1.0.0 Abstract

This paper is based on work done at VLSI Technology on a number of ASIC designs. It addresses several important issues from the ASIC supplier point of view: how to capture the customer's design [intent] in a specification; how to encapsulate the information; how to communicate design issues.

VHDL is used to create a working specification, which is then used by an engineering group. The specification [testbench and models] is expanded and refined through the process. The focus is on creating and maintaining models at different refinement levels. VHDL is used to tie all the tools and models together.

The resulting flow is a step by step design process covering the customer and foundry activities. The complexity of design is simplified by model orientation. The focus is always on producing the next model.

2.0.0 Introduction

ASIC design includes high level, system design as well as low level, physical IC design. High level design issues include functionality, system test strategy, and design specification. IC design issues include layout, routing, and testing.

A model-oriented VHDL flow is shown in Figure 1. This is a simplified view of the flow in that the design is usually broken into parts for many of these steps, and separate testbenches are constructed for those parts. This flow is model-oriented, because we concentrate on how to produce a model at each stage, and how to verify it.

![Model Oriented Design Flow](image)

**FIGURE 1** Model Oriented Design Flow

At each step in ASIC design, a model substitution takes place. This is an important notion: most of the work in creating an ASIC has to do with verifying new models at each step. As new methodologies [like formal verification] emerge, the overall ASIC flow stays about the same, but new tools are added. The VHDL design flow must accommodate new methodologies and tools without causing a lot of rework.

New methodologies and tools will continue to emerge in all phases of design: synthesis, verification, partitioning, design management, technology mapping, layout, etc. One objective of the model-oriented VHDL flow is to be able to adopt
different tools at every level, with a minimum of difficulty.

3.0.0 The Customer/VLSI Interface

It is important to use VHDL wherever possible for specifications. The high level model and testbench are key parts. The high level model contains key behavior and timing information. The testbench contains information about running tests (simulations) on derived models, comparing results to the high level model, and judging whether the derived model is close enough to the ideal model. For obvious reasons, specifications done this way are called executable specifications.

In the VHDL flow, a high level model and a testbench are always required because they make up the design specification. For a User Logical Design, the handoff to VLSI is a gate level model, in combination with the high level model and testbench to prove conformance.

For a “turnkey” design, the handoff is a high level model and the testbench.

![Diagram](image)

**FIGURE 2** Design Specification in VHDL

The timing specification is part of the high level model. Input to output timing is modeled as a simple series of transitions: the output goes to some valid value a specified “maximum time” after a stimulus; it stays at that value a specified “minimum hold time” after another stimulus. After this it should be forced to an unknown value (‘X’ in std_logic). This is so the timing edges on the ideal model can be used to check derived models.

The testbench contains stimulus (either self-generated or read in from vectors); it is able to cosimulate the ideal model against a derived model and check for functional and timing conformance for the stimulus supplied. In the case of turnkey designs, the ideal model could be tested against itself.

The testbench is also used to encapsulate other non-VHDL tools required in the flow. For example, the testbench is able to produce [golden simulator] QSIM driver files (.sim) for “at-speed” and “supersynchronous” design verification. The testbench is also a suitable place to produce driver files for timing verification or formal functional verification.

**VLSI Terminology:**

- Turnkey Design
- Joint Design
- User Logical Design
- User Design

Obviously, how the customer writes the testbench is very important to VLSI. The way in which the high level model is written is also important, as this can impact how easy it is to do layout.
A Model-Oriented Flow for ASIC Design

4.0.0 The High Level Model and Testbench

The high level model and testbench might be created specifically for the ASIC, but more often they come from a system level model. In general the high level model will consist of several interconnected components, roughly representative of the desired layout blocks. The hierarchy should be kept shallow.

In all but the most trivial designs, test considerations are a major part of the design. Test circuitry needs to be introduced as part of the design specification at the high level, and the test methodology for the system is impacted by partitioning choices made at this time. System level models need to be altered to accommodate scan type test circuits.

entity tp is
  port(to_a, to_b: in std_logic_vector;
  out_c: out std_logic_vector);
end tp;
architecture HL of tp is
  component a ...
  component b ...
  component c ...
  begin
  u1:a port map(...
  u2 b port map(...
  u3:c port map(...
end HL;

FIGURE 3 Initial High Level Model (shallow hierarchy)

Note that the ports for the ASIC are all type STD_LOGIC. At this point in the flow, internal signals might be of any type, but the ASIC I/O, and anything interfacing to physical components at the layout block level, needs to be type STD_LOGIC.

At this point the behavior of the model is captured, in the form of a testbench which sequences the model and checks outputs for the proper state transitions. If a testbench exists for the system level model, and data can be captured in an auxiliary file, the creation of a testbench is automated to some degree. A common practise is to create global signals to represent ports of interest in the hierarchy, then post changes; a component, sensitive to these globals, can then write a trace file to be used by a testbench. Note that such a component is not part of components A, B, or C above and it can be removed from the model without changing the system behavior.

It is important to be able to use this model as a specification for the ASIC. JTAG test circuitry and bulk timing needs to be added to create a working specification. JTAG test circuitry should be added as several components, corresponding to the JTAG control section, and at least one block for each interface between the outermost entity and the next level.

4.1.0 16 Bit MAC Example

A 16 bit multiplier-accumulator has a relatively simple architecture which is not complicated by control logic. For simplicity, the JTAG test circuitry has been left out of this design.
A Model-Oriented Flow for ASIC Design

The block diagram shown below is typical of a 16 bit MAC.

FIGURE 4 Multiplier-Accumulator Example (MAC)

The VHDL code for this design consists of 3 files: add2.vhd, mac.vhd, and mr2.vhd. The entity names match the names of the files. This organization is not necessary for VHDL, but it is necessary for RTL synthesis and is common practise [1].

4.1.1 File MAC_HL.VHD:
-- file: mac.hl.vhd
-- multiplier/accumulator example (high level model)
library ieee;
use ieee.std_logic_1164.all;
A Model-Oriented Flow for ASIC Design

use ieee.numeric_std.all;

entity mac_hl is
port(clk,reset_n: in std_logic;
    a,b: in signed(15 downto 0);
    s: out signed(31 downto 0));
end mac_hl;

architecture hilev of mac_hl is
component mr2_hl
port(clk,rst: boolean;a,b:integer;x:out integer);
end component;
component add2_hl
port(clk,rst:boolean;
    a,b:integer;
    s:out integer);
end component;

signal ia,ib,is1,ix: integer;
signal blclk,bclk_d15,breset,breset_d10:boolean;
begin
    -- take care of conversions for I/O
    blclk<= (clk='1');
    breset<= (~reset_n='0');
    ia<= to_integer(a);
    ib<= to_integer(b);
    -- structural
    cmr2: mr2_hl port map(blclk,breset,ia,ib,ix);
    cadd2: add2_hl port map(bclk,breset_d10,ix);
    -- conversion, bulk timing for output
    -- note that outputs show only one transition and do not bounce.
    bclk_d15<= transport blclk after 15 ns;
    breset_d10<= transport breset after 10 ns;
    process(bclk_d15,breset_d10) begin
        if breset_d10 then
            s<= to_signed(0,32); -- initialization
        elsif (bclk_d15'event and bclk_d15) then
            s<= to_signed(is1,32);
        end if;
    end process;
end hilev;

The MAC entity has all conversions necessary to and from the std_logic type, so that the I/ O of this model can be compared with the RTL and lower level models. All of the timing for the MAC is contained in this outermost entity as well, and the outputs are allowed to change only at the specified time of 20 ns.

4.1.2 File MR2_HL.VHD
-- file: mr2_hl.vhd
-- multiplier with registers on input and output
-- entity mr2_hl is
entity mr2_hl is
port(clk,rst: boolean;a,b:integer;x:out integer);
end mr2_hl;
architecture behav of mr2_hl is
signal a1,b1:integer;
begin
process(clk,rst) begin
    if rst then
        a1<=0;
        b1<=0;
        x<=0;
    elsif(clk'event and clk) then
        a1<=a;
        b1<=b;
        x1:= a1*b1;
    end if;
end process;
end behav;

Entity MR2 is a high level VHDL model which uses abstract types: integer and boolean.

There is no timing in this model, so the signal "x" changes immediately after the clock". In VHDL terms, there is a delta delay between cause and effect which can be seen in signal behavior.

It is common to see variables used in behavioral models. The main reason is to speed simulation, because variables have less overhead and simulate faster than signals. A secondary reason is because variables are more familiar to programmers who are used to sequential programming. It is easier to convert algorithms to sequential code with variables than it is to convert to concurrent code. Variables are not used here because the design is too simple.

4.1.3 File ADD2_HL.VHD
-- file add2_hl.vhd
-- adder for MAC_hl
--
entity add2_hl is
port(clk,rst: boolean;
    a,b: integer;
    s: out integer);
end add2_hl;
architecture behav of add2_hl is
begin
process(clk,rst) begin
    if rst then s<=0;
    elsif(clk'event and clk) then
        s<=a+b;
end process;
end behav;
A Model-Oriented Flow for ASIC Design

end if;
end process;
end behav;

Each file, as described so far, contains an entity and an architecture. Together these are called a VHDL "design unit". The VHDL standard does not dictate how many design units can be in a file, but some synthesis tools (Compass included) like to see one design unit in a file, with the file name as <entity_name>.vhd.

![Diagram](image)

**FIGURE 5** A Testbench for the MAC Design

A package is a design unit as well. Package MAC_FUNC is referred to in the context of entity MAC above. A package consists of a header with declarations for functions, components and types, and a package body where the functions and other subprograms are located.

The VHDL standard defines a library context which is not followed by many proprietary tool vendors (like Compass), because these implementations predate the standard in most cases and the library context was not well understood by most vendors until recently. Part of converting to RTL, and managing the testbench, has to do with accommodating different library uses.

4.2.0 A first testbench

The first testbench is what would be completed by the customer for design handoff at the design specification level. The number of tests is known at this point, and the testbench is supplied with a facility to raise or lower test coverage (for example by altering a constant). The testbench is also in a form that makes it easy to concurrently simulate component parts of the design.

In the model-oriented flow, the testbench is maintained and used to verify each model. The testbench is built up as the design progresses. The first testbench exercises the high level model and produces a SIM file for the [golden] QSIM simulator to use later. This shows how to encapsulate NON-VHDL simulators in the testbench.

It is important to incorporate the encapsulated drivers at the beginning. In a conventional ASIC flow, 2 to 3 manweeks are required to convert user test vectors to a useful form. The result is a [test vector] file from 10 to 40 Mb in size. The same work needs to be done for production test vectors.

Model verification is accomplished by concurrent simulation. Outputs of an ideal model are compared, in sequence, with the outputs of the model under test. In practice, the outputs of the ideal model have often been captured ahead of time in a "trace" or "vector" file. During veri-
A Model-Oriented Flow for ASIC Design

Modification, the testbench can then read the stimulus and response from a trace file. Important aspects of the ideal model and testbench are:

1) An initial, verifiable state is established near the beginning of the simulation without relying on VHDL initialization.

2) Timing specifications are embedded in the high level design. Transitions on outputs from the high level design are made to occur at the maximum timing delay, and at the minimum timing delay.

3) The testbench signals used to drive the inputs of models must show a valid state only in the window of interest - i.e. between the setup and hold time specifications.

The reason for rule 1 is to make initialization part of the specification. It is possible for the initialization to take some time, even hundreds of cycles for some input clock, but the start time for verification of the model must be specified and embedded in the testbench.

The reason for rule 2 is to make a specification of the timing of the high level "ideal" model, and to make it possible to maintain this model as timing is discovered in later phases of the flow. Timing must be in the high level, specification model for model verification to work.

The reason for rule 3 is to capture the input timing specifications in a verifiable form.

4.2.1 VHDL Testbench (MAC_TST.VHD)

-- mac_tst.vhd is a testbench for MAC

library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use work.vs_util.all;

entity mac_tst is end;

architecture vflow1 of mac_tst is

component mac_hl
port(clk_reset_n: in std_logic;
     a,b: in signed(15 downto 0);
     s: out signed(31 downto 0));
end component;

signal clk,rst_n: std_logic;
signal a,b: signed(15 downto 0);
signal s_hls_rtl: signed(31 downto 0);

begin

ut:mac_hl port map(clk,rst_n,a,b,s_hl);
utr:mac rtl port map(clk,rst_n,a,b,s_rtl);

process(s_hl)
begin
-- check outputs at the time the ideal model changes
if (now > 1 ns) then
    assert s_rtl = s_hl -- test rtl
    report "s_rtl:" & stdv_to_hex(std_logic_vector(s_rtl)) &
    "expected: " & stdv_to_hex(std_logic_vector(s_hl))
    severity error;
end if;
end process;

file simfile_out : text is out "mac_tst.sim";
constant last_test: integer:=400;
variable rnd_seed: integer:=999;
variable test_count: integer:=0;
variable do_reset boolean:=true;
variable vector_time: time:=0 ns;
variable atmp,btmp: signed(15 downto 0);
variable stmp: signed(31 downto 0);
variable last_s: signed(31 downto 0):=(others=>"1");
variable last_atmp: signed(15 downto 0):=(others=>"1");
variable last_btmp: signed(15 downto 0):=(others=>"1");
variable ctmp,rtmp: std_logic;
variable tflag:integer:=0;
variable lqs:line;

begin
-- header of .sim file:
wstr(simfile_out,
    "#VLSIsim file generated by mac_tst.vhd"
);
wstr(simfile_out,"#"
);
wstr(simfile_out,"#output file: mac_tst.sim"
);
wstr(simfile_out,"#"
);
wstr(simfile_out,"#setup:"""
);
wstr(simfile_out,"set alias L set input low"
);
wstr(simfile_out,"set alias H set input high"
);
wstr(simfile_out,"set alias SV set input vector"
);
wstr(simfile_out,"set alias T test (message)"
);
wstr(simfile_out,"set alias E echo (time)"
);
wstr(simfile_out,"set radix 16"
);
wstr(simfile_out,"#"
);
wstr(simfile_out,"#Begin the simulation"
);
wstr(simfile_out,"#"
);
wstr(simfile_out,"#"
);
wstr(simfile_out,"#"
);
A Model-Oriented Flow for ASIC Design

end if;
vectortime:=vectortime+5 ns;
write(time_string("step ");
write(time_string((vectortime - now)(1 ns));
write(time_string(simfile_out,lqs);
wait for vectortime - now;

-- clock
clock<=clock;
write(time_string(simfile_out,lqs,"clk"); -- write vector (qsim)
wait for vectortime - now;

-- low phase of clock
clock:=0;
write(time_string(simfile_out,lqs,"clk"); -- write vector (qsim)

-- high phase of clock
clock:=clock;
write(time_string(simfile_out,lqs,"clk"); -- write vector (qsim)

-- test outputs now...
if (last_s /= s_hi) then
  write(time_string(simfile_out,lqs,s_hi),"s_hi");
end if;
last_s := s_hi;
do_reset:=false;

if (tflag =10 then
  tflag :=0;
write(time_string("E original sim time: ");
write(time_string(now);write(time_string(simfile_out,lqs);
else tflag:=tflag+1;
end if;

-- perform a test with random numbers
rand_stdv(rand_seed,atmp);

if rand_stdv /= atmp then
  a:=atmp;
write(time_string(simfile_out,lqs,a);last_atmp:=atmp;
end if;
if last_btmp /= btmp then
  b:=btmp;
write(time_string(simfile_out,lqs,b);last_btmp:=btmp;

4.2.2 VS_UTIL package

Numerous operations, such as file I/O, number conversion and parsing need to be done in the testbench. There are common operations which can and should be implemented in functions. The VS_UTIL package is an assortment of such common operations.

The VS_util package is available over internet:

ftp netcom.com

cd rea
A Model-Oriented Flow for ASIC Design

function isabv(val:std_logic_vector) return boolean;
function isab(val:std_logic) return boolean;
end vs_util;

4.2.3 QSIM driver file

A file named “mac_tst.sim” is output by the testbench when a simulation is run. This file is suitable for use by QSIM, the Compass golden simulator, to verify gate level models in NLS or NLE form.

The following listing is a short version of this file:

```vhdl
#VLSIsim file generated by mac_tst.vhd
#
#output file: mac_tst.vhd
#
#setup:
set alias L set input low
set alias H set input high
set alias SV set input vector
set alias T test (message)
set alias E echo (time)
set radix 16
#
#Begin the simulation
#
#
L clk
L reset_n
sv a 'H0000
sv b 'H0000
step 25
T s 'H00000000
H reset_n
sv a 'HEFCC
sv b 'H62D2
step 5
H clk
step 10
L clk
step 10
sv a 'H1F28
sv b 'H2E4E
step 5
H clk
step 10
L clk
step 10
sv a 'HADC4
sv b 'H2F0A
step 5
H clk
```
A Model-Oriented Flow for ASIC Design

5.0.0 Writing and Compiling The RTL Model

The RTL model is the model used for synthesis. Depending on the coding style used in the high level model, there are a few or a lot of changes made in converting to the RTL model. After changes have been made, the RTL model is verified against the high level model.

It is difficult here to list what makes a synthesizable model; synthesis tools are changing and there is a “tools” context which determines how the model is broken down. Nonetheless, the following conversions are made:

- 1) All signal types are changed to types per Std_logic_1164 or closely related types. Variables may still exist and may be of any type the synthesizer can handle.
- 2) Packages used in the high level model are replaced by vendor supplied packages.
- 3) Large processes with large sensitivity lists are broken down into multiple processes with smaller sensitivity lists, corresponding to the ability of the synthesis tool to deal with synchronization. This often forces variables to be turned into signals, because variables cannot communicate between processes.
- 4) Debugging code (e.g. global signals or assert statements or file I/O) has to be surrounded by pragmas to prevent compiling, or simply commented out.
- 5) Pads and level shifters are instantiated. Power pads are also instantiated.
- 6) Synthesis pragmas are placed in the code, and synthesis control files are created.
- 7) RTL or gate models are obtained for any vendor supplied blocks (JTAG, ROM, RAM, etc.).

At this point in the flow it is important to establish:

- 1) realistic timing targets for major components of the design
- 2) load estimates for the output ports of components in the hierarchy
- 3) general load factors and routing factors which determine the choice of gates in synthesis
- 4) which blocks are to be synthesized and optimized; conversely, which blocks are structural
- 5) clocking schemes

6.0.0 The Physical Model

After routing and layout, the design is complete and there is a physical level model.

ASIC layout is also an iterative process, which involves repartitioning and setting positions and sizes of the layout areas. There are usually multiple degrees of accuracy involved in modelling at this level. For example, the Compass tools can either output a model or they can output a timing file to use on the gate level model. These can be output in VHDL or for a golden simulator (QSIM).
The CAE trend is to make backend tools responsible for some aspects of the netlist. In submicron geometries the interconnect effect is very important, and it is better to iterate on a local scale than to try to articulate gate level fixes at the synthesis level. Synthesis tools have to guess at layout effects; this leads to inefficiency because large buffers are used everywhere, whether they are needed or not. New reoptimization tools are becoming available to account for layout effects by altering the netlist. This means that the physical model may not have the same netlist as the gate level.

VITAL compliance and modelling is an orthogonal but related issue. The option to force preservation of the netlist through layout is important. It is always desirable to use acceleratable VITAL models. However, it is important to realize that in general the physical model is different from

7.0.0 Conclusions

VHDL is a general modelling language which can be used in specifications, as well as for synthesis and simulation. Use of the language in specifications is increasing, because VHDL is efficient and has legal semantics (being a standard).

The trend in specification work is to produce executable specifications.

In the future, VHDL will be used to communicate technical specifications between the customer and the ASIC supplier.

8.0.0 References