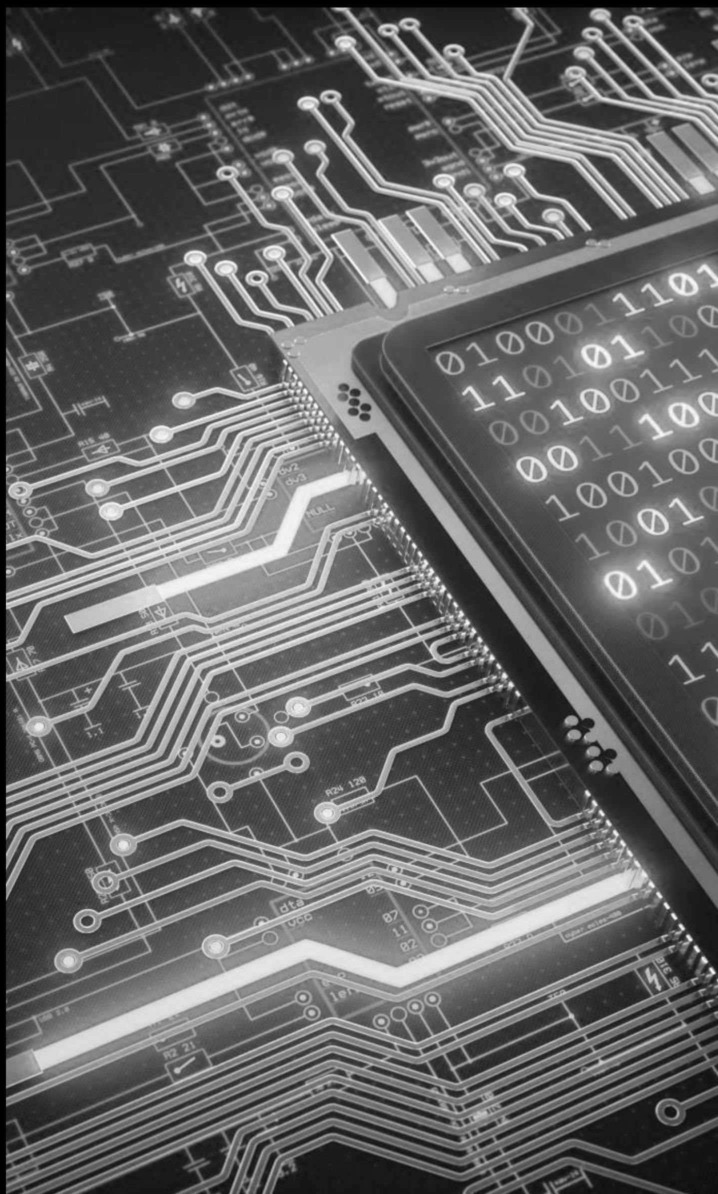


Artgraphics

SVL Premium 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 Premium

- SVLは、検証作業で必要となる検証機能を含む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 Premiumの概要を紹介する資料です。SVL PremiumはTLM (Transaction Level Modeling)をベースにした検証方式を採用しています。本資料では、SVL Standardに含まれていない検証環境構築機能を中心に紹介します。以降では、SVL PremiumをSVLと略称します。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

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 Premium, Copyright © 2025 Artgraphics. All rights reserved.

SVLの検証環境構築機能

- SVLはSystemVerilogによる検証環境を構築する際に必要となる基本的な機能を備えています。特に、SVLはクラスを用いたトランザクションベースの検証手法に適しています。
- 以下に紹介する機能を備えたSVLを利用する事によりユーザはTLMベースの検証環境を構築できます。
  - 検証環境を構築するためのメソドロジークラス
  - TLMを支援する機能
  - テストケースを実行制御するシナリオ機能
  - トランザクション生成機能
  - 検証環境をダイナミックに変更する機能
  - virtualインターフェース操作機能
  - テストケースを実行時に指定する機能

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

RTLとTLM

- RTLではデバイス間をネットで接続し、信号値の変化によりデバイスの動作を確認しますが、TLMでは検証コンポーネントを仮想的に接続します。即ち、ネットの代わりにタスクやファンクション等のメソッドの呼び出しで検証コンポーネント間のデータ授受を行います。
- そのためには、TLMでは検証コンポーネントにTLMポートを定義し、コンポーネントのポート同士に関係を持たせます。この関係樹立は実行時に行えるので、TLMはダイナミックな検証環境を構築する手段となります。 TLMでは対象となるデータをトランザクションと呼びます。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

クライアントとサーバー

- SVLでは、トランザクションの取得にget()、送信 にput()、転送にwrite()メソッドを使用します。何れのメソッドを使用する際にも、クライアントとサーバーがあります。
- クライアントは、命令を発行する側で、その命令を処理する側がサーバーです。そして、サーバーが処理手順をメソッドとして定義します。
- 送信と転送命令は、ポートを介して行われます。SVLでは、この関係を下図のように表現します。サーバー側は、メソッドの内容を埋めるという意味でポートを黒く塗りつぶします。何れの丸もTLMポートを意味します。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

TLMポート

- SVLには多くのTLMポートが備えられています。

ポートクラス	機能
svl_port_t	全てのSVLポートのベースクラスです。全てのポートに共通する情報を定義しています。
svl_io_port_t	一対一のトランザクション処理をするポートのベースクラスです。
svl_get_port_t	トランザクションを取得するポートのベースクラスです。
svl_get_server_t	トランザクションを準備するポートのベースクラスです。
svl_put_port_t	トランザクションを送信するポートのベースクラスです。
svl_put_server_t	送信されたトランザクションを処理するポートのベースクラスです。
svl_nbio_port_t	一対一のトランザクション処理をするポートのベースクラスですが、ノンブロッキング方式を用います。
svl_nbget_port_t	ノンブロッキング方式でトランザクションを取得するポートのベースクラスです。
svl_nbget_server_t	ノンブロッキング方式でトランザクションを準備するポートのベースクラスです。
svl_nbput_port_t	ノンブロッキング方式でトランザクションを送信するポートのベースクラスです。
svl_nbput_server_t	ノンブロッキング方式で送信されたトランザクションを処理するポートのベースクラスです。
svl_pass_port_t	一対Nのトランザクション処理をするポートのベースクラスです。
svl_send_port_t	一対Nのトランザクションを転送するポートのベースクラスです。
svl_receive_port_t	一対Nのトランザクションを受信するポートのベースクラスです。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

get

- トランザクションを取得する手順では、クライアントはsvl_get_port_tを使用し、サーバーはsvl_get_server_tを使用します。
- クライアントが、svl_get_port_tを通してget()メソッドを呼び出すと、サーバーに定義されているget()メソッドが呼ばれます。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

getの使用例

- クライアントがTLMポートを介してget()を呼び出すと、サーバーに定義されているget()メソッドが呼ばれます。

```
class client_t extends svl_component_t;
typedef transaction_t   TR;
svl_get_port_t#(TR,client_t)            m_get_port;

function new(string name,svl_component_t parent);
        super.new(name,parent);
        m_get_port = new("get_port",this);
endfunction
task run();
TR      item;
int     delay;
        repeat( 5 ) begin
                delay = $urandom_range(1,10);
                #delay;
                m_get_port.get(item);
                svl_info($sformatf("@%3t: number = %0d",
                        $time,item.m_number));
        end
endtask
endclass
```

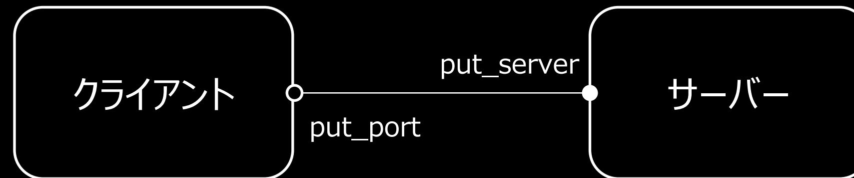
トランザクションの取得要求を出している

```
class server_t extends svl_component_t;
typedef transaction_t   TR;
svl_get_server_t#(TR,server_t)  m_get_server;

function new(string name,svl_component_t parent);
        super.new(name,parent);
        m_get_server = new("get_server",this);
endfunction

task run();
endtask
task get(output TR item);
        item = new;
        item.m_number = $urandom_range(100,200);
endtask
endclass
```

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

put

- トランザクションを送信する手順では、クライアントはsvl_put_port_tを使用し、サーバーはsvl_put_server_tを使用します。
- クライアントが、svl_put_port_tを通してput()メソッドを呼び出すと、サーバーに定義されているput()メソッドが呼ばれます。



SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

putの使用例

- クライアントがTLMポートを介してput()を呼び出すと、サーバーに定義されているput()メソッドが呼ばれます。

```
class client_t extends svl_component_t;
typedef transaction_t   TR;
svl_put_port_t#(TR,client_t)            m_put_port;

function new(string name,svl_component_t parent);
        super.new(name,parent);
        m_put_port = new("put_port",this);
endfunction
task run();
TR      item;
int     delay;
        repeat( 5 ) begin
                delay = $urandom_range(1,10);
                #delay;
                item = new;
                item.m_number = $urandom_range(100,200);
                m_put_port.put(item);
        end
endtask
endclass
```

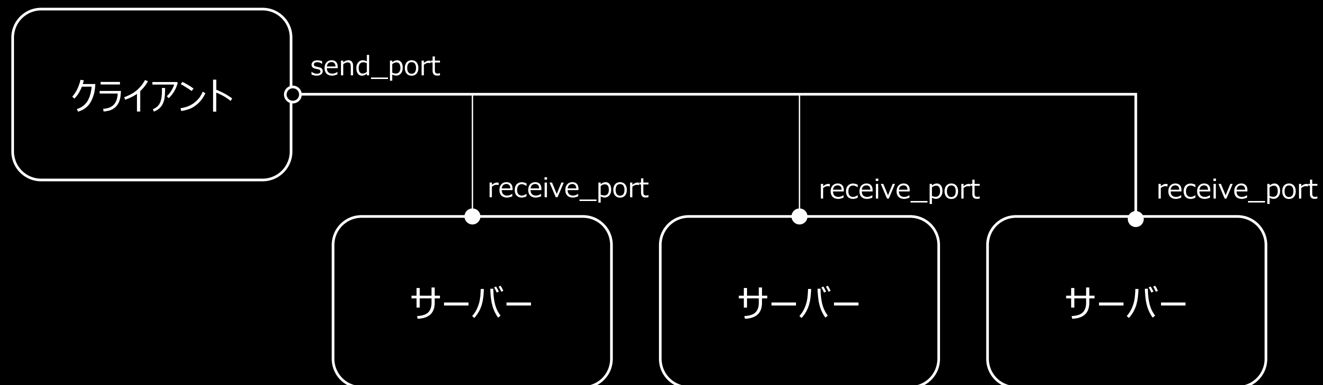
トランザクションの送信
要求を出している

```
class server_t extends svl_component_t;
typedef transaction_t   TR;
svl_put_server_t#(TR,server_t)  m_put_server;

function new(string name,svl_component_t parent);
        super.new(name,parent);
        m_put_server = new("put_server",this);
endfunction

task run();
endtask
task put(input TR item);
        svl_info($sformatf("@%3t: number = %0d",
                $time,item.m_number));
endtask
endclass
```

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

write

- トランザクションを一斉に転送するためにはwrite()メソッドを使用します。転送するコンポーネントはクライアントで、トランザクションを受信するコンポーネントはサーバーになります。
- クライアントは、svl_send_port_tを通してwrite()メソッドでトランザクションを送信します。サーバー側では、svl_receive_port_tを使用してトランザクションを受け取ります。



SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

writeの使用例

- サーバーは、トランザクションの処理方式をwrite()メソッドとして実装します。一般的には、サーバーとしては、チェッカーやスコアボードがあります。最も簡単な構造の場合には、コレクターがクライアントで、モニターがサーバーになります。

```
class client_t extends svl_component_t;
typedef transaction_t   TR;
svl_send_port_t#(TR,client_t)   m_send_port;

function new(string name,svl_component_t parent);
        super.new(name,parent);
        m_send_port = new("send_port",this);
endfunction

task run();
TR      item;
        repeat( 5 ) begin
                #10;
                item = new;
                item.m_number = $urandom_range(100,200);
                m_send_port.write(item);
        end
endtask

endclass
```

トランザクションの送信要求を出している

```
class server_t extends svl_component_t;
typedef transaction_t   TR;
svl_receive_port_t#(TR,server_t)        m_receive_port;

function new(string name,svl_component_t parent);
        super.new(name,parent);
        m_receive_port = new("receive_port",this);
endfunction

function void write(input TR item);
        svl_info($sformatf("@%3t: number = %0d",
                $time,item.m_number));
endfunction

endclass
```

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

try_put / try_get

- putとgetはブロッキング方式なので、サーバーがトランザクション処理を終了するまでクライアントは待たなければなりません。
- SVLには、ノンブロッキング方式でトランザクション処理を行えるtry_putとtry_getを備えています。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

メソドロジークラス

- TLMでは検証コンポーネントにはTLMポートが必要ですが、その都度TLMポートを定義するのは不便なので、SVLでは予めTLMポートを実装している検証コンポーネントのクラスを備えています。それらのクラスを総称してメソドロジークラスと呼びます。
- 例えば、SVLのドライバークラスはgetポートを標準的に備えています。また、コレクタークラスはsendポートを標準的に備えています。
- SVLのメソドロジークラスを使用する事により、煩わしいTLMポートの準備作業を回避できます。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

主なメソドロジークラス

- 主なメソドロジークラスを以下に紹介します。トランザクションを操作する機能を持つクラスには、TLMポートが実装されています。

SVLクラス	機能
svl_driver_t	ドライバーのベースクラスです。ユーザはドライバーをこのクラスのサブクラスとして定義しなければなりません。ドライバーは、ジェネレータからトランザクションを取得する機能を備えています。
svl_generator_t	ジェネレータのベースクラスです。ジェネレータは、ドライバーの要求に応じてトランザクションを生成して戻します。
svl_collector_t	コレクターのベースクラスです。コレクターは、DUTからのレスポンスをサンプリングしてトランザクションに変換して、モニターにトランザクションを送信する役目を持ちます。一般的には、コレクターは検証に関わる作業を担当しません。
svl_monitor_t	モニターのベースクラスです。モニターは、コレクターから受信したトランザクションを他の検証コンポーネントに一斉に転送します。モニター自身も簡単な検証作業を行います。
svl_agent_t	エージェントのベースクラスです。エージェントは、ドライバー、ジェネレータ、コレクター、モニターから構成される最小単位の階層的検証コンポーネントです。
svl_env_t	エンバイロンメントのベースクラスです。エンバイロンメントは、エージェントや他のエンバイロンメントから構成される階層的な検証コンポーネントで、検証内容に応じて様々な規模のエンバイロンメントが開発されます
svl_scoreboard_t	スコアボードのベースクラスです。スコアボードは、DUTからのレスポンスを検証する役目を持ちます。
svl_test_t	テストのベースクラスです。テストは、一般に、複数のエンバイロンメントから構成されますが、テスト同士が共有する資源をベースクラスとして定義するのが一般的です。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

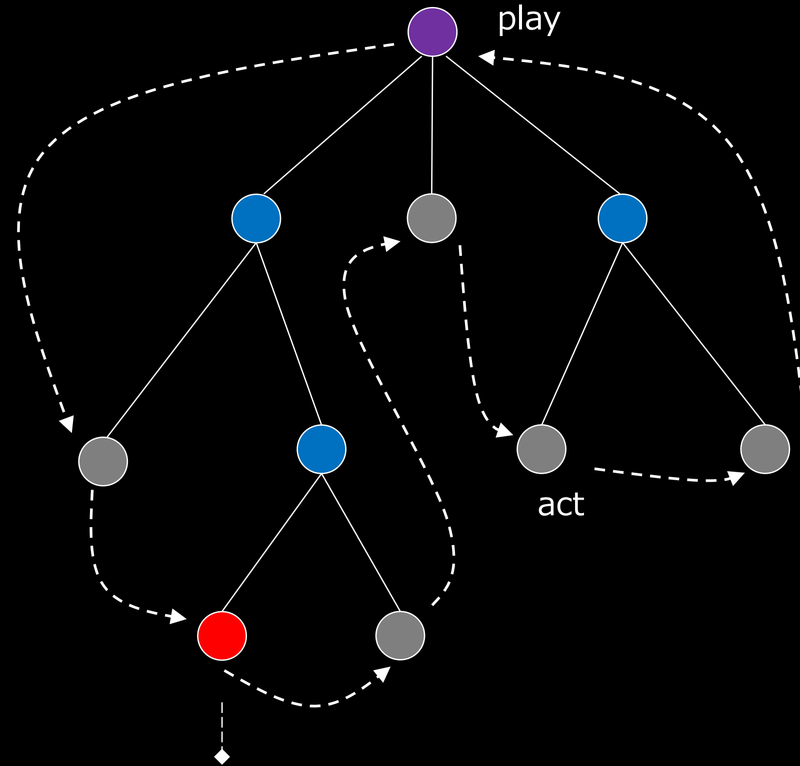
プレイ / アクト / シナリオ / テストケース

- SVLでは、テストデータを生成するオブジェクトをアクトと呼び、階層的に配置して複雑なテストケース生成を実現しています。階層構造のルートにはプレイと呼ばれるオブジェクトが使用されますが、階層構造の内部ノードとリーフノードはアクトが担当します。
- ツリー内部ノードでは、SystemVerilogのプログラミング機能を使用できるので、下位の部分ツリーを複数回繰り返す事ができます。したがって、実質無数の組み合わせを実現できます。
- ツリーのルートをsvl_play_tのオブジェクトで表現し、ツリー内の他のノードをsvl_act_tのオブジェクトで表現します。
- シナリオとは、テストケースを実行するための手順でバッチ処理のスクリプトの役割に相当します。シナリオは、ジェネレータにプレイを割り当てる役目を果たします。ジェネレータは、プレイで表現されている階層に従い、一連のトランザクションを生成する制御をします。
- テストケースを表現するテスト用の検証コンポーネントがシナリオの設定を行います。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

トランザクション生成の流れ

- ドライバーがトランザクションを要求すると、ジェネレータはアクティブなリーフノードを実行してトランザクションを生成しドライバーに戻します。
- 右図の赤いノードがアクティブなノードを示しています。
- 処理されたアクティブノードの次のノードがアクティブになり、次のトランザクション生成の対象になります。



SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

ドライバーの定義例

- 簡単なドライバーの定義を以下に紹介します。

```
class driver_t extends svl_driver_t#(simple_item_t);
vif_config::vif_type    vif;
`svl_auto_component_m(driver_t)
`svl_component_new_m
`svl_extern_connect_step_m
`svl_extern_run_step_m
extern task drive_dut(TR item);
endclass
```

ドライバークラス

```
task driver_t::drive_dut(TR item);
        vif.reset <= item.reset;
        vif.load <= item.load;
        vif.up_down <= item.up_down;
        vif.d <= item.d;
        if( item.reset )
                vif.reset = #1 0;
endtask
```

このタスクはDUTをドライブしています

```
function void driver_t::connect_step(svl_run_param_t param);
        super.connect_step(param);
        vif = vif_config::get();
endfunction
```

virtualインターフェースを設定しています

```
task driver_t::run_step(svl_run_param_t param);
TR      item;
        m_get_port.m_connected_port.set_option(SVL_PORT_REUSE_TRANSACTION);
        forever begin
                m_get_port.get(item);
                drive_dut(item);
                @(negedge vif.clk);
        end
endtask
```

このタスクで$time==0および(negedge clk)のタイミングでトランザクションを生成しています

ベースクラスのメソドロジークラスでTLMポートが定義されているので、このドライバー内で自由に使用できます

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

テストケースの記述例

- テストケースは、以下のようにシナリオを設定します。

```
class test1_t extends test_base_t;
`svl_auto_component_m(test1_t)
function new(string name="test1",svl_component_t parent=null);
        super.new(name,parent);
endfunction


function void build_step(svl_run_param_t param);
        super.build_step(param);
        svl_scenario_t::allocate("test1*generator",
                `svl_create_object_m(play_reset_down_t,"play"));
endfunction
endclass
```

ジェネレータにプレイのオブジェクトを割り当てています

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

virtualインターフェースの操作

- SVLはvirtualインターフェースを操作するためのマクロを準備してあります。インターフェースを定義した際に、このマクロを使用してvirtualインターフェースを操作するクラスを生成しておくと、 virtualインターフェースの設定と取得が容易になります。
- 以下のように、インターフェース定義後にvirtualインターフェースを操作するクラスを定義しておきます。この場合には、vif_configと呼ばれるクラスが生成されます。

```
interface simple_if import pkg_definitions::*; (input logic clk);
logic [UP_DOWN_WIDTH-1:0]       d, q, qn;
logic   reset, load, up_down;
clocking cb @(posedge clk); endclocking
clocking cbr @(posedge reset); endclocking

initial begin
        reset = 0;
        load = 0;
        up_down = 1;
        d = '0;
end
endinterface
`svl_vif_config_m(virtual simple_if,vif_config)
```

virtualインターフェースを操作するクラスを生成

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

virtualインターフェースの設定と取得

- トップモジュールでvirtualインターフェースを設定し、検証コンポーネント内で取得します。設定には、vif_config::set()を、取得にはvif_config::get()を使用します。

```
module top;
import svl_pkg::*;
import pkg_definitions::*;
import pkg::*;
logic           clk;

simple_if SIF(.clk(clk));
up_down_counter #(.NBITS(UP_DOWN_WIDTH))
        DUT(.clk,.reset(SIF.reset),.load(SIF.load),
        .up_down(SIF.up_down),
        .d(SIF.d),.q(SIF.q),.qn(SIF.qn));

initial begin
        svl_clock_gen(clk,20);
        svl_set_timeout(10,100);
        vif_config::set(SIF);
        svl_run_test();
end

endmodule
```

トップモジュールでvirtualインターフェースを設定しています

```
function void driver_t::connect_step(svl_run_param_t param);
        super.connect_step(param);
        vif = vif_config::get();
endfunction
```

ドライバー内でvirtualインターフェースを取得しています

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

シミュレーションの実行

- SVLのシミュレーションは、svl_run_test()メソッドにより起動されます。そして、シミュレーションが開始するとSVLが実行制御を握り、適切なタイミングで検証コンポーネントに定義されているシミュレーションステップを呼び出します。

実行順序	ステップ	説明
1	build_step	コンポーネントの階層を構築するステップです。階層のトップから順に呼ばれて行きます。通常、サブコンポーネントをこのステップで作成します。サブコンポーネントを作ると、そのサブコンポーネントのbuild_step()が次に呼ばれるので、階層構造がトップダウンで構築されます。トップダウンで呼び出されるので、このステップ内では最初にベースクラスのbuild_step()を呼び出す事が必要になります。
2	connect_step	コンポーネント間の接続を完成するステップです。例えば、TLMポートの接続を完了します。あるいは、virtualインターフェースの設定等も行います。このステップはボトムアップの順序で呼ばれます。
3	setup_step	シミュレーションが開始する直前にこのステップが呼ばれます。初期化処理等を行えます。
4	run_step	シミュレーションを行うためのステップです。
5	collect_step	run_step()の実行が終了すると、このステップに制御が移ります。
6	check_step	collect_step()が終了すると、このステップが呼ばれます。
7	conclude_step	最後に呼ばれるステップです。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

コンフィギュレーション設定変更

- SVLでは、ダイナミックにコンフィギュレーションの設定変更を行えます。
- シミュレーションの実行は、幾つかのステップに分かれて行われるので run_stepが実行する前であれば、適切なステップでコンフィギュレーションの設定を変更できます。
- 設定変更には、次の二種類があります。
  - クラスタイプの変更
  - クラスプロパティの変更

クラスタイプの変更例

- 例えば、トップモジュールで以下のようにクラスタイプ変換の準備をできます。

```
module top;
import  svl_pkg::*;
import  pkg::*;
test_t  test;

initial begin
        `svl_change_type_m(monitor_t,monitor_fc_t)
        ...
end
endmodule
```

monitor_t を monitor_fc_t で置き換える指定をしている

- こうすると、monitor_tのインスタンスはmonitor_fc_tのインスタンスに置き換わります。

```
function void agent_t::build_step(svl_run_param_t param);
        super.build_step(param);
        if( m_function == SVL_DRIVE_CHECK ) begin
                driver = `svl_create_component_m(driver_t,"driver",this);
                generator = `svl_create_component_m(generator_t,"generator",this);
        end
        collector = `svl_create_component_m(collector_t,"collector",this);
        monitor = `svl_create_component_m(monitor_t,"monitor",this);
endfunction
```

monitor は monitor_fc_t のインスタンスとなる

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.

まとめ

- 以上、TLMベースの検証環境を構築するクラスライブラリーの機能概要を紹介しました。
- 記述例を通して、多くの点においてUVMよりも簡略化された記述方式が可能である事を観察できたと思います。更に詳しい解説はSVLの仕様書をご覧下さい。
- この他としてSVL Standardの持つ機能も使用できるので、SVLは検証環境構築作業において強力なツールとなります。
- 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。

SVL Premium, Copyright © 2025 Artgraphics. All rights reserved.