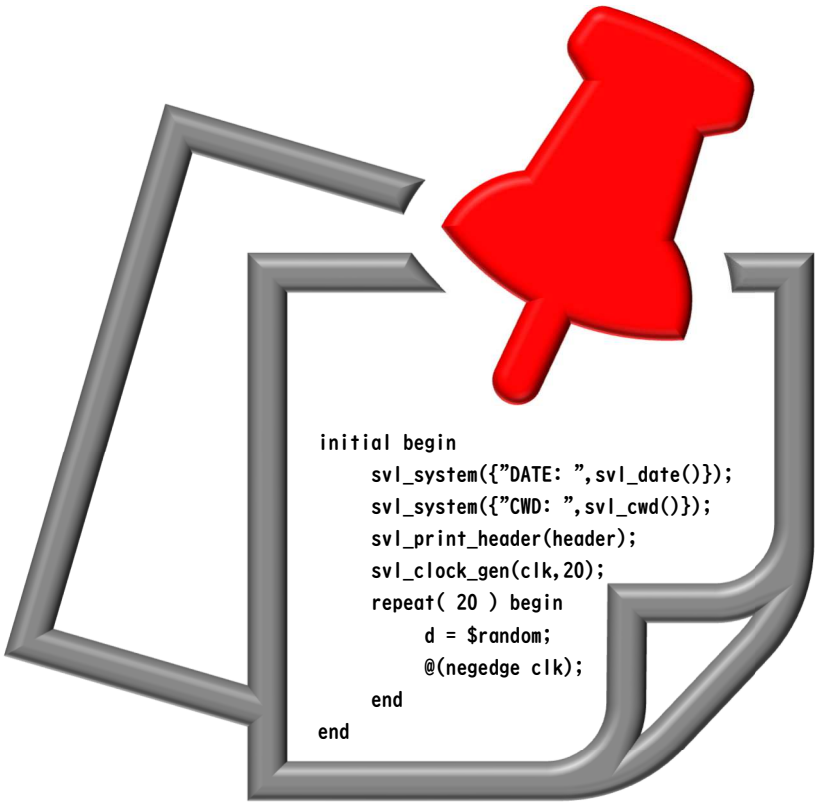
Artgraphics

SVL Standard

検証のための SystemVerilog ライブラリー

生産性向上のための SystemVerilog パッケージ

Document Identification Number: ARTG-TD-005-2024
Document Revision: 1.0, 2024.11.25
アートグラフィックス



SVL Standard
検証のための SystemVerilog ライブラリー

〒124-0012 東京都葛飾区立石 8-14-1
www.artgraphics.co.jp

SystemVerilog Library (SVL) for Verification

8-14-1, Tateishi, Katsushika-ku, Tokyo, 124-0012 Japan
www.artgraphics.co.jp

弊社製品、および、業務案内に関するご質問には下記の連絡先をご利用下さい。
連絡先：contact.us@artgraphics.co.jp

はじめに

本書は、検証のための SystemVerilog ライブラリー（SVL）の解説書です。SVL は、SystemVerilog による検証作業で必要となる基本機能を含む SystemVerilog パッケージです。SVL には SVL Standard と SVL Premium の二種類があります。SVL Standard は検証作業で必要となる基本機能で構成され、SVL Premium は基本機能に加えて、検証環境構築時に必要となる機能で構成されています。本書は SVL Standard の解説書です。

SystemVerilog は豊富な機能を備えていますが、プログラミングに適した機能を備えているわけではありません。例えば、十進数を 123,456 のようにプリントする機能はありません。しかし、検証結果を見易く記録するためには、そのような機能が必要になる時があります。同様に、SystemVerilog には 16 進数を `32'h010a_ff34` のようにプリントする機能はありませんが、検証結果を見易くするためには必要な機能です。これらの一例が示すように、SystemVerilog には検証で必要とされる汎用的なプログラミング機能が不足しています。SVL はその欠乏した機能を補完する役目を持っています。

SVL ではマクロが使用されてプリント書式の自動生成が行われています。その結果、マクロにより汎用的なプログラミング記述が可能となっています。以下に、簡単なマクロ使用例を紹介します。

記述法	生成された書式の実行結果例
`` `svl_sformatfb_m(v) ``	10101011110011010000000100100011
`` `svl_sformatfo_m(v) ``	25363200443
`` `svl_sformatfh_m(v) ``	abcd0123
`` `svl_pbreakupb_m(v) ``	'b1010_1011_1100_1101_0000_0001_0010_0011
`` `svl_pbreakupo_m(v) ``	'o253_6320_0443
`` `svl_pbreakuph_m(v) ``	'habcd_0123
`` `svl_sprintfb_m(v) ``	32'b1010_1011_1100_1101_0000_0001_0010_0011
`` `svl_sprintfo_m(v) ``	32'o253_6320_0443
`` `svl_sprintfh_m(v) ``	32'habcd_0123
`` `svl_breakupd_m(n) ``	123,456,789

このように、使用するマクロを変えるだけで様々なプリント書式を生成する事ができます。これらのマクロはプリント機能のビルディングブロックであり、SVL ではそれらを組み合わせて更に高度なマクロ機能が構築されています。SVL は非常に強力なプリント機能を備えています。その機能は、柔軟性があり、しかも汎用的です。それらの機能を使用する事により、ビット数の変化に影響を受けない汎用的なプリント機能を開発できます。

SVL では、プリントするためのレイアウトを定義し、そのレイアウトをモデルにしてプリントします。したがって、モデルが変化すると自動的にプリント処理も変更されます。例えば、ビット数が変化すると、モデルが更新されるので、プリント処理も自動的に更新されます。以下に簡単なモデル例を紹介します。

```
svl_print_header_s header[] = '{
    '{"time",5,SVL_HEADER_DEFAULT},
    '{"reset",5,SVL_HEADER_DEFAULT},
    '{"d",`svl_full_hex_digits_m(WIDTH),SVL_HEADER_DEFAULT},
    '{"q",`svl_full_hex_digits_m(WIDTH),SVL_HEADER_DEFAULT}
    };
```

この例では DUT を検証する際の信号値に対応してモデルを定義しています。検証結果をプリントする際には、このモデルを指定してプリントします。こうすると、カラム情報や WIDTH が変化しても自動的にカラム調節が行われます。カラムを入れ替えてもプリント処理を変更

する必要はありません。以下は、プリント結果を示しています。

```
最初の仕様（WIDTH==16）          カラム d と q を入れ替えた仕様
=========================  =========================
time  reset d      q       time  reset q      d
-------------------------  -------------------------
@ 10: 0     'hc778 'hc778  @ 10: 0     'hc778 'hc778
@ 30: 0     'hc8d0 'hc8d0  @ 30: 0     'hc8d0 'hc8d0
@ 50: 0     'h1471 'h1471  @ 50: 0     'h1471 'h1471
@ 70: 0     'h2004 'h2004  @ 70: 0     'h2004 'h2004
@ 90: 0     'hea38 'hea38  @ 90: 0     'hea38 'hea38
@100: 1     'h17ef 'h0000  @100: 1     'h0000 'h17ef
@110: 0     'h17ef 'h17ef  @110: 0     'h17ef 'h17ef
@130: 0     'h8562 'h8562  @130: 0     'h8562 'h8562
@150: 0     'hfc26 'hfc26  @150: 0     'hfc26 'hfc26
@170: 0     'h3b02 'h3b02  @170: 0     'h3b02 'h3b02
@190: 0     'h02a2 'h02a2  @190: 0     'h02a2 'h02a2
=========================  =========================
```

カラム名称の左揃え、右揃え、中央揃え等の指定もできます。以下に実行例を紹介します。

```
WIDTH==32 でカラム名称と値を右に揃えた仕様
===================================
time  reset           d           q
-----------------------------------
@ 10:     0 'he093_c778 'he093_c778
@ 30:     0 'h060c_c8d0 'h060c_c8d0
@ 50:     0 'h3154_1471 'h3154_1471
@ 70:     0 'h870a_2004 'h870a_2004
@ 90:     0 'ha9bb_ea38 'ha9bb_ea38
@100:     1 'h700e_17ef 'h0000_0000
@110:     0 'h700e_17ef 'h700e_17ef
@130:     0 'had45_8562 'had45_8562
@150:     0 'haf37_fc26 'haf37_fc26
@170:     0 'h848b_3b02 'h848b_3b02
@190:     0 'hd0cf_02a2 'hd0cf_02a2
===================================
```

検証環境が複雑な検証階層で構成される事は珍しくありませんが、階層の中から特定の検証コンポーネントを検索する際にはワイルドカードによるパターン検索が必要になります。SVL は、その機能を標準的に備えています。以下のように簡単にパターンを指定できます。階層名（pathname）がパターンにマッチすれば if 文の条件は真となります。

```
if( svl_pattern_matching("top.*.driver",pathname) )
```

勿論、SVL にはこの他にも便利な機能が多く備えられています。特に、文字列処理機能が豊富です。また、便利で使い易いファイル入出力機能も備えられています。本書は、SVL が提供する全ての機能を使用例と共に詳しく解説しています。

アートグラフィックス

変更履歴

日付	Revision	変更点
2024.11.25	1.0	初版。

目次

1　概要

SVL は、検証作業で必要となる機能を汎用的に開発した SystemVerilog パッケージです。汎用的であるため、実装に依存する内容はパラメータとして指定される仕組みになっています。したがって、パラメータに割り当てられた値が変化すれば、実装された内容も自動的に変化するように構築されています。例えば、ビット幅が変化しても検証結果のレポート処理を変更する必要はありません。本章では、SVL の概要を解説します。

1.1　SVL の意義と目的

検証環境を構築するためのパッケージとしては UVM が良く知られています。UVM には、検証コンポーネントやトランザクションの定義を容易にする機能が含まれていますが、検証コード記述のための生産性向上技術は含まれていません。SVL は、UVM 等のパッケージに不足している生産性向上技術を提供します。勿論、SVL は UVM とは直接的な関連を持たないため、SVL を単独に生産性向上のためのパッケージとして使用する事ができます。寧ろ、これが SVL の目指す本来の目的です。

先ず、SVL が記述されたコードに柔軟性を持たせる効果から説明を始めます。通常、メッセージをプリントする際には、以下のように`$display` システムタスクを使用します。

```
$display("sample message");
```

このメッセージはターミナルにプリントされると共に、一般的には、シミュレータのログファイルにも記録されます。しかし、ログファイルは一時的な記録ファイルであるため、保存していなければ情報は次回のシミュレーションで消滅してしまいます。SVL では、そのような状況を避けるためのメッセージプリントする機能（`svl_info`、`svl_warning`、`svl_error`、`svl_fatal`、`svl_system` 等）を備えています。

SVL のプリント機能は、指定された状態にしたがい動作するので、プリント内容をユーザ指定のファイルに出力できます。つまり、SVL のプリント機能は、ターミナルモードであれば通常の`$display` のように動作し、ファイルモードであれば`$fdisplay` のように動作します。しかも、メッセージのレベルを指定できるので、重要でないメッセージを抑止する事ができます（図 1-1）。

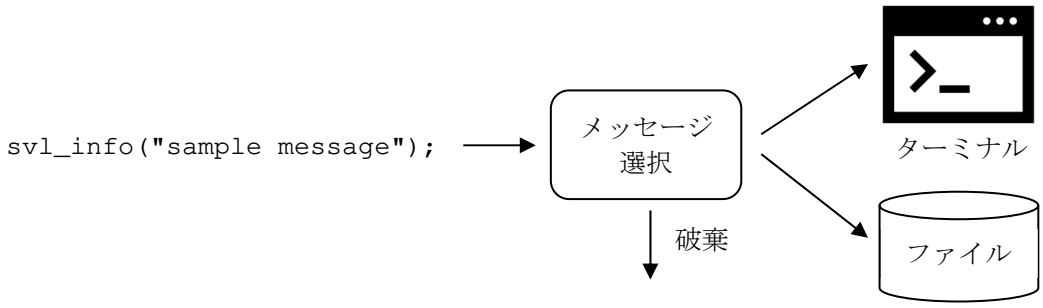


図 1-1　プリント機能の動作

メッセージは、以下のようにメッセージレベル識別子と共にプリントされるので、情報を見分けるのが容易です。

```
SVL_INFO: sample message
```

メッセージレベル識別を以下のように変更する事もできます。

```
#-I: sample message
```

以上は基本的な機能の例でしたが、この他にも重要な基本機能があります。それは、プリント書式機能であり、その多くは SystemVerilog マクロで実現されています。例えば、以下のようにマクロを指定できます。

```
svl_info($sformatf("%s = %s",`svl_nsprintfh_m(a)));
```

こうすると、以下のように見易い書式で変数の値をプリントできます。様々な書式のプリント機能が用意されているので、検証結果の確認が容易になる効果をもたらします。

```
SVL_INFO: a = 32'habcd_ef01
```

プリント書式マクロはベーシックな機能を提供しますが、それらを組み合わせると更に強力な機能となります。以下に、そのような機能の例を紹介します。

SVL では、プリント用のヘッダーモデルを定義し、それに合わせてカラム単位にプリント処理を記述します。そうする事により、汎用性の高いプリント処理を実現できます。利点をまとめると以下のようになります。ここで、カラムはヘッダーの構成単位です。

- カラムの幅の変更はモデルの変更だけで済みます。
- カラムの入れ替えをしてもプリント処理には影響を及ぼしません。
- カラムの挿入や追加・削除は他のカラムに影響を及ぼしません。
- 変数のビット数に変更が発生してもプリント処理を変更する必要はありません。
- 殆どのプリント処理は、同じような形式になるのでテンプレート化し易い。

例えば、以下のようにヘッダーを定義します。

```
svl_print_header_s header[] = '{
    '{"time",5,SVL_HEADER_DEFAULT},
    '{"reset",5,SVL_HEADER_DEFAULT},
    '{"d",`svl_full_hex_digits_m(WIDTH),SVL_HEADER_DEFAULT},
    '{"q",`svl_full_hex_digits_m(WIDTH),SVL_HEADER_DEFAULT}
    };
```

プリント処理は、以下のようにカラム単位に行います。こうすると、プリント処理は、ヘッダーのレイアウトにしたがい動作するので、ヘッダーに変更が起こればプリント処理も自動的に変更されます。

```
function void print();
svl_print_data_s    row[$], column;
    `svl_init_print_row_m(row)
    foreach(header[i]) begin
        case (header[i].m_name)
        "time": column.m_value = $sformatf("@%3t:",$time);
        "reset": column.m_value = `svl_sformatfb_m(reset);
        "d": column.m_value = `svl_pbreakuph_m(d);
        "q": column.m_value = `svl_pbreakuph_m(q);
        endcase
        `svl_add_print_column_m(row,column,header[i])
    end
    `svl_print_row_m(row)
endfunction
```

例えば、WIDTH を 16 から 32 に変更しても、プリント処理には影響がなく、以下のようにプリントされます。同様に、カラムを入れ替えてもプリント処理に影響がありません。

```
WIDTH==16                          WIDTH==32
=========================          ===================================
time  reset d      q               time  reset d           q
-------------------------          -----------------------------------
@ 10: 0     'h9d66 'h9d66          @ 10: 0     'h8f17_9d66 'h8f17_9d66
@ 30: 0     'h9b47 'h9b47          @ 30: 0     'h575e_9b47 'h575e_9b47
@ 50: 0     'h7f20 'h7f20          @ 50: 0     'h0424_7f20 'h0424_7f20
@ 70: 0     'h51a6 'h51a6          @ 70: 0     'hf816_51a6 'hf816_51a6
@ 90: 0     'ha6e2 'ha6e2          @ 90: 0     'h814a_a6e2 'h814a_a6e2
@100: 1     'h4017 'h0000          @100: 1     'h4c12_4017 'h0000_0000
@110: 0     'h4017 'h4017          @110: 0     'h4c12_4017 'h4c12_4017
@130: 0     'hcd05 'hcd05          @130: 0     'h422d_cd05 'h422d_cd05
@150: 0     'h2534 'h2534          @150: 0     'hccef_2534 'hccef_2534
@170: 0     'ha061 'ha061          @170: 0     'h6873_a061 'h6873_a061
@190: 0     'h75d9 'h75d9          @190: 0     'h7f26_75d9 'h7f26_75d9
=========================          ===================================
```

この他にも SVL には便利な機能があります。例えば、検証コンポーネント階層をワイルドカードで検索する際には、以下のように正則表現機能を使用できます。

```
if( svl_pattern_matching("*.driver[0-9]",pathname) )
```

この検索では、階層名（pathname）が.driver0、.driver1、...、driver9 で終了すれば真となります。このように、SVL を使用する事により多くの問題を自然に解消できる事がわかります。これらの例が示すように、SVL には生産性を向上させるための意義があります。

1.2 SVL の機能概要

検証作業には多くの機能が必要になります。SVL には、検証環境で必要な以下に示すような機能が備えられています。

- 記述作業の効率化を促進するマクロ機能
- プロセスのスケジューリングや待ち行列の管理に必要なプライオリティキューを実装するためのデータ構造
- 文字列をパターンで検索する事を可能にする正則表現機能
- 文字列内での検索機能
- 文字列の比較と変換機能
- ファイル入出力機能
- 検証結果をプリントするための汎用的なプリント書式生成機能
- メッセージレベルによるメッセージプリント機能
- プリント出力のターミナルとファイルへの切り替え機能
- プロセス生成機能
- クロック生成機能
- タイムアウト機能
- virtual インターフェース操作機能
- 現在時刻や作業ディレクトリの取得機能
- 環境変数の内容取得

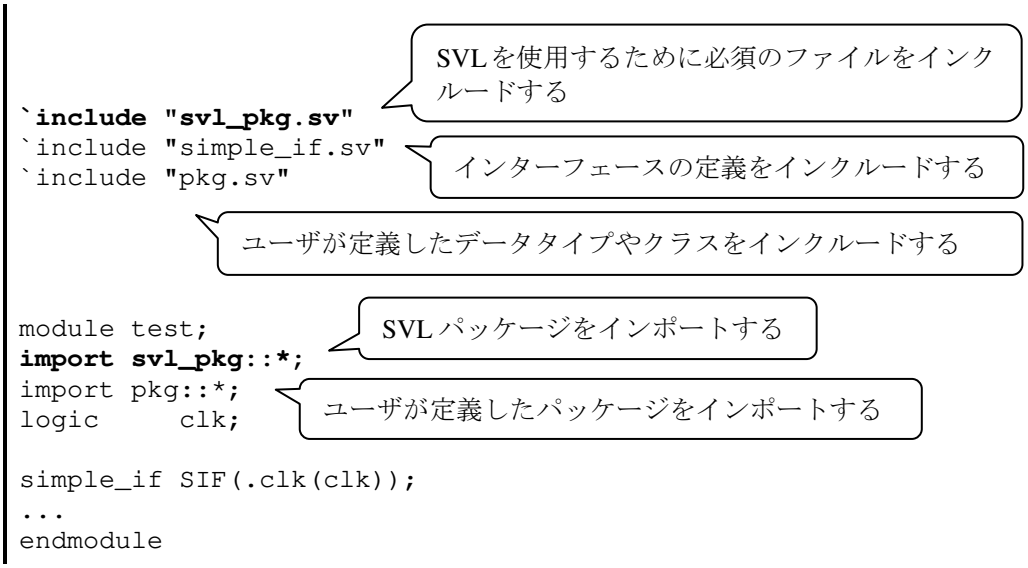
これらの機能を使用する事により、定型的な作業から解放されて本質的な作業への専念へと移行し、より優れた性能と機能を持つ検証環境を構築できるようになります。

1.3 SVL の使用手順

SVL の機能は、svl_pkg.sv ファイルに定義されているので、そのファイルをインクルードする必要があります。まとめると以下のような手順となります。

- svl_pkg.sv ファイルをインクルードする。
- ユーザが定義したファイルをインクルードする。

- SVL を使用するスコープ内で svl_pkg パッケージをインポートする。ユーザが定義したパッケージも同時にインポートする。




```
`include "svl_pkg.sv"
`include "simple_if.sv"
`include "pkg.sv"

module test;
import svl_pkg::*;
import pkg::*;
logic     clk;

simple_if SIF(.clk(clk));
...
endmodule
```

このように準備すると、モジュール test 内で SVL が提供する機能を自由に使用できます。

1.4 SVL の記法

既に気づいたと思いますが、SVL を構成する要素には一定の命名法が適用されています。命名法のルールを表 1-1 にまとめておきます。

表 1-1 SVL の命名法

命名対象	意味
svl_pkg	SVL パッケージを意味します。
svl_*_m	SVL マクロを意味します。
svl_*_t	SVL クラスのタイプを意味します。
svl_*_s	SVL で定義されたストラクチャを意味します。
svl_*_e	SVL で定義された enum を意味します。
m_*	SVL のグローバル変数とクラス内の変数には、接頭辞 m_が使用されています。

例えば、SVL にはメソッドの終了状態を示すコードが enum として定義されていますが、そのデータタイプ名には以下のように _e が付けられています。

```
typedef enum { SVL_RETURN_SUCCESS, SVL_RETURN_WARNING,
       SVL_RETURN_ERROR, SVL_RETURN_FATAL,
       SVL_RETURN_SYSTEM } svl_return_e;
```

1.5 本書の記法

本書で示す説明図では、読み易さのためにクラス名の接頭辞（svl_）と接尾辞（_t）を省略する事があります。例えば、port と書くと svl_port_t を意味します。

4　クラス

検証時に使用するデータタイプを表 4-1 に示します。

表 4-1　実行時に使用するクラス

データタイプ	意味
svl_heap_item_t	ヒープを構成する要素のデータ構造です。
svl_heap_t	ヒープデータ構造を定義するためのクラスです。ヒープはプライオリティキューを実装するために使用されます。 プライオリティキューを使うと、ソートをせずに最大値または最小値を効率良く取り出せます。最大値または最小値を表現するためにプライオリティという用語を使用します。つまり、最も高いプライオリティは、最大または最小の何れかの値を示します。 ヒープにデータを登録するだけで、最も高いプライオリティを持つデータを取り出せる準備ができています。例えば、待ち行列の管理に利用すると便利です。
svl_max_heap_t	最大の値が最も高いプライオリティと見做すヒープデータ構造です。
svl_min_heap_t	最小の値が最も高いプライオリティと見做すヒープデータ構造です。
svl_parallel_array_t	キーを指定してオブジェクトを登録するための辞書ですが、キーの順に検索できる事はもちろんですが、登録された順にオブジェクトを取得できる便利なデータ構造です。
svl_process_t	プロセスを生成する際に使用されるプロセスのベースクラスです。プロセスが生成されると、このクラスに定義されている run()メソッドが呼び出されます。
svl_vif_config_t	virtual インターフェースを操作するためのデータタイプです。

4.1　svl_heap_item_t

svl_heap_item_t は以下のように定義されています。このクラスは、ヒープのアレイ要素として使用されます。通常、ユーザが直接使用する機会はありません。定義されているプロパーティの意味は表 4-2 の通りです。

```
class svl_heap_item_t#(type PRIORITY=int,type DATA=svl_thing_t);
PRIORITY  m_priority;
DATA      m_data;
endclass
```

表 4-2　svl_heap_item_t に定義されているプロパーティ

プロパーティ	意味
m_priority	要素のプライオリティを示します。
m_data	プライオリティを持つオブジェクトを示します。通常の数値だけでなく、クラスのオブジェクトを指定できます。

4.2　svl_heap_t（プライオリティキュー用のデータ構造）

svl_heap_t はアブストラクトクラスで、以下のように定義されています。このクラスにはプライオリティキューに必要な機能が備えられています。

```
virtual class svl_heap_t#(type PRIORITY=int,type DATA=int);
typedef svl_heap_item_t#(PRIORITY,DATA) HEAP_ITEM_TYPE;
```

5　グローバルメソッド

5.1　文字列処理

表 5-1 に示すような文字列処理メソッドがあります。

表 5-1　文字列処理メソッド

メソッド	機能概要
svl_max_len(s)	文字列のアレイに存在する文字列の最大長を求めます。プリントする際の最大幅を算出する際に有効です。
svl_is_alnum(c)	指定された文字が英数字であるか調べます。
svl_is_alpha(c)	指定された文字が英文字であるか調べます。
svl_is_digit(c)	指定された文字が数字であるか調べます。
svl_is_lower(c)	指定された英字が子文字であるか調べます。
svl_is_upper(c)	指定された英字が大文字であるか調べます。
svl_is_space(c)	指定された文字が白空白であるか調べます。
svl_tolower(c)	大文字を小文字に変換して戻します。
svl_toupper(c)	小文字を大文字に変換して戻します。
svl_reverse_string(s)	与えられた文字列を逆順に入れ替えた文字列を戻します。
svl_starts_with(s,prefix)	与えられた文字列が指定した文字列で始まるか調べます。
svl_ends_with(s,suffix)	与えられた文字列が指定した文字列で終了するか調べます。
svl_first_index(s,c)	svl_first_index_from() と同じ機能を持ちますが、このメソッドは文字列の最初から検索を始めます。
svl_first_index_from(s,c,from)	与えられた文字列に指定された文字が存在するか調べ、存在すれば文字列内の最初の出現位置を戻します。検索は、文字列の指定された位置から開始されます。
svl_last_index(s,c)	svl_last_index_from() と同じ機能を持ちますが、このメソッドは文字列の最後から検索を始めます。
svl_last_index_from(s,c,from)	与えられた文字列に指定された文字が存在するか調べ、存在すれば文字列内の最も右の出現位置を戻します。検索は、文字列の指定された位置から左方向に行われます。
svl_find_first_substr(s,substr,from)	与えられた文字列に指定した部分文字列が含まれるかを調べ、最初の出現位置を戻します。
svl_find_last_substr(s,substr,from)	与えられた文字列に指定した部分文字列が含まれるかを調べ、最も右の出現位置を戻します。
svl_trim(s)	文字列の前後にある白空白を削除した文字列を戻します。
svl_replace_head_ws(s,to)	文字列の前にある白空白を指定した文字で置き換えます。
svl_replace_tail_ws(s,to)	文字列の後ろにある白空白を指定した文字で置き換えます。
svl_replace_first(s,from,to)	与えられた文字列に指定された文字が存在すれば、異なる文字で置き換えて戻します。最

	初に一致した文字だけが置き換えられます。
svl_replace_last(s,from,to)	svl_replace_first()の「最初」を「最後」に置き換えた機能と同じです。
svl_replace_all(s,from,to)	与えられた文字列に指定された文字が存在すれば、全ての出現を異なる文字で置き換えて戻します。
svl_replace_substr(s,from,to,replacement)	文字列内の部分文字列を指定した文字列で置き換えます。
svl_separate_string(s,separator,name)	指定した区切り記号を含む文字列から区切り記号で仕切られている部分文字列群を取り出します。例えば、区切り記号がドット（.）であれば、文字列 top.env0.master_agent.driver が与えられれば、top、env0、master_agent、driver のそれぞれを取得できます。
svl_strncmp(s1,s2,n)	二つの文字列の最初の N 文字を比較した結果を戻します。C/C++の strncmp()と同じです。
svl_strnicmp(s1,s2,n)	大文字と小文字の区別をせずに、二つの文字列の最初の n 文字を比較した結果を戻します。C/C++の strnicmp()と同じです。
svl_strnset(s,c,n)	文字列の最初の n 文字を指定した文字で n 回繰り返して置き換えた文字列を戻します。
svl_ascii_to_hex(c,hex)	16 進文字を値に変換します。例えば、"d" は 13 になります。
svl_ascii_to_bin(c,bin)	2 進文字を値に変換します。例えば、"1" は 1 になります。
svl_ascii_to_oct(c,oct)	8 進文字を値に変換します。例えば、"6"は 6 になります。

以下では、これらのメソッドの使用法を解説します。

5.1.1 svl_max_len

```
function int svl_max_len(input string s[]);
```

string 型のアレイ内の文字列の最大長を戻します。

5.1.2 svl_is_alnum

```
function bit svl_is_alnum(byte c);
```

指定された文字が英数字（"a"~"z"、"A"~"Z"、"0"~"9"）であるか調べます。英数字であれば 1、そうでなければ 0 を戻します。

5.1.3 svl_is_alpha

```
function bit svl_is_alpha(byte c);
```

指定された文字が英文字（"a"~"z"、"A"~"Z"）であるか調べます。英文字であれば 1、そうでなければ 0 を戻します。

□

5.4　クロック生成

クロック生成機能は SystemVerilog の以下のようにファンクションとして定義されています。このファンクションはクロック生成とタイムアウトの機能を備えています。

```
function automatic void svl_clock_gen(
    ref logic clk,
    input int cycle,
    int timeout=0,
    svl_finish_code_e finish_code=SVL_FINISH_SUCCESS);
```

タイムアウトが発生すると、finish_code で指定された動作を実行します。このファンクションが呼ばれると、即座に呼び出し側に戻りますが、クロック生成は継続します。引数の意味は表 5-6 の通りです。

表 5-6　svl_clock_gen()の引数

引数	意味
clk	クロック信号を指定します。この信号値は、クロック生成に伴い随時更新されます。
cycle	クロックサイクルを意味します。cycle/2 の間隔でクロック信号が 0 と 1 の値を繰り返します。
timeout	タイムアウトを設定します。timeout==0 と設定すると、タイムアウトの制限はありません。timeout が 0 以外であれば、タイムアウトが発生すると finish_code に従って終了処理が行われます。
finish_code	タイムアウトが発生した際の終了処理を指定します。終了処理の種類は、表 3-3 を参照して下さい。

参考 5-3

このクロック生成機能はファンクションで実装されているので呼び出し側に即座に戻りますが、クロック生成は継続します。この機能をタスクで表現する事も可能ですが、そうすると使用者にとっては svl_clock_gen からいつ戻ってくるかわからないという印象を与えるので好ましくないと思えます。
□

参考 5-4

クロック信号 clk は logic データタイプであるため、clk の初期値が x や z である可能性もあります。その場合、トグル処理が正しく動作しないため clk を 0 に設定してからクロック生成を開始します。
□

5.5　メッセージプリント機能

SVL にはメッセージプリント機能があります。戻りコードにしたがってメッセージに識別子が添えられます（表 5-7）。

表 5-7　戻りコードと識別子

戻りコード	プリントされるメッセージの識別子
SVL_RETURN_SUCCESS	"SVL_INFO"
SVL_RETURN_WARNING	"SVL_WARNING"
SVL_RETURN_ERROR	"SVL_ERROR"

SVL_RETURN_FATAL	"SVL_FATAL"
SVL_RETURN_SYSTEM	空文字列。

例えば、以下のように戻りコードを SVL_RETURN_SUCCESS に設定してメッセージをプリントできます。

```
svl_print_message(SVL_RETURN_SUCCESS,SVL_MEDIUM,
    "Successfully completed.");
```

こうすると、以下のようにメッセージ識別子が SVL_INFO としてプリントされます。

```
SVL_INFO: Successfully completed.
```

メッセージプリント機能としては、表 5-8 に示すメソッドがあります。

表 5-8　メッセージプリント機能のメソッド

メソッド	機能概要
svl_set_print_file(filename)	メッセージをターミナルの代わりにファイルに出力するようにします。
svl_suspend_print_file();	メッセージのファイル出力を一時的に中断してターミナルに出力します。
svl_resume_print_file();	メッセージのファイル出力を再開します。
svl_reset_print_file()	メッセージのファイル出力を無効にします。
svl_flush_print_file()	バッファに残っている内容をプリントファイルに書き込みます。
svl_print_message(rtn,level,msg,option)	rtn で示されているコードとオプションに従いメッセージをプリントします。
svl_info(msg)	SVL_INFO を添えてメッセージをプリントします。
svl_warning(msg)	SVL_WARNING を添えてメッセージをプリントします。
svl_error(msg)	SVL_ERROR を添えてメッセージをプリントします。
svl_fatal(msg)	重大なエラーが発生したメッセージをプリントします。
svl_system(msg)	特別な識別子を省き、指定されたメッセージだけをプリントします。
svl_system_wonl(msg)	svl_system()と同じ機能を持ちますが、改行をしません。
svl_change_message_prefix (svl_message_prefix_code_e code, string prefix)	メッセージをプリントする際に付けられる識別文字列を変更します。例えば、SVL_INFO の代わりに"#-I"を使用するように変更できます。
svl_set_message_level(level)	メッセージレベルを設定します。ここで指定されたレベル以下のメッセージだけがプリントの対象になります。
svl_set_info_message_level(level)	svl_info()のメッセージレベルを変更します。svl_info()の標準値は SVL_MEDIUM です。
svl_set_warning_message_level(level)	svl_warning()のメッセージレベルを変更します。svl_warning()の標準値は

	SVL_MEDIUM です。
svl_set_error_message_level(level)	svl_error ()のメッセージレベルを変更します。svl_error()の標準値は SVL_LOW です。
svl_set_system_message_level(level)	svl_system()のメッセージレベルを変更します。svl_system()の標準値は SVL_MEDIUM です。
svl_make_format(spec,width)	指定した幅を持つ書式を作ります。
svl_sprint_left(msg,width)	メッセージを左詰めでプリントするための書式を作ります。
svl_sprint_right(msg,width)	メッセージを右詰めでプリントするための書式を作ります。
svl_sprint_center(msg,width)	メッセージを中央揃えでプリントするための書式を作ります。
svl_sprint_string(msg,width,option)	幅とオプションに従いメッセージを揃えます。
svl_breakup(text,interval,divider)	数値を表現している文字列に区切り記号を挿入します。
svl_print_header(header)	ヘッダーをプリントします。
svl_sprintf_header(header)	カラム位置に合わせたヘッダー文字列を作ります。
svl_print_footer(header)	フッターをプリントします。
svl_sprintf_footer(header)	フッター情報を作ります。
svl_print_data(row)	カラム情報にしたがい一行をプリントします。
svl_sprintf_data(row)	一行にプリントする文字列を作ります。

5.5.1 svl_set_print_file

```
function bit svl_set_print_file(string filename);
```

メッセージをターミナルに出力する代わりにファイルに出力する機能を持ちます。ファイルを正しくオープンできれば1、オープンできなければ0を戻します。SVLのメソッドを使用してプリントされたメッセージのみファイルに出力されます。即ち、以下の情報はターミナルに出力されます。

- シミュレータがプリントする情報
- `$display` や`$write` でプリントされる情報

参考 5-5

大量の情報をプリントする場合には、ターミナルの代わりにファイルに出力すると処理が高速化されます。
□

5.5.2 svl_suspend_print_file

```
function void svl_suspend_print_file();
```

メッセージのファイル出力が有効になっている場合に使用できます。この機能は、メッセージのファイル出力を一時的に中断してターミナルに出力します。この機能を実行後に出力されるメッセージはターミナルに出力されます。

6　マクロ

6.1　汎用マクロ

記述を簡略化する際や信号値をプリントする際に必要となるマクロが準備されています（表 6-1）。

表 6-1　汎用マクロ機能

マクロ	機能概略
`svl_info_m(MSG)	svl_info()メソッドの機能に加えて、ファイル名、行番号、現在時刻の情報がプリントされます。
`svl_warning_m(MSG)	svl_warning()メソッドの機能に加えて、ファイル名、行番号、現在時刻の情報がプリントされます。
`svl_error_m(MSG)	svl_error()メソッドの機能に加えて、ファイル名、行番号、現在時刻の情報がプリントされます。
`svl_hex_digits_m(n)	n ビットの信号を 16 進で表現するために必要な桁数を計算するマクロです。 例： `svl_hex_digits_m(5)は 2 となります。
`svl_short_bin_digits_m(n)	n ビットの信号を 2 進で表現するために必要な桁数を計算するマクロです。区切り記号の数を桁数に考慮します。 例： 7 ビットであれば、111_0101 のようにプリントされるので、`svl_short_bin_digits_m(7)は 8 となります。
`svl_short_hex_digits_m(n)	n ビットの信号を 16 進で表現するために必要な桁数を計算するマクロです。区切り記号の数を桁数に考慮します。 例： 32 ビットであれば、abcd_0123 のようにプリントされるので、`svl_short_hex_digits_m(32)は 9 となります。
`svl_full_bin_digits_m(n)	`svl_short_bin_digits_m(n)の機能に加えて、'b を先頭に付加した文字列の桁数を求めます。 例： 7 ビットであれば、'b111_0101 のようにプリントされるので、`svl_full_bin_digits_m(7)は 10 となります。
`svl_full_hex_digits_m(n)	`svl_short_hex_digits_m(n)の機能に加えて、'h を先頭に付加した文字列の桁数を求めます。 例： 32 ビットであれば、'habcd_0123 のようにプリントされるので、`svl_full_hex_digits_m(32)は 11 となります。
`svl_init_print_row_m(ROW)	行をプリントする際の初期化メソッドを呼び出すマクロです。
`svl_add_print_column_m (ROW,COLUMN,HEADER)	カラム情報を行に追加します。
`svl_print_row_m(ROW)	行をプリントします。

`svl_sprintf_row_m(ROW)	行の内容を書式に従いフォーマットした文字列を生成します。
`svl_foreach_enum_m(ENUM_TYPE,v)	enum で定義されている定数を foreach 形式で走査します。

6.1.1 `svl_info_m

```
`svl_info_m(MSG)
```

svl_info()メソッドの機能に加えて、マクロを実行したファイル名、行番号、現在時刻がプリントされます。例えば、以下のようにプリントされます。

```
SVL_INFO: Print_N001.sv(11) @10: File opened
```

6.1.2 `svl_warning_m

```
`svl_warning_m(MSG)
```

svl_warning()メソッドの機能に加えて、マクロを実行したファイル名、行番号、現在時刻がプリントされます。例えば、以下のようにプリントされます。

```
SVL_WARNING: Print_N001.sv(11) @10: File opened
```

6.1.3 `svl_error_m

```
`svl_error_m(MSG)
```

svl_error()メソッドの機能に加えて、マクロを実行したファイル名、行番号、現在時刻がプリントされます。例えば、以下のようにプリントされます。

```
SVL_ERROR: Print_N001.sv(11) @10: Name not found
```

6.1.4 `svl_hex_digits_m

```
`svl_hex_digits_m(NBITS)
```

このマクロは 16 進表現として必要な桁数を求めるために使用するマクロです。例えば、7 ビットの信号を 16 進表示する際に必要な桁数は `svl_hex_digits_m(7) として求まります。

6.1.5 `svl_short_bin_digits_m

```
`svl_short_bin_digits_m(NBITS)
```

このマクロは、区切り記号を付けた 2 進表現の桁数を求めるために使用します。

6.1.6 `svl_short_hex_digits_m

```
`svl_short_hex_digits_m(NBITS)
```

このマクロは、区切り記号を付けた 16 進表現の桁数を求めるために使用します。

ENUM_TYPE には enum のタイプを指定し、v にはイタレータ名を指定します。例えば、以下のように使用します。

```
typedef   enum { RED=0, YELLOW, GREEN } color_e;

svl_system_wonl("All values for color_e:");
`svl_foreach_enum_m(color_e,v)
   svl_system_wonl($sformatf(" %s=%0d",v.name,v));
svl_system;
```

こうすると、以下のような結果を得ます。

```
All values for color_e: RED=0 YELLOW=1 GREEN=2
```

ループ内に複数の文を記述する場合には、以下のように begin と end で囲まなければなりません。

```
`svl_foreach_enum_m(color_e,v) begin
...
end
```

6.2　プリント書式マクロ

SVL にはトランザクションや検証コンポーネントの内容をプリントする際に使用されるマクロがありますが、それらの中でも一般的なプリント処理で使用できる便利なマクロがあります。本節では、それらの機能を紹介します。マクロの一覧を表 6-2 に示します。

表 6-2　プリント書式マクロ

マクロ	機能概略
`svl_name_m(name)	指定されたパラメータを名称に変換します。
`svl_sformatfb_m(v) `svl_sformatfo_m(v) `svl_sformatfh_m(v) `svl_sformatfd_m(v) ` svl_sformatfs_m(v)	$sformatf(format_str,var)をマクロで表現した機能です。
`svl_breakupb_m(v) `svl_breakupo_m(v) `svl_breakuph_m(v) `svl_breakupd_m(v)	パラメータ v を数値に変換後、見易くするために区切り記号を挿入します。 例： "0101_1100"
`svl_pbreakupb_m(v) `svl_pbreakupo_m(v) `svl_pbreakuph_m(v)	上記の機能に加えて数値の表現タイプを先頭に付けます。 例： "'b0101_1100"
`svl_sprintfb_m(v) `svl_sprintfo_m(v) `svl_sprintfh_m(v)	上記の機能に加えて数値のビット数が添えられます。 例： "8'b0101_1100"
`svl_nsprintfb_m(v) `svl_nsprintfo_m(v) `svl_nsprintfh_m(v) `svl_nsprintfd_m(v)	カンマで区切られたパラメータ v の名称とその値が生成されます。 例： `svl_nsprintfb_m(sum)とすると以下のように文字列が生成されます。 "sum", "8'b0101_1100"
`svl_nsformatfb_m(v) ` svl_nsformatfo_m(v)	カンマで区切られたパラメータ v の名称とその値が生成され