

# Going the Other Way or How Do You Generate Behavioral VHDL Code From Structure ?

Mehmet A. Cirit  
MAC Associates  
18837 Casa Blanca Lane  
Saratoga, California 95070

## **Abstract**

As VHDL becomes more popular and gets coupled with behavioral synthesis, the question of integrating the immense design data bases built around schematic capture, preserving the investments already made into traditional design methodologies, and utilizing them to the best of one's advantage become of paramount importance in a transition environment. To address some of these important questions, we present **HDLsmith**, a VHDL code generator. In addition to making it possible to recapture existing designs in a pure behavioral model, **HDLsmith** makes existing designs compliant with MIL-STD 454L. The generated code can be used for accelerating functional design verification, and as a basis for formal verification against a functional specification.

## **Section 1. Introduction**

Behavioral modeling has proved itself to be an indispensable design methodology as the design complexity increased tremendously. Along with the increased complexity, the competitive pressures for shorter time to market and further constraints on tighter design cycles are also emerging patterns. The enabling technologies which help in this process are:

1. Increasing use of the synthesis to reduce the design time,
2. Increasing reliance on system level simulation to verify the design,
3. Increasing design reusability.

In this paper we shall introduce a new design tool, **HDLsmith**, which could help reduce the design costs by providing the necessary links between the items listed above. In a nut shell, **HDLsmith** is a behavioral VHDL

3. Which signals should be eliminated completely or partially. In the latter case, only at a limited number of locations a signal substitution is performed. This could enhance legibility and compactness the model considerably.
4. How to organize the model hierarchy. There is a range of hierarchic modeling options, which extend from flat to completely hierarchic. Some sub-blocks can be optionally flattened or preserved.
5. How to name the basic bit and bit\_vector types, and what VHDL simulator interface to use. Not every VHDL simulator accepts the same inputs. Depending on the simulator used, HDLsmith can generate slightly different code to make it acceptable to the host simulator.

Several iterations may be necessary to assess the impact of the various commands on the readability and compactness of the model. Using a SUN SPARC work-station, one can generate a model for 5000 gate design under one minute. Internally, the model generation proceeds in three steps:

- Identify the signal to be used in the model.
- Extract the relationships between them.
- Merge the relationships together using the bussing techniques to compact the model.

The ability of HDLsmith to generate a higher level representation from a structural netlist qualifies it as a *reverse engineering* tool. Anybody who studied schematic representation of circuit diagrams can testify how difficult it is to grasp functionality at one sight even for a relatively trivial case. When there are hundreds of pages of schematic to connect together, the situation becomes impossible. HDLsmith can reduce each page of schematic into a few VHDL statements. By converting a design from pictures into data flow and behavioral model, its functionality can be grasped faster. New engineers can climb up the learning curve faster. By extracting and decoding the functionality of the design in a standard hardware description language, projects can be carried out faster even in the face of high employee turn around.

#### Section 4. Conclusions

For VHDL conversion, recapturing design and product knowledge base, HDLsmith charts a smooth transition path by providing an effective method. It enhances the reliability and maintainability of your products, lets your design teams go up the learning curve faster.

Model development is a costly process. As usually is the case, design specifications may be unclear, out of date, misleading, or simply misstated on purpose. In addition, they need to be verified by exhaustive simulation. As a result of the automatic model generation, one gets the unique opportunity to compare a design against its specification. This may uncover problems traditional verification methods may not be able to catch.

HDLsmith generates data flow and register transfer level behavioral VHDL code in compliance with DoD requirements. The model generated by HDLsmith is a very accurate representation of the gate level design. It can be used as the basis for enhancements and modifications, to the original design, as well as a documentation of the design. It can be resynthesized and simulated. Signal and variable names used in the model are directly

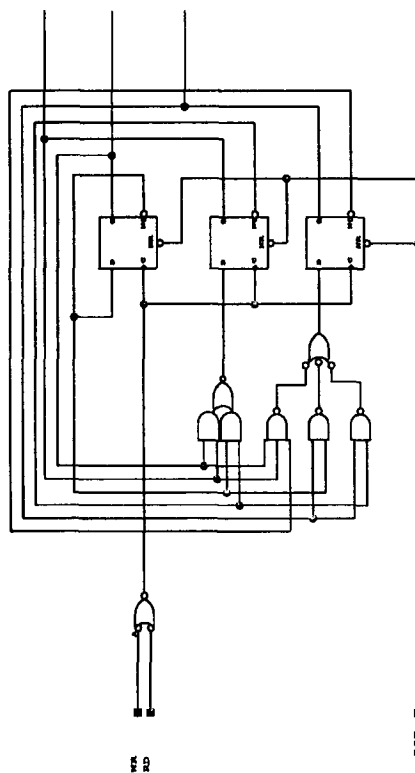


Figure 2: Schematic of a sample circuit. Automatically generated models are shown in the following pages.

derived from the input netlist. Therefore, there is always a direct correspondence between the signal names used in the model and the schematic netlist.

Behavioral model development time for existing designs can be reduced drastically by using HDLsmith. It takes the pain away from retrofitting models to already designed chips by automating the process. With HDLsmith, a model developer need not worry about the logic and the data flow, which are extracted from the netlist, or whether the model is going to be accurate and representative of the original. Models generated by HDLsmith are correct by design.

As a result, HDLsmith can be used for design analysis, design auditing, and for design capture as opposed to “schematic capture” to make it platform independent. For complex designs, the functionally accurate VHDL model can be used to shorten logic simulation and verification period using a behavioral VHDL simulator. Model generation and integration with synthesis become painless tasks with HDLsmith. The complexity of a design could be reduced by orders of magnitude with HDLsmith. A design with thousands of gates may be reduced to a few hundred lines of structured register transfer level VHDL code, an order of magnitude reduction in complexity.

```

use work.hdlsmithbit.all;
entity cntr is
port (
    q : inout bit_vector (0 to 2) ;
    rd : in bit;
    rst : in bit;
    wr : in bit
);
end cntr;
architecture behavior of cntr is
    signal rd_and_wr : bit;
    signal s : bit_vector (0 to 2);
    signal t : bit_vector (0 to 1);
begin
process(q,rd,s(0),t,wr)
begin
    s(2) <= (((bool2bit(q(0 to 1) = "11") and t(1))
        or (q(2) and s(0))) or (q(2) and t(0)));
    s(1) <= ((bool2bit(q(0 to 1) /= "11")
        and (not s(0) or not t(0)));
    rd_and_wr <= (rd and wr);
end process;
proc_0 : process(rd_and_wr,rst)
begin
    if rst = '0'
    then
        q <= "000";
        s(0) <= '1';
        t <= "11";
    elsif (rd_and_wr'event and rd_and_wr = '1')
    then
        q <= s;
        s(0) <= not s(0);
        t <= not s(1 to 2);
    end if;
end process;
end behavior;
configuration ccntr of cntr is
    for behavior
    end for;
end ccntr;

```

Figure 3: VHDL model generated by HDLsmith. The intermediate signal *s* has been renamed and reordered so that assignment to *q* is as compact as possible, eliminating the need for concatenation operator. Comments which map the signal names to schematics are not shown.

```

use work.hdlsmithbit.all;
entity cntr is
port (
    q : inout bit_vector (0 to 2) ;
    rd : in bit;
    rst : in bit;
    wr : in bit
);
end cntr;
architecture behavior of cntr is
    signal din : bit_vector (0 to 5);
begin
process(din(0 to 1),din(4),q,rd,wr)
begin
    din(2) <= (((din(4) and bool2bit(q(0 to 1) = "11"))
        or (din(0) and q(2))) or (din(1) and q(2)));
    din(3) <= ((bool2bit(q(0 to 1) /= "11"))
        and (bool2bit(din(0 to 1) /= "11")));
    din(5) <= (rd and wr);
end process;
proc_0 : process(din(5),rst)
begin
    if rst = '0'
    then
        q <= "000";
        din(0 to 1) <= "11";
        din(4) <= '1';
    elsif (din(5)'event and din(5) = '1')
    then
        q <= din(0)&din(3)&din(2);
        din(0 to 1) <= not din(0)& not din(3);
        din(4) <= not din(2);
    end if;
end process;
end behavior;
configuration ccntr of cntr is
    for behavior
    end for;
end ccntr;

```

Figure 4: VHDL model generated by HDLsmith. This is the “raw” model, where the output has not been changed by automatic signal remapping. It is functionally equivalent to the model in the previous example. Comments which map the signal names to schematics are not shown.