Introduction to Arduino

What is Arduino?

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can be communicating with software running on your computer (e.g. Flash, Processing and MaxMSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.

The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

Why Arduino?

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50

- Cross-platform - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

- Simple, clear programming environment - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino

- Open source and extensible software- The Arduino software and is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

- Open source and extensible hardware - The Arduino is based on Atmel's ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the

module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

Introduction

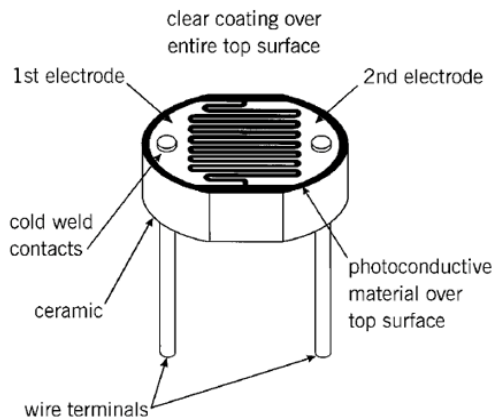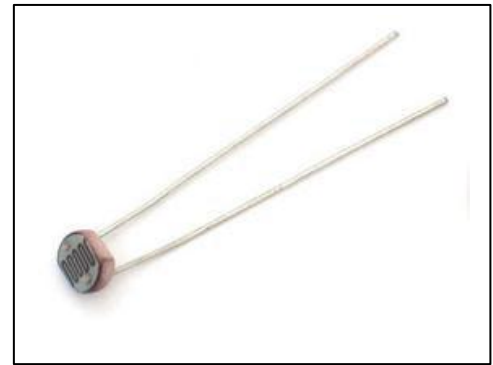## Photocells

Photoresistors, LDR (light dependent resistor)..

Figure 3
Typical Construction of a Plastic Coated Photocell

What is a photocell?
Photocells are sensors that allow you to detect light. They are small, inexpensive, low-power, easy to use and don't wear out. For that reason they often appear in toys, gadgets and appliances. They are often referred to as CdS cells (they are made of Cadmium-Sulfide), light-dependent resistors (LDR), and photoresistors.
Photocells are basically a resistor that changes its resistive value (in ohms Ω) depending on how much light is shining onto the squiggly face.

How to measure light using a photocell
As we've said, a photocell's resistance changes as the face is exposed to more light. When its dark, the sensor looks like an large resistor up to 10MΩ, as the light level increases, the resistance goes down. This graph indicates approximately the resistance of the sensor at different light levels. Remember each photocell will be a little different so use this as a guide only!

The easiest way to measure a resistive sensor is to connect one end to Power and the other to a **pull-down** resistor to ground. Then the point between the fixed pulldown resistor and the variable photocell resistor is connected to the analog input of a microcontroller such as an Arduino (shown)

Resistance vs. Illumination
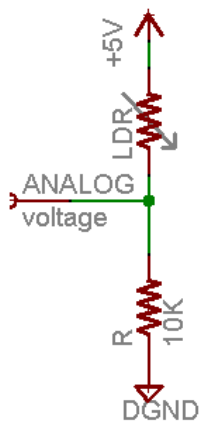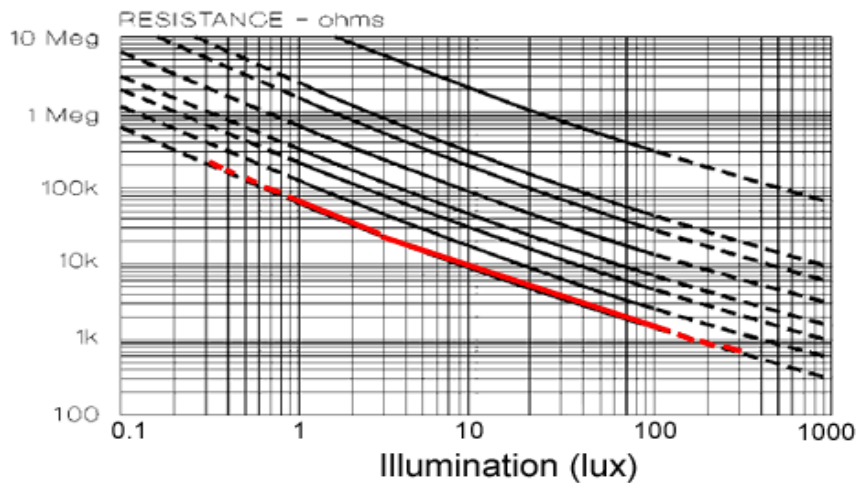
RESISTANCE – ohms

Vo = Vcc (R / (R + Photocell))

That is, the voltage is proportional to the **inverse** of the photocell resistance which is, in turn, inversely proportional to light levels

Interrupts

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a small program called an interrupt handler (or interrupt service routine, ISR) to deal with the event. This interruption is temporary, and after the interrupt handler finishes, the processor resumes execution of the previous thread. There are two types of interrupts: hardware interrupts and software interrupts.

Hardware interrupts

A hardware interrupt is an electronic alerting signal sent to the processor from an external device. Arduino uno board has two external interrupts: int.0 (on digital pin 2) and

int.1 (on digital pin 3). when the triggering event is captured at these pins, the corresponding interrupt service routine is executed.

The attachInterrupt() API is used to specify a function to call when an external interrupt occurs.

Syntax:

attachInterrupt(interrupt, function, mode)

Parameters:

interrupt: the number of the interrupt (int)

function: the function to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered. Four contstants are predefined as valid values:

- LOW to trigger the interrupt whenever the pin is low,
- CHANGE to trigger the interrupt whenever the pin changes value
- RISING to trigger when the pin goes from low to high,
- FALLING for when the pin goes from high to low.

NOTE:

You should declare as volatile any variables that you modify within the attached function.
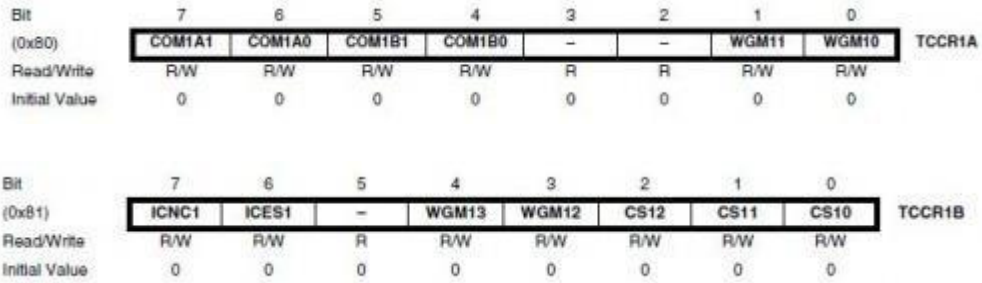
Software interrupts

A software interrupt is caused either by an exceptional condition in the processor itself, or a special instruction in the instruction set which causes an interrupt when it is executed. In this experiment we will focus on the interrupt caused by timer1 overflow.

Timer1 is 16 bit hardware timer on the ATmega168/328. There are 3 hardware timers available on the chip, and they can be configured in a variety of ways to achieve different functionality. The timer can be programmed by some special registers. You can configure the prescaler for the timer, or the mode of operation and many other things. Timer1's clock speed is defined by setting the prescaler, or divisor. This prescale can be set to 1, 8, 64, 256 or 1024.

Timer Register
You can change the Timer behaviour through the timer register. The most important timer registers are:

TCCRx - Timer/Counter Control Register. The prescaler can be configured here.

Bit (0x80)	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit (0x81)	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNTx - Timer/Counter Register. The actual timer value is stored here.

OCRx - Output Compare Register

ICRx - Input Capture Register (only for 16bit timer)

TIMSKx - Timer/Counter Interrupt Mask Register. To enable/disable timer interrupts.

TIFRx - Timer/Counter Interrupt Flag Register. Indicates a pending timer interrupt.

Clock select and timer frequency

Different clock sources can be selected for each timer independently. To calculate the timer frequency (for example 2Hz using timer1) you will need:

1. CPU frequency 16Mhz for Arduino
2. maximum timer counter value (256 for 8bit, 65536 for 16bit timer)
3. Divide CPU frequency through the choosen prescaler (16000000 / 256 = 62500)
4. Divide result through the desired frequency (62500 / 2Hz = 31250)
5. Verify the result against the maximum timer counter value (31250 < 65536 success) if fail, choose bigger prescaler.

| CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Table.1 Clock selection

To start with arduino programming let us first try to run an already existing example

1.Let there be blink!

- You have to open up the workspace
- You have to choose the correct serial port from Tools ⟶ Serial Port in the menu.
- Open blink sketch
- Compile blink sketch by clicking on
- Upload blink by clicking on

2. **Photocells**
Simple demonstration of use
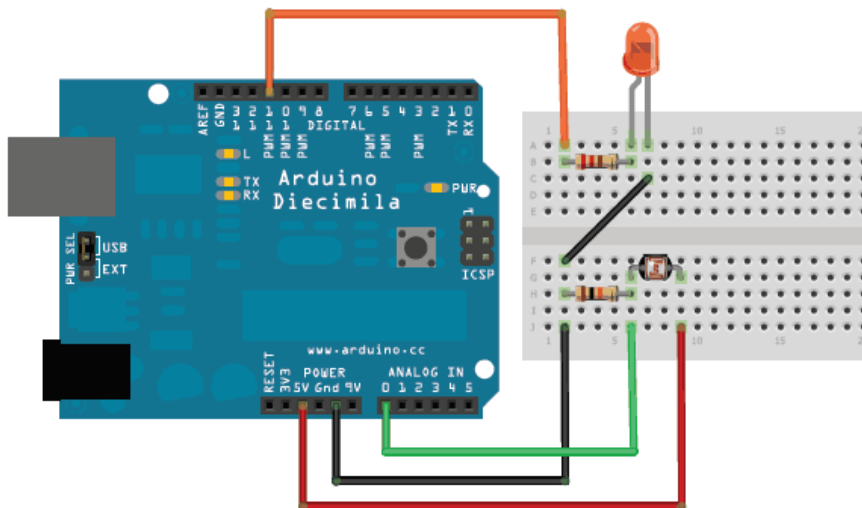
This sketch will take the analog voltage reading and use that to determine how bright the red LED is. The darker it is, the brighter the LED will be! Remember that the LED has to be connected to a PWM pin for this to work, I use pin 11 in this example.

```
/* Photocell simple testing sketch.



Connect one end of the photocell to 5V, the other end to Analog
0.
Then connect one end of a 10K resistor from Analog 0 to ground
Connect LED from pin 11 through a resistor to ground
For more information see www.ladyada.net/learn/sensors/cds.html
*/
```

```
int photocellPin = 0;        // the cell and 10K pulldown are
connected to a0
int photocellReading;        // the analog reading from the sensor
divider
int LEDpin = 11;             // connect Red LED to pin 11 (PWM pin)
int LEDbrightness;           //
void setup(void) {
  // We'll send debugging information via the Serial monitor
  Serial.begin(9600);
}

void loop(void) {
  photocellReading = analogRead(photocellPin);

  Serial.print("Analog reading = ");
  Serial.print(photocellReading);     // the raw analog reading

  if (photocellReading < …??) {
    Serial.println(" - Dark");
  }
  else if (photocellReading < …??) {
    Serial.println(" - Light");
  } else  {
    Serial.println(" - bright");
  }

  // LED gets brighter the darker it is at the sensor
  // that means we have to -invert- the reading from 0-1023 back
to 1023-0
  photocellReading = 1023 - photocellReading;
  //now we have to map 0-1023 to 0-255 since that's the range
analogWrite uses
  LEDbrightness = map(photocellReading, 0, 1023, 0, 255);
  analogWrite(LEDpin, LEDbrightness);



  delay(10000);
}
```
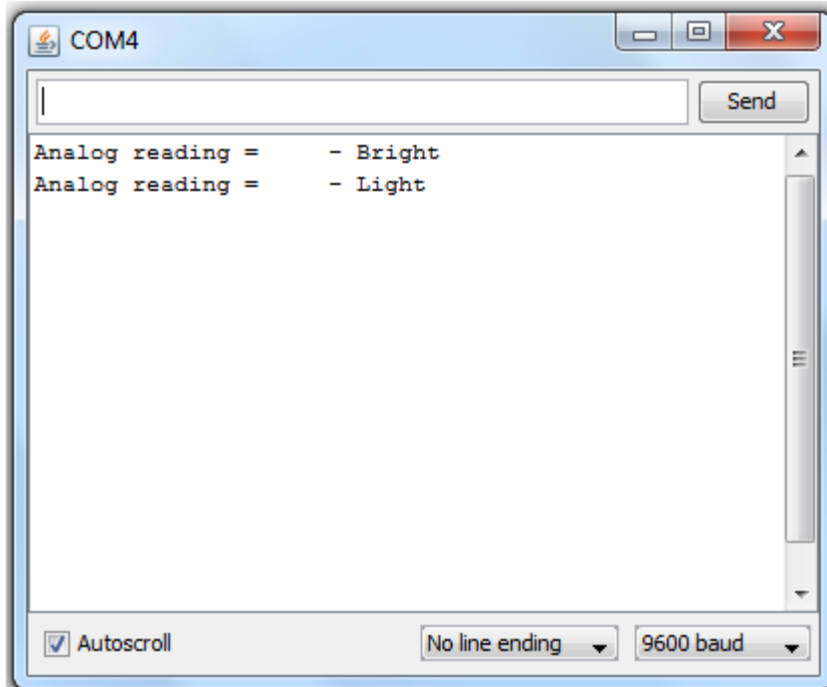
Serial Monitor:

It shows you the result which you wrote in Serial.println().
From **Tools -> Serial Monitor**.

For more information about interfacing Hardware with Arduino

http://arduino.cc/playground/Main/InterfacingWithHardware

Hardware interrupts

1- Connect a push button to digital pin 2 in the arduino uno board.
2- Copy the following code to the arduino IDE then compile and upload it to the board

```
// Hardware interrupt example
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

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

void blink()
{
  state = !state;
}
```

3- Explain how the previous code works.

Software interrupts

1. Copy the following code to the arduino IDE then compile and upload it to the board

```
// Timer1 overflow interrupt example
#define ledPin 13
void setup()
{
  pinMode(ledPin, OUTPUT);

  // initialize timer1
  noInterrupts();         // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;

  TCNT1 = 34286;          // preload timer 65536-16MHz/256/2Hz
  TCCR1B |= (1 << CS12);    // 256 prescaler
  TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt
  interrupts();           // enable all interrupts
}

ISR(TIMER1_OVF_vect)       // interrupt service routine that wraps a user
defined function supplied by attachInterrupt
{
  TCNT1 = 34286;          // preload timer
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}

void loop()
{
  // your program here...
}
```

2. Explain how the previous code works.

To Do (PWM)

Write a code to generate a square wave on the pin 13 with frequency 1 KHz and duty cycle that is controllable by a potentiometer.