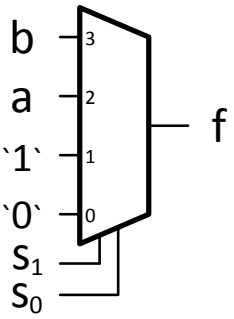
# CE/CZ1005 Digital Logic
# Tutorial 6

Q1 Determine the minimized sum-of-products expression for the 4-1 multiplexer below, and hence sketch the equivalent circuit using just AND and OR gates. Bubble inputs are allowed.




Q2. (a) Draw the circuit represented by the following Verilog module. Label the gates and wires.

```
module whatisit (input a, b, c, d, e,
                 output x, y);
    not  n1  (nb, b);
    not  n2  (ne, e);
    and  a1  (w1, a, b);
    and  a2  (w2, nb, c);
    nor  no1 (w3, d, e);
    nand na1 (w4, w2, w3);
    or   o1  (x, w1, w4);
    and  a3  (y, ne, w1);
endmodule
```

(b) Write down a logic expression for each of the outputs.

Q3. You are required to design a ferry boarding system to direct cars to one of four boarding ramps. The input to the circuit is a 2-bit binary number representing which ramp to use, and a 1-bit signal which is high when all ramps are full. These signals are generated for you (possibly by the human gatekeeper). The circuit has outputs that control four green lights, one above each of the ramps, with only one active at any time. If all ramps are full, no lights should be illuminated indicating that vehicles should not proceed to join any ramp.

(a) Sketch a block diagram for the circuit using a single 2-4 decoder (with enable). A 2-4 decoder takes a 2-bit binary input and produces a one-hot 4-bit output if enable is TRUE.

(b) Write a Verilog module to implement the boarding system that instantiates a predefined decoder module with the following declaration:

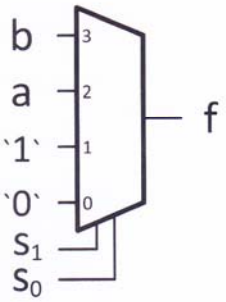
**module** dec2to4 (**input** [1:0] a, **input** enable, **output** [3:0] one-hot);

Q4. Rewrite the Verilog module of Q2 using a single assign statement for each output.

## CE/CZ1005 Digital Logic Tutorial 6: Combinational Logic Answers

Q1. Determine the minimized sum-of-products expression for the 4-1 multiplexer below, and hence sketch the equivalent circuit using just AND and OR gates. Bubble inputs are allowed.

Firstly draw a truth table to describe the function of the multiplexer. Note there are 6 inputs however there is no need to show a 64 entry truth table. Instead, simplify as follows:




| $S_1$ | $S_0$ | b | a | f | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | $f = `0'$ |
| | | 0 | 1 | 0 | |
| | | 1 | 0 | 0 | |
| | | 1 | 1 | 0 | |
| 0 | 1 | 0 | 0 | 1 | $f = `1'$ |
| | | 0 | 1 | 1 | |
| | | 1 | 0 | 1 | |
| | | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | 0 | $f = a$ |
| | | 0 | 1 | 1 | |
| | | 1 | 0 | 0 | |
| | | 1 | 1 | 1 | |
| 1 | 1 | 0 | 0 | 0 | $f = b$ |
| | | 0 | 1 | 0 | |
| | | 1 | 0 | 1 | |
| | | 1 | 1 | 1 | |

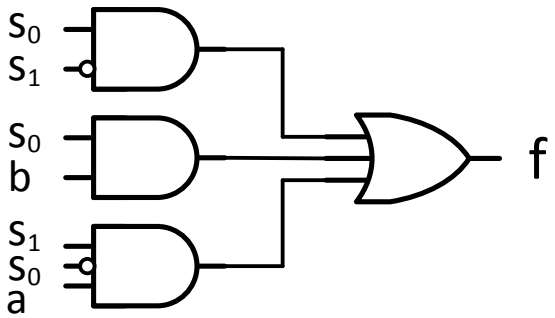
Then it is possible to determine the k-map as:

| $S_1 S_0$ \ ba | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 1 | 1 | 0 |

And hence the expression for f is:

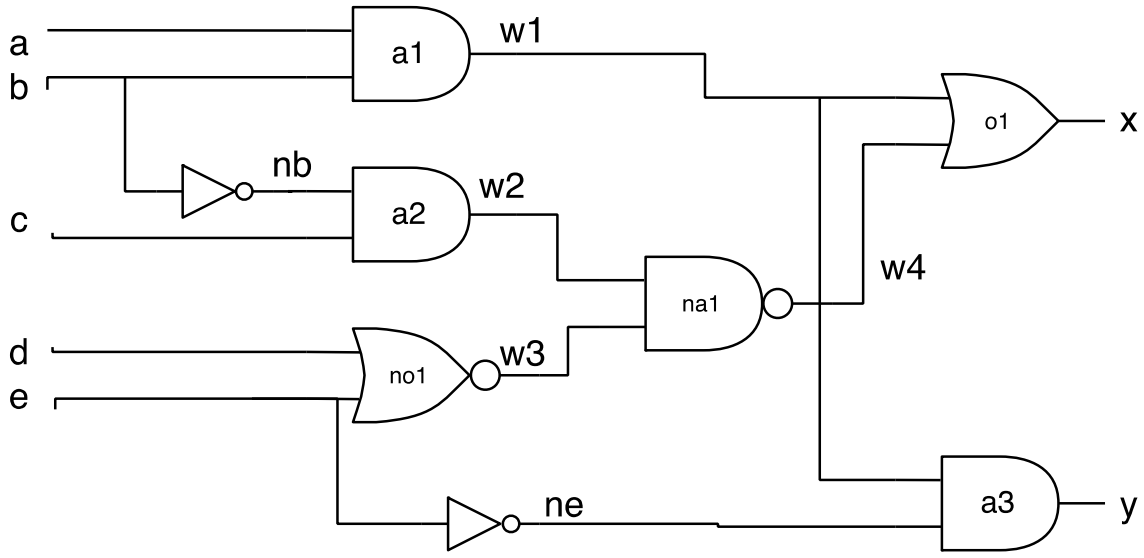
$$f = s_1' \, s_0 + s_0 \, b + s_1 \, s_0' \, a$$

Thus the gate representation is:



Q2. (a) Draw the circuit represented by the following Verilog module. Label the gates and wires.

```
module whatisit (input a, b, c, d, e,
                 output x, y);
    not  n1  (nb, b);
    not  n2  (ne, e);
    and  a1  (w1, a, b);
    and  a2  (w2, nb, c);
    nor  no1 (w3, d, e);
    nand na1 (w4, w2, w3);
    or   o1  (x, w1, w4);
    and  a3  (y, ne, w1);
endmodule
```




Note: since the wires are 1-bit, they don't need to be explicitly declared in the Verilog.

(b) Write down a logic expression for each of the outputs.

x = ab + ((b'c) (d+e)')'

y = abe'

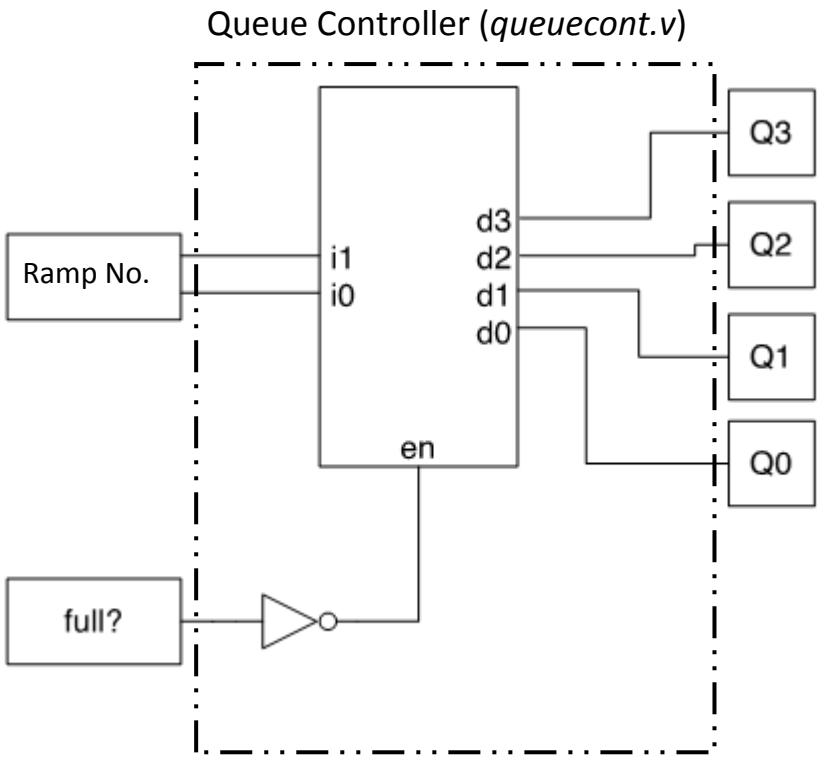
Q4. Rewrite the Verilog module of Q2 using an assign statement for each output.

```
module whatisit2 (input a,b,c,d,e, output x, y);
   assign x = (a & b) | ~((~b & c) & ~(d | e));
   assign y = a & b & ~e;

endmodule
```

Note: The &, |, and ~ operators are bitwise, whereas &&, ||, ! are logical.
They are interchangeable for 1-bit signals.
Also note the operator precedence (~ & | ^ ~^)

Q3.




We want to enable the system when the queue is not full, so we need to negate the full signal. When full, all lights will be OFF.

(b) Write a Verilog module that instantiates a decoder module with the following declaration:

**module** dec2to4 (**input** [1:0] a, **input** enable, **output** [3:0] one-hot);

```
module queuecont (input [1:0] ramp_num,
                  input       full,
                  output [3:0] queue);

  dec2to4 u1 (.a(ramp_num), .enable(~full), .one-hot(queue));

endmodule
```

Note the use of the negated *full* signal in the port connection.