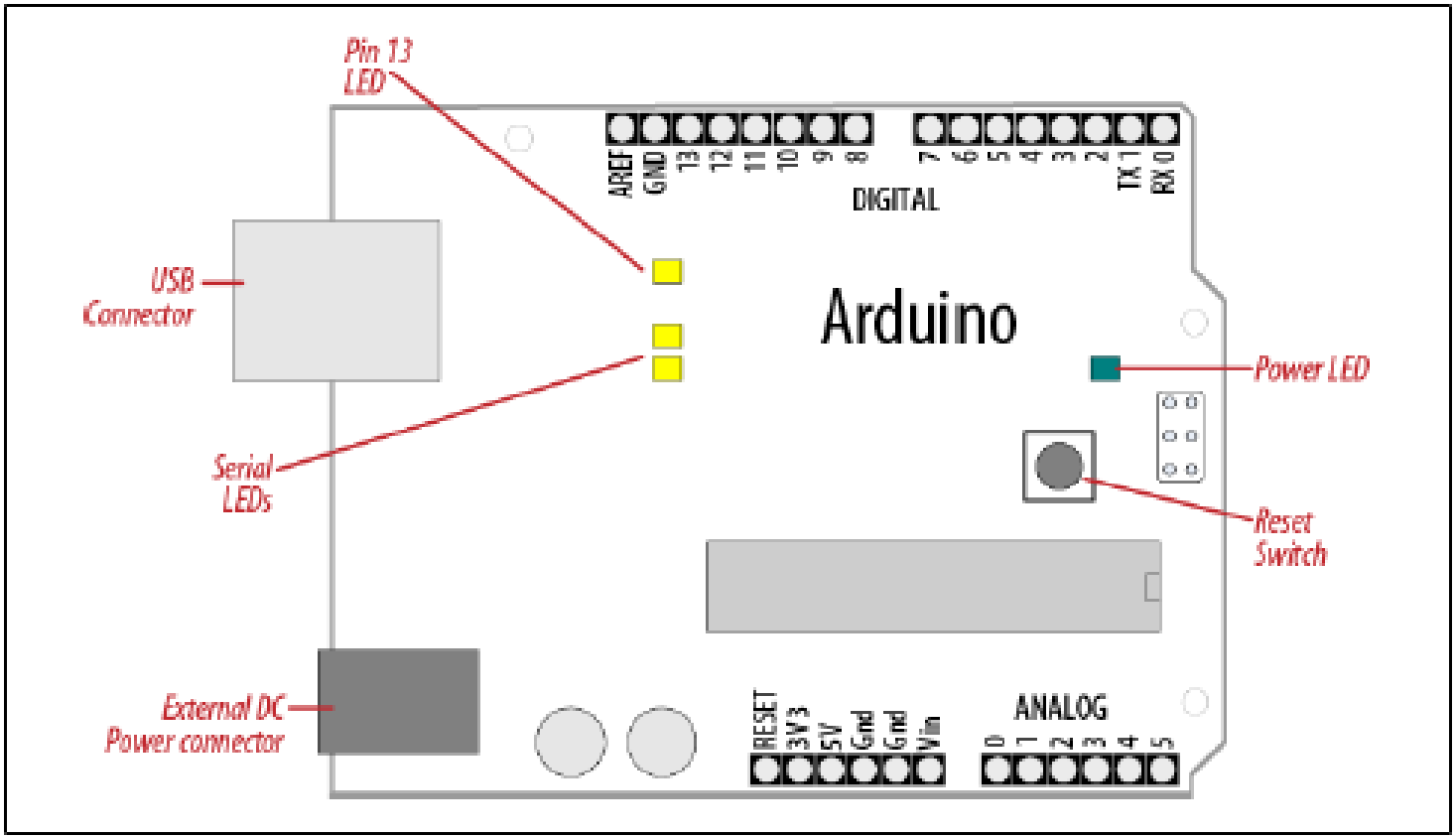
PROGRAMMING WITH ARDUINO

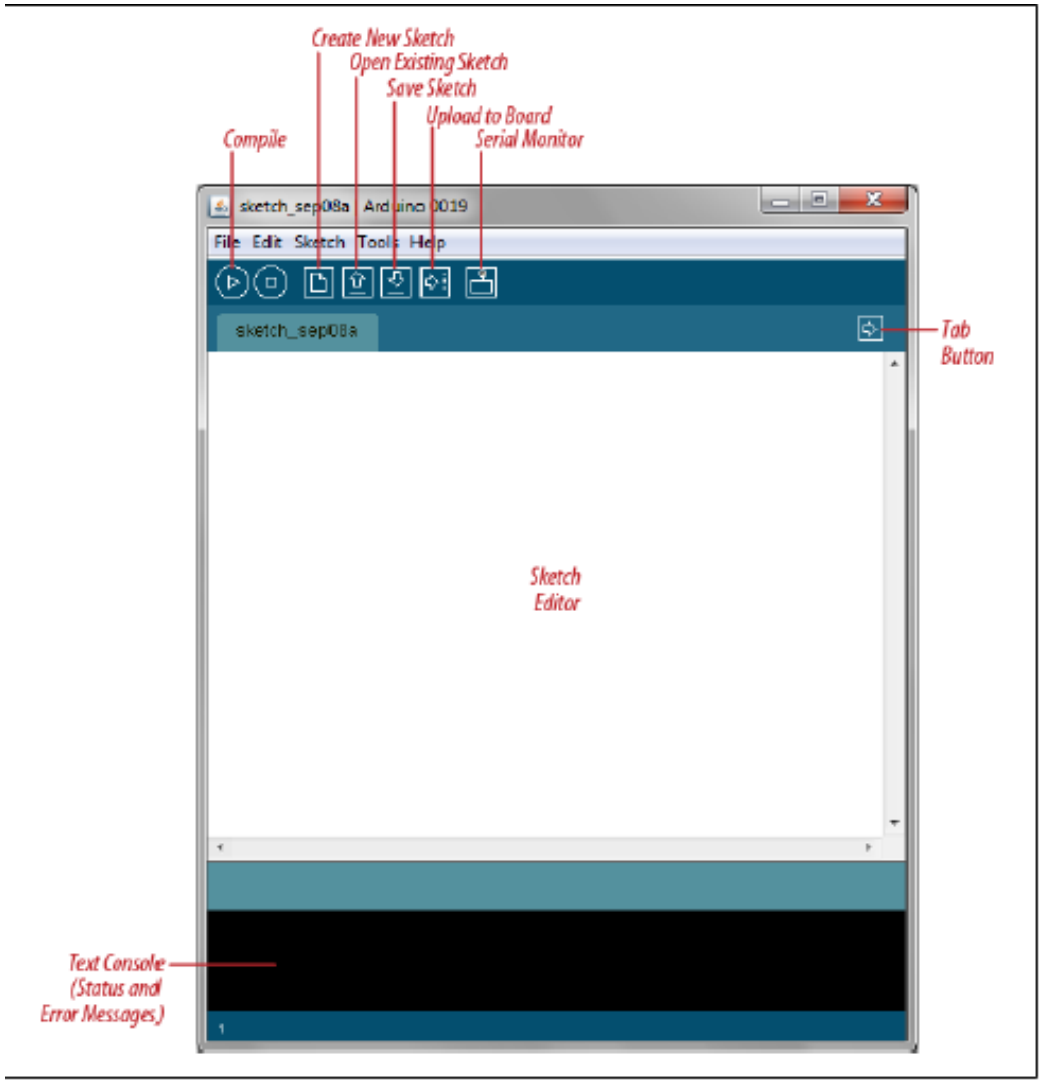
Arduino

- An open-source hardware platform based on an Atmel AVR 8-bit microcontroller and a C++ based IDE
- Over 300000 boards have been manufactured
- Arduino Due is based on a 32-bit ARM Cortex

Typical Arduino Board



Arduino IDE


Create New Sketch
Open Existing Sketch
Save Sketch
Upload to Board
Serial Monitor
Compile
sketch_sep08a | Arduino 0019
File Edit Sketch Tools Help
sketch_sep08a
Tab Button
Sketch Editor
Text Console (Status and Error Messages)

Important functions

- Serial.println(value);
  - Prints the value to the Serial Monitor on your computer
- pinMode(pin, mode);
  - Configures a digital pin to read (input) or write (output) a digital value
- digitalRead(pin);
  - Reads a digital value (HIGH or LOW) on a pin set for input
- digitalWrite(pin, value);
  - Writes the digital value (HIGH or LOW) to a pin set for output

OUTLINE

- Essential Programming Concepts
  - Delay
  - Infinite Loop
- General Input/Output
  - Polling or Busy/Wait I/O
  - Interrupt Processing
- Timers and Internal Inteerrupts
- High-Level Language Extensions
- Code Transformations for Embedded Computing
  - Loop Unrolling
  - Loop Merging
  - Loop Peeling
  - Loop Tiling

DELAY (1/3)

- Delays are essential in embedded systems, unlike high-performance systems where we want the program to execute as fast as possible
- Delays are used to synchronize events, or read inputs with a specific sampling freqency (more on Bus/Wait I/O)

DELAY (2/3)

- Arduino example:

```
delay(int milliseconds)
//creates a delay in ms
delayMicroseconds(int microseconds)
//creates a delay in µs

delay(1000); //one second delay
delayMicroseconds(10); //10 µs delay
```

DELAY (3/3)

- Okay, so how do we build a delay function?
- Our reference is the system clock frequency
- We use a register or a timer to measure ticks
- Each tick is 1/frequency
- Example: Assuming a 16-bit processor, an increment and a jump instruction is 1-cycle each and a 10 MHz system clock, build a 1-sec delay:
- T = 1/10 MHz = 100 ns
- 1 s/100 ns = 10,000,000

```
     int i=5000000; //2 ticks per iteration
BACK: i--;
     if (i!=0) goto BACK;
```

Infinite Loop (1/2)

- Embedded Systems are mostly single-functioned
- Their core application never terminates
- Infinite loops are not forbidden as long as they are done correctly

Infinite Loop (2/2)

```
void main()
{
   light enum {RED, ORANGE, GREEN};
   loop: light = RED;  //no exit from loop!
      delay(20000); //20-sec red
      light = ORANGE;
      delay(2000);  //2-sed orange
      light = GREEN;
      delay(20000); //20-sec green
      goto loop;    //just repeat sequence
}
```

Example: Arduino

- Because goto is considered a bad programming practice and in order to avoid the label, Arduino provides an infinite loop function

```
light enum {RED, ORANGE, GREEN};
void loop(){ //the whole function repeats
  light = RED;  //no need for label!
  delay(20000);   //20-sec red
  light = ORANGE;
  delay(2000);    //2-sed orange
  light = GREEN;
  delay(20000);   //20-sec green
}
```

General Input/Output

- Embedded processors receive input from their environment (sensors) and produce output that change that environment (actuators) or give information (indicators)
- These require reading and writing to single or multiple bit I/O ports

High-Level Language Extensions for Embedded Computing

Language Extensions

- Commands in high-level languages that are not part of the language standard (for example ANSI C)
- Useful for accessing low-level (assembly) functionality from a high-level language or simplifying desirable embedded system functionality that is considered bad programming practice (such as infinite loops)
- Essentially functions the user calls
- Generally machine and compiler-dependent, it is possible to write an equivalent function for another machine

Examples (Arduino)

```
void loop (){
digitalWrite(77,HIGH);
delay(500);
digitalWrite(77, LOW);
}
```

This creates an infinite loop without the programmer using a goto (which is bad programming practice)

These turn pin 77 on and off (assuming there is something like a LED there without using logic instructions and I/O ports

This creates a half-second (500 ms) delay without directly accessing the timers

Using LEDs

```
void setup()
{
pinMode(77, OUTPUT);   //configure pin 77 as output
}
// blink an LED once
void blink1()
{
digitalWrite(77,HIGH); // turn the LED on
delay(500); // wait 500 milliseconds
digitalWrite(77,LOW); // turn the LED off
delay(500); // wait 500 milliseconds
}
```

Creating infinite loops

```
void loop()  //blink a LED repeatedly
{
digitalWrite(77,HIGH); // turn the
  LED on
delay(500); // wait 500 milliseconds
digitalWrite(77,LOW); // turn the LED
  off
delay(500); // wait 500 milliseconds
}
```

Using switches and buttons

```
const int inputPin = 2; // choose the input pin
void setup() {
pinMode(inputPin, INPUT); // declare pushbutton as input
}
void loop(){
int val = digitalRead(inputPin); // read input value
}
```

Reading analog inputs and scaling

```
const int potPin = 0; // select the input pin for the
   potentiometer
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 0 to 100.
```

Creating a bar graph using LEDs

```
const int NoLEDs = 8;
const int ledPins[] = { 70, 71, 72, 73, 74, 75, 76, 77};
const int analogInPin = 0; // Analog input pin const int
   wait = 30;
const boolean LED_ON = HIGH;
const boolean LED_OFF = LOW;
int sensorValue = 0; // value read from the sensor
int ledLevel = 0; // sensor value converted into LED
   'bars'
void setup() {
for (int i = 0; i < NoLEDs; i++)
{
pinMode(ledPins[i], OUTPUT); // make all the LED pins
   outputs
}
}
```

Creating a bar graph using LEDs

```
void loop() {
sensorValue = analogRead(analogInPin); // read the analog
   in value
ledLevel = map(sensorValue, 0, 1023, 0, NoLEDs); // map to
   the number of LEDs
for (int i = 0; i < NoLEDs; i++)
{
if (i < ledLevel ) {
digitalWrite(ledPins[i], LED_ON); // turn on pins less
   than the level
}
else {
digitalWrite(ledPins[i], LED_OFF); // turn off pins higher
   than the level:
}
}
}
```

Measuring Temperature

```
const int inPin = 0; // analog pin
void loop()
{
int value = analogRead(inPin);
float millivolts = (value / 1024.0) *
  3300; //3.3V analog input
float celsius = millivolts / 10; //
  sensor output is 10mV per degree
  Celsius
delay(1000); // wait for one second
}
```

Reading data from Arduino

```
void setup()
{
Serial.begin(9600);
}
void serialtest()
{
int i;
for(i=0; i<10; i++)
Serial.println(i);
}
```

Connecting LCDs

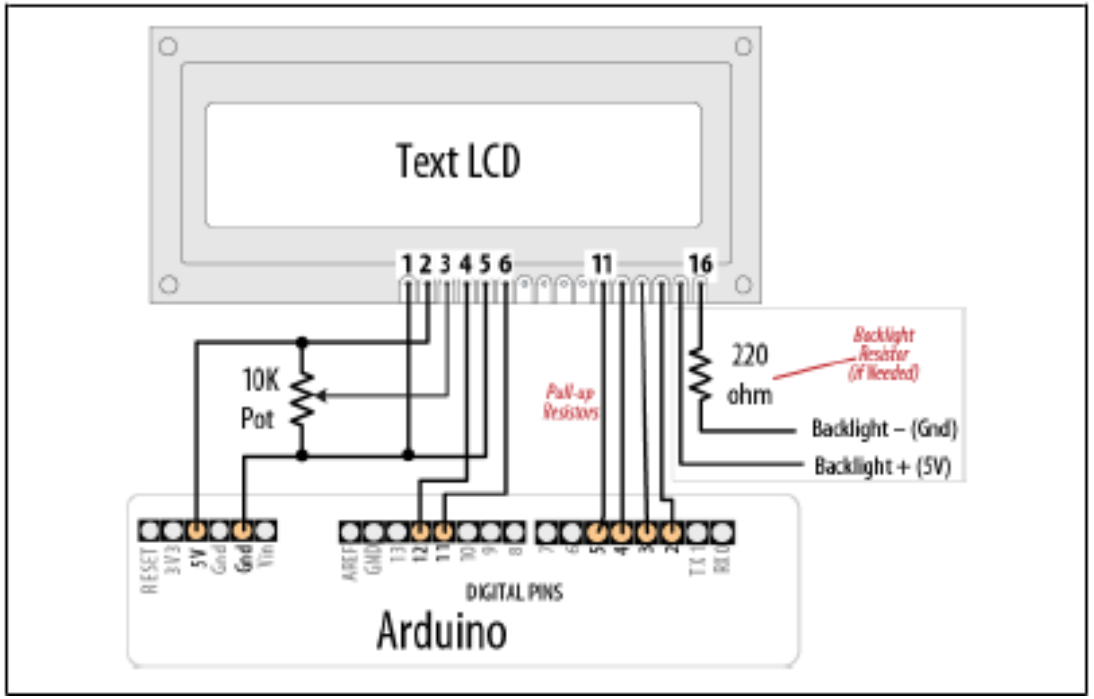


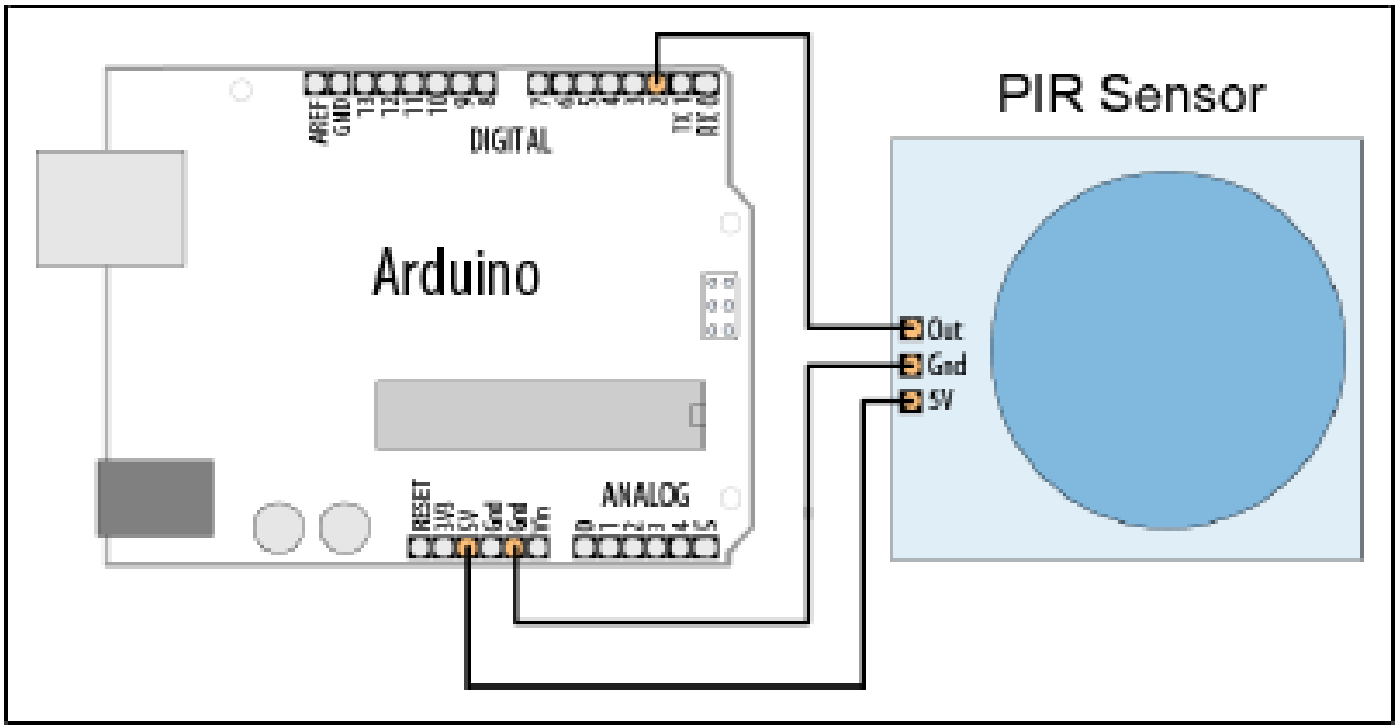
Table 11-1. LCD pin connections

LCD pin	Function	Arduino pin
1	Gnd or 0V or Vss	Gnd
2	+5V or Vdd	5V
3	Vo or contrast	
4	RS	12
5	R/W	
6	E	11
7	D0	
8	D1	
9	D2	
10	D3	
11	D4	5
12	D5	4
13	D6	3
14	D7	2
15	A or analog	
16	K or cathode	

Using LCDs

```
#include <LiquidCrystal.h> // include the library code
//constants for the number of rows and columns in the LCD
const int numRows = 2;
const int numCols = 16;
// initialize the library with the numbers of the interface
   pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup()
{
lcd.begin(numCols, numRows);
lcd.print("hello, world!"); // Print a message to the LCD.
}
```

Using PIR motion sensors



Using PIR motion sensors

```
const int ledPin = 77; // pin for the LED
const int inputPin = 2; // input pin (for the PIR sensor)
void setup() {
pinMode(ledPin, OUTPUT); // declare LED as output
pinMode(inputPin, INPUT); // declare pushbutton as input
}
void loop(){
int val = digitalRead(inputPin); // read input value
if (val == HIGH) // check if the input is HIGH
{
digitalWrite(ledPin, HIGH); // turn LED on if motion
   detected
delay(500);
digitalWrite(ledPin, LOW); // turn LED off
}
}
```

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: RoundTrip = microseconds / 29

Using ultrasonic sensors

```
const int pingPin = 5;
const int ledPin = 77; // 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 10 milliseconds delay
digitalWrite(ledPin, LOW);
delay( cm * 10);
}
```

Using ultrasonic sensors

```
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;
}
```

Audio output

```
tone(speakerPin, frequency, duration); // play the
   tone
delay(duration); //wait for the tone to finish
```

Example

- Write a program that plays an A note (440 Hz) for 1 second when a motion sensor detects motion.

Using Interrupts

- On a standard Arduino board, two pins can be used as interrupts: pins 2 and 3.
- The interrupt is enabled through the following line:
- attachInterrupt(interrupt, function, mode)
  - attachInterrupt(0, doEncoder, FALLING);

Interrupt example

```
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;
   }
```

Timer functions (timer.h library)

- int every(long period, callback)
  - Run the 'callback' every 'period' milliseconds. Returns the ID of the timer event.
- int every(long period, callback, int repeatCount)
  - Run the 'callback' every 'period' milliseconds for a total of 'repeatCount' times. Returns the ID of the timer event.
- int after(long duration, callback)
  - Run the 'callback' once after 'period' milliseconds. Returns the ID of the timer event.

Timer functions (timer.h library)

- int oscillate(int pin, long period, int startingValue)
  - Toggle the state of the digital output 'pin' every 'period' milliseconds. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.
- int oscillate(int pin, long period, int startingValue, int repeatCount)
  - Toggle the state of the digital output 'pin' every 'period' milliseconds 'repeatCount' times. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.

Timer functions (Timer.h library)

- int pulse(int pin, long period, int startingValue)
  - Toggle the state of the digital output 'pin' just once after 'period' milliseconds. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.
- int stop(int id)
  - Stop the timer event running. Returns the ID of the timer event.
- int update()
  - Must be called from 'loop'. This will service all the events associated with the timer.

Example (1/2)

```
#include "Timer.h"
Timer t;
int ledEvent;
void setup()
{
  Serial.begin(9600);
  int tickEvent = t.every(2000, doSomething);
  Serial.print("2 second tick started id=");
  Serial.println(tickEvent);
  pinMode(13, OUTPUT);
  ledEvent = t.oscillate(13, 50, HIGH);
  Serial.print("LED event started id=");
  Serial.println(ledEvent);
  int afterEvent = t.after(10000, doAfter);
  Serial.print("After event started id=");
  Serial.println(afterEvent);
}
```

Example (2/2)

```
void loop()
{
  t.update();
}

void doSomething()
{
  Serial.print("2 second tick: millis()=");
  Serial.println(millis());
}

void doAfter()
{
  Serial.println("stop the led event");
  t.stop(ledEvent);
  t.oscillate(13, 500, HIGH, 5);
}
```

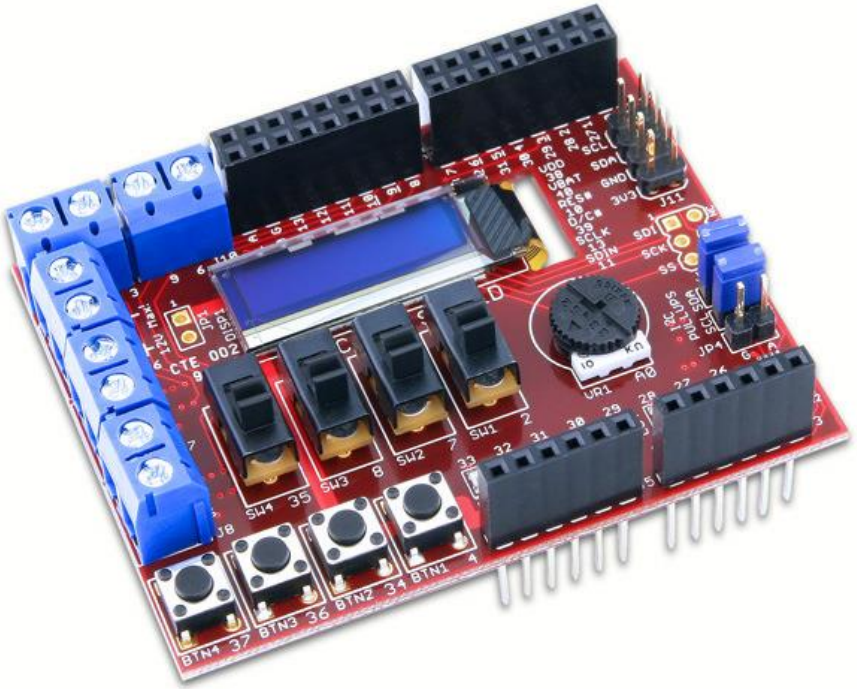
CHIPKIT MAX32 ARDUINO BOARD

ChipKit MAX32

- Microcontroller: PIC32MX795F512L
- Flash Memory: 512K
- RAM Memory: 128K
- Operating Voltage: 3.3V
- Operating Frequency: 80Mhz
- Typical operating current: 90mA
- Input Voltage (recommended): 7V to 15V
- Input Voltage (maximum): 20V
- I/O Pins: 83 total
- Analog Inputs: 16
- Analog input voltage range: 0V to 3.3V
- DC Current per pin: +/-18mA
- Advanced peripherals:
- 10/100 Ethernet MAC
- USB 2.0 Full Speed OTG controller
- 2 CAN controllers.
- **External Interrupts**: Pin 3 (INT0), Pin 2 (INT1), Pin 7 (INT2), Pin 21 (INT3), Pin 20 (INT4)

Basic I/O Shield

- Features
  - 128x32 pixel OLED graphic display
  - I2C temperature sensor
  - 256Kbit I2C EEPROM
  - I2C daisy chain connector
  - 4 push buttons
  - 4 slide switches
  - 8 discrete LEDs
  - 4 open drain FET drivers
  - Analog potentiometer



Temperature Sensor

- A digital temperature sensor is provided using a Microchip TCN75A 2-Wire Serial Temperature Sensor. The temperature sensor, IC2, is an I2C device, and is located in the lower right corner of the board.
- The TCN75A is rated for an accuracy of +/-1ºC and has selectable resolution from 0.5ºC down to 0.0625ºC. The seven bit device address is ‘1001000’.
- Digilent provides a library for accessing the temperature sensor. This library is available on the Digilent web site and in the third party library repository on github.
- Using the temperature sensor with the MAX32 board requires manually connecting the SDA and SCL pins to the basic I/O shield

Configuring Temperature sensor

- **void config(uint8_t configuration)**
- Parameters
- configuration - Value to be written to config register
- This function writes the configuration register with the given value. There are a number of defined
- values as described below that can be or'd together to set multiple parameters. For example if one
- wished to put the device in one shot mode and use 12 bit resolution the following call could be made.
- Config(ONESHOT | RES12)
- IOSHIELDTEMP_ONESHOT 0x80 //One Shot mode
- IOSHIELDTEMP_RES9 0x00 //9-bit resolution (0.5ºC)
- IOSHIELDTEMP_RES10 0x20 //10-bit resolution
- IOSHIELDTEMP_RES11 0x40 //11-bit resolution
- IOSHIELDTEMP_RES12 0x60 //12-bit resolution (0.0625ºC)
- IOSHIELDTEMP_FAULT1 0x00 //1 fault queue bits
- IOSHIELDTEMP_FAULT2 0x08 //2 fault queue bits
- IOSHIELDTEMP_FAULT4 0x10 //4 fault queue bits
- IOSHIELDTEMP_FAULT6 0x18 //6 fault queue bits
- IOSHIELDTEMP_ALERTLOW 0x00 //Alert bit active-low
- IOSHIELDTEMP_ALERTHIGH 0x04 //Alert bit active-high
- IOSHIELDTEMP_CMPMODE 0x00 ///comparator mode.
- IOSHIELDTEMP_INTMODE 0x02 //interrupt mode
- IOSHIELDTEMP_STARTUP 0x00 //Shutdown disabled
- IOSHIELDTEMP_SHUTDOWN 0x01 //Shutdown enabled
- IOSHEIDLTEMP_CONF_DEFAULT //Power up initial configuration

Reading Temperature sensor

- **float getTemp()**
- Retrieves the current temp from the temp sensor and converts the returned value into a signed floating point value.

Example

```
void setup(){
IOShieldTemp.config(IOSHIELDTEMP_ONESHOT |
   IOSHIELDTEMP_RES11 | IOSHIELDTEMP_ALERTHIGH);
}

void loop()
{
   float temp;
   int celsius;
   char sign, msd_char, lsd_char;
   //Get Temperature in Celsius.
   temp = IOShieldTemp.getTemp();
}
```

Potentiometer

- A potentiometer is provided on the board
- to be used as an analog signal source or
- analog control input. The pot is a 10Kohm
- trimmer pot connected between the VCC3V3
- supply and ground. The wiper of the pot is
- connected to analog input A0.
- The pot is read using the analogRead function.

OLED Display

- 128x32 pixels
- Each individual pixel can only be on (illuminated) or off (not illuminated)
- The display is a serial device that is accessed via an SPI interface.
- write-only device

Using the OLED display

- Initialization
- Mode select
  - Character
  - Graphic
  - Drawing

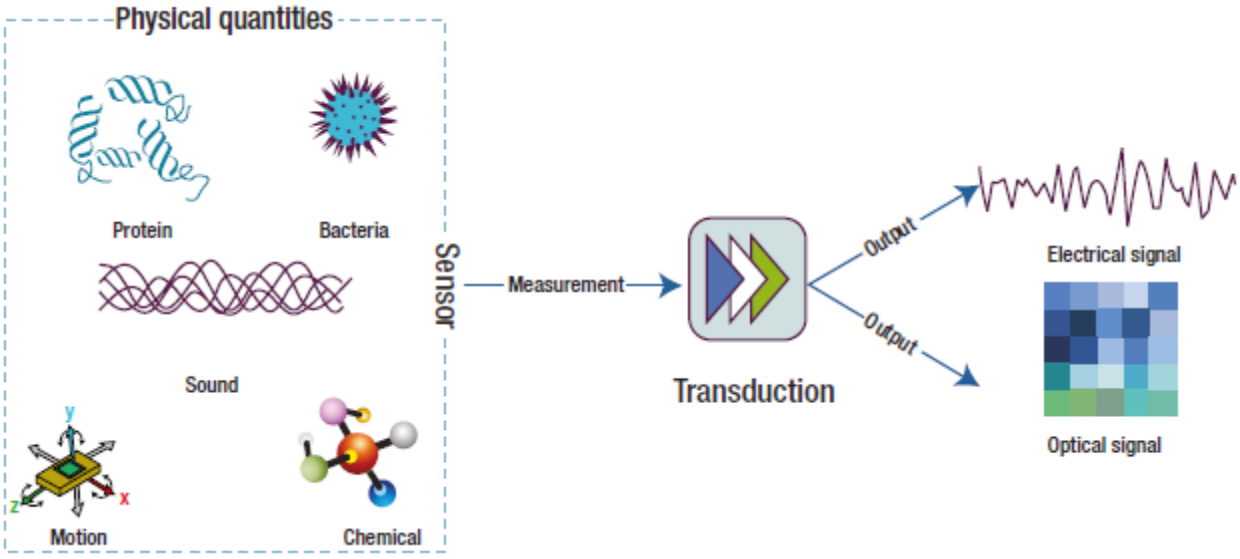
Example

```
void setup()
{
IOShieldOled.begin();
}

void loop()
{
char toprint;

  IOShieldOled.clearBuffer();
    IOShieldOled.setCursor(0, 0);

toprint = ‘A’;

IOShield0led.putChar(toprint);
  }
```

INTRODUCTION TO SENSORS

What is a sensor?

- *A device that receives a stimulus and responds with an electrical signal.*
- *A special type of transducer (device that converts one type of energy into another*



Common Sensors

- Mechanical
  - Accelerometers
  - Gyroscopes
- Optical
  - Photodetectors
  - Infrared
- Semiconductor
  - Gas
  - Temperature
  - Magnetic

Example: Simple temperature sensor

- A RTD is a *thermoresistive temperature sensor. It is a metal element (in a ceramic tube) whose resistance* typically increases with temperature, according to a known function.
- A linear approximation is given by

$$R = R_0(1 + \alpha T)$$

- Where a is the temperature coefficient, T is the temperature in Kelvin and R0 the resistance in a known temperature

Example

- Calculate the temperature for a copper RTD if R0 = 500Ω and room temperature and a = 0.0043. The measured resistance is 500.43Ω

Sensor Characteristics (1/4)

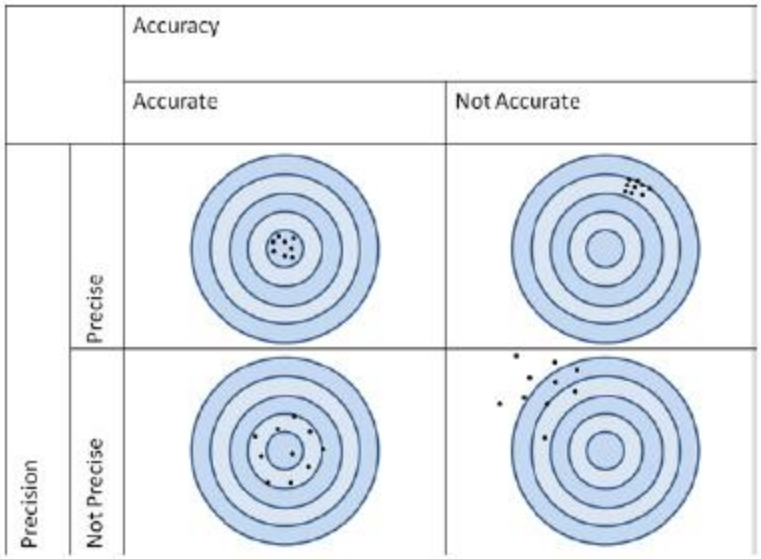
- Range
  - Full Scale Range
  - Operating Voltage Range
- Accuracy
- Transfer Function
  - S=F(x), where x is the measurand and S the electrical signal (commonly Voltage)
- Sensitivity
  - The change in input required to generate a unit change in output

Sensor Characteristics (2/4)

- Accuracy
- Precision

$$Percentage\ Relative\ Error = \frac{(Measured\ Value - True\ Value)}{(True\ Value)} \times 100$$

$$Percentage\ Standard\ Deviation = \frac{(Standard\ Deviation)}{(Mean)} \times 100$$

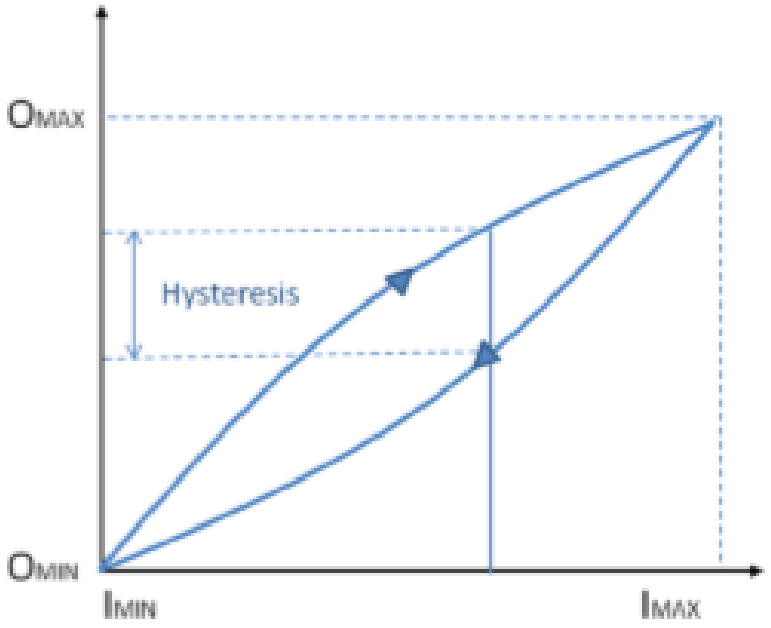


Sensor Characteristics (3/4)

- Error: the difference between the measured value and true value
- Systematic errors are reproducible inaccuracies that can be corrected with compensation methods
  - Interference errors
  - Operator errors etc.
- Random error
  - Noise

Sensor Characteristics (4/4)

- Hysterysis: the difference in output between the rising and falling output values for a given input



Example: Smoke sensor (1/2)

- An MQ-2 smoke sensor reports smoke by the voltage level it puts out.
- The more smoke there is, the higher the voltage.
- built-in potentiometer for adjusting sensitivity
- Three pins:
  - Vdd input
  - Ground input
  - Analog output

Smoke sensor (2/2)

```
const int smokePin = 54; //sensor input
void setup() {
  pinMode(smokePin, INPUT);
  Serial.begin(9600);
}
void loop() {
int smoke_level = analogRead(smokePin); //read sensor
Serial.println(smoke_level);
  if(smoke_level > 120) {     //calibrate accordingly
    Serial.println("Smoke detected");
  }
  delay(100); // ms
}
```

References

- M.J. McGrath, C. N. Scanaill and D. Nafus, “Sensor Technologies Healthcare, Wellness and Environmental Applications”, Apress, 2013
- C.W. de Silva, “Sensors and Actuators: Control System Instrumentation”, 2nd Edition, CRC Press, 2015
- T. Karvinen and K. Karvinen, ” Make: Sensors: A Hands-On Primer for Monitoring the Real World with Arduino and Raspberry Pi”, Maker Media, Inc, 2014

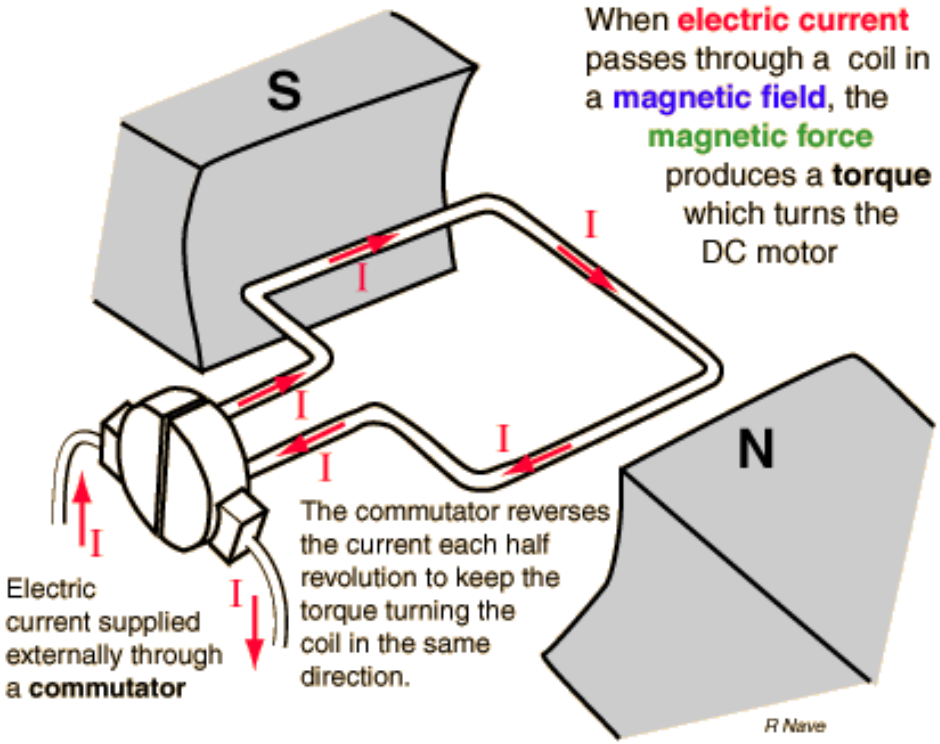
ACTUATORS

Actuators

- Device that turns energy (typically electrical) to motion
- Features
  - Force
  - Speed
  - Torque
  - Power
  - Efficiency

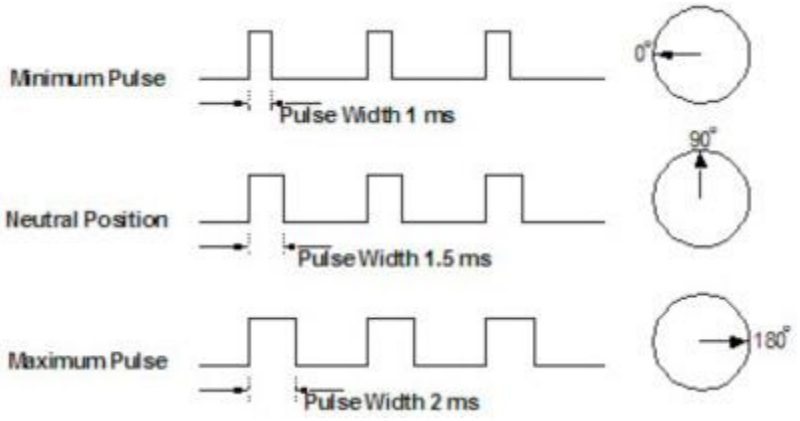
DC motor

- Force is produced (F=ILB) due to the electric current in a wire inside a magnetic field.
- Proportional to the current, therefore can be controlled by potentiometer
- Hard to control precisely



Servo motors

- A DC motor with a control circuit
- Servo motors are controlled by PWM through the control pin

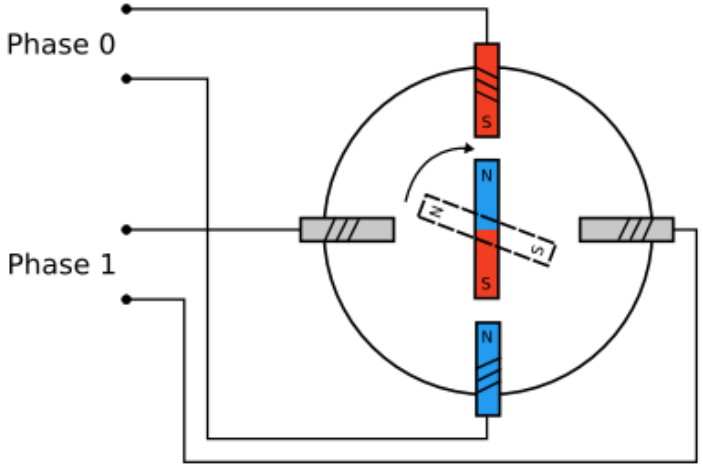


Servo motor control

```
#include <Servo.h>
  Servo myservo;  // create servo object to control a servo
  void setup() {
  int val = 180;    // variable to control servo
    myservo.attach(9);  // pin 9 is a PWM pin
  }
  void loop() {
    myservo.write(val);                  // constant servo speed
    delay(15);                  // waits for the servo to get there
  }
```

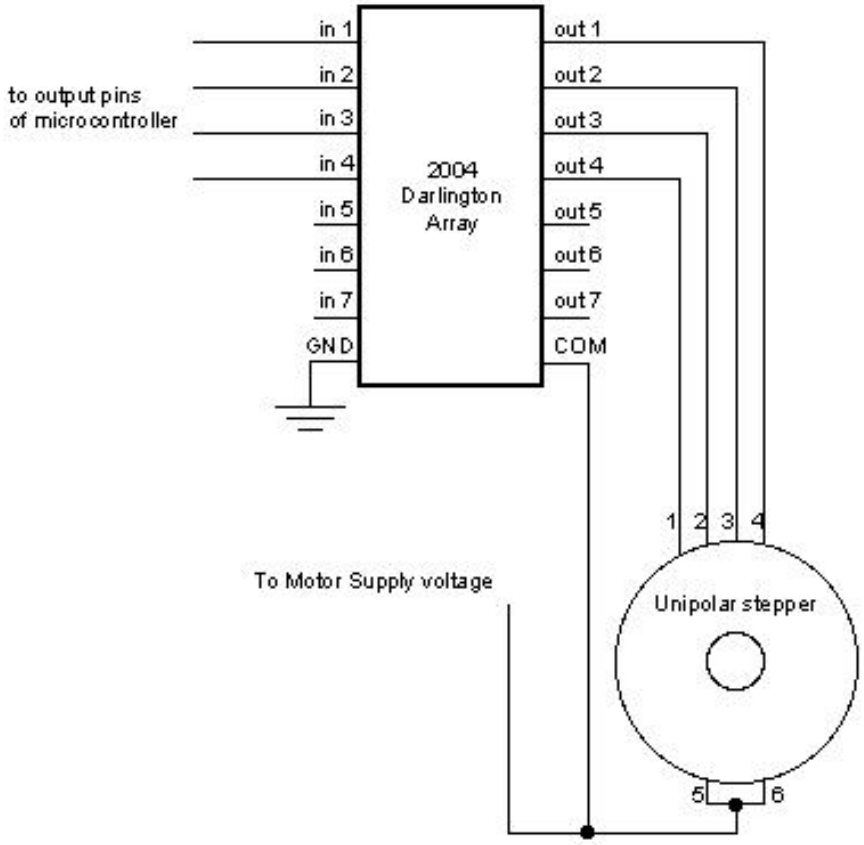
Stepper Motors

- motor controlled by a series of electromagnetic coils.
- The center shaft has a series of magnets mounted on it
- the coils are alternately given current or not, creating magnetic fields which repulse or attract the magnets on the shaft, causing the motor to rotate.
- allows for very precise control of the motor. it can be turned in very accurate steps of set degree increments
- two basic type
  - unipolar
  - bipolar



Example: Unipolar stepper motor

- Four digital pins required to control the motor
- Darlington transistor array used for supplying the power required



Stepper motor control

```
#include <Stepper.h> //the control sequence is in this library

    const int stepsPerRevolution = 200;  // motor-dependent

    Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11); //pins used

    void setup() {
      // set the speed at 60 rpm:
      myStepper.setSpeed(60);  //actually sets the delay between steps
    }

    void loop() {
      // step one revolution  in one direction:
      myStepper.step(stepsPerRevolution);
      delay(500);
}
```

INTERNET OF THINGS WITH ARDUINO

Arduino with GPS (1/4)

- Three modes:
  - Stand-alone: GPS obtains information such as position and altitude with only the signal of the satellites
    - slowest of the three modes
    - AT+CGPSINFO command brings directly latitude, logtitude, date, UTC time, altitude and speed
  - S-GPS: module connects to a GPS server
    - module calculates the position
  - A-GPS: Three modes
    - MS-Assisted: Server sends data and calculates position
    - MS-based: Server sends aiding data, module calculates position
    - Stand-Alone: Module demodulates GPS data and calculates position

Arduino with GPS (2/4)

- AT+CGPSINFO returns the GPS info in a string:
- [<latitude>],[<N/S>],[<longitude>],[<E/W>],[<date>],[<UTC_time>],[<altitude>],[<speedOG>],[<course>]

Arduino with GPS (3/4)

```
int8_t answer;
int onModulePin= 2;
char gps_data[100];   //will perform 100 GPS data reads
int counter;
void setup(){
     pinMode(onModulePin, OUTPUT);
     Serial.begin(115200);
     Serial.println("Starting...");
     power_on();
    delay(5000); // starts GPS session in stand alone mode
   }
}
void power_on(){   //void power_on() should be placed AFTER void loop(), used here for lecture
     digitalWrite(onModulePin,HIGH);    //GPS module requires a 3 sec pulse on onModulePin
     delay(3000);
     digitalWrite(onModulePin,LOW);
}
```

Arduino with GPS (4/4)

```
void loop(){
answer = sendATcommand("AT+CGPSINFO","+CGPSINFO:",1000);
// request info from GPS
if (answer == 1) {
        counter = 0;
        do{
         while(Serial.available() == 0);          //reading GPS data
         gps_data[counter] = Serial.read();
         counter++;
         }
       }
          Serial.print("GPS data:");              //printing GPS data
         Serial.print(gps_data);
}
```

Connecting through 3G (1/2)

```
int8_t answer;
int onModulePin = 2, aux;
char aux_str[50];
void setup(){
pinMode(onModulePin, OUTPUT);
Serial.begin(115200);
Serial.println("Starting...");
power_on();
delay(3000); //sets the PIN code
sprintf(aux_str, "AT+CPIN=%s", pin_number);
sendATcommand(aux_str, "OK", 2000);
delay(3000);
```

Connecting through 3G (2/2)

```
while( (sendATcommand("AT+CREG?", "+CREG: 0,1",
    500) || sendATcommand("AT+CREG?", "+CREG: 0,5",
    500)) == 0 ); // sets APN, user name and password
sprintf(aux_str, "AT+CGSOCKCONT=1,\"IP\",\"%s\"", apn);
sendATcommand(aux_str, "OK", 2000);
sprintf(aux_str, "AT+CSOCKAUTH=1,1,\"%s\",\"%s\"",
    user_name, password);
sendATcommand(aux_str, "OK", 2000);
}
```