Homework 1 Solutions

1 Combinational Logic Review

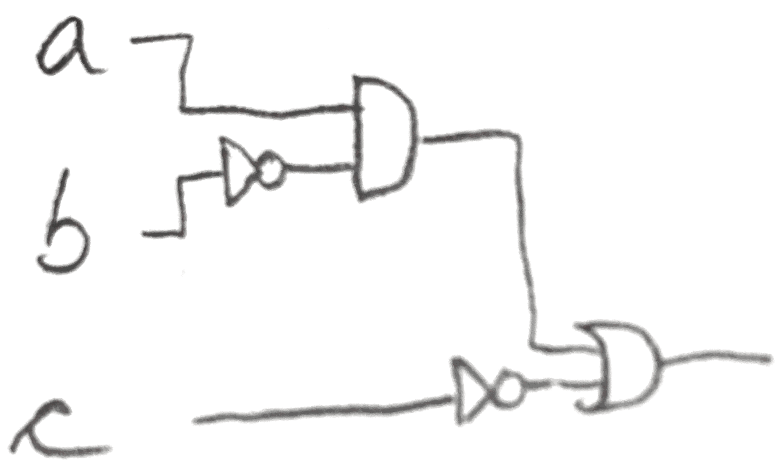
1a)

a	b	c	b'c'	bc'	ab'c	b'c' + bc' + ab'c
0	0	0	1	0	0	1
0	0	1	0	0	0	0
0	1	0	0	1	0	1
0	1	1	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	1	1
1	1	0	0	1	0	1
1	1	1	0	0	0	0

1b)

`c′ + ab′`

1c)

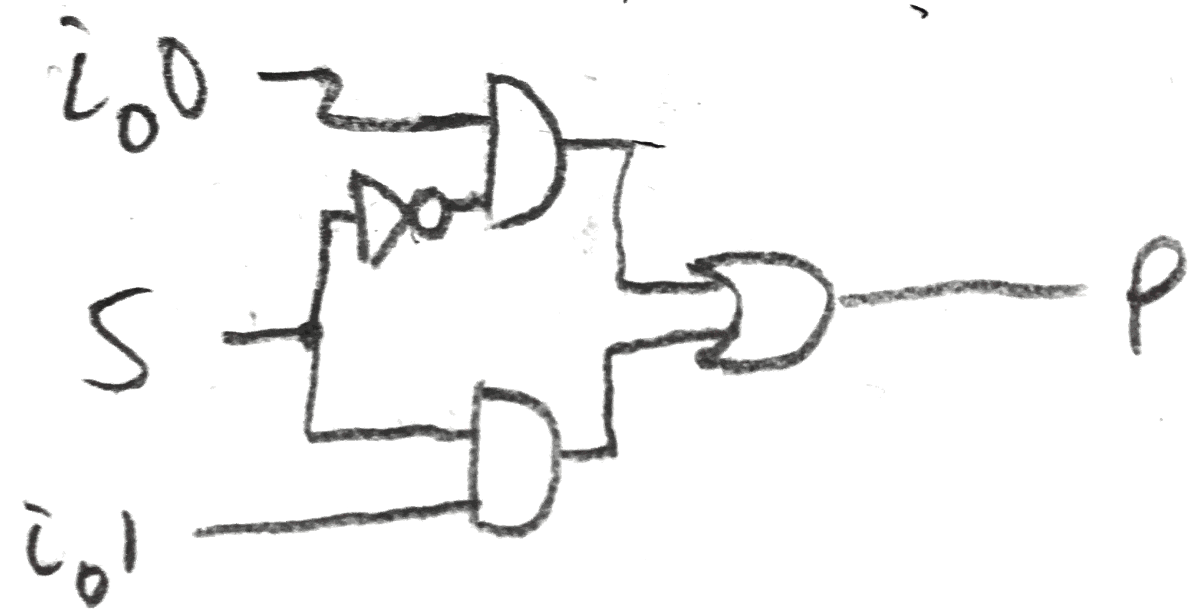


2a)

Io0	**io1**	**s**	**p**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

2b)

io0*s’ + io1*s



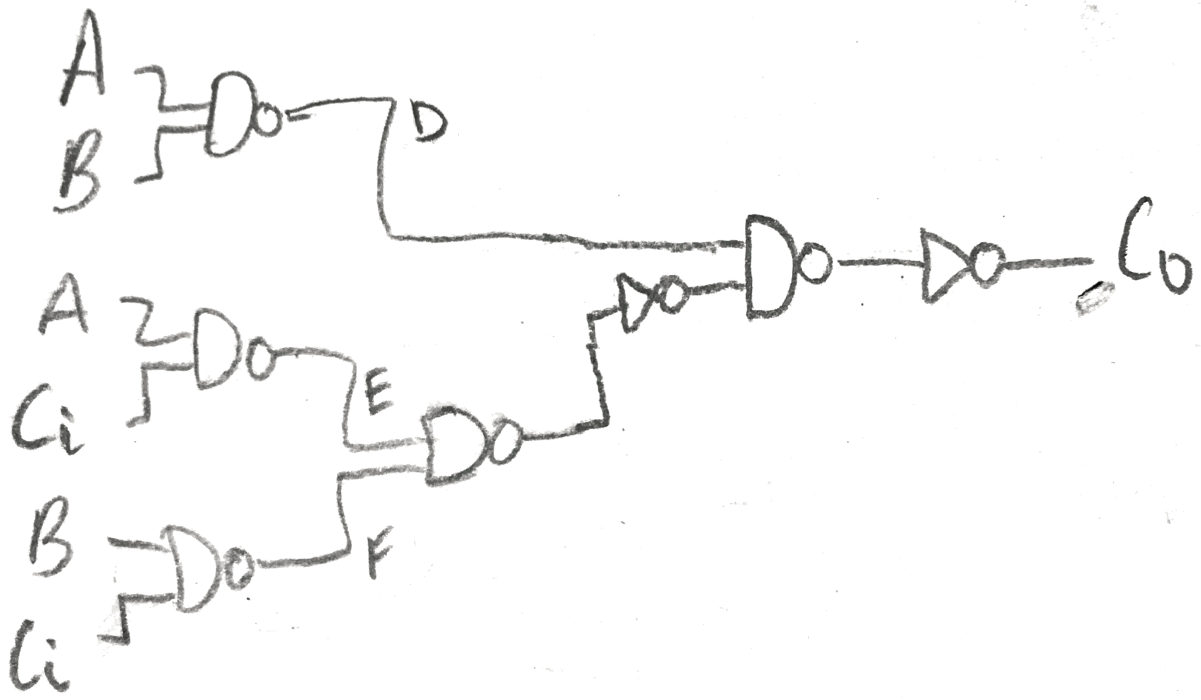
$i_0 0$
S
$i_0 1$
P

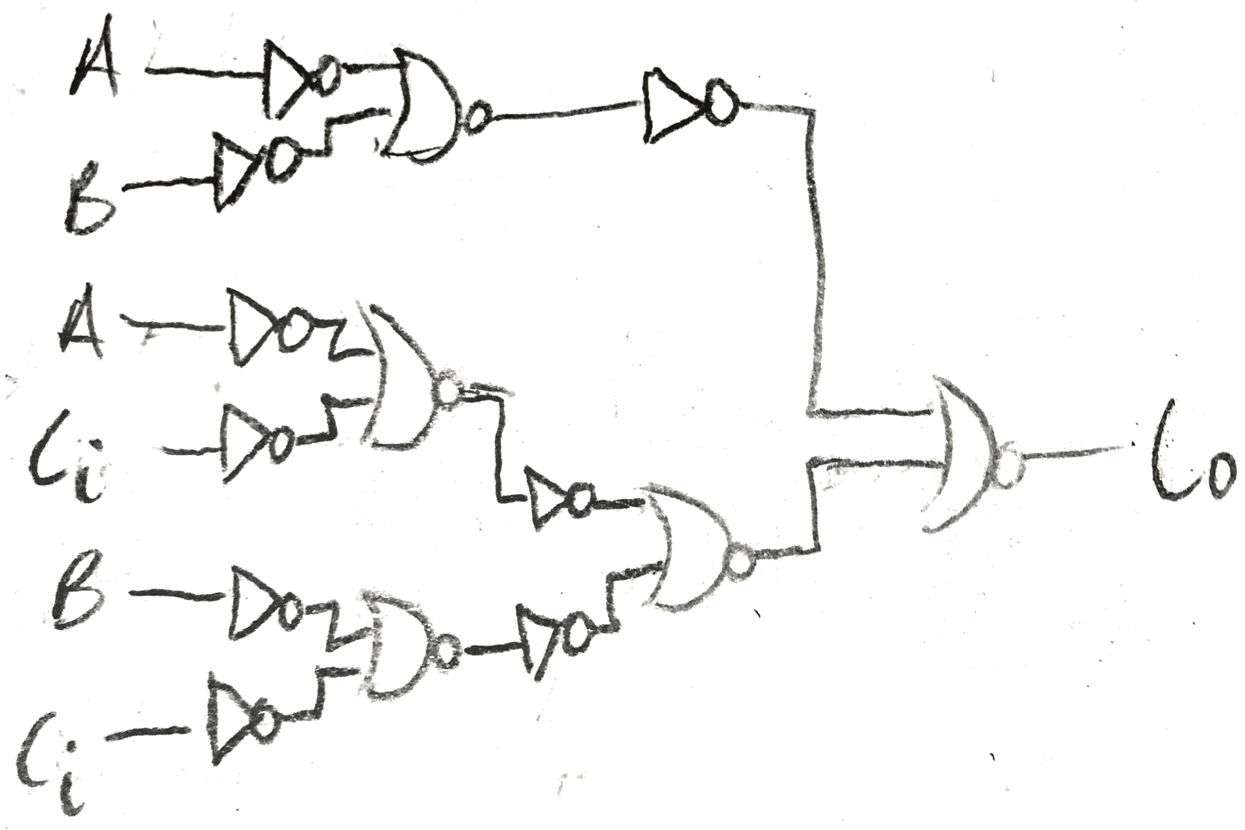
3a)

a	b	ci	co	s	co'
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	1	0

3b)

With generating s, and generating co with all NANDs (I think this is more optimized, by a bit):





Generating co with all NORs:

2 Verilog for Combinational Logic

4a)

To determine proper parentheses use, see http://www.ee.ed.ac.uk/~gerard/Teach/Verilog/manual/Appendices/op_prec.html for the order of operations.

You could also use logical operators (&&, ||, !) instead of bitwise (&, |, ~). Brehob prefers using bitwise for logic like this. Using them interchangeably is inappropriate (but would still work in this instance).

```
assign p = (~a)&b | ~(a&(~c)) | (b^c);
```

4b)

```
assign p = (s & io1) | (~s & io0);
```

5a)

```
module MUX21 (io0, io1, s, p);
```

```
input io0, io1, s;
output P;
wire P;

assign p = (s == 1)? io1: io0;
// alternatively
// assign p = (s & io1) | (~s & io0);

endmodule // end of MUX21 module
```

5b)

```
module MUX41 (io0, io1, io2, io3, s0, s1, p);

input io0, io1, io2, io3, s0, s1;
output p;
wire w0, w1;

MUX21(io0, io1, s0, w0);
MUX21(io2, io3, s0, w1);
MUX21(w0, w1, s1, p);

endmodule // end of MUX41 module
```

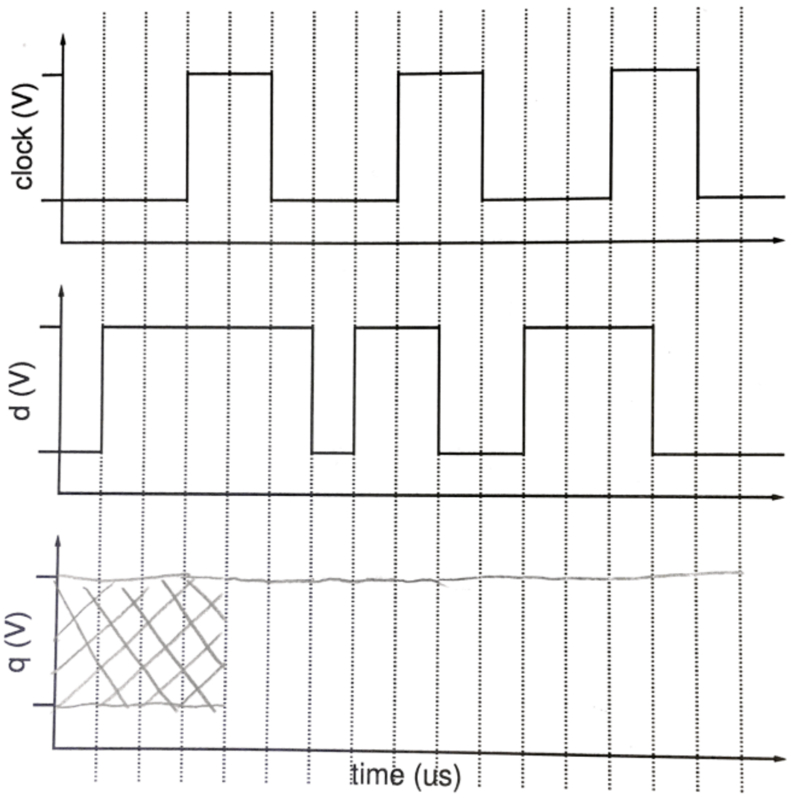
6a)

```
module MUX21 (io, io1, s, p);

input io0, io1, s;
output reg p;

always @*
begin
     p = (s == 1)? io1 : io0;
end

endmodule // end of MUX21 module
```

6b)

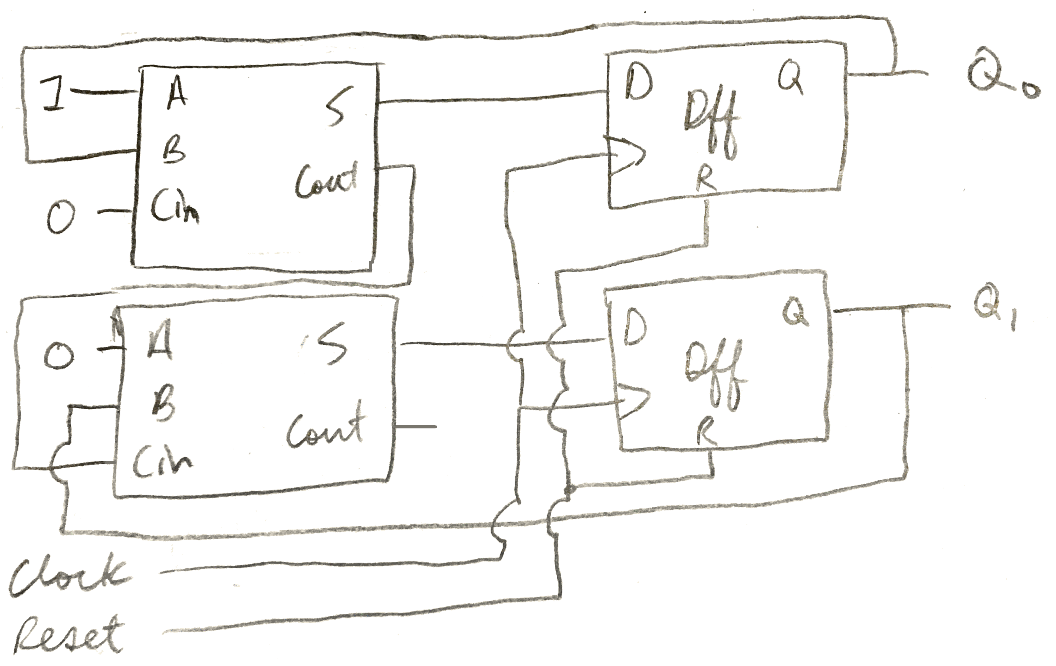
```
module MUX41 (io0, io1, io2, io3, s0, s1, p);

input io0, io1, io2, io3, s0, s1;
wire w0, r1;
// also acceptable: reg r0, r1;
output p;

MUX21(io0, io1, s0, w0);
```

Figure 1: D flip-flop.



8)

The reset part here caused people a lot more grief than was intended. A number of you came



up with really cool ways of handling the reset signal. The intent was to just use a flip-flop with a reset. A number of you came up with some very impressive and creative solutions while using only full-adders and flip-flops without a reset.

4 Verilog for Sequential Logic

9a)

3 bits used - the minimum would be 2.

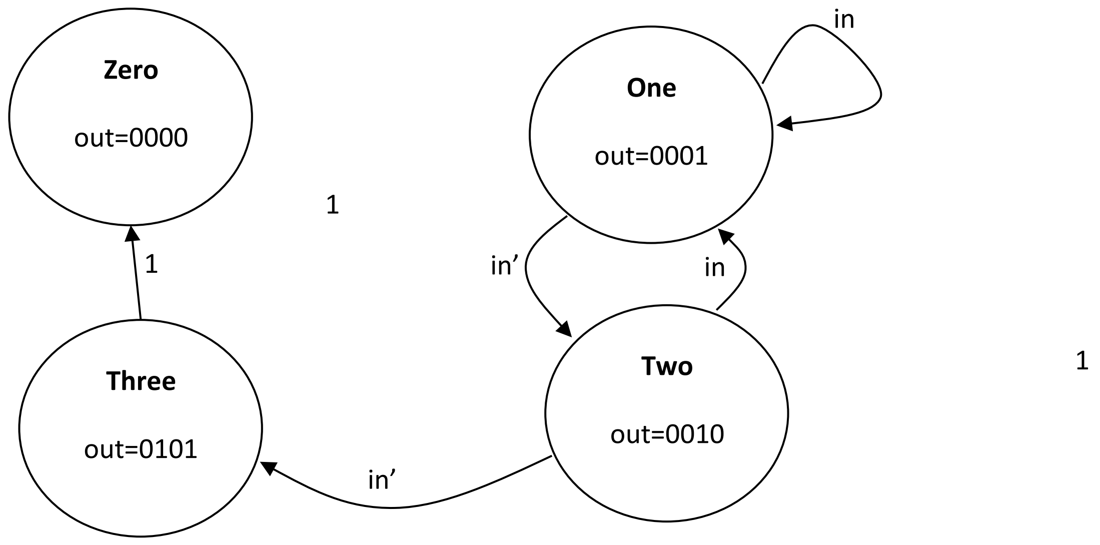
9b)

Only two state variables are required (`state`, and `next_state`). Each one needs to be two or 3 bits wide.

9c)

The assign statement version is a lot shorter, but harder to get and much harder to manipulate if the state machine needs minor changes. On the whole, the case-statement version is strongly preferred - less error-prone, easier to debug and modify.

10)



11a and b)

A tri-state driver is a driver that is capable of putting three different states onto a wire. There are three different states possible: “1” (high), “0” (low”), or “HiZ” (high impedance).

11c)

The Hi-Z state is where the device isn’t actually driving a signal onto the wire. It is, in effect, electrically disconnected from the wire.

11d)

For a device to send a 1 on the bus, its driver would need to be in the “1” (high) state.

11e)

The drivers of the other devices would need to be in the HiZ state.

11f)

If a device drove a wire with a 1 and another drove it with a 0 simultaneously, then a very-low resistance path between power and ground - a short - would be formed. This would cause high current, power, and heat. It’s hard to say what specifically would happen, but you could fry some transistors, or the wire - or it may just get hot.

11g)

If the input to an inverter is Hi-Z, then the input wire is essentially being left to float. Therefore, the output is unknown.

12a)

An open-collector bus design is one where a device's output either connects to ground or it connects to nothing. If any device on the bus connects to ground, the wire is at ground. But if none of them do, the wire will float (HiZ).

12b)

The pull-up resistor prevents the wire from floating if no device is connected to grand. Now, the default state is the wire being at a "1", through the pull-up resistor.

12c)

To send a 0, a device output would be in a "0" state (and everyone else should drive nothing, although it doesn't matter what they drive - they can only drive "0", which would accomplish the same objective). That would pull the wire to "0".

12d)

To send a 1, a device output would be in the "HiZ" state as well as all of the other devices). This would allow the pull-up resistor to bring the bus up to a "1".

12e)

When the device output is not attempting to drive the bus, then it would be in "HiZ" to allow other devices to choose to drive the bus to "0".