VHDL Tutorial: Learn by Example

-- by Weijun Zhang, July 2001

Typical Combinational Components

The following behavior style codes demonstrate the concurrent and sequential capabilities of VHDL. The *concurrent statements* are written within the body of an architecture. They include *concurrent signal assignment*, *concurrent process* and *component instantiations (port map statement)*. *Sequential statements* are written within a *process* statement, *function* or *procedure*. Sequential statement includes *case statement*, *if-then-else statement* and *loop statement*.

Multiplexor	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic
Decoder	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic
Adder	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic
Comparator	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic
ALU	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic
Multiplier	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic

---

Latch & Flip-Flops

Besides from the circuit input and output signals, there are normally two other important signals, *reset* and *clock*, in the sequential circuit. The reset signal is either *active-high* or *active-low* status and the circuit status transition can occur at either clock *rising-edge* or *falling-edge*. Flip-Flop is a basic component of the sequential circuits.

Simple Latch	Behavior Code	Test Bench	Behavior Simulation
D Flip-Flop	Behavior Code	Test Bench	Behavior Simulation
JK Flip-Flop	Behavior Code	Test Bench	Behavior Simulation

Typical Sequential Components

Typical sequential components consist of registers, shifters and counters. The concept of *generics* is often used to parameterize these components. Parameterized components make it possible to construct *standardized libraries* of shared models. In the behavioral description, the output transitions are generally set at the clock rising-edge. This is accomplished with the combination of the VHDL *conditional statements* (clock'event and clock='1'). During the testbench running, the expected output of the circuit is compared with the results of simulation to verify the circuit design.

Register	**Behavior Code**	**Test Bench**	**Behavior Simulation**	**Synthesis Schematic**	**Structural Simulation**
Shift Register	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic	Structural Simulation
Counter	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic	Structural Simulation

---

Sequential Logic Design

The most important description model presented here may be the *Finite State Machine (FSM)*. A general model of a FSM consists of both the combinational Logic and sequential components such as state registers, which record the states of circuit and are updated synchronously on the rising edge of the clock signal. The output function computes the various outputs according to different states. Another type of sequential model is the memory module, which usually takes a long time to be synthesized due to the number of design cells.

FSM Model	**Behavior Code**	**Test Bench**	**Behavior Simulation**	**Synthesis Schematic**

- **Memories**

RAM Module	**Behavior Code**	**Test Bench**	**Behavior Simulation**	**Synthesis Schematic**
ROM Module	Behavior Code	Test Bench	Behavior Simulation	Synthesis Schematic

---

- **Behavior vs. RTL Synthesis** (Y Chart)

RTL stands for *Register-Transfer Level*. It is an essential part of *top-down* digital design process. Logic synthesis offers an automated route from an RTL design to a *Gate-Level* design. In RTL design a circuit is described as a set of registers and a set of transfer functions describing the flow of data between the registers, (ie. *FSM + Datapath*). As an important part of a complex design, this division is the main objective of the hardware designer using synthesis. The Synopsys Synthesis Example illustrates that the RTL synthesis is more efficient than the behavior synthesis, although the simulation of previous one requires a few clock cycles.

GCD Caculator	**Behavior Code**	**RTL Code (FSM+D)**	**Comparison**

Following section illustrates the *RTL (FSM+Datapath)* method further using several design examples.

Custom Single-Purpose Processor Design

The first three examples illustrate the difference between *RTL FSMD model* (*Finite State Machine with Datapath buildin*) and *RTL FSM + DataPath model*. From view of RT level design, each digital design consists of a *Control Unit* (FSM) and a *Datapath*. The datapath consists of storage units such as registers and memories, and combinational units such as ALUs, adders, multipliers, shifters, and comparators. The datapath takes the operands from storage units, performs the computation in the combinatorial units, and returns the results to the storage units during each state. This process typically takes one or two clock cycles.

*Data-flow (*looks more like an Algorithm*) modeling* is presented in the fourth example. The FIR digital filter algorithm is simulated and synthesized using VHDL. A comparison of the coding styles between the *RTL modeling* and *Algorithm level modeling* highlights the different techniques.

- **GCD Calculator**

FSMD Modeling	**RTL Code**	**Test Bench**	**RTL Code Simulation**	**Synthesis Schematic**
FSM + Datapath Modeling	RTL Code	Test Bench	RTL Code Simulation	Synthesis Schematic

- **ISA Bus Interface**

FSM + Datapath Modeling	**RTL Code**	**Test Bench**	**RTL Code Simulation**	**Synthesis Schematic**

- **FIR Digital Filter (DSP Example)**

Data-Flow Modeling	**Behavior Code**	**Test Bench**	**Behavior Simulation(1,2)**	**Synthesis Schematic**
