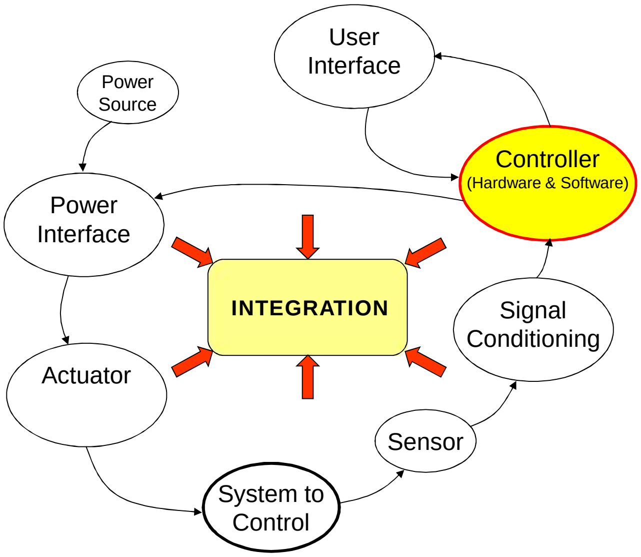
Microcontroller Fundamentals

Learning Objectives

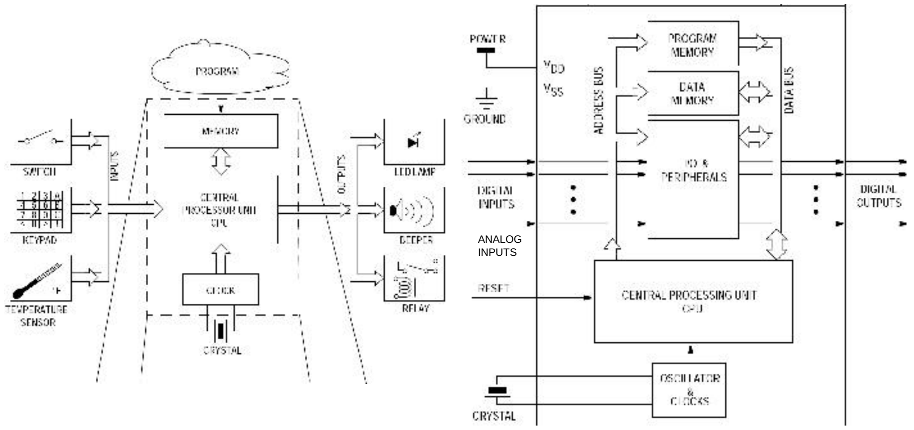
- Explain the general architecture of a microcontroller
- List the key features of the ATmega328 microcontroller
- Explain the features and elements of the Arduino and Spartronics Experimenter Shield (SES)
- Explain the concepts of microcontroller pins as inputs and outputs
- Convert between binary and hexadecimal digits

Mechatronics Concept Map



BJ Furman 22JAN11

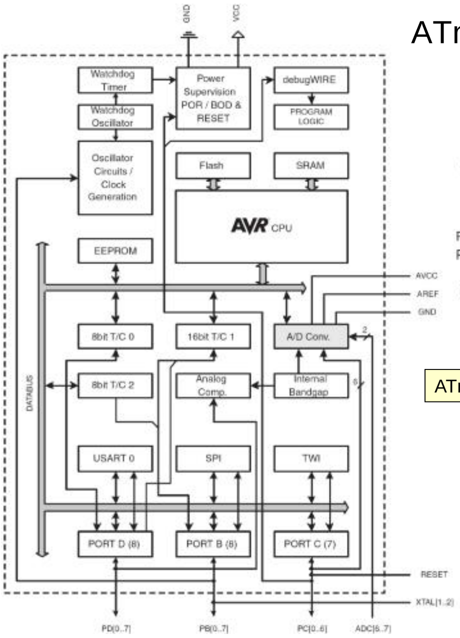
What is a Microcontroller?

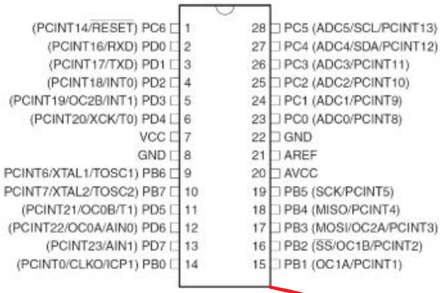


What is the difference between a ‘*Digital* Input’ and an ‘*Analog* Input’?

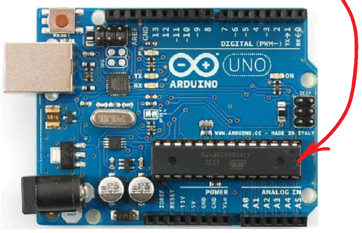
http://www.freescale.com/files/microcontrollers/doc/ref_manual/M68HC05TB.pdf

ATmega328 Internal Architecture



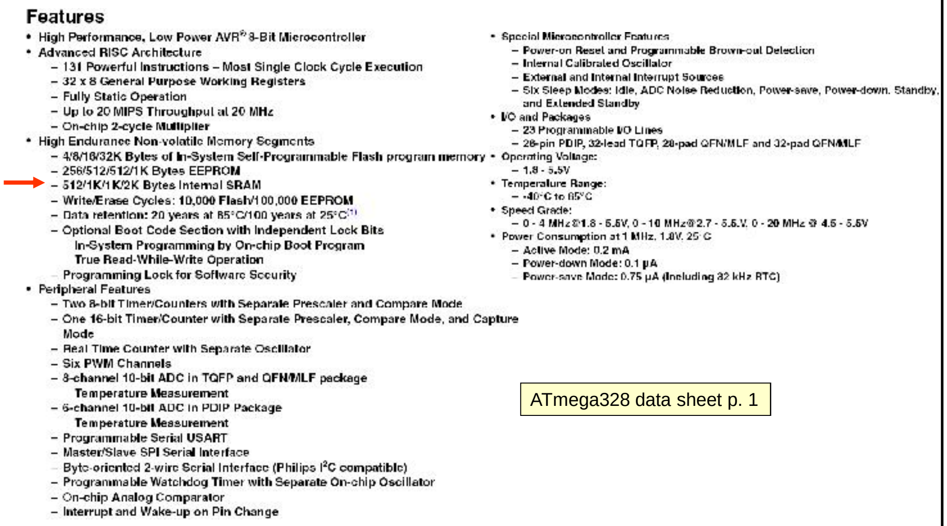


ATmega328 data sheet pp. 2, 5



http://www.adafruit.com/index.php?main_page=popup_image&pID=50

ATmega328 Features



Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1K Bytes EEPROM
  - 512/1K/1K/2K Bytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^{2}C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5.V, 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

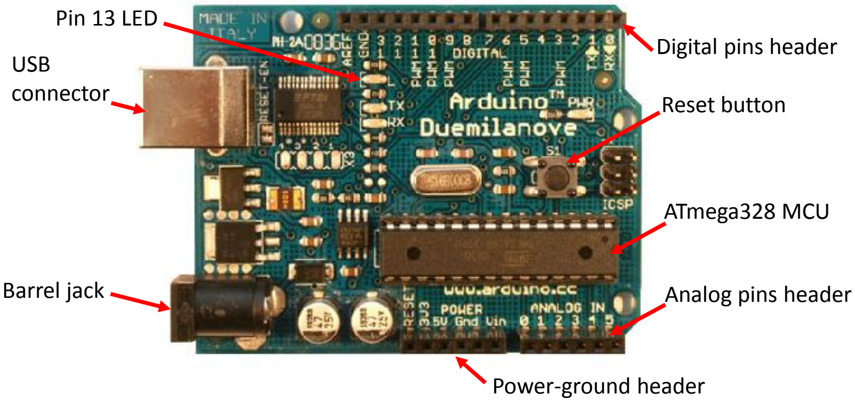
ATmega328 data sheet p. 1

http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf

Arduino Duemilanove

http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove

See the handout: **Arduino_ATmega328_pin_mapping_and_schematic**


Pin 13 LED

USB connector

Barrel jack

Digital pins header

Reset button

ATmega328 MCU

Analog pins header

Power-ground header

http://arduino.cc/en/uploads/Main/ArduinoDuemilanove.jpg

Arduino Uno R3

ATmega16u2 replaces FT232RL for USB-serial communication

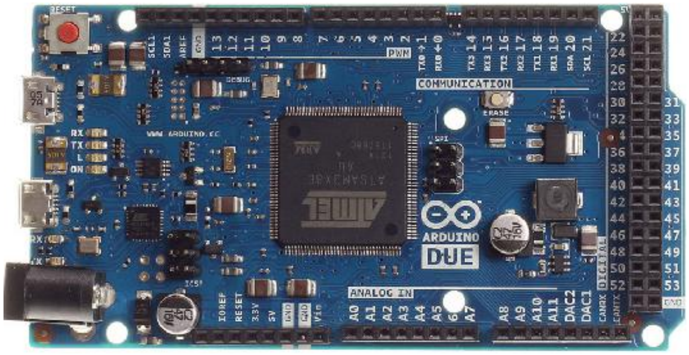
http://www.adafruit.com/index.php?main_page=popup_image&pID=50

See: http://learn.adafruit.com/arduino-tips-tricks-and-techniques/arduino-uno-faq

Arduino Due

Note: **3.3 V** !!

Atmel SAM3X8E processor (32 bit ARM Cortex M3 architecture, 84MHz)



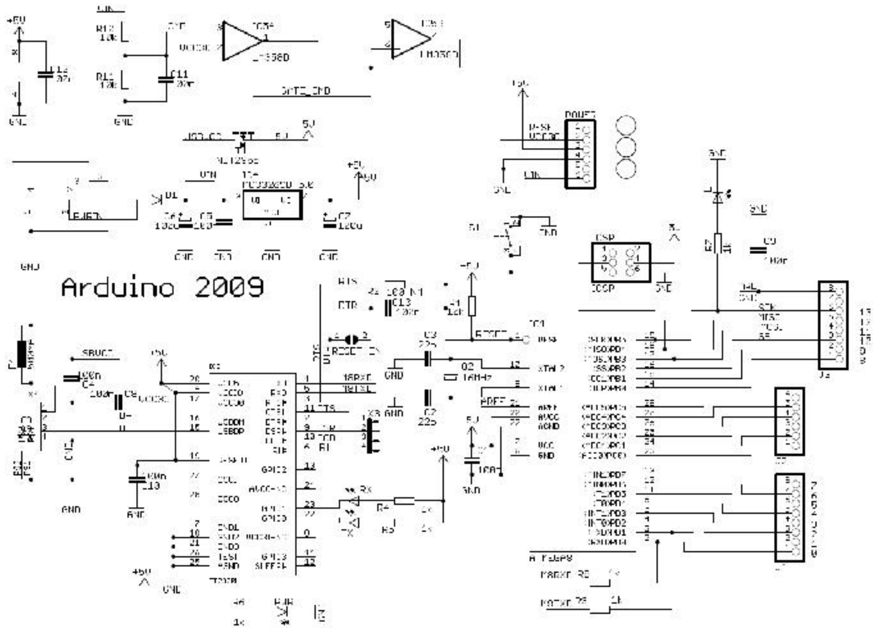
http://www.adafruit.com/index.php?main_page=popup_image&pID=1076

See: http://arduino.cc/en/Main/ArduinoBoardDue

Arduino Duemilanove/Uno Features

Microcontroller	ATmega168/328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz

http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove



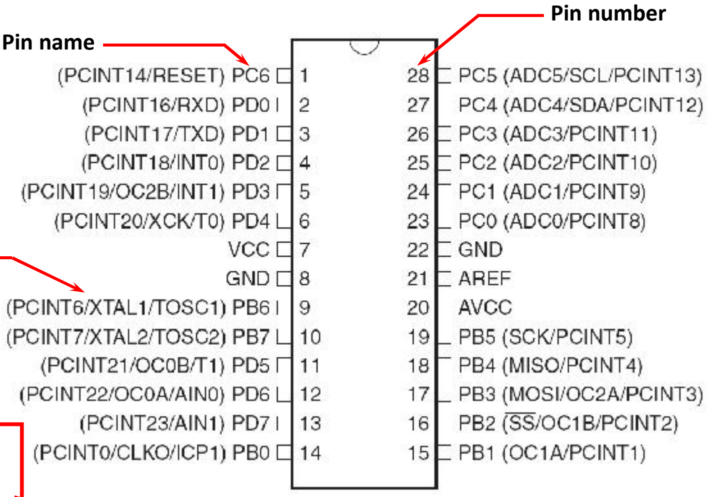
http://arduino.cc/en/uploads/Main/arduino-duemilanove-schematic.pdf

ATmega328 Microcontroller

Pin number

Pin name

Special function



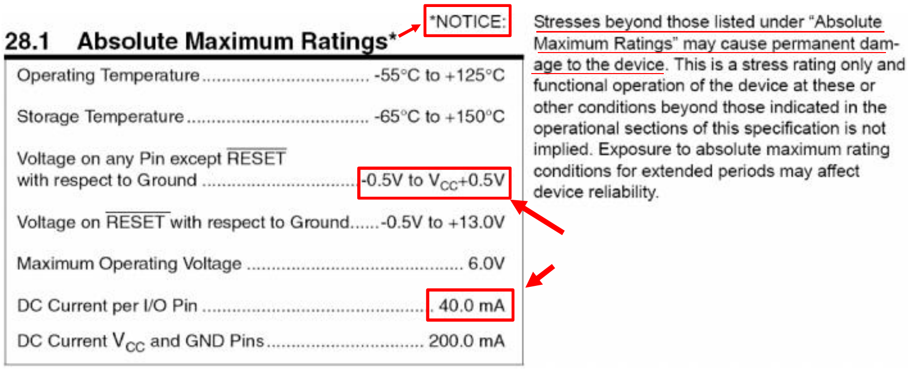
Pin name	Pin	Pin	Pin name
(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 ($\overline{\text{SS}}$/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

Note the *limitations*!

p. 316

Source:http://www.atmel.com/dyn/products/product_card.asp?PN=ATmega328P

Absolute Maximums



28.1 Absolute Maximum Ratings*

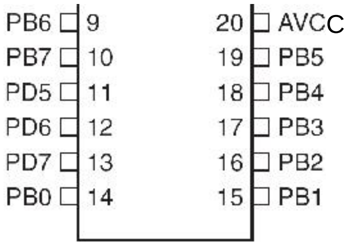
Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ATmega328 data sheet p. 316

Microcontroller Ports and Pins

- The communication channels through which information flows into or out of the microcontroller
  - Ex. PORTB
    - Pins PB0 – PB7
      - May not be contiguous
      - Often bi-*directional*

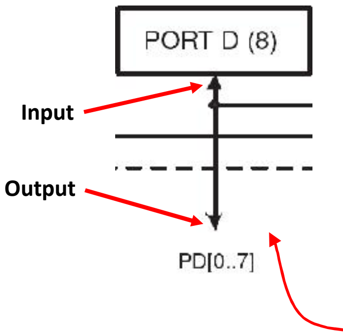


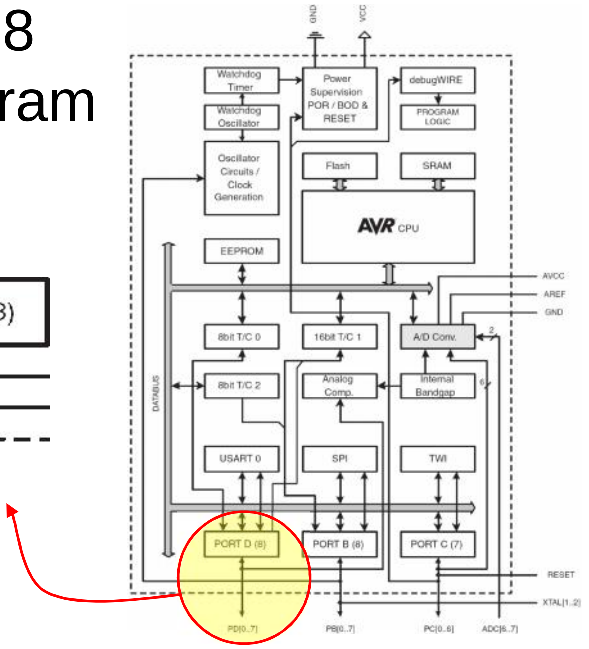
See next slides!

Port Pin Data Directionality

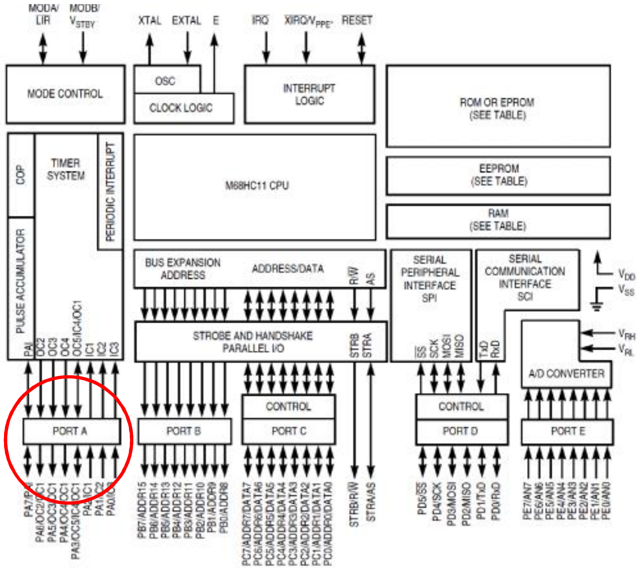
- Input
  - □ When you want to take information from the external world (sensors) ***into*** the MCU
- Output
  - □ When you want to change the state of something ***out*****side** the MCU (turn a motor on or off, etc.)
- Pins default to input direction on power-up or reset
- Your program can set or change the directionality of a pin at any time

ATmega328 Block Diagram





M68HC11 microcontroller



Setting the Pin Data Direction

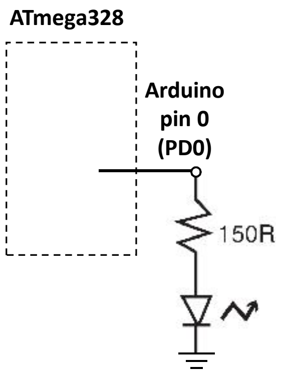
- Arduino
  - pinMode(*pin_no., dir*)
    - Ex. Make Arduino pin 3 (PD3) an *output*
      - **`pinMode(3, OUTPUT);`**
      - **`pinMode(PIN_D3, OUTPUT); // with me106.h`**
  - Note: one pin at a time
    - Suppose you wanted Arduino pins 3, 5, and 7 (PD3, PD5, and PD7) to be outputs?
    - Is there a way to make them all outputs at the same time?
      - Yes! Answer coming later…

Pin Voltages

- Microcontrollers are fundamentally ***digital*** devices. For digital IO pins:
  - □ Information is ‘coded’ in two discrete states:
    - HIGH or LOW (logic: 1 or 0)
    - Voltages
      - □ TTL
        - 5 V (for HIGH)
        - 0 V (for LOW)
      - □ 3.3 V CMOS
        - 3.3 V (for HIGH)
        - 0 V (for LOW)

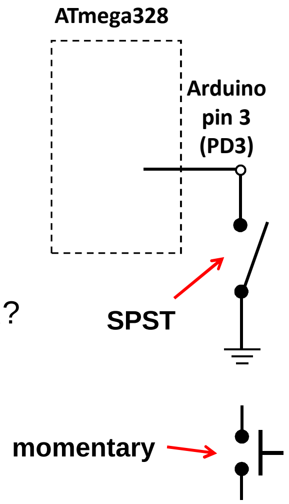
Pin Used as an Output

- Turn on an LED, which is connected to pin Arduino pin 0 (PD0) (note the resistor!)
  - □ What should the data direction be for pin 0 (PD0)?
    - **`pinMode(____, ____);`**
  - □ Turn on the LED
    - **`digitalWrite(PIN_LED,HIGH);`**
  - □ Turn off the LED
    - **`digitalWrite(PIN_LED,LOW);`**



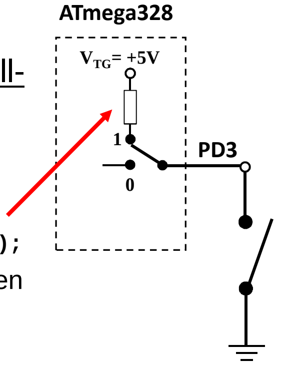
Pins as Inputs and Pull-up Resistors - 1

- Using a switch as a sensor
  - Ex. Seat belt sensor
  - Detect the switch ***state***
    - What should the data direction be for Arduino pin 3 (PD3)?
    - **`pinMode(____, ____);`**
    - What will the voltage be on PD3 when the switch is closed?
    - What will the voltage be on PD3 when the switch is open?
      - Indeterminate!



Pins as Inputs and Pull-up Resistors - 2

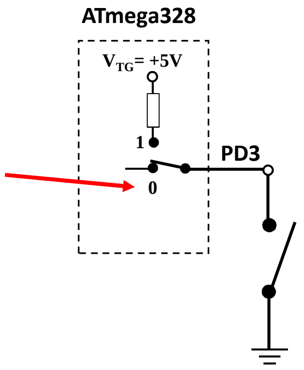
- Switch as a sensor, cont.
  - Make the voltage on the pin *determinate* by turning on the <u>pull-up</u> resistor for PD3
    - Assuming PD3 is an input:
      - **`digitalWrite(PIN_SWITCH,HIGH);`** turns on the “pull-up” resistor
      - **`pinMode(PIN_SWITCH,INPUT_PULLUP);`**
    - What will the voltage on PD3 be when the switch is open?
      - V_{TG}
    - What will the voltage on PD3 be when the switch is closed?



Pins as Inputs and Pull-up Resistors - 3

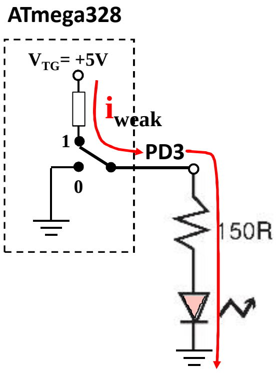
- Switch as a sensor, cont.
  - □ To turn *off* the pull-up resistor
    - Assuming PD3 is an input:

`digitalWrite(PIN_SWITCH,LOW);`
turns the “pull-up” resistor off

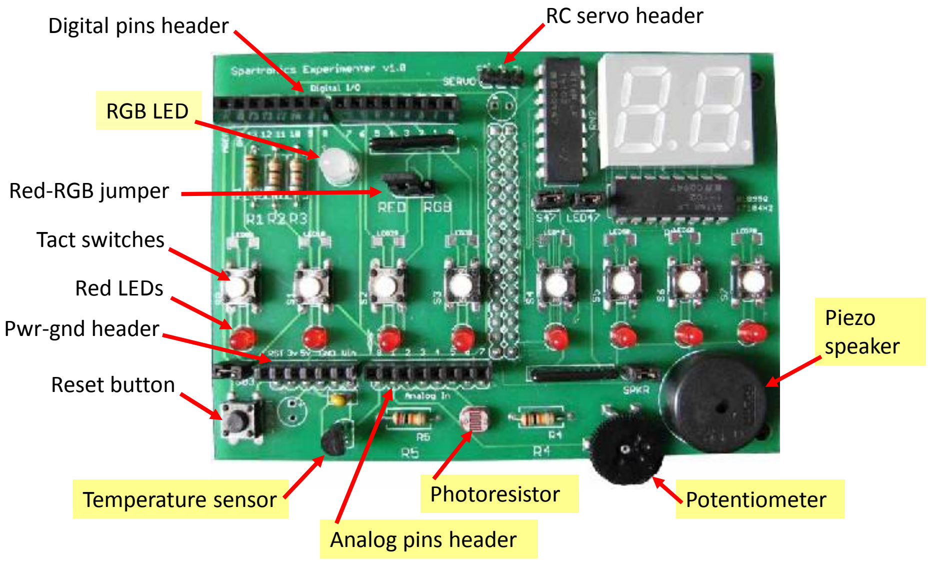


Pins as Inputs and Pull-up Resistors - 4

- Possibility of ‘weak drive’ when pull-up resistor is turned on
  - □ Pin set as an *input* with a pull-up resistor turned on can source a small current
    - Remember this!

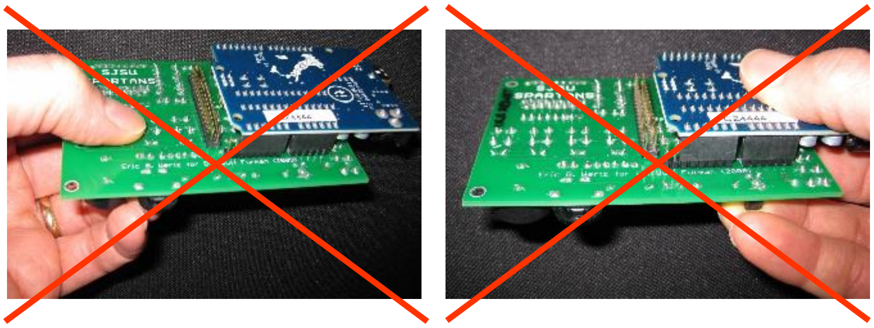


Spartronics Experimenter Shield



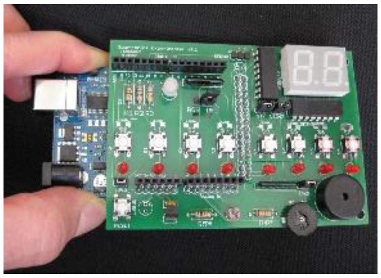
Handling the Arduino - How *NOT* to Do It!

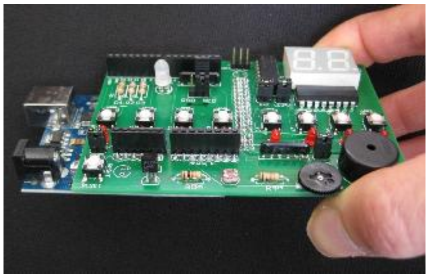
Improper Handling - ***NEVER***!!!



Handling the Arduino - The Proper Way

Proper Handling - by the ***edges***!!!

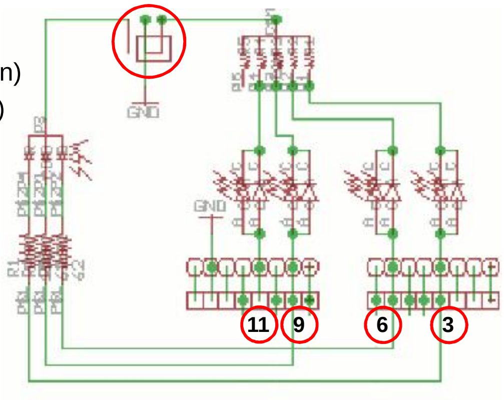


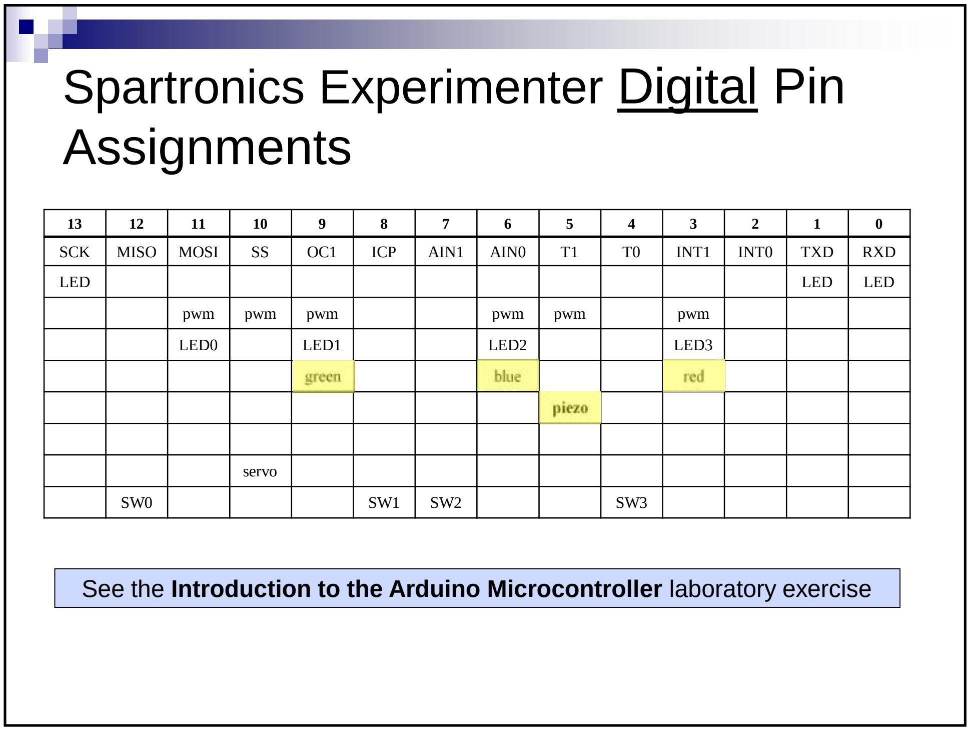


Spartronics Experimenter LED Pinout

- Pin and LED map
  - □ 11 - LED0 (red)
  - □ 9 - LED1 (red) or RGB (green)
  - □ 6 - LED2 (red) or RGB (blue)
  - □ 3 - LED3 (red) or RGB (red)
  - □ 13 - LED on Arduino

Jumper determines whether pins map to red LEDs or the RGB

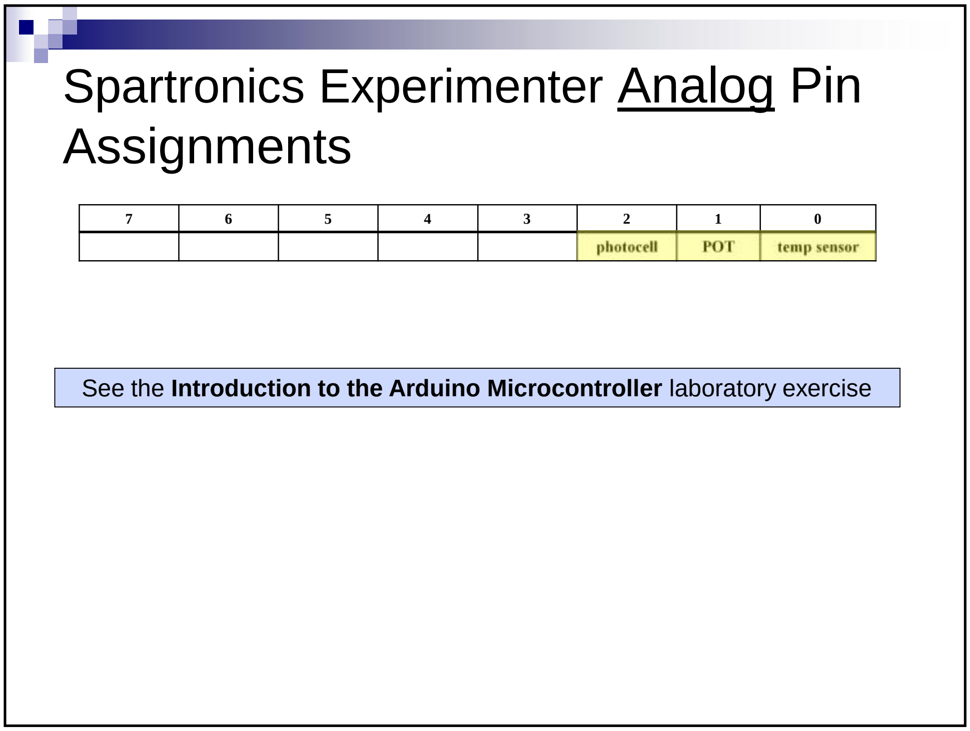




Spartronics Experimenter Digital Pin Assignments

13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCK	MISO	MOSI	SS	OC1	ICP	AIN1	AIN0	T1	T0	INT1	INT0	TXD	RXD
LED												LED	LED
		pwm	pwm	pwm			pwm	pwm		pwm			
		LED0		LED1			LED2			LED3			
				green			blue			red			
								piezo					
			servo										
	SW0				SW1	SW2			SW3				

See the **Introduction to the Arduino Microcontroller** laboratory exercise



Spartronics Experimenter Analog Pin Assignments

7	6	5	4	3	2	1	0
					photocell	POT	temp sensor

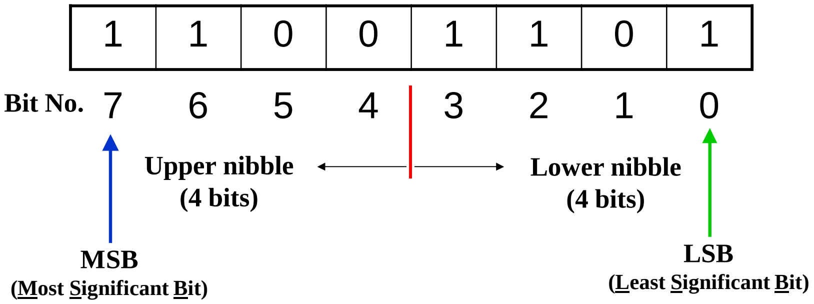
See the **Introduction to the Arduino Microcontroller** laboratory exercise

Binary and Hexadecimal Numbers - 1

- Microcontrollers are fundamentally digital (as opposed to ‘analog’) and use ***binary*** logic
  - Two states: high and low, 1 or 0, on or off
    - Often 5V or 0V
  - One binary digit is called a ***bit***
    - It can take on two possible states: 1 or 0
  - Eight binary digits are called a ***byte***
  - Four binary digits are called a ***nibble***

Binary and Hexadecimal Numbers - 2

- Byte and bits



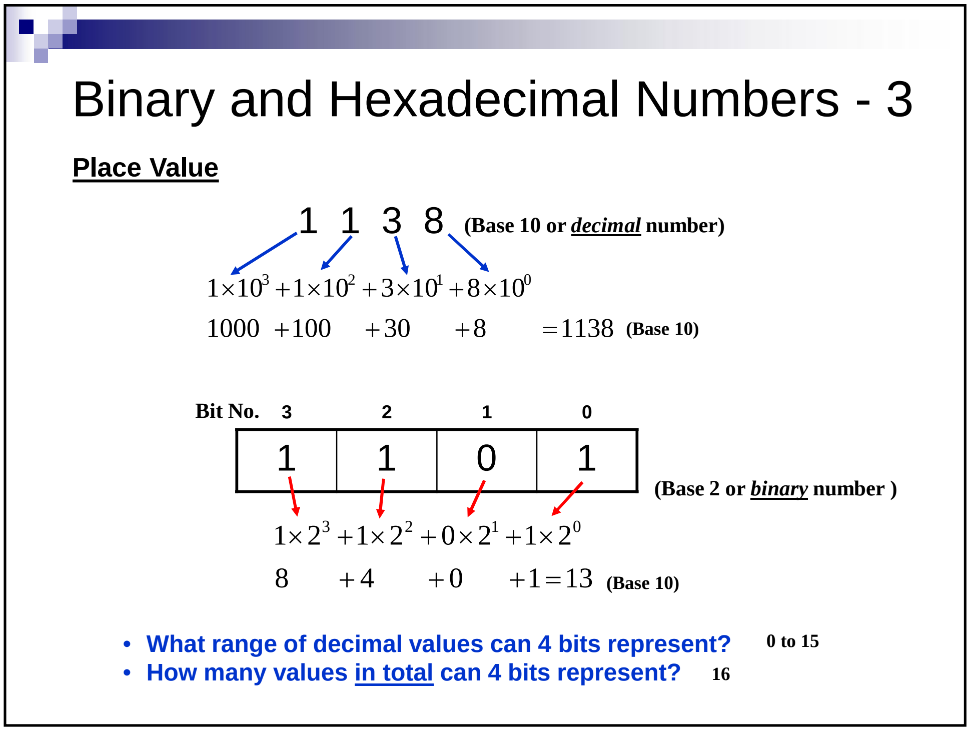
	1	1	0	0	1	1	0	1
Bit No.	7	6	5	4	3	2	1	0

Upper nibble (4 bits)

Lower nibble (4 bits)

MSB (Most Significant Bit)

LSB (Least Significant Bit)



Binary and Hexadecimal Numbers - 3

Place Value

1 1 3 8 **(Base 10 or *decimal* number)**

$$1\times10^{3}+1\times10^{2}+3\times10^{1}+8\times10^{0}$$

$$1000\;+100\;+30\;+8\;=1138$$ **(Base 10)**

Bit No. **3** **2** **1** **0**

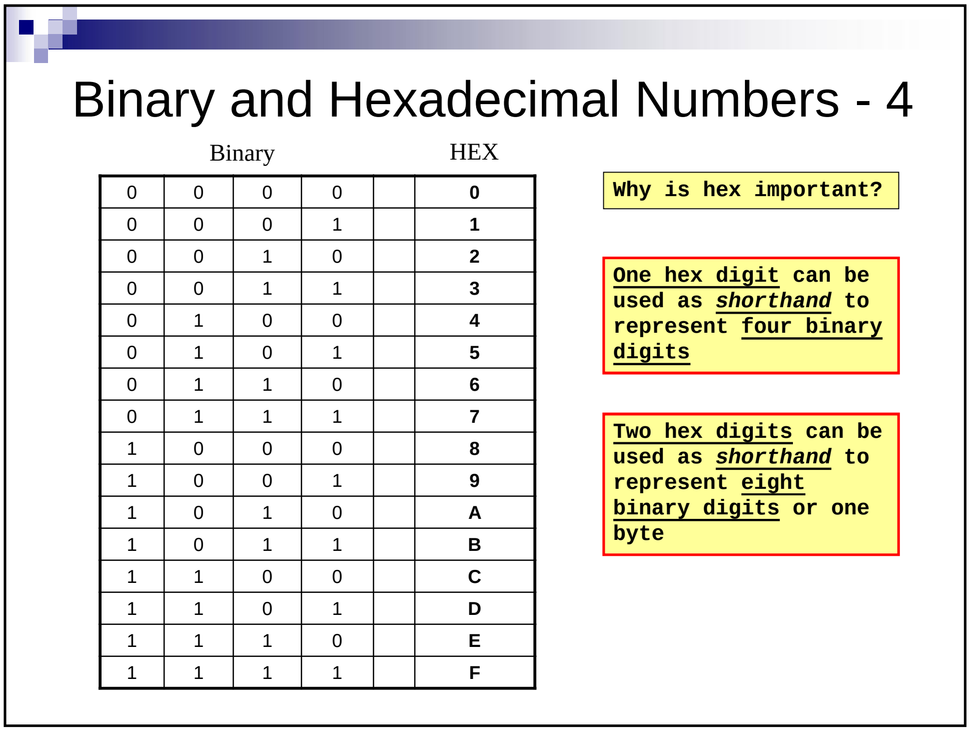
1	1	0	1

(Base 2 or *binary* number)

$$1\times2^{3}+1\times2^{2}+0\times2^{1}+1\times2^{0}$$

$$8\;+4\;+0\;+1=13$$ **(Base 10)**

- **What range of decimal values can 4 bits represent?** **0 to 15**
- **How many values in total can 4 bits represent?** **16**



Binary and Hexadecimal Numbers - 4

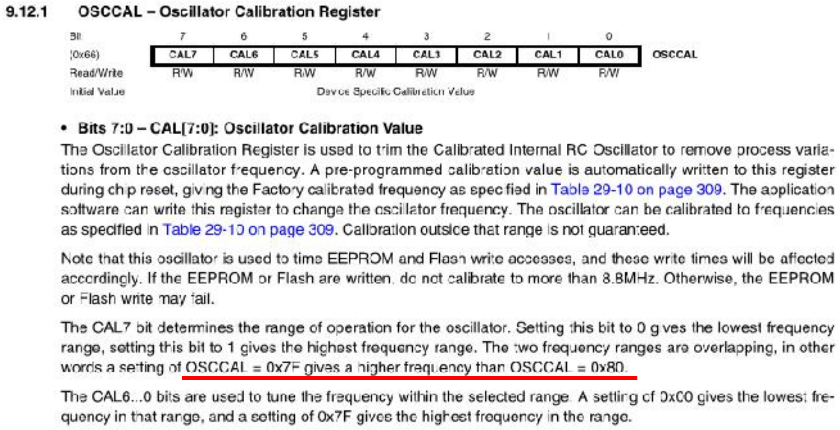
Binary					HEX
0	0	0	0		**0**
0	0	0	1		**1**
0	0	1	0		**2**
0	0	1	1		**3**
0	1	0	0		**4**
0	1	0	1		**5**
0	1	1	0		**6**
0	1	1	1		**7**
1	0	0	0		**8**
1	0	0	1		**9**
1	0	1	0		**A**
1	0	1	1		**B**
1	1	0	0		**C**
1	1	0	1		**D**
1	1	1	0		**E**
1	1	1	1		**F**

Why is hex important?

One hex digit can be used as *shorthand* to represent four binary digits

Two hex digits can be used as *shorthand* to represent eight binary digits or one byte

Using Hex Values



9.12.1 **OSCCAL – Oscillator Calibration Register**

Bit	7	6	5	4	3	2	1	0	
(0x66)	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	Device Specific Calibration Value								

- **Bits 7:0 – CAL[7:0]: Oscillator Calibration Value**

The Oscillator Calibration Register is used to trim the Calibrated Internal RC Oscillator to remove process variations from the oscillator frequency. A pre-programmed calibration value is automatically written to this register during chip reset, giving the Factory calibrated frequency as specified in Table 29-10 on page 309. The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to frequencies as specified in Table 29-10 on page 309. Calibration outside that range is not guaranteed.

Note that this oscillator is used to time EEPROM and Flash write accesses, and these write times will be affected accordingly. If the EEPROM or Flash are written, do not calibrate to more than 8.8MHz. Otherwise, the EEPROM or Flash write may fail.

The CAL7 bit determines the range of operation for the oscillator. Setting this bit to 0 gives the lowest frequency range, setting this bit to 1 gives the highest frequency range. The two frequency ranges are overlapping, in other words a setting of OSCCAL = 0x7F gives a higher frequency than OSCCAL = 0x80.

The CAL6...0 bits are used to tune the frequency within the selected range. A setting of 0x00 gives the lowest frequency in that range, and a setting of 0x7F gives the highest frequency in the range.

Practice

- 0b11000111 in hex (0b is C notation that says, “interpret what follows as a binary number”)
- 0b10011001 in hex
- 0b10011001 as a base 10 number
- 0x5A in binary (use 8 bits)
- 0b11111111 in hex and as a base 10 number
- $(37)_{10}$ in binary and hex

the prefix ’0x’ is C notation that means that the digits which follow are hex digits
the prefix ’0b’ means that the digits which follow are binary digits

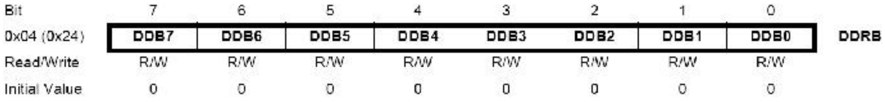
Back to PORT details

Solution

- 1100 0111 in hex = 0xC7
- 1001 1001 in hex = 0x99
- 1001 1001 in base 10 = 153
- 0x5A in binary = 0b0101 1010
- 0b1111 1111 = 0xFF or 255
- (37) = 0b0010 0101 or 0x25

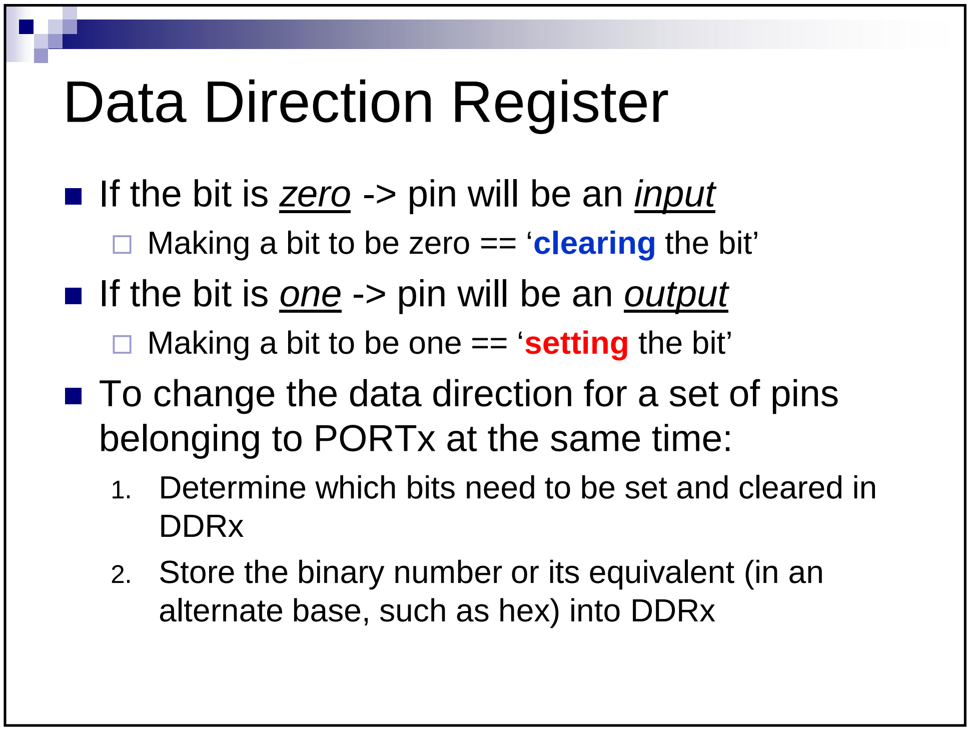
So What?

- Recall the question:
  - □ Is there a way change the data direction for a set of pins all at the same time?
- All the work of MCU happens through *registers* (special memory locations)
  - □ Registers on the Atmega328 are 8-bits wide
- The data direction register (DDRx) handles the data directions for pins in PORTx



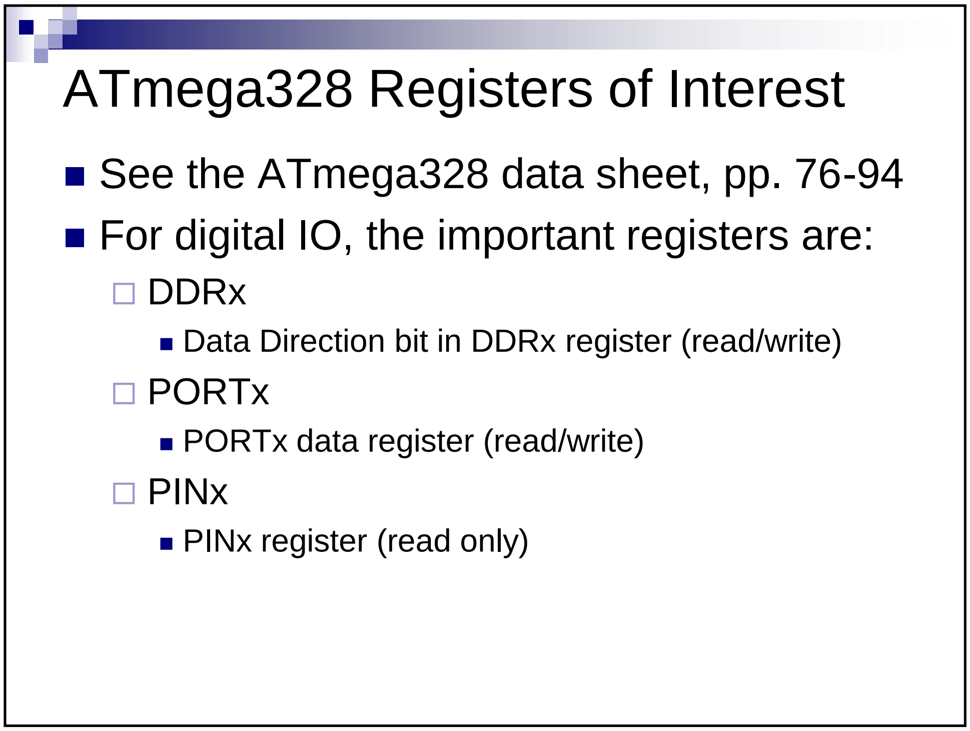
Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Source:http://www.atmel.com/dyn/products/product_card.asp?PN=ATmega328P **p. 93**



Data Direction Register

- If the bit is *zero* -> pin will be an *input*
  - Making a bit to be zero == ‘**clearing** the bit’
- If the bit is *one* -> pin will be an *output*
  - Making a bit to be one == ‘**setting** the bit’
- To change the data direction for a set of pins belonging to PORTx at the same time:
  1. Determine which bits need to be set and cleared in DDRx
  2. Store the binary number or its equivalent (in an alternate base, such as hex) into DDRx



ATmega328 Registers of Interest

- See the ATmega328 data sheet, pp. 76-94
- For digital IO, the important registers are:
  - DDRx
    - Data Direction bit in DDRx register (read/write)
  - PORTx
    - PORTx data register (read/write)
  - PINx
    - PINx register (read only)

PORT Pin and register details

ATmega328 datasheet, pp. 76-94

Figure 13-1. I/O Pin Equivalent Schematic

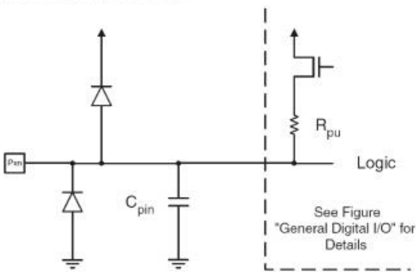
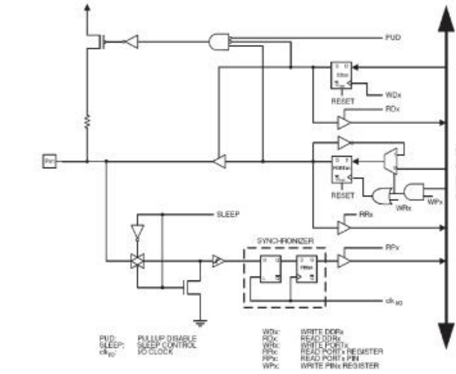


Figure 13-2. General Digital I/O[1]



PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
Read/Write	R	R	R	R	R	R	R	R
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Jump to bits

Example 1

- Make Arduino pins 3, 5, and 7 (PD3, PD5, and PD7) to be outputs
- Arduino approach

```
pinMode(3, OUTPUT);
pinMode(5, OUTPUT);
pinMode(7, OUTPUT);
```

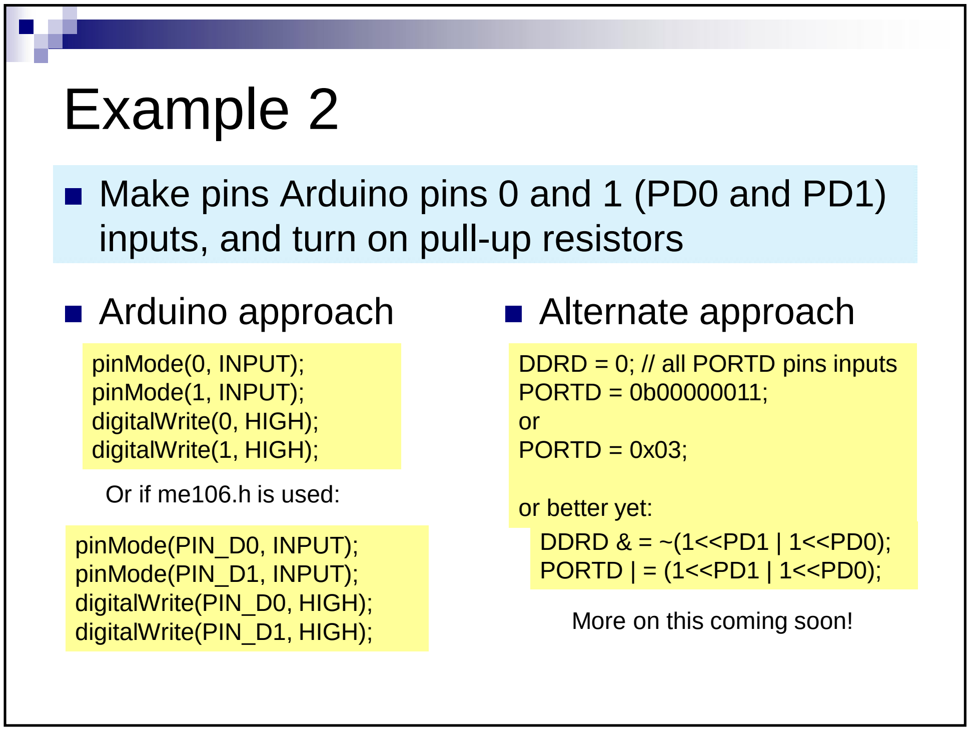
Or if me106.h is used:

```
pinMode(PIN_D3, OUTPUT);
pinMode(PIN_D5, OUTPUT);
pinMode(PIN_D7, OUTPUT);
```

- Alternate approach

```
DDRD = 0b10101000;

or

DDRD = 0xA8;

or

DDRD | = 1<<PD7 | 1<<PD5 | 1<<PD3;
```

More on this coming soon!



Example 2

- Make pins Arduino pins 0 and 1 (PD0 and PD1) inputs, and turn on pull-up resistors

- Arduino approach

```
pinMode(0, INPUT);
pinMode(1, INPUT);
digitalWrite(0, HIGH);
digitalWrite(1, HIGH);
```

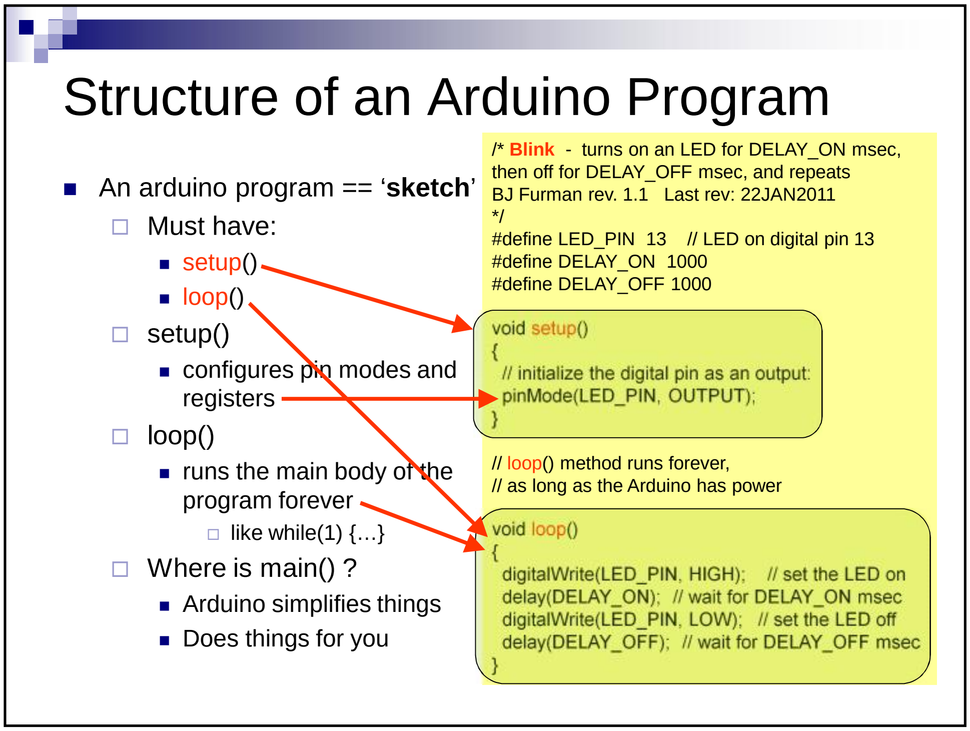
Or if me106.h is used:

```
pinMode(PIN_D0, INPUT);
pinMode(PIN_D1, INPUT);
digitalWrite(PIN_D0, HIGH);
digitalWrite(PIN_D1, HIGH);
```

- Alternate approach

```
DDRD = 0; // all PORTD pins inputs
PORTD = 0b00000011;
or
PORTD = 0x03;

or better yet:
  DDRD & = ~(1<<PD1 | 1<<PD0);
  PORTD | = (1<<PD1 | 1<<PD0);
```

More on this coming soon!



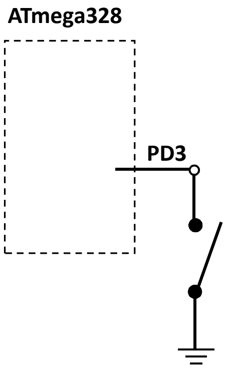
Structure of an Arduino Program

- An arduino program == ‘**sketch**’
  - □ Must have:
    - setup()
    - loop()
  - □ setup()
    - configures pin modes and registers
  - □ loop()
    - runs the main body of the program forever
      - □ like while(1) {…}
  - □ Where is main() ?
    - Arduino simplifies things
    - Does things for you

```
/* Blink  -  turns on an LED for DELAY_ON msec,
then off for DELAY_OFF msec, and repeats
BJ Furman rev. 1.1   Last rev: 22JAN2011
*/
#define LED_PIN  13    // LED on digital pin 13
#define DELAY_ON  1000
#define DELAY_OFF 1000

void setup()
{
 // initialize the digital pin as an output:
 pinMode(LED_PIN, OUTPUT);
}

// loop() method runs forever,
// as long as the Arduino has power

void loop()
{
 digitalWrite(LED_PIN, HIGH);    // set the LED on
 delay(DELAY_ON);  // wait for DELAY_ON msec
 digitalWrite(LED_PIN, LOW);   // set the LED off
 delay(DELAY_OFF);  // wait for DELAY_OFF msec
}
```

Digital IO – Practice 1

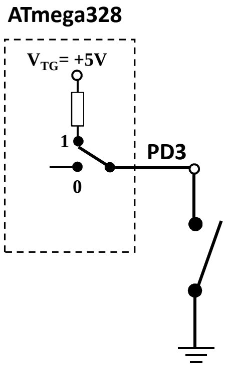
- ‘Reading a pin’
  - Write some lines of C code for the Arduino to determine a course of action if the seat belt has been latched (switch closed).
    - If latched, the ignition should be enabled through a call to a function ig_enable().
    - If not latched, the ignition should be disabled through a call to a function ig_disable()
  - Write pseudocode first

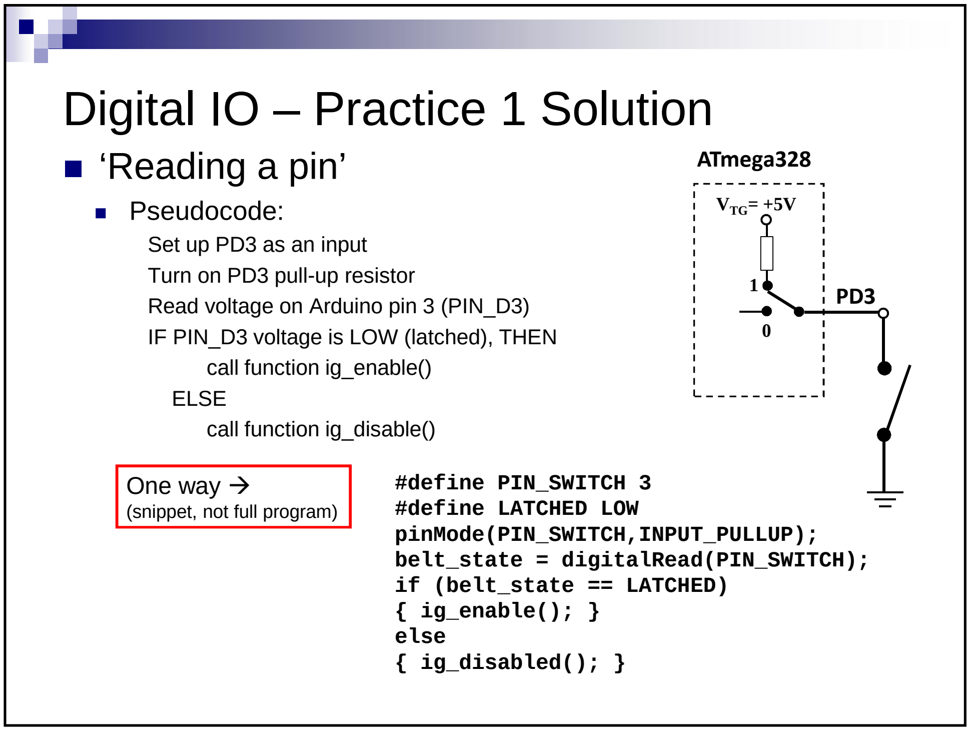


Digital IO – Practice 1 Solution

- ‘Reading a pin’
    - Pseudocode:

```
Set up PD3 as an input
Turn on PD3 pull-up resistor
Read voltage on Arduino pin 3 (PIN_D3)
IF PIN_D3 voltage is LOW (latched), THEN
    call function ig_enable()
ELSE
    call function ig_disable()
```





ATmega328
V_{TG}= +5V
1
0
PD3

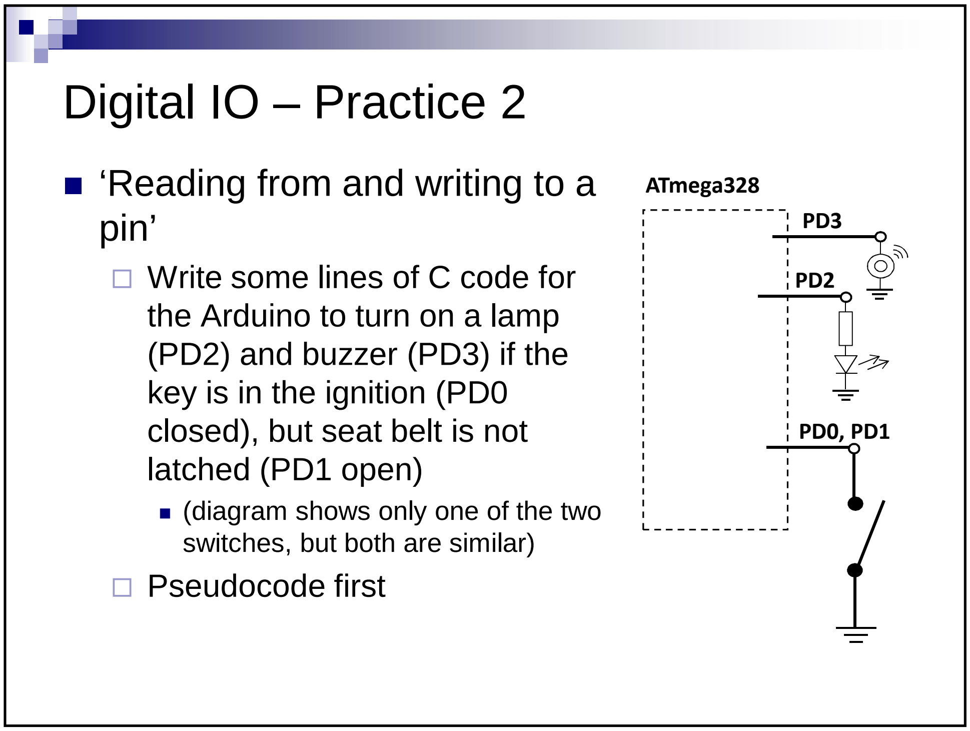
Digital IO – Practice 1 Solution

- ‘Reading a pin’
  - Pseudocode:

```
Set up PD3 as an input
Turn on PD3 pull-up resistor
Read voltage on Arduino pin 3 (PIN_D3)
IF PIN_D3 voltage is LOW (latched), THEN
        call function ig_enable()
    ELSE
        call function ig_disable()
```

One way →
(snippet, not full program)

```
#define PIN_SWITCH 3
#define LATCHED LOW
pinMode(PIN_SWITCH,INPUT_PULLUP);
belt_state = digitalRead(PIN_SWITCH);
if (belt_state == LATCHED)
{ ig_enable(); }
else
{ ig_disabled(); }
```



ATmega328
PD3
PD2
PD0, PD1

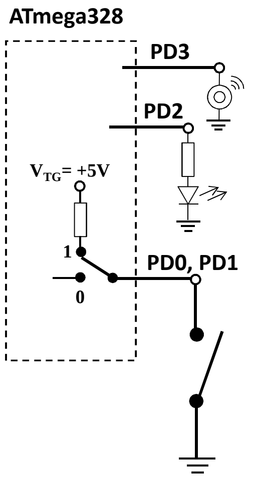
Digital IO – Practice 2

- ‘Reading from and writing to a pin’
  - □ Write some lines of C code for the Arduino to turn on a lamp (PD2) and buzzer (PD3) if the key is in the ignition (PD0 closed), but seat belt is not latched (PD1 open)
    - (diagram shows only one of the two switches, but both are similar)
  - □ Pseudocode first

Digital IO – Practice 2 Pseudocode

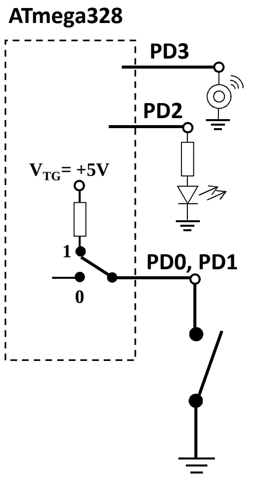
- Pseudocode:

```
Set up data direction of pins
    Make PD0 and PD1 inputs
    Turn on pull up resistors for PD0 and PD1
    Make PD2 and PD3 outputs
Loop forever
    IF key is in ignition THEN
        IF belt is latched, THEN
            Turn off buzzer
            Turn off lamp
        ELSE
            Turn on lamp
            Turn on buzzer
    ELSE
        Turn off buzzer
        Turn off lamp
```



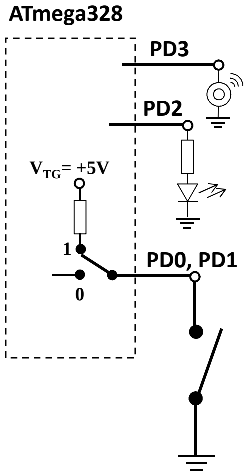
Digital IO – Practice 2 (Arduino style code)

```
#define PIN_IGNITION  0
#define PIN_SEATBELT  1
#define PIN_LED  2
#define PIN_BUZZER  3
#define SEATBELT_LATCHED  LOW
#define KEY_IN_IGNITION LOW
#define LED_ON  HIGH
#define LED_OFF  LOW
#define BUZZER_ON  HIGH
#define BUZZER_OFF  LOW
void setup()
{
  pinMode(PIN_IGNITION, INPUT_PULLUP);  // key switch
  pinMode(PIN_SEATBELT, INPUT_PULLUP); // belt latch switch
  pinMode(PIN_LED, OUTPUT);  // lamp
  pinMode(PIN_BUZZER, OUTPUT);  // buzzer
}
void loop()
{ /* see next page for code */}
```



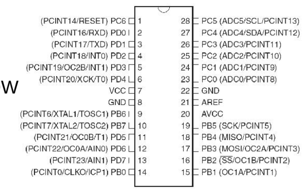
Digital IO – Practice 2 (Arduino style code)

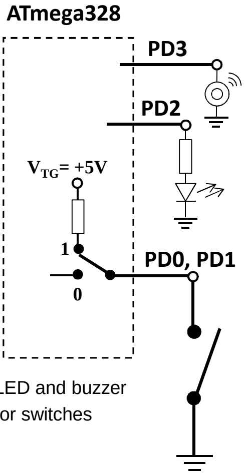
```
/* see previous page for code before loop() */
void loop()
{
  int key_state = digitalRead(PIN_IGNITION);
  int belt_state = digitalRead(PIN_SEATBELT);
  if (key_state == KEY_IN_IGNITION)
    {
      if (belt_state == SEATBELT_LATCHED)
        {
          digitalWrite(PIN_BUZZER, BUZZER_OFF);
          digitalWrite(PIN_LED, LED_OFF);
        }
          else
           {
             digitalWrite(PIN_BUZZER, BUZZER_ON);
             digitalWrite(PIN_LED, LED_ON);
           }
      else
        {
          digitalWrite(PIN_BUZZER, BUZZER_OFF);
          digitalWrite(PIN_LED, LED_OFF);
        }
    }
}
```



Digital IO – Practice 3 (Port style code)

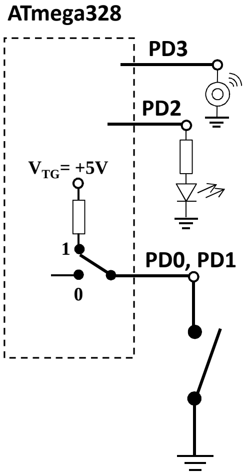
```
/* NOTE: #defines use predefined PORT pin numbers  for ATmega328 */
#define PIN_IGNITION  PD0
#define PIN_SEATBELT  PD1
#define PIN_LED  PD2
#define PIN_BUZZER  PD3
#define SEATBELT_LATCHED  LOW
#define KEY_IN_IGNITION LOW
#define LED_ON  HIGH
#define LED_OFF  LOW
#define BUZZER_ON  HIGH
#define BUZZER_OFF  LOW
#define  _BIT_MASK( bit )  ( 1 << (bit) )  // same as _BV( bit)
void setup()
{
   PORTD = 0;  // all  PORTD pullups off
   DDRD | = _BIT_MASK(PIN_LED)  | _BIT_MASK(PIN_BUZZER);  // LED and buzzer
   PORTD | = _BV(PIN_IGNITION) | _BV(PIN_SEATBELT);  // pullups for switches
}

/* See next page for loop() code */
```





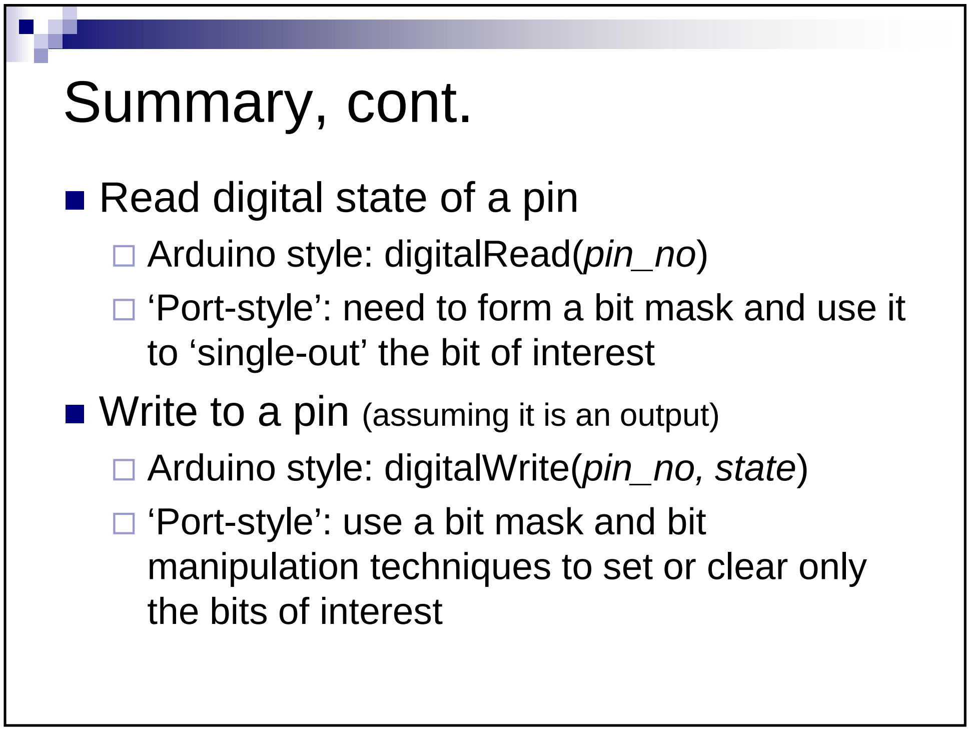
Digital IO – Practice 3 (Port style code)

```
/* see previous page for setup() code */
void loop()
{
    uint8_t current_PORTD_state, key_state, belt_state;
    current_PORTD_state = PIND;  // snapshot of PORTD pins
    key_state = current_PORTD_state  &  _BV(PIN_IGNITION);
    belt_state = current_PORTD_state  &  _BV(PIN_SEATBELT);
    if (key_state  == KEY_IN_IGNITION)
    {
        if (belt_state == SEATBELT_LATCHED)
        {
          PORTD & =  ~( _BV(PIN_LED)  |  _BV(PIN_BUZZER) );
        }
        else
        {
          PORTD | =  ( _BV(PIN_LED)  |  _BV(PIN_BUZZER) );
        }
    }
    else
    {
      PORTD  & = ~( _BV(PIN_LED)  |  _BV(PIN_BUZZER) );
    }
}
```



Summary

- Data direction
  - □ Input is default, but okay to set explictly
  - □ Output
    - Arduino style: pinMode(*pin_no, mode*)
    - Alternate: Set bits in DDRx
- Pull-up resistors
  - □ Pin must be an input
    - Arduino style: digitalWrite(*pin_no, state*)
    - Alternate style: Set bits in PORTx



Summary, cont.

- Read digital state of a pin
  - Arduino style: digitalRead(*pin_no*)
  - ‘Port-style’: need to form a bit mask and use it to ‘single-out’ the bit of interest
- Write to a pin (assuming it is an output)
  - Arduino style: digitalWrite(*pin_no, state*)
  - ‘Port-style’: use a bit mask and bit manipulation techniques to set or clear only the bits of interest