A simple implementation of LCD brightness control using the

Overview

Ambient-light sensor (ALS) ICs are increasingly used in a variety of display and lighting applications to save power and improve the user experience. With ALS solutions, system designers can automatically adjust display brightness based on the amount of ambient light. Since backlighting accounts for a significant portion of the system's power budget, dynamic brightness control can translate into substantial power savings. It can also improve the user experience, allowing screen brightness to be optimized based on ambient-light conditions.

Implementing such a system requires three sections: a light sensor to monitor the amount of ambient light, a device (usually a microcontroller) to process the data, and an actuator to control the current through the backlight.

Backlight control: the ambient-light sensor

Figure 1 provides an example block diagram of a system that implements backlight control. The light sensor is a key part of this setup, as it provides information about the environment's light level to the rest of the system. The light sensor must contain a transducer (e.g., a photodiode or CdS photoresistor) to convert light to an electrical signal, some amplification and/or signal conditioning, and an analog-to-digital converter (ADC).

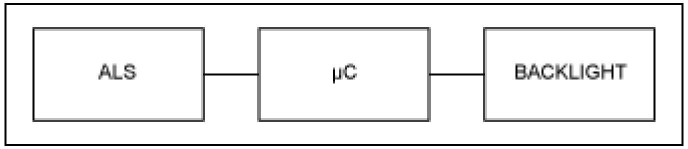


Figure 1. Block diagram for a system that implements backlight control.

Figure 2 shows a discrete implementation of a photodiode circuit. As you can see, the circuit requires one or more operational amplifiers: one for I-to-V conversion and, perhaps, a second for additional gain. It also includes extra routing to power all of these components and ensure a robust signal chain. In applications where space is at a premium, the large number of components required may be problematic.

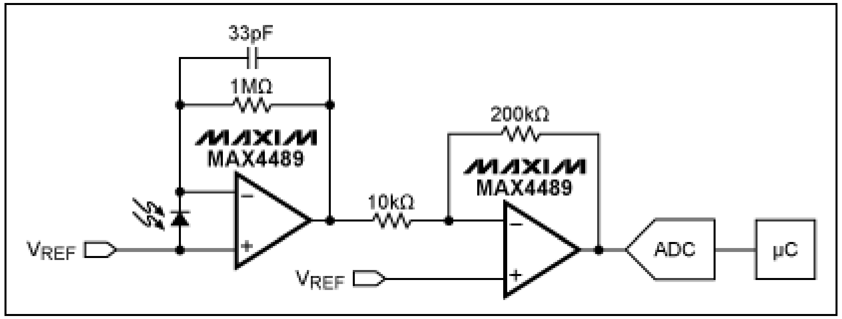


Figure 2. Discrete implementation of a photodiode circuit.

There is a second, more subtle, issue at hand here. Specifically, it is desirable to ensure that the ambient light is measured in a way that replicates the optical response of the human eye to light. This is often described with the CIE photopic curve (**Figure 3**). However, photodiodes rarely replicate this response, since they often have a heavy infrared (IR) sensitivity. This sensitivity causes false readings under IR-heavy light, such as that from incandescent bulbs or the sun.

One way around this is to use two photodiodes: one with a visible light plus infrared component, and one with only an infrared component. It is then possible to subtract the two responses from one another to obtain only the visible light portion, with a minimized infrared section.

Although effective, this solution adds to the space required by the discrete circuit described above. Additionally, it is very difficult, if not impossible, to match the discrete photodiodes closely enough to eliminate infrared interference. Dynamic range would likely be limited without very sophisticated implementations of the amplifier, such as log amps. It is difficult to obtain repeatable results with such a setup.

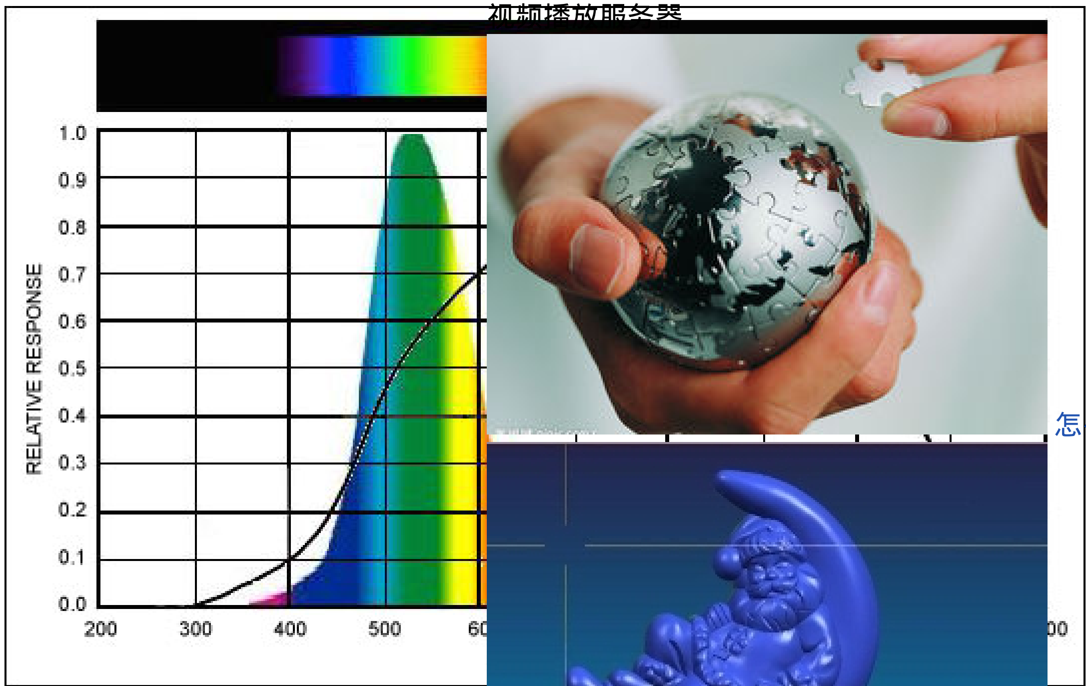


Figure 3. CIE curve vs. a typical phot

An integrated solution not only yields l response of the human eye, but also saves a great deal of space. A device such as the MAX9635 or MAX44009 ambient-light sensor integrates all signal conditioning and A/D conversion circuitry into a small form factor (2mm x 2mm UTDFN), saving considerable board real estate in space-constrained applications.

Figure 4 shows the functional block diagram for the MAX44009. It uses the I2C communication protocol to allow for a fast, simple method of interfacing to a microcontroller. In addition to this, the integrated nature of this solution enables it to be placed on a flex cable, and set in a desired location away from the main circuit board.

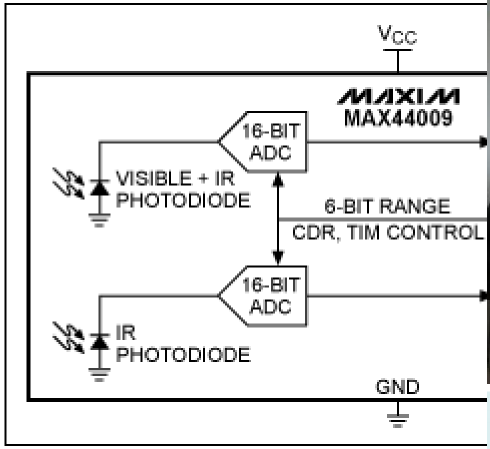


Figure 4. Functional block diagram f

Backlight control: modu

The second part of this control schem ... on the screen. This can be done in many ways, depending upon the screen module used in the ap ... ctly via a pulse-width modulation (PWM) scheme or indirectly by using a screen controller chip

Many display modules now have an i ... o directly set brightness by sending serial commands to the device. If this is not available, howev ... e implemented by controlling the power delivered to a series of white LEDs behind the screen, v ... of implementing this is by directly placing a FET in series with the LEDs and switching it on an ...). However, this can be done more elegantly and robustly with a single chip: the MAX1698 step ... See application note 3866, "Low-power PWM output controls LED brightness" for more details on this implementation.

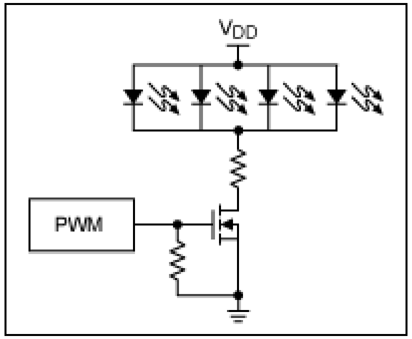


Figure 5. Simple PWM control circuit

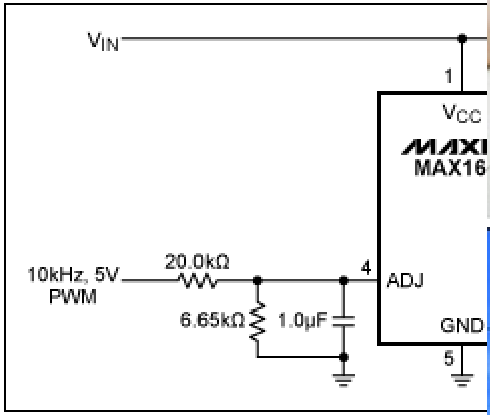
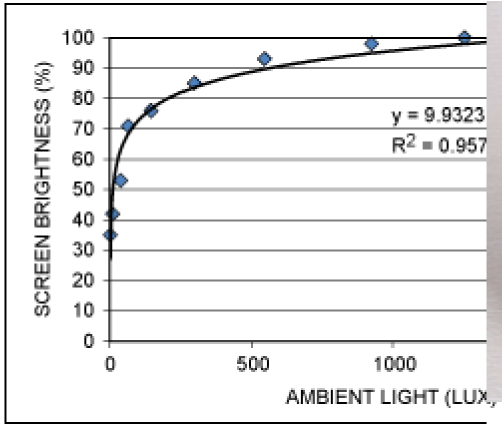


Figure 6. MAX1698-based LED regul

Backlight control: bridg

The final step is to bridge the gap between the sensor and the actuator, which is done in the microcontroller. The first question one may ask is: "How does one map ambient light to backlight brightness?" There are, in fact, specifications that describe how this should be done. One example of a mapping is recommended by Microsoft® for computers running Windows® 7.1 The curve in **Figure 7** is provided by Microsoft to map ambient-light levels to screen brightness as a percentage of full brightness.



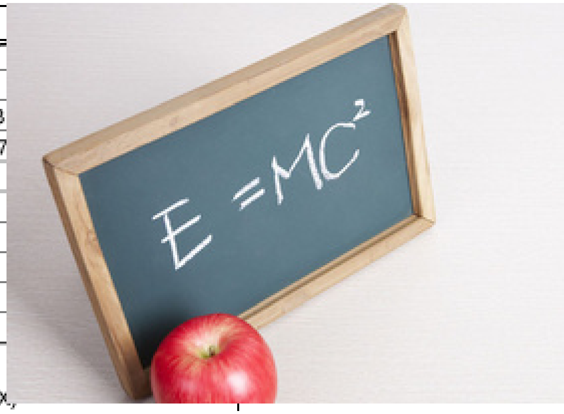


Figure 7. Example brightness curve m ... en brightness.

This particular curve can be describec

$$f(x) = \begin{cases} 9.9323\ln x + 27.059, & x < 1254\text{lux} \\ 100, & x \geq 1254\text{lux} \end{cases}$$

If the application utilizes an LCD con ... ntrol, then the brightness can easily be set by sending a co mmand to the chip with the desired va ... brightness, then one must consider how to map th e percentage signal into brightness.

In the example of the MAX1698, one ... cribed in the datasheet. From there, one can often assume t hat an LED's current is almost linearly ... ply constants into the equation above to account for mappi ng PWM into an effective voltage, wh ... ereby translating into screen brightness.

Notes on implementation

It is best not to jump directly from one setting to another: rather, the backlight brightness should be ramped up and down smoothly to ensure a seamless transition between levels. This is best done by using timed interrupts with either a fixed or variable brightness step size to gradua lly shift either the PWM value used to ... serial command sent to the display controller chip. **Figur e 8** provides an example of such an al...

嵌入式开发

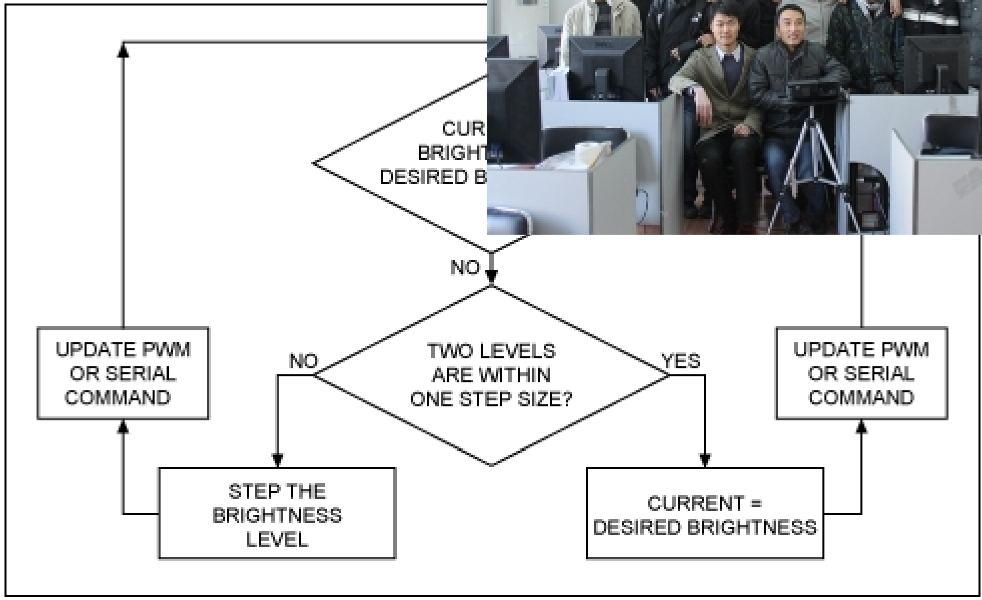


Figure 8. Example of an algorithm to step brightness.

Another concern is how quickly the system should respond to changes in ambient-light levels. One should avoid changing the brightness lev el too quickly. The concern is that transient changes in light (e.g., passing by a window or a lamp) can cause undesired changes in the backli ght brightness, which some users would find irritating. Furthermore, using a slower response time reduces the need to constantly poll the lig ht sensor, freeing some microcontroller resources.

A rudimentary approach is to poll the light sensor once every second or two, and then change the brightness. A better approach is to change the brightness only when the light level leaves a certain region for a specific amount of time. For example, if the current light level is 200lu x, one may only want to change the brightness if the light level falls below 180lux or rises above 220lux for longer than a few seconds. Fort unately, the MAX9635 and MAX44009 have an interrupt pin and threshold registers, making this very easy to do. See application note 478

6, "Interface code implementation for the MAX9635 ambient light sensor" for further details.

Appendix: sample code

```
#define MAX44009_ADDR    0x96
// begin definition of slave addresses for MAX44009
#define INT_STATUS      0x00
#define INT_ENABLE      0x01
#define CONFIG_REG      0x02
#define HIGH_BYTE       0x03
#define LOW_BYTE        0x04
#define THRESH_HIGH     0x05
#define THRESH_LOW      0x06
#define THRESH_TIMER         0x07
// end definition of slave addresses for MAX44009

extern float SCALE_FACTOR; // captures scaling factors to map from % brightness to PWM
float currentBright_pct;    // the current screen brightness, in % of maximum
float desiredBright_pct;    // the desired screen brightness, in % of maximum
float stepSize;             // the step size to use to go from the current
                    // brightness to the desired brightness
uint8 lightReadingCounter;

/**
 *    Function:       SetPWMDutyCycle
```

```
 *
 *    Arguments:    uint16 dc - desired duty cycle
 *
 *    Returns:  none
 *
 *    Description:   Sets the duty cycle of a 16-bit PWM, assuming that in this
 *                   architecture, 0x0000 = 0% duty cycle
 *                   0x7FFF = 50% and 0xFFFF = 100%
**/
extern void SetPWMDutyCycle(uint16 dc);

/**
 *    Function:       I2C_WriteByte
 *
 *    Arguments:    uint8 slaveAddr - address of the slave device
 *                  uint8 command - destination register in slave device
 *                  uint8 data - data to write to the register
 *
 *    Returns:  ACK bit
 *
 *    Description:   Performs necessary functions to send one byte of data to a
 *                   specified register in a specific device on the I2C bus
**/
uint8 2C_WriteByte(uint8 slaveAddr, uint8 command, uint8 data);
```

```
/**
 *   Function:       I2C_ReadByte
 *
 *   Arguments:   uint8 slaveAddr - address of the slave device
 *                uint8 command - destination register in slave device
 *                uint8 *data - pointer data to read from the register
 *
 *   Returns:  ACK bit
 *
 *   Description:   Performs necessary functions to get one byte of data from a
 *                specified register in a specific device  on the I2C bus
**/
uint8 I2C_ReadByte(uint8 slaveAddr, uint8 command, uint8* data);

/**
 *   Function:       getPctBrightFromLuxReading
 *
 *   Arguments:   float lux - the pre-computed ambient light level
 *
 *   Returns:  The % of maximum brightness to which the backlight should be set
 *                given the ambient light (0 to 1.0)
 *
 *   Description:   Uses a function to map the ambient light level to a backlight
 *                brightness by using a predetermined function
**/
```

```
float getPctBrightFromLuxReading(float lux);

/**
 *    Function:       mapPctBrighttoPWM
 *
 *    Arguments:    float pct
 *
 *    Returns:  PWM counts needed to achieve the specified % brightness (as
 *                    determined by some scaling factors)
**/
uint16 mapPctBrighttoPWM(float pct);

/**
 *    Function:       getLightLevel
 *
 *    Arguments:    n/a
 *
 *    Returns:  the ambient light level, in lux
 *
 *    Description:   Reads both the light registers on the device and returns the
 *                    computed light level
**/
float getLightLevel(void);

/**
```

```
 *    Function:       stepBrightness
 *
 *    Arguments:    n/a
 *
 *    Returns:  n/a
 *
 *    Description:    This function would be called by an interrupt. It looks at the
 *                    current brightness setting, then the desired brightness setting.
 *                    If there is a difference between the two, the current brightness
 *                    setting is stepped closer to its goal.
**/
void stepBrightness(void);

/**
 *    Function:       timerISR
 *
 *    Arguments:    n/a
 *
 *    Returns:  n/a
 *
 *    Description:    An interrupt service routine which fires every 100ms or so. This
 *                    handles all the ambient light sensor and backlight
 *                    control code.
**/
void timerISR(void);
```

```
void main() {

    SetupMicro();                  // some subroutine which initializes this CPU

    I2C_WriteByte(MAX44009_ADDR, CONFIG_REG, 0x80);    // set to run continuously

    lightReadingCounter = 0;
    stepSize = .01;
    currentBright_pct = 0.5;
    desiredBright_pct = 0.5;
    SetPWMDutyCycle(mapPctBrighttoPWM(currentBright_pct));
    InitializeTimerInterrupt();        // set this to fire every 100ms

    while(1) {
        // do whatever else you need here, the LCD control is done in interrupts
        Idle();
    }
} // main routine

// the point at which the function clips to 100%
#define MAXIMUM_LUX_BREAKPOINT      1254.0
float getPctBrightFromLuxReading(float lux) {
    if (lux > MAXIMUM_LUX_BREAKPOINT)
        return 1.0;
```

```
    else
        return (9.9323*log(x) + 27.059)/100.0;
} // getPctBrightFromLuxReading

uint16 mapPctBrighttoPWM(float pct) {
    return (uint16)(0xFFFF * pct * SCALE_FACTOR);
} // mapPctBrighttoPWM

float getLightLevel(void) {
    uint8* lowByte;
    uint8* highByte;
    uint8 exponent;
    uint8 mantissa;
    float result;

    I2C_ReadByte(MAX44009_ADDR, HIGH_BYTE, highByte);
    I2C_ReadByte(MAX44009_ADDR, LOW_BYTE, lowByte);

    exponent = (highByte & 0xF0) >> 4;// upper four bits of high byte register
    mantissa = (highByte & 0x0F) << 4;// lower four bits of high byte register =
                        // upper four bits of mantissa
    mantissa += lowByte & 0x0F;     // lower four bits of low byte register =
                        // lower four bits of mantissa

    result = mantissa * (1 << exponent) * 0.045;
```

```
        return result;
} //getLightLevel

void stepBrightness(void) {
        // if current is at desired, don't do anything
        if (currentBright_pct == desiredBright_pct)
                return;
        // is the current brightness above the desired brightness?
        else if (currentBright_pct > desiredBright_pct) {
                // is the difference between the two less than one step?
                if ( (currentBright_pct-stepSize) < desiredBright_pct)
                        currentBright_pct = desiredBright_pct;
                else
                        currentBright_pct -= stepSize;
        } // else if
        else if (currentBright_pct < desiredBright_pct) {
                // is the difference between the two less than one step?
                if ( (currentBright_pct+stepSize) > desiredBright_pct)
                        currentBright_pct = desiredBright_pct;
                else
                        currentBright_pct += stepSize;
        } // else if

        SetPWMDutyCycle(mapPctBrighttoPWM(currentBright_pct));
```

```
        return;
} // stepBrightness

void timerISR(void) {
        float lux;
        float pctDiff;

        stepBrightness();
        if (lightReadingCounter)
                lightReadingCounter--;
        else {
                lightReadingCounter = 20;    // 2 second delay
                lux = getLightLevel();
                desiredBright_pct = getPctBrightFromLuxReading(lux);
                pctDiff = abs(desiredBright_pct - currentBright_pct);
                stepSize = (pctDiff <= 0.01) ? 0.01:pctDiff/10;
        } // else

        ClearInterruptFlag();
} // timerISR
```

1http://www.microsoft.com/whdc/device/sensors/ambient-light-sensors.mspx

相关型号

MAX1698	高效率、升压型稳流器，用于LED驱动	免费样品
MAX44007	低功耗、环境光数字传感器，有效提高灵敏度	
MAX44009	业内功耗最低的环境光传感器，内置ADC	
MAX4489	SOT23封装、低噪声、低失真、宽带、满摆幅运算放大器	免费样品
MAX9635	业内功耗最低的环境光传感器，内置ADC	
MAXQ2000	低功耗LCD微控制器	免费样品

相关阅读:

python 教程 implementation deep learning 人工智能

一种新型滑模变结构观测器的永磁同步电动机矢量控制

控制科学与技术的发展及其思考

一种新型无桥部分有源PFC的实验研究

数字图像处理专业英语词汇

电力系统谐波测量方法综述

由浅入深，蓝牙4.0/BLE协议栈开发攻略大全（1）-网络/

交直流电压及测频板

黑龙江电网电压综合治理对策与应用

Microblaze与片内逻辑分析工具Chipscopepro

转载：uboot分析

单片机最小系统制作教程（转）

i2c读写心得(at24c02篇)

Android[LED]如何配置LCD背光和LED，调试方法

玩转你的MP3播放器之必学的基础知识

中文字库液晶模块在温室环境监控中的应用

笔记本LCD使用中须关注的七个问题

【程序】CAN驱动程序

CPLD为控制核心16位高精度数字电压表设计

笔记本液晶屏使用中应该注意的几个问题

单片机和嵌入式系统linux的区别

拨开迷雾-单片机和嵌入式LINUX开发的那点事儿

登录

来说两句吧...

还没有评论，快来抢沙发吧！

中国百科网正在使用畅言

时间:2011年02月14日 13:57:32 来源:maxim 作者: 上一篇：Hot-swap controller with calibration input can accurately mo 下一篇：Ramp generator controls DC-DC converters (电脑版 手机版)