第3回補足　データパスの基本

前回、テストベンチの意味が良くわからなかったようなので、以下、補足的な解説をする。

下が、前回使ったテストベンチだったが、これが分かりにくかった理由は、多分、(1) クロックと、テストベンチ中の時間経過が独立に宣言されていたので分かりにくかった。(2)com, addr は一度設定すると値を覚えてくれるので、毎回設定するのをサボった。このため、操作と与えるコマンドの対応が付きにくかった。(3) 必要な場所にのみメモリを表示する display 文を入れたために混乱した。(4) reset シーケンス中に最初のアドレスとコマンドの設定を行ったので、分かりにくかった。などによるのではないかと思う。

そこで、今回は以下の test3.v を使って説明する。

```
/* test bench */
`timescale 1ns/1ps
`include "def.h"
module test;
parameter STEP = 10;
   reg clk, rst_n;
   reg [`DATA_W-1:0] datain;
   reg [`SEL_W-1:0] com;
   reg [`ADDR_W-1:0] addr;
   reg we_n;
   wire [`DATA_W-1:0] accout;
   wire [`DATA_W-1:0] dmem2;
   reg [`DATA_W-1:0] dmem [`DEPTH-1:0];
   always @(posedge clk)
   begin
      if(!we_n) dmem[addr] <= accout;
   end
   datapath datapath_1(.clk(clk), .rst_n(rst_n), .com(com),
               .datain(dmem[addr]), .accout(accout));
   initial begin
      $dumpfile("datapath.vcd");
      $dumpvars(0,test);
      $readmemh("dmem.dat", dmem);
      clk <= `DISABLE;
      rst_n <= `ENABLE_N;
      we_n <= `DISABLE_N;
   #STEP
      rst_n <= `DISABLE_N;
      addr <= `ADDR_W'h00;
      com <= `ALU_THB;
   #(STEP*1/2)             // CLK 1 up
      clk <= `ENABLE;
   #(STEP*1/4)
      $display("com:%h addr:%h acc:%h we_n:%b", com, addr, accout, we_n);
      $display("dmem:%h %h %h %h", dmem[0], dmem[1], dmem[2], dmem[3]);
   #(STEP*1/4)
      clk <= `DISABLE;
```

```
      addr <= `ADDR_W'h01;
      com <= `ALU_ADD;
   #(STEP*1/2)             // CLK 2 up
      clk <= `ENABLE;
   #(STEP*1/4)
      $display("com:%h addr:%h acc:%h we_n:%b", com, addr, accout, we_n);
      $display("dmem:%h %h %h %h", dmem[0], dmem[1], dmem[2], dmem[3]);
   #(STEP*1/4)
      clk <= `DISABLE;
     addr <= `ADDR_W'h02;
      com <= `ALU_ADD;
   #(STEP*1/2)  // CLK 3 up
      clk <= `ENABLE;
   #(STEP*1/4)
      $display("com:%h addr:%h acc:%h we_n:%b", com, addr, accout, we_n);
      $display("dmem:%h %h %h %h", dmem[0], dmem[1], dmem[2], dmem[3]);
   #(STEP*1/4)
     clk <= `DISABLE;
     addr <= `ADDR_W'h03;
     com <= `ALU_THA;
     we_n <= `ENABLE_N;
   #(STEP*1/2)  // CLK 4 up
      clk <= `ENABLE;
   #(STEP*1/4)
      $display("com:%h addr:%h acc:%h we_n:%b", com, addr, accout, we_n);
      $display("dmem:%h %h %h %h", dmem[0], dmem[1], dmem[2], dmem[3]);
   #(STEP*1/4)
     clk <= `DISABLE;
     addr <= `ADDR_W'h02;
     com <= `ALU_THB;
     we_n <= `DISABLE_N;
     $finish;
   end
endmodule
```

前回のを反省して、まず CLK の上げ下げを他の信号と同じ initial 文中で行うようにした。どうせ CLK の上げ下げは定期的に行われるので、本来、これは無駄だが、分かりやすさを重視した。まず、リセット時にはコマンドとアドレスの設定はせずに、リセットを解除してから最初のクロックを立ち上げるようにした。実行するコマンドはやや変わっているので注意。

次にタイミング設定を以下のように明確にした。

- クロックの立ち下がりで、入力を設定
- 1/2 周期でクロックを立ち上げる (CLK up)
- 1/4 周期で結果を表示

を繰り返す。この様子を図 1 に示す。

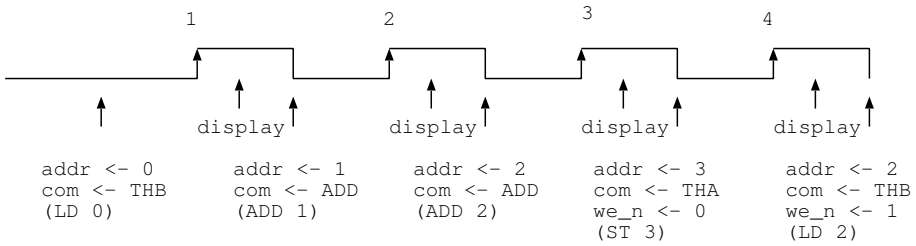

図 1: test3 のタイミング図

ちょうどこれと同じ波形を gtkwave で見ることができるので試みられたい。しかし、display のタイミングは表示できないので注意。このコマンドとアドレスをデータパスに与えることで、

```
LD 0
ADD 1
ADD 2
ST 3
LD 2
```

を実行している。それでは前回の演習（演習 3）ができていない人は test3.v に対して改造する形でこれを実現せよ。
今回より、Subject の最後に演習番号を入れて欲しい。
Subject: PARTHENON 学籍番号　名前　演習 3
のようにしてくれると嬉しい。

演習 3(前回と同じ)

メモリ付きのデータパス

前回改造した ALU を使って、以下の処理を test2.v を用いて行う。X,Y,W,Z が先の例通り、0,1,2,3 番地に格納されているとする。(X-Z) AND (Y-W) の演算を記号を使って示せ。なお提出物は test3.v を改造したものとする。