Debre Berhan University

Real times and Embedded Systems

Lecture III: The Arduino Microcontroller

Outline

- Introduction to Embedded Firmware
- The Arduino Platform and C Programming
- Interfacing with the Arduino

Embedded Firmware

- Firmware is **software that provides basic machine instructions that allow the hardware to function and** communicate with other software running on a device.
- It is an un-avoidable part of an embedded system.
- ✓ The embedded firmware can be developed in various methods like: Write the program in high level languages like Embedded C/C++ using an Integrated Development Environment(IDE).
- ✓ An integrated development environment (IDE) is a **software application that helps programmers develop software code** efficiently.
- The embedded firmware is responsible for controlling the various peripherals of the embedded hardware and generating response in accordance with the functional requirements of the product.
- ✓ The embedded firmware will continue serving the assigned task till hardware breakdown occurs or a corruption in embedded firmware.
- ✓ In case of hardware breakdown , the damaged component may need to be replaced.

- The embedded firmware is usually stored in a permanent memory (ROM).
- The firmware design approaches for embedded product is purely dependent on the complexity of the functions to be performed and speed of operation required
- There **exist two** basic approaches for the design and implementation of embedded firmware, namely;

  1. **The Super loop based approach**

  2. The Embedded Operating System based approach

1. The Super loop:

- The Super loop based firmware development approach is Suitable for applications that are not time critical and where the response time is not so important
- Embedded systems where missing deadlines are acceptable.
- The task listed on top on the program code is executed first and the tasks just below the top are executed after completing the first task

The ‘C’ program code for the super loop is given below

```
void main ()
{
Initializations ();
while (1)
{
Task 1 ();
Task 2 ();
:
Task n ();
} }
```

Super loop approach

- **Doesn’t require an Operating** System for task scheduling and monitoring and free from OS.
- Simple and straight forward design
- Reduced memory space.
- Non Real time in execution behavior.

2. Embedded OS based Approach

The **Embedded OS** is responsible for scheduling the execution of user tasks and the allocation of system resources among multiple tasks:

The embedded device contains an Embedded Operating System which can be a Real Time Operating System (RTOS)

- Flight Control Systems is example of embedded devices that runs on **RTOSs**

Embedded firmware Development Languages/Options

- **Assembly Language**
- **High Level Language**
  - ✓ Subset of C (Embedded C)
  - ✓ Subset of C++ (Embedded C++)
  - ✓ Any other high level language

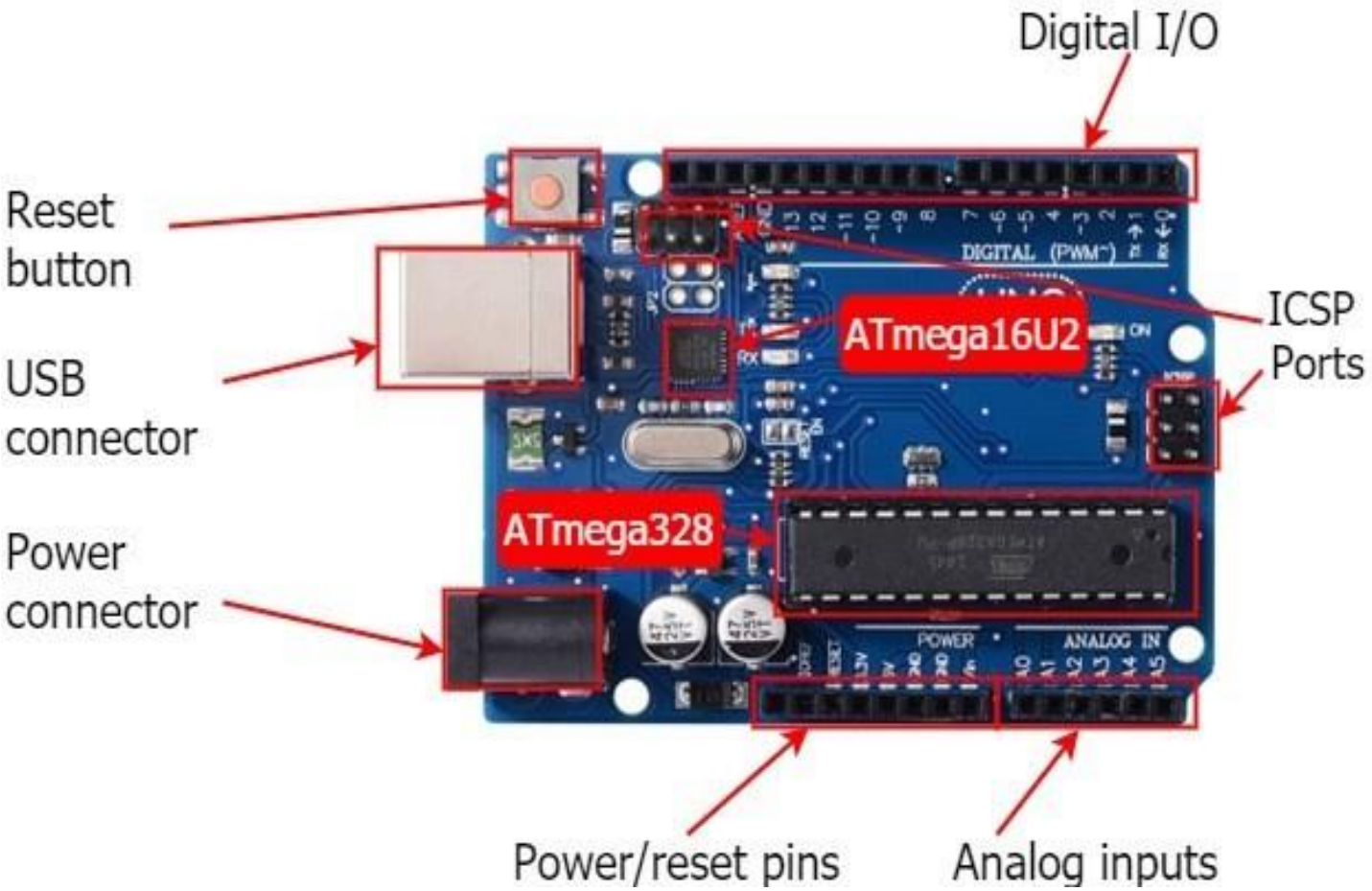
The Arduino Platform and C Programming

Arduino Environment

A development board

- 8-bit microcontroller
- Programming hardware
- USB programming interface
- I/O pins
- Has a microcontroller and USB interface to a PC

Arduino Board

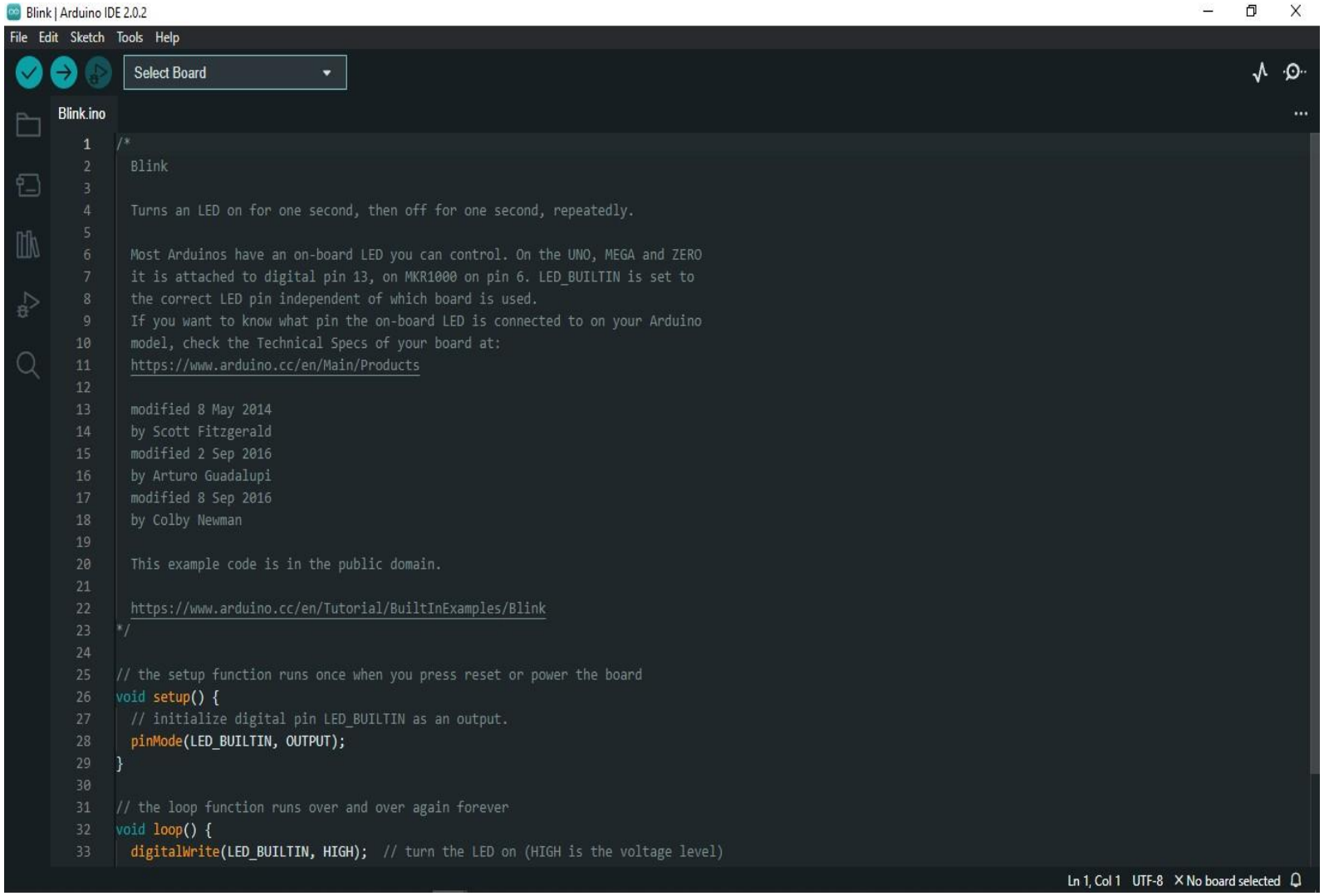


ICSP: In circuit serial programming

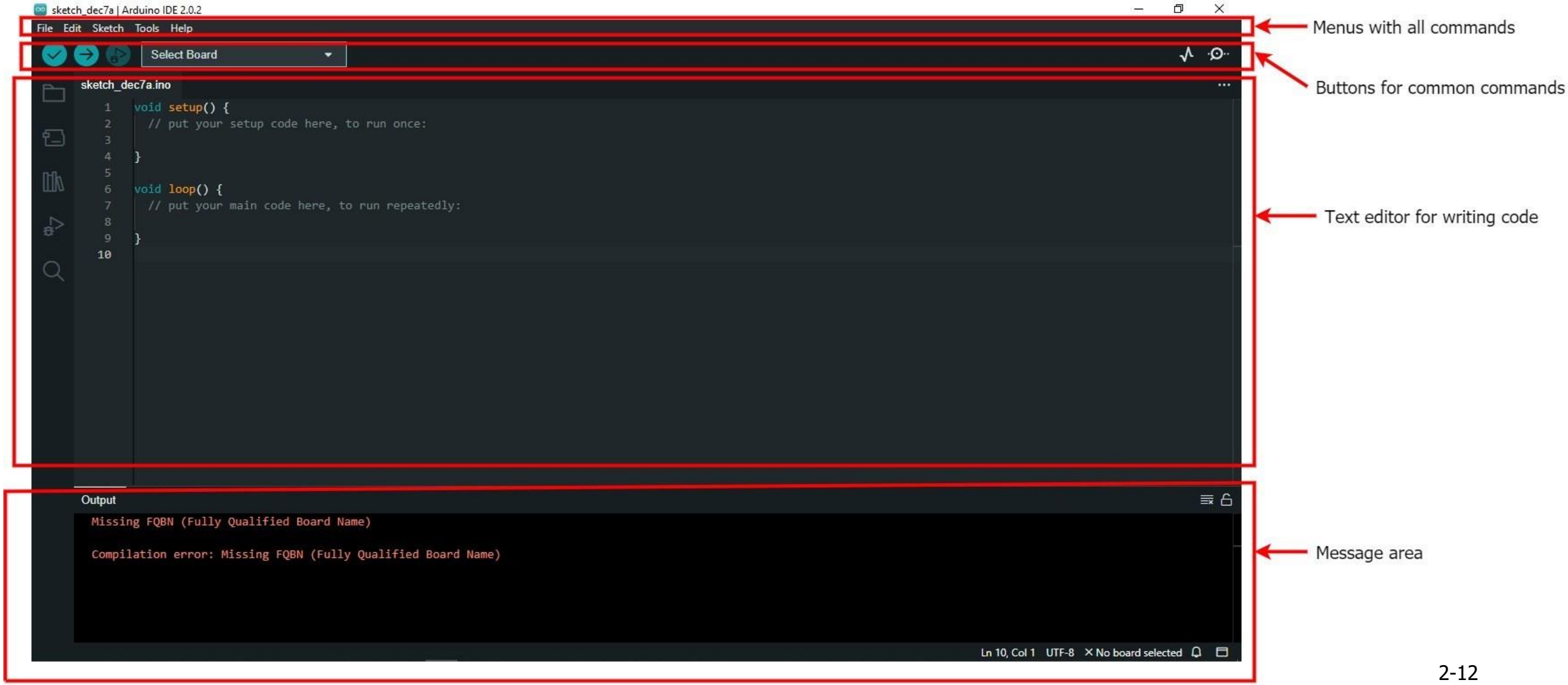
Arduino Environment

A software environment

- Compiler
- Debugger
- Simulator
- Programmer



Arduino Integrated Development Environment (IDE)



Arduino Integrated Development Environment (IDE)

- ✓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

  Display serial data sent from the Arduino

Firmware

Two types of code executing on

A simple microcontroller:

1. Application code

- Executes the system's main functionality
- We write this code

2. Firmware

- supports the main function
- USB interface, Power modes, reset, etc.
- Preprogrammed

Bootloader

- Allows the microcontroller to be programmed
- Manage USB communication, since application programming is via USB

In-Circuit Serial Programming (ICSP)

- A special programming method to program the firmware
- Needed because the bootloader can't reprogram itself

Arduino Board and Microcontroller

Microcontroller	ATmega328
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	40mA
DC Current for 3.3V Pin	50mA
Flash Memory	32 KB (ATmega328) Of which 0.5KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16MHz

C Programming

Getting Started

```c
#include<stdio.h>

int main()
{
    printf("hello, world \n");
    return 0;
}
```

- Prints “hello, world” to the screen
- Type this in with a text editor and save it as hello.c
- An **`#include`** is added to reference basic Arduino libraries

C Programming

Variables

- Names that represent values in the program
- All variables have a type which must be declared

```
int x;
float y;
```

- Type determines how arithmetic is performed, how much memory space is required

Types and Type Qualifiers

- Several built in types, different sizes

Type	Size	Notes
char	1 byte	Fixed size
int	Typically, word size	16 bits minimum
float	Floating point	64 bits, typical
double	Double precision	64, 128 typical

- Type qualifiers exist: short, long
- Char is 8 bits on all platforms

Variable names

- A sequence of visible characters
- Must start with a non numerical character
  - **int testvar1 ✓**
  - **int 1testvar ✗**
- No C language keywords

namespace, using, bool…

- C Programming
- Basic C Operators

Arithmetic/Relational operators

- +,-,*,/
- % is the modulo operator, division
  - remainder
    - Ex. `9%2 = 1, 9%3 = 0`
- ++ (increment), -- (decrement)
- ==,<,>,<=,>=,!=
  - Ex. `If(x < 5)…`

Logical Operators

- &&(AND), ||(OR),!(NOT)
- Treat arguments as one-bit binary values

  0 is FALSE, not-0 is TRUE

  Ex. `If((A==1) && !B)`

C Programming

Conditional Statements (`if else`)

```
if (expression)
    statement1
else
    statement2
```

```
if (expr1)
    stat1
else if (expr2)
    stat2
else
    stat3
```

Example

```
1   #include<stdio.h>
2
3   int main() {
4       int x = 1;
5       if (x == 1)
6           printf("Correct");
7       else
8           printf("Incorrect");
9       return 0;
10  }
```

- *else* is optional
- *expression* is evaluated
  - Execute *statement1* if TRUE, *statement2* if FALSE
- *expr2* evaluated if *expr1* is FALSE

C Programming

Conditional Statements (`switch`)

```
switch (expression) {
    case const_expr1: stat1
    case const_expr2: stat2
    default: stat3
}
```

- *expression* is evaluated, compared to *const_expr*
- Statements are executed corresponding to the first matching expression
- **default** is optional
- Without a **break** statement, the case will not end
- If `x == 0` then both `y = 1;` and `y = 2;` are executed

```
switch (x) {
    case 0:
      y = 1;
    case 1:
      y = 2;
      break;
    case 2:
      Y = 3;
}
```

C Programming

Loops (`for`, `while` and `do-while`)

```
for (expr1; expr2; expr3)
    statement
```

```
expr1;
while (expr2) {
    statement
    expr3;
}
```

```
expr1;
do {
    statement
    expr3;
} while (expr2);
```

- Initialization and increment are built into the `for` loop
- Condition checked at the top of a `for/while` loop
- Condition checked at the bottom of a `do-while` loop

Examples

```c
#include<stdio.h>

int main() {
    int i = 1;
    while (i < 3) {
        printf("%i \n", i);
        i = i + 1;
    }
    return 0;
}
```

```c
#include<stdio.h>

int main() {
    int i;
    for (i = 0; i < 3; i++) {
        printf("%i \n", i);
    }
    return 0;
}
```

C Programming

Loops (`break` and `continue`)

Break jumps to the of a `for`, `while`, `do`, or `case`

```
while (x > 5) {
    y++;
    if (y < 3) break;
    x++;
}
```

`continue` jumps to the next iteration of the loop

```
While (x > 5) {
    y++;
    if (y < 3) continue;
    x++;
}
```

C Programming

Functions

- Functions can replace group of instructions
- Data can be passed to functions are arguments
- Functions can return a value to the caller
- The type of return value must be declared

```c
#include<stdio.h>

int main () {
    int x, y = 2, z = 3;
    y = y + z;
    x = y;
    printf ("%i", x);
    return 0;
}
```

```c
#include<stdio.h>

void foo () {
    int x, y = 2, z = 3;
    y = y + z;
    x = y;
    printf ("%i", x);
}
int main () {
    foo ();
}
```

Function definition

Function call

```c
#include<stdio.h>

void foo (int a, int b) {
    int x, temp;
    temp = a + b;
    x = temp;
    printf ("%i", x);
}
int main () {
    foo (2, 3);
}
```

C Programming

Global Variables

- A variable is global if it's defined outside of any function
- A global variable must be declared as an extern in any function using it
  - Extern not needed if global declaration is before the function
- Variables can be global across files

```
int global_i;

void foo () {
   extern int global_i;
```

Arduino Programs

- A program is called a sketch
- C/C++ program using Arduino library function
- C++ is a super set of C
  - All C programs are legal C++

Classes in Libraries

```
Ethernet.begin(mac);
Serial.begin(speed);
Serial.print("hello");
```

Sketch Structure

`setup()` Function

- A sketch doesn't have a `main()` function
- Every sketch has a `setup()` function
  - Executed once when Arduino is powered up
- Used for initialization operations
- Returns no value, takes no arguments

```
void setup() {
    ...
}
```

`loop()` Function

- Every sketch has a `loop()` function
  - Executes iteratively as long as the Arduino is powered up
  - `loop()` starts executing after setup() has finished
  - `loop()` is the main program control flow
  - Returns no value, takes no argument

```
void loop() {
    ...
}
```

Pins

- Pins are wires connected to the microcontroller
- Pins are the interface of the microcontroller

Digital Vs .Analog

- Some pins are digital only
  - Read digital input, write digital output
  - 0 volts or 5 volts
- Some pins can be analog inputs
  - Can read analog voltage on the pin
  - Useful for analog sensors
- Analog only pins are clearly labeled

Output Pins

Output pins are controlled by theArduino

- Other components can be controlled through outputs

Input Pins

- Input pins are controlled by other components
- Arduino reads the voltage on the pins
- Allows it to respond to events and data

Input/Output (I/O)

- These functions allow accesses to the pins

  `pinMode(pin, mode)`

  - Sets a pin to as act as either an input or an output
  - `pin` is the number of the pin
    - 0 - 13 for the digital pins
    - A0 - A5 for the analog pins
  - `mode` is the I/O mode the pin is set to
    - `INPUT, OUTPUT`

Digital Input

`digitalRead(pin)`

- Returns the state of an input pin
- Returns either **`LOW`** (0 volts) or **`HIGH`** (5 volts)

  Example:

```
int pinval;
pinval = digitalRead(3)
```

- **`pinval`** is set to the state of digital pin = 3

Digital Output

`digitalWrite(pin, mode)`

- Assigns the state of an output pin
- Assigns either **`LOW`**(0 volts) or **`HIGH`**(5 volts)

  Example:

```
digitalWrite(3, HIGH)
```

- Digital pin 3 is set HIGH (5 volts)

Example: Blink sketch(digital output)

```
Void setup () {
     pinMode(13, OUTPUT);
}

Void loop () {
    digitalWrite(13, HIGH);
    delay (1000);
    digitalWrite(13, LOW);
    delay (1000);
}
```

Digital read example

```
//Sets pin 13 as output and pin 7 declared as an input.

int ledPin = 13;  // LED connected to digital pin 13
int inPin = 7;    // pushbutton connected to digital pin 7
int val = 0;      // variable to store the read value
void setup() {
  pinMode(ledPin, OUTPUT);  // sets the digital pin 13 as output
  pinMode(inPin, INPUT);    // sets the digital pin 7 as input
}
void loop() {
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);  // sets the LED to the button's value
}
```

Analog Input

`analogRead(pin)`

- Returns the state of an analog input pin
- Returns an integer from 0 to 1023
- 0 for 0 volts, 1023 for 5 volts

Example:

```
int pinval;
pinval = analogRead(A3);
```

- Pin must be an analog pin

Delay

`delay(msec)`

- Pause the program for msec seconds
- Useful for human interaction

Example:

```
digitalWrite(3, HIGH);
delay(1000);
digitalWrite(3, LOW);
```

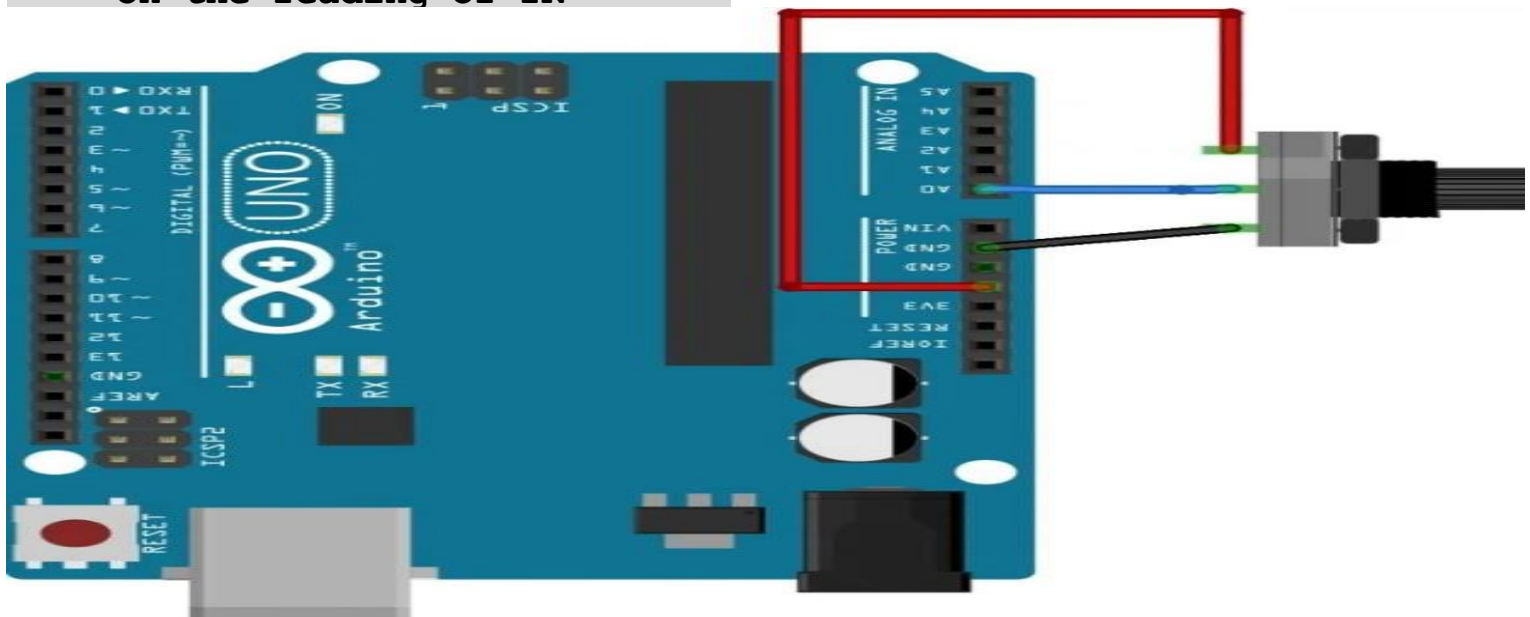
- Pin 3 is **`HIGH`** for 1 second

Example: analog read

int IR= A0

```
Void setup () {
      pinMode(IR, INPUT);
      serial.begin(9600);
}
Void loop () {
 int readval=analogRead(IR);
 serial.println(readval);
  delay(1000);
}
Note: output displays in the
     serial monitor window
     between o to 1023 based
     on the reading of IR
```

```
//Potentiometer is connected at analog pin A0:
int analogPin = A0;
int val = 0;
//variable to store the value read
void setup ()
{
 Serial.begin (9600);
//Setup serial
}
void loop ()
{
    val = analogRead (analogPin);
    //Used to read the input pin
    Serial.println(val);
    delay(1000);
}
```



For loop iteration

```
int timer = 100;  // The higher the number, the slower the timing.
void setup() {
  // use a for loop to initialize each pin as an output:
  for (int thisPin = 2; thisPin < 8; thisPin++) {
    pinMode(thisPin, OUTPUT);
  }
}
void loop() {
    for (int thisPin = 2; thisPin < 8; thisPin++) {
    // turn the pin on:
    digitalWrite(thisPin, HIGH);
    delay(timer);
        digitalWrite(thisPin, LOW);
  }
  // loop from the highest pin to the lowest:
  for (int thisPin = 7; thisPin >= 2; thisPin--) {
    // turn the pin on:
    digitalWrite(thisPin, HIGH);
    delay(timer);
    // turn the pin off:
    digitalWrite(thisPin, LOW);
  }
}
```

conditions

```
// These constants in the const keyword won't change:
const int analogPin = A0;   // pin that the sensor is attached to
const int ledPin = 13;      // pin that the LED is attached to
const int threshold = 400;  //
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize serial communications:
  Serial.begin(9600);
}
void loop() {
  // read the value of the potentiometer:
  int analogValue = analogRead(analogPin);
  // if the analog value is high enough, turn on the LED:
  if (analogValue > threshold) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
  // print the analog value:
  Serial.println(analogValue);
  delay(1);  // delay in between reads for stability
}
```

1.READ ABOUT analog write and pulse width modulation and prepare some note with programming code example within 3-5pages?

2. write an Arduino code to implement blinking of 4 LEDs interchangeably (not at the same time) when push button connected to pin 7 is pressed, with a delay of 1000millisecond each LEDs which are connected to digital output pins 2,4,8,10.

UART

- Universal Asynchronous Receiver/Transmitter
- Used for serial communication between devices
- UART is Asynchronous; no shared clock
- Asynchronous allows longer distance communication
- Used by modems to communicate with network

Serial on Arduino

Arduino Serial Communication

- UART protocol used over the USB cable
- Initialize by using `Serial.begin()`
- `Serial.begin(speed)` or
- `speed` is the baud rate
- `Serial.begin(9600)`
- Usually call `Serial.begin()` in the setup function

Sending Text Over Serial

- Use `Serial.print()` or `Serial.println()` to print text in the serial monitor

```
Serial.print("hello");
```

- Serial monitor interprets bytes as ASCII

Serial on Arduino

Sending Data Over Serial

- Use `Serial.write()`

```
Serial.write(42);
```

- Serial monitor still interprets data as ASCII
- 42 the ASCII value for '*'

Reading Data Over Serial

- Data can be sent to the Arduino via the serial monitor:

- When data is sent it goes into a buffer in the Arduino until it is read
- The Arduino core code contains a nice little data buffer where you can keep throwing data at it and the arduino code will read the data and process it in order.
- `Serial.available()` is used to see how many bytes are waiting in the buffer

```
Int bytenum = Serial.available();
```

- `Serial.read()` returns one byte from the serial buffer

```
Int bval = Serial.read();
```

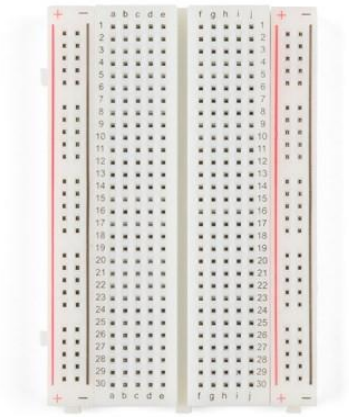
- Returns -1 if no data is available
- `Serial.readBytes()` writes several bytes into a buffer

Interfacing with the Arduino

Electrical Components

- Resistors
- Battery/DC Power
  - Battery, Power, Ground
- Diodes and LEDs
- Switches/Pushbuttons
- Potentiometers

Solderless Breadboard



Sensors

- Allow the microcontroller to receive information about the environment
- Perform operations based on the state of the environment

Sensing the Environment

- Microcontrollers sense voltage
  - `digitalRead(pin)` returns state of a digital pin
  - `analogRead(pin)` returns the analog voltage on a pin
- Sensing logic must convert an environmental effect into voltage

Actuators

- Devices that cause something to happen in the physical world
- Outputs of the embedded devices
  - Visual: LED, LCD, monitor
  - Audio: buzzer, speaker
  - Motion: motor, valve, pump
  - Tactile: heating, cooling
- Many actuators need an analog voltage for complete control
  - DC motor speed controlled by voltage
  - LED brightness controlled by voltage
  - Heating element temperature controlled by voltage
- Arduino cannot generate analog outputs

Arduino Libraries

- Many devices are more complicated than simple sensors/actuators
- Microcontroller (ATMega328) has components which are hard to use
  - Memories, communication interfaces, PWM logic, etc.
- Arduino provides libraries to facilitate their use
- Libraries are also available for external hardware
  - Wifi controller, LCD, controller

EEPROM

- Electronically Erasable Programmable Read-Only Memory
- Only 1024 bytes available on ATMega328

Reading and Writing (EEPROM)

- Access one address at a time
- Each address contains one byte
- `EEPROM.read(address)` returns the content of an address
- `EEPROM.write(address, data)` writes a single byte of data into the address
- Address must be between 0 to 1023

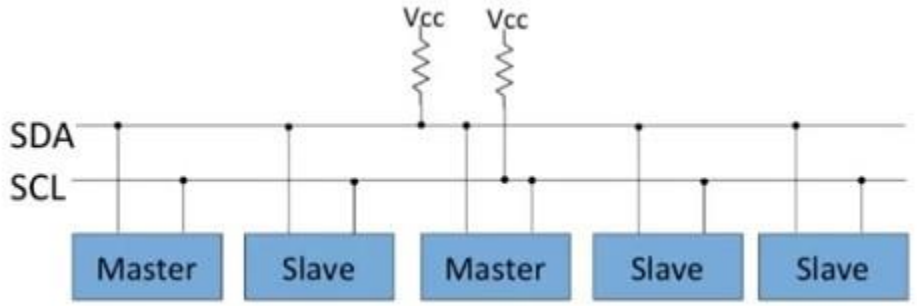
I2C Communication Protocol

- Synchronous, serial protocol
- Multiple masters, multiple slaves
- Two wires: SDA (serial data) and SCL (serial clock)
- SDA and SCL are bidirectional

I2CTerminology

- Master – Initiates and terminates transmission; generates SCL
- Slave – Addressed by the Master
- Transmitter – Placing data on the bus
- Receiver – Reading data from the bus

I2C Network



Wire library

- The wire library is used to access I2C
- `#include <Wire.h>` needed at the top
- `Wire.begin()` function initializes I2C hardware
- Calling `Wire.begin()` with no argument makes the Arduino a Master

Ethernet Library

```
#include
```

- There are many .h files to include
  - `Ethernet.h`
  - `EthernetClient.h`
  - `EthernetServer.h`

Initialize the Ethernet Interface

- Invoke the `Ethernet.begin()` function

Ethernet Client

- Arduino can act as a client, create a client object
  - `EthernetClient client;`
- Needs to connect to a server
  - `result = client.connect(ip, port);`
  - Returns 1 if connection is made, 0 if it is not
- `client.stop();` ends connection