2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

What’s Next for SystemVerilog in the Upcoming IEEE 1800 standard

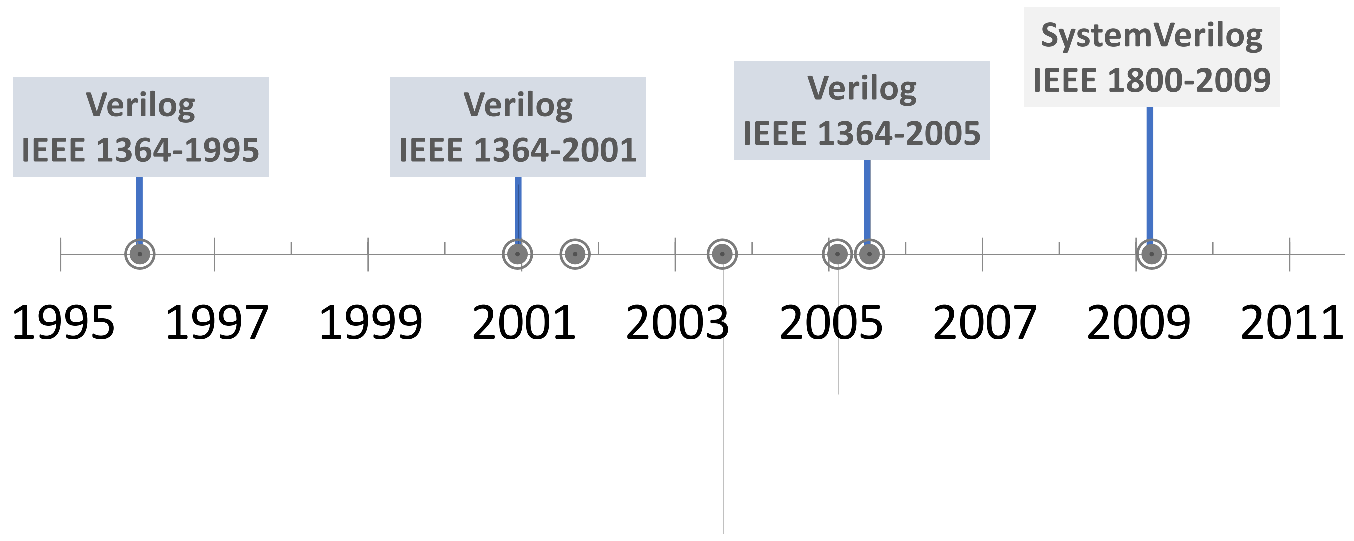
Dave Rich

Siemens EDA

SIEMENS

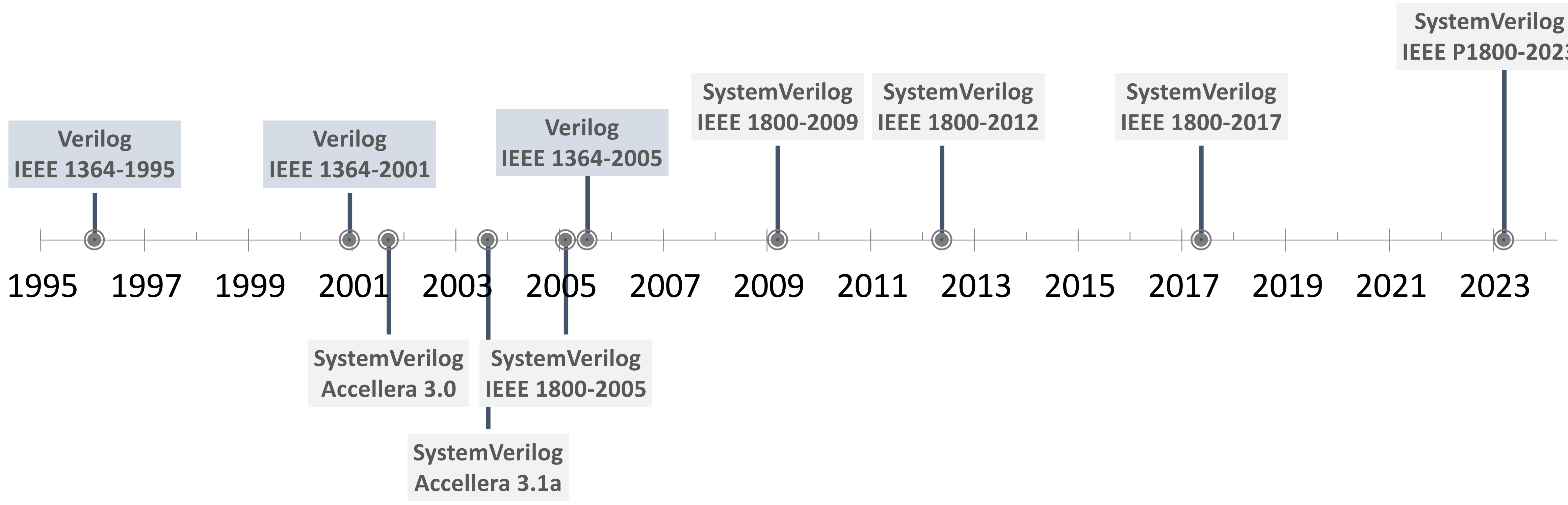
accellera SYSTEMS INITIATIVE

Revisions of Verilog



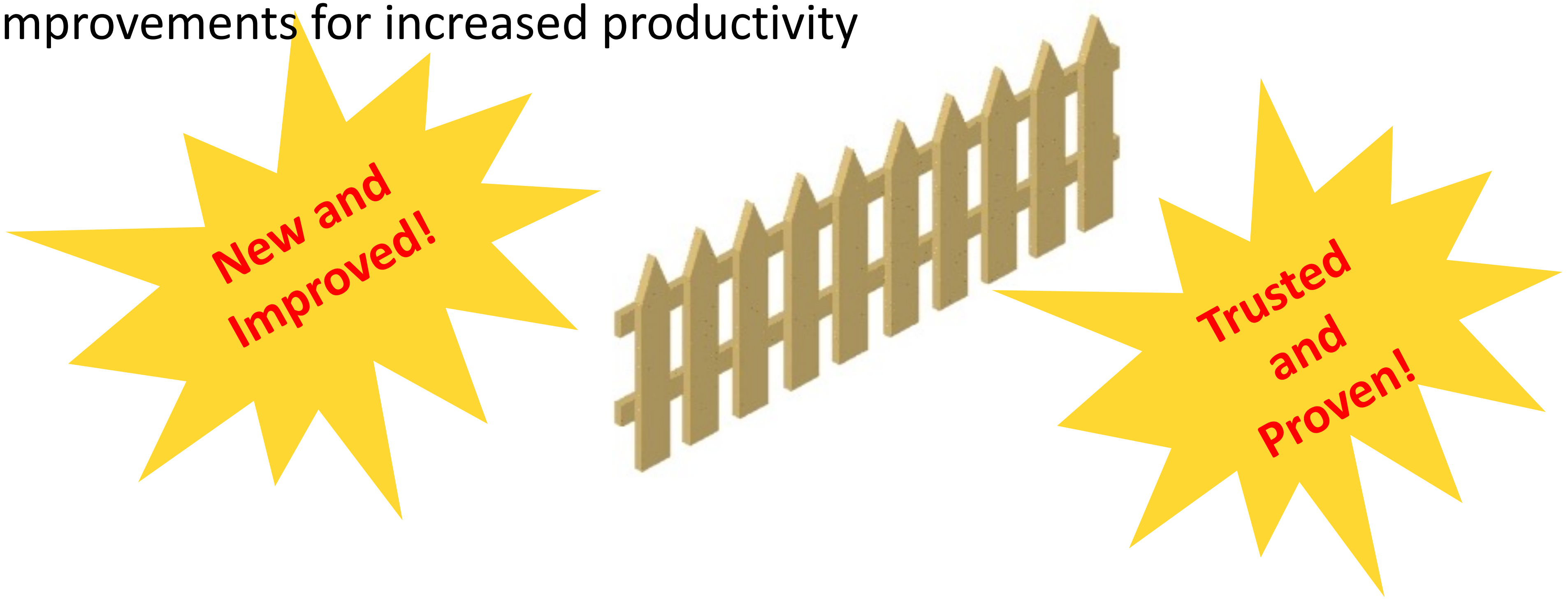
- Verilog 1.0 released 1985
- SystemVerilog 1800-2009 supersedes Verilog

Revisions of SystemVerilog


Verilog IEEE 1364-1995
Verilog IEEE 1364-2001
Verilog IEEE 1364-2005
SystemVerilog IEEE 1800-2009
SystemVerilog IEEE 1800-2012
SystemVerilog IEEE 1800-2017
SystemVerilog IEEE P1800-2023
SystemVerilog Accellera 3.0
SystemVerilog IEEE 1800-2005
SystemVerilog Accellera 3.1a
1995
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015
2017
2019
2021
2023

Which side of the technology fence are you on?

- Stability with existing tool ecosystems
- Need improvements for increased productivity



IEEE P1800 SystemVerilog Working Group

- Entity based group of end users and tool vendors formed in 2020
  - Accellera, ARM, Cadence, Infineon, Intel, JEITA, Marvell, Nvidia, Qualcomm, Siemens, Sigasi, TI, Verific
- Gathered issues to be addressed and published by the end of 2023
  - Publish in 2024
- Mantis Issue Tracking Database
  - https://accellera.mantishub.io
  - 3300+ issues captured since 2004
  - 1100+ issues still open

Issue categories

- Enhancements—New features extending the capabilities of the language
  - May already be implemented in some tools or, completely new features
  - “Borrowed” from other languages; use models are well known
- Errata—Obvious mistakes in the current LRM
  - Small typographical or editing errors
  - Large contradictions in the text between different LRM sections
  - Sometimes current wording of the LRM is obviously incorrect and tools have implemented what the LRM should have said in the first place
- Clarifications—Missing or misleading text where the LRM is ambiguous
  - Normally does not involve a change to a tool’s behavior unless tool implements something very different from original intent

Enhancements

- Extending coverpoints (Mantis 4703)
- Unpacked array mapping function (Mantis 7610)
- `ifdef Boolean combination of identifiers (Mantis 1084)
- Add support for multiline strings (Mantis 7308)
- Real number modeling (Mantis 7295 and 7669)
- Chaining of method calls (Mantis 2735)
- Adding static ref arguments (Mantis 2583)

Extending coverpoints (Mantis 4703)

- Allow adding a new coverpoint in an extended class and crossing it with an existing coverpoint in the base class
- Replacing the bin structure of an existing coverpoint or removing it entirely

```
class pixel;
  bit [7:0] level;
  enum {OFF,ON,BLINK,REVERSE} mode;
  covergroup g1;
    a: coverpoint level;
    b: coverpoint mode;
  endgroup
  function new();
    g1 = new;
  endfunction
endclass
```

```
class colorpixel extends pixel;
  enum {red,blue,green} color;
  covergroup extends g1;
    b: coverpoint mode { // the coverpoint
     // ‘b’ from the base class is changed
      ignore_bins ignore = {REVERSE};
    }
    cross level, color; // ‘level’ comes
                              // from the base class
  endgroup
endclass
```

Unpacked array map function (Mantis 7610)

- Assignments between arrays with inequivalent types
- Create new arrays based on operations with each element

```
int A[3] = {1,2,3};
byte B[3];
int C[3];
// assigns and casts array of int to an array of byte
B = A.map() with ( byte’(item) );
// increments each element of the array (use b instead of item)
B =  B.map(b) with ( b + 8’b1 ); // B becomes {2,3,4}
// Add two arrays
C = A.map(a) with (a + B[a.index] );  // C becomes {3,4,5}
```

`ifdef Boolean combination of identifiers (Mantis 1084)

- Create simple Boolean conditional expressions ‘and’ (&&) ‘or’ (||)
- Replaces layers of intermediate macro definitions

```
// AND
`ifdef A
   `ifdef B
     `define A_and_B
   `endif
`endif
`ifdef A_and_B
   // code for AND
`endif
```

```
// OR
`ifdef A
   `define A_or_B
`endif
`ifdef B
   `define A_or_B
`endif
`ifdef A_or_B
   // code for OR
`endif
```

```
`ifdef (A && B)
   // code for AND condition
`endif
`ifdef (A || B)
   // code for OR condition
`endif
```

Support for multiline strings (Mantis 7308)

- Easier to write informative or self-documenting messages so they can be easily read and maintained within the source code.
- Required addressing many existing, older issues:
  - Multi-line string literals in a text macro (Mantis 1397)
  - Special characters in strings (Mantis 1507)
  - “Printable” characters (Mantis 7562)

```
string x = """
This is one continuous string.
Single ' and double " can
be placed throughout, and
only a triple quote will end it.
"""
```

Real number modeling issues (Mantis 7295 and 7669)

- Real number binary format creates many problems for expressions
  - Decimal 0.1 is binary repeating pattern 0.00011001100110011...
  - Round-off errors to 52-bits of precision means that 0.1 + 0.2 ≠ 0.3
- Constraints and covergroup expressions were designed around discrete values and ranges
- Use tolerances instead of equality for complex expressions with reals

```
rvalue == 0.3 // for simple constraints
rexp inside {[0.3 +/- 0.003]}
rexp inside {[0.3 +%- 1]}
// equivalent to
rexp inside {[0.297:0.303]}
```

Reals in Constraints and Covergroups

- Reals in constraints is just relaxation of rules
  - No new syntax or rules
- Reals in coverpoints requires new bin options and range rules
  - option real_interval instead of discrete values

```
type_option.real_interval = 0.2;
coverpoint rvalue { // requires explicit bins
  bins a    = {0.3 +%- 1}; // 1 bin {[0.297:0.303] }
  bins b[] = {[1.0:1.5]}; // 3 bins
}                            // = 1.0 < 1.2
                             // = 1.2 < 1.4
                             // = 1.4 <= 1.5
```

Chaining of method calls (Mantis 2735)

- Use the result of a function to serve as an intermediate variable for selecting a member of the result
  - Mainly used when the result is a handle to a class.

```
class A;
  int member=123;
endclass
```

```
module top;
  A a;
  function A F(int arg=0);
    int member; // static variable uninitialized value 0
    a = new();
    return a;
  endfunction
  initial begin
    $display(F.member);   // 0 - No "()", Verilog hierarchical reference
    $display(F().member); // 123 - With "()", implicit variable
   end
endmodule
```

Adding static ref arguments (Mantis 2583)

- Removes many restrictions on pass-by-reference arguments
  - Become the target of a nonblocking assignment.
  - Have a reference on the LHS or RHS of force statement.
  - Be referenced from within a fork join_any/join_none process.

```
module top;
  function void monitor(ref static logic arg);
    fork // the reference to arg only becomes legal with a static qualifier
      forever @(arg) $display("arg changed at time %t", arg, $realtime);
    join_none
  endfunction
  logic C;
  initial monitor(C);
endmodule
```

Errata

- ***@(clocking_block_name) is unequal to its associated clocking event (mantis 7172)***
- ***non-integral function actual argument allowed in a constraint expression (mantis 2841)***

@(clocking_block_name) is unequal to its associated clocking event (mantis 7172)

- The current LRM has this brief example in section 14.10:

```
clocking dram @(posedge phi1); inout data;
output negedge #1 address; endclocking
```

The clocking event of the `dram` clocking block can be used to wait for that particular event:

```
@(dram);
```

The preceding statement is equivalent to `@(posedge phi1)`.

- Cannot be true since section 14.13 says the dram event gets triggered in the observed region

```
always @(posedge phi1) $display("clocking event");
always @(dram) $display("clocking block event");
```

The first **`always`** procedure in the preceding example executes in the same Active or Reactive event region as the positive edge of `phi1`. In contrast, the second **`always`** procedure executes after the Observed region following that Active or Reactive event region. This behavior can be used to avoid race conditions when sampling input data, see 14.13 for more details.

Corrected

Non-integral function actual argument allowed in a constraint expression (mantis 2841)

- The current LRM says this about constraints in section 18.3

> Constraints can be any SystemVerilog expression with variables and constants of integral type (e.g., bit, reg, logic, integer, enum, packed struct).

- Section 18.5.13 (Constraint guards) already contradicts this with object handles
- Not a problem as long as non-random expression result in an integral type

```
class A;
  rand bit [8:0] randvar;
  real threshold;
    constraint c{ randvar < int'(threshold); }
endclass
```

Clarifications

- Unclear which compiler directives must be alone on a line (Mantis 1014)
- Definition of term "blocking statement" (Mantis 225)
- Packed array shortcut – (Mantis 325)

Unclear which compiler directives must be alone on a line (Mantis 1014)

- Entangled with many other issues
- LRM mentions “lines of text” but newlines are only relevant in
  - Single quoted strings “I’m a string”
  - Single line comments // I’m a comment
  - End of macro definitions `define

```
   `ifdef NAME text `endif
```

Definition of term "blocking statement" (Mantis 225)

- “block” could be used as a noun to describe a set of connected conceptual elements, or as a verb, a construct that suspends execution of a process.
- Clarified the three terms, “blocking statement”, “non-blocking assignment”, and “blocking assignment”

```
   A <= #1 2; //  non-blocking assignment, not a blocking statement
   A = 3;     //  blocking assignment, not a blocking statement
   A = #4 5;  //  blocking assignment, a blocking statement
```

Packed array shortcut – (Mantis 325)

- Started out as an enhancement or clarification request

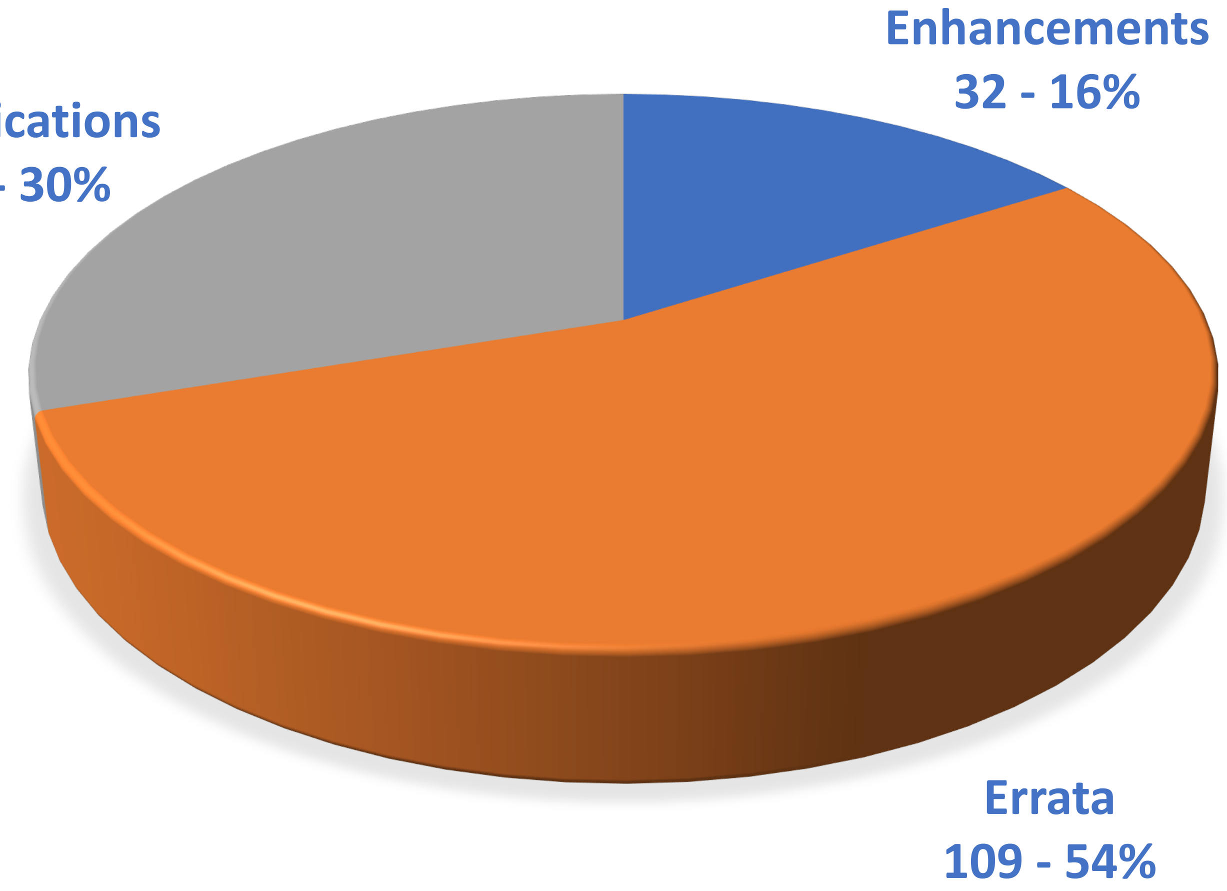
```
bit [7:0] v = 8'b10000000; // 8-bit wide vector, v[7] = 1, 2^7 = 128
int A[0:99];
int A[100]; // This shortcut implies that the left index is 0
```

```
bit [8] v = 8'hab;     // Proposed shorthand for bit [0:7] v = 8'hab;
                       // or should it be       bit [7:0] ???
```

```
int is shortcut for bit signed [7:0]
{expression}[0] is LSB of expression
```

- Now explicitly illegal for packed arrays – not worth the confusion

Work Completed to Date



Closed 180 issues not requiring any action

Thank You

- Questions?