SmartCap

Capacete de segurança automatizado

Thiago Gomes de Sousa Bezerra
Universidade de Brasília
Faculdade Gama
Gama, Distrito Federal
thiagotnd@hotmail.com

Diogo Gomes de Sousa Bezerra
Universidade de Brasília
Faculdade Gama
Gama, Distrito Federal
diogogsb@hotmail.com

I. INTRODUÇÃO

Na sociedade contemporânea é comum a pratica de exploração em minas, as quais tem por objetivo a extração de determinados minérios, como é o caso do carvão e das pedras preciosas. Tal prática é uma das várias responsáveis pelo crescimento da economia mundial, entretanto, os profissionais que trabalham nesses locais podem estar sujeitos a determinados perigos e condições adversas, perigos os quais podem, até mesmo, leva-los a óbito.

Dentre essas ameaças, existem aquelas que são, muitas das vezes, imperceptíveis para os trabalhadores, como é o caso dos vazamentos de gases nocivos à saúde, tais como CO, H_2S, NO, NO_2 e O_2, produzidos pela utilização de explosivos e motores a combustão[4], além do gás natural. Estes são responsáveis por explosões, desmaio de trabalhadores em zonas de difícil acesso e riscos futuros a saúde. Pode-se citar também a exposição a altas temperaturas, que podem causar doenças e/ou agravantes das mesmas, como é o caso de queimadoras de pele.

Além do mais, é comum situações que impossibilitam a prática do trabalho em questão, como é o caso da falta de iluminação em determinados pontos dessas minas, a qual pode ser ocasionada por diversas situações.

II. OBJETIVO

Através de um capacete de segurança, conseguir identificar situações adversas provenientes do meio, como cavernas ou/e ambientes de mineração, tais como a identificação de gases tóxicos prejudiciais a saúde e falta de luminosidade. Assim, quando ocorrer tais identificações o dispositivo enviará ao usuário um alerta de segurança.

Com tudo, o intuito deste projeto é a automação de um equipamento de segurança individual (EPI), mais especificamente o capacete de segurança, com o cuidado de minimizar e prevenir situações adversas ao usuário.

III. BENEFÍCIOS E REQUISITOS

A partir das situações apresentadas, propõem-se o desenvolvimento de um equipamento autônomo para suprir tais necessidades, o SmartCap.

O SmartCap será um equipamento que auxiliará os trabalhadores em determinadas ocasiões. Este auxilio será feito através da comunicação entre um microcontrolador (MSP430) e sensores capazes de captar diversas perturbações, como a presença de gases nocivos e a falta de luminosidade, além de componentes que irão auxiliar e avisar. Quando o sensor de gás nocivo identificar a presença desses gases, o equipamento retornará uma resposta ao usuário através de um sinal sonoro, proveniente de um buzzer. Já quando o sensor de luminosidade (LDR) identificar a falta de luminosidade no ambiente, uma lanterna fixada ao capacete será ligada, afim de iluminar o ambiente.

IV. SOFTWARE E HARDWARE

MATERIAIS UTILIZADOS

Na tabela 1 pode-se observar os materiais utilizados para o desenvolvimento do projeto.

Tabela 1 – Materiais utilizados no desenvolvimento do projeto.

Materiais utilizados
Capacete de Segurança
MSP430
Sensor de gás – MQ2
Sensor de luminosidade – LDR
Jumpers
Protoboard
Led
Buzzer
Resistor ()
Potenciômetro 1kΩ

HARDWARE

O LDR (Light Dependent Resistor) é basicamente um sensor fotossensível cuja resistência varia conforme a incidência de luz sobre seu corpo, este sensor é mostrado na figura 1.1.[6] Este sensor é composto por uma área exposta à luz feita de material semicondutor que varia sua resistência elétrica conforme o nível de incidência de raios luminosos; isto porque todo material semicondutor é sensível à radiação de alguma forma (luminosa ou não, visível ou não). A resistência desse componente diminui ao aumentar a incidência de luz no mesmo, como mostrado no gráfico presente na figura 1.2.

Figura 1.1: LDR

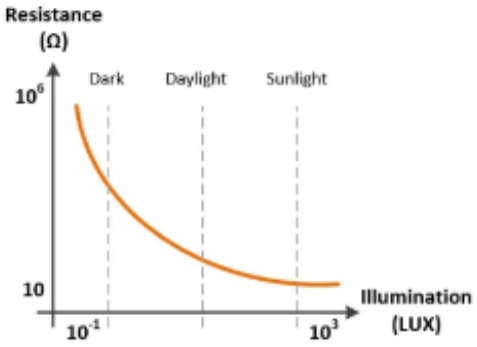

Figura 1.2: Variação da resistência conforme a incidência de luz.

O sensor de gás utilizado para o projeto é o MQ-2, [5] este sensor é capaz de detectar concentrações de gases combustíveis e fumaça no ar. Quando a concentração de gases fica acima do nível, o qual é ajustado por um potenciômetro presente no módulo utilizado, a saída digital D0 fica em estado alto, caso contrário a saída fica em estado baixo. Para ter uma resolução melhor e medir a variação da concentração dos gases no ar é possível usar a saída analógica A0 e conectar a um conversor AD (analógico/digital), como o presente no microcontrolador MSP430 utilizado por exemplo. O MQ-2 é capaz de detectar GLP, Metano, Propano, Butano, Hidrogênio, Álcool, Gás Natural, além de outros gases inflamáveis. Na figura 2 pode-se observar o sensor de gás MQ-2 a ser utilizado no projeto.

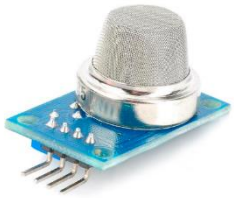

Figura 2 – Sensor de gás MQ-2.

Para a implementação do projeto na placa MSP430, utilizou-se os pinos analógicos A0 e A5, que na placa são descritos como P1.0 e P1.5, e os pinos digitais P1.1 e P1.3 da placa.

Os pinos analógicos foram utilizados a fim de fazer a leitura dos valores de tensão presentes na saída do sensor de gás e no circuito resistivo formado pelo LDR e um resistor (sendo que a resistência do LDR varia com a luz), sendo que esses valores são convertidos internamente no MSP430 (por um conversor A/D) e expressos em valores entre 0 e 1023. No sensor de gás, a tensão aferida pela porta varia a partir da concentração de gases detectados, já no LDR a tensão varia com a luminosidade captada pelo mesmo.

Já os pinos digitais correspondem as entradas do LED e Buzzer, entradas as quais dependem dos valores de tensões, captados no LDR e sensor de gás, e aferidos pelas portas analógicas A0 e A5, respectivamente, de modo que as saídas digitais comecem em baixo (0) e vão para alto (1) quando um determinado valor de tensão for obtido.

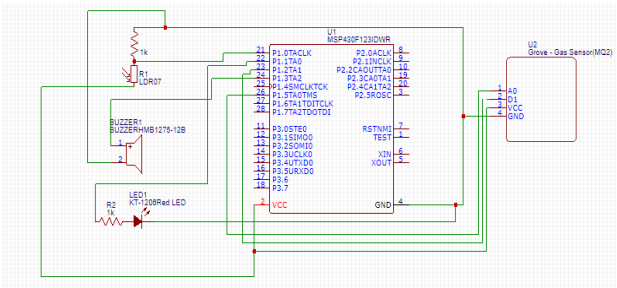

Figura 3.1 – Conexões com os sensores e o microcontrolador.

Como já mencionado, o hardware será desenvolvido e montado em um capacete de segurança, como exibido na figura 3.2, de forma com que não haja qualquer influência do mesmo que comprometa a integridade e segurança do EPI.

Figura 3.2 – Capacete de segurança.

SOFTWARE

Inicialmente incluiu-se algumas bibliotecas e definiu-se algumas variáveis, além de definir algumas portas como entradas e saídas, bem como alguns valores iniciais de saída e uma variável global.

```
#include <msp430.h>
#include <msp430g2553.h>

#define LDR BIT0
#define GAS BIT1
#define BUZZER BIT2
#define LED BIT3

void Init_AD(void);
unsigned int valor[2];
```

Figura 4 – Definições de algumas variáveis.

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; //Desliga Wtch_dg_timer

    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    P1OUT &= ~(BUZZER + LED);
    P1DIR |= BUZZER + LED;
```

Figura 5 – Definições de entradas e saídas.

Os pinos P1.0 e P1.1 correspondem as entradas analógicas do LDR e do sensor de gás, respectivamente, sendo estas entradas, enquanto as saídas BUZZER e LED correspondem aos pinos P1.2 e P1.3 do MSP430, respectivamente.

Como mencionado, o projeto utiliza um sensor analógico (sensor de gás nocivo) e uma resistência que varia com a luz (LDR), utilizados afim de identificar as situações exigidas. Para tais sinais serem reconhecidos pelo microcontrolador, foi necessário o uso de um conversor analógico-digital de 10 bits presente no MSP430 (ADC10), pois, é necessário ler os valores de tensão (sinal analógico) da saída do sensor de gás e no LDR, valores os quais variam para cada fator (gás nocivo e luz). Os valores de tensão podem ser lidos nas portas P1.0 à p1.1 do MSP430 utilizado. Nóta-se que as flags ADC10ON e ADC10IE setadas no registrador ADC10CTL0 habilitam a conversão AD e a interrupção, respectivamente, do vetor ADC10. Os pinos analógicos de entrada (A0 e o A1), foram escolhidos de forma com que a rotina da conversão (ADC), para ambas as entradas, fosse realizada de maneira crescente, a partir do primeiro pino analógico P1.0. Para tal, foram utilizadas as flegs INCH_1 (habilita as entradas analógicas A0 e o A1) e CONSEQ_1 (habilita a conversão em modo sequencial), ambos presentes no registrador ADC10CTL1. A conversão realizada nestes pinos foi habilitada colocando o valor do registrador ADC10AE0 = BUZZER + LDR e configuramos para ADC10DTC1 = 0x2, assim configurando-se duas conversões para cada bloco. Para habilitar a múltipla conversão AD, utilizou-se a flag MSC, setando-a em 1. Esta flag foi determinada para dois canais através da flag ADC10SHT_2, ambas no registrador ADC10CTL0. Na linha 34, observa-se o início da conversão, sendo ela habilitada e iniciada já na linha 36, como observado na figura 6. O resultado obtido através da conversão é

```
33        ADC10CTL0 &= ~ENC;
34        while (ADC10CTL1 & BUSY);
35        ADC10SA = valor;
36        ADC10CTL0 |= ENC + ADC10SC; //inicia a conversão AD
```

armazenado no registrador ADC10SA e passado para a variável "valor", a qual trata-se de um vetor inteiro com valores positivos de duas posições, uma para cada sensor no caso.

Figura 6 – Inicialização da conversão AD.

Os valores obtidos através da conversão AD ficam armazenadas na memória, em um registrador específico do conversor (ADC10SA), como já mencionado. Assim, quando o microcontrolador identificar um nível maior que o limite pré-determinado dos sensores, a resposta será acionada, como o buzzer para o gás e a lanterna para o ldr. Nas interrupções, habilitou-se também o modo de baixo consumo, assim como o desligamento da CUP, visto que para uma conversão múltipla, já existe registradores que armazenam tais valores em lugares específicos da memória. A interrupção foi habilitada no vetor ADC10, como observado na figura 7.

```
54 #pragma vector=ADC10_VECTOR
55 __interrupt void ADC10_ISR(void)
56 {
57 __bic_SR_register_on_exit(CPUOFF);
58 }
```

Figura 7 – Código de interrupção

No pino P1.1 (A1) será lido o valor analógico referente ao sensor de gás. Para obtermos um valor especifico para a nossa aplicação, foram realizados testes e comparações, utilizando gás de cozinha, tendo como base a tensão de referência (1023). Com isso, chegamos em uma faixa de valores consideráveis de tensão para a nossa aplicação, fixando o limite de gás em 115. Já para o LDR, pino P1.0, fixamos um limite de 711, que também foi medido de forma experimental, visto que o LDR varia a sua resistência de acordo com a intensidade da luz. Os valores captados pelos sensores foram armazenados na variável res [], contendo duas componentes, uma para cada sensor.

Assim, observando o valor de tensão na variável res[], pode-se impor as respostas do nosso projeto, sendo ela o buzzer (pino P1.2) e o Led (pino P1.3). Tais respostas foram desenvolvidas de maneira simples no código, no qual se res[0] > 711 (LDR) houve uma perda de intensidade luminosa, logo o BIT3 vai para

```
__bis_SR_register(CPUOFF + GIE);
    if (valor[0]>115){
        P1OUT |= BUZZER;
    }
    else{
        P1OUT &= ~BUZZER;
    }
    if(valor[1]>711){
        P1OUT |= LED;
    }
    else{
        P1OUT &= ~LED;
    }
}
```

alto ligando o LED, caso contrário ele ficará apagado. Já se res[1] > 115 (Gás), indica que o nível de gás está ultrapassando o limite convencionado, setando assim o BIT3 o que ativa o buzzer.

Figura 8 – Código de resposta para os sensores .

REFERENCIAS BIBLIOGRÁFICAS

[1] Ardumotive_com. Arduino smart working helmet. Disponível em: http://www.instructables.com/id/Arduino-Smart-Working-Helmet/

[2] A. C. Ramos Gonçalves. Riscos associados a exploraão mineira. Disponível em: http://www.uc.pt/fluc/depgeo/Cadernos_Geografia/Numeros_publicados/CadGeo30_31/Eixo1_9.

[3] Oliveira OIT – Organizaão Nacional do Trabalho, Sobre saúde e segurana nas minas. Disponível em: http://www.oitbrasil.org.br/content/sobre-seguran%C3%A7a-e-sa%C3%BAde-nas-minas.

[4] P. Cézanne Pinto. Avaliação das condições ambientais na mineração em subsolo. Disponível em: http://www.scielo.br/pdf/rem/v59n3/v59n3a10

[5] FilipiFlop, Sensor de gás e fumaça MQ-2. Disponível em: https://www.filipeflop.com/produto/sensor-de-gas-mq-2-inflamavel-e-fumaca/

[6] FritzenLab, Como funciona um LDR, Disponível em: http://fritzenlab.com.br/2016/01/como-funciona-um-ldr-resistor-dependente-de-luz/

ANEXOS

```
#include <msp430.h>
#include <msp430g2553.h>

#define LDR BIT0
#define GAS BIT1
#define BUZZER BIT2
#define LED BIT3

void Init_AD(void);
unsigned int valor[2];

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; //Desliga Wtch_dg_timer

    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    P1OUT &= ~(BUZZER + LED);
    P1DIR |= BUZZER + LED;

    Init_AD();

    for (;;)
    {
        TACCR0 = 31250-1;
        TACCTL1 = OUTMOD_7;
        TACTL = TASSEL_2 | ID_3 | MC_1;
        while ((TACTL & TAIFG)==0){
        }
        TACTL &= ~TAIFG; //ATRASO DE 0,5s
        ADC10CTL0 &= ~ENC;
        while (ADC10CTL1 & BUSY);
        ADC10SA = valor;
        ADC10CTL0 |= ENC + ADC10SC; //inicia a conversão AD

    __bis_SR_register(CPUOFF + GIE);
        if (valor[0]>115){
            P1OUT |= BUZZER;
        }
        else{
            P1OUT &= ~BUZZER;
        }
        if(valor[1]>711){
            P1OUT |= LED;
        }
        else{
            P1OUT &= ~LED;
        }
    }
}

#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
__bic_SR_register_on_exit(CPUOFF);
}

void Init_AD(void){
    WDTCTL = WDTPW + WDTHOLD;
    ADC10CTL1 = INCH_1 + CONSEQ_1;
    ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
    ADC10AE0 = LDR + GAS;
    ADC10DTC1 = 0x2;
    P1OUT &= ~(BUZZER + LED);
    P1DIR |= BUZZER + LED;
}
```