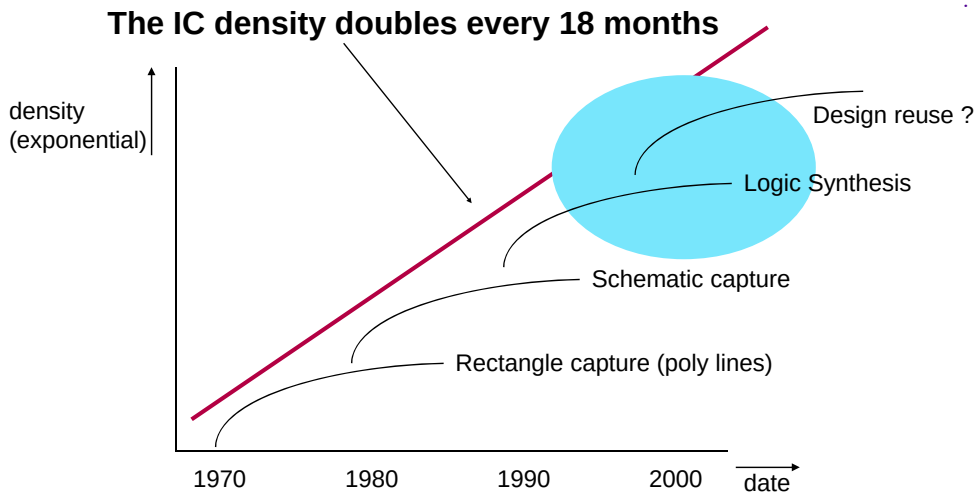
VHDL, an Introduction

Egbert Molenkamp
Department of Electrical Engineering, Mathematics and Computer Science
University of Twente
PO Box 217
7500 AE Enschede
the Netherlands
email: e.molenkamp@utwente.nl

Contents

Moore's Law



Janick Bergeron

Why was VHDL developed (1981) ?

- **Each company uses its own 'language'**
  - **portability problems.**
  - **DoD got different descriptions for their VHSIC (Very High Speed Integrated Circuit) program.**
- **Reusability not easy**
- **Tools from logical level until realization available, but a lack of high level description tools**

History van VHDL

- June 1981: Woods Hole *Summer Study on Hardware Description Languages* Workshop
- Aug. 1983: Start of VHDL project
- Aug. 1985: VHDL version 7.2 (DoD)
- March 1986: IEEE starts standardization
- Dec. 1987: Std. 1076-1987
- June 1990: Preparation reballoting (P1076-1992/A)
- Sept. 1993: Std. 1076-1993
- March 2002: Std. 1076-2002 (major goal: 'bug fix')
- Sept. 2008: Std. 1076-2008
- ..

VHSIC Hardware Description Language

Properties of VHDL

- Hierarchy supported
- Support of different design flows
  - top-down
  - bottom-up
  - mixed
- Technology independent
- Synchrone / asynchrone models
- Different description styles supported
  - finite state machine
  - algorithms
  - boolean equations

Properties of VHDL /2

- Second-sourcing easy
  - CAD /CAE Tool independent
- Higher level of abstraction
  - Handling complexity
  - Suitable for behavioral until logical level
  - Synthesis tools for the realization
- Early capture of design faults
- LSI modeling easy, due to ‘components’, ‘functions’, ‘procedures’ and ‘packages’

Properties of VHDL /3

- No restrictions on the size
- ‘test benches’ in the same language > portable!
- Due to ‘generics’ backannotation is easy
- Strongly typed language. User defined types possible.

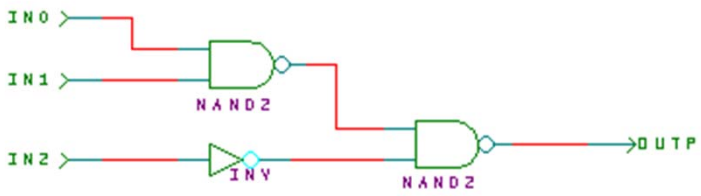
Demanded by the DoD
An IEEE and ANSI standard

Current use of VHDL

- **Modelling**
- **Design flow**
  - **Designer writes VHDL**
  - **Intermediate format (VHDL automatically generated)**

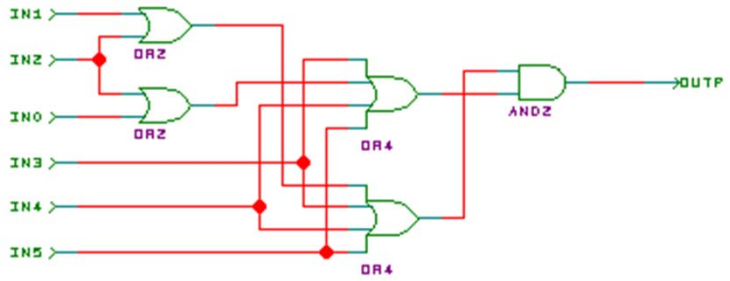
Synthesis, the mapping to hardware,
is **not** part of the VHDL standard.

schematic capture



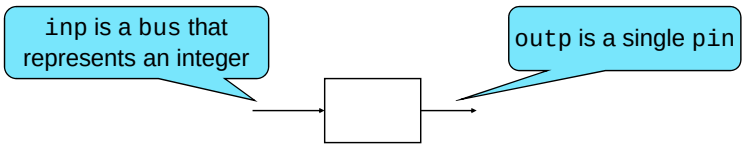
What is the behavior of this circuit ?

schematic capture /2



What is the behavior of this circuit ?

Textual approach

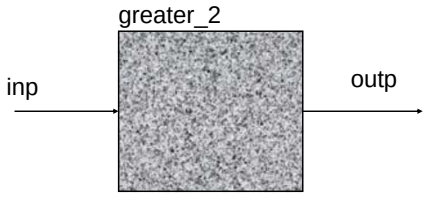



```
outp <= '1' WHEN inp > 2 ELSE
        '0';
```

a VHDL statement

What is the behavior of this “circuit” ?

Interface: the ENTITY




```
-- author: ....this line is comment
ENTITY greater_2 IS
 PORT ( inp  : IN bit_vector(2 DOWNTO 0);
        outp : OUT bit

        );
END greater_2;
```

Alternatives for the last line:

```
-END;
-END ENTITY greater_2;
```


Body: the ARCHITECTURE

Name of the ENTITY

```
ARCHITECTURE behavior OF greater_2 IS
BEGIN
  outp <= '1' WHEN inp > 2 ELSE
          '0';
END behavior;
```

Not really correct because bit_vector has no integer representation. This will be solved later

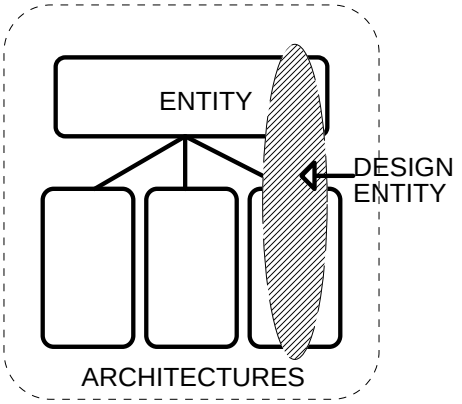
Multiple architectures for the same ENTITY are allowed.

alternative architecture with the same behavior

```
ARCHITECTURE gates OF greater_2 IS
BEGIN
  outp <= inp(2) OR
          (inp(1) AND inp(0));
END gates;
```

"design units" of VHDL

- **Entity**
  - **Architecture**
-
-



Model in VHDL

ARCHITECTURE

```
ARCHITECTURE architecture_name OF entity_name IS….
    CONSTANT level : integer := 2;
    SIGNAL a,b,c : bit := '0';      -- initial value is optional
    FUNCTION, PROCEDURE, COMPONENT declarations
BEGIN
    a <= in1 AND in2;

    b <= in1 OR in2;       -- sensitivity list is implicit

    PROCESS  (a,b)      -- sensitivity list
      VARIABLE loc : bit;
    BEGIN
      loc := a XOR b;
      c <= NOT loc;
    END PROCESS;
END architecture_name;
```

Declarations (for local use only!)

three concurrent statements

Sequential statements, also if *condition* then <else> end if; case, iteration schemes (while, for loop) ..

Some sequential statements

```
IF a>b THEN
  z <= a;
ELSIF a > c THEN
  z <= b;
  y <= a;
ELSE
  z <= c;
END IF;
```

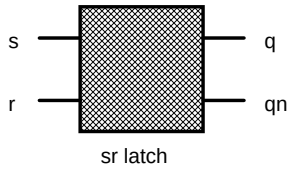
```
CASE inp IS
  WHEN 0          => statement(s)
  WHEN 1 | 4      => statement(s)
  WHEN OTHERS => statement(s)
END CASE;
```

```
FOR i IN 0 TO 100 LOOP
  y := i * z;
  EXIT WHEN y >= 10;
END LOOP;
```

```
WAIT UNTIL clk='1';
```

Wait until rising edge of signal clk

SR Latch



sr latch

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY sr_latch IS
  PORT (s, r    : IN    std_ulogic;
        q, qn : OUT std_ulogic);
END sr_latch;
```

Logical levels in VHDL

Standard:

- In library Std type bit.
- In library IEEE type std_ulogic.

MVL9

```
TYPE std_ulogic IS
 ( 'U',     -- Uninitialized
   'X',     -- Forcing  Unknown
   '0',     -- Forcing  0
   '1',     -- Forcing  1
   'Z',     -- High Impedance
   'W',     -- Weak    Unknown
   'L',     -- Weak    0
   'H',     -- Weak    1
    '-'     -- Don't care
 );
```

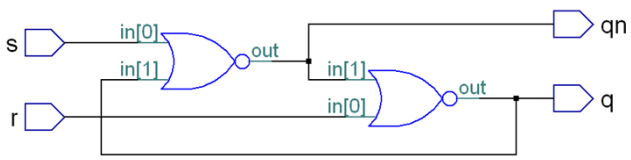
MVL2

```
TYPE bit IS
 ( '0',  -- Forcing  0
   '1'   -- Forcing  1
 );
```


SR latch, data flow; (parallelism)

```
ARCHITECTURE dataflow OF sr_latch IS
  SIGNAL qi, qni : std_ulogic;
BEGIN
  qi  <= r NOR qni AFTER 5 ns;
  qni <= s NOR qi AFTER 3 ns;
  q   <= qi;
  qn  <= qni;
END dataflow;
```

not sequential



Input/output modes

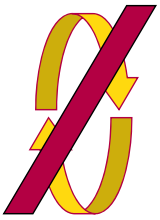
- IN — only reading of port is allowed
- OUT — only writing to port is allowed
- INOUT — reading of and writing to the port is allowed
- BUFFER, LINKAGE — Not part of this course

Do not use mode *inout* if it is not a bidirectional port

SR Latch, structure; (netlist)

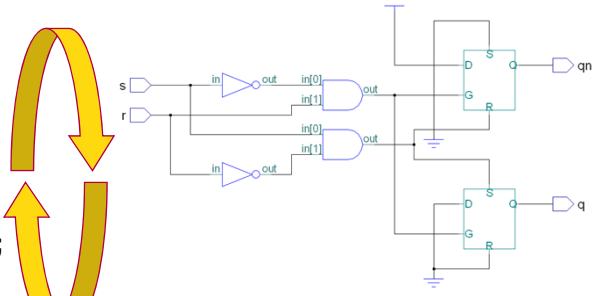
```
ARCHITECTURE structure OF sr_latch IS
  COMPONENT  nor2 IS
    PORT (a,b : IN std_ulogic;
          y   : OUT std_ulogic);
  END COMPONENT;
  SIGNAL qi, qni : std_ulogic;
BEGIN
  n1 : nor2 PORT MAP(r,qni,qi);
  n2 : nor2 PORT MAP(s,qi,qni);
  q  <= qi;
  qn <= qni;
END structure;
```

Tip for component declaration:
- Copy corresponding entity declaration
- Change entity name at the end in “COMPONENT”
- Keyword “IS” allowed since VHDL’93)



SR Latch, behaviour; (sequential!)

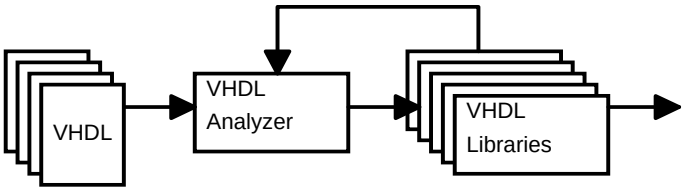
```
ARCHITECTURE behaviour OF sr_latch IS
BEGIN
   PROCESS(s,r)
   BEGIN
      IF s='1' AND r='0'
         THEN q <='1'; qn<='0';
      ELSIF s='0' AND r='1'
         THEN q <='0'; qn<= '1';
      ELSIF s='1' AND r='1'
         THEN q <= '-'; qn <= '-';
      END IF;
   END PROCESS;
END behaviour;
```



Description styles for architectures

- **Structure**
  - **Component declarations**
  - **Component instantiations**
    - A “netlist” description!
    - Error sensitive
    - Use with care
- **Data flow**
  - **Concurrent signal assignment statements**
    - Parallelism of hardware
- **Behaviour**
  - **Process description**
    - Sequential description of the behaviour
    - Simple!
- **Mixture allowed**

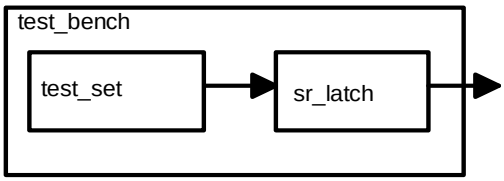
VHDL Analysis



Analysis - Compilation without code generation:

- **Input: *design file* containing one or more *design units***
- **Output: one *library unit* per *design unit***
- **Libraries:**
  - **Resource Libraries: Std, IEEE, ...**
  - **A single working library: *Work***

Test Bench



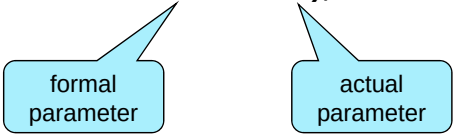

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY test_bench IS

  PORT (q,qn : OUT std_ulogic);

END test_bench;
```

Test Bench /2

```
ARCHITECTURE structure OF test_bench IS
 COMPONENT sr_latch IS
  PORT (s, r    : IN  std_ulogic;
        q, qn : OUT std_ulogic);
 END COMPONENT;
 COMPONENT test_set IS
  PORT (set, reset : OUT std_ulogic := '0');
 END COMPONENT;
 SIGNAL si, ri : std_ulogic;
BEGIN
 latch:sr_latch PORT MAP(si,ri,q,qn);
 testb:test_set PORT MAP(si,ri);
END structure;
```

Association list

- **Positional association:**
  - testb:test_set PORT MAP(si,ri);
- **Named association:**
  - testb:test_set PORT MAP(reset => ri,
    set => si);



Test Bench /3

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY test_set IS
  PORT (set, reset : OUT std_ulogic := '0');
END test_set;
```

```
ARCHITECTURE TestSet1 OF test_set IS
  SIGNAL rs : std_ulogic_vector(1 DOWNTO 0) := "00";
BEGIN
  rs <= "01" AFTER 50 ns,
        "00" AFTER 100 ns,
        "10" AFTER 150 ns,
        "00" AFTER 200 ns;
  reset <= rs(1);
  set <= rs(0);
END TestSet1;
```

Test Bench /4

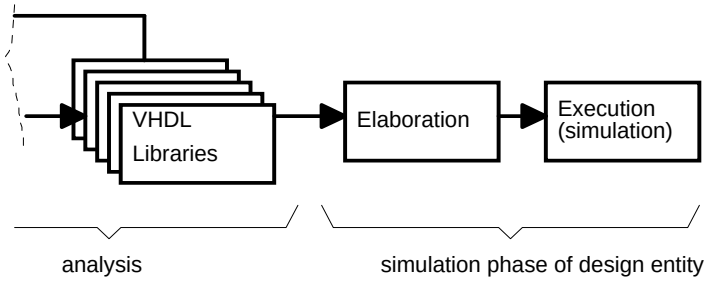
Still problems left:

- **Which *architecture* is tested?**
- **Manual verification of the simulation results is error prone.**

content of library WORK
<entity> (<architecture>)

- sr_latch (dataflow)
- sr_latch (structure)
- sr_latch (behaviour)
- test_bench (structure)
- test_set (TestSet1)

Simulation



Elaboration

- **depends on the selected design entity to simulate/synthesize**
- **signals are created**
  signals are not created/removed in the execution phase
- **components instantiated**
  design entities are bound
- **initial values are assigned to objects**

SR latch, order of statements

```
ARCHITECTURE exa1 OF sr_latch IS
  SIGNAL qi, qni : std_ulogic;
BEGIN
  qi  <= r NOR qni;
  qni <= s NOR qi;
  q   <= qi;
  qn  <= qni;
END exa1;
```

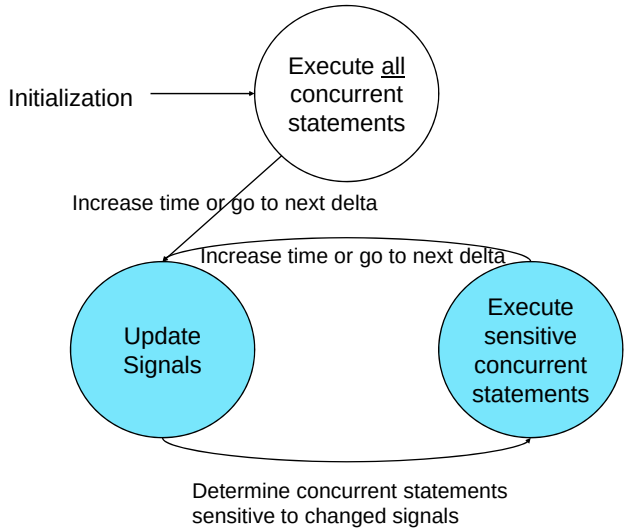
```
ARCHITECTURE exa2 OF sr_latch IS
  SIGNAL qi, qni : std_ulogic;
BEGIN
  qni <= s NOR qi;
  qi  <= r NOR qni;
  q   <= qi;
  qn  <= qni;
END exa2;
```

Delta delay

***Delta delay* makes:**

- **Order independent descriptions possible**
- **Ensures that a circuit is stable before time advances**
  - **Any number of deltas < 1 fs**
  - **Delta delayed oscillators possible**
    No progress in time, hence also waveform display does not show any progress
  - **Most tools can set an upper limit on the number deltas**

Simulation cycle



Simulation cycle /2

A: qi <= r NOR qni;

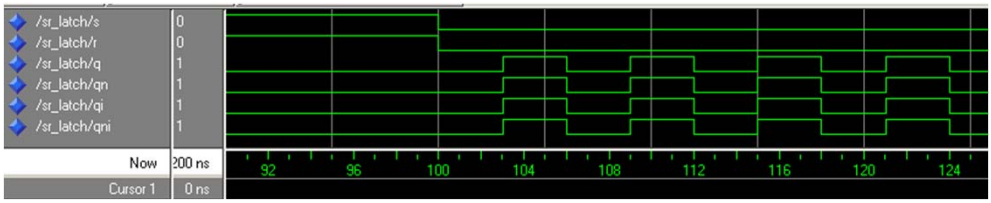
B: qn <= qni;

C: qni <= s NOR qi;

D: q <= qi;

time	s	r	qi	qni	q	qn
t1	1	1	0	0	0	0
t2	0	0	0	0	0	0
	processes A and C are sensitive					
t2+1δ	0	0	1	1	0	0
	processes A, B, C and D are sensitive					
t2+2δ	0	0	0	0	1	1
t2+3δ	0	0	1	1	0	0
t2+4δ	0	0	0	0	1	1

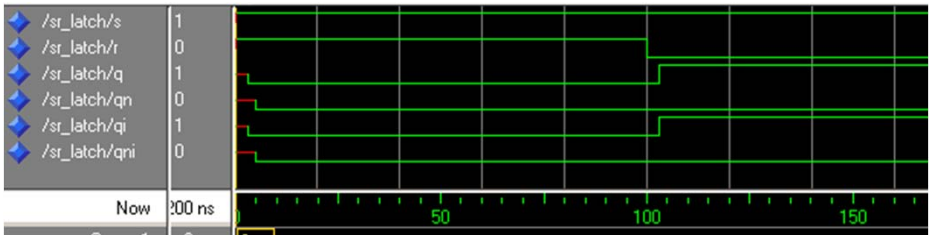
Simulation cycle /3



qi <= r NOR qni AFTER 3 ns;

qni <= s NOR qi AFTER 3 ns;

Simulation cycle /4




```
qi   <= r NOR qni AFTER 3 ns;

qni <= s NOR qi AFTER 5 ns;
```

Variables, Signals, and Constants

variable is updated immediately

signal is updated **after** a DELTA delay, hence the next statements will use the 'old' value!

sig1 is not yet updated here!

```
label_optional:PROCESS
  VARIABLE var : std_ulogic;
  CONSTANT high : std_ulogic := '1';
BEGIN
  var := a NOR b;
  sig1 <= var AND high;
  sig2 <= NOT sig1;
  WAIT ON a,b,sig1;
END PROCESS label_optional;
```

Remember this!

Assignments to

- variables:
  - := as assignment operator
  - always immediately
- signals
  - <= as assignment operator
  - at least after a *delta delay*

Simulation cycle /3

Which behavior is exactly the same?

(The signals *inp* and *outp* are the only signals in the port declaration of the entity. Other signals are declared locally.)

```
A:PROCESS (inp)
  VARIABLE v : bit;
BEGIN
  v := inp;
  outp <= v;
END PROCESS;
```

```
B:PROCESS (inp)
BEGIN
  s <= inp;
  outp <= s;
END PROCESS;
```

```
C:PROCESS (inp,s)
BEGIN
 outp <= s;
 s <= inp;
END PROCESS;
```

```
D: outp <= inp;
```

```
E:PROCESS (inp)
  VARIABLE v : bit;
BEGIN
  outp <= v;
  v :=inp;
END PROCESS;
```

```
F:PROCESS (inp,s)
BEGIN
  s <= inp;
  outp <= s;
END PROCESS;
```

Summary: VHDL is Concurrent and Sequential

- Statements in architecture are *concurrently* executed
  - The ordering in the design file is not important.
  - Concurrent statements communicate via *signals.*
  - Signals are declared at process level (not in a process).
  - An assignment to a signal will at least update the value of that signal after one delta delay.
  - Communication is also possible via *shared variables* (Std. 1076-1993). But not portable anymore!
- A process is internally executed *sequentially*
  - Variables may only be declared in sequential part, e.g. in the declaration region of a process.
  - An assignment to a variable always immediately updates the value.