

## Photoresistor, Transistor, and LED's

### Purpose:

- To introduce photoresistors, LED's, FET's, and transistors used as power switching devices
- To become familiar with the capability of the Atmega 128 to measure the change of resistance of a sensor using analog-to-digital conversion.
- To build and experiment with a light-controlled switch

### Components:

<u>Qty.</u>	<u>Item</u>	<u>Qty.</u>	<u>Item</u>
1	10 k $\Omega$ resistor	1	22 k $\Omega$ resistor
1	Solderless Breadboard	1	470 $\Omega$ resistor
1	Photoresistor	1	220 $\Omega$ resistor
1	2N3904, NPN transistor	1	red or green LED
1	Atmel Atmega 128, STK500 & STK 501 interface boards, and programming cable.		

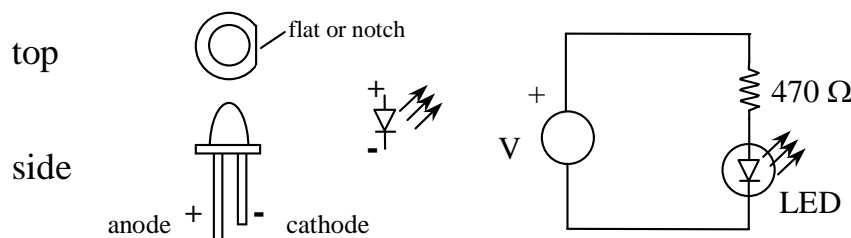
### Introduction:

A photoresistor is simply a resistor whose resistance depends on the amount of light incident upon it. They are used to make light-sensitive switching devices. Photoresistors are often made from cadmium sulfide (CdS). The resistance of a CdS photoresistor varies inversely to the amount of light incident upon it. In other words, its resistance will be high in the dark and low in the light.

The LED behaves like an ordinary diode except that when it is forward biased, it emits light. The LED's forward voltage drop is higher than an ordinary diode. Typical LED's require 5 to 15mA to reach full brightness, but are not designed to handle more than about 20 mA of current (though some can handle upwards of 80 mA or more). You will therefore *always* need to provide a resistor in series with an LED to limit the current to about 20 mA or less, or else you will burn it out. Also, don't make the mistake of trying to substitute an LED where a standard diode is called for! Look at the schematic diagram to see which kind of component is needed.

### Procedure

1. Measure the photo-resistor's resistance in the ambient lighting of the lab. Once this is recorded, repeat the measurement, only this time, cover the cell with your hand. These two extremes will be used in calculations later on.
2. To verify the behavior of the LED, construct the circuit shown in Figure 1, and vary the supply voltage between 1 to 5 volts at 1-volt increments. **At each voltage, measure the voltage across the LED and the 470  $\Omega$  resistor and enter the values into the following table. The LED current can be calculated by applying Ohm's law across the resistor.** A similar table should be entered into the lab report with all voltage values and comments. Remember  $I=R/V$  for current

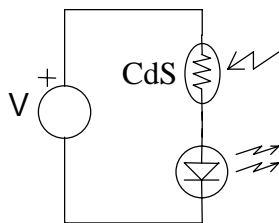


**Figure 1.** LED and typical circuit. Note that the anode lead is longer than the cathode. Sometimes there may be a flat on the cathode side of the LED to help you distinguish anode from cathode.

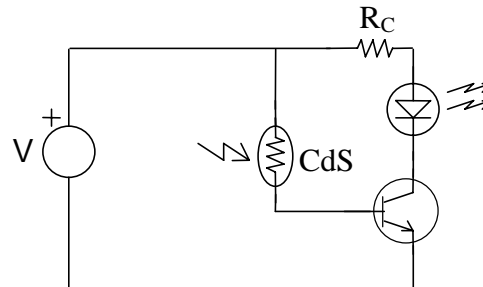
**Table 1.** LED circuit measurements (Refer to Figure 1)

$V_{\text{supply}}, \text{V}$	$V_{\text{LED}}, \text{V}$	$V_R, \text{V}$	Current, mA	Comment on LED brightness
1				
2				
3				
4				
5				

Figure 2 shows a simple ‘light-controlled-LED’. The circuit should turn-off the LED as the photo resistor is covered. **Explain the theory of operation of this circuit.** Based on the information obtained above, what is a good supply voltage to use? (Hint:  $V$  should be high enough so that enough current flows through the LED when the photo resistor has low resistance, and yet should be low enough so that the current is not enough to turn on the LED when the photo resistor has high resistance.) **Build the circuit and check its function.**



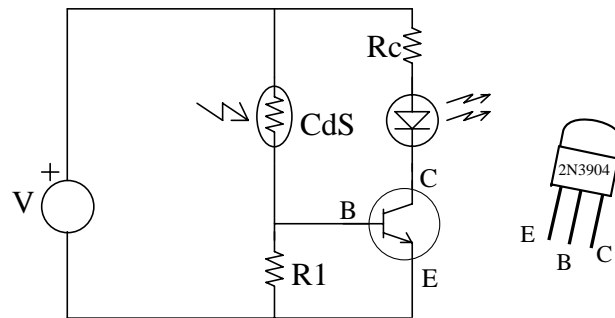
**Figure 2.** Light-controlled LED



**Figure 3.** Light-controlled using a transistor “switch”

### The Light-Controlled Switch Using a Transistor

A transistor can be added to the light-controlled-switch circuit to improve its sensitivity and to eliminate the ‘half-on-half-off’ state of the LED. A rudimentary circuit to do so is shown above in Figure 3 (**you don’t have to build this circuit**). Here the photoresistor controls the transistor’s base current, which is then amplified by the transistor. The collector current of the transistor, in turn, controls the LED. Unfortunately, this circuit may not function properly, because when the photoresistor is in the dark state, (and the LED is supposed to be turned off), the base current may be large enough that the LED may stay lit! **Prove this, by calculating the collector current for the circuit in Figure 3 when  $V=10 \text{ V}$ ,  $R_{\text{CdS}}=100 \text{ k}\Omega$ ,  $R_c=220 \text{ }\Omega$  and  $h_{\text{fe}}=100$ .** Figure 4 shows an improved circuit. This is the circuit that you will build and experiment with next.



**Figure 4.** Improved Light-Controlled Switch Using a Transistor

With a properly selected resistor  $R_1$ , the voltage at the base of the transistor in the dark state is less than 0.7 V, and therefore the transistor is in the cut-off state. As the photoresistor's resistance decreases (as the result of an increase in light intensity), the base voltage increases. Once the base voltage reaches 0.7 V, the base current starts to flow, and any further decrease in the photoresistor's resistance causes an increase of base current. This base current increment will be amplified by the current gain of the transistor.

### Procedure

The following procedure is the instructions to build the circuit in figure 4. Remember that you cannot just select any resistor, try some of the values that you have found in the previous labs since we do not have all the different resistor values available. **This procedure for a light transistor switch should be clear in your report with calculations.**

1. **Choose the supply voltage.** The supply voltage is often a predetermined value rather than a design choice. For example, if a battery is to be used, the voltage will likely be 3, 6, 9, or 12 volts. In the following calculation and in later circuit construction, use any voltage of your choice between **6 V to 15 V**, this will be called **V**.
2. **Select  $R_1$ .** First, measure the value of the photo resistor's resistance (call it  $R_{on}$ ) at which you would like the LED to be turned on. The resistance value can be that for when the photoresistor is covered or uncovered, it's up to you. The value of  $R_1$  should be such that  **$0.7 = \frac{V \cdot R_1}{R_1 + R_{on}}$** .

A variable resistor (trim pot) can be used so that the turn-on value can be adjusted.

3. **Select  $R_c$ ,** the current limiting resistor. With this resistor, the collector current is limited to  **$I_{max} = \frac{V - V_{LED} - 0.4}{R_c}$** , where  $V_{LED}$  is the voltage drop across the LED, and 0.4 V is a typical saturation voltage between the collector and emitter. **Select  $R_c$  so that the LED current is limited to be less than 20 mA (preferably 5-10 mA).** Using  $R_c$ , **construct and test the circuit.**

### Using the Atmega 128 to Make a Programmable Light-Controlled Switch

The circuit in Figure 4 is very simple, but it suffers from the disadvantage that once  $R_1$  is chosen and the circuit is constructed, you're stuck with its performance unless you physically remove  $R_1$  and replace it with a different value. That is not too serious if we are dealing with one circuit on a breadboard, but suppose this circuit were part of a product that you were manufacturing, say 1000 per day. If you wanted to change the performance of the device, you would have to modify the assembly drawings, circuit board artwork, component inventory, rework the entire work-in-process, etc. That would be a big deal! Here we will use the Atmega 128 to make a light-controlled switch whose performance can be modified by simply reprogramming the microcontroller.

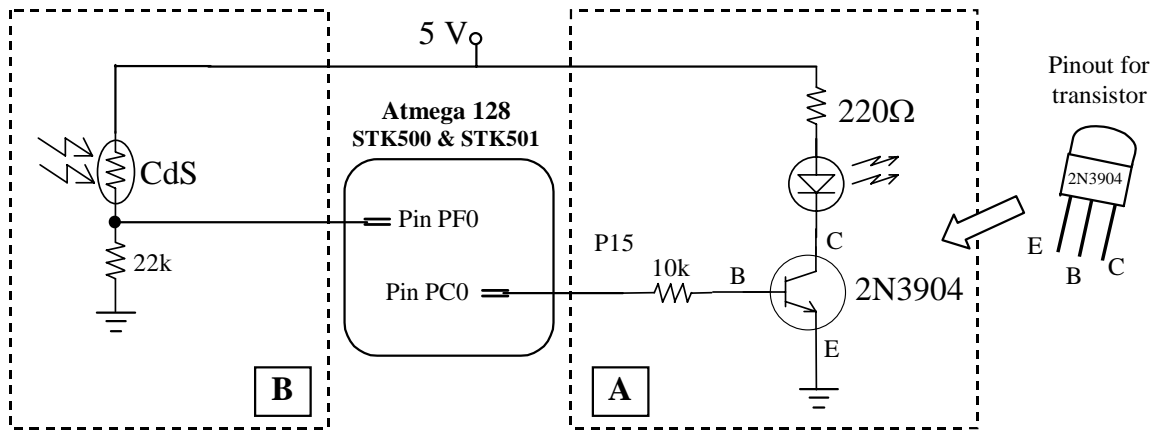
**Procedure**

1. Build the part of the circuit in shown in Figure 5 below labeled **A**. Connect the 10k resistor to pin PC0 on the STK500 board using a jumper from the pin to the solderless breadboard. Don't forget that you need to supply power to the Atmega 128. You may use the +5V from any VTG pin on the STK500 or STK501 boards for your circuit; however, be sure to double check the voltage on the pin you wish to use with a voltmeter **before** connecting it to your circuit. Likewise, you may use a pin labeled GND for ground. The voltage on PC0 will either be 0 or 5 V. **Which of these will turn the transistor on? When the transistor turns on, is it saturated? Based on your earlier measurements on the LED and the characteristics of the transistor, what are  $i_b$ , and  $i_c$  when PC0 is at 5 V?**
2. Using your last two labs as a guide compile (build) and run the **Blink Test Program** (Appendix A), which can be found on the 'Lab 106' network drive >> Lab 4 >> Blink Test Files >> blink\_test.c. For your convenience we have also included the global.h file you used last week in the same folder.

Read through the code in blink\_test.c, and **Explain how the PORTC output is toggled by using the '^=' operator.**

**Tips:**

- *When creating a new project save it to the desktop and move it to a USB flash drive once you complete the lab. Your code may not compile if you save it directly to the USB drive.*
- How did you setup (configure) your last two programs? Why? How does that apply here? Have you included all the libraries (#includes) required for your program to compile?
- Remember to build your program! This step creates your hex file. If you're trying to program your chip but can't find a hex file double check to ensure you didn't skip this step.
- Both programming the chip and outputting data to the terminal happen on the same serial port. You can't do both at the same time. This means that if your programming dialog window is still open you won't be able to establish a terminal emulation session. Nor can you reprogram the chip unless you've disconnected your terminal (In HyperTerminal this can be accomplished by clicking on the hang up icon).
- You need a jumper cable between PE0 & PE1 to the receive and transmit pins for the serial "spare" port you're using. If you don't remember which one goes where consult lab two or the pinout for the Atmega128. If you're not getting any output on the screen double check that you've made this crucial connection.



**Figure 5.** Light-controlled switch using the Atmega 128. The photoresistor (in **B**) is part of a voltage divider, the output of which is connected to pin PF0, one of the pins that can be used for analog-to-digital conversion. Pin PC0 is connected to the base of a transistor (in **A**), which is used as a switch for illuminating the LED when the light level on the photoresistor is beyond a certain threshold value.

- When you have successfully completed step 2, build the circuit in **B** shown in Figure 5, and compile and run the **Photoresistor and A/D test program** in **Appendix A**.

Connect the microcontroller to Hyperterminal as described in the intro to Atmega 128 lab. Reset the microcontroller by pressing the reset button on the STK500 or by cycling the power. **What happens when you cover the photoresistor? What range of values are shown in the communications window? What voltage do these correspond to?** Now try using the 10-bit reading instead of the 8-bit one. **Remember to change the declaration of 'level' to an unsigned 16-bit integer so it can handle the extra bits. State the differences you observe between the two methods.**

- When you have successfully completed step 3, write a program that will turn the LED on when you cover the photoresistor with your hand. (Hint: add a test in the While loop that compares the value of 'level' with a value slightly higher than that value for when the photoresistor is covered. Experiment with the threshold value.) Display the state of your LED on the terminal window.

**What changes need to be made to the software (note: no need to change any hardware) if you want to have the LED stay on under ambient light conditions and turn off when a shadow falls on the photoresistor (i.e. the opposite function to what you programmed in step 4.**

## Appendix A

### Blink test program:

```
// Blink test program
// Put your name here
// Put the date here
// This program turns on an off the PC0 output with a 500ms delay
//----- Include Files -----
#include <avr/io.h> // include I/O definitions (port names, pin names, etc)
#include <avr/interrupt.h> // include interrupt support
#include "global.h" // include our global settings
#include "timer.h" // include timer function library (timing, PWM, etc)
//----- Function Declarations-----
void init(void);
//----- Begin Code -----
int main(void)
```

```

{
    init(); // initialize everything
    while(1)
    {
        PORTC ^= 0x01; // toggle output bit PC0
        timerPause(500); // pause for 500ms
    }
    return 0;
} // end main()
void init(void)
{
    // initialize our libraries
    timerInit(); // initialize the timer system
    // Setup Digital I/O
    DDRC = 0xff; // set PORTC to all outputs
    PORTC = 0xff; // set all output pins to high
} // end init()

```

### **Photoresistor and A/D test program:**

```

// Photoresistor and A/D test program
// Put your name here
// Put the date here
// This program reads the voltage associated with a voltage divider
// that contains a photoresistor and continuously prints the output
// to the Atmega 128 Serial Port
//----- Include Files -----
#include <avr/io.h> // include I/O definitions (port names, pin names, etc)
#include <avr/interrupt.h> // include interrupt support
#include "global.h" // include our global settings
#include "uart.h" // include uart function library
#include "rprintf.h" // include printf function library
#include "timer.h" // include timer function library (timing, PWM, etc)
#include "a2d.h" // include A/D converter function library
#include "vt100.h" // include VT100 terminal support
//----- Function Declarations -----
void init(void);
//----- Begin Code -----
int main(void)
{
    u08 level; // declare unsigned 8-bit integer
    init(); // initialize everything
    // print a little intro message so we know things are working
    vt100ClearScreen();
    vt100SetCursorPos(1,1);
    rprintf("Welcome to the Photoresistor Laboratory!\r\n");

    while(1)
    {
        // sample a2d channel and print them to the terminal
        vt100SetCursorPos(2,1);
    }
}

```

```
        level = a2dConvert8bit(0);
// use a2dConvert10bit(channel#) to get a 10bit a2d reading
    rprintf("Light Level: %d \r\n", level);
    }
    return 0;
} // end main()
void init(void)
{
    // initialize our libraries
    uartInit();                // initialize the UART (serial port)
    uartSetBaudRate(9600);     // set the baud rate of the UART
    rprintfInit(uartSendByte); // make all rprintf statements use uart for output
    timerInit();              // initialize the timer system
// Setup Digital I/O
    DDRC = 0xff;              // set PORTC to all outputs
    PORTC = 0xff;            // turn off all outputs
    DDRF = 0x00;              // configure a2d port (PORTF) as input
    PORTF = 0x00;            // turn off pull-up resistors on PORTF
a2dInit();                    // turn on and initialize A/D converter
    // set the a2d prescaler (clock division ratio)
    // - a lower prescale setting will make the a2d converter go faster
    // - a higher setting will make it go slower but the measurements
    // will be more accurate
    // - other allowed prescale values can be found in a2d.h
    a2dSetPrescaler(ADC_PRESCALE_DIV32);
    // set the a2d reference
    // - the reference is the voltage against which a2d measurements are made
    // - other allowed reference values can be found in a2d.h
    a2dSetReference(ADC_REFERENCE_AVCC);
} // end init()
```