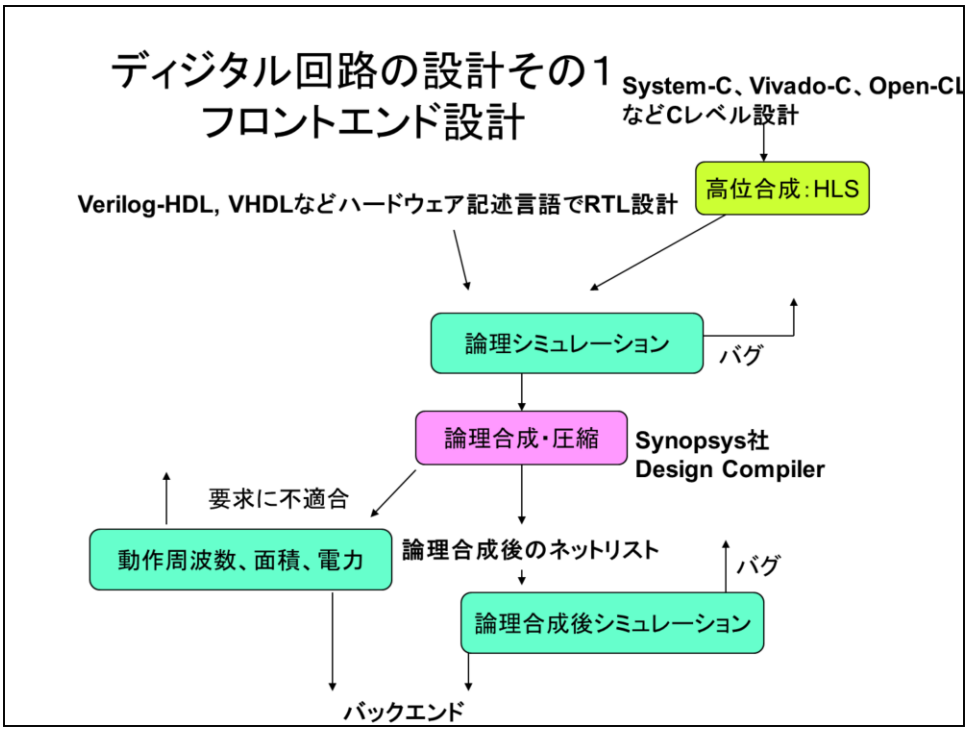
コンピュータアーキテクチャ　第3回
シングルサイクル版の論理合成

情報工学科
天野英晴

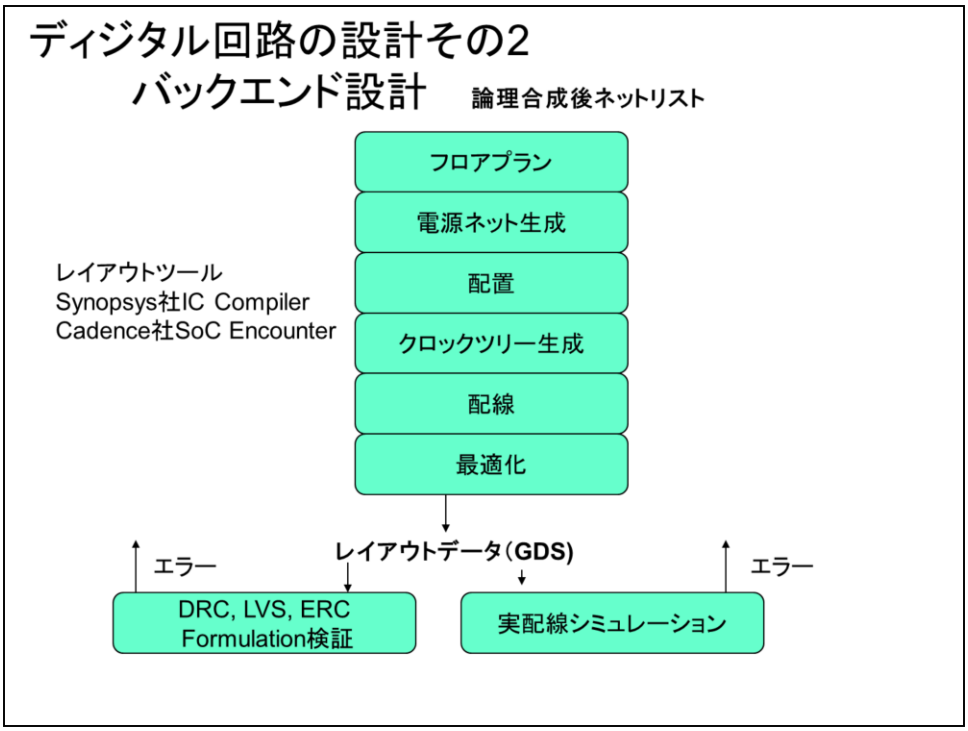
論理合成と圧縮

- VerilogHDLで記述し、シミュレーションしただけでは実際に動くシステムはできない
  - 論理合成、圧縮が必要
  - 対象デバイスのゲート間の接続リスト（ネットリスト）の形に変換
- チップ上でCPUを実現する
  - Synopsys社Design Compiler　→今回使う
- FPGA上でCPUで実現する
  - FPGAベンダのツール　→情報工学実験第2

今までVerilogHDLでMIPSeを記述してシミュレーションして動作を確認してきました。しかし、これだけでは実際に動くシステムはできません。CPUの入った集積回路を作る場合も、書き換え可能なLSIであるFPGA上で実現する場合も、論理合成、圧縮を行って、対象とするデバイスのゲート間の接続リスト（ネットリスト）に変換する必要があります。今回は、Synopsys社のDesign Compilerを用いて、実際の集積回路上にチップを実装する方法を紹介します。実際には書き換え可能なLSIであるFPGA（Field Programmable Gate Array)上で実現する場合が多いですが、これは来年の情報工学実験第2で実際に行います。



ディジタル集積回路の設計は、まずフロントエンド設計を行います。まずハードウェア記述言語でRTL設計を行います。今までやってきたように論理シミュレーションを行って、動作を確認しながら設計を進めます。設計が一段落したら、論理合成・圧縮を行います。CADがRTL記述を解析し、論理ゲート間の接続図の形に変換します。これと同時に動作周波数、面積、電力を見積もります。これが要求を満足しなければ、最初に戻って設計をやり直します。要求を満足すれば、ネットリストをシミュレーションして動作を確認します。基本的にCADで生成されたネットリストのシミュレーションは元のRTLのシミュレーションと一致するはずですが、記述のやり方が悪いと意図通りの動作を行わない場合があります。もしも問題が見つかったら設計をやり直します。最近はSystem-C、Vivado-C、Open-CLなどのCレベル設計が特にFPGAを対象として広まっています。このC言語の記述は高位合成（High Level Synthesis:HLS)によりRTL記述に変換されます。これ以降は同じ流れになります。



フロントエンド設計により、ネットリストが固まったら、次の段階はバックエンド設計です。
これは、4年生のVLSI設計論で実際に演習をしながら学んで行きます。

Design Compilerによる論理合成

- ライセンスの関係で天野研究室のマシン(sirius.am.ics.keio.ac.jp)を使う
  - アカウント情報は注意して管理
  - VDECのライセンスなので教育研究専用
- 対象デバイスは、オクラホマ大のTSMC 0.18um CMOSプロセスを利用
  - ライブラリのセル数が少ない
  - プロセスが時代遅れ
  - しかし、商用プロセスのライブラリを利用するためにはNDA契約が必要、非常に高価
- バッチ処理で用いる
  - tclファイル(ここではmipse.tcl)にやることを書いておく
- 実行

  dc_shell-t –f mipse.tcl | tee mipse.rpt

  レポートファイルがmipse.rptに生成される

今回利用するのはSynopsys社のDesign Compilerという論理合成用のツールです。このツールはチップ設計用に世界中で使われており、実は非常に高価です。しかし、大学の教育研究用に限って、大規模集積設計教育研究センター（VDEC)により安価で提供されています。したがって、管理の都合上、天野研のマシン上で動かすことになります。このツールは教育研究用の利用以外は禁止されているので、これを使って何かを設計して売り出したりしてはいけません。（これをやりたければ天野研に来れば、正規の形で出来るようにしてあげるので、どうぞおいでください）。実際のチップを設計するためには、チップ上で動くゲート、フリップフロップなどの論理素子の一式が必要です。これをセルライブラリと呼びます。セルライブラリは、遅延時間、面積、消費電力が定義されていて、これを基にCADは論理合成・圧縮を行います。

今回は対象デバイスとしてはオクラホマ大の教育用のセルライブラリを使います。これは、TSMC（台湾の世界的半導体の製造メーカー（ファブ））の0.18μmのセルライブラリをモデルとしており、プロセスとしては古いですがリアルです。実際の商用プロセスのライブラリは非常に高価な上、利用にはNDA（秘密保持契約）を結ぶ必要があり、とても授業では使えません。

Design Compilerは、tclファイル（ここではpoco.tcl)にコマンドを書いておき、これをdc_shellと呼ばれるシェルに入れてやり、一度に実行させます。これをバッチ処理と呼びます。合成後のネットリストを見るツールdesign_visionもありますので、今回はこれも使います。

例題

シングルサイクルのmipse.tclを論理合成して結果を確認せよ

動作周波数, Total Cell Area, Total Powerを見てみよう

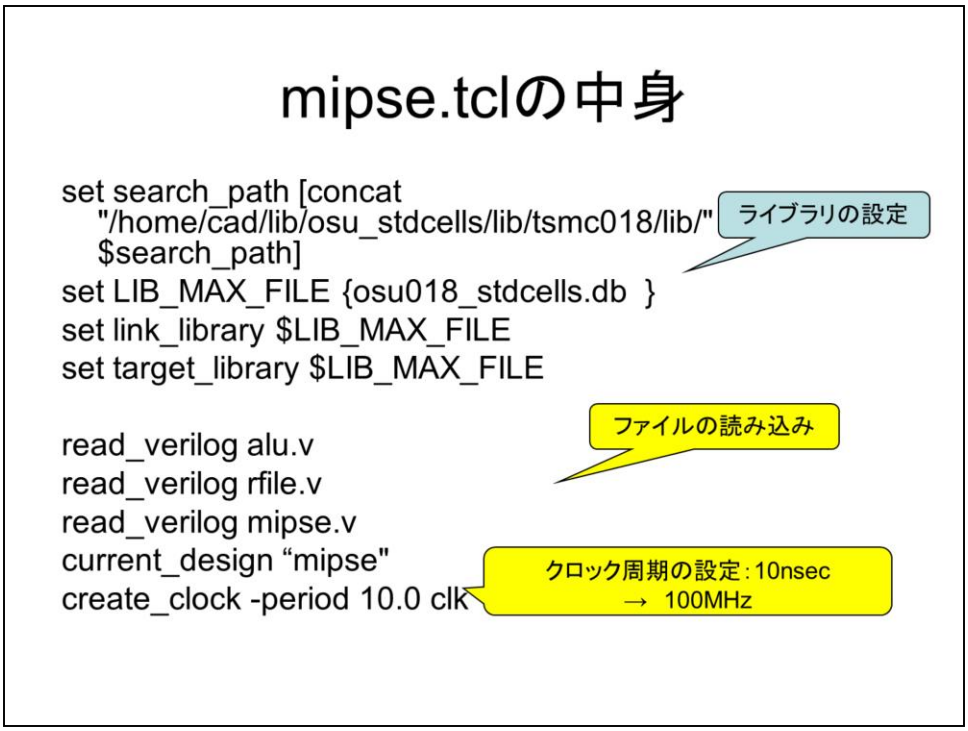
ではちょっと実際に合成をしてみましょう。

siriusへのログインとファイル転送

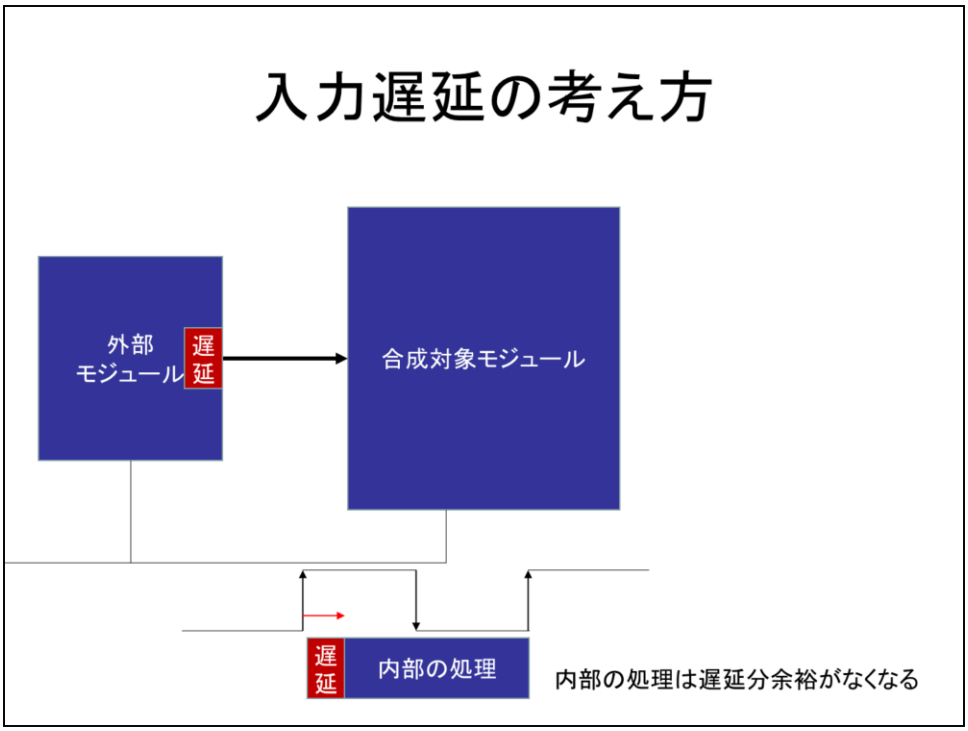
web上の合成用環境の使い方
を参照のこと

ssh –Y exXXXX@sirius.am.ics.keio.ac.jp
passwdでパスワードを変更

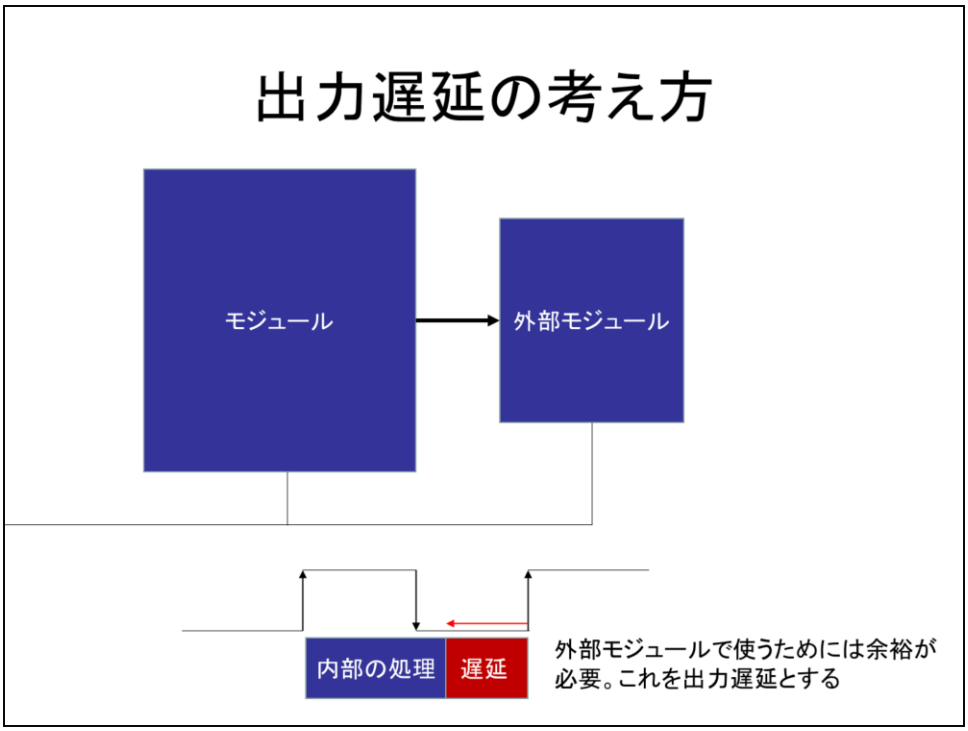
scp synth.tar exXXX@sirius.am.ics.keio.ac.jp:~/.



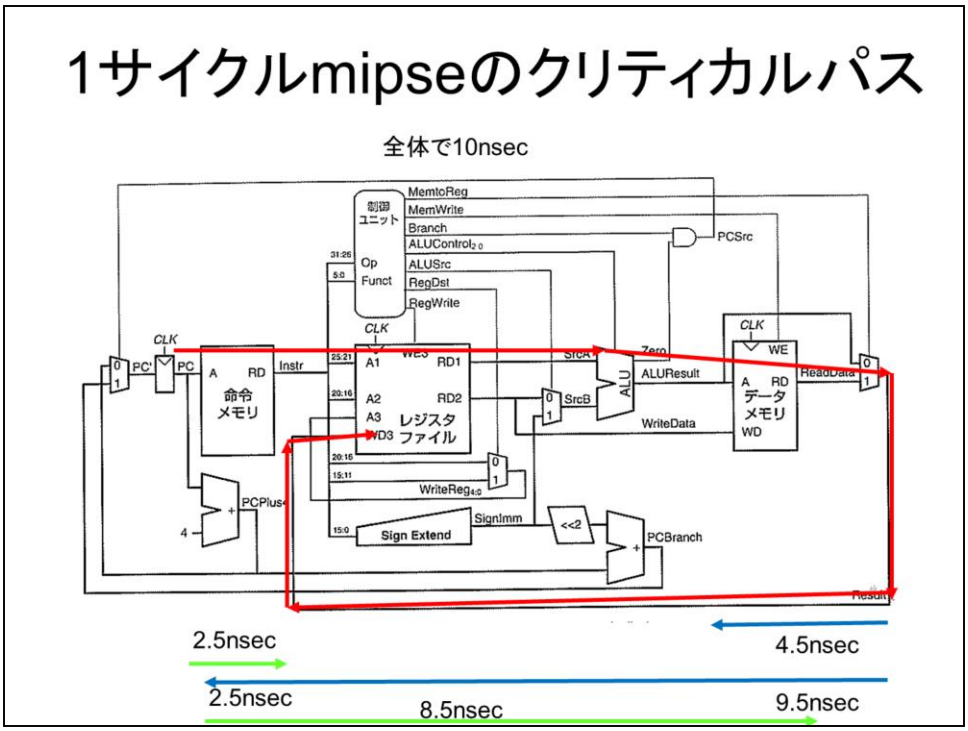
mipse.tclの中身を示します。最初数行は合成に必要なライブラリを指定するもので、演習では変えてはいけません。それからファイルを読み込み、クロックを設定します。ここでは10MHz＝10nsecにしています。



ではまず入力遅延を設定しましょう。外部モジュールと合成対象モジュールは同じクロックで動作していると考えます。入力は外部モジュールから与えられますが、ここには一定の遅延があるはずです。合成対象モジュールは、これに内部の処理の遅延を加えた全体の遅延が、次のクロックの立ち上がりに間に合うように合成しなければなりません。つまり入力遅延が大きいと、内部の動作は厳しくなり、場合によっては設定周期を満足できなくなります。遅延は立ち上がりのクロックを基準として図の赤矢印の方向に設定します。



出力遅延は、入力遅延と逆で、モジュールからの出力が外部モジュールで確実に格納されるために余裕を持って出力しなければならず、このための遅延です。この遅延は次のクロックの立ち上がりを基準として、図の赤矢印の方向に考えて設定します。



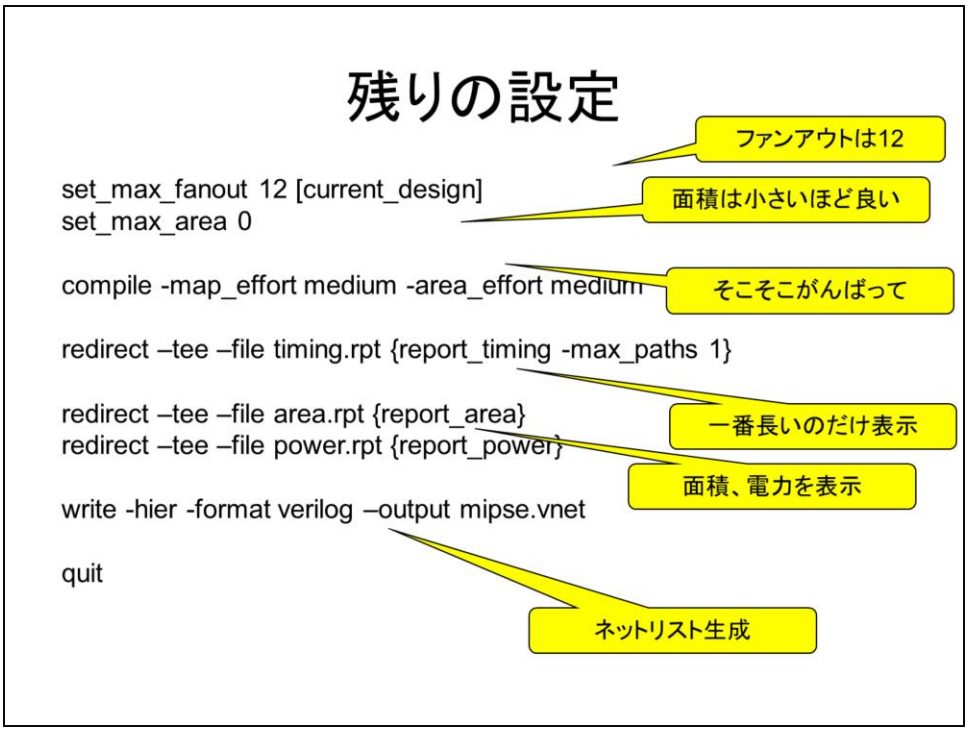
MIPSeの場合、クリティカルパスは、ディスプレースメントの計算とメモリアクセスが両方必要なlw命令で決まります。全体のクリティカルパスが10nsecで、メモリの遅延が2nsecであることを考えて、各部の遅延を設定します。

入出力遅延の設定

```
set_input_delay 7.5 -clock clk [find port "readdata*"]
set_input_delay 2.5 -clock clk [find port "instr*"]
set_output_delay 9.5 -clock clk [find port "pc*"]
set_output_delay 4.5 -clock clk [find port "aluresult*"]
set_output_delay 4.5 -clock clk [find port "writedata*"]
set_output_delay 4.5 -clock clk [find port "memwrite"]
```

メモリの遅延を2nsecと考えた

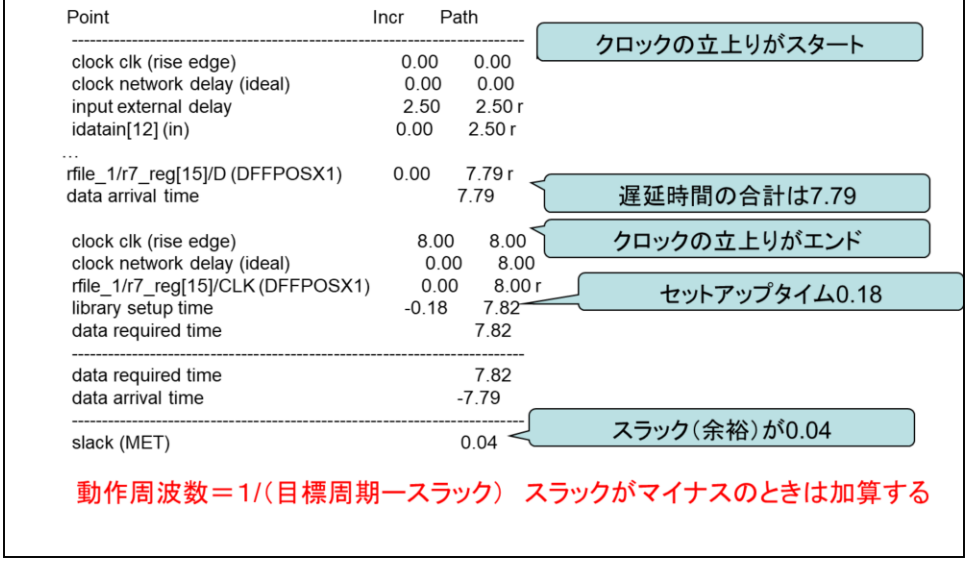
では、この考え方で、入出力の遅延を設定します。1サイクルCPUは、遅延パスが途中で外部メモリを通るので、この設定が面倒です。set_inpit_delayは入力信号の遅延、set_output_delayは出力信号の遅延設定です。それぞれの信号について設定します。



最大ファンアウトは12に設定し、ひとつの出力に繋がる入力数を12に制限します。次に目標面積に0を設定します。これはムチャか気がしますが、これはあくまで目標であり、面積は小さいほど良いので、通常このように設定します。次はコンパイル（ここでは論理合成、圧縮の最適化）レベルを設定するコマンドで、ここではmediumで中程度にがんばって欲しい設定にしています。これをhighにすると、もっと最適化をがんばってくれますが、コンパイル時間が非常に掛かります。さて、次からはレポート文で出力を制御します。report_timingは最も長いパス、すなわちクリティカルパスを表示します。ここでは一番長いののみ表示しています。次に面積、電力を出力します。これらはそれぞれtiming.rpt, area.rpt, power.rptのファイルに格納されます。最後にはネットリストを生成しておしまいです。ネットリストの名前はpoco.vnetとしています。これも実はVerilogの記述ですが、ゲート間の接続の形に変換されています。

では、dc_shell-t –f poco.tcl | tee poco.tclで実行します。Synopsysの実行には初期設定が必要です。これはこのスライドの最後の方に使い方をまとめているのでここを見てください

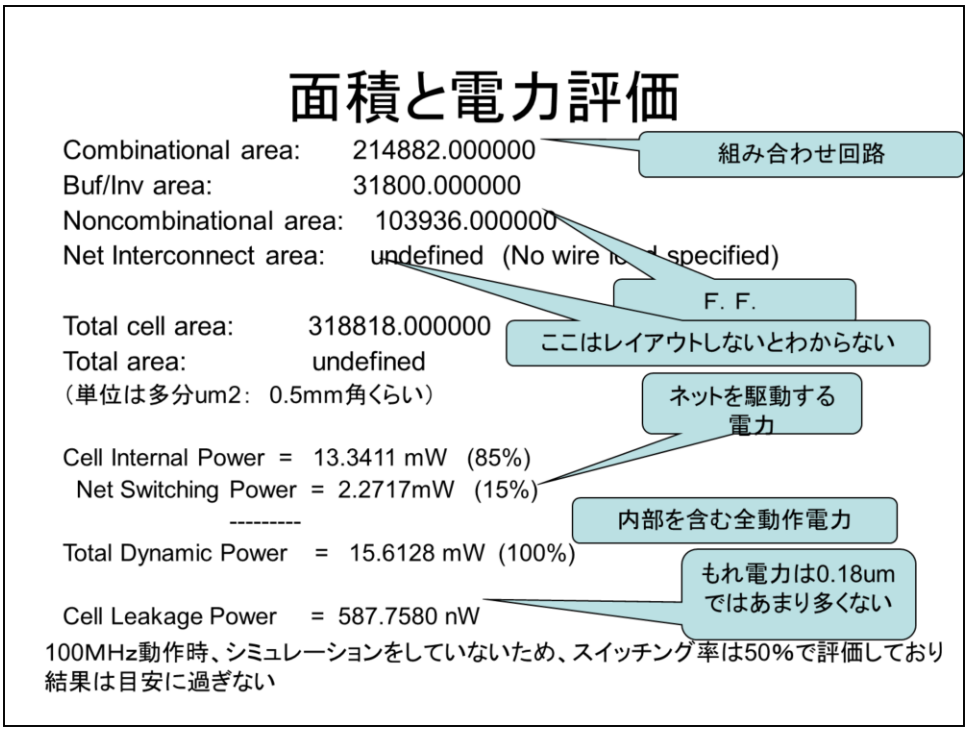
クリティカルパスの表示



timing.rptの中身を見てみましょう。このスライドのようなレポートが出ていると思います。パスは長い順に表示されるので、最初の1本が一番大事で、それだけ表示されています。この表は、クリティカルパスがどのように辿っているか、途中でどのように遅延が増えているかを示しています。スタートポイントはclkの立ち上がりで、idatainに対する入力遅延が2.5nsecがまず加わり、さらにCPUの内部で順に遅延が積み重なっている様子がわかります。この場合、遅延時間の合計は7.79になっています。これにフリップフロップに値を格納するために必要なセットアップタイム0.18nsecが加わります。これは目標周期を8nsecに設定しているため、0.04nsecの余裕（スラック）が生じていることがわかります。（電卓で計算すると0.03になるはずだが、下の桁まで計算に入れているのかと思う）

この回路の動作周波数は、目標周期ースラックの逆数になります。今回は、125.6MHzです。スラックはマイナスになる場合もあり、この場合は加算されることになり、動作周波数は目標周波数よりも落ちます。

目標周期を短くすると、クリティカルパスを短くしようと論理合成、圧縮をがんばってくれるので、性能が上がる可能性がありますが、その分面積が増えてしまいます。スラックが0前後になるように目標周期を設定するのが多くの場合は良いといわれています。



次に面積はarea.rptに電力はpower.rptに格納されています。ここで、単位は平方μmです。つまり、1000000平方μmが1mm角になります。Combinational areaは組み合わせ回路、Noncombinational areaがフリップフロップを表します。Net Interconnect areaは配線のための面積で、これはレイアウトしないと分かりません。セル全体の面積は318818になります。セルの充填率つまり詰め込む割合を70%くらいと考えると、レイアウト全体の面積は455454となり、この平方根は674なので、大体0.67mm角くらいであることが分かります。（去年ちょっと合成をしてみたPOCOは0.25mm角くらいでした。やっぱり結構大きいです）

最後は、動作周波数が出てきます。Cell Internal Powerはセルの貫通電力、Net Switching Powerは負荷を駆動するためのスイッチ電力です。この和がダイナミック電力になり、約15.6mWです。これは125MHzでスイッチング電力を50%としたときの見積もりです。正確には合成後のシミュレーションでスイッチ率を出す必要があり、これは目安と思ってください。リーク電力はnWで多くないです。これは0.18μmプロセスで古いためです。

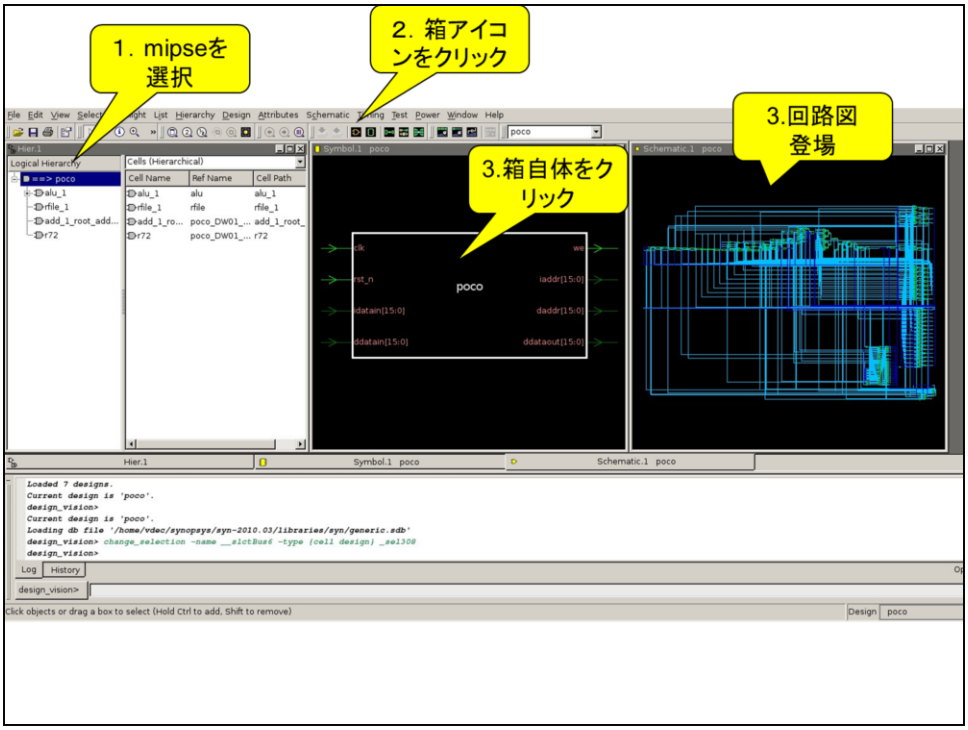
回路Viewer　design_vision

- design_visionはdc_shellのGUIで合成した回路を見たり解析したりするツール
- 実際の回路は巨大すぎるので、設計の現場ではあまり使わない

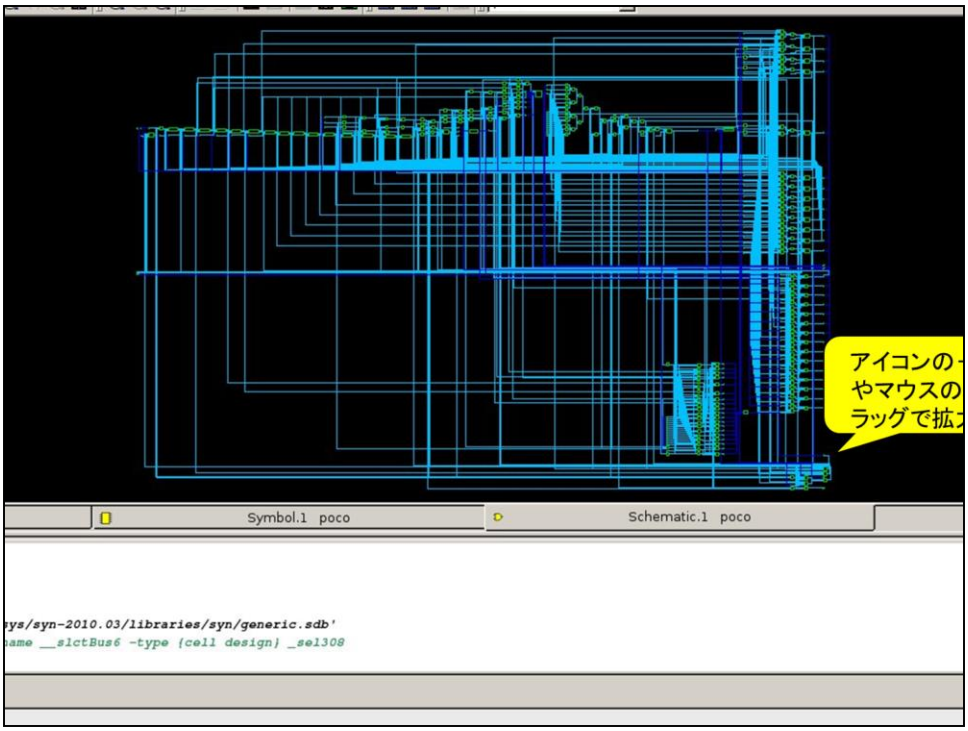
design_visionで立ち上がるので、

File→Read→mipse.ddcを選択して、Openをクリックしてみよう

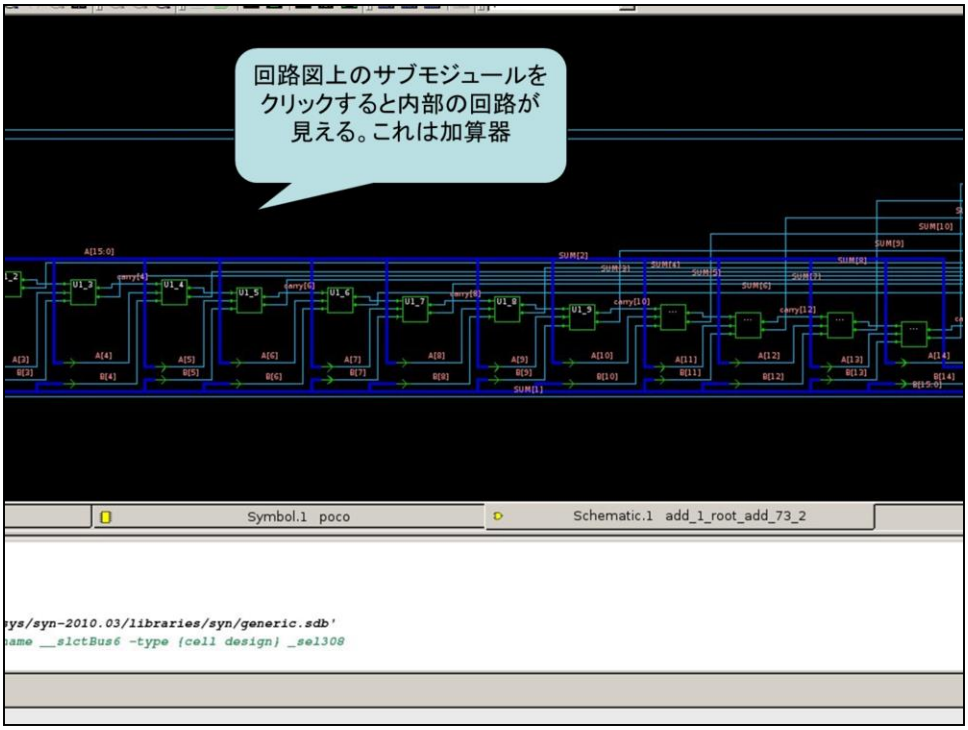
では次にdesign_visionについて解説しましょう。design_visionはdc_shellのGUIで、合成した回路を見たり解析したりするツールです。しかし、実際の設計では巨大すぎ、このツールで見てもわけがわからないので、あまり使われないです。しかしここでは雰囲気を掴むため、使ってみましょう。design_visionと打ち込むと立ち上がるので、File→Read-> mipse.ddcを選択してOpenをクリックしてみてください。



スライドの図に従い、pocoを選択、箱アイコンをクリックします。そうすると箱がサブウィンドウに出てくるのでここをクリックすると隣のウインドウに回路図が登場します。



アイコンの＋やマウスのドラッグで拡大します。ゲートのつなぎ方を見てみましょう。



回路中にはゲートだけではなく箱があり、これはサブモジュールになっています。これをクリックすると内部の回路が見えます。これは加算器です。MIPSeの回路を見てみましょう。

演習4

- サイコロのVerilog記述diceを合成せよ。
- dc_shell-t –f dice.tcl | tee dice.rptを実行
- slackが0になるまで、目標周期を小さくし、最大動作周波数、面積、電力を求めてレポートせよ。
- 合成した回路をdesign_visionで見てみよ。