

Kyle Han, 113799110

ESE280 Laboratory 11

Section 3

Bench 9

Sensors, Basic Analog-to-Digital Conversion, and the 12bit ADC

Design Task 2 Verification Strategy:

Because the pin of PE3 is now an analog input, we can just directly test this on the microcontroller, without pulling up or down the pin. Thus, the pin is floating, and its analog reading will change with time due to the environment. Thus, we only need to step to make sure the logic is correct.

With the potentiometer and the multimeter, we can specify an analog voltage that PE3 should be at. Once maintained, we can step forward to make sure that the ADC correctly digitizes the number (Following the format of a specific fraction of our ref voltage, 2.5V)

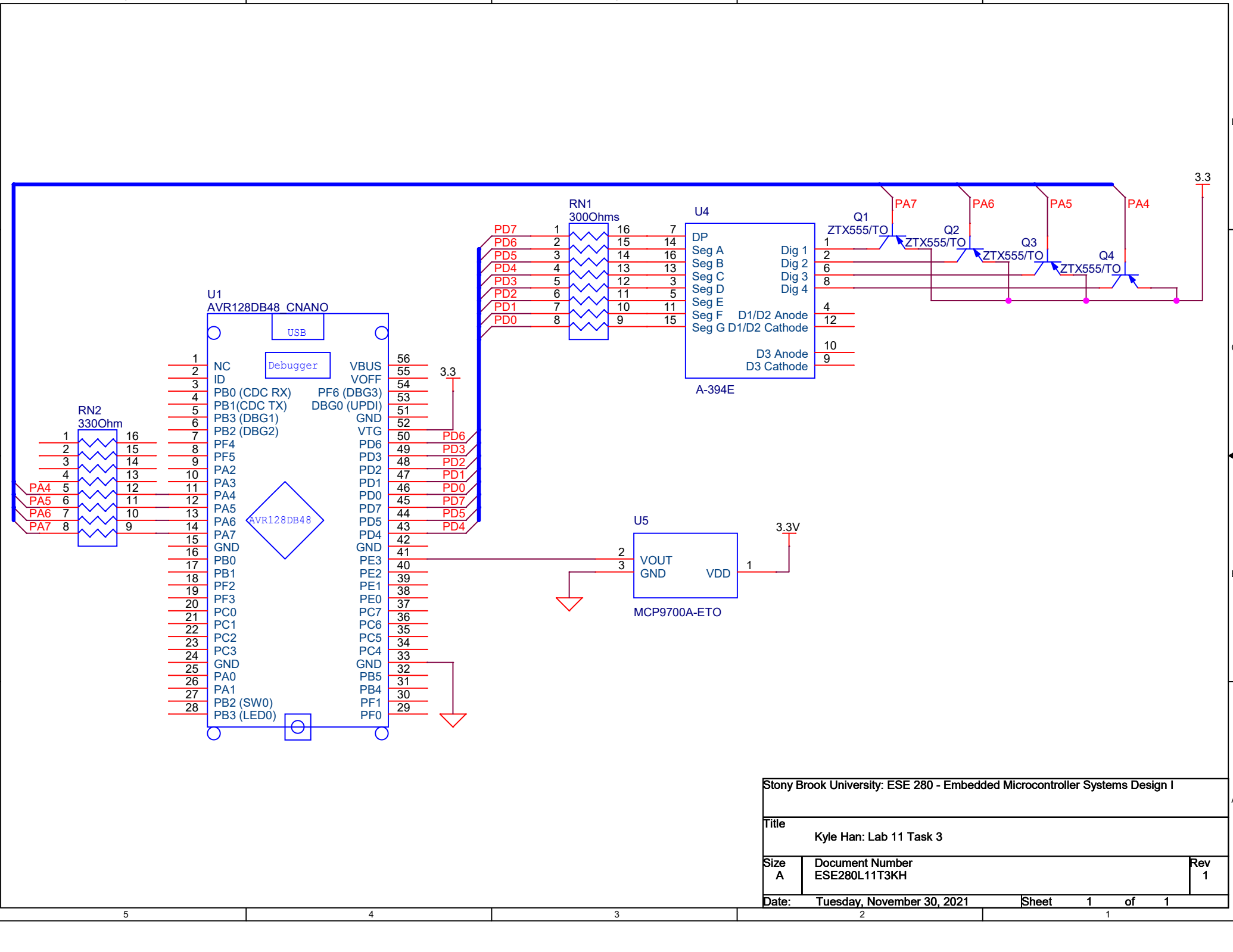
We can check what was analog voltage was read via the debugger. As we step through the program, we can see what was once in the res register of the ADC will now be set in the bcd_entries and led_display arrays in .dseg. Each step of the way, we verify our calculations.

The final result in bcd_entries should be the hand calculated value as hex.

We can repeat this several times to ensure that everything is working correctly.

Design Task 4 Verification Strat:

We can do the same as task 2, hand calculating whatever value that was read from PE3. However, the final result in bcd_entries should be a proper BCD entry that matches the decimal math of the equation $((res * 2500) / 4096) - 500$



```
1 ;
2 ; ADC_sgnl_conv.asm
3 ;
4 ; Created: 11/20/2021 10:40:04 PM
5 ; Author : Kyle Han
6 ;
7
8
9 .equ PERIOD = 97 //((4MHz * (1/160))/256)-1
10
11
12 .dseg
13 bcd_entries: .byte 4
14 led_display: .byte 4
15 digit_num: .byte 1
16
17
18 .cseg
19 reset:
20     rjmp start
21 .org TCA0_OVF_vect
22     rjmp multiplex_display
23
24
25 start:
26
27 //This section deals with port configurations
28 //Set all of PORTD to outputs (For the 7seg)
29     ldi r16, 0xFF
30     sts PORTD_DIR, r16
31
32 //Set PA7-PA4 as outputs (For the multiplexer)
33     ldi r16, 0xF0
34     sts PORTA_DIR, r16
35
36 //Configures all of PORTE as inputs
37     ldi r16, 0x00
38     sts PORTE_DIR, r16
39
40 //This section deals with timer TCA0
41 //Sets up TCA0 timer for the interrupt. 40Hz
42     ldi r16, TCA_SINGLE_WGMODE_NORMAL_gc ;WGMODE normal
43     sts TCA0_SINGLE_CTRLB, r16
44
45     ldi r16, TCA_SINGLE_OVF_bm ;enable overflow interrupt
46     sts TCA0_SINGLE_INTCTRL, r16
47
48 ;load period low byte then high byte
49     ldi r16, LOW(PERIOD) ;set the period
50     sts TCA0_SINGLE_PER, r16
51     ldi r16, HIGH(PERIOD)
52     sts TCA0_SINGLE_PER + 1, r16
```

```

53
54
55 //This section deals with Analog-to-Digital-Converter ADC0
56 //Set the controls as a 12 bit resolution, and enable the ADC
57 ldi r16, ADC_RESSEL_12BIT_gc | ADC_ENABLE_bm
58 sts ADC0_CTRLA, r16
59
60 //Config the voltage ref to 2.5V
61 ldi r16, VREF_REFSEL_2V500_gc
62 sts VREF_ADC0REF, r16
63
64 //Sets the prescaler to 64 bits
65 ldi r16, ADC_PRESC_DIV64_gc
66 sts ADC0_CTRLB, r16
67
68 //Configs PE3 as the positive analog input to the ADC
69 //AIN11 = PE3
70 ldi r16, ADC_MUXPOS_AIN11_gc
71 sts ADC0_MUXPOS, r16
72
73
74 //POST the display. Once posted, then start the timer and continue as normal.
75 rcall post_display
76
77 //set clock and start timer
78 ldi r16, TCA_SINGLE_CLKSEL_DIV256_gc | TCA_SINGLE_ENABLE_bm
79 sts TCA0_SINGLE_CTRLA, r16
80
81 //Start ADC0's conversion
82 ldi r16, ADC_STCONV_bm
83 sts ADC0_COMMAND, r16
84 sei
85
86 loop:
87 rcall pollADC
88 rjmp loop
89
90
91 ;*****
92 ;*
93 ;* "pollADC" - Poll the ADC's result int flag
94 ;*
95 ;* Description: Polls ADC0's INTFLAGS register to check if a result is ready.
96 ;* If it is ready, it will read in the value, and update bcd_entries/led_display ↗
97 ;* accordingly
98 ;* Author: Kyle Han
99 ;* Version: 0.1
100 ;* Last updated: 11/29/2021
101 ;* Target: AVR128DB48
102 ;* Number of words:
103 ;* Number of cycles:
104 ;* Low registers modified: r16-r18

```

```

104 ;* High registers modified:
105 ;*
106 ;* Parameters: None
107 ;*
108 ;* Returns: None
109 ;*
110 ;* Notes: Requires digit_num, bcd_entries, and led_display to be declared
      in .dseg
111 ;*
112 ;*****
113 pollADC:
114     //Saving things to the stack
115     push r16
116     push r17
117     push r18
118     push r19
119
120     //Check the flag. If 1, then continue. If 0, then remove the things from the
      stack and return.
121     lds r16, ADC0_INTFLAGS
122     sbrc r16, 0
123     rjmp retADC
124
125     //Past this point, we know that a conversion is ready for us to read in.
126     lds r19, ADC0_RESL
127     lds r17, ADC0_RESH
128
129     //Will copy everything needed into bcd_entries, and create the corresponding
      values in led_display
130     //Entry 0
131     mov r18, r17
132     andi r18, 0xF0
133     //We need a shift right to get this back to bcd (Since that only requires 4
      bits)
134     lsr r18
135     lsr r18
136     lsr r18
137     lsr r18
138     sts bcd_entries, r18
139     rcall hex_to_7seg
140     sts led_display, r18
141
142     //Entry 1
143     mov r18, r17
144     andi r18, 0x0F
145     sts bcd_entries+1, r18
146     rcall hex_to_7seg
147     sts led_display+1, r18
148
149     //Entry 2
150     mov r18, r19
151     andi r18, 0xF0

```

```

152     lsr r18
153     lsr r18
154     lsr r18
155     lsr r18
156     sts bcd_entries+2, r18
157     rcall hex_to_7seg
158     sts led_display+2, r18
159
160     //Entry 3
161     mov r18, r19
162     andi r18, 0x0F
163     sts bcd_entries+3, r18
164     rcall hex_to_7seg
165     sts led_display+3, r18
166
167     //Returns from the ADC subroutine. Saves things and make sure the registers are
        left unmodified.
168 retADC:
169     pop r19
170     pop r18
171     pop r17
172     pop r16
173
174     //Restart ADC0's conversion
175     ldi r16, ADC_STCONV_bm
176     sts ADC0_COMMAND, r16
177     ret
178
179 ;*****
180 ;*
181 ;* "hex_to_7seg" - Hexadecimal to Seven Segment Conversion
182 ;*
183 ;* Description: Converts a right justified hexadecimal digit to the seven
184 ;* segment pattern required to display it. Pattern is right justified a
185 ;* through g. Pattern uses 0s to turn segments on ON.
186 ;*
187 ;* Author:          Ken Short
188 ;* Version:         0.1
189 ;* Last updated:    101221
190 ;* Target:          AVR128DB48
191 ;* Number of words:
192 ;* Number of cycles:
193 ;* Low registers modified:
194 ;* High registers modified:
195 ;*
196 ;* Parameters: r18: hex digit to be converted
197 ;* Returns: r18: seven segment pattern. 0 turns segment ON
198 ;*
199 ;* Notes:
200 ;*
201 ;*****
202 hex_to_7seg:

```

```

203     push ZH
204     push ZL
205     ldi ZH, HIGH(hextable * 2) ;set Z to point to start of table
206     ldi ZL, LOW(hextable * 2)
207     ldi r16, $00                ;add offset to Z pointer
208     andi r18, 0x0F              ;mask for low nibble
209     add ZL, r18
210     adc ZH, r16
211     lpm r18, Z                  ;load byte from table pointed to by Z
212     pop ZL
213     pop ZH
214     ret
215
216     ;Table of segment values to display digits 0 - F
217     ;!!! seven values must be added
218     hextable: .db $01, $4F, $12, $06, $4C, $24, $20, $0F, $00, $04, $08, $60, $31,  ↗
                $42, $30, $38
219
220
221     ;*****
222     ;*
223     ;* Multiplex_display
224     ;*
225     ;* Description: reads everything in the memory "array" and converts it to a 7seg  ↗
                interpetatoin.
226     ;*
227     ;*
228     ;*
229     ;* Author:          Kyle Han
230     ;* Version:         0.1
231     ;* Last updated:    11032021
232     ;* Target:         AVR128DB48
233     ;* Number of words:
234     ;* Number of cycles:
235     ;* Low registers modified:
236     ;* High registers modified:
237     ;*
238     ;* Parameters: A pointer called array, and pointer X available
239     ;* Returns: Everything inside pointer X
240     ;*
241     ;* Notes:
242     ;*
243     ;*****
244
245
246     multiplex_display:
247
248     //Turns off the whole display by outputting 1s to PORTA. Then, check with  ↗
                digit_num to see which display should be lit
249     turn_off:
250
251     //First, we must push all the registers we are using to the stack. This is so  ↗

```



```

        that the original values are restored later on
252     push r16
253     push r17
254     push r18
255     in r16, CPU_SREG
256     push r16
257     push XL
258     push XH
259
260     //Handles the interrupt and resets the timer
261     ldi r16, TCA_SINGLE_OVF_bm ;clear OVF flag
262     sts TCA0_SINGLE_INTFLAGS, r16
263
264     ldi r16, 0xFF
265     sts PORTA_OUT, r16
266     lds r17, digit_num
267     inc r17
268     //If we are at position 4, return to position 0
269     cpi r17, 0x04
270     brsh overflow
271
272 output:
273     sts digit_num, r17
274     ldi XH, HIGH(led_display)
275     ldi XL, LOW(led_display)
276     add XL, r17
277     ld r18, X
278     //rcall hex_to_7seg
279     sts PORTD_OUT, r18
280
281 //Will check digit_num to decide which display on the 7seg is ebing outputted
282 checking_dig:
283     cpi r17, 0x00
284     breq dig0
285     cpi r17, 0x01
286     breq dig1
287     cpi r17, 0x02
288     breq dig2
289     cpi r17, 0x03
290     breq dig3
291
292     //r18 stores the value of the digit to be displayed
293 dig0:
294     ldi r18, 0x70 //PA7
295     sts PORTA_OUT, r18
296     rjmp restore
297
298 dig1:
299     ldi r18, 0xB0 //PA6
300     sts PORTA_OUT, r18
301     rjmp restore
302

```

```

303 dig2:
304     ldi r18, 0xD0    //PA5
305     sts PORTA_OUT, r18
306     rjmp restore
307
308 dig3:
309     ldi r18, 0xE0    //PA4
310     sts PORTA_OUT, r18
311     rjmp restore
312
313 //Reads off the stack the values that we preserved inside turn_off
314 restore:
315     pop XH
316     pop XL
317     pop r16
318     out CPU_SREG, r16
319     pop r18
320     pop r17
321     pop r16
322
323     reti
324
325 //Sets digit_num back to 0, since we only have 4 digits
326 overflow:
327     ldi r17, 0x00
328     sts digit_num, r17
329     ldi XH, HIGH(led_display)
330     ldi XL, LOW(led_display)
331     rjmp output
332
333
334 ;*****
335 ;*
336 ;* "post_display" - Power On Self Test
337 ;*
338 ;* Description: Will POST the 7segment display hooked up to port D, multiplexed
339 ;* by PA7-PA4.
340 ;* Individually turns on each segment for a brief moment, totalling 1 second
341 ;* Author: Kyle Han
342 ;* Version: 0.1
343 ;* Last updated: 11/29/2021
344 ;* Target: AVR128DB48
345 ;* Number of words:
346 ;* Number of cycles:
347 ;* Low registers modified: r16-r20
348 ;* High registers modified:
349 ;*
350 ;* Parameters: None
351 ;*
352 ;* Returns: None
353 ;*
354 ;* Notes: Requires digit_num to be declared in .dseg

```

```

354 ;*
355 ;*****
356 post_display:
357     //A value of 0 turns on our LED displays
358     ldi r16, 0x00
359     out VPORTD_OUT, r16
360     sts digit_num, r16 ;Store 0x00 to the current digit to be displayed
361 repeatPost:
362     in r17, VPORTA_OUT
363     ori r17, 0xF0
364     out VPORTA_OUT, r17
365     ldi r19, 0b00010000 ;This 1 will turn on a specific digit.
366 leftShiftPOST:
367     cpi r16, 0
368     breq postDispOn
369     lsl r19
370     dec r16
371     rjmp leftShiftPOST
372 postDispOn:
373     in r17, VPORTA_OUT
374     eor r17, r19
375     out VPORTA_OUT, r17
376     ldi r20, 10
377     rcall oneSecDelay
378     //Compares digit_num to 0x04. If greater than or equal to, we've finished posting
379     lds r16, digit_num
380     inc r16
381     sts digit_num, r16
382     cpi r16, 0x04
383     brlo repeatPost
384
385     in r17, VPORTA_OUT
386     ori r17, 0xF0
387     out VPORTA_OUT, r17
388     ret
389
390 ;*****
391 ;*
392 ;* "oneSecDelay" - Delay the microcontroller by 1 second by occupying CPU time
393 ;*
394 ;* Description: Loops through and occupies CPU time to delay by 1 second
395 ;*
396 ;* Author: Kyle Han
397 ;* Version: 1.0
398 ;* Last updated: 11/29/2021
399 ;* Target: AVR128DB48
400 ;* Number of words:
401 ;* Number of cycles:
402 ;* Low registers modified:r17
403 ;* High registers modified:
404 ;*

```

```
405  ;* Parameters: r20 * .1ms
406  ;*
407  ;* Returns: Nothing
408  ;*
409  ;* Notes:
410  ;*
411  ;*****
412  oneSecDelay:
413  outer_loop:
414      ldi r17, 133
415  inner_loop:
416      dec r17
417      brne inner_loop
418      dec r20
419      brne outer_loop
420      ret
```

```
1 ;
2 ; temp_meas.asm
3 ;
4 ; Created: 11/30/2021 5:02:21 AM
5 ; Author : Kyle Han
6 ;
7
8
9 .equ PERIOD = 97 //((4MHz * (1/160))/256)-1
10
11
12 .dseg
13 bcd_entries: .byte 4
14 led_display: .byte 4
15 digit_num: .byte 1
16
17
18 .cseg
19 reset:
20     rjmp start
21 .org TCA0_OVF_vect
22     rjmp multiplex_display
23 .org ADC0_RESRDY_vect
24     rjmp ADC0_response
25
26 start:
27
28 //This section deals with port configurations
29 //Set all of PORTD to outputs (For the 7seg)
30     ldi r16, 0xFF
31     sts PORTD_DIR, r16
32
33 //Set PA7-PA4 as outputs (For the multiplexer)
34     ldi r16, 0xF0
35     sts PORTA_DIR, r16
36
37 //Configures all of PORTE as inputs
38     ldi r16, 0x00
39     sts PORTE_DIR, r16
40
41 //This section deals with timer TCA0
42 //Sets up TCA0 timer for the interrupt. 40Hz
43     ldi r16, TCA_SINGLE_WGMODE_NORMAL_gc ;WGMODE normal
44     sts TCA0_SINGLE_CTRLB, r16
45
46     ldi r16, TCA_SINGLE_OVF_bm ;enable overflow interrupt
47     sts TCA0_SINGLE_INTCTRL, r16
48
49 ;load period low byte then high byte
50     ldi r16, LOW(PERIOD) ;set the period
51     sts TCA0_SINGLE_PER, r16
52     ldi r16, HIGH(PERIOD)
```

```
53     sts TCA0_SINGLE_PER + 1, r16
54
55
56 //This section deals with Analog-to-Digital-Converter ADC0
57 //Set the controls as a 12 bit resolution, and enable the ADC
58     ldi r16, ADC_RESSEL_12BIT_gc | ADC_ENABLE_bm
59     sts ADC0_CTRLA, r16
60
61     //Config the voltage ref to 2.5V
62     ldi r16, VREF_REFSEL_2V500_gc
63     sts VREF_ADC0REF, r16
64
65     //Sets the prescaler to 64 bits
66     ldi r16, ADC_PRESC_DIV64_gc
67     sts ADC0_CTRLB, r16
68
69     //Configs PE3 as the positive analog input to the ADC
70     //AIN11 = PE3
71     ldi r16, ADC_MUXPOS_AIN11_gc
72     sts ADC0_MUXPOS, r16
73
74     //Configures the interrupt for the ADC
75     ldi r16, ADC_RESRDY_bm
76     sts ADC0_INTCTRL, r16
77
78     //POST the display. Once posted, then start the timer and continue as normal.
79     rcall post_display
80
81 //Starting line
82 //set clock and start timer
83     ldi r16, TCA_SINGLE_CLKSEL_DIV256_gc | TCA_SINGLE_ENABLE_bm
84     sts TCA0_SINGLE_CTRLA, r16
85
86     //Start ADC0's conversion
87     ldi r16, ADC_STCONV_bm
88     sts ADC0_COMMAND, r16
89     sei
90
91 loop:
92     nop
93     rjmp loop
94
95
96 ;*****
97 ;*
98 ;* "ADC0_response" - Read the result of ADC0
99 ;*
100 ;* Description: Read in the value, and update bcd_entries/led_display accordingly
101 ;*
102 ;* Author: Kyle Han
103 ;* Version: 0.1
104 ;* Last updated: 11/29/2021
```

```

105 ;* Target: AVR128DB48
106 ;* Number of words:
107 ;* Number of cycles:
108 ;* Low registers modified: r16-r18
109 ;* High registers modified:
110 ;*
111 ;* Parameters: None
112 ;*
113 ;* Returns: None
114 ;*
115 ;* Notes: Requires digit_num, bcd_entries, and led_display to be declared
      in .dseg
116 ;*
117 ;*****
118 ADC0_response:
119     //Saving things to the stack
120     push r16
121     push r17
122     push r18
123     push r19
124     push r20
125     push r21
126     push r22
127
128     //Past this point, we know that a conversion is ready for us to read in.
129     lds r16, ADC0_RES_L
130     lds r17, ADC0_RES_H
131
132     //We will now compute the formula to transform the result into our degrees
      celsius
133     //Eqn: T(0.1C) = ((RES x 2500)/4096)-500
134
135     //Load s the multipleier (2500)(1001 1100 0100) into r18 and r19
136     ldi r18, 0b11000100
137     ldi r19, 0b00001001
138
139     rcall mpy16u
140
141     //Result now stored in r18-r21. Divide by 4096 (2^12) (Anotherwards, ignore
      the 3 least signifigant hexs)
142     //Ignore 18, and the latter half of 19
143     //Result of this is stored (High) r20, (low) r19
144     ldi r22, 4 //Do this shift 4 times
145 Div4096:
146     clc
147     ror r20
148     ror r19
149     dec r22
150     brne Div4096
151     //This shift has been done 4 times. We've now divided by 4096
152
153     //We need to check if the number is smaller than 500 first.

```

```
154 //Larger: Res-500
155 //Smaller: 500-Res, set T bit
156 //if it is larger, we continue as normal
157 //Output is stored in r17 (HIGH), r16 (LOW)
158 ldi r16, LOW(500)
159 ldi r17, HIGH(500)
160 cp r19, r16
161 cpc r20, r17
162 brlo negative
163 sub r19, r16
164 sbc r20, r17
165 mov r16, r19
166 mov r17, r20
167 rjmp continueUpdate
168
169 negative:
170     set
171     sub r16, r19
172     sbc r17, r20
173
174
175 continueUpdate:
176     rcall bin2BCD16
177
178
179
180
181 //Will copy everything needed into bcd_entries, and create the corresponding values in led_display
182 //Entry 0
183 lds r18, tBCD0
184 andi r18, 0x0F
185 sts bcd_entries, r18
186 rcall hex_to_7seg
187 sts led_display, r18
188
189 //Entry 1
190 lds r18, tBCD0
191 andi r18, 0xF0
192 //We need a shift right to get this back to bcd (Since that only requires 4 bits)
193 lsr r18
194 lsr r18
195 lsr r18
196 lsr r18
197 sts bcd_entries+1, r18
198 rcall hex_to_7seg
199 sts led_display+1, r18
200
201 //Entry 2
202 lds r18, tBCD1
203 andi r18, 0x0F
```



```

204     sts bcd_entries+2, r18
205     rcall hex_to_7seg
206     sts led_display+2, r18
207
208     //Entry 3 is special. If the T bit is set, then we know we must display a -  ↗
209     //      (Negative).
210     //Otherwise, display whatever should be displayed.
211     //Entry 3
212     brrc positiveEntry3
213
214 negativeEntry3:
215     ldi r18, 0xFE //For the negative sign
216     sts led_display+3, r18
217     clt
218     rjmp retADC
219
220 positiveEntry3:
221     lds r18, tBCD1
222     andi r18, 0xF0
223     lsr r18
224     lsr r18
225     lsr r18
226     lsr r18
227     sts bcd_entries+3, r18
228     rcall hex_to_7seg
229     sts led_display+3, r18
230
231 //Returns from the ADC subroutine. Saves things and make sure the registers are  ↗
232 //      left unmodified.
233 retADC:
234     pop r22
235     pop r21
236     pop r20
237     pop r19
238     pop r18
239     pop r17
240     pop r16
241
242 //Restart ADC0's conversion
243     ldi r16, ADC_STCONV_bm
244     sts ADC0_COMMAND, r16
245     reti
246
247 ;*****
248 ;*
249 ;* "hex_to_7seg" - Hexadecimal to Seven Segment Conversion
250 ;*
251 ;* Description: Converts a right justified hexadecimal digit to the seven
252 ;* segment pattern required to display it. Pattern is right justified a
253 ;* through g. Pattern uses 0s to turn segments on ON.
254 ;*

```

```

254 ;* Author:          Ken Short
255 ;* Version:         0.1
256 ;* Last updated:    101221
257 ;* Target:          AVR128DB48
258 ;* Number of words:
259 ;* Number of cycles:
260 ;* Low registers modified:
261 ;* High registers modified:
262 ;*
263 ;* Parameters: r18: hex digit to be converted
264 ;* Returns: r18: seven segment pattern. 0 turns segment ON
265 ;*
266 ;* Notes:
267 ;*
268 ;*****
269 hex_to_7seg:
270     push ZH
271     push ZL
272     ldi ZH, HIGH(hextable * 2) ;set Z to point to start of table
273     ldi ZL, LOW(hextable * 2)
274     ldi r16, $00                ;add offset to Z pointer
275     andi r18, 0x0F              ;mask for low nibble
276     add ZL, r18
277     adc ZH, r16
278     lpm r18, Z                  ;load byte from table pointed to by Z
279     pop ZL
280     pop ZH
281     ret
282
283     ;Table of segment values to display digits 0 - F
284     ;!!! seven values must be added
285 hextable: .db $01, $4F, $12, $06, $4C, $24, $20, $0F, $00, $04, $08, $60, $31, ➤
           $42, $30, $38
286
287
288 ;*****
289 ;*
290 ;* Multiplex_display
291 ;*
292 ;* Description: reads everything in the memory "array" and converts it to a 7seg ➤
           interpetatoin.
293 ;*
294 ;*
295 ;*
296 ;* Author:          Kyle Han
297 ;* Version:         0.1
298 ;* Last updated:    11032021
299 ;* Target:          AVR128DB48
300 ;* Number of words:
301 ;* Number of cycles:
302 ;* Low registers modified:
303 ;* High registers modified:

```

```

304  ;*
305  ;* Parameters: A pointer called array, and pointer X available
306  ;* Returns: Everything inside pointer X
307  ;*
308  ;* Notes:
309  ;*
310  ;*****
311
312
313  multiplex_display:
314
315  //Turns off the whole display by outputting 1s to PORTA. Then, check with
    digit_num to see which display should be lit
316  turn_off:
317
318  //First, we must push all the registers we are using to the stack. This is so
    that the original values are restored later on
319  push r16
320  push r17
321  push r18
322  in r16, CPU_SREG
323  push r16
324  push XL
325  push XH
326
327  //Handles the interrupt and resets the timer
328  ldi r16, TCA_SINGLE_OVF_bm ;clear OVF flag
329  sts TCA0_SINGLE_INTFLAGS, r16
330
331  ldi r16, 0xFF
332  sts PORTA_OUT, r16
333  lds r17, digit_num
334  inc r17
335  //If we are at position 4, return to position 0
336  cpi r17, 0x04
337  brsh overflow
338
339  output:
340  sts digit_num, r17
341  ldi XH, HIGH(led_display)
342  ldi XL, LOW(led_display)
343  add XL, r17
344  ld r18, X
345  //rcall hex_to_7seg
346  sts PORTD_OUT, r18
347
348  //Will check digit_num to decide which display on the 7seg is being outputted
349  checking_dig:
350  cpi r17, 0x00
351  breq dig0
352  cpi r17, 0x01
353  breq dig1

```

```

354     cpi r17, 0x02
355     breq dig2
356     cpi r17, 0x03
357     breq dig3
358
359     //r18 stores the value of the digit to be displayed
360 dig0:
361     ldi r18, 0x70 //PA7
362     sts PORTA_OUT, r18
363     rjmp restore
364
365 dig1:
366     ldi r18, 0xB0 //PA6
367     sts PORTA_OUT, r18
368     rjmp restore
369
370 dig2:
371     ldi r18, 0xD0 //PA5
372     sts PORTA_OUT, r18
373     rjmp restore
374
375 dig3:
376     ldi r18, 0xE0 //PA4
377     sts PORTA_OUT, r18
378     rjmp restore
379
380 //Reads off the stack the values that we preserved inside turn_off
381 restore:
382     pop XH
383     pop XL
384     pop r16
385     out CPU_SREG, r16
386     pop r18
387     pop r17
388     pop r16
389
390     reti
391
392 //Sets digit_num back to 0, since we only have 4 digits
393 overflow:
394     ldi r17, 0x00
395     sts digit_num, r17
396     ldi XH, HIGH(led_display)
397     ldi XL, LOW(led_display)
398     rjmp output
399
400
401 ;*****
402 ;*
403 ;* "post_display" - Power On Self Test
404 ;*
405 ;* Description: Will POST the 7segment display hooked up to port D, multiplexed

```

```

    by PA7-PA4.
406 ;*Individually turns on each segment for a brief moment, totalling 1 second
407 ;* Author: Kyle Han
408 ;* Version: 0.1
409 ;* Last updated: 11/29/2021
410 ;* Target: AVR128DB48
411 ;* Number of words:
412 ;* Number of cycles:
413 ;* Low registers modified: r16-r20
414 ;* High registers modified:
415 ;*
416 ;* Parameters: None
417 ;*
418 ;* Returns: None
419 ;*
420 ;* Notes: Requires digit_num to be declared in .dseg
421 ;*
422 ;*****
423 post_display:
424     //A value of 0 turns on our LED displays
425     ldi r16, 0x00
426     out VPORTD_OUT, r16
427     sts digit_num, r16 ;Store 0x00 to the current digit to be displayed
428 repeatPost:
429     in r17, VPORTA_OUT
430     ori r17, 0xF0
431     out VPORTA_OUT, r17
432     ldi r19, 0b00010000 ;This 1 will turn on a specific digit.
433 leftShiftPOST:
434     cpi r16, 0
435     breq postDispOn
436     lsl r19
437     dec r16
438     rjmp leftShiftPOST
439 postDispOn:
440     in r17, VPORTA_OUT
441     eor r17, r19
442     out VPORTA_OUT, r17
443     ldi r20, 10
444     rcall oneSecDelay
445     //Compares digit_num to 0x04. If greater than or equal to, we've finished
    posting
446     lds r16, digit_num
447     inc r16
448     sts digit_num, r16
449     cpi r16, 0x04
450     brlo repeatPost
451
452     in r17, VPORTA_OUT
453     ori r17, 0xF0
454     out VPORTA_OUT, r17
455     ret

```

```

456
457 ;*****
458 ;*
459 ;* "oneSecDelay" - Delay the microcontroller by 1 second by occupying CPU time
460 ;*
461 ;* Description: Loops through and occupies CPU time to delay by 1 second
462 ;*
463 ;* Author: Kyle Han
464 ;* Version: 1.0
465 ;* Last updated: 11/29/2021
466 ;* Target: AVR128DB48
467 ;* Number of words:
468 ;* Number of cycles:
469 ;* Low registers modified:r17
470 ;* High registers modified:
471 ;*
472 ;* Parameters: r20 * .1ms
473 ;*
474 ;* Returns: Nothing
475 ;*
476 ;* Notes:
477 ;*
478 ;*****
479 oneSecDelay:
480 outer_loop:
481     ldi r17, 133
482 inner_loop:
483     dec r17
484     brne inner_loop
485     dec r20
486     brne outer_loop
487     ret
488
489 ;*****
490 ;*
491 ;* "mpy16u" - 16x16 Bit Unsigned Multiplication
492 ;*
493 ;* This subroutine multiplies the two 16-bit register variables
494 ;* mp16uH:mp16uL and mc16uH:mc16uL.
495 ;* The result is placed in m16u3:m16u2:m16u1:m16u0.
496 ;*
497 ;* Number of words :14 + return
498 ;* Number of cycles :153 + return
499 ;* Low registers used :None
500 ;* High registers used :7 (mp16uL,mp16uH,mc16uL/m16u0,mc16uH/m16u1,m16u2,
501 ;*                      m16u3,mcnt16u)
502 ;*
503 ;*****
504
505 ;***** Subroutine Register Variables
506
507 .def    mc16uL    =r16            ;multiplicand low byte

```

```

508 .def      mc16uH  =r17      ;multiplicand high byte
509 .def      mp16uL  =r18      ;multiplier low byte
510 .def      mp16uH  =r19      ;multiplier high byte
511 .def      m16u0   =r18      ;result byte 0 (LSB)
512 .def      m16u1   =r19      ;result byte 1
513 .def      m16u2   =r20      ;result byte 2
514 .def      m16u3   =r21      ;result byte 3 (MSB)
515 .def      mcnt16u =r22      ;loop counter
516
517 ;***** Code
518
519 mpy16u: clr m16u3      ;clear 2 highest bytes of result
520         clr m16u2
521         ldi mcnt16u,16 ;init loop counter
522         lsr mp16uH
523         ror mp16uL
524
525 m16u_1: brcc  noad8      ;if bit 0 of multiplier set
526         add m16u2,mc16uL ;add multiplicand Low to byte 2 of res
527         adc m16u3,mc16uH ;add multiplicand high to byte 3 of res
528 noad8:  ror m16u3        ;shift right result byte 3
529         ror m16u2        ;rotate right result byte 2
530         ror m16u1        ;rotate result byte 1 and multiplier High
531         ror m16u0        ;rotate result byte 0 and multiplier Low
532         dec mcnt16u      ;decrement loop counter
533         brne m16u_1      ;if not done, loop more
534         ret
535
536
537 ;*****
538 ;*
539 ;* "bin2BCD16" - 16-bit Binary to BCD conversion
540 ;*
541 ;* This subroutine converts a 16-bit number (fbinH:fbinL) to a 5-digit
542 ;* packed BCD number represented by 3 bytes (tBCD2:tBCD1:tBCD0).
543 ;* MSD of the 5-digit number is placed in the lowermost nibble of tBCD2.
544 ;*
545 ;* Number of words :25
546 ;* Number of cycles :751/768 (Min/Max)
547 ;* Low registers used :3 (tBCD0,tBCD1,tBCD2)
548 ;* High registers used :4(fbinL,fbinH,cnt16a,tmp16a)
549 ;* Pointers used :Z
550 ;*
551 ;*****
552
553 ;***** Subroutine Register Variables
554
555 .dseg
556 tBCD0: .byte 1 // BCD digits 1:0
557 tBCD1: .byte 1 // BCD digits 3:2
558 tBCD2: .byte 1 // BCD digits 4
559

```

```

560 .cseg
561 .def    tBCD0_reg = r13      ;BCD value digits 1 and 0
562 .def    tBCD1_reg = r14      ;BCD value digits 3 and 2
563 .def    tBCD2_reg = r15      ;BCD value digit 4
564
565 .def    fbinL = r16          ;binary value Low byte
566 .def    fbinH = r17          ;binary value High byte
567
568 .def    cnt16a =r18           ;loop counter
569 .def    tmp16a =r19           ;temporary value
570
571 ;***** Code
572
573 bin2BCD16:
574     push fbinL
575     push fbinH
576     push cnt16a
577     push tmp16a
578
579
580     ldi cnt16a, 16 ;Init loop counter
581     ldi r20, 0x00
582     sts tBCD0, r20 ;clear result (3 bytes)
583     sts tBCD1, r20
584     sts tBCD2, r20
585 bBCDx_1:
586     // load values from memory
587     lds tBCD0_reg, tBCD0
588     lds tBCD1_reg, tBCD1
589     lds tBCD2_reg, tBCD2
590
591     lsl fbinL          ;shift input value
592     rol fbinH          ;through all bytes
593     rol tBCD0_reg      ;
594     rol tBCD1_reg
595     rol tBCD2_reg
596
597     sts tBCD0, tBCD0_reg
598     sts tBCD1, tBCD1_reg
599     sts tBCD2, tBCD2_reg
600
601     dec cnt16a          ;decrement loop counter
602     brne bBCDx_2        ;if counter not zero
603
604     pop tmp16a
605     pop cnt16a
606     pop fbinH
607     pop fbinL
608     ret                ; return
609 bBCDx_2:
610     // Z Points tBCD2 + 1, MSB of BCD result + 1
611     ldi ZL, LOW(tBCD2 + 1)

```



```

612     ldi ZH, HIGH(tBCD2 + 1)
613     bBCDx_3:
614         ld tmp16a, -Z        ;get (Z) with pre-decrement
615         subi tmp16a, -$03    ;add 0x03
616
617         sbrc tmp16a, 3        ;if bit 3 not clear
618         st Z, tmp16a         ;store back
619
620         ld tmp16a, Z          ;get (Z)
621         subi tmp16a, -$30     ;add 0x30
622
623         sbrc tmp16a, 7        ;if bit 7 not clear
624         st Z, tmp16a         ; store back
625
626         cpi ZL, LOW(tBCD0)    ;done all three?
627         brne bBCDx_3
628         cpi ZH, HIGH(tBCD0)  ;done all three?
629         brne bBCDx_3
630     rjmp bBCDx_1

```