Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

9. プロセッサの基本構成要素:レジスタファイルとメモリ
Basic Components of Processor: Register File and Memory

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
W621 講義室　月 10:45-12:15, 木 9:00-12:15

プロセッサが命令を処理するための基本的な5つのステップ

- IF (Instruction Fetch)
  メモリから命令をフェッチする.
- ID (Instruction Decode)
  命令をデコード(解読)しながら, レジスタの値を読み出す.
- EX (Execution)
  命令操作の実行またはアドレスの生成を行う.
- MEM (Memory Access)
  必要であれば, データ・メモリ中のオペランドにアクセスする.
- WB (Write Back)
  必要であれば, 結果をレジスタに書き込む.

Inside module m_memory

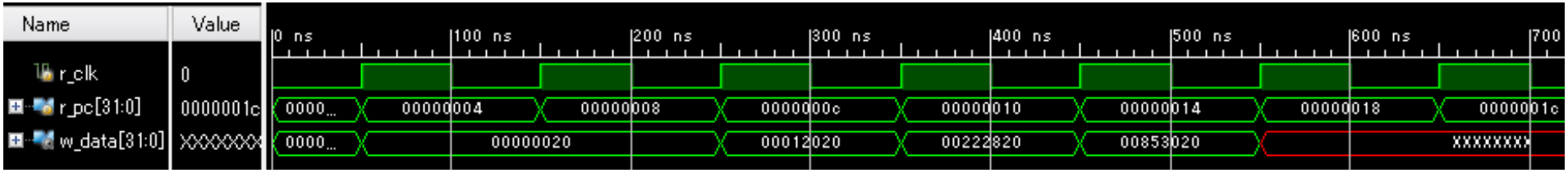
- メモリの記述の例を示す.
- main.vを code084.v と code091.v の内容となるように入力して, シミュレーションする.

code084.v

```
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 <= r_pc + 4;

  wire [31:0] w_data;
  m_memory m_memory0 (r_clk, r_pc[13:2], 0, 0, w_data);

  always@(*) #1 $write("%3d %d %x¥n", $time, r_pc, w_data);
endmodule
```

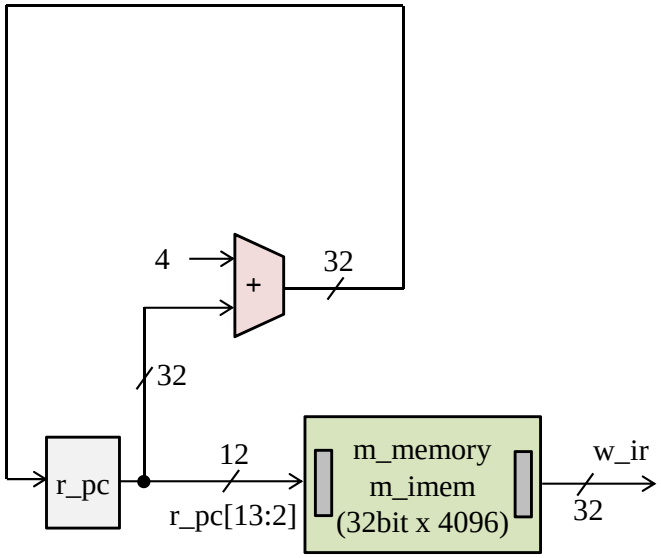
code091.v

```
module m_memory (w_clk, w_addr, w_we, w_din, r_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output reg  [31:0] r_dout;
  reg r_we=0;
  reg [11:0] r_addr=0;
  reg [31:0] r_din=0;
  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) begin
    r_addr <= w_addr;
    r_din  <= w_din;
    r_we   <= w_we;
    r_dout <= cm_ram[r_addr];
    if (r_we) cm_ram[r_addr] <= r_din;
  end
  initial begin
    r_dout = 0;
    cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add $0, $0, $0
    cm_ram[1] = {6'h0, 5'd0, 5'd1, 5'd4, 5'h0, 6'h20}; // add $4, $0, $1
    cm_ram[2] = {6'h0, 5'd1, 5'd2, 5'd5, 5'h0, 6'h20}; // add $5, $1, $2
    cm_ram[3] = {6'h0, 5'd4, 5'd5, 5'd6, 5'h0, 6'h20}; // add $6, $4, $5
  end
endmodule
```



Inside module m_proc01

- プロセッサの実装に向けた最初の版．命令をフェッチ (fetch) するだけのモジュール．
- 命令メモリから命令を読み出し，取得することを命令フェッチと呼ぶ．



code092.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc01 p (r_clk, 0, 0, w_led);
  always@(*) #1 $write("%3d %x¥n", $time, p.w_ir);
endmodule

module m_proc01 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  assign w_led = (w_btnu | w_btnd) ? w_ir[31:16] : w_ir[15:0];
endmodule
```

Inside module m_proc01

- プロセッサの実装に向けた最初の版．命令をフェッチ (fetch) するだけのモジュール．命令メモリから命令を読み出し，取得することを命令フェッチと呼ぶ．
- main.vを code092.v と code091.v の内容となるように入力して，シミュレーションする．

code092.v

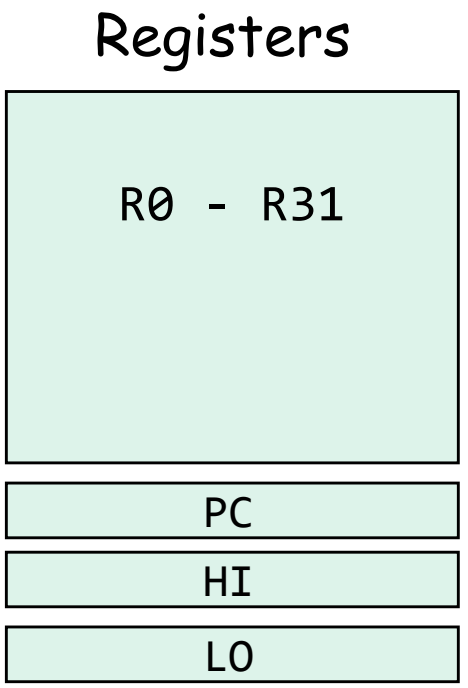
```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc01 p (r_clk, 0, 0, w_led);
  always@(*) #1 $write("%3d %x¥n", $time, p.w_ir);
endmodule

module m_proc01 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  assign w_led = (w_btnu | w_btnd) ? w_ir[31:16] : w_ir[15:0];
endmodule
```

code091.v

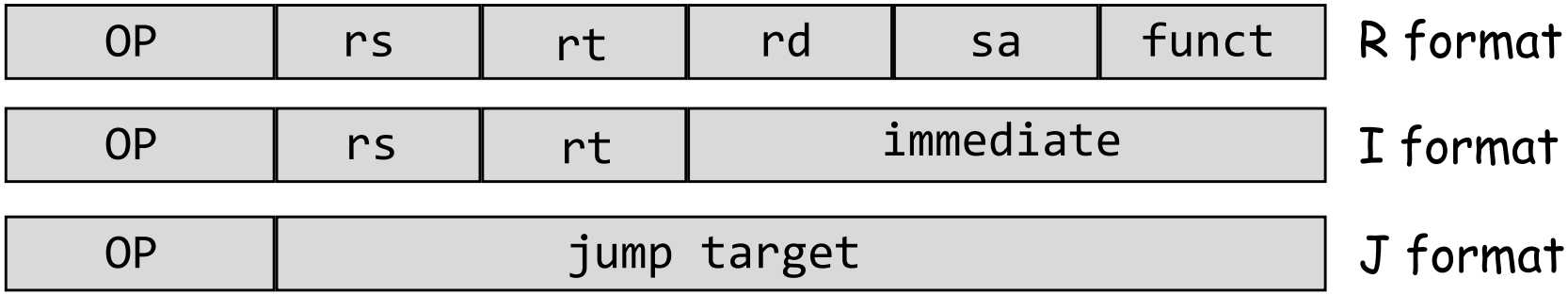
```
module m_memory (w_clk, w_addr, w_we, w_din, r_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output reg  [31:0] r_dout;
  reg r_we=0;
  reg [11:0] r_addr=0;
  reg [31:0] r_din=0;
  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) begin
    r_addr <= w_addr;
    r_din  <= w_din;
    r_we   <= w_we;
    r_dout <= cm_ram[r_addr];
    if (r_we) cm_ram[r_addr] <= r_din;
  end
  initial begin
    r_dout = 0;
    cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add $0, $0, $0
    cm_ram[1] = {6'h0, 5'd0, 5'd1, 5'd4, 5'h0, 6'h20}; // add $4, $0, $1
    cm_ram[2] = {6'h0, 5'd1, 5'd2, 5'd5, 5'h0, 6'h20}; // add $5, $1, $2
    cm_ram[3] = {6'h0, 5'd4, 5'd5, 5'd6, 5'h0, 6'h20}; // add $6, $4, $5
  end
endmodule
```

MIPS R3000 Instruction Set Architecture (ISA)

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

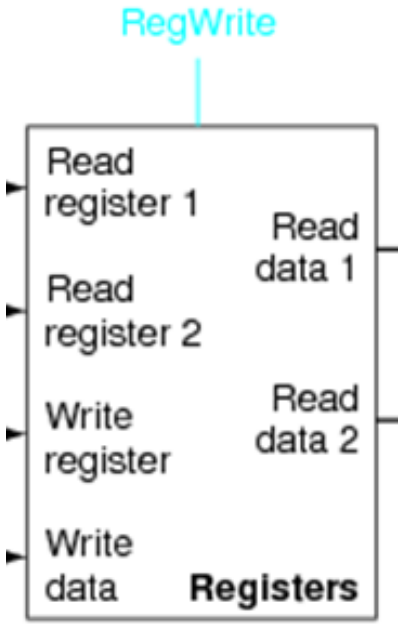


3 Instruction Formats: all 32 bits wide



Register file, module m_regfile

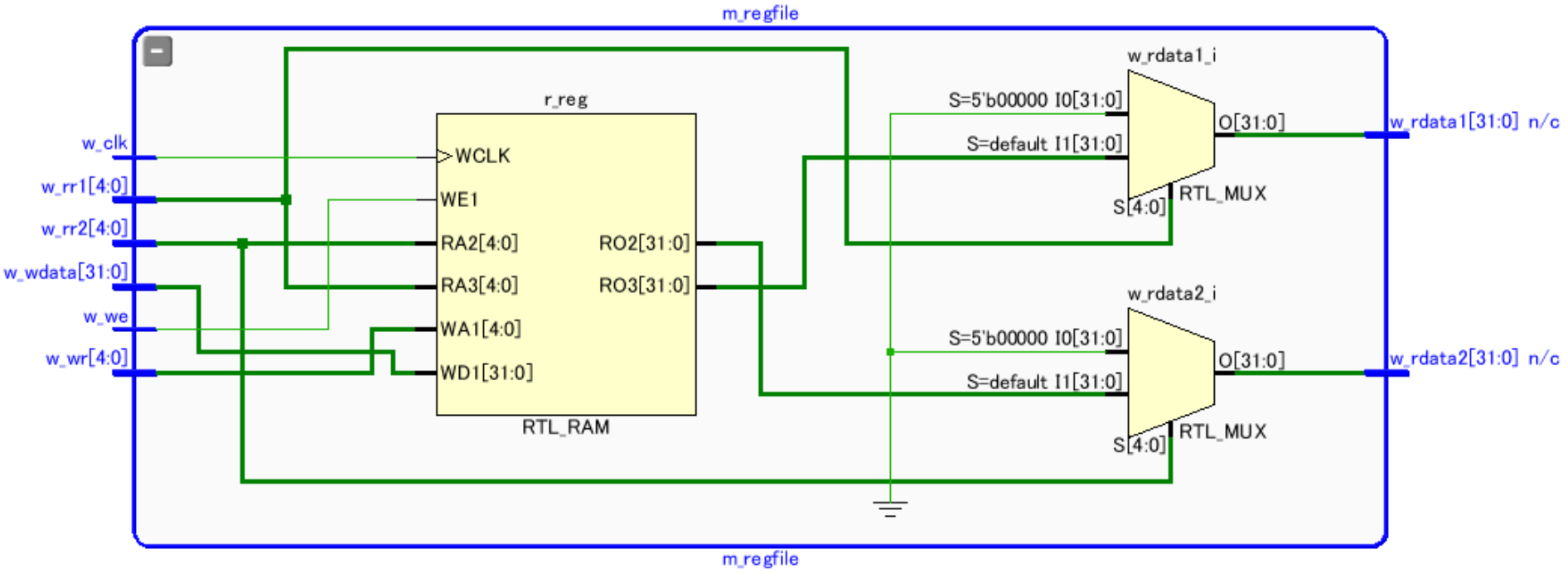
- w_rw が1の時に，w_rw で指定されたレジスタに w_wdata の値を書き込む.
- w_rr1 で指定したレジスタの値を読み出し w_rdata1 に出力する.
- w_rr2 で指定したレジスタの値を読み出し w_rdata2 に出力する.
  - ただし，レジスタ番号0の読み出しは，値0を出力する.



code093.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 begin
    r[1] = 1;
    r[2] = 2;
  end
endmodule
```

Register file, module m_regfile

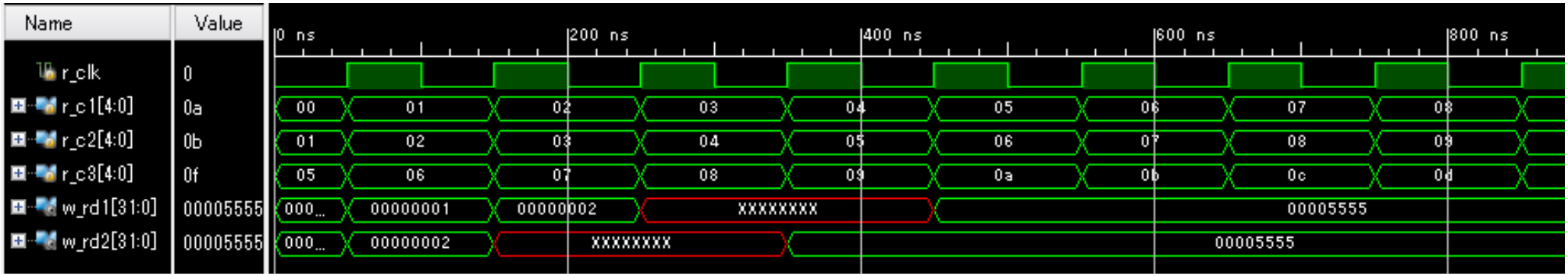


Register file, module m_regfile

- main.vを code094.v と code093.v の内容となるように入力して，シミュレーションする．
- どうしてこのような波形になるのか．

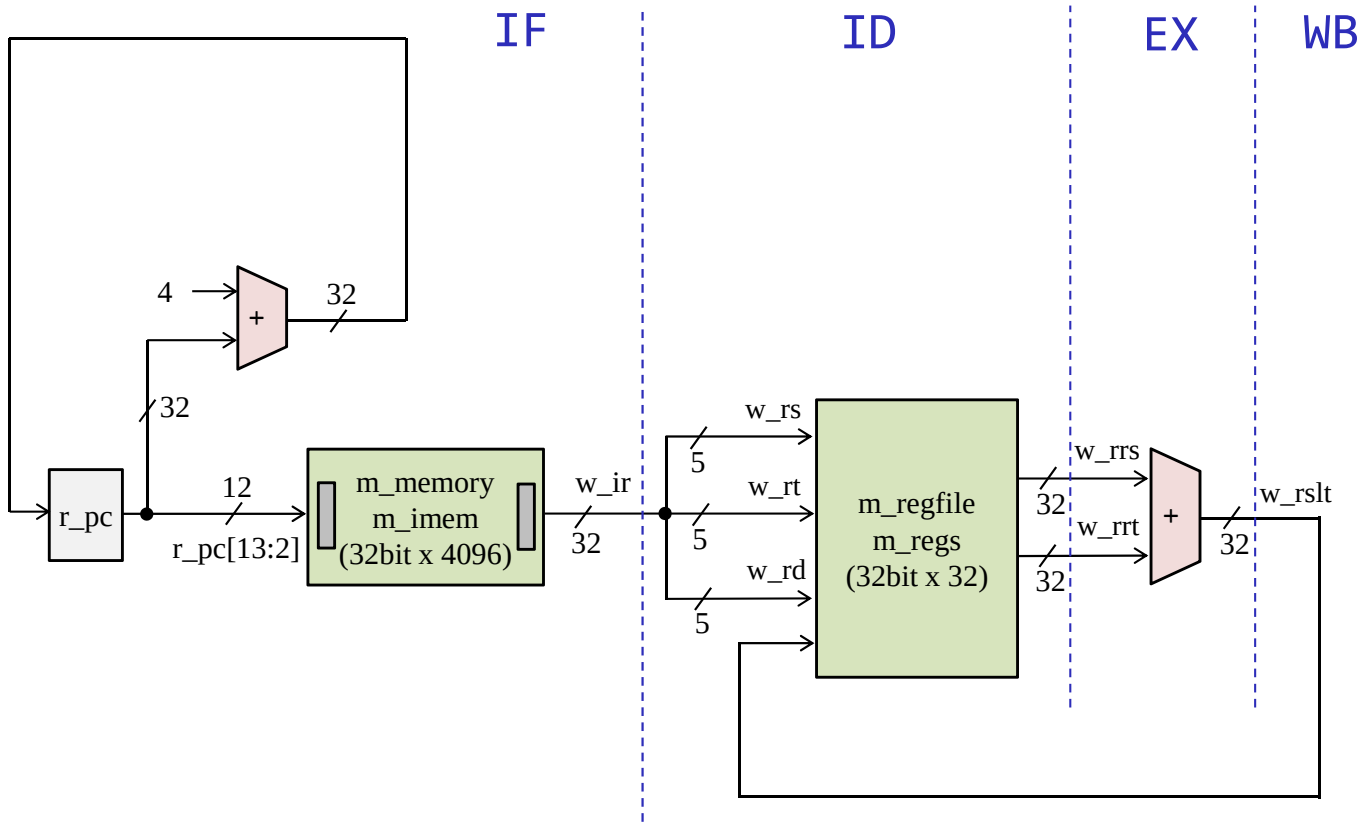
code094.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  reg [4:0] r_c1 = 0, r_c2 = 1, r_c3 = 5;
  always @(posedge r_clk) r_c1 <= r_c1 + 1;
  always @(posedge r_clk) r_c2 <= r_c2 + 1;
  always @(posedge r_clk) r_c3 <= r_c3 + 1;

  wire [31:0] w_rd1, w_rd2;
  m_regfile m_regs (r_clk, r_c1, r_c2, r_c3, 1, 32'h5555, w_rd1, w_rd2);

  always @(posedge r_clk) #1 $write("%4d %2d %2d %2d %x %x¥n",
                                    $time, r_c1, r_c2, r_c3, w_rd1, w_rd2);
endmodule
```



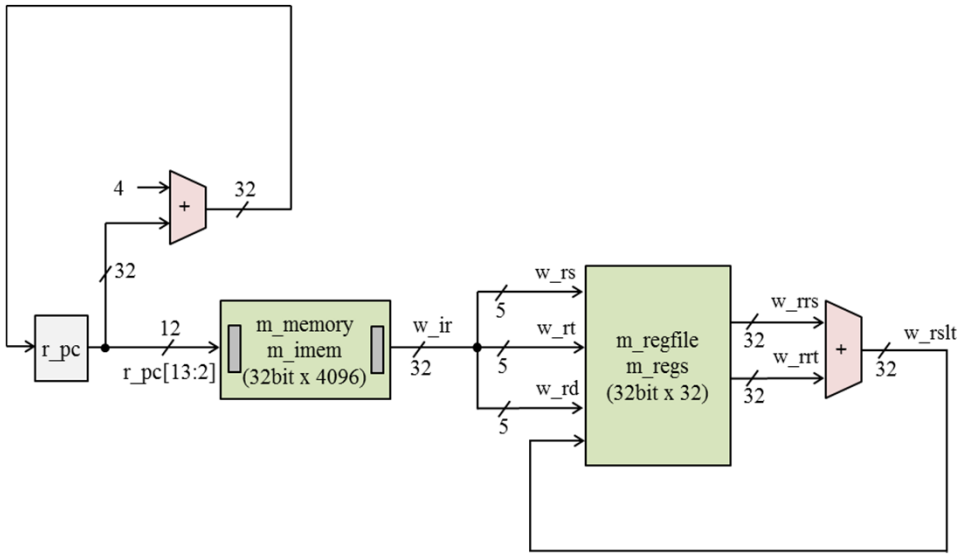
Inside module m_proc02

- 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のみに対応したプロセッサ



Inside module m_proc02

- 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のみに対応したプロセッサ



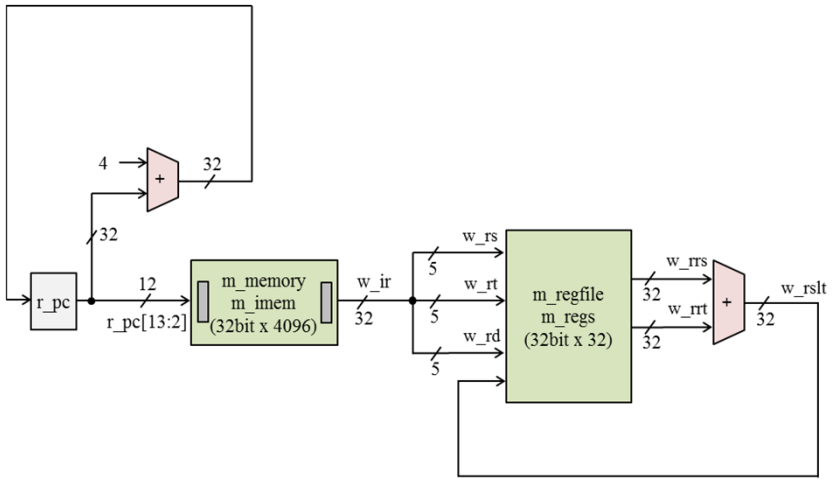
code095.v

```
module m_proc02 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  wire  [4:0] w_rs = w_ir[25:21];
  wire  [4:0] w_rt = w_ir[20:16];
  wire  [4:0] w_rd = w_ir[15:11];
  wire [31:0] w_rrs, w_rrt, w_rslt;
  m_regfile m_regs (w_clk, w_rs, w_rt, w_rd, 1, w_rslt, w_rrs, w_rrt);

  assign w_rslt = w_rrs + w_rrt;

  assign w_led = (w_btnu | w_btnd) ? w_rslt[31:16] : w_rslt[15:0];
endmodule
```

R format

Inside module m_proc02

- main.vを code096.v code093.v code091.v の内容となるように編集して，シミュレーションする．



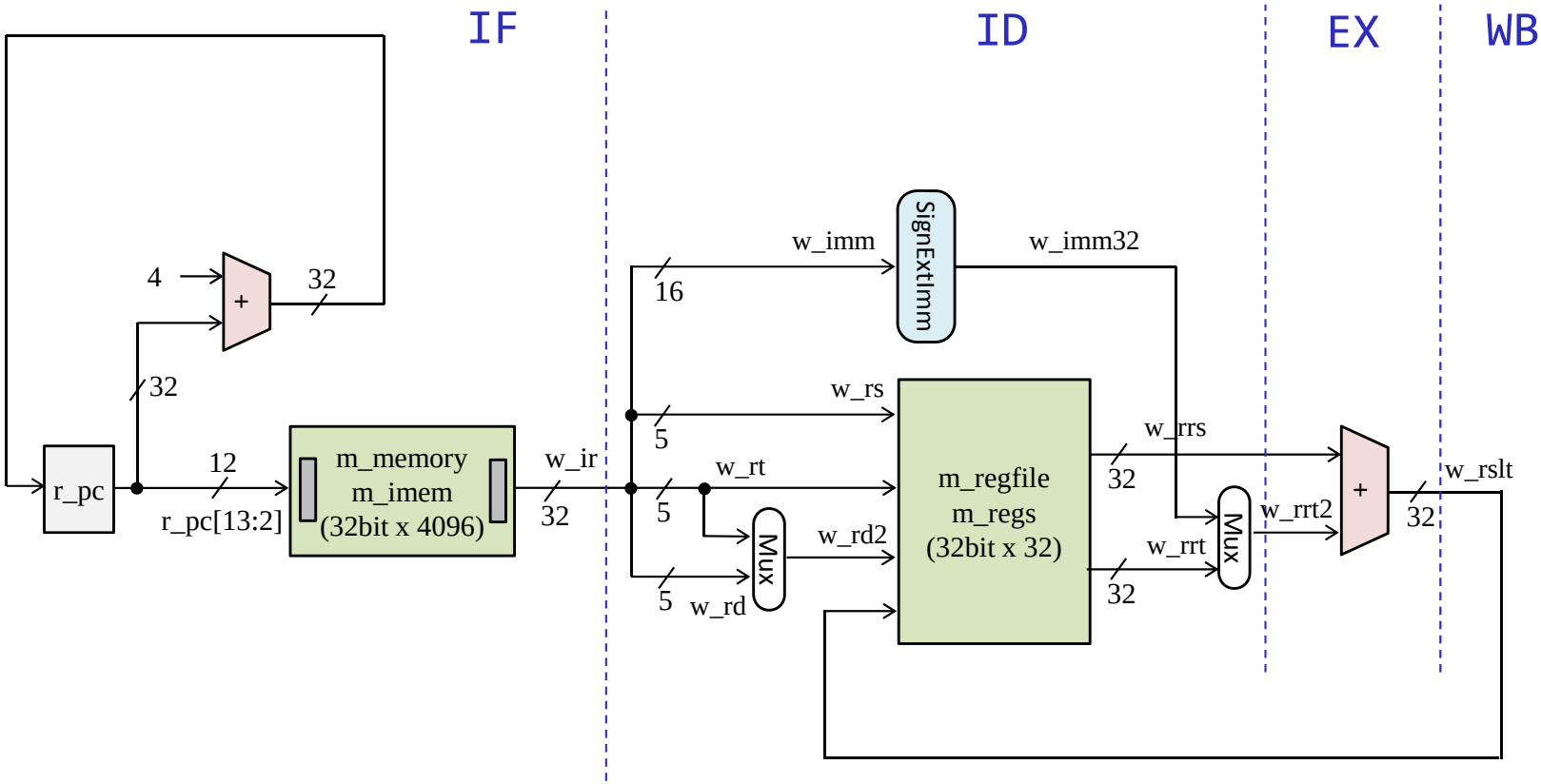

```
  1 00000000 00000000 00000000 00000000 00000000
 51 00000004 00000020 00000000 00000000 00000000
151 00000008 00000020 00000000 00000000 00000000
251 0000000c 00012020 00000000 00000001 00000001
351 00000010 00222820 00000001 00000002 00000003
451 00000014 00853020 00000001 00000003 00000004
551 00000018 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
651 0000001c xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
751 00000020 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
851 00000024 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
951 00000028 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

code096.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc02 p (r_clk, 0, 1, w_led);
  always@(*) #1 $write("%4d %x %x %x %x %x¥n", $time, p.r_pc,
                        p.w_ir, p.w_rrs, p.w_rrt, p.w_rslt);
endmodule

module m_proc02 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  wire  [4:0] w_rs = w_ir[25:21];
  wire  [4:0] w_rt = w_ir[20:16];
  wire  [4:0] w_rd = w_ir[15:11];
  wire [31:0] w_rrs, w_rrt, w_rslt;
  m_regfile m_regs (w_clk, w_rs, w_rt, w_rd, 1, w_rslt, w_rrs, w_rrt);

  assign w_rslt = w_rrs + w_rrt;

  assign w_led = (w_btnu | w_btnd) ? w_rslt[31:16] : w_rslt[15:0];
endmodule
```

Inside module m_proc03

- 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



Inside module m_proc03

- main.vを code097.v code098.v code093.v の内容となるように編集して，シミュレーションする.

code098.v

```
module m_memory (w_clk, w_addr, w_we, w_din, r_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output reg  [31:0] r_dout;
  reg r_we=0;
  reg [11:0] r_addr=0;
  reg [31:0] r_din=0;
  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) begin
    r_addr <= w_addr;
    r_din  <= w_din;
    r_we   <= w_we;
    r_dout <= cm_ram[r_addr];
    if (r_we) cm_ram[r_addr] <= r_din;
  end
  initial begin
    r_dout = 0;
    cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add  $0, $0, $0
    cm_ram[1] = {6'h9, 5'd0, 5'd4, 16'd7};             // addi $4, $0, 7
    cm_ram[2] = {6'h0, 5'd1, 5'd2, 5'd5, 5'h0, 6'h20}; // add  $5, $1, $2
    cm_ram[3] = {6'h0, 5'd4, 5'd5, 5'd6, 5'h0, 6'h20}; // add  $6, $4, $5
  end
endmodule
```

code097.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc03 p (r_clk, 0, 1, w_led);
  always@(*) #1 $write("%4d %s %2d %2d %x %x %x %x¥n", $time, p.r_pc,
                       p.w_rs, p.w_rd2, p.w_ir, p.w_rrs, p.w_rrt2, p.w_rslt);
endmodule

module m_proc03 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  wire  [5:0] w_op = w_ir[31:26];
  wire  [4:0] w_rs = w_ir[25:21];
  wire  [4:0] w_rt = w_ir[20:16];
  wire  [4:0] w_rd = w_ir[15:11];
  wire  [4:0] w_rd2 = (w_op==6'h9) ? w_rt : w_rd;
  wire [15:0] w_imm = w_ir[15:0];
  wire [31:0] w_rrs, w_rrt, w_imm32, w_rrt2, w_rslt;
  m_regfile m_regs (w_clk, w_rs, w_rt, w_rd2, 1, w_rslt, w_rrs, w_rrt);

  assign w_imm32 = {{16{w_imm[15]}}, w_imm};
  assign w_rrt2 = (w_op==6'h9) ? w_imm32 : w_rrt; // addi ?

  assign w_rslt = w_rrs + w_rrt2;

  assign w_led = (w_btnu | w_btnd) ? w_rslt[31:16] : w_rslt[15:0];
endmodule
```

Inside module m_proc03

- code098.v の命令メモリに格納する命令を変更して，正しく動作するか確認する．
- addi命令の即値を負の数に設定したらどうなるか．

code098.v

```
module m_memory (w_clk, w_addr, w_we, w_din, r_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output reg  [31:0] r_dout;
  reg r_we=0;
  reg [11:0] r_addr=0;
  reg [31:0] r_din=0;
  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) begin
    r_addr <= w_addr;
    r_din  <= w_din;
    r_we   <= w_we;
    r_dout <= cm_ram[r_addr];
    if (r_we) cm_ram[r_addr] <= r_din;
  end
  initial begin
    r_dout = 0;
    cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add  $0, $0, $0
    cm_ram[1] = {6'h9, 5'd0, 5'd4, 16'd7};             // addi $4, $0, 7
    cm_ram[2] = {6'h0, 5'd1, 5'd2, 5'd5, 5'h0, 6'h20}; // add  $5, $1, $2
    cm_ram[3] = {6'h0, 5'd4, 5'd5, 5'd6, 5'h0, 6'h20}; // add  $6, $4, $5
  end
endmodule
```

code097.v

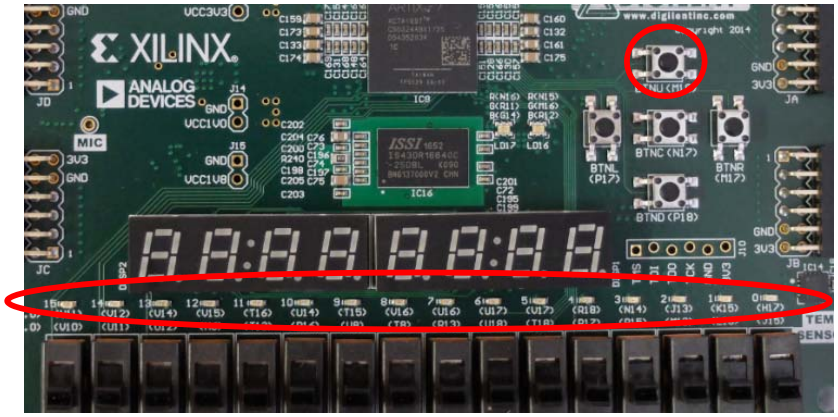
```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc03 p (r_clk, 0, 1, w_led);
  always@(*) #1 $write("%4d %s %2d %2d %x %x %x %x¥n", $time, p.r_pc,
                  p.w_rs, p.w_rd2, p.w_ir, p.w_rrs, p.w_rrt2, p.w_rslt);
endmodule

module m_proc03 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  reg [31:0] r_pc = 0;
  always @(posedge w_clk) r_pc <= r_pc + 4;
  wire [31:0] w_ir;
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  wire  [5:0] w_op = w_ir[31:26];
  wire  [4:0] w_rs = w_ir[25:21];
  wire  [4:0] w_rt = w_ir[20:16];
  wire  [4:0] w_rd = w_ir[15:11];
  wire  [4:0] w_rd2 = (w_op==6'h9) ? w_rt : w_rd;
  wire [15:0] w_imm = w_ir[15:0];
  wire [31:0] w_rrs, w_rrt, w_imm32, w_rrt2, w_rslt;
  m_regfile m_regs (w_clk, w_rs, w_rt, w_rd2, 1, w_rslt, w_rrs, w_rrt);

  assign w_imm32 = {{16{w_imm[15]}}, w_imm};
  assign w_rrt2 = (w_op==6'h9) ? w_imm32 : w_rrt; // addi ?

  assign w_rslt = w_rrs + w_rrt2;

  assign w_led = (w_btnu | w_btnd) ? w_rslt[31:16] : w_rslt[15:0];
endmodule
```

Inside code043.v

- 32ビットカウンタを用いて16個のLEDを点滅させる回路の例.
- main.vをcode043.vとなるように入力する.
- FPGAのコンフィギュレーションし，演習(2)と同様にLEDが点滅することを確認する.

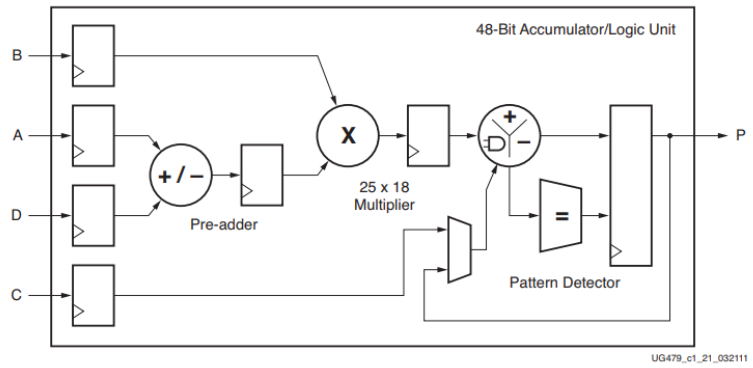
code043.v

```
module m_main (w_clk, w_rst, w_led);
  input  wire w_clk;
  input  wire w_rst;
  output wire [15:0] w_led;

  reg [31:0] r_cnt;
  always@(posedge w_clk) begin
    if (w_rst) r_cnt <= 0;
    else r_cnt <= r_cnt + 1;
  end
  assign w_led = r_cnt[31:16];
endmodule
```

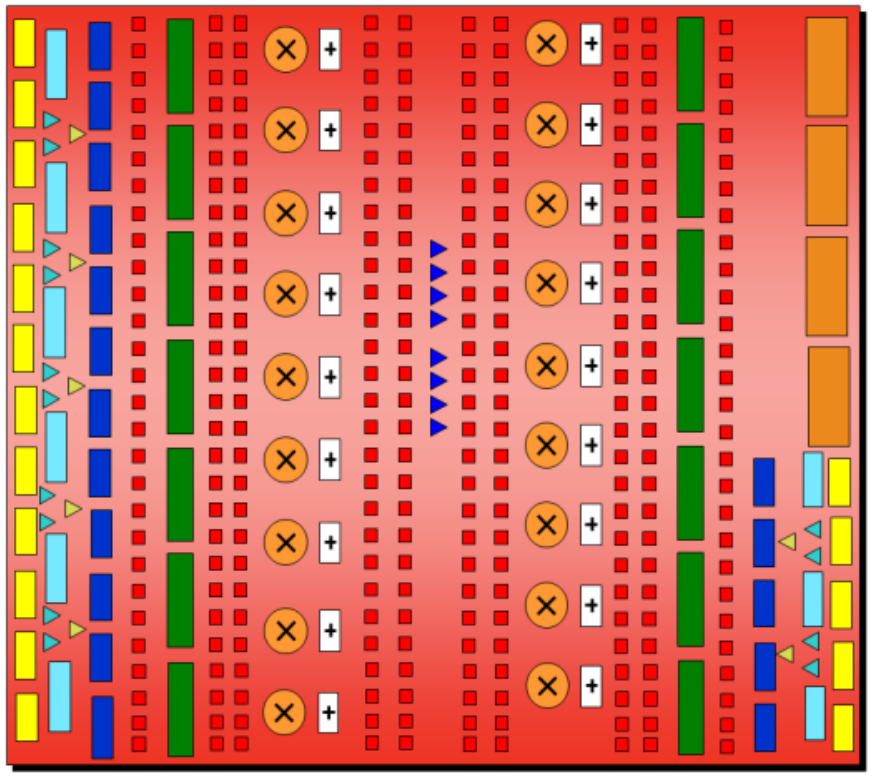


Artix-7 Architecture Overview

- CLB (Configurable Logic Block)
- BRAM (Block RAM, embedded memory)
- DSP (Digital Signal Processing)
- CMT (Clock Management Tile)
- Routing fabric

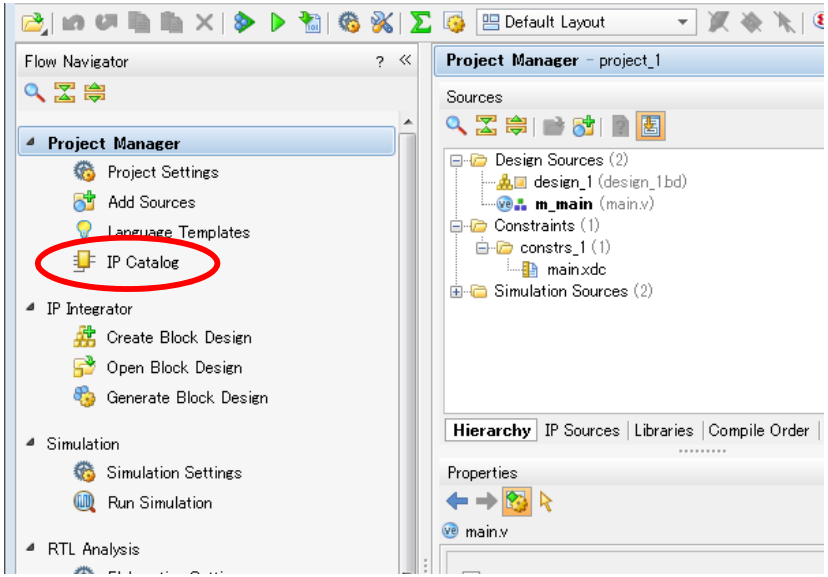


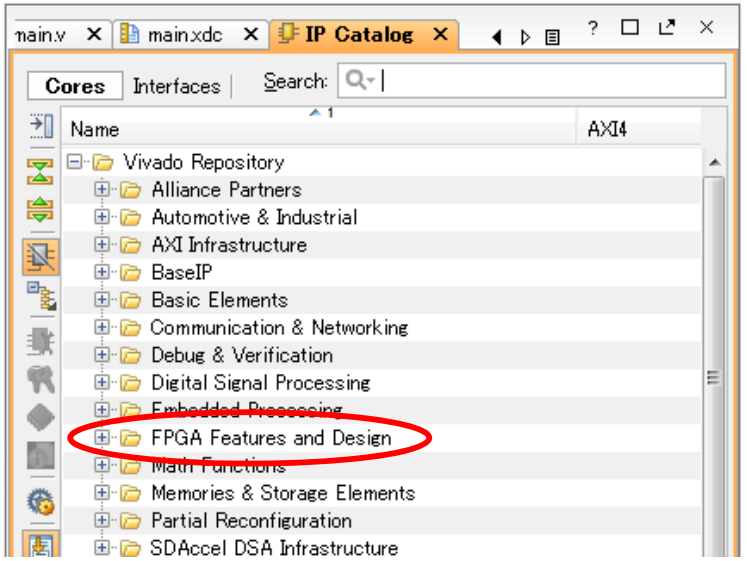
DSP Slice

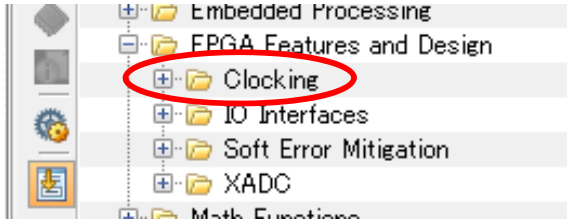


CMT (Clock Management Tile)

- Click **IP Catalog**
  - IP (Intellectual Property)
- Click **FPGA Features and Design**
- Click **Clocking**

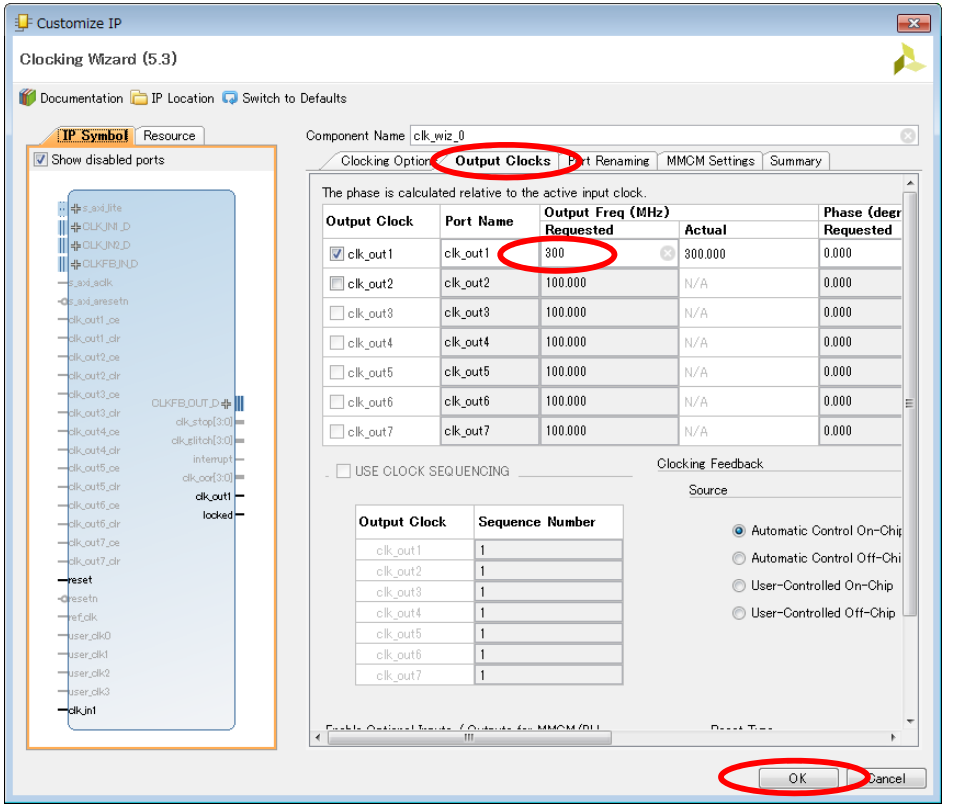






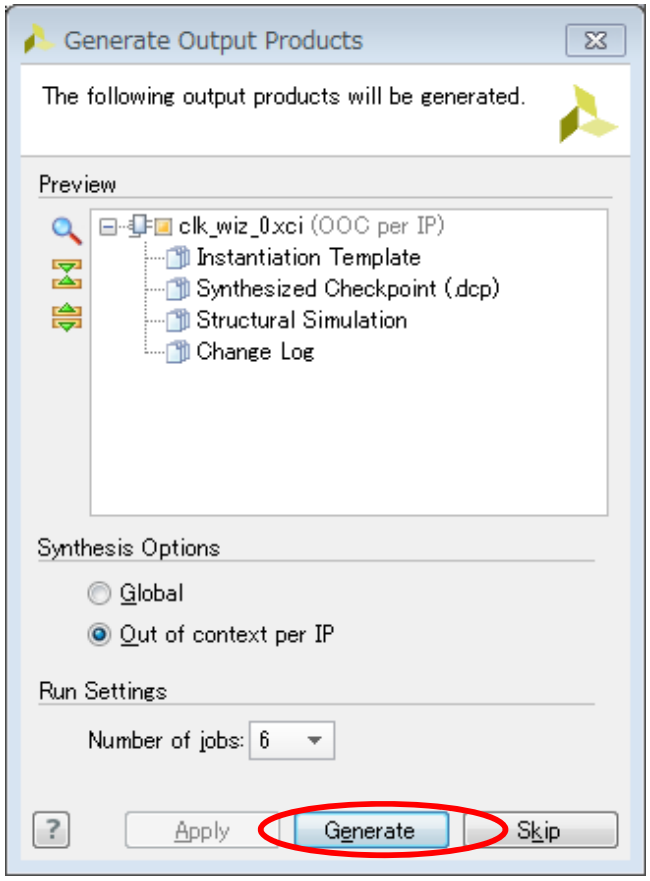
CMT (Clock Management Tile)

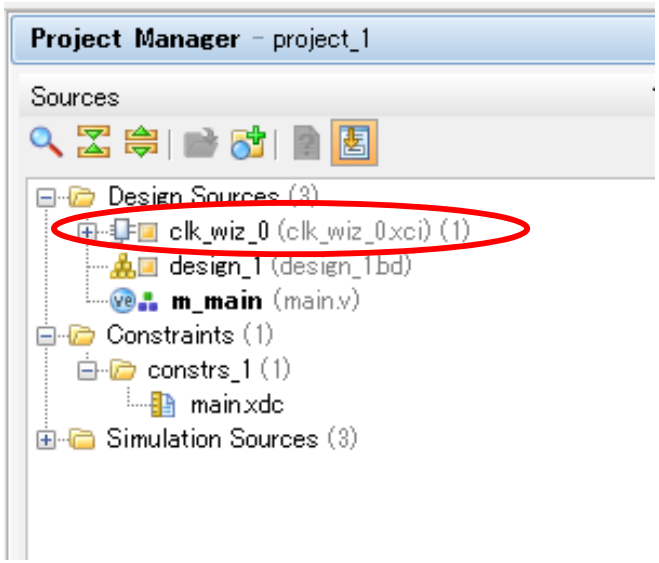
- Double click **Clocking Wizard**
- Select **Output Clocks**
  - Set **300** in Output Freq (MHz) Requested
  - In this project, we generate 300MHz clock signal
- Select **OK**
- Select **OK** in Create Directory window



CMT (Clock Management Tile)

- Click **Generate** in Generate Output Products window
- You will see **clk_wiz_0** in Sources



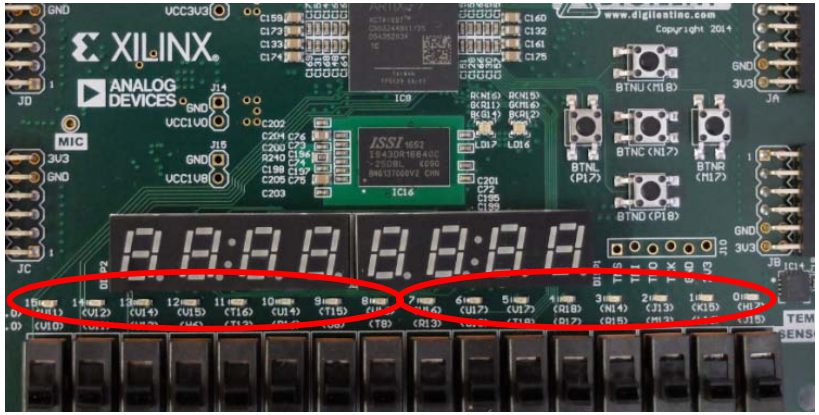


Inside code101.v

- main.vをcode101.vとなるように入力する.
  - w_clk は100MHzのクロック信号，w_clk2 は300MHzのクロック信号
- 論理合成，ビットストリームを作成する.
- FPGAのコンフィギュレーションができたら，担当のTAを呼んで確認してもらう（Check Point 5）.

code101.v

```
module m_main (w_clk, w_rst, w_led);
  input  wire w_clk;
  input  wire w_rst;
  output wire [15:0] w_led;

  wire w_clk2, w_locked;
  clk_wiz_0 clk_wiz (w_clk2, w_rst, w_locked, w_clk);

  reg [31:0] r_cnt=0;
  always@(posedge w_clk) r_cnt <= r_cnt + 1;

  reg [31:0] r_cnt2=0;
  always@(posedge w_clk2) r_cnt2 <= r_cnt2 + 1;

  assign w_led = {r_cnt2[29:22],  r_cnt[29:22]};
endmodule
```



Inside code101.v

- Double click **clk_wiz**
- w_clk2 のクロック信号を 400MHz, 500MHz, 600MHz に変更して論理合成する．タイミング制約を満たすだろうか？

code101.v

```
module m_main (w_clk, w_rst, w_led);
  input  wire w_clk;
  input  wire w_rst;
  output wire [15:0] w_led;

  wire w_clk2, w_locked;
  clk_wiz_0 clk_wiz (w_clk2, w_rst, w_locked, w_clk);

  reg [31:0] r_cnt=0;
  always@(posedge w_clk) r_cnt <= r_cnt + 1;

  reg [31:0] r_cnt2=0;
  always@(posedge w_clk2) r_cnt2 <= r_cnt2 + 1;

  assign w_led = {r_cnt2[29:22],  r_cnt[29:22]};
endmodule
```