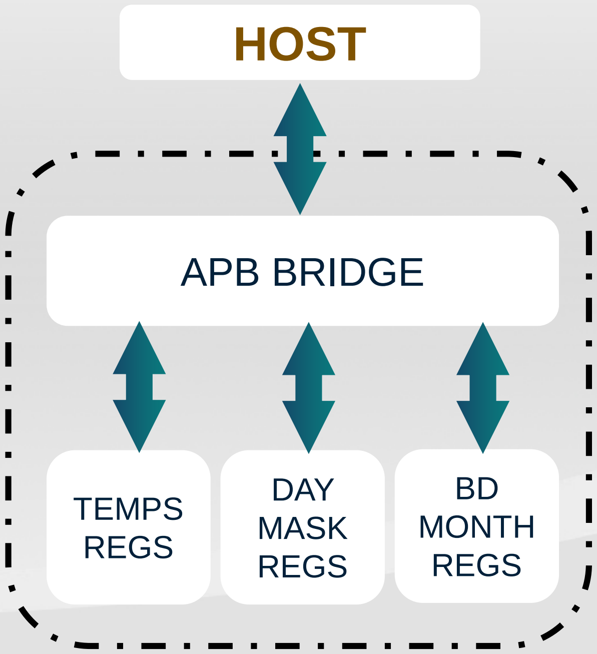
SystemVerilog Design: User Experience Defines Multi-Tool, Multi-Vendor Language Working Set

APB Example Code

Tool vendors and designers: Use this code to screen your tools

APB EXAMPLE CODE

APB Example



apb_pkg.sv

Package definition.

```
package apb_pkg;
    // AMBA 3 APB Protocol Specification v1.0

    // Addr width unspecified by APB
    // so make a type for flexibility.
    typedef logic [19:2] apb_addr_t;

    typedef enum apb_addr_t {
        REG_BD_MONTH_MOM        = 'h40,
        REG_BD_MONTH_DAD        = 'h41,
        REG_BD_MONTH_DAUGHTER   = 'h42,
        REG_BD_MONTH_SON        = 'h43,
        REG_DAY_MASK            = 'h50,
        REG_TEMP_JAN            = 'h61,
        REG_TEMP_FEB            = 'h62,
        REG_TEMP_MAR            = 'h63,
        REG_TEMP_APR            = 'h64,
        REG_TEMP_MAY            = 'h65,
        REG_TEMP_JUN            = 'h66,
        REG_TEMP_JUL            = 'h67,
        REG_TEMP_AUG            = 'h68,
        REG_TEMP_SEP            = 'h69,
        REG_TEMP_OCT            = 'h6a,
        REG_TEMP_NOV            = 'h6b,
        REG_TEMP_DEC            = 'h6c
    } apb_addr_e;
```

User defined type as enum base type.

Explicit value assignments.

User defined type.

Logic type.

```
    // Data width unspecified by APB
    // so make a type for flexibility.
    typedef logic [31:0] apb_data_t;

    // APB request (master to slave).
    typedef struct packed {
        apb_addr_e addr;
        logic      enable;
        logic      write;
        apb_data_t wdata;
    } apb_req_s;

    // APB response (slave to master).
    typedef struct packed {
        logic      ready;
        apb_data_t rdata;
    } apb_resp_s;

    typedef enum int {
        SLAVE_TEMPS,
        SLAVE_BD_MONTH,
        SLAVE_DAY_MASK,
        SLAVE_NUM
    } SLAVE_e;

endpackage
```

Packed struct.

Int as enum type base.

Implicit value assignments.

other_pkg.sv

```
package other_pkg;

    typedef logic [3:0] MONTH_t;
    typedef enum MONTH_t {
        JAN, FEB, MAR, APR, MAY, JUN,
        JUL, AUG, SEP, OCT, NOV, DEC
    } MONTH_e;

    typedef enum int { LO, HI } RANGE_e;

    typedef enum int {
        MOM, DAD, DAUGHTER, SON,
        FAMILY_SIZE
    } FAMILY_e;

    typedef logic [7:0] temp_t;

endpackage
```

apb_if.sv

Interface definition.

Wildcard package import.

Function within an interface.

Interface modport.

```
interface apb_if;

    import apb_pkg::*;

    logic       sel;
    apb_req_s   req;
    apb_resp_s resp;

    function automatic logic WriteReg( input apb_addr_e addr );
        WriteReg = sel & req.write & req.enable & (req.addr == addr);
    endfunction

    function automatic logic ReadReg( input apb_addr_e addr );
        ReadReg = sel & ~req.write & req.enable & (req.addr == addr);
    endfunction

    modport master( output req, sel, input  resp );
    modport slave ( input  req, sel, output resp,
                    import WriteReg, import ReadReg );

endinterface
```

reg_day_mask.sv

Interface as a port.

always_ff

Package import within a module.

‘1 literal fills vector with ones.

Struct with default assignment.

Call to a function inside an interface.

$bits system function.

Casting to pad to desired vector width..

Struct assignment by field.

Referencing items inside an interface.

```
module reg_day_mask ( input logic clk, input logic ares,
                      apb_if.slave apb, output logic [31:1] dayMask );

    import apb_pkg::*;

    always_ff @( posedge clk or posedge ares )
        if ( ares ) begin
            dayMask  <= '1;
            apb.resp <= '{ default: 0 };
        end
        else begin
            if ( apb.WriteReg( REG_DAY_MASK ) )
                dayMask <= apb.req.wdata[0+:$bits(dayMask)];

            if ( apb.ReadReg( REG_DAY_MASK ) )
                apb.resp <= '{ ready: 1, rdata: apb_data_t'(dayMask) };
            else
                apb.resp <= '{ default: 0 };
        end

endmodule
```

reg_bd_month.sv

Package import outside of a module because definitions are needed in the port list.

```
import other_pkg::*;

module reg_bd_month ( input logic clk, input logic ares,
                      apb_if.slave apb,
                      output MONTH_e [FAMILY_SIZE-1:0] bdMonth );

   import apb_pkg::*;

   apb_resp_s resp;
   const apb_addr_e start_reg = REG_BD_MONTH_MOM;

   always_comb begin
      resp = '{ default: 0 };
      foreach ( bdMonth[i] ) begin
         if ( apb.ReadReg( start_reg.next(i) ) )
            resp = '{ ready: 1,
                      rdata: apb_data_t'( bdMonth[i] ) };
      end
   end
```

always_comb

Packed array of enum as a port.

Struct variable declaration.

Constant enum declaration.

foreach loop.

reg_bd_month.sv (continued)

```
    localparam int width = $bits(MONTH_e);

    always_ff @( posedge clk or posedge ares ) begin
        if ( ares ) begin
            bdMonth  <= '{ default: bdMonth[$low(bdMonth)].first };
            apb.resp <= '{ default: 0 };
        end
        else begin
            foreach ( bdMonth[i] ) begin
                if ( apb.WriteReg( start_reg.next(i) ) )
                    bdMonth[i] <= MONTH_e'(apb.req.wdata[0+:width]);
            end
            apb.resp <= resp;
        end
    end

endmodule
```

Enumerated method .first

Packed array element default value assignment.

$low system function.

Enumerated method .next
with increment value.

Casting a vector to
an enumerated type.

reg_temps.sv

```
import other_pkg::*;

module reg_temps ( input logic clk, input logic ares,
                   apb_if.slave apb,
                   output temp_t [DEC:JAN][HI:LO] temps );

    import apb_pkg::*;

    apb_resp_s resp;
    const apb_addr_e start_reg = REG_TEMP_JAN;
    const int shift = 16;

    always_comb begin
        resp = '{ default: 0 };
        foreach ( temps[month,range] ) begin : rd_loop
            apb_data_t rdata;
            rdata = apb_data_t'(temps[month][range]);
            if ( apb.ReadReg( start_reg.next(month) ) ) begin
                resp.ready = 1;
                resp.rdata = resp.rdata | ( rdata << (range*shift) );
            end
        end
    end
```

Multidimensional packed array as a port.

foreach loop with multiple dimensions.

reg_temps.sv (continued)

```
    always_ff @( posedge clk or posedge ares ) begin
        if ( ares ) begin
            temps    <= 0;
            apb.resp <= '{ default: 0 };
        end
        else begin
            foreach ( temps[month,range] ) begin : wr_loop
                localparam int width = $bits(temp_t);
                apb_data_t wdata;
                wdata = apb.req.wdata[shift+:width];
                if ( apb.WriteReg( start_reg.next(month) ) )
                    temps[month][range] <= wdata;
            end
            apb.resp <= resp;
        end
    end

endmodule
```

Packed array assigned directly to zero.

apb_bridge.sv

Array of interfaces as a port.

$size system function.

Unpacked array element default value assignment.

Inside operator with range bounds.

Inside operator with individual match.

```
import apb_pkg::*;

module apb_bridge ( apb_if.slave master,
                    apb_if.master slave[SLAVE_NUM] );

    logic sel [$size(slave)];
    always_comb begin
        if ( ~master.sel )
            sel = '{ default: 0 };
        else begin
            sel[SLAVE_TEMPS]    = master.req.addr
                inside { [REG_TEMP_JAN:REG_TEMP_DEC] };
            sel[SLAVE_BD_MONTH] = master.req.addr
                inside { REG_BD_MONTH_MOM, REG_BD_MONTH_DAD,
                         REG_BD_MONTH_SON, REG_BD_MONTH_DAUGHTER };
            sel[SLAVE_DAY_MASK] = master.req.addr == REG_DAY_MASK;
        end
    end
```

apb_bridge.sv (continued)

Genvar without the use of the generate keyword.

Loop variable declaration inside a for loop.

Packed array of a struct.

```
   apb_resp_s resp [$size(slave)];

   for ( genvar i = 0; i < $size(slave); i++ ) begin : gen_select
      always_comb begin
         slave[i].sel = sel[i];
         slave[i].req = sel[i] ? master.req
                                : '{ addr: apb_addr_e'(0),
                                     default: 0 };
         resp[i]      = slave[i].resp;
      end
   end

   assign master.resp = resp.or;

endmodule
```

Plus-plus operator.

Struct assignment with enum and default.

Decomposition of array of interfaces using a genvar.

Unpacked array reduction method.

apb_top.sv

Direct package scope resolution.

```
module apb_top ( input logic clk, input logic ares,
                 apb_if.slave host );

    import apb_pkg::SLAVE_TEMPS;
    import apb_pkg::SLAVE_BD_MONTH;
    import apb_pkg::SLAVE_DAY_MASK;

    apb_if slave[apb_pkg::SLAVE_NUM]();

    apb_bridge   bridge (.master(host), .slave);

    reg_temps    temps  (.apb(slave[SLAVE_TEMPS]),    .temps(),   .*);
    reg_bd_month bdmonth(.apb(slave[SLAVE_BD_MONTH]), .bdMonth(), .*);
    reg_day_mask daymask(.apb(slave[SLAVE_DAY_MASK]), .dayMask(), .*);

endmodule
```

Array of interfaces declaration.

Implicit (.name) port assignment with an array of interfaces.

Wildcard port assignments.

Each slide represents an actual tool issue overcome during project development

TOOL WORKAROUNDS

Enums as index

- **Original code:**

```
typedef enum int { BUS_A, BUS_B, BUS_C } bus_targets_e;
logic [BUS_C:BUS_A] all_valid;
logic valid;
assign valid = |all_valid;
block_a u_block_a ( all_valid[BUS_A] );
block_b u_block_b ( all_valid[BUS_B] );
block_c u_block_c ( all_valid[BUS_C] );
```

- **Unsuccessful workaround attempt:**

```
always_comb begin
     valid = 0;
     for ( int i = BUS_A; i <= BUS_C; i++ )
         valid = valid | all_valid[i];
end
```

- **Workaround code:**

```
assign valid = all_valid[BUS_A] | all_valid[BUS_B] | all_valid[BUS_C];
```

- **Conclusion:**
  - Tool complains that bits of all_valid are not driven
  - Explicitly OR the bits together
  - May also work with localparam instead of enum

Use of enum in an interface select

- **Original code:**

```
interface readback_if;
    logic [31:0] data;
    logic valid;
endinterface

typedef enum int { UART, SPI } target_e;
readback_if rdata[SPI:UART]( );

logic valid;
logic [31:0] data;
always_comb begin
    valid = rdata[UART].valid | rdata[SPI].valid;
    data  = rdata[UART].data  | rdata[SPI].data;
end
```

- **Workaround code:**

```
valid = rdata[int'(UART)].valid | rdata[int'(SPI)].valid;
data  = rdata[int'(UART)].data  | rdata[int'(SPI)].data;
```

- **Conclusion:**
  - Cast interface select enums to an int

Broadside struct default assignments

- **Original code:**

```
typedef struct packed {
    logic [4:0] hour;
    logic [5:0] minute, second;
} time_s;

  time_s noon;

  always_comb begin
      noon      = '0; // Should assign all fields to zero
      noon.hour = 12;
  end
```

- **Workaround code:**

```
  always_comb begin
      noon.second = 0;
      noon.minute = 0;
      noon.hour   = 12;
  end
```

- **Conclusion:**
  - Tool complains that noon.second and noon.minute are unassigned
  - Explicitly assign each field of the struct

Parameter keyword in parameter port lists

IEEE Std 1800-2005 → 6.3.4

- **Original code:**

```
package test_case_pkg;
    typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
endpackage

import test_case_pkg::*;
module test_case_module #( days_e THIS_DAY = days_e'( 0 ), int WIDTH = 1 )
    // Code here…
endmodule
```

- **Workaround code:**

```
import test_case_pkg::*;
 module test_case_module #( parameter days_e THIS_DAY = days_e'( 0 ), parameter int WIDTH = 1 )
  // Code here…
 endmodule
```

- **Conclusion:**
  - Add parameter keyword even though LRM says it isn't needed

Loss of type in parameter port lists after an enum

- **Original code:**

```
package test_case_pkg;
    typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
endpackage

import test_case_pkg::*;
module test_case_module #( days_e THIS_DAY = days_e'( 0 ), int WIDTH = 1 )
    // Code here…
endmodule
```

- **Workaround code:**

```
import test_case_pkg::*;
 module test_case_module #( days_e THIS_DAY = days_e'( 0 ), int WIDTH = int'( 1 ) )
    // Code here…
 endmodule
```

- **Conclusion:**
  - Add parameter keyword even though LRM says it isn't needed

Macro with open parenthesis on different line

- **Original code:**

```
always_ff @( posedge clk or posedge ares )
    if `RESET_MACRO
    (
         qTraffic_light <= RED;
     )
```

- **Workaround code:**

```
always_ff @( posedge clk or posedge ares )
    if `RESET_MACRO(
         qTraffic_light <= RED;
    )
```

- **Conclusion:**
  - Open parenthesis must be on the same line as the macro
  - Just plain Verilog

Use of enum with a genvar for loop

- **Original code:**

```
typedef enum int { HEADS, TAILS } coin_e;
logic [1:0] coin_side;
for ( genvar i = HEADS; i <= TAILS; i++ ) begin : gen_coin
    assign coin_side[i] = i;
end
```

- **Workaround code:**

```
for ( genvar i = int’( HEADS ); i <= int'( TAILS ); i++ ) begin : gen_coin
    assign coin_side[i] = i;
end
```

- **Conclusion:**
  - Recast enum to int

Casting using $bits as the vector size

- **Original code:**

```
typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;

days_e day_as_enum;
logic [7:0] day_as_byte;

assign day_as_enum = THU;
assign day_as_byte = $bits(day_as_byte)'( day_as_enum );
```

- **Code to fix one tool's order of operation issue:**

```
assign day_as_byte = ($bits(day_as_byte))'( day_as_enum );
```

- **Workaround code:**

```
typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
typedef logic [7:0] byte_t;

days_e day_as_enum;
byte_t day_as_byte;

assign day_as_enum = THU;
assign day_as_byte = byte_t'( day_as_enum );
```

- **Alternate code (do not use):**

```
assign day_as_byte = type(day_as_byte)'( day_as_enum );
```

- **Conclusion:**
  - Different issues in multiple tools
  - Recommend casting with `typedef` types or constants

accellera SYSTEMS INITIATIVE

Packed array of a type defined logic vector

- **Original code:**

```
typedef logic [7:0] byte_t;
byte_t [3:0] dword;
```

- **Workaround code:**

```
logic [3:0][7:0] dword;
 // Or…
 typedef logic [3:0][7:0] dword_t;
 dword_t dword;
```

- **Conclusion:**
  - Combine into a single declaration or typedef

Package import is forgotten

- **Original code:**

```
package test_case_pkg;
    typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
endpackage

import test_case_pkg::*;
module test_case_module;
    days_e day;
    assign day = WED;
endmodule
```

- **Workaround code:**

```
import test_case_pkg::*;
module test_case_module;
    days_e test_case_pkg::day;
    assign day = test_case_pkg::WED;
endmodule
```

- **Conclusion:**
  - Rare occurrence – no pattern to when or where
  - In such cases explicitly specify the source package

Enumerated module parameters loose their type when assigned to a struct or interface member

- **Original code:**

```
package test_case_pkg;
    typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
    typedef struct packed {
        days_e day;
        logic [4:0] hour;
    } time_s;
endpackage

import test_case_pkg::*;
module test_case_module #( parameter days_e THIS_DAY = SUN )
    ( input logic [4:0] hour; output time_s present );

    assign present.hour = hour;
    assign present.day  = THIS_DAY;

endmodule
```

- **Workaround code:**

```
    assign present.day  = days_e'( THIS_DAY );
```

- **Conclusion:**
  - Re-cast to remind the tool of the member's type

Enumerated methods (i.e. .first, .next)

IEEE Std 1800-2005 → 23.2

- **Original code:**

```
always_ff @( posedge clk or posedge ares )
    if ( ares )
        qTraffic_light <= qTraffic_light.first;
    else if ( timer_expired )
        qTraffic_light <= qTraffic_light.next;
```

- **Workaround code:**

```
always_ff @( posedge clk or posedge ares )
     if ( ares )
         qTraffic_light <= RED;
     else if ( timer_expired )
         case ( qTraffic_light )
             RED:     qTraffic_light <= GREEN;
             GREEN:   qTraffic_light <= YELLOW;
             default: qTraffic_light <= RED;
         endcase
```

- **Conclusion:**
  - Tool's documents explicitly state that enumerated methods aren't supported
  - No choice but to be explicit which decreases coding efficiency

ACCELLERA SYSTEMS INITIATIVE

Use .num method in constant expression

- **Original code:**

```
typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
days_e day;
localparam int days_per_week = day.num; // produces a value of 7
logic [days_per_week-1:0] busy_that_day;
```

- **Workaround code:**

```
localparam int days_per_week = 2**$bits( days_e ); // produces a value of 8
```

- **Conclusion:**
  - Brute force the vector creation by any number of methods
  - Method chosen is flexible and responds to changes in enumeration
    - But likely results in superfluous bits

Modport in instantiation port connectivity

- **Original code:**

```
interface payload_if;
    logic payload, rts, rtr;
    modport initiator( output payload, rts, input rtr );
    modport target   ( input payload, rts, output rtr );
endinterface

payload_if my_payload;
initiator_submodule u_initiator ( .my_payload( my_payload.initiator ) );
target_submodule    u_target    ( .my_payload( my_payload.target    ) );
```

- **Workaround code:**

```
payload_if my_payload;
initiator_submodule u_initiator ( .my_payload ); // Wildcard is now possible
target_submodule    u_target    ( .my_payload );
```

- **Conclusion:**
  - Only workaround is to be less specific and eliminate the modport in instance
  - The fringe benefit is that this allows the use of instance wildcarding

Implied flop enable to a flop that is always zero

- **Original code:**

```
logic [31:0] irq, set_irq;
logic set_src_a, set_src_b;
assign set_irq = { 15’b0, set_src_b, 15’b0, set_src_a };
always_ff @( posedge sclk_g or posedge ares ) begin
    if ( ares )
        irq <= '0;
    else
        for ( int i = 0; i < $bits( irq ); i++ )
            if ( set_irq[i] ) irq[i] <= 1'b1;
end
```

- **Workaround code:**

```
always_ff @( posedge sclk_g or posedge ares ) begin
    if ( ares )
        irq <= '0;
    else
        irq <= irq | set_irq;
end
```

- **Conclusion:**
  - Tool cannot separate an always zero flop from the coding style
  - Just plain Verilog

Enumerated constants as select in array of interfaces

- **Original code:**

```
typedef enum int { CABLE_BOX, BLURAY, NETWORK } video_sources_e;
interface stream_if;
    logic [31:0] data;
    logic rts, rtr;
endinterface

stream_if video_stream[2:0]();

assign video_stream[CABLE_BOX].rtr = cable_box_selected & rtr;
```

- **Workaround code:**

```
localparam int CABLE_BOX=0, BLURAY=1, NETWORK=2;
interface stream_if;
    logic [31:0] data;
    logic rts, rtr;
endinterface

stream_if video_stream[2:0]();

assign video_stream[CABLE_BOX].rtr = cable_box_selected & rtr;
```

- **Conclusion:**
  - Use a localparam in place of enum

Unpacked array of interfaces

- **Original code:**

```
interface stream_if;
    logic [31:0] data;
    logic rts, rtr;
endinterface

stream_if my_stream[3]();
```

- **Workaround code:**

```
interface stream_if;
    logic [31:0] data;
    logic rts, rtr;
endinterface

stream_if my_stream[2:0]();
```

- **Conclusion:**
  - Packed array format is accepted

Use of $clog2 in parameter definition to override a module

- **Original code:**

```
module inv_addr #( parameter int WIDTH = 8 )
    ( input [WIDTH-1:0] iAddr, output [WIDTH-1:0] oAddr );
    assign iAddr = ~oAddr;
    $display( “Addr width %d”, $bits( iAddr ) ); // Should be 8 but Conformal says 1
endmodule

localparam int WIDTH = $clog2( 256 );
inv_addr #( WIDTH ) uinv_addr ( .iAddr( addr_in ), .oAddr( addr_out ) );
```

- **Workaround code:**

```
localparam int WIDTH = 8;
inv_addr #( WIDTH ) uinv_addr ( .iAddr( addr_in ), .oAddr( addr_out ) );
```

- **Conclusion:**
  - Remove the $clog2 from the parameter definition

Generate loops for RAM instances

- **Original code:**

```
for ( genvar i = 0; i < 4; i++ ) begin : gen_ram
    cache_data u_ram
             (
              .clk        (ifRam.clk),
              .cs_n       (ifRam.csn),
              .addr       (addr[i]),
              .din        (wdata[i]),
              .we_n       (ifRam.wen),
              .dout       (rdata[i])
              );
```

- **Conclusion:**
  - Manually unroll the loop (too large to include code here)

Multi-line loops in macros

IEEE Std 1800-2005 → 23.2

- **Original code:**

```
`define RESET_MACRO(reset_code) (ares) begin reset_code end
`define RESET_MACRO_SYNC(reset_code,sres) \
            `RESET_MACRO(reset_code) else if (sres) begin reset_code end

logic [7:0][2:0] qArray1, qArray2;

always_ff @( posedge clk or posedge ares )
    if `RESET_MACRO_SYNC(
         for ( int i = 0; i < 8; i++ ) begin
              qArray1[i] <= i;
              qArray2[i] <= 7-i;
         end
         ,software_reset
    )
```

- **Workaround code:**

```
always_ff @( posedge clk or posedge ares )
    if `RESET_MACRO_SYNC(
         for ( int i = 0; i < 8; i++ ) qArray1[i] <= i;
         for ( int i = 0; i < 8; i++ ) qArray2[i] <= 7-i;
         ,software_reset
    )
```

- **Conclusion:**
  - Keep code on a single line until the semicolon
  - No begin/end allowed

`define using the `` syntax in macro

IEEE Std 1800-2005 → 23.2

- **Original code:**

```
`ifdef VENDOR_SPECIFIC_COVERAGE_TOOL_DEFINE
 `define COVERAGE(cov) /``/ pragma coverage cov
`else
 `define COVERAGE(cov) /``/ coverage comment for other tools
`endif
```

- **Workaround code:**

```
`ifdef VENDOR_SPECIFIC_COVERAGE_TOOL_DEFINE
 `define COVERAGE(cov) /``/ pragma coverage cov
`else
 `define COVERAGE(cov)
`endif
```

- **Conclusion:**
  - An empty macro seems to work

SYSTEMS INITIATIVE

Inline comments in nested macros

- **Original code:**

```
`define RESET_MACRO(reset_code) (ares) begin reset_code end
`define ENUM_FIRST(enumtype) enumtype'( 0 )

typedef enum logic [1:0] { RED, GREEN, YELLOW } traffic_light_e;
typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
traffic_light_e qTraffic_light;
days_e qDay;

always_ff @( posedge clk or posedge ares )
    if `RESET_MACRO(
         qTraffic_light <= `ENUM_FIRST( traffic_light_e ); // SVWORKAROUND qTraffic_light.first;
         qDay           <= `ENUM_FIRST( days_e );          // SVWORKAROUND qDay.first;
    )
```

- **Workaround code:**

```
always_ff @( posedge clk or posedge ares )
    // SVWORKAROUND qTraffic_light.first;
    // SVWORKAROUND qDay.first;
    if `RESET_MACRO(
         qTraffic_light <= `ENUM_FIRST( traffic_light_e );
         qDay           <= `ENUM_FIRST( days_e );
    )
```

- **Conclusion:**
  - Move comments out of the outermost macro
  - Loss of context for the comment

Use of default assign to struct with enum field

- **Original code:**

```
typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
typedef struct packed {
    days_e day;
    logic [4:0] hour;
} time_s;
time_s start_time;
assign start_time = '{ default: 0 };
```

- **Workaround code:**

```
assign start_time = '{ day: SUN, default: 0 };
```

- **Conclusion:**
  - Be explicit about enum assignments
  - Other tools complain with warnings not errors

Reuse of genvar for loop variable

- **Original code:**

```
typedef enum int { HEADS, TAILS } coin_e;
typedef enum int { ROCK, PAPER, SCISSORS } roshambo_e;
logic [1:0] coin_side;
for ( genvar i = HEADS; i <= TAILS; i++ ) begin : gen_coin
    assign coin_side[i] = i;
end
logic [2:0] roshambo_turn;
for ( genvar i = ROCK; i <= SCISSORS; i++ ) begin : gen_roshambo
    assign roshambo_turn[i] = i;
end
```

- **Workaround code:**

```
for ( genvar j = ROCK; j <= SCISSORS; j++ ) begin : gen_roshambo
    assign roshambo_turn[j] = j;
end
```

- **Conclusion:**
  - Scoping rules allow reuse
  - Change to a different genvar name to make it work

`begin_keywords/`end_keywords

- **Original code:**

```
`begin_keywords “1800-2005”
module test_case;
    // Only 1800-2005 compliant keywords allowed
endmodule
`end_keywords
```

- **Workaround code:**

```
module test_case;
    // No keyword protection
endmodule
```

- **Conclusion:**
  - Only provides keyword checking, not full syntax checking
  - With limited value, no harm in eliminating

Set operator inside used in continuous assignment

- **Original code:**

```
typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
days_e day;
logic is_weekend;

assign is_weekend = day inside { SUN, SAT };
```

- **Workaround code:**

```
 always_comb is_weekend = day inside { SUN, SAT };
```

- **Conclusion:**
  - Must use procedural assignment

Packed array of interfaces that doesn’t start at zero

- **Original code:**

```
interface month_if;
    logic [4:0] day;
endinterface

month_if calendar_Q4_months[12:10]();
```

- **Workaround code:**

```
month_if calendar_Q4_months[2:0]();
```

- **Conclusion:**
  - Some tools mistake [12:10] as a bit slice
    - Bit-sliced interface usage is explicitly called out as illegal in the LRM
  - Reverting to [n:0] loses its self-documenting appearance

`define using the `` syntax as non-macro

IEEE Std 1800-2005 → 23.2

- **Original code:**

```
`ifdef VENDOR_SPECIFIC_COVERAGE_TOOL_DEFINE
 `define COVERAGE_ON   /``/ pragma coverage on
 `define COVERAGE_OFF  /``/ pragma coverage off
`else
 `define COVERAGE_ON   /``/ coverage comment for other tools
 `define COVERAGE_OFF  /``/ coverage comment for other tools
`endif
```

- **Workaround code:**

```
`ifdef VENDOR_SPECIFIC_COVERAGE_TOOL_DEFINE
 `define COVERAGE(cov) /``/ pragma coverage cov
`else
 `define COVERAGE(cov) /``/ coverage comment for other tools
`endif
```

- **Conclusion:**
  - No issue with `` syntax when used in a `define macro

Non-overridden parameter in an interface

- **Original code:**

```
interface stream_if;
    parameter NDATA = 32;
    logic [NDATA-1:0] data;
    logic rts, rtr;
endinterface
```

- **Workaround code:**

```
interface stream_if;
    logic [31:0] data;
    logic rts, rtr;
endinterface
```

- **Conclusion:**
  - Parameter will throw an error even if the parameter is not overriden

Enumerated interface members loose their type

- **Original code:**

```
typedef enum logic [2:0] { SUNNY, OVERCAST, FOG, RAIN, SNOW } weather_e;
interface weather_report_if;
    logic update;
    weather_e condition;
endinterface

weather_report_if weather_report();
weather_e current_weather;

always_latch begin
    if ( weather_report.update )
        current_weather <= weather_report.condition;
end
```

- **Workaround code:**

```
always_latch begin
    if ( weather_report.update )
        current_weather <= weather_e'( weather_report.condition );
end
```

- **Conclusion:**
  - Re-cast to remind the tool of the member’s type

Enumerated module parameters assigned to an enumerated constant

- **Original code:**

```
package test_case_pkg;
    typedef enum logic [2:0] { SUN, MON, TUE, WED, THU, FRI, SAT } days_e;
endpackage

import test_case_pkg::*;
module test_case_module #( parameter days_e THIS_DAY = SUN )
    // Code here…
endmodule
```

- **Workaround code:**

```
import test_case_pkg::*;
module test_case_module #( parameter days_e THIS_DAY = days_e’( 0 ) )
 // Code here…
endmodule
```

- **Conclusion:**
  - Re-cast to remind the tool of the member’s type

Ternary after a for loop

- **Original code:**

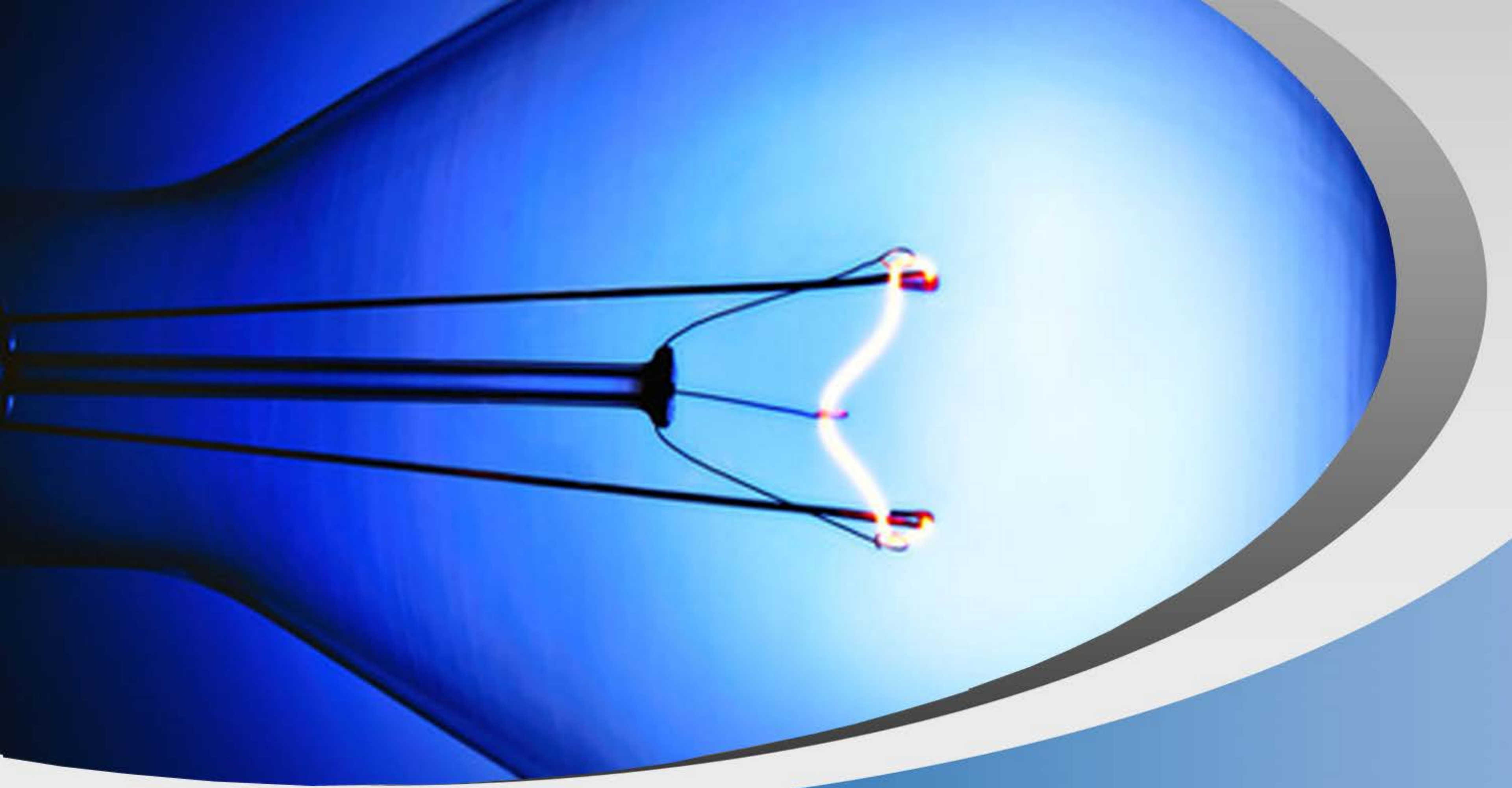
```
 logic [4:0] hour;
 logic [1:0] data;

always @( posedge hour_clk or posedge ares )
    if ( ares )
        data <= 1;
        hour <= ‘0;
    else begin
        for ( int i = 0; i < 2; i++ ) data[i] <= ~data[i];
        hour <= ( hour == 23 ) ? 0 : hour + 1; // Tool sees two coverage blocks
    end
```

- **Workaround code:**

```
    else begin
        hour <= ( hour == 23 ) ? 0 : hour + 1; // Tool sees one coverage block
        for ( int i = 0; i < 2; i++ ) data[i] <= ~data[i];
    end
```

- **Conclusion:**
  - Not a parsing issue!
  - Probably just a Verilog issue
  - Tool creates a superfluous coverage block which is never covered
  - Move the for loop after the code

ACCELLERA SYSTEMS INITIATIVE



Thank you!

SYSTEMS INITIATIVE