EGC221
Class Notes
4/19/2023

Baback Izadi
Division of Engineering Programs
bai@engr.newpaltz.edu

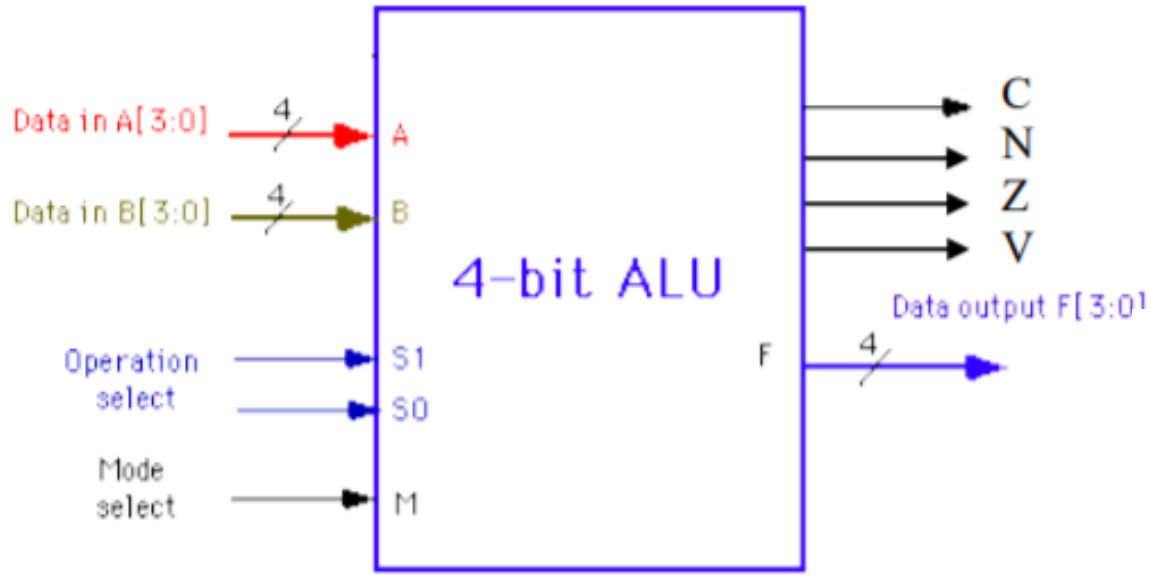


Figure 1: Block diagram of the 4-bit ALU.

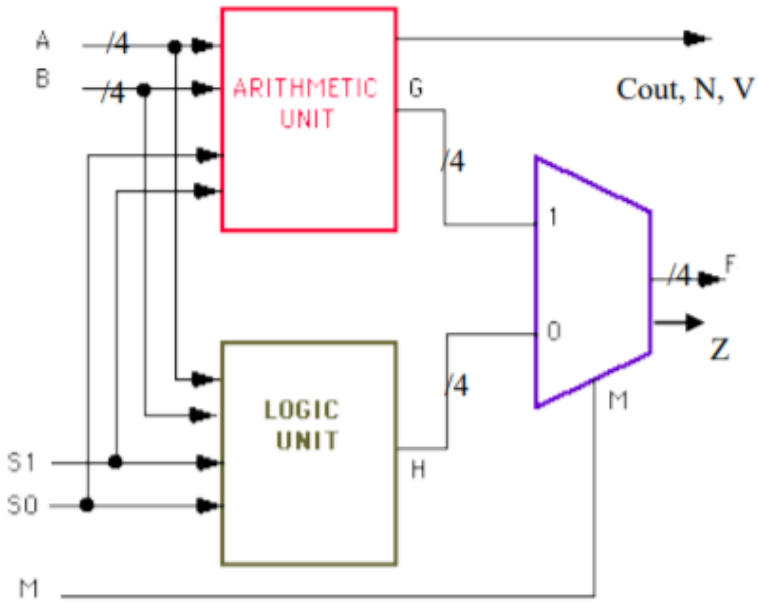


Figure 2: Block diagram of the ALU

Table 1: Functions of ALU				
Logic				
M2	S1	S0	FUNCTION	OPERATION (bit wise)
0	0	0	A · B	AND
0	0	1	A + B	OR
0	1	0	A ⊕ B	XOR
0	1	1	A’	NOT
Arithmetic				
M2	S1	S0	FUNCTION	OPERATION
1	0	0	A + B	Addition
1	0	1	A - B	Subtract
1	1	0	A + 1	Increment
1	1	1	A - 1	Decrement

V = CY ∧ temp[3]

3 bit adder

A[2:0] + B[2:0]

temp[3:0]

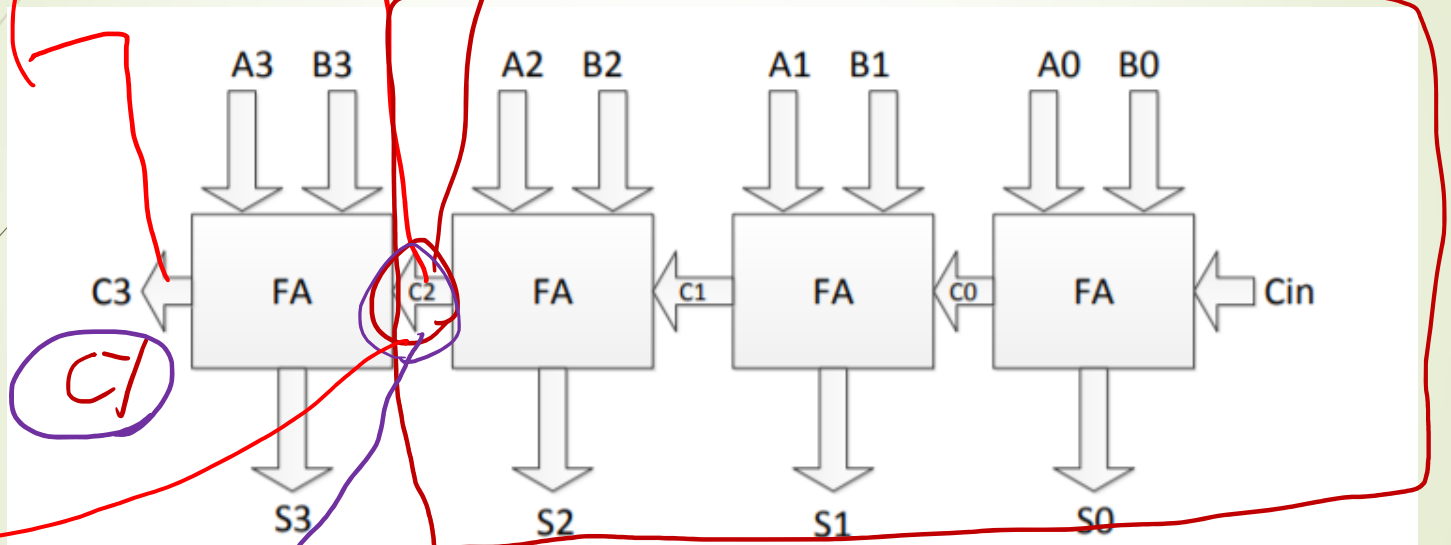


Figure 3. Hierarchical block diagram of 4-bit adder

C3

temp[3]

{CY, S} = A + B

A_3 A_2 A_1 A_0

B_3 B_2 B_1 B_0

CY S_3 S_2 S_1 S_0

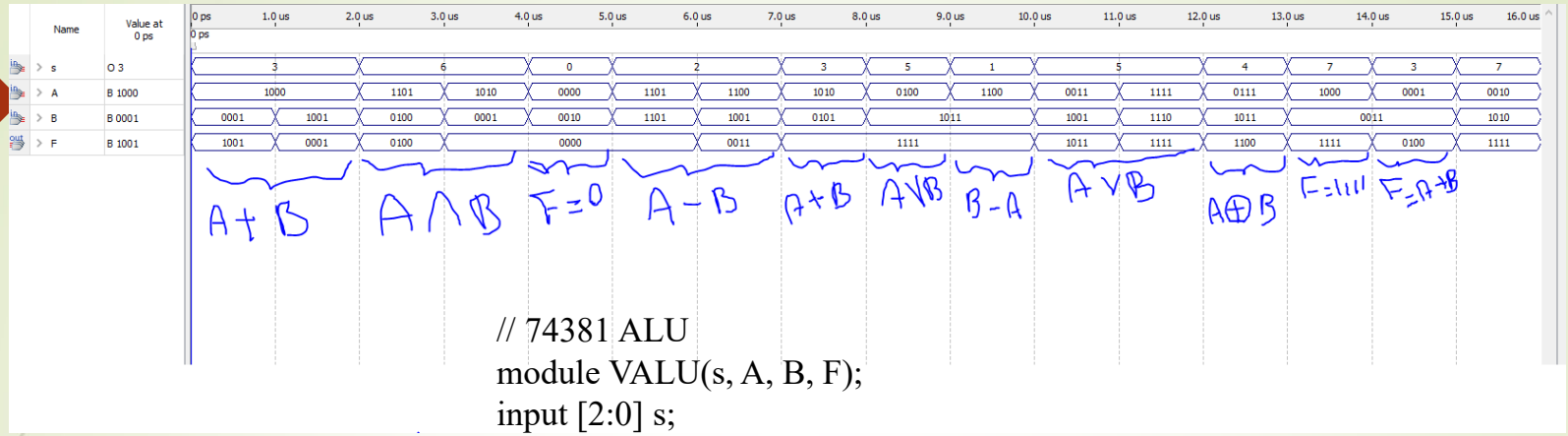
Design of an ALU using Case Statement

S	Function
0	A AND B
1	A OR B
2	A XOR B
3	A'
4	A+B
5	A-B
6	A+1
7	A-1

Z = (F == 4'b0);

```
// 74381 ALU
module alu(s, A, B, F);
input [2:0] s;
input [3:0] A, B;
output [3:0] F;
reg [3:0] F;
always @(s or A or B)
case (s)
0: F = A & B;
1: F = A | B;
2: F = A ^ B;
3: F = ~A;
4: F = A + B;
5: F =A-B;
6: F =A+4'b0001;
7: F = A+4'b1111;
endcase
endmodule
```

```
4: begin
    {CY, F} = A + B;
     temp[3:0] = A[2:0] + B[2:0];
    sign = F[3];
     if(F == 0)
       z = 1;
      else
       z = 0;
     V = temp[3] ^ CY;
end
```




```
// 74381 ALU
module VALU(s, A, B, F);
input [2:0] s;
input [3:0] A, B;
output [3:0] F;
reg [3:0] F;
always @(s or A or B)
case (s)
0: F = 4'b0000;
1: F = B - A;
2: F = A - B;
3: F = A + B;
4: F = A ^ B;
5: F = A | B;
6: F = A & B;
7: F = 4'b1111;
endcase
endmodule
```

. Concatenation and Replication in Verilog

- The concatenation operator "{ , }" combines (concatenates) the bits of two or more data objects. The objects may be scalar (single bit) or vectored (multiple bit). Multiple concatenations may be performed with a constant prefix and is known as replication.

```
module Concatenation (A, B, Y);
        input [2:0] A, B;
        output [14:0] Y;
        parameter C=3'b011;
        reg [14:0] Y;
        always @(A or B)
        begin
                Y={A, B, {2{C}}, 3'b110};
        end
endmodule
```

Bit-wise Operations in Verilog

```
module Bitwise (A, B, Y);
        input [6:0] A;
        input [5:0] B;
        output [6:0] Y;
        reg [6:0] Y;
        always @(A or B)
        begin
                Y[0]=A[0]&B[0]; //binary AND
                Y[1]=A[1]|B[1]; //binary OR
                Y[2]=!(A[2]&B[2]); //negated AND
                Y[3]=!(A[3]|B[3]); //negated OR
                Y[4]=A[4]^B[4]; //binary XOR
                Y[5]=A[5]~^B[5]; //binary XNOR
                Y[6]=!A[6]; //unary negation
        end
endmodule
```

```
module ALU8bit( Opcode, Operand1, Operand2,
Result, flagC, flagZ); input [2:0] Opcode;
input [7:0] Operand1, Operand2;
output reg [15:0] Result = 16'b0;
output reg flagC = 1'b0, flagZ = 1'b0;
parameter [2:0] ADD = 3'b000, SUB = 3'b001, MUL =
3'b010, AND = 3'b011, OR = 3'b100, NAND = 3'b101,
NOR = 3'b110, XOR = 3'b111;
always @ (Opcode or Operand1 or Operand2)
begin
case (Opcode)
ADD: begin
Result = Operand1 + Operand2;
flagC = Result[8];
flagZ = (Result == 16'b0);
end
SUB: begin
Result = Operand1 - Operand2;
flagC = Result[8];
flagZ = (Result == 16'b0);
end
MUL: begin
Result = Operand1 * Operand2;
flagZ = (Result == 16'b0);
end
```

```
AND: begin
Result = Operand1 & Operand2;
flagZ = (Result == 16'b0);
end
OR: begin
Result = Operand1 | Operand2;
flagZ = (Result == 16'b0);
end
NAND: begin
Result = ~(Operand1 & Operand2);
flagZ = (Result == 16'b0);
end
NOR: begin
Result = ~(Operand1 | Operand2);
flagZ = (Result == 16'b0);
end
XOR: begin
Result = Operand1 ^ Operand2;
flagZ = (Result == 16'b0);
end
default: begin
Result = 16'b0;
flagC = 1'b0; flagZ = 1'b0;
end
endcase
end
endmodule
```