*svlib* User Guide  
and Programmer's Reference

# About this document

## Summary

This document provides a specification and programmer's reference for the SystemVerilog utility library *svlib*.

## Revision information

|  |  |  |  |
| --- | --- | --- | --- |
| Rev | Date | Author | Description |
| 0.0 | 10 Feb 2014 | J Bromley | Initial release for discussion |
|  |  |  |  |

## Scraps of useful formatting

###### A/D result high byte (offset 0xB)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | MSB |  |  |  |  |  |  | LSB |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | Most significant bits of ADC conversion result (bits [11:4]) | | | | | | | |

# A few notes about general principles of use

svlib has been designed to be as un-selfish and un-intrusive as possible for use in any SystemVerilog environment. To achieve these goals it was necessary to introduce some underlying behaviours that are common to the whole library. It is important for svlib users to be aware of these behaviours to avoid unpleasant surprises.

## Constructing svlib objects

Many parts of svlib use SystemVerilog classes. User code must of course create new objects of these class types in order to make use of svlib features. However, in order to avoid unexpected disturbance to random stability and to improve memory management efficiency,

**it is very important that user code should *never* directly call the constructor, new, of any svlib class.** All svlib objects should be created using their built-in static create method, which is documented individually for each class.

This issue is discussed in more detail in the accompanying conference paper .

For SystemVerilog tools that support it, the class constructors are declared protected so that they cannot be called from user code. However, some tools do not yet support this feature, and if you are using a tool with this limitation, it will be possible for your code to call the constructor directly. Take great care to avoid doing this, as it will make your code non-portable and will compromise both performance and functionality of svlib.

## Handling errors

# String manipulation

The SystemVerilog language provides a number of string operations natively. However, experience has shown that the built-in set is not sufficient for many practical string processing tasks, and svlib provides a further set of operations to help meet these requirements.

String operations are, in most cases, available in two different forms, and a programmer is free to choose whichever form is more appropriate to their needs.

* The first form is straightforward functions on string values, often (but not always) returning a string result. These functions are defined in the svlib package and consistently have names that begin with the prefix str\_.
* The second form is methods of an object of class Str (note the uppercase S). The Str class is a wrapper for a SystemVerilog string, allowing a string to be passed around by reference and making some sequences of operations more convenient.

The obvious drawback to using Str objects rather than simple functions is that an object must be constructed before any operation is performed. This drawback is often outweighed by the convenience of being able to offer a string object by reference to many different successive operations. As already noted, programmers are free to choose the representation that is most convenient for them.

# Regular expression processing

svlib supports regular expression matching and substitution within strings.

After a regular expression match has succeeded, there are many different things that a user might wish to do with the results. To support this variety of needs, regular expressions in svlib are invariably represented as an object of class Regex.

Often, you need to apply the same expression multiple times to a given string – typically because you want to locate not just the first, but every occurrence of a match within the string. To make this more efficient and convenient, regular expression matching works not on a native SystemVerilog string, but on a svlib Str object (see section \*\*\*).

The basic steps in performing a regular expression match are:

* Construct a Regex object and set it up to contain your chosen regular expression, together with options such as case sensitivity and end-of-line handling.
* Construct a Str object containing the string that you want to search in.
* Call the test() method of the Regex object to perform the match, returning information about whether the match succeeded (found a match) or failed.
* Call other methods of the Regex object to retrieve more detailed results such as matches corresponding to parenthesized groups, or to perform substitution operations.

However, some convenience functions exist to simplify some of these steps in situations where only standard matching operations are required.

First we describe the more flexible approach in which objects are created explicitly. Later in this section we cover the

## Retrieving matches and sub-matches

After using a Regex object to perform a successful match, you can call methods of the object to get information about the various matches and sub-matches that were found by the match attempt.

function string getMatchString(int m);

function int getMatchStart (int m);

function int getMatchLength(int m);

These methods extract and return the match specified by the value m. A value of zero indicates the string that corresponds to the whole regular expression match. Values between 1 and 9 correspond to strings that matched sub-expression groups in the regular expression, numbered in left-to-right order of their opening left parenthesis in the usual way.

* getMatchString returns the matching string itself (a slice of the original string).
* getMatchStart returns the left-most character position of the match.
* getMatchLength returns the number of characters in the match.

If you call any of these functions on a Regex object whose most recent match was unsuccessful, or if you supply a value of m that is larger than the number of sub-matches in the original regular expression, then there will be no error, but:

* getMatchString returns an empty string.
* getMatchStart returns -1.
* getMatchLength returns zero.

getMatchString(m) is always exactly equivalent to calling the range method on the Str object containing the string that was searched:

range(getMatchStart(m), getMatchLength(m))