機能設計仕様書

2018 年 4 月 26 日

1029-28-9483 勝田 峻太朗

p2

コンポーネントの外部仕様

概要

このコンポーネントはレジスタを含む. このコンポーネントの基本的な動作としては, 命令を解釈すると同時に, 後続の p3 モジュールが必要とする値をレジスタから読み出す.

入力

このコンポーネントは, 入力として, 以下の 2 つを受け取る.

clock(2bit) クロック
command(16bit) p1 によって読み出された命令を入力とする.
readflag 通常の命令を解釈する時は 1, メモリから読んだ値をレジスタに書き込む時は 0.
writetarget(3bit) メモリから読んだ値を格納するレジスタの番号.(load 命令のときのみ)
writeval(16bit) レジスタに書き込む値.(load 命令のときのみ)

出力

alu1, alu2(16bit) 演算命令の場合,ALU が使用する値 2 つをレジスタや即値から取得する.
opcode(4bit) ALU で処理をする場合に, 行うべき演算を示している.

code	計算
0000	in1 + in2
0001	in1 - in2
1000	in1 & in2 (bitwise)
1001	in1 | in2 (bitwise)
1010	in1 « i2
1011	in1 » in2

writereg(1bit) 演算または, ロード命令の場合, 結果をレジスタに書き込む必要がある. 書き込む場合は 1, 書き込まない場合は 0 である.
regaddress(3bit) writereg が 1 の場合, 書き込む対象となるレジスタの番号を示す.
memwrite(2bit) メモリに行う操作をコードで表す. 何もしない場合は 00, 読み込み時は 01, 書き込み時は 10 を示す.
address(16bit) メモリに操作をを行う場合, どの番地に行うかを示している.
storedata(16bit) メモリに書き込みを行う場合, 書き込む内容を示す.

内部仕様

入力として命令とクロックをを受け取り, 出力レジスタに対して, クロックの立ち上がりとともに, 対応するデータを書き込む. 書き込む値は, 全て場合分け関数をもちいて出力される.

以下では, 各出力を決定するために用いている関数を示す.

レジスタ

レジスタをモジュール内に定義し, 値を読み取るための関数 read を定義した.

また, 各レジスタの初期値は, テスト時の便利のため, レジスタ番号と同じ値に初期かしている.

```
////////////////
/// registers ///
////////////////

reg [15:0] r0, r1, r2, r3, r4, r5, r6, r7;

// initial assignments for testing
initial begin
    r0 = 16'b0;
    r1 = 16'b1;
    r2 = 16'b10;
    r3 = 16'b11;
    r4 = 16'b100;
    r5 = 16'b101;
    r6 = 16'b110;
    r7 = 16'b111;
end

// read register value
function [15:0] read;
input [2:0] addressin;
    case (addressin)
    0: read = r0;
    1: read = r1;
    2: read = r2;
    3: read = r3;
    4: read = r4;
    5: read = r5;
    6: read = r6;
    7: read = r7;
    default: read = 16'b0;
    endcase
endfunction
```

alu1,alu2,opcode

```
////////////////
/// functions ///
```

```
////////////////

//..

function [2:0] getaluaddress1;
input [15:0] command;
case (command[15:14])
    3: getaluaddress1 = command[13:11];
    0: getaluaddress1 = command[13:11];
    1: getaluaddress1 = command[13:11];
    default: getaluaddress1 = 3'b000;
endcase
endfunction

function [2:0] getaluaddress2;
input [15:0] command;
case (command[15:14])
    3: if (command[7:4] <= 4'd8) begin
            getaluaddress2 = command[10:8];
        end else begin
            getaluaddress2 = command[3:0];
        end
    0: getaluaddress2 = command[10:8];
    1: getaluaddress2 = command[10:8];
    default: getaluaddress2 = 3'b000;
endcase
endfunction

//..

always @(posedge clock) begin
    //..
    // get alu1 and 2
    alu1address = getaluaddress1(command);
    alu2address = getaluaddress2(command);
    alu1 = alu1val;
    alu2 = alu2val;
    opcode = command[7:4];
end
```

writereg, regaddress

```
// function to get writereg
function getwritereg;
input [15:0] command;
```

```
    case (command[15:14])
        3: getwritereg = 1'b1;
        0: getwritereg = 1'b1;
        1: getwritereg = 1'b0;
        2: getwritereg = 1'b1;
        default: getwritereg = 1'b0;
    endcase
endfunction

// function to get regaddress
function [2:0] getregaddress;
input [15:0] command;
    case (command[15:14])
        3: getregaddress = command[10:8];
        0: getregaddress = command[13:11];
        2: getregaddress = command[10:8];
        default: getregaddress = 2'b00;
    endcase
endfunction

always @(posedge clock) begin
    //..
    // get register things
    writereg = getwritereg(command);
    regaddress = getregaddress(command);
end
```

memwrite, address

```
// function to get memwrite
function [1:0] getmemwrite;
input [15:0] command;
    case (command[15:14])
        3: getmemwrite = 2'b00;
        0: getmemwrite = 2'b01;
        1: getmemwrite = 2'b10;
        2: getmemwrite = 2'b01;
        default: getmemwrite = 2'b00;
    endcase
endfunction

// function to get memory address
function [15:0] getaddress;
input [15:0] alu2;
input [15:0] command;
    case (command[15:14])
```

```
    0: getaddress = alu2 + signext8(command[7:0]);
    1: getaddress = alu2 + signext8(command[7:0]);
    2: getaddress = signext8(command[7:0]);
    endcase
endfunction

//..

always @(posedge clock) begin
    //..
    // get memory things
    memwrite = getmemwrite(command);
    address = getaddress(alu2val, command);
end
```

Register.v

内部仕様

概要

レジスタを実装しており, 値の読み (同時に最大 2 つ), 書き (同時に最大 1 つ) を行う.

外部仕様

入力

clock クロック
rs(2bit) レジスタの値を読みときは, 読み対象のレジスタの番号を表す. 値は `read1` から出力される. 書き込む場合は, 書き込む対象のレジスタの番号.
rd(3bit) レジスタを読み場合, 読む対象のレジスタの番号を示す. 値は `read2` から出力される.
readflag 行う動作を指定する. 読み出し時は 1, 書き込み時は 0.
value(16bit) 書き込みを行う場合, 書き込む値

出力

read1 値を読む時,`rs` で指定されたレジスタの値をここから返す.
read2 値を読む時,`rd` で指定されたレジスタの値をここから返す.

内部仕様

概要

内部では, レジスタを以下のように保持している.

```
// the registers
reg [15:0] r0, r1, r2, r3, r4, r5, r6, r7;
```

値を読む

値を読むため, レジスタの値を返す関数を以下のように実装している.

```
// read register value
function [15:0] read;
input [2:0] addressin;
    case (addressin)
    0: read = r0;
    1: read = r1;
    2: read = r2;
    3: read = r3;
    4: read = r4;
    5: read = r5;
    6: read = r6;
    7: read = r7;
    default: read = 16'b0;
    endcase
endfunction
```

また, 入力クロックの立ち上がり時に, 上で示した関数を用いて, 以下のように読み書きを行う.

```
// main
always @(posedge clock) begin
    if (readflag == 1'b1) begin
        read1 <= read(rs);
        read2 <= read(rd);
    end else begin
        case (rs)
        0: r0 <= value;
        1: r1 <= value;
        2: r2 <= value;
        3: r3 <= value;
        4: r4 <= value;
        5: r5 <= value;
        6: r6 <= value;
        7: r7 <= value;
        endcase
    end
end
```

値を書き込む

readflag の値が 0 である時には, レジスタに値を書き込むということであるから, 下記のように値を書き込む.

```
always @(posedge clock) begin
    if (readflag == 1'b1) begin
        /..
    end else begin // write to register
        case (writetarget)
        0: r0 <= writeval;
        1: r1 <= writeval;
        2: r2 <= writeval;
        3: r3 <= writeval;
        4: r4 <= writeval;
        5: r5 <= writeval;
        6: r6 <= writeval;
        7: r7 <= writeval;
        endcase
    end
end
```