

Some ideas regarding Verilog-AMS/SystemVerilog interoperability to support AMS assertions

Version 1.2

It has been proposed that: “The AL host language may be either IEEE 1800-2009 SystemVerilog, Accellera Verilog-AMS v2.3 or a designated successor language. A future SystemVerilog-AMS formulated with the compatibility of AL in mind will also be a host. The objects in AL formulas are restricted to those of the host language.”

Above is the original proposed restriction. We propose several relaxations to the restriction. These relaxations improve the utility and user friendliness of the resulting assertions. The idea is to reduce the amount of indirection and user created “infrastructure” code required to write assertions using both SV and VAMS values by allowing specific types to be passed between SV and VAMS using the module instantiation. Under this proposal, some changes will be required in both SV and VAMS. We propose that:

- SVA would be extended with the necessary constructs to support AMS assertions. This extension will be referred to as AMS-SVA and is considered part of SV for purposes of the following description.
- SV modules can be instantiated in VAMS. We are not proposing the instantiation of VAMS modules within SV modules.
- The actuals in the SV module instantiation within a VAMS module can be any legal VAMS `expression` or `analog_expression` that results in an SV compatible type. The SV compatible types would be: integer, real, realtime, time, reg, wreal, and event. The receiving SV formal should be an input port whose type is assignment compatible with the actual’s type. This capability requires that the larger Verilog-AMS committee define the meaning of a continuous assignment of an `analog_expression` in a discrete context. Some examples of what would be legal are described below:
 - VAMS wreals can connect to SV input var real ports.
 - VAMS V(a) can connect to SV input var real ports.
 - VAMS reg variables can connect to SV input logic ports.
 - VAMS real variables can connect to SV input var real ports.
 - VAMS integer variables can connect to SV input integer ports.
- Events are a legal VAMS type, but not currently allowed in module instantiations. We would like to add events and `analog_event_control` (cross, timer, etc.) to the list of allowable expressions in a VAMS module instantiation.

- Any port types allowed by SV can be present in the module port list, but types not compatible with VAMS must be accessed via a hierarchical or out of module reference to another SV module.
- We also propose the addition of a **bind** construct to VAMS.

Proposed changes for VAMS:

- Capability for any legal VAMS `expression` or `analog_expression` to be passed via module instantiation to an assignment compatible SV input port.
- Addition of `event` and `analog_event_control` to the list of allowed expressions in module instantiation.
- Addition of VAMS `bind`.

Proposed changes for SV:

- Allow compatible VAMS types to connect to compatible SV input port types.
- Additions to SVA necessary to support AMS assertions.

An example illustrating several of the above items follows.

```
<BEGIN top.v>
module top;
  mVAMS mVAMS1(.rSV(rSV),.lSV(lSV),.rVAMS(rVAMS),.lVAMS(lVAMS));
  mSV mSV1(.rVAMS(rVAMS),.lVAMS(lVAMS),.rSV(rSV),.lSV(lSV));
endmodule
<END top.v>
```

```
<BEGIN mVAMS.vams>
`include "constants.vams"
`include "disciplines.vams"

module mVAMS (rSV,lSV,rVAMS,lVAMS);
  input rSV;
  input lSV;
  output rVAMS;
  output lVAMS;

  wreal rSV;
  logic lSV;
```

```

wreal rVAMS;
reg lVAMS;
electrical n;
event timerEv;
real rVal;
integer i;

assign rVAMS = rVal;

initial rVal = 0.0;

always @(posedge lSV) begin
    if (rSV > 2.5)
        lVAMS <= 1'b1;
    else
        lVAMS <= 1'b0;
end

always @(cross(V(n) - 2.0)) begin
    -> timerEv;
    if(rVal > 2.5)
        rVal = 0.0;
    else
        rVal = 4.0;
end

analog begin
    V(n) <+ transition(rVal,1n,2n,5n);
end

mAmsSva(.rVAMS(rVAMS),
        .lVAMS(lVAMS),
        .rSV(rSV),
        .lSV(lSV),
        .Vn_VAMS(V(n)),
        .cross_VAMS(cross(V(n) - 2.5)),
        .timer_VAMS(timerEv),
        .i(i),
        .r(rVal)
    );

```

```

endmodule
<END mVAMS.vams>

<BEGIN mSV.sva>
module mSV (
    input real rVAMS,
    input logic lVAMS,
    output real rSV,
    output logic lSV
);

initial begin
    rSV = 0.0;
    lSV = 1'b1;
    forever #10 lSV = ~lSV;
end

always @(posedge lVAMS) begin
    if (rVAMS > 2.5)
        rSV = 1.2;
    else
        rSV = 4.8;
end

endmodule
<END mSV.sva>

<BEGIN mAmsSva.sva>
module mAmsSva (
    input real rVAMS,
    input logic lVAMS,
    input real rSV,
    input logic lSV,
    input var real Vn_VAMS,
    input var event cross_VAMS,
    input var event timer_VAMS,
    input var integer i,
    input var real r
);

```

```
default clocking CLK @(posedge lSV); endclocking

assert property(lVAMS |-> rVAMS > 0.0 || rSV < 2.5);
assert property(@cross_VAMS (Vn_VAMS == 0.0 || Vn_VAMS == 4.0));
assert property(@timer_VAMS (Vn_VAMS == 0.0 || Vn_VAMS == 4.0));
endmodule
<END mAmsSva.sva>
```