キーワード: 環境光センサー, als, アルゴリズム, バックライト, I2C, 光センサー, 割込み, ヒューマンアイ, 人間の目, LCD, LEDバックライト, 疑似コード, タブレット, 電話, ノートブック, ネットブック, スマートフォン, TV

Feb 22, 2011

アプリケーションノート 4786

MAX9635環境光センサー用のインタフェースコードの実装

筆者：Prashanth Holenarsipur
Ilya Veygman, Strategic Applications Engineer

要約：このアプリケーションノートでは、ポータブル機器(スマートフォンやタブレットなど)のLCDバックライト管理のためのMAX9635環境光センサーにおける割込み出力機能の正しい使用法について説明します。この割込み機能を使えば、複数の照明ゾーンに対して光のスレッショルド値を調整することができるため、光センサーに繰り返しポーリングする必要がなくなります。この機能を正しく使用すると、システムを低電力スリープモードに保持することが可能で、またユーザーが定義したその他のタスクに他のリソースを割り当てることができます。割込み機能によってエネルギー効率、システム性能、およびさまざまな照明条件下でのユーザー体験が大幅に向上されます。このアプリケーションノートでは、割込みをプログラミングするためのI²CのC擬似コードの例も示しています。

はじめに

環境光センサーのMAX9635は、高度な手法を使用して環境光の明るさを検出します。このデバイスは、ポータブルエレクトロニクスやホームエレクトロニクスおよび室内照明でのディスプレイLCDのバックライト調整など他にも多くのアプリケーションで役立ちます。MAX9635は、超低動作電力(わずか0.65µA)で動作電圧が1.8Vであるため(マイクロコントローラのI/Oポートと簡単に連動可能)、いくつかのセンサーやセキュリティのアプリケーションで使用するのに魅力的なデバイスです。バックライトが調整可能で、低電力で動作するため、バッテリ寿命が伸び、照明アプリケーションのエネルギー効率が増大します。

MAX9635の最も有用な機能の1つが汎用性の高い割込み出力ピンです。このピンによって、システムを低電力スリープ状態に保持することが可能で、またユーザーにとって重要な他のタスクにリソースを割り当てることができます。

このアプリケーションノートでは、最適なシステム性能を確保するためにこの割込み出力機能をコーディングする方法について説明します。いくつかのCによる擬似コードを示します。

レジスタのプリセット

以下の表は、パワーオンリセット(POR)の各状態とともに、MAX9635のレジスタマップを示しています。

Register	Bit								Register address	Power-on RESET state	R/W
	7	6	5	4	3	2	1	0			
Status											
Interrupt status	—	—	—	—	—	—	—	INTS	0x00	0x00	R
Interrupt enable	—	—	—	—	—	—	—	INTE	0x01	0x00	R/W
Configuration											
Configuration	CONT	MANUAL	—	—	CDR	TIM[2:0]			0x02	0x03	R/W
LUX reading											
LUX high byte	E3	E2	E1	E0	M7	M6	M5	M4	0x03	0x00	R
LUX low byte	—	—	—	—	M3	M2	M1	M0	0x04	0x00	R
Threshold set											
Upper threshold high byte	UE3	UE2	UE1	UE0	UM7	UM6	UM5	UM4	0x05	0xFF	R/W
Lower threshold high byte	LE3	LE2	LE1	LE0	LM7	LM6	LM5	LM4	0x06	0x00	R/W
Threshold timer	T7	T6	T5	T4	T3	T2	T1	T0	0x07	0xFF	R/W

プリセットするレジスタは、Configuration、Interrupt Enable、およびThreshold Timerです。

ユーザーのほとんどのアプリケーションでは、Configurationレジスタ(アドレス0x02)のパワーアップ設定、CONT=0とMANUAL=0で十分です。これらの設定によって、MAX9635は、該当の環境光のレベルに基づいて感度を自動的にアップダウンするようになります。

割込み機能を有効にするため、マスターすなわちマイクロコントローラは最初にInterrupt Enableレジスタ(アドレス0x01)に1を書き込みます。

次にマスターは、Threshold Timerレジスタ(アドレス0x07)に適切な遅延を書き込みます。通常、この設定値は変更されません。このスレッショルド遅延を書き込むのには、2つの大きな理由があります。1つ目は、このレジスタにゼロ以外の値を書き込むことで、一瞬または一時的な照明条件の変化による厄介な作動を防止することができます。一瞬の照明の変化は、ユーザーの身振りや機器の移動の後に、その影が光センサーを横切るときに生じます。2つ目は、ディスプレイの明るさに応じて意図的に遅らせることによって、定義したユーザインタフェースのアルゴリズムに時間の余裕を持たせることできます。この例として、iPad™機器などのモバイルアプリケーションがあります。このような例では、断続的に照明が点在する地下道のような暗い通路を通過するときにディスプレイの明るさが急激に変動するのは好ましくありません。

Thresholdレジスタの設定

通常の動作時、ユーザーは、Upper Thresholdレジスタ(アドレス0x05)とLower Thresholdレジスタ(アドレス0x06)を繰り返しプログラムします。割込みは、これらのレジスタによって設定されたウィンドウレベルを環境光が超えたときにトリガされます(レジスタ0x00のINTSビットが1にセットされ、アクティブローINTハードウェアピンがローにプルダウンされる)。この割込みは、Threshold Timerレジスタ(アドレス0x07)によって設定された遅延よりも長く続きます。

Thresholdレジスタの設定をプログラムするため、マスターは最初にデータレジスタLUX High Byte (アドレス0x03)とLUX Low Byte (アドレス0x04)からルクスカウントを読み出して、現在の動作ゾーンを検出します。次にマスターは、Upper ThresholdレジスタとLower Thresholdレジスタの適切なカウントを設定します。

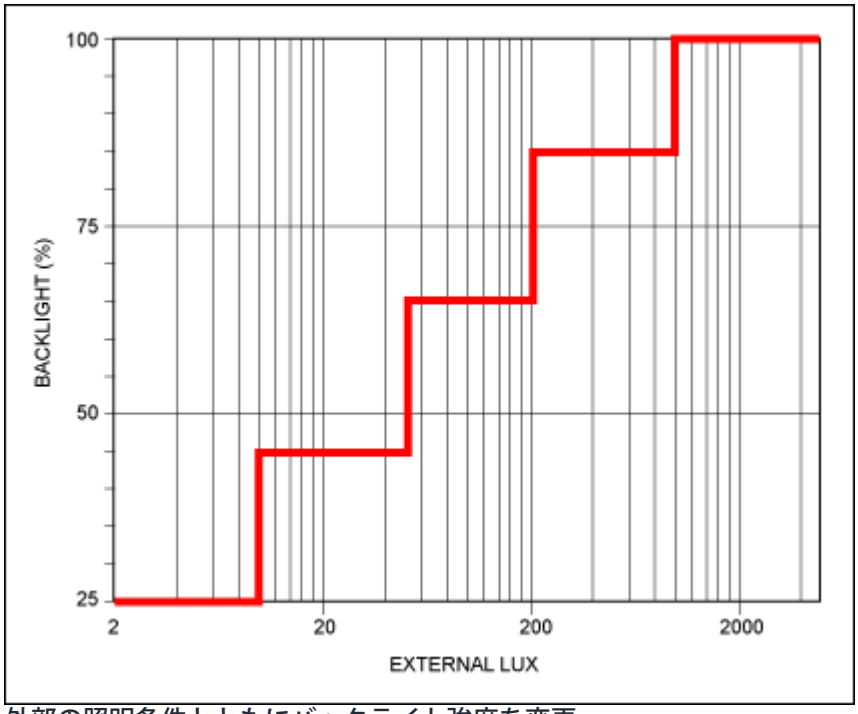
バックライト制御のアルゴリズム例

人間の目は、対数的に明るさを感じとります。これは人間の耳が音の大きさを感じとるのとまったく同じです。このため、バックライトの強度も通常、環境光のレベルに対数的に応答するようにプログラムされます。つまり、微光レベルでのステップは細かくなりますが、明るい環境光ではバックライトの強度はあまり変化しません。理想的には、ホストプロセッサが、この環境光レベルの情報に基づいてコントラストと色を調整するなど、高度な画像処理アルゴリズムを追加で実装することになります。

明るさを制御するための一般的なアルゴリズムは、制御のために5つのスレッショルドを備えています。ほとんどの場合、表面ガラスの種類と物理的な開口部のサイズによっては光が減少し、光センサーは外部環境光の5%～10%しか検出できなくなります。スレッショルドレベルを設定するときには、この光の減少を考慮に入れる必要があります。

以下の表は、バックライト強度および上側と下側のスレッショルドの一例です。スレッショルドのルクスをスレッショルドのカウントに変換するには、単に対象のルクス設定値を0.045で割るだけです。

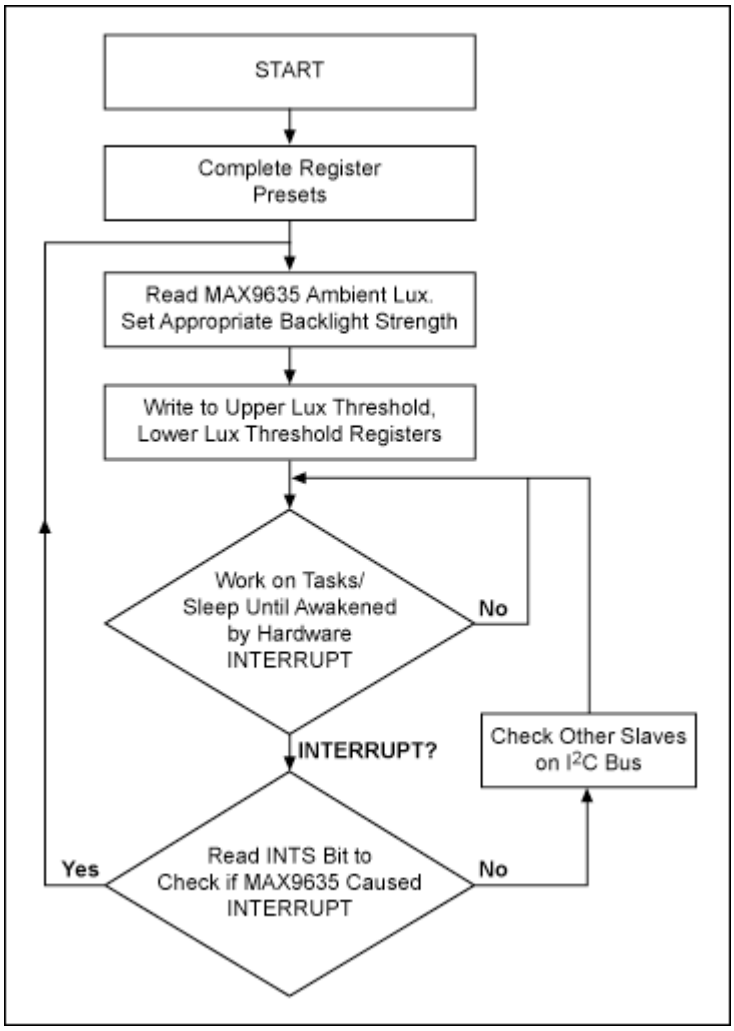
Illumination Zone	External Lux (typ)	Backlight Strength (%)	External Lux, Lower Threshold (typ)	External Lux, Upper Threshold (typ)	Lower Threshold (10% Glass)	Upper Threshold (10% Glass)
Dark	4	25	< 0	> 10	< 0	> 1
Dim	20	45	< 10	> 50	< 1	> 5
Home	100	65	< 50	> 200	< 5	> 20
Office	400	85	< 200	> 1000	< 20	> 100
Sunlight	> 2000	100	< 1000	> Maximum	< 100	> Maximum



外部の照明条件とともにバックライト強度を変更

割込みの実装

以下の図は、マスターのマイクロコントローラによって実装されたフローチャートの標準的な例を示しています。



アルゴリズムのスレッショルドレベルと周囲の測定：カウント対ルクス

ルクス値よりもカウントでアルゴリズムを実装する方が簡単です。こうすることで浮動小数点の計算を使用する必要がなく、マイクロコントローラの単純な固定小数点コードが可能です。

環境光カウント	2^(指数) × 仮数
	指数 = 8xE3 + 4xE2 + 2xE1 + E0
	仮数 = 128xM7 + 64xM6 + 32xM5 + 16xM4 + 8xM3 + 4xM2 + 2xM1 + M0
上側スレッショルドカウント	2^(指数) × 仮数
	指数 = 8xE3 + 4xE2 + 2xE1 + E0
	仮数 = 128xM7 + 64xM6 + 32xM5 + 16xM4 + 15
下側スレッショルドカウント	2^(指数) × 仮数
	指数 = 8xE3 + 4xE2 + 2xE1 + E0
	仮数 = 128xM7 + 64xM6 + 32xM5 + 16xM4

上の表から所望のスレッショルドを使用すれば、各照明ゾーンの擬似コードの限界として使用するThresholdレジスタのバイトを計算することができます。これらのスレッショルドは、上式で計算した環境光カウントと単純に比較されます。

Zone	Lower Threshold, 10% Glass (Lux)	Upper Threshold, 10% Glass (Lux)	Desired Lower Threshold Counts	Desired Upper Threshold Counts	Lower Threshold Register Byte	Upper Threshold Register Byte	Actual Lower Threshold Counts	Actual Upper Threshold Counts	Actual Lower Threshold (Lux)	Actual Upper Threshold (Lux)
Dark	< 0	> 1	0	22	0000 0000	0000 0001	0	31	< 0	> 1.395
Dim	< 1	> 5	22	111	0000 0001	0000 0110	16	111	< 0.72	> 4.995
Home	< 5	> 20	111	556	0000 0110	0010 1001	96	636	< 4.32	> 28.62
Office	< 20	> 100	556	2222	0010 1001	0100 1000	576	2288	< 25.92	> 102.96
Sunlight	< 100	> Maximum	2222	4177920	0100 1000	1110 1111	2048	4177920	< 92.16	> 188006

留意が必要ですが、光の動作レベルが、定義した照明ゾーンの境界に極めて近い場合、バックライトのレベルが頻繁に変動する可能性があり、ユーザーを不快にするおそれがあります。このため、ある照明ゾーンの上側スレッショルドと次の上位の照明ゾーンの下側スレッショルドとの間に小さなオーバーラップゾーンが設けられています。これによって自然なヒステリシスが得られ、光の小さな変動に対するシールドとしての役割を果たします。このオーバーラップは必要に応じてさらに拡張することができます。

ここに記したアルゴリズムは、バックライト輝度制御の1つの可能な実装についての一般的なガイドラインに過ぎません。バックライト制御の技術の進歩によって、多くのさまざまなアルゴリズムが開発され、洗練された光の透過感をエンドユーザーに与えています。

C擬似コードの実装サンプル

```
// begin definition of slave device address
#define MAX9635_WR_ADDR          0x96
#define MAX9635_RD_ADDR          0x97

// begin definition of slave register addresses for MAX9635
#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 MAX9635

// define some lookup tables for the upper and lower thresholds as well as the
// brightness. All tables values are taken from text of application notes
#define NUM_REGIONS              5
uint8 upperThresholds[NUM_REGIONS] = {0x01, 0x06, 0x29, 0x48, 0xEF};
```

```
uint8 lowerThresholds[NUM_REGIONS] = {0x00, 0x01, 0x06, 0x29, 0x48};
uint8 backlightBrightness[NUM_REGIONS] = {0x40, 0x73, 0xA6, 0xD9, 0xFF};

/**
        Function:       SetPWMDutyCycle

        Arguments:      uint8 dc - desired duty cycle

        Returns:        none

        Description:    sets the duty cycle of a 8-bit PWM, assuming that in this
                        architecture, 0x00 = 0% duty cycle 0x7F = 50% and 0xFF = 100%
**/
extern void SetPWMDutyCycle(uint8 dc);
extern void SetupMicro(void);
extern void Idle(void);

/**
        Function:       I2C_WriteByte

        Arguments:      uint8 slaveAddr - address of the slave device
                        uint8 regAddr - 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
**/
extern uint8 I2C_WriteByte(uint8 slaveAddr, uint8 regAddr, uint8 data);

/**
        Function:       I2C_ReadByte

        Arguments:      uint8 slaveAddr - address of the slave device
                        uint8 regAddr - 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
**/
extern uint8 I2C_ReadByte(uint8 slaveAddr, uint8 regAddr, uint8* data);

/**
        Function:       findNewThresholdsAndBrightness

        Arguments:      uint8 luxCounts - light counts High Byte
                        uint8 *highThresh - pointer to memory storing upper threshold byte
                        uint8 *lowThresh - pointer to memory storing lower threshold byte

        Returns:        none

        Description:    Based on what the lux reading was (in counts), this routine
                        determines the current operating illumination zone. The zones
                        are defined by upper and lower bounds in a lookup table. After
                        knowing the operating zone, this function may set new interrupt
                        thresholds and a backlight brightness. Since the interrupt only
                        fires when the lux reading is outside the defined region, these
                        threshold and brightness settings are not overwritten with the
                        same data repeatedly.
**/
void findNewThresholdsAndBrightness(uint8 luxCounts, uint8 *highThresh,
                                    uint8 *lowThresh);

void main() {

        uint8 *highThresholdByte;       // upper and lower threshold bytes
        uint8 *lowThresholdByte;
        uint8 *timerByte;
        uint8 max9635Interrupt = 0;     // status of MAX9635 interrupt register
        uint8 luxCounts;                // computed as shown below

        SetupMicro();                   // some subroutine which initializes this CPU
        *highByte = 0;
        *lowByte = 0;
        *highThresholdByte = 0xEF;      // upper threshold counts
                                        // initially = POR setting (maximum possible = 0xEF)
        *lowThresholdByte = 0x00;       // lower threshold counts
                                        // initially POR setting (minimum possible = 0x00)
        *timerByte = 0x14;              // initial timer delay for thresholds:
                                        // 0x14 * 100ms = 2 seconds

                                        // initialize MAX9635 threshold and timer registers
        I2C_WriteByte(MAX9635_WR_ADDR, THRESH_HIGH, *highThresholdByte);
```

```
        I2C_WriteByte(MAX9635_WR_ADDR, THRESH_LOW, *lowThresholdByte);
        I2C_WriteByte(MAX9635_WR_ADDR, THRESH_TIMER, *timerByte);
        I2C_WriteByte(MAX9635_WR_ADDR, INT_ENABLE, 0x01);// enable sensor interrupts

        while(1) {

                // do other tasks until an interrupt fires
                // assume that this function waits for the status of a GPIO-type pin to
                // change states
                while (! GPIO_StatusChanged() ) {
                        // some idling subroutine, shown with polling a port for
                        // simplicity - but alternate interrupt-based routines are more
                        // efficient
                        Idle();
                } // loop until an interrupt occurs

                // ok... an interrupt fired! was it from the MAX9635?
                I2C_ReadByte(MAX9635_RD_ADDR, INT_STATUS, max9635Interrupt);

                /**
                        Place code to check other devices here, if desired
                **/

                if (max9635Interrupt) {
                        // get the current lux reading from the MAX9635
                        I2C_ReadByte(MAX9635_RD_ADDR, HIGH_BYTE, luxCounts);
                        findNewThresholdsAndBrightness(luxCounts, highThresholdByte,
                                                        lowThresholdByte);

                        // write to the threshold and timer registers with new data
                        I2C_WriteByte(MAX9635_WR_ADDR, THRESH_HIGH, *highThresholdByte);
                        I2C_WriteByte(MAX9635_WR_ADDR, THRESH_LOW, *lowThresholdByte);

                        max9635Interrupt = 0;   // interrupt serviced, clear the bits
                } // only executes if the MAX9635's interrupt fired

                // perform other tasks which are only done after change of a GPIO pin
        } // loop forever

} // main routine

void findNewThresholdsAndBrightness(uint8 luxCounts, uint8 *highThresh, uint8 *lowThresh) {

        uint8 i;

        for (i=0; i < NUM_REGIONS; ++i) {
                if ((luxCounts >= lowerThresholds[i]) && (luxCounts <= upperThresholds[i])){
                        *highThresh = upperThresholds[i];
                        *lowThresh = lowerThresholds[i];
                        // PWM duty cycle sets the brightness of the backlight
                        SetPWMDutyCycle(backlightBrightness[i]);
                        return;  // found the region -- no point in continuing the loop
                } // found the right region
        } // check where the lux reading lies in terms of threshold regions

} // findNewThresholdsAndBrightness
```

関連製品

MAX44007	感度が向上された、低電力ディジタル環境光センサー
MAX9635	業界最低の消費電力、ADC内蔵環境光センサー

自動アップデート

お客様が関心のある分野でアプリケーションノートが新規に掲載された際に自動通知Eメールの受信を希望する場合は、EE-Mail™にご登録ください。

その他の情報

テクニカルサポート：http://japan.maxim-ic.com/support
サンプル請求：http://japan.maxim-ic.com/samples
その他の質問およびコメント：http://japan.maxim-ic.com/contact

アプリケーションノート4786: http://japan.maxim-ic.com/an4786
AN4786, AN 4786, APP4786, Appnote4786, Appnote 4786

法的お知らせ：http://japan.maxim-ic.com/legal