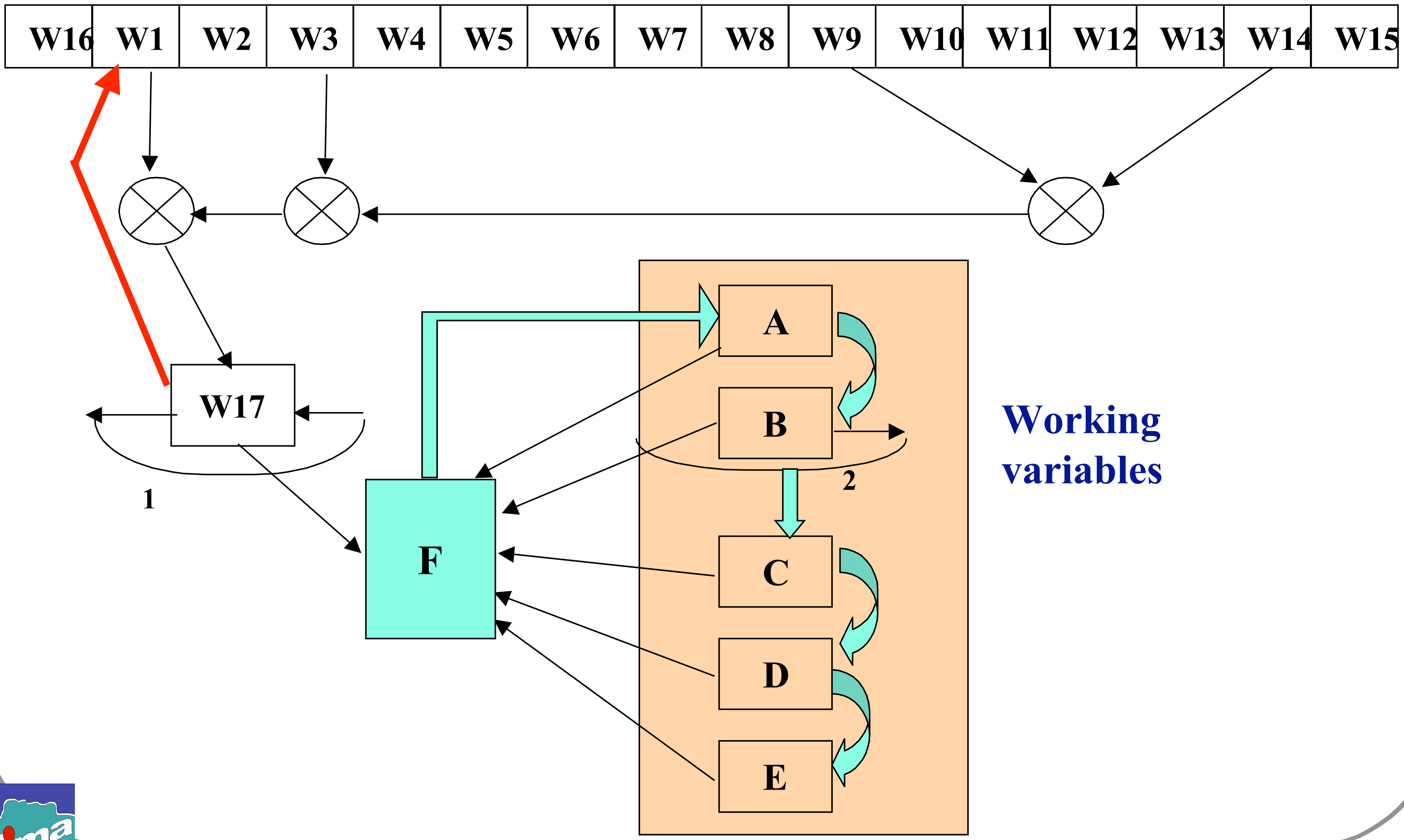
Computation of one block digest

Next 64 iterations



Computation of one block digest

Next 64 iterations



Computation of the message digest

- **For each 512 bit-block**

  Apply the block digest step 80 times

  Compute the hash values for the next block

```
(defun digest-one-block-spec (j working-variables m-i)
    …
  (let ((new-m-i (if (<= 16 j)
                     (repl (s j) (word-spec j m-i) m-i)
                    m-i))))
   (if (<= 80 j) working-variables)
       (digest-one-block-spec (+ 1 j))
              (list (temp-spec j working-variables new-m-i)
                    (nth 0 working-variables)
                    (rotl 30 (nth 1 working-variables))
                    (nth 2 working-variables)
                    (nth 3 working-variables))
              new-m-i))))
```

Computation of the message digest

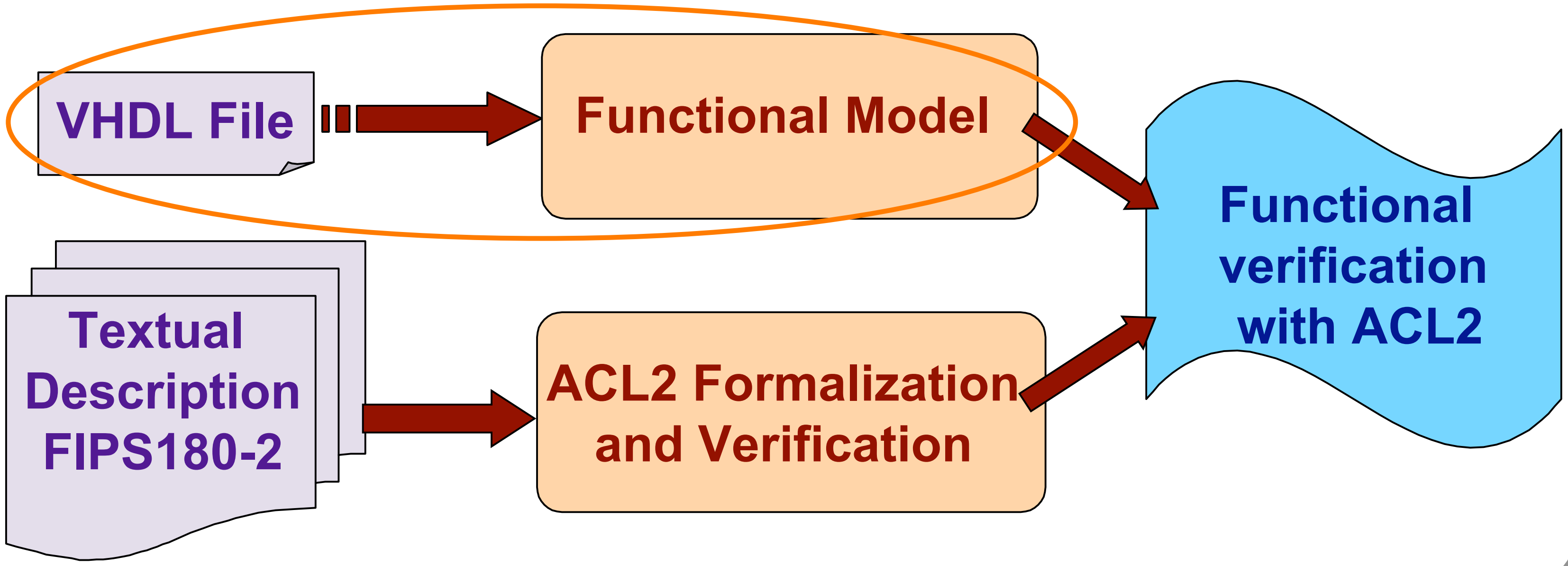
- Global function
  - Recursive in the number of blocks of M
  - Direct translation of the standard

  SHA1 (M) = digest-spec (parsing (padding(M), 512), H_INIT)

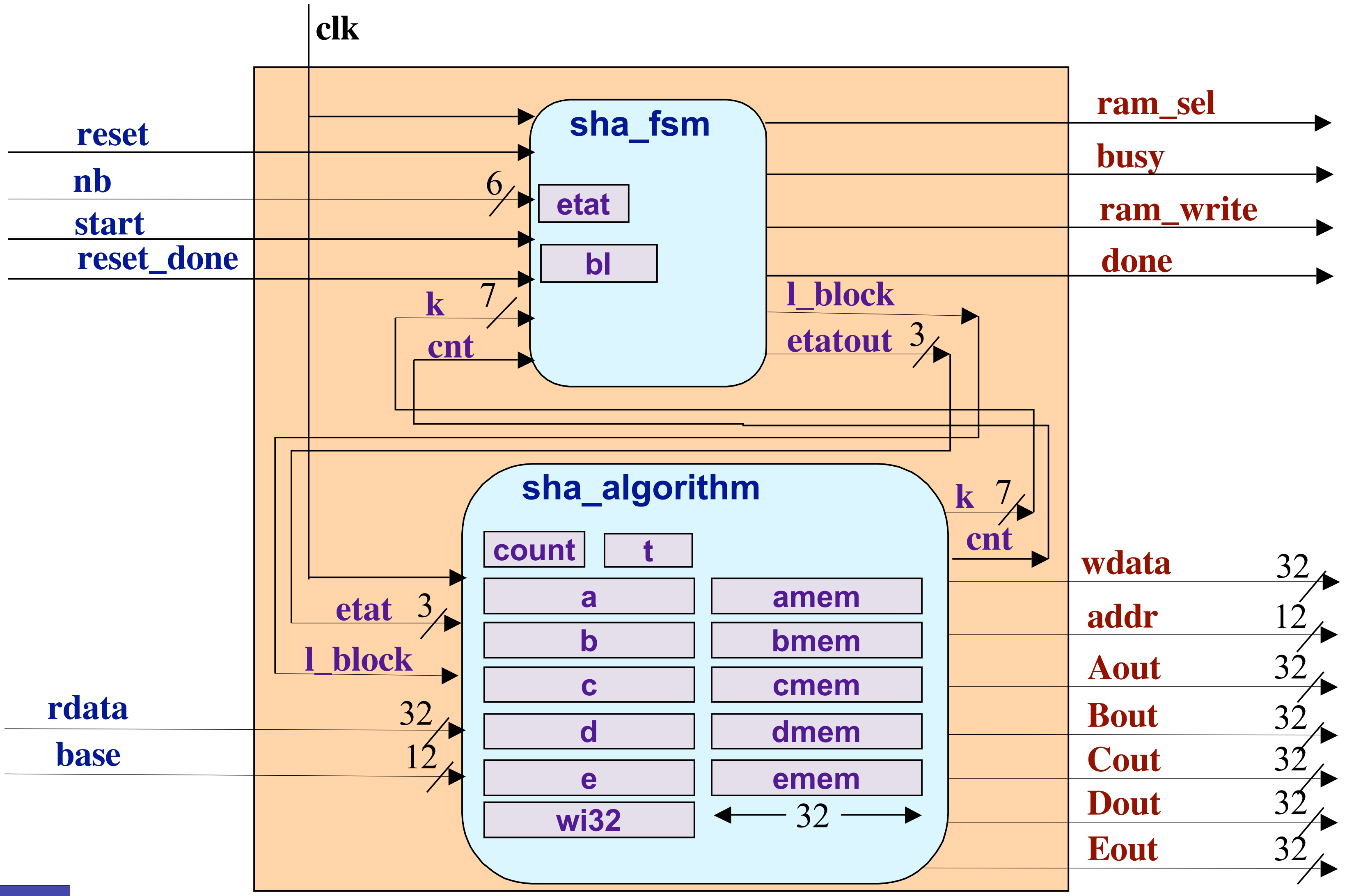
```
(defun digest-spec (m hash-values)
    (if (endp m) hash-values
        (digest-spec (cdr m)
                (intermediate-hash hash-values
                     (digest-one-block-spec 0 hash-values
                            (parsing (car m) 32))))))
```

Extracting the model of the implementation

- Should be automatic
- Should provide same results as VHDL on same numeric test vectors
- Same kind of formalization as the specification

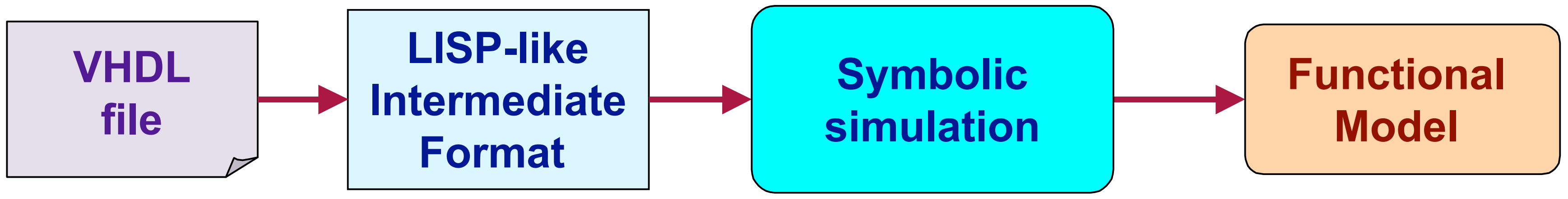

VHDL File
Functional Model
Textual Description FIPS180-2
ACL2 Formalization and Verification
Functional verification with ACL2

SHA-1


clk
reset
nb
start
reset_done
sha_fsm
etat
bl
6
k
7
cnt
ram_sel
busy
ram_write
done
l_block
etatout
3
sha_algorithm
count
t
a
b
c
d
e
wi32
amem
bmem
cmem
dmem
emem
32
k
7
cnt
etat
3
l_block
rdata
32
base
12
wdata
32
addr
12
Aout
32
Bout
32
Cout
32
Dout
32
Eout
32

Cycle-level VHDL model

- Execution of the VHDL simulation algorithm for one clock cycle
  - Intermediate signals and non-memorising variables of the source VHDL design are eliminated
  - Symbolic simulation system and symbolic rewriting of expressions performed with Mathematica
- Extraction of one transition function for each output and each state element of the design
- No limitation to the logic data types



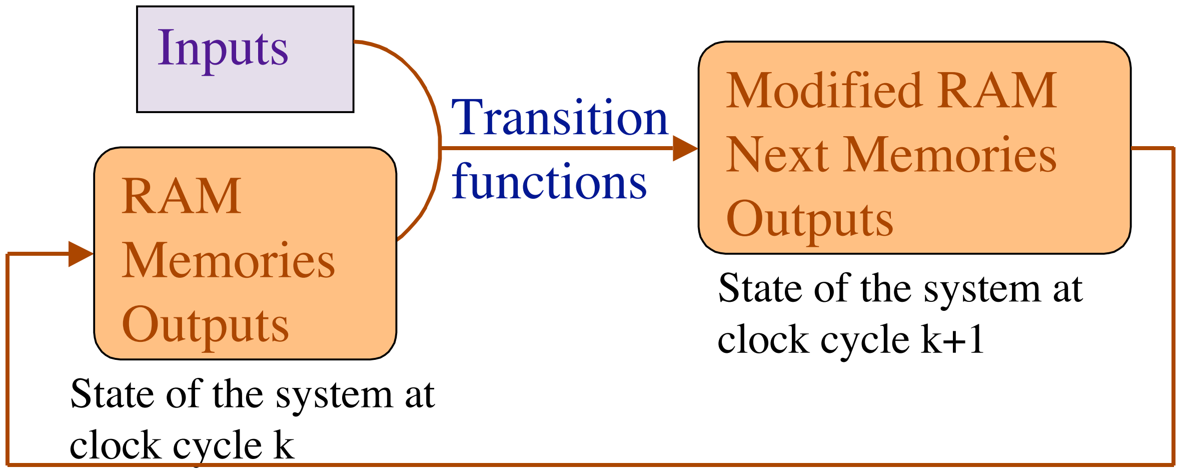
ACL2 model of the VHDL design

```
(defun Sim-step (input memory)
…
(list (nextsig_a bl a_mem state t1 rdata a b c d e wi32 count)
      (nextsig_b bl b_mem state a b t1)
      (nextsig_c bl c_mem state b c t1)
       …
      (nextsig_done reset reset_done start state cnt bl done)
)
```

Nextsig_x function is: if-expression, boolean or arithmetic expression

```
(defun nextsig_done (reset reset_done start state cnt bl done)
 (if (equal reset 1) 0
    (if (equal state *idle*)
        (if (or (equal start 1) (equal reset_done 1))
          0 done)
      (if (and (equal state *resultw*) (equal cnt 0)
               (equal bl (list 0 0 0 0 0 0)))
        1 done))))
```

ACL2 model of the VHDL design



```
(defun sha-vhdl (L-input st)
(if (atom L-input) st
 (let* ((memory (car st))
        (ram (cdr st))
        (new-mem
          (Sim-step (cons (read-ram memory ram) (car L-input))
                memory)))
   (sha-vhdl (cdr L-input)
          (cons new-mem (write-ram new-mem ram))))))
```

Functional verification

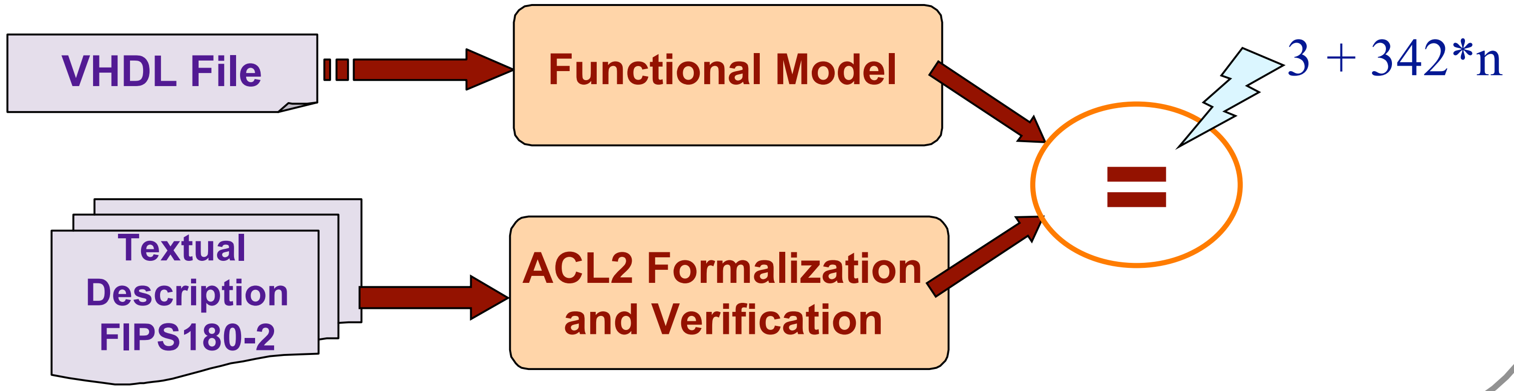
Main Theorem

For all

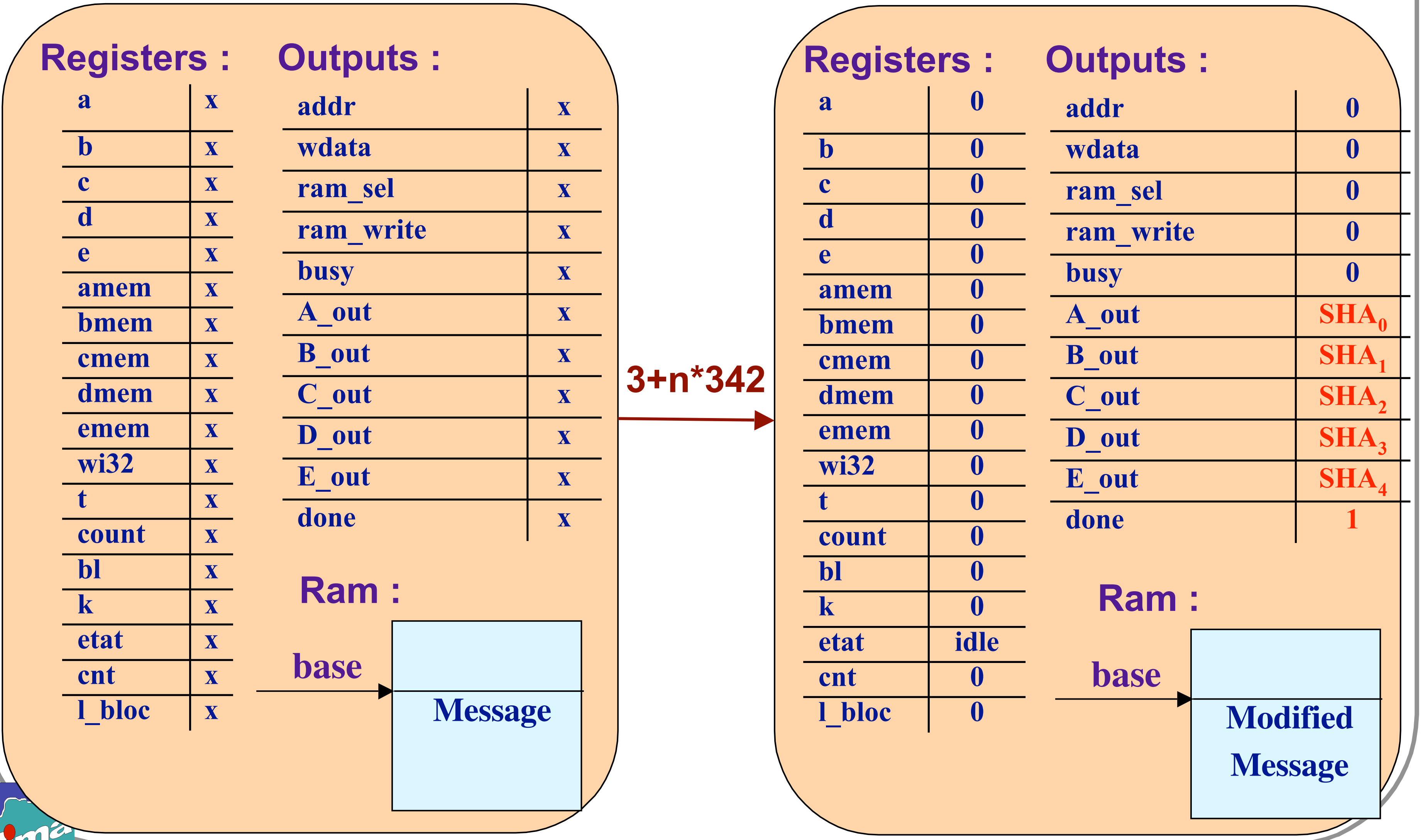
- n, positive integer
- RAM(*base*),
- message of size n blocks

Cycle	1	2	3 …
reset	1	0	0
start	x	1	x
nb	x	x	*n*
base	x	x	*base*

After executing the VHDL SHA1 circuit model, for 3 + (342 * n) clock cycles, the system is in its final state (*done*=1) and the output registers Aout, Bout, Cout, Dout, Eout hold the expected message digest.


VHDL File
Functional Model
Textual Description FIPS180-2
ACL2 Formalization and Verification
=
3 + 342*n

Main theorem



Registers :

a	x
b	x
c	x
d	x
e	x
amem	x
bmem	x
cmem	x
dmem	x
emem	x
wi32	x
t	x
count	x
bl	x
k	x
etat	x
cnt	x
l_bloc	x

Outputs :

addr	x
wdata	x
ram_sel	x
ram_write	x
busy	x
A_out	x
B_out	x
C_out	x
D_out	x
E_out	x
done	x

Ram :

base → Message

Registers :

a	0
b	0
c	0
d	0
e	0
amem	0
bmem	0
cmem	0
dmem	0
emem	0
wi32	0
t	0
count	0
bl	0
k	0
etat	idle
cnt	0
l_bloc	0

Outputs :

addr	0
wdata	0
ram_sel	0
ram_write	0
busy	0
A_out	SHA_0
B_out	SHA_1
C_out	SHA_2
D_out	SHA_3
E_out	SHA_4
done	1

Ram :

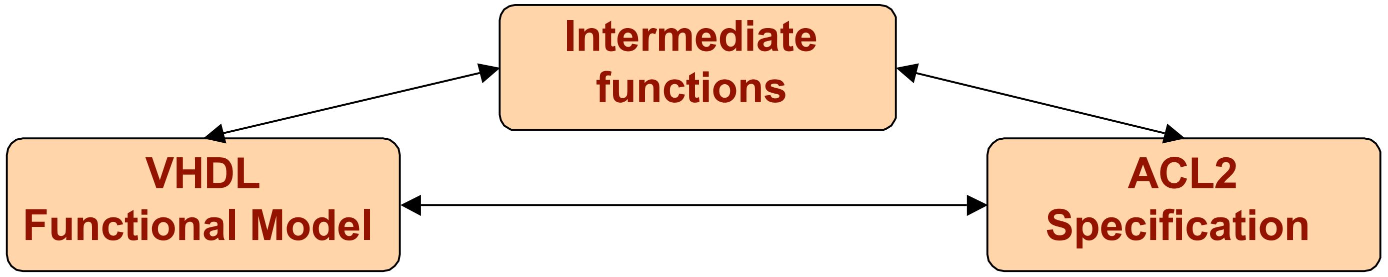
base → Modified Message

Principle of the proof

- Stepwise process, details are circuit specific
- For SHA1 :

– 3 cycles	reset + initialization of internal registers
– 342 cycles	digest computation for one block
• 16 cycles	read 16 32-bit words of the block and compute the first 16 iterations of the digest algorithm
• 320 cycles	compute intermediate digest (5*64)
• 5 cycles	combine with hash values
• 1 cycle	ready for next block

- Techniques: generalization followed by induction



Results of VHDL model for one block

```
(defun digest-one-block-impl (i count a b c d e ram base nb bl)
 (if (zp i) (list a b c d e)
     (digest-one-block-impl (- i 1) (plus count 1)
        (temp-impl count a b c d e
           (new-word count base nb bl ram)
        a (next-b b) c d
        (new-ram count base nb bl ram)
        base nb bl)))

(defthm sha-vhdl-for-one-block
 (implies (and (hyp (memory st)) (ramp (ram st) (base input)) …
               (hyp-input input) (init (memory st)))
  (equal (abcde-mem (sha-vhdl (fistn 342 input) st))
         (intermediate-digest (a st) (b st) (c st) (d st) (e st)
              (digest-one-block-impl 80 `(0 0 0 0 0 0 0 0)
                     (a st) (b st) (c st) (d st) (e st)
                     (ram st) (base input) (nb input) (bl st))))))
```

Results of VHDL model for n blocks

```
(defun digest-impl (hash-values ram base nb bl)
(if (equal (bv-nat-be bl) 0) hash-values
    (digest-impl
      (intermediate-digest hash-values
          (digest-one-block-impl 80 `(0 0 0 0 0 0 0 0)
              (car hash-values) (nth 1 hash-values)
              (nth 2 hash-values) (nth 3 hash-values)
              (nth 4 hash-values) ram base nb bl))
      (modified-ram 80 `(0 0 0 0 0 0 0 0) ram base nb bl)
      base nb (minus bl 1))))

(defthm sha-vhdl-for-n-blocks
 (implies (and (hyp (memory st)) (ramp (ram st) (base st))
               (hyp-input input) (init (memory st)) …
               (<= n (bv-nat-be (bl st))))
      (equal (abcde-mem (sha-vhdl (fistn (* n 342) input) st))
             (digest-impl (abcde-mem st)
                          (ram st) (base input)
                          (nb input) (nat-bv-be n 6)))))
```

Main Theorem

```
(defthm digest-implementation=digest-specification
 …
  (equal (digest-impl (list *h0* *h1* *h2* *h3* *h4*)
                      (ram st) (base input)
                      (nb input) (nb input))
         (digest-spec
              (parsing (get-message-from-ram
                          (nb input) (base input) (ram st))) 512)
              (list *h0* *h1* *h2* *h3* *h4*)))
```

Generalized form:

```
(defthm digest-implementation=digest-specification->general
 …
  (equal (digest-impl hash-values ram base nb bl)
         (digest-spec
              (parsing (get-message-from-ram bl
                         (plus base (* 16 (bv-nat-be (minus nb bl))))
                         ram)) 512)
         hash-values)
```

Ancillary theorems

- The digest of one block is the same in both specification and implementation

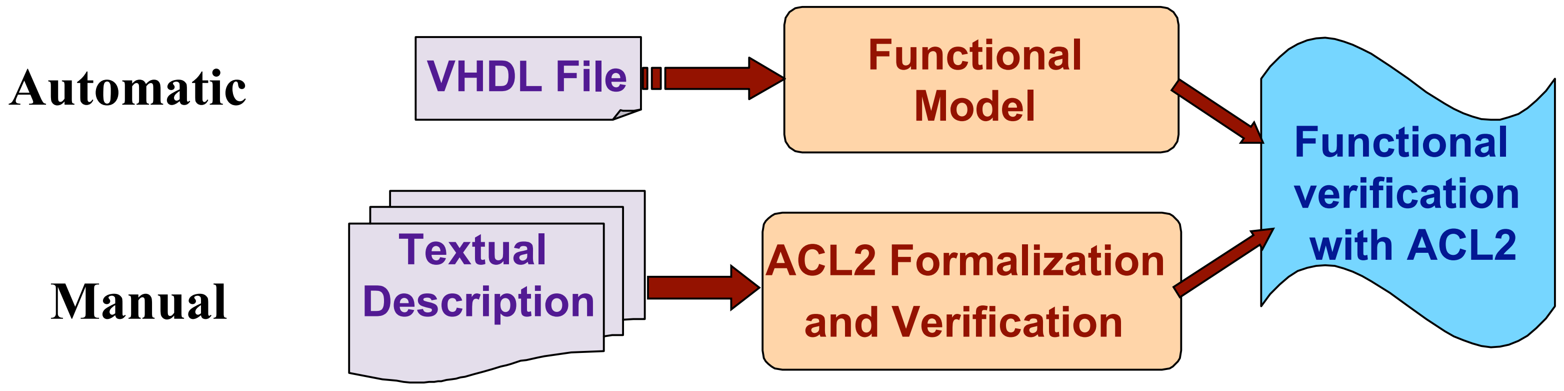
```
(defthm digest-implementation=digest-specification->one-block
…    (equal (digest-one-block-impl 80 `(0 0 0 0 0 0 0 0)
                             a b c d e ram base nb bl)
              (digest-one-block-spec 0 (list a b c d e)
                    (get-message-from-ram 1
                          (plus base (* 16 (bv-nat-be (minus nb bl))))
                          ram)))
```

- After processing k blocks, the memory storing the rest of the message to be processed is unaltered

```
(defthm ram-is-unaltered
 (implies (and …
               (<= (plus base (* 16 (bv-nat-be (minus nb bl))))
                    address))
      (equal (get-message-from-ram n address
                    (modified-ram j count ram base nb bl))
               (get-message-from-ram n address ram))
```

Conclusion

- Prove correctness of the VHDL design vs specification for SHA-1
  - Couple of errors in the VHDL design were uncovered
  - 150 functions, 750 theorems -> 45% reusable
- Development of a stepwise method based on the state machine of the RTL
- Synergy between various symbolic techniques :
  - Symbolic simulation and theorem proving
- The use of executable logic was helpful


Automatic
Manual
VHDL File
Textual Description
Functional Model
ACL2 Formalization and Verification
Functional verification with ACL2

Thank you