EDA for Digital Circuits

II. Simulation of Digital Circuits
II. Simulation of Digital Circuits

1 Introduction

• Why use simulation?
  1. Hardware design validation (logic function, timing, power consumption)
  2. Fault simulation (fault coverage)
  3. Early co-validation of system hardware and software

• Simulation levels:
2 Logic simulation

- Overview of simulation system:

![Simulation System Diagram]

- **Netlist**
- **Stimuli**
- **Component models**
  - logic function
  - timing behavior
- **Simulator**

**Signal waveforms**

**typical signal models:**
- Boolean logic \{'0', '1'\}
- Three-valued logic \{'0', '1', 'X'\}
- IEEE standard logic \{'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'\}
Simulation of logic function („Zero Delay“)

Example circuit:

\[
x \rightarrow z_1 \rightarrow z_2 \rightarrow z_3 \rightarrow z_5 \rightarrow y
\]

<table>
<thead>
<tr>
<th>x</th>
<th>0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>z_1</td>
<td></td>
</tr>
<tr>
<td>z_2</td>
<td></td>
</tr>
<tr>
<td>z_3, z_4</td>
<td></td>
</tr>
<tr>
<td>z_5</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

Boolean function: \[ y = \overline{x} \cdot \overline{x} = x \cdot \overline{x} = 0 \]
Timing behavior: Modeling the delays between inputs and outputs of a component. Different alternatives:

- Gate delay model

- Gate and wire delays

- Pin-to-pin delays

\[ \Delta : \text{basic time unit} \]
\[ \tau_d = n \cdot \Delta : \text{propagation delay (gate delay)} \]
\[ \tau_w = m \cdot \Delta : \text{wire delay} \]
Example with signal traces (gate delay model)

Delay dependent effects:

a) Race: „run“ between signal changes on different wires leading to a single component

b) Hazard: short signal pulse which does not conform to the pure logic function of a circuit; caused by internal delays.
Evaluation of timing (timing verification, race analysis)

„logic“ is not considered

Assuming a signal edge on x at t = 0, the following „time table“ for the propagation of this edge to the circuit output results:

At the output we can now determine the maximum delay uncertainty (important for determining the clock period of synchronous circuits).
Simulation with slew rate (slope) uncertainty

A signal change does not have an ideal edge (infinite slope). Therefore a transition period results, during which the signal value is undefined. In simulation this is represented by a new signal value "X".

Example:

Let the signal uncertainty be one basic time unit:
Truth tables with the new signal value „X“:

<table>
<thead>
<tr>
<th>AND</th>
<th>0</th>
<th>1</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>0</th>
<th>1</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th>0</th>
<th>1</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Simulation with delay uncertainty

Let the minimum and maximum delay of a gate be known. In between the output value is undefined.

\[ \tau_{d_{\text{min}}} = \Delta \]
\[ \tau_{d_{\text{max}}} = 3\Delta \]

<table>
<thead>
<tr>
<th>x (ideal)</th>
<th>0 0 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1 1 1 X X 0 0</td>
</tr>
</tbody>
</table>
Circuit example:

\[ \tau_d \text{ min } = 3 \Delta \]
\[ \tau_d \text{ max } = 4 \Delta \]

<table>
<thead>
<tr>
<th>x</th>
<th>0 X 1 1 1 1 1 1 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 X 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>z_1</td>
<td>1 1 1 1 X X 0 0 0 0 0 X X 1 1 1</td>
</tr>
<tr>
<td>z_2</td>
<td>0 0 0 X X X 1 1 1 1 X X X 0 0 0</td>
</tr>
<tr>
<td>z_3, z_4</td>
<td>1 1 X X X X 0 0 0 X X X X 1 1 1</td>
</tr>
<tr>
<td>z_5</td>
<td>0 0 X X X 1 1 X X X 0 0 0</td>
</tr>
<tr>
<td>z_6</td>
<td>0 0 X 0 0 X X 0 0 0</td>
</tr>
<tr>
<td>y</td>
<td>0 0 X X 0 0 X X 0 0</td>
</tr>
</tbody>
</table>

\( \text{hazard possible} \)
3 Simulation methods

Simulation program

a) Modeling circuit components using basic elements (primitives) of the simulation system
b) Circuit replication (compilation or list creation)
c) Simulation execution

3.1 Compiler driven simulation

The circuit under test is compiled into an executable program. In each simulation cycle, the logic function of the complete circuit is evaluated, based on the input signals and the internal circuit states (registers, feedback signals).
Very fast, usually no timing analysis (zero delay simulation).
3.2 Event driven simulation

Event driven simulation is based on signal changes within the circuit. In each simulation step, components are evaluated only if a signal change occurs on their corresponding inputs. Signal changes are coded as events.

• Signal change: \( \text{val}(z, t_v) \neq \text{val}(z, t_{v-1}) \)

• Event:
A component evaluation at simulation time \( t_{\text{gen}} \) causes that signal \( z \) is set to a new value \( \text{val}(z, t_{\text{exe}}) \) at simulation time \( t_{\text{exe}} \).

\[
E = (z, \text{val}(z, t_{\text{exe}}), t_{\text{gen}}, t_{\text{exe}})
\]

- \( z \) : signal name
- \( \text{val}(z, t_{\text{exe}}) \) : value of signal \( z \) at time \( t_{\text{exe}} \)
- \( t_{\text{gen}} \) : generation time of event
- \( t_{\text{exe}} \) : execution time of event
- \( t_{\text{exe}} - t_{\text{gen}} = \tau \) : imposed delay
• Simulation cycle:

**event execution (change signal values)** → **find target components** → **evaluate these components** → **event generation**

*signal update*  *sensitivity list*  *process evaluation*  *signal assignment*

**zero delay events**

+ 1 *delta cycle*

**advance simulation time to next time stamp** *(delayed events, event queue)*

*next transactions in event queue*

(italic font indicates VHDL simulation terminology)
Example simulation run: Multiplexer

- Circuit:

```
\begin{circuit}\hspace{5cm} \text{Delays: } \tau = 2
\end{circuit}
```
- Simulation run:

<table>
<thead>
<tr>
<th>t</th>
<th>A</th>
<th>B</th>
<th>Sel</th>
<th>Sel_n</th>
<th>S_1</th>
<th>S_2</th>
<th>Q</th>
<th>evaluated components</th>
<th>new events (signal, val, t_gen, t_exe)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'0'</td>
<td>'0'</td>
<td>'1'</td>
<td>'0'</td>
<td>'1'</td>
<td>'1'</td>
<td>'0'</td>
<td>initial state (stimuli)</td>
<td>(A, '1', 0, 20); (B, '1', 0, 10); (Sel, '0', 0, 30)</td>
</tr>
<tr>
<td>10</td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_b</td>
<td>(S_2, '0', 10, 12)</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>'0'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_c</td>
<td>(Q, '1', 12, 14)</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_a</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>'0'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inv, Nand_b</td>
<td>(Sel_n, '1', 30, 32); (S_2, '1', 30, 32)</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>'1'</td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_a, Nand_c</td>
<td>(S_1, '0', 32, 34); (Q, '0', 32, 34)</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>'0'</td>
<td>'0'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_c</td>
<td>(Q, '1', 34, 36)</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
• Signal waveforms:
4 VHDL

**VHSIC Hardware Description Language** (VHSIC = Very High Speed Integrated Circuit)

- widely used hardware description language for simulation, documentation, and synthesis
- standardized format for libraries and interoperability of design tools
- similar to programming language; special extensions for modeling hardware:
  - Concurrency (description of concurrent processes)
  - Timing model (delays; event driven simulation)
- supports modeling of hardware at different levels of abstraction
- supports behavioral and structural description styles
VHDL model of a multiplexer

1. Specification:

<table>
<thead>
<tr>
<th>Sel</th>
<th>A</th>
<th>B</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>

Inputs: A, B, Sel
Output: Q

Sel = '0' ⇒ Q = A
Sel = '1' ⇒ Q = B

2. Realization:

- Interface description: ENTITY Declaration

- Description of an implementation: ARCHITECTURE

ENTITY Multiplexer IS
PORT (A, B, Sel: in bit;
     Q: out bit);
END Multiplexer;

Entity = Entity Declaration + Architecture
3. Behavioral description (dataflow, zero delay):

ENTITY Multiplexer IS
  PORT (A, B, Sel: in bit;
       Q: out bit);
END Multiplexer;

ARCHITECTURE dataflow OF Multiplexer IS
BEGIN
  PROCESS (A, B, Sel)
  BEGIN
    CASE Sel IS
      WHEN '0' => Q <= A;
      WHEN '1' => Q <= B;
    END CASE;
  END PROCESS;
END dataflow;
4. Structural description:

(A) Circuit interface:

ENTITY Multiplexer IS
  PORT (A, B, Sel: in bit;
       Q: out bit);
END Multiplexer;

(B) Circuit implementation:

LIBRARY BasicGates;
USE BasicGates.ALL;

ARCHITECTURE structure OF Multiplexer IS
  COMPONENT Inverter
    PORT (i: in bit; o: out bit);
  END COMPONENT;
  COMPONENT Nand2
    PORT (i1, i2: in bit; o: out bit);
  END COMPONENT;
  SIGNAL Sel_n, S1, S2: bit;
  BEGIN
    Inv: Inverter PORT MAP (Sel, Sel_n);
    Nand_a: Nand2 PORT MAP (A, Sel_n, S1);
    Nand_b: Nand2 PORT MAP (B, Sel, S2);
    Nand_c: Nand2 PORT MAP (S1, S2, Q);
  END structure;
5. Validation:

Testbench:

- applies stimuli to circuit inputs
- monitors circuit outputs
- checks correctness via assert statements (optional)

Configuration:

- binding of architectures to entities
- flexible mechanism for using alternative implementations
**VHDL Modell of a Multiplexer**

**Testbench for a Multiplexer**

*Interface Description:*

ENTITY Testbench IS
  --
  -- no inputs, no outputs.
  --
END Testbench;

*Implementation - instantiating the MUX as a COMPONENT:*

ARCHITECTURE Functiontest OF Testbench IS
COMPONENT Multiplexer
  PORT (A, B, Sel: in bit;
        Q: out bit);
END COMPONENT;
SIGNAL A, B, Sel, Q: bit;
BEGIN
  MUX_1: Multiplexer PORT MAP (A, B, Sel, Q);
  A <= '0', '1' after 20 ns, '0' after 60 ns, '1' after 100 ns;
  B <= '1', '0' after 30 ns, '1' after 70 ns;
  Sel <= '0', '1' after 40 ns, '0' after 80 ns;
END Functiontest;
**VHDL model of a multiplexer**

**Configuration, Entity / Architecture Binding:**

CONFIGURATION Test_Setup OF Testbench IS
FOR Funktionstest
   FOR MUX_1: Multiplexer USE ENTITY WORK.Multiplexer(dataflow);
   END FOR;
END FOR;
END Test_Setup;
Processes:

- Statements within a \texttt{PROCESS} are executed sequentially within an infinite loop.
- All processes are concurrent.
- A \texttt{PROCESS} must have either a \texttt{WAIT}-statement or a \texttt{sensitivity-List}.
- Each signal assignment outside of a \texttt{PROCESS} will be modified into an individual \texttt{PROCESS} by the simulator.
- In a \texttt{PROCESS} \textit{variables} can be used in addition to \textit{signals}.

\textbf{Example: Clock Generator}

\textbf{Example for signal assignment}
ARCHITECTURE Simple OF CLKgen IS
BEGIN
    CLK <= NOT CLK AFTER 10 ns;
END Simple;

\textbf{Example for SENSITIVITY LIST}
ARCHITECTURE SList OF CLKgen IS
BEGIN
    PROCESS (CLK)
    BEGIN
        CLK <= NOT CLK AFTER 10 ns;
    END PROCESS;
END SList;

\textbf{Example for \texttt{WAIT ON} statement}
ARCHITECTURE Wait1 OF CLKgen IS
BEGIN
    PROCESS
    BEGIN
        CLK <= NOT CLK AFTER 10 ns;
        WAIT ON CLK;
    END PROCESS;
END Wait1;

\textbf{Example for \texttt{WAIT FOR} statement}
ARCHITECTURE Wait2 OF CLKgen IS
BEGIN
    PROCESS
    BEGIN
        CLK <= NOT CLK;
        WAIT FOR 10 ns;
    END PROCESS;
END Wait2;
Two process descriptions:

ENTITY XXX IS
  PORT (a, b: IN bit; c: OUT bit);
END XXX;

ARCHITECTURE example1 OF XXX IS
BEGIN
  PROCESS
  BEGIN
    IF a = '1' AND b = '1' THEN
      c <= '1';
      WAIT ON a, b;
    ELSIF a = '0' THEN
      c <= '0';
      WAIT ON a;
    ELSIF b = '0' THEN
      c <= '0';
      WAIT ON b;
    END IF;
  END PROCESS;
END example1;

ENTITY XXX IS
  PORT (a, b: IN bit; c: OUT bit);
END XXX;

ARCHITECTURE example2 OF XXX IS
BEGIN
  PROCESS
  BEGIN
    IF a = '1' AND b = '1' THEN
      c <= '1';
      WAIT ON a, b;
    ELSE
      c <= '0';
      WAIT ON a, b;
    END IF;
  END PROCESS;
END example2;

• Which function is described by these processes?

• Which modelling is better suited for a simulator?
Signals and variables:

Signals:

• Have a temporal „memory“. Future signal values are planned by entry of a transaction 
  \((w_n, t_n)\) in the signal driver. 
  \((w_n: \text{planned signal value}, t_n: \text{timestamp})\)  \(\text{(transaction} \neq \text{event)}\)

• In value assignments „\(<= \text{waveform}“\) the new value will be planned. It will be incurred only when the 
  \text{PROCESS will be suspended (during the signal update phase).}

Variables:

• Do not have a temporal memory.

• In value assignments „\(:= \text{expression}“\) the new value will be incurred immediately.
Delay Mechanisms:

**Transport-Delay:** \((\text{signal} \leq \text{transport} \ \text{waveform};)\)

- Modeling of an ideal component with unlimited bandwidth.
- Rule for insertion of a new transaction \((w_n, t_n)\) into the signal driver:
  - All (later) transactions \((w_i, t_i)\) with \(t_i \geq t_n\) will be deleted.
  - The new transaction \((w_n, t_n)\) will be appended at the end of the list.

**Inertial-Delay:** \((\text{signal} \leq \text{waveform};)\)

- Modeling of real components with finite bandwidth.
  (impulses, which are shorter than the component’s delay, will be suppressed.)
- Rule for insertion of a new transaction \((w_n, t_n)\) into the signal driver:
  - All (later) transactions \((w_i, t_i)\) with \(t_i \geq t_n\) will be deleted.
  - The new transaction \((w_n, t_n)\) will be appended at the end of the list.
  - Transactions \((w_i, t_i)\) within the interval \(t_{\text{now}} \leq t_i \leq t_n\) will be deleted, if \(w_i \neq w_n\).
ENTITY inverter IS
PORT (  
   inp : IN std_logic;
   outp : OUT std_logic);
END inverter;

ARCHITECTURE delayMechanism OF inverter IS
BEGIN
   outp <= TRANSPORT NOT inp AFTER 10 ns;
END ARCHITECTURE;
ENTITY inverter IS
PORT (  
    inp : IN std_logic;
    outp : OUT std_logic);
END inverter;

ARCHITECTURE delayMechanism OF inverter IS
BEGIN  
    outp <= INERTIAL NOT inp AFTER 10 ns;
END ARCHITECTURE;
ENTITY inverter IS
PORT (  
  inp : IN std_logic;
  outp : OUT std_logic);
END inverter;

ARCHITECTURE delayMechanism OF inverter IS
BEGIN
  outp <= REJECT 2 ns INERTIAL NOT inp AFTER 10 ns;
END ARCHITECTURE;
Event-driven Simulation:

Execution of all transactions at time step $t_n$ in $\Delta$-cycles. Advancing the simulation time $t_n \rightarrow t_{n+1}$, when all transactions at time $t_n$ have been executed.

$\Delta$-cycle:

• Signal-Update-Phase:
  - Execution of all transactions for the current time step.
  - If a transaction changes a signal value, this is an event.

• Process-Evaluation-Phase:
  - Processes, for which an event is present, will be executed.
  - Value assignments to signals will be scheduled in the signal driver.