



SystemVerilog 3.1a
Language Developments for Design

Matt Maidment
Intel Corporation



Overview

- SystemVerilog 3.1a Clean-up
- Next Major Milestones
- Primary New Features
 - Extension of memory system tasks
 - Operator Overloading
- BlueSpec Donation

3.1 Clean-up

- Very Busy Group
 - 141 issues filed over wide range of LRM
- 18 BNF Issues Resolved
- Enhanced Features
 - e.g \$readmem{b,h}
- Clarified Features
 - interfaces
 - enumerated types
 - unique/priority

Status	Count
Proposal	12
Committed	5
Open	35
Pass	23
Update	18
Deferred	3
Closed	45
Total	141

Next Major Milestones

- 24 December 2003 – End of LRM development, start of LRM review
- 24 January 2004 – Send to board
- 24 February 2004 – Release 3.1a LRM

- Due to tight schedule and the number of remaining issues, group will accelerate its meeting schedule

Primary New Features

- Extension of memory system tasks
- Operator Overloading
- Separate Compilation
 - Will be covered by David Smith

Extension of Memory System Tasks

- \$readmemh, \$readmemb, \$writememh, \$writememb extended to include SV data types
 - unpacked arrays of any packed data
 - associative arrays of packed data
 - dynamic arrays of packed data
 - 2-value types
- Input/output file formats extended accordingly

Extension of Memory System Tasks

- Syntax

```
$readmemb ("file_name",memory_name[,start_addr[,finish_addr]]);  
$readmemh ("file_name",memory_name[,start_addr[,finish_addr]]);  
$writememb("file_name",memory_name[,start_addr[,finish_addr]]);  
$writememh("file_name",memory_name[,start_addr[,finish_addr]]);
```

- memory_name can be sliced according to SV rules (right-most dimension)

```
$readmemb("file",mem[0][1][5:8]);
```

- start_addr, finish_addr apply to highest dimension of data in the file

```
$readmemb("file",mem[0][1],5,8);
```

Extension of Memory System Tasks

- File Format

- Organized in row-major order, compatible with Verilog-2001
- Right-most dimension varies most rapidly

```
reg [31:0] mem[0:2][0:4][5:8];
```

W Z Y X

zyx	zyx	zyx	zyx
w005	w006	w007	w008
w015	w016	w017	w018
w025	w026	w027	w028
w035	w036	w037	w038
w045	w046	w047	w048
w105	w106	w107	w108
w115	w116	w117	w118
w125	w126	w127	w128
w135	w136	w137	w138
w145	w146	w147	w148
w205	w206	w207	w208
w215	w216	w217	w218
w225	w226	w227	w228
w235	w236	w237	w238
w245	w246	w247	w248

Extension of Memory System Tasks

- File Format

- Organized in row-major order, compatible with Verilog-2001
- Right-most dimension varies most rapidly
- May use address for left-most dimension

```
reg [31:0] mem[0:2][0:4][5:8];
```

W **Z** **y** **x**

	zyx	zyx	zyx	zyx
@0	w005	w006	w007	w008
	w015	w016	w017	w018
	w025	w026	w027	w028
	w035	w036	w037	w038
	w045	w046	w047	w048
@1	w105	w106	w107	w108
	w115	w116	w117	w118
	w125	w126	w127	w128
	w135	w136	w137	w138
	w145	w146	w147	w148
@2	w205	w206	w207	w208
	w215	w216	w217	w218
	w225	w226	w227	w228
	w235	w236	w237	w238
	w245	w246	w247	w248

Operator Overloading

- Enable use of simple operators with Complex SV Types

```
struct3 = struct1 + struct2
```

- Operator Overloading is allowed for type combinations not already defined by SV
- Syntax

```
bind overload_operator
```

```
function data_type function_identifier  
    (overload_proto_formals)
```

Operator Overloading Example

```
typedef struct {
    bit sign;
    bit [3:0] exponent;
    bit [10:0] mantissa;
} float;

bind + function float faddif(int, float);
bind + function float faddfi(float, int);
bind + function float faddrf(real, float);
bind + function float faddrf(shortreal, float);
bind + function float faddfr(float, real);
bind + function float faddfr(float, shortreal);
bind + function float faddff(float, float);

float A, B, C, D;
assign A = B + C; //equivalent to A = faddff(B, C);
assign D = A + 1.0; //equivalent to A = faddfr(A, 1.0);
```

BlueSpec Donation

- BlueSpec donation is based on mathematical semantics to advance Behavioral Synthesis:
 - Originated by research activities in MIT
- Donation is composed of two main parts:
 - Tagged Unions
 - Pattern Matching
- Under-going second revision since donation

Tagged Unions

- Motivation:
 - Provide type-safety and brevity
- Benefits:
 - Improves correctness
 - Improves ability to reason about programs for formal verification

Example:

```
typedef tagged union {
    struct {
        bit [4:0] reg1, reg2, regd;
    } Add;
    tagged union {
        bit [9:0] JmpU;
        struct {
            bit [1:0] cc,
            bit [9:0] addr;
        } JmpC;
    } Jmp;
} Instr;
```

Pattern Matching

- Context: case statements, if statements and conditional expressions
- Benefits:
 - Improve brevity and readability
 - Complete type-safety
 - Makes it easier to implement tagged unions without runtime checks
 - Improves formal reasoning

Pattern Matching Example

```
Instr instr;  
...  
casep (instr)  
  tagged Add {r1,r2,rd}: rf[rd] = rf[r1] + rf[r2];  
  tagged Jmp j: casep (j)  
    tagged JmpU a : pc = pc + a;  
    tagged JmpC {c,a}: if (rf[c]) pc = a;  
  endcase  
endcase
```

Q&A