SysAdmin Group Presentation

Field-Programmable Gate Array Development

Igor Senderovich
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
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:

Software

Coding → Compilation (Interpretation) → Native Testing

Hardware

Coding → Synthesis → Behavioral Model → Implementation → Device → Programming → In-Circuit Testing
More Detailed FPGA Design Flow:

“conceptual” schematic generates Register Transfer Language (RTL) code.

Planning of routing and placement of above-conceived components
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** – Hardware Description Language (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

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CLKswitch is
  Port ( CLK : in STD_LOGIC;
          D  : in STD_LOGIC;
          Q  : out STD_LOGIC_VECTOR (3 downto 0));
end CLKswitch;

architecture Behavioral of CLKswitch is
begin
  Q(3) <= not CLK or D;
  Q(2) <= not CLK and D;

  edge_trig : process (CLK, D)
  begin
    if rising_edge(CLK) then             --forces sync
      Q(1) <= not CLK or D;
      Q(0) <= not CLK and D;
    end if;
  end process;

end Behavioral;
```
Ex. 1: Basic Logic and Synchronization

RTL Schematic:
Ex. 1: Basic Logic and Synchronization

Current Simulation
Time: 1000 ns

<table>
<thead>
<tr>
<th></th>
<th>0 ns</th>
<th>100 ns</th>
<th>200 ns</th>
<th>300 ns</th>
<th>400 ns</th>
<th>500 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q[3:0]</td>
<td>4'h8</td>
<td>4'h8</td>
<td>4'h8</td>
<td>4'h8</td>
<td>4'h8</td>
<td>4'h8</td>
</tr>
<tr>
<td>q[3]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q[2]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q[1]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q[0]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

asynchronous response

async

sync

undefined!

first sync. evaluation

no synchronous evaluations performed
Ex. 2: 3-bit Register

```vhdl
architecture Behavioral of stateReg is
  signal state : STD_LOGIC_VECTOR (2 downto 0);
begin
  LatchInput : process (Clk, Rst, En)
  begin
    if (Rst = '1') then
      state <= "000";
    else
      if (falling_edge(Clk) and En='1') then
        state <= D;
      else
        state <= state;
      end if;
    end if;
  end process LatchInput;

  Q <= state;
end Behavioral;
```
Ex. 2: 3-bit Register

A Standard Register Component!

[Diagram of a 3-bit register with inputs D(2:0), En, Clk, Rst, and outputs D, Q, CLR, Q(2:0)]
Ex. 2: 3-bit Register

<table>
<thead>
<tr>
<th>Current Simulation</th>
<th>Time: 10000 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>1</td>
</tr>
<tr>
<td>rst</td>
<td>0</td>
</tr>
<tr>
<td>en</td>
<td>0</td>
</tr>
<tr>
<td>d[2:0]</td>
<td>3'h0 3'h3 3'h0 3'h3 3'h0 3'h0 3'h1 3'h0 3'h1 3'h0 3'h1 3'h2</td>
</tr>
<tr>
<td>d[2]</td>
<td>0</td>
</tr>
<tr>
<td>d[1]</td>
<td>0</td>
</tr>
<tr>
<td>d[0]</td>
<td>0</td>
</tr>
<tr>
<td>q[2:0]</td>
<td>3'h0 3'hU 3'h0 3'h3 3'h0 3'h1 3'h2</td>
</tr>
<tr>
<td>q[2]</td>
<td>0</td>
</tr>
<tr>
<td>q[1]</td>
<td>0</td>
</tr>
<tr>
<td>q[0]</td>
<td>0</td>
</tr>
</tbody>
</table>

- latched on falling clock edge
- ignored
- register zeroed on reset
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
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