Computation II: embedded system design (5EIB0)

Available from:
Due date:
Requested files: -
Type of work:

Introduction

The code below performs selctive biasing. Selective biasing checks an image pixel-by-pixel. All the pixels above a threshold are biased down (subtracted) by 100 whereas all the pixels below the threshold are biased up (added) by 100. The C code below handles this with an if-statement resulting in many processor instructions.

```
#define WIDTH   32
#define HEIGHT  32

void main(void)
{
    int a, b, result;
    unsigned char *buf_i = (unsigned char*)0x401000, *buf_o = (unsigned char*)0x402000;

    for (a = 1; a < HEIGHT - 1; a++)
    {
        for (b = 1; b < WIDTH - 1; b++)
        {
            /*biasing*/
            result = buf_i[a * WIDTH + b];
            if(result > 128) buf_o[a * WIDTH + b] = result-100;
            else buf_o[a * WIDTH + b] = result+100;

        }
    }
}
```

To reduce the number of instructions, the C code is modified to use a custom instruction (biasing), giving the following C code:

```
#define WIDTH   32
#define HEIGHT  32
#define biasing(p, q) ((p) + ((q) - *(int *) 0x12344321)) //HEX Code: 0x32

void main(void)
{
    int a, b, result, threshold;
    unsigned char *buf_i = (unsigned char*)0x401000, *buf_o = (unsigned char*)0x402000;

    for (a = 1; a < HEIGHT - 1; a++)
    {
        for (b = 1; b < WIDTH - 1; b++)
        {
            /*biasing*/
            result = buf_i[a * WIDTH + b];
                        buf_o[a * WIDTH + b] = biaising(result, threshold);

        }
    }
}
```

Assignment

Add hardware support for the custom instruction (**biasing**) in the current mMIPS implementation by modifying all or some of the provided files (ctrl.v, aluctrl.v and alu.v). You can infer the functionality of **biasing** by comparing the C codes above.

Note: The biasing function code is 0x32.

Debug

For debugging the design, you can use the "**$display**" command in the RTL code to print out the values of interest. **Note: "$display" will only work in the debug mode of VPL.**

Requested files

ctrl.v

```verilog
/////////////////////////////////////////////////
// CTRL.V
//
// TU/e Eindhoven University Of Technology
// Eindhoven, The Netherlands
//
// Created: 21-11-2013
// Author: Bergmans, G (g.bergmans@student.tue.nl)
// Based on work by Sander Stuijk
//
// Function:
//     Single cycle controller
//
// Version:
//     (27-01-2014): initial version
//
///////////////////////////////////////////////!/

module CTRL(
        enable,
        en,
        Opcode,
        FunctionCode,
        RegDst,
        Target,
        Branch,
        MemRead,
        MemtoReg,
        ALUop,
        MemWrite,
        ALUSrc,
        RegWrite,
        SignExtend,
        c4,
        c1,
        c31,
        HiLoWrite,
        AluSel
    );

    input   enable;
    output  [0:0]   en;
    reg     [0:0]   en;
    input   [5:0]   Opcode;
    input   [5:0]   FunctionCode;
    output  [1:0]   RegDst;
    reg     [1:0]   RegDst;
    output  [1:0]   Target;
    reg     [1:0]   Target;
    output  [1:0]   Branch;
    reg     [1:0]   Branch;
    output  [1:0]   MemRead;
    reg     [1:0]   MemRead;
    output  [1:0]   MemtoReg;
    reg     [1:0]   MemtoReg;
    output  [4:0]   ALUop;
    reg     [4:0]   ALUop;
    output  [1:0]   MemWrite;
    reg     [1:0]   MemWrite;
    output  [0:0]   ALUSrc;
    reg     [0:0]   ALUSrc;
    output  [0:0]   RegWrite;
    reg     [0:0]   RegWrite;
    output  [0:0]   SignExtend;
    reg     [0:0]   SignExtend;
    output  [31:0]  c4;
    reg     [31:0]  c4;
    output  [0:0]   c1;
    reg     [0:0]   c1;
    output  [4:0]   c31;
    reg     [4:0]   c31;
    output  [0:0]   HiLoWrite;
    reg     [0:0]   HiLoWrite;
    output  [1:0]   AluSel;
    reg     [1:0]   AluSel;


    always @(Opcode or FunctionCode or enable)
        begin

            //Write constant 4 to output
            c4 = 32'b00000000000000000000000000000100;
            //Write constant 1 to output
            c1 = 1'b1;
            //Write constant 31 to output
            c31 = 5'b11111;

            if (enable == 1)
                en = 1'b1;
            else
                en = 1'b0;

            RegDst      = 2'b00;
            Target      = 2'b00;
            ALUSrc      = 1'b0;
            MemtoReg    = 2'b00;
            RegWrite    = 1'b0;
            MemRead     = 2'b00;
            MemWrite    = 2'b00;
            Branch      = 2'b00;
            ALUop       = 5'b00000;
            SignExtend  = 1'b0;

```

```verilog
//Determine the output
case (Opcode)
0: // R-format instruction: check functioncode
    case (FunctionCode)
        'h8:  // Instruction: Jr
            begin
                RegDst      = 2'b01;
                Target      = 2'b10;
                ALUSrc      = 1'b0;
                MemtoReg    = 2'b00;
                RegWrite    = 1'b0;
                MemRead     = 2'b00;
                MemWrite    = 2'b00;
                Branch      = 2'b11;
                ALUop       = 5'b00010;
                SignExtend  = 1'b1;
                HiLoWrite   = 1'b0;
                AluSel      = 2'b00;
            end
        'h9:  // Instruction Jalr
            begin
                RegDst      = 2'b01;
                Target      = 2'b10;
                ALUSrc      = 1'b0;
                MemtoReg    = 2'b00;
                RegWrite    = 1'b1;
                MemRead     = 2'b00;
                MemWrite    = 2'b00;
                Branch      = 2'b11;
                ALUop       = 5'b00010;
                SignExtend  = 1'b1;
                HiLoWrite   = 1'b0;
                AluSel      = 2'b11;
            end
        'h10:  // Instruction: Move hi register
            begin
                RegDst      = 2'b01;
                Target      = 2'b00;
                ALUSrc      = 1'b0;
                MemtoReg    = 2'b00;
                RegWrite    = 1'b1;
                MemRead     = 2'b00;
                MemWrite    = 2'b00;
                Branch      = 2'b00;
                ALUop       = 5'b00010;
                SignExtend  = 1'b1;
                HiLoWrite   = 1'b0;
                AluSel      = 2'b10;
            end
        'h12:  // Instruction: Move lo register
            begin
                RegDst      = 2'b01;
                Target      = 2'b00;
                ALUSrc      = 1'b0;
                MemtoReg    = 2'b00;
                RegWrite    = 1'b1;
                MemRead     = 2'b00;
                MemWrite    = 2'b00;
                Branch      = 2'b00;
                ALUop       = 5'b00010;
                SignExtend  = 1'b1;
                HiLoWrite   = 1'b0;
                AluSel      = 2'b01;
            end
        'h19:  // Instruction: Multiply unsigned
            begin
                RegDst      = 2'b00; //No destination
                Target      = 2'b00;
                ALUSrc      = 1'b0;
                MemtoReg    = 2'b00;
                RegWrite    = 1'b1;
                MemRead     = 2'b00;
                MemWrite    = 2'b00;
                Branch      = 2'b00;
                ALUop       = 5'b00010;
                SignExtend  = 1'b1;
                HiLoWrite   = 1'b1;
                AluSel      = 2'b00;
            end
        default: // Others
            begin
                RegDst      = 2'b01;
                Target      = 2'b00;
                ALUSrc      = 1'b0;
                MemtoReg    = 2'b00;
                RegWrite    = 1'b1;
                MemRead     = 2'b00;
                MemWrite    = 2'b00;
                Branch      = 2'b00;
                ALUop       = 5'b00010;
                SignExtend  = 1'b1;
                HiLoWrite   = 1'b0;
                AluSel      = 2'b00;
                end
        endcase
2:  // Instruction: J
    begin
        RegDst      = 2'b00;
        Target      = 2'b01;
        ALUSrc      = 1'b0;
        MemtoReg    = 2'b00;
        RegWrite    = 1'b0;
        MemRead     = 2'b00;
```

```
                    MemWrite    = 2'b00;
                    Branch      = 2'b11;
                    ALUop       = 5'b00010;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            3:  // Instruction; Jal
                begin
                    RegDst      = 2'b10;
                    Target      = 2'b01;
                    ALUSrc      = 1'b0;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b11;
                    ALUop       = 5'b00010;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b11;
                end
            4:  // Instruction: BEQ
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b0;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b0;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b01;
                    ALUop       = 5'b00001;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            5:  // Instruction: BNE
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b0;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b0;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b10;
                    ALUop       = 5'b00001;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            9:  // Instruction: ADDIU
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b00011;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            10:  // Instruction: SLTI
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b00111;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            11:  // Instruction: SLTUI
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b01000;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            12:  // Instruction: ANDI
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b00100;
                    SignExtend  = 1'b0;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            13:  // Instructino: ORI
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b00101;
                    SignExtend  = 1'b0;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            14:  // Instruction: XORI
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b00110;
                    SignExtend  = 1'b0;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            15:  // Instruction: LUI
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b01001;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            32:  //Instruction: LB
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b10;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b10;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b00000;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            35:  // Instruction: LW
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b01;
                    RegWrite    = 1'b1;
                    MemRead     = 2'b01;
                    MemWrite    = 2'b00;
                    Branch      = 2'b00;
                    ALUop       = 5'b00000;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            40:  // Instruction: SB
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b0;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b10;
                    Branch      = 2'b00;
                    ALUop       = 5'b00000;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            43:  // Instruction: SW
                begin
                    RegDst      = 2'b00;
                    Target      = 2'b00;
                    ALUSrc      = 1'b1;
```

```verilog
                    MemtoReg    = 2'b00;
                    RegWrite    = 1'b0;
                    MemRead     = 2'b00;
                    MemWrite    = 2'b01;
                    Branch      = 2'b00;
                    ALUop       = 5'b00000;
                    SignExtend  = 1'b1;
                    HiLoWrite   = 1'b0;
                    AluSel      = 2'b00;
                end
            default: //No default case
                begin
                end
            endcase
        end

endmodule

```

aluctrl.v

```verilog
//////////////////////////////////////////////////
// ALUCTRL.V
//
// TU/e Eindhoven University Of Technology
// Eindhoven, The Netherlands
//
// Created: 21-11-2013
// Author: Bergmans, G (g.bergmans@student.tue.nl)
// Based on work by Sander Stuijk
//
// Function:
//     ALU controller
//
// Version:
//     (27-01-2014): initial version
//
////////////////////////////////////////////////!/

module ALUCTRL(functionCode, ALUop, Shamt, ALUctrl);
    input   [5:0]   functionCode;
    input   [4:0]   ALUop;
    input   [4:0]   Shamt;
    output  [5:0]   ALUctrl;
    reg     [5:0]   ALUctrl;

    always @(functionCode or ALUop or Shamt)
        begin : aluctrl_thread
            case (ALUop) //synopsys parallel_case
                'h0:    // Add signed
                    ALUctrl = 'h2;

                'h1:    // Subtract unsigned
                    ALUctrl = 'h6;

                'h2:    // R-type instruction, look to functionCode
                    begin
                        case (functionCode)
                            'h0:    // SLL
                                case (Shamt) //Check shift amount
                                    1:
                                        ALUctrl = 'hA;
                                    2:
                                        ALUctrl = 'hB;
                                    8:
                                        ALUctrl = 'hC;
                                    default:
                                        ALUctrl = 'h0;
                                endcase

                            'h2:    // SRL
                                case (Shamt) //Check shift amount
                                    1:
                                        ALUctrl = 'hD;
                                    2:
                                        ALUctrl = 'hE;
                                    8:
                                        ALUctrl = 'hF;
                                    default:
                                        ALUctrl = 'h0;
                                endcase

                            'h3:    // SRA
                                case (Shamt) //Check shift amount
                                    1:
                                        ALUctrl = 'h10;
                                    2:
                                        ALUctrl = 'h11;
                                    8:
                                        ALUctrl = 'h12;
                                    default:
                                        ALUctrl = 'h0;
                                endcase

                            'h10:   // Move hi register (nop in ALU)
                                ALUctrl = 'h0;

                            'h12:   // Move hi register (nop in ALU)
                                ALUctrl = 'h0;

                            'h19:   // Multiply unsigned
                                ALUctrl = 'h13;

                            'h20:   // Add signed
                                ALUctrl = 'h2;

                            'h21:   // Add unsigned
                                ALUctrl = 'h3;

                            'h23:   // Subtract unsigned
                                ALUctrl = 'h6;

                            'h24:   // And
                                ALUctrl = 'h0;

                            'h25:   // Or
                                ALUctrl = 'h1;

                            'h26:  // Xor
                                ALUctrl = 'h4;

                            'h2A:   //Set-on-less-than (2's complement)
                                ALUctrl = 'h7;

                            'h2B:   //Set-on-less-than (unsigned)
                                ALUctrl = 'h8;

                            default:
                                ALUctrl = 'h0;
                        endcase
                    end
                'h3:    // Add unsigned
                    ALUctrl = 6'b000011;

                'h4:    // And
                    ALUctrl = 6'b000000;

                'h5:    // Or
                    ALUctrl = 6'b000001;

                'h6:    // Xor
                    ALUctrl = 6'b000100;

                'h7:    //Slt
                    ALUctrl = 6'b000111;

                'h8:    //Sltu
                    ALUctrl = 6'b001000;

                'h9:    //Load upper immediate
                    ALUctrl = 6'b001001;

                default:
                    ALUctrl = 6'b000000;
            endcase
        end

endmodule

```

alu.v

```verilog
///////////////////////////////////////////////
// ALU.V
//
// TU/e Eindhoven University Of Technology
// Eindhoven, The Netherlands
//
// Created: 21-11-2013
// Author: Bergmans, G (g.bergmans@student.tue.nl)
// Based on work by Sander Stuijk
//
// Function:
//     Arithmetic  Logic Unit
//
// Version:
//     (27-01-2014): initial version
//
/////////////////////////////////////////////!/

module ALU(ctrl, a, b, r, r2, z);
    input       [5:0]   ctrl;
    input       [31:0]  a;
    input       [31:0]  b;
    output      [31:0]  r;
    reg         [31:0]  r;
    output      [31:0]  r2;
    reg         [31:0]  r2;
    output      [0:0]   z;
    reg         [0:0]   z;
    reg         [31:0]  s;
    reg         [31:0]  t;
    reg signed  [31:0]  s_int;
    reg signed  [31:0]  t_int;
    reg         [31:0]  result;
    reg         [31:0]  result_hi;
    reg         [0:0]   sign;
    reg signed  [63:0]  c;
    reg         [0:0]   zero;

    always @(ctrl or a or b)
        begin : alu_thread

            //Read the inputs
            s           = a;
            t           = b;
            s_int       = s;
            t_int       = t;
            result      = 0;
            result_hi   = 0;

            // Calculate result using selected operation
            case (ctrl)
                'h0:    // And
                    result = s & t;

                'h1:    // Or
                    result = s | t;

                'h2:    // Add signed
                    result = s_int + t_int;

                'h3:    // Add unsigned
                    result = s + t;

                'h4:    // Xor
                    result = s ^ t;

                'h6:    // Substract signed
                    result = s - t;

                'h7:    // Set-on-less-than
                    if (s_int < t_int)
                        result = 1;
                    else
                        result = 0;

                'h8:    // Set-on-less-than unsigned
                    if (s < t)
                        result = 1;
                    else
                        result = 0;

                'h9:    // Load upper immediate
                    result = (t << 16);

                'hA:    // SLL (1 bit)
                    result = (t << 1);

                'hB:    // SLL (2 bit)
                    result = (t << 2);

                'hC:    // SLL (8 bit)
                    result = (t << 8);

                'hD:    // SRL (1 bit)
                    result = (t >> 1);

                'hE:    // SRL (2 bit)
                    result = (t >> 2);

                'hF:    // SRL (8 bit)
                    result = (t >> 8);

                'h10:   // SRA (1 bit)
```

```
                    begin
                        sign = t[31:31];
                        result = (t >> 1);
                        result[31:31] = sign;
                    end

                'h11:   // SRA (2 bit)
                    begin
                        sign = t[31:31];
                        result = (t >> 2);
                        result[31:30] = {sign, sign};
                    end

                'h12:   // SRA (8 bit)
                    begin
                        sign = t[31:31];
                        result = (t >> 8);
                        result[31:24] = {sign, sign, sign, sign, sign, sign, sign, sign};
                    end

                'h13:   //Multu
                    begin
                        c = s * t;
                        result = c[31:0];
                        result_hi = c[63:32];
                    end

                default: //No default case: invallid opcode!
                    begin
                    end
            endcase

            // Calculate zero output
            if (result == 0)
                zero = 1;
            else
                zero = 0;

            // Write results to output
            r = result;
            r2 = result_hi;
            z = zero;
        end

endmodule

```

VPL