
EEL 5722C

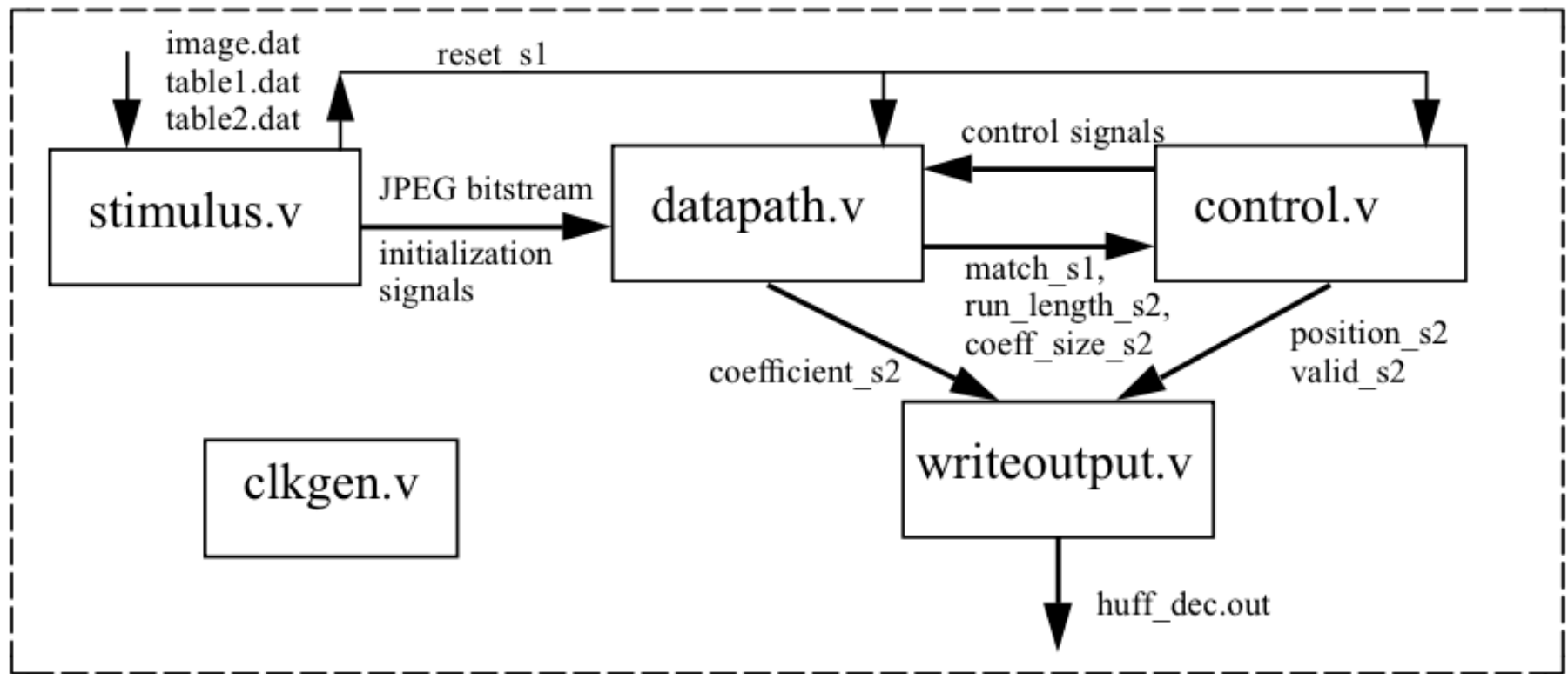
Field-Programmable Gate Array Design

Lecture 13: Mid-Term and PA2

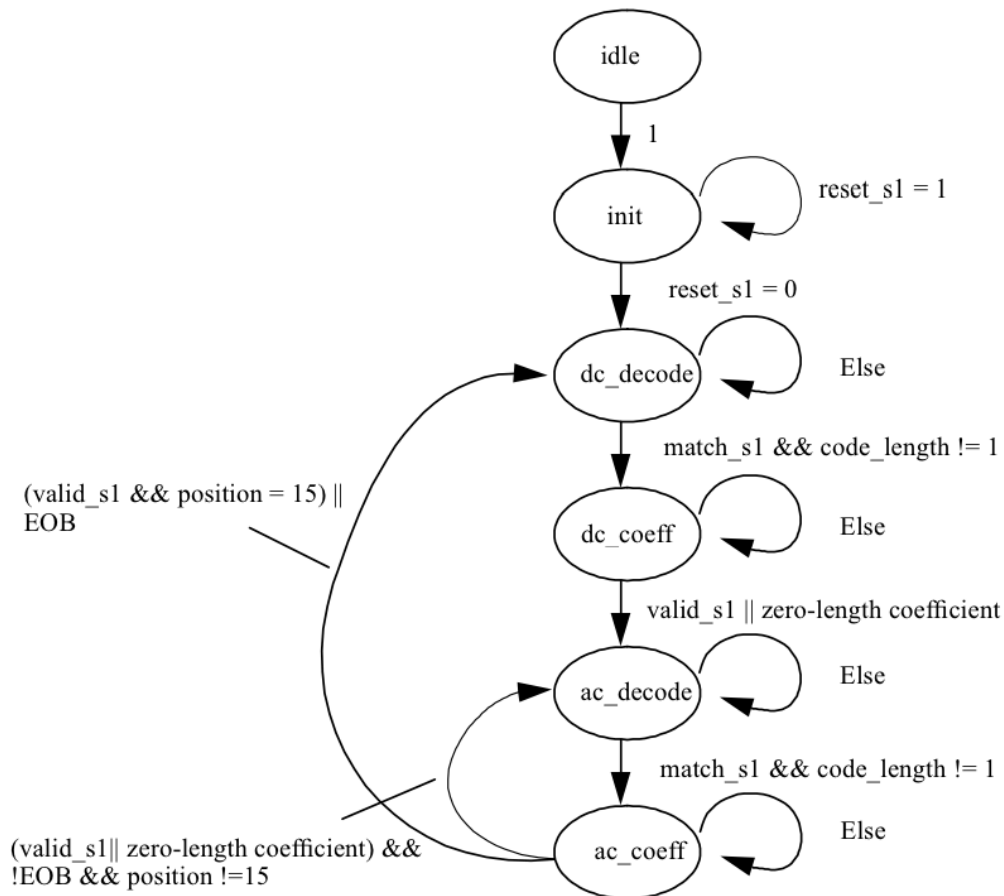
Prof. Mingjie Lin



top.v

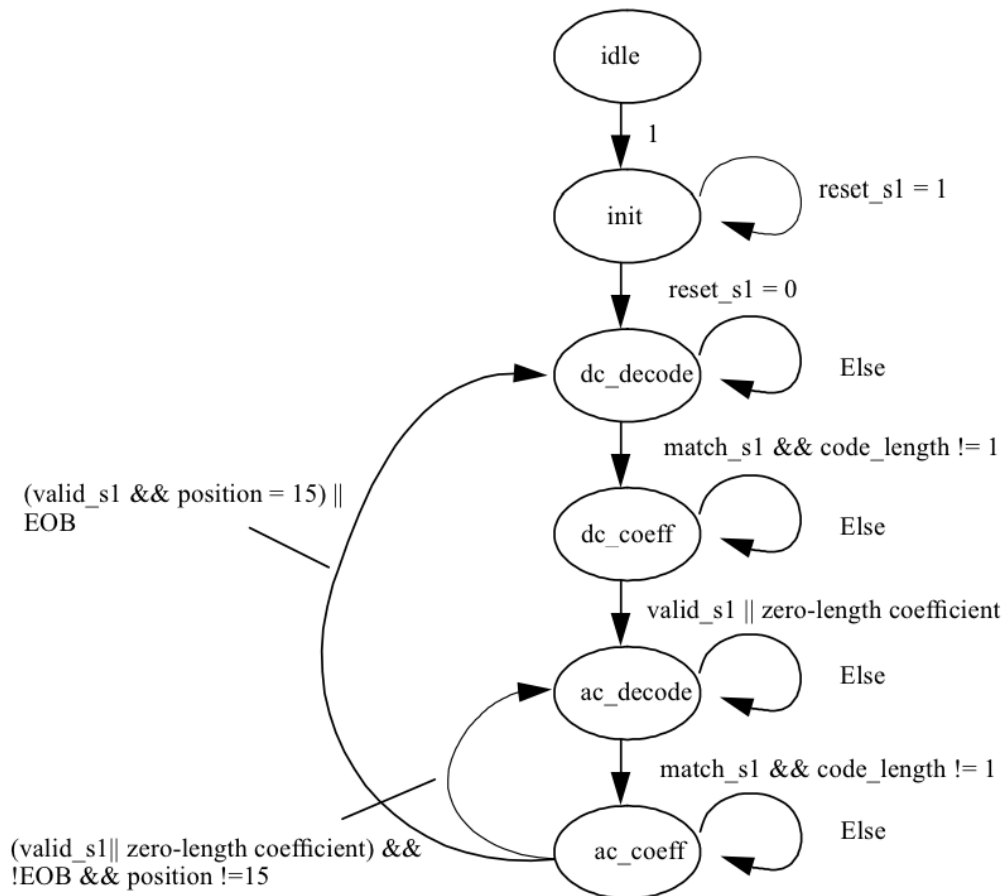


Control.v



```
always @(curr_state_s1 or match_s1 or valid_s1 or
coeff_size_s1 or position_s1 or reset_s1 or
match_del_s1 or reset_sr_s1 or run_length_s1)
begin
  case(curr_state_s1)
    `idle:
      if (reset_s1==1'b1)
        next_state_s1=`init;
      else
        next_state_s1=`idle;
    `init:
      if (reset_s1==1'b0)
        next_state_s1=`dc_decode;
      else
        next_state_s1=`init;
    `dc_decode:
      if (reset_sr_s1==1'b0 && match_s1==1'b1)
        next_state_s1 = `dc_coeff;
      else
        next_state_s1=`dc_decode;
```

Control.v



```

`dc_coeff:
  if (coeff_size_s1 == 4'b0 && match_del_s1 == 1'b1 ||
      valid_s1==1'b1)
    next_state_s1= `ac_decode;
  else
    next_state_s1=`dc_coeff;

`ac_decode:
  if (reset_sr_s1==1'b0 && match_s1==1'b1)
    next_state_s1= `ac_coeff;
  else
    next_state_s1=`ac_decode;

`ac_coeff:
  if (coeff_size_s1 == 4'b0 && match_del_s1 == 1'b1 ||
      valid_s1==1'b1)
    begin
      if (position_s1==4'b1111 || run_length_s1 == 4'b0 &&
          coeff_size_s1 == 4'b0 && match_del_s1 == 1'b1)
        next_state_s1=`dc_decode;
      else
        next_state_s1=`ac_decode;
    end
  else
    next_state_s1=`ac_coeff;

endcase
end
    
```

Datapath.v

```
1 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 //      Block Name      : datapath.v
3 //      Block Description :
4 //      This block implements the datapath portion of the Huffman decoder
5 // and shift registers, adder, comparator and latches that are
6 // necessary to implement the Huffman Code
7 //
8 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
9
10 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
11 module    datapath(bitstream_s1,maxcode_v1, base_v1,
12      reset_sr_s2,coeff_en_b_s2,match_s1, address_s1,
13      coefficient_s2,phi1, phi2);
14 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
15
16 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
17 // Port Declarations
18 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
19
20 input     phi1,phi2; // - 2 phase clocks
21
22 //INPUT
23 input     bitstream_s1; // - JPEG bitstream
24 input     reset_sr_s2; // - Signal resets shift-reg's when a new
25           // huffman code or coefficient is identified.
26 input     coeff_en_b_s2; // - This signal controls whether the bitstream
27           // shift register is reset to a 1 or 0.
28 input [8:0] maxcode_v1; // - Table 1 Data: Maxcode
29 input [5:0] base_v1; // - Table 1 Data: Base
30
31
32 output    match_s1; // - Indicates that the Huffman code length has
33           // been determined.
34 output [9:0] coefficient_s2;
35 output [5:0] address_s1; // - Table 2 Address that selects the corresponding
36
37 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
38 // Datapath Variable Declarations
39 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
40
41 reg [5:0] address_s1; // - Table 2 Address that selects the corresponding
42 wire [5:0] address_s2; // coeff_size and run_length
43 reg [8:0] maxcode_s1; // - Largest Huffman code for a given code_length.
44 reg [8:0] maxcode_s2;
45 wire [8:0] maxcode_v1;
46 reg [5:0] base_s2; // - Value from Table 1 that is used to calculate the
47 wire [5:0] base_v1; // the Table 2 address.
48 wire [5:0] bits_s2; // - 6 LSB's of the 10-bit shift register
49 reg [9:0] bits_s1; // - Contents of 10-bit shift register
50 reg [9:0] coefficient_s2;
```

```
53 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
54 //                                DATAPATH                                //
55 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
56
57 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
58 // 10-bit Shift Register for Jpeg Bitstream:
59 // This block receives the jpeg bitstream and sends latched output to
60 // a) Table 2 address calculation
61 // b) Huffman Code-Length determination. (match_s1 generation)
62 // The shift register is reset after a huffman code or coefficient has been
63 // shifted in. The shift register should NOT be reset if the coefficient
64 // size is zero. Note that the first bit of the shift register is never reset!
65 //
66 // A shift register consists of a number of latches connected in series with
67 // alternating phi1 and phi2 clock signals. The data is serially shifted from
68 // latch to latch.
69 // Inputs: bitstream_s1 -- encoded bitstream
70 //         reset_sr_s2 -- When this signal is 1 the shift register is reset.
71 //         coeff_en_b_s2 -- When a coefficient is shifted in this signal=0
72 //                                and when a huffman code is shifted in it's 1.
73 // Outputs: bits_s1, bits_s2 signals used by the datapath.
74 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
75
76 // Interval Variable Declarations
77 reg [9:0] par_out_s2;
78 wire [9:0] par_out_tmp_s2;
79 wire reset_tmp_s2;
80
81 always @(phi1 or bitstream_s1 or bits_s1[9:0])
82   if (phi1)
83     begin
84       par_out_s2[0] = bitstream_s1;
85       par_out_s2[9:1] = bits_s1[8:0];
86     end
87 always @(phi2 or par_out_tmp_s2)
88   if (phi2)
89     begin
90       bits_s1 = par_out_tmp_s2;
91     end
92
```

Datapath.v

```
93 // The coefficients that are placed on the bitstream may be positive or negative. 123 ////////////////////////////////////////////////////////////////////
94 // It may be determined if the coefficient is positive or negative by looking at the 124 // Table 2 address calculation:
95 // first bit on the bitstream. The bitstream is reset to all 1's if the coefficient 125 // The Huffman code in the bitstream shift register is added with base
96 // is negative and all zeroes if the coefficient is positive. 126 // to create the lookup table 2 address.
97 assign reset_tmp_s2 = ~(coeff_en_b_s2 | par_out_s2[0]); 127 ////////////////////////////////////////////////////////////////////
98 assign par_out_tmp_s2[9:1] = reset_sr_s2 ? {9{reset_tmp_s2}} : par_out_s2[9:1]; 128 assign address_s2 = base_s2 + bits_s2;
99 129
100 // The first bit of the shift register is never reset because a new bit is 130 always @(phi2 or address_s2)
101 // shifted in each cycle. 131     if (phi2)
102 assign par_out_tmp_s2[0] = par_out_s2[0]; 132         address_s1 = address_s2;
103 133 // End of address calculation
104 // bits_s2 only needs to be 6 bits long. 134
105 assign bits_s2 = par_out_tmp_s2[5:0]; 135 ////////////////////////////////////////////////////////////////////
106 // End of 10-bit shift Register 136 // Compare Maxcode to Bits:
107 137 // If bits <= maxcode them match=1 and the Huffman code length has been
108 138 // determined. This comparison between maxcode and bits is always made
109 //////////////////////////////////////////////////////////////////// 139 // but match_s1 is only sampled by the control when a Huffman code is being
110 // Latches for Table1/Table2 outputs 140 // read from the bitstream and code_length is greater than 1.
111 //////////////////////////////////////////////////////////////////// 141 ////////////////////////////////////////////////////////////////////
112 142 always @(phi2 or maxcode_s2)
113 // Latch Base and Maxcode after Read 143     if (phi2)
114 always @(phi1 or base_v1) 144         maxcode_s1 = maxcode_s2;
115     if (phi1) 145
116         base_s2 = base_v1; 146 assign match_s1=(maxcode_s1 < bits_s1) ? 1'b0 : 1'b1;
117 always @(phi1 or maxcode_v1) 147
118     if (phi1) 148 // End of comparator
119         maxcode_s2 = maxcode_v1; 149
120 150 ////////////////////////////////////////////////////////////////////
121 // End of Latches of Table1/Table2 outputs 151 // Latch for Coefficient_s2 Output
122 152 ////////////////////////////////////////////////////////////////////
153 always @(phi1 or bits_s1)
154     if (phi1)
155         coefficient_s2 = bits_s1;
156 endmodule // datapath
```

Regfile.v

```
128 ////////////////////////////////////////////////////////////////////
129 // 8-bit Code_length SR:
130 // This functional block generates the wordline selects for lookup table 1.
131 // Only one wordline will be high at one time.
132 //
133 // In SRAMs the wordline selects which of the 8 DC/AC values will be read
134 // on the bit-lines (maxcode_v1 and base_v1). Because the wordlines are selected
135 // in order (code_length = 2-9) a shift register is used to select the wordlines
136 // instead of an incrementer.
137 //
138 // When the shift register is reset by reset_sr_s2, the input to the shift register,
139 // count_in_s2, is high for one cycle. The output, wordnum_s1 selects one of the
140 // bitlines each cycle until the huffman code length is determined.
141 //
142 // Inputs: reset_sr_s2, phi1, phi2
143 // Output: wordnum_s1
144 ////////////////////////////////////////////////////////////////////
145 // Internal Variable Declarations
146 reg [7:1] state_s2;
147 wire [7:0] state_tmp_s2;
148
149 // Implement all the phi1 latches in the shift register.
150 always @(phi1 or wordnum_s1)
151     if (phi1)
152         state_s2[7:1] = wordnum_s1[6:0];
153
154 // Implement all the phi2 latches in the shift register.
155 always @(phi2 or state_tmp_s2)
156     if (phi2)
157         wordnum_s1 = state_tmp_s2;
158
159 // When reset_sr_s2 is high, the shift register is reset to 00000001.
160 assign state_tmp_s2[7:1] = reset_sr_s2 ? 7'b1 : state_s2[7:1];
161 assign state_tmp_s2[0] = reset_sr_s2;
162 // End of code_length shift register.
163
164
165 endmodule // regfile
```

Midterm

Final issues

- Come by my office hours (right after class)
- Any questions or concerns?