

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 prática 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, H₂S, NO, NO₂ e O₂, 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 queimaduras 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 auxílio 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

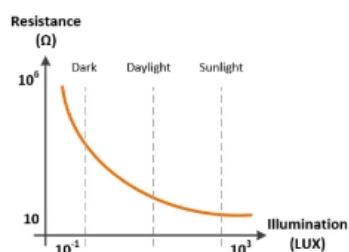


Figura 1.2: Variação da resistência conforme a incidência de luz.

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.

Nas figuras 3.1 e 3.2 pode-se observar o esquemático do circuito a ser desenvolvido, contendo os sensores e suas devidas conexões.

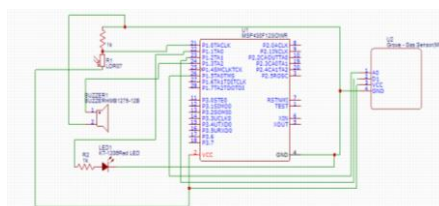


Figura 3.1 – Conexões com os sensores e o microcontrolador.

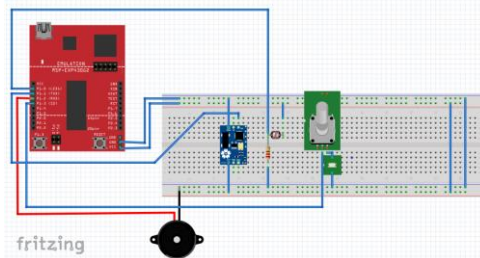


Figura 3.2 – Esquemático do circuito a ser desenvolvido.

Como já mencionado, o hardware será desenvolvido e montado em um capacete de segurança, como exibido na figura 3.3, de forma com que não haja qualquer influência do mesmo que comprometa a integridade e segurança do EPI.



Figura 3.3 – 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.

```
1 #include <msp430.h>
2 #include <msp430g2553.h>
3
4 #define LDR BIT0
5 #define GAS BIT1
6 #define BUZZER BIT2
7 #define LED BIT3
8
9 void Init_AD(void);
10 unsigned int valor[2];
```

Figura 4 – Definições de algumas variáveis.

```
11 void main(void)
12 {
13     MDCTCL = MDTPW + MDTHOLD; //Desliga Watch dog timer
14
15     BCSCTL1 = CALDC1_1MHZ;
16     DCOCTL = CALDC0_1MHZ;
17
18     P1OUT &= ~(BUZZER + LED);
19     P1DIR |= BUZZER + LED;
20
21 }
```

Figura 5 – Definições de entradas e saídas.

```
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

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. Nota-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 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 específico 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 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 convencional, setando assim o BIT3 o que ativa o buzzer.

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

Figura 8 – Código de resposta para os sensores.

V. RESULTADOS

O circuito foi montado no capacete de segurança, além de construído uma placa de circuito impresso observando a viabilidade do projeto, assim como a otimização de espaço. A placa de circuito impresso (PCI) e suas devidas conexões podem ser observadas na figura 9.

O capacete apresentou uma ótima resposta as situações propostas a ele, como na conversão AD realizada de forma simultânea para cada sensor. Os sensores foram posicionados de forma estratégica no capacete, buscando-se o melhor funcionamento cada sensor.

Nas figuras a seguir, pode-se observar o protótipo do capacete desenvolvido, assim como os sensores e suas devidas ligações.



Figura 9 – PCI e conexões do capacete de segurança.

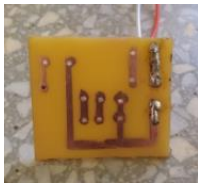


Figura 10 – Placa de circuito impresso.



Figura 11 – Capacete de segurança automatizado.



Figura 12 – Capacete de segurança finalizado.

VI. RESULTADOS CONCLUSÃO

Acerca dos resultados obtidos, constata-se que o SmartCap foi capaz de atender todas as especificações propostas. Os componentes necessários foram dispostos no capacete de modo a não prejudicar as propriedades do mesmo, assegurando a integridade de quem o utiliza. Todos os componentes foram fixados ao capacete, assim como os fios que os interliga, afim de evitar possíveis curtos e desconexões.

Ao decorrer do projeto, a utilização de diversos conhecimentos obtidos em sala de aula durante toda o semestre torna-se explícito, conhecimentos de microcontroladores, bem como a disposição de seus registradores e utilização dos mesmos, além de diversas outras técnicas de programação.

REFERENCIAS BIBLIOGRÁFICAS

- [1] Arduomotive_com. Arduino smart working helmet. Disponível em: <http://www.instructables.com/id/Arduino-Smart-Working-Helmet/>
- [2] A. C. Ramos Gonçalves. Riscos associados a exploração mineira. Disponível em: http://www.uc.pt/fluc/depgeo/Cadernos_Geografia/Numeros_publicados/CadGeo30_31/Eixo1_9.
- [3] Oliveira OIT – Organização Nacional do Trabalho, Sobre saúde e segurança nas minas. Disponível em: <http://www.oitbrasil.org.br/content/sobre-seguran%C3%A7a-e-sa%C3%BAde-nas-minas.>
- [4] P. Cézarne 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.filipiflop.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; //ATRASSO 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;
}

```