Computation II: embedded system design (5EIB0)

Available from: Friday, 5 April 2024, 2:00 PM
Requested files: ctrl.v, aluctrl.v, alu.v (Download)
Type of work: Individual work

Introduction

Image binarization converts an image of up to 256 gray levels to a black and white image. The simplest way to use image binarization is to choose a threshold value, and classify all pixels with values above this threshold as white (255), and all other pixels as black (0). 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;
    int threshold = 128;
    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++)
        {

            result = buf_o[a * WIDTH + b];
            /* Thresholding */
            if(result > 128) buf_o[a * WIDTH + b] = (char)255;
            else buf_o[a * WIDTH + b] = 0;


        }
    }
}
```

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

```
#define WIDTH   32
#define HEIGHT  32
#define thresholding(result, threshold) ((result) - ((threshold) + *(int *) 0x12344321)) // HEX Code: 0x31

void main(void)
{
    int a, b, result;
    int threshold = 128;
    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++)
        {

            result = buf_o[a * WIDTH + b];
            /* Thresholding */
            buf_o[a * WIDTH + b] = thresholding(result, threshold);

        }
    }
}
```

Assignment

Add hardware support for the custom instruction (**thresholding**) 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 **thresholding** by comparing the C codes above.

Note: The thresholding function code is 0x31.

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

```
1  ////////////////////////////////////////////////
2  // CTRL.V
3  //
4  // TU/e Eindhoven University Of Technology
5  // Eindhoven, The Netherlands
6  //
7  // Created: 21-11-2013
8  // Author: Bergmans, G (g.bergmans@student.tue.nl)
9  // Based on work by Sander Stuijk
10 //
11 // Function:
12 //     Single cycle controller
13 //
14 // Version:
15 //     (27-01-2014): initial version
16 //
17 ////////////////////////////////////////////////!/
18
19 module CTRL(
20          enable,
21          en,
22          Opcode,
23          FunctionCode,
24          RegDst,
25          Target,
26          Branch,
27          MemRead,
28          MemtoReg,
29          ALUop,
30          MemWrite,
31          ALUSrc,
32          RegWrite,
33          SignExtend,
34          c4,
35          c1,
36          c31,
37          HiLoWrite,
38          AluSel
39      );
40
41      input   enable;
42      output  [0:0]   en;
43      reg     [0:0]   en;
44      input   [5:0]   Opcode;
45      input   [5:0]   FunctionCode;
46      output  [1:0]   RegDst;
47      reg     [1:0]   RegDst;
48      output  [1:0]   Target;
49      reg     [1:0]   Target;
50      output  [1:0]   Branch;
51      reg     [1:0]   Branch;
52      output  [1:0]   MemRead;
53      reg     [1:0]   MemRead;
54      output  [1:0]   MemtoReg;
55      reg     [1:0]   MemtoReg;
56      output  [4:0]   ALUop;
57      reg     [4:0]   ALUop;
58      output  [1:0]   MemWrite;
59      reg     [1:0]   MemWrite;
60      output  [0:0]   ALUSrc;
61      reg     [0:0]   ALUSrc;
62      output  [0:0]   RegWrite;
63      reg     [0:0]   RegWrite;
64      output  [0:0]   SignExtend;
65      reg     [0:0]   SignExtend;
66      output  [31:0]  c4;
67      reg     [31:0]  c4;
68      output  [0:0]   c1;
69      reg     [0:0]   c1;
70      output  [4:0]   c31;
71      reg     [4:0]   c31;
72      output  [0:0]   HiLoWrite;
73      reg     [0:0]   HiLoWrite;
74      output  [1:0]   AluSel;
75      reg     [1:0]   AluSel;
76
77
78      always @(Opcode or FunctionCode or enable)
79          begin
80
81              //Write constant 4 to output
82              c4 = 32'b00000000000000000000000000000100;
83              //Write constant 1 to output
84              c1 = 1'b1;
85              //Write constant 31 to output
86              c31 = 5'b11111;
87
88              if (enable == 1)
89                  en = 1'b1;
90              else
91                  en = 1'b0;
92
93              RegDst      = 2'b00;
94              Target      = 2'b00;
95              ALUSrc      = 1'b0;
96              MemtoReg    = 2'b00;
97              RegWrite    = 1'b0;
98              MemRead     = 2'b00;
99              MemWrite    = 2'b00;
100             Branch      = 2'b00;
101             ALUop       = 5'b00000;
102             SignExtend  = 1'b0;
103
```

```
104             //Determine the output
105             case (Opcode)
106             0: // R-format instruction: check functioncode
107                 case (FunctionCode)
108                     'h8:  // Instruction: Jr
109                         begin
110                             RegDst      = 2'b01;
111                             Target      = 2'b10;
112                             ALUSrc      = 1'b0;
113                             MemtoReg    = 2'b00;
114                             RegWrite    = 1'b0;
115                             MemRead     = 2'b00;
116                             MemWrite    = 2'b00;
117                             Branch      = 2'b11;
118                             ALUop       = 5'b00010;
119                             SignExtend  = 1'b1;
120                             HiLoWrite   = 1'b0;
121                             AluSel      = 2'b00;
122                         end
123                     'h9:  // Instruction Jalr
124                         begin
125                             RegDst      = 2'b01;
126                             Target      = 2'b10;
127                             ALUSrc      = 1'b0;
128                             MemtoReg    = 2'b00;
129                             RegWrite    = 1'b1;
130                             MemRead     = 2'b00;
131                             MemWrite    = 2'b00;
132                             Branch      = 2'b11;
133                             ALUop       = 5'b00010;
134                             SignExtend  = 1'b1;
135                             HiLoWrite   = 1'b0;
136                             AluSel      = 2'b11;
137                         end
138                     'h10:  // Instruction: Move hi register
139                         begin
140                             RegDst      = 2'b01;
141                             Target      = 2'b00;
142                             ALUSrc      = 1'b0;
143                             MemtoReg    = 2'b00;
144                             RegWrite    = 1'b1;
145                             MemRead     = 2'b00;
146                             MemWrite    = 2'b00;
147                             Branch      = 2'b00;
148                             ALUop       = 5'b00010;
149                             SignExtend  = 1'b1;
150                             HiLoWrite   = 1'b0;
151                             AluSel      = 2'b10;
152                         end
153                     'h12:  // Instruction: Move lo register
154                         begin
155                             RegDst      = 2'b01;
156                             Target      = 2'b00;
157                             ALUSrc      = 1'b0;
158                             MemtoReg    = 2'b00;
159                             RegWrite    = 1'b1;
160                             MemRead     = 2'b00;
161                             MemWrite    = 2'b00;
162                             Branch      = 2'b00;
163                             ALUop       = 5'b00010;
164                             SignExtend  = 1'b1;
165                             HiLoWrite   = 1'b0;
166                             AluSel      = 2'b01;
167                         end
168                     'h19:  // Instruction: Multiply unsigned
169                         begin
170                             RegDst      = 2'b00; //No destination
171                             Target      = 2'b00;
172                             ALUSrc      = 1'b0;
173                             MemtoReg    = 2'b00;
174                             RegWrite    = 1'b1;
175                             MemRead     = 2'b00;
176                             MemWrite    = 2'b00;
177                             Branch      = 2'b00;
178                             ALUop       = 5'b00010;
179                             SignExtend  = 1'b1;
180                             HiLoWrite   = 1'b1;
181                             AluSel      = 2'b00;
182                         end
183                     default: // Others
184                         begin
185                             RegDst      = 2'b01;
186                             Target      = 2'b00;
187                             ALUSrc      = 1'b0;
188                             MemtoReg    = 2'b00;
189                             RegWrite    = 1'b1;
190                             MemRead     = 2'b00;
191                             MemWrite    = 2'b00;
192                             Branch      = 2'b00;
193                             ALUop       = 5'b00010;
194                             SignExtend  = 1'b1;
195                             HiLoWrite   = 1'b0;
196                             AluSel      = 2'b00;
197                             end
198                 endcase
199             2:  // Instruction: J
200                 begin
201                     RegDst      = 2'b00;
202                     Target      = 2'b01;
203                     ALUSrc      = 1'b0;
204                     MemtoReg    = 2'b00;
205                     RegWrite    = 1'b0;
206                     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;
```

```
414                     MemtoReg    = 2'b00;
415                     RegWrite    = 1'b0;
416                     MemRead     = 2'b00;
417                     MemWrite    = 2'b01;
418                     Branch      = 2'b00;
419                     ALUop       = 5'b00000;
420                     SignExtend  = 1'b1;
421                     HiLoWrite   = 1'b0;
422                     AluSel      = 2'b00;
423                 end
424             default: //No default case
425                 begin
426                 end
427             endcase
428         end
429
430 endmodule
431
```

aluctrl.v

```
1  ////////////////////////////////////////////////
2  // ALUCTRL.V
3  //
4  // TU/e Eindhoven University Of Technology
5  // Eindhoven, The Netherlands
6  //
7  // Created: 21-11-2013
8  // Author: Bergmans, G (g.bergmans@student.tue.nl)
9  // Based on work by Sander Stuijk
10 //
11 // Function:
12 //     ALU controller
13 //
14 // Version:
15 //     (27-01-2014): initial version
16 //
17 ////////////////////////////////////////////////!/
18
19 module ALUCTRL(functionCode, ALUop, Shamt, ALUctrl);
20     input   [5:0]   functionCode;
21     input   [4:0]   ALUop;
22     input   [4:0]   Shamt;
23     output  [5:0]   ALUctrl;
24     reg     [5:0]   ALUctrl;
25
26     always @(functionCode or ALUop or Shamt)
27         begin : aluctrl_thread
28             case (ALUop) //synopsys parallel_case
29                 'h0:    // Add signed
30                     ALUctrl = 'h2;
31
32                 'h1:    // Subtract unsigned
33                     ALUctrl = 'h6;
34
35                 'h2:    // R-type instruction, look to functionCode
36                     begin
37                         case (functionCode)
38                             'h0:    // SLL
39                                 case (Shamt) //Check shift amount
40                                     1:
41                                         ALUctrl = 'hA;
42                                     2:
43                                         ALUctrl = 'hB;
44                                     8:
45                                         ALUctrl = 'hC;
46                                     default:
47                                         ALUctrl = 'h0;
48                                 endcase
49
50                             'h2:    // SRL
51                                 case (Shamt) //Check shift amount
52                                     1:
53                                         ALUctrl = 'hD;
54                                     2:
55                                         ALUctrl = 'hE;
56                                     8:
57                                         ALUctrl = 'hF;
58                                     default:
59                                         ALUctrl = 'h0;
60                                 endcase
61
62                             'h3:    // SRA
63                                 case (Shamt) //Check shift amount
64                                     1:
65                                         ALUctrl = 'h10;
66                                     2:
67                                         ALUctrl = 'h11;
68                                     8:
69                                         ALUctrl = 'h12;
70                                     default:
71                                         ALUctrl = 'h0;
72                                 endcase
73
74                             'h10:   // Move hi register (nop in ALU)
75                                 ALUctrl = 'h0;
76
77                             'h12:   // Move hi register (nop in ALU)
78                                 ALUctrl = 'h0;
79
80                             'h19:   // Multiply unsigned
81                                 ALUctrl = 'h13;
82
83                             'h20:   // Add signed
84                                 ALUctrl = 'h2;
85
86                             'h21:   // Add unsigned
87                                 ALUctrl = 'h3;
88
89                             'h23:   // Subtract unsigned
90                                 ALUctrl = 'h6;
91
92                             'h24:   // And
93                                 ALUctrl = 'h0;
94
95                             'h25:   // Or
96                                 ALUctrl = 'h1;
97
98                             'h26:  // Xor
99                                 ALUctrl = 'h4;
100
101                            'h2A:   //Set-on-less-than (2's complement)
102                                ALUctrl = 'h7;
103
```

```
                                '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

```
////////////////////////////////////////////////
// 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)
```

```
104                     begin
105                         sign = t[31:31];
106                         result = (t >> 1);
107                         result[31:31] = sign;
108                     end
109
110                 'h11:   // SRA (2 bit)
111                     begin
112                         sign = t[31:31];
113                         result = (t >> 2);
114                         result[31:30] = {sign, sign};
115                     end
116
117                 'h12:   // SRA (8 bit)
118                     begin
119                         sign = t[31:31];
120                         result = (t >> 8);
121                         result[31:24] = {sign, sign, sign, sign, sign, sign, sign, sign};
122                     end
123
124                 'h13:   //Multu
125                     begin
126                         c = s * t;
127                         result = c[31:0];
128                         result_hi = c[63:32];
129                     end
130
131                 default: //No default case: invallid opcode!
132                     begin
133                     end
134             endcase
135
136             // Calculate zero output
137             if (result == 0)
138                 zero = 1;
139             else
140                 zero = 0;
141
142             // Write results to output
143             r = result;
144             r2 = result_hi;
145             z = zero;
146         end
147
148 endmodule
149
```

VPL