Topic 11

Examples

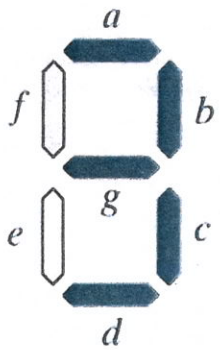
國立高雄大學電機工程學系

陳春僥

cychen@ieee.org

June 13, 2019

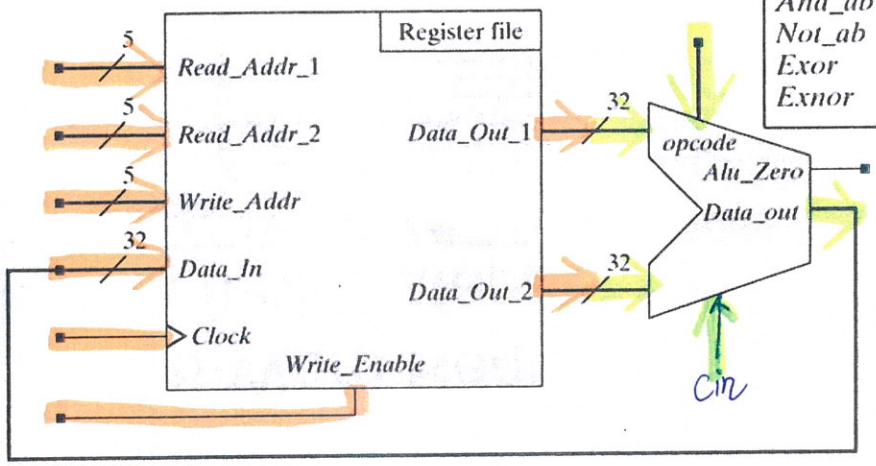
Seven-Segment LED Controller




```
module Seven_Seg_Display ( output reg [6: 0] Display,  input [3: 0] BCD);
                            //  abc_defg
  parameter BLANK          = 7'b111_1111;
  parameter   ZERO         = 7'b000_0001;                    // h01
  parameter   ONE          = 7'b100_1111;                    // h4f
  parameter   TWO          = 7'b001_0010;                    // h12
  parameter   THREE        = 7'b000_0110;                    // h06
  parameter   FOUR         = 7'b100_1100;                    // h4c
  parameter   FIVE         = 7'b010_0100;                    // h24
  parameter   SIX          = 7'b010_0000;                    // h20
  parameter   SEVEN        = 7'b000_1111;                    // h0f
  parameter   EIGHT        = 7'b000_0000;                    // h00
  parameter   NINE         = 7'b000_0100;                    // h04
  always @ (BCD)
     case (BCD)
        0:                 Display = ZERO;
        1:                 Display = ONE;
        2:                 Display = TWO;
        3:                 Display = THREE;
        4:                 Display = FOUR;
        5:                 Display = FIVE;
        6:                 Display = SIX;
        7:                 Display = SEVEN;
        8:                 Display = EIGHT;
        9:                 Display = NINE;
       default:  Display = BLANK;
     endcase
endmodule
```

A 32-Word Register File + an ALU with a 32-bit Datapath

Operand	Function
Add	*a + b + c_in*
Subtract	*a + ~b + c_in*
Subtract_a	*b + ~a + ~c_in*
Or_ab	{1'b0, *a* | *b*}
And_ab	{1'b0, *a* & *b*}
Not_ab	{1'b0, (~*a*) & *b*}
Exor	{1'b0, *a* ^ *b*}
Exnor	{1'b0, *a* ~^ *b*}



A 8-Word Register File + an ALU with an 8-bit Datapath

1

```
module alu_8b (
  output reg [8: 0] alu_out,
  input  [7: 0] a, b,
  input  c_in,
  input  [2: 0] opcode
);
  parameter [2: 0] add = 0;
  parameter [2: 0] subtract = 1;
  parameter [2: 0] subtract_a = 2;
  parameter [2: 0] or_ab = 3;
  parameter [2: 0] and_ab = 4;
  parameter [2: 0] not_ab = 5;
  parameter [2: 0] exor = 6;
  parameter [2: 0] exnor = 7;

  always @ (a, b, c_in, opcode)
    case (opcode)
      add:  alu_out = a + b + c_in;
      subtract:  alu_out = a + (~b) + c_in;
      subtract_a:  alu_out = b + (~a) + ~c_in;
      or_ab:  alu_out = {1'b0, a | b};
      and_ab:  alu_out = {1'b0, a & b};
      not_ab:  alu_out = {1'b0, (~a) &b};
      exor:  alu_out = {1'b0, a^b};
      exnor:  alu_out = {1'b0, a ~^ b};
    endcase
endmodule
```

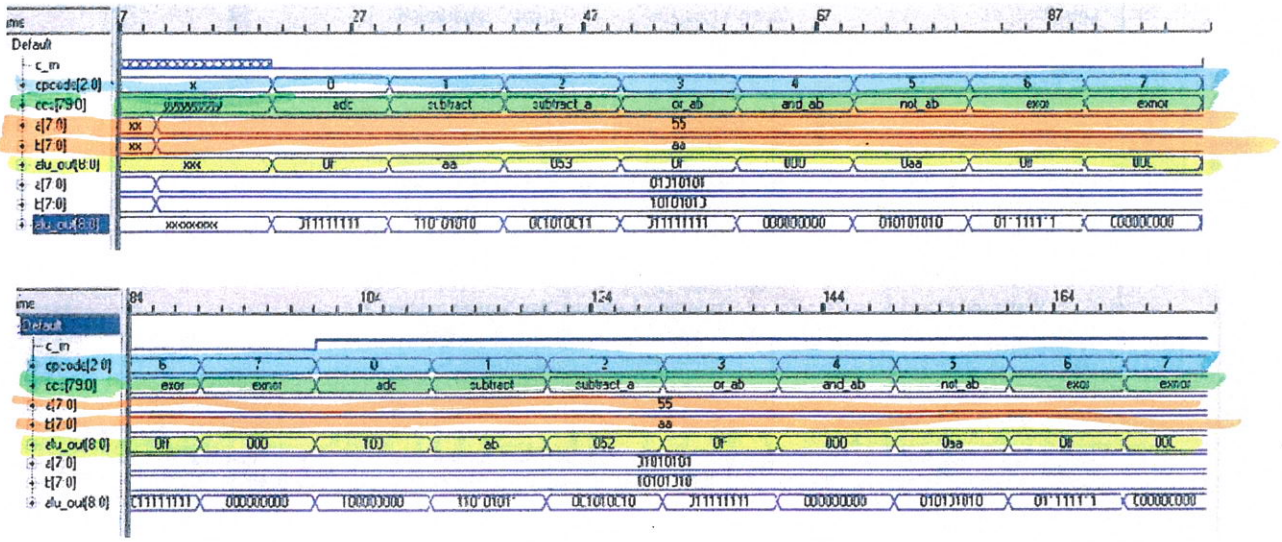
2

```
module t_alu_8b;
 reg  [7: 0]  a, b;
 reg   c_in;
 reg  [2: 0] opcode;
 reg  [79: 0] ocs;
 wire  [8: 0] alu_out;
 integer  j, k;

 parameter [2: 0] add = 0;
 parameter [2: 0] subtract = 1;
 parameter [2: 0] subtract_a = 2;
 parameter [2: 0] or_ab = 3;
 parameter [2: 0] and_ab = 4;
 parameter [2: 0] not_ab = 5;
 parameter [2: 0] exor = 6;
 parameter [2: 0] exnor = 7;

 parameter [79: 0] ocs_0 = "add";
 parameter [79: 0] ocs_1 = "subtract";
 parameter [79: 0] ocs_3 = "or_ab";
 parameter [79: 0] ocs_4 = "and_ab";
 parameter [79: 0] ocs_5 = "not_ab";
 parameter [79: 0] ocs_6 = "exor";
 parameter [79: 0] ocs_7 = "exnor";

    alu_8b      M0 (alu_out, a, b, c_in, opcode);
```

3

```
initial #1000 $finish;
initial begin
 #10    a = 8'h55; //0101_0101
  b = 8'haa; //1010_1010
 for (j = 0; j <= 1;j = j + 1)
 for (k = 0; k <=7; k = k + 1)
begin
 #10  c_in = j;
        case (k)
          0: opcode = add;
          1: opcode = subtract;
          2: opcode = subtract_a;
          3: opcode = or_ab;
          4: opcode = and_ab;
          5: opcode = not_ab;
          6: opcode = exor;
          7: opcode = exnor;
 endcase
end
end

 always @(opcode)
  case (opcode)
   add: ocs = ocs_0;
   subtract: ocs = ocs_1;
   subtract_a: ocs = ocs_2;
   or_ab: ocs = ocs_3;
   and_ab: ocs = ocs_4;
   not_ab: ocs = ocs_5;
   exor: ocs = ocs_6;
   exnor: ocs = ocs_7;
 endcase
endmodule
```

A 8-Word Register File + an ALU with an 8-bit Datapath



A 8-Word Register File + an ALU with an 8-bit Datapath

3 (整個CUT)

```
module alu_8b_register (
output [8: 0] alu_out,          // Re-sized
input [7: 0]  Data_in,
input  [2: 0] Read_Addr_1, Read_Addr_2, Write_Addr,
input [2: 0] opcode,
input   Write_Enable, Clock,
input   c_in
);
 wire  [7: 0]  Data_Out_1, Data_Out_2;

 alu_8b              M0_ALU (alu_out, Data_Out_1, Data_Out_2, c_in, opcode);
Register_File M1_Reg_File (Data_Out_1, Data_Out_2, Data_in, Read_Addr_1,
   Read_Addr_2, Write_Addr, Write_Enable, Clock);
endmodule
```

1

```
module Register_File (
 output  [7: 0] Data_Out_1, Data_Out_2,
 input [7: 0]  Data_in,
 input  [2: 0] Read_Addr_1, Read_Addr_2, Write_Addr,
 input   Write_Enable, Clock
);
 reg [7: 0] Reg_File [31: 0]; // 8 x 32 Memory declaration

 assign Data_Out_1 = Reg_File[Read_Addr_1];
 assign Data_Out_2 = Reg_File[Read_Addr_2];

 always @ (posedge Clock) begin
  if (Write_Enable) Reg_File [Write_Addr] <= Data_in;
 end
endmodule
```

正緣寫入

2

```
// Test Plan: Register File
// Write to memory with walking ones
// Verify read of walking ones from each port
/*
module t_Register_File ();
 wire  [7: 0] Data_Out_1, Data_Out_2;
 reg  [7: 0]  Data_in;
 reg  [2: 0]  Read_Addr_1, Read_Addr_2, Write_Addr;
 reg    Write_Enable, Clock;
 integer   k;

 Register_File M0(Data_Out_1, Data_Out_2, Data_in,
Read_Addr_1, Read_Addr_2,
Write_Addr, Write_Enable, Clock);

 initial #500 $finish;
 initial begin Clock = 0; forever #5 Clock = ~Clock; end
 initial begin
   Data_in = 8'b1000_0000;
   Write_Enable = 1;
   Write_Addr = 0;
   Read_Addr_1 = 0;
   Read_Addr_2 = 0;
   for (k = 0; k <= 31; k = k + 1) begin
     @ (negedge Clock)
      if (Data_in == 8'b1000_0000) Data_in = 8'b0000_0001;
        else Data_in <= Data_in << 1;
      Write_Addr <= Write_Addr + 1;
      Read_Addr_1 <= Read_Addr_1 + 1;
      Read_Addr_2 <= Read_Addr_2 + 1;
   end
 end
endmodule
```

負緣讀出&更新位址

running 1's

A 32-Word Register File +an ALU with an 8-bit Datapath

```
// Test plan: Register file and ALU
//Verify write of walking ones to all registers
//Verify alu operations for data read from registers

 module t_alu_8b_register;
  wire [8: 0] alu_out;  // Re-sized
  reg  [7: 0]  Data_in;
reg  [2: 0]  Read_Addr_1, Read_Addr_2
reg  [2: 0] opcode;
reg    Write_Enable, Clock;
reg    c_in;
reg  [79: 0] ocs;
integer    k;
parameter [2: 0] add = 0;
parameter [2: 0] subtract = 1;
parameter [2: 0] subtract_a = 2;
parameter [2: 0] or_ab = 3;
parameter [2: 0] and_ab = 4;
parameter [2: 0] not_ab = 5;
parameter [2: 0] exor = 6;
parameter [2: 0] exnor = 7;

parameter [79: 0] ocs_0 = "add";
parameter [79: 0] ocs_1 = "subtract";
parameter [79: 0] ocs_2 = "subtract_a";
parameter [79: 0] ocs_3 = "or_ab";
parameter [79: 0] ocs_4 = "and_ab";
parameter [79: 0] ocs_5 = "not_ab";
parameter [79: 0] ocs_6 = "exor";
parameter [79: 0] ocs_7 = "exnor";
```

4

```
alu_8b_register   M0 (alu_out, Data_in, Read_Addr_1, Read_Addr_2, Write_Addr, opcode,
Write_Enable, c_in, Clock);


 initial #500 $finish;
 initial begin Clock = 0; forever #5 Clock = ~Clock; end
 initial begin
  #1 Data_in = 8'b1000_0000;
  opcode = add;
  c_in = 0;
  Write_Enable = 1;
  Write_Addr = 0;
  Read_Addr_1 = 0;
  Read_Addr_2 = 0;
  for (k = 0; k <= 31; k = k + 1) begin
   @ (negedge Clock)
     if (Data_in == 8'b1000_0000) Data_in = 8'b0000_0001;
      else Data_in <= Data_in << 1;
     Write_Addr <= Write_Addr + 1;
     Read_Addr_1 <= Read_Addr_1 + 1;
     Read_Addr_2 <= Read_Addr_2 + 1;
  end
 end
```

5

```
 always @(opcode)
  case (opcode)
  add:  ocs = ocs_0;
  subtract: ocs = ocs_1;
  subtract_a: ocs = ocs_2;
  or_ab:  ocs = ocs_3;
  and_ab:  ocs = ocs_4;
  not_ab:  ocs = ocs_5;
   exor:  ocs = ocs_6;
   exnor:  ocs = ocs_7;
  endcase
endmodule
```

6

A 8-Word Register File + an ALU with an 8-bit Datapath