Debouncing and Multiplexing

ECE 362
https://engineering.purdue.edu/ece362/

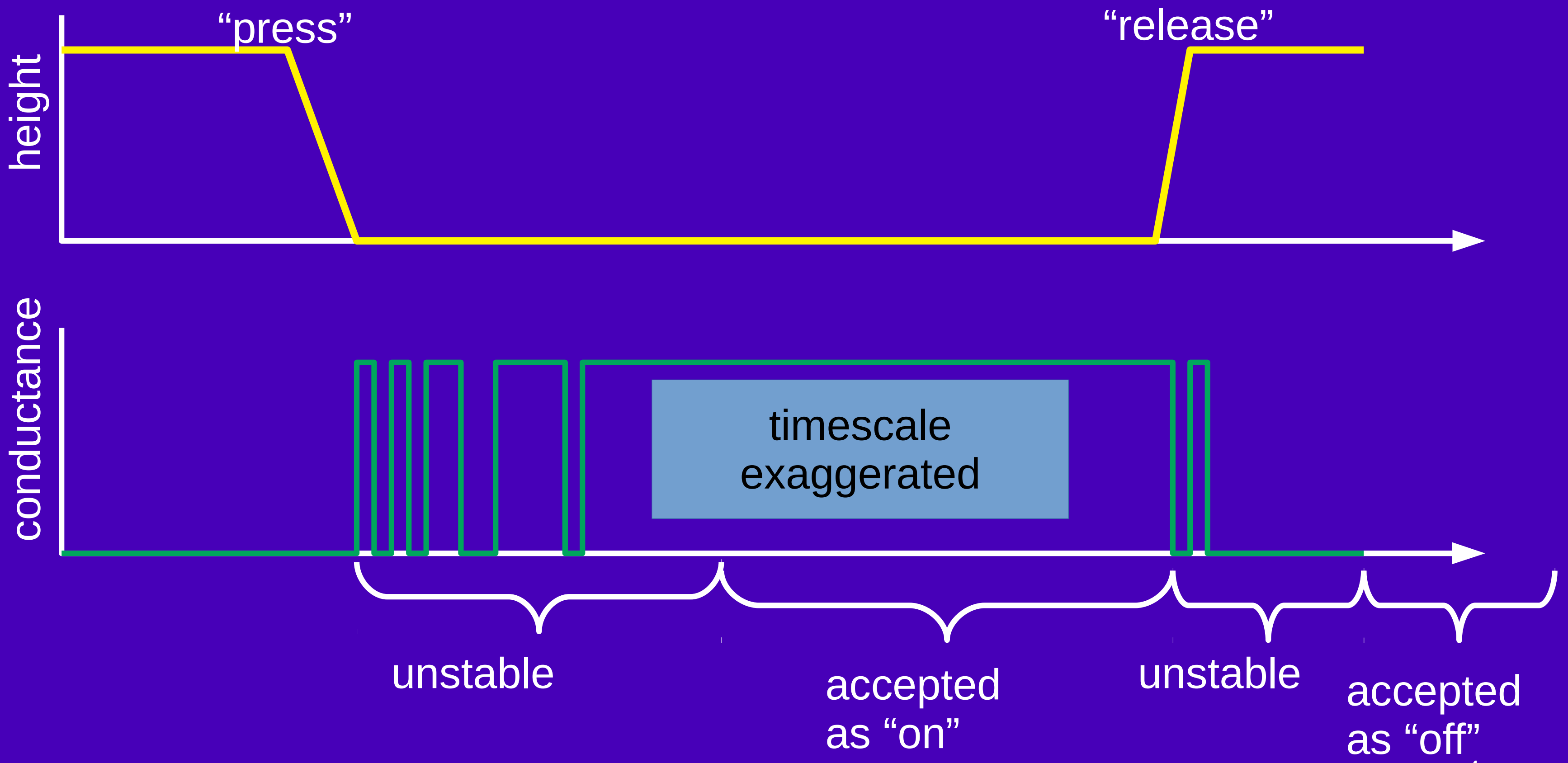
Reading Assignment

- Reading assignment:
  - Textbook, Chapter 21, Digital-to-Analog Conversion, pp. 507 – 526.
    - Probably read this first.
  - FRM, Chapter 14, Digital-to-analog converter (DAC), pp. 269 – 281.
    - Scan. Learn basics like I/O registers, enabling, use.
  - Textbook, Chapter 20, Analog-to-Digital Conversion, pp. 481 – 506.
    - Read this later.
  - FRM, Chapter 14, Digital-to-analog converter (DAC), pp. 269 – 281.
    - Scan. Learn basics like I/O registers, enabling, use.
  - Family Reference Manual, Chapter 17, "General purpose timers (TIM2 and TIM3)", pages 377 – 443.
  - Textbook, Chapter 15, “General-purpose Timers”, pages 373 – 414.

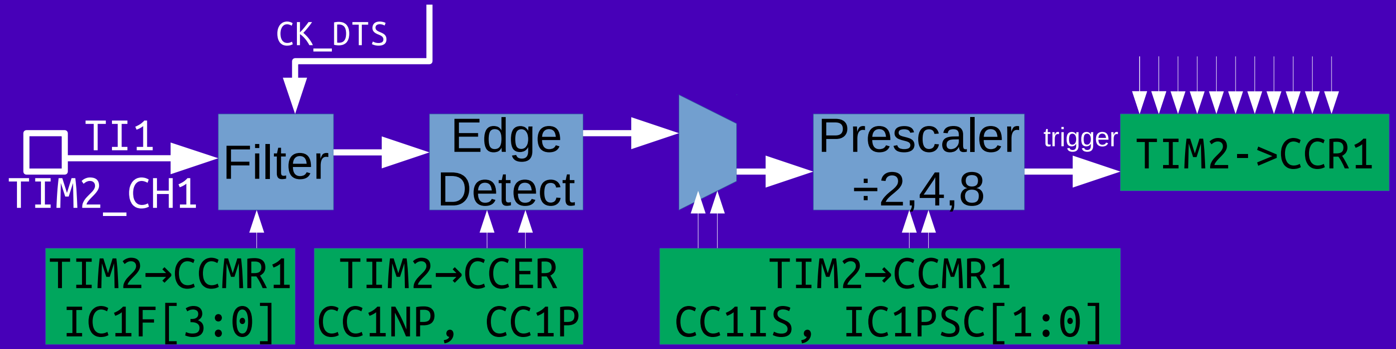
Everything bounces

- Most mechanical switches consist of a conductive plate that closes a circuit between two contacts.
  - Press the switch, and bounces.

What does a bounce look like?

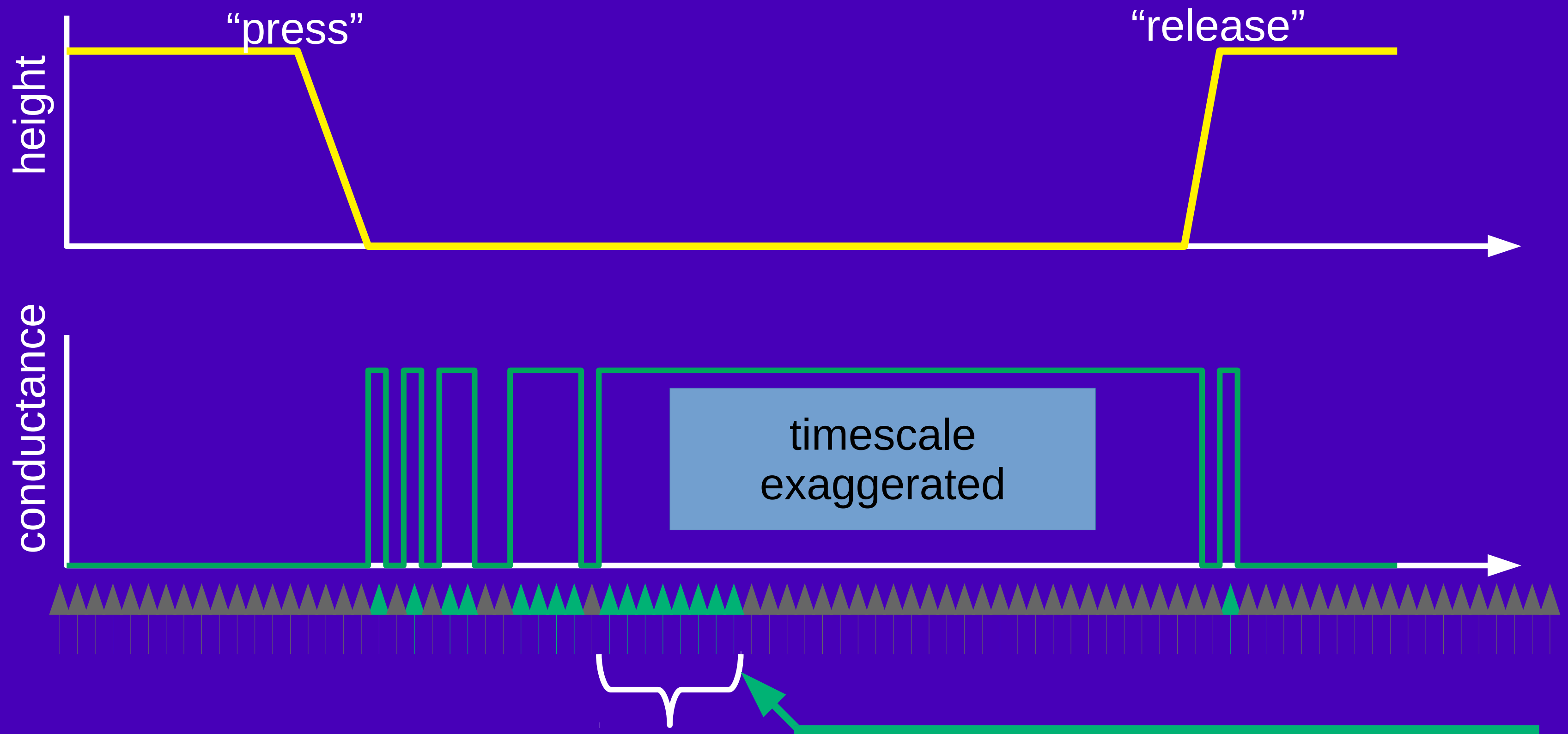


How can input filtering help us?



- In the past, I’ve explained how to set up input filtering on the general purpose timers to:
  - Sample at every M ticks of CK_DTS
  - Require N positive (“on”) samples before accepting.
  - Generate an event only on rising edge.
  - No prescaler division.

Sampling the bounce



searching for 8 positive reads in a row

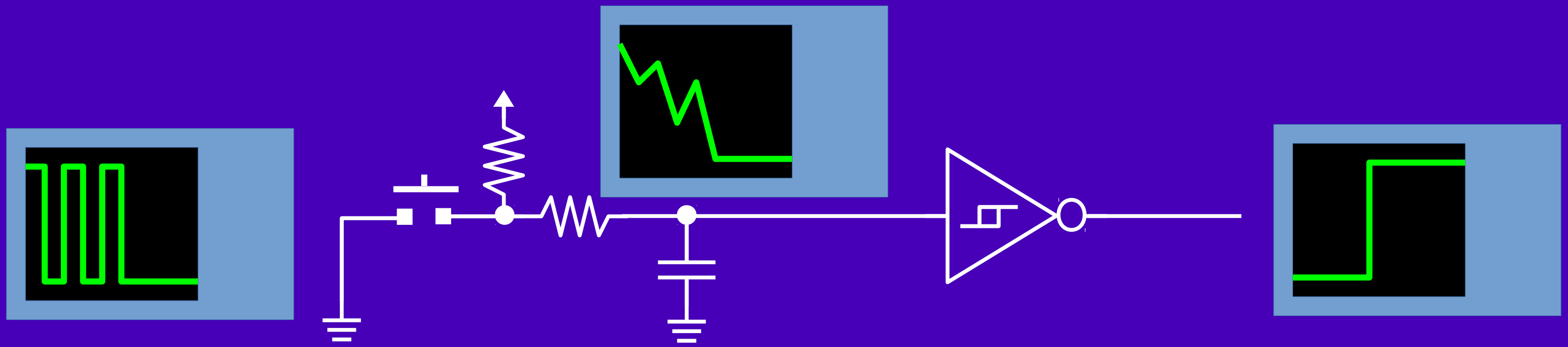
Success. Positive edge registered and interrupt request raised.

Debouncing is still not perfect

- This works great when the system clock is 8MHz.
  - 8 sample intervals take about .1ms
- At 48MHz, it’s better than nothing, but not perfect.
  - 21.3µs is too short of a sample interval for many types of buttons.
  - There are no other options for fixing how to do this using built-in hardware and configuration.
  - We will do software solutions instead.
  - See the textbook, pages 360 – 371, for more ideas.

Effective Debouncing of One Button

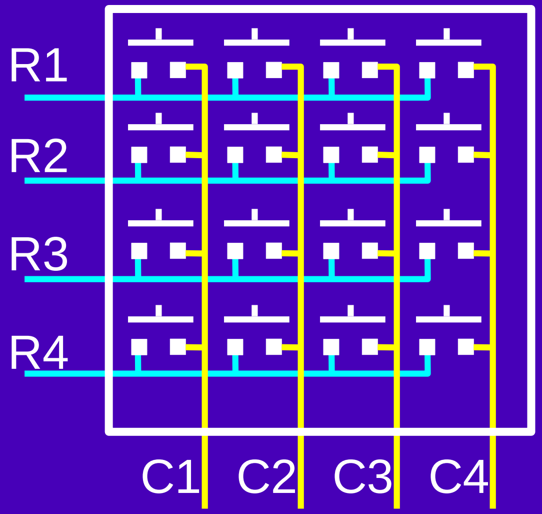
- Consider the following circuit:



- The on-off-on bouncing of the switch is "smoothed" by the R-C network. Normally, the slow rise and fall time causes problems for digital inputs.
  - The Schmitt Trigger doesn’t mind slow inputs.
  - As long as the RC constant is much larger than the bounce time, the output is a bounce-free digital signal.

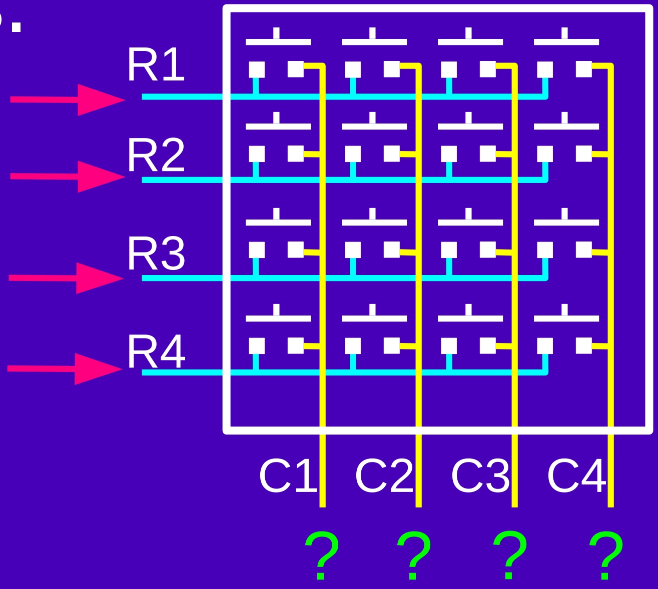
Keyboard Matrix

- Not many situations call for a single button.
- Most of the time, you have a <u>matrix</u> of keys.
  - For 16 buttons, you don’t want to waste 16 pins (and 16 Schmitt inverters) to read them all. Arrange them in a matrix.
  - And they still bounce.
  - You must scan them.
  - You don’t have to watch every button all the time. Just check each rapidly enough to notice a push soon after it happens.



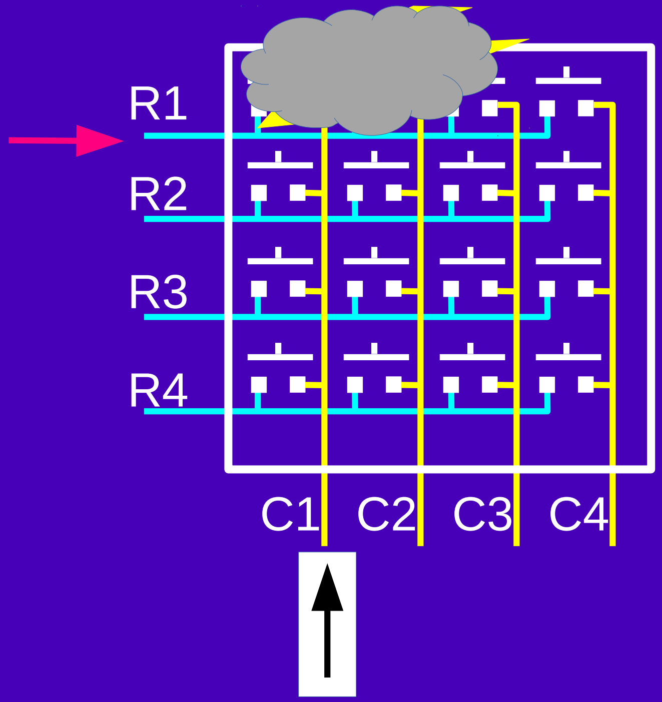
How do we scan keys?

- Apply voltage to one row.
- Check for voltage on columns.
- Turn off voltage.
- Turn on voltage for next row.
- And so on...



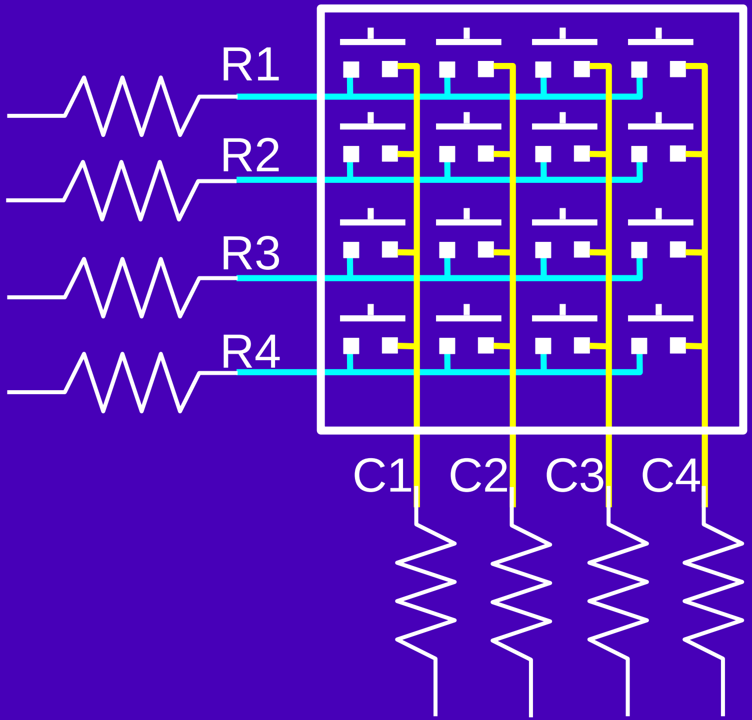
Small danger

- You use the STM32 pins to scan by changing pins from input to output. But what if you...
  - Apply power to one row...
  - Apply ground to one column…
  - And then you push the upper-left button?



Safety First

- At least put some resistors here… to limit how much current flows through a button <u>when</u> you make a mistake.



Use of timer/interrupt

- The work of scanning can be done incrementally using a timer interrupt.
- On each interrupt, the ISR will:
  - read all the columns
  - put the value read for each key of the current row into its own *history byte*
  - turn off the voltage for the row
  - turn on the voltage for the next row (for the next ISR invocation)
  - return

- Why do it in this order?
  - You could turn on a row and immediately read the columns, but this way gives the voltage on the row/column connection time to settle in between ISR invocations.

Debouncing a matrix

- As key matrix is scanned, keep track of what the last 8 values read for each key. (1 for currently pressed, 0 for currently released)
  - left shift its latest reading into a byte of memory called a history byte.
- If key idle for a long time, the byte for the key will be 00000000
- The first time a key is pressed, its history will become 00000001
- If it bounces, it may be 00000101 or 00010101
- After it is pressed and stable for a long time, it will be 11111111
- The moment it is first released, it will be 11111110
- If it bounces on release, it may be 11111010 or 11101010

Detection

- To detect a press or release, search all the history bytes that represent the keys:
  - `00000001`: key pressed
  - `11111110`: key released
  - ignore any other values

How Quickly Should We Scan?

- Much faster than keys can be pressed and released.
  - It’s possible to repeatedly press and release a single button 10 times per second (maybe).
- Slower than the total bounce time for any key.
  - Don’t scan so fast that you can read 00000001 multiple times for a single (bouncing) press.
- If a button can bounce for 10ms, and we scan one of four rows every 1ms, then the worst possible history byte for a single press would be 00000101. (Individual bounces separated by 2 bits.)

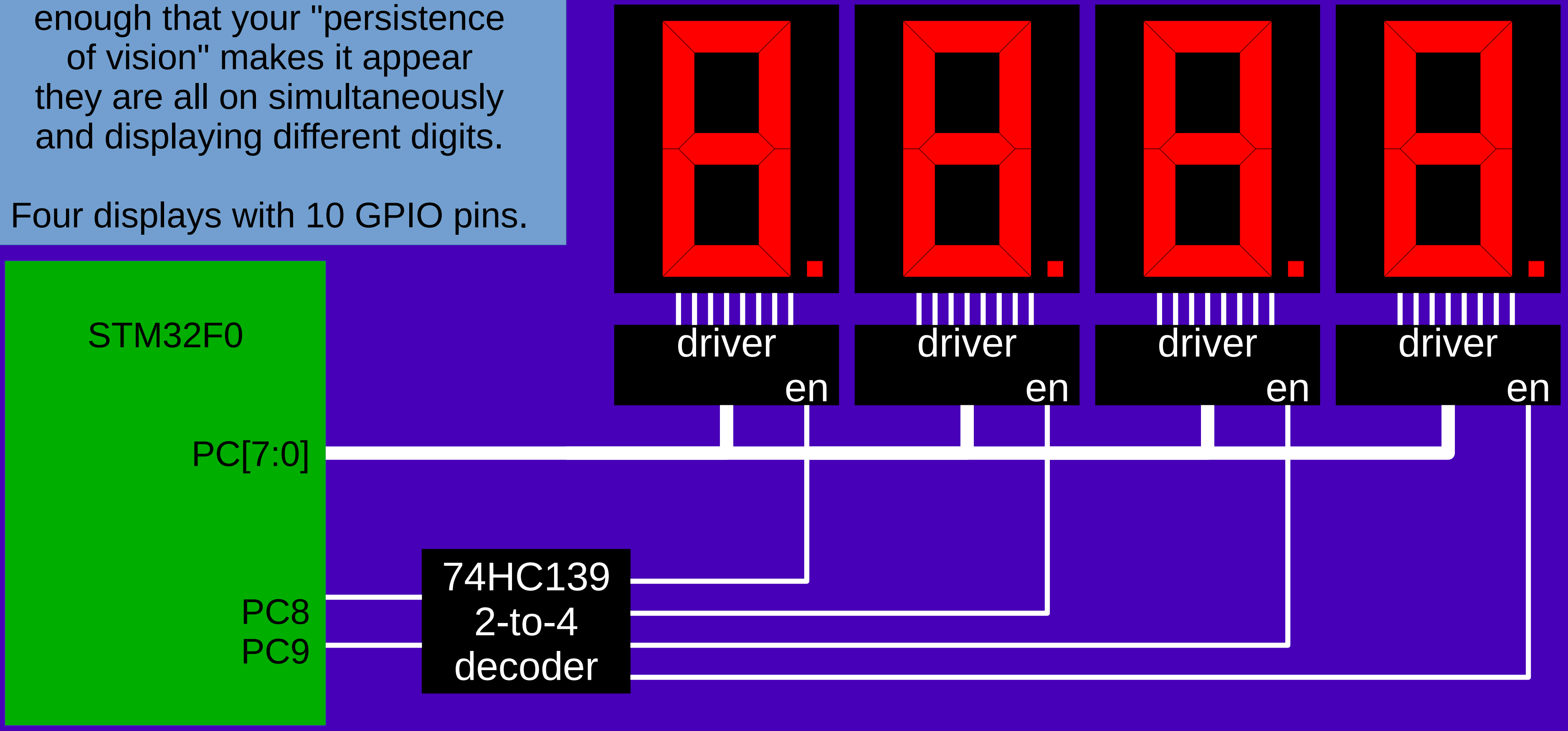
Output Multiplexing

- Key scanning is a specific example of input multiplexing and encoding.
  - If you use microcontrollers, you may spend a lot of time doing things like this.
- Another example: driving displays.
- There are eight 7-segment displays in your lab kit.
  - You do not want to use 64 STM32 pins to drive segments individually.
  - You can multiplex them with far fewer pins.
  - They are already configured in two groups of 4 to allow this.

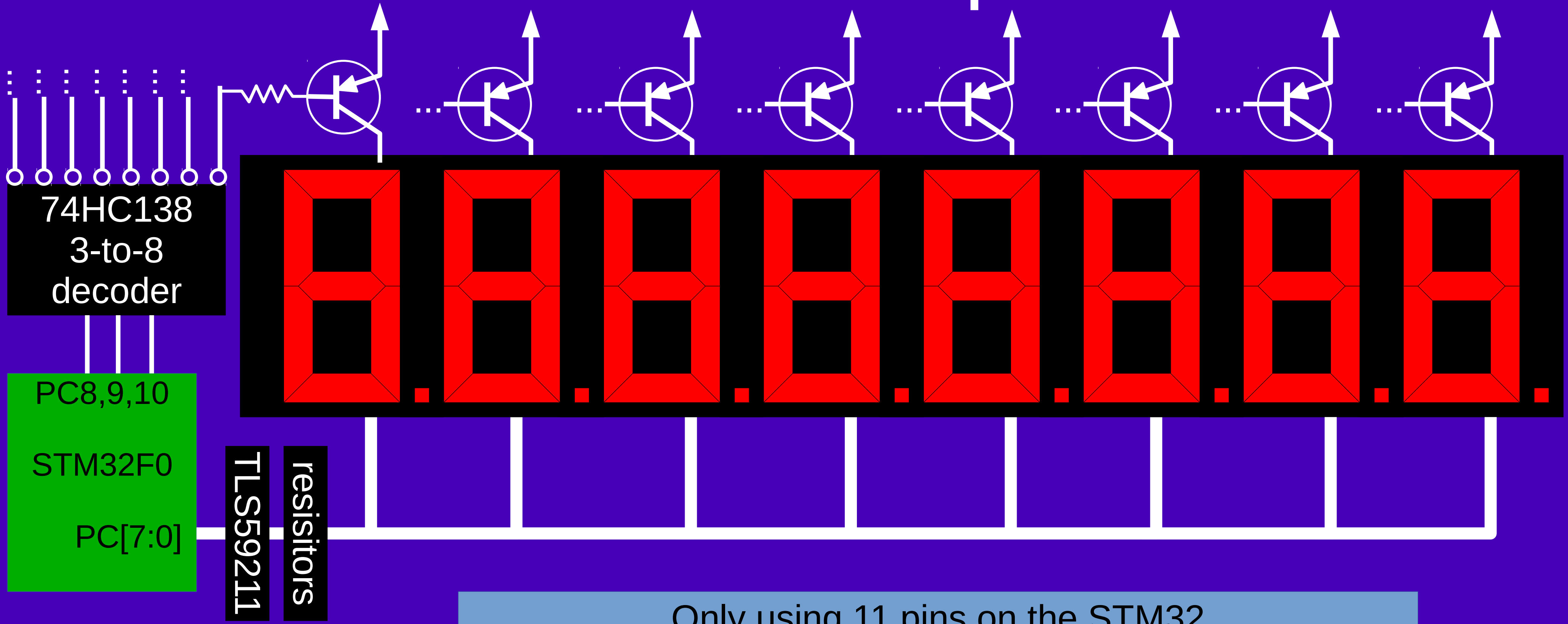
Example

Turn on one display at a time. Rotate through them rapidly enough that your "persistence of vision" makes it appear they are all on simultaneously and displaying different digits.

Four displays with 10 GPIO pins.

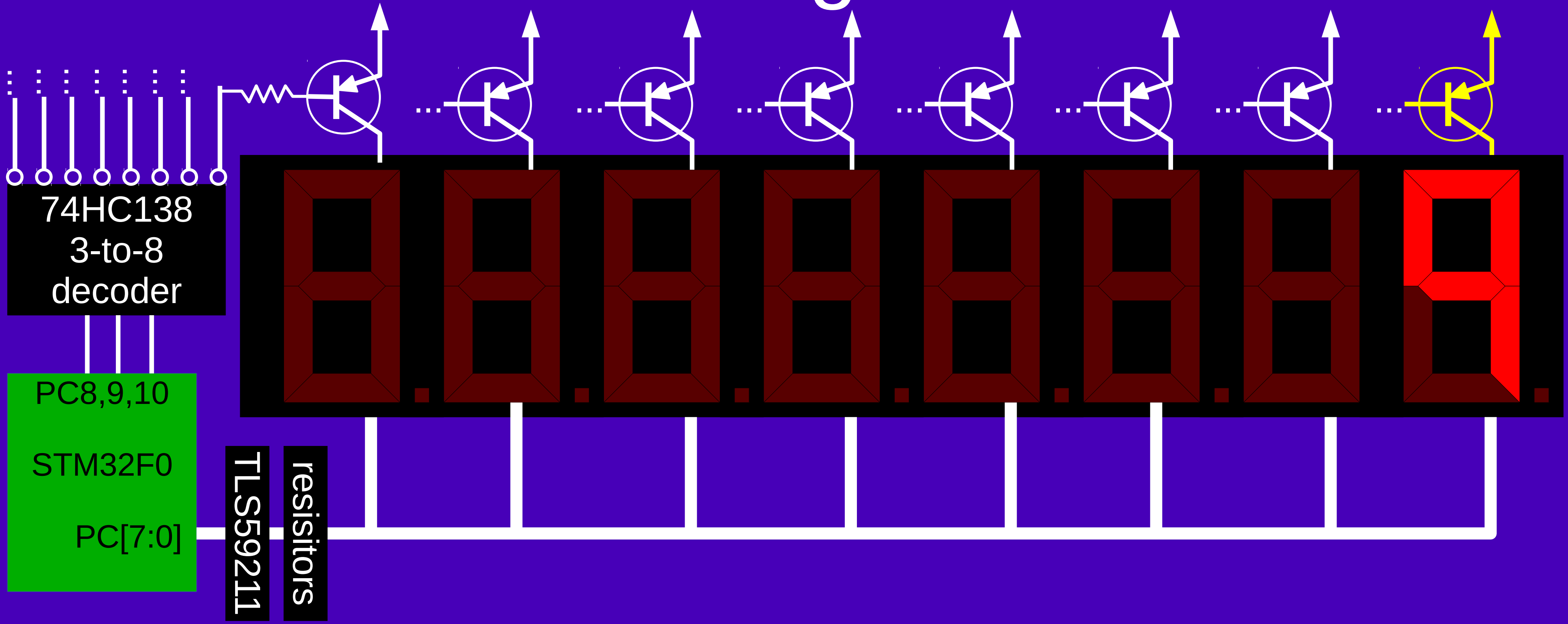


Better Example

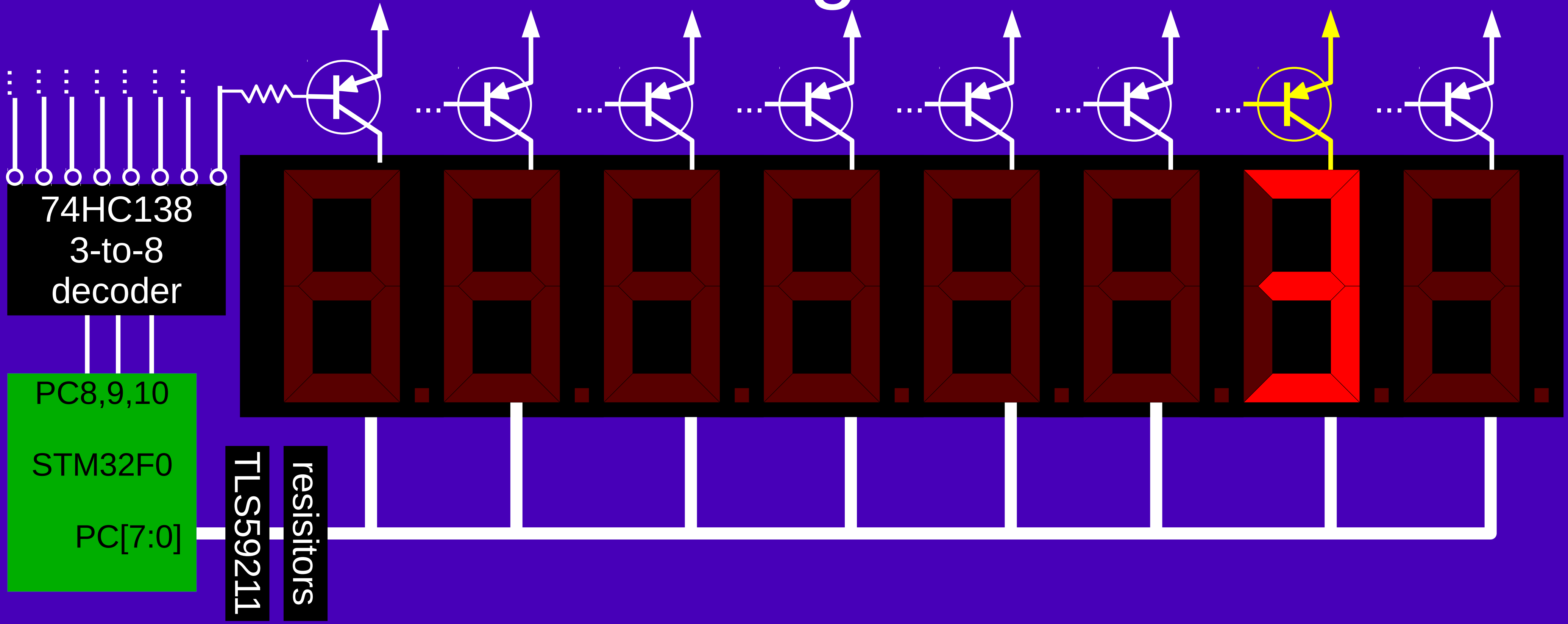


Only using 11 pins on the STM32
(Generally, $\log_2$(digits) + 8 pins)

Select One Digit At A Time



Select One Digit At A Time



Much of Microcontroller Development is Multiplexing

- We’ll use the multiplexing input and output systems shown in this lecture in multiple lab experiments