How to use the new System Verilog Nettypes to address the Analog SoC Integration Verification

Joachim Geishauser

Thomas Theisen

Freescale Halbleiter GmbH

June 25, 2015

Munich

Agenda

Verification Use Cases

LRM Nettype Introduction

User-defined Nettype: Realnet

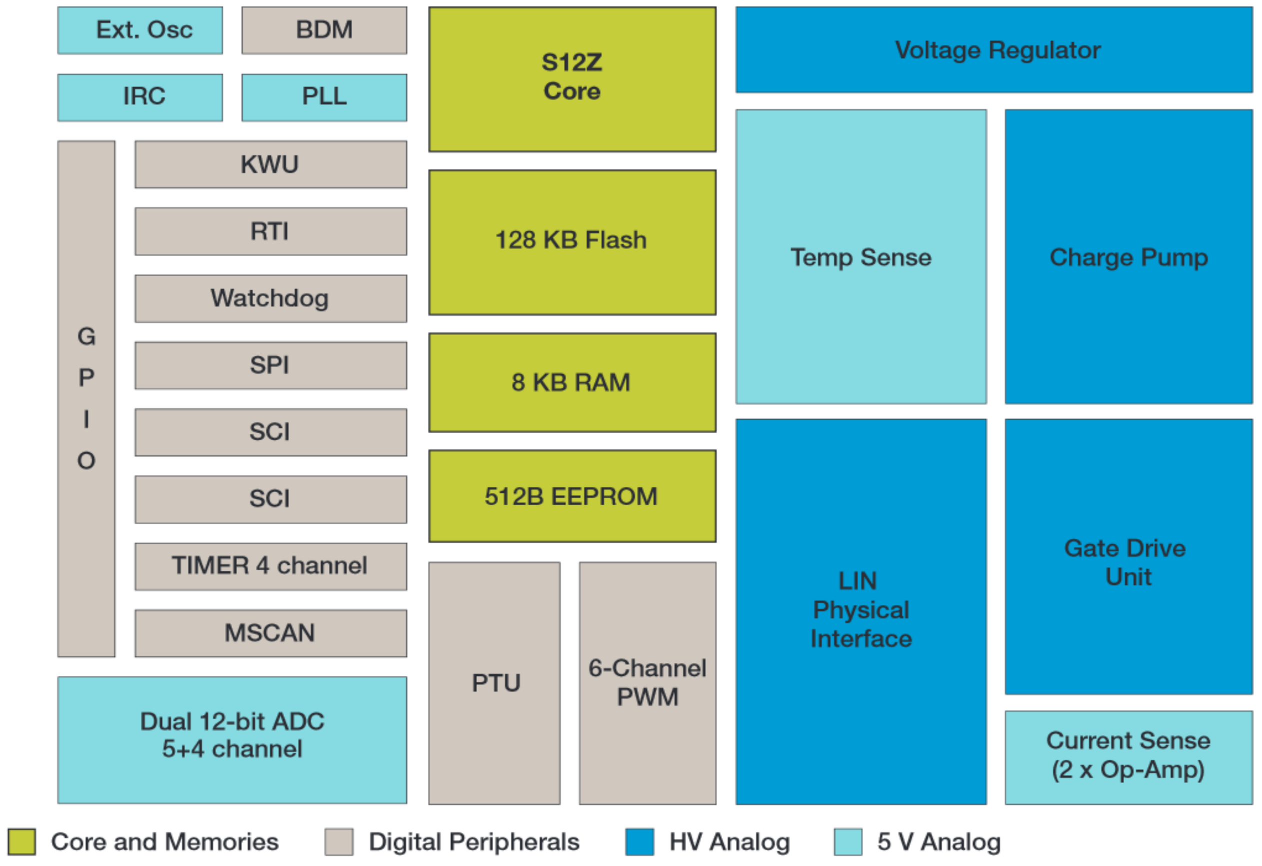
Using the Nettype

Conclusion

Verification Use Cases

The Analog Integration Verification Problem

The Use Case
Setup

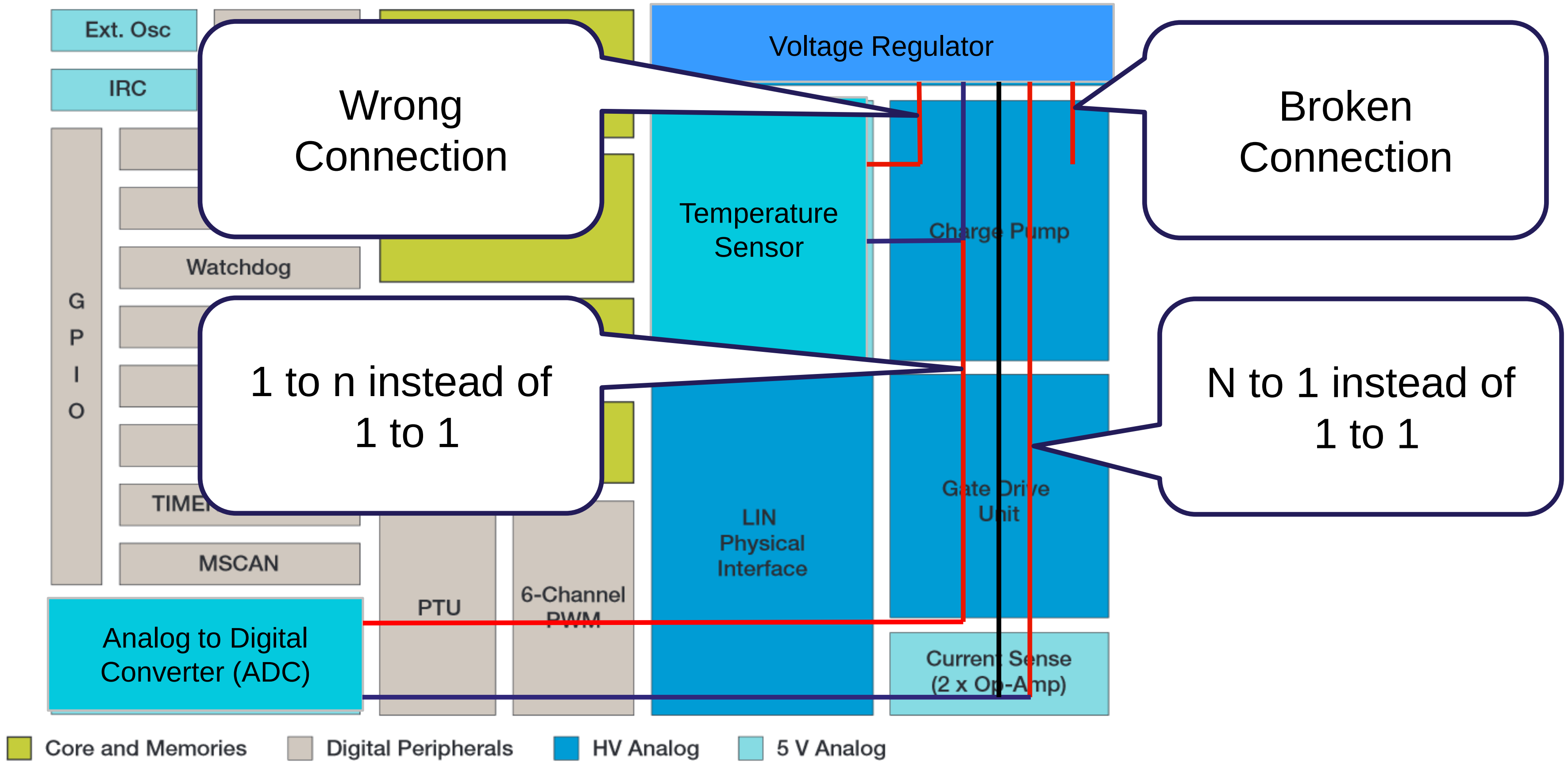


Freescale MagniV S12ZVML128 SoC

- Many analog blocks
- Existing Verilog Behavioral models
- Existing testbench
- Concurrent support of SV Nettype models and Verilog behavioral models
- Analog design team co-located with digital design and verification team
- No change to RTL code

The Use Case

Verification View



LRM Nettype Introduction

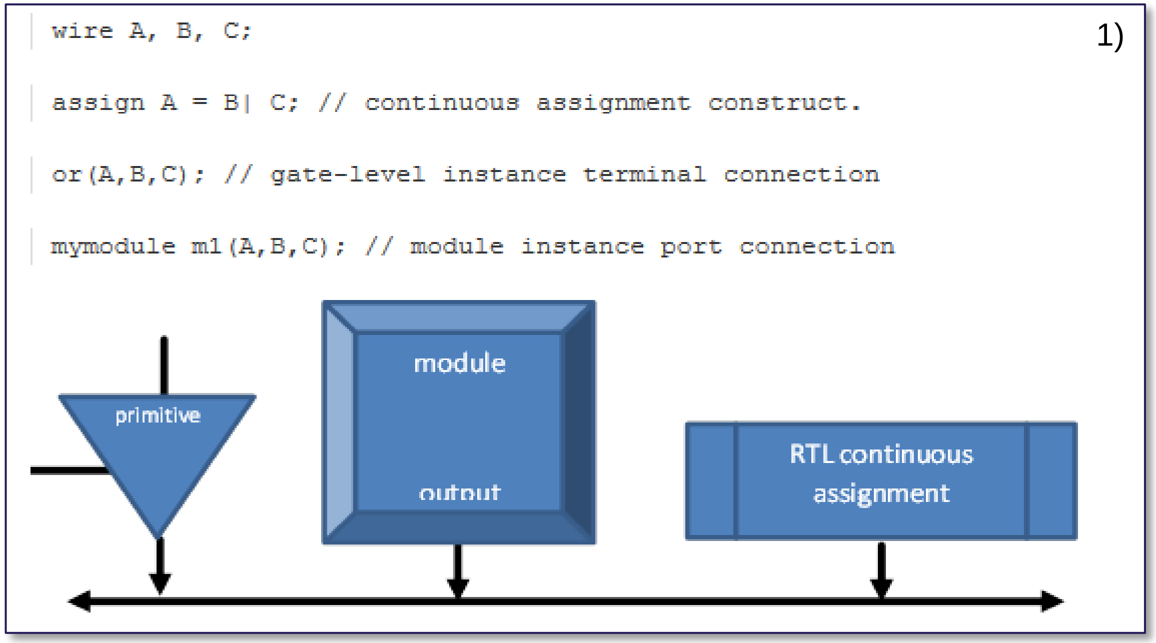
LRM Nettype Introduction

- The Nettype construct was introduced with the SystemVerilog LRM release 2012.
- Allows to define user-defined data types on top of built-in Nettypes e.g. like
  - wire
  - supply0
- The Nettype is constructed out of 3 parts
  - Data Structure
  - Resolution function
  - Declaration

LRM Nettype Introduction

Data Structure

```
typedef struct {
      real data;
} data_type_t;
```



1)

```
wire A, B, C;

assign A = B| C; // continuous assignment construct.

or(A,B,C); // gate-level instance terminal connection

mymodule m1(A,B,C); // module instance port connection
```

Resolution Function

```
function automatic data_type_t my_res_function (
                       input data_type_t driver[] );
```

Nettype declaration

```
nettype data_type_t data_type with my_res_function;
```

1) http://blogs.mentor.com/verificationhorizons/blog/2013/05/03/wire-vs-reg

LRM Nettype Introduction

New LRM Interconnect Keyword

```
interconnect out_tmp;
logic_driver my_logic_driver(
    .in (hi_low),
    .out (out_tmp)
  );
transmitter my_transmitter(
    .in (out_tmp),
    .out (result)
  );


module logic_diver (in, out);
output interconnect out;
driver my_driver(
    .in (hi_low),
    .out (out)
  );
```

Works

```
module diver (in, out);
output reg out;

module transmitter (in, out);
input reg in;
```

Fails Compilation

```
module driver (in, out);
output data_type out;

module transmitter (in, out);
input data_type in;
```

Usage of the VCS switch solves the problem

`-xlrm coerce_nettype`

VCS replaces the connections with interconnect

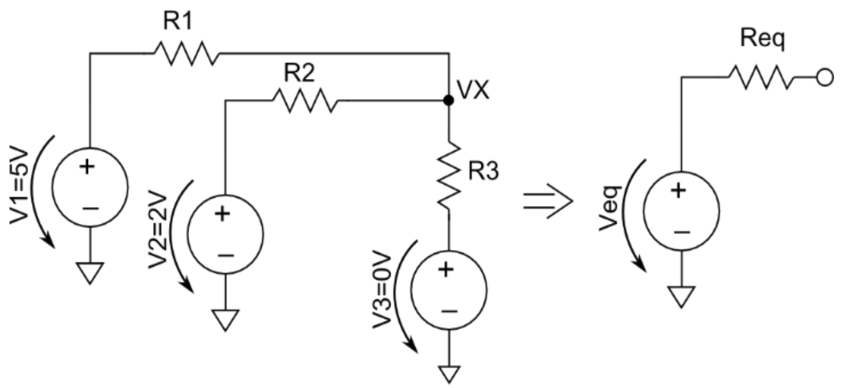
User-defined Nettype: Realnet

User-defined Nettype Realnet

Definition Background (Voltage Example)

- How to address a realistic analog value propagation

```
wire VX;
module M1  (VX);
module M2  (VX);
module M3  (VX);
```



- Thevenins theorem equivalent resistor

$$R_{eq_{new}} = \frac{R_{eq}R_3}{R_{eq} + R_3}$$

- Thevenins theorem equivalent voltage

$$V_{eq_{new}} = \frac{V_{eq}R_3 + V_3R_{eq}}{R_{eq} + R_3}$$

User-defined Nettype Realnet

SystemVerilog Implementation

Data Structure

```
typedef struct {
     real v_eq;          //Thevenins theorem equivalent voltage
     real r_eq;          //Thevenins theorem equivalent resistance
     real load;          //Number of drivers
                         } voltage_t;
```

Declaration

```
nettype voltage_t realnet with res_electrical;
```

4 state support

```
`define realnetZState 1e23
`define realnetXState 1e22
```

Defines can be used for built-in to user-defined nettype conversion (and vice versa)

User-defined Nettype Realnet

SystemVerilog Implementation

```
function automatic voltage_t res_electrical (input voltage_t driver[]);
    voltage_t aDrv, res0, res= '{0.0,`realnetZState, 0.0};
    foreach (driver[i]) begin : for_each_driver
       aDrv = driver[i];
       if((`ABS(aDrv.r_eq-`realnetZState)< 1.0) ||
         ((`ABS(aDrv.v_eq-`realnetZState))< 1.0)) begin
       end
       else if((`ABS(aDrv.r_eq-`realnetXState)< 1.0) ||
             (`ABS(aDrv.v_eq-`realnetXState)) < 1.0) begin
           res = '{0.0, `realnetXState,0.0};
           break;
       end
       else if((`ABS(res.r_eq-`realnetZState)< 1.0) ||
             (`ABS(res.v_eq-`realnetZState) < 1.0 ))
           res = aDrv;
       else begin
           res0    = res;
           res.v_eq = (aDrv.v_eq * res0.r_eq + res0.v_eq * aDrv.
                      (aDrv.r_eq + res0.r_eq);
           res.r_eq = res0.r_eq * aDrv.r_eq / (res0.r_eq+aDrv.r_
       end
    end : for_each_driver
    res.load          = driver.size;
    if (($realtime == 0) && (res.r_eq == 0)) res.r_eq = `realnetZState;
    res_electrical = res;
endfunction : res_electrical
```

High imp driver has no influence to node. If there is only one driver to the node, the result will be high imp since `res` is init with Z!

Result is unknown

Do not compare to 0

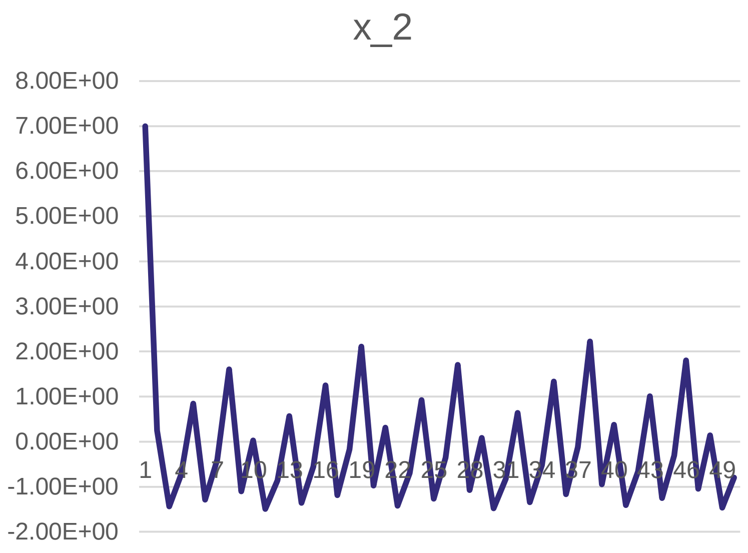
If `res` is high Imp. (=no infuence to node) the actual driver will drive the node !!! `res` is init as Z, so we go here when we enter the first time the loop!!!

Init R to high impedance

User-defined Nettype Realnet

Real Number Convergence Problem

```
real low  = 0;
real high = 3;
real x_1 , x_2 , delta;
for (int ii = 0; ii < 50; ii++) begin
    x_1 = foo (low);
    x_2 = foo (high);
    delta = (high - low)/2;
    if ((x_1<0)&&(x_2>0)) high = delta + low;
    if ((x_1>0)&&(x_2<0)) high = delta + low;
    if ((x_1>0)&&(x_2>0)) low  = low - delta;
    if ((x_1<0)&&(x_2<0)) high = delta + high;
   $display("%e,%e,%e,%e",x_1,x_2,low,high);
end

function real foo (real x);
      foo = x * x -2;
endfunction
```



```
res_electrical = result;
`ifdef TRUNC
     res_electrical.v_eq = $bitstoreal (
         ($realtobits (res_electrical.v_eq) & 64'hFFFFFFFFFFFFFF00));
`endif
```

User-defined Nettype Realnet

Technology Dependent Net Types

```
package nettype_l118_uhv_pkg;
   import nettype_pkg::*;
   function automatic voltage_t res_electrical_1v8(
                                 input voltage_t driver[]);
      res_electrical_1v8 = res_electrical (driver);
     endfunction : res_electrical_1v8

   // AMS: electrical_5v0
   function automatic voltage_t res_electrical_5v0(
                                 input voltage_t driver[]);
      res_electrical_5v0 = res_electrical (driver);
   endfunction : res_electrical_5v0
   // ---------------------------------------------------------------
   // Nettype defintions
   // AMS: electrical_1v8
   nettype voltage_t sv_electrical_1v8 with res_electrical_1v8;
   // AMS: electrical_5v0
   nettype voltage_t sv_electrical_5v0 with res_electrical_5v0;
```

- Nettype connection is strongly typed.

User-define Nettype Realnet

Technology Dependent Nettypes

```
module driver #(MAXLOAD=1)
       (in, out);
   import nettype_ll18_uhv_pkg::*;

   input   wire             in;
   output  sv_electrical_1v8 out;
```

```
module transmitter
       (in, out);
   import nettype_ll18_uhv_pkg::*;

   input  sv_electrical_5v0  in;
```

```
wire out_tmp;
driver my_driver(
    .in  (hi_low),
    .out  (out_tmp)
  );
transmitter my_transmitter(
    .in  (out_tmp),
    .out  (result)
  );
```

Wrong Connection

```
Error-[PCTM] Port connection type mismatch top.sv, 78
"driver my_driver ( .in (hi_low),  .out (out_tmp));"
The following expression is illegally connected to port "out" of module
"driver", instance “my_driver". The type of the port does not match that
of the port connect.
        Expression: sv_electrical_5v0 out_tmp;
        Port      : sv_electrical_1v8 out;
```

Using the Nettype

Using the Nettype

- Nettype usage will be shown on the example blocks
  - PAD
  - ADC
- Goal is to allow usage of the same code for 4 state simulation as well as for user-defined Nettype usage
- A common development branch for the development was defined – use_realnet
- A common define was agreed upon - USE_REALNET

Using the Nettype

ADC example - Ports

Analog to Digital Converter (ADC)

- The usage of conditional code supports the usage as digital and Nettype model
- DVE collapses not selected code, thus keeps the code small

```
module adc12b9c_5m1t (
…
  vrh0,vrh1, vrl0, vrl1,
…
);

`ifdef USE_REALNET
   inout  realnet vrh0;
   inout  realnet vrh1;
   inout  realnet vrl0;
   inout  realnet vrl1;
`else
   inout  vrh0;
   inout  vrh1;
   inout  vrl0;
   inout  vrl1;
`endif
```

Using the Nettype

ADC example – Ports Load Checking

Analog to Digital Converter (ADC)

```
`ifdef USE_REALNET
   inout  realnet vrh0;
`else
   inout  vrh0;
`endif

`ifdef USE_REALNET
initial
  begin : load_check
    #0;
    if (vrh0.load > 1)
      $display("ERROR, vrh0 load > 1");
  end : load_check
`endif
```

N to 1 instead of 1 to 1

- `load` variable allows to add a simple check to the model
- `load` value is provided inside the resolution function

Using the Nettype

ADC example – Internal Real Variables

Analog to Digital Converter (ADC)

```
`ifdef USE_REALNET
   `define vrefh_val  vrh.v_eq
`else
   real vrefh_val;
   `define vrefh_val  vrefh_val
`endif

…
always @(posedge sar_az1_scan)
  vin_analog_val= vin_analog_val + `vrefh_val/256.0;
```

- Usage of defines for variables allows to keep algorithmic code almost unchanged – except the "`"

Using the Nettype

ADC example – Verilog Primitives

Analog to Digital Converter (ADC)

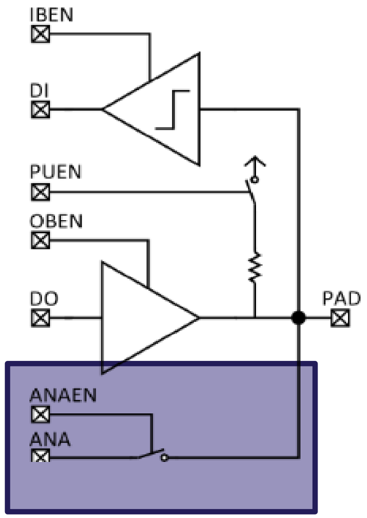
- Nettype primitive allows to keep code structure

```
module realnet_nmos (out, data, control);
   import nettype_pkg::*;

   input  wire    control;
   input  realnet data;
   output realnet out;
   assign out = control ?
        data : '{0,`realnetZState,1};
endmodule : realnet_nmos
```

```
inout adc_temp_sense_out;
real  adc_temp_sense_out_val;

`ifdef USE_REALNET
  realnet adc_temp_sense_out_r;
  assign  adc_temp_sense_out_r = '{adc_temp_sense_out_val,1000,1};
  realnet_nmos nmos_temp_sense_out (
       adc_temp_sense_out, adc_temp_sense_out_r, 1'b1);
`else
  nmos(adc_temp_sense_out, adc_temp_sense_out_i, 1'b1);
`endif
```

Model had integer and real values for temperature

Using the Nettype
PAD Example – Analog In

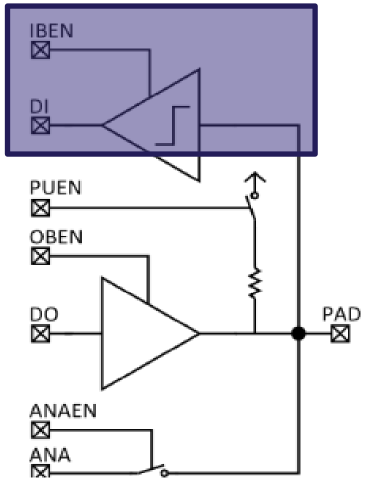


- Bidirectional assignment requires iterative solving
- Bidirectional signals create some simulation performance slow down
- Alternative implementations have functional drawbacks

```
assign pad =  anaen ? '{ana.v_eq,300.0,0.0} : '{gnd,`realnetZState,0.0};
assign ana =  anaen ? '{pad.v_eq,300.0,0.0} : '{gnd,`realnetZState,0.0};
```

Using the Nettype

PAD Example – Digital In

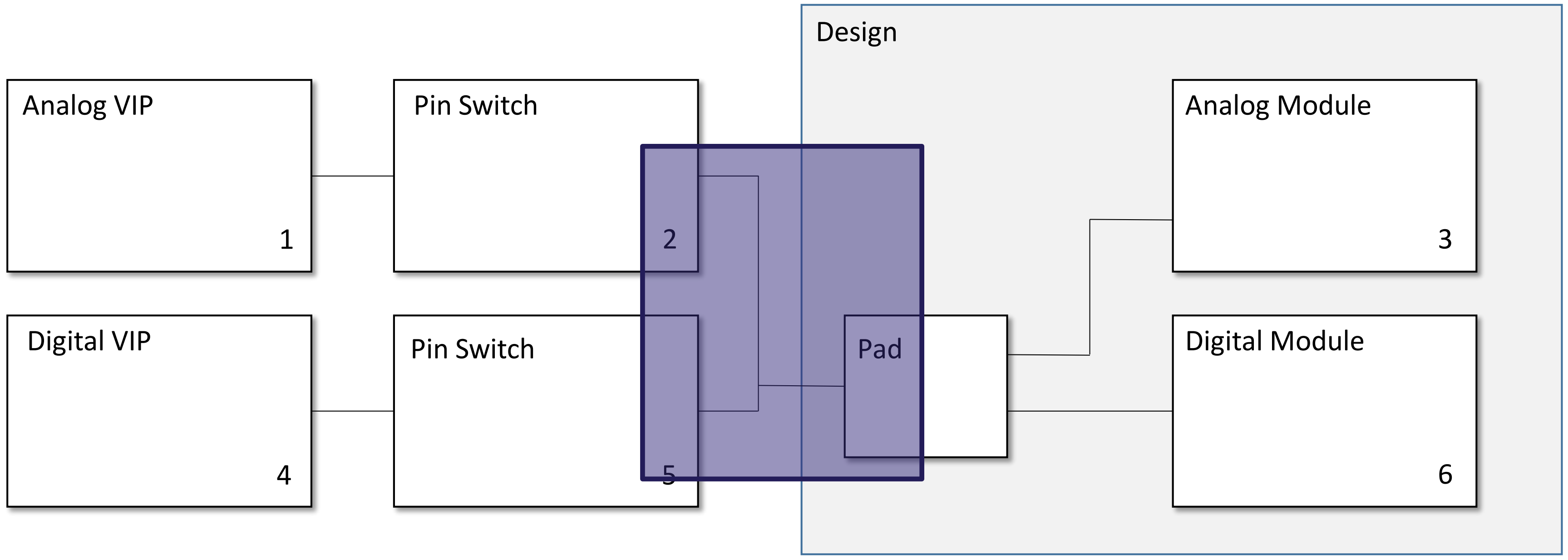


```
always @*  begin
   if (iben) begin
      if ((analogIn.r_eq - `realnetZState) < 1.0)
         di_reg = 1'bz;
      else if ((analogIn.r_eq - `realnetXState) < 1.0)
         di_reg = 1'bx;
      else if (analogIn.v_eq < Vth)
         di_reg = 1'b0;
      else
         di_reg = 1'b1;
   end
   else begin
      di_reg = 1'bz;
   end     //if
end //always
```

- PAD model does convert Realnet to 4 state value on digital input port

Using the Nettype

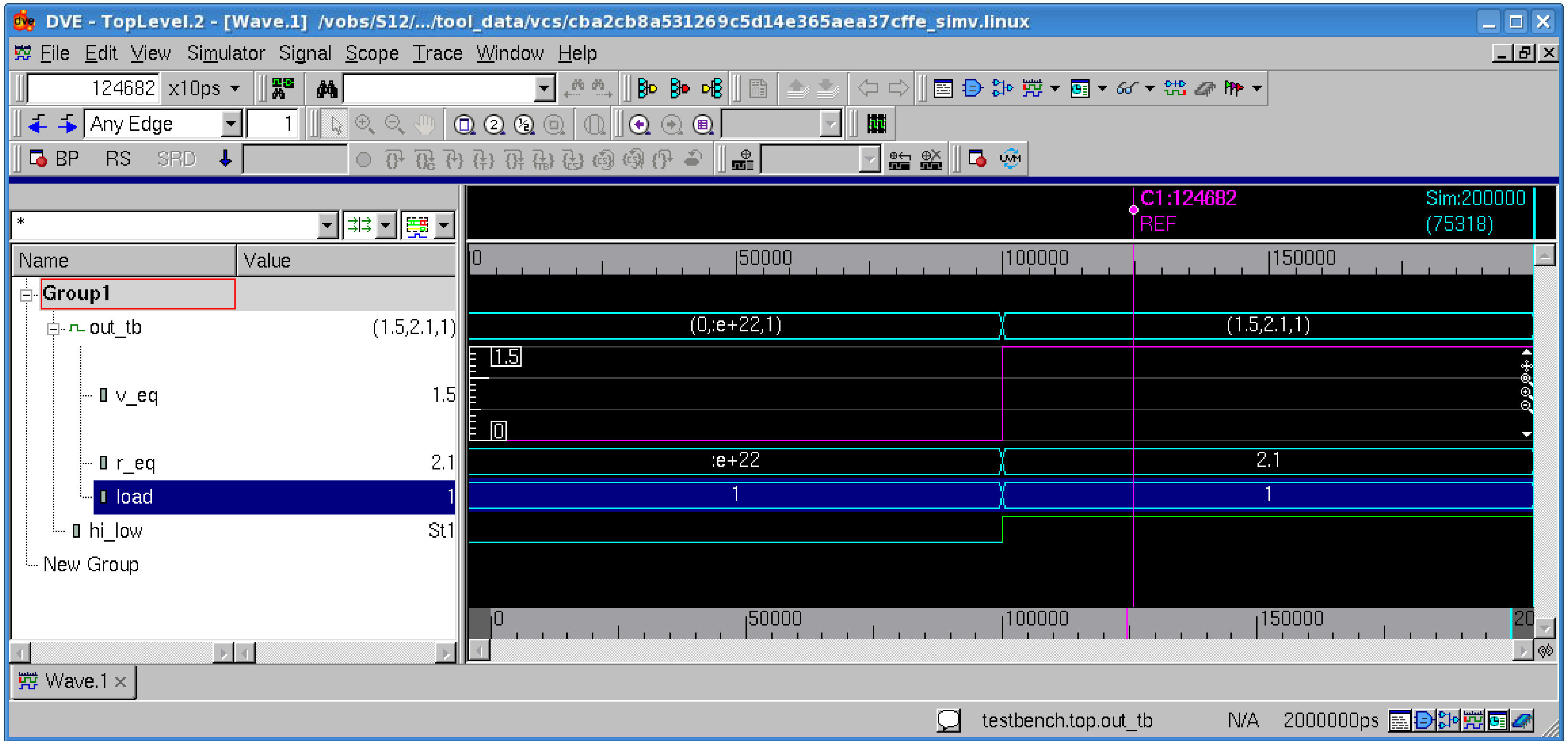
Testbench Considerations



- Since PAD model supports digital and analog function the testbench to design connection is done as a user-defined Nettype

Using the Nettype

Debug View


DVE - TopLevel.2 - [Wave.1] /vobs/S12/.../tool_data/vcs/cba2cb8a531269c5d14e365aea37cffe_simv.linux

Name	Value
Group1	
out_tb	(1.5,2.1,1)
v_eq	1.5
r_eq	2.1
load	1
hi_low	St1
New Group	

- Real values can be displayed as analog values

Conclusion

Conclusion
Verification Mapping

	Load Value	Specific Nettype	Real Value Modeling
Wrong connectivity		**X**	X
Broken connectivity	**X**		X
1 to n connection Instead of 1 to 1	X		**X**
N to 1 connection Instead of 1 to 1	X		**X**

- The specific Nettype creates a compile time error
- The Load Value method allows to add a check with little effort
- The Realnet modeling requires the most effort to implement

Conclusion

- The usage of the SV Nettype allows to verify analog integration behavior in a single digital simulator engine
- We did update the 24 behavioral models with manageable effort
- It is a good idea to start conversion from the bottom up – divide and conquer
- Runtime improvement was seen as expected, detailed data was not yet captured
- Missing LRM support required some workaround
- VCS Error messages need to be improved

Thank You