2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

MICROCHIP

It Should Just Work!
Tips and Tricks for Creating Flexible, Vendor Agnostic Analog Behavioral Models

Chuck McClish

Microchip Technology Inc.

accellera SYSTEMS INITIATIVE

Motivation for paper

- It should just work!... But it doesn't always
  - Lots of little gotchas to consider for true portability
  - Who supports what when?
- Different UDNs don't play well together
- Useful features that aren't yet LRM 'compliant'
  - But are supported by the tools

What are UDNs?

- User Defined Nettypes == UDNs, AKA the shiny new wheel!
  - Introduced SV 1800.2012 LRM
  - Abstract representation of a SV ‘wire’
  - Made of single or fixed structure of reals or 2 or 4 state integral types
  - User defined resolution function
- Replaces the non-LRM ‘wreal’

Simple UDN

- Single 4 state nettype?
  - Legal, but not supported by most!

```
nettype logic ana_net_t;
```

- Is SV ‘wire’ a nettype?
  - No!

```
nettype wire ana_net_t;
```

- What is portable?
  - Scalar or structures of reals

```
nettype real ana_net_t;

typedef struct {
  real        value;
  real        stren;
} ana_value_t;
nettype ana_value_t ana_net_t;
```

Guideline #1: Always use scaler or structures of reals for all nettype definitions

What about X and Z?

- Old style ‘wreal’ types had `wrealXState and `wrealZState
- UDNs are just a value set, there is no special X or Z value predefined
  - If you want it, you have to define it! Pick some obnoxious value.
  - Tools don’t understand X or Z, can make waveform plotting with these difficult
- Simple resolution function to handle X and Z:

```
`define ZSTATE      -100
`define XSTATE       100
`define ana_value_t real

function automatic `ana_value_t res (input `ana_value_t driver[]);
  res = `ZSTATE;
  for(int i = $low(driver); i < $size(driver); i++) begin: res_loop
    if (driver[i] > res) res = driver[i];
  end
endfunction

nettype `ana_value_t ana_net_t with res;
`define ana_net_t ana_net_t
```

Vendor Suggestion #1: Waveform viewer variable to define X and Z for pretty plotting

Type Matching

- All UDN types MUST match. This is simpler, but not always desirable:
  - Communication challenges, external IP
  - Different complexities at IP level vs SoC level
  - UDN license costs
- Flexible UDN examples:

```
// simple wire
`define ana_value_t logic
`define ZSTATE  1'bz
`define XSTATE  1'bx
`define ana_net_t wire
```

Guideline #2: Define and utilize X/Z values, UDN value set, and UDN as `defines

```
typedef struct {
  real        value;
  real        stren;
} ana_value_t;
`define ana_value_t ana_value_t
`define ZSTATE -100
`define XSTATE  100

function automatic `ana_value_t res (input `ana_value_t driver[]);
  res.value = `ZSTATE;
  res.stren = 0;
  for(int i = $low(driver); i < $size(driver); i++)
    if (driver[i].value != `ZSTATE)
      if (res.stren == driver[i].stren) begin
        res.value = `XSTATE;
        $warning("Contention on net. %m");
      end
      else if (res.stren < driver[i].stren)
        res = driver[i];
endfunction

nettype `ana_value_t ana_net_t with res;
`define ana_net_t ana_net_t
```

Value Setting and Getting

- Because UDN types are flexible, helper functions are required
- Superset set function that can set all possible fields
- Superset get functions covering all possible fields
  - 'Reasonable' defaults

Guideline #3: Define superset of functions to ensure compile time compatibility

```
`define ana_value_t logic
`define ZSTATE  1'bz
`define XSTATE  1'bx
`define ana_net_t wire

//NOTE: value input must be of type 'logic' because
//      1'bx and 1'bz do not map to real values
function automatic `ana_value_t set_n (input logic value = 0,
                                       input real stren = 6);
  set_n = value;
endfunction

function automatic logic get_v (input `ana_value_t ana_net);
  get_v = ana_net;
endfunction

function automatic real get_s (input `ana_value_t ana_net);
  get_s = 6; //No strength, everything 'strong'
endfunction
```

```
typedef struct {
  real          value;
  real          stren;
} ana_value_t;
`define ana_value_t ana_value_t
`define ZSTATE -100
`define XSTATE  100

function automatic `ana_value_t set_n (input real value = 0,
                                       input real stren = 6);
  set_n.value = value;
  set_n.stren = stren;
endfunction

function automatic real get_v (input `ana_value_t ana_net);
  get_v = ana_net.value;
endfunction

function automatic real get_s (input `ana_value_t ana_net);
  get_s = ana_net.stren;
endfunction
```

UDN Type Checking

- Flexibility requires additional model complexity
  - Different behavior based on type
  - Additional UDNs for model level verification
- Many times, incoming signals are ‘OK’ if they fall within a certain range:

```
assign vref_ok  = (get_udn_type() == LOGIC) ? get_v(vref)  : (get_v(vref) inside {[MIN:MAX]});
```

- Organize these ‘OK’ level checks in a common location in the model

Guideline #4: Maintain minimal set of SoC and subsystem level UDNs with explicit type naming scheme and organize checks in common location in models

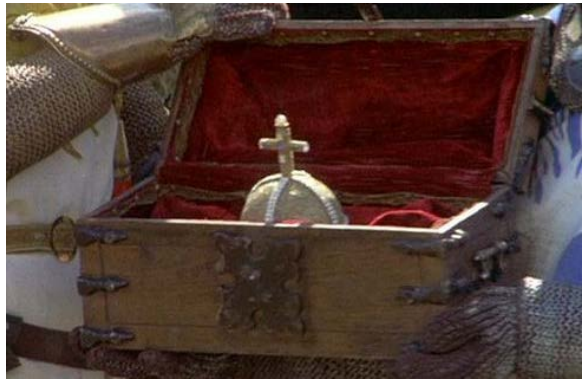
UDN Bi-Dir Switches

- How to model a bidirectional switch with UDNs?

So sayeth the LRM

Burn the tools!!

UDN capable switch primitives



- With simple wrapper, can use tran, tranif0, and tranif1 gates with UDNs
- Will likely be added to the LRM in a future release

Images from Monty Python Quest for the Holy Grail 1974

Integration Tips

- Along with UDNs, the LRM introduced ‘interconnect’
  - Typeless hierarchy connectors
  - All end points must be of the same type
- Synthesis and APR tools replace these with ‘wire’
  - To use UDNs in a LRM compliant manner, need to write a script to replace wire with interconnect
- All the simulation tools treat ‘wire’ as ‘interconnect’ as long as interconnect rules are followed

Guideline #5: Don’t use interconnect, always use wires for hierarchical transport of UDNs

Real Coverage and Stimulus

- The LRM says that only integral types can be used for randomization/coverage
  - but all vendors now support constrained random real stimulus and coverage!
- Be aware that it doesn't always just work!
  - There are several gotchas noted in the paper
  - Some tools pull a special license

Guideline #6: Use constrained random real stimulus and real coverage collection if you have the license capacity

```
rand real value;
constraint value_dist {
  value dist { `ZSTATE    := 1,
               [0.7:0.9] :/ 1,
               [0.9:1.2] :/ 20};
}

covergroup cg_value;
  cvp_real_value: coverpoint value {
  `ifdef VENDOR_C
    bins highz  = {[`ZSTATE-0.1:`ZSTATE+0.1]};
  `else
    bins highz  = {`ZSTATE};
  `endif
    bins low   = {[0.7:0.9]};
    bins valid = {[0.9:1.2]};
  }
endgroup
```

UPF for Behavioral Models

- What are UPF aware behavioral models?
  - React according to UPF supply connections
  - Complex corruption semantics handled entirely by the model
  - Generate supply voltages for other blocks (e.g. regulators)

```
module foo (
            input UPF::supply_net_type vdd, vss,
            inout `ana_net_t           vref_out,
            output wire                rdy);
  real vdd_real, vss_real;
  assign vdd_real = $itor(vdd.voltage) / (10**6);
  assign vss_real = $itor(vss.voltage) / (10**6);
  assign vdd_ok   = (vdd.state == UPF::FULL_ON) && (vdd_real inside {[ 1.8 : 3.3]});
  assign vss_ok   = (vss.state == UPF::FULL_ON) && (vss_real inside {[-0.1 : 0.1]});
  assign rdy      = 1'b1;
  assign vref_out = (get_udn_type() == LOGIC) ? set_n(1'b1) : set_n(0.700);
  assign rdy      = (vdd_ok && vss_ok) ? 'z      : 'x;
  assign vref_out = (vdd_ok && vss_ok) ? `ZSTATE : `XSTATE;
endmodule
```

Disabling Automatic Corruption

- Disabling automatic corruption semantics must be done for UPF aware models with real UDNs
  - Corruption sets outputs to default state, for reals, this is 0
  - Initial blocks are not retriggered when power comes back up!
  - Automatic corruption occurs immediately, no delays!
- Setting module attribute is portable and behavior is self contained:

```
(* UPF_simstate_behavior = "DISABLE" *) module ldo (
```

Guideline #7: Always disable automatic UPF corruption in behavioral models with the UPF_simstate_behavior SV attribute

UPF Modelling Tips, Tricks, and Gotchas

- Each vendor interpreted the UPF LRM differently for SV implementation:
  - Always prefix UPF package types and enumerations with 'UPF::'
  - Avoid using UPF package functions like get_supply_state(), supply_on(), etc.
  - Create a common define for the state enumeration
    - Can be either UPF::upfSupplyStateE or UPF::state depending on your tool
- Always drive or read the UPF supply net struct fields directly:

```
UPF::supply_net_type vout, vin;
real vin_real;
assign vin_real = $itor(vin.voltage) / (10**6);
assign vin_ok   = (vin.state == UPF::FULL_ON) && (vin_real inside {[ 1.8 : 3.3]});
always @*
  if (vin_ok) begin
    vout.state   = UPF::FULL_ON;
    vout.voltage = $rtoi(1.0 * 10**6);
  end else begin
    vout.state   = UPF::OFF;
    vout.voltage = 0;
  end
```

Guideline #8: Follow UPF modelling rules on this page

Conclusion

- UDN based, UPF aware behavioral models add many new modelling and verification capabilities
- The vendors are working to add features to make our lives easier
- Following the guidelines in this paper will ensure that every just works!