The Ten Commandments of Excellent Design—VHDL Code Examples

Peter Chambers
Engineering Fellow
VLSI Technology

This short paper will give you some VHDL code examples that will help you design synchronous circuits that work first time.

Those Ten Commandments

Just in case you forgot, here are the Ten Commandments of Excellent Design:

1. **All state machine outputs shall *always* be registered**
2. **Thou shalt use registers, never latches**
3. **Thy state machine inputs, including resets, shall be synchronous**
4. **Beware fast paths lest they bite thine ankles**
5. **Minimize skew of thine clocks**
6. **Cross clock domains with the greatest of caution. Synchronize thy signals!**
7. **Have no dead states in thy state machines**
8. **Have no logic with unbroken asynchronous feedback lest the fleas of myriad Test Engineers infest thee**
9. **All decode logic must be crafted carefully—eschew asynchronicity**
10. **Trust not thy simulator—it may beguile thee when thy design is junk**

How to Write Ten-Commandment Code

Conforming to the Ten Commandments is not difficult. In this section you'll see how to write VHDL (your author doesn't do Verilog, but the translation is easy)

© 1997, VLSI Technology

that complies with the rules. Robust design and first-silicon success are the goals!

The philosophy behind Ten-Commandment code is that synthesizers are not to be trusted too much. Most of the code you will see is close to the structural level; some more overtly than others.

Most of the code is self-explanatory. It is assumed that the reader is familiar with VHDL. Signal names are also obvious to anyone “skilled in the art.”

Ten-Commandment Code

How to Create a Flip-Flop

One of the basic primitives that we need to create robust synchronous designs is the D-type flip-flop. Look at the code in Code Sample 1:

```
-- VHDL Code for a D-Type Flip-Flop with an
-- Asynchronous Clear


D_Type_Flip_Flop: process(Reset_n, Clock_In)
    begin
        if (Reset_n = '0') then
            Q_Output <= '0' after 1 ns;
        elsif (Clock_In'event and Clock_In = '1') then
            Q_Output <= D_Input after 1 ns;
        end if;
end process D_Type_Flip_Flop;
```

Code Sample 1. A D-Type Flip-Flop

This flip-flop has the following properties:

- An asynchronous active-low clear input sets the Q output to zero.
- It is triggered on the rising edge of the clock.

How to Create a Latch

While the Ten Commandments specifically forbid the use of latches, there are still those heretics who will insist on the use of latches. The code to instantiate a transparent latch is shown in Code Sample 2:

```
-- VHDL Code for a Transparent Latch

Latch_Data: process(Latch_Open, D_Input)
    begin
        if (Latch_Open = '1') then
            Latched_Data <= D_Input;
-- If Latch_Open = 0, then Latched_Data keeps its old value,
-- i.e. the latch is closed.
        end if;
end process Latch_Data;
```

Code Sample 2. A Transparent Latch

This latch has the following properties:

- A latch control that opens the latch when high (the latch is transparent).

How to Create a Metastable-Hardened Flip-Flop

The use of a metastable-hardened flip flop is nothing more than the direct instantiation of a suitable library element—in this case, a “dfntns” flip-flop. This is pure structural VHDL. The component declaration is shown in Code Sample 3:

```
-- VHDL Code for a Nice Metastable-Hardened Flip-Flop

component dfntns
    Port (
        CP  : In    std_logic;
        D   : In    std_logic;
        Q   : Out   std_logic
    );
end component;
```

Code Sample 3. **A Metastable-Hardened Flip-Flop, Component Declaration**

To use the flip-flop in your circuit, instantiate it as shown in Code Sample 4:

```
-- VHDL Code to Instantiate the Metastable-Hardened Flip-Flop

Metastable_Hardened_Flip_Flop_Please: dfntns port map (
    D   =>  D_Input,
    CP  =>  Clock_In,
    Q   =>  Q_Output
);
```

Code Sample 4. **A Metastable-Hardened Flip-Flop, Instantiation**

This flip-flop has the following properties:

- A maximum clock-to-out time under worst-case setup and hold time violations. This time is available in the library element specifications.

The Care and Feeding of Toggle Signals

Receiving a Toggle Signal

The Ten Commandments paper suggested that a method for exchanging single-point information across clock domains is by the use of toggle signals. Here, it is assumed that the toggle event should generate an active-high pulse to pass to a state machine. Every toggle—rising edge and falling edge—must create the pulse. In addition, the pulse must be synchronized correctly to the receiver's clock. The code to accomplish this is shown in Code Sample 5:

```
-- VHDL Code to Create a Pulse from an Asynchronous
-- Toggle Signal


-- First, use a metastable-hardened flip-flop to synchronize the
-- toggle input
Metastable_Hardened_Flip_Flop_Please: dfntns port map (
    D   =>  Handshake_T,
    CP  =>  Clock_In,
    Q   =>  Sync_Handshake_T
);


-- Now pass the synchronized toggle through another flip-flop
Toggle_Reg_Proc: process(Clock_In)
    begin
        if (Clock_In'event and Clock_In = '1') then
            Reg_Handshake_T <= Sync_Handshake_T after 1 ns;
        end if;
end process Toggle_Reg_Proc;


-- Finally XOR the two synchronized signals to create a pulse
Toggle_Pulse <= Reg_Handshake_T xor Sync_Handshake_T;
```

Code Sample 5. **Receiving a Toggle Signal**

When synthesizing this code, remember to use the “fix hold” option so a fast path doesn't occur between the two flip-flops in this circuit.

Generating a Toggle Signal

Recall that a toggle signal is generated by simply inverting a level to pass the information. The trivial code to do this is shown in Code Sample 6. The suffix "_T" is used to denote a toggle signal.

```
-- VHDL Code to Create a Toggle Signal

Handshake_T    <= not (Handshake_T) after 1 ns;
```

Code Sample 6. **Generating a Toggle Signal**

The Beginner's Guide to State Machines

Introduction

The creation of state machines is a mixture of art and science. A well-crafted state machine will possess a sense of elegance; it will be appealing, both functionally and visually.

Here, a very simple example is presented as an illustration of state machine design. The state diagram for the Flintstones State Machine is shown in Figure 1

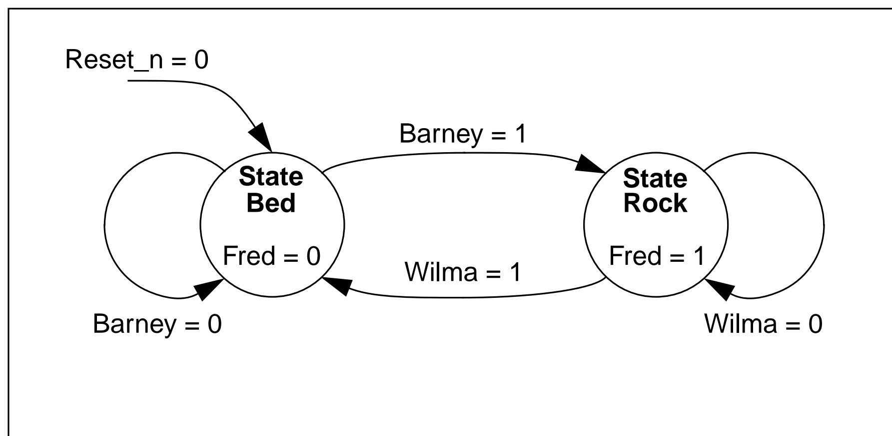

Figure 1. **The Flintstones State Machine**

The Flintstones State Machine operates as follows:

1. The State Machine has two states, State Bed and State Rock.
2. There is one output, Fred, which takes the value 0 in State Bed and 1 in State Rock.
3. A reset, caused by a low level on Reset_n, puts the State Machine into State Bed.
4. The State Machine waits in State Bed while Barney is low, and enters State Rock when Barney goes high.
5. The State Machine then waits in State Rock while Wilma is low, and returns to State Bed when Wilma goes high.

Implementing the Flintstones State Machine

An example implementation of the Flintstones State Machine is shown in Code Sample 7 and Code Sample 8

```
-- VHDL Code to Implement the Flintstones State Machine

Flintstones_SM_Proc: process(Sync_Reset_n, Clock_In)

-- Enumerate state types:
type    Flintstones_Statetype is (
            Bed, Rock
        );

-- define the state variable:
variable    Flint_State:    Flintstones_Statetype;

-- Here's the state machine:
begin
-- Define the asynchronously set reset states...
    if (Sync_Reset_n = '0') then
        Fred        <= '0' after 1 ns;
        Flint_State := Bed
-- Default conditions for each output, in this case identical to the
-- reset state:
    elsif (Clock_In'event and Clock_In = '1') then
        Fred        <= '0' after 1 ns;
-- Here are the state transitions:
    (Continued on next Code Sample listing)
```

Code Sample 7. **Implementation of the Flintstones State Machine (First Part)**

```
        case Flint_State is
            when Bed =>
-- Transition from Bed to Rock:
                if (Barney = '1') then
                    Fred         <= '1' after 1 ns;
                    Flint_State  := Rock;
-- Holding term in Bed:
                else
                    Flint_State  := Bed;
                end if;

            when Rock =>
-- Transition from Rock to Bed:
                if (Wilma = '1') then
                    Fred         <= '0' after 1 ns;
                    Flint_State  := Bed;
-- Holding term in Rock:
                else
                    Fred         <= '1' after 1 ns;
                    Flint_State  := Rock;
                end if;

-- Default term for dead states:
            when others =>
                Flint_State  := Bed;
        end case;
    end if;
end process Flintstones_SM_Proc;
```

Code Sample 8. **Implementation of the Flintstones State Machine (Second Part)**

Notes on the State machine Implementation

For the most part, the Flintstones State Machine's operation should be clear. A few points are worth noting, however:

1. The reset signal (Sync_Reset_n) is synchronized with Clock_In before being sent to the State Machine.
2. Barney and Wilma must also be synchronous to Clock_In; at the very least, there must be an assurance that the State Machine's state and output register's setup and hold times are not violated.
3. This design assigns a default value to each output and to the state variable before entering the case statement. This ensures that only those signals that are not taking default (usually inactive) values need be listed in the case statement. This is optional; it is entirely reasonable to list every signal under each transition term, including inactive signals.
4. Note that the output signal Fred comes directly from a D-type flip-flop: it is not a decode of the state variable. This ensures Fred's cleanliness (so to speak).
5. The "when others" in the case statement handles the possibility that the State Machine might end up in a dead state.

Conclusions

The code examples in this document should be considered as examples only. There are many ways to code excellent VHDL; this code is a place to start. If you have a neat snippet of VHDL to add to the list, please contact the author!

Contact Information

Here's how to contact the author:

Peter Chambers

VLSI Technology, Inc.
8375 South River Parkway, M/S 250
Tempe, Arizona 85284

Phone: 602 752 6395

Email: peter.chambers@vlsi.com