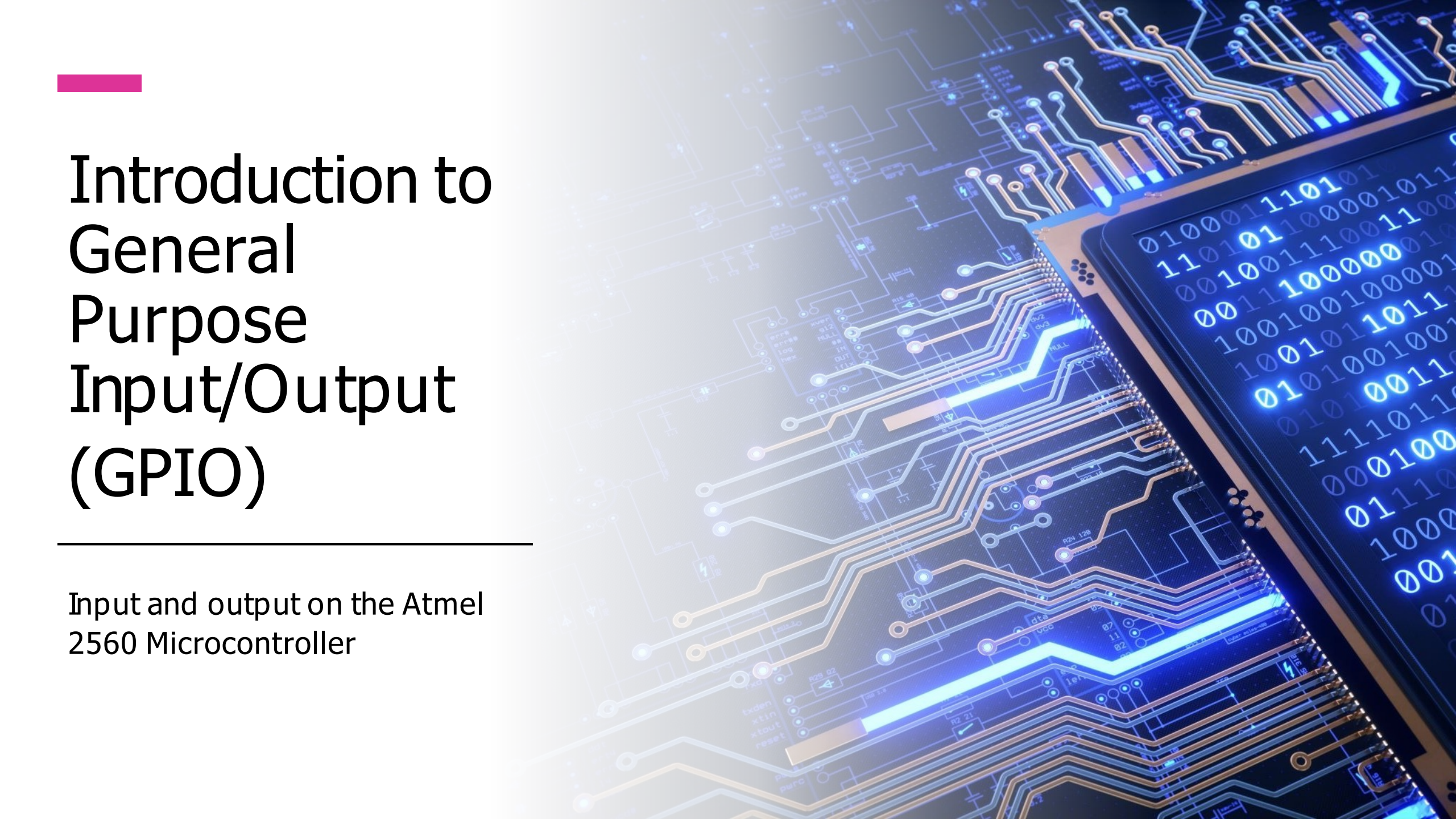
Introduction to General Purpose Input/Output (GPIO)

Input and output on the Atmel 2560 Microcontroller

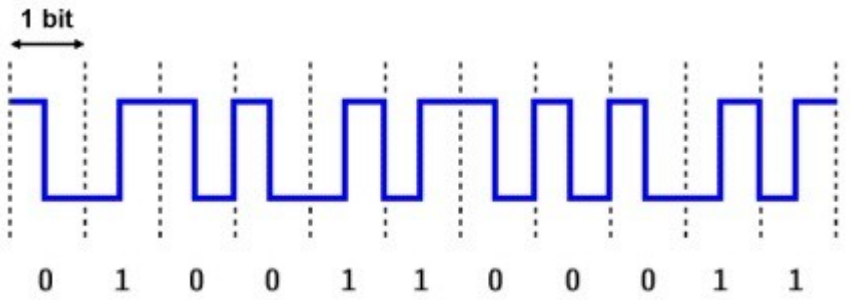


Lecture Outline

- Overview of Digital Signals
- The Atmel 2560 and the Arduino Architecture
- GPIO in Detail

Digital Signals

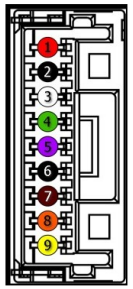
- Digital signals have two states: 1 and 0, High or Low
- The signals are voltages that are *interpreted* as being high or low
  - Example: 0-1 volts -> low, > 1.5 -> high
- Uses
  - Control
    - Turning on or off a motor or lights
  - Status
    - User press of a button
    - Camera shutter closed
  - Communication
    - Serial transmission (ex. MIDI)
    - Parallel



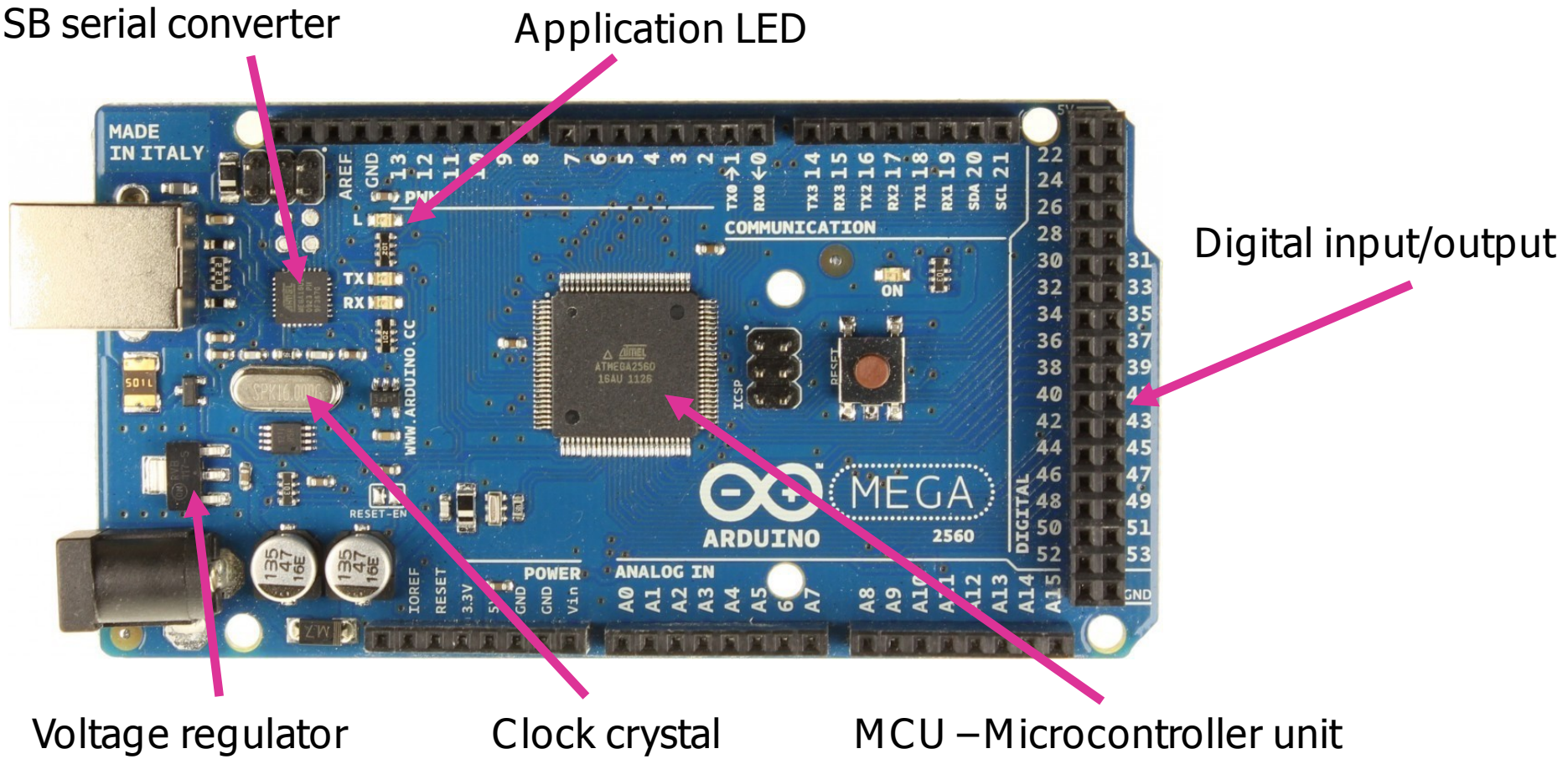
Digital Signal Example

FLIR Chameleon 3 USB Camera

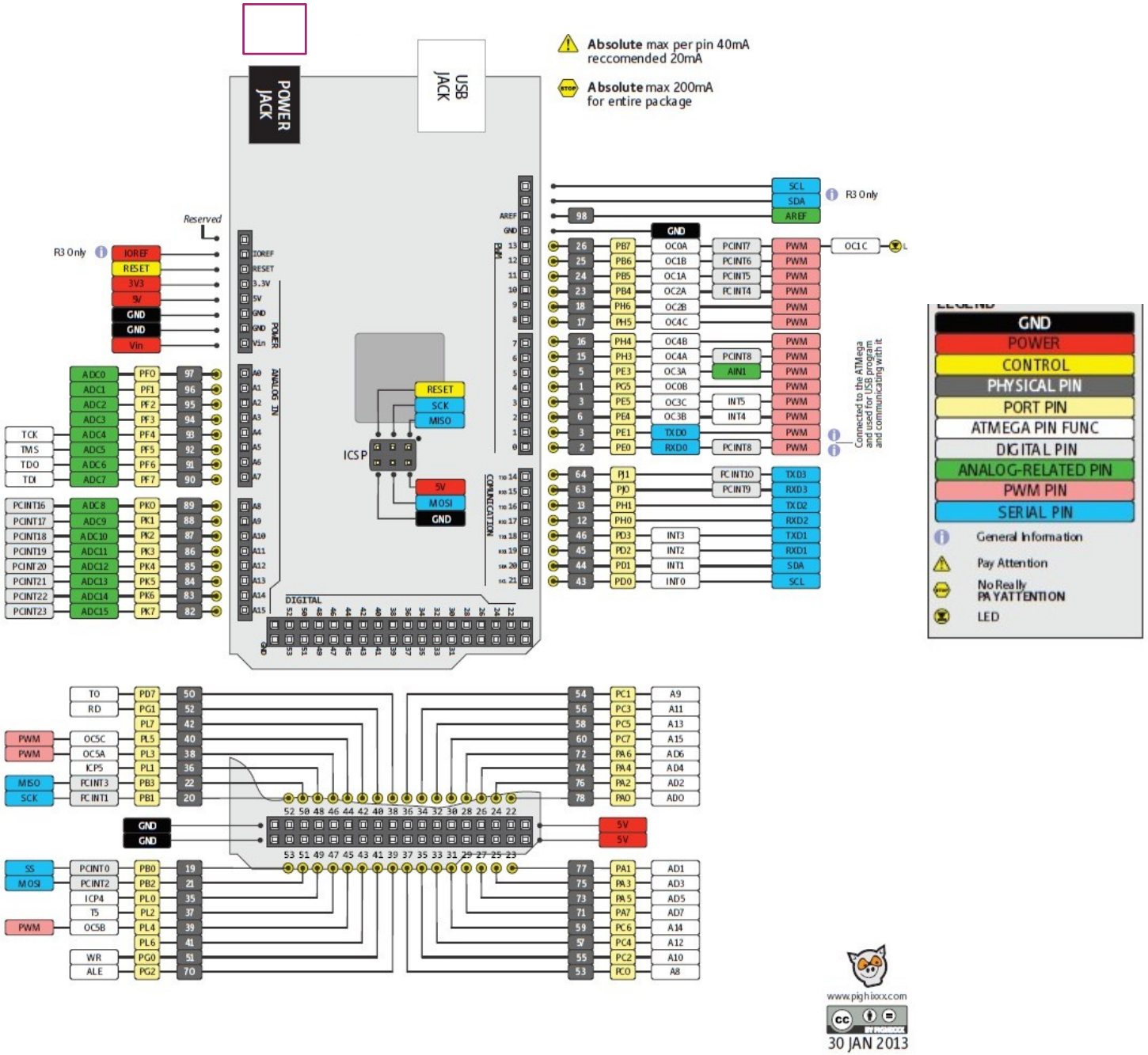
4.9.4 General Purpose Input/Output (GPIO)

Diagram	Color	Pin	Function	Description
	Red	1	V_{EXT}	Allows the camera to be powered externally 5 - 24 VDC
	Black	2	GND	Ground for Input/Output, V_{EXT}, +3.3 V pins
	White	3	+3.3 V	Power external circuitry fused at 150 mA maximum
	Green	4	GPIO3 / Line3	Input/Output/Tx
	Purple	5	GPIO2 / Line2	Input/Output/Rx
	Black	6	GND	Ground for Input/Output, V_{EXT}, +3.3 V pins
	Brown	7	OPTO_GND	Ground for opto-isolated IO pins
	Orange	8	OPTO_OUT / Line1	Opto-isolated output
	Yellow	9	OPTO_IN / Line0	Opto-isolated input

The Arduino Mega

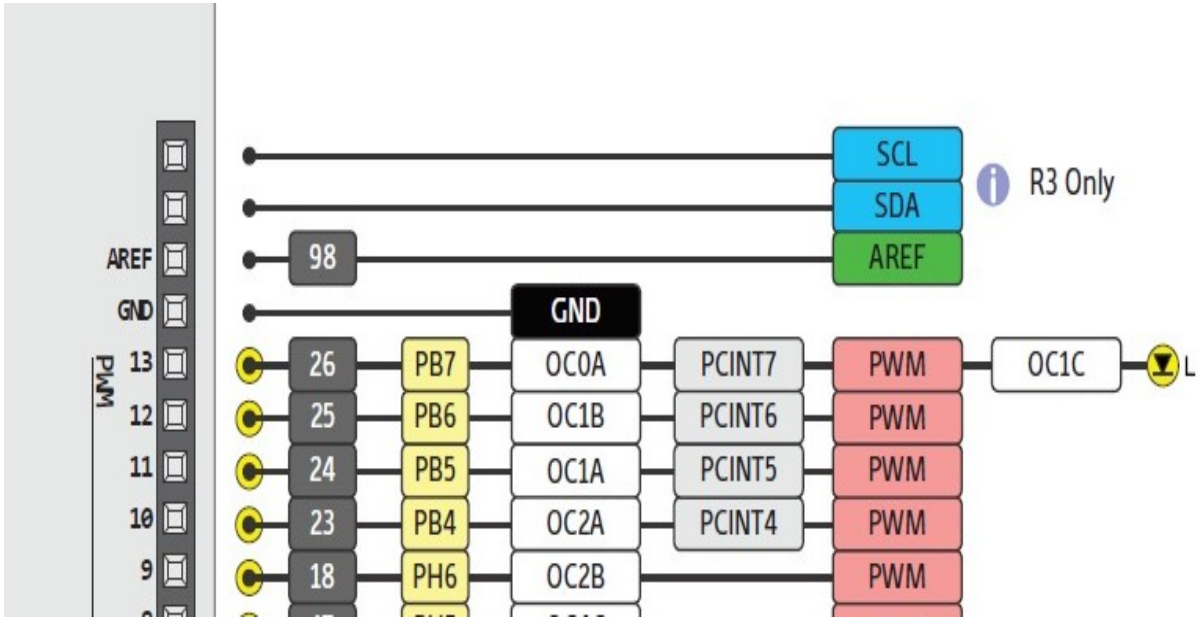


Arduino Mega Pinout Diagram



Pins Serve Multiple Functions

- Multiplexing is used to give pins multiple functions
- Example - Pin 13 is one of
  - PORT B Pin 7: In/Out
  - OC0A: output from timer 0 register A compare
    - Much more on this when we discuss pulse width modulation
  - OC1C: timer 1 register C compare
  - PCINT5 – interrupt #7 input
  - On the Arduino, also connected to an LED
- Which function is dependent on the settings in one or more registers



GPIO Modes

- Three modes of operation
  - Outputting data over the port
  - Reading input from the **P**ort **IN**put
  - Input with pullup for disconnected input pin
    - When connected to a switch, the pin is effectively disconnected and may return a random value

GPIO MODE	DDR	PORT	PIN
OUTPUT	1	DATA OUT	N/A
INPUT	0	0	DATA IN
INPUT with Pullup	0	1	DATA IN

```
pinMode(13, OUTPUT)
digitalWrite(13, HIGH)
```

Setting Modes using Special Registers

- PORTB: Data to be written to the output
- DDRB: Controls whether a pin is an input or output
- PINB: The value present on the input pins

GPIO MODE	DDR	PORT	PIN
OUTPUT	1	DATA OUT	N/A
INPUT	0	0	DATA IN
INPUT with Pullup	0	1	DATA IN

Bit	7	6	5	4	3	2	1	0
0x25	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTB7-0: GPIO data value stored in bit *n*.

Bit	7	6	5	4	3	2	1	0
0x24	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRB7-0: selects the direction of pin *n*. If DDRB*n* is written '1', then PORTB*n* is configured as an output pin. If DDRB*n* is written '0', then PORTB*n* is configured as an input pin.

Bit	7	6	5	4	3	2	1	0
0x23	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Read/Write	R	R	R	R	R	R	R	R
Default	-	-	-	-	-	-	-	-

- PINB7-0: logic value present on external pin *n*.

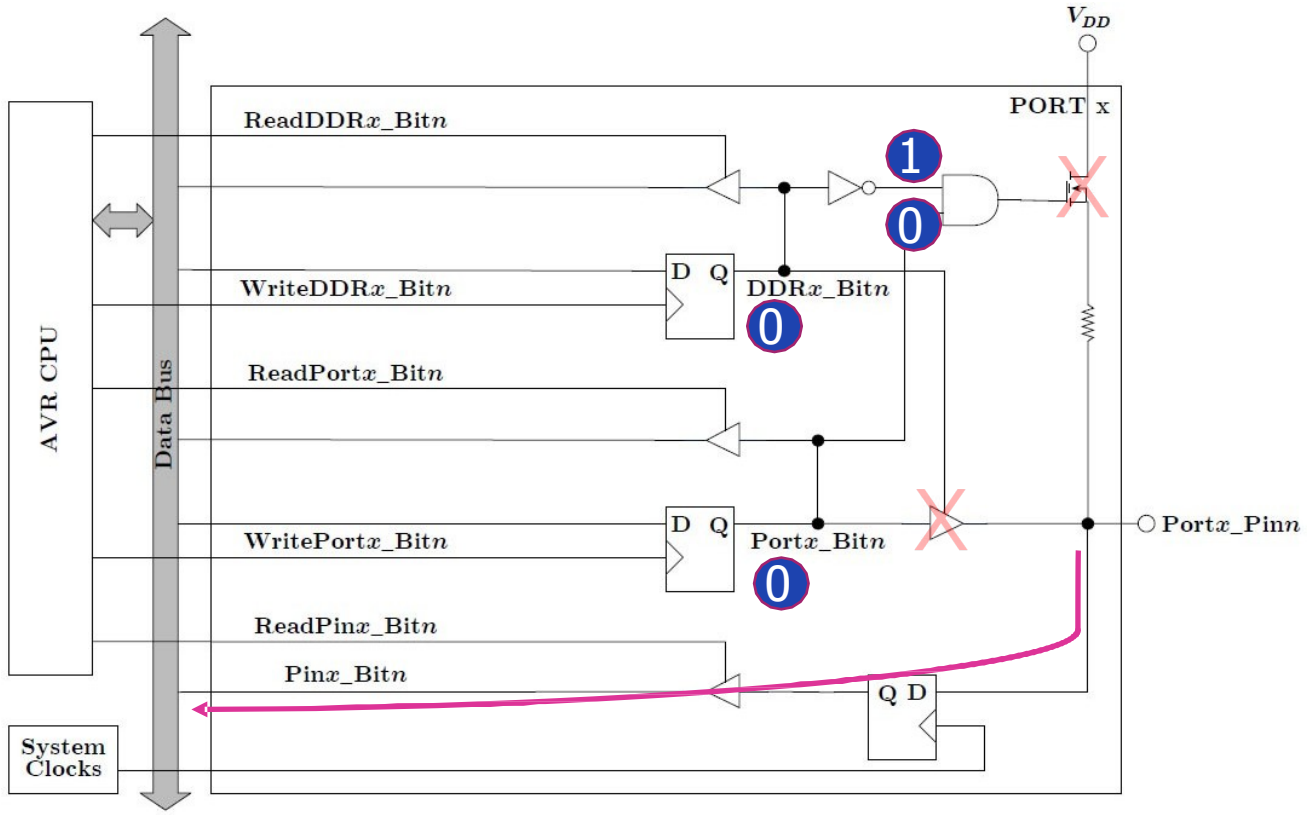
Register Settings for Port B

Read and Write Operations

Reading	Reading a value from PORT B bit 7 (pin 13 on the Arduino, pin 26 on the Atmel 2560) • Write a '0' into the DDRB7 • Read the value from PINB7
Writing	Writing a value to PORT B bit 7 • Write a '1' into DDRB7 • Write '1' or '0' into PORTB7

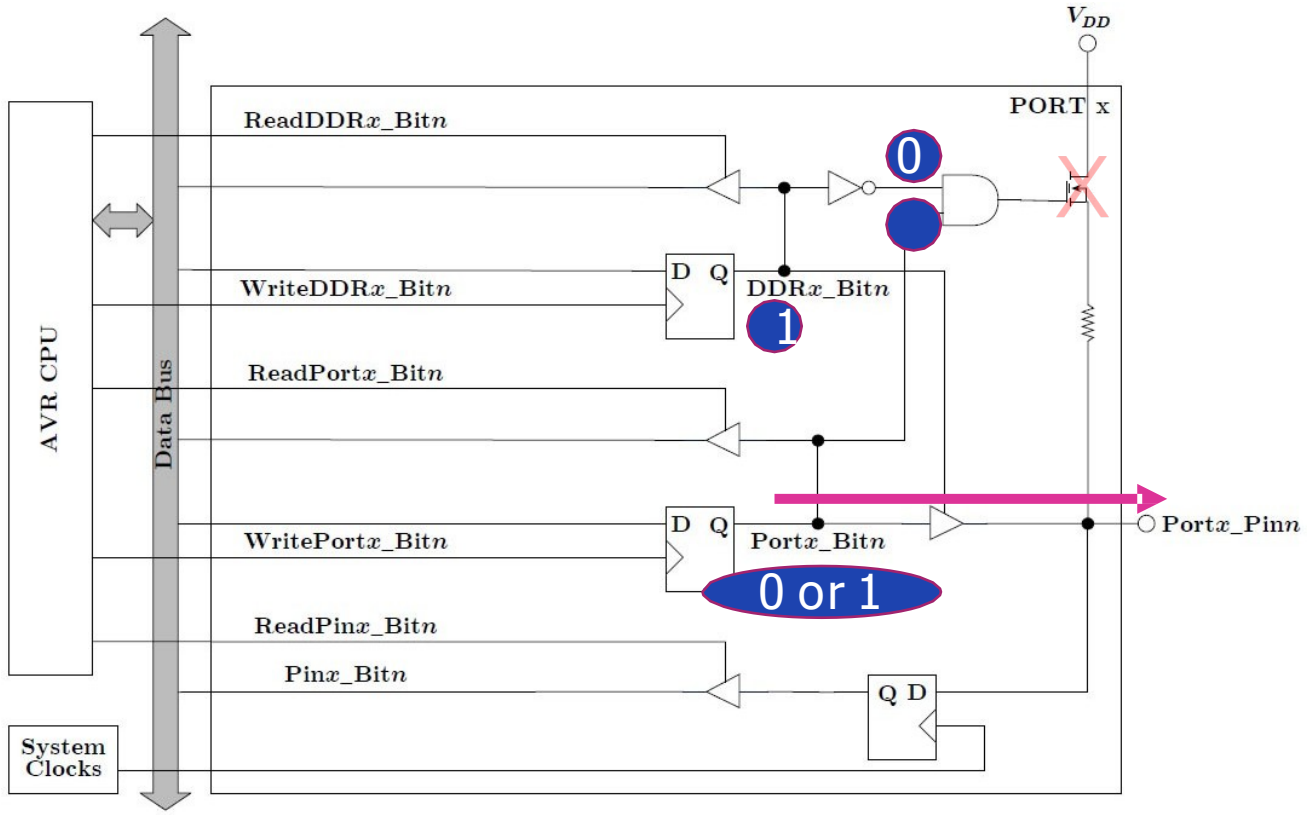
Operation of a Single Port Bit

- Read
- Write
- Input with pullup



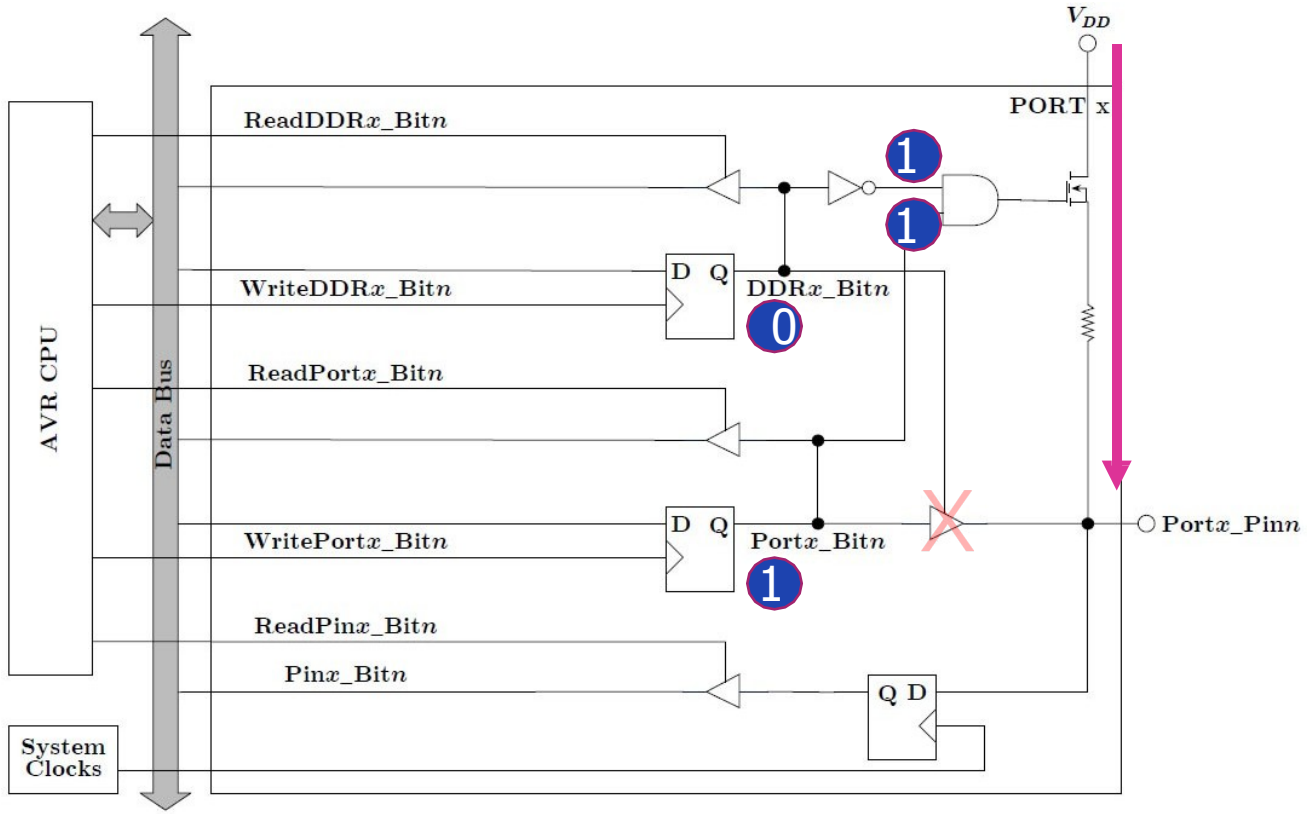
Operation of a Single Port Bit

- Read
- Write
- Input with pullup



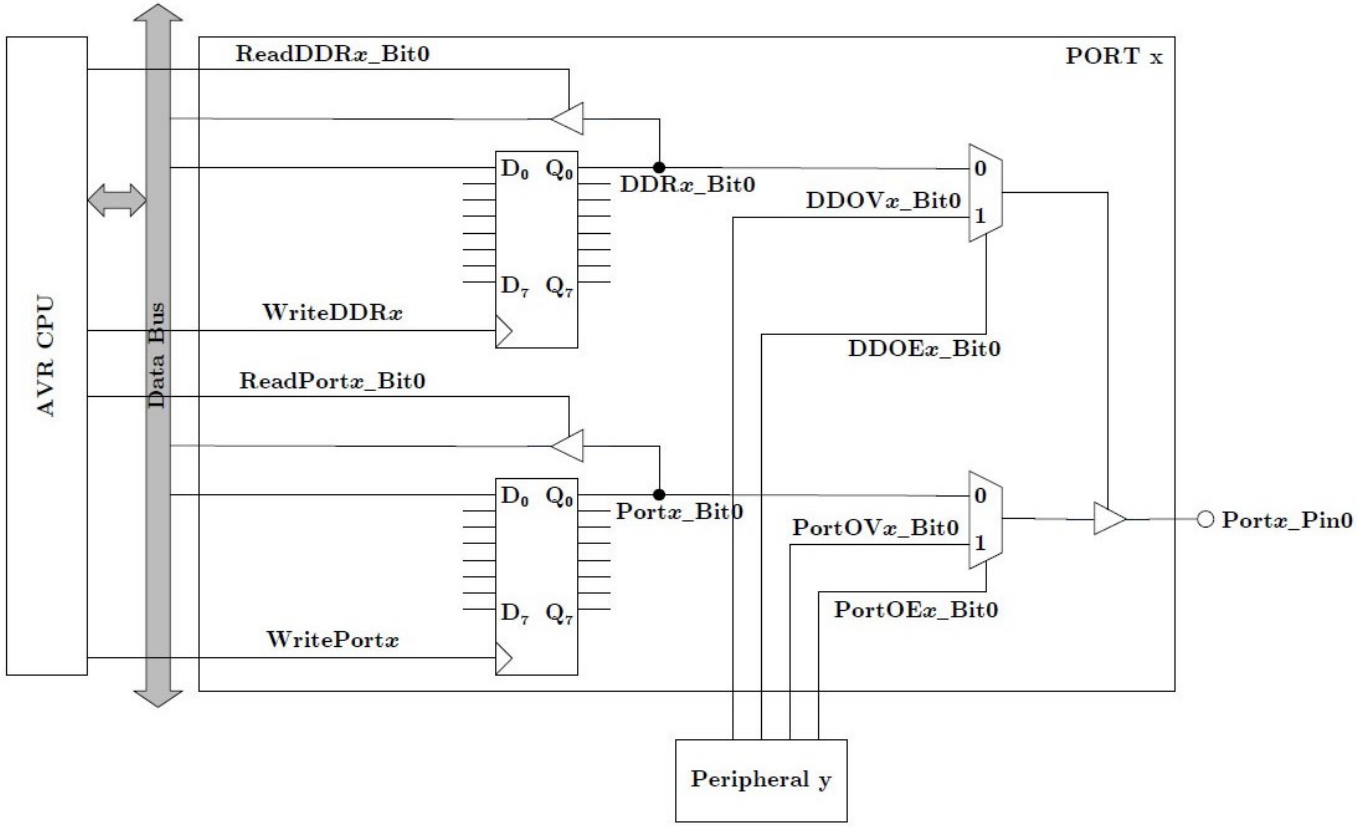
Operation of a Single Port Bit

- Read
- Write
- Input with pullup



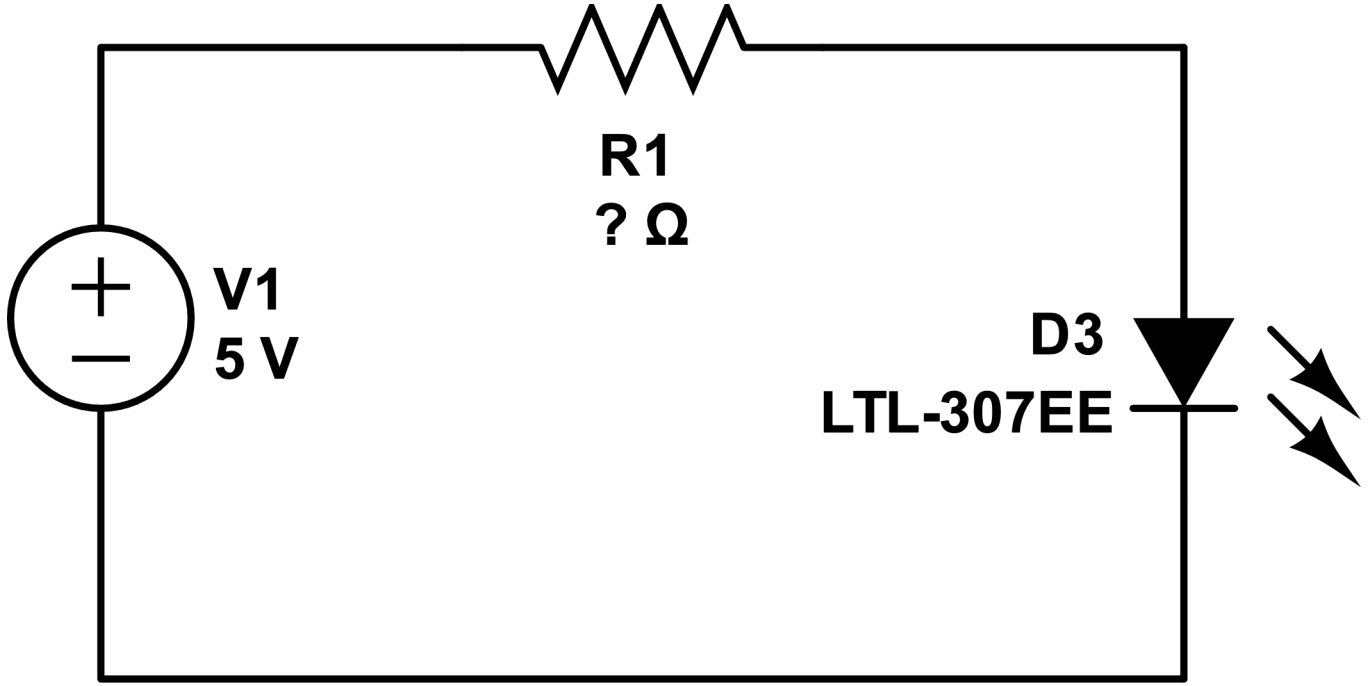
Port Showing Mux for Peripherals

- Peripherals can be enabled through MUX
- The MUX routes the peripheral to the pin, turning off the port functionality



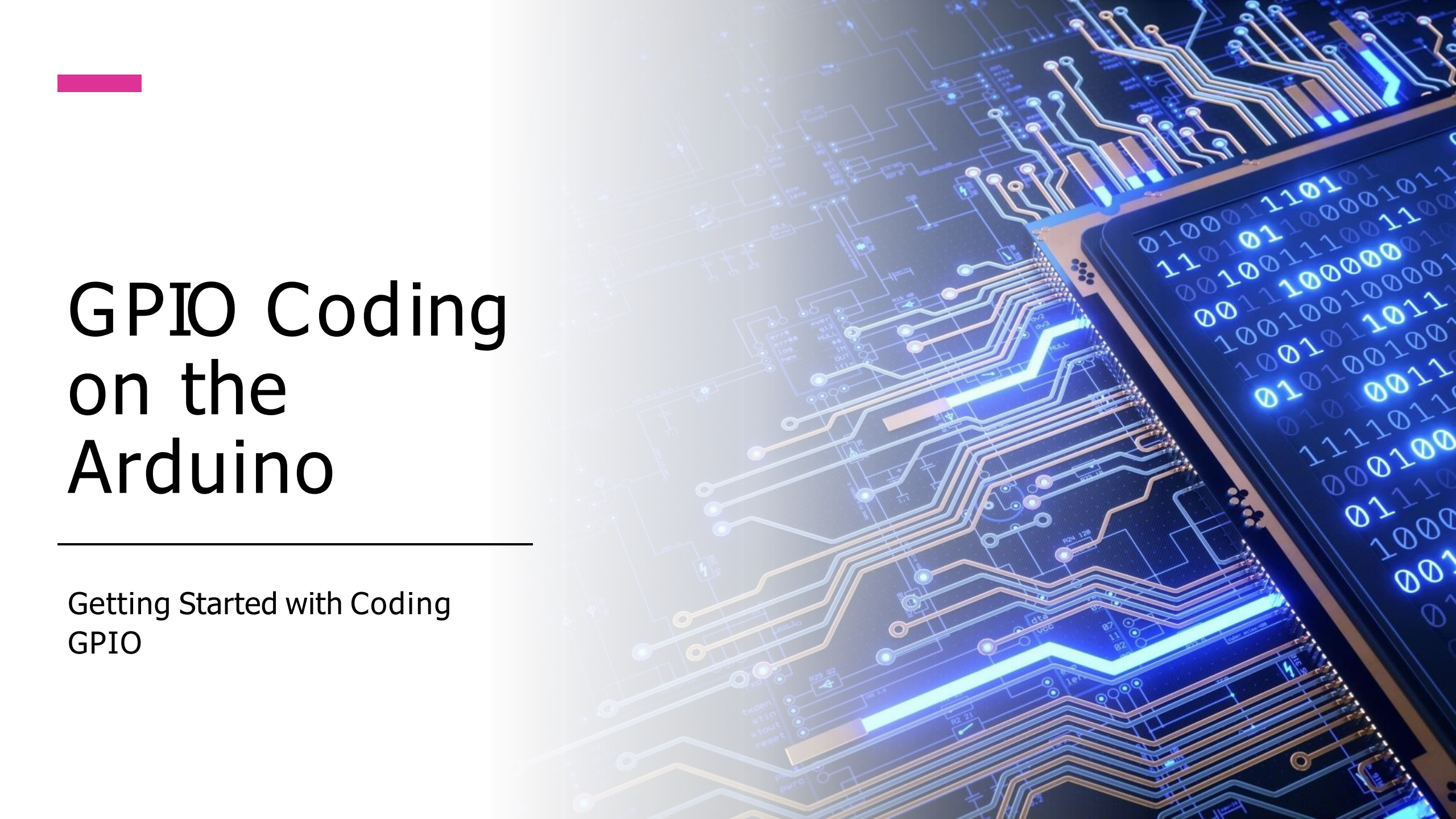
IO Current Limitations

- The AVR MCUs can source and sink up to 40ma, but it is recommended to keep the values around 20ma.
- This is important when connecting circuits to the GPIO pins
- The voltage drop Vd across LEDs varies by color (!)
  - Red may have a drop of 1.8V, while blue might be 3.0V
- Choose R1 so that (V1+Vd)/R1 is no more than 40 ma.



GPIO Coding on the Arduino

Getting Started with Coding GPIO

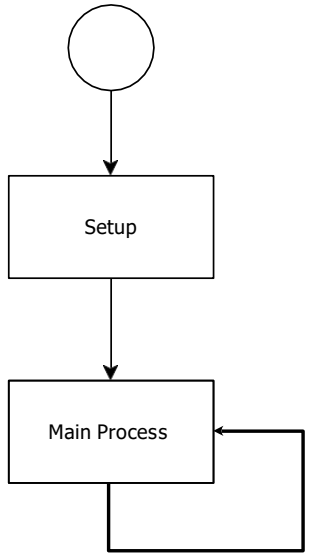


Blink – Hello World for Arduino

- The blink program illustrates the most basic GPIO function
  - Turn on an LED
  - Turn off the LED
- Allows us to explore the Arduino code structure and impress our friends and parents

The Structure of an Arduino Program

- Unlike simple programs that start, process some data, and end, embedded programs typically loop forever
- In the Arduino IDE, the *main* function from C is hidden
- Instead, there are two default functions: *setup* and *loop*




```
void setup() {
  // code in here runs only once
}
void loop() {
  // code in this section runs repeatedly
}
```

GPIO Modes

- Three modes of operation
  - Outputting data over the port
  - Reading input from the **P**ort **IN**put
  - Input with pullup for disconnected input pin
    - When connected to a switch, the pin is effectively disconnected and may return a random value

GPIO MODE	DDR	PORT	PIN
OUTPUT	1	DATA OUT	N/A
INPUT	0	0	DATA IN
INPUT with Pullup	0	1	DATA IN

Setting Modes using Special Registers

- PORTB: Data to be written to the output
- DDRB: Controls whether a pin is an input or output
- PINB: The value present on the input pins

Bit	7	6	5	4	3	2	1	0
0x25	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTB7-0: GPIO data value stored in bit *n*.

Bit	7	6	5	4	3	2	1	0
0x24	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRB7-0: selects the direction of pin *n*. If DDRB*n* is written '1', then PORTB*n* is configured as an output pin. If DDRB*n* is written '0', then PORTB*n* is configured as an input pin.

Bit	7	6	5	4	3	2	1	0
0x23	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Read/Write	R	R	R	R	R	R	R	R
Default	-	-	-	-	-	-	-	-

- PINB7-0: logic value present on external pin *n*.

Register Settings for Port B

Tasks to Blink the LED

- Setup
  - Identify the port and pin where the built-in LED is connected
    - This varies based on the version of the Arduino
    - For the Arduino Mega 2560, this is pin 13, port B pin 7
  - Set DDRB7 to 1 to enable writing
- Loop
  1. Write a 1 to PORTB7 to turn on the LED
  2. Wait a while
  3. Write a 0 to PORTB7 to turn the LED off

Make sure to check the port and pin for different version of Arduino.
For example, for Arduino UNO the built in LED is connected to Port B Pin 5.

Setting the DDR for Port B Bit 7 for Output

- Using the Wired library, just use
  `pinMode(LED_BUILTIN, OUTPUT);`
- To use registers directly,
  - Look up memory address of DDRB –0x24 (see page 108 in Russell)
  - Declare a pointer to the register

    ```
    unsigned char* ddr_b = (unsigned char*) 0x24
    ```
  - Modify bit 7 using a bit mask

    ```
    *ddr_b |= 0x01 << 7;
    ```

Output mode:
DDR = 1
Port = DATA OUT
Pin = N/A

```
*ddr_b = b00010011
 0x01 << 7 =  0x80  = b1000 0000

   00010011
|= 10000000
   10010011
```

Writing Data to Port B Bit 7

Output mode:
DDR = 1
Port = DATA OUT
Pin = N/A

- Declare a pointer to the register
  - `unsigned char* port_b = (unsigned char*) 0x25`
- Write a 1 or 0 to bit 7 of the port to turn on or off the LED

1: `*port_b |= 0x01 << 7`

0: `*port_b &= ~(0x01 << 7)`

The Completed Blink Program using Registers

```
// Define Port B register pointers as global variables
unsigned char* port_b = (unsigned char*) 0x25;
unsigned char* ddr_b  = (unsigned char*) 0x24;

void setup()
{
 //set PB7 to OUTPUT
 *ddr_b |= 0x01 << 7;
}

void loop()
{
 // drive PB7 HIGH
 *port_b |= (0x01 << 7);
 // wait 500ms
 delay(500); // this is a built-in Wired library function
 // drive PB7 LOW
 *port_b &= ~(0x01 << 7);
 // wait 500ms
 delay(500);
}
```

Output mode:
DDR = 1
Port = DATA OUT
Pin = N/A

Frequency

In general, Pulse Width Modulation (PWM) is a modulation technique used to encode a message into a pulsing signal.
A PWM is comprised of two key components: **frequency** and **duty cycle**.

The PWM frequency dictates how long it takes to complete a single cycle (period) and how quickly the signal fluctuates from high to low.

The duty cycle determines how long a signal stays high out of the total period. The duty cycle is represented in percentage.

For a single cycle,
One cycle period = how long it was HIGH and how long it was LOW
Frequency = 1/Cycle period

```
void loop()
{
  // drive PB7 HIGH
  *port_b |= (0x01 << 7);
  // wait 500ms
  delay(500); // this is a built-in Wired library function
  // drive PB7 LOW
  *port_b &= ~(0x01 << 7);
  // wait 500ms
  delay(500);
}
```

It is HIGH for 500 ms

It is LOW for 500 ms

Frequency = 1/ (500 ms + 500 ms)
= 1/ (0.5s + 0.5s) = 1 Hz

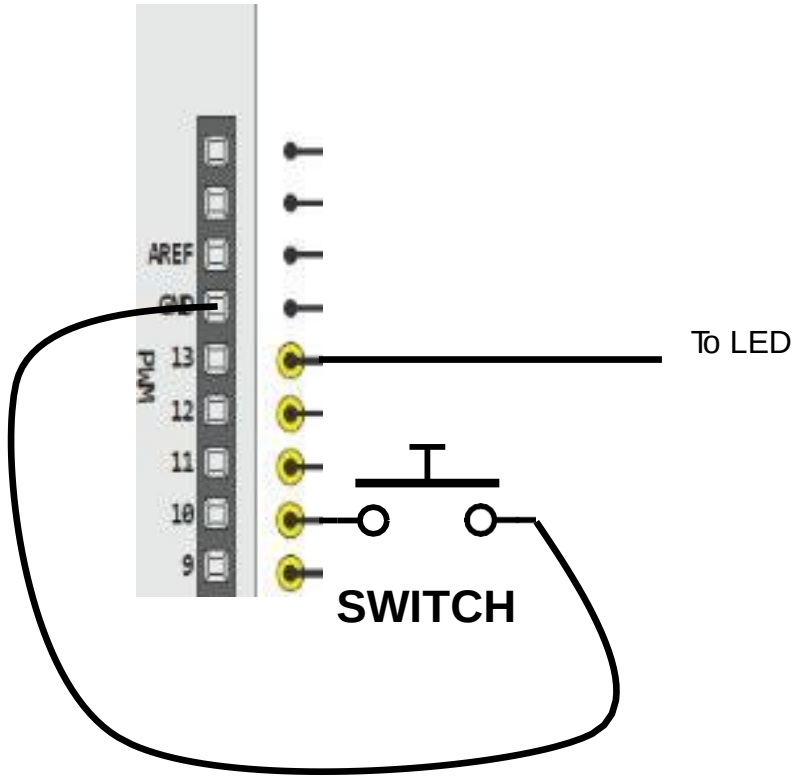
https://www.tinkercad.com/things/9164cnMzzgi-gpiobuiltinled/editel?sharecode=QV5no2qN_-PxSNYzGzpQiYKYHgRO9Q0TqsPozEkEe1E

Some Alternate Syntax for Setting Values

- 0x01 << 7 can be written as
  - 0b10000000
  - 0x80
  - 128 in decimal (not recommended)
- ~(0x01 << 7) can be written as
  - 0b01111111
  - 0x7F

Blink with a Button

- Goal: turn LED off when button connected to PB4 is pressed
- Requires reading from an input and setting modifying the LED port value

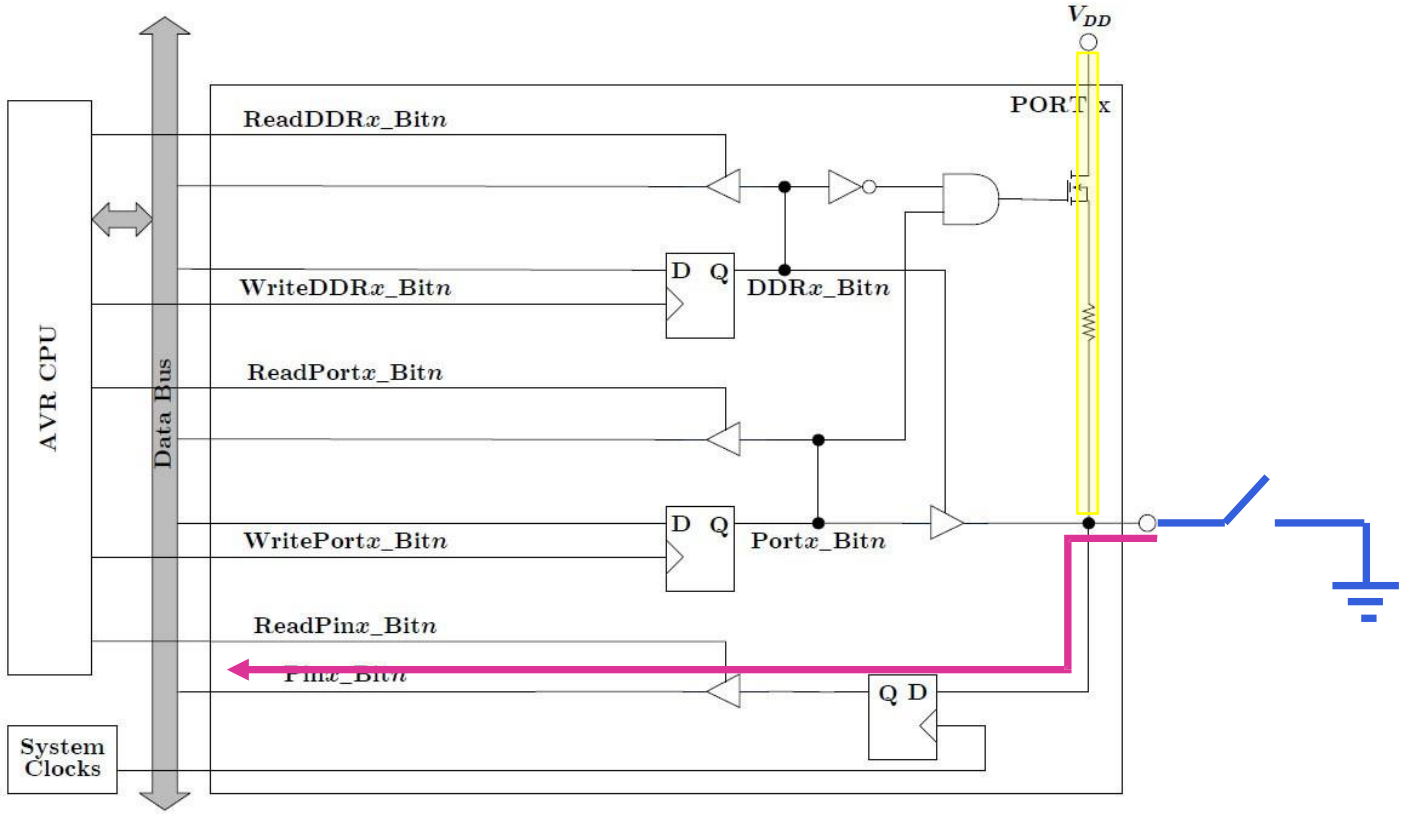


The GPIO Modes Revisited

- The three modes of GPIO are
  - Output
  - Input
  - Input with pullup
- Modes are set using the data direction registers (DDR) and the port register

Operation of a Single Port Bit

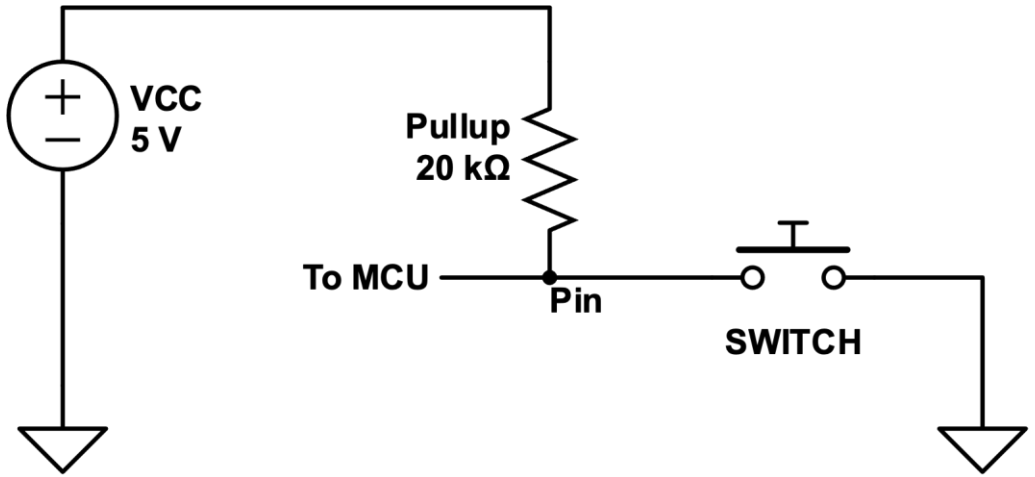
- With pullup disabled and no connection to the pin, the state is unpredictable
- When the switch is open, the pin is effectively disconnected
  - Reading the pin value can result in random values (0 or 1)
- Enabling the pull-up sets the pin to high by default
- The switch then pulls the pin low

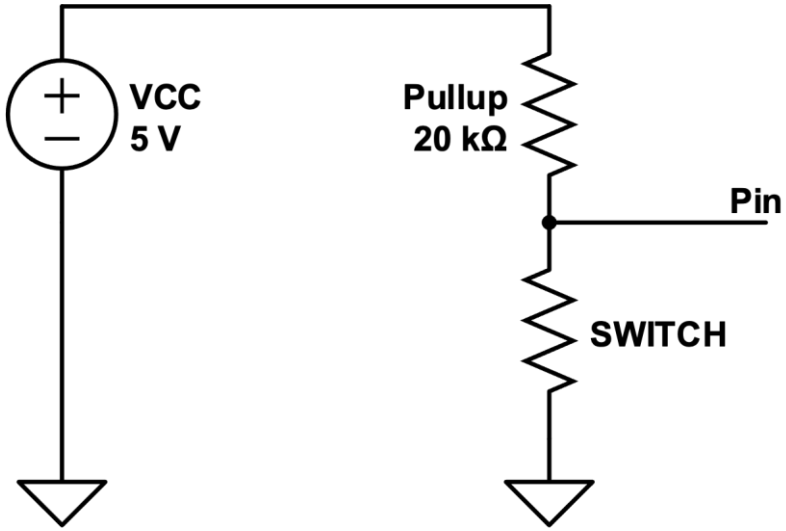


Why is Pin = 0 when Switch Closed?

- Think of resistor and switch as voltage divider
- When the switch is open, it is an infinite resistor
- When the switch is closed, it is a resistor with value 0 Ohms

$$V_{pin} = V_{CC} \frac{R_{switch}}{R_{switch} + R_{pullup}}$$





Blink with Button Code

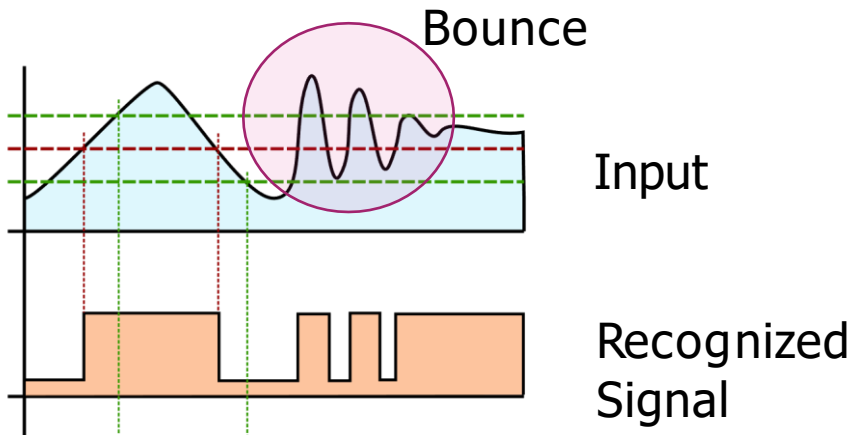
Output mode:
DDR = 1 Port = DATA OUT Pin = N/A
Input mode:
DDR = 0 Port = 0 Pin = Data IN
Input with pullup mode:
DDR = 0 Port = 1 Pin = Data IN

```
// Define Port B Register Pointers
unsigned char* port_b = (unsigned char*) 0x25;
unsigned char* ddr_b  = (unsigned char*) 0x24;
volatile unsigned char* pin_b  = (unsigned char*) 0x23;
void setup()
{
*ddr_b &= 0xEF; //set PB4 to INPUT
 *ddr_b |= 0x80; //set PB7 to OUTPUT
//  // enable the pullup resistor on PB4
   *port_b |= 0x10;
}
```

```
void loop()
{
 // if the pin is high
 if(*pin_b & 0x10)    // PB4 is the 5th pin!
 {
   *port_b |= (0x01 << 7); // set PORTB7 to 1
 }
 // if the pin is low
 else
 {
   *port_b &= ~(0x01 << 7); // set PORTB7 to 0
 }
 delay(100);
}
```

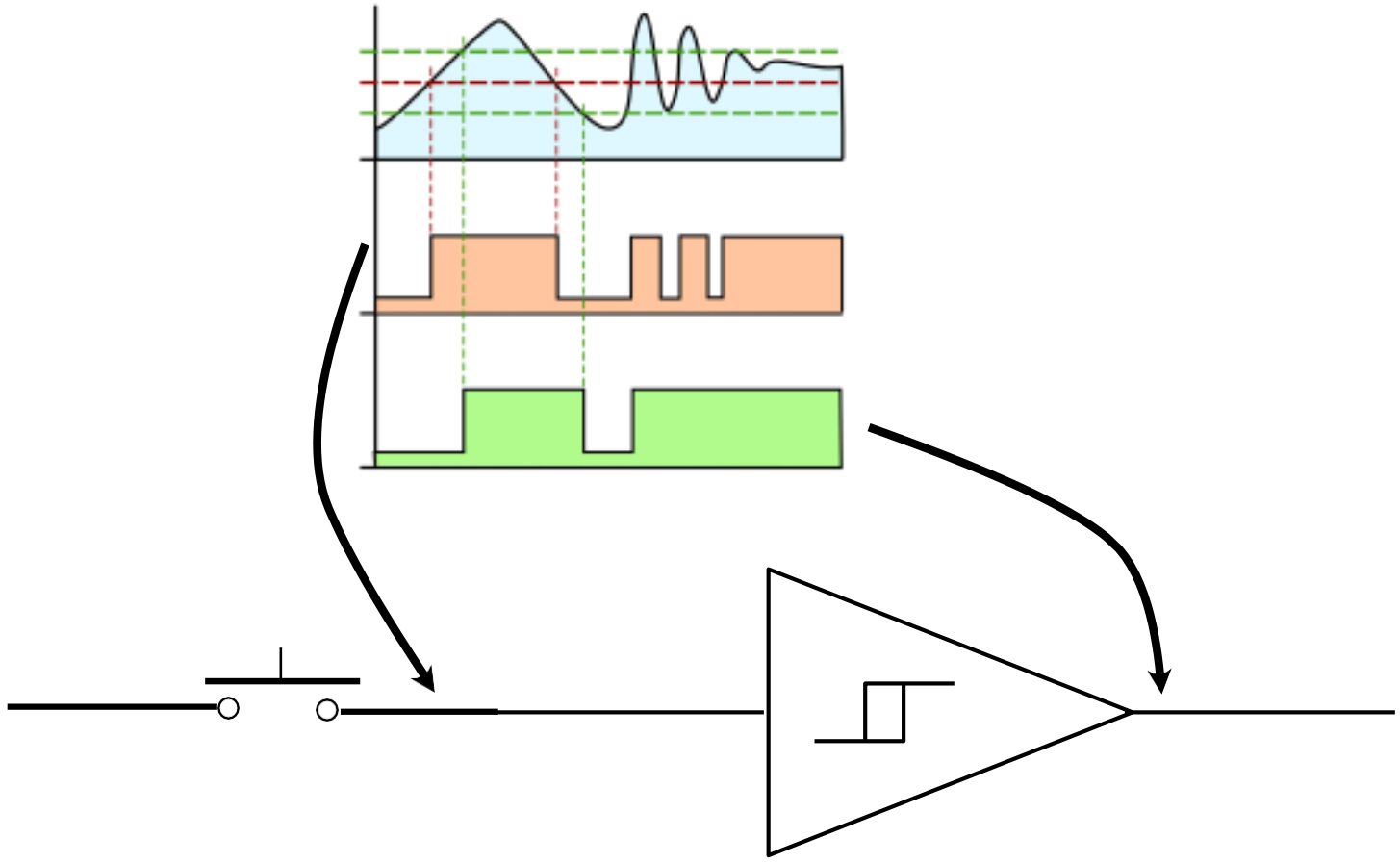
Debouncing

- Electronic switches exhibit *bounce*
  - The switch may make and break contact multiple times before the signal stabilizes
- *Debouncing* refers to using either hardware or software to remove the unwanted noise
  - Bouncing of a switch is effectively the same as noise on the line



Debouncing

- Hardware: Use a filter or a Schmitt Trigger



Debouncing in Software

- Strategy
  - Most contact bounces have a duration of < 10mSec
  - Read the input once, wait ~15mSec, read the input again
  - If the two reads are the same, the reading is valid
- Drawbacks
  - Bounce durations can vary on larger mechanisms (use a scope to find the duration for a particular application)
  - The processor is idle for the wait duration
  - Increased complexity for multiple inputs