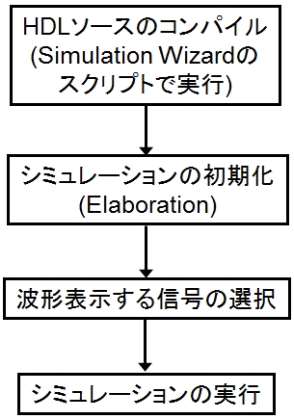
第 19 章　論理シミュレーション

本章では Lattice Diamond にバンドルされている論理シミュレータ Active-HDL を使用して論理シミュレーションを実行する方法や、その他のシミュレータを使用する場合のライブラリの作成方法等について説明します。

図 19-1. シミュレーションの実行手順



各作業の実行には、Active-HDL の GUI 上で行う操作による方法と、コンソールにコマンドを入力する方法があります。後者には個別にコマンドを対話的に入力する場合と、事前に用意するスクリプトで一連のコマンドを一挙に実行する方法があります。

19.1 Simulation Wizard を使用したシミュレーションの実行

Lattice Diamond には、HDL ソースのコンパイル等を行うスクリプトを作成するためのツール [Simulation Wizard] が用意されています。本節ではその使用方法について記述します。

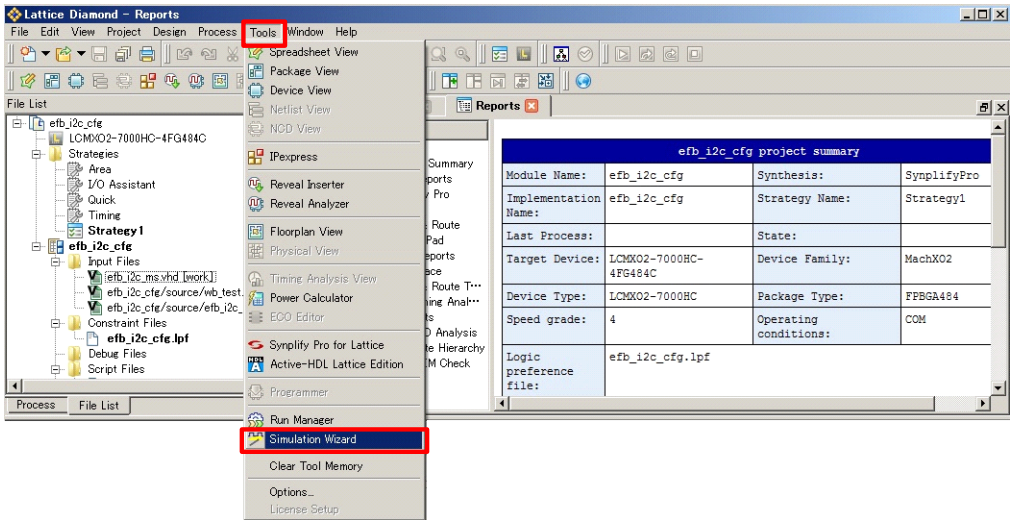
Diamond 2.1 以降では 4 ステップ全てが自動で実行されます。波形表示する信号の指定も含めてユーザが独自にスクリプトを作成して実行する場合は、後述する最初の 2 ステップのみを実行する指定をすれば、効率の良い繰り返し作業ができます。以下、個別に詳細説明をします。

19.1.1 Simulation Wizard の起動とコンパイルスクリプトの作成

Simulation Wizard は、ツールバー上のアイコン をクリックするか、メニューバーから [Tools] => [Simulation Wizard] の順に選択すると起動します。

起動した Simulation Wizard では、まず Simulation Wizard の用途についてのメッセージが表示されます。設定する項目はないので、ウインドウ右下の [Next>] ボタンをクリックし次へ進みます。次はシミュレータのプロジェクト名とフォルダパスを設定するウインドウが開きます。ここで適当なプロジェクト名とフォルダを選択してください。

図 19-2. Simulation Wizard の起動



*****----------

・ Mentor Graphics 社の ModelSim がインストールされており、かつ環境設定で ModelSim のパス設定が行われていれば、ここでシミュレータとして ModelSim を選択することもできます

*****----------

図 19-3. シミュレーションプロジェクトのパス設定

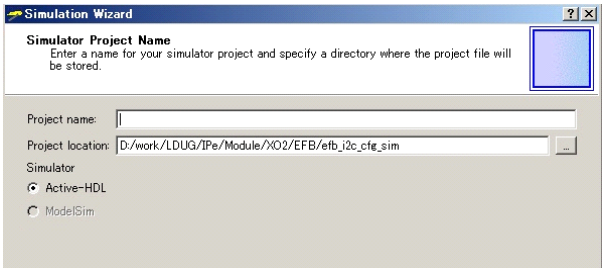
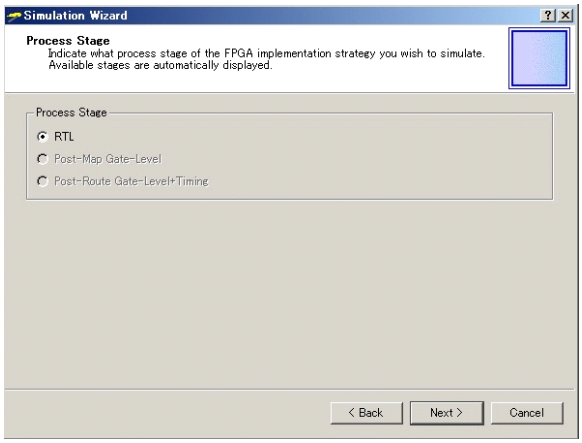


図 19-4. 実行するシミュレーション内容の選択



次へ進むと、シミュレーション内容の選択を行います（図 19-4）。選択肢とシミュレーション内容の対応は、表 19-1 のようになります。

表 19-1. Process Setup 設定とシミュレーション内容

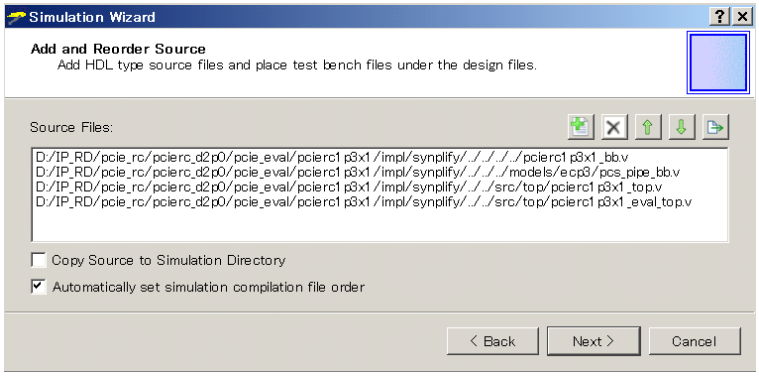
Process Setup	シミュレーション対象	遅延情報（sdf）
RTL	RTL の HDL ソース	なし
Post-Map Gate-Level	Map Design 後のネットリスト	なし
Post-Route Gate-Level+Timing	Place & Route 後のネットリスト	あり

Post-Map Gate-Level と Post-Route Gate-Level+Timing は、それぞれのシミュレーション実行に必要なネットリストが Lattice Diamond で生成されている場合のみ選択できます。

シミュレーション内容を選択すると、次はシミュレーションに使用するソースの選択を行います（図 19-5)。選択ウインドウが開くと、デフォルトでシミュレーション内容に対応した HDL ソースが選択された状態になっています。RTL シミュレーションの場合はプロジェクトにインポートされているソースが、ネットリストを使用するシミュレーションの場合は、対応するネットリストが表示されています。

シミュレーションの実行には、これにテストベンチを追加する必要があります。ウインドウ上の ボタンをクリックすると、HDL ソースを選択するウインドウが開きますので、必要なテストベンチの HDL ソース全てを選択しインポートします。

図 19-5. RTL ファイルリスト表示



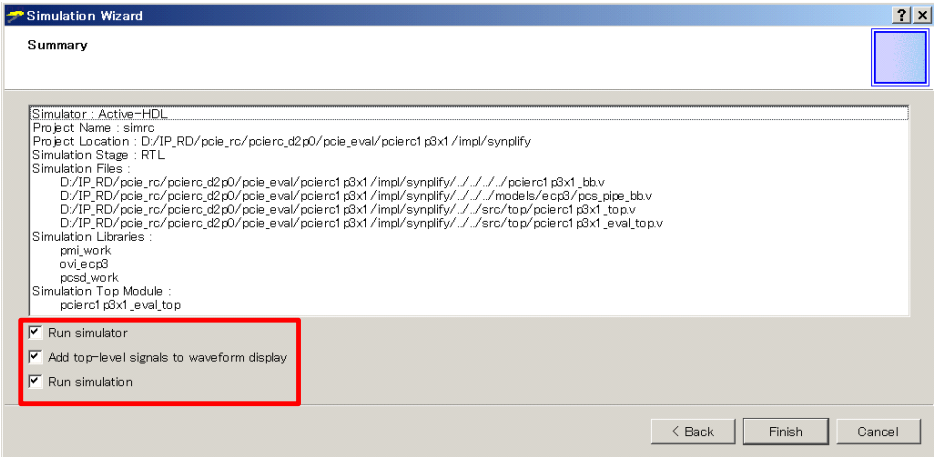
第 2 章「プロジェクト管理」の **2.6.6 節**に示すように、テストベンチなどシミュレーションのみに用いるファイルを Diamond プロジェクトにインポートすることができます。この場合、図 19-5 に示すソースファイル一覧にはそうしたシミュレーション用ファイル一式も含まれますので、毎回新たにインポートする手間が省けます。

VHDL の場合は、コンパイルソースがリストの上にくるようにファイルの並び順を変更します。並び替えるソースを選択し、 や ボタンをクリックして並び順を変えます。

[Next>] ボタンをクリックすると、"Parse HDL files for simulation" というウインドウが表示されます。さらに [Next>] をクリックします。最後にこれまでの設定の確認画面になります。内容に問題がなければ [Finish] ボタンをクリックします。

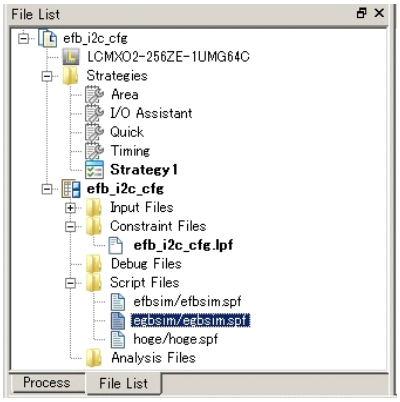
Diamond 2.1 以降では、赤枠の 3 項目がデフォルトとして全てチェック入りで表示されます。下二つが追加になったオプションです。

図 19-6. シミュレーション設定確認ウインドウ



[Run Simulation] にチェックが入っている状態で [Finish] をクリックするとシミュレータが起動し、インポートしたソースのコンパイルが行われます。チェックが入っていない場合は、コンパイルスクリプトの作成のみが行われます。作成されたスクリプトは、Lattice Diamond の File List ウインドウに自動的にインポートされます。

図 19-7. Simulation Wizard で作成されたコンパイルスクリプト



一度スクリプトを作成した後は、コンパイルする対象の HDL ソースファイルに変更がなければ Simulation Wizard を起動する必要はありません。以降は File List ウインドウ上のスクリプトをダブルクリックすれば、図 19-6 のウインドウが起動しますので [Finish] ボタンをクリックすればチェックのあるボックスの項目が自動実行されます。

"Add top-level signals to waveform display" にチェックが入っている状態で [Finish] をクリックするとシミュレータの波形表示ウインドウにテストベンチのトップレベル信号が自動的に取り込まれて表示されます。

"Run Simulation" にチェックが入っている状態で [Finish] をクリックするとシミュレータが自動的に実行されます。実行時間も自動的にツールが設定します。

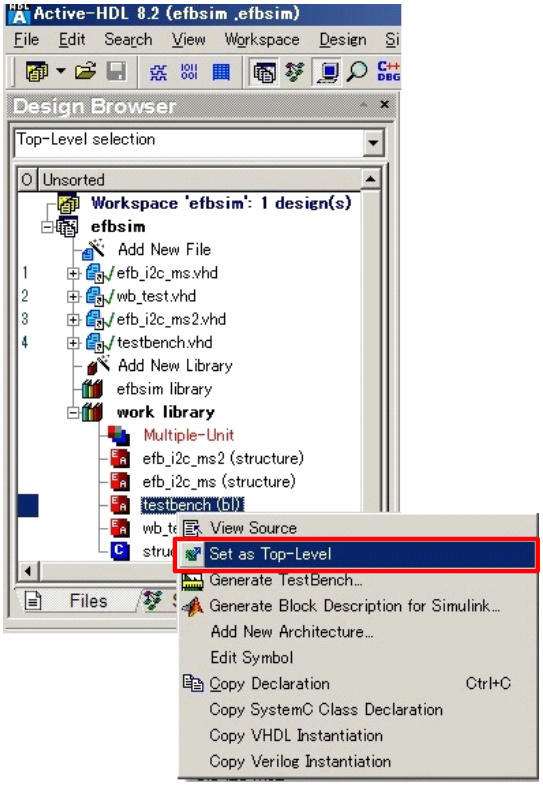
19.1.2 GUI でのシミュレーション実行手順

19.1.2.1 シミュレーションの準備と初期化

図 19-6 の左下オプション 3 つがすべて非選択状態になっている状態で [Finish] をクリックしたり、単独で Active-HDL を起動したりする場合には本アクションが必要です。

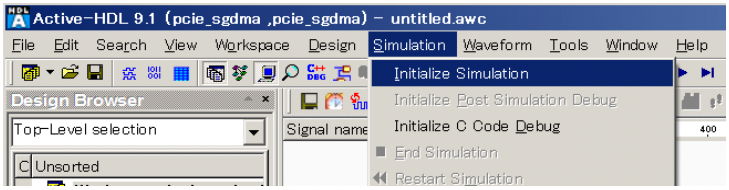
まず、プロジェクトを構成するソースファイルを全てインポートしてコンパイルしたソースの中から、最上位階層になるソースを選択します。コンパイルされたモジュールは、左上のウインドウの［Files］タブの［work library］ツリーに表示されます（図 19-8）。この中から、最上位階層に指定するモジュールを右クリックします。これで表示されるメニューの中から［Set as Top-Level］を選択すると、自動的にこのモジュール以下の階層の構築が行われます。

図 19-8. GUI 上でのトップモジュールの指定



次にシミュレーションの初期化を行います。メニューバーの Simulation から［Initialize Simulation］を選択します。インクルードするライブラリなど、シミュレーションの実行に問題があると初期化は失敗しますので、それを解消して次に進みます。

図 19-9. GUI 上でのシミュレーションの初期化



19.1.2.2 波形表示する信号の選択

信号の選択を行うには、Active-HDL のツールバーからアイコン  をクリックして波形 Viewer を起動します。

Viewer 上で右クリックすると表示されるメニューから［Add Signals］を選択すると、表示する信号の選択ウインドウが起動します。ウインドウの左側には、シミュレーション対象のインスタンスが表示されています。この中からインスタンスを選択すると、そのインスタンスのポートや信号名がウインドウの右側に表示されます。波形を表示させたい信号を選択後、ウインドウ右下の［Add］ボタンをクリックすると、選択した信号名が波形 Viewer に表示されます。

図 19-10. 波形 Viewer の起動

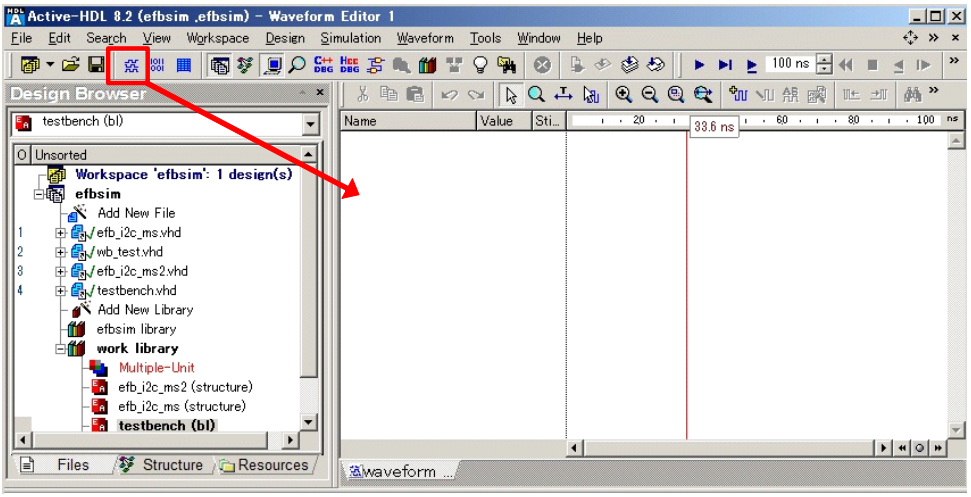
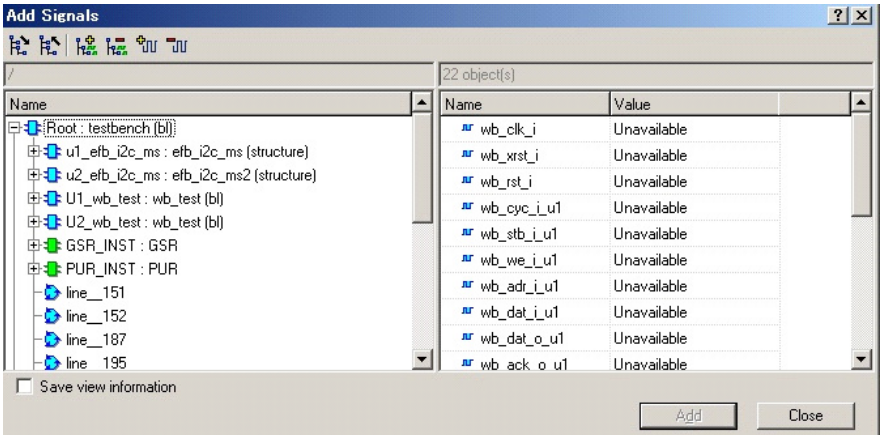


図 19-11. 波形表示信号の選択ウインドウ

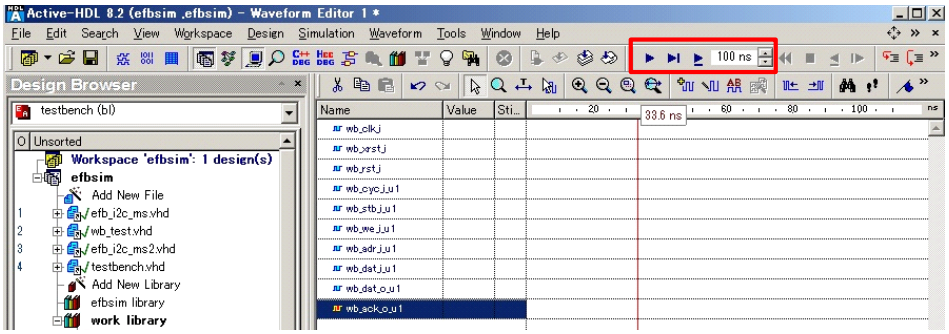


19.1.2.3 シミュレーションの実行

シミュレーションを実行するには、Active-HDL のツールバーからアイコンをクリックします。シミュレーション実行アイコンは 3 種類あり、それぞれ以下のような動作をします。

- ：シミュレーション開始実行（ ■ をクリックするまで継続）
- ：現在のシミュレーション時間から、指定した時間まで実行
- ：現在のシミュレーション時間から、右側の欄で指定した時間の間実行

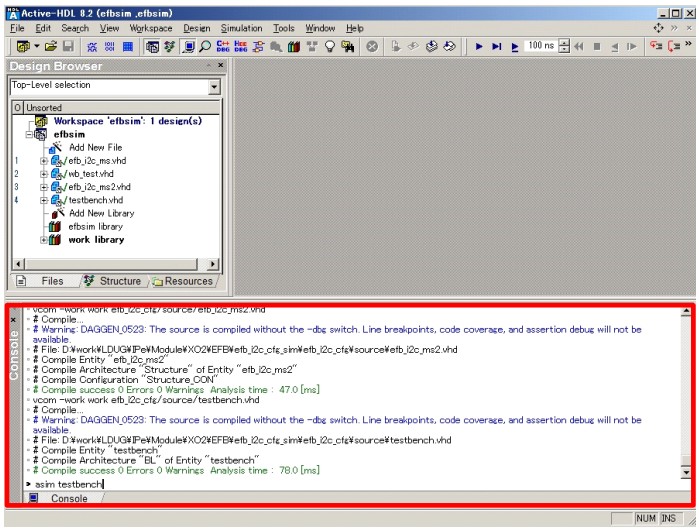
図 19-12. シミュレーション開始アイコン



19.1.3 コマンド入力によるシミュレーション実行

コマンド入力でシミュレーションを行う場合、以下に紹介するコマンドを Active HDL の下部のコンソールに入力します。

図 19-13. Active-HDL のコンソール表示例



19.1.3.1 シミュレーションの初期化

シミュレーションの初期化は、以下のコマンドで行います。

-- コマンド入力ルール

asim［最上位階層のモジュール名］［オプション］

オプション

・時間単位の指定

-t［時間単位］

・sdf ファイルと適用する値（min/typ/max）の指定 - 実負荷遅延 SIM の場合のみ

-sdfmax/min/typ モジュール名 =[sdf ファイル名]

-- コマンド入力例

```
asim testbench
asim testbench  -t 1 fs  -sdfmax U1_module=example.sdf
```

19.1.3.2 波形表示する信号の選択

信号の選択は、以下のコマンドで行います。

-- コマンド入力ルール

add wave [信号名 /port 名 ※]

※ 階層は [/] で区切って記述します。アスタリスクを使用することもできます。

-- コマンド入力例

```
add wave testbench/signal1
add wave *
add wave testbench/U1_test/*
```

19.1.3.3 シミュレーションの実行

シミュレーションの実行は、以下のコマンドで行います。

run [シミュレーション時間]

-- コマンド入力例

```
run 100 us
```

19.1.3.4 スクリプトの実行

以上で紹介したコマンドをスクリプトファイルに記述し、Active HDL のコンソールでそのスクリプトの実行コマンドを入力することで、コマンド入力を省略することもできます。

-- スクリプトファイル記述例

```
asim testbench
add wave *
add wave testbench/U1_test/*
run 300 us
```

-- コマンド入力ルール

do [スクリプトファイル名]

-- コマンド入力例

```
do sample_script.do
```

19.1.4 ECP5 のシミュレーション実行

ECP5 をターゲットとするインプリメンテーションで SERDES（PCS）を用いる場合のシミュレーション実行についての留意点・例を補足説明します。

Clarity Designer でモジュールを生成・プランニングしますが、リセットシーケンスを実行するソフトロジックのモジュール”RSL”が必ず PCS と同時に生成されます。これは生成する（トップ）モジュールの言語指定に拘わらず、必ず Verilog で、ファイル名は <PCS モジュール名 >_rsl.v です。

まず、RSL をコンパイルするコマンドは以下のようにします。リセットシーケンスは実シリコンの動作としては数十 msec 以上の規定時間を待つためのカウンタが幾つか含まれています。これらはシミュレーション時間的には許容で着ない程度に長いため、短縮するディレクティブを明示します。二つの方法があります。

①コマンドライン（スクリプト内）： vlog -dbg +define+RSL_SIM_MODE <WORK_DIR>/<PCS mod>_rsl.v

②パラメータ指定ファイル等を用意して‘define RSL_SIM_MODE を記述

ここで、<WORK_DIR> はスクリプトやコマンドを実行する作業フォルダからの相対パス、或いはフルパス指定です。

次に、シミュレーション実行時のライブラリ指定方法例です。<tb_top> はテストベンチのトップモジュール名を示すものとします。

(Verilog) asim -L ovi_ecp5u -L pmi_work -L pcsc_work +access +r <tb_top>

(VHDL) asim -L ecp5u -L pmi_work -L pcsc_work +access +r <tb_top>

”ovi_ecp5u” や ”ecp5u” は ECP5 の Verilog/VHDL ライブラリ、”pcsc_work” は PCS を含む場合に必要となるコンパイル済みライブラリです。”+access +r” はモジュール内部（下層）のノードにアクセス可能にするオプションです。

19.2 シミュレーション・ライブラリのコンパイル

Lattice FPGA 固有の機能ブロック（ブロックメモリ、PLL 等）を使用している場合、バンドル版以外のシミュレータではこれらの機能ブロックのためのライブラリが必要になります。

バンドル版の Active HDL では、全ての Lattice PLD のコンパイル済みライブラリがインストールされているので、ライブラリを生成する必要はありません。また、シミュレーション対象のソース内にブロックメモリや PLL 等の機能ブロックが含まれていない場合は、ライブラリ作成の必要はありません。

19.2.1 ライブラリのコンパイルが必要とされるケース

バンドル版の Active HDL 以外を使用する場合でも、デバイス固有のマクロを使用しない RTL シミュレーションの場合はライブラリを作成する必要はありません。しかし、以下の条件に 1 つでも当てはまる場合は、ライブラリのコンパイルが必要になります。

1. Map Design/Place & Route Design19.1.4 プロセス実行後に Lattice Diamond が出力したネットリストを使用するシミュレーション。
2. IPexpress で生成したマクロのソースを使用した RTL シミュレーション。
3. デバイス固有のマクロを使用した RTL シミュレーション

19.2.2 シミュレーションライブラリのパス

Lattice Diamond をインストールすると、以下のフォルダにシミュレーションライブラリの HDL ソースもインストールされます。

[Lattice Diamond Install Path]¥cae_library¥simulation¥[モデルタイプ ※1]¥[デバイスファミリ]¥src

-- 例えば Lattice Diamond 3.3 をデフォルトパスにインストールした場合、MachXO2 の VHDL ソース

C:¥lscc¥diamond¥3.3¥cae_library¥simulation¥vhdl¥machxo2¥src

※1: VHDL or Verilog or blackbox : EBR や PLL 等は VHDL または Verilog、PCS や JTAG モジュールは blackbox（19.1.4 項関連記述）

19.2.3 ライブラリ名

ライブラリのソースファイルをコンパイルする際は、ライブラリ名を表 19-2 のように設定します。

表 19-2. コンパイルライブラリ名

デバイス	Verilog ライブラリ名	VHDL ライブラリ名
LatticeXP2	ovi_xp2	xp2
MachXO2	ovi_machxo2	machxo2
LatticeECP3	ovi_ecp3	ecp3
MachXO3L	ovi_machxo3	machxo3
ECP5	ovi_ecp5u	ecp5u

19.2.4 ライブラリの呼び出し例

シミュレーションライブラリを呼び出す記述例を VHDL と Verilog それぞれについて示します。本ガイドでの主な記述目的である、Diamond から Simulation Wizard によってシミュレータを起動する場合は特に意識する必要はありませんが、Active-HDL を単独で立ち上げてプロジェクト生成・管理、シミュレーション実行する場合には、以下に留意してください。

19.2.4.1 VHDL

MachXO2 の場合の例を以下に示します。デバイス固有のマクロをインスタンスする場合は必ず宣言する必要があります。他のファミリの場合も同様です。

--VHDL 記述例（合成対象のモジュール記述に含める場合）

```
-- synopsys translate_off
library MACHXO2;
use MACHXO2.components.all;
-- synopsys translate_on
```

また、Verilog ビヘービア記述のシミュレーションマクロを呼び出す必要があるデザインの場合は、以下も含めます（MachXO2 の例）。

```
library ovi_machxo2;
use ovi_machxo2.all;
```

19.2.4.2 Verilog HDL

Active-HDL の GUI から対象ライブラリをインクルード設定しても良いのはもちろんですが、ここでは MachXO2 の場合に do スクリプト内で定義する例を示します。他のファミリでも同様にできます。

//Verilog RTL の do スクリプト内記述例

```
set LatticeTool C:/lscc/diamond/<version_number>
set XO2_LIB $LatticeTool/cae_library/simulation/verilog/machxo2
…
alog -y $XO2_LIB [< 相対パスとフォルダ名 >/RTL ソースファイル .v] [（同様に全てのリスト）]
asim -L ovi_machxo2 +access +r <test-bench top module 名 >
```

19.2.4.3 その他固有の留意点

MachXO2 など一部デバイスファミリでは、テストベンチで以下に示すデバイス初期化マクロに相当する GSR と PUR のインスタンス記述を含める必要があります。特に VHDL ではインスタンス名をここに記述する例の通りでないと期待動作しませんのでで、留意してください。

GSR の ″外部リセット入力信号名 ″ を実デザインの信号名に置き換えます。Low Active が有効ですので、極性に注意します（High Active のリセット入力であれば論理反転して記述）。

--VHDL 記述例（テストベンチ内）

```
-- コンポーネント宣言部
COMPONENT GSR
PORT（
    GSR: IN std_logic
 ）;
END COMPONENT;

COMPONENT PUR
PORT（
    PUR: IN std_logic
 ）;
END COMPONENT;
…
Begin  -- モジュールボディ内
…
```

```
GSR_INST: GSR
  port map （
    GSR => [ 外部リセット入力信号名 ]
  ）;

PUR_INST: PUR
  port map （
    PUR => c_vcc
  ）;

// Verilog HDL 記述例（テストベンチ内）
PUR PUR_INST （1'b1）;
GSR GSR_INST （< 外部リセット入力信号名 >）;
```

19.3 改訂履歴

Ver.	Date	page	内　容
3.3	Mar. 2015	--	第 19 章に変更（内容に変更はなし）
3.3.1 (3.3 rev1.1)	June 2016	8	19.1.4 ECP5 シミュレーション実行について補足を追記

--- *** ---