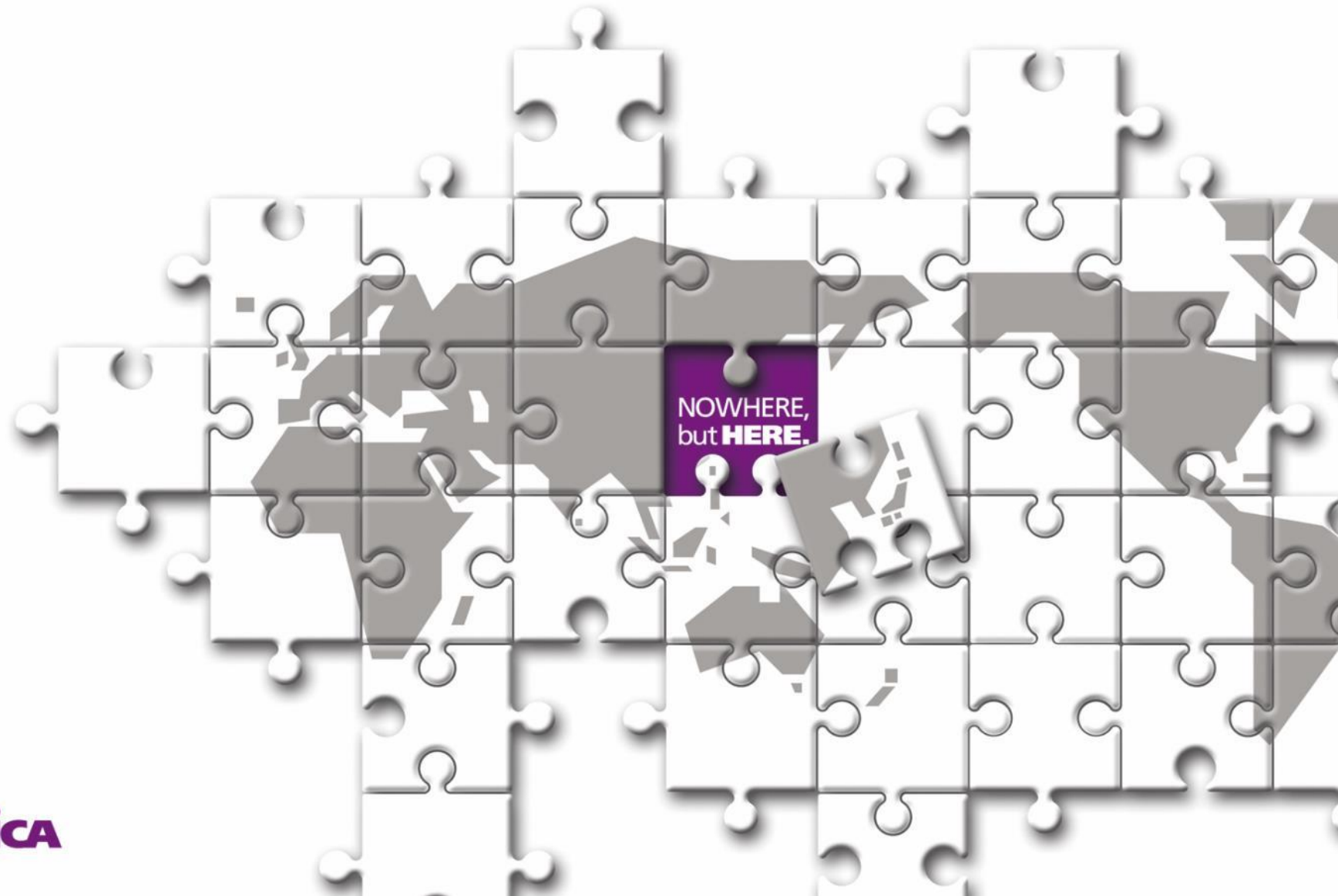


ModelSim LE DO マクロ ユーザーガイド Rev.0.4



はじめに

- 本ガイドは『[do マクロ](#)』を用いる RTL シミュレーションについての詳細です
 - 『[マクロ](#)』とは一連の実行コマンド ([TCL スクリプト](#)) を羅列した[テキストファイル](#)です
 - ✓ 拡張子は慣例に従って (便宜上) “.do” とします
 - 下に記述する GUI ベースの手法に比較して大幅に作業効率が向上します
 - ✓ RTL デバッグ・検証では一般的に繰り返し実行が必須であり、特に効果があります
 - ✓ 再利用性に優れます
- 次の手法に対して、以下のような種々の点から本 UG の手法を推奨します
 - 主に GUI 上でのマウス操作によるプロジェクト生成からシミュレーション実行まで
 - ✓ マウス操作がやや煩雑です
 - ✓ マウス操作のみでは完結せず、キーボード入力を伴うステップもあります
 - ✓ 手順詳細については別途『[ModelSim_LE_GUI_JUG](#)』をご参照ください
 - Radiant / Diamond の統合機能 Simulation Wizard によるシミュレーション実行
 - ✓ 呼び出すごとにシミュレーションが初期化され、前回作業の継続・再開ができません
 - ✓ 波形表示する信号と順序が毎回デフォルトのみで、前回までに編集・保存した形式にするためには別途手順が必要です
 - ✓ 手順詳細については『[ModelSim_LE_for_old_version_tools](#)』をご参照ください

内容

NOWHERE,
but **HERE.**

- 事前準備 p. 4
- マクロを用いたシミュレーション手順の概要 p. 6
- ModelSim LE GUI の単独起動 p. 7
- init.do マクロ実行方法の確認 p. 9
- init.do マクロの記述（概要と詳細） p. 10
 - * ソースファイルのインポートについて
 - * ソースファイルのコンパイルについて
 - * init.do 実行の補足
- 波形表示リストを do マクロに書き出し、編集 p. 18
- 繰り返し実行用 do マクロの生成 p. 20
- シミュレーション再実行・停止・再開 p. 21
- 作業の終了 p. 23
- プロジェクトの再オープン p. 24
- タイミング（遅延）シミュレーション p. 25
- SERDES、暗号化 IP のシミュレーション p. 27
- まとめ p. 28
- 変更履歴 p. 29
- 補足 p. 30

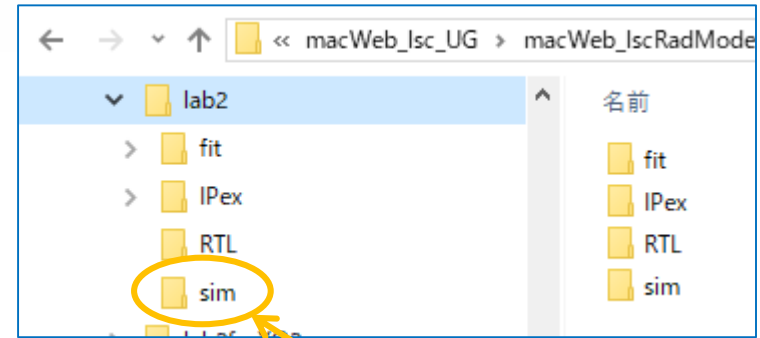
A1. ライブラリの名称 A2. “vlog” コンパイル・ディレクティブ A3. マウス操作による観測信号の指定方法

A4. Simulation Wizard 生成マクロの流用 A5. do マクロの変換 A6. 各種アイコン

事前準備：サンプルデザインについて

- 本ユーザガイドで例として用いるデザイン概要は以下の通りです
 - 動作クロックは外部クロック入力を PLL で2てい倍して使用
(ファイル名 "x2PLL.v" : ツール生成)
 - リセット入力は Low Active
 - 4ビットのフリーラン・カウンタ
 - 出力はカウンタ値が "1010" のときに High
 - ファイル名 "ug_lab2.vhd"
- ターゲットは iCE40 Ultra-Plus とします (Radiant 使用)
- 関連ソースファイル一式を zip で用意しています

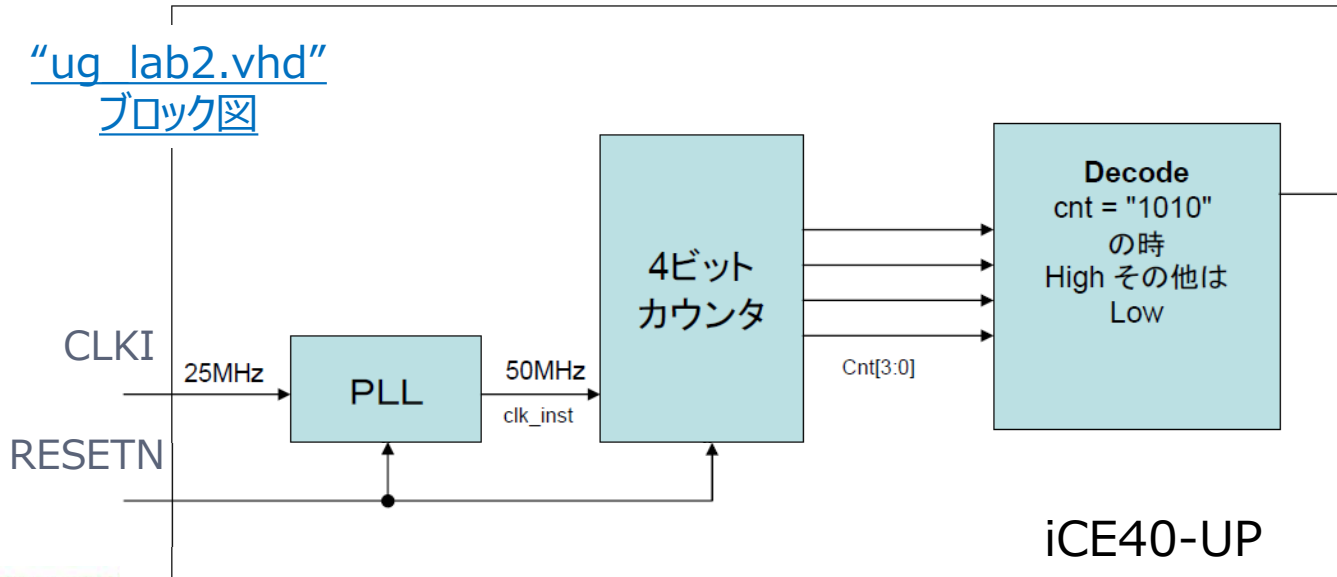
サンプルデザインのフォルダ構成



シミュレーション作業
のフォルダ

(TB 内インスタンス名 "u_ug_lab2")

"ug_lab2.vhd" ブロック図



DOUT

- * 作業フォルダ sim 下にはテストベンチ "tb_lab2.vhd"、及び以降に説明する do マクロを置きます
- * フォルダ fit はフィッティング用です



事前準備 (つづき)

- サンプルデザインを用いた演習のため、シミュレーション開始前に以下が準備してあるものとします
 - シミュレーション作業用フォルダを作成
 - サンプルデザイン (DUT) の RTL 一式 (zip ファイルに含まれる)
 - ☞ PLL はツールで RTL を生成し、フォルダ IPex 下に置くものとします (“x2PLL.v”)
 - ☞ Radiant に VHDL マクロ (ラッパー) 生成機能がないため Verilog を使用します
 - ☞ DUT トップはフォルダ RTL 下にあるものとします (“ug_lab2.vhd”)
 - DUT のテストベンチ (TB) を作成 (zip ファイルに含まれる)
 - ☞ シミュレーション作業フォルダと同じ sim 下にあるものとします (“tb_lab2.vhd”)
 - ☞ Verilog-HDL / VHDL の RTL / TB 記述法については別途資料をご用意していますので、FAE までお問い合わせください
- 最初に実行する do マクロ名を本ガイドでは “init.do” とします
 - 用意しているテンプレートの該当する箇所 (二行目) を、実際の作業フォルダ (ディレクトリ) に編集します
- 二回目以降のシミュレーション実行に使用する do マクロは基本的に二つです
 - “init.do” には初回のみ実行するコマンドが含まれますので、繰り返し実行時には必要なコマンド・シーケンスのみのマクロ (本ガイドでは “run.do”) を用います
 - 波形リストを表示するマクロ (本ガイドでは “waves.do”) を作成します
 - 次ページ以降で具体的に説明します



マクロを用いたシミュレーション手順の概要

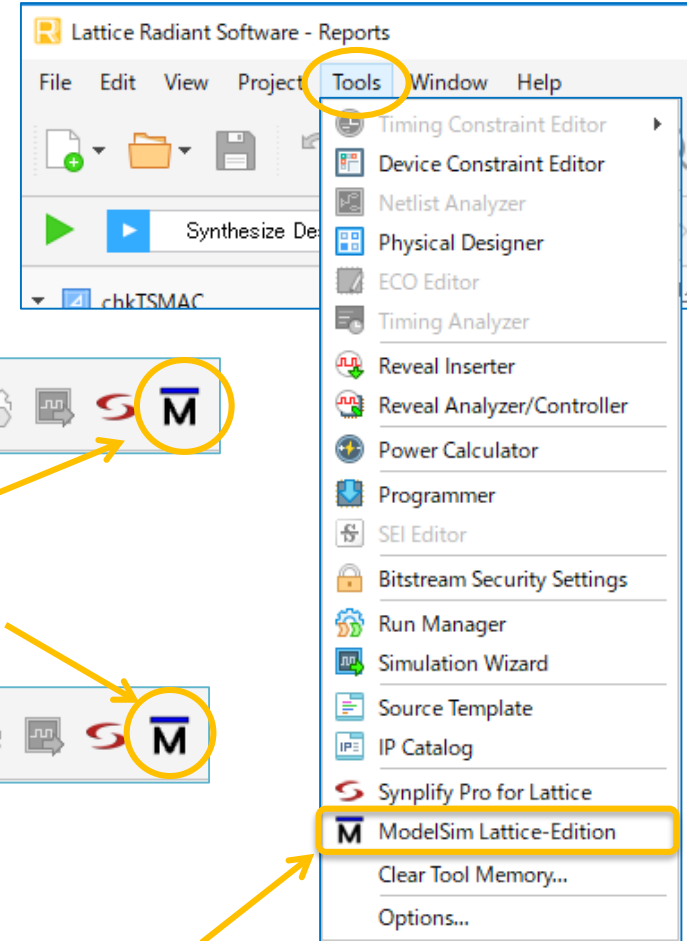
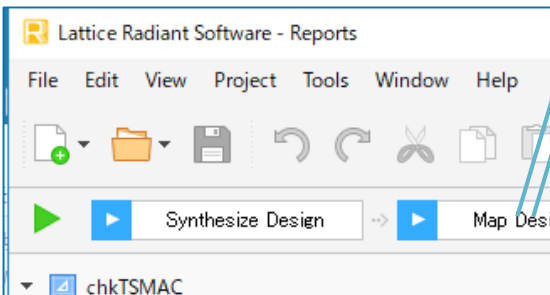
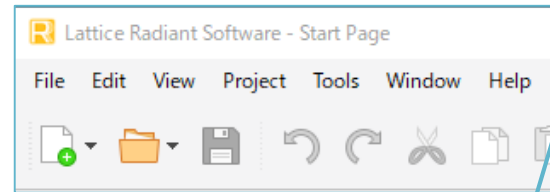
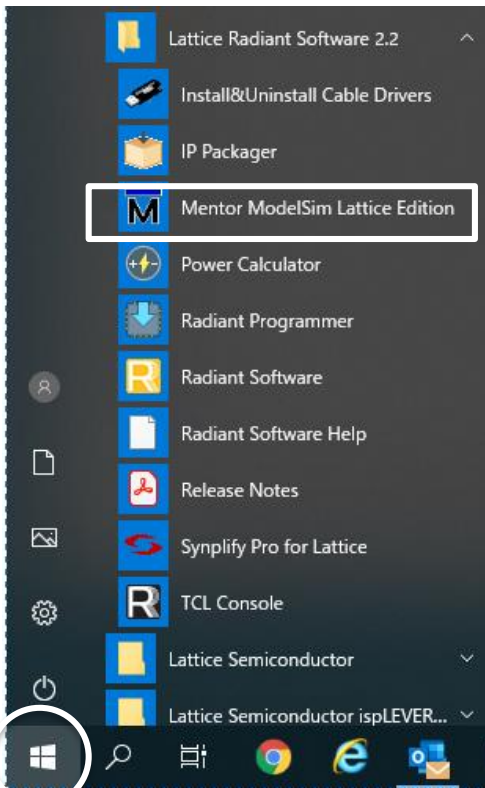
1. “init.do” マクロのテンプレートを元に、当該デザイン用に編集します
 - ✓ プロジェクトの作成から、RTL のコンパイル、sim 実行まで行います
2. ModelSim LE GUI を起動します
 - ✓ “init.do” がない状態で起動した場合、次のステップの前に作成します
3. “init.do” を実行します
 - ☞ エラーがなければシミュレーションが実行され、波形が表示されます
4. 適宜波形表示マクロを変更・保存し、シミュレーションを終了&クローズします
 - ✓ 表示波形（順序とフォーマット）をマクロ “waves.do” に書き出します
 - ☞ シミュレーション実行後に生成できます。シミュレーション終了前に行います
5. “init.do” を編集し、繰り返し実行用マクロ “run.do” を生成します
 - ☞ 最低限必要な部分のみを実行するバージョンのマクロです
6. デザイン（RTL）、TB や “run.do” “waves.do” などを適宜編集して繰り返し実行します
 - ☞ RTL / TB ソースを変更した場合は再コンパイル・ステップから行います

（注：本ガイドで使用している特定のファイル名は全て例）

ModelSim LE GUI の起動 (Radiant 2.2~)




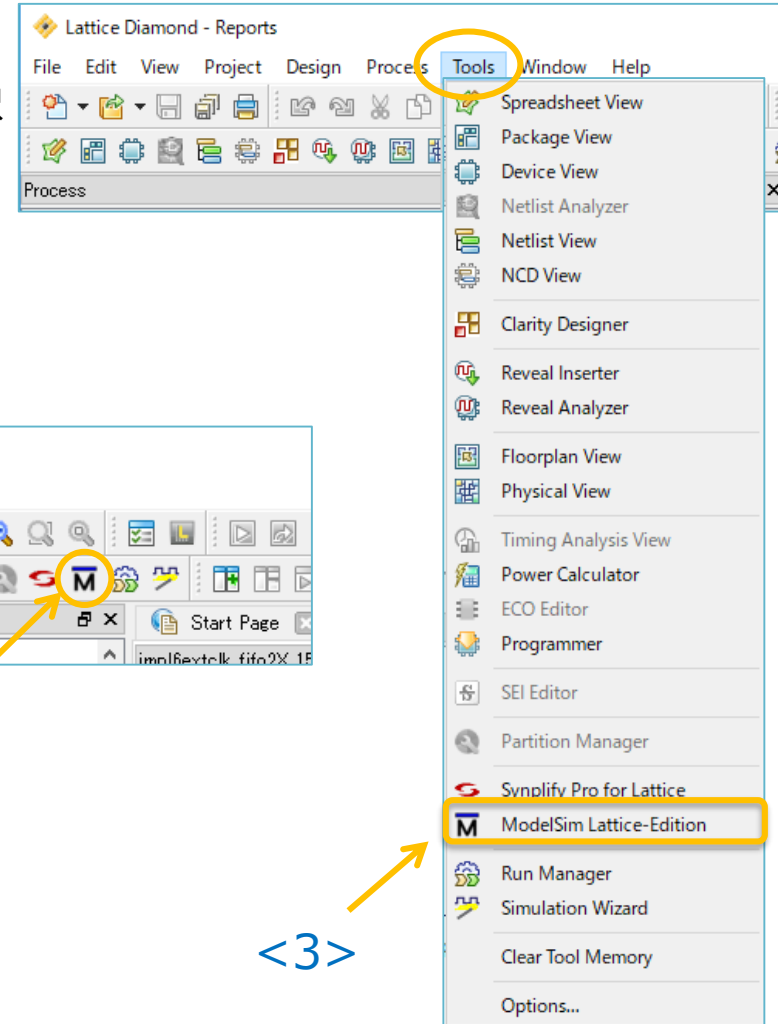
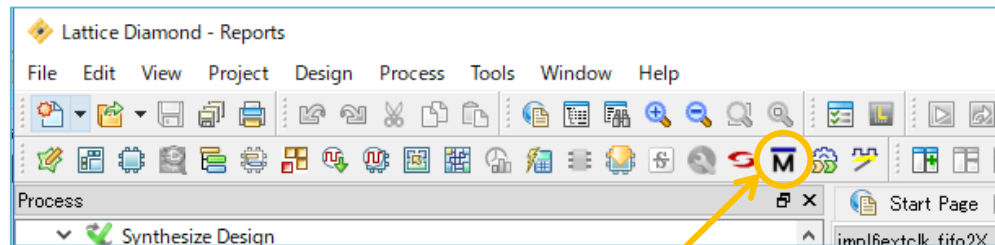
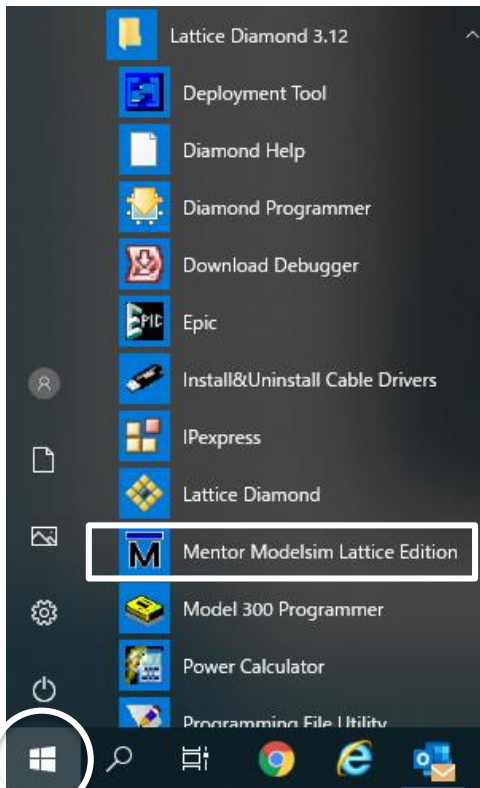
- 一般のアプリと同様に ModelSim LE GUI の単独起動方法は複数あります
 - Windows スタートメニューで **Mentor ModelSim Lattice Edition** を選択 (<1>)、又は
 - Radiant を起動後、トップのアイコン列からクリック (<2>)
 - Radiant を起動後、何らかのプロジェクトをオープンした状態で：
 - <3> **M** アイコンをクリック、または
 - <4> **Tools** メニュー → **ModelSim Lattice-Edition** を選択



ModelSim LE GUI の起動 (Diamond 3.12)

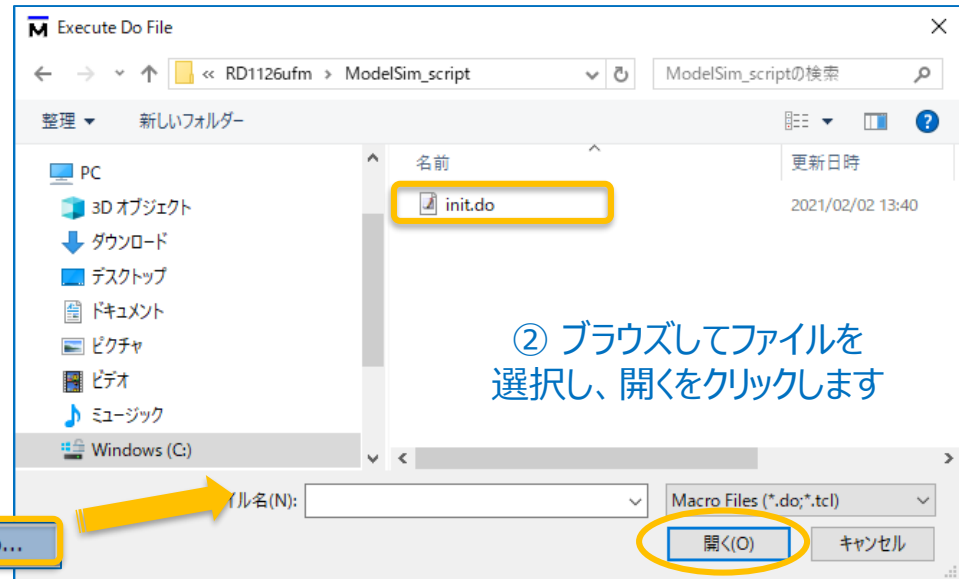
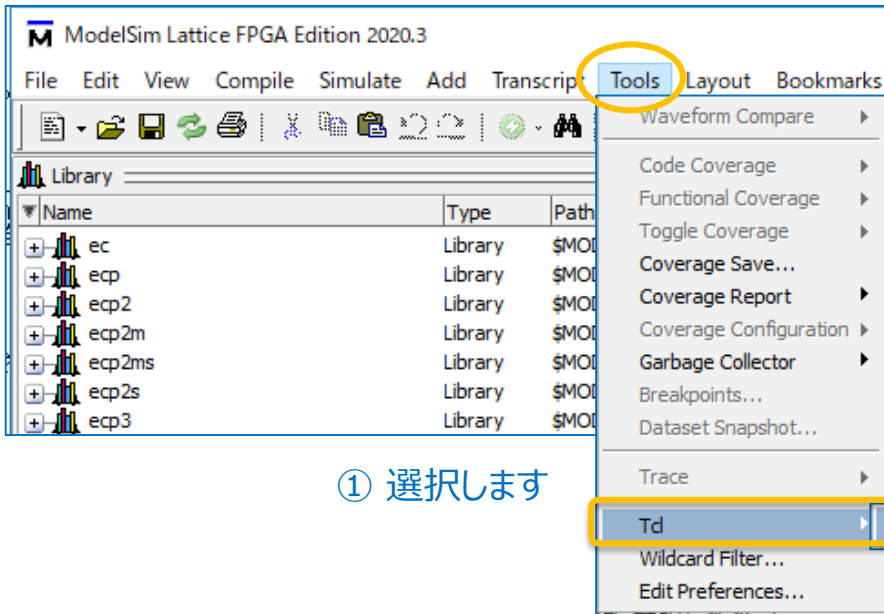


- 一般のアプリと同様に ModelSim LE GUI の単独起動方法は複数あります
 - スタートメニューで **Mentor ModelSim Lattice Edition** を選択 (<1>)、又は
 - Diamond を起動後は、何らかのプロジェクトをオープンした状態で：
 - <2>  アイコンをクリック、または
 - <3> **Tools** メニュー → **ModelSim Lattice-Edition** を選択



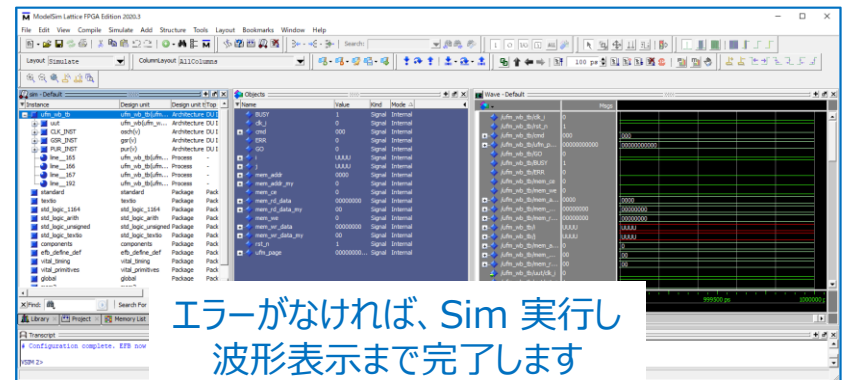
“init.do” マクロ実行方法の確認

- 選択対象は“プロジェクト生成から行うマクロ”です
 - ① メニューバーで Tools → Tcl → “Execute Macro...” を選択します
 - ② ファイルブラウザで所望のマクロファイルを選択して『開く』をクリックします
 - ✓ 右下の例は“init.do”（詳細例は次ページから）



☞ Transcript 窓に以下例のようなコマンドをタイプしても同等の作業ができます（ディレクトリ表記の区切りは“¥”ではなく“/”を使用します）

```
ModelSim> cd C:/macWeb_lsc_UG/sim
ModelSim> do init.do
```



“init.do” マクロ記述概要

- 右マクロ例は新規シミュレーション・プロジェクトの生成からシミュレーション実行までを行うための、最も簡易な例の命令・シーケンスを記述したテンプレートです
 - 必須ではないものを省略しています（③を除く）
- “カレントディレクトリ” は常に “*.mpf”（プロジェクト・ファイル）のあるディレクトリです
- 各ステップの詳細説明は、次ページから**サンプル・デザイン用**に記述しています
 - 記述例の **青字** は該当する文字列に書き換えることを意図しています
 - ✓ ディレクトリやソースファイル名など
 - 行頭の “#” はコメント行を示します

👉 do マクロの間違い易いところ（1）
* 作業フォルダ指定後プロジェクト作成前にディレクトリ変更し忘れる（①の‘cd’）

```
# “init.do”
① { set WORK_DIR C:/<work_directory>
    set RTL_DIR C:/<RTL_source_directory>
    cd $WORK_DIR
    #
    ② { project new $WORK_DIR <project_name>
        #
        ③ { project addfile $RTL_DIR/jk.vhd
            project addfile $RTL_DIR/mn.v
            project addfile tb_zz.v
            #
            ④ { vlib work
                vdel -lib work -all
                vlib work
                #
                # compile Verilog source files
                ⑤ { vlog $RTL_DIR/jk.vhd
                    vcom $RTL_DIR/mn.vhd tb_zz.vhd
                    #
                    # initialize simulation
                    ⑥ { vsim -L work -L ovi_iCE40UP <tb_name>
                        #
                        # set up signals to display
                        ⑦ { add wave /<tb_name>/<top_mod_name>/*
                            #
                            ⑧ { # run the simulation
                                run 100 us
```

“init.do” マクロ記述例①～③

① 作業ディレクトリを変数 WORK_DIR (任意名) として定義し、そのディレクトリに移動します。また、他の変数も適宜定義します

* “cd” はコマンド “change directory”

* “set” で変数を定義し、それを参照する際は “\$” をつけます (右例で WORK_DIR 定義以外は、参照される③の前であれば、どこに記述しても構いません)

② プロジェクトを作成します

* 作成するディレクトリと共にプロジェクト名を指定します

👉 ファイル *.mpf、*.mti の二つが生成されます

👉 右例で生成されるのは “mdlsim_uglab2.mpf” と “mdlsim_uglab2.mti” です

* “work” がデフォルトの作業ライブラリ名です。固有の名称も与えることができますが、意図しない動作を防ぐためにデフォルト名を使用することとします

③ ソースファイルを全てインポートします

* 相対パス記述でも良いですが、本例では変数を定義した上で絶対パス記述しています

👉 本ステップは必須ではありませんが、GUI 表示で確認できるという観点から、記述しておくことが望ましいです

作成するプロジェクト名

作成するディレクトリ

```
# "init.do"
set WORK_DIR C:/macWeb_lsc_UG/lab2/sim
cd $WORK_DIR
set RTL_DIR C:/macWeb_lsc_UG/lab2/RTL
set IPex_DIR C:/macWeb_lsc_UG/lab2/IPex
#
project new $WORK_DIR mdlsim_uglab2
#
project addfile $IPex_DIR/x2PLL/rtl/x2PLL.v
project addfile $RTL_DIR/ug_lab2.vhd
project addfile tb_lab2.vhd
#
```

“init.do” マクロ記述例④～⑤

- ④ 念のためプロジェクト内のコンパイル済み作業ライブラリを全てクリアしておきます

☞ 例えば二回目の実行時以降に何らかのオブジェクトを更新した場合、前回の作業ライブラリに含まれる、更新前のオブジェクトを呼び出してしまうことで、意図しないシミュレーション結果になることがあるため、それを防ぎます

- ⑤ RTL ソースファイルを全てコンパイルします

- * VHDL ソースは “vcom” コマンドです
- * Verilog ソースは “vlog” コマンドです
- * 相対 or 絶対パス指定で記述します
 - ☞ 定義済み変数を用いることができます
- * 単一コマンドで対象を単一ソース、或いは複数の対象ソースとします
- * 複数ソースを複数行に亘って記述する場合、行末に “¥” を記述します（コマンドが次の行まで継続することを指示します）
 - ☞ “¥” の後には空白文字を含め何もないことが必要です
- ☞ 他に種々コンパイル・オプションを指定する方法があります（pp.32-33）

```
④ { vlib work
      vdel -lib work -all
      vlib work
      #
      # compile VHDL source files
⑤ { vlog $IPex_DIR/x2PLL/rtl/x2PLL.v
      vcom $RTL_DIR/ug_lab2.vhd ¥
          tb_lab2.vhd
      #
```



“init.do” マクロ記述例⑥

⑥ シミュレーションの初期化

* ターゲットデバイスのライブラリを ‘-L’ で含めます（デバイス固有のマクロを含む場合）

☞ “固有マクロ” とは例えば OSC / PLL、EBR、EFB（MachXO2/3 シリーズ）などです

☞ “ovi_” 有は Verilog、無しは VHDL です

☞ ライブラリ名は p.31 をご参照ください（以下に含まれる “サブフォルダ名” が相当し、デバイス名はある程度類推できます）

C:/lsc/radiant/2.2/modeltech/lib

C:/lsc/diamond/3.12/modeltech/lib

☞ 混在言語の場合は Verilog ライブラリのみ指定で構いません

* SERDES や暗号化 IP（Radiant）、また特定のハードマクロを含むデザインでは “pmi_work” ライブラリも ‘-L’ 指定します

* 他のデバイスで PLL などをデザインでインスタンスしている場合、時間精度を指定する必要がある場合があります

☞ `vsim -t 10ps -L work ...`

時間精度指定

```
⑥ #  
# initialize simulation  
vsim -L work -L ovi_iCE40UP TB_LAB2
```

ICE40UP Verilog ライブラリ指定
（ターゲットデバイスのライブラリ）

↑
“tb_lab2.vhd” 内で
entity 宣言している
テストベンチ名



“init.do” マクロ記述例⑦～⑧

⑦ Wave 窓に波形表示する信号を指示します

- * 1行目はテストベンチ階層の全信号を波形表示する指定
- * 2行目は TB でインスタンスされる DUT トップモジュール階層の全オブジェクトを表示する指定

- ☞ ここではワイルドカード “*” を使用して全オブジェクトを指定していますが、個別に書くことも可能です (次ページ)
- ☞ 個別オブジェクト名を列記したものを (“waves.do” など) をここでは『波形表示マクロ』と呼んでいます)
- ☞ 三行目は、次ページから記述する方法で生成した “波形表示マクロ” を実行する場合の記述例です (ここでは、まだ生成していないのでコメントアウトしています)

```

⑦ { # set up signals to display
      add wave /TB_LAB2/*
      add wave /TB_LAB2/u_UG_LAB2/*
      #do waves.do
      #
⑧ { # run the simulation
      run 5 us
  
```

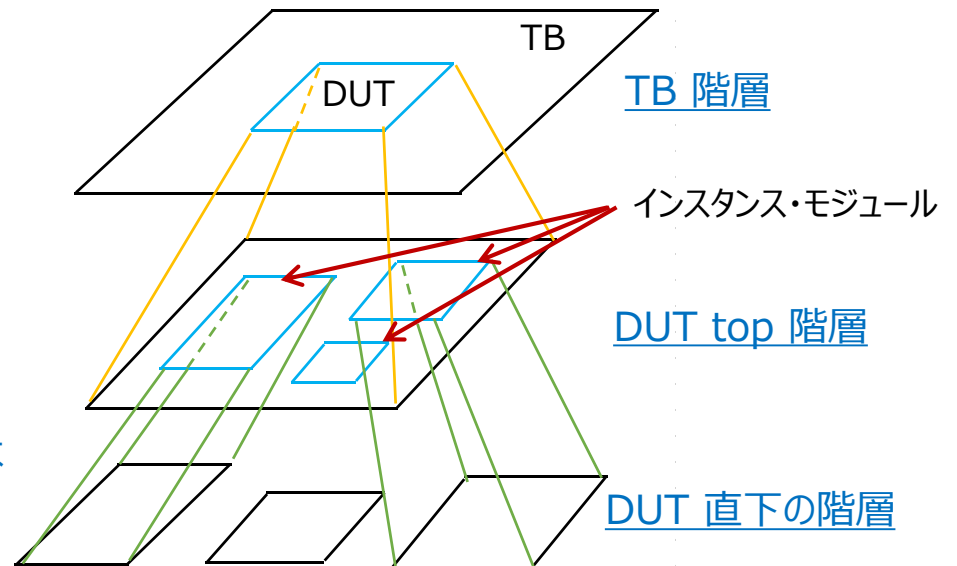
⑧ シミュレーション実行

- * TB 記述のシナリオを全て実行する場合は以下です :

`run -all`

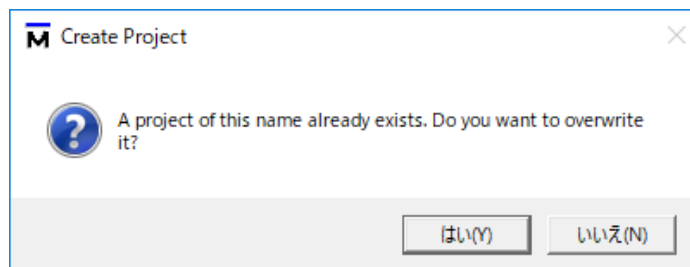
- ☞ 時間と単位の間スペースは無くても構いません

『階層 (scope) 』
論理シミュレータの実行では
常に “階層” に留意



注：“init.do” マクロ実行に関する補足

- init.do に修正を加えた後、①部から再実行する場合、カレントディレクトリにはプロジェクト・ファイル（*.mti *.mpf）が既に存在するので、右下のようなポップアップが表示されます
 - ☞ 『はい』をクリックして先に進みます
- P.9 で示したような Tool メニューから **init.do** を選択して進める方法ではなく、Transcript 窓でディレクトリ変更後に **do** コマンドで実行することも可能です
 - ☞ カレントディレクトリの確認は Transcript 窓で以下をタイプします（print working directory）
ModelSim> **pwd**



注：ソースファイルのインポートについて

- ソースファイル・インポート **addfile** 文の記述（③部）に関する留意点です：

* ファイルが指定フォルダに見つからない場合、GUI 表示では“Modified” 欄の表示が“… …” になります

☞ ソースファイルが期待通りにインポートできているかどうかの判別ができます

☞ 当該文を実行した時点ではエラーにならず、また何らかのメッセージも出ません（少なくともコンパイル・ステップ以降ではエラーになります）

☞ 前ページで“必須ではないが記述することを推奨する”と記述した理由の一つです

誤ったディレクトリ情報を含む do マクロ例
(addfile 関連部のみ)

```
#
set SRC_DIR C:/LSC_RDs/RD1126ufm/source/vhdl
set TB_DIR C:/LSC_RDs/RD1126ufm/testbench/vhdl
#
project addfile ../efb_define_def.vhd
project addfile $SRC_DIR/ipexpress/USR_MEM.vhd
project addfile $SRC_DIR/ipexpress/EFB_UFM.vhd
project addfile $SRC_DIR/ufm_wb_top.vhd
project addfile $TB_DIR/ufm_wb_tb.vhd
#
```

実行後

ファイルが見つからないことを示す


Name	Status	Type	Order	Modified
efb_define_def.vhd	?	VHDL	0	---
USR_MEM.vhd	?	VHDL	1	01/21/2015 05:45:12 ...
EFB_UFM.vhd	?	VHDL	2	01/21/2015 05:45:08 ...
ufm_wb_top.vhd	?	VHDL	3	01/21/2015 05:45:04 ...
ufm_wb_tb.vhd	?	VHDL	4	01/21/2015 05:45:14 ...

☞ do マクロの間違い易いところ（2）

* ソースファイルの所在ディレクトリが正しくない（特に相対パス記述）



注：ソースファイルのコンパイルに関して

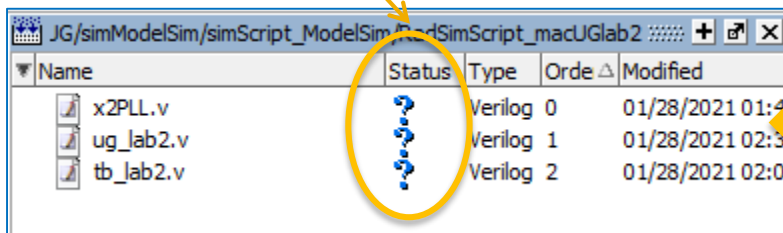
- ソースファイル・コンパイル `vlog` / `vcom` 文の実行（⑤部）に関する留意点です：
 - * コンパイル結果が PASS / FAIL に関わらず GUI のステータス・アイコンは  のままです
 - 👉 Transcript 窓のメッセージで判断します




do マクロ例（コンパイル関連部のみ）

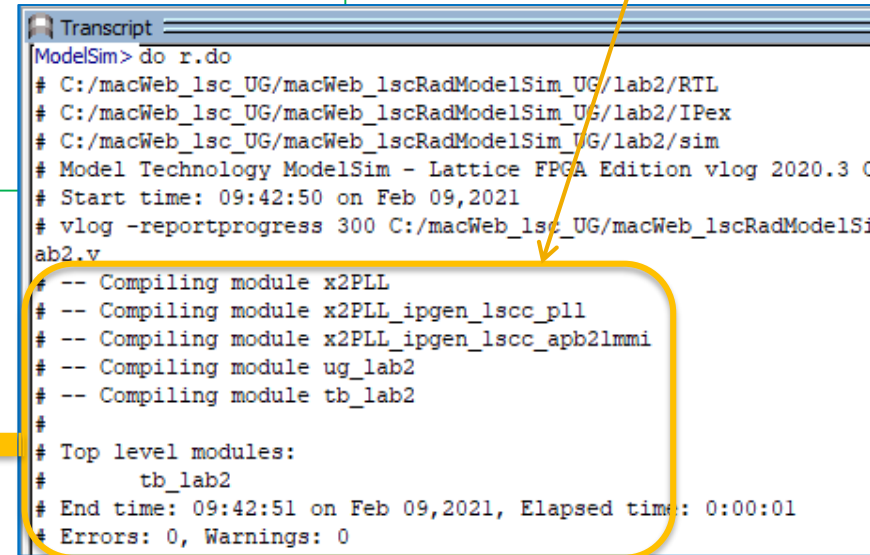
```
#
set RTL_SRC C:/macWeb_lsc_UG/macWeb_lscRadModelSim_UG/lab2/RTL
set IPex_SRC C:/macWeb_lsc_UG/macWeb_lscRadModelSim_UG/lab2/IPex
#set TB_DIR C:/macWeb_lsc_UG/macWeb_lscRadModelSim_UG/lab2/sim
#
(略)
#
vlog $IPex_SRC/x2PLL/rtl/x2PLL.v
vcom $RTL_SRC/ug_lab2.vhd tb_lab2.vhd
#
```

コンパイル結果は
Errors 0, Warnings 0

アイコンは変わりません



Name	Status	Type	Orde Δ	Modified
x2PLL.v		Verilog	0	01/28/2021 01:4
ug_lab2.v		Verilog	1	01/28/2021 02:3
tb_lab2.v		Verilog	2	01/28/2021 02:0




```
Transcript
ModelSim> do r.do
# C:/macWeb_lsc_UG/macWeb_lscRadModelSim_UG/lab2/RTL
# C:/macWeb_lsc_UG/macWeb_lscRadModelSim_UG/lab2/IPex
# C:/macWeb_lsc_UG/macWeb_lscRadModelSim_UG/lab2/sim
# Model Technology ModelSim - Lattice FPGA Edition vlog 2020.3 C
# Start time: 09:42:50 on Feb 09,2021
# vlog -reportprogress 300 C:/macWeb_lsc_UG/macWeb_lscRadModelSi
ab2.v
# -- Compiling module x2PLL
# -- Compiling module x2PLL_ipgen_lscc_pll
# -- Compiling module x2PLL_ipgen_lscc_apb2lmmi
# -- Compiling module ug_lab2
# -- Compiling module tb_lab2
#
# Top level modules:
#   tb_lab2
# End time: 09:42:51 on Feb 09,2021, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

波形表示リストを do マクロに書き出し

- 論理検証は RTL/TB の修正とシミュレーション実行の繰り返しですので、波形表示マクロを活用することによって容易に同じ表示形式にできます。以下の二通りの方法があります：

1. Wave 窓で順序の変更や信号の追加・削除、表示属性の指定等を行います
- 2 A. Transcript 窓で以下例を参考にしてタイプします（出力ファイルが “waves.do” の場合）

VSIM > **write format wave** waves.do

- 2 B. または ① メニューバーで File → Save Format... を選択します（アイコン  をクリックしても同じです）

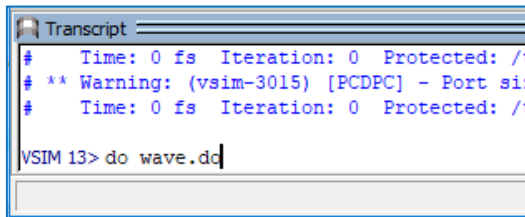
☞ メニューに “Save Format...” がない場合は Wave 窓内のどこかを一度クリックします

- ② “Save Format” ポップアップの Pathname 欄にパスとファイル名を入力後 OK をクリックします

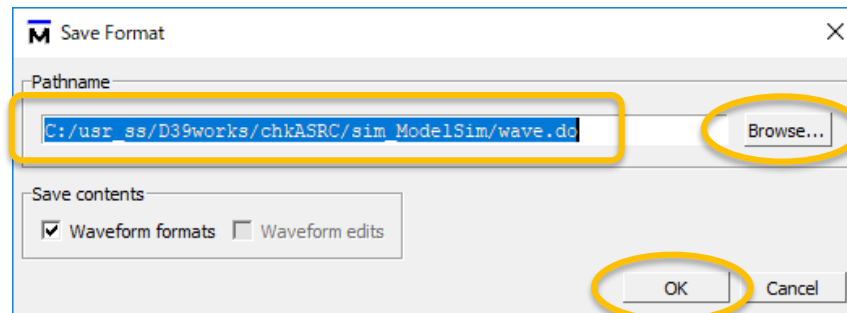
☞ デフォルト名 wave.do が自動的に表示されています（編集可）

☞ 繰り返し作業で波形表示を操作・変更した場合も、同様に操作します

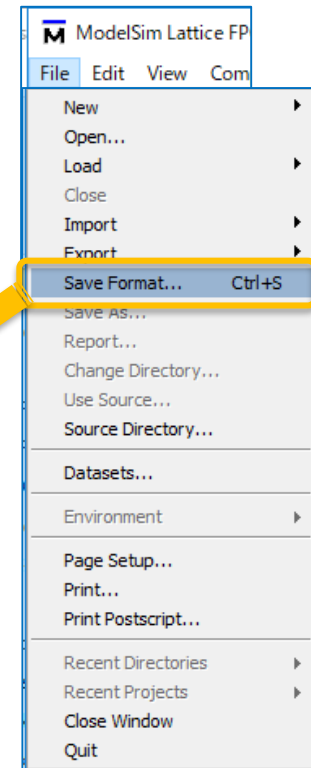
- p.20 に示す繰り返し実行用マクロで、保存した波形表示マクロを実行します



波形表示をロード（復元）
するコマンド入力



パスとファイル名を入力して OK をクリック
（必要に応じてブラウズする = 非推奨）





波形表示マクロの編集

- 書き出された波形表示マクロ [waves.do](https://www.macnica.com/tech/2018/07/20/waves-do/) をテキストエディタで編集する方法もあります
 - 既にある波形表示マクロを流用する場合などは、マウス操作をせずにこのアプローチが採用できます
 - 表示順の変更や属性の指定、信号の追加・削除が容易です
 - ☞ デザインの複雑さに依存して、テストシナリオごとに複数の波形表示マクロを使い分けて、検証作業を効率向上できます
 - 階層関係とインスタンス名に留意して信号名を記述します
 - ☞ 階層は "/" で区切ります (モジュール名ではなくインスタンス名であることに留意します)
 - Wave 窓左端のオブジェクト名欄は階層名も表示されるため冗長です。表示名を指定する場合は **label** を用います


```
add wave -noupdate -label counter /tb_lab2/u_ug_lab2/counter (例)
```
 - 表示波形の間にテキストで区切り行 (セパレータ) を挿入する場合の記述例を示します


```
add wave -divider "u_UG_LAB2"
```

■ その他書式指定の詳細はオンラインヘルプをご参照ください

"-hexadecimal" は多ビットオブジェクトを16進表記 (十進表記は "-decimal")

- メニューバーで [Help](#) → [Product Help](#)

編集・再利用の際はこれら行は必須ではないので無くても良い (十分に理解した後、数値等を編集して活用しても良い)

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate /tb_lab2/resetn
add wave -noupdate /tb_lab2/clki
add wave -noupdate /tb_lab2/u_ug_lab2/pll_lock
add wave -noupdate -hexadecimal /tb_lab2/u_ug_lab2/counter
add wave -noupdate /tb_lab2/dout
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {2748042630 fs} 0}
quietly wave cursor active 1
configure wave -namecolwidth 362
... (略)
update
WaveRestoreZoom {932462130 fs} {6110371730 fs}
```

生成されたマクロ [waves.do](https://www.macnica.com/tech/2018/07/20/waves-do/) 例



繰り返し実行用 do マクロの生成

- “init.do” を編集して繰り返し実行用マクロ “run.do” を作成します
 - ① の “cd ...” 行と②③はプロジェクト生成時にのみ必要なのでコメントアウトします
 - ✓ ① の使用する変数の定義行は有効なままにしておきます
 - ④ の最初の行の **vlib** work はコメントアウトします
 - ⑤ **vcom**、**vlog** ⑥ **vsim** ⑧ **run** は残します
 - ☞ ソースファイルに変更がなく、波形表示のみ再実行したい場合は⑥～⑧のみでも可（通常は TB / RTL の変更を伴うことが多いので③～⑧の繰り返し実行になります）
 - ⑦ の波形表示指定について：
 - ✓ **init.do** の “add wave /<tb_name>/*” 行などの一括指定部はコメントアウトします
 - ✓ 以下を追記します（**waves.do** を p.18 のように作業ディレクトリに保存した前提）

do waves.do

- 編集したマクロを、例えば “run.do” として保存します

- ☞ 保存先は作業（カレント）ディレクトリにします
- ☞ 右例はコマンド行のみを抜粋しています
- ☞ “#” マークを先頭にしてコメント行を適宜挿入することを推奨します

保存した “run.do” 例
(残すコマンド行のみ抜粋)

```
① { set RTL_DIR C:/macWeb_lsc_UG/lab2/RTL
    set IPex_DIR C:/macWeb_lsc_UG/lab2/IPex
    #
④ { vdel -lib work -all
    vlib work
    #
⑤ { vlog $IPex_DIR/x2PLL/rtl/x2PLL.v
    vcom $RTL_DIR/ug_lab2.vhd tb_lab2.vhd
    #
⑥ { vsim -L work -L ovi_iCE40UP TB_LAB2
    #
⑦ { do waves.do
    #
⑧ { run 10 us
```


シミュレーションの再実行

- 観測信号を追加した直後は、その時点以降しか波形が表示されないので、一旦シミュレーションを終了して再実行します

[1] RTL ソースファイルを変更している場合 ([2] の “RTL を変更している場合” でも適用可)

1 - 1. Transcript 窓に以下のコマンドを入力してシミュレーションを終了します

```
VSIM > quit -sim
```

☞ メニューバーで Simulate → End Simulation を選択しても同じです

1 - 2. Transcript 窓に以下のコマンドを入力して以下のように繰り返し実行用 do マクロを実行し、コンパイルからシミュレーション実行までを行います


```
ModelSim > do run.do
```

[2] RTL ソースファイルを一切変更していない場合

2 - 1. Transcript 窓に以下のコマンドを入力します

```
VSIM > restart -f -nowave (波形表示が同じなら restart のみ)
```

☞ 同じリスタート動作となる別の方法が二通りあります：

- ・ アイコン  をクリックするか、又は
- ・ メニューバーの Simulate → Restart を選択します

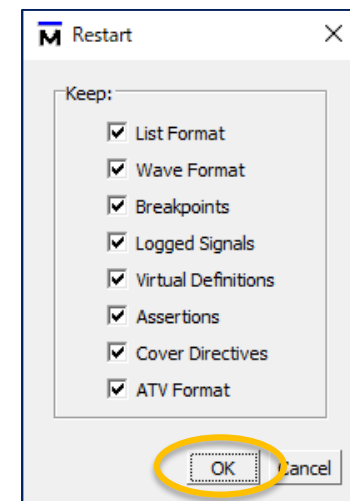
2 - 2. “Restart” ウィンドウがポップアップ (右図) しますので、OK をクリックします

☞ シミュレーションが初期化されます (シミュレーション実行前の表示に戻ります)

2 - 3. Transcript 窓に以下のようなコマンドを入力し、波形表示とシミュレーションを実行します


```
VSIM > do waves.do (波形表示マクロ waves.do を実行)
```

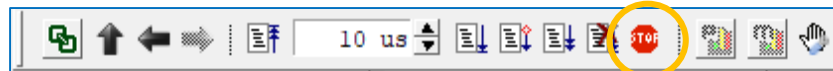
```
VSIM > run 10us (シミュレーションを 10us 実行)
```



Restart 確認窓
(OK をクリック)

シミュレーションの停止・再開

- シミュレーション実行の途中で中断したい場合はアイコン  をクリックします
 - TB での “\$stop” と同じ (Verilog)
 - ☞ 前ページの “quit -sim” (またはメニュー Simulate → End Simulation) による終了 (\$finish) とは異なります
 - ☞ シミュレーション時間が長くない場合はアイコンをクリックする前に終わります
 - 再開するには以下の例のように Transcript 窓に時間と共に run コマンドをタイプします
VSIM > run 10us



このアイコンで実行を中断

作業の終了

■ シミュレーションの終了:

- Transcript 窓に以下コマンドをタイプします

```
VSIM> quit -sim
```

☞ メニューバーで Simulation → End Simulation を選択しても同じです

■ プロジェクトのクローズ:

- Transcript 窓に以下コマンドをタイプします

```
VSIM> project close
```

☞ メニューバーの File → Close でも同じです (Close Window では終了しません)

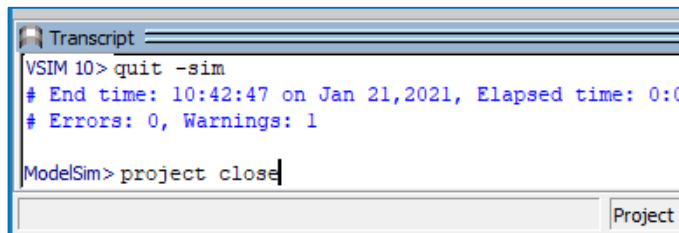
☞ 作業していたプロジェクトをクローズしないで ModelSim LE を終了した場合、その次に立ち上げると終了直前のプロジェクトを自動的にオープンした状態になることにご留意ください

■ ModelSim LE の終了:

- Transcript 窓に以下コマンドをタイプします

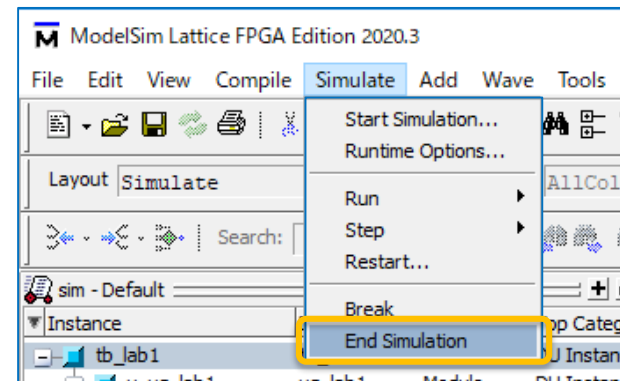
```
VSIM> quit
```

☞ メニューバーで File → Quit を選択しても同じです

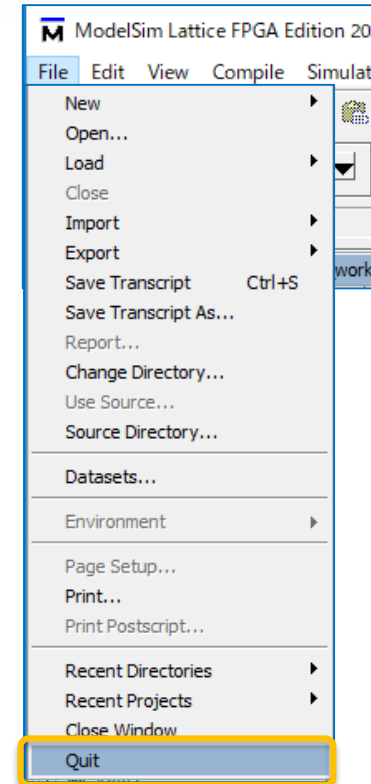


```
Transcript
VSIM 10> quit -sim
# End time: 10:42:47 on Jan 21,2021, Elapsed time: 0:0
# Errors: 0, Warnings: 1
ModelSim> project close
```

プロジェクトのクローズ




シミュレーションの終了メニュー



ModelSim LE の
終了メニュー

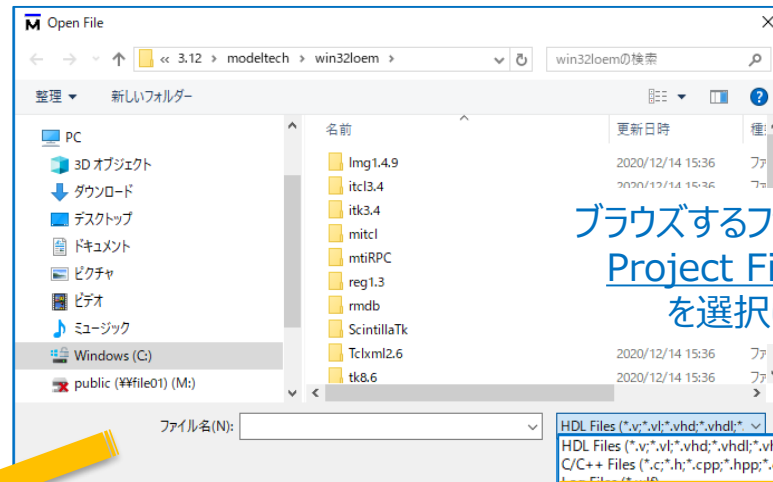
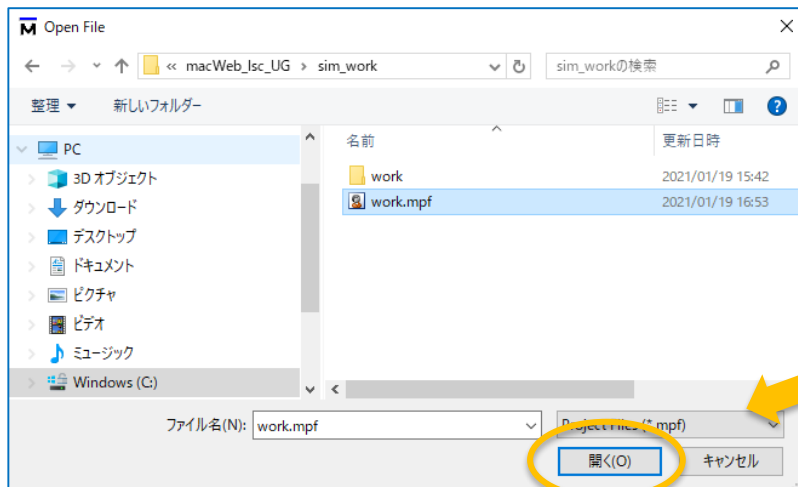
プロジェクトの再オープン

- File → Recent Projects で表示されるプロジェクトから選択します
 - 直近作業した 8 プロジェクト が候補としてリストされます
 - プロジェクト名は Radiant / Diamond 両環境が識別されずに、混在してリストされることにご留意ください
 - ☞ ツールが併存する環境下ではプロジェクト名を工夫し、識別できるようにすることを推奨します
 - プロジェクト・リストにない場合は以下です（下図）
 - ✓ メニューバーの File → Open... を選択するか、アイコン  をクリックします
 - ✓ 表示される “Open File” の右下ファイルタイプを “Project Files (*.mpf)” にし、ブラウザ・選択して OK します
- または Transcript 窓に以下例のようなコマンドをタイプします（ディレクトリはフルパス指定の必要があります）

ModelSim> **project open** C:/<directory (full-path)>/<project_name>

☞ プロジェクト名には拡張子 .mpf 無しでも良いです

☞ オープン後はカレント・ディレクトリが自動で移動します



ブラウズするファイルタイプとして
Project Files (*.mpf)
を選択します ↓

← ブラウズして意図するファイル
(.mpf) を選択後、OK します



タイミング（遅延）シミュレーション

- PAR 結果のネットリストを対象とするゲートレベル・シミュレーションで遅延ファイル sdf を用いる『実負荷』シミュレーションです（青字は実際の文字列に要編集）
 - 次ページに do マクロ記述例を示します
 - “Export Files” プロセスで “Gate-Level Simulation File” (Radiant) 、 “Verilog/VHDL Simulation File” (Diamond) を実行
 - ✓ sdf 及びシミュレーション用 RTL 記述ゲートレベル・ネットリスト .vo / .vho を生成します
 - ☞ 拡張子 .vo は Verilog、.vho は VHDL です
 - ☞ Radiant では .vo のみが生成されます
 - コンパイル対象のファイルはテストベンチと .vo / .vho のみです（以下は Verilog 例）
`vlog <design_impl>_vo.vo`（☛ “<design_impl>_vo.vo” は Radiant/Diamond 生成ファイル名）
 - シミュレーション初期化コマンド `vsim` で sdf ファイルと遅延条件を指定します
 - ✓ 下の例では sdf ファイルはカレントディレクトリにあるものとします
 - ☞ 別のディレクトリにある場合は下例の <design_impl>_vo.sdf の箇所を次のようにします
“C:/<folder-with-sdf>/<design_impl>_vo.sdf”
 - ✓ 遅延条件は “sdfmax” “sdfmin” “sdftyp” のいずれかです（同時にはできません）

```
vsim -t 10ps -sdfmax <DUT top> =<design_impl>_vo.sdf -L ovi_iCE40UP <test_bench>
```

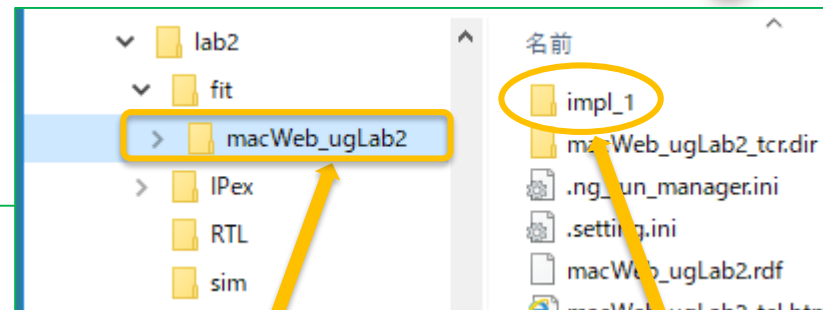
時間精度 指定可 TB 内 DUT インスタンス名 ターゲットデバイス Verilog ライブラリ テストベンチ名

タイミング（遅延）シミュレーション（つづき）

- 演習（サンプルデザイン）“lab2” について do マクロ記述例を示します（zip ファイル内 “init_sdf.do”）
 - この例では RTL シミュレーションと別にプロジェクトを作成します
 - ✓ 別プロジェクトとすることは任意です
 - .vo / .sdf はインプリ名の “impl_1” 下に生成します

```
# "init_sdf.do"
set WORK_DIR C:/macWeb_lsc_UG/lab2/sim
cd $WORK_DIR
set SRC_DIR $WORK_DIR/../../fit/macWeb_ugLab2/impl_1
#
project new $WORK_DIR mdlsim_uglab2_sdf
project addfile $SRC_DIR/macWeb_ugLab2_impl_1_vo.vo
project addfile tb_lab2.vhd
#
vlib work
vdel -lib work -all
vlib work
#
vlog $SRC_DIR/macWeb_ugLab2_impl_1_vo.vo
vcom tb_lab2.vhd
#
vsim -t 10ps -sdfmax u_UG_LAB2=$SRC_DIR/macWeb_ugLab2_impl_1_vo.sdf -L ovi_iCE40UP TB_LAB2
#
add wave /TB_LAB2/*
run 5us
```

サンプルデザインのフォルダ構成例



Radiant
プロジェクト名

デフォルトの
インプリメンテーション名
(vo, sdf ファイルの在りか)



SERDES、暗号化 IP などのシミュレーション

- Radiant で Crosslink-NX 用など暗号化 IP 有りデザインのコンパイルコマンドです
`vsim -L work -L ovi_lifcl -L pmi_work <tb_name>`
 - 必ず “pmi_work” を含めます
 - 例えば DDR3 Mem. Cont. や PCIe IP が対象ですが、その他特定のマクロも該当します
- これらマクロ / IP の実体は Verilog 記述ですので `vsim` コマンドを用います
 - Diamond で生成する VHDL は “VHDL 記述ラッパー” を top とし、その中で Verilog 本体をインスタンスする形式です (ラッパー top は `vcom` コマンドでコンパイルします)
- SERDES (PCS) 有りデザインのコンパイル・コマンドは以下のとおりです
ECP5/Diamond: `vsim -L work -L ovi_ecp5um -L pcsd_work -L pmi_work <tb_name>`
Crosslink-NX/Radiant: `vsim -L work -L ovi_lifcl -L pmi_work <tb_name>`
 - Diamond 3.12: PCS ライブラリ (ECP5 の場合 `pcsd_work`) および “pmi_work” ライブラリを含めます
 - ☞ PCS ライブラリ名は対象デバイスによって異なります (ECP3 は `pcsc_work`、など)
 - ☞ “pmi” とは parameterized module inferencing の略で、モジュールをインスタンスする手法。[この手法でインスタンス記述されている PLL や EBR / Dist.Mem を含むマクロが含まれている場合は、-L 指定が必須です](#)
 - ☞ ECP5 のシミュレーション時は必ず “+define+RSL_SIM_MODE” を与えるようにします (組み込まれている初期化マクロである `RSL_core / SLL_core` の有意な動作開始までの時間を短縮します)
 - Radiant 2.2~: Crosslink-NX / Certus-NX の SERDES マクロはマルチプロトコル対応の PCS ではなく、PCIe ハードマクロのため、Diamond のような PCS ライブラリはありません。ターゲットデバイス用 Verilog ライブラリと `pmi_work` を `-L` 指定します

まとめ

- do マクロベースのシミュレーション実行には多くの利点があります
 - 繰り返し実行の伴うデバッグ・検証の作業効率に優れます
 - 結果的に設計品質を高めることができます
 - シミュレーションに必要なファイルを全てまとめても比較的小さいサイズのため扱いが容易で、シミュレーション環境や結果波形ファイルを他者と共有したり、作業を再現することが容易です

- ここに示した例を変形・改善して、**各自の定型**を作り込むことを推奨します
 - 自分なりに流用・活用できる形式を確立しておくことで、新プロジェクトにも毎回最小の労力で適用できます

- ☞ do マクロで使用するコマンド詳細はツールを起動して Help をご参照下さい
 - メニューバーの [Help](#) → [Product Help](#) → “[Using Macro Commands](#)” でサーチ
 - マクロコマンドは ModelSim® / Mentor 社と等価ですので、既存マクロを流用する際の作業にも本ガイドの内容をご参照ください

変更履歴 History



Date	Revision	Page	Change Information	Updated by
2021/2/16	0.2		Draft Version for review (VHDL/Radiant as Lab.2). P.26 yet-to-be-done.	S.S.
2021/3/15	0.3	5,11,14,19	Minor edits/corrections	S.S.
		15 - 17	Changed the order of pages	
		25	Minor edit at the line #8	
2021/3/16	0.4	25, 26	Minor edits on p.25, and inserted new p.26 with an example (page# referenced in some pages are updated accordingly)	S.S.

A. 補 足



- A1. ライブラリの名称
- A2. “vlog” コンパイル・ディレクティブ
- A3. マウス操作による観測信号の指定方法
- A4. Simulation Wizard 生成のマクロの流用
- A5. 各種アイコン
- A6. do マクロ記述 : Active-HDL と ModelSim の比較

A 1 : ライブラリ名



- Radiant 2.2~ / Diamond 3.12 のサポートする各デバイス・ファミリのライブラリ名は以下のとおりです

Radiant

Family	Verilog	VHDL
iCE40 Ultra Plus	ovi_iCE40UP	iCE40UP
Crosslink-NX	ovi_lifcl	lifcl
Certus-NX	ovi_lfd2nx	lfd2nx

Diamond

Family	Verilog	VHDL
Crosslink	ovi_lifmd	lifmd
Crosslink Plus	ovi_lifmdf	lifmdf
Mach-NX	ovi_lfmnx	lfmnx
MachXO2	ovi_machxo2	machxo2
MachXO3L/LF	ovi_machxo3l	machxo3l
MachXO3D	ovi_machxo3d	machxo3d
ECP5U	ovi_ecp5u	ecp5u
ECP5UM	ovi_ecp5um	ecp5um
LatticeECP3	ovi_ecp3	ecp3

A2: “vlog” コンパイル・ディレクティブ (1)



■ 特に Verilog コンパイル・コマンド (p.10 ⑤部) では、言語仕様と相俟って種々のオプション (ディレクティブ、マクロ) 指定が可能です。特に比較的頻繁に用いる例を次に挙げます (別デザインでのマクロ例)

👉 Active-HDL と全く同じです

[1] “+incdir+xxx”

変数定義

- * 対象 RTL 内に `include 文があり、かつ記述されているディレクトリに当該ファイルがない場合のサーチディレクトリを xxx で指定します
- * 当該 vlog コマンドで対象としているソースファイルにのみ有効です (右例は “ddr3_dim_16.v”)
- * 同じファイルが複数の RTL で include されている場合は RTL 名を続けて記述しても有効です

```
set PROJ_DIR "<work-dir>/VIP_DDR3_Demo"  
set ULOGIC_SRC "$PROJ_DIR/rtl_ulogic"  
#  
# compile Verilog src which contain "`include" directive  
vlog +incdir+$PROJ_DIR/inst1/src/params ¥  
$PROJ_DIR/mem/ddr3_dimm_16.v
```

include されているファイルの場所は
“\$PROJ_DIR/inst1/src/params”

対象 RTL ソース中で include している
記述例 (“ddr3_dimm_16.v”)

```
...  
`include "ddr3_sdram_mem_params.v"  
`include "tb_config_params.v"  
  
module ddr3_dimm_16 (  
    rst_n, ddr_clk,  
    ...
```

注: include されているファイル中でさらに include していると “+incdir+” ディレクティブで指定しても効果がないように見受けられるので、そのファイルを include 元のファイルがあるディレクトリにコピーしておくとい良いでしょう

A2: “vlog” コンパイル・ディレクティブ (2)



[2a] “+define+<macro_name>”

[2b] “+define+<name>=<value>”

* 固有文字列の定義や、文字列に値を与えます (2bでは “=” を忘れないこと)

- ☞ RTL 内 `define 記述文と同等ですが、その場合は論理合成ツールにも作用します
- ☞ ディレクティブを活用すれば、シミュレーターにのみ指示が与えられるので、作業によって RTL を毎回編集する必要がありません

RTL 内で記述した場合の例

```
`define SIM_MODE  
`define FRAME_PATTN 2
```

← この場合は論理合成も
論理シミュレーションにも
共に作用します

```
...  
`ifdef SIM_MODE  
    localparam ELAPSE1 = 18'h000C0;  
`else  
    localparam ELAPSE1 = 18'h3F000;  
`endif
```

```
set PROJ_DIR "<work-dir>/VIP_DDR3_Demo  
set ULOGIC_SRC "$PROJ_DIR/rtl_ulogic"  
#  
# compile Verilog src with macro  
vlog +define+SIM_MODE $ULOGIC_SRC/ddr_ulogic.v
```

“ddr_ulogic.v”

```
...  
`ifdef SIM_MODE  
    localparam ELAPSE1 = 18'h000C0;  
`else  
    localparam ELAPSE1 = 18'h3F000;  
`endif  
...
```

← シミュレーションで有効

← 論理合成で有効

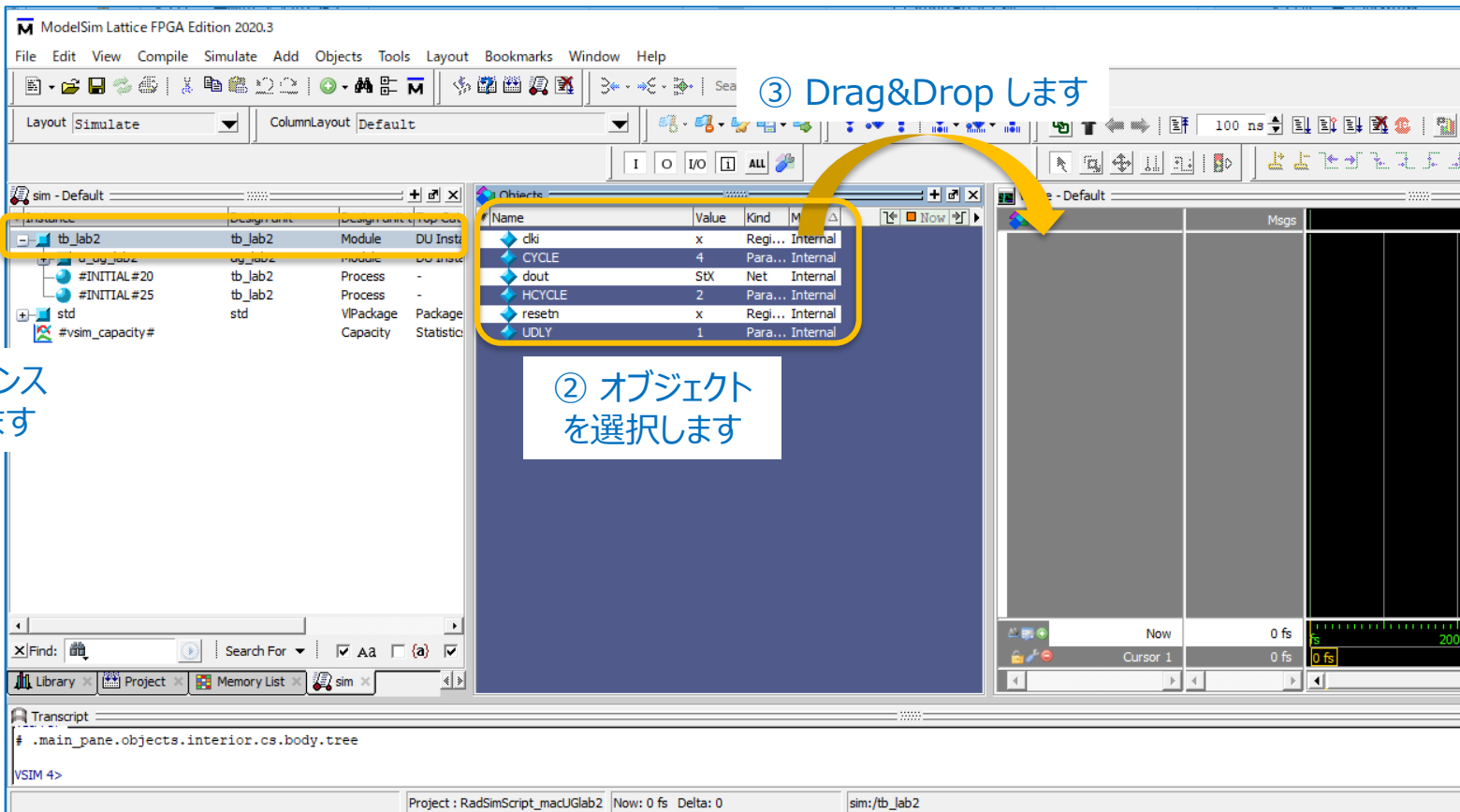
RTL ソース中で長いタイマー待ちがあるデザインで、シミュレーション実行時間を短くするための例

A3. マウス操作による観測信号の指定方法（1）



(第一の方法) ドラッグ&ドロップ操作による信号表示の指定方法です

- ① “Wave” 窓で観測する信号を含むモジュール（エンティティ）のインスタンスを “Instance” 窓で選択します
- ② 選択されたインスタンスのモジュール記述に含まれる信号 = オブジェクト（ネット、ポート）が “Objects” 窓にリストされますので、観測したい信号（複数可）を選択します
- ③ “Wave” 窓にドラッグしてドロップします



① インスタンス
を選択します

② オブジェクト
を選択します


③ Drag&Drop します

A3. マウス操作による観測信号の指定方法（2）



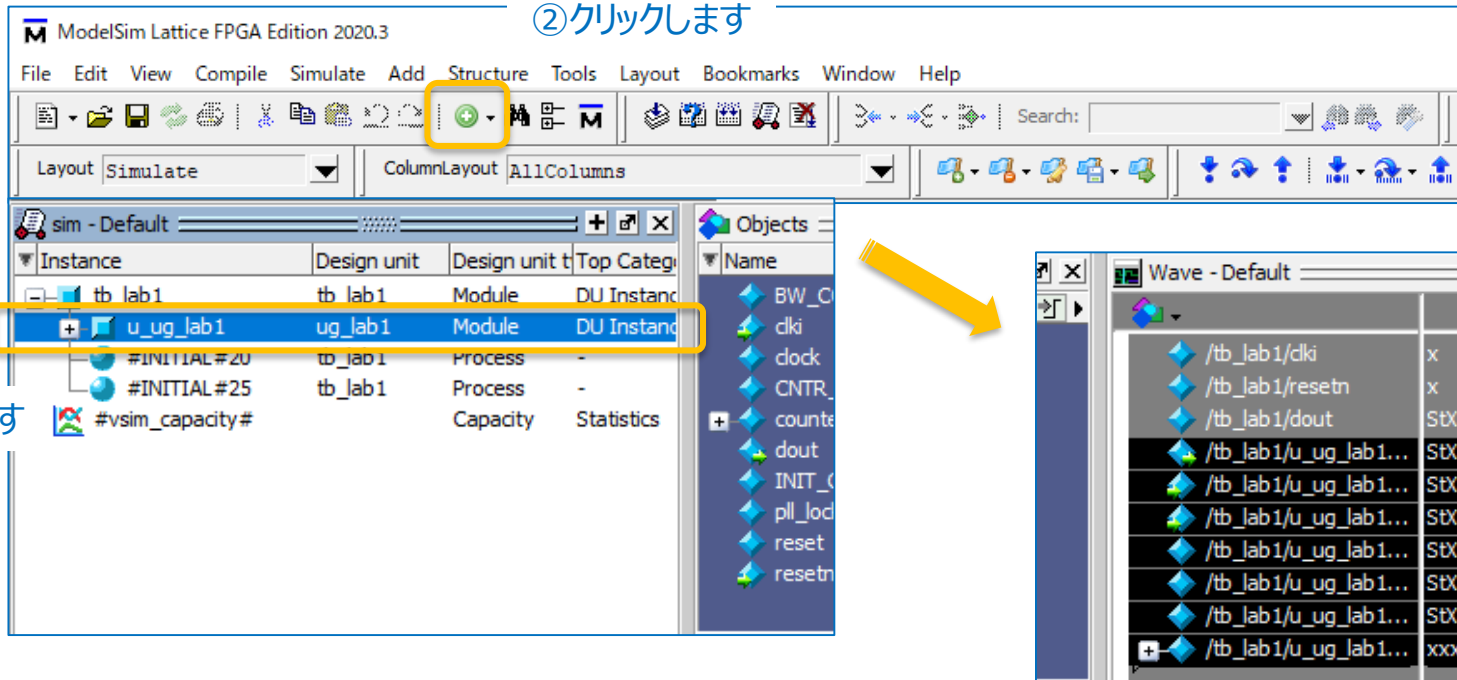
(第二の方法) アイコンを用いる方法です

① “Wave” 窓で観測する信号を含むモジュール（エンティティ）のインスタンスを “Instance” 窓で選択します（前頁と同様）

② アイコン  をクリックします

☞ 選択モジュール（アーキテクチャ）階層にある信号（オブジェクト）が**全て**挿入されます

② クリックします



① 選択します

Instance	Design unit	Design unit t	Top Categori	Name
tb_lab1	tb_lab1	Module	DU Instance	BW_O
u_ug_lab1	ug_lab1	Module	DU Instance	clk
#INITIAL#20	tb_lab1	Process	-	clock
#INITIAL#25	tb_lab1	Process	-	CNTR_
#vsim_capacity#		Capacity	Statistics	counte
				dout
				INIT_C
				pll_loc
				reset
				resetn

Wave - Default	Msgs
/tb_lab1/dki	x
/tb_lab1/resetn	x
/tb_lab1/dout	StX
/tb_lab1/u_ug_lab1...	StX
/tb_lab1/u_ug_lab1...	StX
/tb_lab1/u_ug_lab1...	StX
/tb_lab1/u_ug_lab1...	StX
/tb_lab1/u_ug_lab1...	StX
/tb_lab1/u_ug_lab1...	StX
/tb_lab1/u_ug_lab1...	xxxxxx

追加されるオブジェクトは、Verilog では wire / reg 宣言されているネットとポート，VHDL では signal 宣言されているネットおよびポートです（いずれも重複分を除く）

A3. マウス操作による観測信号の指定方法（3）



(第三の方法) メニューバーを用いる方法です

- ① “Wave” 窓で観測する信号を含むモジュール（エンティティ）のインスタンスを “Instance” 窓で選択します（前頁と同様）
- ② 選択された行の上で右クリックして Add to → Wave → All Items in region を選択します
 - ☞ 選択モジュール（アーキテクチャ）階層にある信号（reg / wire / signal / port）が**全て**挿入されます
 - ☞ “All Items...” の他の二つのいずれかを選択しても良いですが、やや煩雑になります

① 選択します

② 右クリックして 選択します

Wave - Default

	Msgs
/tb_lab1/dki	x
/tb_lab1/resetn	x
/tb_lab1/dout	STX
/tb_lab1/u_ug_lab1...	STX
/tb_lab1/u_ug_lab1...	STX
/tb_lab1/u_ug_lab1...	STX
/tb_lab1/u_ug_lab1...	STX
/tb_lab1/u_ug_lab1...	STX
/tb_lab1/u_ug_lab1...	xxxxxx

当該モジュール（アーキテクチャ）階層にあるオブジェクト=信号（reg / wire / signal / port）が全て追加されます

A3. マウス操作による観測信号の指定方法（4）



(第四の方法) メニューバーを用いるもう一つの方法です

- ① “Wave” 窓で観測する信号を含むモジュール（エンティティ）のインスタンスを “Instance” 窓で選択します（前頁までと同様）
- ② メニューバーの Add → To Wave → All Items in region を選択します
☞ “All Items...” の他の二つのいずれかを選択しても良いですが、やや煩雑になります

① 選択します

② Add をクリックして選択します

当該モジュール（アーキテクチャ）階層にあるオブジェクト=信号（reg / wire / signal / port）が全て追加されます

☞ 観測信号の追加は Transcript 窓で次例のようなコマンドをタイプすることと等価です

VSIM > add wave -noupdate /* (TB 階層の信号全て)

VSIM > add wave -noupdate /<TB_name>/uut/* (TB 内のインスタンス uut の信号全て)

VSIM > add wave -noupdate /<TB_name>/uut/u0/abc (uut 内インスタンス u0 の信号 abc)

A4: Sim.Wizard を用いた do マクロの生成と実行



- Radiant / Diamond 組み込みの Simulation Wizard を出発点にして “シミュレーションの繰り返し実行用” do マクロを生成し、実行する方法です
 - ✓ Sim. Wizard から ModelSim GUI 起動し、シミュレーション実行まで行います。その際にマクロ “[xxx.mdo](#)” が自動生成されますので、これを編集して用います
- 本 UG で述べた手法を適用するためのアプローチとしては有用です
- 次ページ以降にその手順概要を示します

A4. Sim. Wizard 生成の “.mdo” を編集 (1)



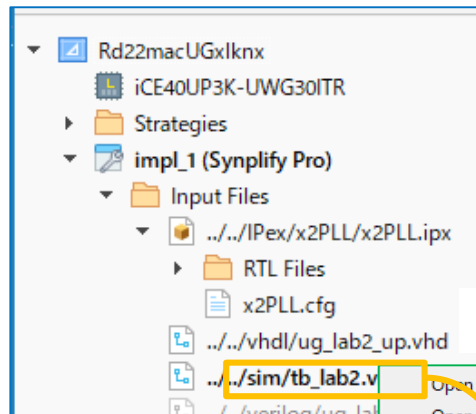
1-1. テストベンチ (および sim. に必要なファイル全て) を **Input Files** にインポートします

✓ ファイルを選択 → 右クリック → Include for 属性を “Simulation” に変更します

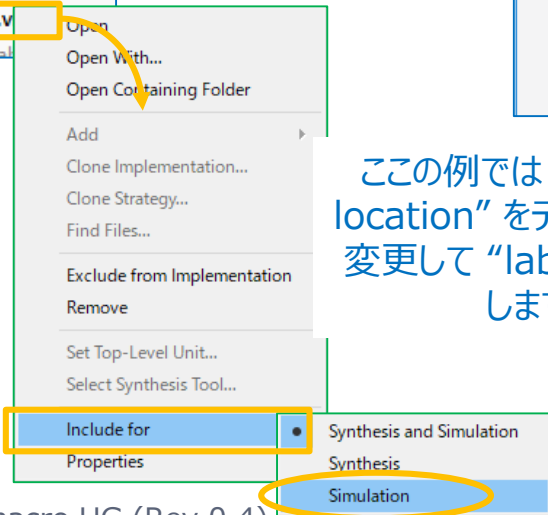
1-2. アイコン  をクリックし Simulation Wizard を起動します

✓ 問題なく Active-HDL が起動してシミュレーション実行されると、**Simulator Project Name** 窓での “Project name” 欄入力と同名のフォルダが作成されます

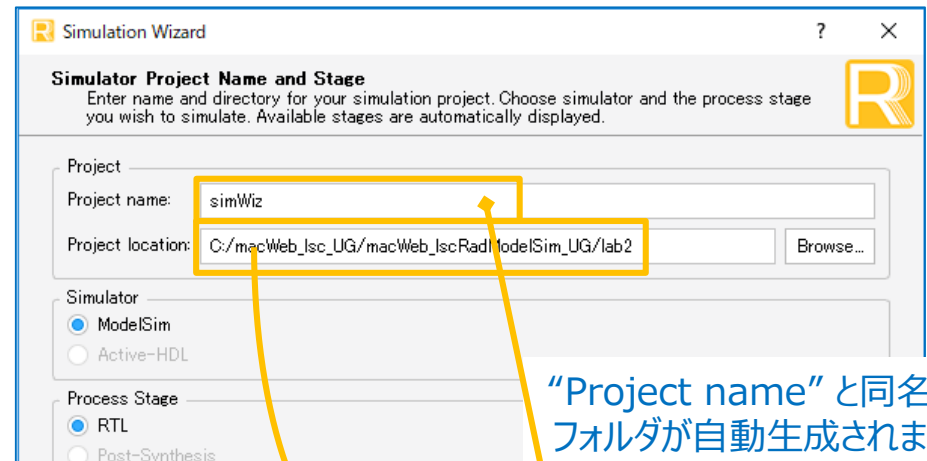
👉 詳細手順は別途ユーザガイド『ModelSim_LE_for_old_version_tools』をご参照ください



右クリックします

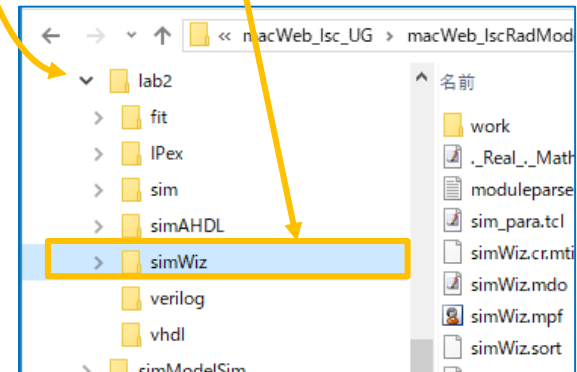


Simulation 属性でテストベンチ・ファイル一式をインポートします



“Project name” と同名のフォルダが自動生成されます

この例では “Project location” をデフォルトから変更して “lab2” に移動します

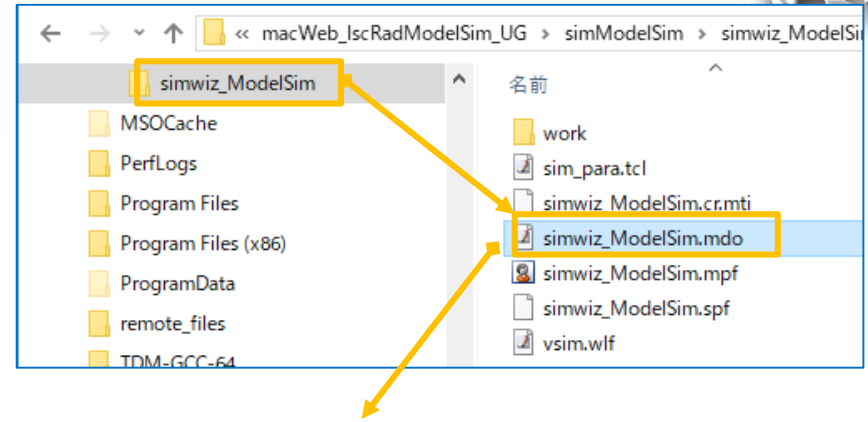


A4. Sim. Wizard 生成の “.mdo” を編集 (2)



1-3. 生成フォルダ下の “.mdo” を編集します

- ✓ Sim. プロジェクトは生成されたものとして
- ✓ 編集後に適切な別名で保存します (例: “run.do”)
- ✓ 保存先は Sim. Wizard プロジェクト・フォルダです (ここでは “simwiz_ModelSim”)
- ✓ この例では黒字が必要な箇所です (殆どの例と同様)
- ✓ vlog / vcom 行に +incdir+ ディレクティブが長々と記述されますが、殆どのケースでは不要です (黒字以外)。必要なケースでのみ記述するようにします



```
if {![file exists "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/simModelSim/simwiz_ModelSim/simwiz_ModelSim.mpf"]} {
    project new "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/simModelSim/simwiz_ModelSim" simwiz_ModelSim
    project addfile "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/sim/tb_lab2.vhd"
    project addfile "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/IPex/x2PLL/rtl/x2PLL.v"
    project addfile "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/verilog/ug_lab2_up.v"
    vlib work
    vdel -lib work -all
    vlib work
    vcom -work work "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/sim/tb_lab2.vhd"
    vlog +incdir+C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/IPex/x2PLL/rtl +incdir+(omit) +incdir+(omit) -work work
    "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/IPex/x2PLL/rtl/x2PLL.v"
    vcom +incdir+C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/verilog +incdir+(omit) +incdir+(omit) -work work
    "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/lab2/verilog/ug_lab2_up.vhd"
} else {
    project open "C:/macWeb_Isc_UG/macWeb_IscRadModelSim_UG/simModelSim/simwiz_ModelSim/simwiz_ModelSim"
    project compileoutofdate
}
vsim -L work TB_LAB2 -L pmi_work -L ovi_ice40up
view wave
add wave /*
run 1000ns
```

繰り返し
実行には
不要な
部分

生成する波形表示マクロ
の実行に変更する

A4. Sim. Wizard 生成の “.mdo” を編集 (3)



1-5. 既に起動されている GUI で pp.35-37 に示すような手順で所望の表示波形になるように操作して、マクロ (例 : [waves.do](#)) として保存します

1-6. 一旦シミュレーションを終了します

VSIM > **quit** -sim シミュレーション終了 (“run.do” の冒頭に入れても良い)

1-7. Console 窓で以下のように一連のコマンドを

入力後、作成したマクロを実行します

VSIM > **do** run.do コンパイルから再実行

```
# "run.do"
#quit -sim
vdel -lib work -all
vlib work
#
vcom "C:/macWeb_lsc_UG/lab2/sim/tb_lab2.vhd"
vlog "C:/macWeb_lsc_UG/lab2/IPex/x2PLL.v"
vcom "C:/macWeb_lsc_UG/lab2/RTL/ug_lab2.vhd"
#
vsim -L work TB_LAB2 -L ovi_ice40up
#
do waves.do
run 5 us
```

[編集後の do マクロ例](#)

A5. do マクロ : A-HDL 用を ModelSim 用に変換①



Active-HDL 用 do マクロ最少記述

```
# "init.do"
set WORK_DIR "C:¥<work_directory>"
cd $WORK_DIR
set RTL_DIR "C:¥<source_directory>"
#
workspace create <ws_name>
design create -a <dws_name>
design open <dws_name>
cd $WORK_DIR
#
#
#
#
adel -all
#
```



ModelSim 用 do マクロ最少記述

```
# "init.do"
set WORK_DIR "C:/<work_directory>"
cd $WORK_DIR
set RTL_DIR "C:/<source_directory>"
#
project new $WORK_DIR <project_name>
#
#
#
project addfile $IPex_DIR/x2PLL/rtl/x2PLL.v
project addfile $RTL_DIR/ug_lab2.vhd
project addfile tb_lab2.vhd
#
vlib -work
vdel -lib work -all
vlib -work
```

やや異なる

①

要変更 →

②

要追加 →

③

要変更 →

① set コマンドは同等です ("" はあっても無くても可)。ただし、A-HDL で '¥' を使用していた場合は '/' に変更する必要があります

② Sim. プロジェクトの作成方法はことなり、それぞれの作法になります

③ ModelSim では、必須記述ではありませんが、ソースファイルを全てインポートしておくことを推奨します

(次ページにつづく)

A5. do マクロ : A-HDL 用を ModelSim 用に変換②



Active-HDL 用 do マクロ最少記述

```
#
# compile VHDL source files
vcom ../RTL/abc.vhd ../RTL/efg.vhd
#
# compile Verilog source files
vlog ../RTL/jk.v ../RTL/lm.v ../RTL/nn.v ¥
  tb_zz.v
#
# initialize simulation
vsim -L ovi_machxo2 +access +r <tb_name>
#
# set up signals to display
add wave *
#add wave <top_module_name>/*
#do waves.do
#
# run the simulation
run 100 us
```



ModelSim 用 do マクロ最少記述

```
#
# compile VHDL source files
vcom ../RTL/abc.vhd ../RTL/efg.vhd
#
# compile Verilog source files
vlog ../RTL/jk.v ../RTL/lm.v ../RTL/nn.v ¥
  tb_zz.v
#
# initialize simulation
vsim -L ovi_machxo2 <tb_name>
#
# set up signals to display
add wave /<tb_name>/*
#add wave /<tb_name>/<top_module_name>/*
#do waves.do
#
# run the simulation
run 100 us
```

同じ
④

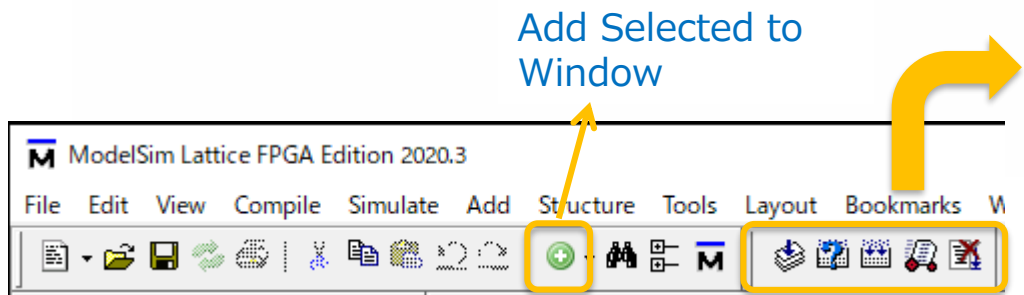
削除
⑤

若干違う
⑥

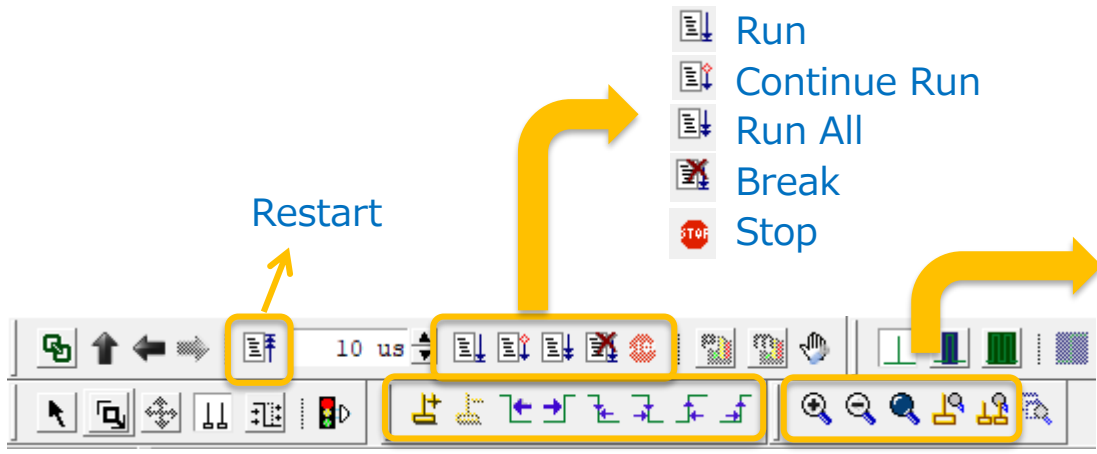
同じ

- ④ A-HDL で vlog/vcom コマンドに '-dbg' オプションを付加している場合は削除します
- ⑤ A-HDL の +access +r は不要です。その他は同じです
- ⑥ 信号の階層に関する表記が若干異なります (do waves.do は同じです)

A6. 各種アイコン

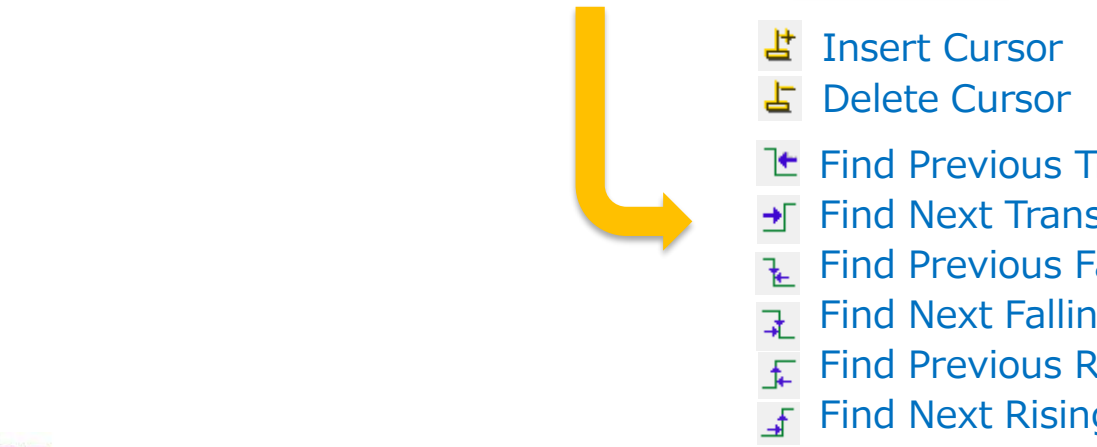


- Add Selected to Window
- Compile
- Compile Out of Date
- Compile All
- Simulate
- Break



- Run
- Continue Run
- Run All
- Break
- Stop

- Zoom In 拡大
- Zoom Out 縮小
- Zoom Full フル表示
- Zoom In on Active Cursor
- Zoom Between Cursors



- Insert Cursor
- Delete Cursor
- Find Previous Transition
- Find Next Transition
- Find Previous Falling Edge
- Find Next Falling Edge
- Find Previous Rising Edge
- Find Next Rising Edge