プロジェクト実験 P2 CPU の論理設計　　岡本 吉央
平成 18 年 12 月 5 日　　1 回目資料

1 スケジュール

12/5 VHDL 入門，(一般論) モデル化について
12/8 VHDL 入門，(一般論) 検証・検査について
12/12 CPU の設計，(一般論) 仕様・設計について
12/15 CPU の設計，中間レポート〆切
12/19 CPU の設計
12/22 特別実験 (卒業研究) 報告会　報告会参加を強く推奨
出席を取るので，報告会の休憩時に岡本まで来ること
1/9 CPU の設計，発表のための班分け
1/12 CPU の設計
1/16 CPU の設計，発表原稿のチェック
(1/19) 午後休講
1/23 CPU の設計，発表練習 (C-203)
1/26 CPU の設計
1/30 CPU の設計，発表練習 (C-203)
2/2 CPU の設計
2/6 プロジェクト実験発表会 (A2-101)
(2/9) 予備日
2/13 レポート提出〆切

2 発表会について

- 発表時間：30 分 (発表 20 分，質疑 10 分)
- 形式：4 班 (1 班 3 ~ 4 人) に分けて，各班から 1 人ずつ発表者を選出．当日は 4 人で発表．
- 詳細は追って (おそらく 1 月 9 日に) 連絡．

3 P2 実験 Web ページ

- http://www.algo.ics.tut.ac.jp/~okamotoy/lect/P2/
- 配布資料の置場になるかもしれないが未定．

4 モデル化について

- (ソフトウエア工学における) モデル : 現実世界の問題領域を抽象化し，何らかの記述体系で表したもの．(論理学の「モデル」とは違うので注意)
- (ソフトウエア工学における) モデル化 : モデルの表現方法．現実世界の何に注目して抽象化を行なうかによって，様々なモデル化技法が存在する
  - データフローモデル :「データの流れが合流し，プロセスによって，変換・分岐され，新たなデータの流れを生む」として現実世界をモデル化．データフロー図では，頂点がプロセス，辺がデータの流れに対応
  - 制御フローモデル :「意味的にまとまりのある作業 (処理) が順次実行されている」として現実世界をモデル化．フローチャートで記述されるものを念頭に．UML の活動図 (activity diagram) では，頂点が処理や判断，辺が制御の流れに対応
  - オブジェクト指向モデル :「オブジェクトの集まり」として現実世界をモデル化．UML のクラス図 (class diagram) では，頂点がクラス，辺がクラス同士の汎化，集約，関連，依存などの関係に対応し，静的な関係が記述される．
  - 状態遷移モデル :「内部状態を持ち，環境との相互作用で内部状態を変化させる機械」として現実世界をモデル化．有限状態オートマトンで記述されるものを念頭に．UML の状態図 (state diagram) では，頂点が状態，辺が遷移に対応
  - 協調モデル :「動作主体が互いに情報をやりとりしながら動作している」として現実世界をモデル化．UML の系列図 (sequence diagram) では時間の流れが縦方向の辺で，情報の流れが横方向の辺で記述される．UML の協調図 (collaboration diagram) では頂点がオブジェクト，辺がオブジェクト間の情報の流れに対応
- 注意 1 : これらのモデルのいずれかが優れていて，いずれかが劣っているというものではない．各モデル化技法が互いの欠点を補いあって利用されるべきであり，実際そうされている．
- 注意 2 : どのモデルもグラフ構造を持っているが，グラフ構造を用いるためにグラフ理論を知る必要はない．しかし，各モデルではグラフの頂点や辺に与える「意味」が定められている．(そのようなものを意味論 (セマンティクス，semantics) と呼ぶ．) これによってモデル化における曖昧性を出来る限り排除しようとしている．
- 注意 3 : これらがモデル化技法のすべてではない．特にこれらのモデルと異質なものに「形式的モデル」と呼ばれるものがあり，これは論理や代数を中心して厳密にモデルを記述しようとするものである．例として，VDM，Z，CafeOBJ，LOTOS といったものがあり，いずれも処理系を備えた仕様記述言語である．
- 注意 4 : 論理回路の回路図はデータフローモデルと考えられる．このとき，頂点が論理ゲートであり，辺が真理値の流れに対応する．また，内部状態を持つ論理回路 (順序回路) は状態遷移モデルで記述できる．クロックによって同期がとられる回路は協調モデルで記述できる．本実験では，この 3 つのモデル化技法を通して CPU を扱うことになる．

5 VHDL について

5.1 概要

実験指導書 P2 の第 2 節 (101 ~ 106 ページ)，および，サブテキストの第 1 章を参照のこと．

- HDL (Hardware Description Language，ハードウエア記述言語) : ハードウエアを記述するためのプログラミング言語．
  - 利点 1 : ハードウエアをプログラムによって抽象化した記述が得られる．これによって，シミュレーションやデバッグが容易になり，設計が効率化される．
  - 利点 2 : ハードウエア合成ツールを用いることで，ゲート・レベルの設計を生成することができる (論理合成と呼ばれる)．
- VHDL と Verilog-HDL : 主要なハードウエア記述言語．本実験では VHDL を用いる．VHDL に関する詳しい情報は `http://www.vhdl.org/`で得られる．
- 本実験では Synopsys 社の提供するツールを使用する．他にも，GPL の下で GHDL というシミュレータが開発されている (現在は β バージョン)．

5.2 VHDL の基本概念

- エンティティ，エンティティ宣言とアーキテクチャ宣言
  - エンティティ：記述する 1 つ 1 つのモジュール
  - エンティティ宣言：エンティティの外部とのインタフェースを記述する部分
  - アーキテクチャ宣言：エンティティの内部動作を記述する部分．記述は部品の「接続関係 (配線)」と「動作」の 2 種類の情報によって行なうが，記述する機能レベルによって次のように分類される
    - ビヘイビア・レベル :「動作」のみを記述する．動作の細かいタイミングまでは規定せず，最も抽象度の高い記述である．
    - RTL (Register Transfer Level，レジスタ転送レベル) : 基本的な組み合わせ回路を単位として，その間の接続関係と動作タイミングを規定する．
    - ゲート・レベル : 論理ゲートとそれらの組合せを規定する．回路図をプログラムとして書き直しただけであり，最も抽象度が低い (すなわち，最も具体的な) 記述である．
- コンフィグレーション記述
  - コンフィグレーション記述 : 1 つのエンティティに対して複数のアーキテクチャが設計されているときに，その中のどれを選択するか指定する
- 同時処理文と逐次処理文
  - 同時処理文：普通の論理回路と同様に同時に処理される文．
  - 逐次処理文：普通のプログラミング言語と同様に上から順番に処理される文．

5.3 例：ハーフ・アダー

サブテキストのリスト 1.1 (19 ページ)，配布ファイルの test/HA.vhd．

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity HALF_ADDER is
5    port(
6      A, B  : in std_logic;
7      S, C0 : out std_logic);
8  end HALF_ADDER;
9
10 architecture DATAFLOW of HALF_ADDER is
11   signal C,D : std_logic;
12 begin
13   C <= A or B;
14   D <= A nand B;
15   C0 <= not D;
16   S <= C and D;
17 end DATAFLOW;
```

説明

- 1 行目：ライブラリ宣言．IEEE というライブラリを使用する．ちなみに IEEE とは The Insititute of Electrical and Electronics Engineers (電気電子工学会) の略であり，「アイトリプルイー」と発音される．電気電子分野最大の学会であり，様々な標準化活動も行なっている．
- 2 行目：パッケージ呼び出し．
- 4 ~ 8 行目：エンティティ宣言．
- 4 行目：このエンティティの名前を HALF_ADDER としている．
- 5 ~ 7 行目：外部とのインタフェースを記述する port 文．
- 6 行目：A，B は入力 (in) であり，そのデータ・タイプが std_logic であることを宣言．
- 7 行目：S，C0 は出力 (out) であり，そのデータ・タイプが std_logic であることを宣言．
- 8 行目：エンティティ宣言の終了．
- 10 ~ 17 行目：アーキテクチャ宣言．
- 10 行目：エンティティ HALF_ADDER のアーキテクチャとして DATAFLOW という名前を持つものをこのアーキテクチャ宣言で記述することを表明している．(1 つのエンティティに対して複数のアーキテクチャを設計可能であることに再度注意．)
- 11 行目：信号宣言．C，D が std_logic というデータ・タイプを持つ信号 (signal) であることを宣言している．このような宣言文は「begin ~ end」の前に置かれる．

- 12 行目 ~ 17 行目：begin と end で囲まれた箇所にある文 (13 行目 ~ 16 行目) はすべて同時に処理される (同時処理文)．普通の論理回路と同じような動作だと考えればよい．
- 13 行目：A と B の OR が C に代入される．信号の代入は「<=」で表す．
- 14 行目：A と B の NAND が D に代入される．
- 15 行目：D の NOT が C0 に代入される．
- 16 行目：C と D の AND が S に代入される．
- 17 行目：アーキテクチャ宣言の終了．

VHDL では大文字と小文字の区別をしないので，それも注意すること．

5.4 ソフトウエアの検証と検査，シミュレーションについて

総論については次回．今日は実験手順だけ．

設計した回路の動作確認用のプログラムも同じく VHDL で記述する．本実験で用いる Synopsys 社提供のツールにはシミュレータも付いているので，それを用いて動作確認を行なう．動作確認において予想しなかった振る舞いが観察された場合，プログラムに誤りがないか調べる．どのような動作確認プログラムを書くとよいか，という疑問からソフトウエア検証理論が生まれる．

5.5 例：ハーフ・アダーのシミュレーション

配布ファイルの `test/addertest.vhd`．(サブテキストで対応する箇所はなし．)

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity ADDER_TEST is
5  end ADDER_TEST;
6
7  architecture TESTBENCH of ADDER_TEST is
8    component HALF_ADDER
9      port (
10       A, B  : in  std_logic;
11       S, C0 : out std_logic);
12   end component;
13   signal IN1, IN2, S, C0 : std_logic;
14 begin  -- TESTBENCH
15   U0 : HALF_ADDER port map ( IN1, IN2, S, C0 );
16   process begin
17             IN1 <= '0'; IN2 <= '0';
18             wait for 30 ns;
19             IN1 <= '0'; IN2 <= '1';
20             wait for 30 ns;
21             IN1 <= '1'; IN2 <= '0';
22             wait for 30 ns;
23             IN1 <= '1'; IN2 <= '1';
24             wait;
25           end process;
26 end TESTBENCH;
27
28 configuration SIM_HA of ADDER_TEST is
29   for TESTBENCH
30   end for;
31 end SIM_HA;
```

説明

- 1 行目：ライブラリ宣言．IEEE というライブラリを使用する．
- 2 行目：パッケージ呼び出し．
- 4 ~ 5 行目：エンティティ宣言．
- 4 行目：このエンティティの名前を ADDER_TEST とする．
- 5 行目：エンティティ宣言の終了．シミュレーションの記述では，エンティティ宣言を空にする．(シミュレーションにおいては，入力や出力もシステムの構成要素と見ていて，そのシステムへの入出力はないため，ポートの記述が不要になるわけである．)
- 7 ~ 26 行目：アーキテクチャ宣言．
- 7 行目：エンティティ ADDER_TEST のアーキテクチャとして TESTBENCH という名前のものを記述していく．
- 8 ~ 12 行目：コンポーネント宣言．
- 8 行目：HALF_ADDER というコンポーネントの使用を宣言．これで，HA.vhd に記述した HALF_ADDER を参照できる．現在作業中のディレクトリにあるファイルは自動的に参照パスに入る．ただし，参照のためには HA.vhd が予めコンパイルされていないといけない．(詳しくはサブテキストの第 4.5 節．)
- 9 ~ 11 行目：このコンポーネントの入出力を port 文で記述．
- 10 行目：A，B は HALF_ADDER への入力 (in) であり，そのデータ・タイプは std_logic である．
- 11 行目：S，C0 は HALF_ADDER の出力 (out) であり，そのデータ・タイプは std_logic である．
- 12 行目：コンポーネント宣言の終了．
- 13 行目：信号宣言．IN1，IN2，S，C0 という信号を宣言し，それらのデータ・タイプは std_logic である．
- 14 ~ 26 行目：アーキテクチャ動作の本体．
- 14 行目：「--」はコメントの始まりを表す．「--」の直後から行末までがコメントとなる．
- 15 行目：コンポーネント・インスタンス文．U0 という名前を持つコンポーネントを作成する．コロンの後を見ると，そのアーキテクチャが HALF_ADDER であることが分かる．丸カッコの中の IN1，IN2，S，C0 がそれぞれ A，B，S，C0 に対応する．対応を明示する場合には

  ```
  port map ( A => IN1, B => IN2, S => S, C0 => C0 )
  ```

  と書く．(矢印の向きに注意．) サブテキストの第 1.4 節を参照のこと．
- 16 ~ 25 行目：プロセス文 (サブテキスト第 2 章)．このプロセス文の中身 (begin から end まで) は逐次処理文 であり，上から順に実行される．
- 17 行目：IN1 に'0'，IN2 に'0' を代入する．
- 18 行目：30 ns (ナノ秒) だけ待つ．

- 19 行目：IN1 に'0'，IN2 に'1' を代入する．
- 20 行目：30 ns (ナノ秒) だけ待つ．
- 21 行目：IN1 に'1'，IN2 に'0' を代入する．
- 22 行目：30 ns (ナノ秒) だけ待つ．
- 23 行目：IN1 に'1'，IN2 に'1' を代入する．
- 24 行目：永遠に待つ．
- 25 行目：プロセス文の終了．
- 26 行目：アーキテクチャ宣言の終了．
- 28 ~ 31 行目：コンフィグレーション文．(これは HA.vhd に無かった部分なので注意．) この部分でどのアーキテクチャをシミュレーションに用いるか記述する．
- 28 行目：エンティティ ADDER_TEST のコンフィグレーションとして SIM_HA という名前のものを記述するという宣言．
- 29 ~ 30 行目：TESTBENCH というアーキテクチャを指定している．ここで「for」が出て来ているが，これは反復処理とは全く関係がない．
- 31 行目：コンフィグレーション文の終了．

6 実験の進め方，ツールの使い方

- 初回に行なうこと
  - 実験で用いるツールは Redhat の古いバージョンでしか動かないため，再起動して，古いバージョンを用いる．
  - 必要なファイルのコピー
    > cp␣/home/ics/okamoto/P2.tgz␣. [enter]
    > tar␣xfvz␣P2.tgz [enter]
    これで p2.nagoya，p2.tokyo というファイルと test，p2 というディレクトリがカレントディレクトリに作られる．
  - ライセンスの取得
    > source␣p2.tokyo [enter]
  - ディレクトリ test 内のファイルで練習
    > cd␣test [enter]
    > vhdlan␣half.vhd [enter] (コンパイル)
    > vhdlan␣addertest.vhd [enter] (コンパイル)
    > scs␣SIM_HA [enter] (シミュレーション用ファイルの作成)
    > scirocco␣& [enter] (ウィンドウが開く)
    ウィンドウを操作してシミュレーションを行なう．
    1. はじめに「Simulator Invocation Dialog」というウィンドウが表示される．そのテキストダイアログの中の「scsim」を「./scsim」に書き換えて OK をクリックする．(スラッシュの前にピリオドが必要なので注意．)

2. 「Virsim」というウィンドウが表示される．
3. 2 段目の左から 5 番目のアイコンをクリックする．そうすると，「Hierarchy」というウィンドウが表示される．
4. 2 段目の左から 6 番目のアイコンをクリックする．そうすると，「Waveform」というウィンドウが表示される．
5. 動作確認をしたい信号を次のように選択する．Hierarchy ウィンドウの左側の囲みで，例えば，シミュレーション用ファイルのエンティティ名 (上の例では `ADDER_TEST`) をクリックする．そして，右側の囲みに現れる信号をクリックして選択する．複数選択したい場合はシフトを押しながらクリックする．最後に `Add` をクリックすると，選択された信号が Waveform ウィンドウの左の囲みに現れる．
6. Virsim ウィンドウの一番下の欄の Simulator Control で時間を進めながら，Waveform ウィンドウ上で波形の確認を行なう．Simulator Control の Step Time を 20ns に設定して `OK` をクリックすると時間が 20ns 進む．

- 毎回行なうこと (の手順)
  - 古い Redhat の起動
  - ライセンスの取得
  - vhdlan，scs，scirocco の実行
  - 実験終了時，古い Redhat を終了し，通常の Redhat を再起動
- 注意
  - 計算機室の旧バージョン Redhat が不調のため起動に失敗することが頻繁に起こる．そのときは教員または TA に申し出て，別のマシンを用いるようにする．
  - ライセンス取得において技科大外部にネットワーク接続するため，ネットワークが不調であるときや立ち上がっていないときには実験が行なえない．
  - `source␣p2.tokyo` を実行した kterm から Emacs を立ち上げると，漢字入力が行なえないことがある．そのときには，別の kterm を起動してそこから Emacs を実行すればよい．
  - 計算機室の初期設定環境では，Emacs で VHDL ファイルを開くと，そのプログラム中の予約語などに色が付いたり，入力時に自動補完が行なわれたりする．これはいわゆる vhdl-mode というものだが，それを解除するには，Emacs で「[Esc] x fundamental-mode」と入力すればよい．
- 実験の手順
  - 手始めに，サブテキスト 65 ページのリスト 3.2「イネーブル付き 12 進カウンタ」と 73 ページのリスト 3.6「イネーブル付き 12 進カウンタのシミュレーション記述」を入力，コンパイル，シミュレーションしてみる．
    (中間レポートの課題)
  - 実験指導書に従って進める (各回の配布資料も参照のこと)
    各モジュールを作成したら，その検証をシミュレーションによって随時行なう
    * 演習 1：ALU の設計，および，そのシミュレーション
    * 演習 2：1 ビットシフト回路の設計，および，そのシミュレーション
    * 演習 3：レジスタファイルの設計，および，そのシミュレーション
    * 演習 4：命令デコード回路の設計，および，そのシミュレーション

* 演習 5：プログラムカウンタの設計，および，そのシミュレーション
* 演習 6：メモリ回路の設計，および，そのシミュレーション
* 演習 7：シーケンサの設計，および，そのシミュレーション
* 演習 8：CPU 全体のシミュレーション (動作確認用のプログラムによる)
* 演習 8：CPU 全体のシミュレーション (オリジナル・プログラムによる)
* 演習 8：CPU 全体のシミュレーション (他学生作成プログラムによる)

7 経過報告

- 毎週火曜日の実験終了時，メーリングリストへ自分の進捗状況を簡単に報告する．
- 具体的には，12/5，12/12，12/19，1/9，1/16，1/23，1/30 の実験終了時．
- この経過報告は必ず行なうこと．行なわないと欠席扱いになるので注意．
- メーリングリスト：cpu@algo.ics.tut.ac.jp
- 質問などがあるときや情報共有をしたいときは，このメーリングリストを活用してもらいたい．このメーリングリストで緊急の連絡を行なう場合もある．
- 経過報告は，例えば「演習 4 命令デコード回路の設計が終了．次回は命令デコード回路のシミュレーションから行なう」などの記述で．表題 (Subject) は適当なもの，例えば 12/5 の経過報告については「経過報告 12/5」などとする．
- 他の人に比べて著しく遅れている場合は質問や相談をして，進めるように努力を．
- 安泰な進度の目安
  - 演習 1：~ 12/12
  - 演習 2：~ 12/12
  - 演習 3：~ 12/15
  - 演習 4：~ 12/19
  - 演習 5：~ 12/19
  - 演習 6：~ 1/9
  - 演習 7：~ 1/9
  - 演習 8 (動作確認)：~ 1/16
  - 演習 8 (オリジナル)：~ 1/23

8 成績評価

出席，レポート，実験態度により 40 点満点で評価する．(ちなみに，ハードウエア実験，ソフトウエア実験はどちらも 1 テーマ 10 点満点で評価する．)

- 出席：1 回でも欠席があると，「情報工学実験 I」としての単位を認められない．やむを得ない状況がある場合は，事前に申し出ること．
- レポート：中間レポートと最終レポートを課す．中間レポートと最終レポートの双方が提出され，受理された場合にのみレポートは採点される．どちらか一方でも提出されない場合は，「情報工学実験 I」としての単位を認められない．
- 実験態度：13:35 ~ 16:15 は実験時間なので集中すること．ただし，適宜休憩を取ってよい．2 月 6 日に行なわれるプロジェクト実験発表会での活発な議論も期待する．