

**Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores**

**Trabalho Prático 4 - Jogo Squash para 1**

47206 : Tiago Pardal (47206@alunos.isel.pt)

47202 : Manuel Henriques (47202@alunos.isel.pt)

48052 : Manuel Fonseca (48052@alunos.isel.pt)

Relatório para a Unidade Curricular de Arquitetura de Computadores  
da Licenciatura em Engenharia Informática e de Computadores

Professor : Engenheiro Jorge Fonseca



# Índice

<b>Lista de Figuras</b>	<b>v</b>
<b>Lista de Listagens</b>	<b>vii</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Problema</b>	<b>3</b>
2.1 Squash para Um . . . . .	3
2.2 Sistema usado . . . . .	4
<b>3 Detalhes Implementação</b>	<b>5</b>
3.1 Lógica programa . . . . .	5
3.1.1 Ciclo de jogo . . . . .	6
3.1.2 Contagem e apresentação indicador novo ponto . . . . .	6
3.1.3 Movimento da bola . . . . .	6
3.1.4 Verificação de jogada, aquando da bola se encontrar em frente ao jogador . . . . .	7
3.1.5 Game Over . . . . .	7
3.1.6 Implementação dos níveis distintos . . . . .	8
3.1.7 Rotina Interrupção . . . . .	9

<b>4</b>	<b>Resposta a questões colocadas no enunciado</b>	<b>11</b>
4.1	Solução adotada para ligar o circuito pTC à placa SDP16 . . . . .	11
4.2	Cálculos para determinar as temporizações envolvidas no trabalho . . .	12
4.3	Latência máxima do sistema no atendimento dos pedidos de interrupção gerados pelo circuito pTC . . . . .	12
4.4	Tempo de execução no pior caso da rotina de atendimento da interrupção externa . . . . .	13
<b>5</b>	<b>Conclusão</b>	<b>15</b>
<b>A</b>	<b>Appendice</b>	<b>i</b>

# Lista de Figuras

2.1	Diagram de Blocos do Sistema . . . . .	4
3.1	Indicador ponto . . . . .	7
3.2	Esquema Game Over . . . . .	8
4.1	Esquema Ligações . . . . .	11



# Lista de Listagens

3.1	Rotina Interrupção . . . . .	9
4.1	Acesso Rotina Interrupção . . . . .	12
A.1	Código Fonte . . . . .	iii





# 1

## Introdução

Este trabalho tem por objetivo consolidar todos os pontos centrais lecionados ao longo da cadeira.

Passando pela programação em *assembly*, o acesso a memória, a comunicação com portos de saída e de entrada, assim como o uso da rotina de interrupções.

O código desenvolvido encontra-se disponível em anexo, assim sendo este relatório parte do pressuposto do acesso por parte do leitor ao código desenvolvido no âmbito do mesmo, não sendo assim necessário enunciá-lo em extensão, bastando apenas mencionar trechos do mesmo.

O código disponível em anexo encontra-se ligeiramente modificado em relação ao entregue, isto por se ter reparado em pequenas falhas, que nesta nova versão se encontram corrigidas.

As falhas principais são as seguintes:

- era efetuada um *branch link* para a rotina *game\_continue*, que corretamente deveria ser apenas um *branch*
- eram efetuadas paragens do *picoTimer*, entre jogos, tal como nos foi chamado à atenção pelo professor, tal não deveria ocorrer
- era mantido o valor do registo *r2* na rotina de interrupção, através de um *push* e um *pop* no final para o *stack*, que era desnecessário, visto não o utilizarmos nesta rotina





## Problema

### 2.1 Squash para Um

O problema proposto é o da realização do jogo "Squash para Um".

O objetivo do jogo é do jogador é manter a bola em movimento entre ele mesmo e a parede.

Na versão que se pretende implementar o jogo é unidimensional, sendo representado graficamente por 6 LEDS, representando cada um uma posição distinta entre o jogador e a parede.

O objetivo do jogador é manter a bola em movimento, durante o jogo, quando a bola se posicionar em frente ao jogador, o mesmo tem de efetuar uma tacada, antes que o tempo dessa jogada acabe.

Existem 2 interruptores, para alternar entre os 4 níveis distintos existentes: velocidade lenta, velocidade média, velocidade rápida, e velocidade variável.

O jogador faz uso de um interruptor para efetuar uma tacada.

O deslocamento da bola é visível pelos leds.

## 2.2 Sistema usado

O hardware usado é a placa de desenvolvimento SDP16, o circuito Pico Timer/Couter (pTC), implementado sobre uma pal e a ATB, tal como demonstrado na figura 2.1.

A placa SDP16, é o processador em si, responsável por armazenar o programa, todas as suas variáveis e por o correr.

A ATB providencia o *clock* para o Pico Timer.

E por fim o *Pico Timer*, é um contador, responsável pela contagem de tempo, um elemento essencial para a implementação do projeto. É através do *Pico Timer* que será feita a gestão de pedidos de interrupção ao processador, consoante as configurações do dispositivo. De acordo com a nossa implementação do projeto, considerámos que as configurações mais adequadas para o resultado que pretendíamos, seria estabelecer um *clock* de 1Khz, juntamente com um limite de contagem de 50. Assim, o dispositivo *Pico Timer* irá realizar um pedido de interrupção a cada 50 *clocks*, o que dado o *clock* 1khz, se traduz em um pedido de interrupção a cada 50 ms, unidade de tempo que utilizamos para realizar a contagem de tempo.

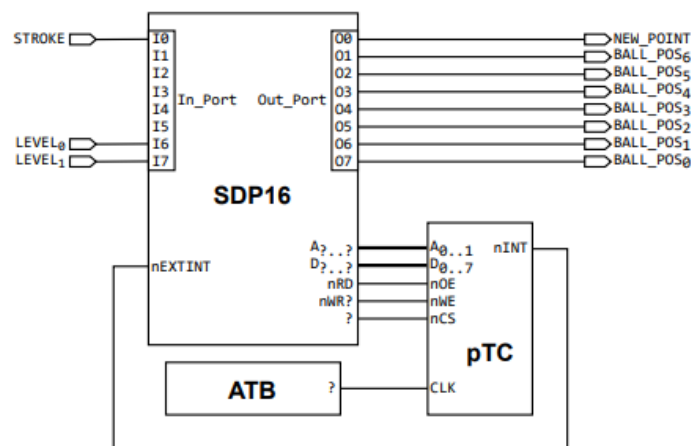


Figura 2.1: Diagram de Blocos do Sistema



## Detalhes Implementação

### 3.1 Lógica programa

A lógica do programa encontra-se descrita num asm chart no final deste documento.

O desenvolvimento deste programa focou se em três grandes pontos:

- Ciclo de jogo
- Contagem e apresentação indicador novo ponto
- Movimento da bola
- Verificação de jogada, aquando da bola se encontrar em frente ao jogador
- Game Over
- Implementação dos níveis distintos
- Rotina de Interrupção

Para além dos pontos previamente referidos existem outros, que consideramos, no entanto, de menor relevo, e assim não serão descritos com o mesmo detalhe que os mencionados.

### 3.1.1 Ciclo de jogo

O Ciclo de jogo baseia-se em 3 verificações. Passou 1 segundo, passou o tempo do nível, se a bola está no player.

- Caso tenha passado 1 segundo aumentamos o *score*, ligamos o led de pontos, atualizamos o a variável do temporizador. Caso o led esteja ligado, e tenham passado 0.25s desligamos.
- Caso o tempo o nível tenha passado, se a bola ainda estiver no jogador perdemos o jogo, caso não, verificamos se está na parede se sim invertemos a direção, e finalmente movemos a bola para a posição seguinte, e atualizamos o temporizador.
- Caso a bola está no jogador verificamos a tacada e caso exista, invertemos a direção, movemos a bola e atualizamos o temporizador do nível.

### 3.1.2 Contagem e apresentação indicador novo ponto

Tal como é visível na imagem seguinte, começamos por verificar através da nossa variável *timer\_1s* se já passou 1 segundo, se tal se tiver sucedido procedemos a incrementar o *score*, ligar o led indicador de *score* e reiniciar essa variável.

Passados 250ms do led estar ligado apagamo-lo.

O valor usado para contabilizar 1 segundo em função do pico *timer* é 20, pois a cada 20 interrupções causadas pelo pico *timer*, obtemos 1 segundo, visto cada interrupção ter a duração de 50ms.

### 3.1.3 Movimento da bola

A rotina *set\_ball\_leds* passa como parâmetro a máscara *BALL\_LEDS\_MASK* que corresponde a 0xFE, pois não afeta o bit de menor peso do porto de saída, visto este estar destinado ao indicador de ponto.

A direção é guardada em memória como uma variável de tipo *byte*, que pode possuir o valor 0 ou 1.

Sendo que o valor 0 representa a direção de jogador para a parede, e o valor 1 a direção oposta, da parede para o jogador.

O *invert\_dir* procede à inversão da direção da bola, com base num xor entre a direção atual com 1, o que vai necessariamente alterar o valor de direção.

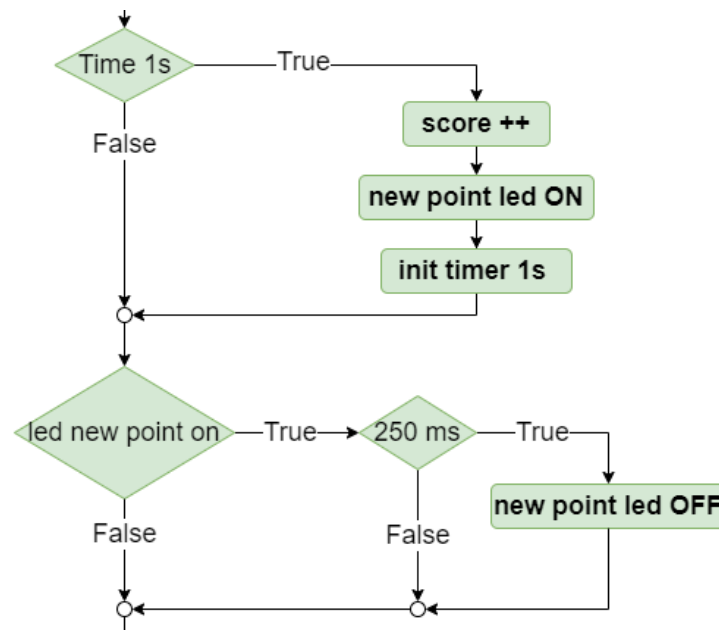


Figura 3.1: Indicador ponto

### 3.1.4 Verificação de jogada, aquando da bola se encontrar em frente ao jogador

Na presente lógica do programa começamos por verificar se o tempo do nível, ou seja, o tempo para a bola se movimentar, já passou.

Se sim procedemos à verificação de se a bola se encontra em frente ao jogador, se este for o caso, o jogo acabou, pois, o jogador não deu a tacada durante o tempo que tinha para o fazer.

Caso contrário verificamos se a bola se encontra na parede e se esse for o caso mudamos de direção.

Independentemente da bola se encontrar ou não na parede movimentamos a bola.

Caso o tempo de nível ainda não tenha terminado, verificamos se a bola se encontra no jogador, e se este for o caso verificamos se houve tacada.

Se esta ocorrer o jogo prossegue, alterando se a bola de direção, para esta passar a ir outra vez em direção à parede, e movimentando a mesma.

### 3.1.5 Game Over

No caso do jogador perder, invertemos a direção da bola, para a preparar para uma nova jogada, apresentamos, apresentamos o score durante 5 segundos e após estes 5 segundos voltamos ao início, onde aguardamos uma nova jogada.

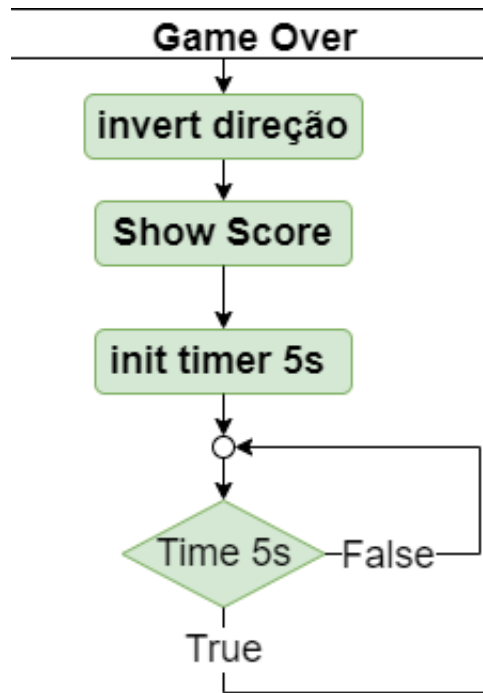


Figura 3.2: Esquema Game Over

### 3.1.6 Implementação dos níveis distintos

Os tempos usados para cada nível são os seguintes:

- Nível 0 Velocidade lenta - 1s (20 ticks)
- Nível 1 Velocidade média - 0,5s (10 ticks)
- Nível 2 Velocidade rápida - 0,25s (5 ticks)
- Nível 3 Velocidade variável - Esta não se encontra implementada, no entanto deixamos preparado de modo a que basta alterarmos uma variável com um valor e o programa funciona.

O jogo implementado permite que antes do início da partida, se escolha o nível de dificuldade pretendido de entre os 4 disponíveis. De modo a permitir a escolha do nível, foram disponibilizados os PINs I6 e I7 do porto de entrada para a seleção.

Visto que cada um dos vários níveis possui o nível de ticks correspondente guardado num array em memória, os valores obtidos nos dois bits do porto entrada são utilizados como index do array e assim é selecionada a velocidade da bola durante a partida.



### 3.1.7 Rotina Interrupção

A rotina de interrupções é a seguinte:

```

1 ; -----
2 ; Rotina:      isr
3 ; Descricao:  Rotina responsavel pelo processamento do pedido de
4               interrupcao.
5 ; Entradas:   -
6 ; Sairas:     -
7 ; Efeitos:    Incrementa o valor da variavel global ticks
8 ; void isr() {
9 ;     ticks++;
10 ;     //clear Interrupt Request
11 ; }
12 isr:
13     ; Prologo
14     push    r0
15     push    r1
16     ; Corpo da rotina
17     ldr     r0, ticks_addr_ext
18     ldr     r1, [r0, #0]
19     add     r1, r1, #1
20     str     r1, [r0, #0]
21     ; clear Interrupt Request
22     mov     r1, 0xFF
23     ldr     r0, ptc_addr
24     strb    r1, [ r0, #pTC_TIR ]
25     ; Epilogo
26     pop     r1
27     pop     r0
28     movs    pc, lr
29
30 timer_clearInterrupt:
31     mov     r0, 0
32     ldr     r1, timer_addr
33     strb    r0, [ r1, #pTC_TIR ]
34     mov     pc, lr

```

Listagem 3.1: Rotina Interrupção

A rotina é relativamente simples, na medida em que cada interrupção aceite se limita a

incrementar uma variável de memória, que conta o número de interrupções ocorridas, e de seguida, limpa o pedido de interrupção executando um comando store em pTC-TIR.

Na presente rotina não salvaguardamos lr, pois o mesmo não é usado dentro da interrupção, visto que não efetuamos branches de qualquer tipo dentro desta rotina. Ou seja, não usamos braches with link e por isso não alteramos o valor de lr, sendo assim não temos necessidade de o salvar.

## Resposta a questões colocadas no enunciado

### 4.1 Solução adotada para ligar o circuito pTC à placa SDP16

A solução adotada para ligar o circuito pTC à placa SDP16 encontra se descrita na seguinte imagem.

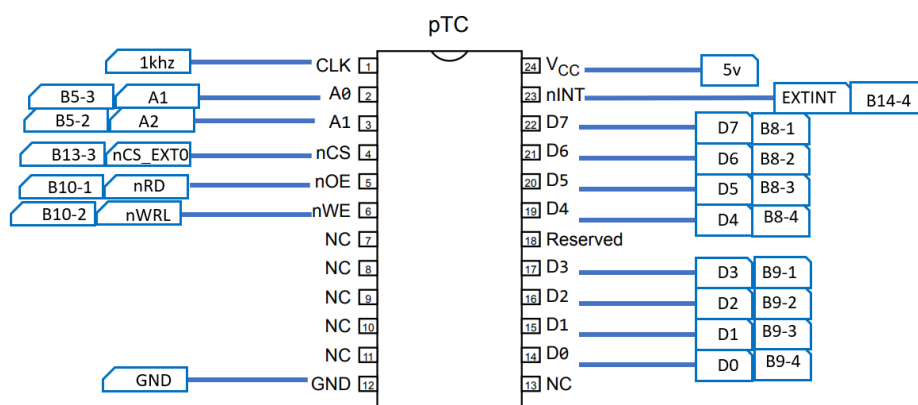


Figura 4.1: Esquema Ligações

## 4.2 Cálculos para determinar as temporizações envolvidas no trabalho

Ocorre uma interrupção a cada 50 milissegundos.

Visto usarmos 50 para configurar o pico timer.

$f_{in\_pTC} = 1kHz$  com o ptc configurado a 50 ficamos com:

$$f_{out\_ptc} = \frac{1kHz}{50} = 20Hz$$

Assim os tempos usados no programa obtêm se a partir do seguinte cálculo:

$$ticks = \frac{tempo}{0,05s}$$

Que resulta nos seguintes ticks para cada tempo

- 5s - 100
- 1s - 20
- 0,5s - 10
- 0,25s - 5

## 4.3 Latência máxima do sistema no atendimento dos pedidos de interrupção gerados pelo circuito pTC

No *worst case scenario*, no caso de nos encontrarmos no início de uma instrução com acesso à memória temos 6 ciclos de *clock*, um ciclo para tratar das *flags*, e colocar o PC a 2 e como no nosso programa temos:

```
1 ldr pc, isr_addr
2 isr_addr:
3     .word  isr
```

Listagem 4.1: Acesso Rotina Interrupção

Devido a não termos a nossa rotina de interrupção logo na linha 2, e em lugar disso fazermos o termos de carregar a mesma de memória, efetuamos mais 6 *clocks* levando a um total de 13 ciclos de *clock* isto com 50khz ficamos com uma latência de 0,26 ms.

$$n\_ciclos\_relogio = 6 + 1 + 6 = 13$$

$$lat\_max = 0,02ms \times 13 = 0,26ms$$

#### 4.4 Tempo de execução no pior caso da rotina da atendimento da interrupção externa

A nossa função de interrupção tem 12 instruções (incluindo o prologo e o epilogo) em que 9 destas acedem a memoria.

Assim o tempo de execução é calculado seguinte modo:

$$n\_ciclos = (3 \times 3) + (9 \times 6) = 63$$

Com 50khz ficamos com um tempo de execução de 1.26 ms.

$$max\_time\_isr = 0,02 \times 63 = 1,26ms$$





## Conclusão

A realização deste trabalho permitiu-nos consolidar os conhecimentos recentemente obtidos relativos a interrupções, assim como a todos os tópicos, já abrangidos nos trabalhos anteriores, pertencentes a esta uc.

Compreendemos assim a utilidade das interrupções como ferramenta onde em algumas situações é vantajoso o seu uso para acesso a dados de um periférico, visto a maior simplicidade do código e a garantia de sincronismo dada pela rotina de interrupção.

No presente programa não desenvolvemos o último nível, com velocidade variável, visto não termos tido tempo para o mesmo.



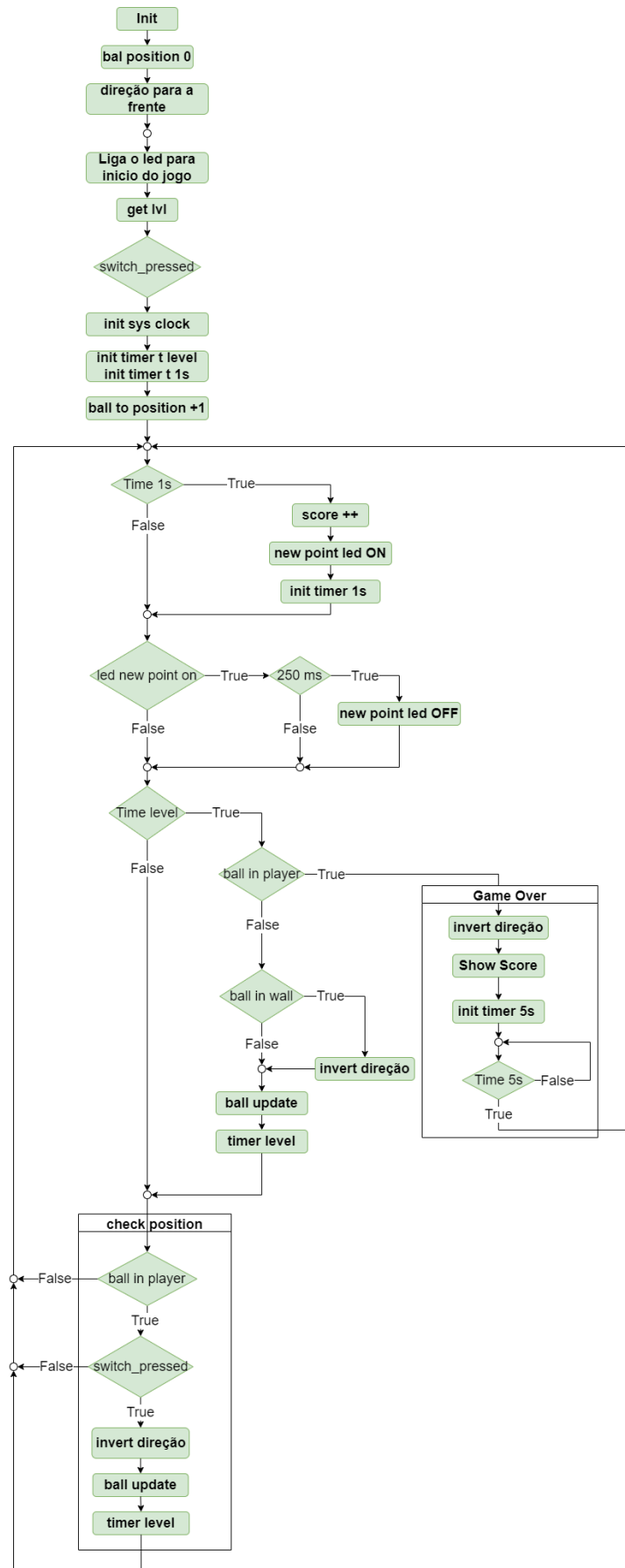




# Appendice

Aqui encontra-se na íntegra o *ASM chart* do programa, e o código na íntegra tal como foi exigido.

O *ASM chart* só se apresenta na página seguinte, de modo a permitir a apresentá-lo na íntegra sem cortes.



```
1
2 ; Autor:      Manuel Fonseca    n: 48052
3 ; Autor:      Manuel Henriques n: 47202
4 ; Autor:      Tiago Pardal     n: 47206
5
6
7
8
9 ; Definicao dos valores dos simbolos utilizados no programa
10 ; valores calculados para o pico timer ligado a 1khz
11
12 .equ    STACK_SIZE, 64          ; Dimensao do stack (em bytes)
13
14 .equ    INPORT_ADDRESS, 0xFF00  ; Endereço do porto de entrada da placa
15      SDP16
16 .equ    OUTPORT_ADDRESS, 0xFF00 ; Endereço do porto de saida da placa
17      SDP16
18
19 .equ    CPSR_BIT_I, 0x10        ; Mascara para a flag I do registo
20      CPSR
21 .equ    PTC_VALUE, 50           ; Intervalo de contagem do circuito pTC
22      ; valores calculados para o pico timer ligado a 1khz
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
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
26
```

```
38 .equ WALL_MASK, 0X02
39 .equ NEW_POINT_LED_MASK, 0X1
40 .equ LEVEL_INPUT_MASK, 0xc0
41 .equ BALL_LEDS_MASK, 0xfe
42 .equ RAKET_MASK, 0x01
43 .equ LVL_MASK, 0xc0
44 .equ VALUE_OF_1S, 20 ;0.05 * 20 = 1s
45 .equ VALUE_OF_5S, 100 ;0.05 * 100 = 5s
46 .equ VALUE_OF_25, 5 ;0.05 * 5 = 2.5s
47
48 .equ VARIANT_LEVEL, 3
49
50 ; Seccao: .startup
51 ; Descricao: Guarda o código de arranque do sistema
52 ;
53 .section .startup
54 b _start
55 ldr pc, isr_addr
56
57 _start:
58 ldr sp, tos_addr
59 bl SYS_init
60 ldr pc, main_addr
61
62 tos_addr:
63 .word tos
64 main_addr:
65 .word main
66 isr_addr:
67 .word isr
68
69 ;-----
70 ;# define OUTPORT_INIT_VALUE 0
71 ;# define LED0_MASK 1
72 ;
73 ;uint16_t ticks = 0;
74 ;
75 ;void main() {
76 ;uint16_t t;
77 ; outport_init ( OUTPORT_INIT_VALUE );
78 ; timer_init ( SYSCLK_FREQ );
79 ; //Habilitar o atendimento das interrupcoes
80 ; while(1) {
```

```
81 ;      outport_set_bits(LED0_MASK);
82 ;      t = sysclk_get_value ();
83 ;      while ( sysclk_elapsed ( t ) < LED_TOGGLE_TIME );
84 ;      outport_clear_bits(LED0_MASK);
85 ;      t = sysclk_get_value ();
86 ;      while ( sysclk_elapsed ( t ) < LED_TOGGLE_TIME );
87 ;  }
88 ;}
89 ;-----
90 .text
91 SYS_init:
92     push lr
93     bl outport_init
94     ;bl timer_stop
95     ldr    r0, ticks_addr
96     ldr    r1, [r0, #0]
97     mov    r1, #0
98     str    r1, [r0, #0]
99     mov r0, PLAYER_MASK ;posição de inicio do jogo
100     ldr r1, ball_pos_addr
101     strb r0, [r1]
102
103     mov r0, PTC_VALUE
104     bl timer_init
105
106     mov r0, IE_MASK
107     msr cpsr, r0
108     pop pc
109
110 ball_pos_addr:
111     .word  ball_pos
112
113 ticks_addr:
114     .word ticks
115
116 main:
117     push lr
118     ;bl timer_stop
119     ldr r0, direction_addr
120     mov r1, 0
121     strb r1, [r0]
122     b main_while
123
```

```
124 direction_addr:
125     .word    direction
126
127 main_while:
128     bl reset_all
129     bl set_ball_leds
130
131 wait_for_init_stroke:
132     mov r0, RAKET_MASK
133     bl sw_is_pressed
134     add r0,r0,0
135     bzc start_game
136     b    wait_for_init_stroke
137
138 start_game:
139     bl set_level_dif
140     bl timer_start
141     bl init_timer_lvl
142     bl init_timer_ls
143     bl mov_ball
144     bl set_ball_leds
145
146 game_loop:
147     mov r0, RAKET_MASK
148     bl sw_is_pressed
149     bl get_timer_ls
150     bl sysclk_elapsed
151     mov r1, VALUE_OF_1S
152     cmp r0, r1 ;20
153     blo one_second_pass_spik
154     bl one_second_pass
155
156 one_second_pass_spik:
157     ldr r0, new_point_led_addr
158     ldrb r0,[r0]
159     sub r0,r0,0
160     bzs time_lvl
161
162     bl get_timer_ls
163     bl sysclk_elapsed
164     mov r1, VALUE_OF_25
165     cmp r0, r1 ;5
166     blo time_lvl
```

```
167     mov r0, 0
168     bl set_led_newpoint
169
170 time_lvl:
171     bl get_timer_lvl
172     bl sysclk_elapsed
173     mov r1, r0
174     bl get_level_dif
175     cmp r1, r0
176     blo await_time_or_player
177
178     bl init_timer_lvl
179     bl get_ball_position
180     mov r2, PLAYER_MASK
181     sub r0, r0, r2
182     bzc next_move_dir
183     b game_over
184
185 ; averigua direção em que a bola se deve movimentar
186 next_move_dir:
187     bl get_ball_position
188     sub r0, r0, WALL_MASK
189     bzc continue_game
190     bl invert_dir
191
192 continue_game:
193     bl mov_ball
194     bl set_ball_leds
195     bl init_timer_lvl
196
197 ; verifica se o jogador já jogou no caso de a bola se encontrar à frente dele
198 await_time_or_player:
199     ;verifica se a bola se encontra no jogador
200     bl get_ball_position
201     mov r2, PLAYER_MASK
202     sub r0, r0, r2
203     bzc game_loop
204
205 ;verifica se botão foi premido
206 ;se o jogador moveu a raquete
207     mov r0, RAKET_MASK
208     bl sw_is_pressed
```

```
209     add r0,r0,0
210     bzs game_loop
211
212     bl invert_dir
213     b continue_game
214     b game_loop
215
216
217 ;apresenta do score no porto de saída durante 5 segundos
218 game_over:
219     bl invert_dir
220     mov r0, 0xff
221     bl outport_clear_bits
222     bl get_score
223     bl outport_set_bits
224     bl init_timer_5s
225     mov r4, VALUE_OF_5S
226
227 game_over_loop:
228     bl get_timer_5s
229     bl sysclk_elapsed
230     cmp r0, r4           ;wait 5s
231     blo game_over_loop
232     ;bl timer_stop
233     mov r0, 0xff
234     bl outport_clear_bits
235     b main_while
236
237 new_point_led_addr:
238     .word new_point_led
239
240
241 ; incrementa score e apresenta indicador de ponto
242 one_second_pass:
243     ;SCORE ++
244     ;LED ON NEW POINT
245     ;INIT TIMER
246     push lr
247     bl init_timer_1s
248     mov r0, 1
249     bl set_led_newpoint
250     bl add_score
251     pop pc
```



```
252
253
254 ; set led new point to the valu of r0
255 set_led_newpoint:
256     push lr
257     ldr r1, new_point_led_addr_ext
258     strb r0, [r1]
259     mov r1, r0
260     mov r0, NEW_POINT_LED_MASK
261     bl  outport_write_bits
262     pop pc
263
264 new_point_led_addr_ext:
265     .word  new_point_led
266
267
268 ;-----
269 ; Funções relacionadas com a bola
270 ;-----
271
272 set_ball_leds:
273     push lr
274     ldr r1, ball_pos_addr_ext1
275     ldrb r1, [r1]
276     mov r0, BALL_LEDS_MASK
277     bl  outport_write_bits
278     pop pc
279
280 invert_dir:
281     push lr
282     ldr r0, direction_addr_ext
283     ldrb r1, [r0]
284     mov r2, 1
285     eor r1, r1, r2
286     strb r1, [r0]
287     pop pc
288
289 get_direction:
290     ldr r0, direction_addr_ext
291     ldrb r0, [r0]
292     mov pc, lr
293
294 get_ball_position:
```

```
295     ldr r0, ball_pos_addr_ext1
296     ldrb r0, [r0]
297     mov pc, lr
298
299 ball_pos_addr_ext1:
300     .word    ball_pos
301
302 direction_addr_ext:
303     .word    direction
304
305 ;-----
306 ; Funções relacionadas com timers
307 ;-----
308
309 init_timer_1s:
310     push lr
311     bl sysclk_get_value
312     ldr r1, timer_1s_addr
313     str r0, [r1]
314     pop pc
315
316 get_timer_1s:
317     ldr r0, timer_1s_addr
318     ldr r0, [r0]
319     mov pc, lr
320
321 timer_1s_addr:
322     .word    timer_1s
323
324
325 init_timer_5s:
326     push lr
327     bl sysclk_get_value
328     ldr r1, timer_5s_addr
329     str r0, [r1]
330     pop pc
331
332 get_timer_5s:
333     ldr r0, timer_5s_addr
334     ldr r0, [r0]
335     mov pc, lr
336
337 timer_5s_addr:
```

```
338     .word    timer_5s
339
340 ;-----
341 ; Funções relacionadas com nivel
342 ;-----
343
344 init_timer_lvl:
345     push lr
346     bl sysclk_get_value
347     ldr r1, timer_level_addr
348     str r0, [r1]
349     pop pc
350
351 get_timer_lvl:
352     ldr r0, timer_level_addr
353     ldr r0, [r0]
354     mov pc, lr
355
356 get_level_dif:
357     ldr r0, current_lvl_addr
358     ldrb r0, [r0]
359     mov pc, lr
360
361 set_level_dif:
362     push lr
363     bl inport_read
364     mov r1, LEVEL_INPUT_MASK
365     and r0, r1, r0
366     lsr r0, r0, #6
367     mov r1, VARIANT_LEVEL
368     cmp r0, r1
369     beq set_level_dif //TODO
370
371     ldr r1, lvl_list_addr
372     ldrb r0, [r1, r0] //lvl_list + input lvl as offset
373     ldr r1, current_lvl_addr
374     strb r0, [r1]
375     pop pc
376
377
378 timer_level_addr:
379     .word timer_level
380
```

```
381 lvl_list_addr:
382     .word lvl_in_time
383
384 current_lvl_addr:
385     .word current_lvl_difficult_in_time
386
387 ;-----
388 ; Funções relacionadas com score
389 ;-----
390
391 get_score:
392     push lr
393     ldr r1, score_addr
394     ldr r0, [r1]
395     pop pc
396
397 add_score:
398     push lr
399     ldr r0, score_addr
400     ldr r1, [r0]
401     add r1, r1, 1
402     str r1, [r0]
403     pop pc
404
405
406 ;-----
407 ; Rotina:    mov_ball
408 ; Descricao: R
409 ; Entradas:  -
410 ; Sidas:    -
411 ; Efeitos:   Move bola em função de direção
412 ;           Na direção do player ou da parede
413 ; void mov_ball() {
414 ;
415 ;}
416 mov_ball:
417     push lr
418     bl get_direction
419     mov r1, r0
420     bl get_ball_position
421     mov r2, 1
422     and r1, r1, r2
423     sub r1, r1, 0
```

```
424     bzs mov_away
425     ;move from wall to player (BALL_POS6)01 -> (BALL_POS0)07
426     lsl r0, r0,1
427     b finish_mov
428 mov_away:
429     lsr r0, r0,1
430
431 finish_mov:
432     ldr r1, ball_pos_addr_ext2
433     strb r0, [r1]
434     pop pc
435
436
437 ball_pos_addr_ext2:
438     .word ball_pos
439
440 ;-----
441 ; Funcao para preparar o inicio de um novo jogo
442 ; Para o contador, limpa o e limpa o score
443 ;-----
444 reset_all:
445     push    lr
446     ;bl timer_stop
447     ;timer sysclk = 0
448     ldr     r1, ticks_addr_ext
449     mov r0, 0
450     str     r0, [r1, #0]
451     ;score = 0
452     ldr     r1, score_addr
453     mov r0, 0
454     str     r0, [r1, #0]
455
456     pop     pc
457
458
459 score_addr:
460     .word   score
461
462 ;-----
463 ; Rotina:    isr
464 ; Descricao: Rotina responsavel pelo processamento do pedido de
465               interrupcao.
466 ; Entradas:  -
```

```
466 ; Saidas:      -
467 ; Efeitos:      Incrementa o valor da variavel global ticks
468 ; void isr() {
469 ;     ticks++;
470 ;     //clear Interrupt Request
471 ;}
472 isr:
473     ; Prologo
474     push    r0
475     push    r1
476     ; Corpo da rotina
477     ldr     r0, ticks_addr_ext
478     ldr     r1, [r0, #0]
479     add     r1, r1, #1
480     str     r1, [r0, #0]
481     ; clear Interrupt Request
482     mov     r1, 0xFF
483     ldr     r0, ptc_addr
484     strb    r1, [ r0, #pTC_TIR ]
485     ; Epilogo
486     pop     r1
487     pop     r0
488     movs    pc, lr
489
490
491 timer_clearInterrupt:
492     mov     r0, 0
493     ldr     r1, timer_addr
494     strb    r0, [ r1, #pTC_TIR ]
495     mov     pc, lr
496
497
498 ;-----
499 ;Funcao para devolver o valor corrente da variável global ticks.
500 ;uint16_t sysclk_get_value ( void );
501 ;     return ticks;
502 ;-----
503 sysclk_get_value:
504     ldr     r1, ticks_addr_ext
505     ldr     r0, [r1, #0]    ; r0 = ticks
506     mov     pc, lr
507
508 ;-----
```

```
509 ;Funcao para devolver o tempo decorrido desde o instante last_read.
510 ;O tempo e medido em unidades de contagem ( ticks ).
511 ;uint8_t sysclk_elapsed ( uint16_t last_read ){
512 ;   return ( ticks - last_read )
513 ;}
514 ;-----
515 sysclk_elapsed:
516     ldr    r1, ticks_addr_ext
517     ldr    r2, [r1, #0]    ; r0 = ticks
518     sub    r0, r2, r0
519     mov    pc,lr
520
521 ticks_addr_ext:
522     .word ticks
523
524 ;-----
525 ;Funcao para iniciar a contagem no periferico.
526 ;void timer_start ( void );
527 ;-----
528 timer_start:
529     mov    r1, #pTC_CMD_START
530     ldr    r0, ptc_addr
531     strb   r1, [ r0, #pTC_TCR ]
532     mov    pc, lr
533
534
535 timer_write:
536     ldr    r2, timer_addr
537     add    r0, r0, r0
538     strb   r1, [r2,r0]
539     mov    pc,lr
540
541 timer_addr:
542     .word  pTC_ADDRESS
543 ;-----
544 ;Funcao para parar a contagem no periferico.
545 ;Colocando o contador com o valor zero.
546 ;void timer_stop ( void );
547 ;-----
548 timer_stop:
549     mov    r1, #pTC_CMD_STOP
550     ldr    r0, ptc_addr
551     strb   r1, [ r0, #pTC_TCR ]
```

```
552     mov pc, lr
553
554     ;-----
555     ;Funcao que faz a iniciacao do periferico para habilitar o
556     ;funcionamento em modo continuo e com intervalo de contagem
557     ;interval, em ticks.
558     ;void timer_init ( uint8_t interval );
559     ;-----
560 timer_init:
561     push lr
562     push r0
563     ; Parar contagem
564     bl timer_stop
565     ; Programar intervalo de contagem
566     pop r0
567     ldr r1, ptc_addr
568     strb r0, [ r1, #pTC_TMR ]
569     ; Clear Interrupt Request
570     ldr r1, ptc_addr
571     strb r0, [ r1, #pTC_TIR ]
572     pop pc
573
574 ptc_addr:
575     .word pTC_ADDRESS
576
577     ;
578     ;-----
579
580 ;uint8_t sw_is_pressed(uint8_t pin_mask) {
581 ;uint8_t sw_new_state;
582 ;    sw_new_state = inport_test_bits( pin_mask );
583 ;    if ( sw_state == sw_new_state )
584 ;        return 0;
585 ;    sw_state = sw_new_state;
586 ;    if ( sw_new_state == 0 )
587 ;        return 0;
588 ;    return 1;
589 ;}
590 ;
591 ;-----
592
593 ; Rotina:    sw_is_pressed
594 ; Descricao:
```



```
591 ; Entradas: pins_mask
592 ; Sidas:   devolve 1 se detecta uma transição 0 -> 1 no pino identificado
           em pin_mask
593 ;         e 0 se não detecta.
594 ; Efeitos:
595 ;
           -----

596 sw_is_pressed:
597     push    lr
598     bl      inport_test_bits
599     ; r0 = sw_new_state = inport_test_bits(pins_mask)
600     ldr     r1, sw_state_address
601     ldrb    r2, [r1, #0]    ; r2 = sw_state
602     cmp     r0, r2          ; sw_state == sw_new_state
603     beq     sw_is_pressed_0
604     strb    r0, [r1, #0]    ; sw_state = sw_new_state;
605     sub     r0, r0, #0
606     beq     sw_is_pressed_0
607     mov     r0, #1
608     b       sw_is_pressed_1
609 sw_is_pressed_0:
610     mov     r0, #0
611 sw_is_pressed_1:
612     pop     pc
613
614 sw_state_address:
615     .word   sw_state
616
617 ;
           -----

618 ;uint16_t inport_test_bits(uint16_t pins_mask) {
619 ;    return ((inport_read() & pins_mask) == pins_mask);
620 ;}
621 ;
           -----

622 ; Rotina:   inport_test_bits
623 ; Descricao: Devolve um se todos dos pinos do porto de entrada
           identificados com o valor um
624 ; em pins_mask tomaremm o valor logico um , ou zero no caso contrario .
625 ; Entradas: Mascara com os bits a testar
```

```
626 ; Saidas:      Devolve um ou zero conforme a descrição.
627 ; Efeitos:
628 ;
-----

629 inport_test_bits:
630     push    lr
631     push    r4
632     mov     r4, r0
633     bl      inport_read
634     and     r0, r0, r4
635     cmp     r0, r4
636     beq     end_inport_test_bit_1
637     mov     r0, #0
638     b       end_inport_test_bit
639 end_inport_test_bit_1:
640     mov     r0, #1
641 end_inport_test_bit:
642     pop     r4
643     pop     pc
644
645 ;
-----

646 ;uint16_t inport_read() {
647 ;     return [INPORT_ADDRESS];
648 ;}
649 ;
-----

650 ; Rotina:      inport_read
651 ; Descricao: Devolve o valor corrente do estado dos pinos do porto de
        entrada.
652 ; Entradas:
653 ; Saidas:      Valor corrente do porto
654 ; Efeitos:
655 ;
-----

656 inport_read:
657     ldr     r0, inport_address_local
658     ldrb    r0, [r0, #0]
659     mov     pc, lr
```

```
660
661 inport_address_local:
662     .word    INPORT_ADDRESS
663 ;
-----
664 ;uint8_t outport_init(uint8_t initial_value) {
665 ;     outport_img = initial_value;
666 ;     outport_write(outport_img);
667 ;}
668 ;
-----
669 ; Rotina:     outport_init
670 ; Descricao: Inicia o porto de saida, atribuindo-lhe o valor do argumento
        passado
671 ;
        a rotina.
672 ; Entradas:  Valor para iniciar o porto de saida
673 ; Saidas:
674 ; Efeitos:   Atualiza o valor da variavel imagem do porto
675 ;
-----
676 outport_init:
677     push    lr
678     mov r0 , #0
679     ldr     r1, outport_img_address
680     strb    r0, [r1, #0]
681     bl      outport_write
682     pop     pc
683
684 ;
-----
685 ;void outport_set_bits(uint8_t pins_mask) {
686 ;     outport_img |= pins_mask;
687 ;     outport_write(outport_img);
688 ;}
689 ;
-----
690 ; Rotina:     outport_set_bits
```

```
691 ; Descricao: Atribui o valor logico '1' aos pinos do porto de saida
        identificados
692 ;           com o valor 1 no argumento passado a rotina. O estado dos
        restantes
693 ;           bits nao e alterado.
694 ; Entradas: Mascara com os bits a alterar
695 ; Saidas:
696 ; Efeitos:  Atualiza o valor da variavel imagem do porto
697 ;
        -----

698 outport_set_bits:
699     push    lr
700     ldr     r1, outport_img_address
701     ldrb    r2, [r1, #0]
702     orr     r0, r2, r0
703     strb    r0, [r1, #0]
704     bl      outport_write
705     pop     pc
706
707 ;
        -----

708 ;void outport_clear_bits(uint8_t pins_mask) {
709 ;    outport_img &= ~pins_mask ;
710 ;    ourport_write(outport_img);
711 ;}
712 ;
        -----

713 ; Rotina:    outport_clear_bits
714 ; Descricao: Atribui o valor logico '0' aos pinos do porto de saida
        identificados
715 ;           com o valor 1 no argumento passado a rotina. O estado dos
        restantes
716 ;           bits nao e alterado.
717 ; Entradas: Mascara com os bits a alterar
718 ; Saidas:
719 ; Efeitos:  Atualiza o valor da variavel imagem do porto
720 ;
        -----

721 outport_clear_bits:
```

```
722     push    lr
723     ldr     r1, outport_img_address
724     ldrb    r2, [r1, #0]
725     mvn     r0, r0
726     and     r0, r2, r0
727     strb    r0, [r1, #0]
728     bl      outport_write
729     pop     pc
730
731 ;
-----
732 ;void outport_write_bits(uint8_t pins_mask, uint8_t value) {
733 ;     value &= pins_mask;
734 ;     outport_img &= ~pins_mask;
735 ;     outport_img |= value;
736 ;     outport_write(outport_img)
737 ;}
738 ;
-----
739 ; Rotina:      outport_write_bits
740 ; Descricao:  Atribui aos pinos do porto de saida identificados com o valor
741 ;             lógico
742 ;             um em pins_mask o valor dos bits correspondentes de value. O
743 ;             estado
744 ;             dos restantes bits não é alterado.
745 ; Entradas:  Mascara com os bits a alterar
746 ;           : valor dos bits a alterar
747 ; Sidas:
748 ; Efeitos:   Atualiza o valor da variável imagem do porto
749 ;
-----
748 outport_write_bits:
749     push    lr
750     and     r1, r0, r1           ; r1 = pins_mask & value
751     ldr     r2, outport_img_address
752     ldrb    r3, [r2, #0]
753     mvn     r0, r0               ; ~pins_mask
754     and     r3, r3, r0           ; outport_img &= ~pins_mask;
755     orr     r0, r3, r1           ; outport_img |= pins_mask & value;
756     strb    r0, [r2, #0]
```

```
757     bl      outport_write
758     pop     pc
759
760 ;
761 -----
762 ;void outport_write(uint8_t value) {
763 ;     outport_img = value;
764 ;     [OUTPORT_ADDRESS] = outport_img;
765 ;}
766 ;
767 -----
768 ; Rotina:      outport_write
769 ; Descricao: Atribui aos pinos do porto de saida o valor dos bits
770 ;              correspondentes de value.
771 ; Entradas:  Valor a escrever no porto
772 ; Saidas:
773 ; Efeitos:   Atualiza o valor da variavel imagem do porto
774 ;
775 -----
776 outport_write:
777     ldr     r1, outport_addr
778     strb    r0, [r1, #0]
779     mov     pc, lr
780
781 outport_img_address:
782     .word   outport_img
783
784 outport_addr:
785     .word   OUTPORT_ADDRESS
786
787 ; Seccao:      .data
788 ; Descricao: Guarda as variáveis globais com valor inicial definido
789 ;
790 .data
791 timer_level:
792     .word   0
793
794 timer_ls:
795     .word   0
796
```

```
793 timer_5s:
794     .word    0
795
796 current_lvl_difficult_in_time:
797     .word    10
798
799 score:
800     .word    0
801
802 ticks:
803     .word    0           ; uint16_t ticks;
804
805 ball_pos:
806     .byte    0x80
807 lvl_in_time:
808     .byte    20, 10, 5      ; 1s / 0.5s / 0.25s
809
810 new_point_led:
811     .byte    0x00
812
813 direction: ; 0 away from player 1 into the player
814     .byte    0x00
815 sw_state:
816     .byte    0
817 ; Seccao:      .bss
818 ; Descricao: Guarda as variáveis globais sem valor inicial definido
819 ;
820     .section .bss
821 outport_img:      ; Imagem do porto de saida no programa
822     .space    1
823
824 ; Seccao:      .stack
825 ; Descricao: Implementa a pilha com o tamanho definido pelo simbolo
826     STACK_SIZE
827 ;
828     .section .stack
829     .space    STACK_SIZE
830 tos:
```

Listagem A.1: Código Fonte

