Design Description Languages

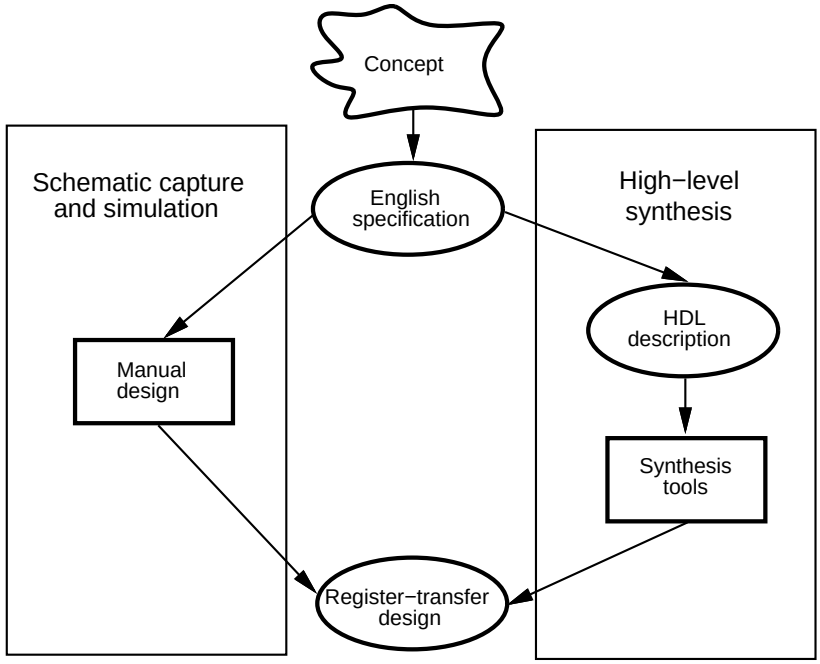
Chapter 4

Source: Gajski, Dutt, Wu, Lin

"High−Level Synthesis"

Kluwer Academic Publishers, 1992

NEED FOR HARDWARE DESCRIPTION LANGUAGE



Design specification

Documentation/Redesign

Verification/Simulation

Communication between designers

DESIGN SPECIFICATION

1. Conceptual capture

2. Higher abstraction level

3. Detect Early Design Errors

4. Model hardware realistically

5. Facilitate synthesis, simulation and verification

6. Good spec ==> Good design
   Poor spec ==> ?

SPECIFICATION: Programming Language Features

1. Data types

Integer, boolean, arrays

2. Operators

Arithmetic, logic, manip, access

3. Control

If, case, repeat, decode

4. Conciseness

Macros, subroutines

5. Extensibility

Operator overloading, user definitions

6. Expressivity

Hardware constructs, constraints

7. Bindings, user annotations

Component/Time allocation and binding

SPECIFICATION: Design Features

1. Design model

Target architecture: DSP, uproc

2. Execution ordering

Sequential, parallel, pipeline

3. Hierarchy

Complex descriptions

4. Timing specification

Clocks, delays

5. Synchronization

Communication protocols

6. Asynchrony

Global events, resets

HDL FORMATS

Textual

- Programming languages
  - Pascal, Ada, ISPS, VHDL, Verilog
- Applicative
  - DSL
- Structural
  - EDIF, VHDL
- Formal
  - HOP, CIRCAL

Graphical

- Hierarchical FSM
  - StateCharts
- Petri−Nets
  - GDL
- FlowCharts
  - ASM, EXEL
- Waveforms

Tabular

- Symbolic MicroCode
- State Tables

TYPICAL HDL'S

ISPS (Barb 81)

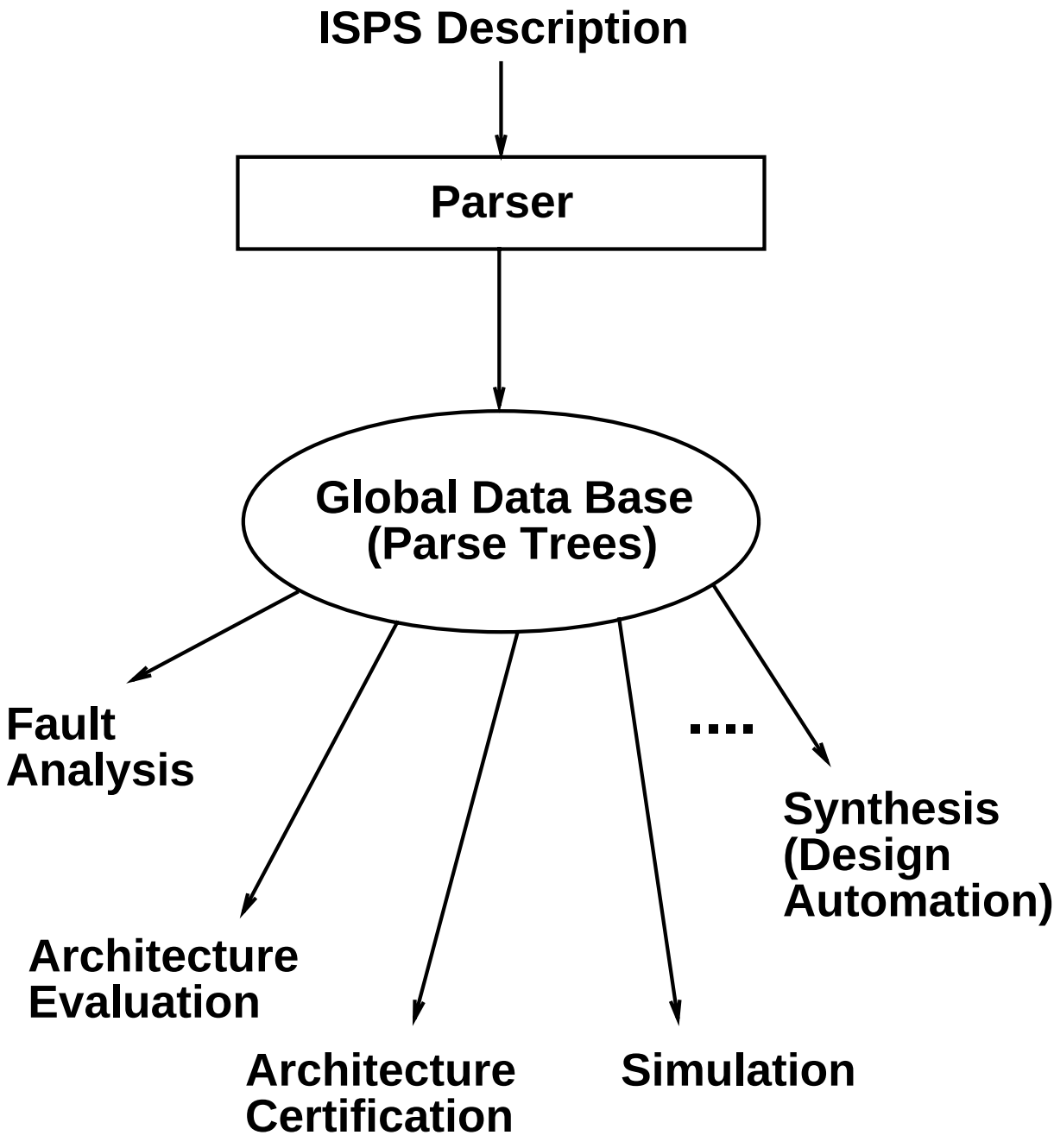
Hardware C (Ku DE 91)

Silage (Hilf 85)

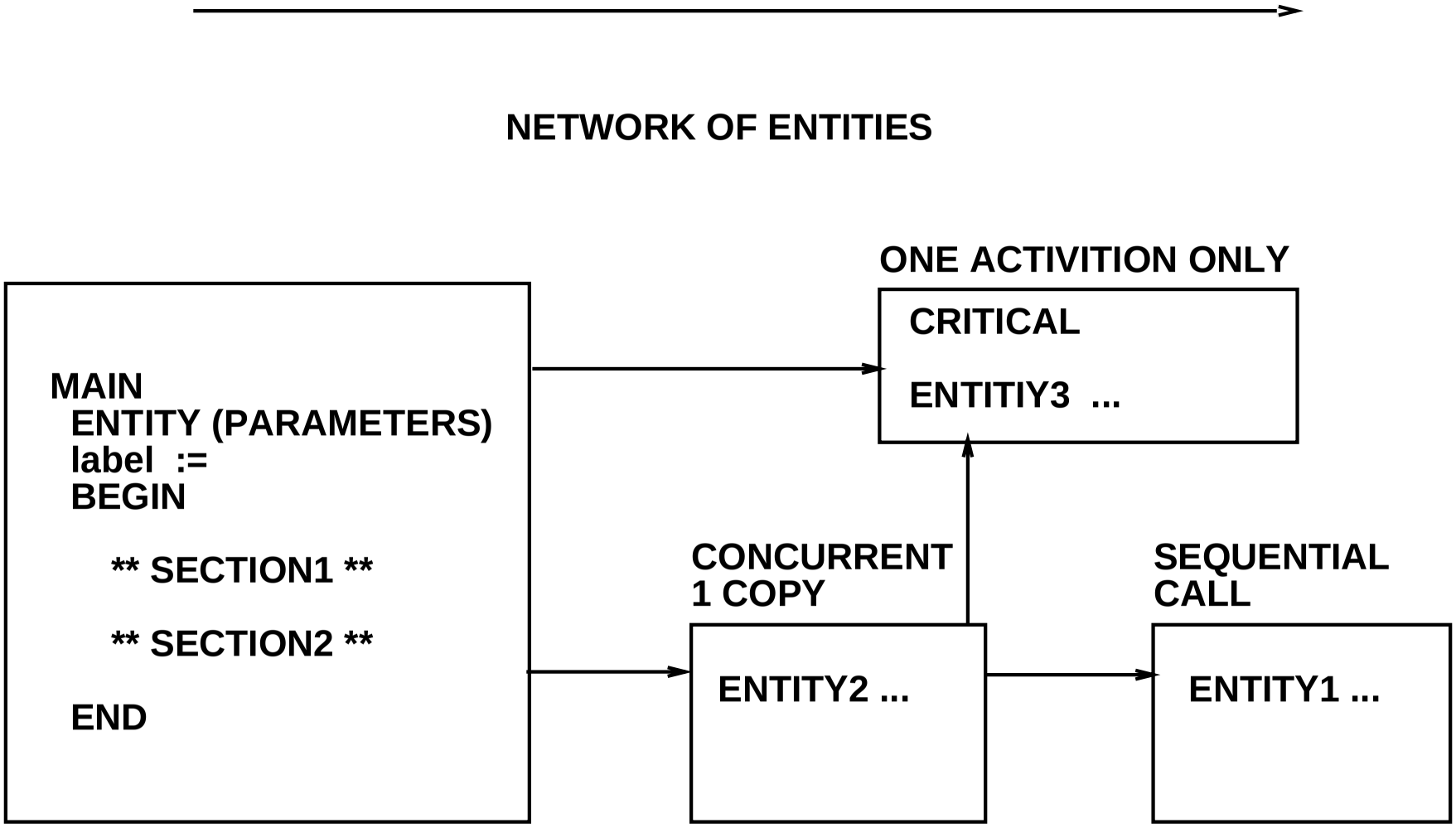
VHDL (IEEE 88)

Verilog (ThMo 91)

ISPS PARADIGM [Barb81]



ISPS MODEL



ISPS: FEATURES

Data types

constants

prefix	*meaning*	*example*
'	Boolean	'10?0
#	Octal	#17
{0−9}	Decimal	12
"	Hexadecimal	"A

bit−vectors & arrays

```
Acc\accumulator<0:31>
Mem\register.file[0:15]<0:31>
```

Control constructs

```
if x => PC = PC + 1
```

```
decode x =>
begin
  0 := Acc = 0,
  1 := Acc = Acc + 1,
end
```

```
Label := repeat
begin
  if (done) => leave Label
end
```

ISPS: FEATURES (cont'd)

Operation execution

All operations executed in parallel

Squentiality enforced by NEXT

```
PC = PC + 1;
ACC = 0;
next
IR = M[PC]
```

Concurrency

a PROCESS entity executes asynchronously

a CRITICAL entity is queued

Synchronization

```
process Master := begin
    ⋮
 nbsend (5) {Messg:Inp1};
 nbsend (1) {Messg:Start};

 L1 := repeat begin
    nbrecv(:Done) {Messg:Done};
    if (Done) => leave L1
 end;
 nbsend (0) {Mesg:Start};

    ⋮
end
```

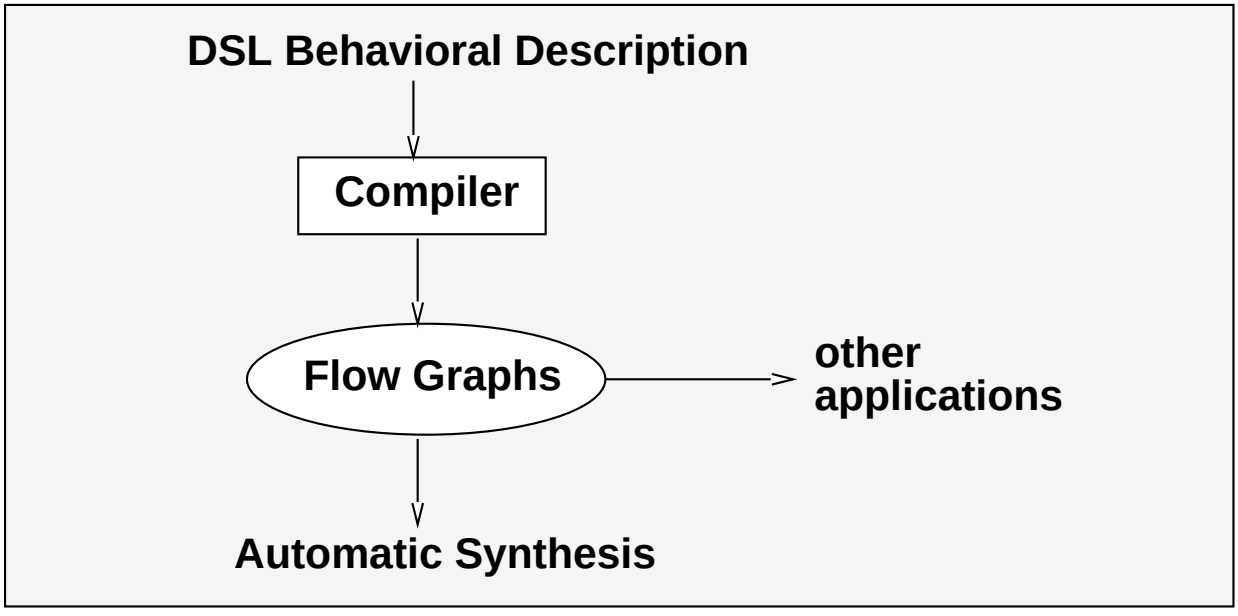
```
process Slave := begin
  L2 := repeat begin
     nbrecv(:Start)
     if (Start) => leave L2
  end;
  nbsend (0) {Messg:Done};
  nbrecv (:X) {Messg:Inp1};

      ⋮
  nbsend (1) {Messg:Done};

end
```

ISPS MARK 1 DESCRIPTION

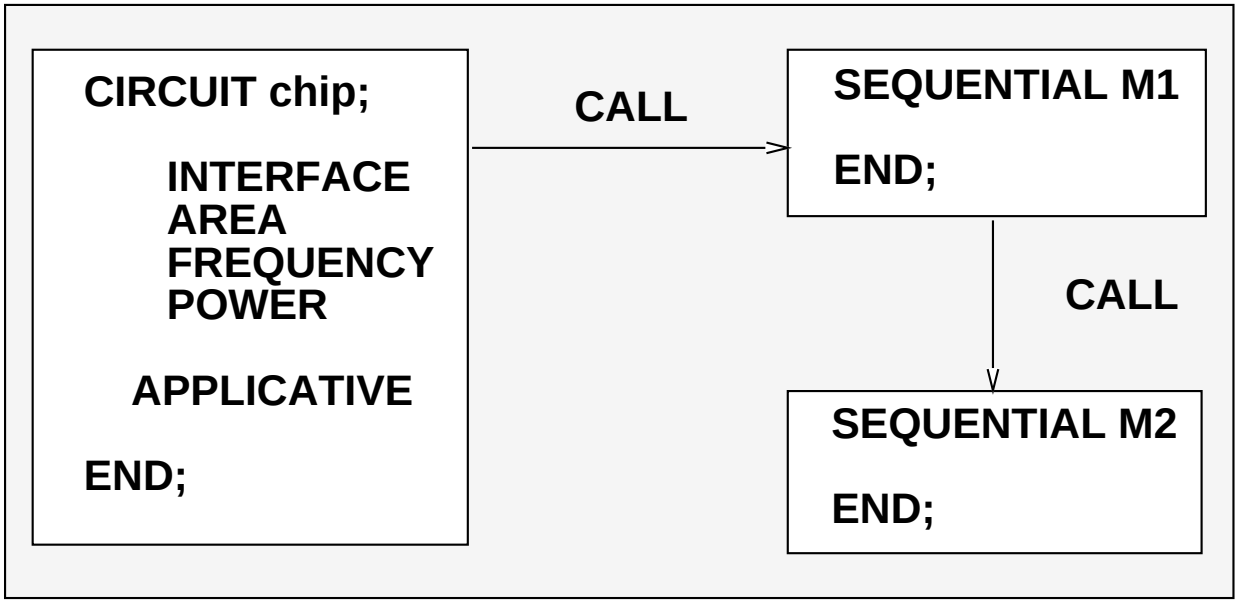
```
Mark1 :=
begin
**Memory.Primary.State**
      M[0:8191]<31:0>,
**Central.Processor.State**
      PI\Present.Instruction<0:15>,
      f\function = PI<0:2>,
      s<0:12> := PI<3:15>,
      Acc\Accumulator<0:31>,

**Instruction Execution**
  Icycle\Instruction.Cycle(main) :=
  begin
      repeat
        PI = M[CR]<0:15> next
        decode f =>
        begin
            #0  :=  CR = M[s]
            #1  :=  CR = CR + M[s]
            #2  :=  Acc = Acc − M[s]
            #3  :=  M[s] = Acc
            #4, #5  :=  Acc = Acc + M[s]
            #6  :=  if Acc < 0 =>
                        CR = CR + 1
            #7  :=  stop()
        end next
      CR = CR + 1
      end
  end
end
```

DSL: Paradigm and Model

Paradigm



Model



DSL FEATURES

Description styles

Applicative

Single assignment, concurrent
Global resets, interrupts

Imperative

Pascal−like procedures
Default sequential execution
Operation−level concurrency:
FORK A := B, C := D JOIN;

Delay specification

```
CLOCK CYCLES
        (A := A + 5;  B := 3  CYCLES < 4);

ABSOLUTE DELAY
        (IF  X  THEN  A := B  DELAY < 20);
```

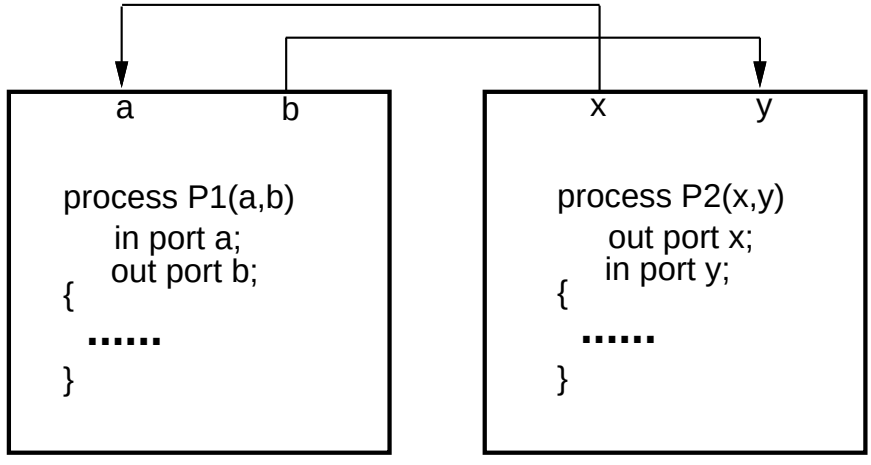
Chip−level constraints

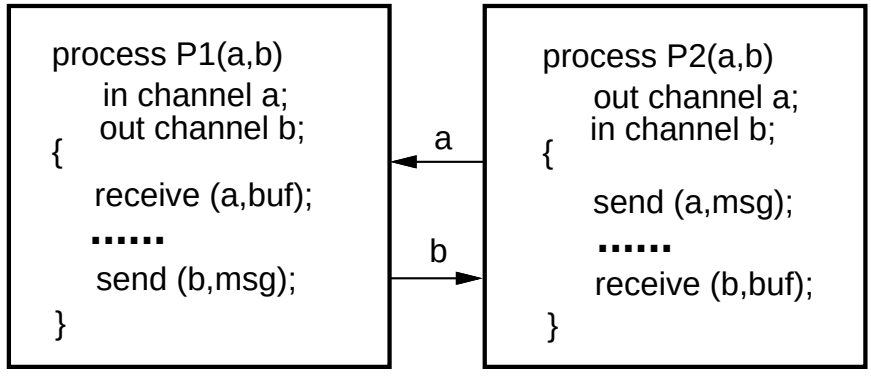
POWER
VOLTAGE
LEVELS
AREA
FREQUENCY

DSL SPECIFICATION (CaRo 85)

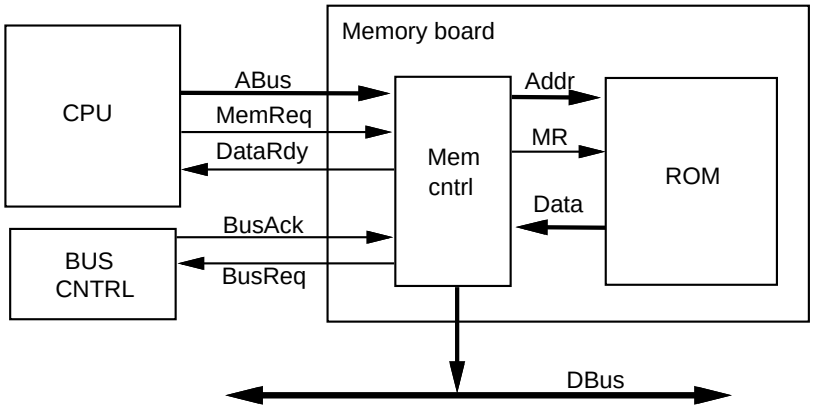
```
CIRCUIT  exponentiation;

INTERFACE
 vcc              :12v;
 gnd              :GND;
 input(15..0)     :INPUT FANIN 1;
 output(15..0)    :OUTPUT FANOUT 10;
 enable           :INPUT FANIN 1;
 clk              :CLOCK FANIN 1;

 POWER            100 mW;
 VOLTAGE          12.00V;
 TECHNOLOGY       CMOS;
 AREA             30 sq. mm;
 FREQUENCY        0 to 500 kHz;
 CLOCKBASE        clk;

 PERFORMED FUNCTION
   output := #exp(input);
   CONTROL
        (enable := 0  CYCLES = 1);
        (enable := 1  CYCLES = 48);
 END;

   VAR  x,y      :FIXED(8,8);
        I        :LOGICAL(4..0);

APPLICATIVE
    output := y;

    IF enable = 0 THEN x :=1, y := 1
                  ELSE Start calc;
    FI;
 END APPLICATIVE;

IMPERATIVE  calc;
    (FOR i := 1 to 16 DO
          x := x * input / i;
          y := y + x;
    OD  CYCLES = 3);
END IMPERATIVE;

END.
```

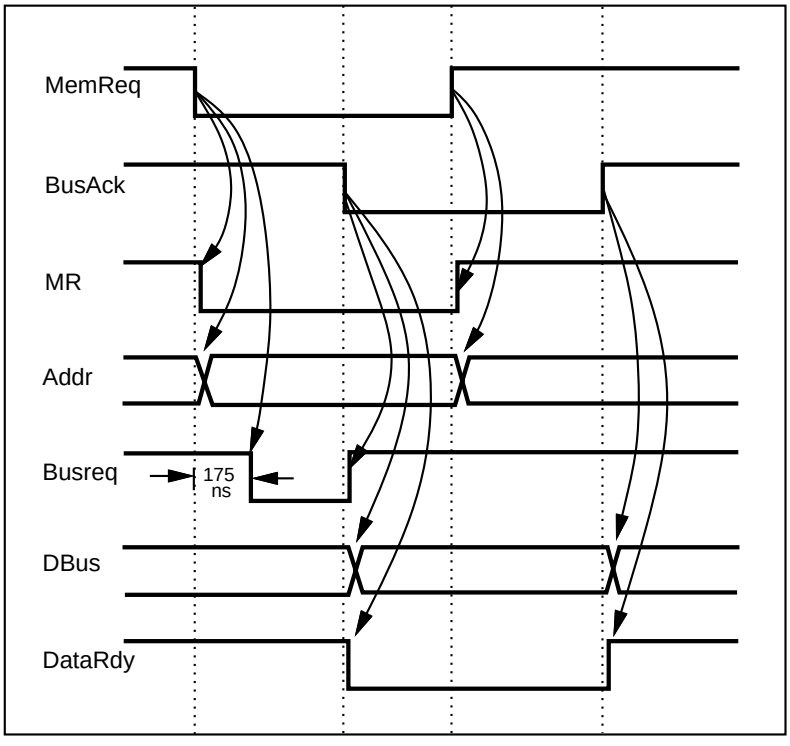
PROCESS SYNCHRONIZATION IN HARDWARE C



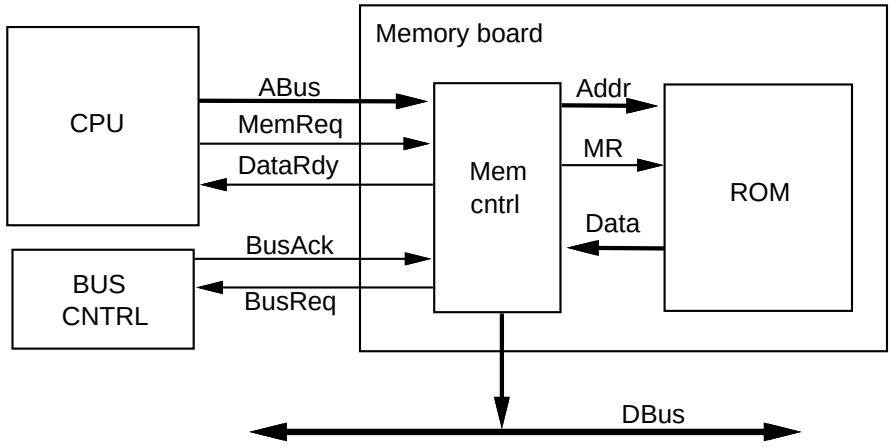


MEMORY BOARD DESIGN AND TIMING DIAGRAM





MEMORY BOARD DESCRIPTION IN BIF

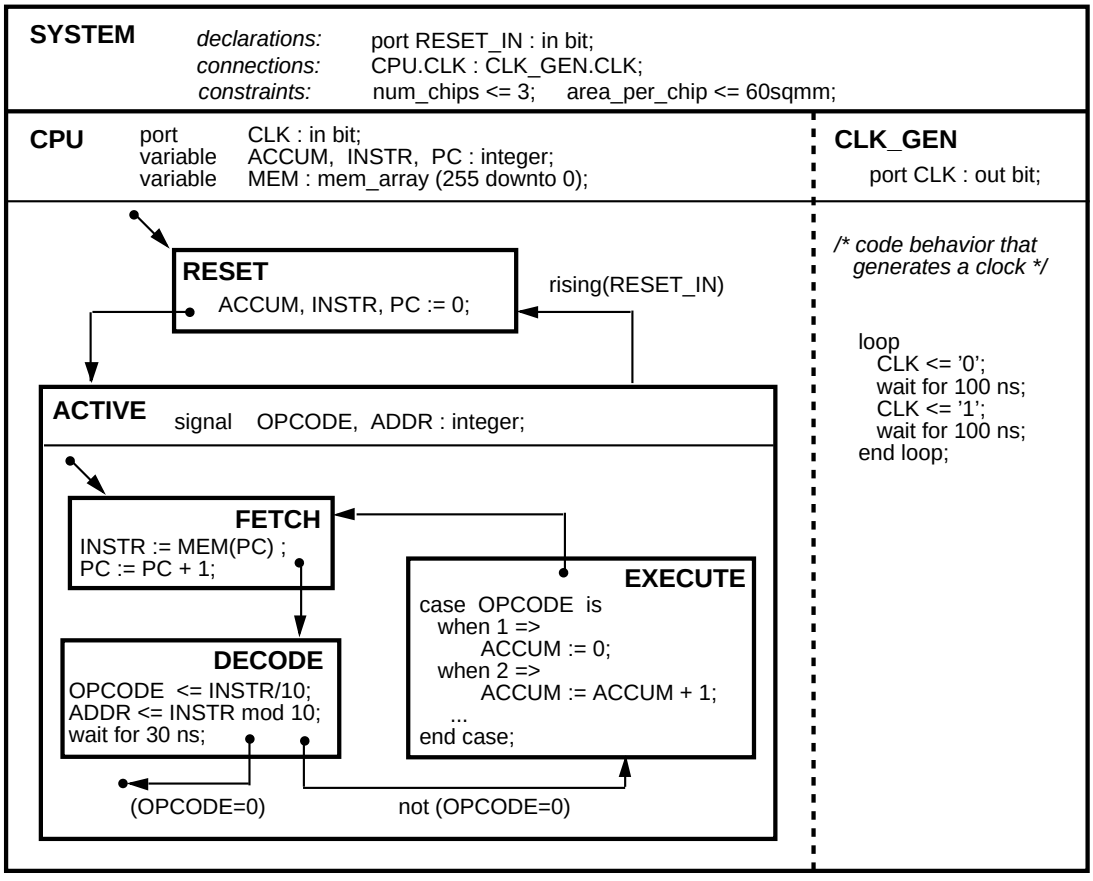


CPU−memory board block diagram

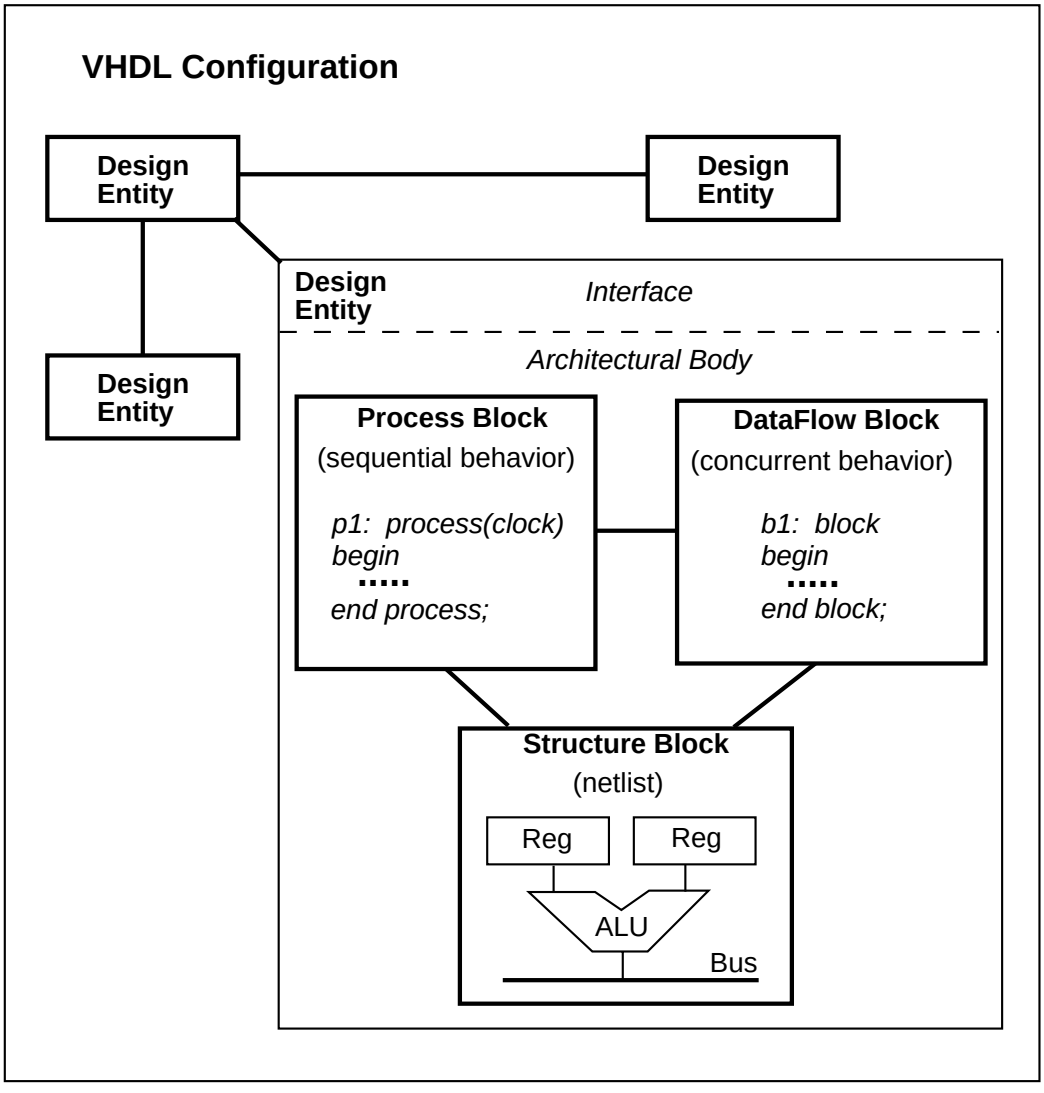
Present State	Cond	Val	Actions	Next State	Event
0		T	DBUS = 'X'; DataRdy = 1; BusReq = 1;	1	Falling(MemReq)
1	Abus {18..16} == Board_Id	T	MR = 0; Addr = ABus; BusReq|(delay 175ns) = 0;	2	Falling(BusAck)
		F		1	Falling(MemReq)
2		T	BusReq = 1; DBus = Data; DataRdy = 0;	3	Rising(MemReq)
3		T	MR = 1; Addr = 'X';	0	Rising(BusAck)

Memory board read cycle in BIF

BEHAVIORAL HIERARCHY IN SPEC CHARTS (NaVG 91)



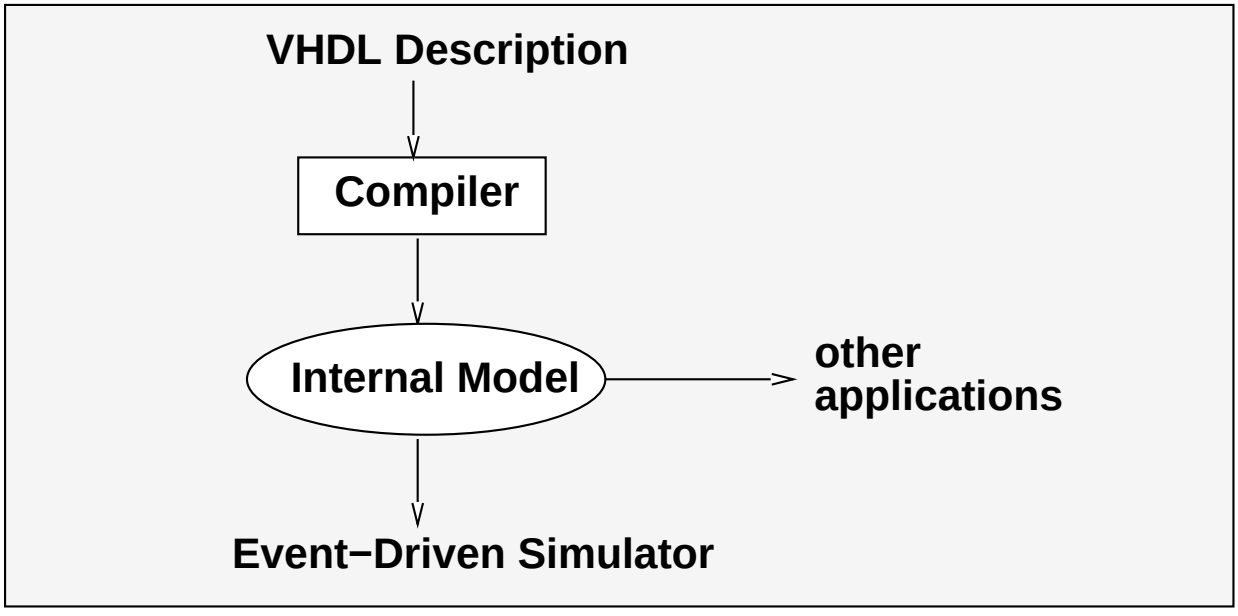
DESIGN HIERARCHY IN VHDL



VHDL [IEEE87]

VHDL Hardware Description Language

Paradigm



Simulation Language

Signals, Registers, Ports: containers w/ drivers

Driver Values Scheduled as Events in Simulation Time

Bus Resolution Function for Multiple Drivers

VHDL FEATURES

Strongly Typed Language

Operator Overloading

Concurrency

Process Level

Operation Level:

DataFlow Blocks

Timing Specification

Transport & Inertial Delays

A <= {TRANSPORT} B AFTER 5 ns

WAIT

WAIT FOR 20 ns

VHDL FEATURES

Resolution Function

Resolve multiple drivers for a signal

Packages

Definitions, Macros

Attributes

Signals:

S'STABLE

S'QUIET

User Extensions

Constraints, Annotations (not simulated)

attribute Performance : Integer

attribute LayoutSize : Integer

attribute Power : Integer

SEQUENTIAL AND PARALLEL EXECUTION IN VHDL

```
P1:  PROCESS (clock)
begin
     A  <=  B;
     B  <=  A;
end PROCESS P1;
```

```
B1:  BLOCK (clock'event  AND
                          clock = '1')
begin
     A  <=  guarded  B;
     B  <=  guarded  A;
end BLOCK B1;
```

VHDL DESCRIPTION STYLES

Behavior

- Abstract algorithm
- Sequential description of functionality
- No implied structural implementation

DataFlow

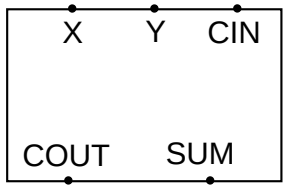
- Parallel execution of operations
- Data transformations, register transfers
- Hints at structural implementations

Structural

- Component instantiations, interconnections
- Netlist description

FULL ADDER: DATAFLOW DESCRIPTION

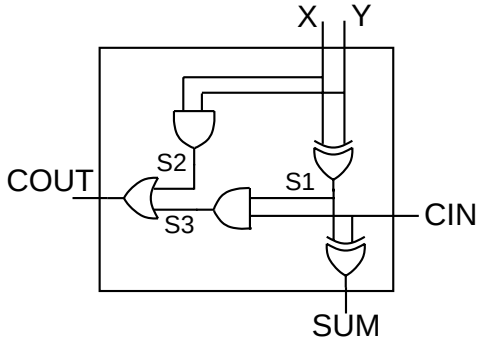
```
entity FULL_ADDER is
     port (X,Y:  in BIT;
           CIN:  in BIT;
           SUM:  out BIT;
          COUT:  out BIT );
end FULL_ADDER;
```



Entity interface description in VHDL

```
architecture FA_BOOLEAN of
                FULL_ADDER is
signal S1, S2, S3: BIT;
begin
    S1  <=  X xor Y;
    SUM  <=  S1 xor CIN  after 3 ns;
    S2  <=  X and Y;
    S3  <=  S1 and CIN;
    COUT  <=  S2 or S3  after 5 ns;
end;
```

Data flow

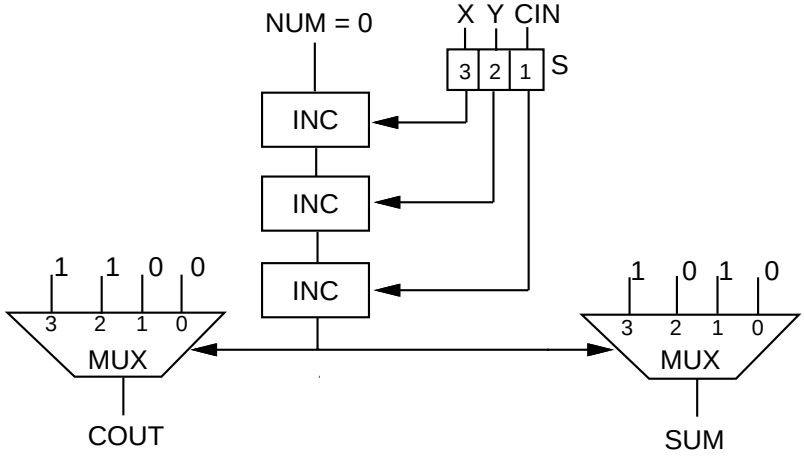


Synthesized structure

FULL ADDER: BEHAVIORAL DESCRIPTION

```
architecture FA_BEHAV  of
                FULL_ADDER is
begin
   process(X,Y,CIN)
     variable BV: BIT_VECTOR(1 to 3);
     variable NUM,I: INTEGER;
     variable Stemp, Ctemp: BIT;

   begin
     NUM := 0;
     BV := X & Y & CIN;
     for I := 1 to 3 loop
          if (BV(I) = '1') then
               NUM := NUM + 1;
          end if;
     end loop;

     case NUM is
          when 0 => Ctemp:='0'; Stemp:='0';
          when 1 => Ctemp:='0'; Stemp:='1';
          when 2 => Ctemp:='1'; Stemp:='0';
          when 3 => Ctemp:='1'; Stemp:='1';
     end case;
     SUM <= Stemp after 3 ns;
     COUT <= Ctemp after 5 ns;
   end process;
end FA_BEHAV;
```

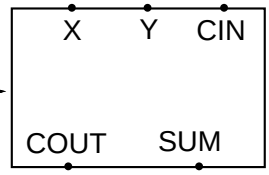
VHDL description



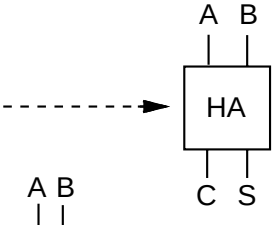
Synthesized full adder

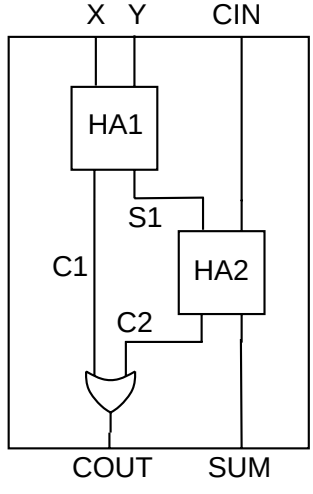
FULL ADDER: STRUCTURAL DESCRIPTION

```
entity FULL_ADDER is
    port (X,Y:  in BIT;
          CIN:  in BIT;
          SUM:  out BIT;
          COUT:  out BIT );
end FULL_ADDER;
```




```
architecture Structure_View  of
                    FULL_ADDER is

component Half_adder
    port (A,B: in BIT;
          S,C: out BIT);
end component;

component Or_gate
    port (A,B: in BIT;
          O: out BIT);
end component;

signal  C1, S1, C2:  BIT;

begin

HA1: Half_adder  port map
          (A=>X, B=>Y, S=>S1, C=>C1);

HA2: Half_adder  port map
          (A=>S1, B=>CIN, S=>SUM, C=>C2);

OR!: Or_gate  port map
          (A=>C1, B=>C2, O=>COUT);

end;
```





MODELING

1. Language and architecture matching
2. General languages induce modeling styles
3. Description disambiguation and design optimization
4. Real vs simulated delays
5. Language constructs with no hardware realization
6. Modeling guidelines

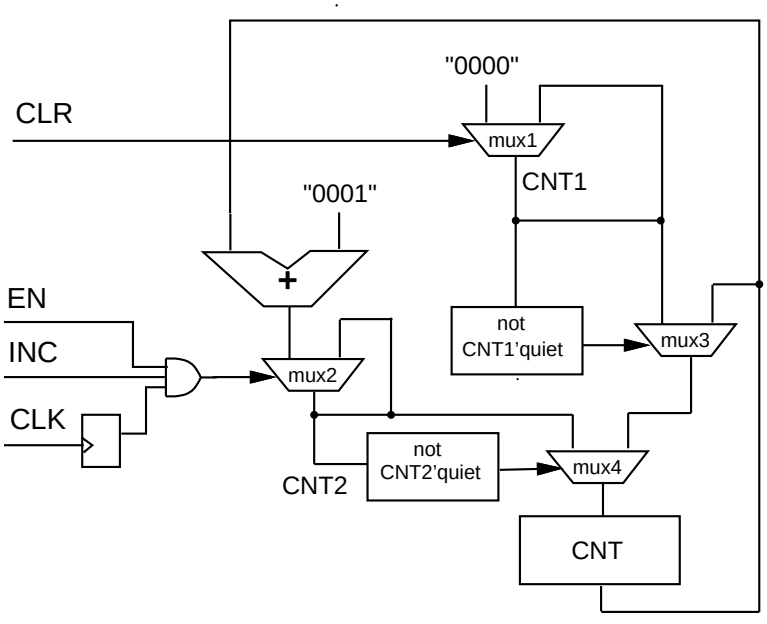
LANGUAGE INDUCED UNNECESSARY HARDWARE

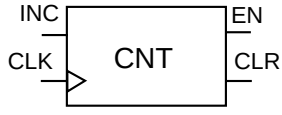
```
CNT_CLR: block(CLR = '1')
begin
    CNT1 <= guarded B"0000"  after CLRDEL;
end block;
```

```
CNT_UP: block(EN = '1'  and  CLK = '1'
                    and  CLK'event  and  INC = '1')
begin
    CNT2 <= guarded CNT + B"0001" after INCDEL;
end block;
```

```
SEL:  CNT <=  CNT1  when not CNT1'quiet  else
              CNT2  when not  CNT2'quiet  else
              CNT;
```

VHDL counter description





Initial hardware synthesized

RT counter

VHDL SYNTHESIS PROBLEMS

Identification of storage elements, signals

Language constructs with no realizable hardware

Collecting, identifying component attributes

Specification of asynchronous events

Use of multiple blocks/processes to describe one component

MODELING GUIDELINES

1. Matching of semantic models to architectural models

(a) specialized languages
(b) modeling styles (structured properties)

2. Combinatorial style

(a) concurrent execution semantics
(b) connection of logic gates
(c) no clocks
(d) dataflow VHDL constructs

3. Functional style

(a) one−state FSMD
(b) synchronous and asynchronous behavior
(c) signal typing
(d) dataflow block VHDL constructs

4. Register−transfer style

(a) FSMD model
(b) states, condition, actions
(c) no explicit VHDL constructs

5. Behavioral style

(a) communicating processes
(b) shared memory or message passing
(c) no allocation, no binding, no schedule
(d) VHDL process statements, wait statements

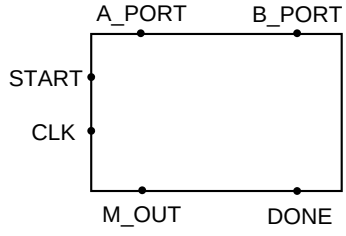
VHDL FUNCTIONAL DESCRIPTION STYLE

```
CNT_UP_CLR: block(  CLR = '1'
                        or (EN = '1'  and  CLK'event  and  CLK = '1'))
begin
   CNT  <=  guarded
                B"0000" after CLRDEL   when CLR='1'  else
                CNT + B"0001" after INCDEL  when INC='1' else
                CNT;
end block;
```

VHDL RT DESCRIPTION STYLE

```
State_Fetch:  block ( (CLK'event and CLK='1')  and  (state=S0))
begin
        IR <= M(PC);
        state <= S1;
end block;

State_Decode:  block ( (CLK'event and CLK='1')  and  (state=S1))
begin  case  IR  is
              when "0000"  =>    ACC <= ACC + 1;
                                 state <= S2;
              when "0001"  =>    ACC <= 0;
                                 state <= S3;
              .....

        end case;
end block;
```

BEHAVIORAL DESCRIPTION STYLE




```
entity MULT is
 port ( A_PORT,
          B_PORT:  in bit_vector(3 downto 0);
          M_OUT:  out bit_vector(7 downto 0);
          CLK:  in CLOCK;
          START:  in BIT;
          DONE:  out BIT;
        );
end MULT;
```

```
architecture SHIFT_MULT  of  MULT  is
begin
      process
          variable A, B, M  :  BIT_VECTOR;
          variable COUNT  :  INTEGER;
      begin
          wait until (START = 1);
          A := A_PORT;   COUNT := 0;
          B := B_PORT;   DONE <= '0';
          M := B"0000";
          while  (COUNT < 4)  loop
              if (A(0) = '1')  then
                        M  :=  M + B;
              end if;
              A := SHR(A, M(0));
              M := SHR(M, '0');
              COUNT := COUNT + 1;
          end loop;
          M_OUT  <=  M & A;
          DONE <= '1';
      end process;
end SHIFT_MULT;
```

CURRENT ISSUES

Raise abstraction level

User−interaction / annotation

Design frameworks

Unified design representation

Constraint specification & representation

Design verification

FUTURE DIRECTIONS

Specialized languages

Variety of Intermediate forms

Architectural taxonomies

Modeling guidelines

Design scenarios