```
1   ;   Basil Moledina
2   ;   ELEC 2220, Final Project
3   ;   April 29th, 2022
4
5
6   ;   A program that turns the STM32F11 into a function generator, capable of different periods and
    waveforms.
7
8
9
10
11
12
13
14  ;   Program (main function) Setup, starts at 0x08000000
    ____________________________________________________________________________
15
16
17
18
19  ;   GET TIMER_INITIALIZATION.s
20
21      EXPORT __main
22      EXPORT TIM4_IRQHandler
23      EXPORT EXTI0_IRQHandler
24
25      GET TIMER_INITIALIZATION.s
26      GET EXTI_SETUP.s
27      GET LED_control.s
28
29      AREA PROGRAM25, CODE
30      THUMB
31      ENTRY
32
33
34
35
36
37
38
39  ;   Driver Function here ___________________________________________________________________________
40
41  __main
42
43
44
45  ;------------ GLOBAL VAR INITIALIZATION -----------------
46
47      ldr r0, =TIM4_cnt       ;   timer-interrupt count starts at 0
48      mov r1, #0x00           ;
49      str r1, [r0];           ;
50
51
52      ldr r0, =btn_cnt        ;   button-press count starts at 0
53      mov r1, #0              ;
54      str r1, [r0]            ;
55
56
57
58
59      ldr r0, =DACvalue       ;   waveform starts at 0
60      mov r1, #0x00           ;
61      str r1, [r0]            ;
62
63
64
65      ldr r0, =saw_slope      ;   Saw Tooth Slope = 00
66      mov r1, #0x00           ;
67      str r1, [r0]            ;
68
69
70      ldr r0, =plot_delay     ;   # of timed delays between samples = 0
```

```
71      mov r1, #0x00           ;
72      str r1, [r0]            ;
73
74
75      ldr r0, =current_wave   ;   current wave = sawtooth, 0
76      mov r1, #0x00           ;
77      str r1, [r0]            ;
78
79
80
81      ldr r0, =btn_tmr        ;   button timer = 0
82      mov r1, #0x00           ;
83      str r1, [r0]            ;
84
85
86      ldr r0, =chng_wav       ;   change wave = 0
87      mov r1, #0x00           ;
88      str r1, [r0]            ;
89
90
91      ldr r0, =tri_slope      ;   triangle slope = 0
92      mov r1, #0x00           ;
93      str r1, [r0]            ;
94
95
96      ldr r0, =LED_pattern    ;   LED_pattern = 0000
97      mov r1, #0              ;
98      str r1, [r0]            ;
99
100 ;-------------------------------------------------
101
102
103     bl EXTI_SETUP           ; Set up external interrupts
104     bl SET_LCLK             ; Set up LED GPIO clock
105     bl LED_WMODE            ; Set pinmode of LEDs to write
106     bl SETUP_TIM4           ; set up TIMER
107
108     CPSIE I                 ; Change Processor State / Interrupts Enabled
109
110
111
112
113
114 ; ----------------------STATE_BASED LOOP--------------------------
115
116 main_loop
117
118 main_sawtooth   bl gen_saw;
119 main_triangle   bl gen_tri;
120
121     b main_loop
122
123 ; ------------------------------------------------------------------
124
125
126
127
128
129 here b here                 ;
130
131 ;_____________________________________END OF MAIN_____________________________________________
132
133
134
135
136
137
138
139
140
141
142
```

```
143
144
145
146
147
148 ;   Supporting subroutines _____________________________________________________________________
149
150
151
152
153 ;-----------------------------------------------------------------------
154 ; gen_saw- generate sawtooth - A subroutine that generates a sawtooth waveform
155 ;
156 ;           inputs: none (as far as parameters go)
157 ;           outputs: updates global var waveform
158 ;
159 ;-----------------------------------------------------------------------
160
161
162 gen_saw ldr r0, =TIM4_cnt           ; r0 -> address of TIM4_cnt
163     ldr r1, [r0]                    ; r1 = TIM4_cnt, an interrupt counter
164
165     ldr r0, =plot_delay             ; r0 -> address of plot_delay
166     ldr r2, [r0]                    ; r2 = plot_delay, interrupts between two samples
167
168
169
170 ; -----IF Statement------------------
171     cmp r1, r2                      ;
172     blo saw_IF2                     ;
173 ;-------------------------------------
174
175 ;   If (#_interrupts_occurred >= #_interrupts_between_plots)
176
177
178     ldr r0, =DACvalue               ; r0 -> address of DACvalue
179     ldr r1, [r0]                    ; r1 = DACvalue, current value of the digital waveform
180
181     ldr r0, =saw_slope                  ; r0 -> address of slope
182     ldr r2, [r0]                    ;   r2 = slope, the change in the waveform
183
184     add r1, r2                      ; Increase waveform by slope (DAC + m)
185
186     ldr r0, =DACvalue               ; r0 -> address of DACvalue
187     str r1, [r0]                    ; Write new DACvalue to memory
188
189     ldr r0, =TIM4_cnt               ; r0 -> TIM4_cnt addrs.
190     mov r1, #0                      ; r1 = 0
191     str r1, [r0]                    ; Reset TIM4_cnt ( = 0)
192
193
194
195
196
197
198
199
200
201 ; -----IF Statement------------------
202 saw_IF2 ldr r0, =DACvalue           ;   r0 -> DACvalue Addrs.
203     ldr r1, [r0]                    ;   r1 = DACvalue
204                                     ;
205     mov r0, #0xFFF                  ;   r0 = 0xFFF, 4095 (decimal)
206                                     ;
207                                     ;
208     cmp r1, r0                      ;
209     blo saw_IF3                     ;
210 ;-------------------------------------
211
212 ;   if (current waveform >= MAX_VALUE)
213
214
```

```
215       ldr r0, =DACvalue               ; r0 -> DACvalue addr.
216       mov r1, #0x00                   ; r1 = 0
217       str r1, [r0]                    ; write DACvalue = 0 to memory
218
219
220       ldr r0, =LED_pattern          ; Toggle blue LED
221       ldr r1, [r0]                  ; r1 = LED_pattern
222       EOR r1, #0x08                 ;
223       str r1, [r0]                  ; write back to LED_pattern
224
225       push {r1}                     ; pass LED_pattern as a parameter
226       ldr r0, =lr_copy              ; Store copy of link register
227       str lr, [r0]                  ;
228       bl L_WRITE                    ; subroutine to update LEDs onboard
229       ldr r0, =lr_copy              ;
230       ldr lr, [r0]                  ; Restore link register
231
232
233
234
235   ; if (chng_wav == 1)
236
237    ;-----IF STATEMENT---------------
238   saw_IF3 ldr r0, =chng_wav         ;  r0 -> chng_wav addrs
239       ldr r1, [r0]                  ;  r1 = chng_wav
240       mov r0, #0x01                 ;  r0 = #1
241       cmp r1, r0                    ;  if (chng_wav == 1)
242       bleq wave_setup               ;  branch to wave_setup function
243    ;--------------------------------
244
245
246
247   ;   Otherwise
248
249       b gen_saw                       ; Go back to top, generate another waveform
250
251
252
253   ;---------------------------------------END-------------------------------------------------------
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273   ;-----------------------------------------------------------------------------------------------
274   ;   TIM4_IRQHandler - Timer 4 interrupt Handler
275   ;
276   ;       Purpose: responsible for timing data plots and button-presses in an interval
277   ;
278   ;
279   ;       Inputs:  None
280   ;
281   ;       Outputs:
282   ;                   Modifies TIM4_cnt
283   ;                   Modifies btn_timer (elapsed time between button presses)
284   ;                   Modifies change_wave boolean
285   ;
286   ;
```

```
287 ;       Notes: Tested with break points in debug mode, works
288 ;-------------------------------------------------------------------------------------------
289
290 TIM4_IRQHandler mov r4, lr          ; Reset Timer interrupt-pending bit
291     bl CLR_TIM4_UIF;                ;
292     mov lr, r4;                     ;
293
294
295     ldr r0, =TIM4_cnt               ; r0 -> address of tmr_cnt
296     ldr r1, [r0]                    ; r1 = tmr_cnt
297     add r1, #1                      ; tmr_cnt++
298     str r1, [r0]                    ; write new tmr_cnt back to memory
299
300
301
302 ;    if (btn_cnt > 0), add to btn_timer
303
304 ;--------------IF/ELSE statement--------------------------------
305     ldr r0, =btn_cnt                ; r0 -> btn_cnt addrs
306     ldr r1, [r0]                    ; r1 = btn_cnt
307     cmp r1, #0                      ; if (btn_cnt == 0)
308     beq TIM4_IF2                    ; skip to next IF statement
309 ;-------------------------------------------------------------
310
311 ;           btn_tmr++
312
313     ldr r0, =btn_tmr                ; r0 -> btn_tmr adrs
314     ldr r1, [r0]                    ; r1 = btn_tmr
315     add r1, #1                      ; btn_tmr++
316     str r1, [r0]                    ; write new btn_tmr count back to memory
317
318
319
320
321
322
323
324
325 ;    if (btn_tmr == 2 seconds, ) set chng_wav == 1, reset btn_tmr (reset btn_cnt later)
326
327 ;       in other words:
328 ;
329 ;   if 400 0.005-second-interrupts occur
330
331 ;-------------IF/ELSE statement--------------------------------
332 TIM4_IF2 ldr r0, =btn_tmr           ; r0, -> btn_tmr address
333     ldr r1, [r0]                    ; r1 = btn_tmr
334     cmp r1, #0x190                  ; compare btn_tmr to 400 (in decimal)
335     bxlo lr                         ; Skip below section if btn_tmr < 400 interrupts
336
337 ;-------------------------------------------------------------
338
339 ;   Set change_wave staus to "yes"
340
341
342     ldr r0, =chng_wav               ; r0 -> chng_wav addrs
343     mov r1, #1                      ; r1 = #1
344     str r1, [r0]                    ; write #1 to chng_wav
345
346
347 ;   Reset button_timer
348
349     ldr r0, =btn_tmr                ; r0 -> btn_tmr addrs
350     mov r1, #0                      ; r1 = 0
351     str r1, [r0]                    ; write #0 to btn_tmr (reset it)
352
353
354
355     bx lr                           ;
356
357
358 ; ---------------------------------------END OF INTERRUPT HANDLER-------------------------------
```

```
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 ; wave_setup - Modifies / sets up slope and time between analog waveform plots
386 ;
387 ;   input:
388 ;           reads several global variables
389 ;
390 ;
391 ;   output:
392 ;           modifies several global variables
393 ;
394 ;
395 ; -------------------------------------------------------------------------------
396
397 wave_setup add r0, #0                       ; Delete me
398
399
400
401
402
403     ; reset chng_wav status to "NO"/0
404
405     ldr r0, =chng_wav           ; r0 -> chng_wav
406     mov r1, #0                  ; r1 = 0
407     str r1, [r0]                ; write #0 to chng_wav
408
409
410
411
412
413     ; reset btn_cnt to 0 (r2 contains a copy of btn_cnt)
414
415     ldr r0, =btn_cnt            ; r0 -> btn_cnt addrs
416     ldr r2, [r0]                ; r2 = btn_cnt
417     mov r1, #0                  ; r1 = 0
418     str r1, [r0]                ; write btn_cnt = 0 to memory
419
420
421
422 ; if (btn_cnt >= 2) branch to new waveform (in main)
423
424     cmp r2, #2                  ; compare btn_cnt to #2
425     blo update_graphing         ; skip section if btn_cnt < 2
426
427
428
429
430         ; if( current_wave == sawtooth [0])
```

```
431
432         ldr r0, =current_wave       ; r0 -> current_wave address
433         ldr r1, [r0]                ; r1 = current wave
434
435         cmp r1, #0                  ; compare current wave to #0
436         bne WAVS_ELSE1              ; branch if current wave =/= 0
437
438         mov r1, #1                  ; r1 = 1
439         str r1, [r0]                ; write #1 to current_wave
440         b main_triangle             ; branch to generating triangle
441
442
443
444             ; else if (current_wave == triangle [1])
445
446     WAVS_ELSE1
447
448         mov r1, #0                  ; r1 = 0
449         str r1, [r0]                ; write 0 to current_wave
450         b main_sawtooth             ; branch to generating sawtooth
451
452
453
454
455
456
457
458
459
460     ; otherwise, update sawtooth and triangle's plot_delay and slopes.
461     ;   NOTE: There is only 1 plot delay variable
462
463
464     update_graphing
465
466     ;-------------------------------------------------------------------
467         ; if plot_delay is max (7), reset to 0. Also reset DACvalue to 0
468         ;   otherwise, skip the section below and increase delay. Slopes = 0
469
470         ldr r0, =plot_delay         ; r0 -> plot_delay addrs
471         ldr r1, [r0]                ; r1 = plot_delay
472         cmp r1, #7                  ; if (plot_delay < 7)
473         blo WAVS_ELSE2              ; skip the below section
474     ;-------------------------------------------------------------------
475
476
477
478
479         mov r1, #0                  ; r1 = 0
480         str r1, [r0]                ; Write to plot_delay addrs
481
482
483         ldr r0, =DACvalue           ; r0 -> DACvalue addrs
484         mov r1, #0                  ; r1 = 0
485         str r1, [r0]                ; write 0 to DACvalue
486
487
488         ldr r0, =LED_pattern        ; LED_pattern = 0
489         mov r1, #0                  ;
490         str r1, [r0]                ;
491
492         push {r1}                   ;
493         ldr r0, =lr_copy            ; Save Link Register
494         str lr, [r0]                ;
495         bl L_WRITE                  ; Update LEDs onboard
496         ldr r0, =lr_copy            ;
497         ldr lr, [r0]                ;
498
499         ldr r0, =saw_slope          ;   Saw Tooth Slope = 0 (decimal)
500         mov r1, #0x00               ;
501         str r1, [r0]                ;
502
```

```
503
504         ldr r0, =tri_slope          ;   triangle Slope = 0 (decimal)
505         mov r1, #0x00               ;
506         str r1, [r0]                ;
507
508
509
510
511         bx  lr                      ; return back to wave (which is now off)
512
513
514
515         ; Otherwise
516         ;   Increase data plot delay by 1, reactivate slopes, adjust LED pattern
517
518     WAVS_ELSE2
519
520         ldr r0, =DACvalue           ; r0 -> DACvalue addrs
521         mov r1, #0                  ; r1 = 0
522         str r1, [r0]                ; write 0 to DACvalue
523
524
525
526         ldr r0, =LED_pattern        ; r0 -> LED_pattern addrs
527         ldr r1, [r0]                ; r1 = LED_pattern
528         add r1, #1                  ; LED_pattern++
529         str r1, [r0]                ; write new plot_delay back
530
531
532         push {r1}                   ;
533         ldr r0, =lr_copy            ;
534         str lr, [r0]                ;
535         bl L_WRITE                  ;
536         ldr r0, =lr_copy            ;
537         ldr lr, [r0]                ;
538
539
540         ldr r0, =plot_delay         ; r0 -> plot_delay addrs
541         ldr r1, [r0]                ; r1 = plot_delay
542         add r1, #1                  ; plot_delay++
543         str r1, [r0]                ; write new plot_delay back
544
545
546         ldr r0, =saw_slope      ;   Saw Tooth Slope = 0x29
547         mov r1, #0x29           ;
548         str r1, [r0]            ;
549
550         ldr r0, =tri_slope      ;   triangle Slope = 0x52
551         mov r1, #0x51           ;
552         str r1, [r0]            ;
553
554
555         ; Then branch back to current waveform
556         bx lr;
557
558
559     ;-----------------------------------------END-----------------------------------------
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 ;   EXTI0_IRQHandler - Interrupt handler for interrupt line 0
602 ;
603 ;   inputs - Any inputs required for pressing a button
604 ;   outputs - none
605 ;
606 ;-------------------------------------------------------------------------------------------------------
    -------
607
608 EXTI0_IRQHandler
609
610 ;       Do a quick timed delay to debounce the button
611
612     mov r0, #0x00                               ; r0 = 0
613     mov r1, #0xFFFF                             ; r1 = 0xFFFFF
614
615     lsl r1, #1                                  ;
616     add r1, #0xFF                               ;
617
618
619 btn_dbnc cmp r0, r1                             ; Loop until r0 == r1
620     add r0, #1                                  ; r0++
621     blo btn_dbnc                                ; if (r0 < r1), keep waiting
622
623
624
625 ;       Add to btn_cnt
626
627     ldr r0, =btn_cnt                            ; r0 -> btn_cnt addrs.
628     ldr r1, [r0]                                ; r1 = btn_cnt
629     add r1, #1                                  ; btn_cnt++
630     str r1, [r0]                                ; Write new btn_cnt back to memory
631
632
633
634
635 ;       Reset EXTI pending-bit
636
637     ldr r0, =EXTI                               ; r0 -> points to EXTI memory block
638     ldr r1, [r0, #PR]                           ; r1 = EXTI_PR (pending register)
639     ORR r1, #0x00000001                         ; Reset the pending interrupt request for line 0 (by
    setting it to HIGH)
640     str r1, [r0, #PR]                           ; Write the new pending register code back to the EXTI_PR
641
```

```
642
643 ;       RESET NVIC pending bit
644
645     ldr r0, =NVIC_ICPR0                         ; r0 -> NVIC interrupt clear pending register 0
646     ldr r1, [r0]                                ; r1 = NVIC interrupt clear pending register 0
647     ORR r1, #0x40                               ; Write bit 6 to HIGH
648     str r1, [r0]                                ; Write new NVIC interrupt clear pattern back to memory
649
650
651
652     bx lr                                       ; return back to stopping point
653
654
655
656 ; ------------------------------------END of Subroutine
    ----------------------------------------------------------------------
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 ; gen_tri - generate triangle - A subroutine that generates a triangle waveform
688 ;
689 ;           inputs: none (as far as parameters go)
690 ;           outputs: updates global var waveform
691 ;
692 ;-----------------------------------------------------------------------
693
694
695 gen_tri
696
697 ;   // This segment is for when the triangle is increasing
698
699     ldr r0, =TIM4_cnt           ; r0 -> address of TIM4_cnt
700     ldr r1, [r0]                    ; r1 = TIM4_cnt, an interrupt counter
701
702     ldr r0, =plot_delay             ; r0 -> address of plot_delay
703     ldr r2, [r0]                    ; r2 = plot_delay, interrupts between two samples
704
705
706
707 ; -----IF Statement------------------
708     cmp r1, r2                      ;
709     blo tri_IF2                     ;
710 ;-------------------------------------
711
712 ;   If (#_interrupts_occurred >= #_interrupts_between_plots)
```

```
713
714
715         ldr r0, =DACvalue               ; r0 -> address of DACvalue
716         ldr r1, [r0]                    ; r1 = DACvalue, current value of the digital waveform
717
718         ldr r0, =tri_slope              ; r0 -> address of slope
719         ldr r2, [r0]                    ; r2 = slope, the change in the waveform
720
721         add r1, r2                      ; Increase waveform by slope (DAC + m)
722
723         ldr r0, =DACvalue               ; r0 -> address of DACvalue
724         str r1, [r0]                    ; Write new DACvalue to memory
725
726         ldr r0, =TIM4_cnt               ; r0 -> TIM4_cnt addrs.
727         mov r1, #0                      ; r1 = 0
728         str r1, [r0]                    ; Reset TIM4_cnt ( = 0)
729
730
731
732
733
734
735
736
737
738     ; -----IF Statement------------------
739     tri_IF2 ldr r0, =DACvalue           ;   r0 -> DACvalue Addrs.
740         ldr r1, [r0]                    ;   r1 = DACvalue
741                                         ;
742         mov r0, #0xFFF                  ;   r0 = 0xFFF, 4095 (decimal)
743                                         ;
744                                         ;
745         cmp r1, r0                      ;
746         blo tri_IF3                     ;
747     ;-------------------------------------
748
749     ;   if (current waveform > MAX_VALUE)
750
751
752         b tri_dec                       ; Branch to triangle decrease loop
753
754
755
756
757     ; if (chng_wav == 1)
758
759      ;-----IF STATEMENT---------------
760     tri_IF3 ldr r0, =chng_wav         ;  r0 -> chng_wav addrs
761         ldr r1, [r0]                  ;  r1 = chng_wav
762         mov r0, #0x01                 ;  r0 = #1
763         cmp r1, r0                    ;  if (chng_wav == 1)
764         bleq wave_setup               ;  branch to wave_setup function
765      ;---------------------------------
766
767
768
769     ;   Otherwise
770
771         b gen_tri                       ; Go back to top, generate another waveform
772
773
774
775
776
777
778
779
780
781
782
783
784
```

```
785
786
787   ;   ------------THE DECREASING LOOP-----------------
788
789
790
791   tri_dec
792
793
794
795
796
797       ldr r0, =TIM4_cnt               ; r0 -> address of TIM4_cnt
798       ldr r1, [r0]                    ; r1 = TIM4_cnt, an interrupt counter
799
800       ldr r0, =plot_delay             ; r0 -> address of plot_delay
801       ldr r2, [r0]                    ; r2 = plot_delay, interrupts between two samples
802
803
804   ; -----IF Statement------------------
805       cmp r1, r2                      ;
806       blo saw_IF4                     ;
807   ;-------------------------------------
808
809   ;   If (#_interrupts_occurred >= #_interrupts_between_plots)
810
811
812       ldr r0, =DACvalue               ; r0 -> address of DACvalue
813       ldr r1, [r0]                    ; r1 = DACvalue, current value of the digital waveform
814
815       ldr r0, =tri_slope              ; r0 -> address of slope
816       ldr r2, [r0]                    ;   r2 = slope, the change in the waveform
817
818       sub r1, r2                      ; decrease waveform by slope (DAC - m)
819
820       ldr r0, =DACvalue               ; r0 -> address of DACvalue
821       str r1, [r0]                    ; Write new DACvalue to memory
822
823       ldr r0, =TIM4_cnt               ; r0 -> TIM4_cnt addrs.
824       mov r1, #0                      ; r1 = 0
825       str r1, [r0]                    ; Reset TIM4_cnt ( = 0)
826
827
828
829
830
831
832
833
834
835
836
837
838   ; if (chng_wav == 1)
839
840    ;-----IF STATEMENT---------------
841   saw_IF4 ldr r0, =chng_wav         ;  r0 -> chng_wav addrs
842       ldr r1, [r0]                  ;  r1 = chng_wav
843       mov r0, #0x01                 ;  r0 = #1
844       cmp r1, r0                    ;  if (chng_wav == 1)
845       bleq wave_setup               ;  branch to wave_setup function
846    ;--------------------------------
847
848
849
850
851   ; -----IF Statement------------------
852   saw_IF5 ldr r0, =DACvalue           ;    r0 -> DACvalue Addrs.
853       ldr r1, [r0]                    ;    r1 = DACvalue
854                                       ;
855                                       ;
856       cmp r1, #0x51                   ; compare DACvalue to decreasing slope
```

```
857         bgt tri_dec                     ; Go back to top of dec, generate another waveform                  ;
858     ;-----------------------------------
859
860     ;   if (current waveform < MINIMUM)
861
862         ldr r0, =LED_pattern        ; Toggle blue LED
863         ldr r1, [r0]                ; r1 = LED_pattern
864         EOR r1, #0x08               ;
865         str r1, [r0]                ; write back to LED_pattern
866
867         push {r1}                   ; pass LED_pattern as a parameter
868         ldr r0, =lr_copy            ; Store copy of link register
869         str lr, [r0]                ;
870         bl L_WRITE                  ; subroutine to update LEDs onboard
871         ldr r0, =lr_copy            ;
872         ldr lr, [r0]                ; Restore link register
873
874         b gen_tri                       ; Branch to triangle decrease loop
875
876
877
878
879
880
881
882
883
884
885
886
887
888     ;---------------------------------------END-----------------------------------------------------------
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     ;   Data Area, starting at 0x20000000 ____________________________________________________________
918
919
920         EXPORT TIM4_cnt
921         EXPORT DACvalue
922         EXPORT btn_cnt
923         EXPORT plot_delay
924         EXPORT saw_slope
925         EXPORT LED_pattern
926
927         AREA Data1, DATA
928
```

```
929   TIM4_cnt dcd 0              ; Total interrupts by Timer 4
930                               ;   (since this var's reset to 0)
931
932   btn_cnt dcd 0               ; Button-press counter
933
934
935   plot_delay dcd 0            ; Total interrupts between data plots
936
937   DACvalue dcd 0              ; Instantaneous/current value of the waveform (data)
938   saw_slope dcd 0             ; Linear slope of the waveform
939
940
941
942   current_wave dcd 0          ; current waveform
943                               ; 0 = sawtooth
944                               ; 1 = triangle
945
946
947
948   btn_tmr dcd 0               ; Counts the amount of interrupts after a button press
949                               ;   and then resets and changes waveform after 2 seconds
950
951
952
953
954   chng_wav dcd 0              ; Tells if ready to change the waveform (1 or 0)
955                               ;   Rather than updating waveform solely based off btn_cnt
956
957
958
959
960   tri_slope dcd 0             ; #0x52
961
962
963   LED_pattern dcd 0           ; 4 bit LED pattern
964
965
966   lr_copy dcd 0               ; Holds a copy of the link register
967
968
969   ;_____________________HAS YET TO BE INITIALIZED IN MAIN PROGRAM_____________________
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984       END; End of assembly program.
985
```

Page left intentionally blank to indicate the start of a new program/subroutine library

```
1   ;   Basil Moledina
2   ;   April 10th, 2022
3   ;   Setup file containing subroutines to set up system interrupts
4
5
6   ;   This file is intended to be adjustable as needed
7   ;   If you import this file with get, no need to export any labels
8
9
10
11      AREA INTER_SETUP, CODE
12
13
14
15
16  ;
    ----------------------------------------------------------------------------------------------------
    ---------------------
17  ;   Setup Labels and Constants
18
19
20  SYSCONFIG EQU 0x40013800            ;   Start of SYSCONFIG memory block, 0x4001 3800
21  CR1 EQU 0x08                        ;   Offset to Control register 1 (Lets you set interrupts for pins
    0 - 3)
22                                      ;   From there, load int and adjust bits 0 - 3 (corresponding to
    EXTI0 and which pin 0 will interrupt)
23
24
25
26
27  EXTI EQU 0x40013C00                 ;   Start of EXTI memory block, 0x4001 3C00
28  IMR EQU 0x00                        ;   Offset of interrupt mask register (IMR, AKA "Enable") is 0
29                                      ;   Load int (possibly byte if there's an access violation) and
    change bit 0 to HIGH
30
31
32
33
34  FTSR EQU 0x0C                       ;   Offset to Falling Trigger Selection Register, 0x0C
35                                      ;   Load integer, Set bit 0 to HIGH
36
37
38  PR EQU 0x14                         ;   Pending bit register offset, update bit 0 to
    HIGH.
39
40
41
42
43  NVIC EQU 0xE000E000                 ;   Start of NVIC memory block, 0xE000E100
44  ISER0 EQU 0x100                     ;   Offset to interrupt set-enable register 0
45                                      ;   To set EXTI0, load integer and write bit 6 to HIGH (FROM NVIC
    Table in RM, CH. 10)
46
47
48
49  NVIC_ICPR0 EQU 0XE000E280           ;   Interrupt clear register 0
50                                      ;   load int, write bit 6 to HIGH
51
52
53
54
55  ;
    ----------------------------------------------------------------------------------------------------
    -----------------------
56
57
58
59
60
61
62
63
```

```
 64
 65
 66
 67
 68  ;
     ------------------------------------------------------------------------------------------------------------
     -----
 69  ;   EXTI_SETUP - Sets up interrupt line 0 to be activated by push button
 70  ;
 71  ;   Inputs - None
 72  ;   Outputs - Modifies various EXTI-related registers (see labels and constants)
 73  ;
 74  ;   Note: Does not save registers (advised to run at start of main program)
 75  ;
     ------------------------------------------------------------------------------------------------------------
     ----
 76
 77  EXTI_SETUP ldr r0, =SYSCONFIG               ; r0 points to SYSCONFIG
 78      ldr r1, [r0, #CR1]                      ; r1 = SYSCONFIG_CR1
 79      AND r1, #0xFFFFFFF0                     ; Mask to get 0x#######0 (EXIT0 uses port A)
 80      str r1, [r0, #CR1]                      ; Write new Port-to-Interrupt configuration back
 81
 82
 83
 84      ldr r0, =EXTI                           ; r0 -> points to EXTI memory block
 85      ldr r1, [r0, #IMR]                      ; r1 = EXTI_IMR
 86      ORR r1, #0x00000001                     ; Mask to get 0x#######1
 87      str r1, [r0, #IMR]                      ; write pattern back to interrupt (enable) mask register
 88
 89
 90
 91      ldr r1, [r0, #FTSR]                     ; r1 = EXTI_RTSR
 92      ORR r1, #0x00000001                     ; Set bit 0 of RTSR to HIGH
 93      str r1, [r0, #FTSR]                     ; Write new Rising Trigger code back to RTS register
 94
 95
 96
 97      ldr r0, =NVIC                           ; r0 -> points to NVIC memory block
 98      ldr r1, [r0, #ISER0]                    ; r1 = NVIC_ISER0
 99      ORR r1, #0x00000040                     ; Set bit 6 to HIGH
100      str r1, [r0, #ISER0]                    ; Write new pattern back to interrupt set-enable register
101
102
103
104      bx lr;
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122      END                         ;       End of EXTI0_SETUP file
123
124
```

Page left intentionally blank to indicate the start of a new program/subroutine library

```
;   Basil Moledina
;   LED Control File
;   March 31st, 2022

;   A File that contains subroutines and address labels for setting/Controlling LEDS on the STM32F411VETx board
;   onboard LEDs are at port D

;   The pins for the LEDs are on port D
;   Pins corresponding to LEDs are 12-15









    AREA LED_CONTROL, CODE









;-------------------------------------------------------------------------------------------------------------------
; Constants and address label list


;   ______Clock Setting Addresses_________________:

RCC EQU 0x40023800;     Starting address of Reset & Clock Control Registers
RCC_AHB1ENR EQU 0x30;       OFFSET from RCC, gives you the 32-bit register which enables port D's clock
;                           From here, you can just read the clock register, modify bit 3 (set it to 1), and overwrite back to register




;   _______Pin Mode Register Adresses_____________:


GPIOD EQU 0x40020C00;       Starting address for all registers pertaining to Port D

;       No offset needed to access Port D mode register (offset = 0x00), just LDR GPIOD.
;           See subroutine for more info.



;   ______Addresses for Writing to pins __________:

; Still uses GPIOD address to access Port D's pins
ODR_D EQU 0x14;         OFFSET to access the output data register of port D. See subroutine for more info.




```

```
 69
 70
 71  ;-----------------------------------------------------Label List
     End--------------------------------------------------------------------
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86  ;-----------------------------------------------------------------------------------------------------
     -
 87  ; SET_LCLK  -  (Set LED clock) Activate the clock on PORT D, allowing the on-board LEDs to be modified.
 88  ;
 89  ;   Input:  None
 90  ;
 91  ;   Output: Updates the RCC register to actiavte GPIO port D
 92  ;
 93  ;   Note:   Does not save registers
 94  ;
 95  ;-----------------------------------------------------------------------------------------------------
     -
 96
 97
 98
 99
100  SET_LCLK ldr r0, =RCC;                          ; Load start address of the RCC "partition"
101          ldr r1, [r0, #RCC_AHB1ENR];             ; Read clock register data that controls port d into r1.
102          ORR r1, #0x08;                          ; Use Mask 0x08 ( binary 1000) to set bit 3 (enable
     bit) to 1.
103          str r1, [r0, #RCC_AHB1ENR];             ; Write new data to the clock register
104          bx lr;                                  ; Return to main program
105
106
107
108  ;-----------------------------------------------------------------------------------------------------
     --
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125  ;-----------------------------------------------------------------------------------------------------
     --------
126  ; LED_WMODE - (LED Write Mode) - Sets the mode of the LED pins to "write," (instead of reading data on
     the pins)
127  ;
128  ;   Input:  None
129  ;
130  ;   Output: Updates Mode Register on port D
131  ;
132  ;   Notes: Does not save registers
133  ;
```

```
134 ;-------------------------------------------------------------------------------------------------------------
    --------
135
136
137
138 LED_WMODE   ldr r0, =GPIOD                          ;   Load starting address of Port D data area
139             ldr r1, [r0]                            ;   Read Port D's mode register data (no offset
    needed)
140             bic r1, #0xFF000000                     ;   Clear bits (please note that each pin mode is 2
    bits wide)
141             orr r1, #0x55000000                     ;   Set bits with the following 32 bit mask (
    Binary 0101 0101 ... LSB )
142             str r1, [r0]                            ;   Write new mode data to Port D's mode register
143             bx lr                                   ;   Branch back to main program
144
145
146
147
148 ;---------------------------------------- END
    ----------------------------------------------------------------
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166 ;-------------------------------------------------------------------------------------------------------------
    ------
167 ;  L_WRITE - (LED Write)  Writes an LED pattern to all 4 LEDs, turning them on or off as specified.
168 ;
169 ;       Input:
170 ;               STACK (Push in this order):
171 ;               1) 4-bit code to be written to LEDs (pin 15 -> pin 12)
172 ;
173 ;
174 ;       Output:
175 ;               Updates the output data register of port D
176 ;               LED lights light up or are turned off on the board.
177 ;
178 ;
179 ;       Notes:  For the code, 1 = ON and 0 = OFF. Does not save registers
180 ;
181 ;-------------------------------------------------------------------------------------------------------------
    ------
182
183
184 L_WRITE POP {r0}                        ; r0 = 4-bit LED code
185     LSL r0, #12                         ; Shift the 4-bit code 12 bits to the left.
186     ldr r1, =GPIOD                      ; Load start of port D's data area into r1
187     ldrh r2, [r1, #ODR_D]               ; Read (half-word) Port D's data register
188     bic r2, #0xF000                     ; Clear last four pins (15 - 12)
189     orr r2, r0                          ; Use the shifted parameter as a mask to set bits
190     str r2, [r1, #ODR_D]                ; Write the new output data to the ODR
191     bx lr;                              ; Return to main program or whatever
192
193
194
195 ;------------------------------------ END
    ----------------------------------------------------------------
196
197
```

```
198
199
200
201
202
203
204
205
206
207
208
209 ;-------------------------------------------------------------------------------------------------
    --
210 ;   For Activating and Reading
    Buttons                                                                 |
211 ;
      |
212 ;-------------------------------------------------------------------------------------------------
    --
213
214
215 GPIOA EQU 0x40020000    ; Starting Address of GPIO A data area
216 IDR_A EQU 0x10          ; OFFSET to access the input data register of PORT A.
217
218
219 ;-------------------------------------------------------------------------------------------------
    -
220 ; SET_BCLK  -  (Set button clock) Activate the clock on PORT A, allowing on-board button to be read.
221 ;
222 ;   Input:  None
223 ;
224 ;   Output: Updates the RCC register to actiavte GPIO port A
225 ;
226 ;   Note:   Does not save registers
227 ;
228 ;-------------------------------------------------------------------------------------------------
    -
229
230
231
232
233 SET_BCLK ldr r0, =RCC;                          ; Load start address of the RCC "partition"
234         ldr r1, [r0, #RCC_AHB1ENR];             ; Read clock register data that controls port A into r1.
235         ORR r1, #0x01;                          ; Use Mask 0x01 ( binary 0001) to set bit 0 (enable
    bit) to 1.
236         str r1, [r0, #RCC_AHB1ENR];             ; Write new data to the clock register
237         bx lr;                                  ; Return to main program
238
239
240
241 ;-------------------------------------------------------------------------------------------------
    --
242
243
244
245
246
247
248
249
250
251
252
253 ;-------------------------------------------------------------------------------------------------
    --------
254 ; BTN_RMODE - (Button Read Mode) - Sets the mode of the button pin to "read" (instead of writing data
    on the pin)
255 ;
256 ;   Input:  None
257 ;
258 ;   Output: Updates Mode Register on port A
259 ;
```

```
260 ;   Notes: Does not save registers
261 ;
262 ;-------------------------------------------------------------------------------------------------------
    --------
263
264
265
266 BTN_RMODE   ldr r0, =GPIOA                          ;   Load starting address of Port A data area
267             ldr r1, [r0]                            ;   Read Port A's mode register data (no offset
    needed)
268             bic r1, #0x03                           ;   Clear first 2 bits (please note that each pin
    mode is 2 bits wide)
269             str r1, [r0]                            ;   Write new mode data to Port A's mode register
    (pin mode 00 = Read)
270             bx lr                                   ;   Branch back to main program
271
272
273
274
275 ;---------------------------------------- END
    ----------------------------------------------------------------
276
277
278
279
280
281
282
283
284
285
286 ;-------------------------------------------------------------------------------------------------------
    ------
287 ;  B_READ - ()  Reads data from the button pin.
288 ;
289 ;       Input:
290 ;                   INPUT data register from Port A
291 ;                   Onboard pin 0 associated with GPIO A
292 ;
293 ;       Output:
294 ;                   STACK (pop in this order)
295 ;                   1) on (pressed) or off (not being pressed) status of button
296 ;
297 ;
298 ;       Notes:  For the code, 1 = ON and 0 = OFF. Does not save registers
299 ;
300 ;-------------------------------------------------------------------------------------------------------
    ------
301
302
303 B_READ
304     ldr r1, =GPIOA                      ; Load start of port A's data area into r1
305     ldrh r2, [r1, #IDR_A]               ; Read (half-word) Port A's  INPUT DATA REGISTER
306     AND r2, 0x0001                      ; Use bit-clearing mask (binary 0000 0000 0000 0001) since we
    only care about bit 0
307     PUSH {r2}                           ; Push R2 onto stack to return the button status
308
309     bx lr;                              ; Return to main program or whatever
310
311
312
313 ;------------------------------------ END
    ----------------------------------------------------------------
314
315
316
317
318
319
320
321
322
```

```
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337         END; End of assembly program.
338
```

Page left intentionally blank to indicate the start of a new program/subroutine library

```
1   ; Basil Moledina
2   ; TIMER INITIALIZATION FILE (FINAL PROJECT)
3   ; April 24th, 2022
4
5
6
7   ; A file that includes all subroutines, addresses, and offsets to set up the timer for the final project
8   ; Note: 5 ms is a multiple of every period's time between samples
9
10  ;   Timer Period calculation
11  ;
12  ; Tout = (ARR + 1) x (PSC + 1) x (Tclk)
13  ;
14  ; Tout = (799 + 1) x (99 + 1) x (1 / 16 MHz)
15  ; 5 ms =
16
17
18  ; Process:
19  ;
20  ;   Turn on clock in RCC
21  ;   Set UIE bit (bit 0) in TIMx_DIER        - ENABLE 1
22  ;   Set enable bit in NVIC (NVIC_ISERx)     - ENABLE 2
23  ;   SET CPU enable, CSSIE (done in main program)
24
25  ;   Set counter, prescaler, and autoreload
26
27  ;   Set counter enable (TIMx_CR1)
28
29
30
31
32
33
34
35
36
37      AREA TIMER_INIT, CODE
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52  ;   ------------------------------------------------------------------------------------------------
53  ;   Addresses, Constants, and Offsets
54  ;
55
56  RCC_APB1ENR EQU 0x40023840      ;   RCC Advanced Peripheral Bus 1 Enable register
57                                  ;   Change bit 2 (TIM4 enable)
58
59
60
61  TIM4_DIER EQU 0x4000080C        ;   Timer 4 Interrupt enable register
62                                  ;   Update bit 0
63
64
65  NVIC_ISER0 EQU 0xE000E100       ;   Interrupt set-enable register
66                                  ;   Set bit 30
67
68
69  TIM4_PSC EQU 0x40000828         ;   Timer 4 prescaler register
70                                  ;   16 bits
71
72
```

```
 73  TIM4_CNT EQU 0x40000824          ;   Timer 4 current count (load 16 bits)
 74
 75
 76  TIM4_ARR EQU 0x4000082C          ;   Timer 4 Auto-reload value (16 bits)
 77
 78
 79  TIM4_CR1 EQU 0x40000800          ;   TIMER 4 control register 1
 80                                   ;   Set bit 0 to HIGH
 81
 82
 83
 84  TIM4_SR EQU 0x40000810           ;   TIMER 4 STATUS REGISTER
 85                                   ;   clear bit 0 when interrupt is handled
 86
 87
 88
 89  ;       ---- Constants ---
 90
 91  PRESCALAR EQU 0x63                   ; Prescaler Value (99 in decimal)
 92  AUTORELOAD EQU 0x31F                 ; Auto-Reload Value (799 in decimal)
 93
 94
 95
 96
 97  ; -------------------------------------------------------------------------------------------------------
 98
 99
100
101
102
103
104
105
106
107
108
109
110
111  ; -------------------------------------------------------------------------------------------------------
112  ; SETUP_TIM4  -  Subroutine to set up timer 4 to trigger system interrupts
113  ;
114  ;   Inputs:  NONE (to update the prescale and auto-reload, adjust them above)
115  ;
116  ;   Outputs:
117  ;            No Return value
118  ;            Modifies some registers + clock functions (see addresses + labels)
119  ;
120  ; -------------------------------------------------------------------------------------------------------
121
122
123  SETUP_TIM4 ldr r0, =PRESCALAR                ; r0 = Prescalar
124      ldr r1, =AUTORELOAD                      ; r1 = Autoreload
125
126
127
128      ldr r2, =RCC_APB1ENR                     ; r2 -> points to Reset and Clock Control Register
129                                               ; (Adv. Periph Bus 1 Enable)
130
131      ldr r3, [r2]                             ; r3 = RCC_APB1ENR
132      ORR r3, #0x04                            ; Set bit 2 to HIGH
133      str r3, [r2]                             ; Write new device enable code back to register
134
135
136
137      mov r4, lr                               ; Save link register
138      bl CLR_TIM4_UIF                          ; Clear the interrupt pending bit (starts HIGH for some
     reason)
139      mov lr, r4                               ; Restore link register back
140
141
142
143
```

```
144       ldr r2, =TIM4_DIER                    ; r2 -> Timer 4 interrupt enable register
145       ldr r3, [r2]                          ; r3 = TIM4_DIER
146       ORR r3, #0x01                         ; Set bit 0 to HIGH
147       str r3, [r2]                          ; Write new data back to DIER
148
149
150
151
152       ldr r2, =NVIC_ISER0                   ; r2 -> points to NVIC enable register 0
153       ldr r3, [r2]                          ; r3 = Data in NVIC ISER0
154       ORR r3, #0x40000000                   ; Set bit 30 to HIGH
155       str r3, [r2]                          ; WRITE NVIC interrupt enable code back
156
157
158
159       ldr r2, =TIM4_PSC                     ; r2 -> Timer 4 Prescalar
160       strh r0, [r2]                         ; Store r0 to Timer 4 Prescalar register
161
162
163
164       ldr r2, =TIM4_ARR                     ; r2 -> Timer 4 Auto-Reload Value register
165       strh r1, [r2]                         ; Store r1 to Auto-Reload Register
166
167
168
169
170       ldr r2, =TIM4_CNT                     ; r2 -> Timer 4 current count register
171       mov r3, #0x00                         ; r3 = 0
172       strh r3, [r2]                         ; Store r3 to the current count register (reset to 0)
173
174
175
176
177
178
179       ldr r2, =TIM4_CR1                     ; r2 -> Timer 4 control register
180       ldr r3, [r2]                          ; r3 = Timer 4 CR1 data
181       ORR r3, #0x01                         ; Set bit 0 to HIGH
182       str r3, [r2]                          ; WRITE this data back to CR1 (start the count)
183
184
185
186   ; ---------------------------------------END OF SETUP-----------------------------------------------------
187
188
189
190
191
192
193
194
195   ;-------------------------------------------------------------------------------------------------------
196   ; CLR_TIM4_UIF - Clears the timer 4 interrupt bit (SR, UIF)
197   ;
198   ;   Input - none
199   ;   Output - none
200   ;
201   ;-------------------------------------------------------------------------------------------------------
202
203   CLR_TIM4_UIF    ldr r2, =TIM4_SR          ; r2 -> TImer 4 status register
204       ldrb r3, [r2]                         ; r3 = SR data
205       AND r3, #0xFE                         ; Clear bit 0 to LOW
206       strb r3, [r2]                         ; Write SR data back
207
208       bx lr;
209
210   ;
      -------------------------------------------------------------------------------------------------------
211
212
213
214
```

```
215
216
217
218
219
220
221
222
223
224
225
226         END                                     ; END of timer setup file
```