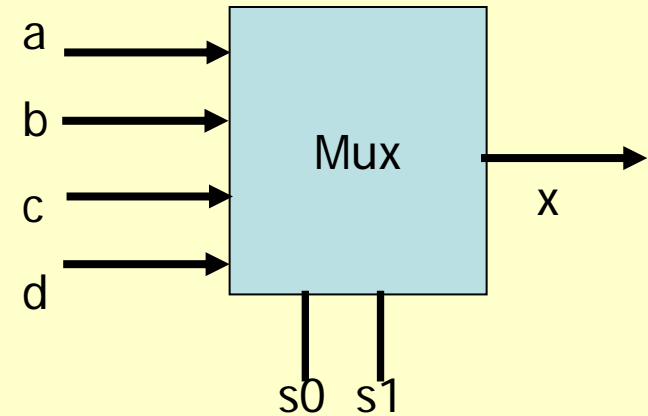


VHDL Front end

Entity, Architecture and Process

```
entity mux is
  port( a, b, c, d : in bit;
        s0, s1: in bit;
        x: out bit);
end mux;
```



```
architecture arch_mux of mux is
begin
  process (a, b, c, d, s0, s1)
  variable sel: std_logic_vector(1 downto 0);
  begin
    if s0='0' and s1= '0' then
      sel<="00";
    elsif s0='1' and s1= '0' then
      sel<="01";
    elsif s0='0' and s1= '1' then
      sel<="10";
```

```
    else sel<="11";
    end if;
    case sel is
    when "00" => x<= a;
    when "10" => x<= c;
    when "01" => x<= b;
    when others => x<= d;
    end case;
  end process;
end arch_mux;
```

Behavioral & Structural coding styles

architecture structural_mux of mux is

signal s0_inv, s1_inv, x1, x2, x3, x4 : bit;

```
component andgate
port ( a, b, c: in bit; d: out bit);
d=>x2);
end component;
```

```
component inverter
port ( in1: in bit; x: out bit);
end component;
```

```
component orgate
port ( a, b, c, d : in bit; x: out bit);
end component;
```

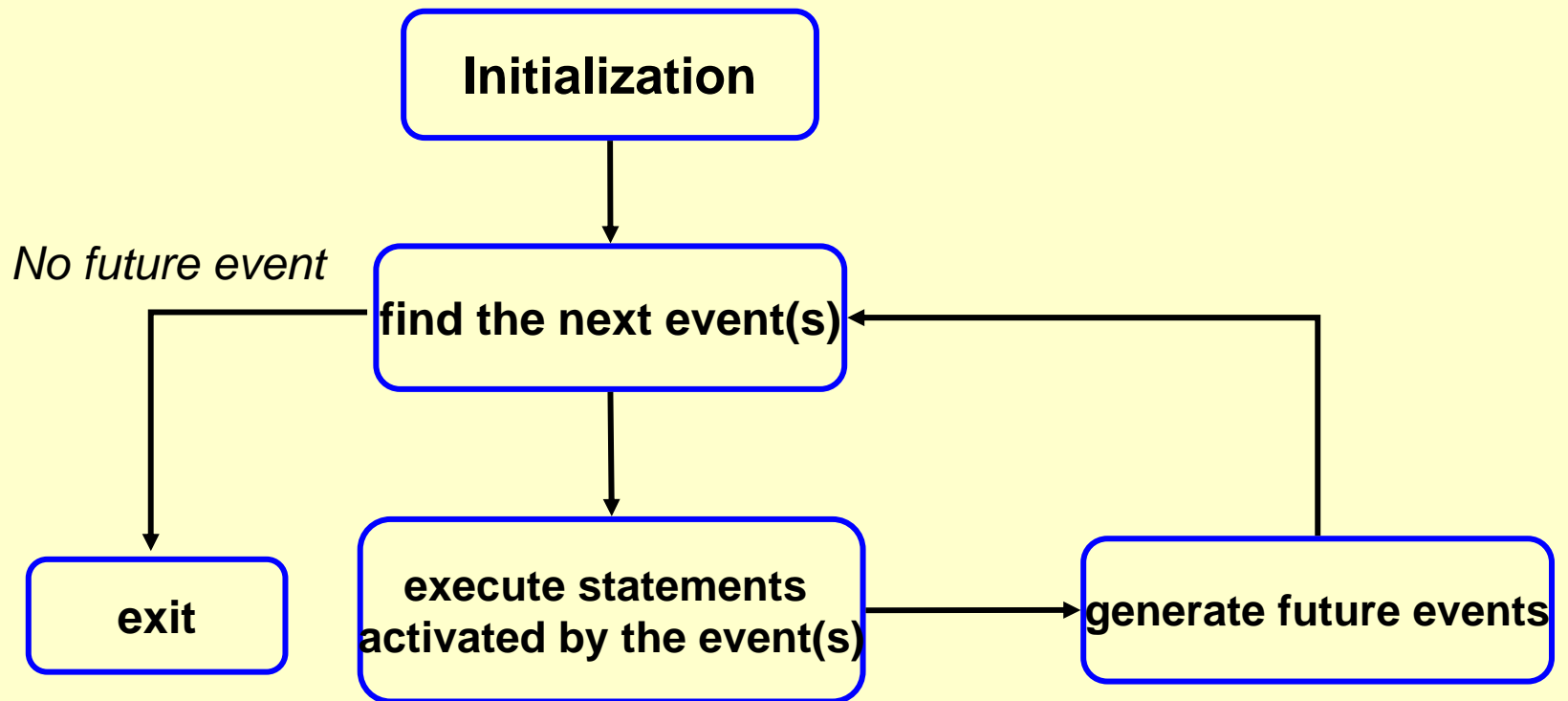
```
begin
U1: inverter port map (in1 =>s0,
x=>s0_inv);
U2: inverter (in1=>s1, x=>s1_inv);
U3: andgate (a=>a, b=>s0_inv,
c=>s1_inv, d=>x1);
U4: andgate (a=>b, b=>s0, c=>s1_inv,

U5: andgate (a=>c, b=>s0_inv, c=>s1,
d=>x3);
U6: andgate (a=>d, b=>s0, c=>s1,
d=>x4);
U7: orgate (a=>a, b=>b, c=>c, d=>d,
x=>x);
end structural_mux;
```

Event-driven Simulation

- Event: occurrence of a signal change (transaction)
- When an event occurs, need to handle it (execution)
 - execute any concurrent statements that are sensitive to the change
 - activate a waiting process if the condition becomes true
- The execution triggered by an event may generate future events
 - `sig_a <= a_in after 5 ns;`

Event-driven Simulation



Wait Statements

wait_stmt <=

**[label :] wait [on signal_name{ , ... }]
[until boolean_expr]
[for time_expr] ;**

- **wait;**
- **wait on a, b, c;**
- **wait until x = 1;**
- **wait for 100 ns;**

Wait on

- process being suspended until an event takes place on any one of the signals.
- The list of signals is also called a sensitivity list.

```
half_adder: process
is begin
    s <= a xor b after 10 ns;
    c <= a and b after 10 ns;
    wait on a, b;
end process;
```

```
half_adder: process (a, b)
is begin
    s <= a xor b after 10 ns;
    c <= a and b after 10 ns;
end process;
```

Wait until

wait on s1, s2, s3 until *condition*;

- Condition evaluated only when an event occurs on a signal in the sensitivity list
- Process is resumed when the condition evaluates to TRUE.

Example Use of Multiple Wait Statements: CPU and Memory Handshaking

Memory: process is

begin

DAV <= '0';

wait until Mem_Req = '1';

Data <= ROM_DATA(Address) after 50 ns;

DAV <= '1' after 60 ns;

wait until Mem_Req = '0';

end process;

CPU_Read: process is

begin

Mem_Req <= '0';

wait until ... the need for memory read ;

Address <= . . . address value . . .

Mem_Req <= '1' after 10 ns;

wait until DAV = '1';

MD_Reg <= Data;

end process;

label: process (a, b, c, d) is

where signals a, b, c, d are the sensitivity list

- equivalent to a single WAIT with a sensitivity list at the bottom of the process:

```
process
begin
    . . . . .
    wait on a, b, c, d;
end process;
```

- Whenever any of the signals in the sensitivity list change value, process executed

Signals and Variables

```
ARCHITECTURE test1 OF mux IS
    SIGNAL x : BIT := '1';
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, x, y)
    BEGIN
        x <= in_sig XOR y;
        y <= in_sig XOR x;
    END PROCESS;
END test1;
```

```
ARCHITECTURE test2 OF mux IS
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, y)
    VARIABLE x : BIT := '1';
    BEGIN
        x := in_sig XOR y;
        y <= in_sig XOR x;
    END PROCESS;
END test2;
```

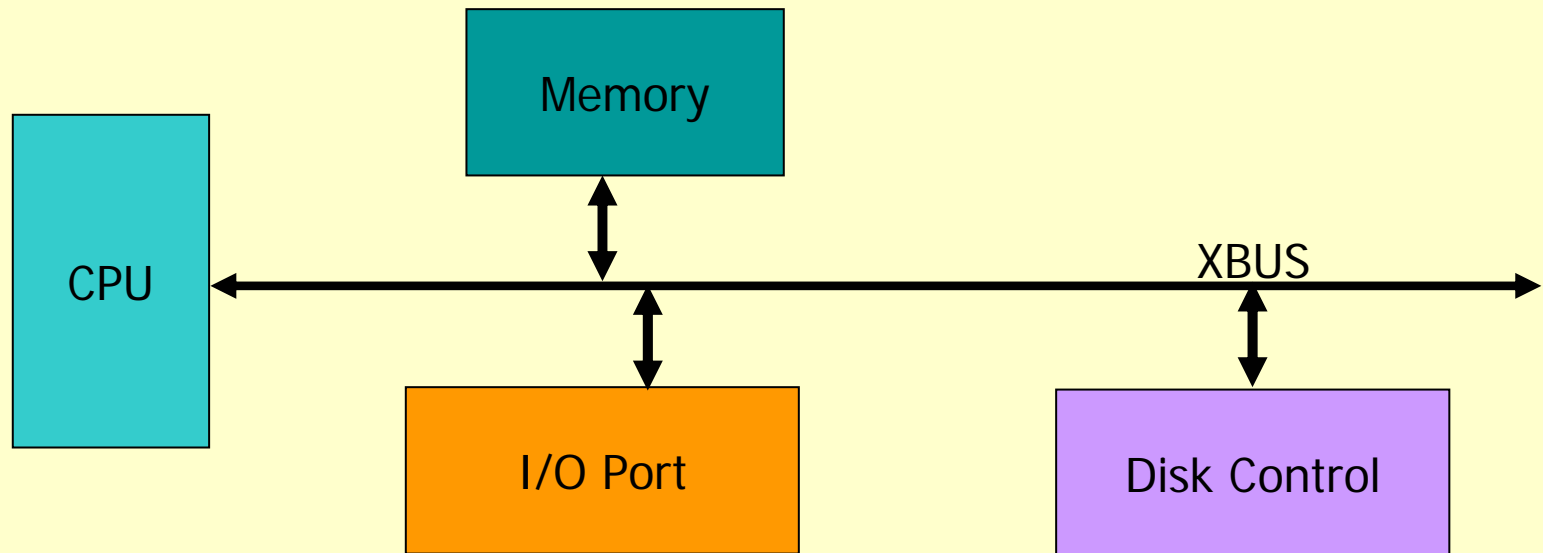
- Assuming a 1 to 0 transition on *in_sig*, resulting values for *y* in the both cases differ

Data types

- integers (huge 32 bit range, can be limited)
- reals (-1.0E+38 to +1.0E+38, no range clause available)
- unconstrained arrays (range/size not specified completely during type declaration)
- access types (akin to pointers)
- incomplete types (akin to linked lists)
- files

Resolution functions

- Cater to multiple driver problem of a signal
- are non-commutative



Attributes

- provide information about certain items
 - E.g. types, subtypes, procedures, functions, signals, variables
 - General form of attribute use :

```
name'attribute_identifier -- read as "tick"
```

- several predefined attributes
 - X'EVENT -- TRUE when there is an event on signal X
 - X'LAST_VALUE -- returns the previous value of signal X
 - Y'HIGH -- returns the highest value in the range of Y

Interface Modes

- Represent direction of value flow
- Frequently used modes:
 - IN: within the design unit (both entity and body) the value may be read, but not written.
 - OUT: within the design unit (both entity and body) the value may be written, but not read.
 - INOUT: within the design unit (both entity and body) the value may be both read and written.
 - BUFFER: within the design unit (both entity and body) the value may be both read and written.
Reading attributes also allowed

VHDL'93 updates

- many more attributes
- notion of postponed process
 - Executed after processing of all delta cycles
 - Benefit: each signal gets the final value of a simulation time
 - Typical use: timing checks, execute postponed process after input signals stabilize
- use of shared variables
 - Global sharing of variables permitted

Tools/Options considered

CV Model checker

- open-source and academic
- developed at CMU in 1996
- no users since 1998
- problem we faced: not getting installed
- Fires “library missing” error
- author cited Endian-ness incompatibility, yet to offer us a solution

CV Model checker

- VHDL subset it caters to:
 - **Design entities**: entity declaration, architecture body, package declaration, package body.
 - **Declarations**: type, subtype, signal, variable and constant and constant declarations.
 - **Types**: integer and enumeration types.
 - **Concurrent statements**: process statement, block statement, conditional signal assignment statement and selected signal assignment statement.
 - **Sequential statements**: if statement, case statement, null statement, wait statement (no timeout clause), variable assignment statement, signal assignment statement (only one, untimed, transaction in the source).
 - **Libraries and context clauses**.

VAUL

(VHDL Analyzer and Utility Library)

- Claims to be compliant with VHDL'87 and VHDL'93, needs testing for completeness
- aim is to be used as part of a simulation effort for VHDL
- generates a collection of C++ objects from the input VHDL RTL description
- needed lot of tweaking for installation
- no English documentation
- no way to view the AST
- had provided "wddl" feature of producing output in CDFG format which is "untested, incomplete and undocumented".
- more of a simulation tool than one for formal verification

Sample run with VAUL

```
Architecture behav_mux of mux is begin
process (a, b, c, d, s0, s1)
variable sel: integer;
begin
if s0='0' and s1= '0' then
    sel:=0;
elsif s0='1' and s1= '0' then
    sel:=1;
elsif s0='0' and s1= '1' then sel:=2;
else sel:=3;
end if;
case sel is
when 0 => x<= a;
when 1 => x<= b;
when 2 => x<= c;
when others =>    x<= d;
end case;
end process;
end behav_mux;
```

```
#include <freehdl/kernel.h>
#include <freehdl/std.h>

/* External declarations */
/* Definitions for enumeration type
:IEEE:std_logic_1164:std_ulogic */
class
L4ieee_Q14std_logic_1164_I10std_ulogic:pu
blic enum_info_base{
    static const char *values[];
public:
L4ieee_Q14std_logic_1164_I10std_ulogic():e
num_info_base(0,8,values) {};
    static const char **get_values() { return
values; }
    static int low() { return 0; }
    static int high() { return 8; }
    static int left() { return 0; }
    static int right() { return 8; }
}; ... (shown partly here)
```

VIS

(Verification Interacting with Synthesis)

- open-source
- accepts Verilog RTL as input
- “vl2mv” translator to generate BLIF-MV file used for further processing
- attempted interface for VHDL through
 - a) Synopsys DC (commercial):
 - produces synthesised Verilog netlist rather than HDL
 - unwanted side-effects of optimisation (we would rather not have)
 - b) EDIF2BLIF (open-source):
 - EDIF is a popular netlist standard
 - interpretation left to vendors
 - side-effect: EDIF outputs are technology specific (-- again not feasible for our line of work)

VST2BLIF

(VHDL structural -> BLIF)

- Open source
- feature available with SIS (primarily used for synthesis)
- covers only a subset of structural VHDL style
- data types supported: bit, bit_vector
- only boolean operators
- no support for IEEE library
- need to use accompanying *.genlib kind of libraries
- further restriction on how to write the code: each entity must be immediately followed by its architecture specification, else signals an error.
- works well if one knows a priori which genlib is to be used for the translation.

Sample run with VST2BLIF

```
architecture structural_mux of mux is
signal s0_inv, s1_inv, x1, x2, x3, x4 : bit;

--component declarations come here

begin
U1: inverter port map (in1 =>s0, x=>s0_inv);
U2: inverter (in1=>s1, x=>s1_inv);
U3: andgate (a=>a, b=>s0_inv, c=>s1_inv,
d=>x1);
U4: andgate (a=>b, b=>s0, c=>s1_inv,
d=>x2);
U5: andgate (a=>c, b=>s0_inv, c=>s1,
d=>x3);
U6: andgate(a=>d, b=>s0, c=>s1, d=>x4);
U7: orgate(a=>a, b=>b, c=>c, d=>d, x=>x);
end structural_mux;
```

```
# *-----*
# |   File created by Vst2Blif v 1.1 |
# |                                   |
# |       by Roberto Rambaldi       |
# |   D.E.I.S. Universita' di Bologna |
# *-----*/

.model mux
.input a b c d s0 s1
.output x

.subckt inverter in1=s0 x=s0_inv
.subckt inverter in1=s1 x=s1_inv
.subckt andgate a=a b=s0_inv c=s1_inv d=x1
.subckt andgate a=b b=s0 c=s1_inv d=x2
.subckt andgate a=c b=s0_inv c=s1 d=x3
.subckt andgate a=d b=s0 c=s1 d=x4
.subckt orgate a=a b=b c=c d=d x=x
```

Sample run with VST2BLIF

Genlib Used: lib2.genlib (shown partly here)

GATE inv1x 928.00 O = ! a;

PIN a INV 0.0514 999.0 0.4200 4.7100 0.4200 3.6000

GATE inv2x 928.00 O = ! a;

PIN a INV 0.1009 999.0 0.3000 1.9800 0.2900 1.8200

GATE inv4x 1392.00 O = ! a;

PIN a INV 0.1897 999.0 0.2300 1.0800 0.2700 0.8500

GATE xor 2320.00 O = ((!a * b) + (a * !b));

PIN a UNKNOWN 0.1442 999.0 1.7700 5.2300 0.9600 4.6400

PIN b UNKNOWN 0.1381 999.0 1.9400 4.6500 1.1400 5.2200

GATE xnor 2320.00 O = ((!a * !b) + (a * b));

PIN a UNKNOWN 0.1502 999.0 1.1100 4.8600 1.0700 3.3900

PIN b UNKNOWN 0.1352 999.0 1.5500 4.8700 1.0700 3.3900

GATE nand2 1392.00 O = ! (a * b);

PIN a INV 0.0777 999.0 0.6400 4.0900 0.4000 2.5700

PIN b INV 0.0716 999.0 0.4600 4.1000 0.3700 2.5700

GATE nand3 1856.00 O = ! (a * b * c);

PIN a INV 0.1000 999.0 0.8900 3.6000 0.5100 2.4900

PIN b INV 0.0828 999.0 0.7100 4.1100 0.4200 2.5000

PIN c INV 0.0777 999.0 0.5600 4.3900 0.3500 2.4900

VHD2VL

(VHDL RTL -> Verilog RTL)

- open-source from a company OceanLogic Ltd.
- Relatively new tool: released in 2001
- code and documentation seem OK to read and understand
- supports a reasonable enough subset of VHDL: no integers and reals
- no support for "FOR" loops, "WHILE" loops, and "WAIT" statements
- supports IEEE library
- support for attributes limited to: event, lastvalue, rising_edge and falling_edge
- allows both behavioral and structural style of VHDL coding

Sample run with VHD2VL

```
architecture behav_mux of mux is
process (a, b, c, d, s0, s1)
variable sel: std_logic_vector(1 downto 0);
begin
if s0='0' and s1= '0' then
    sel<="00";
elsif s0='1' and s1= '0' then
    sel<="01";
elsif s0='0' and s1= '1' then
    sel<="10";
else sel<="11"; end if;
case sel is
when "00" => x<= a;
when "01" => x<= b;
when "10" => x<= c;
when others => x<= d;
end case; end process;
end behav_mux;
```

```
module mux( a, b, c, d, s0, s1, x);
input a, b, c, d; input s0, s1; output x;
wire a; wire b; wire c; wire d; wire s0;
wire s1; reg x;
always @(a or b or c or d or s0 or s1) begin :
P1
    reg [1:0] sel;
    if(s0 == 1'b 0 && s1 == 1'b 0) begin
        sel <= 2'b 00;    end
    else if(s0 == 1'b 1 && s1 == 1'b 0) begin
sel <= 2'b 01;    end
    else if(s0 == 1'b 0 && s1 == 1'b 1) begin
sel <= 2'b 10;    end
    else if(s0 == 1'b 1 && s1 == 1'b 1) begin
sel <= 2'b 11;    end
    case(sel)
2'b 00 : begin    x <= a;    end
2'b 01 : begin    x <= b;    end
2'b 10 : begin    x <= c;    end
default : begin    x <= d;    end
endcase end endmodule
```

Sample run with VIS with input taken from VHD2VL

```
.model mux
.root mux
.inputs c d s0 s1 a b
.outputs x
.latch _n46<1> sel<1>
.reset ->sel<1>
.latch _n46<0> sel<0>
.reset ->sel<0>
.latch _n58 x
.reset ->x
0
1
.table ->_n1
0

.table s0 _n1 ->_n2
.default 0
0 1 1
1 0 1
.table _n2 ->_n0
0 1
1 0
.table ->_n5
0
.table s1 _n5 ->_n6
.default 0
0 1 1
1 0 1
```