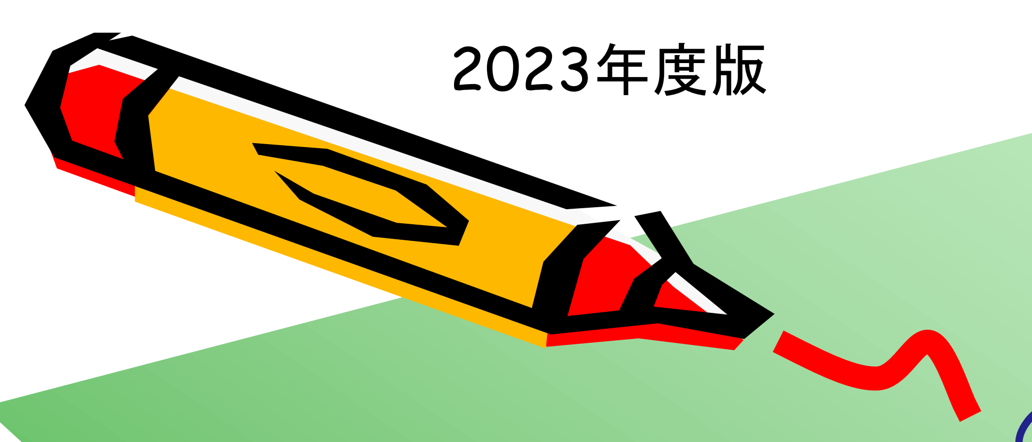
2023年度版

Ver. 2023-05-05a



Course number: CSC.T341

コンピュータ論理設計 演習(4)
Computer Logic Design Exercise(4)

情報工学系　荒堀喜貴

Yoshitaka ARAHORI, Department of Computer Science

arahori_at_c.titech.ac.jp

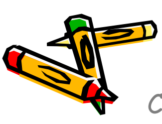
Computer Logic Design support page https://www.arch.cs.titech.ac.jp/lecture/CLD/

コンピュータ論理設計 演習(Exercise)の注意点

- 演習はACRiルームを利用します。
- 3～4人のグループを作成します．そのグループ内で情報を共有しながら演習を進めてください．
- 問題はグループ内で相談して解決する，あるいは，担当のTA(Teaching Assistant)や教員に質問してください．
- 演習には出席点があります．休まずにきちんと出席しましょう．
- 演習スライドにチェックポイントの図がある場所は，作業を確認してもらう場所です．すべてのチェックポイントをクリアしましょう．

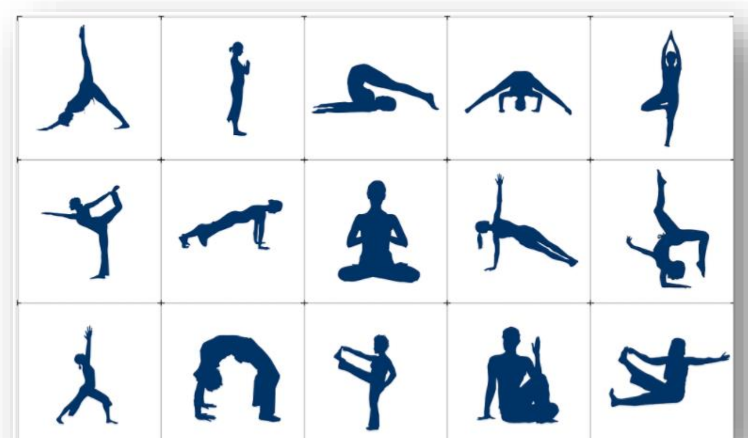
Check Point 1

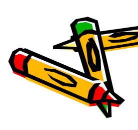
- 演習時間でなくてもACRiルームを利用できます．現在は，1日に4枠（3時間 x 4枠 = 12時間）を利用できます。独自のハードウェア設計などに挑戦しましょう．

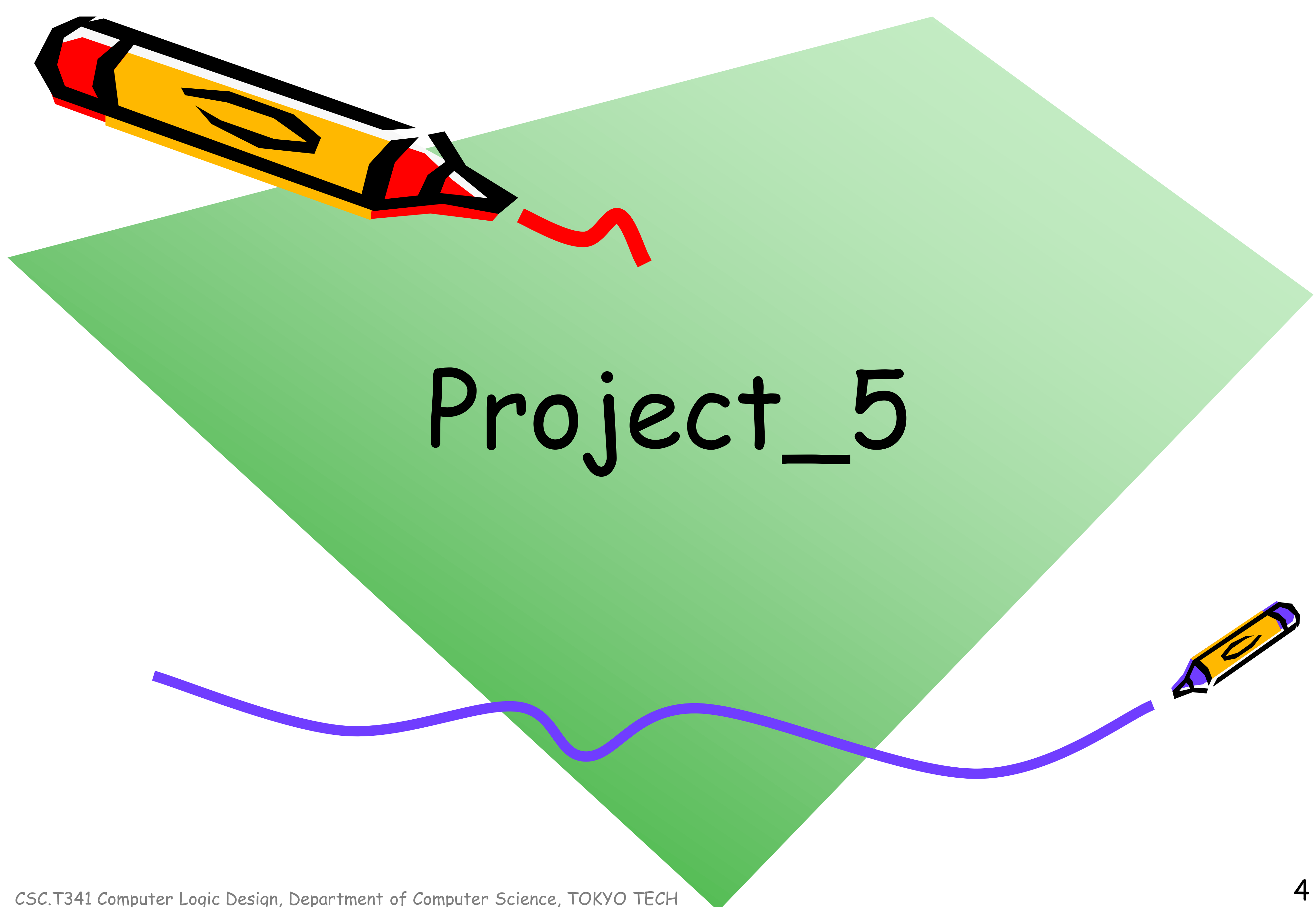


Exercise(4)

- Project_5
  - シングルサイクルプロセッサ m_proc02 の仕組みを理解する.
  - その正しさをシミュレーションで確認する.
  - プロセッサを FPGA で動作させ，挙動が正しいことを VIO で確認する.
- Project_6
  - シングルサイクルプロセッサ m_proc03 の仕組みを理解する.
  - Verilog HDLのコードを修正する.
  - その正しさをシミュレーションと波形で確認する.






Project_5

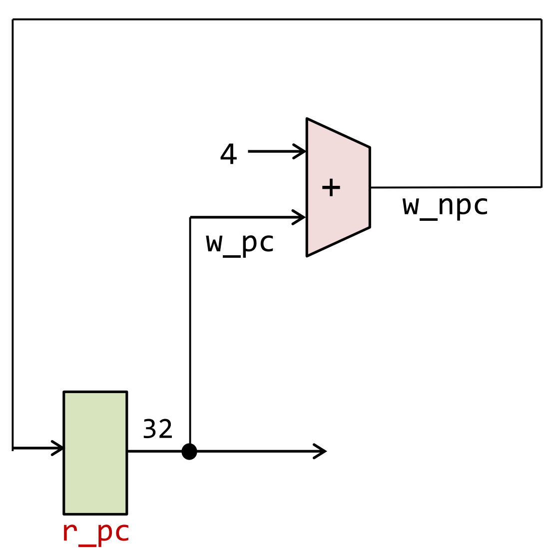
新しい Vivado プロジェクトの作成とファイルの登録

- 前回の演習を参考に，Vivado で新しいプロジェクト project_5 を作成する.
- Ubuntu で起動したターミナルで，次のコマンドを実行してファイルをコピーする.
  - /home/tu_kise は automount のディレクトリなので，アクセスしないとファイルが見えない．tabキーによる補完がうまく動作しないことがあるので注意する.
  - 最後の ls コマンドで，code102.v, code111.v, code112.v, code113.v, code115.v, main11.xdc が表示されることを確認.

```
$ ls /home/tu_kise
$ cd ~/cld/project_5
$ cp /home/tu_kise/cld/2023/code102.v .
$ cp /home/tu_kise/cld/2023/code111.v .
$ cp /home/tu_kise/cld/2023/code112.v .
$ cp /home/tu_kise/cld/2023/code113.v .
$ cp /home/tu_kise/cld/2023/code115.v .
$ cp /home/tu_kise/cld/2023/main11.xdc .
$ ls
```

- Vivado で，project_5 の制約ファイルとして main11.xdc を登録する.
  - main11.xdc ファイルの内容を変更する必要はない.
- Vivado で，project_5 のVerilog HDLファイルとして code115.v を登録する

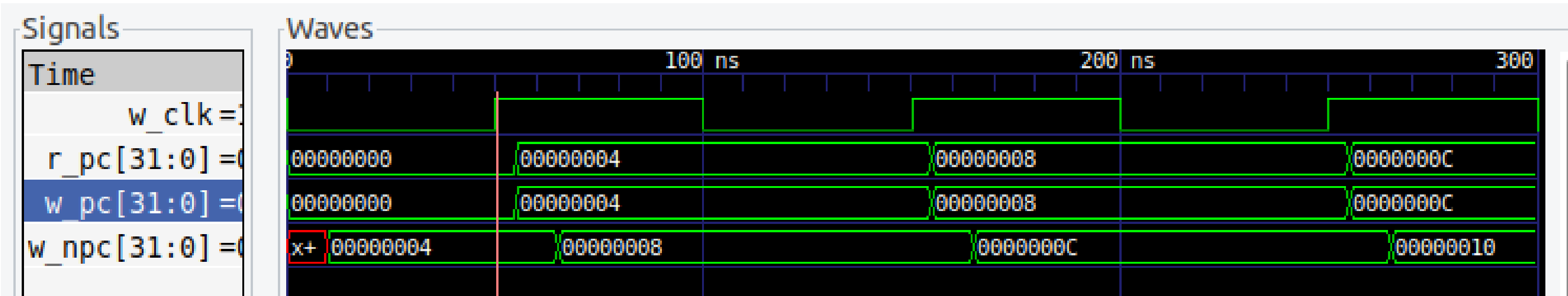
m_proc01 プロセッサの設計と実装に向けた一歩



code102.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [31:0] w_pc;
  m_main m_main0 (r_clk, w_pc);
  always@(*) #1 $write("%3d %x¥n", $time, w_pc);
  initial $dumpfile("main.vcd"); /* file name for GTKWave */
  initial $dumpvars(0, m_top);   /* module    for GTKWave */
  initial #300 $finish();
endmodule

module m_main (w_clk, w_pc);
  input  wire w_clk;
  output wire [31:0] w_pc;

  reg [31:0] r_pc = 0;
  assign w_pc = r_pc;
  wire [31:0] #10 w_npc = w_pc + 4;
  always@(posedge w_clk) #5 r_pc <= w_npc;
endmodule
```

```
  1 00000000
 56 00000004
156 00000008
256 0000000c
```



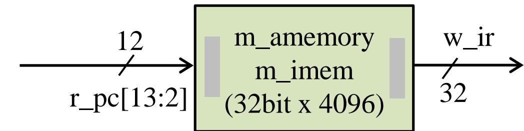
m_amemory 非同期式メモリの記述とシミュレーション

- Verilog HDLでは，ビット幅BでワードWのメモリ m を reg [B-1:0] m [0:W-1] として宣言できる.
- 読み出す動作でクロック信号を利用しないメモリを非同期メモリ (asynchronous memory) と呼ぶ.
- 非同期式メモリの記述例を示す．シミュレーションでの読み出しの遅延を 20nsec とした．w_addr で指定されたアドレスの内容を読み出す．posedge w_clk のタイミングで，w_we (write enable) が1の時に，w_addr で指定されたアドレスに w_din (data in) の値を書き込む.
- このコード (code111.v) をシミュレーションして，波形を確認すること.

```
module m_amemory (w_clk, w_addr, w_we, w_din, w_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output wire [31:0] w_dout;

  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) if (w_we) cm_ram[w_addr] <= w_din;
  assign #20 w_dout = cm_ram[w_addr];

  initial begin
    cm_ram[0]={7’d0, 5’d0, 5’d0, 3’d0, 5’d0, 7’b0110011}; // add x0, x0, x0
    cm_ram[1]={7’d0, 5’d1, 5’d0, 3’d0, 5’d4, 7’b0110011}; // add x4, x0, x1
    cm_ram[2]={7’d0, 5’d2, 5’d1, 3’d0, 5’d5, 7’b0110011}; // add x5, x1, x2
    cm_ram[3]={7’d0, 5’d5, 5’d4, 3’d0, 5’d6, 7’b0110011}; // add x6, x4, x5
    cm_ram[4]={7’d0, 5’d0, 5’d0, 3’d0, 5’d0, 7’b0110011}; // add x0, x0, x0
  end
endmodule
```

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  reg [31:0] r_pc = 0;
  always @(posedge r_clk) r_pc <= #3 r_pc + 4;

  wire [31:0] w_data;
  m_amemory m (r_clk, r_pc[13:2], 1'd0, 32'd0, w_data);

  initial $dumpfile("main.vcd"); /* file name for GTKWave */
  initial $dumpvars(0, m_top);   /* module    for GTKWave */
  initial #1000 $finish;
  always@(*) #80 $write("%3d %d %x¥n", $time, r_pc, w_data);
endmodule
```

code111.v



31 — 25	24 — 20	19 — 15	14 — 12	11 — 7	6 — 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type

Register file, レジスタファイル m_regfile の実装

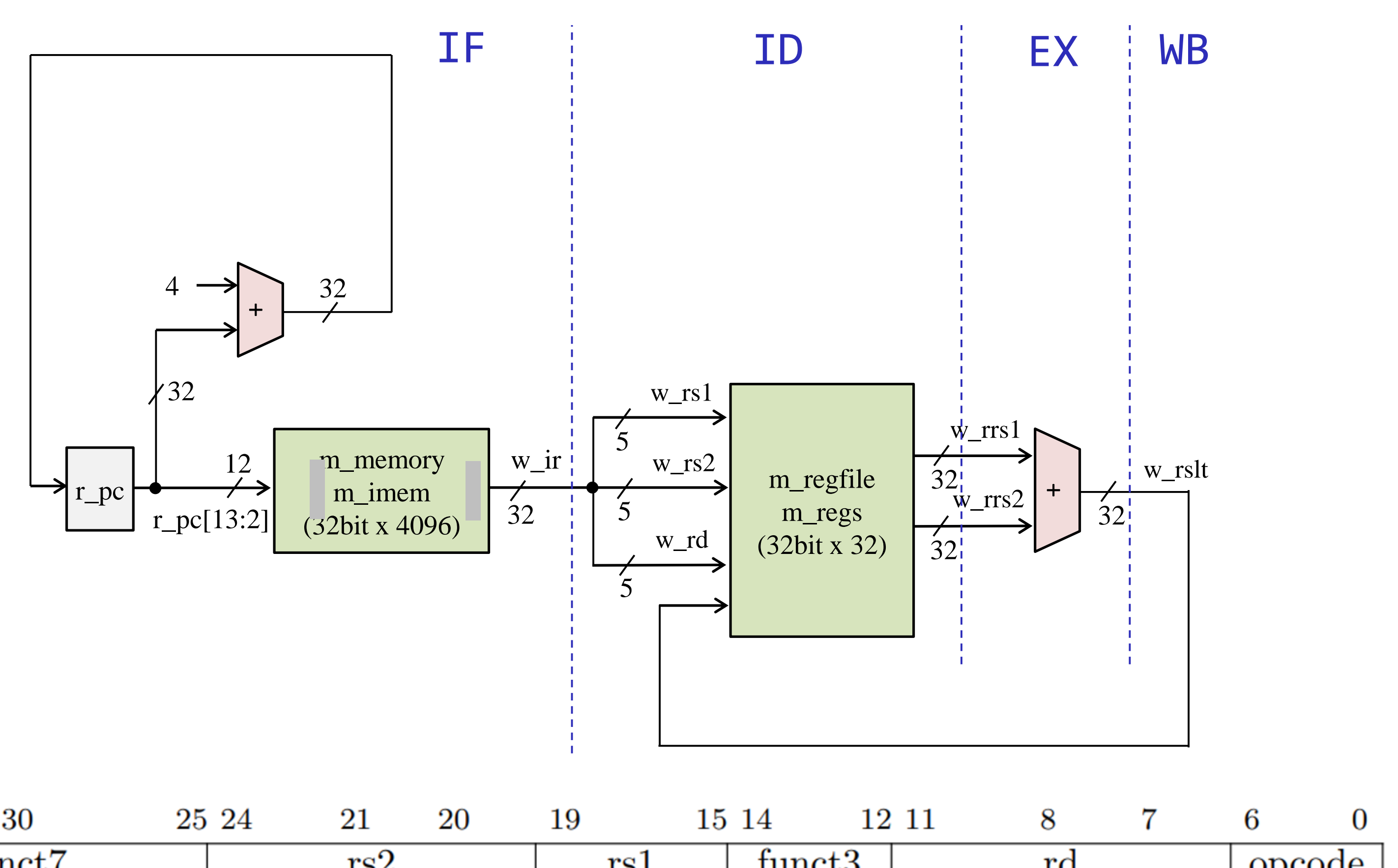
- Verilog HDLでは，ビット幅Bでワード数Wのメモリ m を reg [B-1:0] m [0:W-1] として宣言できる.
- w_rr1 で指定したレジスタの値を読み出し w_rdata1 に出力する．非同期の読み出し.
- w_rr2 で指定したレジスタの値を読み出し w_rdata2 に出力する．非同期の読み出し.
  - ただし，x0 (zero) の読み出しは，値0を出力する.
- posedge w_clk のタイミングで，w_we (write enable) が1の時に， w_wr (write register) で指定されたレジスタに w_wdata (write data) の値を書き込む.
- このモジュールではadd命令の動作確認のために x1 を 1 で，x2 を 2 で初期化している.

code112.v

```
module m_regfile (w_clk, w_rr1, w_rr2, w_wr, w_we, w_wdata, w_rdata1, w_rdata2);
  input  wire        w_clk;
  input  wire [4:0]  w_rr1, w_rr2, w_wr;
  input  wire [31:0] w_wdata;
  input  wire        w_we;
  output wire [31:0] w_rdata1, w_rdata2;

  reg [31:0] r[0:31];
  assign w_rdata1 = (w_rr1==0) ? 0 : r[w_rr1];
  assign w_rdata2 = (w_rr2==0) ? 0 : r[w_rr2];
  always @(posedge w_clk) if(w_we) r[w_wr] <= w_wdata;

  initial r[1] = 1;
  initial r[2] = 2;
endmodule
```

m_proc02 addを処理するシングルサイクルのプロセッサ

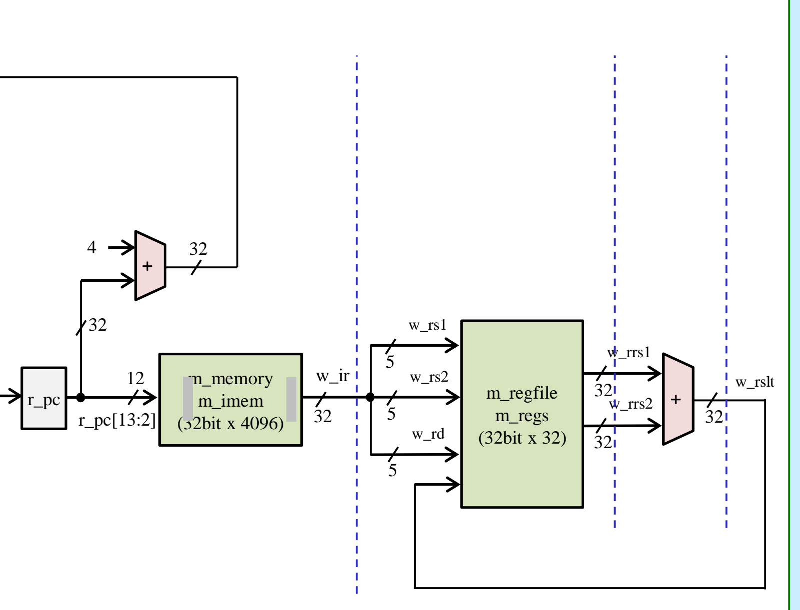
- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), ライトバック(WB) の処理をおこなう加算命令 (add) のみに対応したプロセッサのブロック図



m_proc02 addを処理するシングルサイクルのプロセッサ

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), ライトバック(WB)の処理をおこなう加算命令 (add) のみに対応したプロセッサのブロック図
- このプロセッサ (code113.v)で, code111.v の命令列(p.7)を実行するときの配線の値を考える.

code113.v の一部




```
module m_proc02 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) #5 if(w_ce & r_pc[13:2]!=4) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);

  wire [4:0]  w_rs1 = w_ir[19:15];
  wire [4:0]  w_rs2 = w_ir[24:20];
  wire [4:0]  w_rd  = w_ir[11:7];
  wire [31:0] w_rrs1, w_rrs2, w_rslt;
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_ce, w_rslt, w_rrs1, w_rrs2);

  assign #10 w_rslt = w_rrs1 + w_rrs2;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_rd==6) r_led <= w_rslt;
  assign w_led = r_led;
endmodule
```

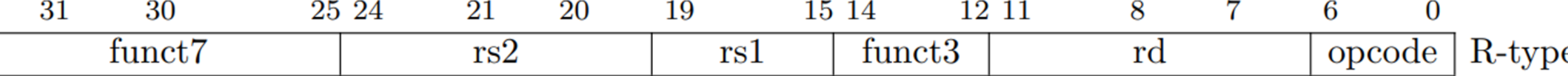
31 ... 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type

m_proc02 addを処理するシングルサイクルのプロセッサ

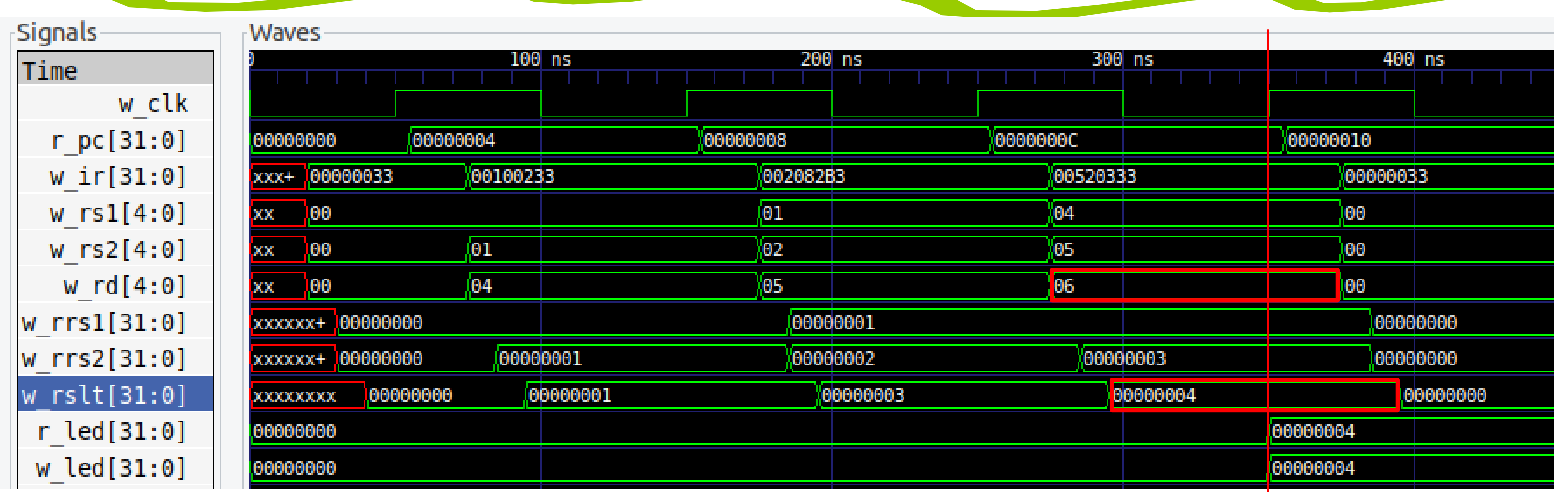
- code113.v をシミュレーションして，その波形を確認すること.
- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), ライトバック(WB) の処理をおこなう加算命令 (add) のみに対応したプロセッサ
- m_proc02 のインスタンス名を p とする. p の内部の r_pc は，ピリオドを用いて p.r_pc として参照できる.
- 同様に，p に含まれるインスタンス m_reg の内部の r[1] は， p.m_reg.r[1] として参照できる。

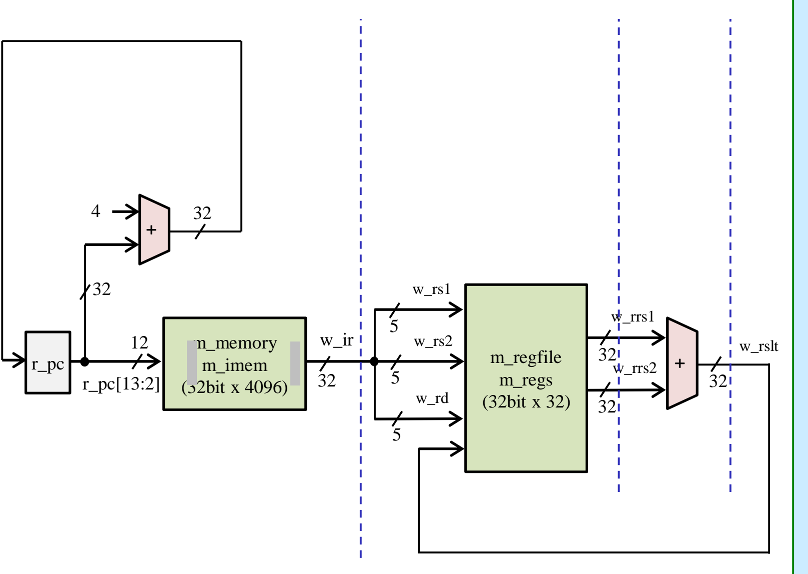
code113.v の一部

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [31:0] w_led;
  m_proc02 p (r_clk, 1'b1, w_led);
  initial $dumpfile("main.vcd"); /* file name for GTKWave */
  initial $dumpvars(0, m_top);   /* module    for GTKWave */
  initial #550 $finish;
  always@(posedge r_clk) #1 $write("%4d %x: %x  %x -> %x¥n",
                                  $time, p.r_pc, p.w_rrs1, p.w_rrs2, p.w_rslt);
endmodule
```



m_proc02 addを処理するシングルサイクルのプロセッサ





code113.v の一部

```
module m_proc02 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) #5 if(w_ce & r_pc[13:2]!=4) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);

  wire [4:0]  w_rs1 = w_ir[19:15];
  wire [4:0]  w_rs2 = w_ir[24:20];
  wire [4:0]  w_rd  = w_ir[11:7];
  wire [31:0] w_rrs1, w_rrs2, w_rslt;
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_ce, w_rslt, w_rrs1, w_rrs2);

  assign #10 w_rslt = w_rrs1 + w_rrs2;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_rd==6) r_led <= w_rslt;
  assign w_led = r_led;
endmodule
```

w_led の出力の値はどうなるか？

m_proc02 addを処理するシングルサイクルのプロセッサ

- FPGA で動作させるためのコード code115.v の内容を理解すること.
- 50MHz のクロック信号を生成するように clk_wiz_0 を生成する.
- 32ビットの入力を持つように vio_0 を生成する.
- FPGA で動作させたときの VIO の値はどうなるか？

```
module m_proc02 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) #5 if(w_ce & r_pc[13:2]!=4) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);

  wire [4:0]  w_rs1 = w_ir[19:15];
  wire [4:0]  w_rs2 = w_ir[24:20];
  wire [4:0]  w_rd  = w_ir[11:7];
  wire [31:0] w_rrs1, w_rrs2, w_rslt;
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_ce, w_rslt, w_rrs1, w_rrs2);

  assign #10 w_rslt = w_rrs1 + w_rrs2;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_rd==6) r_led <= w_rslt;
  assign w_led = r_led;
endmodule
```

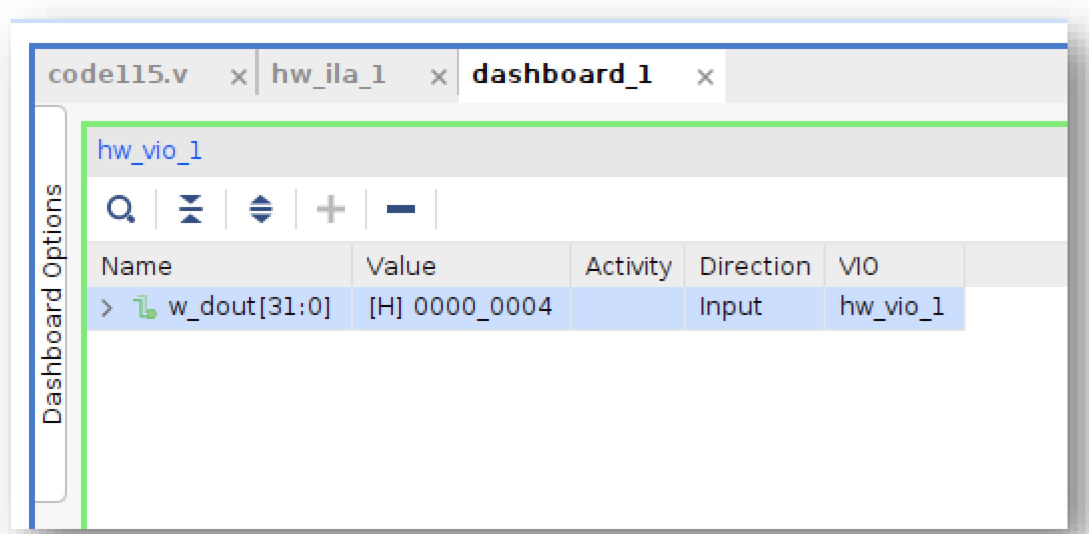
w_led の出力の値はどうなるか？

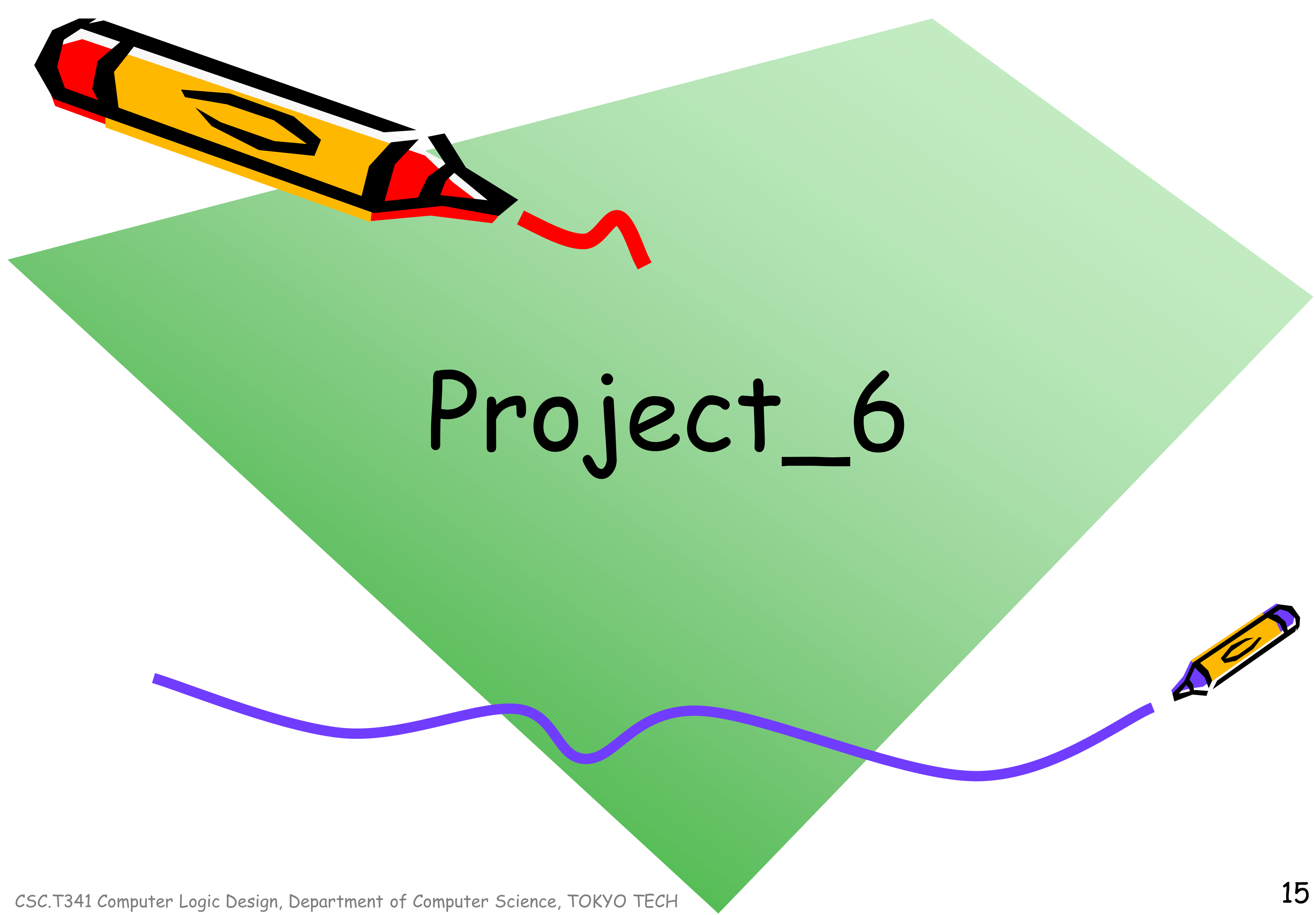
```
module m_main (w_clk, w_led);
  input  wire w_clk;
  output wire [3:0] w_led;

  wire [31:0] w_dout;
  wire w_clk2, w_locked;
  clk_wiz_0 clk_w0 (w_clk2, 0, w_locked, w_clk);
  vio_0 vio_00(w_clk2, w_dout);

  m_proc02 p (w_clk2, w_locked, w_dout);

  reg [3:0] r_led = 0;
  always @(posedge w_clk2)
    r_led <= {^w_dout[31:24], ^w_dout[23:16], ^w_dout[15:8], ^w_dout[7:0]};
  assign w_led = r_led;
endmodule
```

code115.v の一部

code115.v のFPGAへの実装と動作確認

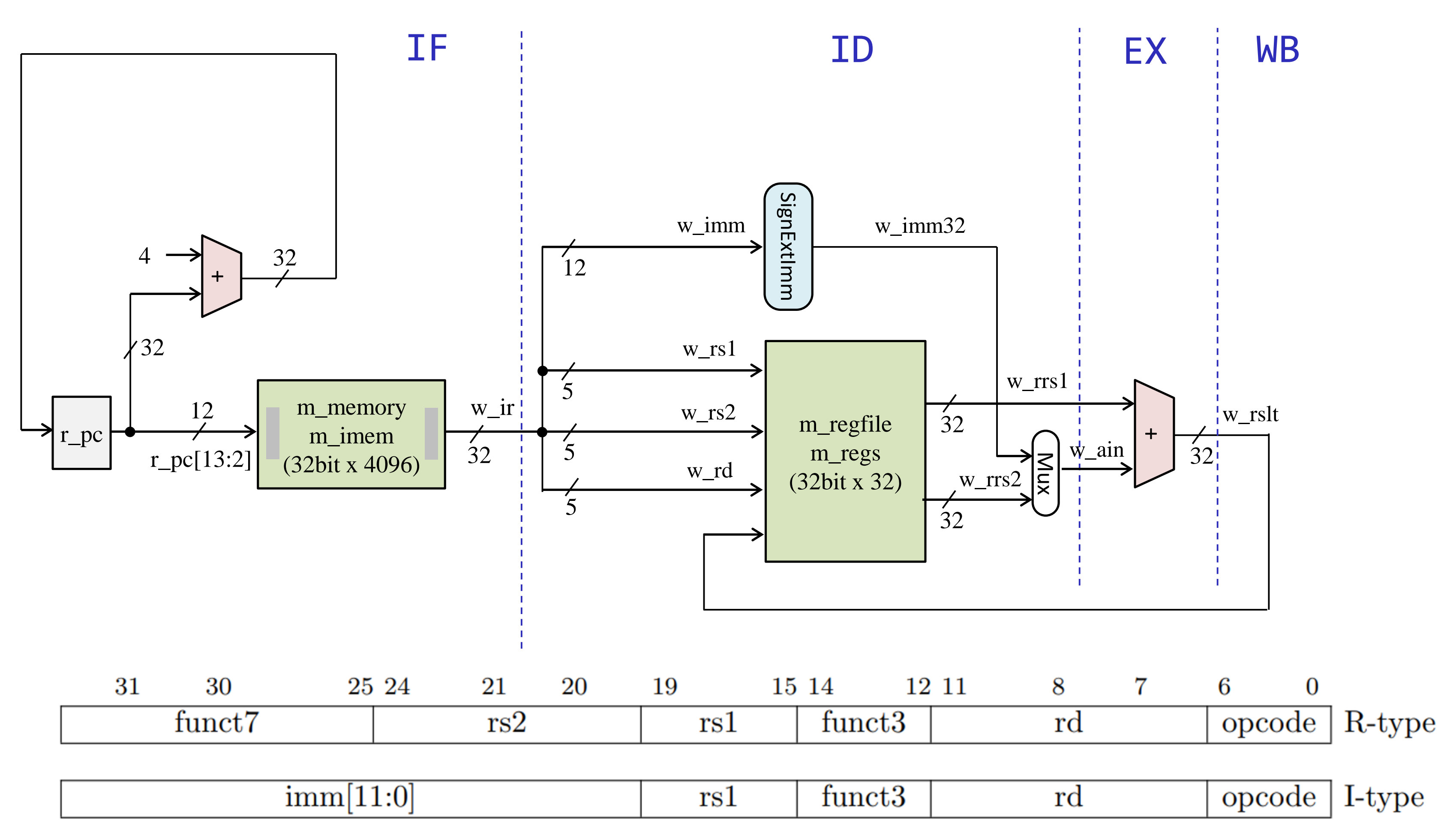
- Vivado のプロジェクトを開き，code115.v, main11.xdc を登録する.
- 50MHz のクロック信号を生成するように clk_wiz_0 を生成する.
- 32ビットの入力を持つように vio_0 を生成する.
- Vivado で論理合成，配置・配線してbitstreamファイルを生成する.（ここでエラーになる場合には，デバッグすること．）
- FPGAをコンフィギュレーションして， VIOの値が想定通りになることを確認する.
- 「担当の教員あるいはTA」にVIOの値を確認してもらうこと.




Project_6

m_proc03 add と addi を処理するプロセッサ

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), ライトバック(WB) の処理をおこなう加算命令(add, addi)に対応したプロセッサのブロック図



m_proc03 add と addi を処理するプロセッサ

- /home/tu_kise/cld/2023/code150.v を作業用のディレクトリにコピーする.
- code150.v を修正すること.
- 波形を確認すること.

```
module m_proc03 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) #5 if(w_ce & r_pc[13:2]!=4) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);

  wire [4:0]  w_rs1 = w_ir[19:15];
  wire [4:0]  w_rs2 = w_ir[24:20];
  wire [4:0]  w_rd  = w_ir[11:7];
  wire [31:0] w_rrs1, w_rrs2, w_rslt;
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_ce, w_rslt, w_rrs1, w_rrs2);

  /***** Please describe this part by yourself *****/


  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_rd==6) r_led <= w_rslt;
  assign w_led = r_led;
endmodule
```

```
module m_amemory (w_clk, w_addr, w_we, w_din, w_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output wire [31:0] w_dout;

  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) if (w_we) cm_ram[w_addr] <= w_din;
  assign #20 w_dout = cm_ram[w_addr];

  initial begin
    cm_ram[0]={7'd0, 5'd0, 5'd0, 3'd0, 5'd0, 7'b0110011}; // add  x0, x0, x0
    cm_ram[1]={12'h008,      5'd0, 3'd0, 5'd4, 7'b0010011}; // addi x4, x0,  8
    cm_ram[2]={12'hffe,      5'd0, 3'd0, 5'd5, 7'b0010011}; // addi x5, x0, -2
    cm_ram[3]={7'd0, 5'd5, 5'd4, 3'd0, 5'd6, 7'b0110011}; // add  x6, x4, x5
    cm_ram[4]={7'd0, 5'd0, 5'd0, 3'd0, 5'd0, 7'b0110011}; // add  x0, x0, x0
  end
endmodule
```

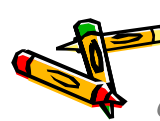
code150.v の一部

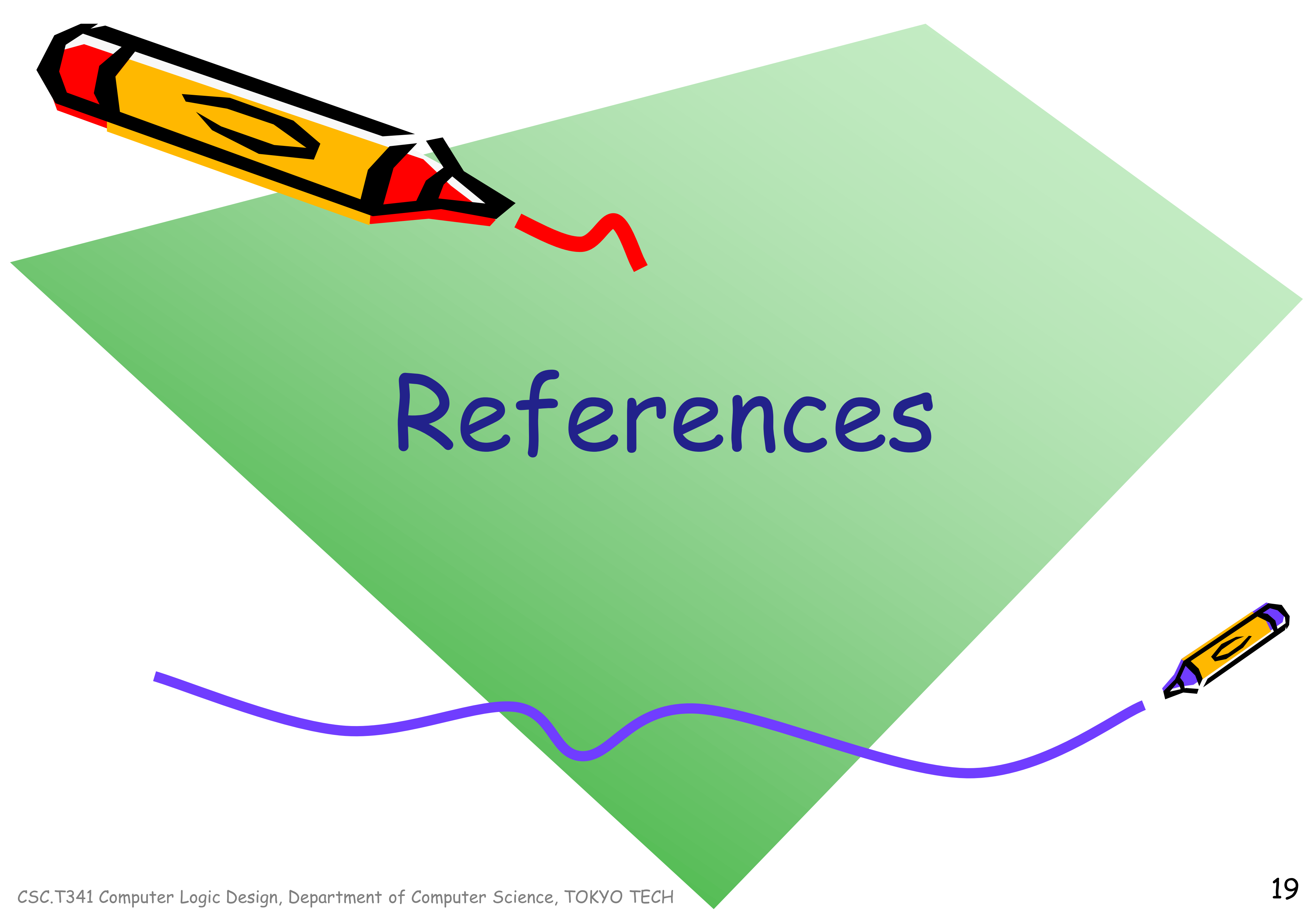
31 30 25	24 21 20	19 15	14 12	11 8 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type

code150.v の修正とシミュレーションによる動作確認

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), ライトバック(WB) の処理をおこなう加算命令(add, addi)に対応したプロセッサの動作になるように, code150.v のモジュール m_proc03 を修正すること.
- iverilog でシミュレーションして, 得られる波形が正しいものであることを確認すること.
- 「担当の教員あるいはTA」に波形と記述した m_proc03 のコードを確認してもらうこと.

Check Point 6




References

References

- Computer Logic Design support page
  - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
  - https://gw.acri.c.titech.ac.jp
- ACRi Blog
  - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
  - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog