Systems Architecture

Lab: Introduction to VHDL

Jeremy R. Johnson
Anatole D. Ruslanov
William M. Mongan
Introduction

- Objective: To provide an introduction of digital design and simulation. To introduce the hardware description language VHDL and the VHDL simulator and design tools from SymphonyEDA: Sonata and VHDLSimili.

- Behavioral models
- Structural models
- Discrete event simulation
  - signals
  - events
  - waveforms
  - concurrency
VHDL

- VHSIC Hardware Description Language
  - Very High Speed Integrated Circuit
- IEEE standard
  - IEEE 1076-1987
  - IEEE 1076-1993

- A language for describing digital designs with several levels of abstraction
  - behavioral (describe what the design does)
  - structural (describe how the design is implemented)

- Tools exist for verifying designs and synthesizing hardware layout from designs specified in VHDL
VHDL Simulation

- Since a VHDL program is a description of a design, it, by itself, does not compute anything.

- To verify a design, it must be simulated on a set of inputs.

- Simulation is “event driven”
  - When inputs change (an event) the corresponding outputs are computed using rules specified in the design
  - The state of the design may also change (sequential logic - e.g. memory, registers, or any state variables)
  - Changes propagate throughout the system
  - There may be delays
  - Operations may occur concurrently
Intro to VHDL Lab

- **Objective:** Prepare for the implementation of the MIPS ALU. To learn how to implement simple designs (both behavioral and structural) in VHDL using the VHDL simulator and design tools from SymphonyEDA: Sonata and VHDLSimili.

- **Components**
  - multiplexor
  - full adder
  - and, or, not gates
Boolean Functions

- A boolean variable has two possible values (true/false) (1/0).
- A boolean function has a number of boolean input variables and has a boolean valued output.
- A boolean function can be described using a truth table.
- There are $2^n$ boolean function of n variables.

<table>
<thead>
<tr>
<th>s</th>
<th>x₀</th>
<th>x₁</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Multiplexor function
Boolean Expressions

• A boolean expression is a boolean function.

• Any boolean function can be written as a boolean expression
  
  – Disjunctive normal form (sums of products)
  – For each row in the truth table where the output is true, write a product such that the corresponding input is the only input combination that is true
  – Not unique

• E.G. (multiplexor function)

\[
s \cdot \overline{x_0} \cdot \overline{x_1} + s \cdot x_0 \cdot x_1 + s \cdot \overline{x_0} \cdot x_1 + s \cdot x_0 \cdot x_1
\]

<table>
<thead>
<tr>
<th>s</th>
<th>x_0</th>
<th>x_1</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Simplification of Boolean Expressions

- Simplifying multiplexor expression using boolean algebra

\[
\overline{s} \cdot x_0 \cdot \overline{x_1} + \overline{s} \cdot x_0 \cdot x_1 + s \cdot \overline{x_0} \cdot x_1 + s \cdot x_0 \cdot x_1 \\
= \overline{s} \cdot x_0 \cdot \overline{x_1} + \overline{s} \cdot x_0 \cdot x_1 + s \cdot x_1 \cdot \overline{x_0} + s \cdot x_1 \cdot x_0 \quad \text{(commutative law)}
\]

\[
= \overline{s} \cdot x_0 \cdot (x_1 + \overline{x_1}) + s \cdot x_1 \cdot (x_0 + \overline{x_0}) \quad \text{(distributive law)}
\]

\[
= \overline{s} \cdot x_0 \cdot 1 + s \cdot x_1 \cdot 1 \quad \text{(inverse law)}
\]

\[
= \overline{s} \cdot x_0 + s \cdot x_1 \quad \text{(identity law)}
\]

- Verify that the boolean function corresponding to this expression as the same truth table as the original function.
Logic Circuits

- A single line labeled $x$ is a logic circuit. One end is the input and the other is the output. If $A$ and $B$ are logic circuits so are:

- and gate

- or gate

- inverter (not)
Logic Circuits

- Given a boolean expression it is easy to write down the corresponding logic circuit
- Here is the circuit for the original multiplexor expression
Logic Circuits

• Here is the circuit for the simplified multiplexor expression
A multiplexor is a switch which routes n inputs to one output. The input is selected using a decoder.
VHDL constructs

- **Entity**
- **Port**
- **Architecture**
  - implementation of an entity
  - support for both behavioral and structural models
- **Signal**
  - `std_ulogic` (from IEEE 1164 library)
  - input/output signals
  - internal signals inside an architecture
  - assignment
  - delays
- **Vectors of signals (bus)**
entity half_adder is
port(a, b : in std_ulogic;
    sum, carry : out std_ulogic);
end half_adder;
entity mux is
  port(I0, I1, I2, I3 : in std_ulogic;
       Sel        : in std_ulogic_vector (1 downto 0);
       Z          : out std_ulogic);
end mux;
entity ALU32 is
port(a,b : in std_ulogic_vector (31 downto 0);
    Op : in std_ulogic_vector (2 downto 0);
    result : out std_ulogic_vector (31 downto 0);
    zero : out std_ulogic;
    CarryOut : out std_ulogic;
    overflow : out std_ulogic);
end ALU32;
Half Adder Behavioral Implementation

library IEEE;
use IEEE.std_logic_1164.all;
entity half_adder is
port(a,b : in std_ulogic;
     sum,carry : out std_ulogic);
end half_adder;

architecture concurrent_behavior of half_adder is
begin
  sum <= (a xor b) after 5 ns;
carry <= (a and b) after 5 ns;
end concurrent_behavior;
Half Adder Structural Implementation

library IEEE;
use IEEE.std_logic_1164.all;
entity half_adder is
port(a,b : in std_logic;
       sum,carry : out std_logic);
end half_adder;

architecture structural of half_adder is
component and_2
port(x,y : in std_logic;
     c : out std_logic);
end component;
Half Adder Implementation (cont.)

component xor_2
port(x,y : in std_logic;
    c : out std_logic);
end component;

begin
X1 : xor_2 port map(x => a, y => b, c => sum);
A1 : and_2 port map(x => a, y => b, c => carry);
end structural;
MUX Behavioral Implementation

entity mux4 is
 port(I0, I1, I2, I3 : in std_ulogic;
      Sel : in std_ulogic_vector (1 downto 0);
      Z  : out std_ulogic);
end mux;

architecture behavioral of mux4 is
begin
  with Sel select
  Z <= I0 after 5 ns when "00",
       I1 after 5 ns when "01",
       I2 after 5 ns when "10",
       I3 after 5 ns when "11",
       'U' after 5 ns when others;
end behavioral;
Simulation Model

- Signal changes are called events

- Whenever an input signal changes the output is recomputed
  - Changes are propagated
  - There may be a delay

- Updates occur concurrently

- The history of changes on a signal produces a waveform (value vs. time)
  - Shows delays and dependencies