LOW LEVEL PROGRAMMING

IOT Course

Vijay Janapa Reddi
Intel (UT Austin)
Goals

- Programming in Sketch
  - Getting started
  - Language
  - GPIO
  - Debugging
  - Libraries
  - Examples
  - Interrupts

- Communication
  - SPI, I2C
“Arduino”

- Arduino is a popular “open source” single board microcontroller. It is designed to make the process of using electronics in multidisciplinary projects more accessible.

This idea began in Italy and its initial purpose was to make STUDENT design projects more affordable than other prototyping projects at the time.
“Arduino” “Arduino” “Arduino”

• **Arduino** is an open source electronics prototyping platform

• **Open source** – intellectual property, especially computer source code, that is made freely available to the general public by its creators

• **Prototype** – an original or first model of something from which other forms are copied or developed or extended

• **Platform** – hardware design and software framework for developing prototypes
“Galileo” Arduino
What is a *Galileo Arduino*?

The board contains (about 3x4 ¼”):

- 400MHz Intel Quark processor
- 256M RAM, 8M SPI EEROM
- 20 I/O pins
- 2 UARTs
- 2 pushbutton
- 1 LED
- USB client and host
- Ethernet
- micro PCIE
- SD Card
- Open source

“Working with Galileo Course Material” – Great deck of slides!
What do we need?
Getting Started (1)

- Download and install the Arduino Software
Getting Started (2)

- Configure the Arduino software for the correct chip
  - Select the Microcontroller
Getting Started (3)

- Configure the Serial Port
Getting Started (4)

- Open Blink Sketch
Getting Started (5)

• Verify/Compile

• Once it compiles, you must see the following messages in the Status bar and the Program notification Area:

  - Done Compiling.
  - Binary sketch size: 1108 bytes (of a 14336 byte maximum)
Getting Started (6)

• Upload the code to the board

• Once it upload, you must see the following messages in the Status bar and the Program notification Area
Getting Started (7)

- Check… blinking?
Getting Started (7)

- Check… blinking?
On Intel Galileo Board
On Intel Galileo Board
General Flow

```c
void setup() {
  pinMode(ledPin, OUTPUT);  // sets t
}

void loop() {
  digitalWrite(ledPin, HIGH);  // sets t
  delay(1000);  // waits
  digitalWrite(ledPin, LOW);  // sets t
  delay(1000);  // waits
}
```

1. **compile**
2. **Done compiling.**
3. **upload**
4. **TX/RX flash**
5. **blink blink**
6. **sketch runs**
IDE Layout

```c
int ledPin = 13; // LED connected to digital pin 13

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

void loop()
{
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000); // waits for a second
  digitalWrite(ledPin, LOW); // sets the LED off
  delay(1000); // waits for a second
}
```
Status Messages

**Uploading worked**
Done uploading.
Binary sketch size: 1110 bytes (of a 14336 byte maximum)

**Wrong serial port selected**
Serial port '/dev/tty.usbserial-A4001qa8' not found. Did you select the right port?

Wrong microcontroller found. Did you select the right board from the Tools menu?

**Wrong board selected**
neddy cryptic error messages
Key IDE Features

```
// Blink
// Turns on an LED on digital pin 13, then off for one second, repeatedly.
// This example can be used with the public domains

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 is the LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);  // set the LED on
  delay(1000);             // wait for a second
  digitalWrite(13, LOW);   // set the LED off
  delay(1000);             // wait for a second
}
```

- Sketch Name
- Toolbar
- Upload
- Save
- Serial Monitor
- Open
- New
- Stop
- Compile
- Actual Code
Arduino Software

- The Arduino programming platform was designed in JAVA to help newcomers become familiar with programming.

- The language used to write code is C/C++ and only uses TWO functions to make a functionable program.
Arduino Language

- C like syntax, but simplified
- Abstracts the pin naming to numbers
- Trades efficiency for ease of use
- Easy to learn, yet powerful
- Lots of example code
- Easy to reuse C-code from other projects
- Libraries can be written in C++
- Lots of libraries available

**Structure**
- setup()
- loop()

**Control Structures**
- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

**Further Syntax**
- ; (semicolon)

**Variables**

**Constants**
- HIGH
- LOW
- INPUT
- OUTPUT
- INPUT_PULLUP
- LED_BUILTIN
- true
- false
- integer constants
- floating point constants

**Data Types**
- void
- boolean
- char
- unsigned char
- byte
- int
- unsigned int

**Functions**

**Digital I/O**
- pinMode()
- digitalWrite()
- digitalRead()

**Analog I/O**
- analogReference()
- analogRead()
- analogWrite() - PWM

**Due only**
- analogReadResolution()
- analogWriteResolution()

**Advanced I/O**
- tone()
- noTone()
Programming - Syntax

• Similar to ROBOTC, the formatting requirement is the same.

  //  - Single line comment

  /* */  - Multiline comment

  { }  – used to define a block of code that starts and ends.

  ;  - used to define the end of a line of code.
Programming - Variables

- **int (integer)** – this stores a number in 2 bytes (16 bits) and has no decimal places. The value must be between -32,768 and 32,768.

- **long (long)** – Used when the integer is NOT large enough. This takes 4 bytes (32 bits) of RAM and has a range of -2,147,483,648 and 2,147,483,648.

- **boolean (boolean)** – A simple true and false variable. It is useful because it only takes up 1 bit of RAM.

- **float (float)** – Used for floating decimals. It takes 4 bytes of RAM and has a range of -3.4028235E+38 and 3.4028235E+38

- **char (character)** – Stores one character using ASCII code (“A” = 65). Uses 1 byte of RAM
Programming – Math Operators

= (assignment)
% (modulo)
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
Comparison Operators

- These are used to make logical comparisons.

  `==` *(equal to)*
  `!=` *(not equal to)*
  `<` *(less than)*
  `>` *(greater than)*
Programming – Control Structures

• Execute code based on CONDITIONS.

  if(condition) {   } else if (condition) {   } else(condition) {   }

  This will execute the code between the curly braces if the condition is true, and if not test the condition of the “else if”. If that is false, the “else” code will execute.

  for (int i =0; i < #repeats; i ++)  {   }

  Used when you would like to repeat a line of code a specific # of times. Often called a 
  \textit{FOR LOOP}.  

LED Code

```c
// Blink
// Turns on an LED on for one second, then off for one second, repeatedly.

int led = 13;

// the setup routine runs once when you press reset:
void setup() {
    // initialize the digital pin as an output.
    pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000); // wait for a second
    digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
    delay(1000); // wait for a second
}
```
Parts of the Sketch

```cpp
/* Blink
Turns on an LED on for one second, then off for one second, repeatedly.
This example code is in the public domain.
*/

void setup() {
    // initialize the digital pin as an output.
    // Pin 13 has an LED connected on most Arduino boards:
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);  // set the LED on
    delay(1000);             // wait for a second
    digitalWrite(13, LOW);   // set the LED off
    delay(1000);             // wait for a second
}
```
```cpp
void setup() {
    // initialize the digital pin as an output.
    // Pin 13 has an LED connected on most Arduino boards:
    pinMode(13, OUTPUT);
}
```

- The setup function comes before the loop function and is necessary for all Arduino sketches

- The setup header will never change, everything else that occurs in setup happens inside the curly brackets
Setup

#define SETUP

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

- Outputs are declared in setup, this is done by using the pinMode function
  - This particular example declares digital pin # 13 as an output, remember to use CAPS
Loop

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

This function is run AFTER setup has finished. All of the code within the curly braces will be run again, and again, until the power is removed.
Terminology

“sketch” – a program you write to run on an Arduino board

“pin” – an input or output connected to something. e.g. output to an LED, input from a knob.

“digital” – value is either HIGH or LOW. (aka on/off, one/zero) e.g. switch state

“analog” – value ranges, usually from 0-255. e.g. LED brightness, motor speed, etc.
Serial Communication

• Compiling turns your program into binary data (ones and zeros)
• Uploading sends the bits through USB cable to the Arduino
• The two LEDs near the USB connector blink when data is transmitted
  • RX blinks when the Arduino is receiving data
  • TX blinks when the Arduino is transmitting data
Setup Serial

```cpp
void setup() {
    // initialize the digital pin as an output.
    // Pin 13 has an LED connected on most Arduino boards:
    pinMode(13, OUTPUT);
    Serial.begin(9600);
}
```

- Serial communication also begins in setup

- This particular example declares Serial communication at a baud rate of 9600
Setup Serial (2)

```cpp
void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
    Serial.println("Serial Counting");
}

void loop() {
    // put your main code here, to run repeatedly:
    for(int i = 1; i < 1000; i++){
        Serial.println(i);
        delay(1000);
    }
}
Setup Serial (3)

```cpp
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Serial Counting");
}

void loop() {
  // put your main code here, to run repeatedly:
  for(int i = 1; i < 1000; i++){
    Serial.println(i);
    delay(1000);
  }
}
```
Useful Serial Functions

- Serial.begin()
  - Serial.begin(9600)
- Serial.print() or Serial.println()
  - Serial.print(value)
- Serial.read()
- Serial.available()
- Serial.write()
- Serial.parseInt()
IO Pins

- Two states (binary signal) versus multiple states (continuous signal)
- Digital – (LEDs, switches)
- Analog – resistive sensor data
Programming – Digital

- **pinmode (pin, mode)**
  - Used to address the pin # on the Arduino board you would like to use 0-19. The mode can either be INPUT or OUTPUT.

```c
void setup()
{
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```

- **digitalwrite (pin, value)**
  - Once a pin is set to output it can be set to either HIGH (5 Volts) or LOW(0 volts). This basically means turn ON and OFF.

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

Note: There are ways to use the board as analog. Those will be explained later.
Programming – Digital (2)

- **pinMode(pin, mode)**
  - Sets pin to either INPUT or OUTPUT

- **digitalRead(pin)**
  - Reads HIGH or LOW from a pin

- **digitalWrite(pin, value)**
  - Writes HIGH or LOW to a pin

- **Electronic stuff**
  - Output pins can provide 40 mA of current
  - Writing HIGH to an input pin installs a 20KΩ pullup
Programming – Analog

- Can a digital devise produce analog output?

- Analog output can be simulated using pulse width modulation (PWM)

Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley
Programming - Analog

- What if you want to output a voltage other than 0V or 5V?
- Can’t using digital outputs!
- Need a digital to analog converter
Programming - Analog

- Think of the LED dimmer
  - E.g. Galileo does not have any true analog outputs.
  - Dim the LED by blinking it really fast and our eyes will perceive the average intensity
  - If we drive the LED with the top signal it will look dim (10% on) where as if we drive it with the bottom signal it will look bright (90% on)

  \[
  \text{analogWrite(pin, drive)}
  \]
Pulse Width Modulation

- Can’t use digital pins to directly supply say 2.5V, but can pulse the output on and off really fast to produce the same effect.

- The on-off pulsing happens so quickly, the connected output device “sees” the result as a reduction in the voltage.
PWM Duty Cycle

output voltage = 
(on_time / cycle_time) * 5V

Fixed cycle length; constant number of cycles/sec
PMW Pins

- **Command**
  - `analogWrite(pin, value)`

- **Value is duty cycle**
  - between 0 and 255

- **Examples**
  - `analogWrite(9, 128)`
    - for a 50% duty cycle
  - `analogWrite(11, 64)`
    - for a 25% duty cycle
LED Fading Sketch Code

void setup()
{
    pinMode(LED, OUTPUT);  // Set the LED pin as an output
}

void loop()
{
    for (int i=0; i<256; i++)
    {
        analogWrite(LED, i);
        delay(10);
    }
    for (int i=255; i>=0; i--)
    {
        analogWrite(LED, i);
        delay(10);
    }
}
const int potPin = 0; // select input pin for pot. meter
void loop() {
  int val; // The value coming from the sensor
  int percent; // The mapped value
  val = analogRead(potPin); // read the voltage on
  // the pot (val ranges
  // from 0 to 1023)
  percent = map(val, 0, 1023, 0, 100); // percent will
  // range from
  // values 0 to 100
Using Interrupts

- On a standard Arduino board, two pins can be used as interrupts: pins 2 and 3

- This is a way around the linear processing of Arduino
  - Useful to implement state machines using switch statements

- The interrupt is enabled through the following line:

  ```c
  attachInterrupt(interrupt, function, mode)
  ```
  - E.g. `attachInterrupt(0, doEncoder, FALLING);`
Interrupt Example

```c
int led = 77;
volatile int state = LOW;

void setup()
{
    pinMode(led, OUTPUT);
    attachInterrupt(1, blink, CHANGE);
}

void loop()
{
    digitalWrite(led, state);
}

void blink()
{
    state = !state;
}
```
void setup () {
    attachInterrupt (interrupt, function, mode) }

• **LOW** whenever pin state is low
• **CHANGE** whenever pin changes value
• **RISING** whenever pin goes from low to high
• **FALLING** whenever pin goes from low to high
Arduino Timing

- `delay(ms)`
  - Pauses for a few milliseconds

- `delayMicroseconds(us)`
  - Pauses for a few microseconds

Timers

int pin = 13;

void setup()
{
    pinMode(13, OUTPUT);
    digitalWrite(pin, HIGH);
    delay(10 * 60 * 1000);
    digitalWrite(pin, LOW);
}

void loop()
{
}
It's important to keep in mind that there is no such thing as “true” simultaneous execution on an Arduino.

```c
#include <TimerOne.h>

const int LED = 13;

void setup() {
  pinMode(LED, OUTPUT);
  Timer1.initialize(1000000); // Set a timer of length 1000000us
  Timer1.attachInterrupt( blinky); // Runs "blinky" on each interrupt
}

void loop() {
  // Put any other code here.
}

// Timer interrupt function
void blinky()
{
  digitalWrite(LED, !digitalRead(LED)); // Toggle LED State
}
```
Standard Libraries

- EEPROM - reading and writing to "permanent" storage
- Ethernet - for connecting to the internet using the Arduino Ethernet Shield
- Firmata - for communicating with applications on the computer via serial protocol.
- GSM - for connecting to a GSM/GRPS network with the GSM shield.
- LiquidCrystal - for controlling liquid crystal displays (LCDs)
- SD - for reading and writing SD cards
- Servo - for controlling servo motors
- SPI - for communicating with devices using Serial Peripheral Interface (SPI) Bus
- Stepper - for controlling stepper motors
- TFT - for drawing text, images, and shapes on the Arduino TFT screen
- WiFi - for connecting to the internet using the Arduino WiFi shield
- Wire - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.
Create Your Own Libraries

### HEADER FILE

```c
#ifndef Morse_h
#define Morse_h
#include "Arduino.h"

class Morse {
    public:
        Morse(int pin);
        void dot();
        void dash();
    private:
        int _pin;
};
#endif
```

### C FILE

```c
#include "Arduino.h"
#include "Morse.h"

Morse::Morse(int pin) {
    pinMode(pin, OUTPUT);
    _pin = pin;
}

void Morse::dot() {
    digitalWrite(_pin, HIGH);
    delay(250);
    digitalWrite(_pin, LOW);
    delay(250);
}

void Morse::dash() {
    digitalWrite(_pin, HIGH);
    delay(1000);
    digitalWrite(_pin, LOW);
    delay(250);
}
```
Create Your Own Libraries (2)

#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
    morse.dot(); morse.dot(); morse.dot();
    morse.dash(); morse.dash(); morse.dash();
    morse.dot(); morse.dot(); morse.dot();
    delay(3000);
}
Using Switches and Buttons

```cpp
const int inputPin = 2; // choose the input pin

void setup() {
    pinMode(inputPin, INPUT); // declare pushbutton
        // as input
}

void loop() {
    int val = digitalRead(inputPin); // read the
        // input value
}
```
Using Ultrasonic Sensors

- The “ping” sound pulse is generated when the pingPin level goes HIGH for two microseconds.
- The sensor will then generate a pulse that terminates when the sound returns
- The width of the pulse is proportional to the distance the sound traveled
- The speed of sound is 340 meters per second, which is 29 microseconds per centimeter
  - The formula for the distance of the round trip is:
    - \( \text{RoundTrip} = \frac{\text{microseconds}}{29} \)
Using Ultrasonic Sensors (2)

```c
const int pingPin = 5;
const int ledPin = 7; // pin connected to LED

void setup()
{
    Serial.begin(9600);
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    int cm = ping(pingPin);
    Serial.println(cm);
    digitalWrite(ledPin, HIGH);
    delay(cm * 10); // each centimeter adds 10ms delay
    digitalWrite(ledPin, LOW);
    delay(cm * 10);
}
```
Using Ultrasonic Sensors (3)

```c
int ping(int pingPin)
{
    long duration, cm;
    pinMode(pingPin, OUTPUT);
    digitalWrite(pingPin, LOW);
    delayMicroseconds(2);
    digitalWrite(pingPin, HIGH);
    delayMicroseconds(5);
    digitalWrite(pingPin, LOW);
    pinMode(pingPin, INPUT);
    duration = pulseIn(pingPin, HIGH);
    // convert the time into a distance
    cm = microsecondsToCentimeters(duration);
    return cm ;
}

long microsecondsToCentimeters(long microseconds)
{
    // The speed of sound is 340 m/s or 29 microseconds per centimeter.
    // The ping travels out and back, so to find the distance of the
    // object we take half of the distance travelled.
    return microseconds / 29 / 2;
}
```
Motors

• DC Motors
  • just a regular silly ole motor that turns round and round 😊

• Servo Motors
  • smart motor because it moves based on orientation/angles

• Stepper Motors
  • much more flexible and accurate in control mechanisms
int motorPin = 3;

void setup()
{
    pinMode(motorPin, OUTPUT);
    Serial.begin(9600);
    while (! Serial);
    Serial.println("Speed 0 to 255");
}

void loop()
{
    if (Serial.available())
    {
        int speed = Serial.parseInt();
        if (speed >= 0 && speed <= 255)
        {
            analogWrite(motorPin, speed);
        }
    }
}
DC Motor Example

Remember that it isn’t all about programming! Wire your circuit wrongly, and you can fry some board pins and smell them too!
DC Motor Example

Remember that it isn’t all about programming! Wire your circuit wrongly, and you can fry some board pins and smell them too!
Servo Motor Behavior – PWM!
Servo Motor Example to “Sweep”

```c
#include <Servo.h>

int servoPin = 9;
Servo servo;

int angle = 0; // servo position in degrees

void setup()
{
    servo.attach(servoPin);
}

void loop()
{
    // scan from 0 to 180 degrees
    for(angle = 0; angle < 180; angle++)
    {
        servo.write(angle);
        delay(15);
    }
    // now scan back from 180 to 0 degrees
    for(angle = 180; angle > 0; angle--)
    {
        servo.write(angle);
        delay(15);
    }
}
```
Stepper Motor Example

```cpp
#include <Stepper.h>

int in1Pin = 12;
int in2Pin = 11;
int in3Pin = 10;
int in4Pin = 9;

Stepper motor(512, in1Pin, in2Pin, in3Pin, in4Pin);

void setup()
{
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  pinMode(in3Pin, OUTPUT);
  pinMode(in4Pin, OUTPUT);

  // this line is for Leonardo's, it delays
  // the serial interface
  // until the terminal window is opened
  while (!Serial);

  Serial.begin(9600);
  motor.setSpeed(20);
}

void loop()
{
  if (Serial.available())
  {
    int steps = Serial.parseInt();
    motor.step(steps);
  }
}
```
Communication Interfaces
Communication Protocols

**I2C**

**Serial Peripheral Interface (SPI)**
Inter-Integrated Circuit

• 2 Wire serial communication bus, with unique slave IDs

• There are only two lines common to all the devices:
  • Serial Clock (SCL)
  • Serial Data (SDA)

• Can connect multiple devices
Master Writer/Slave Receiver Example
I2C Code Example

Master

```c
#include <Wire.h>

void setup()
{
    Wire.begin(); // join i2c (address optional for master)
}

byte x = 0;

void loop()
{
    Wire.beginTransmission(4); // transmit to device #4
    Wire.write("x is "); // sends five bytes
    Wire.write(x); // sends one byte
    Wire.endTransmission(); // stop transmitting
    x++;
    delay(500);
}
```

Slave

```c
#include <Wire.h>

void setup()
{
    Wire.begin(4); // join i2c bus with address #4
    Wire.onReceive(receiveEvent); // register event
    Serial.begin(9600); // start serial for output
}

void loop()
{
    delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
    while(1 < Wire.available()) // loop through all but the last
    {
        char c = Wire.read(); // receive byte as a character
        Serial.print(c); // print the character
    }
    int x = Wire.read(); // receive byte as an integer
    Serial.println(x); // print the integer
}
```
Serial Peripheral Interface (SPI)

• 3-Wire (plus 1 chip-select) with unique chip-select lines

• Typically there are three lines common to all the devices:
  • MISO (Master In Slave Out)
    • The Slave line for sending data to the master,
  • MOSI (Master Out Slave In)
    • The Master line for sending data to the peripherals,
  • SCK (Serial Clock)
    • The clock pulses which synchronize data transmission generated by the master
  • SS (Slave Select)
    • The pin on each device that the master can use to enable and disable specific devices.
    • When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.
SPI Peripheral Types

- Converters (ADC, DAC)
- Memories (EEPROM, RAM’s, Flash)
- Sensors (Temperature, Humidity, Pressure)
- Real Time Clocks
- Misc- Potentiometers, LCD controllers, UART’s, USB controller, CAN controller, amplifiers
SPI Code Example

#include <SPI.h>

// By default, 11 = MOSI, 12 = MISO, 13 = CLK
// set pin 10 as the slave select
const int slaveSelectPin = 10;

void setup() {
  pinMode(slaveSelectPin, OUTPUT);
  // initialize SPI:
  SPI.begin();
}

void loop() {
  …
  digitalPotWrite(channel, level);
  …
}

void digitalPotWrite(int address, int value) {
  // take the SS pin low to select the chip:
  digitalWrite(slaveSelectPin, LOW);
  // send in the address and value via SPI:
  SPI.transfer(address);
  SPI.transfer(value);
  // take the SS pin high to de-select the chip:
  digitalWrite(slaveSelectPin, HIGH);
}
The End!