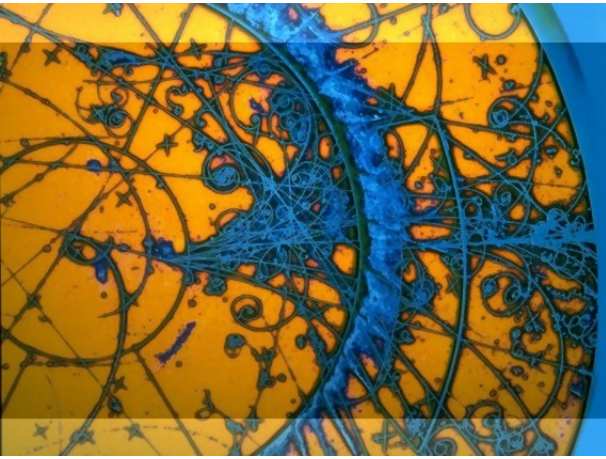
SysAdmin Group Presentation



Field-Programmable Gate Array Development

Igor Senderovich

April 17, 2008

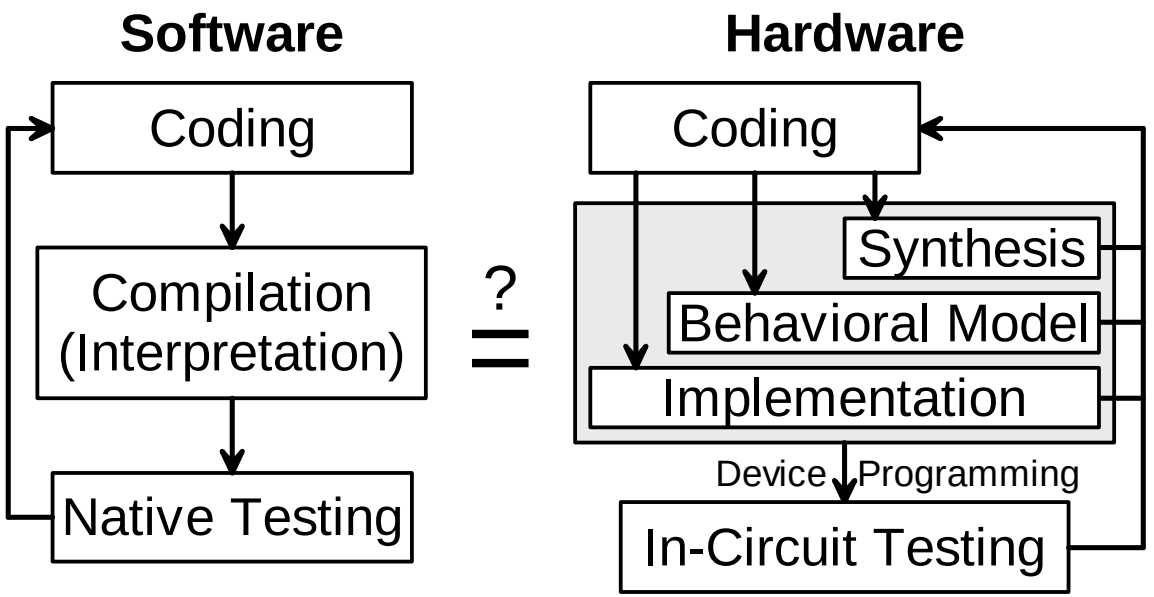
Outline

1. Introduction: Computer-Aided Hardware Design
2. FPGA Design Flow
3. General hardware programming issues
   i. Hardware programming *state of mind*
   ii. Hardware Description Languages
   iii. Component Specification
4. VHDL
   i. Example 1: Basic Logic and Synchronization
   ii. Example 2: 3-bit register
5. Implementation
6. Deployment

Computer-Aided Hardware Design

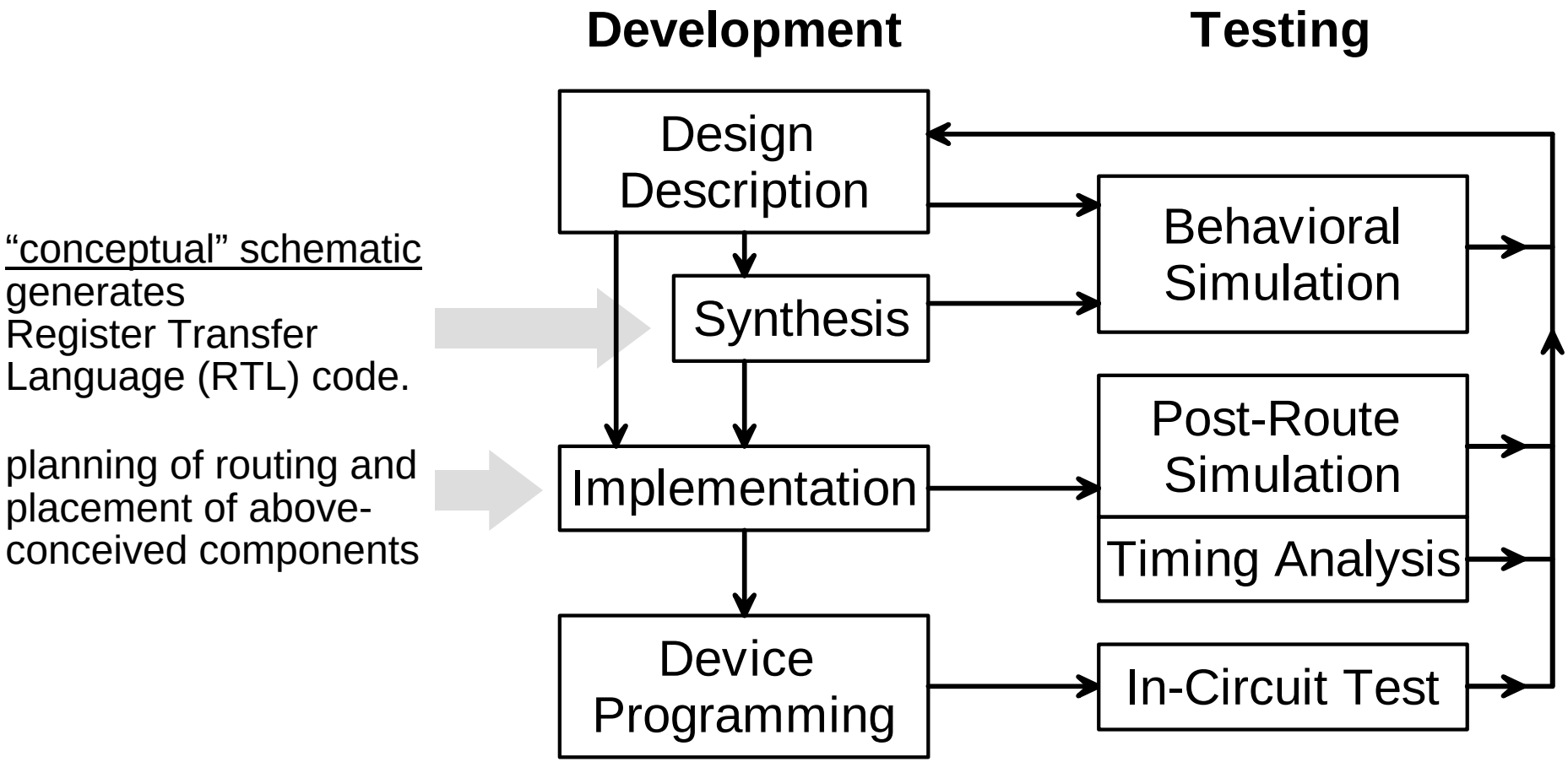
- The key phrase is Computer-Aided
  - Computer provides:
    - Design aids: programming environment, visualization etc.
    - Simulation environment: functionality and performance prediction
  - Computer does NOT provide a native test environment

- Comparison to software development:



FPGA Design Flow

More Detailed FPGA Design Flow:



Hardware programming *state of mind*

- No expectations of sequential processing!
  - All components are “on” and responding to stimulus; all parts of the code are working at once.
  - Order restored with Chip Enable (CE) pins, gates, component’s internal counters: quiet until a “Go” signal
- Scope:
  - Component’s signals: internal or patched outside for sharing
  - Shared signal access: “whoever cares to connect” → constant state of “sharing violation”. Solutions:
    - Separate communication lines
    - Components writing to same line are enabled one at a time. Pins of quiet components set to high impedance.

Hardware Description

- **HDL** – **H**ardware **D**escription **L**anguage (generic term)
  - **Verilog** - designed to resemble C
  - **VHDL** - language based on Ada
    - 'V' for VHSIC (Very-High-Speed Integrated Circuits)
    - product of a 1980s U.S. Defense project
  - Other, less common languages
    - **ABEL** (Adv. Boolean Expression Lang.)
    - **AHDL** (Altera's proprietary language)
    - **Atom, Bluespec, Hydra, Lava** (Haskell-based)
    - **CUPL** (proprietary: Logical Devices, Inc.)
    - **HDCaml** (based on Objective Caml)
    - **Hardware Join Java**
    - **HML** (based on SML)
    - **JHDL** (based on Java)
    - **Lola** (a pedagogical tool)
    - **MyHDL** (based on Python)
    - **PALASM** (for Prog. Array Logic (PAL) devices)
    - **Ruby** (hardware description language)
    - **RHDL** (based on the Ruby prog. language)
    - **SystemVerilog** (superset of Verilog+)
    - **SystemC** (C++ libraries for system-level modeling)

Component Specification

- **Instantiation** – Components added from libraries much like functions from API dynamic libraries.
  - Offers full control of the use of components
  - Useful (and sometimes only allowed) for complicated parts with standard, well-circumscribed behavior
  - Works well with visual schematic design/programming

- **Inference** – Components are *inferred* from functionality described in the programming.
  - Readable/portable code
  - The only approach to simple components (likely majority of design)

Ex. 1: Basic Logic and Synchronization

"include" statements

Define "black box" → (external signals)

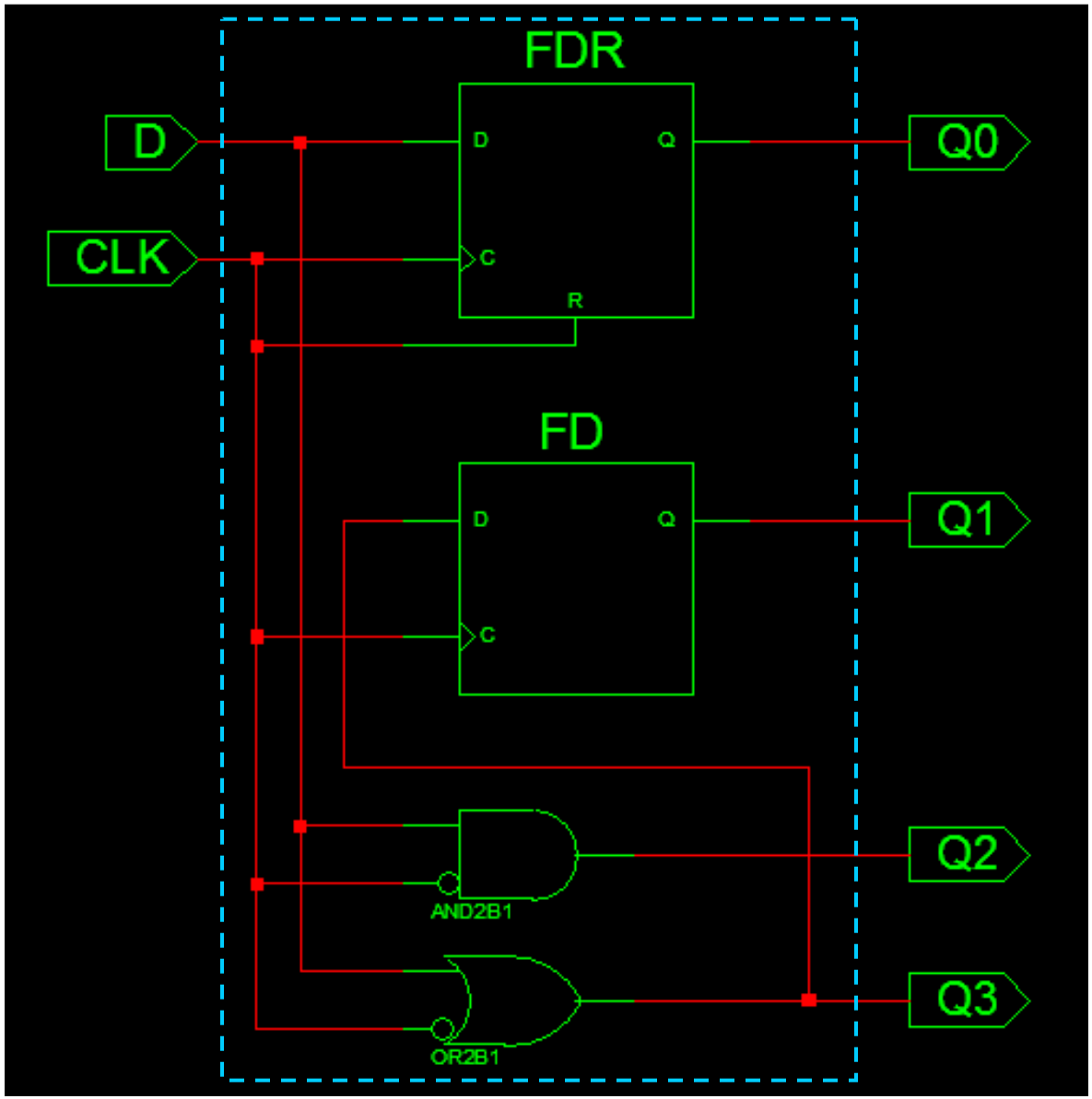
"Sensitivity List" to aid simulator →

rising edge-driven statements (synchronization!)

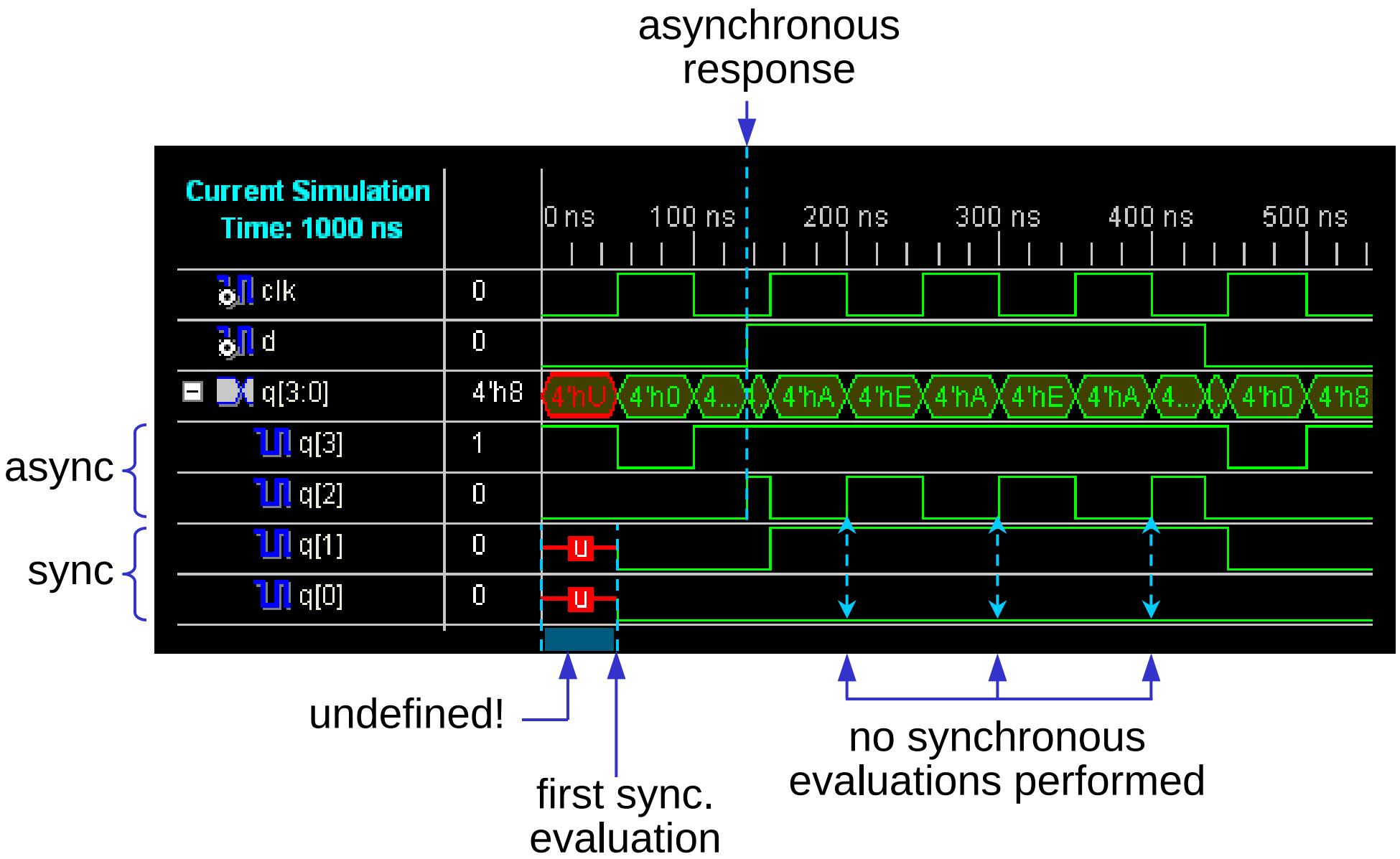
```
01  library IEEE;
02  use IEEE.STD_LOGIC_1164.ALL;
03
04  entity CLKswitch is
05      Port ( CLK : in  STD_LOGIC;
06             D : in  STD_LOGIC;
07             Q : out STD_LOGIC_VECTOR (3 downto 0));
08  end CLKswitch;
09
10  architecture Behavioral of CLKswitch is
11  begin
12      Q(3) <= not CLK or D;
13      Q(2) <= not CLK and D;
14
15      edge_trig : process (CLK, D)
16      begin
17          if rising_edge(CLK) then      --forces sync
18              Q(1) <= not CLK or D;
19              Q(0) <= not CLK and D;
20          end if;
21      end process;
22
23  end Behavioral;
```

Ex. 1: Basic Logic and Synchronization

RTL Schematic:



Ex. 1: Basic Logic and Synchronization



Ex. 2: 3-bit Register

Internal signal declared →

"Sensitivity List" to aid simulator →

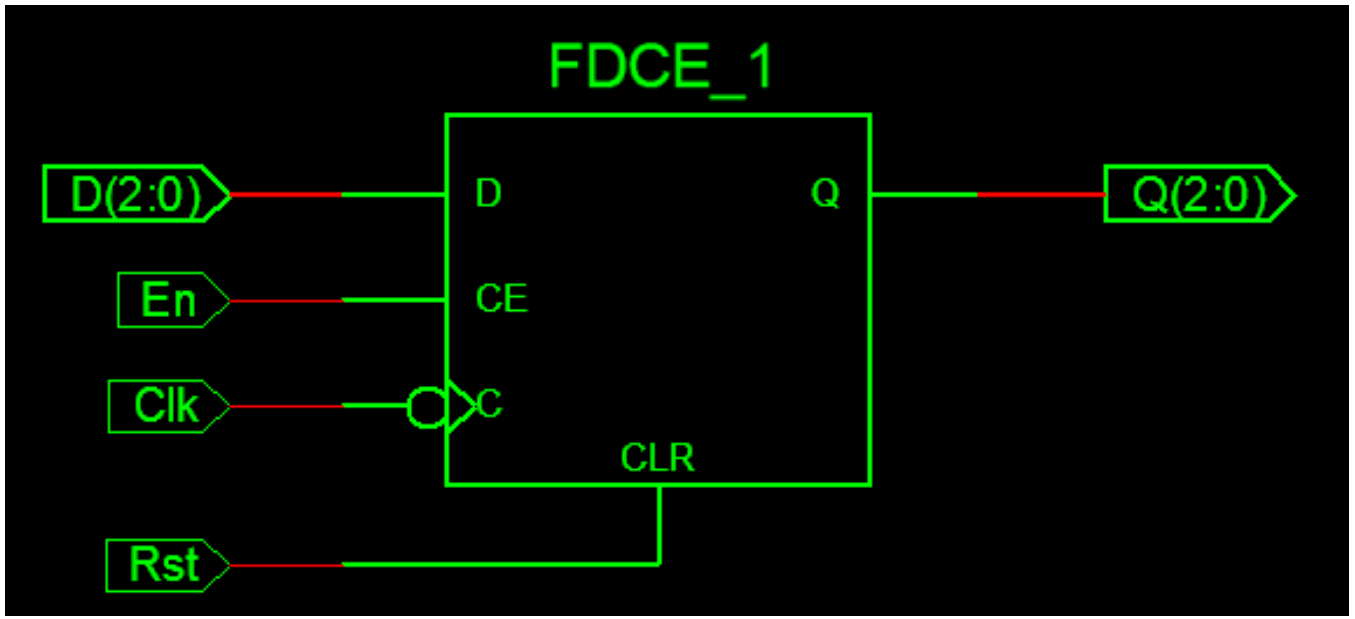
Reset to avoid undefined behavior →

Explicit request for storage →
(often inferred anyway)

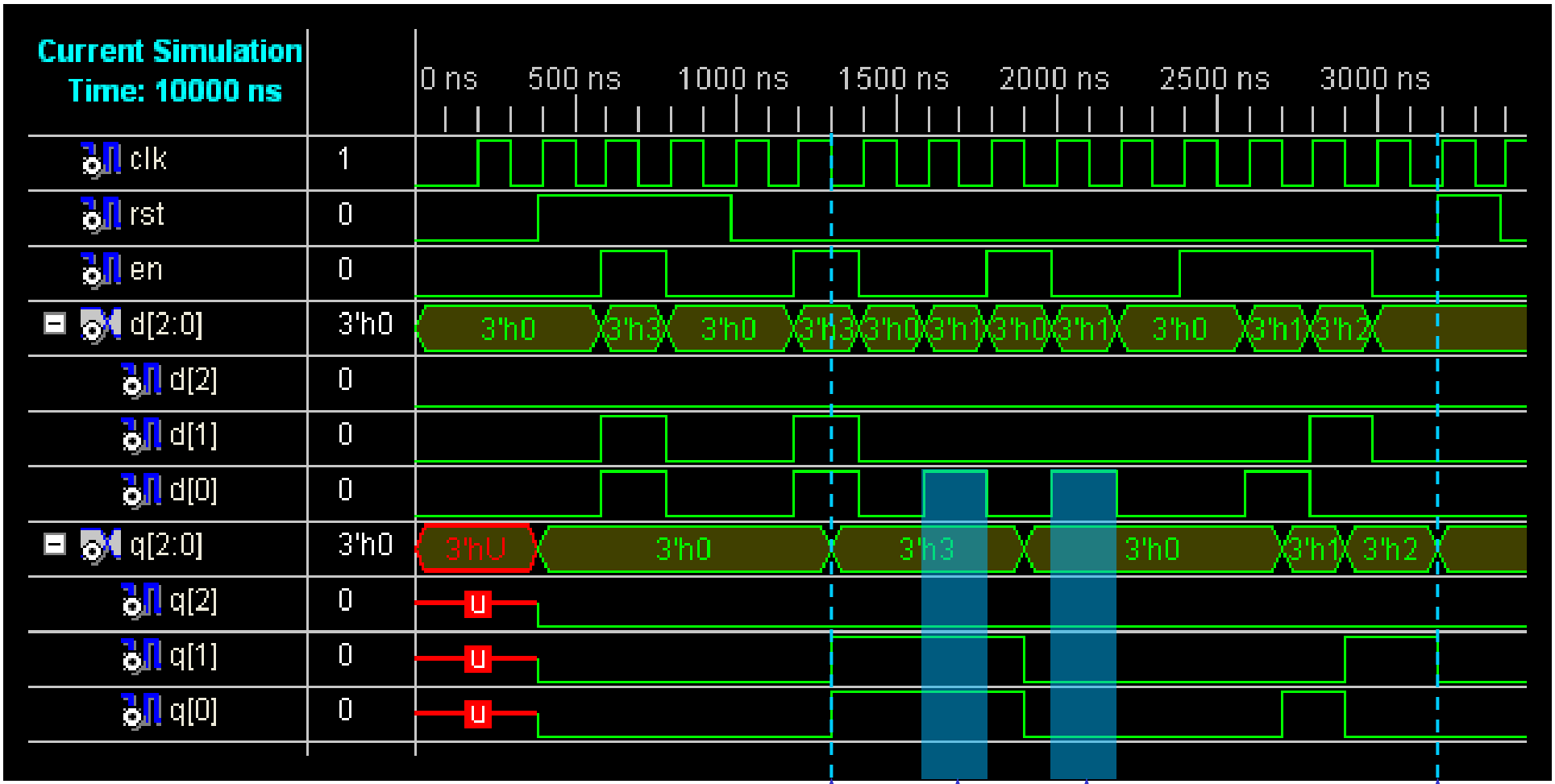
```
01  architecture Behavioral of stateReg is
02      signal state : STD_LOGIC_VECTOR (2 downto 0);
03  begin
04
05      LatchInput : process (Clk, Rst, En)
06      begin
07          if (Rst = '1') then
08              state <= "000";
09          else
10              if (falling_edge(Clk) and En='1') then
11                  state <= D;
12              else
13                  state <= state;
14              end if;
15          end if;
16      end process LatchInput;
17
18      Q <= state;
19
20  end Behavioral;
```

Ex. 2: 3-bit Register

A Standard Register Component!



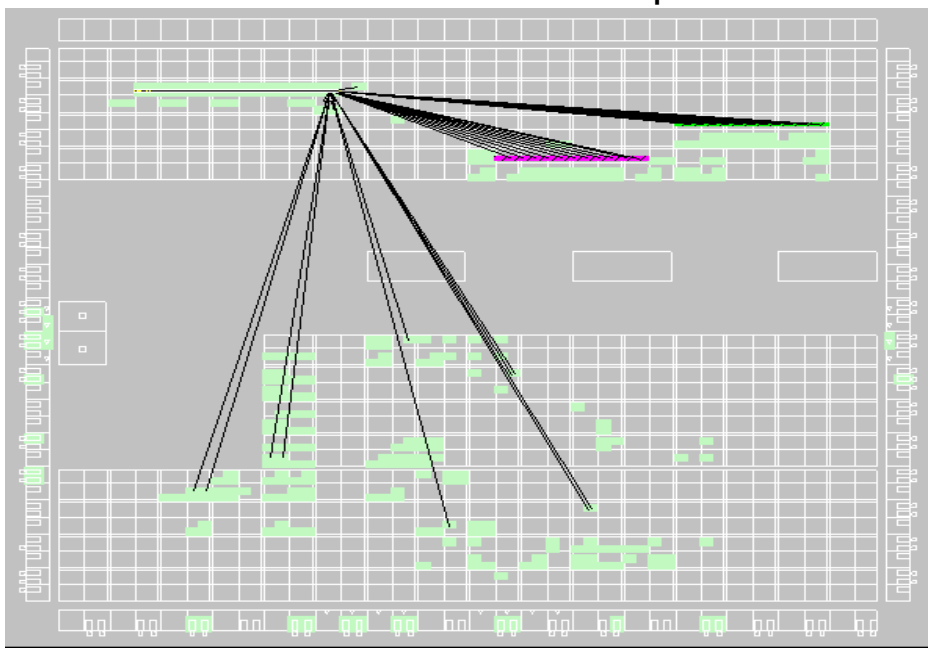
Ex. 2: 3-bit Register



Implementation

- A more concrete plan of components, their placement and routing in the Gate Array is generated
- Constraint and optimization details specified
- Auto-generated maps adjusted
- Timing and power consumption analyzed
- More realistic simulations
  - Functionality verified
  - Timing tested

Sample Floor Plan



FPGA Configuration Schemes

- FPGA is fed a programming bit stream. Sources:
  - **In-System**: using computer interface: useful in the prototyping stage
  - **In the Field** (after deployment): $E^{n}PROM$, $n = 0,1,2$ – on-board and activated on power cycle*
  - More custom solutions possible (e.g. microcontroller or CPLD-directed programming on startup)

* Non-volatile FPGA's are available, eliminating the need for programming sources after the development stage