VHDL: Basic Concepts

Madhav Desai

January 18, 2018

VHDL

- Purpose: to describe hardware in a machine-readable form.
- The description should be unambiguous and
  - it should be possible to simulate the description.
  - it should be possible to derive (synthesize) a physical implementation of a logic circuit which is equivalent to the description.

A two bit adder

- The inputs are x_1, x_0, y_1, y_0 and the outputs are s_1, s_0.
- After some effort, we obtain the following formulas.

$$\begin{aligned} s_0 &= ((y_0.\overline{x_0}) + (\overline{y_0}.x_0)) \\ w &= ((y_1.\overline{x_1}) + (\overline{y_1}.x_1)) \\ z &= (y_0.x_0) \\ s_1 &= ((w.\overline{z}) + (\overline{w}.z)) \end{aligned}$$

A visualization of the two bit adder logic circuit

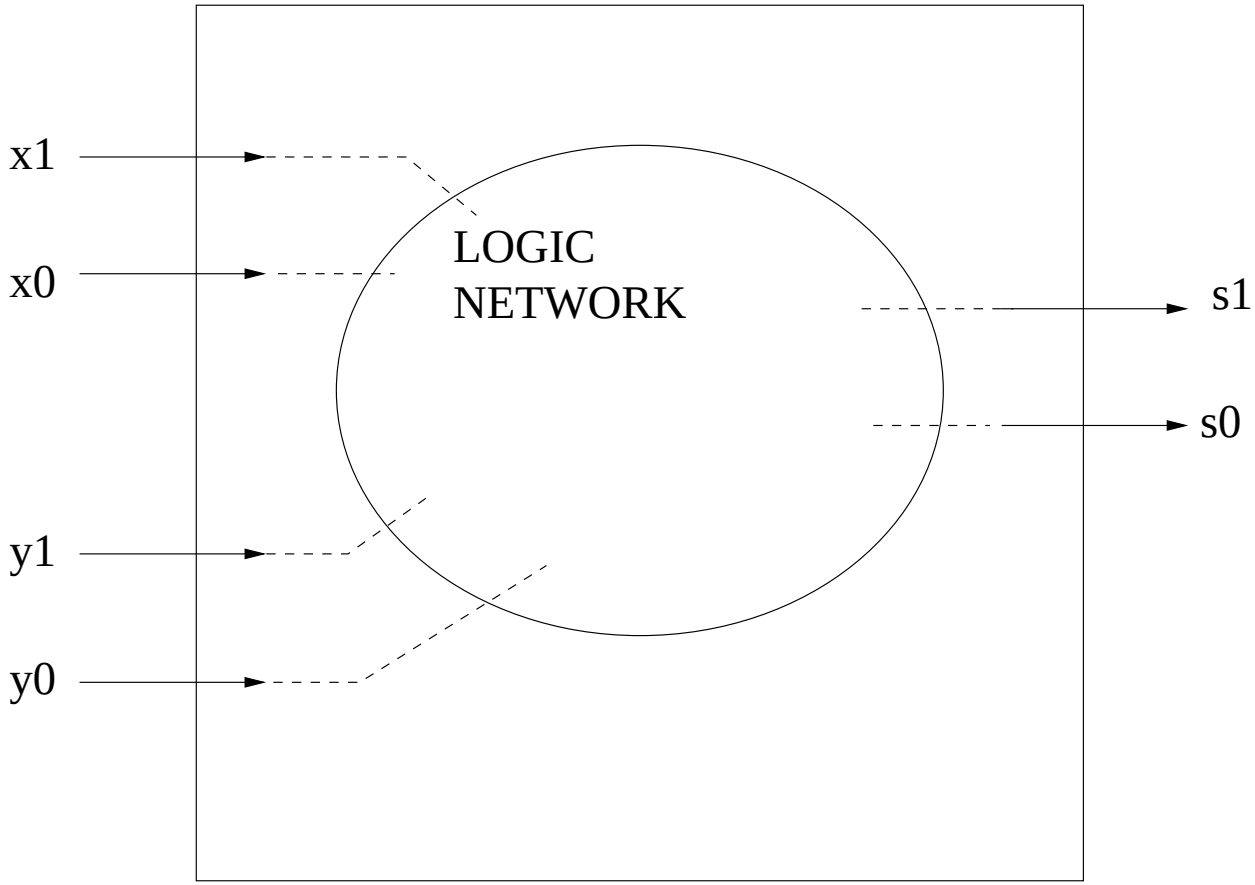

Figure : Visualization of two bit adder logic circuit

A VHDL description of these formulas.

```
entity TwoBitAdder is
   port(x1,x0,y1,y0: in bit;
        s1,s0: out bit);
end entity;
architecture Formulas of TwoBitAdder is
   signal w, z: bit;
begin
   s0 <= (y0 and (not x0)) or ((not y0) and x0);
   w  <= (y1 and (not x1)) or ((not y1) and x1);
   z  <= (y0 and x0);
   s1 <= (w  and (not z)) or ((not w) and z);
end Formulas;
```

The Boundary of the two bit adder

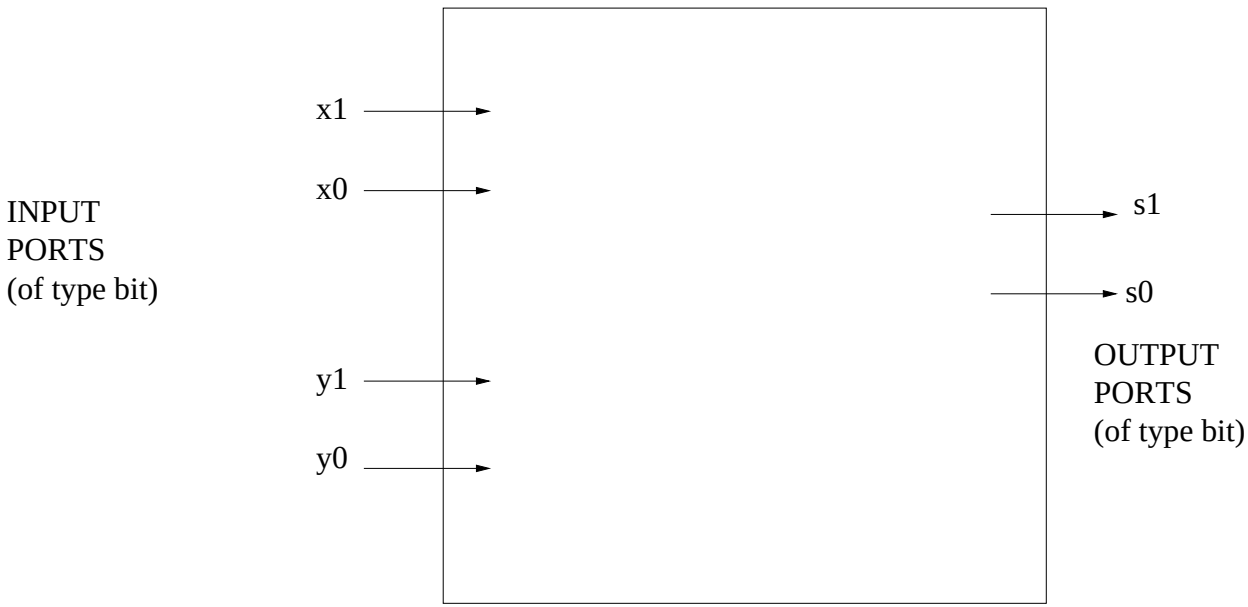

Figure : Boundary of the two bit adder logic circuit

The Entity Design-unit

```
entity TwoBitAdder is
   port(x1,x0,y1,y0: in bit;
        s1,s0: out bit);
end entity;
```

Defines the interface of your system with the outside world.

The Interior of the two bit adder

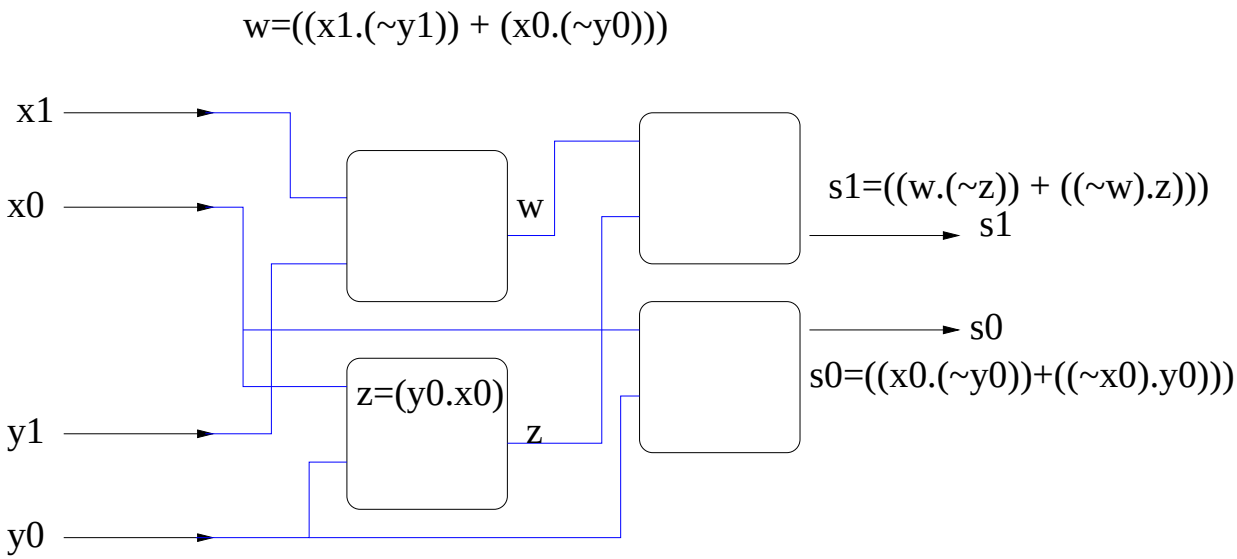

Figure : Interior of the two bit adder logic circuit

Ports and Signals

These are models for *wires* and

- At every time instant, a port/signal has a value from the set of possible values defined by their type.

```
type bit is {'0','1'};
```

- Ports/signals are driven by drivers (to be discussed) and can be sensed at each time instant (note: output ports cannot be read inside the architecture).

Drivers

The concurrent statement:

```
s0 <= (y0 and (not x0)) or ((not y0) and x0);
```

defines a *driver* which forces a value on the port s0.

- A driver is evaluated whenever there is an event on one of the objects on which it depends.
- The evaluated value of the RHS expression is then forced on to the object being driven.

So drivers are like *gates*, whose functionality is decided by the driver expression.

Expressions

Expressions such as

```
(y0 and (not x0)) or ((not y0) and x0)
```

are constructed using operators (functions) such as and/or/not/xor (there are more!).
These operators are strongly typed (and can be overloaded based on both argument and return types).

VHDL Simulation Algorithm

```
At t=0, evaluate all drivers (in any order),
put all transactions in a transaction queue.
```

VHDL Simulation Algorithm: continued

```
while (transaction-queue is non-empty) {

   move time forward to that corresponding
   to the transaction at the head of the queue.

   apply all transactions active at this
   time (in any order).

   execute all drivers which are triggered
   by events as a result of the applied transactions.
   Add resulting transactions to transaction-queue.
}
```

Observations

- The VHDL simulation algorithm is unambiguous.
- Drivers can be defined in any order, they are concurrent (that is, their behaviour at a particular instant is independent of each other).
- The algorithm attempts to mimic *ideal* gates which have a delay δ.
- Delays can be associated with a driver:

```
w <= (y0 and x0) after 5 ns;
```

VHDL Simulation Algorithm: example

```
entity Top is
  port (c: out bit);
end entity Top;
architecture Absurd of Top is
  signal a, b: bit;
begin
  a <= not a after 5 ns;
  b <= not a;
  c <= a or b;
end Absurd;
```

Recap

Do we understand the following concepts?

- Entity.
- Port.
- Type.
- Signal.
- Waveform.
- Driver.
- Transaction.
- Simulation Algorithm.

Simulating the TwoBitAdder

Write a test-bench:

```
entity Testbench is
end entity;
architecture Behave of Testbench is
  signal x0,x1,y0,y1,s0,s1: bit := '0';
  component TwoBitAdder
     port(x0,x1,y0,y1: in bit; s0,s1: out bit);
begin
   -- set values for x0,x1,y0,y1

   -- instantiate the adder.
   dut: TwoBitAdder
      port map(x0 => x0, x1 => x1,
               y0 => y0, y1 => y1,
                s0 => s0, s1 => s1);
end Behave;
```

A closer look at the Testbench

- A design unit which *instantiates* the entity/architecture that we wish to test (the DUT).
- Produces events on the DUT input ports and observes the response on the DUT output ports.

Check out the testbench provided in the VHDL sample.

Process statements with WAIT

This is a special kind of driver, which is very useful in constructing test-benches. As an example:

```
process
begin
   a <= '1';
   wait for 5 ns;
   a <= '0';
   wait for 5 ns;
end process;
```

This produces a sequence of transactions on the signal **a**.
In general, one can describe very complicated sequences using this construct (as an example, see the sample VHDL code).

Using a simulator

GHDL (free.ghdl.fr) is an open source VHDL simulator. The basic flow when using this simulator is

- Compile the VHDL files.
- Build a simulation executable.
- Run the simulation executable (dump waveforms if you wish).
- Observe the waveforms (using the public-domain waveform viewer gtkwave).

Looking ahead

- Structural descriptions.
  - VHDL models of commonly occurring logic building blocks.
  - Interconnections of logic building blocks to construct more complex blocks.
- Behavioural descriptions and writing your own functions.
- Sequential system description: state-machines, register-transfer-level descriptions, memories etc.
- The verification problem: WRITING TESTBENCHES!

Useful reading material

- D. Perry, "VHDL Programming by Example", Tata McGraw-Hill.
- P. Ashenden, "A VHDL Tutorial", hep.uchicago.edu/ tangjian/SVT_sub/FTK_ATLAS/.../vhdl-tutorial.pdf.