Embedded Programming for Beginners

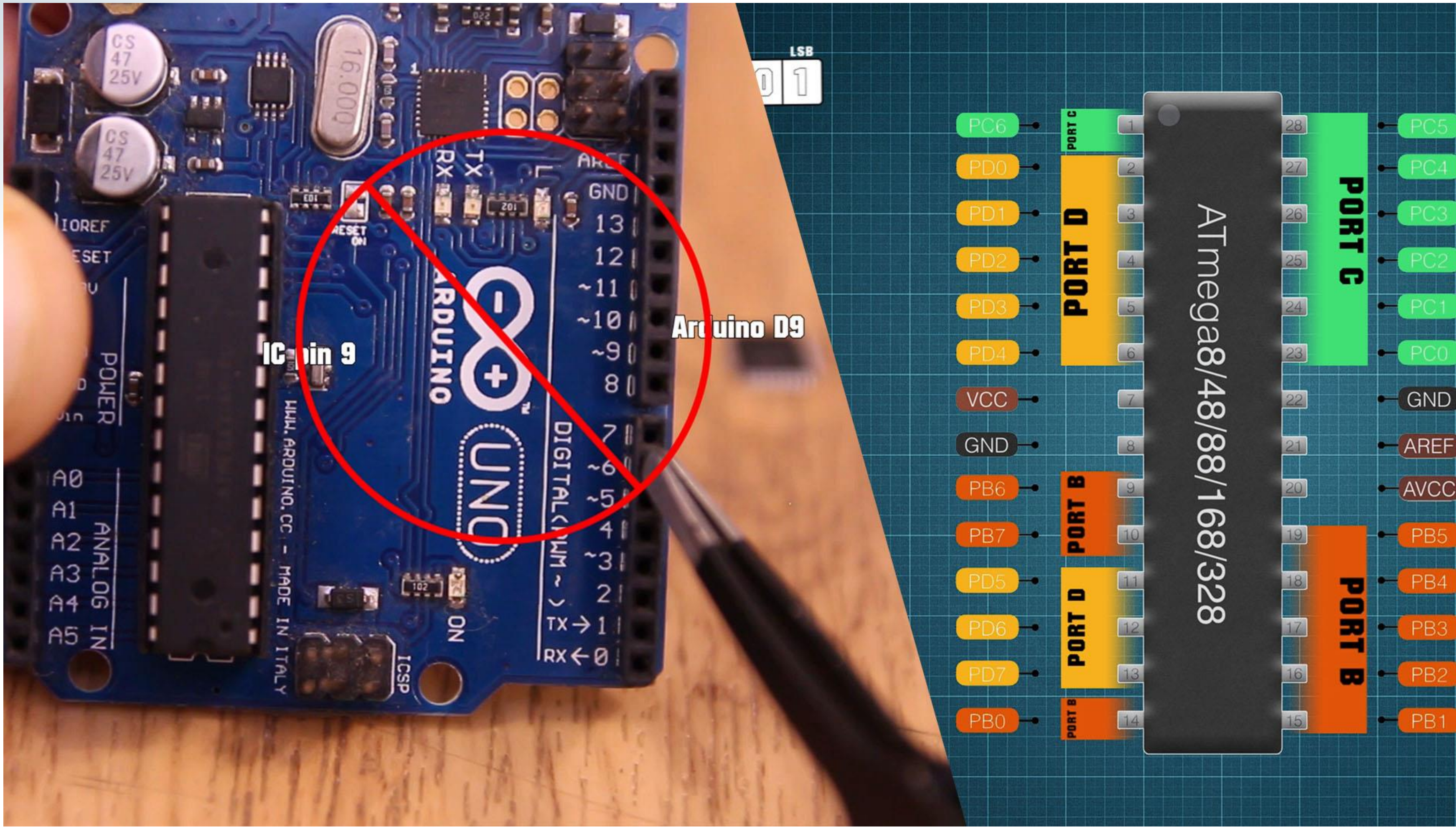
Implementing an embedded application using Arduino

Session 2

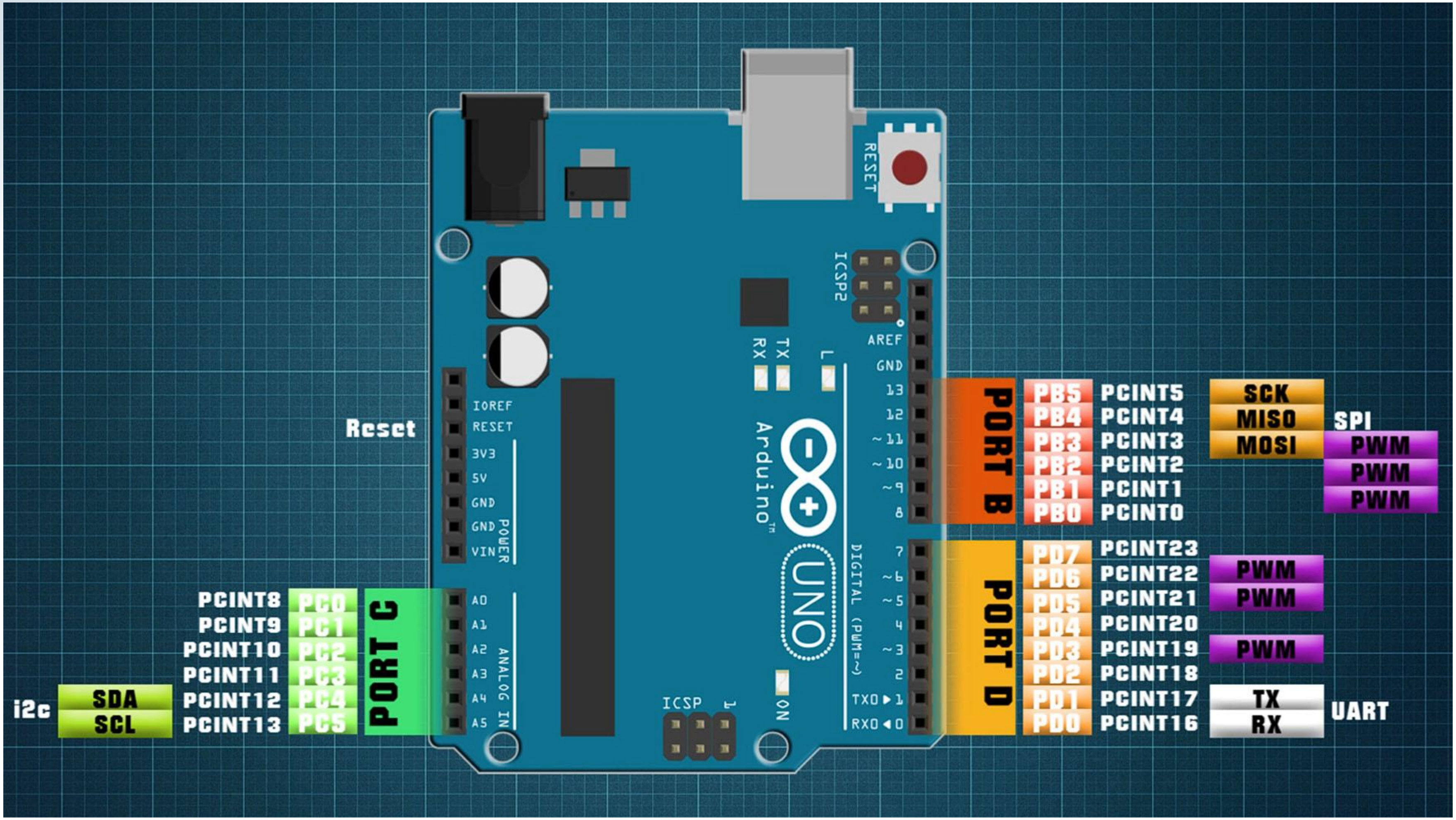
Course Goals

- Objective
  - Introduction to datasheets
  - Integrate registers and bitwise operations into the mix
  - Building applications with or without an IDE

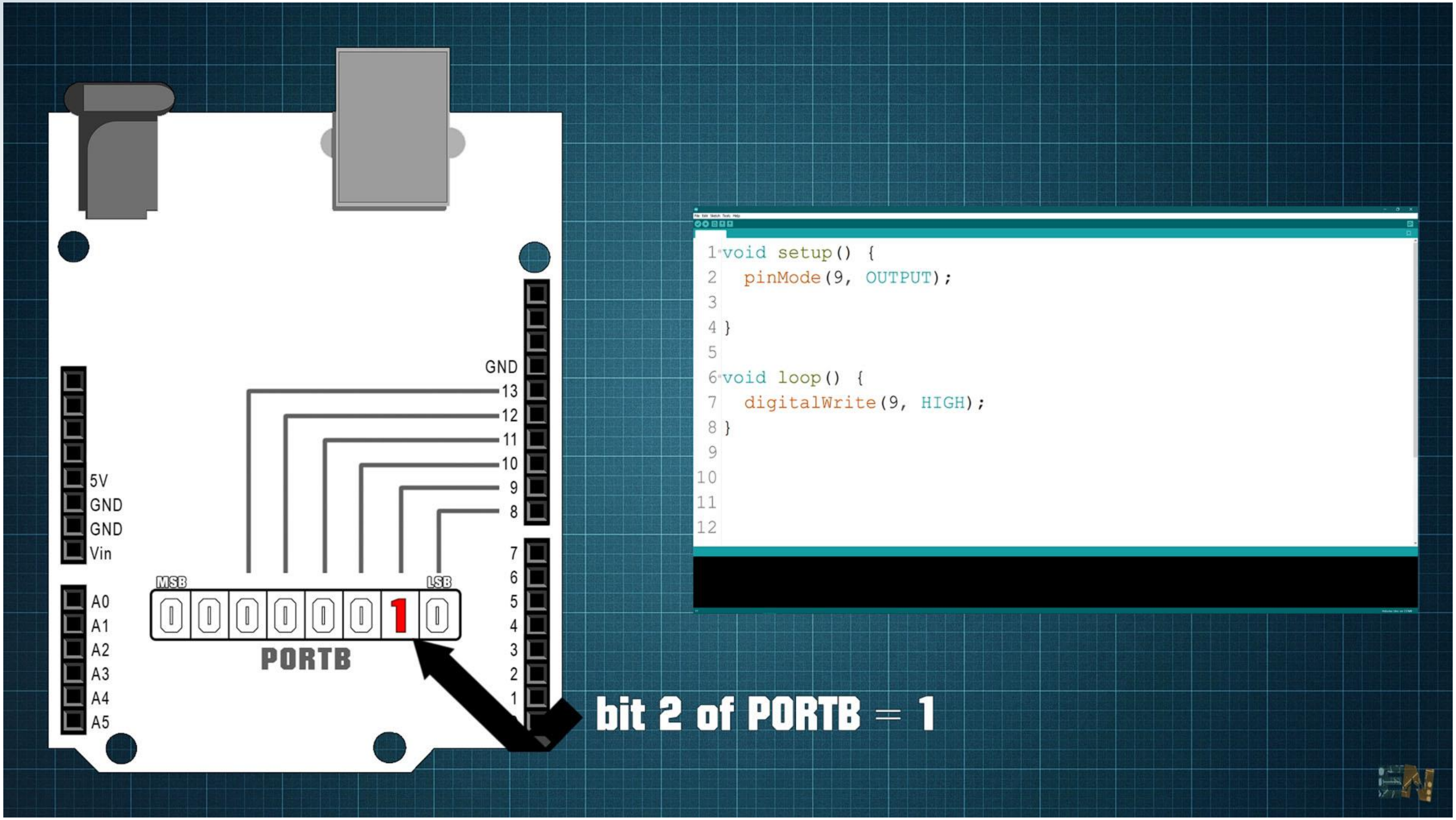
ATmega328p-PU pins



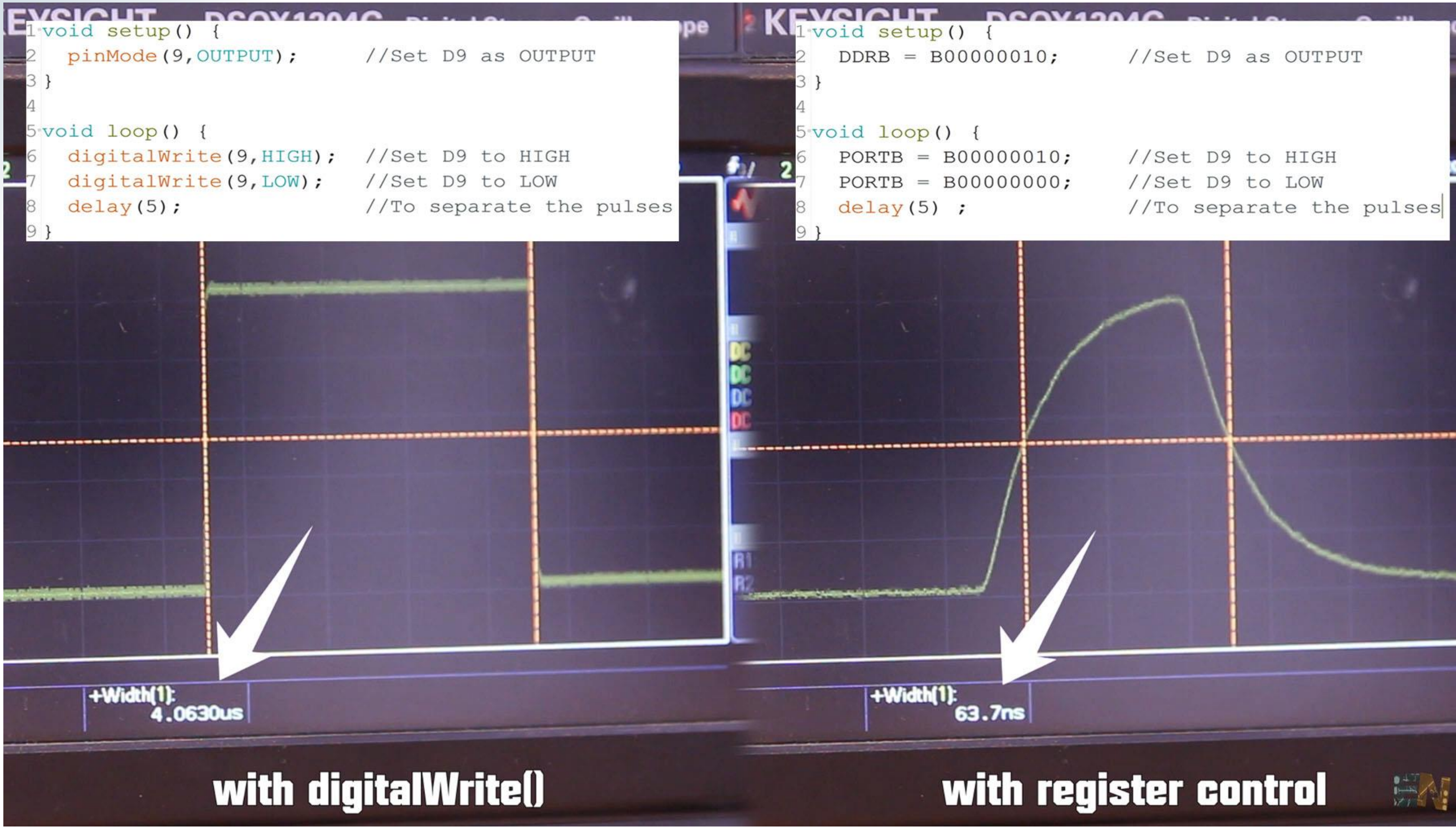
Arduino UNO ports



digitalWrite vs Register control



Speed test


void setup() {
pinMode(9,OUTPUT); //Set D9 as OUTPUT
}
void loop() {
digitalWrite(9,HIGH); //Set D9 to HIGH
digitalWrite(9,LOW); //Set D9 to LOW
delay(5); //To separate the pulses
}
+Width(1):
4.0630us
with digitalWrite()
void setup() {
DDRB = B00000010; //Set D9 as OUTPUT
}
void loop() {
PORTB = B00000010; //Set D9 to HIGH
PORTB = B00000000; //Set D9 to LOW
delay(5) ; //To separate the pulses
}
+Width(1):
63.7ns
with register control

Register Control: Set pin as Output/Input

```
void setup() {
  DDRD = B00010100;         //Sets D2 and D4 as OUTPUT and the rest as INPUT (not recommended)
}

void loop() {
  //your code here...
}
```

```
void setup() {
  DDRD = B00010100;         //Sets D2 and D4 as OUTPUT and the rest as INPUT (not recommended)
  DDRD = DDRD | B00010100;  //This is safer as it sets ONLY pins D2 and D4 as OUTPUT

  //Other exampoles
  DDRB |= B00000101;        //Sets only D8 and D10 as OUTPUT
  DDRB &= B11100111;        //Sets only D11 and D12 as INPUT
  DDRC &= B11110110;        //Sets only A0 and A3 as INPUT
}

void loop() {
  //your code here...
}
```

Register Control: Set pin as LOW/HIGH

```
void setup() {
  PORTD |= B00000100;         //Sets only D2 to HIGH

  PORTB &= B11111001;         //Sets only D9 and D10 to LOW
  PORTB |= B00110000;         //Sets only D12 and D13 to HIGH

  PORTC |= B00000010;         //Sets A1 to HIGH
}

void loop() {
  //your code here...
}
```

Blink Example

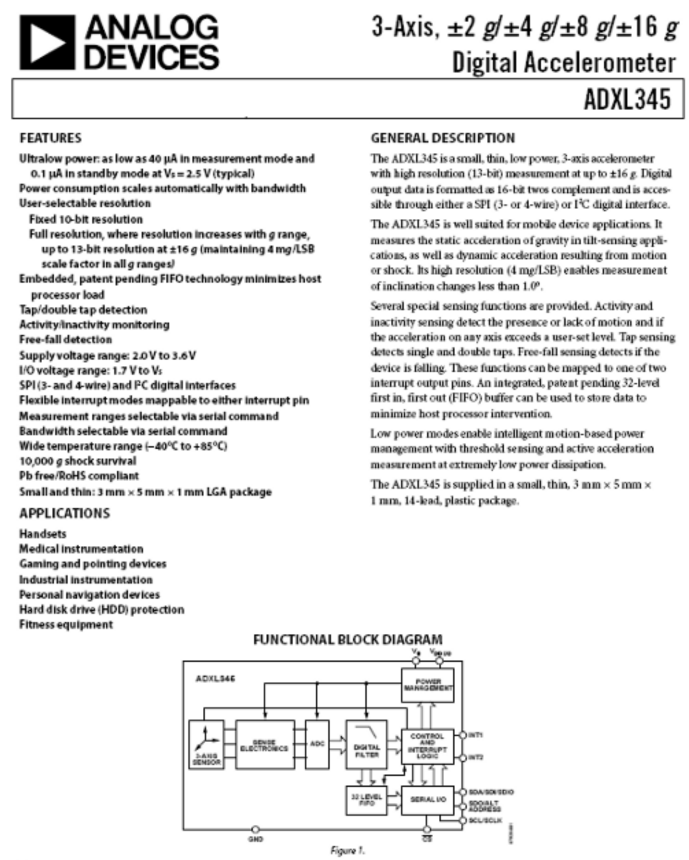
```
void setup() {
  DDRD |= B00000100;  //Set only D2 as OUTPUT
}

void loop() {
  PORTD |= B00000100;         //Sets only D2 to HIGH
  delay(1000);
  PORTD &= ~B00000100;        //Sets only D2 to LOW (we use ~ to invert the byte)
  delay(1000);
}
```

Register Control: Read one input

```
void setup() {
  DDRD &= ~B00100000;  //Set only D5 as INPUT
}

void loop() {
  //This is NOT equal to: int value = digitalRead(5);
  //Why? Because value = an 8 bit value. For only 1 bit, you need to shif regisers
  int value = PIND & B00100000;


  //This is equal to  int value2 = digitalRead(5);
  //Since we read the fifth bit, we need to shift 5 times to the right
  int value2 = (PIND >> 5 & B00100000 >> 5);

  //In case you want to make an if decision you don't have to shift
  //Next line equal to: if(digitalRead(5))
  if(PIND & B00100000){
    //Add your code...
  }

  //We can invert the result: Next line equal to: if(!digitalRead(5))
  if(!(PIND & B00100000)){
    //Add your code...
  }
}
```

Datasheet

- Instruction manual for electronic components
- Written from developers for developers
- First page: summary
- Basic specifications description
- Functional block diagram



ANALOG DEVICES

3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer

ADXL345

FEATURES

Ultralow power: as low as 40 µA in measurement mode and 0.1 µA in standby mode at V_S = 2.5 V (typical)
Power consumption scales automatically with bandwidth
User-selectable resolution
Fixed 10-bit resolution
Full resolution, where resolution increases with *g* range, up to 13-bit resolution at ±16 *g* (maintaining 4 m*g*/LSB scale factor in all *g* ranges)
Embedded, patent pending FIFO technology minimizes host processor load
Tap/double tap detection
Activity/inactivity monitoring
Free-fall detection
Supply voltage range: 2.0 V to 3.6 V
I/O voltage range: 1.7 V to V_S
SPI (3- and 4-wire) and I^2C digital interfaces
Flexible interrupt modes mappable to either interrupt pin
Measurement ranges selectable via serial command
Bandwidth selectable via serial command
Wide temperature range (−40°C to +85°C)
10,000 *g* shock survival
Pb free/RoHS compliant
Small and thin: 3 mm × 5 mm × 1 mm LGA package

APPLICATIONS

Handsets
Medical instrumentation
Gaming and pointing devices
Industrial instrumentation
Personal navigation devices
Hard disk drive (HDD) protection
Fitness equipment

GENERAL DESCRIPTION

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I^2C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins. An integrated, patent pending 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor intervention.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

FUNCTIONAL BLOCK DIAGRAM

Figure 1.

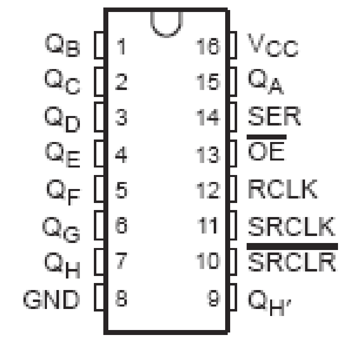
Pinouts & absolute maximum ratings

ABSOLUTE MAXIMUM RATINGS

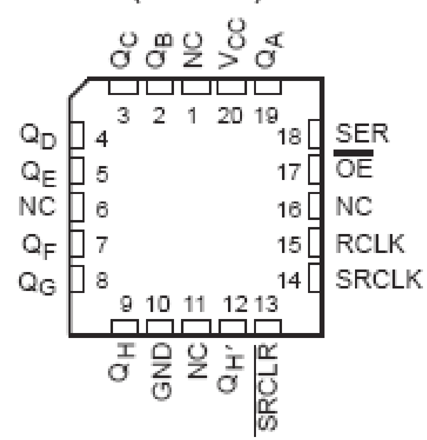
Table 2.

Parameter	Rating
Acceleration	
Any Axis, Unpowered	10,000 *g*
Any Axis, Powered	10,000 *g*
V_S	−0.3 V to +3.6 V
$V_{DD\ I/O}$	−0.3 V to +3.6 V
Digital Pins	−0.3 V to $V_{DD\ I/O}$ + 0.3 V or 3.6 V, whichever is less
All Other Pins	−0.3 V to +3.6 V
Output Short-Circuit Duration (Any Pin to Ground)	Indefinite
Temperature Range	
Powered	−40°C to +105°C
Storage	−40°C to +105°C

SN54HC595 . . . J OR W PACKAGE
SN74HC595 . . . D, DB, DW, N, OR NS PACKAGE
(TOP VIEW)



SN54HC595 . . . FK PACKAGE
(TOP VIEW)



NC − No internal connection

Recommended operating conditions

recommended operating conditions (see Note 3)

			SN54HC595 MIN	SN54HC595 NOM	SN54HC595 MAX	SN74HC595 MIN	SN74HC595 NOM	SN74HC595 MAX	UNIT
V_{CC}	Supply voltage		2	5	6	2	5	6	V
V_{IH}	High-level input voltage	V_{CC} = 2 V	1.5			1.5			V
		V_{CC} = 4.5 V	3.15			3.15			
		V_{CC} = 6 V	4.2			4.2			
V_{IL}	Low-level input voltage	V_{CC} = 2 V			0.5			0.5	V
		V_{CC} = 4.5 V			1.35			1.35	
		V_{CC} = 6 V			1.8			1.8	
V_I	Input voltage		0		V_{CC}	0		V_{CC}	V
V_O	Output voltage		0		V_{CC}	0		V_{CC}	V
$\Delta t/\Delta v$‡	Input transition rise/fall time	V_{CC} = 2 V			1000			1000	ns
		V_{CC} = 4.5 V			500			500	
		V_{CC} = 6 V			400			400	
T_A	Operating free-air temperature		−55		125	−40		85	°C

NOTE 3: All unused inputs of the device must be held at V_{CC} or GND to ensure proper device operation. Refer to the TI application report, *Implications of Slow or Floating CMOS Inputs*, literature number SCBA004.

‡ If this device is used in the threshold region (from V_{IL}max = 0.5 V to V_{IH}min = 1.5 V), there is a potential to go into the wrong state from induced grounding, causing double clocking. Operating with the inputs at t_t = 1000 ns and V_{CC} = 2 V does not damage the device; however, functionally, the CLK inputs are not ensured while in the shift, count, or toggle operating modes.

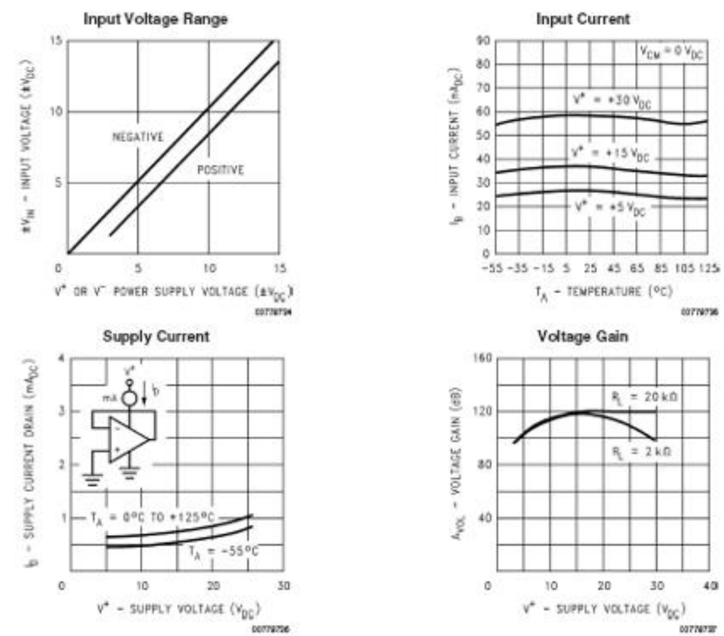
Graphs & truth tables

SN54HC595, SN74HC595
8-BIT SHIFT REGISTERS
WITH 3-STATE OUTPUT REGISTERS
SCLS041G – DECEMBER 1982 – REVISED FEBRUARY 2004

FUNCTION TABLE

INPUTS					FUNCTION
SER	SRCLK	$\overline{\text{SRCLR}}$	RCLK	$\overline{\text{OE}}$	
X	X	X	X	H	Outputs Q_A–Q_H are disabled.
X	X	X	X	L	Outputs Q_A–Q_H are enabled.
X	X	L	X	X	Shift register is cleared.
L	↑	H	X	X	First stage of the shift register goes low. Other stages store the data of previous stage, respectively.
H	↑	H	X	X	First stage of the shift register goes high. Other stages store the data of previous stage, respectively.
X	X	X	↑	X	Shift-register data is stored in the storage register.

Typical Performance Characteristics



Timing diagrams & packaging information

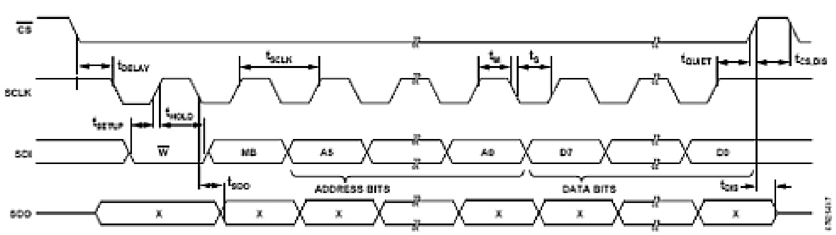
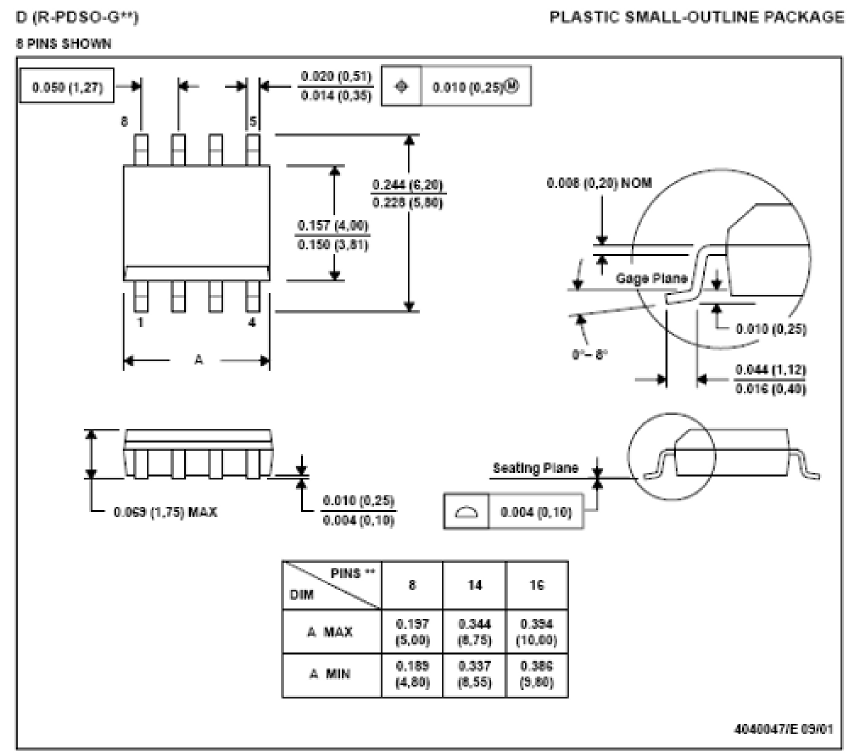


Figure 5. SPI 4-Wire Write



Application information

ADXL345

I²C

With $\overline{CS}$ tied high to $V_{DD\,I/O}$, the ADXL345 is in I²C mode, requiring a simple 2-wire connection as shown in Figure 8. The ADXL345 conforms to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, available from NXP Semiconductor. It supports standard (100 kHz) and fast (400 kHz) data transfer modes if the timing parameters given in Table 11 and Figure 10 are met. Single- or multiple-byte reads/writes are supported, as shown in Figure 9. With the SDO/ALT ADDRESS pin high, the 7-bit I²C address for the device is 0x1D, followed by the R/$\overline{W}$ bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I²C address of 0x53 (followed by the R/$\overline{W}$ bit) can be chosen by grounding the SDO/ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

If other devices are connected to the same I²C bus, the nominal operating voltage level of these other devices cannot exceed $V_{DD\,I/O}$ by more than 0.3 V. External pull-up resistors, R_P, are necessary for proper I²C operation. Refer to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, when selecting pull-up resistor values to ensure proper operation.

Table 10. I²C Digital Input/Output Voltage

Parameter	Limit[1]	Unit
Digital Input Voltage		
Low Level Input Voltage (V_{IL})	$0.25 \times V_{DD\,I/O}$	V max
High Level Input Voltage (V_{IH})	$0.75 \times V_{DD\,I/O}$	V min
Digital Output Voltage		
Low Level Output Voltage (V_{OL})[2]	$0.2 \times V_{DD\,I/O}$	V max

[1] Limits based on characterization results; not production tested.
[2] The limit given is only for $V_{DD\,I/O}$ < 2 V. When $V_{DD\,I/O}$ > 2 V, the limit is 0.4 V max.

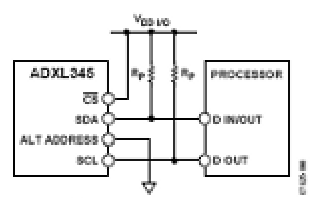


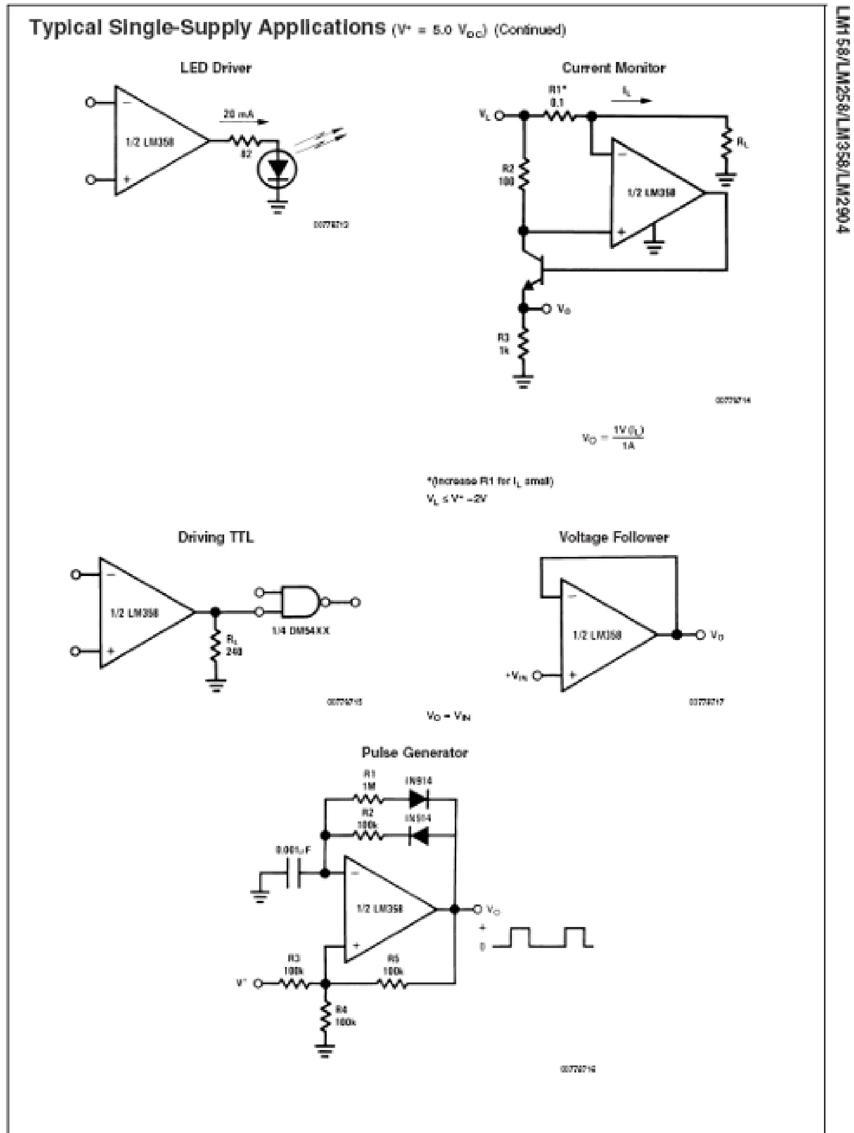
Figure 8. I²C Connection Diagram (Address 0x53)

¹THIS START IS EITHER A RESTART OR A STOP FOLLOWED BY A START.

NOTES
1. THE SHADED AREAS REPRESENT WHEN THE DEVICE IS LISTENING.

Figure 9. I²C Device Addressing

Example schematics



Hands-on exercise: Datasheet question

- A datasheet is the best place to find:
  A. What voltage a part needs to run
  B. How fast a part will run
  C. How to communicate with a part
  D. All of the above

Bitwise operations

- 0b11 = 3
- Bitwise AND: &
- Bitwise OR: |
- Bitwise XOR: ^
- Bitwise NOT: ~
- Bit shift operators: << (left shift), >> (right shift)
- Assignment operators: +

Common bitwise operations mistakes

- ”!” vs ”~”
- ”&” vs ”&&”
- ”|” vs ”||”
- Byte = 1 << Bit_index

  vs

  Byte |= 1 << Bit_index
- Byte = ~(1 << Bit_index)

  vs

  Byte &= ~(1 << Bit_index)

Hands-on exercise: bitwise operations

- int x = 5; int y = x & 1; What is the value of y?
- int x = 4; int y = x & 1; What is the value of y?
- int a = 92; int b = 101; What is the value of a | b?
- int x = 12; int y = 10; What is the value of x ^ y?
- int a = 103; What is the value of ~a?
- int a = 5; What is the value of a << 3?

Arduino I/O operations

- DDRn - Data Direction Register
- PORTn - Data Register
- PINn - Input Pins Address
- pinMode()
- digitalRead()
- digitalWrite()

Operation	Formula
Write bit on 1	register |= (1 << bit_index)
Write bit on 0	register &= ~(1 << bit_index)
Toggle bit	register ^= (1 << bit_index)
Read bit	register & (1 << bit_index)

Arduino software stack

Arduino preprocessing

- .ino to .C++
- Adds Arduino.h
- Adds #Line directive
- Add function prototype for all the functions

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}


void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

INO

```
#include <Arduino.h>
#line 1 "C:\\Users\\Sabarish\\AppData\\Local\\Temp\\arduino_modified_sketch_875345\\Blink.ino"
#line 1 "C:\\Users\\Sabarish\\AppData\\Local\\Temp\\arduino_modified_sketch_875345\\Blink.ino"
void setup();
#line 6 "C:\\Users\\Sabarish\\AppData\\Local\\Temp\\arduino_modified_sketch_875345\\Blink.ino"
void loop();
#line 1 "C:\\Users\\Sabarish\\AppData\\Local\\Temp\\arduino_modified_sketch_875345\\Blink.ino"
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}


void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Header File

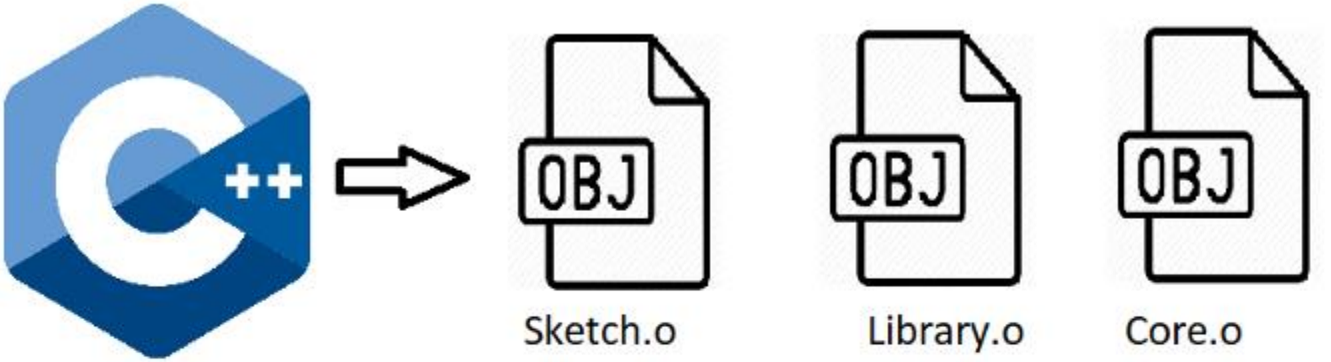
Function Prototype

#Line Directive

C++

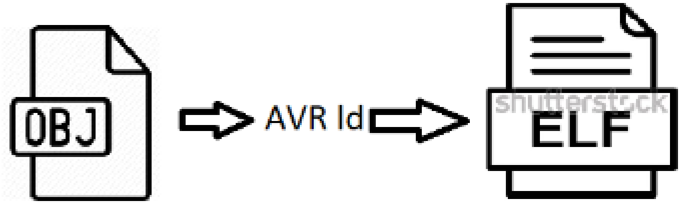
Arduino compiling

- AVR-GCC Toolchain
- Compiler, assembler, linker, Standard C, math libraries and many others…



Arduino linking

- AVR-ld (GNU linker)
- ELF (Executable Linkable Format) format



Arduino ELF extraction

- AVR-Objcopy: convert ELF to EEP and HEX file
- EEP file – stores info on EEPROM (Electrically Erasable Programmable Read-Only Memory) media

Arduino flashing

- AVRDUDE: download and upload on-chip memories of Atmel’s AVR microcontrollers
- Works with Flash and EEPROM memory
- Select the correct board and serial port for upload
  - Tools > Board
  - Tools > Port
- Reset automatically and begin the upload
- Arduino bootloader

AVR programming without IDE: prerequisites

- Windows:
  - pacman -S make mingw-w64-i686-avrdude
- Linux:
  - Ubuntu: sudo apt-get install gcc-avr avr-libc avrdude
  - Arch Linux: sudo pacman -S avr-gcc avr-libc avrdude
- Mac:
  - brew install avrdude
  - xcode-select --install
  - brew tap osx-cross/avr
  - brew install avr-gcc
- Prerequisites:

AVR programming without IDE

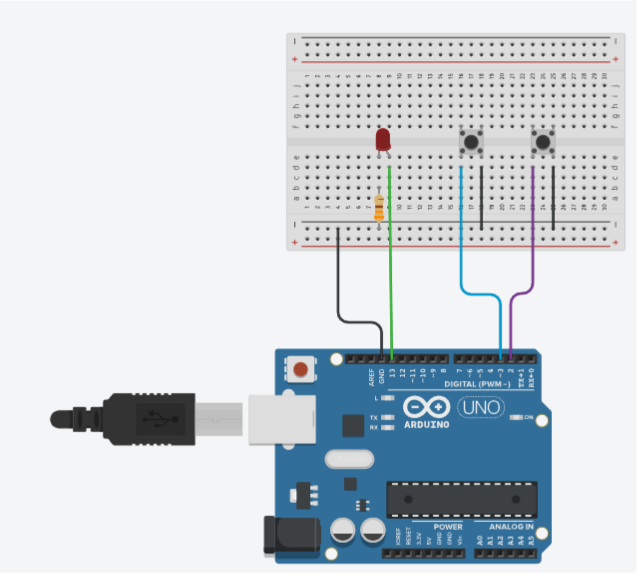
- main.c – blink LED connected to PD1
- Makefile – make; make program

```c
#define F_CPU 20000000    // AVR clock frequency in Hz, used by util/delay.h
#include <avr/io.h>
#include <util/delay.h>

int main() {
    DDRD |= (1<<1);        // set LED pin PD1 to output
    while (1) {
        PORTD |= (1<<1);   // drive PD1 high
        _delay_ms(100);    // delay 100 ms
        PORTD &= ~(1<<1);  // drive PD1 low
        _delay_ms(900);    // delay 900 ms
    }
}
```

```make
MCU=atmega328p
PORT=$(shell pavr2cmd --prog-port)
CFLAGS=-g -Wall -mcall-prologues -mmcu=$(MCU) -Os
LDFLAGS=-Wl,-gc-sections -Wl,-relax
CC=avr-gcc
TARGET=main

all: $(TARGET).hex

clean:
	rm -f *.o *.elf *.hex

%.hex: %.elf
	avr-objcopy -R .eeprom -O ihex $< $@

$(TARGET).elf: $(TARGET).o
	$(CC) $(CFLAGS) $(LDFLAGS) $^ -o $@

program: $(TARGET).hex
	avrdude -c stk500v2 -P "$(PORT)" -p $(MCU) -U flash:w:$<:i
```

Hands-on exercise: Enhanced hello world

- Add a button to pin 2 (PD2), another one to pin 3 (PD3), and an LED connected to pin 13:
  - Use PD2 to decrease the duration of the delay
  - Use PD3 to increase the duration of the delay
- Open Tinkercad: https://www.tinkercad.com
  - Reproduce the hardware setup in the simulator



Closing remarks

- Registers control
- Arduino functions
- Datasheet info
- Bitwise operations
- Arduino programming

- Q&A