Development of VITAL - compliant VHDL models for functionally complex devices

A.Poliakov, A.Sokhatski, SEVA Technologies, Inc.

The paper is based on the development experience of VITAL level 0 compliant VHDL models of IDT FIFO memory family IC for the Free Model Foundation (FMF) [http://vhdl.org/fmf/]. The source data for the development was Integrated Device Technology (IDT) datasheets including timing parameters, VITAL specification, the FMF Guide Lines for VHDL VITAL models. Timing data was prepared in the SDF files. The paper dwells upon approaches and examples of describing functionally complex devices in VITAL standard.

Introduction

Let us try to imagine a bright future: a certain chip manufacturer, for example, IDT, manufacturer of memory chips, provides every new developed chip with documentation containing behavioral VHDL and Verilog models which precisely reflect both functionality and timing characteristics of a chip. This models will use the standard format of timing data presentation (the SDF) and corresponding HDL standards as VITAL VHDL. It would give the following advantages to engineers who apply the chips:

1) Behavioral level of the model ensures speedy board simulation.
2) Precise timing data ensures quality verification.
3) The standard timing data format ensures simplicity of uniting timing data and exchange with various tools.
4) The models can be used for research of chip behavior which have been incompletely reflected in the informal documentation (as datasheets).

VITAL consists of the following standardized parts:

1) SDF to VHDL mapping specification
2) Model style constraints and development methodology
3) Timing routines
4) Primitive (AND, OR,…) timing models

Until now VITAL standard was mostly used for functionally simple IC. We shortly describe model structures and requirements for VITAL standard, the FMF style demands.

VITAL level 0 defines the model external interface (VHDL entity). The main purpose is to allow timing back-annotation by transferring data from the SDF file to the VHDL model through generic parameters.

VITAL level 1 defines an internal modeling methodology (VHDL architecture). It specifies additional constraints to ensure support for simulation acceleration. It makes impossible to describe functionally complex device at behavioral level in this standard. However model structure with some changes was used in IDT FIFO memory models.

The FMF Guide Lines describe constraints and style requirements for VITAL-0 and VITAL-1 models as following:

1) Only scalar ports shall be enabled.
2) Reserved words shall be in capital letters.
3) All input ports shall be initially assigned to 'X' and output to 'U'.
4) Low active signals shall carry names with 'Neg' postfix, as qNeg, CSANeg etc.

VITAL - 0 model structure for IDT FIFO memories

We had to solve problems describing complex behavior, using multi-bit bidirectional buses, specific timing parameters. Our approach and the model structure for each section are described below.

Entity declaration structure

Entity declaration is structured in accordance with VITAL level 0 and the FMF requirements. Generics are presented by input delays (tipd prefix), propagation input— output port delays (tpd), timing constraints (tsetup, thold, tperiod, twidth).

Preparing Entity Declaration we came across two problems

1) IDT datasheets describe timing parameters united in groups, for example: tA - Access Time, CLKA posedge to A0-A35 and CLKB posedge to B0-B35. It's possible to present the generic and SDF definition for each path. To decrease model volume we left one representative, as tpd_CLKA_A0 and used alias declaration in architecture part, as

```
ALIAS   tA : VitalDelayType01 IS tpd_CLKA_A0;
```

and further used tA for parameter access.

2) IDT datasheets describe tSKEW parameters which are not timing constraint and which are used to determine behavior like this:

if Skew time between CLKA and CLKB is more than tSKEW1 and FIFO is not full then IRA flag will be deas-serted at the second CLKA positive edge - else it will occur in the third edge.
The same is for tSKEW2 and almost-full flag. Actually tSKEW1and tSKEW2 present delays on the internal comparators. On the other hand, SDF and VITAL determines tSKEW time only as a strict timing constraint for proper functionality. It does not suit the parameters sense. Therefore we had to declare model components SKEW1 and SKEW2, and describe the component VITAL device parameters as tdevice_SKEW1, tdevice_SKEW2. In the SDF file we had to declare the SKEW1 and SKEW2 instances for VITALBuf cell type.

An example of entity declaration and the SDF file structure for IDT723644 model is shown below.

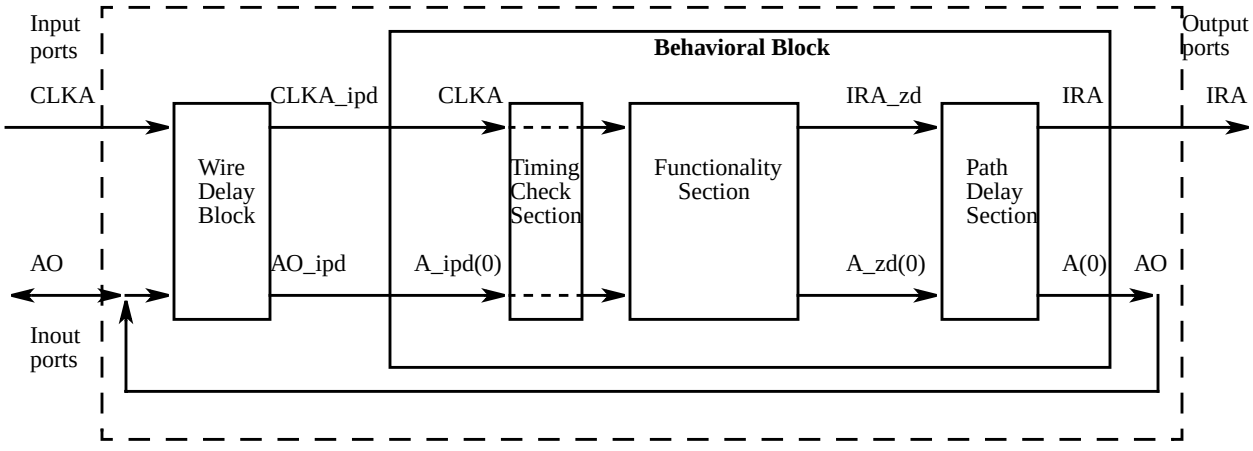
```
LIBRARY ieee;   USE ieee.std_logic_1164.ALL;
                USE ieee.std_logic_arith.ALL;
                USE ieee.vital_primitives.ALL;
                USE ieee.vital_timing.ALL;
LIBRARY fmf;    USE fmf.ff_package.ALL;
                USE fmf.gen_utils.ALL;
ENTITY IDT723644 IS
    GENERIC (
    -- tipd delays            :interconnect path delays
    -- (there must be one generic for each input pin)
        tipd_A0               : VitalDelayType01    := VitalZeroDelay01;
        tipd_A1               : VitalDelayType01    := VitalZeroDelay01;
        ...
        tipd_CLKA             : VitalDelayType01    := VitalZeroDelay01;
        ...
    -- tpd delays             : propagation delays between inputs and outputs
        tpd_CLKA_A0           : VitalDelayType01    := UnitDelay01; -- tA
        tpd_CLKA_IRA          : VitalDelayType01    := UnitDelay01; -- tWFF
        ...
    -- tperiod                : period constraint
        tperiod_CLKA_posedge  : VitalDelayType      := UnitDelay; -- tCLK
        ...
    -- tpw values             : pulse widths
```

```
        tpw_CLKA_posedge      : VitalDelayType      := UnitDelay; -- tCLKL
        tpw_CLKA_negedge      : VitalDelayType      := UnitDelay; -- tCLKH
        ...
    -- tsetup values          : setup times
        tsetup_A0_CLKA        : VitalDelayType      := UnitDelay; -- tDS
        ...
    -- thold values           : hold times
        thold_A0_CLKA         : VitalDelayType      := UnitDelay; -- tDH
    -- tskew values           : skew times
        tdevice_SKEW1         : VitalDelayType      := UnitDelay; -- tSKEW1
        tdevice_SKEW2         : VitalDelayType      := UnitDelay;-- tSKEW2

    -- generic control parameters
        InstancePath          : STRING              := DefaultInstancePath;
        TimingChecksOn        : BOOLEAN             := DefaultTimingChecks;
        MsgOn                 : BOOLEAN             := DefaultMsgOn;
        XOn                   : BOOLEAN             := DefaultXOn;
        TimingModel           : STRING              := DefaultTimingModel);

  PORT (
        A0      : INOUT std_logic;        -- 36 pin bidirectional Port-A data bus
        A1      : INOUT std_logic;        --
        ...
        CLKA    : IN    std_logic  := 'X'; -- clock syncronization for port-A
        ...
        IRA     : OUT   std_logic  := 'U'; -- input ready flag for port A
        ...
        );
  ATTRIBUTE vital_level0 OF IDT723644: ENTITY IS True;
END IDT723644;
```

SDF file structure example shown in the article “Making Verilog models compatible with VHDL VITAL level0 models” by Yuri Tatarnikov

Architecture body structure

Architecture body structure follows the FMF requirements and correlates with VITAL level 1 structure which allows to simplify description and make it more reliable. The block diagram together with VHDL code gives you the idea how the architecture body structure looks like.The figure below shows an example of VITAL Model Structure.




```
ARCHITECTURE vhdl_behavioral OF IDT723644 IS
  ATTRIBUTE vital_level1 OF vhdl_behavioral  : ARCHITECTURE IS False;
  -- Main constants declaration
```

```
    CONSTANT FIFOSize          : POSITIVE     := 1024;
    CONSTANT OffsetSize        : POSITIVE     := 10;
    CONSTANT partID            : String       := "IDT723644";
  -- delayed inputs and bidirectional ports
    SIGNAL A0_ipd              : std_ulogic   := 'X';
    ...
    SIGNAL CLKA_ipd            : std_ulogic   :'X'
  -- Additional signals
    SIGNAL OpenIn, OpenOut : std_logic;
    ...
  -- Aliases for timing parameters
    ALIAS   tA             : VitalDelayType01 IS tpd_CLKA_A0;
    ...
BEGIN
  -- Skew Components
    SKEW1                  : VitalBuf (OpenOut, OpenIn, (tdevice_SKEW1, tdevice_SKEW1));
    SKEW2                  : VitalBuf (OpenOut, OpenIn, (tdevice_SKEW2, tdevice_SKEW2));
  -- Wire Delays Section
    WireDelay              : BLOCK
    BEGIN
    w_1                    : VitalWireDelay (A0_ipd, A0, tipd_A0 );-- Interconnect delay from
    ...                                                            -- port A0 to internal A0_ipd port
    END BLOCK;
 -- Main Behavior Block
    VITALBehavior          : BLOCK ...
END vhdl_behavioral;
```

Wire Delay Block reflects external interconnect delays to the input ports. Skew components are artificial: they are only assigned to transfer *tdevice_SKEW* parameters to the model.

Behavior block structure

The block allows to unite scalar port in buses and separates behavioral part from the others. The block structure is shown below.

```
    VITALBehavior          : BLOCK
    <Port List and Port Map>
    <Zero-delayed Output Signal Declarations>
    <Constant, Type, Internal Signal, Procedures Declarations>
    <Timing Checking Section>
    <Functionality Section>
    <Path Delay Section>
END BLOCK;
```

Block port list and port map

There are the following differences between external port list in entity declaration and the block port list which we used in FIFO models:

1) Entity scalar ports like A0, A1 are united into buses.
2) There are two ports for bidirectional entity ports: the first input port like A_ipd models input interconnect delay, the second output port like A drives value of bidirectional port.

```
PORT(
    A_ipd      : IN std_logic_vector(35 downto 0)    := (OTHERS => 'X');
    A          : OUT std_logic_vector(35 downto 0)   := (OTHERS => 'U');
    CLKA       : IN    std_logic                     := 'X';
    IRA        : OUT   std_logic                     := 'U';
    ... );
PORT MAP (
    A_ipd(0)=> A0_ipd,
```

```
...
A(0)    => A0,
CLKA    => CLKA_ipd,
...
IRA     => IRA,
... );
```

Zero-delayed output signal declarations

The section describes zero-delayed signals for each output and bidirectional ports such as

```
SIGNAL A_zd        : std_logic_vector (35 downto 0);
SIGNAL IRA_zd    : std_logic;
```

Constant, type, internal signal, procedure declarations

This declarative part of behavioral description shows declarations for internal registers and wires, additional constants, types, procedures such as

```
CONSTANTFIFOWordLength: positive         := 36;
SUBTYPE FIFOWord   IS std_logic_vector(FIFOWordLength - 1 DOWNTO 0);
TYPE    FIFOArray   IS ARRAY (0 TO FIFOSize - 1) OF FIFOWord;

SIGNAL FIFOMemory1int  : FIFOArray        := (FIFOArray'range => FIFOWord'(OTHERS => 'X'));
```

Timing checking section

Timing checking section is presented as process TimingChecks. It contains VITALBehavior block input signals in sensitivity list and looks as follows:

```
TimingChecks                  : PROCESS (A_ipd, B_ipd, CLKA,...)
-- Timing Check Variables
  -- Pulse Width Check Variables
    VARIABLE Pviol_CLKA       : X01                 := '0';
    VARIABLE PD_CLKA          : VitalPeriodDataType  := VitalPeriodDataInit;
    ...
  -- Setup/Hold Check Variables
    VARIABLE Tviol_A0_CLKA    : X01 := '0';
    VARIABLE TD_A0_CLKA       : VitalTimingDataType;
    ...
-- Violation variable (used to OR all individual violation variables)
    VARIABLE Violation        : X01                 := '0';
BEGIN
 IF (TimingChecksOn) THEN
  -- CLKA period and pulse width check (high & low)
    VitalPeriodPulseCheck (TestSignal        => CLKA, TestSignalName    => "CLKA",
     Period        => tCLK, PulseWidthHigh   => tCLKH, PulseWidthLow    => tCLKL,
     CheckEnabled  => TRUE, HeaderMsg        => InstancePath & partID,
     PeriodData    => PD_CLKA, XOn           => XOn, MsgOn              => MsgOn,
     Violation     => Pviol_CLKA);
    ...
  -- A/CLKA setup/hold time checks
  VitalSetupHoldCheck (
    TestSignal     => A_ipd, TestSignalName  => "A", RefSignal => CLKA,
    RefSignalName  => "CLKA",SetupHigh       => tDS, SetupLow  => tDS,
    HoldHigh       => tDH,   HoldLow         => tDH,
    CheckEnabled   => (CSANeg = '0') AND      (WRA = '1')
                      AND (ENA = '1'),
   RefTransition   => '/',  HeaderMsg        => InstancePath & partID,
   TimingData      => TD_A0_CLKA, XOn        => XOn,  MsgOn     => MsgOn,
```

```
       Violation       => Tviol_A0_CLKA);
          ...
        Violation      := Pviol_CLKA         OR ...
                          Tviol_A0_CLKA      OR ...

      ASSERT   Violation = '0'
               REPORT   InstancePath & partID & " : signal values may be" &
                        " incorret due timing violation(s)"
               SEVERITY Warning;
     END IF;
   END PROCESS TimingChecks;
```

Two variables should be declared for each timing constraint. They are violation flag Pviol_CLKA and timing data PD_CLKA. Besides VITAL procedure actual parameters, CheckEnable should describe condition when timing constraint should be checked.

Functionality section

This usual behavioral description consists of process and assignment statements which calculates output zero-delayed signals and internal signals.

Path delay section

This section models a delay between internal zero-delayed output signals and external output signals. We used VITAL procedures VitalPathDelay, VitalPathDelay01, VitalPathDelay01Z. The section consists of processes for scalar output ports and generate statements for bus.

The below example shows implementation of path delay from CLKA to scalar signal `IRA`.

```
PathDelayIRA:
   PROCESS (IRA_zd)
     VARIABLE EFAORA_GlitchData   : VitalGlitchDataType;
   BEGIN
     VitalPathDelay01 (OutSignal  => IRA, OutSignalName => "IRA",
        OutTemp                   => IRA_zd, Mode => VitalInertial,
        GlitchData                => IRA_GlitchData,
        Paths=> (
                0 => (InputChangeTime    => CLKA'LAST_EVENT,
                PathDelay                => tREF,
                PathCondition            => TRUE)));
   END PROCESS;
```

The below example shows implementation of path delay for bidirectional 36-bit port A.

```
PathDelaysA_Gen:
    FOR i IN 35 DOWNTO 0 GENERATE
PathDelayA:
    PROCESS (A_zd(i))
      VARIABLE A_GlitchData       : VitalGlitchDataType;
    BEGIN
      VitalPathDelay01Z (OutSignal=> A(i), OutSignalName => "A",
        OutTemp                   => A_zd(i), Mode       => VitalInertial,
        GlitchData                => A_GlitchData,
        Paths=> (
                0 =>   (InputChangeTime   => CSANeg'LAST_EVENT,
                   PathDelay              => tEN_DIS, PathCondition                    => TRUE),
                1 =>   (InputChangeTime   => WRA'LAST_EVENT,
                   PathDelay              => tEN_DIS, PathCondition                    => TRUE),
                2 =>   (InputChangeTime   => MBA'LAST_EVENT,
                   PathDelay              => VitalExtendToFillDelay(tMDV), PathCondition=> TRUE),
                3 =>   (InputChangeTime   => CLKB'LAST_EVENT,
```

```
                PathDelay             => VitalExtendToFillDelay(tPMR),
                PathCondition         => CSANeg = '0' AND WRA = '0' AND MBA = '1'),
             4 =>   (InputChangeTime  => CLKA'LAST_EVENT,
                PathDelay             => VitalExtendToFillDelay(tA),
                PathCondition         => (CSANeg = '0' AND WRA = '0' AND MBA = '0'))));
  END PROCESS;
END GENERATE;
```

There are 5 sources of changing signal on port A: changing CSANeg, WRA, MBA, CLKB, CLKA. PathCondition parameters present path enabling condition. Parameter tEN_DIS has VitalPathDelay01Z type and keeps data about transitions 0->1, 1->0, Z->0, Z->1, 0->Z, 1->Z. Changing MBA, CLKB, CLKA can cause transitions 0->1, 1->0. VitalExtendToFillDelay procedure is used to adjust VitalPathDelay01 and VitalPathDelay01Z data.

Conclusions

The hell and haven of standardization model development process is well known. VITAL standard is on the right way of precision timing description of complex devices but it should be improved.

We widely applied VITAL package procedures and extended VITAL level 1 model structure which helped us create VITAL level 0 models with complex functionality.

On the other hand we had some difficulties in describing internal delays in VITAL models and further assembling SDF files. It is necessary to be careful while describing TSKEW timing because in some cases VITAL and SDF semantics of this parameter does not correspond to datasheet semantics.

Additional effort has to be made in model style standardization. Our experience shows that some constraints should be corrected for functionally complex devices. For example, the FMF Guide Lines require to use only scalar ports. We saw that it does not simplify model description: we had to use block port map to transfer scalar ports to vector and then we used VHDL facilities as generate statements instead of copying the same constructions.

Another FMF requirement to use capital letters for keywords does not make the code more readable. The experience of programming and HDL language application proves it.

The result of our work is that more than a half of VITAL model code consists of VITAL procedure calls. For IDT FIFO memory model with 16 scalar inputs, 6 outputs, two 36-bit bidirectional buses with 2600 lines of its VHDL code only 1000 lines represent directly functional description, 1100 lines represent VITAL procedure calls. It would be great to have options in CAD tools which could allow to generate VITAL description.

We used Cadence Leapfrog, MODELTECH, Synopsys VHDL simulators and came across some problems working with VITAL and SDF. For example, Leapfrog does not generate error message checking VITAL-0 entity which contains constant declarations; MODELTECH V-system and Synopsys VSS did not completely check correspondence between SDF parameters and VHDL generics.

Another problem is selection of timing parameter types. VITAL description contains information about 3 values of timing parameters: minimum/typical/maximum. Timing parameters are loaded from SDF at the elaboration phase and it is possible to use either minimum or typical or maximum values simultaneously for all timings. Using various combinations of values is a special issue of dynamic timing verification.

And the last but not the least. We would like to thank Alex Pejenkov, former employee of SEVA Technologies, Inc. for his ideas in development of VITAL models of IDT FIFO memory chips.

* * *

Arkadi Poliakovis is Senior CAE Consulting Engineer at SEVA Technologies, INC., Fremont, CA. He previously worked in Russia and has more than 25 years experience in development of digital simulation systems and teaching at

a Russian University.

Anatoli Sokhatski is Senior CAE Consulting Engineer at SEVA Technologies, INC., Fremont, CA. He has more than 15 years experience in development of CAD tools and teaching in Russia.