Digital Techniques 2

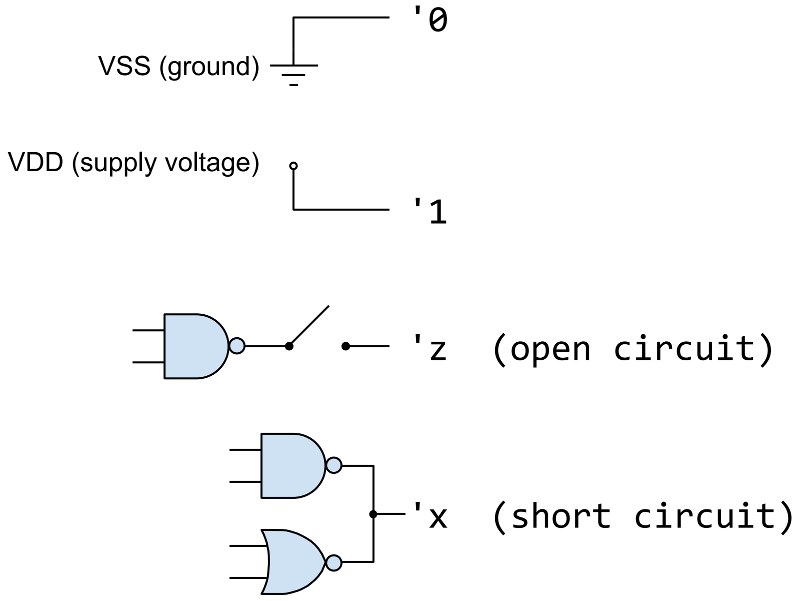
L1: SystemVerilog Data Types and Operators

UNIVERSITY OF OULU
CIRCUITS and SYSTEMS

logic Data Type

- In SystemVerilog, you can almost always use the **logic** data type to represent 1-bit data
- **logic** value set:
  '0 (logical 0)
  '1 (logical 1)
  'z (high-impedance)
  'x (unknown)
- Examples:
  logic my_variable;
  input logic my_input_port:

Examples of data type interpretations:



Packed Arrays (1)

- Bit vectors are represented as **packed arrays** in SystemVerilog
- Only bit types allowed (logic)
- Dimensions as a range [left_bit : right_bit] before the variable name

```
logic [7:0] p1, p2;
p1 = 8'b01011010;   // Set 8-bit binary value
p2 = 8'hA0;         // Set 8-bit hex value
```

- Ranges can be accessed as vectors

```
x = p1[3:2];        // x == 2'b10
y = { p1[3:0], p1[7:4] }; // y == 8'b10100101
```

- Works like unsigned integer by default

```
z = p1 + p2;        // z == 8'b11111010
```

Bit-vector operations

Concatenation:

```
logic [1:0] a = 2'b01, b = 2'b10;
logic [3:0] c;
c = { a, b }; // c = 4'b0110:
```

Splitting:

```
logic [3:0] c = 4'b0110:
logic [1:0] a, b;
{ a, b } = c; // a = 2'b01, b = 2'b10
```

Replication :

```
logic [1:0] a = 2'b01;
logic [7:0] c;
c = { 4{a} }; // a = 8'b01010101
```

Packed Arrays (2)

- Packed arrays are guaranteed to be represented as a contiguous set of bits → can be used to divide a bit-vector into fields that can be accessed using indices

- Assigning a bit literal to a packed array sets all bits, which is a handy way to initialize large arrays:

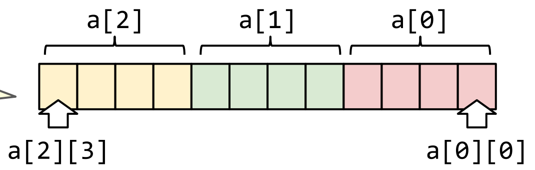
```
logic [7:0] p3;
p3 = '1;   // p3 == 8'b11111111
logic [7:0][7:0] mem = '0; // Sets all
                           // 64 to '0!
```

- Notice the difference:

```
p3 = 1;    // p3 == 8'b00000001
           // (1 = integer value!)
```

```
logic [2:0][3:0] a; // 2-dimensional
```

Conceptually an array that contains three 4-bit elements.



```
logic [11:0] b = 12'b111100001010;
a = b; // a and b have equal number of bits
$display("%b", a[2]) ; // Prints out 1111
```

Notice! SystemVerilog also has conventional arrays, such as:

```
logic x[8];
int  y[1024][32];
```

but these **unpacked arrays** do not have the "hardware-friendly" properties of packed arrays. Use them in testbenches only.

UNIVERSITY OF OULU
CIRCUITS and SYSTEMS

logic Vectors as Unsigned and Signed Numbers

- **logic vectors** work like unsigned integers:

```
logic [7:0] A, B;
logic [8:0] C;
A = 1;      // 00000001
B = -1;     // 11111111 (-1 in 2's compl.)
C = A + B; // 100000000
```

- You can define a **logic** vector to be **signed**:

```
logic signed [7:0] A, B;
logic signed [8:0] C;
A = 1;      //  00000001
B = -1;     //  11111111
C = A + B; // 000000000
```

- You can also use **$signed** and **$unsigned** system functions to change the interpretation:

```
logic [7:0] A, B;
logic [8:0] C;
A = 1;  // 00000001
B = -1; // 11111111
C = $signed(A) + $signed(B);
        // C is now 000000000
```

Enumerated Types

- An enumerated type declares a set of integral named constants

```
enum { RED, GREEN, BLUE, WHITE } my_color; // default coding: RED = 0, GREEN = 1, ...
my_color = GREEN; // Easier to understand than 1
```

- Default base type of enums is **int** (32-bits!), but you can choose a better type and also specify your own constant value encoding:

```
enum logic [1:0] { RED = 2'b00, GREEN = 2'b01, BLUE = 2'b10, WHITE = 2'b11 } my_color;
```

- If you plan to use an *enum* type in many places, define it as a new type:

```
typedef enum logic [1:0] { RED = 2'b00, GREEN = 2'b01,
                           BLUE = 2'b10, WHITE = 2'b11 } my_color_t;
my_color_t my_color; // Variable whose type is my_color_t
my_color_t my_other_color;
```

Packages

- You can create a package to hold your design-wide definitions:

```
package my_package;
  typedef enum logic [1:0] { RED = 2'b00, GREEN = 2'b01,
                             BLUE = 2'b10, WHITE = 2'b11 } my_color_t;
  localparam NBITS = 7:
enpackage
```

- In your design, import the package and use it:

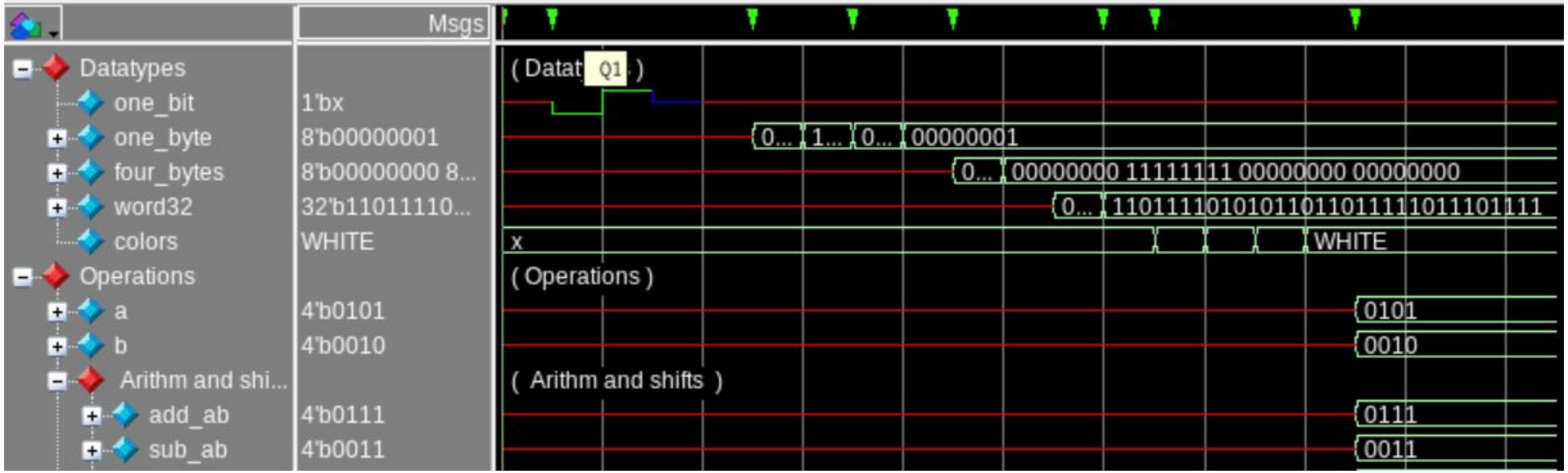
```
import my_package::*;  // Make everything defined in the package visible here
my_color_t my_colors;
logic [NBITS-1:0] my_bitvector;
```

SystemVerilog Operators

binary assignment operator	=							
binary arithmetic assignment operators	+=	-=	/=	*=				
binary arithmetic modulus assignment	%=							
binary bitwise assignment operators	&=	|=	^=					
binary logical shift assignment operators	>>=	<<=						
binary arithmetic shift assignment operators	>>>=	<<<=						
conditional operator	?:							
unary arithmetic operators	+	-						
unary logical negation operator	!							
unary logical reduction operators	~	&	~&	|	~|	^	~^	^~
binary arithmetic operators	+	-	*	/	**			
binary arithmetic modulus operator	%							
binary bitwise operators	&	|	^	^~	~^			
binary logical shift operators	>>	<<						
binary arithmetic shift operators	>>>	<<<						
binary logical operators	&&	||	->	<->				
binary relational operators	<	<=	>	>=				
binary case equality operators	===	!==						
binary logical equality operators	==	!=						
binary wildcard equality operators	==?	!=?						
unary increment, decrement operators	++	--						
binary set membership operator	inside							
binary distribution operator	dist							
concatenation, replication operators	{}	{{}}						
stream operators	{<<{}}	{>>{}}						

Lab 2: Data Types

- Use the simulator to you find the answers to the questions presented in the lab workbook



```
Datatypes                     ( Datat Q1 )
  one_bit       1'bx
  one_byte      8'b00000001        0... 1... 0... 00000001
  four_bytes    8'b00000000 8...   0... 00000000 11111111 00000000 00000000
  word32        32'b11011110...    0... 11011111010101101101111011101111
  colors        WHITE              x                      WHITE
Operations                    ( Operations )
  a             4'b0101            0101
  b             4'b0010            0010
  Arithm and shi...           ( Arithm and shifts )
    add_ab      4'b0111            0111
    sub_ab      4'b0011            0011
```

References

1. IEEE Standard for SystemVerilog
   - Ch 6 Data Types (logic type)
   - Ch 7.4 Packed and unpacked arrays
   - Ch 11.1 - 11.4 Operators