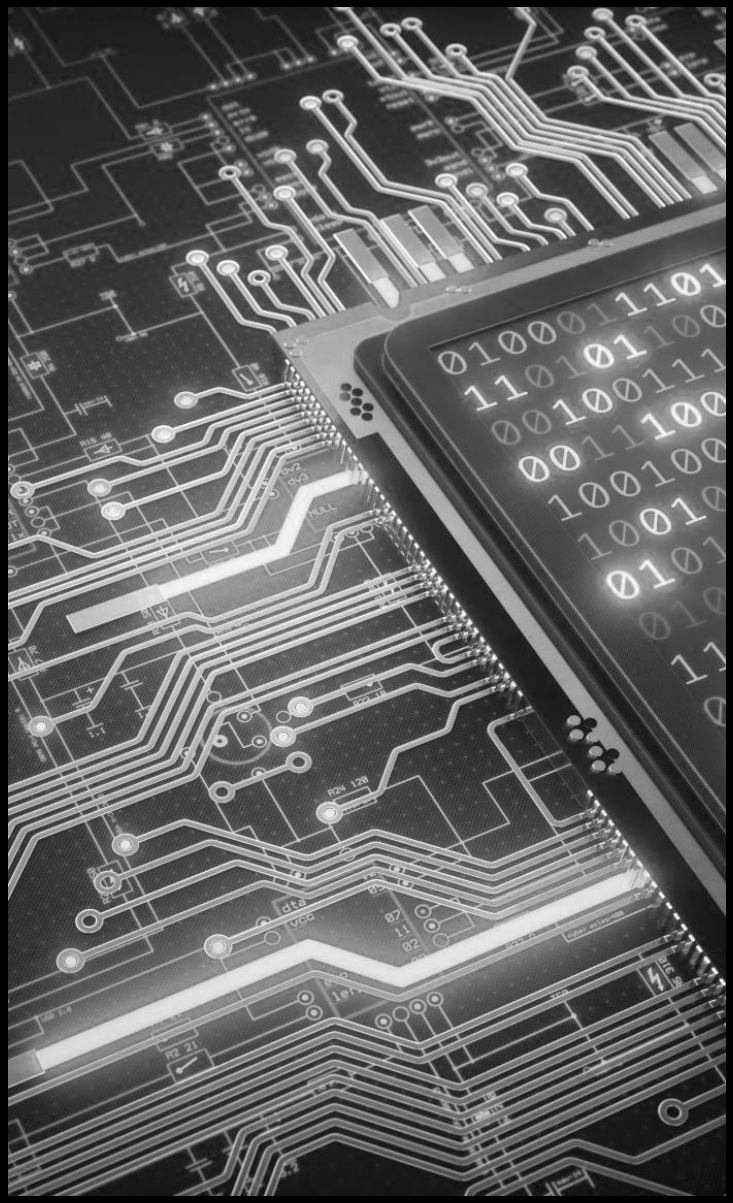


Artgraphics

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

SVL Standard

- SVLは、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SVL Standardの基本機能に加えて、検証環境構築時に必要となる機能で構成されています。
- 本資料はSVL Standardの概要を紹介する資料です。以降では、SVL StandardをSVLと略称します。

SVLの意義と目的

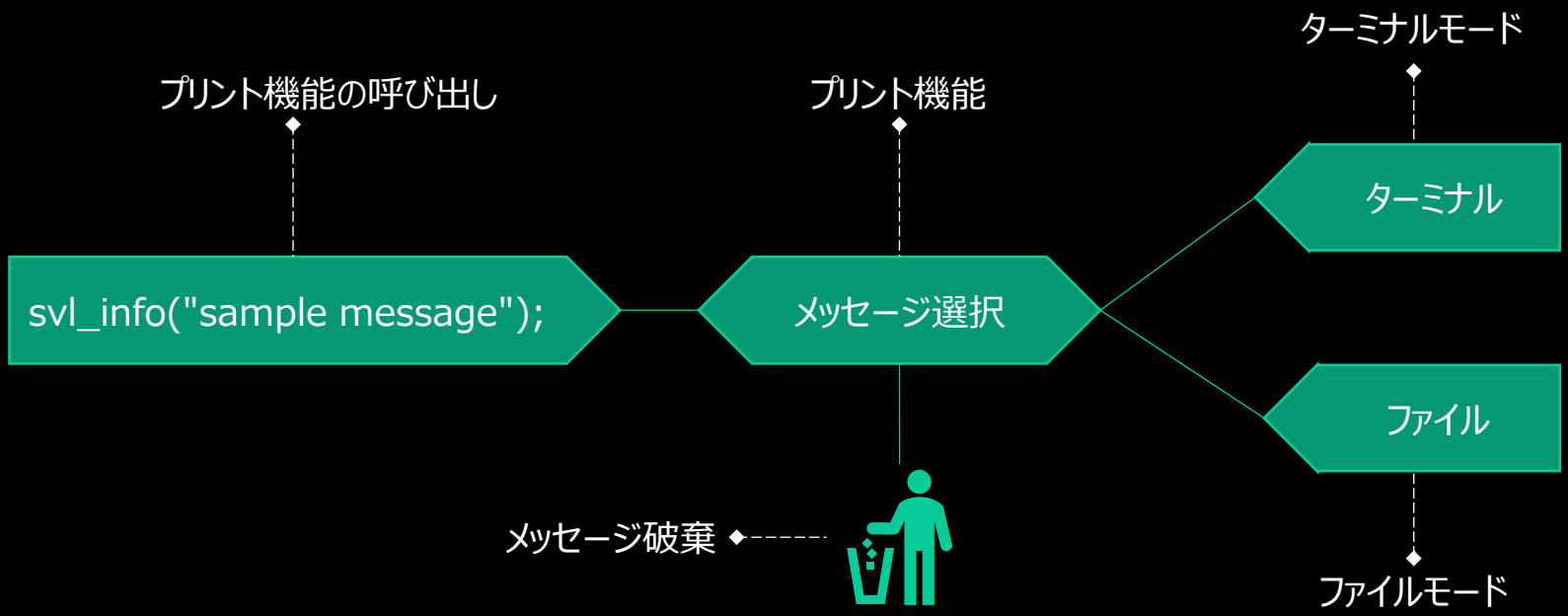
- 検証環境を構築するためのパッケージとしてはUVMが良く知られています。UVMには、検証コンポーネントやトランザクションの定義を容易にする機能が含まれていますが、検証コード記述のための生産性向上技術は含まれていません。
- SVLは、UVM等のパッケージに不足している生産性向上技術を提供します。勿論、SVLはUVMとは直接的な関連を持たないため、SVLを単独に生産性向上のためのパッケージとして使用する事ができます。寧ろ、これがSVLの目指す本来の目的です。

SVLによる記述の柔軟性の促進

- メッセージをプリントするために$displayシステムタスクを使うとターミナルに出力されます。ファイルに出力するためには$fdisplayを使用しなければなりません。したがって、状況に応じて出力先が変化する訳ではないので、ユーザが使い分けをしなければなりません。
- SVLのプリント機能は、状況に応じてターミナルまたはファイルに出力が行われます。同じプリント機能が使用されていても、モードを変更する事によりターミナル出力を一時的にファイル出力に切り替える事ができます。

プリント機能の動作

- 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。
- しかも、メッセージレベルを指定できるので、重要ではないメッセージを抑止できます。メッセージレベルはいつでも自由に変更できます。



プリント機能処理

- プリント処理を行う主な機能は以下の通りです。

記述法	機能
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_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_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。

プリント書式機能

- 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。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。
- SVLは、殆どの必要性に応じられるプリント書式機能を備えています。

プリント書式機能の使用例

- プリント書式機能はSystemVerilogマクロとして実装されています。
  以下に代表的な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

汎用プリント処理機能

- 検証結果をプリントする際、プリント用のレイアウトを定義し出力結果のカラムが揃う様に書式の定義とプリント処理を記述しなければなりません。また、データのビット数に変更が発生したり、プリントカラム位置の変更が起これば、面倒な作業の繰り返しが起こります。
- SVLのプリント機能を使用すると、面倒なカラム揃え作業から解放されます。また、データ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。
- 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。

汎用プリント処理機能の使用例

- プリントレイアウトを定義するモデルを右のように定義します。

```
svl_print_header_s header[] = '{
        '{COLUMN_TIME,5,SVL_HEADER_DEFAULT},
        '{COLUMN_RESET,5,SVL_HEADER_DEFAULT},
        '{COLUMN_D,`svl_full_hex_digits_m(WIDTH),SVL_HEADER_DEFAULT},
        '{COLUMN_Q,`svl_full_hex_digits_m(WIDTH),SVL_HEADER_DEFAULT}
        };
```

- モデルに合わせてプリント処理を記述します。

```
function void print();
svl_print_data_s        row[$], column;
        `svl_init_print_row_m(row)
        foreach(header[i]) begin
                case (header[i].m_name)
                COLUMN_TIME: column.m_value = $sformatf("@%3t:",$time);
                COLUMN_RESET: column.m_value = `svl_sformatfb_m(reset);
                COLUMN_D: column.m_value = `svl_pbreakuph_m(d);
                COLUMN_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
```

汎用プリント処理機能の実行例

- モデルに従いプリント処理を記述しているので、モデルの変更に従いプリントされる結果も変更されます。
- 下記の結果はモデルの変更に対応して得られたプリント結果です。何れの場合にも同じprint()が変更されずに使用されています。

```
=========================
time  reset d      q
-------------------------
@ 10: 0     'hc778 'hc778
@ 30: 0     'hc8d0 'hc8d0
@ 50: 0     'h1471 'h1471
@ 70: 0     'h2004 'h2004
@ 90: 0     'hea38 'hea38
@100: 1     'h17ef 'h0000
@110: 0     'h17ef 'h17ef
@130: 0     'h8562 'h8562
@150: 0     'hfc26 'hfc26
@170: 0     'h3b02 'h3b02
@190: 0     'h02a2 'h02a2
=========================
```

WIDTH==16

```
===================================
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
===================================
```

WIDTH==32

```
=========================
time  reset q      d
-------------------------
@ 10: 0     'hc778 'hc778
@ 30: 0     'hc8d0 'hc8d0
@ 50: 0     'h1471 'h1471
@ 70: 0     'h2004 'h2004
@ 90: 0     'hea38 'hea38
@100: 1     'h0000 'h17ef
@110: 0     'h17ef 'h17ef
@130: 0     'h8562 'h8562
@150: 0     'hfc26 'hfc26
@170: 0     'h3b02 'h3b02
@190: 0     'h02a2 'h02a2
=========================
```

WIDTH==16でdとqのカラムを入れ替えた例

ユーティリティ機能の使用例

- 使用方法は、以下に示すように簡単です。

記録として現在の時刻
をプリントする

```
initial begin
        svl_system({"DATE: ",svl_date()});
        svl_system({"CWD: ",svl_cwd()});
        svl_print_header(header);
        svl_clock_gen(clk,20);
end
```

実行ディレクトリを取得し
てプリントする

クロック生成

正則表現（パターンマッチング）

- 標準的なパターンマッチング機能が備わっています。

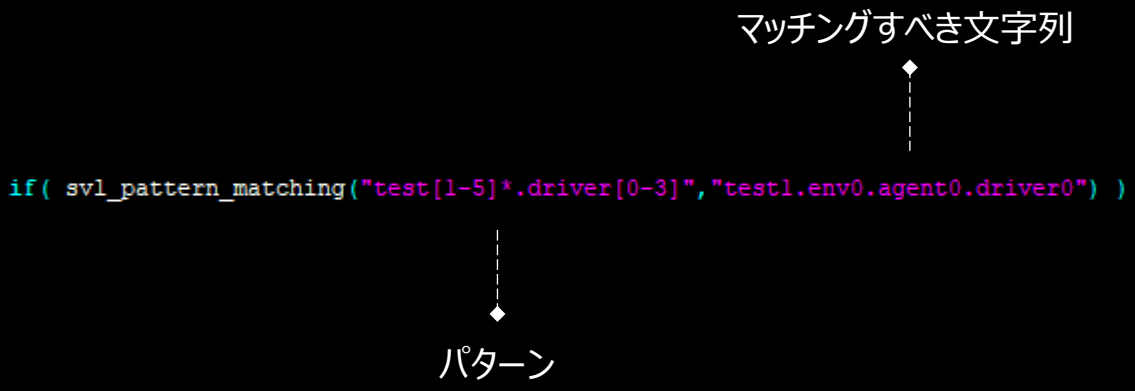
使用できるパターン	機能
[…]	[]内に文字を列挙できます。指定された文字の何れかとマッチします。連続した文字を表現するためにハイフン（-）を使用できます。例えば、a-cのように使用できます。 使用例： [agw]：aかgかwの何れかを意味します。 [D-G]：DかEかFかGの何れかを意味します。 [a-zA-Z]：アルファベットを意味します。
*	ゼロ個以上の文字を表現します。
?	任意の一文字を表現します。
¥c	cを通常の意味を持つ文字として扱うように指示します。例えば、¥[とすると文字 [はパターンの制御文字ではなく通常の意味を持ちます。
その他の文字	指定された文字自身を示します。

パターンマッチングの例

- パターンマッチングをするための機能は幾つかありますが、最も簡単な方法は、以下のメソッドを呼び出す事です。

```
function bit svl_pattern_matching(string pattern,string name);
```

- patternにパターンを指定し、nameにマッチングすべき文字列を指定します。



ファイル入出力機能

- SystemVerilogのファイル操作機能は直感的に明白でないため使いづらいという欠点があります。
- SVLのファイル入出力機能は簡単に使用できるように設計されています。
- ファイル入出力には以下に示すメソッドを利用できます。

メソッド	機能概要
svl_get_line(filename,line)	指定されたファイルの最初の一行を読み込みlineにセットします。
svl_get_lines(filename,lines)	指定されたファイルの全ても内容を読み込みlinesにセットします。
svl_head(filename,n,lines)	指定されたファイルの最初のn行を読み込みlinesにセットします。
svl_tail(filename,n,lines)	指定されたファイルの最後のn行を読み込みlinesいセットします。
svl_put_lines(filename,lines)	指定されたファイルに文字列を書き込みます。

ファイル入出力機能の使用例

- ファイルから文字列を入力するには、svl_get_lines()を使用するのが最も簡単です。

```
module test;
import svl_pkg::*;
string  filename = "SystemData/tmp_file.dat";
string  lines[$], s;

initial begin
        if( svl_cmd("ls",filename) ) begin
                if( svl_get_lines(filename,lines) ) begin
                        foreach(lines[i]) begin
                                s = lines[i];
                                if( svl_ends_with(s,".dat\n") ||
                                        svl_ends_with(s,".log\n") )
                                        svl_system_wonl($sformatf("%s",lines[i]));
                        end
                end
        end
end

endmodule
```

Linuxのlsコマンドを実行し結果をファイルに出力している

lsコマンドの結果をキューに読み込む

.datや.logファイルがリストされていればそれらの情報をプリントする

文字列処理機能

- 文字列処理機能は豊富で殆どの需要に応えます。
- 例えば、文字を調べる機能は以下のように準備されています。

メソッド	機能
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。

文字列処理機能（続）

- 文字列を扱う代表的な機能を以下に紹介します。

メソッド	機能
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。

ランダム文字列生成機能

- SystemVerilogには乱数を発生する機能はありますが、ランダム文字列を生成する機能はありません。SVLには以下の仕様を持つランダム文字列生成機能があります。

```
function string svl_random_string(
     input svl_ascii_mode_e ascii_mode=SVL_ANYCHAR,
     int min_len=8,max_len=16);
```

- ascii_modeは以下に示す中から選択できます。

ラベル	意味
SVL_ANYCHAR	任意の文字を使用して文字列を構成する。
SVL_ALNUM	英数字のみを使用して文字列を構成する。
SVL_ALPHA	英文字だけを使用して文字列を構成する。
SVL_NUMERIC	数字だけを使用して文字列を構成する。
SVL_BINARY	0と1だけを使用してバイナリー文字列を構成する。
SVL_HEXADECIMAL	16進数の文字列を構成する。
SVL_SPECIALCHAR	特殊文字だけを使用して文字列を構成する。

ランダム文字列生成機能の使用例

- 使用法は以下のように簡単です。

```
function void test_random_string(svl_ascii_mode_e ascii_mode);
        svl_system($sformatf("--- %s ---",ascii_mode.name));
        repeat (5) begin
                svl_system($sformatf("%s",svl_random_string(ascii_mode,10,20)));
        end
        svl_system();
endfunction
```

ランダム文字列生成機能の呼び出し

- 以下は、ランダム文字列の生成例です。

```
--- SVL_ANYCHAR ---
~4%*CQa"/y01!=:@K_d
rJ;n7T$'{UfQ
(qk/n¥T)'M`4eDY_vTD"
pI5/`Hy'iq$T
e/,X(S~N~cuD1t^)]5
```

```
--- SVL_ALNUM ---
BHqVt1BCptd
Ae9H2y8Xn0aKSre4BPfa
jkIxIXbOaa
iTxJxXZcqyW0emTZdFgT
8BuXSKN65uZE
```

SVLの主な機能

- 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

まとめ

- SVLの意義と目的を中心として概要を紹介しました。そして、検証作業で必要となる機能が汎用的な仕様でSVLに備わっている事を具体例と共に紹介しました。
- ここで紹介した機能の他に多くの有用な機能がSVLには備わっています。更に詳しい解説はSVLの仕様書をご覧下さい。
- SVLはSystemVerilogのパッケージとして定義されているので、通常のパッケージの使用法と全く同じです。
- SVLはSystemVerilogソースコードとして提供されるので、インストールに手間がかかる事はありません。