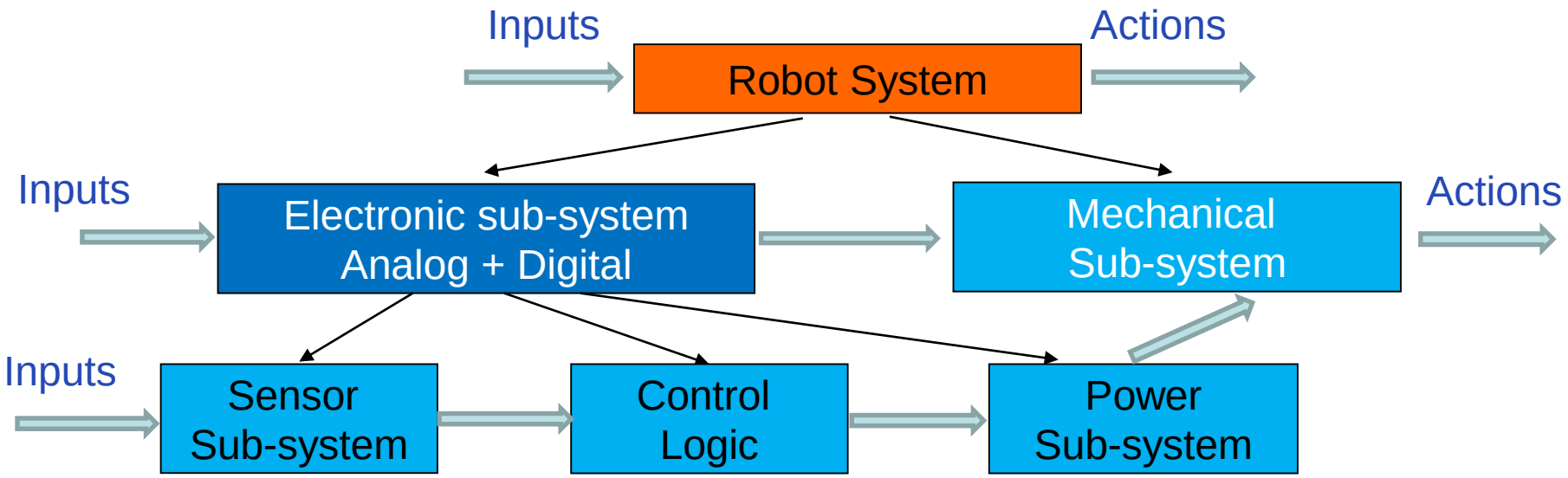
ELEC 1100: Introduction to Electro-Robot Design

Lecture 6: Arduino Hardware & Software + Programming Language (I&II)

CHEN Wei, Dept. of ECE, HKUST

ELEC1100 ROADMAP



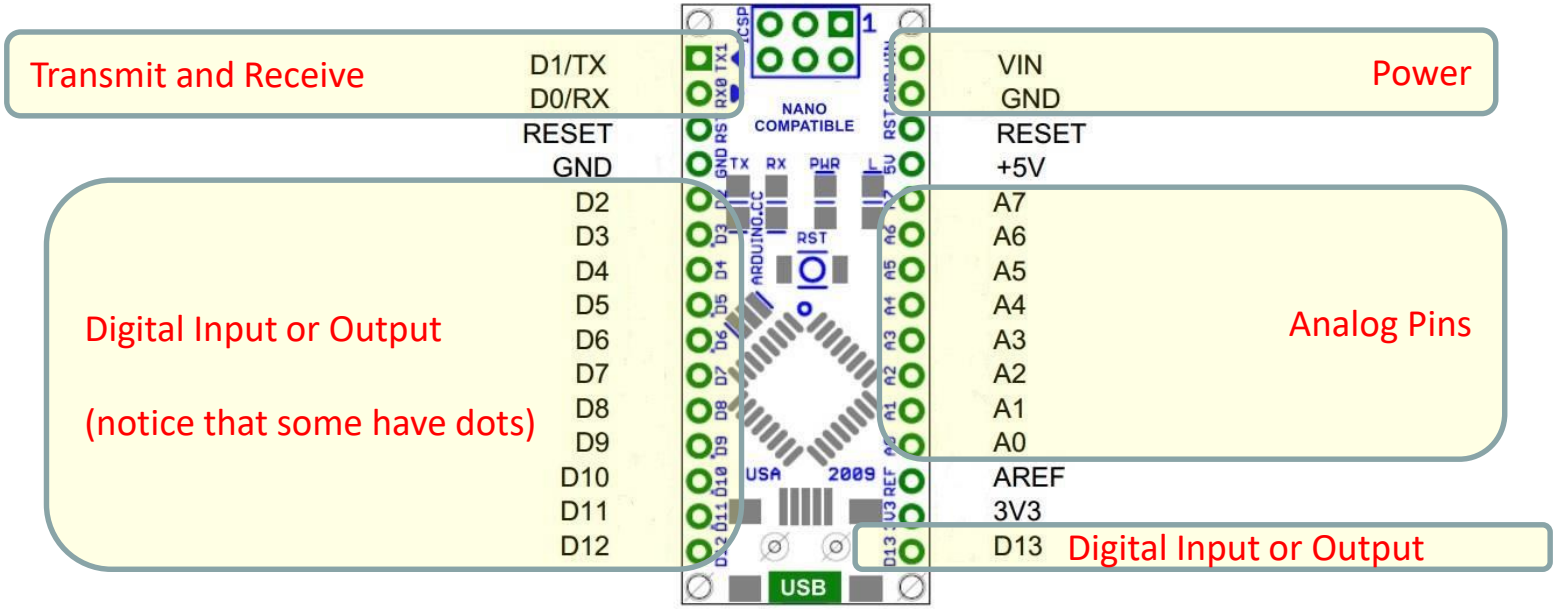
Sensor Basics:
Wk2: Sensor Basic

Combinational/Sequential Logic:
Wk3: Robot Brain: Logic Gate & Logic Operation
Wk4:Sequential Logic
Wk5:MCU & Programming Language

Basic electronics:
Wk1: Basic Electronics & DC Sources

Motor Power Supply:
Wk1: Pulse Signal & PWM Control
Wk2: Transistor and H-Bridge

Nano Board


Transmit and Receive
D1/TX
D0/RX
RESET
GND
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
Digital Input or Output
(notice that some have dots)
VIN
GND
RESET
+5V
A7
A6
A5
A4
A3
A2
A1
A0
AREF
3V3
D13
Power
Analog Pins
Digital Input or Output
NANO
COMPATIBLE
USB

A SAMPLE PROGRAM

```
// Library
//#include <communication.h>
//#include <motor_control.h>

// System setup: Run when we start or reset the Ardunio board:
void setup()
{
        pinMode(13, OUTPUT);            // initialize the digital pin as an output.
}

//  Control the Ardunio board
void loop()                                     // the loop routine runs over and over again forever:
{
        digitalWrite(13, HIGH);         // turn the LED on (HIGH/LOW are the voltage levels)

        delay(1000);                    // wait for a second

        digitalWrite(13, LOW);          // turn the LED off by making the voltage LOW

        delay(1000);                     // wait for a second
}
```

A SAMPLE PROGRAM

```
// Library
//#include <communication.h>
//#include <motor_control.h>

// System setup: Run when we start or reset the Ardunio board:
void setup()
{
        pinMode(13, OUTPUT);            // initialize the digital pin as an output.
}

//  Control the Ardunio board
void loop()                             // the loop routine runs over and over again forever:
{
        digitalWrite(13, HIGH);         // turn the LED on (HIGH/LOW are the voltage levels)

        delay(1000);                    // wait for a second

        digitalWrite(13, LOW);          // turn the LED off by making the voltage LOW

        delay(1000);                     // wait for a second
}
```

A SAMPLE PROGRAM

How to include the library files?

```
// Library
//#include <communication.h>
//#include <motor_control.h>
```

- This section of the program “includes” the “library” files to the program.
- Those libraries are predefined for the board to provide some commonly used functions.
- For this example, we do not need any libraries. Note that
  - “//” is the comment operator. What follows it will be treated as comments. So, the two libraries <communication.h> <motor_control.h> are not included here.
  - You may need to include them for your future lab exercises and projects.

A SAMPLE PROGRAM

```
// Library
//#include <communication.h>
//#include <motor_control.h>

// System setup: Run when we start or reset the Ardunio board:
void setup()
{
        pinMode(13, OUTPUT);            // initialize the digital pin as an output.
}

//  Control the Ardunio board
void loop()                                     // the loop routine runs over and over again forever:
{
        digitalWrite(13, HIGH);         // turn the LED on (HIGH/LOW are the voltage levels)

        delay(1000);                    // wait for a second

        digitalWrite(13, LOW);          // turn the LED off by making the voltage LOW

        delay(1000);                     // wait for a second
}
```

A SAMPLE PROGRAM

How to setup the board?

```
// System setup: Run when we start or reset the Ardunio board:
void setup()
{
        pinMode(13, OUTPUT);        // initialize the digital pin as an output.
}
```

- This section of the program sets up the Ardunio board (initialization).
- In the above example, we initialize pin13 to be the “OUTPUT” pin.
- The function utilized is
  - pinMode(index, mode);
  - It configures a specified pin to behave either as an INPUT or OUTPUT pin.
  - “index” denotes the index number of the pin. In total, there are 14 digital pins.
  - “mode” denotes the operation mode (INPUT or OUTPUT).

A SAMPLE PROGRAM

```
// Library
//#include <communication.h>
//#include <motor_control.h>

// System setup: Run when we start or reset the Ardunio board:
void setup()
{
        pinMode(13, OUTPUT);            // initialize the digital pin as an output.
}

//  Control the Ardunio board
void loop()                                     // the loop routine runs over and over again forever:
{
        digitalWrite(13, HIGH);         // turn the LED on (HIGH/LOW are the voltage levels)

        delay(1000);                    // wait for a second

        digitalWrite(13, LOW);          // turn the LED off by making the voltage LOW

        delay(1000);                     // wait for a second
}
```

A SAMPLE PROGRAM

Board Control

```
//  Control the Ardunio board
void loop()                                          // the loop routine runs over and over again forever:
{
        digitalWrite(13, HIGH);                      // turn the LED on (HIGH/LOW are the voltage levels)
        delay(1000);                                 // wait for a second
        digitalWrite(13, LOW);                       // turn the LED off by making the voltage LOW
        delay(1000);                                  // wait for a second
}
```

- This section of the program controls the board with the defined operations.
- There are two functions utilized for this example program.
  - digitalWrite(13, HIGH);
  - delay(1000);

FUNCTIONS

Function: digitalWrite(int pin, int value)

- Inputs
  - int pin: Integer type pin number
    - Range: 0-13
  - int value: integer type value
    - Range: HIGH, LOW
    - HIGH and LOW are built-in constant integers

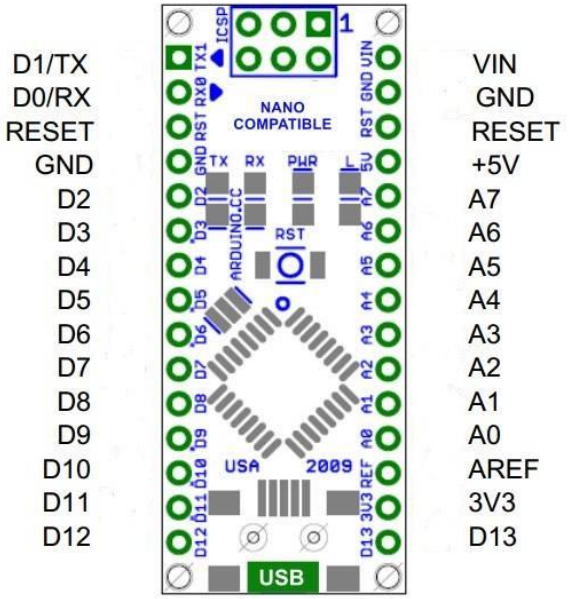
- Function
  - Output one of the two values (HIGH/LOW=5V/0V) to the desired pin
  - We call it digitalWrite because there only two voltage levels can be set as in logic circuit

FUNCTIONS

Function: analogWrite(int pin, int value)

- Inputs
  - int pin: Integer type pin number
    - Range: 3, 5, 6, 9, 10, 11
      (only 6 of the 14 digital pins support analogWrite)
  - int value: integer type value
    - Range: 0~255

- Function
  - Output a signal whose power is controlled by the an integer value (0~255)
  - Recall the power control by pulse signal



FUNCTIONS

Function: delay(int time)

- Inputs
  - int time: Integer type time value
    - Range: 0 - 4,294,967,295 (i.e. 1,193 hours or 49.7 days)
    - Unit: Milliseconds

- Function
  - Pause the program for a given amount of time
  - Normally utilized in loop{} to achieve a periodic operation

- A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output.
- Programming languages generally consist of instructions for a computer.

ELEMENTS

- All programming languages have some primitive building blocks for the description of data and the processes or transformations applied to them.
- These primitives are defined by syntactic and semantic rules which describe their structure and meaning respectively.
- The syntax of a language describes the possible combinations of symbols that form a syntactically correct program.
- Semantics refers to the meaning of languages, as opposed to their form (syntax).

Syntax vs. Semantics

- Syntax: The set of legal structures and commands that can be used in a particular programming language.
- Syntax error or compiler error: A problem in the structure of a program that causes the compiler to fail.
  - "I you love!"
- Not all syntactically correct programs are semantically correct.
  - "The sun sleeps well in day time."

THE LANGUAGE OF A COMPUTER

- Uses digital signals
  - all 0's and 1's (binary)
  - bits (**BI**nary digi**T**s)
- Data and commands stored in binary
  - 8 bits in a byte
  - ASCII character stored in a byte
  - Integers stored in 2 or 4 bytes

EVOLUTION OF PROGRAMMING LANGUAGE

- Early computers programmed in machine languages
  - All binary numbers

- Assembly language used mnemonic codes
  - Codes translated into machine language by a program called the "assembler"

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

EVOLUTION OF PROGRAMMING LANGUAGE

- High level languages read like combination of English and algebra

```
void loop()
{
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```

- They will be translated into machine language by a program called a compiler

ARDUINO PROGRAMMING LANGUAGE

- Arduino programming language can be divided in three main parts: structure, values (variables and constants), and functions.
- Structure: the elements of Arduino (C++) code.
  - Sketch
    - loop()
    - setup()
  - Control Structure
    - break continue do...while
    - for if...else switch...case
  - Operators
    - % (remainder) * (multiplication)
    - + (addition) - (subtraction)
    - / (division) = (assignment operator)

ARDUINO PROGRAMMING LANGUAGE: VALUES

- Values: Arduino data types and constants.
  - Constants
    - HIGH | LOW
    - INPUT | OUTPUT | INPUT_PULLUP
    - LED_BUILTIN
    - true | false
  - Data types
    - String
    - array
    - bool
    - boolean
    - byte
    - char
    - double
    - float
    - int
    - long
    - short
    - unsigned char
    - void
    - …….

ARDUINO PROGRAMMING LANGUAGE: FUNCTIONS

- Functions: controlling the Arduino board and performing computations.
  - Digital/Analog I/O
    - digitalRead()
    - digitalWrite()
    - pinMode()
  - Math
    - % (remainder)
    - * (multiplication)
    - + (addition)
    - - (subtraction)
    - / (division)
    - = (assignment operator)
  - Math
    - serial
    - stream

VARIABLES

- There are several categories of concepts for variables.
  - Constants
  - Conversion
  - Data Types
  - Variables Scope & Qualifiers
  - Utilities
- We will only introduce those key to our project. Please read the others by yourself.

CONSTANTS

- Constants are predefined expressions in the Arduino language.
  - They are used to make the programs easier to read.

- Logical Levels: true and false (lower case)
  - false is defined as 0 (zero)
  - True is often said to be defined as 1. In a Boolean sense, any non-zero integer is true.

Digital Pins Modes: INPUT and OUPUT

- INPUT
  - Arduino pins configured as INPUT with pinMode() are said to be in a high-impedance state.
  - Pins configured as INPUT make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 Megohms in front of the pin.
  - This makes them useful for reading a sensor.

Digital Pins Modes: INPUT and OUPUT

- OUTPUT
  - Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state.
  - They can provide a substantial amount of current to other circuits, i.e., to source (provide current) or sink (absorb current) up to 40 mA (milliamps) of current to other devices/circuits.
  - This makes them useful for powering LEDs because LEDs typically use less than 40 mA.
  - Pins configured as outputs can be damaged or destroyed if they are connected to either the ground or positive power rails.

Pin Levels: HIGH and LOW (upper case)

- HIGH
  - If a pin is configured as INPUT with pinMode() and read with digitalRead(), Auduino will report HIGH if
    - a voltage greater than 3.0V is present (5V boards)
    - a voltage greater than 2.0V is present (3.3V boards)
  - If a pin is configured as OUTPUT and set to HIGH with digitalWrite(), the pin is at
    - 5.0V (5V boards)
    - 3.3V (3.3V boards)
    - In this state, it can source current, e.g. light an LED.

Pin Levels: HIGH and LOW (upper case)

- LOW
  - If a pin is configured as INPUT with pinMode() and read with digitalRead(), Auduino will report LOW if
    - a voltage less than 1.5V is present (5V boards)
    - a voltage less than 1.0V is present (3.3V boards)
  - If a pin is configured as OUTPUT and set to LOW with digitalWrite(), the pin is at
    - 0V (5V and 3.3V boards)
    - In this state, it can sink current.

DATA TYPES

- Data Type: classification of data which tells the compiler or interpreter how the programmer intends to use the data.
  - String
  - String()
  - array
  - bool
  - boolean
  - byte
  - char
  - double
  - float
  - int
  - long
  - short
  - unsigned char
  - unsigned int
  - unsigned long
  - void
  - word

We only introduce several important types for this course.

DATA TYPES

- int: Integers are your primary data-type for number storage.
  - Description: For nano, an int stores a 16-bit (2-byte) value, -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).
  - Syntax
    - int var = val
    - var - your int variable name
    - Val – the value you assign to that variable
    - Example: int ledPin = 13;

DATA TYPES

- A bool holds one of two values, true or false.
- Example Code

```
int LEDpin = 5;        // LED on pin 5
int switchPin = 13;    // momentary switch on 13, other side connected to ground

bool running = false;

void setup()
{
  pinMode(LEDpin, OUTPUT);
  pinMode(switchPin, INPUT);
  digitalWrite(switchPin, HIGH);      // turn on pullup resistor
}

void loop()
{
  if (digitalRead(switchPin) == LOW)
  {  // switch is pressed - pullup keeps pin high normally
    delay(100);                         // delay to debounce switch
    running = !running;                 // toggle running variable
    digitalWrite(LEDpin, running);      // indicate via LED
  }
}
```

VARIABLE SCOPE AND QUALIFIERS

- const: is a variable qualifier that modifies the behavior of the variable, making it “read-only”. This means that the variable can be used just as any other variable of its type, but its value cannot be changed.
- Example Code:

```
const float pi = 3.14;
float x;

// ....

x = pi * 2;     // it's fine to use consts in math

pi = 7;         // illegal - you can't write to (modify) a constant
```

SCOPE

- A global variable is one that can be seen by every function in a program. Local variables are only visible to the function in which they are declared. In the Arduino environment, any variable declared outside of a function (e.g. setup(), loop(), etc.), is a global variable.
- Local variables are a useful way to insure that only one function has access to its own variables.
- Example Code:

```
int gPWMval;  // any function will see this variable

void setup()
{
  // ...
}

void loop()
{
  int i;    // "i" is only "visible" inside of "loop"
  float f;  // "f" is only "visible" inside of "loop"
  // ...

  for (int j = 0; j <100; j++){
  // variable j can only be accessed inside the for-loop brackets
  }

}
```

EXAMPLE CODE FOR PROJECT

```
// assign meaningful names to pins to be used

// analog pins to take sensor input
const int pinWallSensor = A0;      //pin A0 to take the wall sensor input
const int pinLeftSensor = A1;      //pin A1 to take the left sensor input
const int pinMidSensor = A2;       //pin A2 to take the mid sensor input
const int pinRightSensor = A3;     //pin A3 to take the right sensor input

// digital pins to output control signal
const int pinLDir = 2;           //pin D2 to control direction of the left motor
const int pinRDir = 3;           //pin D3 to control direction of the right motor
const int pinLQ0 = 5;            //pin D5 pin 5~8 to control the power of the left motor
const int pinLQ1 = 6;            //pin D6
const int pinLQ2 = 7;            //pin D7
const int pinLQ3 = 8;            //pin D8

// define signal input variables
int wallSensor = 1;
int leftSensor = 1;
int midSensor = 1;
int rightSensor = 1;

void setup() {
// define pins as input and output.
pinMode(pinLeftSensor, INPUT);
pinMode(pinRightSensor, INPUT);
pinMode(pinLDir, OUTPUT);
pinMode(pinRDir, OUTPUT);
...}
```

Questions ?!