ECE:3360 – Lab 3 Report

Oliver Emery and Austin Wittenburg

9 March 2022

1 Introduction

The goal of this lab is to get more experience with timers, single-wire communication, and rotary pulse generators. We will do this by building a thermostat using seven-segment displays, a DHT11 sensor, a rotary pulse generator, and a push button switch with a software debounce. The thermostat should have two modes. The first mode should allow the user to use the RPG to set the desired temperature on the thermostat. The user should know that they are in this mode because both of the decimal points on the seven-segment displays will be on. When the user pushes the button, the mode should change to displaying the actual temperature. Also if the actual temperature is lower than the desired temperature, then the yellow LED "L" on the Arduino board should be turned on, indicating that there is some sort of heating element being activated.

2 Schematic

Figure 1 shows our circuit design as it was implemented. All resistor values are $1\,\mathrm{k}\Omega$, with the exception of the resistor between the ATmega328P and the DHT11, which is $330\,\Omega$. The circuit is running on $5\,\mathrm{V}$.

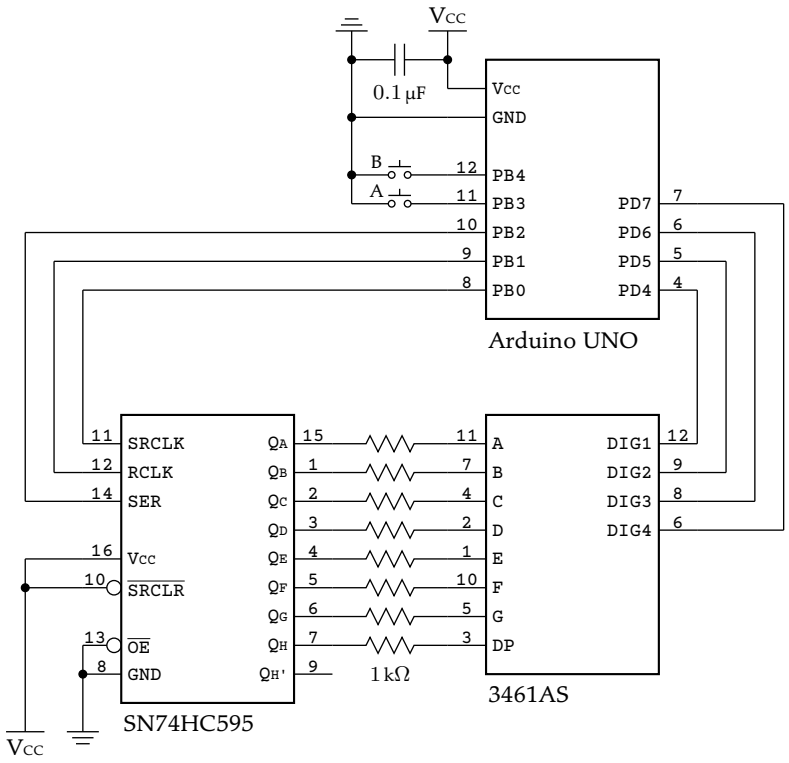

Figure 1: schematic as implemented

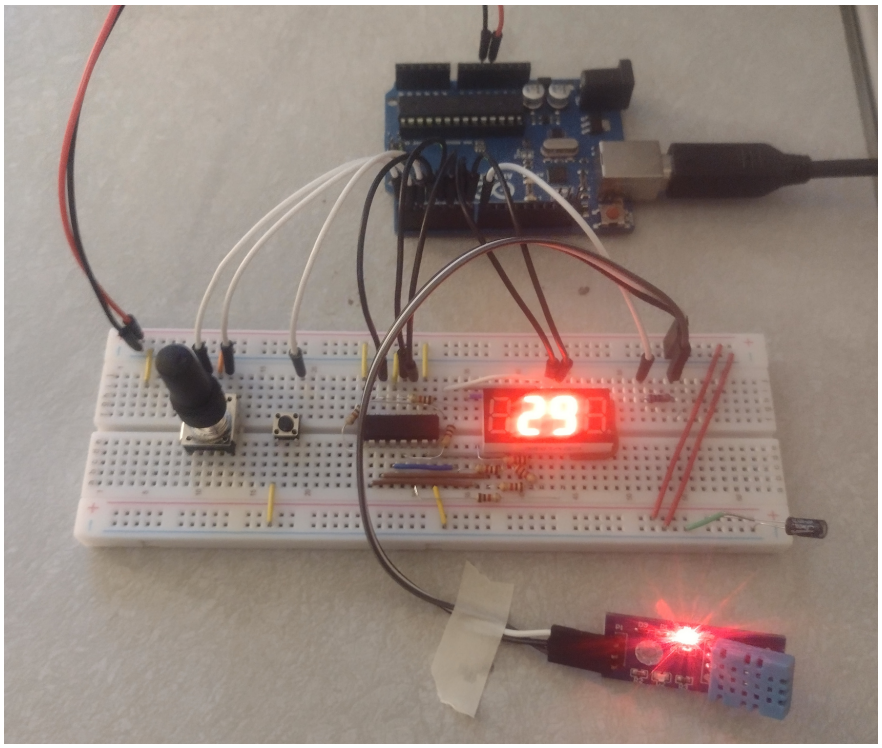

Figure 2: physical implementation

3 Discussion

The software debounce we used in this lab is similar to the debounce method we used in lab 2, but we essentially just treated the RPG as two buttons when we were debouncing it. In fact a lot of our setup was carried over from lab 2 including the placement and wiring of the shift register and seven-segment display. Using the serial connection to communicate with the DHT11 was trickier than expected. We ran into a lot of problems with timing the signal to be able to read it reliably every time.

Timer Usage

We used one 8-bit timer for all time-based operations. For the majority of each second during which the DHT11 is not being read, we have the timer configured in clear timer on compare match mode. The compare register is set to target a $250\,\mu s$ interval between compare match flags.

While reading the DHT11, we use a /8 prescaler for $0.5\,\mu s$ resolution, and compare-timer-on-match mode as well.

Reading the DHT11

As an aid in designing the sensor data reading subroutine, we pulled a transaction sample from an oscilloscope and loaded it into PulseView, a frontend for `sigrok`. We wrote a protocol decoder for DHT11 messages to gain a better understanding of the signal format.

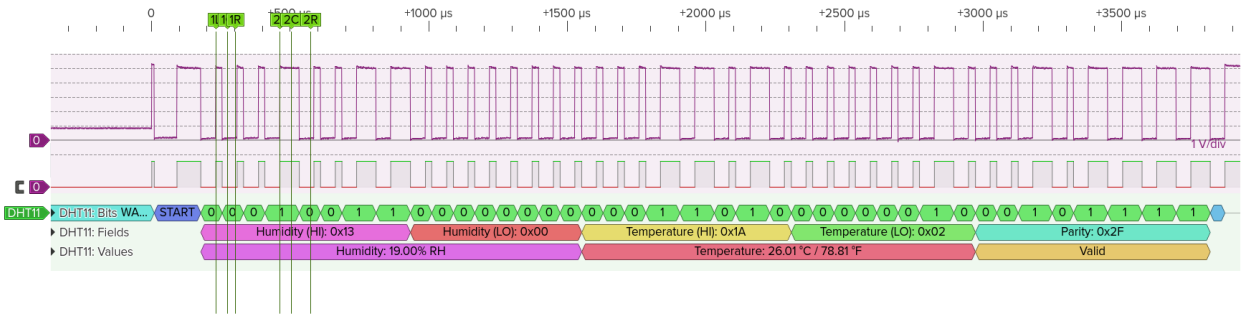

Figure 3: PulseView display with custom protocol decoder

Our timing algorithm is illustrated in Figure 4. We first synchronize to the positive clock edge of the current bit (1L, 2L), then delay for $40\,\mu s$. At this time, the subroutine reads the sensor data pin value (1C, 2C). If the pin is lo (1C), this indicates a 0, and vice versa. Finally, we wait for the next positive clock edge.

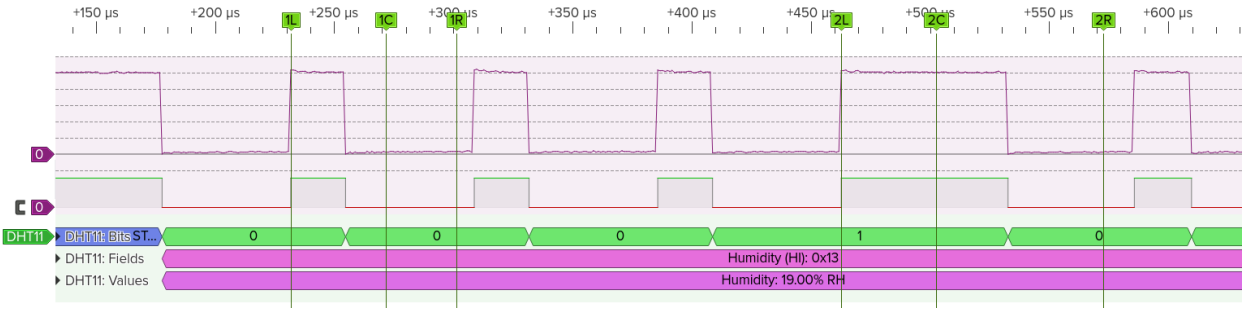

Figure 4: sensor read subroutine intervals

4 Conclusion

This lab helped us better understand the serial communications through the DHT11, and gave us much more experience with using timers. We were able to implement the RPG very easily by treating it similar to just two buttons which allowed us to reuse our previous methods for debouncing which were already very effective.

A Source Code Listing

```
;; project: ece3360-lab03
;; file:    main.S
;; date:    20220309
;; author:  Oliver Emery

.include "m328Pdef.inc"

;; *** Defines
        .equ P_RPG_A    = PIND0
        .equ P_RPG_B    = PIND1
        .equ P_BTN      = PIND2
        .equ P_SER      = PIND3
        .equ P_RCLK     = PIND4
        .equ P_SRCLK    = PIND5
        .equ P_DIG0     = PIND7
        .equ P_DIG1     = PIND6

        .equ P_DHT11    = PINB0
        .equ P_LED      = PINB5

        .equ MODE_ACQUIRE       = 0x01
        .equ MODE_SET           = 0x02

        .equ STATE_IDLE         = 0x01
        .equ STATE_WAKE         = 0x02
        .equ STATE_DATA         = 0x03

        .equ ACQUIRE_WINDOW     = 4000
        .equ WAKE_WINDOW        = (4*18)


        ; struct btn_s {
        .equ    btn_pressed     = 0x00  ; 1 if button is pressed, else 0
        .equ    btn_mask        = 0x01  ; 1 << PIN#
        .equ    btn_dwnd        = 0x02  ; detect window
        .equ    btn_duration    = 0x03  ; duration pressed
        .equ    btn_handler     = 0x04  ; change handler subroutine
        ; }
        .equ    sz_btn          = 6

.dseg   ; data segment
.org 0x0100
        threshold:      .byte 2
        temperature:    .byte 2
        digit:          .byte 1
```

```
46
47          mode:           .byte 1
48          acquire_state:  .byte 1
49          acquire_ctr:    .byte 2
50
51          rpg_a:          .byte sz_btn
52          rpg_b:          .byte sz_btn
53          btn:            .byte sz_btn
54
55          sensor_data:    .byte 5
56
57  .cseg   ; code segment
58  .org 0x0000     jmp __entry
59
60  .org INT_VECTORS_SIZE
61  digit_bits: .db \
62          0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110, \
63          0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111, \
64          0b01000000, 0
65
66  __entry: ; entrypoint
67          ldi     r16, high(RAMEND)
68          out     SPH, r16
69          ldi     r16, low(RAMEND)
70          out     SPL, r16
71
72          clr     r0
73
74          call    init
75          jmp     main
76
77  ;; void memclr(Y: void*, r16: len)
78  ;;
79  ;;     clear up to r16 bytes of SRAM at YH:YL
80  ;;
81  memclr:
82          push    r16
83          push    r17
84          push    YL
85
86          clr     r17
87  memclr_loop:
88          st      Y+, r17
89          dec     r16
90          brne    memclr_loop
91
92          pop     YL
```

```
 93          pop     r17
 94          pop     r16
 95          ret
 96
 97  init:
 98          ; outputs
 99          ldi     r16, 1 << P_SER | 1 << P_RCLK | 1 << P_SRCLK | 1 << P_DIG0 | 1 << P_DIG1
100          out     DDRD, r16
101
102          ; inputs
103          ldi     r16, 1 << P_RPG_A | 1 << P_RPG_B | 1 << P_BTN   | 1 << P_DIG0 | 1 << P_DIG1
104          out     PORTD, r16
105
106          ; led
107          ldi     r16, 1 << P_LED
108          out     DDRB, r16
109          cbi     PORTB, P_LED
110
111          ldi     r16, 61
112          out     OCR0A, r16
113          ldi     r16, 1 << WGM01
114          out     TCCR0A, r16
115          ldi     r16, 1 << CS01 | 1 << CS00
116          out     TCCR0B, r16
117
118          ret
119
120
121  main:
122          ldi     YH, 0x01
123          ldi     ZH, high(digit_bits << 1)
124
125          ldi     r16, MODE_ACQUIRE
126          sts     mode, r16
127
128          ldi     r16, STATE_IDLE
129          sts     acquire_state, r16
130
131          ldi     r16, high(ACQUIRE_WINDOW)
132          sts     acquire_ctr, r16
133          ldi     r16, low(ACQUIRE_WINDOW)
134          sts     acquire_ctr+1, r16
135
136
137          ldi     YL, low(threshold)
138          st      Y+, r0
139          st      Y, r0
```

```
140
141         ldi     YL, low(temperature)
142         st      Y+, r0
143         st      Y, r0
144
145         sts     digit, r0
146
147         ; Initialize button structures
148         ldi     r16, sz_btn
149
150         ldi     YL, low(rpg_a)
151         rcall   memclr
152         ldi     r17, 1 << P_RPG_A
153         ldi     r18, high(rpg_a_changed)
154         ldi     r19, low(rpg_a_changed)
155         std     Y+btn_mask, r17
156         std     Y+btn_handler, r18
157         std     Y+btn_handler+1, r19
158
159         ldi     YL, low(rpg_b)
160         rcall   memclr
161         ldi     r17, 1 << P_RPG_B
162         ldi     r18, high(rpg_b_changed)
163         ldi     r19, low(rpg_b_changed)
164         std     Y+btn_mask, r17
165         std     Y+btn_handler, r18
166         std     Y+btn_handler+1, r19
167
168         ldi     YL, low(btn)
169         rcall   memclr
170         ldi     r17, 1 << P_BTN
171         ldi     r18, high(btn_changed)
172         ldi     r19, low(btn_changed)
173         std     Y+btn_mask, r17
174         std     Y+btn_handler, r18
175         std     Y+btn_handler+1, r19
176
177 main_loop:
178         call    every_interval
179 main_wait_ocf:
180         sbis    TIFR0, OCF0A
181         rjmp    main_wait_ocf
182         sbi     TIFR0, OCF0A
183
184         clr     r16
185         out     TCCR0B, r16
186         ldi     r16, 10
```

```
187 main_adjust_cycles:     ; perfect 250us intervals
188         dec     r16
189         brne    main_adjust_cycles
190
191         ldi     r16, 1 << CS01 | 1 << CS00
192         out     TCCR0B, r16
193
194         ; reset prescaler
195         in      r16, GTCCR
196         sbr     r16, PSRSYNC
197         out     GTCCR, r16
198
199         rjmp    main_loop
200
201
202 every_interval:
203         lds     r16, mode
204
205         cpi     r16, MODE_SET
206         brne    every_interval_mode_elsif_acquire
207         ldi     YL, low(threshold)
208         ldi     r18, 1
209         rjmp    every_interval_write_digits
210
211 every_interval_mode_elsif_acquire:
212         cpi     r16, MODE_ACQUIRE
213         brne    every_interval_debounce
214
215         ldi     YL, low(temperature)
216         ldi     r18, 0
217
218 every_interval_write_digits:
219         rcall   write_digits
220
221 every_interval_debounce:
222         ldi     YL, low(rpg_a)
223         rcall   debounce
224         ldi     YL, low(rpg_b)
225         rcall   debounce
226         ldi     YL, low(btn)
227         rcall   debounce
228
229         lds     r16, acquire_state
230         cpi     r16, STATE_DATA
231         breq    every_interval_ret
232
233         ldi     YL, low(acquire_ctr)
```

```
234         ld      r24, Y+
235         ld      r25, Y
236         sbiw    r25:r24, 1
237         st      Y, r25
238         st      -Y, r24
239         brne    every_interval_ret
240
241
242         lds     r16, acquire_state
243         cpi     r16, STATE_IDLE
244         brne    every_interval_state_elsif_wake
245
246         ldi     r16, WAKE_WINDOW
247         st      Y, r16
248
249         ldi     r16, STATE_WAKE
250         sts     acquire_state, r16
251
252         ; pull data pin lo
253         sbi     DDRB, P_DHT11
254
255         rjmp    every_interval_ret
256 every_interval_state_elsif_wake:
257         cpi     r16, STATE_WAKE
258         brne    every_interval_ret
259
260         ldi     r24, low(ACQUIRE_WINDOW)
261         ldi     r25, high(ACQUIRE_WINDOW)
262         st      Y+, r24
263         st      Y, r25
264
265         ldi     r16, STATE_DATA
266         sts     acquire_state, r16
267
268         ; release data pin
269         cbi     DDRB, P_DHT11
270
271         rcall   do_acquire
272 every_interval_ret:
273         ret
274
275
276 wait_for_data_lo:
277         sbic    PINB, P_DHT11
278         rjmp    wait_for_data_lo
279         ret
280
```

```
281 wait_for_data_hi:
282         sbis    PINB, P_DHT11
283         rjmp    wait_for_data_hi
284         ret
285
286
287 do_acquire:
288         ldi     r20, 1 << CS01  ; /8 scaling aka 0.5us
289         out     TCCR0B, r20
290
291         ldi     r20, 50
292         rcall   wait_for_nus
293         rcall   wait_for_data_hi
294         rcall   wait_for_data_lo
295         rcall   wait_for_data_hi
296
297         ldi     YL, low(sensor_data)
298
299         ldi     r16, 5
300 do_acquire_bytes:
301         ldi     r17, 8
302 do_acquire_bits:
303         ldi     r20, 40
304         rcall   wait_for_nus
305
306         sbic    PINB, P_DHT11
307         rjmp    do_acquire_bits_hi
308
309         ldi     r20, 30
310         rcall   wait_for_nus
311         clc
312         rjmp    do_acquire_bits_shift
313 do_acquire_bits_hi:
314         ldi     r20, 60
315         rcall   wait_for_nus
316         sec
317 do_acquire_bits_shift:
318         rol     r18
319         rcall   wait_for_data_hi
320
321         dec     r17
322         brne    do_acquire_bits
323
324         st      Y+, r18
325
326         dec     r16
327         brne    do_acquire_bytes
```

```
328
329         ldi     YL, low(threshold)
330         ld      r17, Y+
331         ld      r18, Y
332         ldi     r19, 10
333         mul     r18, r19
334         mov     r18, r0
335         clr     r0
336         add     r17, r18
337
338         ldi     YL, low(sensor_data)
339         ldd     r16, Y+2
340
341         cp      r16, r17
342         brlo    do_acquire_heat
343         cbi     PORTB, P_LED
344         rjmp    do_acquire_div
345 do_acquire_heat:
346         sbi     PORTB, P_LED
347 do_acquire_div:
348         ldi     r17, 10
349         rcall   div8u
350
351         ldi     YL, low(temperature)
352         st      Y+, r15
353         st      Y, r16
354
355         ldi     r16, STATE_IDLE
356         sts     acquire_state, r16
357
358 do_acquire_ret:
359         ; reset timer for normal mode
360         ldi     r16, 61
361         out     OCR0A, r16
362         ldi     r16, 1 << CS01 | 1 << CS00
363         out     TCCR0B, r16
364
365         ret
366
367
368 wait_for_nus:
369         lsl     r20
370         out     OCR0A, r20
371         out     TCNT0, r0
372 wait_for_nus_wait:
373         sbis    TIFR0, OCF0A
374         rjmp    wait_for_nus_wait
```

```
375         sbi     TIFR0, OCF0A
376         ret
377
378 rpg_a_changed:
379         push    YL
380         tst     r16
381         brne    rpg_a_changed_ret
382         ldi     YL, low(rpg_b)
383         ldd     r16, Y+btn_pressed
384         tst     r16
385         brne    rpg_a_changed_ret
386         ldi     YL, low(mode)
387         ld      r16, Y
388         cpi     r16, MODE_SET
389         brne    rpg_a_changed_ret
390         rcall   inc_threshold
391 rpg_a_changed_ret:
392         pop     YL
393         ret
394
395
396 rpg_b_changed:
397         push    YL
398         tst     r16
399         brne    rpg_b_changed_ret
400         ldi     YL, low(rpg_a)
401         ldd     r16, Y+btn_pressed
402         tst     r16
403         brne    rpg_b_changed_ret
404         ldi     YL, low(mode)
405         ld      r16, Y
406         cpi     r16, MODE_SET
407         brne    rpg_b_changed_ret
408         rcall   dec_threshold
409 rpg_b_changed_ret:
410         pop     YL
411         ret
412
413
414 btn_changed:
415         push    YL
416         tst     r16
417         breq    btn_changed_ret
418         ldi     YL, low(mode)
419         ld      r16, Y
420         cpi     r16, MODE_SET
421         brne    btn_changed_mode_elsif_acquire
```

```
422         ldi     r16, MODE_ACQUIRE
423         rjmp    btn_changed_mode_store
424 btn_changed_mode_elsif_acquire:
425         cpi     r16, MODE_ACQUIRE
426         brne    btn_changed_ret
427         ldi     r16, MODE_SET
428 btn_changed_mode_store:
429         sts     mode, r16
430 btn_changed_ret:
431         pop     YL
432         ret
433
434
435 inc_threshold:
436         lds     r16, threshold
437         inc     r16
438         cpi     r16, 10
439         brne    inc_threshold_nowrap
440         clr     r16
441         lds     r17, threshold+1
442         inc     r17
443         cpi     r17, 10
444         brne    inc_threshold_nw2
445         clr     r17
446 inc_threshold_nw2:
447         sts     threshold+1, r17
448 inc_threshold_nowrap:
449         sts     threshold, r16
450         rcall   write_digit
451         ret
452
453 dec_threshold:
454         lds     r16, threshold
455         dec     r16
456         brpl    dec_threshold_nowrap
457         ldi     r16, 9
458         lds     r17, threshold+1
459         dec     r17
460         brpl    dec_threshold_nw2
461         ldi     r17, 9
462 dec_threshold_nw2:
463         sts     threshold+1, r17
464 dec_threshold_nowrap:
465         sts     threshold, r16
466         rcall   write_digit
467         ret
468
```

```
469
470 ;; void debounce(YL: *button)
471 ;;
472 ;;     Sample and process raw button input data to reliably detect and handle
473 ;;     button events. Big idea: register a change in button state if and only
474 ;;     if it holds the changed state steady for a specified window of time.
475 ;;
476 debounce:
477         push    r2
478         push    r1
479         push    r16
480         push    ZH
481         push    ZL
482         clr     r16
483         in      r2, PIND
484         ldd     r1, Y+btn_mask
485         and     r2, r1
486         brne    debounce_notpressed
487         inc     r16
488 debounce_notpressed:            ; byte pressed = (PINB & btn->mask) ? 0 : 1;
489         ldd     r2, Y+btn_pressed
490         cp      r16, r2
491         breq    debounce_coda           ; if (btn->pressed != pressed) {
492         ldd     r2, Y+btn_dwnd
493         dec     r2
494         std     Y+btn_dwnd, r2
495         brne    debounce_ret            ; if (--btn->dwnd) return;
496         std     Y+btn_pressed, r16      ;     btn->pressed = pressed;
497         ldd     ZH, Y+btn_handler
498         ldd     ZL, Y+btn_handler+1
499         icall                           ;     btn->handler();
500 debounce_coda:                          ; }
501         ldi     r16, 8
502         std     Y+btn_dwnd, r16         ; btn->dwnd = WND_MSC;
503 debounce_ret:
504         pop     ZL
505         pop     ZH
506         pop     r16
507         pop     r1
508         pop     r2
509         ret
510
511 write_digits:
512         lds     r17, digit
513         add     YL, r17
514         ld      r16, Y
515         tst     r17
```

```
516         brne    write_digits_d1
517         sbi     PORTD, P_DIG1
518         rcall   write_digit
519         cbi     PORTD, P_DIG0
520         ldi     r17, 1
521         rjmp    write_digits_dun
522 write_digits_d1:
523         sbi     PORTD, P_DIG0
524         rcall   write_digit
525         cbi     PORTD, P_DIG1
526         ldi     r17, 0
527 write_digits_dun:
528         sts     digit, r17
529         ret
530
531
532 ;; void write_digit(r16: charn, r18: decimal)
533 write_digit:
534         ldi     ZH, high(digit_bits << 1)
535         ldi     ZL, low(digit_bits << 1)
536         add     ZL, r16
537         lpm     r19, Z
538         tst     r18
539         breq    write_digit_no_dp
540         ori     r19, 1 << 7
541 write_digit_no_dp:
542         rcall   put_sr_byte
543         ret
544
545
546 ;; void put_sr_byte(r19: byte)
547 ;;
548 ;;     Put a byte into the shift register.
549 ;;
550 put_sr_byte:
551         ldi     r20, 8
552 put_sr_byte_while:
553         rol     r19
554         brcs    put_sr_byte_while_hibit
555         cbi     PORTD, P_SER
556         rjmp    put_sr_byte_wend
557 put_sr_byte_while_hibit:
558         sbi     PORTD, P_SER
559 put_sr_byte_wend:
560         ; trigger SRCLK, shifting SER into the shift register. note that there
561         ; is no need for a delay: even if SBI/CBI only took 1 clock cycle, the
562         ; SN74HC595N supports up to 20 MHz while the UNO runs at only 16 MHz
```

```
563         sbi     PORTD, P_SRCLK
564         cbi     PORTD, P_SRCLK
565         dec     r20
566         brne    put_sr_byte_while
567         ; trigger RCLK to transfer shift register data to the storage register
568         sbi     PORTD, P_RCLK
569         cbi     PORTD, P_RCLK
570         ret
571
572
573 .include "div8u.inc" ; subroutine from atmel AVR200 library
574
575 .exit
```

B References

1. "DHT11 Datasheet" <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>

2. "SN74HC595 Datasheet" <https://www.sparkfun.com/datasheets/IC/SN74HC595.pdf>

3. "3461AS Datasheet" <http://www.xlitx.com/datasheet/3461AS.pdf>