

Synchronous Design

Lecture 04

Josh Brake
Harvey Mudd College

Outline

- Review of basic synchronous design
- Review of the dynamic discipline
- FSM Design Steps
- FSM Activity: Level to pulse converter
- Diode and transistor review

Learning Objectives

By the end of this lecture you should be able to...

- Recall the dynamic discipline and timing specs for designing synchronous digital systems.
- Properly condition asynchronous signals using synchronizers.
- Recall how to use transistors to drive large currents.

Synchronous Digital Systems

- Timing problems are usually the #1 source of difficult bugs
- We can almost completely eliminate the timing problems with a synchronous discipline
 - Like digital vs. analog: digital is a subset of analog
 - Synchronous is a subset of asynchronous timing methodologies
 - Limiting choice makes design easier to understand and avoid sneaky bugs
- Also, will make testability easier (we'll see that later)

Basic Synchronous Design Rules

- Use only one **clock signal** (named something clear like `clk`)
- Use only **flip-flops** as state elements (no latches!)
- Put this clock signal into the clock terminal of every flip-flop in the system.

Some common gotchas

Q: How do we begin in a known state?

A: Use a reset signal.

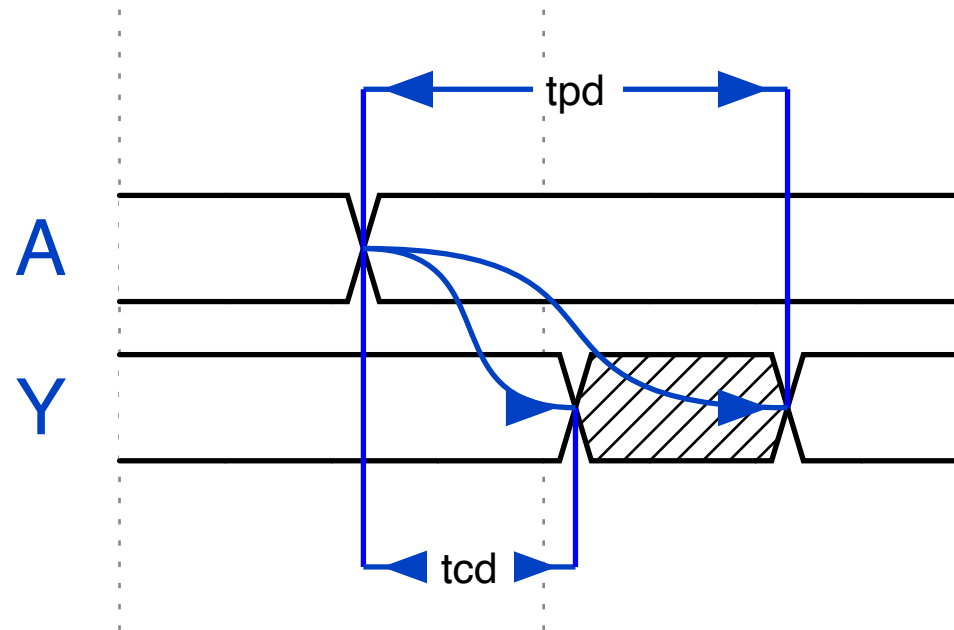
Q: How do we avoid changing the contents of a flip-flop on every clock cycle?

A: Use an enable.

Dynamic Discipline Review

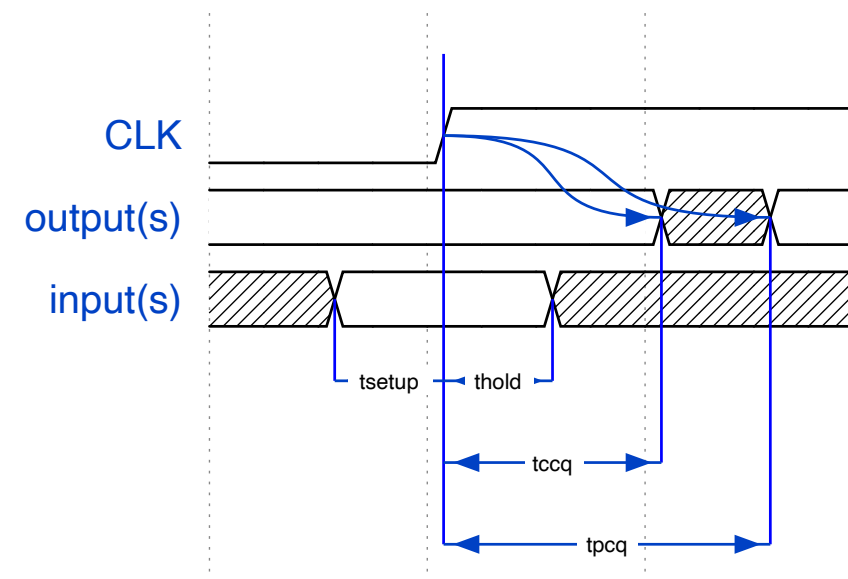
Propagation delay (t_{pd}) – the **maximum** time from when an input changes until the output(s) reach their final value.

Contamination delay (t_{cd}) – the **minimum** time from when an input changes until any output starts to change its value.



Dynamic Discipline Review: Sequential Logic

- **Propagation Clock-to-Q** (t_{pcq}) – **upper** bound on the time from the rising edge of the clock until the output changes.
- **Contamination Clock-to-Q** (t_{ccq}) – **lower** bound on the time from the rising edge of the clock until the output changes.
- **Setup time** (t_{setup}) – the amount of time an input to a flop must be stable **before** the clock edge.
- **Hold time** (t_{hold}) – the amount of time an input to a flop must be stable **after** the clock edge.



Synchronous Timing Constraints

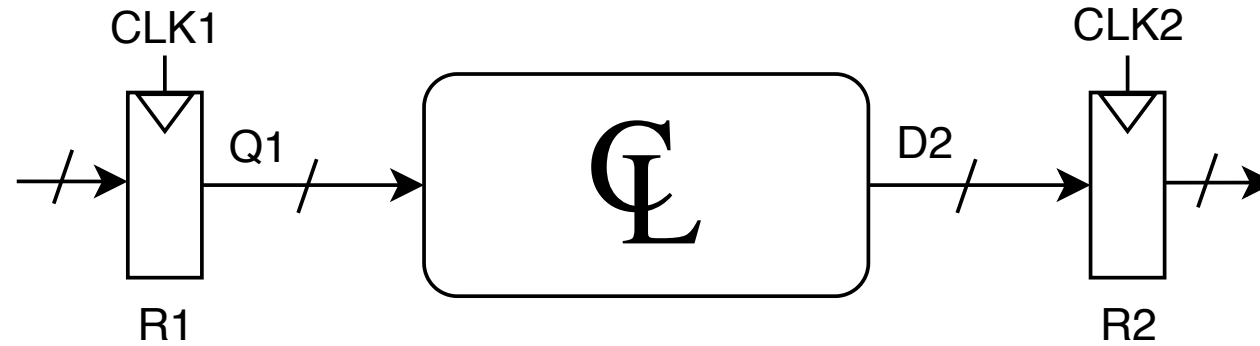
Setup Time Constraint

$$t_{pcq} + t_{pd} + t_{skew} + t_{setup} \leq T_c$$

Hold Time Constraint

$$t_{ccq} + t_{cd} \geq t_{hold} + t_{skew}$$

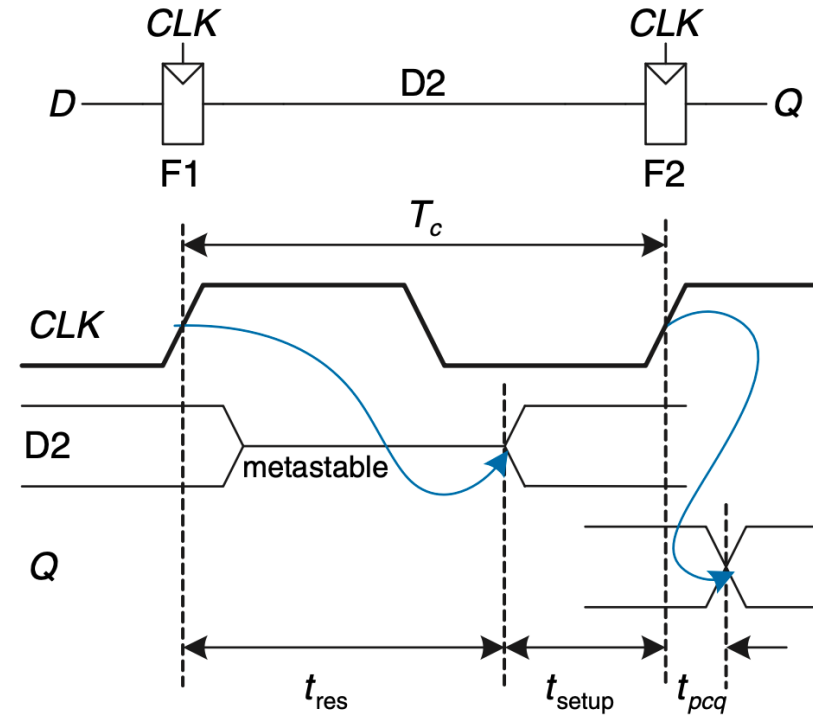
Understanding Timing Constraints



Synchronizers

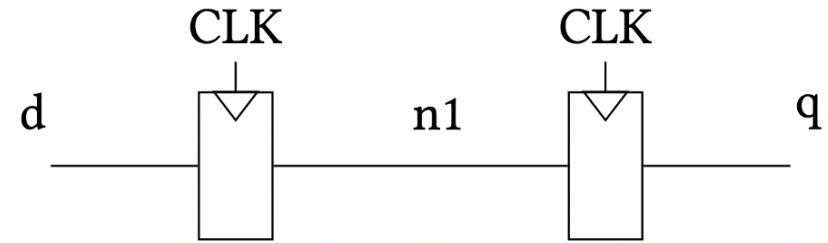
Synchronizer

- The world is asynchronous – how can we cope? Synchronizers!
- Simplest case is a 2-stage synchronizer made of two flops in series.
- If the output of flop F1 goes metastable, we have some time for it to resolve before the next clock edge and the second flop F2.
- This avoids passing metastable inputs out to combinational logic.



Synchronizer

```
1 module sync(input logic clk,  
2             input logic d,  
3             output logic q);  
4  
5     logic n1;  
6  
7     always_ff @(posedge clk)  
8     begin  
9         n1 <= d;  
10        q <= n1;  
11    end  
12 endmodule
```



Another Synchronizer

What if I replace the non-blocking assignments with blocking assignments?

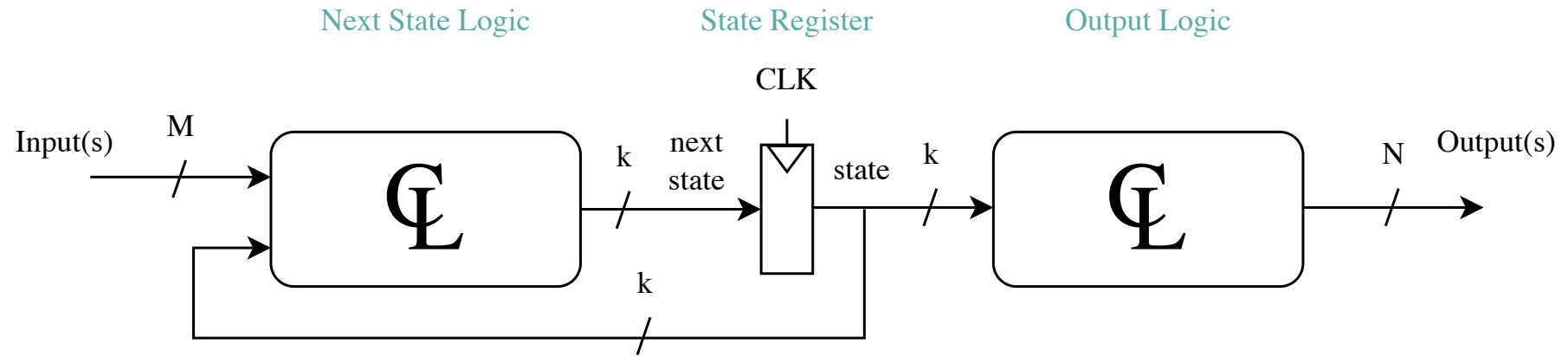
What logic does this imply?

No, this implies only a single flop. The synthesizer will see that `n1` gets `d` and then `q` gets `n1` before the block exits and just eliminate the unnecessary intermediate signal.

```
1 module sync(input logic clk,  
2             input logic d,  
3             output logic q);  
4  
5     logic n1;  
6  
7     always_ff @(posedge clk)  
8     begin  
9         n1 = d;  
10        q = n1;  
11    end  
12 endmodule
```

Finite State Machine (FSM) Review

FSM Design



FSM Design Process

1. List System Parameters (inputs & outputs)
2. Sketch State Transition Diagram
3. Write State Transition Table
4. Choose Encoding Scheme
5. Write Boolean Next State Equations
6. Write Output Table
7. Implement Logic

FSM Activity

FSM Project: Strobe Signal Generator (Level-to-pulse convertor)

You have been tasked with creating circuitry for a single photon detector. When a photon arrives, it generates a pulse of a random length. We want to convert this random-length pulse to an output pulse of a fixed duration whenever a photon hits the detector.

Your task (should you choose to accept it): Design a system which generates a pulse for a single clock cycle when the input goes from low to high. Add a synchronizer to reduce the probability that the input will cause metastability.

List Out Specifications

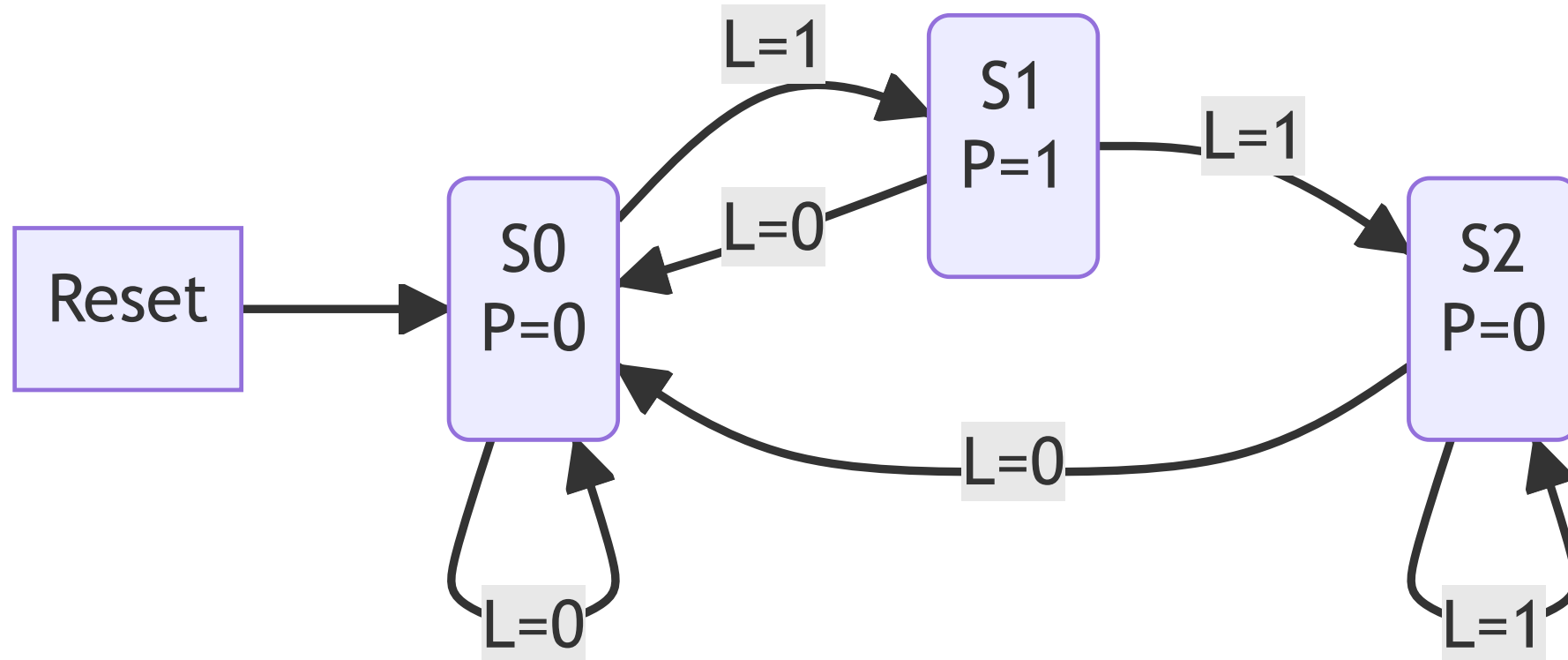
Inputs

- Photon signal: L

Outputs

- pulse: P

State Transition Diagram



State Transition Table

Current State	L	Next State
S0	0	S0
S0	1	S1
S1	0	S0
S1	1	S2
S2	0	S0
S2	1	S1

Output Logic

P=1 when in state S1.

FSM Verilog Template

Five elements:

1. Inputs and outputs
2. Internal signal definition
3. State register: `always_ff` block. Make sure you have a reset!
4. Next state logic: `always_comb` block or `assign` statements.
5. Output logic: `always_comb` block or `assign` statements

FSM Verilog Template

Module and signal declaration.

```
1 // This module converts a level change on an input signal to
2 // a single clock cycle output pulse.
3
4 module level_to_pulse_converter(
5     input logic clk, reset,
6     input logic L,
7     output logic P
8 );
9
10    logic [2:0] state, nextstate;
11
12    parameter S0 = 3'b001;
13    parameter S1 = 3'b010;
14    parameter S2 = 3'b100;
15
16    // Could also use something like the following for specifying the
17    // state encodings.
18    // typedef enum logic [1:0] {S0, S1, S2} statetype;
19    // statetype state, nextstate;
20    ...
```

FSM Verilog Template

State register.

```
1  ...
2  // State register
3  always_ff @(posedge clk, posedge reset)
4      if (reset) state <= S0;
5      else      state <= nextstate;
6
7  ...
```

FSM Verilog Template

Next state and output logic.

```
1  ...
2  // Next state logic
3  always_comb
4      case (state)
5          S0: if(L) nextstate = S1;
6              else nextstate = S0;
7          S1: if(L) nextstate = S2;
8              else nextstate = S0;
9          S2: if(L) nextstate = S2;
10             else nextstate = S0;
11         default: nextstate = S0;
12     endcase
13
14     // Output logic
15     assign P = (state == S1);
16 endmodule
```

Develop a testbench for this project

Steps to create a testbench

1. Create clock signal which toggles continuously for any synchronous elements.
2. Initial statement to apply reset and set inputs to desired initial values.
3. Another initial block to apply input signals.

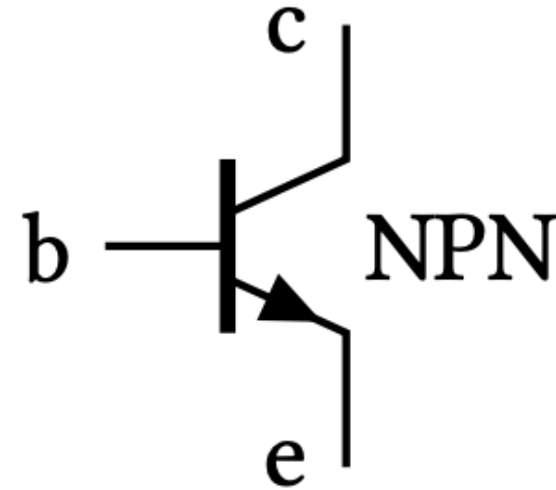
Don't apply signals on a clock edge! (e.g., make sure that if you are using a clock period of 10 timesteps that you don't apply your inputs at multiple of 10.

Testbench Code

```
1 // Compiler directive to set unit and resolution.
2 `timescale 1ns/1ns
3
4 module l2p_tb();
5     logic clk, reset;
6     logic L, P;
7
8     // Instantiate device under test (DUT)
9     level_to_pulse_converter l2pc(clk, reset, L, P, debug);
10
11    // Generate clock signal
12    always begin
13        clk = 1; #5; clk = 0; #5;
14    end
15
16    // Apply reset signal
17    initial begin
18        reset = 1; #44; reset = 0;
19    end
20
21    // Apply input signals
22    initial begin
23        L = 0; #12; L = 1; #12;
24        L = 0; #44; L = 1; #20; L = 0; #32; L = 1;
25    end
26
```

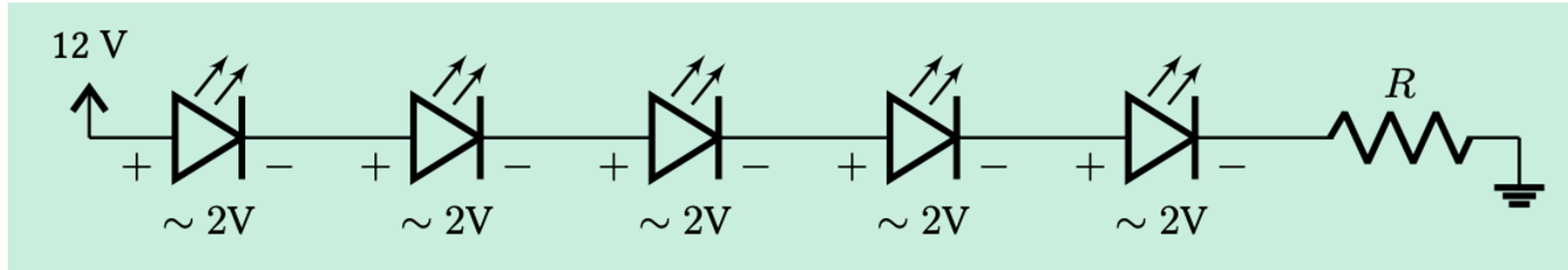
Diode and Transistor Review

- $i_D = i_0 \left(\exp\left(\frac{v_D}{n \cdot v_T}\right) - 1 \right)$
- n and i_0 are scaling factors and v_T is the thermal voltage which is $v_T = kT/q$ (25.4 mV at room temperature).
- For a silicon diode, $v_{on} \approx 0.7V$ and for an LED $v_{on} \approx 1.7 - 2.1V$



NPN Symbol

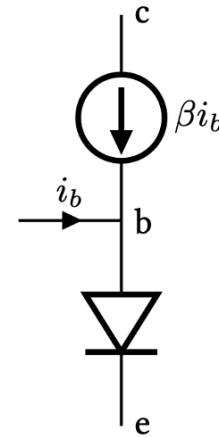
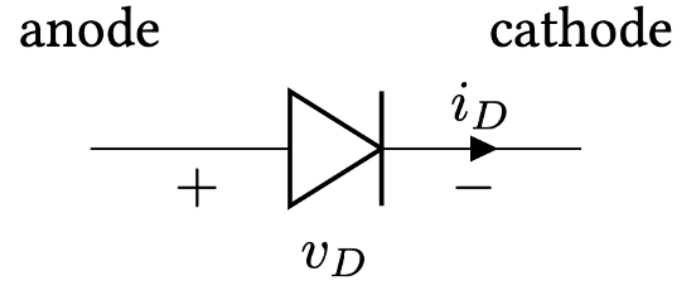
How many LEDs can you light up in series from a 12 V source?



$$i_D = \frac{12 - 5 \cdot 2}{R} = \frac{2}{R}$$

Transistors

Used to pull load **low**. (i.e., connect load to **ground**)



BJT small signal model

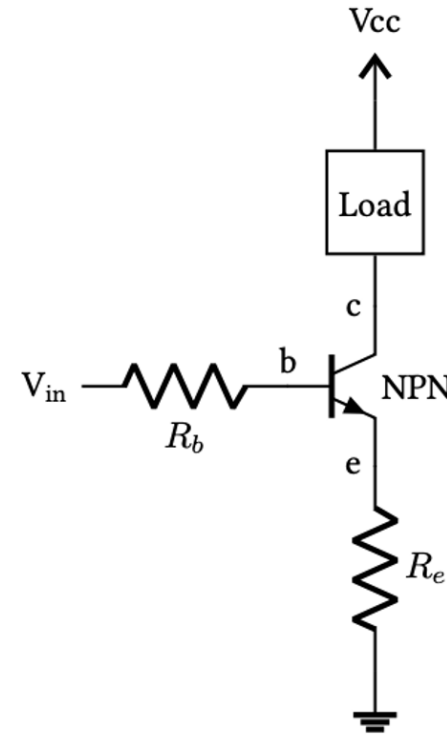
Driving a load with an NPN transistor

How do we choose R_b and R_e ?

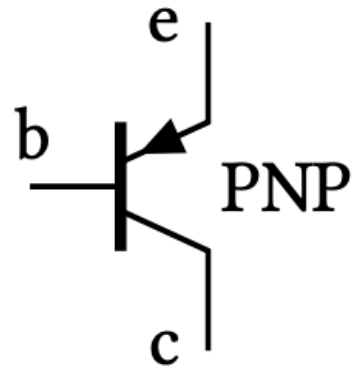
Set R_e to zero.

$$i_b = \frac{V_{in} - 0.7}{R_b}$$

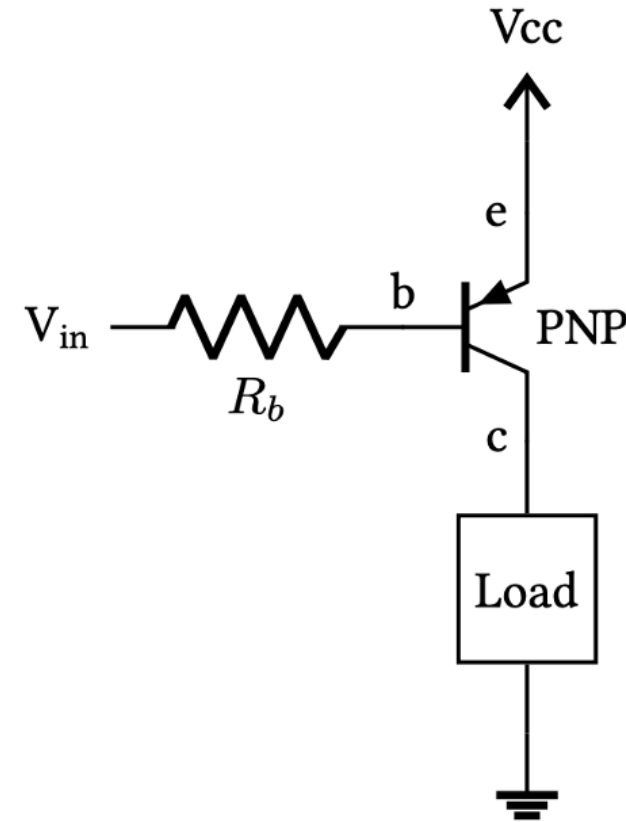
$$R_b = \frac{3.3 - 0.7}{0.1} = \frac{2.6}{0.1} = 26\Omega$$



Driving a load with an PNP transistor



PNP Transistor Symbol



Used to pull load **high**. (i.e., connect load to **power**)

Wrap up

- Synchronous sequential design enables us to design simple and robust digital systems.
 - Only one clock signal to all flops (single clock domain)
 - Ensure that the setup and hold time constraints are observed.
- We need to synchronize asynchronous inputs to avoid metastability. Price is an additional clock cycle of latency.
- Transistors are like electrically controlled switches and enable us to drive larger loads from weak source (e.g., FPGA/MCU I/O pins)

Announcements/Reminders

- Checkoffs continue today – don't delay starting on Lab 2. Can reuse code from Lab 1
 - Only **one** seven_seg Verilog module.
 - Make sure LEDs are consistent brightness no matter how many segments are on
 - Develop a testbench to confirm your circuit is working. See tutorial on the website.