Digital Circuit Design

Li Bai

Theorem of Boolean Algebra

- Idempotence laws:

$$x + x = x \tag{2.11}$$

$$x \cdot x = x \tag{2.12}$$

- Further identity laws:

$$x + 1 = 1 \tag{2.13}$$

$$x \cdot 0 = 0 \tag{2.14}$$

- Absorption laws:

$$x + (x \cdot y) = x \tag{2.15}$$

$$x \cdot (x + y) = x \tag{2.16}$$

- DeMorgan laws:

$$\overline{(x + y)} = \overline{x} \cdot \overline{y} \tag{2.17}$$

$$\overline{(x \cdot y)} = \overline{x} + \overline{y} \tag{2.18}$$

Equivalents

EXERCISE 2.8 Show, using truth tables, that the two Boolean expressions in each of the following pairs are equivalent:

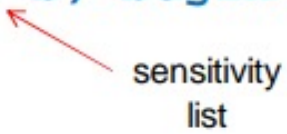
a) $x \cdot \overline{(y \cdot z)}$ and $x \cdot \overline{y} + x \cdot \overline{z}$

b) $\overline{x \oplus y}$ and $\overline{\overline{x} \oplus \overline{y}}$

Verilog `always` Procedural Block

- Example (block of combinational logic):

```
always @* begin          … old  always @(a, b) begin
   … verilog logic statements
end
```



  - a concurrent or parallel block inside a module
  - for multiple always blocks in the same module, these are parallel blocks – ***independent of their order in the module***. (Just like assign statements)
  - maps very nicely into describing hardware
  - inside the block, the sequence of the statements can have significance (to be covered later)

Verilog `always` Procedural Block – Combinational Logic

- Examples (within a single module)

```
…
assign f = a & b;
assign g = a | c;
…
```

```
…
always @* begin
   f = a & b;
end

always @(a, c) begin
   g = a | c;
end
…
```

All of these implementations are equivalent!!

```
…
always @* begin
   f = a & b;
   g = a | c;
end
…
```

Verilog `if-else` Construct

- Again, very similar to C code. Verilog example:

```
if (a == b) begin
  f = 1;
end else begin
  f = 0;
end
```

- For single statements:

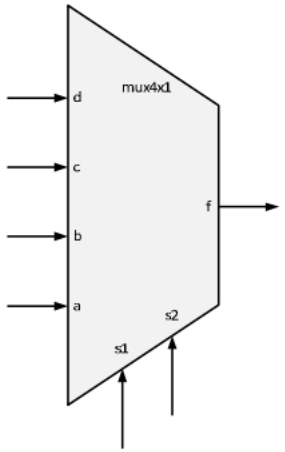
```
if (a == b) f = 1;
else f = 0;
```

- Multiple case items
  - If you have an identical statement (or statement block) for multiple items, those items can be grouped as a single case item.
- Syntax:

```
case(<argument>)
    <case1>: <statementA>
    <case2>: <statementB>
    <case3>, <case4>, <case5>: <statementC>
…
    default: <statementZ>
endcase
```

Example: 4x1 Mux Design

- 4 Inputs, 1 Output, (and 2 Select inputs)



d	c	b	a	s2	s1	f
x	x	x	0	0	0	0
x	x	x	1	0	0	1
x	x	0	x	0	1	0
x	x	1	x	0	1	1
x	0	x	x	1	0	0
x	1	x	x	1	0	1
0	x	x	x	1	1	0
1	x	x	x	1	1	1

Example: 4x1 Mux Design – with `case`

```
…
    always @* begin
        case ({s2, s1})
          2'b00: f = a;
          2'b01: f = b;
          2'b10: f = c;
          default: f = d;
        endcase
    end
…
```

Notes:

the concatenation operator { , } is used;

the default case covers the condition 2'b11.

```
module first(output d,
    output reg e, input a, b, c);

    assign d = a & b;

    always @* begin
        e = b | c;
    end

endmodule
```

```
module second(output reg d, e,
    input a, b, c);

    d = 1'b1;
    if (a == b) begin
        d = 0;
    end
    e = a ^ c;

endmodule
```

```
module third(output reg d, e,
    input a, b, c);

    always @* begin
        case ({b, c})
            2'b10: assign d = a;
            default: assign e = b | c;
        endcase
    end

endmodule
```

```
module fourth(output wire d, e,
    input a, b, c);

    assign a = b & c;

    assign d = e | ~c;

endmodule
```

```
module fifth(output reg d, e,
    input a, b, c);

    assign e = a ^ b ^ c;

    always @* begin
        assign d = b & a | c;
    end

endmodule
```

```
module sixth(output reg d, e,
    input a, b, c);

    always @* begin
        d = a;
        e = c;
        if (b == 0) e = 1'b1;
        if (c == 1) d = 0;
    end

endmodule
```

Encoding and decoding

How to represent information into digital data (encoding)

BCD encoding

0: 0000	1: 0001	2: 0010	3: 0011	4: 0100
5: 0101	6: 0110	7: 0111	8: 1000	9: 1001

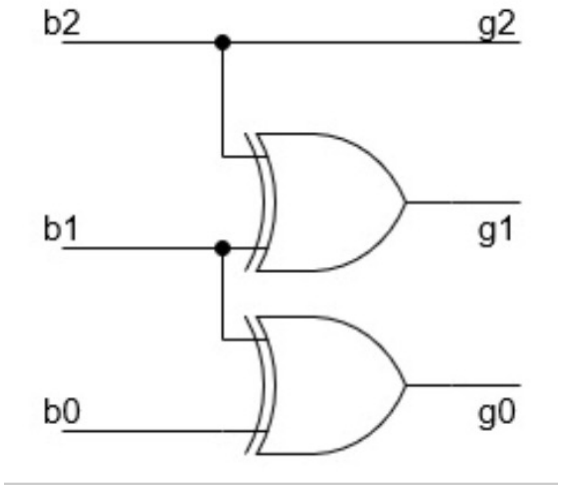
$$\lceil \log_2(N) \rceil = \left\lceil \frac{\ln(10)}{\ln(2)} \right\rceil \approx 3.32$$

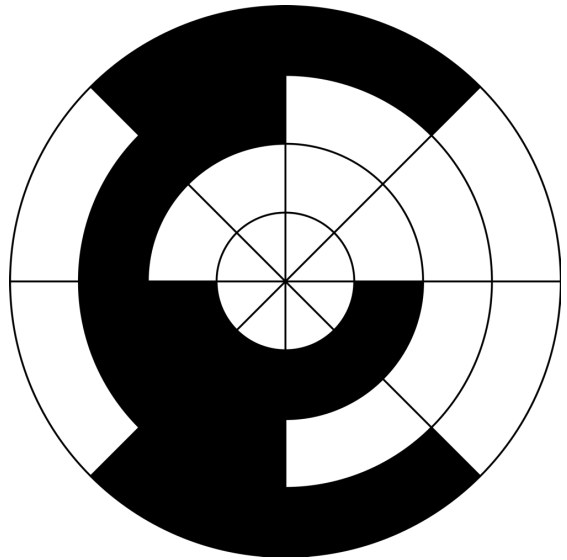
Gray encoding

0: 0000 1: 0001 2: 0010 3: 0011 4: 0100
5: 0101 6: 0110 7: 0111 8: 1000 9: 1001

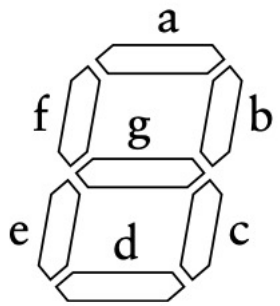
Can I create an encoding with only one bit transition from the near by numbers

b2	b1	b0	g2	g1	g0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0





7 segment display



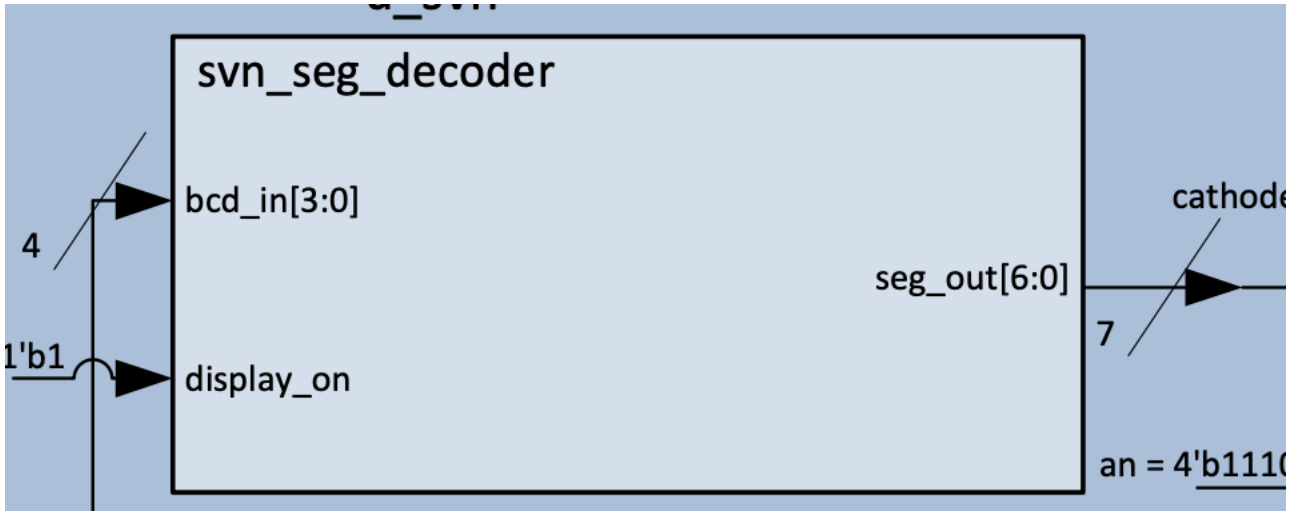
- gfedcba
- active high vs active low
- For the labs, we used the active low 7 segment display

0111111 0000110 1011011 1001111 1100110

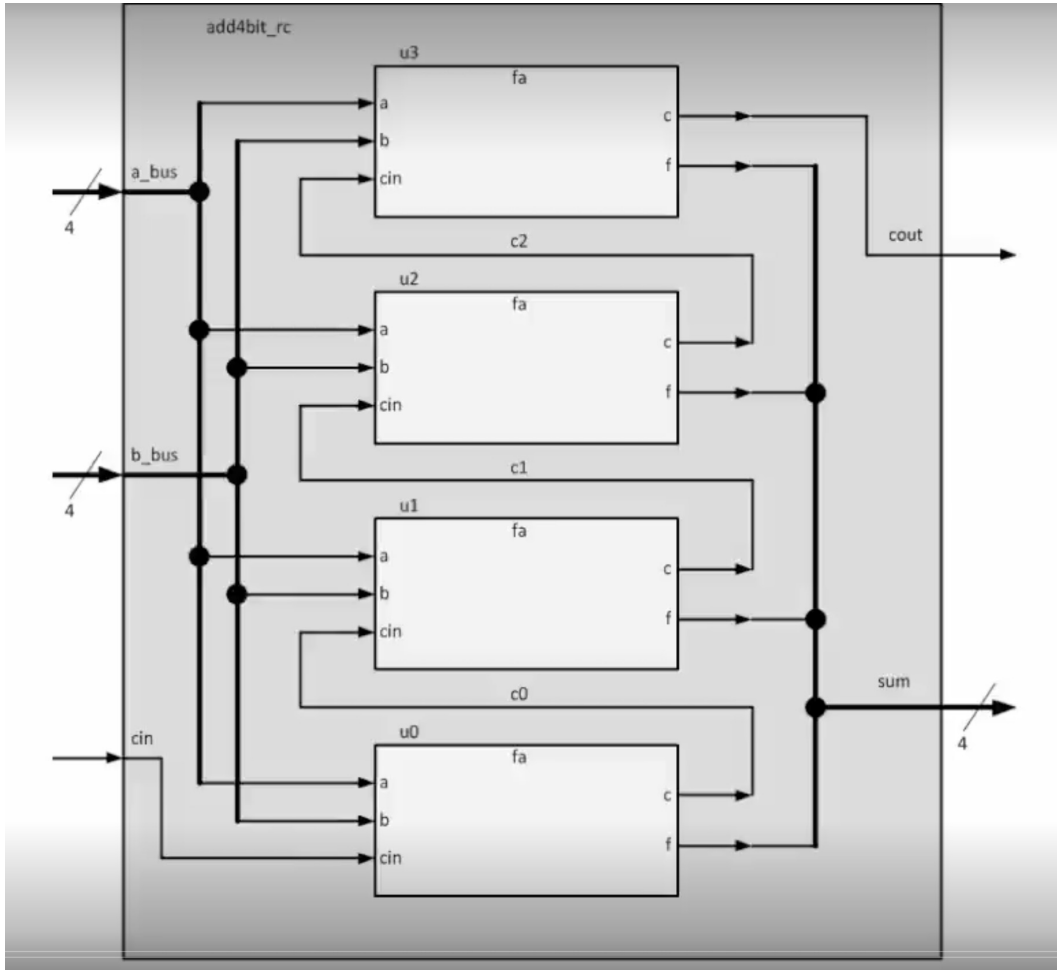
1101101 1111101 0000111 1111111 1101111

Provide a Verilog code for BCD to 7-seg

- Convert the input BCD to a 7-seg encoding scheme (active low)

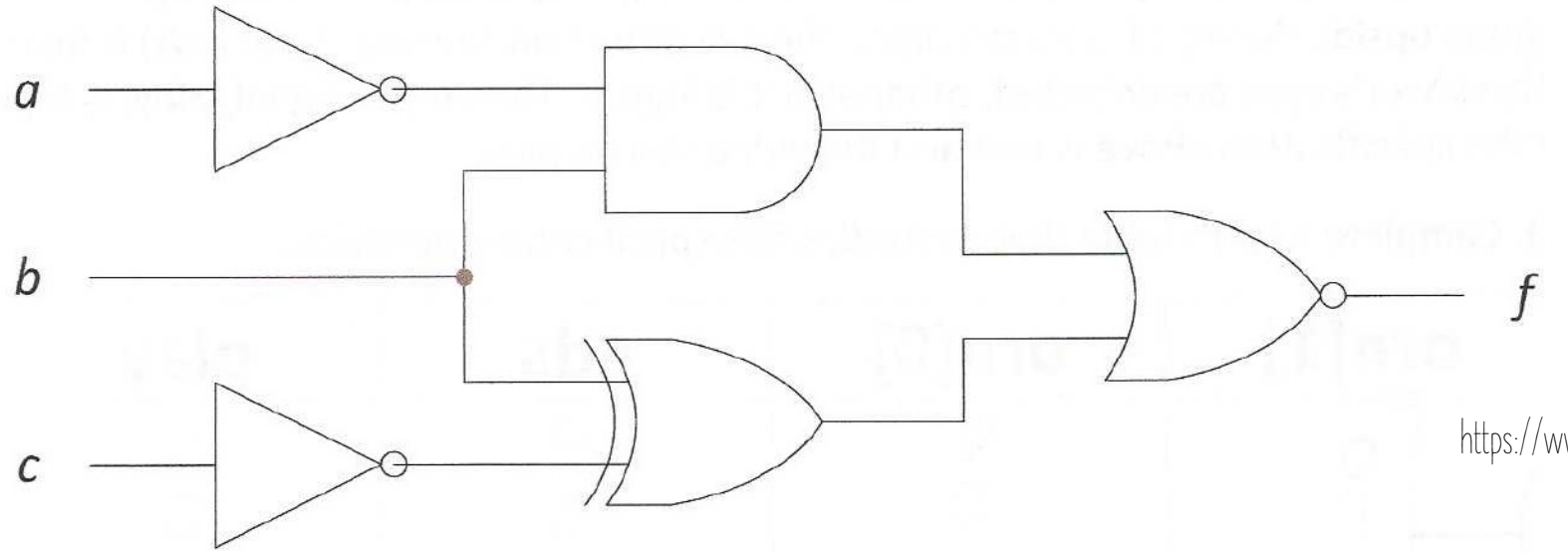


verilog excercise



Example problem 4

Given the schematic below (the new gate with the double curved input is an *exclusive or* gate):



https://www.edaplayground.com/x/xr8

- Based on the AND, OR, XOR, … logic, write a *single* Verilog **assign** statement for the schematic using &, |, ^, …
- Create a truth table for an *exclusive or* gate, then for the full schematic.
- Depending on instructions and templates provided by the instructor, use a skeleton module and matching testbench and enter your assign statement. Run the simulation to test for mismatches.
- From the truth table for the full schematic, create an equivalent SystemVerilog design using *Sum of Products* (SOP) technique described in the videos. Comment out your assign statement from a) and enter your SOP design.
- Sketch a schematic of the *Sum of Products* implementation using AND and OR gates. It is easier to use bubble notation for the invert operation.

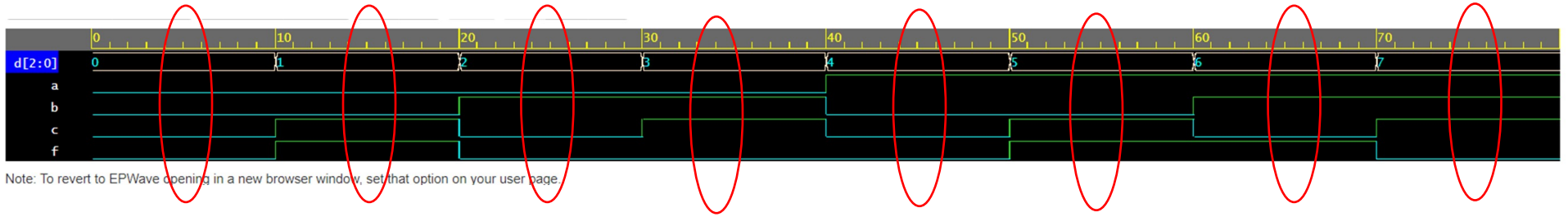
3. Verilog module and testbench

Log | Share

```
[2020-09-08 22:07:22 EDT] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile dump.vcd opened for output.
Testing input pattern: 000
Testing input pattern: 001
Testing input pattern: 010
Testing input pattern: 011
Testing input pattern: 100
Testing input pattern: 101
Testing input pattern: 110
Testing input pattern: 111
Simulation complete!
Done
```

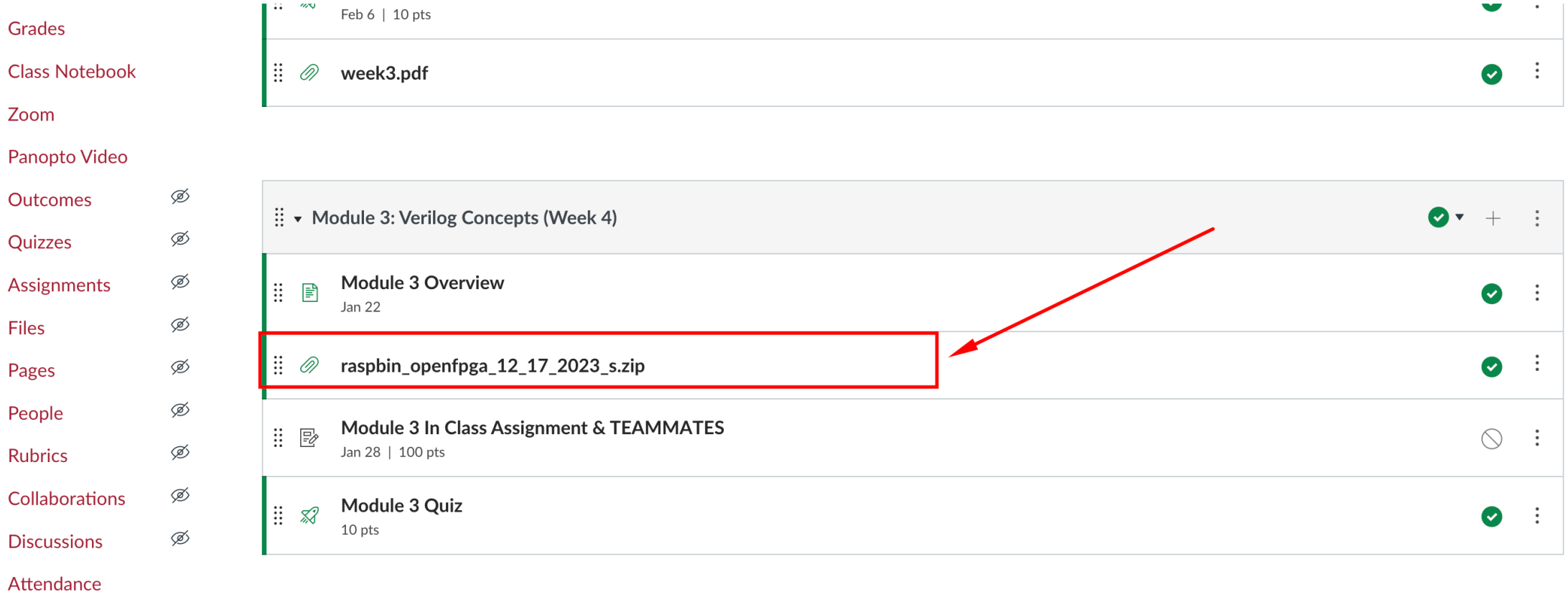
https://embed.edaplayground.com/x/6JKp

abc_f

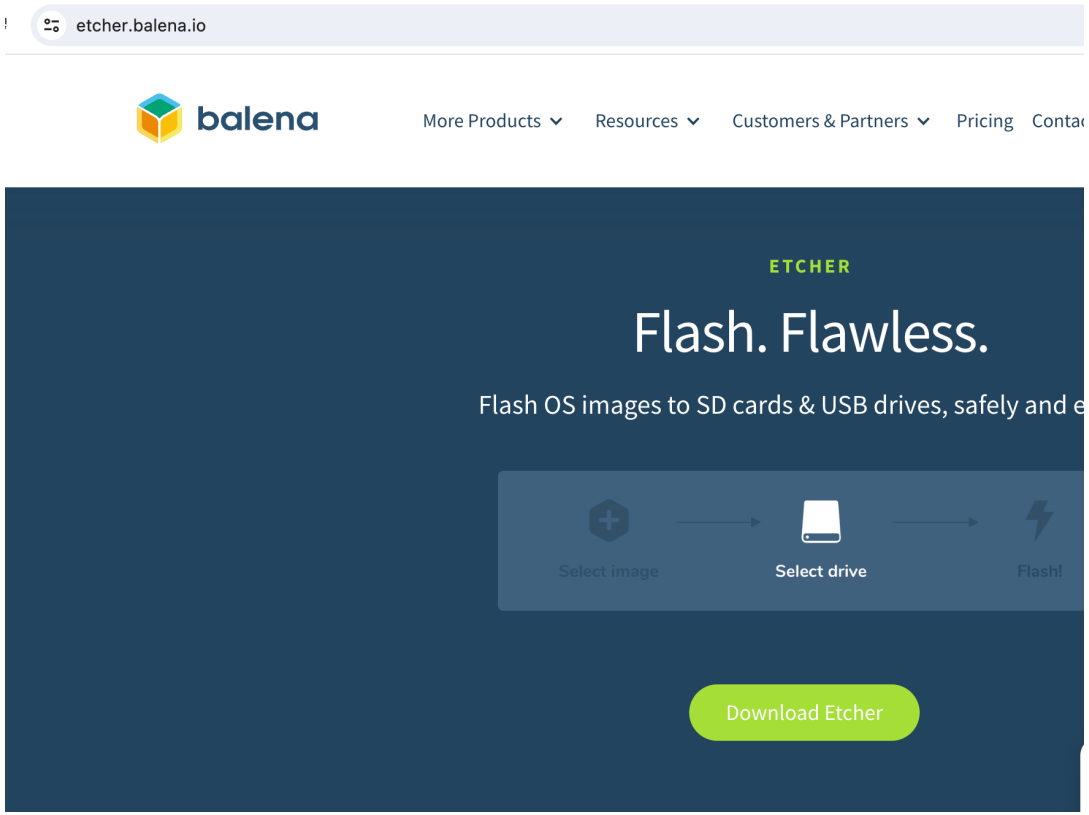


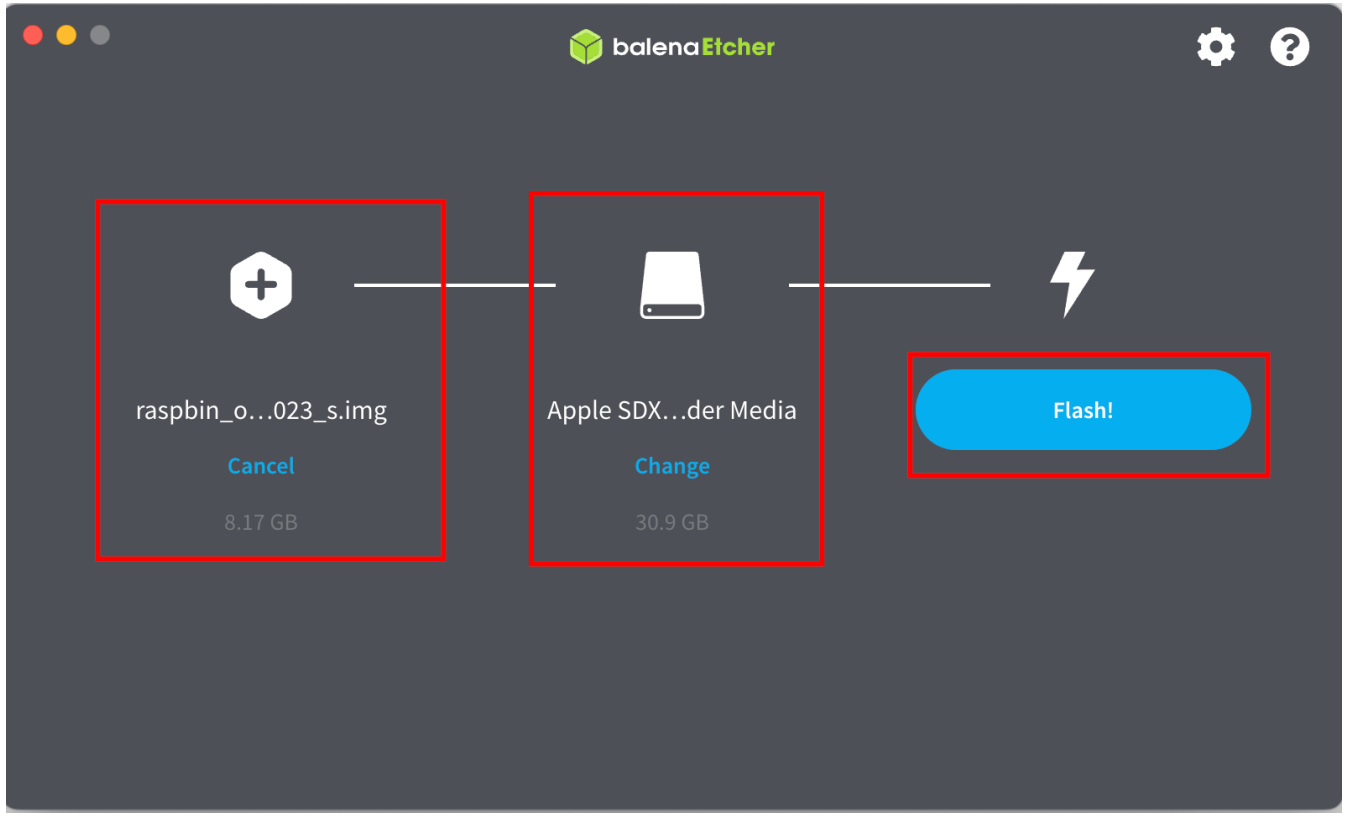
000_0 001_1 010_0 011_0 100_0 101_1 110_1 111_0

Download the image file from canvas



Flash card using etcher





Wireless setup

Get a password hash using AWS cloud

git clone -b spring24 http://github.com/lbaitemple/ece2613 wireless

cd wireless

bash ./rundocker.sh

sudo reboot

```
ubuntu@ip-172-31-63-189:~/environment$ cd wireless/
ubuntu@ip-172-31-63-189:~/environment/wireless$ bash ./rundocker.sh
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker-compose is already the newest version (1.29.2-1).
0 upgraded, 0 newly installed, 0 to remove and 23 not upgraded.
cmdshell uses an image, skipping
Building my_image
[+] Building 0.1s (6/6) FINISHED                                    docker:default
 => [internal] load build definition from Dockerfile                          0.0s
```

Create image

docker image list

docker run -it bionic-bai:latest /bin/bash

```
ubuntu@ip-172-31-63-189:~/environment/wireless$ docker image list
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
bionic-bai   latest    503368ad021f   2 days ago   145MB
ubuntu@ip-172-31-63-189:~/environment/wireless$ docker run -it bionic-bai:latest /bin/bash
root@c902a7fa2ea9:/#
```

echo -n "password" | iconv -t UTF-16LE | openssl md4

Add password hash

echo -n "password" | iconv -t UTF-16LE | openssl md4