**Declaring Vectors**

Vectors must be declared:

type [upper:lower] vector\_name;

type specifies the datatype of the vector. This is usually wire or reg. If you are declaring a input or output port, the type can additionally include the port type (e.g., input or output) as well. Some examples:

wire [7:0] w; // 8-bit wire

reg [4:1] x; // 4-bit reg

output reg [0:0] y; // 1-bit reg that is also an output port (this is still a vector)

input wire [3:-2] z; // 6-bit wire input (negative ranges are allowed)

output [3:0] a; // 4-bit output wire. Type is 'wire' unless specified otherwise.

wire [0:7] b; // 8-bit wire where b[0] is the most-significant bit.

The *endianness* (or, informally, "direction") of a vector is whether the the least significant bit has a lower index (little-endian, e.g., [3:0]) or a higher index (big-endian, e.g., [0:3]). In Verilog, once a vector is declared with a particular endianness, it must always be used the same way. e.g., writing vec[0:3] when vec is declared wire [3:0] vec; is illegal. Being consistent with endianness is good practice, as weird bugs occur if vectors of different endianness are assigned or used together.

**Implicit nets**

Implicit nets are often a source of hard-to-detect bugs. In Verilog, net-type signals can be implicitly created by an assign statement or by attaching something undeclared to a module port. Implicit nets are always one-bit wires and causes bugs if you had intended to use a vector. Disabling creation of implicit nets can be done using the `default\_nettype none directive.

wire [2:0] a, c; // Two vectors assign a = 3'b101; // a = 101 assign b = a; // b = 1 implicitly-created wire assign c = b; // c = 001 <-- bug my\_module i1 (d,e); // d and e are implicitly one-bit wide if not declared.  
// This could be a bug if the port was intended to be a vector.

Adding `default\_nettype none would make the second line of code an error, which makes the bug more visible.

**Unpacked vs. Packed Arrays**

You may have noticed that in *declarations*, the vector indices are written *before* the vector name. This declares the "packed" dimensions of the array, where the bits are "packed" together into a blob (this is relevant in a simulator, but not in hardware). The *unpacked* dimensions are declared *after* the name. They are generally used to declare memory arrays. Since ECE253 didn't cover memory arrays, we have not used packed arrays in this course. See <http://www.asic-world.com/systemverilog/data_types10.html> for more details.

reg [7:0] mem [255:0]; // 256 unpacked elements, each of which is a 8-bit packed vector of reg.

reg mem2 [28:0]; // 29 unpacked elements, each of which is a 1-bit reg.

**Accessing Vector Elements: Part-Select**

Accessing an entire vector is done using the vector name. For example:

assign w = a;

takes the entire 4-bit vector *a* and assigns it to the entire 8-bit vector *w* (declarations are taken from above). If the lengths of the right and left sides don't match, it is zero-extended or truncated as appropriate.

The part-select operator can be used to access a portion of a vector:

w[3:0] // Only the lower 4 bits of w

x[1] // The lowest bit of x

x[1:1] // ...also the lowest bit of x

z[-1:-2] // Two lowest bits of z

b[3:0] // Illegal. Vector part-select must match the direction of the declaration.

b[0:3] // The \*upper\* 4 bits of b.

assign w[3:0] = b[0:3]; // Assign upper 4 bits of b to lower 4 bits of w. w[3]=b[0], w[2]=b[1], etc.