Minority Combinational Project

Capture, Create, Implement Process (Recap)

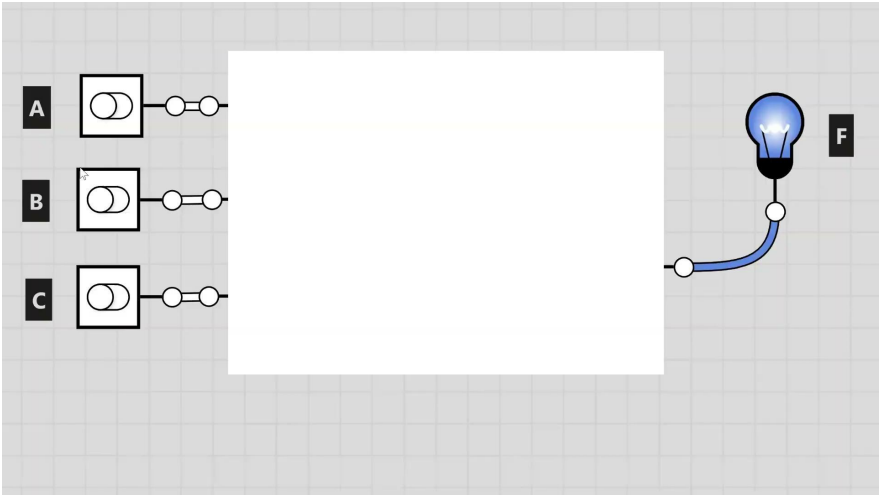
1. **Capture** the behavior of the function
   a. Either through a truth table or equation (Whichever is easier)

2. **Create** equations (If you haven't already)
   a. Canonical Sums-of-Products

3. **Implement** a gate-based circuit for <u>each</u> output
   a. Can then translate that gate directly into Verilog

Minority Function Implementation

- **Minority Function**
  - Inputs: 3 1-bit inputs (A, B, C)
  - Outputs: 1-bit output (F)
  - **F will go high if there are one or fewer logical high across the 3 inputs**



Capture the Function Behavior (Truth Table)

- **Minority Function**
  - Inputs: 3 1-bit inputs (A, B, C)
  - Outputs: 1-bit output (F)
  - **F will go high if there are one or fewer logical high across the 3 inputs**

A	B	C	F

Capture the Function Behavior (Truth Table)

- **Minority Function**
  - Inputs: 3 1-bit inputs (A, B, C)
  - Outputs: 1-bit output (F)
  - **F will go high if there are one or fewer logical high across the 3 inputs**

A	B	C	F

Capture the Function Behavior (Truth Table)

- **Minority Function**
  - Inputs: 3 1-bit inputs (A, B, C)
  - Outputs: 1-bit output (F)
  - **F will go high if there are one or fewer logical high across the 3 inputs**

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Create Your Equation(s)

The 3-step process produces a Canonical SOP!!

1. Determine which rows resulted in ‘F’ being True (1)

2. Generate an minterm for Each Combination
   a. Only that combination should result in output being true

For combination (0, 0, 0) → A’ ● B’ ● C’

The only time this expression is True is when (0, 0, 0) is passed

Fot combination (0, 0, 1) → A’ ● B’ ● C

The only time this expression is True is when (0, 0, 1) is passed

3. OR each of the minterms

F = ?

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Create Your Equation(s)

The 3-step process produces a Canonical SOP!!

1. Determine which rows resulted in ‘F’ being True (1)

2. Generate an minterm for Each Combination
   a. Only that combination should result in output being true

For combination (0, 0, 0) → A’ ● B’ ● C’

The only time this expression is True is when (0, 0, 0) is passed

Fot combination (0, 0, 1) → A’ ● B’ ● C

The only time this expression is True is when (0, 0, 1) is passed

3. OR each of the minterms

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

F = (A’ ● B’ ● C’) + (A’ ● B’ ● C) + (A’ ● B ● C’) + (A ● B’ ● C’)

Implement Your Gate-Based Circuit

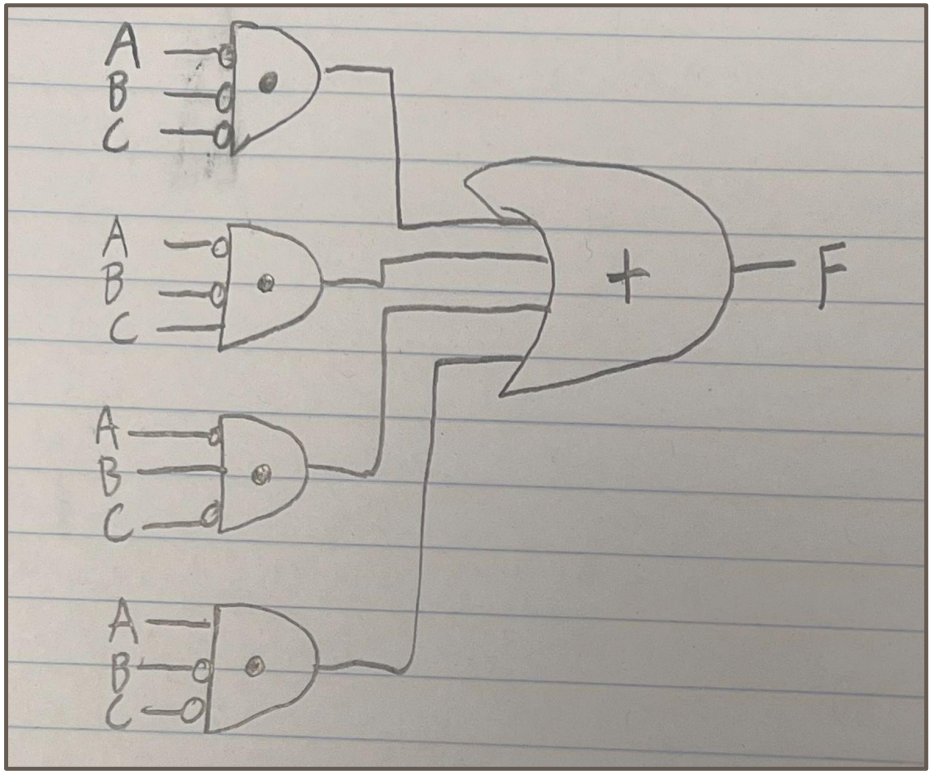
F =

(A’ ● B’ ● C’) +

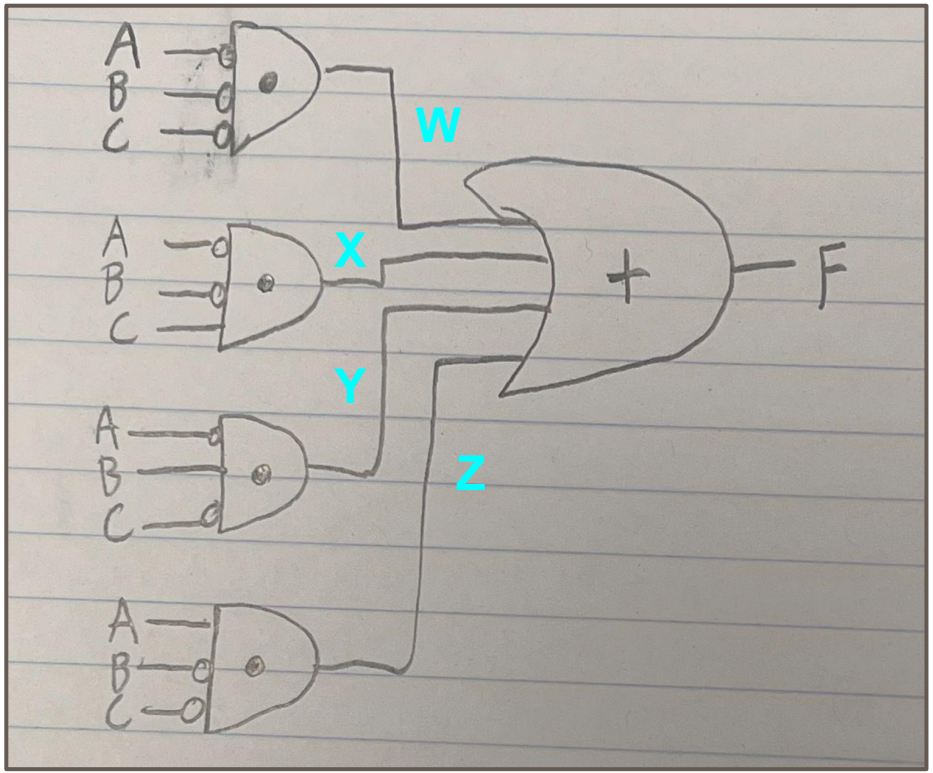
(A’ ● B’ ● C) +

(A’ ● B ● C’) +

(A ● B’ ● C’)



Implementation in **STRUCTURAL** Verilog




```
module Minority(
  input a, b, c,
  output out
);

  // Step 1. Declare your wires
  wire na, nb, nc;
  wire w, x, y, z;

  // Step 2. Drive some not a, b, c signals
  not n1(na, a);  // a'
  not n2(nb, b);  // b'
  not n3(nc, c);  // c'

  // Step 3. Drive w, x, y, z
  and a1(w, na, nb, nc);  // a' & b' & c'
  and a2(x, na, nb, c);   // a' & b' & c
  and a3(y, na, b, nc);   // a' & b & c'
  and a4(z, a, nb, nc);   // a & b' & c'

  // Step 4. Drive our output
  or o1(out, w, x, y, z);

endmodule
```

Implementation in **BEHAVIORAL** Verilog

Can just use your boolean equation

F = (A’ • B’ • C’) +
(A’ • B’ • C) +
(A’ • B • C’) +
(A • B’ • C’)

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b & ~c) |
               (~a & ~b & c) |
               (~a & b & ~c) |
               (a & ~b & ~c);

endmodule
```

Simplifying Boolean Equation (w/ Behavioral Verilog)

OLD

$$F = (A' \bullet B' \bullet C') + (A' \bullet B' \bullet C) + (A' \bullet B \bullet C') + (A \bullet B' \bullet C')$$

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b & ~c) |
               (~a & ~b & c) |
               (~a & b & ~c) |
               (a & ~b & ~c);

endmodule
```

NEW

F = ?

Simplifying Boolean Equation (w/ Behavioral Verilog)

OLD

F = (A' • B' • C') +
(A' • B' • C) +
(A' • B • C') +
(A • B' • C')

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b & ~c) |
               (~a & ~b & c) |
               (~a & b & ~c) |
               (a & ~b & ~c);

endmodule
```

NEW (Distribution)

F = [(A' • B') • (C' + C)] +
(A' • B • C') +
(A • B' • C')

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b & (~c | c)) |
               (~a & b & ~c) |
               (a & ~b & ~c);

endmodule
```

Simplifying Boolean Equation (w/ Behavioral Verilog)

OLD

$$F = [(A' \bullet B') \bullet (C' + C)] + (A' \bullet B \bullet C') + (A \bullet B' \bullet C')$$

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b & (1)) |
               (~a & b & ~c) |
               (a & ~b & ~c);

endmodule
```

NEW (Complements & Null)

$$F = (A' \bullet B') + (A' \bullet B \bullet C') + (A \bullet B' \bullet C')$$

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b) |
               (~a & b & ~c) |
               (a & ~b & ~c);

endmodule
```

Simplifying Boolean Equation (w/ Behavioral Verilog)

OLD

F = (A’ • B’) +
(A’ • B • C’) +
(A • B’ • C’)

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b & (1)) |
               (~a & b & ~c) |
               (a & ~b & ~c);

endmodule
```

NEW (Distribution)

F = [(A’ • (B’ + (B • C’))] +
(A • B’ • C’)

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & (~b | (b & ~c))) |
               (a & ~b & ~c);

endmodule
```

Simplifying Boolean Equation (w/ Behavioral Verilog)

OLD

F = [(A' • (B' + (B • C'))] +
(A • B' • C')

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & (~b | (b & ~c))) |
               (a & ~b & ~c);

endmodule
```

NEW ("No Name" & Distribution)

F = (A' • B') +
(A' • C') +
(A • B' • C')

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b) |
               (~a & ~c) |
               (a & ~b & ~c);

endmodule
```

Simplifying Boolean Equation (w/ Behavioral Verilog)

OLD

F = (A’ • B’) +
(A’ • C’) +
(A • B’ • C’)

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b) |
               (~a & ~c) |
               (a & ~b & ~c);

endmodule
```

NEW (Distribution)

F = (A’ • B’) +
[(C’ • (A’ + (A • B’))]

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b) |
               (~c & (~a | (a & ~b)));

endmodule
```

Simplifying Boolean Equation (w/ Behavioral Verilog)

OLD

$F = (A' \bullet B') + [(C' \bullet (A' + (A \bullet B')))]$

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b) |
               (~c & (~a | (a & ~b)));

endmodule
```

NEW (Distribution)

$F = (A' \bullet B') + (A' \bullet C') + (B' \bullet C')$

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b) |
               (~a & ~c) |
               (~b & ~c);

endmodule
```

FINAL Implementation in **BEHAVIORAL** Verilog

F = (A' • B') + (A' • C') + (B' • C')

```
module Minority(
  input a, b, c,
  output out
);
  assign out = (~a & ~b) | (~a & ~c) | (~b & ~c);

endmodule
```