Enumeration Proposals by Cliff Cummings

Proposed change to BNF section A.2.2.1 - data_type

```
data_type ::=
        integer_vector_type [ signing ] { packed_dimension } [ range ]
      | integer_atom_type [ signing ] { packed_dimension }
      | type_declaration_identifier
      | non_integer_type
      | struct { { struct_union_member } }
      | union { { struct_union_member } }
      | enum [enum_type] { enum_identifier [ = constant_expression ]
          { , enum_identifier [ = constant_expression ] } }
      | void

enum_type ::=
        integer
      | logic [ range ]
      | reg [ range ]
      | <what else??>
```

PROPOSAL: Add the following paragraphs to the beginning of section 3.6, page 8 of draft 5 as shown:

Enumerated data types provide the capability to abstractly declare strongly typed variables without either a data type or data value(s) and later add the required data type and value(s) for designs that require more definition. Enumerated data types also can be easily referenced or displayed using the enumerated names as opposed to the enumerated values.

In the absence of a data type declaration and if no values are assigned, the default data type shall be **`int`**. If only binary values are assigned to the **`enum`** names, the default data type shall be **`int`**, If no data type is assigned and the enumerated names have assigned values that include **`x`**'s and **`z`**'s, the default data type shall be **`logic`**. This means that an enumeration that defaults to a data type of **`int`** because the **`enum`** names were either unassigned or were originally assigned binary values will default to a data type of **`logic`**, after the design is recompiled, if **`x`**-assignments or **`z`**-assignments are added to the enumeration when the design file is modified. Any other data type used with enumerations requires an explicit data type declaration.

An unassigned **`enum`** name that follows an **`enum`** name with **`x`** or **`z`** assignments shall be a syntax error.

```
// Syntax error: IDLE=2'b00, XX=2'bx, S1=??, S2=??
enum {IDLE, XX='x, S1, S2} state, next;
```

Draft 5 - page 8 - Section 3.6 - Enumerations

Proposal: Expand the first paragraph as follows:

An enumerated type has one of a set of named values. In the following example, "light1" and "light2" are defined to be variables of the anonymous (unnamed) enumerated type that includes the three members: "red", "yellow" and "green."

Draft 5 - page 8 - Section 3.6 - Enumerations

Proposal: Add the following clarification paragraphs and examples after the second enum example on page 8.

The values can be set for some of the names and not set for other names. A name without a value is automitically assigned an increment of the value of the previous name.

```
// c is automatically assigned the increment-value of 8
enum (a=3, b=7, c) alphabet;
```

If an automatically incremented value is assigned elsewhere in the same enumeration, this shall be a syntax error.

```
// Syntax error: c and d are both assigned 8
enum (a=0, b=7, c, d=8) alphabet;
```

If the first name is not assigned a value, it is given the initial value of 0.

```
// a=0, b=7, c=8
enum (a, b=7, c) alphabet;
```

Draft 5 - page 9 - Section 3.6 - Enumerations

Proposal: for the third example, change the comment to:

```
// silver=4'h4, gold=4'h5 (all are 4 bits wide)
enum {bronze=4'h3, silver, gold} medal4;
```

Draft 5 - page 9 - Section 3.6 - Enumerations

Proposed wording: Add the following before the paragraph starting with "The type name can be given ...," just before the typedef example.

Adding a constant range to the enum declaration can be used to set the size of the type. If any of the enum members are defined with a different sized constant, this shall be a syntax error.

```
// Error in the bronze and gold member declarations
enum [3:0] {bronze=5'h13, silver, gold=3'h5} medal4;

// Correct declaration - bronze and gold sizes are redundant
enum [3:0] {bronze=4'h13, silver, gold=4'h5} medal4;
```