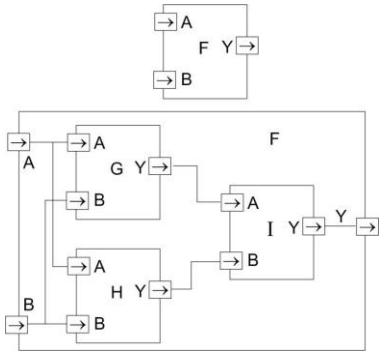
Modelling of discrete events using Verilog language

[1]Dr. T.C. Manjunath, [2]Rajashekar M. Koyyeda, [2]Satvik M. Kusagur, [2]Pavithra G., [2]Arunkumar M.
[1]Prof. & Head, ECE Dept., DSCE, Bangalore, Karnataka,
[2]Research Scholars, VTU RRC, Belagavi, Karnataka

Extended Abstract : Design is a complex process which can be thought of as a top down refinement of a specification. The main aim of this algorithm is to propose a VHDL code generator for real-time systems. Its importance lies in giving graphical interface to the designer. Here, a discrete event model for a digital circuit using VHDL is designed and implemented.

Introduction: VHDL *Verilog Hardware Description Language is* a language for describing digital electronic systems. A VHDL is designed in this paper to fill a number of needs in the design process. Firstly, it allows description of the structure of a design, i.e. how it is decomposed into sub-designs, and how those sub-designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping.

A digital electronic system is designed as a module with inputs and/or outputs. The electrical values on the outputs are some function of the values on the inputs. Figure below shows an example of this view of a digital system as a structural description. The module F has two inputs, A and B**,** and an output Y. Using VHDL terminology, we call the module F a design *entity*, and the inputs and outputs are called *ports.* One way of describing the function of a module is to describe how it is composed of sub-modules. Each of the sub-modules is an *instance* of some entity, and the ports of the instances are connected using *signals.* Figure also shows how the entity F might be composed of instances of entities G, H and I. This kind of description is called a *structural* description. Note that each of the entities G, H and I might also have a structural description.



Describing Behavior: In many cases, it is not appropriate to describe a module structurally. One such case is a module which is at the bottom of the hierarchy of some other structural description. For example, if you are designing a system using IC packages bought from an IC shop, you do not need to describe the internal structure of an IC. In such cases, a description of the function performed by the module is required, without reference to its actual internal structure. Such a description is called *a functional* or *behavioral* description. To illustrate this, suppose that the function of the entity F in figure above is the exclusive-or function. Then a behavioral description of F could be the Boolean function:

$$Y = \overline{A}.B + A.\overline{B}$$

More complex behaviors cannot be described purely as a function of inputs. In systems with feedback, the outputs are also a function of time. VHDL solves this problem by allowing description of behavior in the form of an executable program.

Discrete Event Time Model : Once the structure and behavior of a module have been specified, it is possible to simulate the module by executing its behavioral description. This is done by simulating the passage of time in discrete steps. At some simulation time, a module input may be stimulated by changing the value on an input port. The module reacts by running the code of its behavioral description and scheduling new values to be placed on the signals connected to its output ports at some later simulated time. This is called scheduling a *transaction* on that signal. If the new value is different from the previous value on the signal, an *event* occurs, and other modules with input ports connected to the signal may be activated.

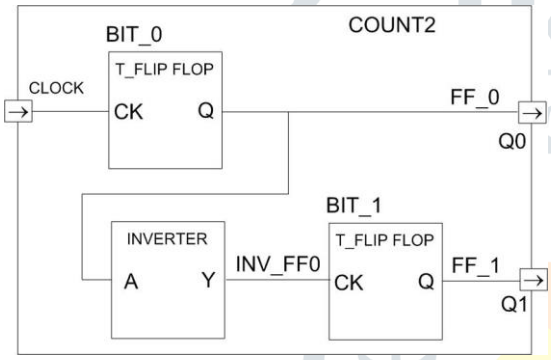
The simulation starts with an *initialization phase,* and then proceeds by repeating a two-stage *simulation cycle.* In the *initialization phase,* all signals are given initial values, the simulation time is set to zero, and each module's behavior program is executed. This usually results in transactions being scheduled on output signals for some later time. In the *first stage* of a simulation cycle, the simulated time is advanced to the earliest time at which a transaction has been scheduled. All transactions scheduled for that time are executed, and this may cause events to occur on some signals. In the *second stage,* all modules which react to events occurring in the first stage have their behavior program executed. These programs will usually schedule further transactions on their output signals. When all of the behavior programs have finished executing, the simulation cycle repeats. If there are no more scheduled transactions, the whole simulation is completed.

The purpose of the simulation is to gather information about the changes in system state over time. This can be done by running the simulation under the control of a *simulation monitor*. The monitor allows signals and other state information to be viewed or stored in a trace file for later analysis. It may also allow interactive stepping of the simulation process, much like an interactive program debugger. We start the description of an entity by specifying its external interface, which includes a

description of its ports. So, the counter is defined as:

```
entity count2 is
    generic (prop_delay: Time := 10 ns);
    port (clock: in bit;
        q1, q0: out bit);
end count2;
```

This specifies that the *entity* count2 has one input and two outputs, all of which are bit values, that is, they can take on the values '0' or '1'. It also defines a generic constant called prop_delay which can be used to control the operation of the entity (in this case its propagation delay). If no value is explicitly given for this value when the entity is used in a design, the default value of 10 ns will be used. An implementation of the entity is described in an architecture body. There may be more than one architecture body corresponding to a single entity specification, each of which describes a different view of the entity. The behavioral description of the counter is written as:




```
architecture behavior of count2 is
begin
  count_up: process (clock)
    variable coun_value : natural := 0;
  begin
    if clock = '1' then
      count_value:= (count_value + 1) mod 4;
      q0 <= bit'val(count_value mod 2) after prop_delay;
      q1 <= bit'val(count_value / 2) after prop_delay;
    end if;
  end process count_up;
end behavior;
```

In this description of the counter, the behavior is implemented by a process called count_up, which is sensitive to the input clock. A process is a body of code which is executed whenever any of the signals it is sensitive to changes value. This process has a variable called count_value to store the current state of the counter. The variable is initialized to zero at the start of simulation, and retains its value between activations of the process. When the clock input changes from '0' to '1', the state variable is incremented, and transactions are scheduled on the two output ports based on the new value. The assignments use the generic constant prop_delay to determine how long after the clock change the transaction should be scheduled. When control reaches the end of the process body, the process is suspended until another change occurs on clock. The two-bit counter is also designed alternatively as a combination of 2 flops and an inverter, as shown in figure below. This can be written in VHDL.

In this architecture, two component types are declared, t_flipflop and inverter, and three internal signals are declared. Each of the components is then instantiated, and the ports of the instances are mapped onto signals and ports of the entity. For example, bit_0 is an instance of the t_flipflop component, with its ck port connected to the clock port of the count2 entity, and its q port connected to the internal signal ff0. The last two signal assignments update the entity ports whenever the values on the internal signals change.

References

[1] Ganesh Babu T.R., Ganesh S., Shenbagadevi, "Automatic detection of glaucoma using fundus image", European Jour. of Scientific Res., ISSN 1450-216X, Vol. 59, Issue 1, pp. 22-32, 2011.

[2] Murthi A. and M. Madheswaran, "Enhancement of optic cup to disc ratio detection in glaucoma diagnosis", IEEE Int. Conf. on Comp. Communication & Informatics (ICCCI-12), Coimbatore, Tamil Nadu, India, pp. 1-5, Jan. 10-12, 2012.

[3] Tan N.M., J. Liu, D.W.K. Wong, F. Yin, J.H. Lim, T.Y. Wong, "Mixture model-based approach for optic cup segmentation", IEEE Annual Int. Conf. on Engg. in Medicine & Biology Society (EMBC), Buenos Aires, Argentina, pp. 4817 – 4820, 31 Aug-4 Sept. 2010.

[4] Yuji Hatanaka, Atsushi Noudo, Chisako Muramatsu, Akira Sawada, Takeshi Hara, Tetsuya Yamamoto, Hiroshi Fujita, "Automatic measurement of cup to disc ratio based on line profile analysis in retinal images", 33rd Annual Int. Conf. of the IEEE Engg. in Medicine & Biology Society, pp. 3387 – 3390, Boston, MA, USA, 30 Aug.-3 Sept. 2011.

[5] Fengshou Yin, Jiang Liu, Damon Wing Kee Wong, Ngan Meng Tan, Carol Cheung, Mani Baskaran, Tin Aung, Tien Yin Wong, "Automated segmentation of optic disc and optic cup in fundus images for glaucoma diagnosis", 25th IEEE Int. Symp. on Comp. Based Medical Systems (CBMS), Rome, Italy, pp. 1-6, 20-22 Jun. 2012.

[6] Sandra Morales, Valery Naranjo, Jesús Angulo, Mariano Alcañiz, "Automatic detection of optic disc based on PCA and mathematical morphology", IEEE Trans. on Medical Imaging, Vol. 32, Issue 4, pp. 786-796, Apr. 2013.

[7] Jun Cheng, Jiang Liu, Yanwu Xu, Fengshou Yin, Damon Wing Kee Wong, Ngan-Meng Tan, Dacheng Tao, Ching-Yu Cheng, Tin Aung, Tien Yin Wong, "Superpixel classification based optic disc and optic cup segmentation for glaucoma screening", IEEE Trans. on Medical Imaging, Vol. 32, Issue 6, pp. 1019-1032, Jun. 2013.

[8] Aquino, Arturo, Manuel Emilio Gegúndez-Arias, and Diego Marín, "Detecting the optic disc boundary in digital fundus images using morphological, edge detection, and feature extraction techniques", IEEE Transactions on Medical Imaging, Vol. 29, Issue 11, pp. 1860-1869, Nov. 2010.