**Week1**

* Arduino is an open-source computer hardware/software platform for building digital devices and interactive objects that can sense and control the physical world around them
* Arduino boards are able to read inputs - light on sensor, a finger on button, or a twitter message - and turn to it into output - activating a motor, turning an LED light, publishing something online
* The least expensive version of the Arduino module can be assembled by hand
* IoT - internet of things

Arduino Board

Machine generated alternative text:
Input/0utput Pins 
/Digital I/O 
ARDUXNO 
Power/reset pins 
Analog inputs 

* Digital I/O: draws and output 0 to 5 volts
* Analog inputs: draws a volt between 0 and 5
* Power/reset pins: have 5v, 3.3V, ground, so you can use those pins with the outside world

Machine generated alternative text:
Microcontrollers 
ATmega328 
• TN"ATmega161J2 
• ATmega328 is the processor 
programmed by the user 
• handles USB 
communication 

* You code is written only to the ATmega328, the ATmega16U2 talks to the USB and has pre written code which you don't touch it

Machine generated alternative text:
Firmware 
Two types of code executing on a simple 
microcontroller: 
2. 
Application code 
Executes the system's main functionality 
We write this code 
Firmware 
Low-level code: supports the main 
function 
IJS8 interface. power modes. reset, etc. 
The distinction is a matter of perspective 
Arduino firmware is pre-programmed 

* A bootloader is an example of Firmware, the code you write is the application
* Update the bootloader
  + Machine generated alternative text:
    Bootloader 
    • Firmware on a microcontroller 
    • Allows the Flash and EEPROM to 
    be programmed 
    • Manages USB communication, 
    since application programming 
    is via USB 
  + To program the circuit with a new bootloader you do not use the normal USB programming. That is done with In-Circuit Serial Programming (ICSP)

Machine generated alternative text:
In-Circuit Serial Programming (ICSP) 
• A special programming 
method to program the 
firmware 
• Needed because the 
bootloader can't 
reprogram itself 

Machine generated alternative text:
ICSP Headers 
ATmega161]2 
ATmega328 
One ICSP header for each processor 

<https://www.electronicwings.com/arduino/basics-to-developing-bootloader-for-arduino>

**Arduino Schematics**

Machine generated alternative text:
Microcontrollers and I/O 
pev3 
ATmega161J2 
Power pins 
Analog in 
igital I/O 
ATmega328 

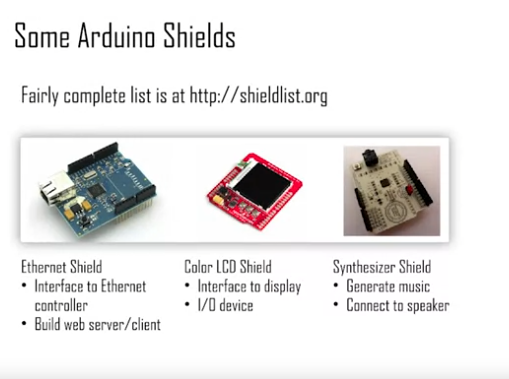
Machine generated alternative text:
USB, Power, ICSP 
3 
ICS I 
USB connector 
Powe 
conn tor 
ATmega328 

**IDE**

Machine generated alternative text:
Arduino Integrated Development Environment (IDE) 
Menus with all commands 
Buttons for common commands 
Text editor for writing code 
e—Message area 

Machine generated alternative text:
Verify: Compiles code. checks for errors 
Upload: Compiles code. checks for errors, uploads to board 
New: Creates a new sketch 
Open: Opens an existing sketch 
Save: Saves your sketch to a file 
Serial Monitor: Opens a window to communicate with the board 

**Arduino Shields**



**Week2**

**Setting up your environment**

The benefit of an IDE is that it puts all the tools together in a nice graphical user interface



Machine generated alternative text:
Types and Type [lualifiers 
• Several built-in types, different sizes 
Type Size 
char I byte 
int word Size 
float Floating point 
double Double-precision 
Notes 
16 bit minimum 
64 bits, typical 
64, 128 typical 
• Type qualifiers exist: short, long 
• Char is 8 bits on all platforms 

Machine generated alternative text:
Conditional Statements 
if (exprl) 
if (expressi on) 
s ta temen el 
else 
s ta temen e2 
stat 1 
else if (expr2) 
sta t2 
else 
• else is optional 
• expression is evaluated 
— Execute statementl if TRUE, statement2 if 
FALSE 
• expr2 evaluated if exprl is FALSE 

Machine generated alternative text:
Functions 
int main ( ) ( 
printf , 
int x, r2, z—3, 
printf ( x) 
Function 
definition 
Function 
• Functions can replace groups of instructions 
• Define a function; call a function 

$ serup() function

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. the setup() function will only run once, after each powerup or reset of the Arduino board.

Example Code

$ int buttonPin = 3;

$ void setup() {

$ Serial.begin(9600);

$ pinMode(buttonPin, INPUT);

$ }

$ void loop() {

$ // ….

$ }

$ pinMode()

[Digital I/O]

Configure the specfied pin to behave either as an input or an output.

Syntax

$ pinMode(pin, mode)

Prameters

$ pin: the Arduino pin number to set the mode of

$ mode: INPUT, OUTPUT, OR INPUT\_PULLUP.

Returns

Nothing

Machine generated alternative text:
Example Code 
The code makes the digital pin 13 OUTPUT and Toggles it HIGH and LON 
void setup() { 
pinmode(13, OUTPUT); 
void loop() { 
digitallérite(13, 
delay ; 
digitallérite(13, 
delay (leea); 
HIGH) ; 
LOW); 
sets the digital pin 13 as output 
sets the digital pin 13 on 
waits for a second 
sets the digital pin 13 off 
waits for a second 

$ loop()

Description

After creating the setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Machine generated alternative text:
Example Code 
int buttonPin 
// setup initializes serial and the button pin 
void setup() { 
Serial . begin(g6ae) ; 
pinMode(buttonPin, INPUT); 
// loop checks the button pin each time, 
// and will send serial if it is pressed 
void loop() { 
if (digitalRead(buttonPin) 
= HIGH) { 
Serial.write('H'); 
else 
Serial.write('L'); 
delay (leea); 

$ digitalWrite()

Description

Write a HIGH or a LOW value to a digital pin

Syntax

digitalWrite(pin, value)

Parameters

pin: the Arduino pin number.

value: HIGH or LOW.

Returns

Nothing

Example Code

The code makes the digital pin 13 an OUTPUT and toggles it by alternating between HIGH and LOW at one second pace.

void setup() {  
 pinMode(13, OUTPUT); // sets the digital pin 13 as output  
}

void loop() {  
 digitalWrite(13, HIGH); // sets the digital pin 13 on  
 delay(1000); // waits for a second  
 digitalWrite(13, LOW); // sets the digital pin 13 off  
 delay(1000); // waits for a second  
}

$ digitalRead()

Description

Reads the value from a specified digital pin, wither HIGH or LOW

Syntax

$ digitalRead(pin)

Parameters

$ pin: the Arduino pin number you want to read

Returns

HIGH or LOW

Machine generated alternative text:
Example Code 
Sets pin 13 to the same value as pin 7, declared as an input. 
int ledPin = 13; 
int inPin = 7; 
int val = e; 
void setup() { 
pinMode(1edPin, 
pinMode(inPin, 
void loop() { 
// LED connected to digital pin 13 
// pushbutton connected to digital pin 7 
// variable to store the read 
OUTPUT), 
INPUT); 
digital Read (inpin); 
digitallérite(ledpin, Val); 
sets the digital 
sets the digital 
read the input 
value 
pin 13 as output 
pin 7 as input 
pin 
sets the LED to the button's value 

Arduino Toolchain

Machine generated alternative text:
Verify and Upload 
Combine & 
Compde 
rans 
libraries 
Link 
Hex Fie 
Croat' 

Cross Compilation

Arduino Programs

* A program is called a sketch
* C++ program using Arduino library functions
* C++ is a superset of C

Classes

Classes basically bring data and functions together and give them name

**Week3**

Sketch

$ setup() function

* A sketch does not have a main() function
* Every sketch (Arduino program) has a setup() function
  + Executed once when Arduino is powered up
  + Used for initialization operation
  + Returns no value, takes no arguments
  + Void setup()
  + {
  + ….
  + }

$ loop() function

* Every sketch (Arduino program) has a loop() function
  + Executed iteratively as long as the Arduino is powered up
  + loop() starts executing after the setup() has finished
  + loop() is the main() program control flow
  + Return no value, takes no arguments
  + void loop()
  + {
  + ….
  + }

Pins

* Pins are wires connected to the microcontroller
* Pins are the interface of the microcontroller
* Pins voltages are controlled by a sketch
* Pin voltages can be read by a sketch

Pins can be designated as INPUT or OUTPUT

Output pins

Output pins are controlled by the Arduino

Voltage is determined by your sketch

Other components can be controlled through outputs

Input pins

Input pin are controlled by other components

Arduino reads the voltage on the pins

Allows it to respond to events and data

**Week4**

 Debugging

Debug and Trace

Controllability and observability are required

Controllability

* Ability to control sources of data used by the system
* Input pins, input interfaces (serial, ethernet, etc.)
* Registers and internal memory

Observability

* Ability to observe intermediate and final results
* Output pins, output interfaces
* Registers and internal memory
* Oscilloscope (voltage over time)

Properties of a Debugging Environment

1. Run control of the target

* Start and stop the program execution
* Observe data at stop points

1. Real-time monitoring of target execution

* Non-intrusive in terms of performance

1. Timing and functional accuracy

* Debugged system should act like real system

Debug

Remote Debugger

Machine generated alternative text:
Remote Debuqqer 
• Frontend running on the host 
• Debug Monitor hidden on target 
— Typically triggered when debug events 
occur 
— Hitting a breakpoint. receiving request from 
host, etc. 
• Debug monitor maintains communication link 

Machine generated alternative text:
Remote Debug Tradeoffs 
Advantages: 
l. Good run control using breakpoints to stop 
execution 
2. Debug monitor can alter memory and 
registers 
3. Perfect functional accuracy 
Disadvantages: 
l. Debug interrupts alter timing so real-time 
monitoring is not possible 
2. Need a spare communication channel 
3. Need program in RAM (not flash) to add 
breakpoints 

'

Machine generated alternative text:
Embedded Debug Interfaces 
• Many modern processors include 
embedded debug logic 
• Typically an optional IP block 
• Embedded trace macrocell (ARM) 
• Background debug mode 
(Freescale) 
• Debug logic permanently built into the 
processor 
• A few dedicated debug pins are added 

Machine generated alternative text:
Debug and Trace Features 
• Breakpoints. stopping points in the code 
• Watchpoints. memory locations which trigger 
stop 
• On-the-fly memory access 
• Examine/change internal processor values 
• Single-step through the code 
• Export exceptions to the debugger (hit a 
watchpoint) 
• Export software-generated data (printf) 
• Timestamp information for each event 
• Instruction trace (special purpose HW needed) 

Debug via Serial

Serial Protocols

* Data is transmitted serially
  + Only 1 bit needed (plus common ground)
* Parallel data transmitted serially
* Original bytes/words regrouped by receiver
* Many protocols are serial to reduce pin usage
  + Pins are precious

UART

* Universal Asynchronous receiver/Transmitter
* Used for serial communication between devices
* UART is asynchronous no shared clock (no clock), hence it work over lock distance
* Clock is a square wave
* Asynchronous allows longer distance communication
  + Clock skew is not a problem

UART Applications

* Used by modems to communicate with network
* Computers used to have an RS232 port standard
* Not well used anymore, outside of embedded systems
  + Replaced by USB, ethernet, 12C, SPI
* Simple , low HW overhead
* Built into most microcontrollers

UART Protocol

Machine generated alternative text:
Simple IJART Structure 
Parallel In 
Serial Out 
status 
Parallel Out 
Serial In 
Rx 
status 

* Data is serialized by Tx, deserialized by Rx
* Status indicates the state of the transmit/receive buffers
  + Useful for flow control

Machine generated alternative text:
8 bits 

* First bit is the start bit; indicates the transfer
* Next bits are the data
* Last are two bits are the Stop Bits

Bit duration

* Each bit is transmitted for fixed duration
* The duration must be known to Tx and Rx
* Baud rate (f) determines the duration (T)
* Baud rate is the number of transitions per second
  + Typically measured in "bits per second (bps)"
* T = 1/f
  + F = 9600 baud, T = ~104 microsecond
* Transmission rate is less than baud rate

UART Synchronization

The receiver should know when data send

Machine generated alternative text:
Expected Bit, correct 
Expected Bit, incorrect 
bit 7 
bit 7 
1 
bit 8 
bit 8 
o 
stop 
stop 
stop 

Machine generated alternative text:
Just a glitch 
Start bit detected 
Detection of the start bit is used to synchronize 
Synchronization based on falling edge of start bit 
Start bit is a falling edge 
• Following 0 must be long duration to screen out noise 
Receiver samples faster than baud rate (16x typical) 
• Start bit is indicated by a 0 of at least half period 

UART Parity and Stop

Machine generated alternative text:
Parity Bit 
• Transmission medium is assumed to be 
error-prone 
E-M radiation noise, synchronization 
accuracy 
• Parity bit may be transmitted to check 
for errors 
• Even Parity: Number of I's is even 
• Odd Parity: Number of I's is odd 
• Parity bit is added to ensure even/odd 
parity 
• After data. before stop bit(s) 
• Data 
• Parity bit = l. total parity is odd 

Machine generated alternative text:
Stop Bit 
8 bits 
1 bit 
Stop Bit 
• Receiver expects a I after all data bits and parity bits 
• If I is not received. an error has occurred 

Machine generated alternative text:
Data Throughput vs. Baud 
• Every transmission involves sending signaling bits 
Stop. start. parity 
• Data throughput rate is lower than baud rate 
Signaling bits must be sent 
• 8 data bits. I parity bit, baud rate = 9600 
Send Il bits to send 8 data bits 
Transmission efficiency = 8/11 = 73% 
• Data throughput rate = 9600 * 0.73 = 6981.8 bps 

Machine generated alternative text:
Arduino Serial Communication 
• UART protocol used over the USB cable 
Initialize by using Serial . begin () 
• 
Serial . begin (speed) or 
• 
Serial . begin (speed, config) 
speed is the baud rate 
• 
config sets the data bits, parity, and stop 
• 
bits 
Serial . begin (9600) 
• 
Serial . begin (9600, SERIAL 8N1) 
• 8 data, no parity, 1 stop 
• Usually call Serial. begin in the setup 
function 

Machine generated alternative text:
Sending Text Over Serial 
Use Serial . print ( ) or Serial . println ( ) tC 
print text in the serial monitor 
Serial . print ( "hello") ; 
COM3 (Arduino/Genuino Uno) 
i •he i 0.he 
• Strings are converted to ASCII and sent 
using UART 
Serial monitor interprets bytes as ASCII 
• 