Introduction to Design (2)

Microcontrollers and Interfacing

Week 08

using LED arrays
direct access to I/O registers
parallel digital output

KUAS 京都先端科学大学
KYOTO UNIVERSITY of ADVANCED SCIENCE
Department of Mechanical and Electrical System Engineering

this week

parallel digital output

- connecting many LEDs
- software techniques

fast parallel output

- directly accessing memory-mapped hardware registers
- configuring, writing and reading I/O pins in parallel

LED arrays: bar graph

many kinds of display are just ‘lots of LEDs’

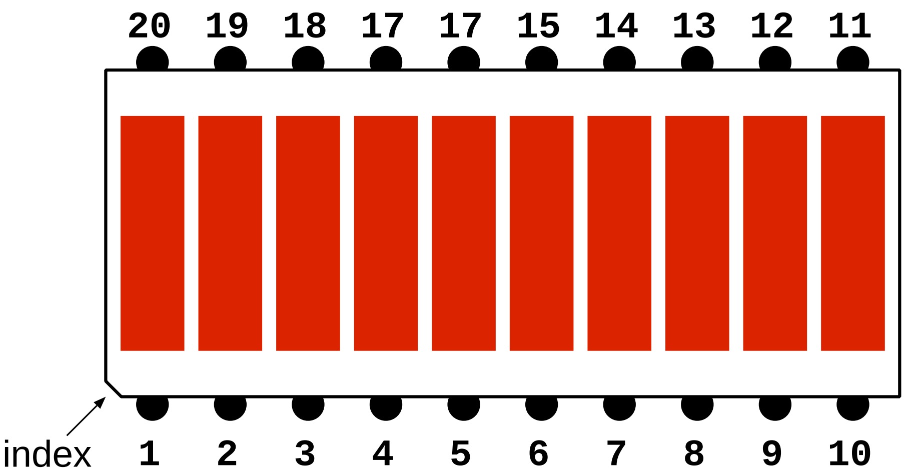
- LED *arrays*

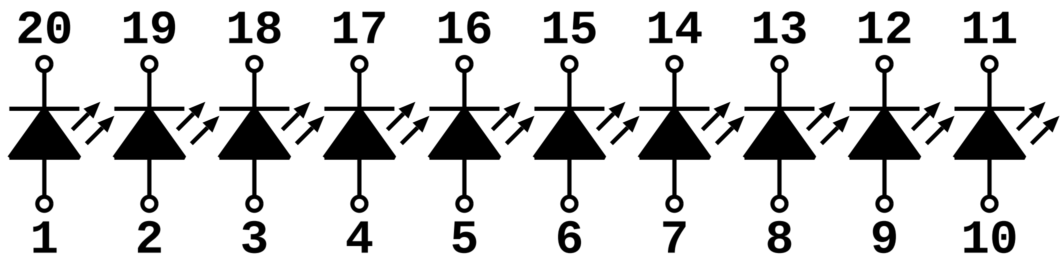
the simplest is the *bar graph* display; typically

- $\approx$ ten identical LEDs, all in one package
- each LED independent of the others (no common connections)
  $\Rightarrow$ 10 LEDs $\times$ 2 terminals each $=$ 20 pins on the package

polarisation is important: an *index* tells us where pin 1 is located

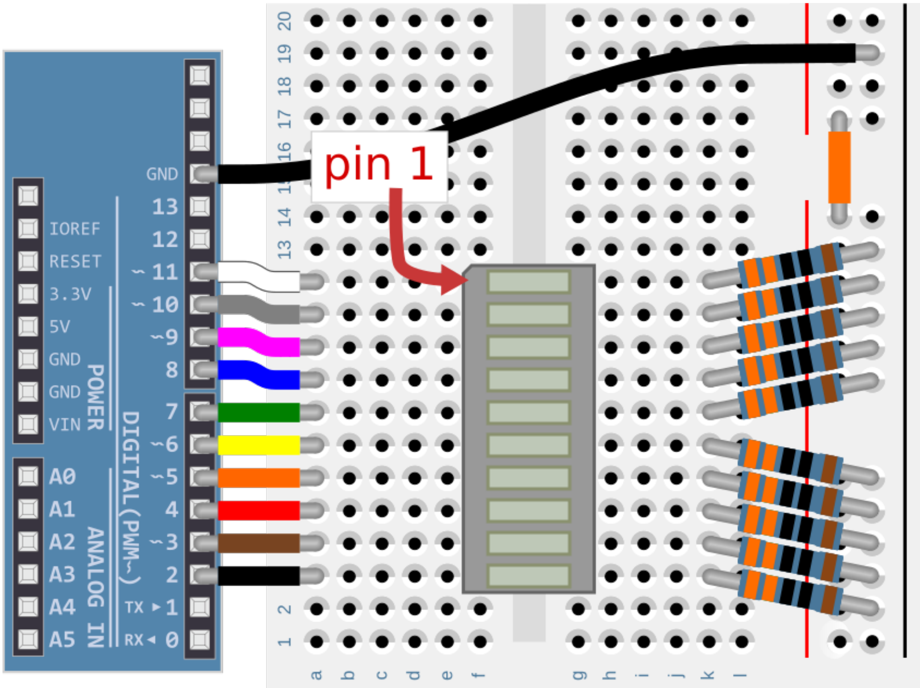
- pins 1–10 are the anodes, and pins 11–20 are the corresponding cathodes





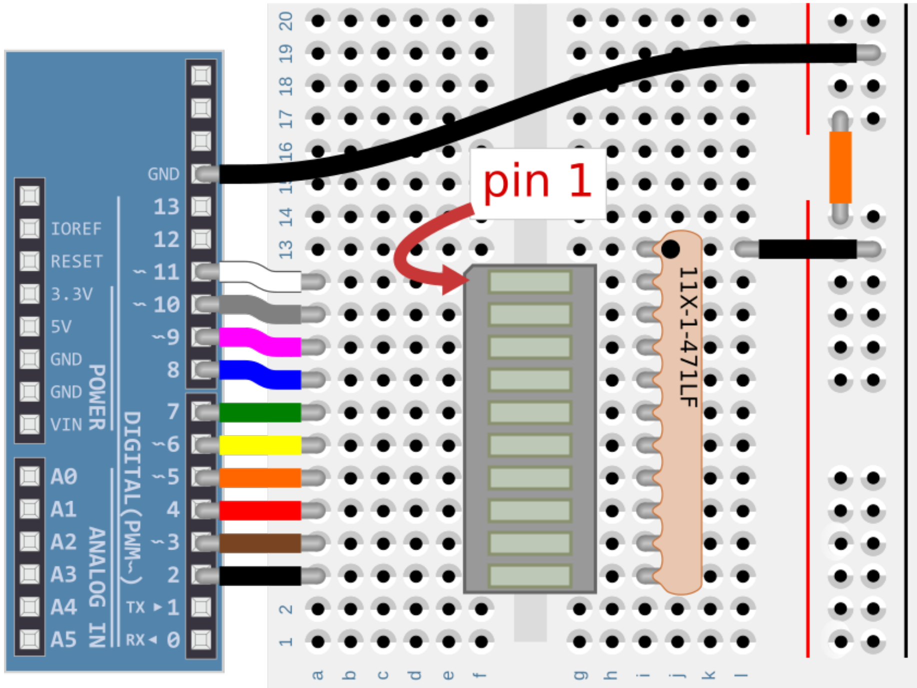
LED arrays: bar graph

each LED requires its own series resistor: *ouch!!*



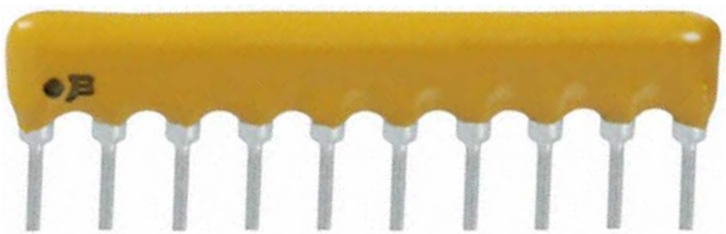
Individual, discrete current-limiting series resistors.

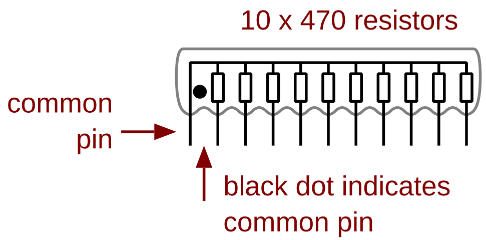
→



10 current-limiting resistors in a single network.

you have two ‘parallel resistor networks’ that make this painless

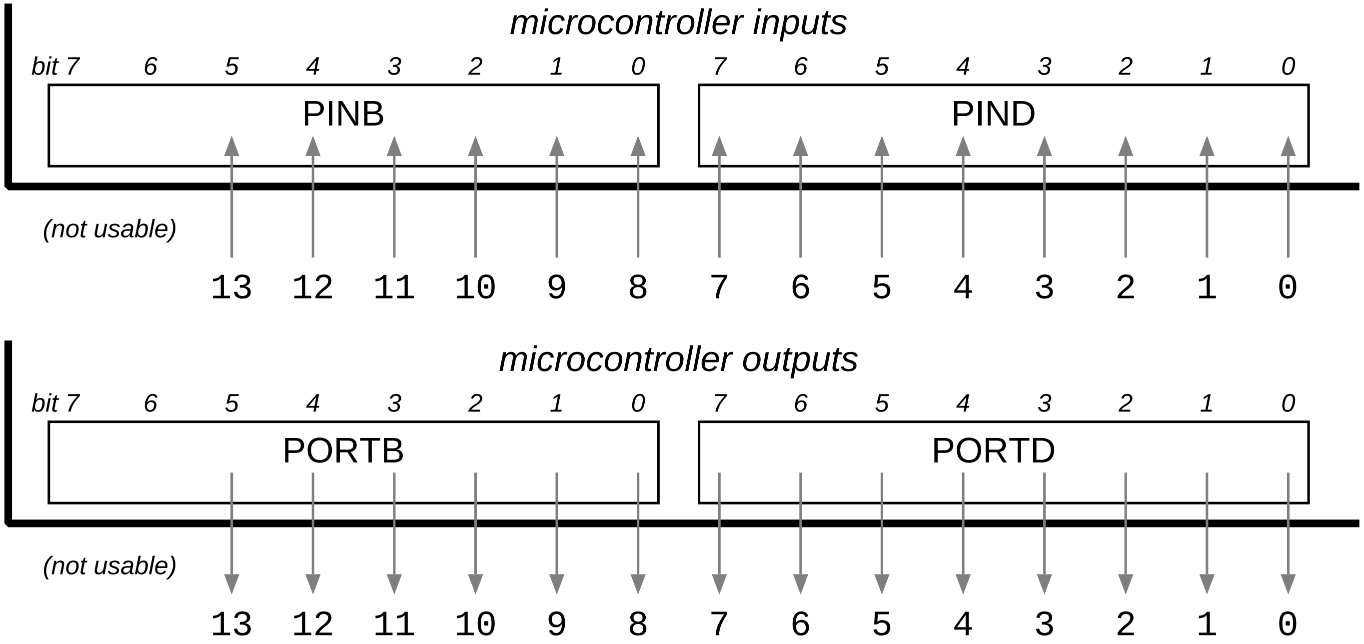




direct access to I/O pins

the I/O pins are connected to *pin* and *port* registers

- each register is 8 bits wide
- PIN registers are for input *only*, PORT registers are for output *only*
- digital pins 0 to 7 are connected to PIND/PORTD
- digital pins 8 to 13 are connected to PINB/PORTB

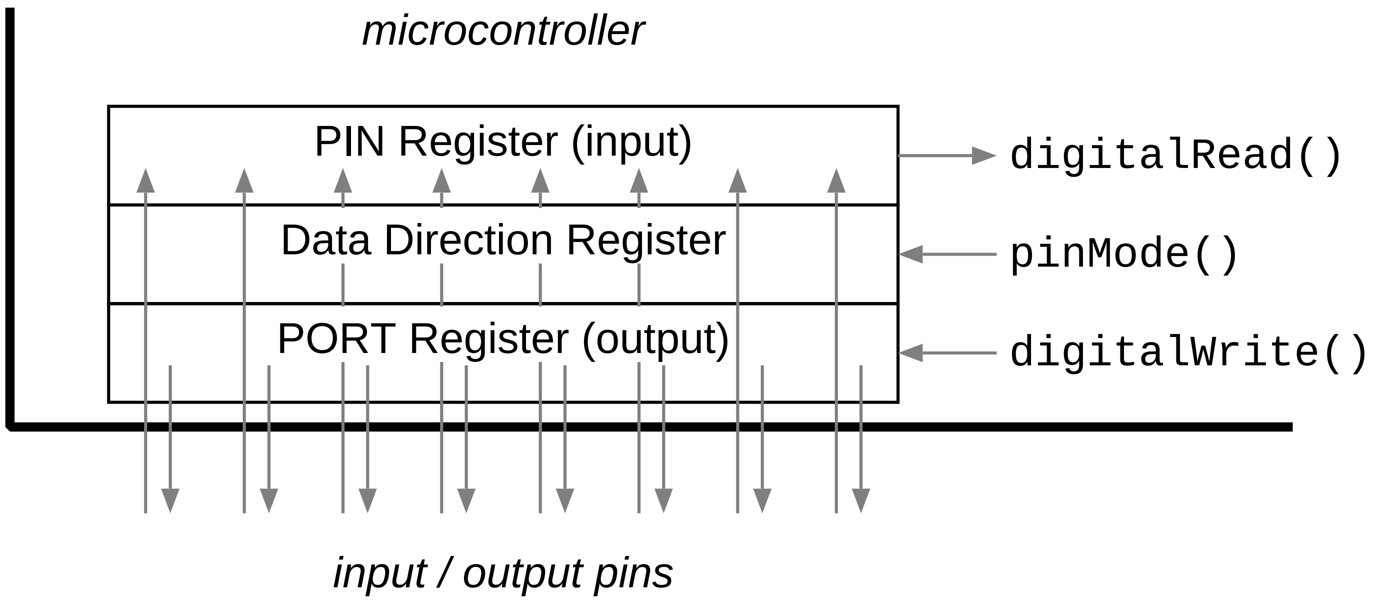


direct access to I/O pins

each physical pin is either an input or an output, but *not both* at the same time

pin configuration (input or output) is done using the *data direction register* (DDR)

- each DDR is 8 bits wide: each bit configures one digital pin
  - `0` for input (default): corresponding PORT bit can be set to `1` or `0`
  - `1` for output: corresponding PIN bit can be read as `1` or `0`
- DDRD configures digital pins 0–7; DDRB configures digital pins 8–13



direct access to I/O pins

the registers can be read/written *directly* by programs using symbolic names

		registers		
pins	*port*	*direction*	*output*	*input*
0 – 7	PORTD	`DDRD`	`PORTD`	`PIND`
8 – 13	PORTB	`DDRB`	`PORTB`	`PINB`

reading a PORT register returns the last value you wrote there

binary constants can be written, e.g: `B0110` (equal to 6), `B00100000` (bit 5 set)

use 'bit-wise' operators (`&`, `|`, `^`, and `~`) to modify only relevant pins

`~x`	inverts each bit in `x`		`~B00100000 == B11011111`
`x &= y`	clears bits in `x` where `y` has `0`s		`x &= B11011111`
		or	`x &= ~B00100000`
`x |= y`	sets bits in `x` where `y` has `1`s		`x |= B00100000`
`x ^= y`	inverts bits in `x` where `y` has `1`s		`x ^= B00100000`

direct access to I/O pins

using ‘bit-wise’ operators for pin configuration and I/O

```
void setup() {
  DDRB |= B00111000;   // set pins 13, 12, 11 as outputs
}

void loop() {
  PORTB &= ~B00100000; // set pin 13 LOW, LED is off
  delay(100)
  PORTB |=  B00100000; // set pin 13 HIGH, LED is on
  delay(100)
  PORTB ^=  B00010000; // toggle pin 12
  int p9 = (PINB >> 1) & 1;         // copy input pin 9...
  PORTB |= (p9 << 3);               // ...to output pin 11
  PORTB |= (PINB & B00000010) << 2; // same thing in one line
}
```

this is approximately 25 times faster than using `digitalWrite()`

- e.g., PWM frequencies of hundreds of kHz are possible