Section 3.7 String data type

There are several issues that were recently raised in a series of emails on the SV-EC reflector with this section of the LRM. This errata is intended to capture those issues. I have attempted to gather the discussion for each issue together. My apologies to anyone that I have misquoted since I did take the liberty of slightly rewording some of the comments after taking them out of the original context.

There are a couple of other errata that are related to this one.
197 - Is a String an array
271 - Incorrect example of assigning a string to a register in Section 3.7

1. The Leftmost character of a string has index 0 - [Re: Neil]

   This is not mentioned in the LRM even though it appears that this was the intent.

   The string data type is based upon the C++ Standard Template Library implementation of its string class. In the STL implementation the leftmost character has index 0. This is consistent with how C-strings are used.

This is mentioned in the 4th paragraph of section 3.7, but it should be made more explicit. Also the type of a character in the string must be made explicit since this is needed to pass by reference an element of a string variable.

Proposal

Change Section 3.7, Page 12 (1st paragrah)

(incorporates changes due to previously approved errata 197 and 203)

SystemVerilog includes a string data type, which is an ordered collection of characters. The length of a string variable is the number of characters in the collection. Variables of type string are dynamic as their length may vary during simulation. A single character of a string variable may be ~~read~~selected for reading or writing ~~from a string variable~~ by indexing the variable ~~from 0 to N-1, where N is the length of the variable~~. ~~Reading a~~ A single character of a string variable ~~yields~~is of type byte. SystemVerilog also includes a number of special methods to work with strings.

Change Section 3.7, Page 12 (4th paragrah)

(this paragraph had been deleted by erratum 197 and the modified by erratum 334)

FROM

~~Variables of type **`string`** can be indexed from 0 to N-1 (the last element of the array), and they can take on the special value “”, which is the empty string. Reading an element of a string yields a byte.~~

TO

The indices of **`string`** variables shall be numbered from 0 to N-1 (where N is the length of the string), such that index 0 corresponds to the first (leftmost) character of the string, and index N-1 corresponds to the last (rightmost) character of the string. String variables can take on the special value “”, which is the empty string. Indexing an empty string variable shall be an out-of-bounds access.

A string shall not contain the special character “\0”. Assigning the value 0 to a string character shall be ignored.

2. putc taking the first character of a string is inconsistent with assignments

3.7.2 putc() says that

> str.putc(i,s) replaces the i'th character in str with the first character in s.

Does str[j] = x actually take the LAST, not FIRST, character if x is a string, as in the following example. [Re: Shalom]

```
bit[1:4][7:0] h = "hello"; //assigns to h "ello"
```

It appears that the reason that the 'h' is dropped is due to the following paragraph in section 3.7 that describes truncation rules when the lengths of strings are unequal. [Re: Neil]

> "A string literal can be assigned to a string or an integral type. If their size differs the literal is right justified and either truncated on the left or zero filled on the left, as necessary."

This characteristic, that under certain conditions, you take the first characters of a string, and in other cases, you take the last characters, is confusing and not well explained. I've been unable to figure out the rules clearly. [Re: Shalom]

The from of the putc method that accepts a string is confusing and unnecessary. It was originally defined in order to allow putc to accept a string literal, for example, str.putc(3, "A"). Now the type of the string characters is defined to be of type "byte", it is no longer necessary. The standard conversion rules to the type "byte" completely govern the outcome of the operation. If the argument is an integral value, the rightmost 8-bits are used. A string variable cannot be used directly as the 2nd argument; instead, an indexed string variable or a cast must be used.

Proposal(incorporates the changes from approved erratum 334)

Change Section 3.7.2, Page 14

3.7.2 putc()

~~`task putc(int i, string s)`~~

`task putc(int i, byte c)`

— `str.putc(i, c)` replaces the *i*th character in *str* with the given integral value.

~~— `str.putc(i, s)` replaces the *i*th character in *str* with the first character in *s*.~~

~~— *s* can be any expression that can be assigned to a string.~~

— `putc` does not change the size of `str`: If `i` < 0 or `i` >= `str.len()`, then *str* is unchanged.

— If the second argument to `putc` is zero, the string is unaffected: if `c` is 0, then *str* is unchanged.

Note: `str.putc(j, x)` is semantically equivalent to `str[j] = x`.

3. Assigning a string variable to a byte select of a string variable

Questions about the "Note:" in section 3.7.2
Which of the putc() calls does the note apply to?

```
task putc(int i, string s)
task putc(int i, byte c)

Note: str.putc(j,x) is semantically equivalent to str[j] = x.
```

Then there was a question as to whether the statement within the note should be allowed at all.

Below is feedback from Surrendra on this point.

```
string x = "abc";
str[2] = "abc"; (1)
str[2] = x; (2) // should this be illegal? [Re: Surrendra]
```

str[2] is a character which in SV is a byte.
Statement (1) is a Verilog statement, so it must follow Verilog semantics.
Statement (2) should be illegal as an SV string variable cannot be assigned directly to a byte, logic, etc. [Re: Surrendra]

The note that Shalom pointed out is wrong if x is a string wider than 1 character. [Re: Steven Sharp]

Is str.putc(j, x) semantically equivalent to str[j] = x?

I think yes, only if x is an integral value.
Otherwise str.putc(j, x)is semantically equivalent to str[j] = x[0] if x is a string. [Re: Dave Rich]

If the proposal to issue 1 and 2 is accepted then the note does apply to all types of arguments and this is a non-issue.

str.putc(j, x) is semantically equivalent to str[j] = x

is true regardless of the type of x:
- string - illegal
- string literal - uses the rightmost character
- integral value - uses the rightmost 8-bits after applying standard extension/truncation

4. The rules of assignment are not explicitly specified in this section

Erratum 334, which was approved, defines specific assignment and casting rules.

This is no longer an issue.

5. Typo in example - [Re: Neil]

The following example is shown right before table 3-2 in section 3.7

```
a[0]="h"; // OK same as a[0]="hi" ) <--- ) should be ;
```

This example is showing a string being indexed. Section 3.7, doesn't seem to say anything about writing to an index of a string. There is only this one example to say it is legal, and there is no description of its meaning in the text. [Re: Steven Sharp]

The example should be modified to be consistent with issue 2,

Proposal (incorporates changes from previously approved errata 203 and 334)

Change Section 3.7, Pages 12-13

For example:

```
string s1 = "hello";          // sets s1 to "hello"
bit [11:0] b = 12'ha41;
~~string s2 = b;~~                // ~~sets s2 to 'h0a41~~
string s2 = string'(b);       // sets s2 to 16'h0a41
```

As a second example:

```
typedef reg [15:0] r_t;
r_t  r;
integer i = 1;
string b = "";
string a = {"Hi", b};
r = ~~a~~ r_t'(a);          // OK
b = ~~r~~ string'(r)        // OK ~~(implicit cast, implementations can issue a warning)~~
b = "Hi";                   // OK
b = {5{"Hi"}};              // OK
a = {i{"Hi"}};              // OK (non constant replication)
r = {i{"Hi"}};              // invalid (non constant replication)
a = {i{b}};                 // OK
a = {a,b};                  // OK
a = {"Hi",b};               // OK
r = {"H",""};               // yields "H\0" => "" is converted to 8'b0
b = {"H",""};               // yields "H" => "" is the empty string
a[0] = "h";                 // OK same as a[0] = "cough~~hi~~" ;~~)~~
a[0] = b;                   // invalid, requires a cast
a[1] = "\0";                // ignored, a is unchanged
```

6. Writing to an index of a string

The following example is shown right before table 3-2 in section 3.7

```
a[0]="h"; // OK same as a[0]="hi";
```

This example is showing a string being indexed. Section 3.7, doesn't seem to say anything about writing to an index of a string. There is only this one example to say it is legal, and there is no description of its meaning in the text. [Re: Steven Sharp]

String indexing is defined in issue (1).

7. Assigning a string literal to a byte select of a string variable

   The following example is shown right before table 3-2 in section 3.7

```
   a[0]="h"; // OK same as a[0]="hi";
```

   Why is this being defined inconsistently with other assignments in the language? [Re: Steven Sharp]

   The text about reading from an index in a string says that the result is a byte. If we assume that it is also a byte when written, then an assignment to it should act like any other assignment to an integral type. The rightmost bits would be kept.

   This comment saying a[0] = "h" is the same as a[0] = "hi" is simply wrong. It should be fixed. [Re: Steven Sharp]

   'a[0] = "hi";' should be the same as 'a[0] = "i";' Since a[0] is an integral byte type, and integral type rules apply, therefore the left-most 8 bits of "hi" are truncated. [Re: Dave Rich]

This is addressed by the proposal to issue (5).

8. Usage of the word string

   By the way, it is also a Bad Practice to refer to a string variable or any other specific type of string as simply "string" with no modifier. The unmodified word "string" should be reserved for situations where it is equally applicable to all types of strings. [Re: Shalom]