

Prelab Report: Design and Implementation of a Programmable Digital Clock

Arye Mindell, LMU EECE 3200

Abstract—This report outlines the development of a programmable digital clock using ARM Assembly Language on a Raspberry Pi. The clock operates in "military" time and is updated every second. The program consists of modular subroutines for setting the clock, updating the time, implementing delays, and displaying the output. This project demonstrates the integration of hardware and software in embedded systems, emphasizing modular programming and real-time operation.

I. INTRODUCTION

THIS lab focuses on creating a programmable digital clock on a Raspberry Pi using ARM Assembly Language. The clock allows the user to set the initial time in 'military' format (e.g., "23:59") via a keyboard and continuously updates and displays the time every second on the terminal. The design employs modular subroutines for input, time updating, delay implementation, and output display. This prelab outlines the design and implementation approach, as well as the testing methodology for validating the functionality.

II. SYSTEM DESCRIPTION

THE system is implemented on a Raspberry Pi microcontroller. The key hardware components include:

- Raspberry Pi 4B
- USB keyboard and monitor for input/output

The software consists of four assembly source files:

- `clock.s`: The main routine that calls subroutines for input, updating, and output.
- `setClock.s`: A subroutine to input the initial time (hours and minutes).
- `delay.s`: A subroutine implementing a one-second delay loop.
- `display.s`: A subroutine to display the current time on the terminal.

III. PROGRAM CODE OVERVIEW

THE modular structure of the clock program begins with the main routine, which initializes the clock by invoking the `setClock` subroutine to capture user input. A continuous loop is then used to update the time by incrementing seconds, minutes, and hours as needed. To maintain accurate timing, the `delay` subroutine is employed to create a one-second delay between updates. Finally, the `display` subroutine outputs the current time in the "HH:MM:SS" format on the terminal.

The program is developed and simulated using Geany, GNU Assembler, and GNU Project Debugger.

IV. FLOWCHART

THE flowchart in Fig. 1 outlines the overall structure of the clock program, including initialization, updating, and displaying time.

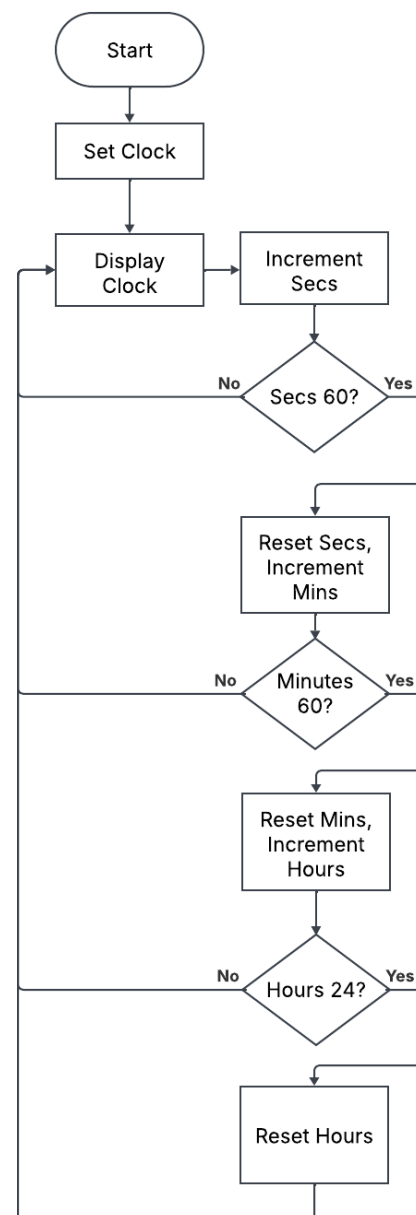


Fig. 1. Flowchart of the clock program algorithm.

V. TEST CASES

TO validate the clock program, the following critical test cases will be simulated:

- Transition from 12:59:59 to 13:00:00.
- Transition from 23:59:59 to 00:00:00.
- Normal operation over several minutes to ensure accuracy.

An additional script, `clockSim.s` simulates each subroutine for testing. Simulation results will be captured using CPUlator.

VI. LIMITATIONS AND CHALLENGES

POTENTIAL limitations in this project include the accuracy of the delay subroutine, which is highly dependent on the timing of instruction cycles. Another challenge is ensuring robust user input validation to accept only valid time formats. Additionally, if the clock loses 10 microseconds per second, it will accumulate a loss of 864 milliseconds per day, resulting in a total loss of 5.26 minutes per year. For the clock to maintain an accuracy of one second per year, it must lose less than 0.032 microseconds per second.

REFERENCES

- [1] Loyola Marymount University, Department of Electrical Engineering and Computer Science, "EECE 3200: Lab 2 - Programmable Digital Clock," Spring 2025.

APPENDIX A ASSEMBLY CODE FILES

```

1
2  @EECE 3200
3  @Arye Mindell
4  @clock main function: runs the main count loop
   of the clock. calls set_clock to initialize
   the clock and display_clock to print the
   clock output
5
6  .global _start
7  .extern set_clock
8  .extern sleep_one_second
9  .extern display_clock
10
11 _start:
12     bl set_clock
13
14 count_loop:
15     CMP r0, #60                @ Check if the
   seconds counter has reached 60
16     BNE check_minutes         @ If not, check
   minutes
17
18     ADD r6, r6, #1             @ Increment
   minutes counter
19     MOV r0, #0                @ Reset the
   seconds counter
20
21 check_minutes:
22     CMP r6, #60                @ Check if the
   minutes counter has reached 60
23     BNE check_hours           @ If not, check
   hours
24
25     ADD r7, r7, #1             @ Increment hours
   counter
26     MOV r6, #0                @ Reset the
   minutes counter

```

```

27
28 check_hours:
29     CMP r7, #24                @ Check if the
   hours counter has reached 24
30     BNE continue_count        @ If not, continue
   counting
31
32     MOV r7, #0                @ Reset the hours
   counter
33
34 continue_count:
35     bl sleep_one_second        @ Pause for 1
   second
36     ADD r0, r0, #1             @ Increment the
   seconds counter
37     bl display_clock           @ Repeat the loop
38     B count_loop

```

Listing 1. `clock.s`

```

1  @EECE 3200
2  @Arye Mindell
3  @setClock subroutine: accepts input from user to
   set the initial values of the clock
4
5  .text
6  .global set_clock
7  set_clock:
8     MOV r0, #0                @ Initialize
   seconds counter to 0
9     MOV r6, #58                @ Initialize
   minutes counter to 58
10    MOV r7, #23                @ Initialize
   hours counter to 23
11    bx lr

```

Listing 2. `setClock.s`

```

1  @EECE 3200
2  @Arye Mindell
3  @sleep_one_second subroutine: waits roughly one
   second based on processor clock rate. must
   be adjusted based on system
4
5  .section .text
6  .global sleep_one_second
7
8  sleep_one_second:
9     PUSH {r4, r5, lr}         @ Save registers and
   return address
10
11    LDR r4, =0x00100000        @ Load 1,000,000 (1
   MHz clock, 1 second delay)
12  loop_outer:
13    SUBS r4, r4, #1            @ Decrement outer
   loop counter
14    BNE loop_outer             @ If not zero, keep
   looping
15
16    POP {r4, r5, lr}           @ Restore registers
   and return address
17    BX lr                      @ Return to caller

```

Listing 3. `delay.s`

```

1  @EECE 3200
2  @Arye Mindell
3  @display_clock subroutine: converts clock values
   to ascii and outputs them to terminal
4
5  .section .text
6  .global display_clock
7
8  display_clock:
9     PUSH {r0, r5, r6, r7, lr} @
   Save the current seconds counter value
10

```

```

11      @ Calculate seconds for the display
12      BL get_led_code          @ Call the
13      subroutine to calculate display value
14      MOV r5, r1              @ Store the
15      seconds display code in r5
16
17      @ Calculate minutes for the display
18      MOV r0, r6              @ Load the minutes
19      counter into r0
20      BL get_led_code          @ Call the
21      subroutine to calculate display value
22      LSL r1, r1, #8           @ Shift the minutes
23      display code left by 16 bits
24      ORR r5, r1, r5          @ Combine minutes
25      and seconds into r1
26
27      @ Calculate hours for the display
28      MOV r0, r7              @ Load the hours
29      counter into r0
30      BL get_led_code          @ Call the
31      subroutine to calculate display value
32      LSL r1, r1, #16          @ Shift the hours
33      display code left by 16 bits
34      ORR r1, r1, r5          @ Combine hours
35      with minutes and seconds in R1
36
37      @ Write the clock value to display
38
39      POP {r0, r5, r6, r7, lr} @
40      Restore the seconds counter value
41      Bx lr
42
43      @ Subroutine to calculate the seven-segment
44      display value
45      get_led_code:
46          PUSH {r4, lr}        @ Save return
47          address
48
49          BL divide_by_10       @ Call the
50          division subroutine
51
52          @ After the call:
53          @ r1 = tens digit (quotient)
54          @ r0 = ones digit (remainder)
55
56          @ Convert the ones digit to ascii
57          MOV r2, r1            @ Move tens
58          digit to r2
59          BL number_to_ascii    @ Call the
60          function to get the segment code
61          MOV r4, r1            @ Store
62          converted ones digit in R4
63
64          @ Convert the tens digit to ascii
65          MOV r0, r2
66          BL number_to_ascii    @ Call the
67          function to get the segment code
68          LSL r1, r1, #4        @ Shift the
69          tens digit value left by 4 bits
70
71          @ Combine tens and ones digits into a single
72          word
73          ORR r1, r1, r4        @ Combine tens
74          (shifted) and ones into r1
75
76          POP {r4, lr}         @ Restore
77          return address
78          BX lr                @ Return to
79          caller
80
81      number_to_ascii:
82          PUSH {lr}            @ Save return
83          address
84          CMP r0, #9            @ Check if the
85          number is greater than 9
86          BHI invalid_input     @ Branch if
87          invalid input
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

```

Listing 4. display.s