BI-DIRECTIONAL MIXED SIGNAL CONNECTION MODULES FOR AUTOMATIC INSERTION

Olaf Zinke

Cadence Design Systems
San Diego, CA, USA
ozinke@cadence.com

Abstract

Efficient simulation of mixed signal designs requires the ability to quickly exchange analog and digital representations of a cell during the design and verification process. For the interconnection between analog nets and digital signals, connect modules are used. Mostly these connections elements are limited to being uni-directional when inserted automatically. The paper describes how true bi-directional connect modules can be realized in VERILOG-AMS by using special language features.

Introduction

It is common to simulate analog and digital represented parts of a SOC simultaneously using Mixed Signal Simulators like the Cadence AMS Simulator. Mostly mixed signal standard languages (VERILOG-AMS, VHDL-AMS) are used throughout such designs. They describe the structure as well as the behavior of the design or IC.

The simulation tool decides based on the kind of description inside of the model or of a part of the model, which structure or behavior will be evaluated in the digital domain or in the analog domain during the mixed signal simulation.

What about when analog and digital interconnect? - At these connection points discrete digital signals have to be transformed into continuous analog information and vice versa. Both VERILOG-AMS and VHDL-AMS are capable describing of such interaction behavior. Both are true mixed signal languages with model-internal cross-domain read access of values.

VHDL-AMS requires manual or netlister based insertion of these connection elements, whereat types and natures must match or be compatible (1). VERILOG-AMS with its additional automatic insertion of connect elements during the design elaboration phase gives the user more flexibility in terms of using the language as a design language. The user is able to exchange the implementation of a model very quickly without the necessity of an afterwards manual type-matching or re-netlisting of major parts of the design. The user just interconnects digital and analog wires in the schematic or in the text based design and specifies the rules defining what connect modules should be used for the possible cases of inter-connection. During design elaboration the disciplines get resolved and based on that the connect modules get inserted following the above mentioned connect rules (3).
This paper talks about connect element modeling in VERILOG-AMS.

Uni-Directional and Bi-Directional Connect Elements

In a simple mixed signal case analog nets and digital signals are connected and only one of them is driving the other one. The connection is uni-directional. In the A-to-D case the connection element detects the analog voltage level and applies the appropriate logic state to the digital receivers. In the D-to-A case the connection element reads the state of the digital drivers at the input of the connection element and applies the voltage to the analog output net accordingly.
The connection module features built into the VERILOG-AMS language enhance the D-to-A case by adding the capability of applying a different state than what the output of the digital driver delivers to the digital receivers being on the same wire like the digital drivers and the logic input of the connection element. This is useful for example, applying delay time caused by a certain analog load on a digital net. Because of the definition of the VERILOG-AMS language this is possible with connection modules having one single input pin.
Especially in large SOC designs a third case is very common. There are blocks, represented as analog behavioral or as schematic, connected to bi-directional communication or data busses. That is, the analog block receives information and is also able to send information over the same connection. On the digital side there could be several blocks with INOUT connections at this bus. Each bus wire needs an INOUT connection module. The difficulty is that the connection module for automatic insertion during the elaboration phase can only have one pin on the logic side. That is, the connect module needs to be able to read and write at the same time via one single port to ensure true bi-directional behavior.

Uni-Directional Example

In the uni-directional analog-to-digital (d2a) case every change of the digital input state is detected. The input state information is used to control voltage and impedance of the analog output. In the simplest case the output structure consists of a controlled ideal voltage source and a controlled serial output resistor.
The discrete real variables *r2set* and *v2set* (see EXAMPLE I) are set to the output impedance and output voltage levels that represent the analog output behavior for the given digital input state. Both trigger transition functions. A change of *v2set* lets the ideal voltage source V(x) change to the appropriate voltage level with the specified rise/fall time. At the same time a change of *r2set* triggers the transition of the output resistance

R = V(x, aVal) / I(x, aVal).

```
                    EXAMPLE I
        SIMPLE DIGITAL-TO-ANALOG CONNECT MODULE

`include "disciplines.vams"
`timescale 1ns / 10ps

connectmodule d2a(dVal, aVal);
input  dVal;
output aVal;
logic dVal;
electrical aVal;

// parameter declaration and initialization not shown

always @dVal begin
 case(dVal)
  1'b0: begin v2set=vldrive;r2set=rldrive; end
  1'b1: begin v2set=vhdrive;r2set=rhdrive; end
  1'bx: begin v2set=vxdrive;r2set=rxdrive; end
  1'bz: begin v2set=vzdrive;r2set=rzdrive; end
 endcase
end

analog begin
 V(x) <+ transition(v2set,0,vrise,vfall);
  I(x,aVal) <+ V(x,aVal) /
                transition(r2set,0,rrise,rfall);
end

endmodule
```

Creating a Bi-Directional Connect Module

In the above example there was no feedback from the analog side back to the digital side. The connection was uni-directional. In cases like shown in Fig. 1 the connect module has to work in both directions. How can we feed back information equivalent to the analog solution to the digital side? The idea is to use the driver/receiver segregation feature of the VERILOG-AMS language. Digital drivers and receivers are separated at digital wires connected to connect modules. The connect module detects what state the digital drivers connected to the digital wire apply. Using this information and other conditions the connect module can decide what logic state the digital receivers connected to the same wire will actually see.

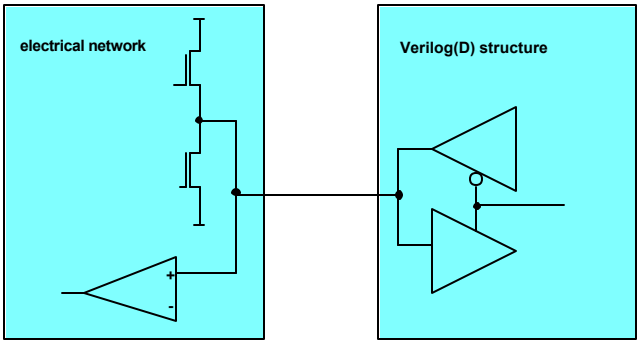

Fig. 1 Bi-Directional Connection

A. From Digital to Analog

Assuming for the moment there is only one logic block connected to the wire, the connection module inserted for this wire reads what the digital block drives on the net. The state could be an active 0 or 1, a Z or it could be undefined X. This information needs to be translated to analog and applied with the right driver strength at the analog output node. The 0 and 1 states should apply an appropriate voltage *Vout* via a lower resistance *Rout* to the analog output node, Z should apply a voltage in a high resistive way, which could vary dependent on the process and is parameterized (Fig. 2). The undefined X state is handled in a special way. If driven to the logic input of the connection module, an X is directly applied to the digital receivers. In this case the voltage source and resistance applied to the analog output node will be set by user-defined parameters. Because there is no undefined state in the continuous analog domain, it is the choice of the simulation user to decide about the voltage he wants having generated in this case. He may want to apply this voltage in a very low resistive way to force the analog voltage settling at the desired voltage level relatively independent of the behavior of the connected analog circuitry.

B. From Analog to Digital

The analog solver of the mixed signal simulator resolves the analog output node of the connection element under the full recognition of the internal supply network and the analog structure of the connected analog cells. The resulting voltage *V(aVal)* from *aVal* to *ground* and current flow through *Rout* and through the *Vout* source represent the logic resolution on the analog side of the wire. This logic resolution available as analog voltage/current information is detected by the connection module (*dVal_out*) and fed back to the receivers in the logic block connected to this INOUT wire.

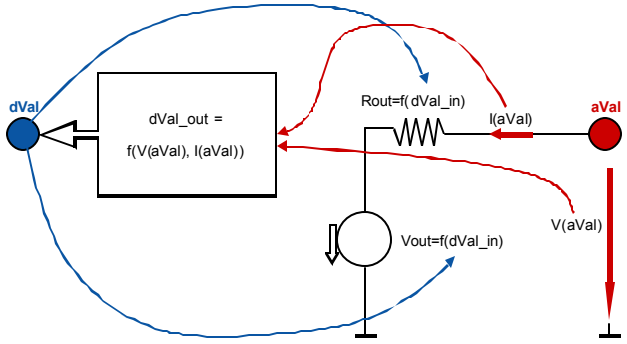

Fig. 2 Bi-Directional Connect Module

If the voltage at the analog connection is below the 0-threshold or above the 1-threshold, the 0 and 1 states are assigned to the digital INOUT port accordingly. If *Vout* leaves the 0 or 1 range i.e. after a rising edge of the digital input signal was detected, *dVal_out* stays unchanged for the first moment. If the voltage remains in the X-range for longer than a preset time limit *xdelay* the output *dVal_out* will change to X and stay X until the voltage moves back into the 0 or 1 range. This way too slow rise or fall transitions get detected.

This mechanism is able to detect the Z-condition at the analog connection module node as well. Therefore the current information is used. Detection is possible only if the designated Z-voltage recognition level is within the X-range. If this is the case when a Z is detected at the digital input and the absolute current into the analog connect module port is less than a certain limit, then *dVal_out* will be set to Z.

C. Implementation

The statement "A*ssign dVal=dValout"* applies the digital output state to the receiver connected to the logic INOUT port of the connect module (Example II). Cross statements are used for a precise detection of the voltage and the current threshold crossings related to the analog connect module port.

The signals *vstate* and *istate* are used to monitor the analog output voltage and current.

EXAMPLE II
BASIC BI-DIRECTIONAL CONNECT MODULE

```
`include "disciplines.vams"
`timescale 1ns / 10ps

connectmodule bidir(aVal, dVal);
inout aVal, dVal;
electrical aVal;
logic dVal;
parameter real maxHiZcurrent = 0.1u;
parameter real xdelay = 1;
parameter real zdelay = 0.5 * xdelay;

// some parameter declarations not shown

electrical x;
reg dValout, istate, inXrange, outOfZcurrent;
logic dValout;
real v2set, r2set;
integer vstate;

assign dVal = dValout;

// initialization not shown

always @dVal begin
 case(dVal)
 1'b0: begin v2set=vldrive;r2set=rldrive; end
 1'b1: begin v2set=vhdrive;r2set=rhdrive; end
 1'bx: begin v2set=vxdrive;r2set=rxdrive;
             dValout=1'bx;
       end
 1'bz: begin v2set=vzdrive;r2set=rzdrive; end
 endcase
end

always @(cross(V(aVal)-thresholdHi,1))
     begin vstate=1; inXrange=0; end
always @(cross(V(aVal)-thresholdHi,-1)) inXrange=1;
always @(cross(V(aVal)-thresholdLo,1))  inXrange=1;
always @(cross(V(aVal)-thresholdLo,-1))
     begin vstate=3; inXrange=0; end

always @(posedge inXrange) begin:XRangeDelay
    #xdelay
    vstate=2;
    inXrange=0;
end

always @(negedge inXrange) disable XRangeDelay;

always @(posedge outOfZcurrent)
begin:outOfZCurrentDelay
    #zdelay
    istate=1;
    outOfZcurrent=0;
end

always @(negedge outOfZcurrent)
       disable outOfZCurrentDelay;

endmodule
```

EXAMPLE II
CONTINUED

```
always @(cross(abs(I(x,aVal))-maxHiZcurrent,1))
  outOfZcurrent=1;

always @(cross(abs(I(x,aVal))-maxHiZcurrent,-1))
  begin istate=0; outOfZcurrent=0;end

always @(vstate or istate) begin
 case(vstate)
 1: dValout= (dVal===1'bx) ? 1'bx : 1'b1;
 2: dValout= ((istate===1'b0)&(dVal===1'bz))?1'bz:1'bx;
 3: dValout= (dVal===1'bx) ? 1'bx : 1'b0;
 endcase
end

analog begin
 V(x) <+ transition(v2set,0,vrise,vfall);
 I(x,aVal) <+ V(x,aVal) /
                transition(r2set,0,rrise,rfall);
end

endmodule
```

The signal *inXrange* is used to show the analog voltage being in the X-range but for not longer than the time limit parameter *xdelay*. When the connect module read an undefined X on the logic side it also applies this X state back to the receivers. Table I summarizes the functionality around the *istate* and *vstate* signals.

D. Example

Fig. 3 shows the simulated waveform of the bi-directional case where the two digital drivers (signals *digdrv1* and *digdrv2*) are connected to the digital port of the connect module. At point *M1* both drivers apply 0. The signal *dVal* is the logic

TABLE I
FUNTION TABLE

dVal_in	VSTATE	ISTATE	dVal_out
x	d	d	x
z	1	d	1
z	2	0	z
z	2	1	x
z	3	d	0
1	1	d	1
1	2	d	x
1	3	d	x
0	1	d	x
0	2	d	x
0	3	d	0

VSTATE
1 V(aVal) > thresholdHi
2 thresholdHi >= V(aVal) >= thresholdLo
3 V(aVal) < thresholdLo

ISTATE
0 abs(I(aVal)) <= maxHiZcurrent
1 abs(I(aVal)) > maxHiZcurrent

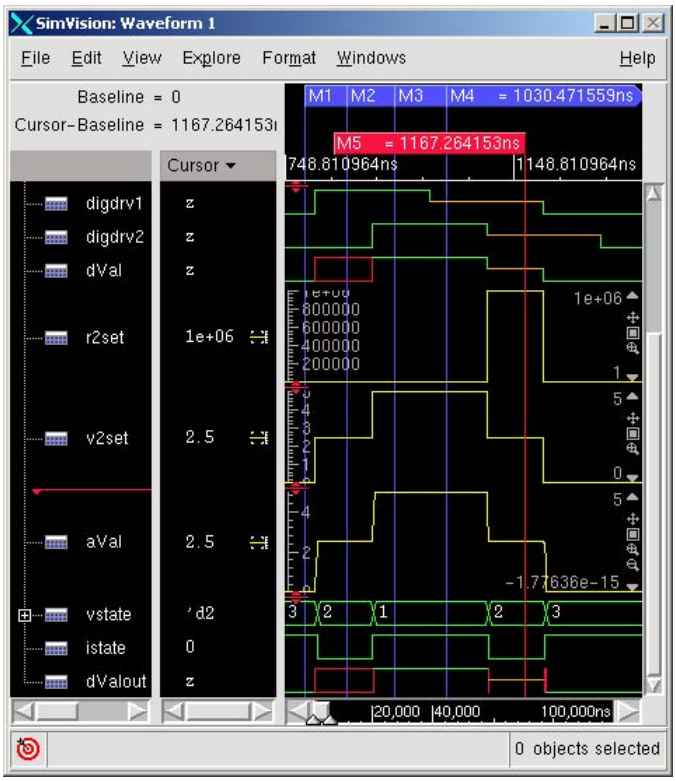

Fig. 3 Example Waveform A

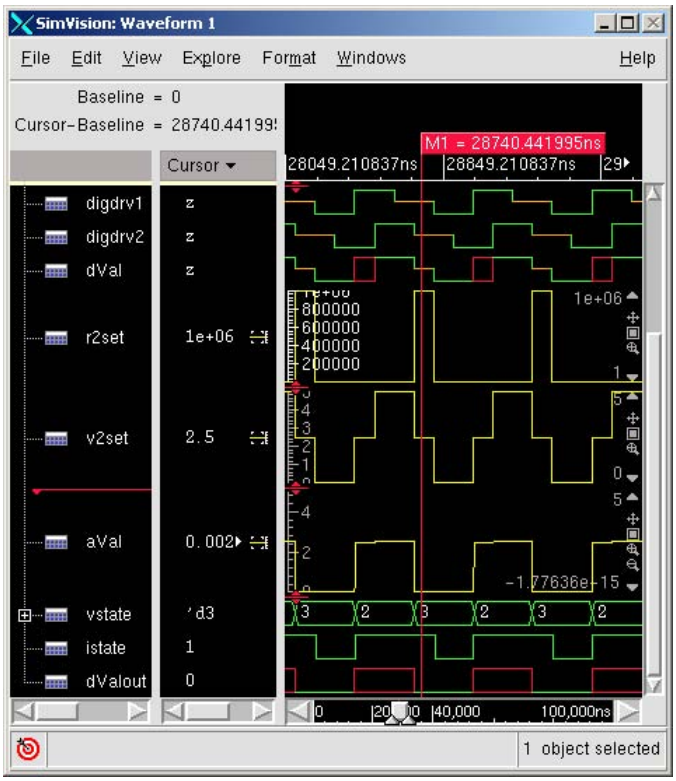

Fig. 4 Example Waveform B

resolution of both driver signals. This is the logic input of the connect module. Accordingly *v2set* is set to the low voltage level. The voltage *aVal* at the analog port of the connect module is almost at the same low voltage level. Note that this voltage level is determined by the analog circuitry connected to the analog port of the connect module as well. This results in *vstate* being at 3 and *dValout* (the logic state the connect module is applying to the digital receivers connected to the digital port) being at 0. In this case *dVal* and *dValout* are equivalent. At point *M2* the digital drivers force opposite states. This results in a resolution to X on *dVal*. Therefore also *dValout* turns X immediately. Obviously the analog circuitry at the analog port is very high resistive, because only a low current is necessary to force 2.5 Volts to this node. That is why *istate* is 0. But *istate* does not influence *dValout*, because *dVal* is at X.

At point *M5* both drivers are HiZ. Also *dValout* shows Z after the current through the analog port has settled and *istate* confirms that the analog circuitry at the analog port is in the Z-state also.

In Fig. 4 the external analog circuitry at the analog port of the connect module drives a voltage close to 0 Volts. This results in a 0 logic state at *dValout* when the digital drivers are at Z (see point M1) or at 0 also. In all other cases *dValout* delivers an X to the digital receivers

Enhancements

The above example contains a minimum set of connect module features. Enhancing this model means adding features bringing the model closer to the reality. The user should be aware that adding features means also lowering the simulation speed. There could be very many connect modules inserted in the design, so adding additional features could slow down the simulation remarkable. It would be necessary to supply the user with several connect modules with different detail level. That way the user could choose a model with respect to the desired detail-to-speed ratio.

A. Power Supply Dependency

One of the first enhancements the user would think of is power supply dependency of the build-in thresholds and the output voltage levels. This could be done easily by binding the connect module parameters to global power supply parameter. This way a static power supply dependency is realized. If the simulation of the design requires dynamic power supply dependencies of the connect module, this can be implemented in VERILOG-AMS by referencing or connecting to global power supply nodes from inside of the connect module. Effects like power on/off behavior or transient power supply noise become visible. For this purpose the Cadence AMS Simulator allows the use of inherited connections (via VERILOG-AMS attributes) inside of connect modules as well.

B. Dependency on The Number of Digital Drivers

The connect module shown in EXAMPLE II uses the resolved logic state of the drivers on the digital pin as the logic input signal. The analog network inside of the connect module is built independently of the number of drivers on the digital net. EXAMPLE III shows a bi-directional connect module containing this dependency.

From inside of a connect module information about the digital drivers on a digital net can be accessed. The access functions are defined in the VERILOG-AMS Language Reference Manual (3). The function *$driver_count(<wire_name>)* returns the number of drivers connected. The favorable way to ask for the number of drivers is the digital initial block, because the number of drivers will not change during the simulation.

EXAMPLE III

BI-DIRECTIONAL CONNECT MODULE WITH DEPENDENCY ON NUMBER OF DIGITAL DRIVERS

```
`include "disciplines.vams"
`timescale 1ns / 10ps

connectmodule bidir(aVal, dVal);
inout aVal, dVal;
electrical aVal;
logic dVal;

electrical x, y;
reg dValout;

// some declarations not shown

parameter real ghondrive=1.0/rhondrive;
parameter real ghoffdrive=1.0/rhoffdrive;
parameter real glondrive=1.0/rlondrive;
parameter real gloffdrive=1.0/rloffdrive;
real rh2set, rl2set;
integer DrCount, i;
integer XCount, ZCount, LCount, HCount;

assign dVal = dValout;

initial DrCount=$driver_count(dVal);

// initialization not shown

 always @(driver_update(dVal)) begin
    XCount=0; ZCount=0; LCount=0; HCount=0;
    for (i=0; i<DrCount; i=i+1)
     case($driver_state(dVal,i))
     1'b0: LCount=LCount+1;
     1'b1: HCount=HCount+1;
     1'bx: begin XCount=XCount+1; dValout=1'bx; end
     1'bz: ZCount=ZCount+1;
  endcase
  rh2set=1.0/((XCount+HCount)*ghondrive+
              (ZCount+LCount)*ghoffdrive);
  rl2set=1.0/((XCount+LCount)*glondrive +
              (ZCount+HCount)*gloffdrive);
 end

always @(cross(V(aVal)-thresholdHi,1))
         begin vstate=1; inXrange=0; end
always @(cross(V(aVal)-thresholdHi,-1)) inXrange=1;
always @(cross(V(aVal)-thresholdLo,1))  inXrange=1;
always @(cross(V(aVal)-thresholdLo,-1))
         begin vstate=3; inXrange=0; end

 always @(posedge inXrange)
   begin : XRangeDelay
   #xdelay
   vstate=2;
   inXrange=0;
 end

 always @(negedge inXrange) disable XRangeDelay;

 always @(posedge outOfZcurrent)
   begin : outOfZCurrentDelay
   #zdelay
   istate=1;
   outOfZcurrent=0;
 end

always @(negedge outOfZcurrent) disable
outOfZCurrentDelay;

always @(cross(abs(I(x,aVal)-I(aVal,y))-
           maxHiZcurrent,1)) outOfZcurrent=1;
always @(cross(abs(I(x,aVal)-I(aVal,y))-
          maxHiZcurrent,-1))
          begin istate=0;  outOfZcurrent=0; end
```

EXAMPLE III

CONTINUED

```
 always @(vstate or istate) begin
     case(vstate)
     1: dValout= (dVal===1'bx) ? 1'bx : 1'b1;
     2: dValout= ((istate===1'b0) & (dVal===1'bz)) ?
1'bz : 1'bx;
     3: dValout= (dVal===1'bx) ? 1'bx : 1'b0;
     endcase
 end

analog begin
 V(x) <+ vhdrive;
 V(y) <+ vldrive;
 I(x,aVal) <+ V(x,aVal) /
transition(rh2set,0,rrise,rfall);
 I(aVal,y) <+ V(aVal,y) /
transition(rl2set,0,rrise,rfall);
end

endmodule
```

The function *$driver_state(<wire_name>,<driver_counter>)* returns the status of the specified driver and the function *driver_update(<wire_name>)* returns *true* if at least one of the connected drivers on this net changes its state.

If we would stay with our current analog network implementation (serial resistance and voltage source to ground) it would get difficult to add dependency on the number of digital drivers and their state. For every digital driver we would need to switch such kind of resistance/voltage source branch in parallel. This leads to a topology change dependent on the number of digital drivers. It is better if we use in this case a different analog output structure (Fig. 5) that can be made easily dependent on the number of drivers without changing the topology.

Vhdrive and *Vldrive* are constant voltage sources. The *Rh* and *Rl* resistors are dependent on the number of digital drivers and on their current states, i.e. *Rh* is calculated based on the number of digital drivers that are X or 1 multiplied with the switched-on conductance and added to the switched-off conductance multiplied with the number of drivers being Z or L.

The voltage on the analog port and the absolute current difference of the current through both of the resistances are used to determine the logical state applied to the digital receivers.

After characterizing according to the technology used, this connect module models the actual circuit behavior already very realistic. An additional enhancement could be to take the

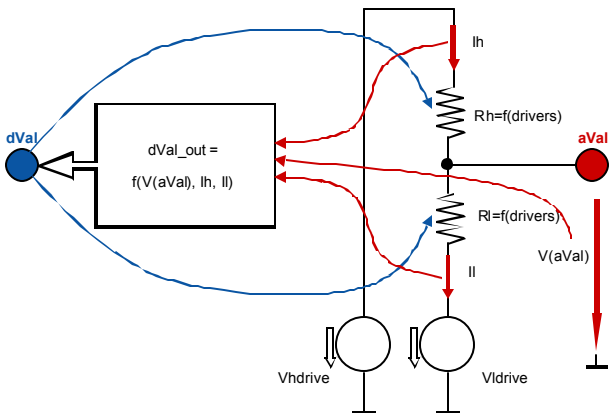

Fig. 5 Connect Module Structure

strength of each digital driver into account and varying both resistances accordingly.

Summary

Bi-directional connect modules have been created using VERILOG-AMS language features. Connect module examples with different level of modeled detail were shown. These modules can be used for rule based automatic insertion according to the VERILOG-AMS language reference manual. Automatic Insertion of connect modules during the design elaboration phase enables an efficient mixed-signal simulation of schematic as well as text based designs.

References

(1) "IEEE Standard VHDL Analog and Mixed-Signal Extensions," Institute of Electrical and Electronics Engineers, New York, December 1999.

(2) " IEEE Standard Verilog Hardware Description Language," Institute of Electrical and Electronics Engineers, IEEE 1364-2001, New York, September 2001.

(3) "Verilog-AMS Language Reference Manual," Open Verilog International, Los Gatos, Version 2.0.