```
C:\Users\OscylO\AppData\Local\Temp\build8586335133903286403.tmp\UnoLight.cpp.elf:     file format elf32-avr


Disassembly of section .text:

00000000 <__vectors>:
	SREG = oldSREG;

	return m;
}

unsigned long micros() {
   0:	0c 94 63 00 	jmp	0xc6	; 0xc6 <__ctors_end>
		#endif
	}
}

void digitalWrite(uint8_t pin, uint8_t val)
{
   4:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>

HardwareSerial::HardwareSerial(ring_buffer *rx_buffer, ring_buffer *tx_buffer,
  volatile uint8_t *ubrrh, volatile uint8_t *ubrrl,
  volatile uint8_t *ucsra, volatile uint8_t *ucsrb,
  volatile uint8_t *udr,
  uint8_t rxen, uint8_t txen, uint8_t rxcie, uint8_t udrie, uint8_t u2x)
   8:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
    n += write(*buffer++);
  }
  return n;
}

size_t Print::print(const __FlashStringHelper *ifsh)
   c:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  10:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  14:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  18:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  1c:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  20:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  24:	0c 94 8d 00 	jmp	0x11a	; 0x11a <__vector_9>
  28:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  2c:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  30:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  34:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  38:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  3c:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  40:	0c 94 87 04 	jmp	0x90e	; 0x90e <__vector_16>
  44:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  48:	0c 94 3f 05 	jmp	0xa7e	; 0xa7e <__vector_18>
  4c:	0c 94 86 05 	jmp	0xb0c	; 0xb0c <__vector_19>
  50:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  54:	0c 94 8b 00 	jmp	0x116	; 0x116 <__bad_interrupt>
  58:   0c 94 8b 00     jmp     0x116   ; 0x116 <__bad_interrupt>
  5c:   0c 94 8b 00     jmp     0x116   ; 0x116 <__bad_interrupt>
  60:   0c 94 8b 00     jmp     0x116   ; 0x116 <__bad_interrupt>
  64:   0c 94 8b 00     jmp     0x116   ; 0x116 <__bad_interrupt>

00000068 <port_to_mode_PGM>:
  68:   00 00 00 00 24 00 27 00 2a 00                       ....$.'.*.

00000072 <port_to_output_PGM>:
  72:   00 00 00 00 25 00 28 00 2b 00                       ....%.(.+.

0000007c <port_to_input_PGM>:
  7c:   00 00 00 00 23 00 26 00 29 00                       ....#.&.).

00000086 <digital_pin_to_port_PGM>:
  86:   04 04 04 04 04 04 04 04 02 02 02 02 02 02 03 03     ................
  96:   03 03 03 03                                         ....

0000009a <digital_pin_to_bit_mask_PGM>:
  9a:   01 02 04 08 10 20 40 80 01 02 04 08 10 20 01 02     ..... @...... ..
  aa:   04 08 10 20                                         ...

000000ae <digital_pin_to_timer_PGM>:
  ae:   00 00 00 07 00 02 01 00 00 03 04 06 00 00 00 00     ................
  be:   00 00 00 00                                         ....

000000c2 <__ctors_start>:
  c2:   9c 01           movw    r18, r24
  c4:   fd 06           cpc     r15, r29

000000c6 <__ctors_end>:
  c6:   11 24           eor     r1, r1
  c8:   1f be           out     0x3f, r1        ; 63
  ca:   cf ef           ldi     r28, 0xFF       ; 255
  cc:   d4 e0           ldi     r29, 0x04       ; 4
  ce:   de bf           out     0x3e, r29       ; 62
  d0:   cd bf           out     0x3d, r28       ; 61

000000d2 <__do_copy_data>:
  d2:   13 e0           ldi     r17, 0x03       ; 3
  d4:   a0 e0           ldi     r26, 0x00       ; 0
  d6:   b1 e0           ldi     r27, 0x01       ; 1
  d8:   ee ea           ldi     r30, 0xAE       ; 174
  da:   ff e0           ldi     r31, 0x0F       ; 15
  dc:   02 c0           rjmp    .+4             ; 0xe2 <.do_copy_data_start>

000000de <.do_copy_data_loop>:
  de:   05 90           lpm     r0, Z+
  e0:   0d 92           st      X+, r0

000000e2 <.do_copy_data_start>:
  e2:   a8 31           cpi     r26, 0x18       ; 24
  e4:   b1 07           cpc     r27, r17
  e6:   d9 f7           brne    .-10            ; 0xde <.do_copy_data_loop>
```

```
000000e8 <__do_clear_bss>:
  e8:   14 e0           ldi     r17, 0x04       ; 4
  ea:   a8 e1           ldi     r26, 0x18       ; 24
  ec:   b3 e0           ldi     r27, 0x03       ; 3
  ee:   01 c0           rjmp    .+2             ; 0xf2 <.do_clear_bss_start>

000000f0 <.do_clear_bss_loop>:
  f0:   1d 92           st      X+, r1

000000f2 <.do_clear_bss_start>:
  f2:   ac 34           cpi     r26, 0x4C       ; 76
  f4:   b1 07           cpc     r27, r17
  f6:   e1 f7           brne    .-8             ; 0xf0 <.do_clear_bss_loop>

000000f8 <__do_global_ctors>:
  f8:   10 e0           ldi     r17, 0x00       ; 0
  fa:   c6 ec           ldi     r28, 0xC6       ; 198
  fc:   d0 e0           ldi     r29, 0x00       ; 0
  fe:   04 c0           rjmp    .+8             ; 0x108 <.do_global_ctors_start>

00000100 <.do_global_ctors_loop>:
 100:   22 97           sbiw    r28, 0x02       ; 2
 102:   fe 01           movw    r30, r28
 104:   0e 94 d1 07     call    0xfa2   ; 0xfa2 <__tablejump__>

00000108 <.do_global_ctors_start>:
 108:   c2 3c           cpi     r28, 0xC2       ; 194
 10a:   d1 07           cpc     r29, r17
 10c:   c9 f7           brne    .-14            ; 0x100 <.do_global_ctors_loop>
 10e:   0e 94 4d 07     call    0xe9a   ; 0xe9a <main>
 112:   0c 94 d5 07     jmp     0xfaa   ; 0xfaa <_exit>

00000116 <__bad_interrupt>:
 116:   0c 94 00 00     jmp     0       ; 0x0 <__vectors>

0000011a <__vector_9>:
#error : There is no timer initialization code for your version of ATmega. Only ATmega168, ATmega328p and
AVR_ATmega2560 are supported.
#endif
}
//----------------------------------------------------------------------------------------------------
//--------------- TIMER2 Overflow interrupt---------------
ISR(TIMER2_OVF_vect)
 11a:   1f 92           push    r1
 11c:   0f 92           push    r0
 11e:   0f b6           in      r0, 0x3f        ; 63
 120:   0f 92           push    r0
 122:   11 24           eor     r1, r1
 124:   2f 93           push    r18
 126:   3f 93           push    r19
 128:   8f 93           push    r24
 12a:   9f 93           push    r25
{
  static word pwmCounter = 1024;
  word * ledValPtr = &ledChannels[0][0];

  if( !( pwmCounter & 0xfc00) )  // works faster then (pwmCounter < 1024)
 12c:   20 91 00 01     lds     r18, 0x0100
 130:   30 91 01 01     lds     r19, 0x0101
 134:   c9 01           movw    r24, r18
 136:   80 70           andi    r24, 0x00       ; 0
 138:   9c 7f           andi    r25, 0xFC       ; 252
 13a:   89 2b           or      r24, r25
 13c:   09 f0           breq    .+2             ; 0x140 <__vector_9+0x26>
 13e:   7a c0           rjmp    .+244           ; 0x234 <__vector_9+0x11a>
  {
#if !defined (LEDS_UNDEFINED)  // if LEDS are defined for this Arduino board in arduinoPins2Ports.h
    //LED 1
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED1_RED);
 140:   80 91 1c 03     lds     r24, 0x031C
 144:   90 91 1d 03     lds     r25, 0x031D
 148:   82 17           cp      r24, r18
 14a:   93 07           cpc     r25, r19
 14c:   08 f4           brcc    .+2             ; 0x150 <__vector_9+0x36>
 14e:   2c 98           cbi     0x05, 4 ; 5
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED1_GREEN);
 150:   80 91 1e 03     lds     r24, 0x031E
 154:   90 91 1f 03     lds     r25, 0x031F
 158:   82 17           cp      r24, r18
 15a:   93 07           cpc     r25, r19
 15c:   08 f4           brcc    .+2             ; 0x160 <__vector_9+0x46>
 15e:   2a 98           cbi     0x05, 2 ; 5
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED1_BLUE);
 160:   80 91 20 03     lds     r24, 0x0320
 164:   90 91 21 03     lds     r25, 0x0321
 168:   82 17           cp      r24, r18
 16a:   93 07           cpc     r25, r19
 16c:   08 f4           brcc    .+2             ; 0x170 <__vector_9+0x56>
 16e:   28 98           cbi     0x05, 0 ; 5

    //LED 2
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED2_RED);
 170:   80 91 22 03     lds     r24, 0x0322
 174:   90 91 23 03     lds     r25, 0x0323
 178:   82 17           cp      r24, r18
 17a:   93 07           cpc     r25, r19
 17c:   08 f4           brcc    .+2             ; 0x180 <__vector_9+0x66>
 17e:   5e 98           cbi     0x0b, 6 ; 11
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED2_GREEN);
 180:   80 91 24 03     lds     r24, 0x0324
 184:   90 91 25 03     lds     r25, 0x0325
 188:   82 17           cp      r24, r18
 18a:   93 07           cpc     r25, r19
 18c:   08 f4           brcc    .+2             ; 0x190 <__vector_9+0x76>
 18e:   5c 98           cbi     0x0b, 4 ; 11
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED2_BLUE);
 190:   80 91 26 03     lds     r24, 0x0326
 194:   90 91 27 03     lds     r25, 0x0327
 198:   82 17           cp      r24, r18
 19a:   93 07           cpc     r25, r19
 19c:   08 f4           brcc    .+2             ; 0x1a0 <__vector_9+0x86>
 19e:   5a 98           cbi     0x0b, 2 ; 11

   //LED 3
   if(*ledValPtr++ < pwmCounter)  LEDOFF(LED3_RED);
 1a0:   80 91 28 03     lds     r24, 0x0328
 1a4:   90 91 29 03     lds     r25, 0x0329
 1a8:   82 17           cp      r24, r18
 1aa:   93 07           cpc     r25, r19
 1ac:   08 f4           brcc    .+2             ; 0x1b0 <__vector_9+0x96>
 1ae:   5d 98           cbi     0x0b, 5 ; 11
   if(*ledValPtr++ < pwmCounter)  LEDOFF(LED3_GREEN);
 1b0:   80 91 2a 03     lds     r24, 0x032A
 1b4:   90 91 2b 03     lds     r25, 0x032B
 1b8:   82 17           cp      r24, r18
 1ba:   93 07           cpc     r25, r19
 1bc:   08 f4           brcc    .+2             ; 0x1c0 <__vector_9+0xa6>
 1be:   29 98           cbi     0x05, 1 ; 5
   if(*ledValPtr++ < pwmCounter)  LEDOFF(LED3_BLUE);
 1c0:   80 91 2c 03     lds     r24, 0x032C
 1c4:   90 91 2d 03     lds     r25, 0x032D
 1c8:   82 17           cp      r24, r18
 1ca:   93 07           cpc     r25, r19
 1cc:   08 f4           brcc    .+2             ; 0x1d0 <__vector_9+0xb6>
 1ce:   5f 98           cbi     0x0b, 7 ; 11

   //LED 4
   if(*ledValPtr++ < pwmCounter)  LEDOFF(LED4_RED);
 1d0:   80 91 2e 03     lds     r24, 0x032E
 1d4:   90 91 2f 03     lds     r25, 0x032F
 1d8:   82 17           cp      r24, r18
 1da:   93 07           cpc     r25, r19
 1dc:   08 f4           brcc    .+2             ; 0x1e0 <__vector_9+0xc6>
 1de:   2d 98           cbi     0x05, 5 ; 5
   if(*ledValPtr++ < pwmCounter)  LEDOFF(LED4_GREEN);
 1e0:   80 91 30 03     lds     r24, 0x0330
 1e4:   90 91 31 03     lds     r25, 0x0331
 1e8:   82 17           cp      r24, r18
 1ea:   93 07           cpc     r25, r19
 1ec:   08 f4           brcc    .+2             ; 0x1f0 <__vector_9+0xd6>
 1ee:   5b 98           cbi     0x0b, 3 ; 11
   if(*ledValPtr++ < pwmCounter)  LEDOFF(LED4_BLUE);
 1f0:   80 91 32 03     lds     r24, 0x0332
 1f4:   90 91 33 03     lds     r25, 0x0333
 1f8:   82 17           cp      r24, r18
 1fa:   93 07           cpc     r25, r19
 1fc:   08 f4           brcc    .+2             ; 0x200 <__vector_9+0xe6>
 1fe:   2b 98           cbi     0x05, 3 ; 5

#if USE_ANALOG_PINS
   //LED 5
```

```
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED5_RED);
 200:   80 91 34 03     lds     r24, 0x0334
 204:   90 91 35 03     lds     r25, 0x0335
 208:   82 17           cp      r24, r18
 20a:   93 07           cpc     r25, r19
 20c:   08 f4           brcc    .+2             ; 0x210 <__vector_9+0xf6>
 20e:   42 98           cbi     0x08, 2 ; 8
    if(*ledValPtr++ < pwmCounter)  LEDOFF(LED5_GREEN);
 210:   80 91 36 03     lds     r24, 0x0336
 214:   90 91 37 03   lds     r25, 0x0337
 218:   82 17           cp      r24, r18
 21a:   93 07           cpc     r25, r19
 21c:   08 f4           brcc    .+2             ; 0x220 <__vector_9+0x106>
 21e:   41 98           cbi     0x08, 1 ; 8
    if(*ledValPtr   < pwmCounter)  LEDOFF(LED5_BLUE);
 220:   80 91 38 03     lds     r24, 0x0338
 224:   90 91 39 03     lds     r25, 0x0339
 228:   82 17           cp      r24, r18
 22a:   93 07           cpc     r25, r19
 22c:   08 f0           brcs    .+2             ; 0x230 <__vector_9+0x116>
 22e:   6f c0           rjmp    .+222           ; 0x30e <__vector_9+0x1f4>
 230:   40 98           cbi     0x08, 0 ; 8
 232:   6d c0           rjmp    .+218           ; 0x30e <__vector_9+0x1f4>
#endif
   }
   else
   {
    pwmCounter = 0;
 234:   10 92 01 01     sts     0x0101, r1
 238:   10 92 00 01     sts     0x0100, r1

    //LED 1
    if(*ledValPtr++) LEDON(LED1_RED);
 23c:   80 91 1c 03     lds     r24, 0x031C
 240:   90 91 1d 03     lds     r25, 0x031D
 244:   89 2b           or      r24, r25
 246:   09 f0           breq    .+2             ; 0x24a <__vector_9+0x130>
 248:   2c 9a           sbi     0x05, 4 ; 5
    if(*ledValPtr++) LEDON(LED1_GREEN);
 24a:   80 91 1e 03     lds     r24, 0x031E
 24e:   90 91 1f 03     lds     r25, 0x031F
 252:   89 2b           or      r24, r25
 254:   09 f0           breq    .+2             ; 0x258 <__vector_9+0x13e>
 256:   2a 9a           sbi     0x05, 2 ; 5
    if(*ledValPtr++) LEDON(LED1_BLUE);
 258:   80 91 20 03     lds     r24, 0x0320
 25c:   90 91 21 03     lds     r25, 0x0321
 260:   89 2b           or      r24, r25
 262:   09 f0           breq    .+2             ; 0x266 <__vector_9+0x14c>
 264:   28 9a           sbi     0x05, 0 ; 5

    //LED 2
    if(*ledValPtr++) LEDON(LED2_RED);
 266:   80 91 22 03     lds     r24, 0x0322
 26a:   90 91 23 03     lds     r25, 0x0323
 26e:   89 2b           or      r24, r25
 270:   09 f0           breq    .+2             ; 0x274 <__vector_9+0x15a>
 272:   5e 9a           sbi     0x0b, 6 ; 11
   if(*ledValPtr++) LEDON(LED2_GREEN);
 274:   80 91 24 03     lds     r24, 0x0324
 278:   90 91 25 03     lds     r25, 0x0325
 27c:   89 2b           or      r24, r25
 27e:   09 f0           breq    .+2             ; 0x282 <__vector_9+0x168>
 280:   5c 9a           sbi     0x0b, 4 ; 11
   if(*ledValPtr++) LEDON(LED2_BLUE);
 282:   80 91 26 03     lds     r24, 0x0326
 286:   90 91 27 03   lds     r25, 0x0327
 28a:   89 2b           or      r24, r25
 28c:   09 f0           breq    .+2             ; 0x290 <__vector_9+0x176>
 28e:   5a 9a           sbi     0x0b, 2 ; 11

   //LED 3
   if(*ledValPtr++) LEDON(LED3_RED);
 290:   80 91 28 03     lds     r24, 0x0328
 294:   90 91 29 03     lds     r25, 0x0329
 298:   89 2b           or      r24, r25
 29a:   09 f0           breq    .+2             ; 0x29e <__vector_9+0x184>
 29c:   5d 9a           sbi     0x0b, 5 ; 11
   if(*ledValPtr++) LEDON(LED3_GREEN);
 29e:   80 91 2a 03     lds     r24, 0x032A
 2a2:   90 91 2b 03     lds     r25, 0x032B
 2a6:   89 2b           or      r24, r25
 2a8:   09 f0           breq    .+2             ; 0x2ac <__vector_9+0x192>
 2aa:   29 9a           sbi     0x05, 1 ; 5
   if(*ledValPtr++) LEDON(LED3_BLUE);
 2ac:   80 91 2c 03     lds     r24, 0x032C
 2b0:   90 91 2d 03     lds     r25, 0x032D
 2b4:   89 2b           or      r24, r25
 2b6:   09 f0           breq    .+2             ; 0x2ba <__vector_9+0x1a0>
 2b8:   5f 9a           sbi     0x0b, 7 ; 11

   //LED 4
   if(*ledValPtr++) LEDON(LED4_RED);
 2ba:   80 91 2e 03     lds     r24, 0x032E
 2be:   90 91 2f 03     lds     r25, 0x032F
 2c2:   89 2b           or      r24, r25
 2c4:   09 f0           breq    .+2             ; 0x2c8 <__vector_9+0x1ae>
 2c6:   2d 9a           sbi     0x05, 5 ; 5
   if(*ledValPtr++) LEDON(LED4_GREEN);
 2c8:   80 91 30 03     lds     r24, 0x0330
 2cc:   90 91 31 03     lds     r25, 0x0331
 2d0:   89 2b           or      r24, r25
 2d2:   09 f0           breq    .+2             ; 0x2d6 <__vector_9+0x1bc>
 2d4:   5b 9a           sbi     0x0b, 3 ; 11
   if(*ledValPtr++) LEDON(LED4_BLUE);
 2d6:   80 91 32 03     lds     r24, 0x0332
 2da:   90 91 33 03     lds     r25, 0x0333
 2de:   89 2b           or      r24, r25
 2e0:	09 f0       	breq	.+2      	; 0x2e4 <__vector_9+0x1ca>
 2e2:	2b 9a       	sbi	0x05, 3	; 5

#if USE_ANALOG_PINS
    //LED 5
    if(*ledValPtr++) LEDON(LED5_RED);
 2e4:	80 91 34 03 	lds	r24, 0x0334
 2e8:	90 91 35 03 	lds	r25, 0x0335
 2ec:	89 2b       	or	r24, r25
 2ee:	09 f0       	breq	.+2      	; 0x2f2 <__vector_9+0x1d8>
 2f0:	42 9a       	sbi	0x08, 2	; 8
    if(*ledValPtr++) LEDON(LED5_GREEN);
 2f2:	80 91 36 03 	lds	r24, 0x0336
 2f6:	90 91 37 03 	lds	r25, 0x0337
 2fa:	89 2b       	or	r24, r25
 2fc:	09 f0       	breq	.+2      	; 0x300 <__vector_9+0x1e6>
 2fe:	41 9a       	sbi	0x08, 1	; 8
    if(*ledValPtr)   LEDON(LED5_BLUE);
 300:	80 91 38 03 	lds	r24, 0x0338
 304:	90 91 39 03 	lds	r25, 0x0339
 308:	89 2b       	or	r24, r25
 30a:	09 f0       	breq	.+2      	; 0x30e <__vector_9+0x1f4>
 30c:	40 9a       	sbi	0x08, 0	; 8
#endif
#endif
  }

  pwmCounter++;
 30e:	80 91 00 01 	lds	r24, 0x0100
 312:	90 91 01 01 	lds	r25, 0x0101
 316:	01 96       	adiw	r24, 0x01	; 1
 318:	90 93 01 01 	sts	0x0101, r25
 31c:	80 93 00 01 	sts	0x0100, r24
  TCNT2 = 0xff; // necessery for not triggering commperator ISR
 320:	8f ef       	ldi	r24, 0xFF	; 255
 322:	80 93 b2 00 	sts	0x00B2, r24
}
 326:	9f 91       	pop	r25
 328:	8f 91       	pop	r24
 32a:	3f 91       	pop	r19
 32c:	2f 91       	pop	r18
 32e:	0f 90       	pop	r0
 330:	0f be       	out	0x3f, r0	; 63
 332:	0f 90       	pop	r0
 334:	1f 90       	pop	r1
 336:	18 95       	reti

00000338 <_GLOBAL__I_setup>:
 338:	0e 94 cf 04 	call	0x99e	; 0x99e <millis>
 33c:	60 93 18 03 	sts	0x0318, r22
 340:	70 93 19 03 	sts	0x0319, r23
 344:	80 93 1a 03 	sts	0x031A, r24
 348:	90 93 1b 03 	sts	0x031B, r25
     }
    }
    return true;
  }
  return false;
}
 34c:   08 95          ret

0000034e <loop>:
  initSerialCommunication();
  initLeds();
  interrupts();              // Enable global interrupts
}
//-----------------------------------------------------------------------------------------------------
void loop()
 34e:   cf 92          push    r12
 350:   df 92          push    r13
 352:   ef 92          push    r14
 354:   ff 92          push    r15
 356:   0f 93          push    r16
 358:   1f 93          push    r17
 35a:   cf 93          push    r28
 35c:   df 93          push    r29
  {
    valueToReturn = false;
  }
#elif (COMMUNICATION_PROTOCOL == AMBLONE_PROTOCOL)
  // AMBLONE Packet (max 4-ch)
  if( (Serial.available() > 0)  && isWaitingForFirstCommandByte() && (Serial.peek() == 0xff) )
 35e:   8d e2          ldi     r24, 0x2D       ; 45
 360:   94 e0          ldi     r25, 0x04       ; 4
 362:   0e 94 55 06    call    0xcaa   ; 0xcaa <_ZN14HardwareSerial9availableEv>
 366:   18 16          cp      r1, r24
 368:   19 06          cpc     r1, r25
 36a:   0c f0          brlt    .+2             ; 0x36e <loop+0x20>
 36c:   e6 c1          rjmp    .+972           ; 0x73a <__stack+0x23b>
  byteCount = 0;
}
//-----------------------------------------------------------------------------------------------------
inline boolean isWaitingForFirstCommandByte()
{
  return (state == S_WAIT_FOR_SF) ? true : false;
 36e:   80 91 49 03    lds     r24, 0x0349
 372:   88 23          and     r24, r24
 374:   09 f0          breq    .+2             ; 0x378 <loop+0x2a>
 376:   e1 c1          rjmp    .+962           ; 0x73a <__stack+0x23b>
  {
    valueToReturn = false;
  }
#elif (COMMUNICATION_PROTOCOL == AMBLONE_PROTOCOL)
  // AMBLONE Packet (max 4-ch)
  if( (Serial.available() > 0)  && isWaitingForFirstCommandByte() && (Serial.peek() == 0xff) )
 378:   8d e2          ldi     r24, 0x2D       ; 45
 37a:   94 e0          ldi     r25, 0x04       ; 4
 37c:   0e 94 66 06    call    0xccc   ; 0xccc <_ZN14HardwareSerial4peekEv>
 380:   8f 3f           cpi     r24, 0xFF       ; 255
 382:   91 05           cpc     r25, r1
 384:   09 f0           breq    .+2             ; 0x388 <loop+0x3a>
 386:   d9 c1           rjmp    .+946           ; 0x73a <__stack+0x23b>
  {
    if( Serial.available() >= 19 && Serial.read() == 0xff && Serial.read() == 0x00 )
 388:   8d e2           ldi     r24, 0x2D       ; 45
 38a:   94 e0           ldi     r25, 0x04       ; 4
 38c:   0e 94 55 06     call    0xcaa   ; 0xcaa <_ZN14HardwareSerial9availableEv>
 390:   43 97           sbiw    r24, 0x13       ; 19
 392:   0c f4           brge    .+2             ; 0x396 <loop+0x48>
 394:   50 c1           rjmp    .+672           ; 0x636 <__stack+0x137>
 396:   8d e2           ldi     r24, 0x2D       ; 45
 398:   94 e0           ldi     r25, 0x04       ; 4
 39a:   0e 94 86 06     call    0xd0c   ; 0xd0c <_ZN14HardwareSerial4readEv>
 39e:   8f 3f           cpi     r24, 0xFF       ; 255
 3a0:   91 05           cpc     r25, r1
 3a2:   09 f0           breq    .+2             ; 0x3a6 <loop+0x58>
 3a4:   48 c1           rjmp    .+656           ; 0x636 <__stack+0x137>
 3a6:   8d e2           ldi     r24, 0x2D       ; 45
 3a8:   94 e0           ldi     r25, 0x04       ; 4
 3aa:   0e 94 86 06       call    0xd0c   ; 0xd0c <_ZN14HardwareSerial4readEv>
 3ae:   89 2b           or      r24, r25
 3b0:   09 f0           breq    .+2             ; 0x3b4 <loop+0x66>
 3b2:   41 c1           rjmp    .+642           ; 0x636 <__stack+0x137>
    {
      switch ( Serial.read() )
 3b4:   8d e2           ldi     r24, 0x2D       ; 45
 3b6:   94 e0           ldi     r25, 0x04       ; 4
 3b8:   0e 94 86 06     call    0xd0c   ; 0xd0c <_ZN14HardwareSerial4readEv>
 3bc:   8a 3a           cpi     r24, 0xAA       ; 170
 3be:   91 05           cpc     r25, r1
 3c0:   d9 f0           breq    .+54            ; 0x3f8 <loop+0xaa>
 3c2:   8c 3c           cpi     r24, 0xCC       ; 204
 3c4:   91 05           cpc     r25, r1
 3c6:   09 f0           breq    .+2             ; 0x3ca <loop+0x7c>
 3c8:   36 c1           rjmp    .+620           ; 0x636 <__stack+0x137>
    *(settings + i++) = EEPROM.read(i);
}
//-----------------------------------------------------------------------------------------------------
inline boolean getUpdateSettingsCommand()
{
  if( Serial.read() == UNOLIGHT_VERSION )
 3ca:   8d e2           ldi     r24, 0x2D       ; 45
 3cc:   94 e0           ldi     r25, 0x04       ; 4
 3ce:   0e 94 86 06     call    0xd0c   ; 0xd0c <_ZN14HardwareSerial4readEv>
 3d2:   02 97           sbiw    r24, 0x02       ; 2
 3d4:   09 f0           breq    .+2             ; 0x3d8 <loop+0x8a>
 3d6:   2f c1           rjmp    .+606           ; 0x636 <__stack+0x137>
 3d8:   ca e3           ldi     r28, 0x3A       ; 58
 3da:   d3 e0           ldi     r29, 0x03       ; 3
  {
    for(byte i = 0; i<NUMBER_OF_SETTINGS; i++)
    {
     byte newSetting = Serial.read();
 3dc:   8d e2           ldi     r24, 0x2D       ; 45
 3de:   94 e0           ldi     r25, 0x04       ; 4
 3e0:   0e 94 86 06     call    0xd0c   ; 0xd0c <_ZN14HardwareSerial4readEv>
 3e4:   98 2f           mov     r25, r24
     if( *(settings + i) != newSetting )
 3e6:   88 81           ld      r24, Y
 3e8:   89 13           cpse    r24, r25
     {
       *(settings + i) = newSetting;
 3ea:   98 83           st      Y, r25
 3ec:   21 96           adiw    r28, 0x01       ; 1
//-------------------------------------------------------------------------------------------------------
inline boolean getUpdateSettingsCommand()
{
  if( Serial.read() == UNOLIGHT_VERSION )
  {
    for(byte i = 0; i<NUMBER_OF_SETTINGS; i++)
 3ee:   83 e0           ldi     r24, 0x03       ; 3
 3f0:   c9 34           cpi     r28, 0x49       ; 73
 3f2:   d8 07           cpc     r29, r24
 3f4:   99 f7           brne    .-26            ; 0x3dc <loop+0x8e>
 3f6:   aa c1           rjmp    .+852           ; 0x74c <__stack+0x24d>
  Serial.begin(SERIAL_BAUD_RATE);  // Setting serial speed correct for used protocol
}

static inline boolean sendInfo()
{
  Serial.write((byte)0xff);
 3f8:   8d e2           ldi     r24, 0x2D       ; 45
 3fa:   94 e0           ldi     r25, 0x04       ; 4
 3fc:   6f ef           ldi     r22, 0xFF       ; 255
 3fe:   0e 94 c1 06     call    0xd82   ; 0xd82 <_ZN14HardwareSerial5writeEh>
  Serial.write((byte)0x00);
 402:   8d e2           ldi     r24, 0x2D       ; 45
 404:   94 e0           ldi     r25, 0x04       ; 4
 406:   60 e0           ldi     r22, 0x00       ; 0
 408:   0e 94 c1 06     call    0xd82   ; 0xd82 <_ZN14HardwareSerial5writeEh>
  Serial.write((byte)0xaa);
 40c:   8d e2           ldi     r24, 0x2D       ; 45
 40e:   94 e0           ldi     r25, 0x04       ; 4
 410:   6a ea           ldi     r22, 0xAA       ; 170
 412:   0e 94 c1 06     call    0xd82   ; 0xd82 <_ZN14HardwareSerial5writeEh>
  Serial.write((byte)UNOLIGHT_VERSION);
 416:   8d e2           ldi     r24, 0x2D       ; 45
 418:   94 e0           ldi     r25, 0x04       ; 4
 41a:   62 e0           ldi     r22, 0x02       ; 2
 41c:   0e 94 c1 06     call    0xd82   ; 0xd82 <_ZN14HardwareSerial5writeEh>
 420:   ca e3           ldi     r28, 0x3A       ; 58
 422:   d3 e0           ldi     r29, 0x03       ; 3

  for(byte i = 0; i<NUMBER_OF_SETTINGS; i++)
    Serial.write((byte)*(settings + i));
 424:   8d e2           ldi     r24, 0x2D       ; 45
 426:   94 e0           ldi     r25, 0x04       ; 4
 428:   69 91           ld      r22, Y+
 42a:   0e 94 c1 06     call    0xd82   ; 0xd82 <_ZN14HardwareSerial5writeEh>
  Serial.write((byte)0xff);
  Serial.write((byte)0x00);
  Serial.write((byte)0xaa);
  Serial.write((byte)UNOLIGHT_VERSION);

  for(byte i = 0; i<NUMBER_OF_SETTINGS; i++)
 42e:   93 e0           ldi     r25, 0x03       ; 3
 430:   c9 34           cpi     r28, 0x49       ; 73
 432:   d9 07           cpc     r29, r25
 434:   b9 f7           brne    .-18            ; 0x424 <loop+0xd6>
 436:   8a c1           rjmp    .+788           ; 0x74c <__stack+0x24d>
// The amount of RGB channels we are using
static byte channelMode;
//-----------------------------------------------------------------------------------------------------
boolean inline getAmbloneCommand()
{
  recv = Serial.read();
 438:   8d e2           ldi     r24, 0x2D       ; 45
 43a:   94 e0           ldi     r25, 0x04       ; 4
 43c:   0e 94 86 06     call    0xd0c   ; 0xd0c <_ZN14HardwareSerial4readEv>
 440:   98 2f           mov     r25, r24
 442:   80 93 4a 03     sts     0x034A, r24

  switch (state) {
 446:   20 91 49 03     lds     r18, 0x0349
 44a:   21 30           cpi     r18, 0x01       ; 1
 44c:   19 f1           breq    .+70            ; 0x494 <loop+0x146>
 44e:   21 30           cpi     r18, 0x01       ; 1
 450:   20 f0           brcs    .+8             ; 0x45a <loop+0x10c>
 452:   22 30           cpi     r18, 0x02       ; 2
 454:   09 f0           breq    .+2             ; 0x458 <loop+0x10a>
 456:   ef c0           rjmp    .+478           ; 0x636 <__stack+0x137>
 458:   e1 c0           rjmp    .+450           ; 0x61c <__stack+0x11d>
    case S_WAIT_FOR_SF:
      // =============================== Wait for start flag state
      switch (recv) {
 45a:   82 3f           cpi     r24, 0xF2       ; 242
 45c:   81 f0           breq    .+32            ; 0x47e <loop+0x130>
 45e:   83 3f           cpi     r24, 0xF3       ; 243
 460:   20 f4           brcc    .+8             ; 0x46a <loop+0x11c>
 462:   81 3f           cpi     r24, 0xF1       ; 241
 464:   09 f0           breq    .+2             ; 0x468 <loop+0x11a>
 466:   e7 c0           rjmp    .+462           ; 0x636 <__stack+0x137>
 468:   06 c0           rjmp    .+12            ; 0x476 <loop+0x128>
 46a:   83 3f           cpi     r24, 0xF3       ; 243
 46c:   79 f0           breq    .+30            ; 0x48c <loop+0x13e>
 46e:   84 3f           cpi     r24, 0xF4       ; 244
 470:   09 f0           breq    .+2             ; 0x474 <loop+0x126>
 472:   e1 c0           rjmp    .+450           ; 0x636 <__stack+0x137>
 474:   0d c0           rjmp    .+26            ; 0x490 <loop+0x142>
        case C_SF1:
```

```
        // Start flag for 1-channel mode
        channelMode = 1;
 476:   81 e0           ldi     r24, 0x01       ; 1
 478:   80 93 4b 03     sts     0x034B, r24
 47c:   04 c0           rjmp    .+8             ; 0x486 <loop+0x138>
        state = S_RECV_RGB;
        byteCount = 0;
        return false;
      case C_SF2:
        // Start flag for 2-channel mode
        channelMode = 2;
 47e:   82 e0           ldi     r24, 0x02       ; 2
 480:   80 93 4b 03     sts     0x034B, r24
        state = S_RECV_RGB;
 484:   81 e0           ldi     r24, 0x01       ; 1
 486:   80 93 49 03     sts     0x0349, r24
 48a:   b8 c0           rjmp    .+368           ; 0x5fc <__stack+0xfd>
        byteCount = 0;
        return false;
      case C_SF3:
        // Start flag for 3-channel mode
        channelMode = 3;
 48c:   83 e0           ldi     r24, 0x03       ; 3
 48e:   f8 cf           rjmp    .-16            ; 0x480 <loop+0x132>
        state = S_RECV_RGB;
        byteCount = 0;
        return false;
      case C_SF4:
        // Start flag for 4-channel mode
        channelMode = 4;
 490:   84 e0           ldi     r24, 0x04       ; 4
 492:   f6 cf           rjmp    .-20            ; 0x480 <loop+0x132>
        return false;
    }
    break;
  case S_RECV_RGB:
    // =============================== RGB Data reception state
    switch (recv) {
 494:   81 3f           cpi     r24, 0xF1       ; 241
 496:   81 f0           breq    .+32            ; 0x4b8 <loop+0x16a>
 498:   82 3f           cpi     r24, 0xF2       ; 242
 49a:   30 f4           brcc    .+12            ; 0x4a8 <loop+0x15a>
 49c:   83 33           cpi     r24, 0x33       ; 51
 49e:   c9 f0           breq    .+50            ; 0x4d2 <loop+0x184>
 4a0:   89 39           cpi     r24, 0x99       ; 153
 4a2:   09 f0           breq    .+2             ; 0x4a6 <loop+0x158>
 4a4:   b0 c0           rjmp    .+352           ; 0x606 <__stack+0x107>
 4a6:   ad c0           rjmp    .+346           ; 0x602 <__stack+0x103>
 4a8:   83 3f           cpi     r24, 0xF3       ; 243
 4aa:   59 f0           breq    .+22            ; 0x4c2 <loop+0x174>
 4ac:   83 3f           cpi     r24, 0xF3       ; 243
 4ae:   38 f0           brcs    .+14            ; 0x4be <loop+0x170>
 4b0:   84 3f           cpi     r24, 0xF4       ; 244
 4b2:   09 f0           breq    .+2             ; 0x4b6 <loop+0x168>
 4b4:	a8 c0       	rjmp	.+336    	; 0x606 <__stack+0x107>
 4b6:	07 c0       	rjmp	.+14     	; 0x4c6 <loop+0x178>
      case C_SF1:
        // Start flag for 1-channel mode
        channelMode = 1;
 4b8:	20 93 4b 03 	sts	0x034B, r18
 4bc:	07 c0       	rjmp	.+14     	; 0x4cc <loop+0x17e>
        state = S_RECV_RGB;
        byteCount = 0;
        return false;
      case C_SF2:
        // Start flag for 2-channel mode
        channelMode = 2;
 4be:	82 e0       	ldi	r24, 0x02	; 2
 4c0:	03 c0       	rjmp	.+6      	; 0x4c8 <loop+0x17a>
        state = S_RECV_RGB;
        byteCount = 0;
        return false;
      case C_SF3:
        // Start flag for 3-channel mode
        channelMode = 3;
 4c2:	83 e0       	ldi	r24, 0x03	; 3
 4c4:	01 c0       	rjmp	.+2      	; 0x4c8 <loop+0x17a>
        state = S_RECV_RGB;
        byteCount = 0;
        return false;
      case C_SF4:
        // Start flag for 4-channel mode
        channelMode = 4;
 4c6:	84 e0       	ldi	r24, 0x04	; 4
 4c8:	80 93 4b 03 	sts	0x034B, r24
        state = S_RECV_RGB;
 4cc:	20 93 49 03 	sts	0x0349, r18
 4d0:	95 c0       	rjmp	.+298    	; 0x5fc <__stack+0xfd>
        byteCount = 0;
        return false;
      case C_END:
        // End Flag
        // For each channel, we should have received 3 values. If so, we have received a valid packet
        if (byteCount == channelMode * 3) {
 4d2:	50 91 4c 03 	lds	r21, 0x034C
 4d6:	85 2f       	mov	r24, r21
 4d8:	90 e0       	ldi	r25, 0x00	; 0
 4da:	20 91 4b 03 	lds	r18, 0x034B
 4de:	43 e0       	ldi	r20, 0x03	; 3
 4e0:	24 9f       	mul	r18, r20
 4e2:	90 01       	movw	r18, r0
 4e4:	11 24       	eor	r1, r1
 4e6:	82 17       	cp	r24, r18
 4e8:	93 07       	cpc	r25, r19
 4ea:	09 f0       	breq	.+2      	; 0x4ee <loop+0x1a0>
 4ec:	85 c0       	rjmp	.+266    	; 0x5f8 <__stack+0xf9>
 4ee:	80 e0       	ldi	r24, 0x00	; 0
 4f0:	90 e0       	ldi	r25, 0x00	; 0
```

```
static inline void loadNewLedValues(byte numOfValues)
{
  byte i = 0;
  while( i < NUM_OF_LEDS )
  {
    *(incomingData + i) = (i < numOfValues) ? *(payload + i) : 0;
 4f2:   85 17       cp      r24, r21
 4f4:   10 f0       brcs    .+4             ; 0x4fa <loop+0x1ac>
 4f6:   20 e0       ldi     r18, 0x00       ; 0
 4f8:   04 c0       rjmp    .+8             ; 0x502 <__stack+0x3>
 4fa:   fc 01       movw    r30, r24
 4fc:   e3 5b       subi    r30, 0xB3       ; 179
 4fe:   fc 4f       sbci    r31, 0xFC       ; 252
 500:   20 81       ld      r18, Z
 502:   fc 01       movw    r30, r24
 504:   e3 59       subi    r30, 0x93       ; 147
 506:   fc 4f       sbci    r31, 0xFC       ; 252
 508:   20 83       st      Z, r18
 50a:   01 96       adiw    r24, 0x01       ; 1
}
//-----------------------------------------------------------------------------------------------------
static inline void loadNewLedValues(byte numOfValues)
{
  byte i = 0;
  while( i < NUM_OF_LEDS )
 50c:   8f 30       cpi     r24, 0x0F       ; 15
 50e:   91 05       cpc     r25, r1
 510:   81 f7       brne    .-32            ; 0x4f2 <loop+0x1a4>
  {
    *(incomingData + i) = (i < numOfValues) ? *(payload + i) : 0;
    i++;
  }

  word *ledChannelAndColorPointer = (*isSmoothEnabled) ? &ledChannelsNew[0][0] : &ledChannels[0][0];
 512:   80 91 3e 03     lds     r24, 0x033E
 516:   88 23       and     r24, r24
 518:   19 f4       brne    .+6             ; 0x520 <__stack+0x21>
 51a:   cc e1       ldi     r28, 0x1C       ; 28
 51c:   d3 e0       ldi     r29, 0x03       ; 3
 51e:   02 c0       rjmp    .+4             ; 0x524 <__stack+0x25>
 520:   cd e7       ldi     r28, 0x7D       ; 125
 522:   d3 e0       ldi     r29, 0x03       ; 3
  byte channel = 0;

  noInterrupts();
 524:   f8 94       cli
  if( *useGammaTable )
 526:   80 91 3a 03     lds     r24, 0x033A
 52a:   88 23       and     r24, r24
 52c:   a9 f1       breq    .+106           ; 0x598 <__stack+0x99>
 52e:   43 e0       ldi     r20, 0x03       ; 3
 530:   51 e0       ldi     r21, 0x01       ; 1
  {
    while( channel < NUM_OF_RGB_LEDS )
```

```
   {
     byte *incomingValuePointer = incomingData + *(channelOrder + channel++) * 3;
 532:   63 e0         ldi     r22, 0x03       ; 3
 534:   fa 01         movw    r30, r20
 536:   a1 91         ld      r26, Z+
 538:   af 01         movw    r20, r30
 53a:   a6 9f         mul     r26, r22
 53c:   d0 01         movw    r26, r0
 53e:   11 24         eor     r1, r1
 540:   a3 59         subi    r26, 0x93       ; 147
 542:   bc 4f         sbci    r27, 0xFC       ; 252

     *ledChannelAndColorPointer++ = *(gammaTable + *incomingValuePointer++); // red
 544:   fd 01             movw    r30, r26
 546:   81 91         ld      r24, Z+
 548:   9f 01         movw    r18, r30
 54a:   e8 2f         mov     r30, r24
 54c:   f0 e0         ldi     r31, 0x00       ; 0
 54e:   ee 0f         add     r30, r30
 550:   ff 1f         adc     r31, r31
 552:   e8 5f         subi    r30, 0xF8       ; 248
 554:   fe 4f         sbci    r31, 0xFE       ; 254
 556:   80 81         ld      r24, Z
 558:   91 81         ldd     r25, Z+1        ; 0x01
 55a:   99 83         std     Y+1, r25        ; 0x01
 55c:   88 83         st      Y, r24
     *ledChannelAndColorPointer++ = *(gammaTable + *incomingValuePointer++); // green
 55e:   11 96         adiw    r26, 0x01       ; 1
 560:   ec 91         ld      r30, X
 562:   f0 e0         ldi     r31, 0x00       ; 0
 564:   ee 0f         add     r30, r30
 566:   ff 1f         adc     r31, r31
 568:   e8 5f         subi    r30, 0xF8       ; 248
 56a:   fe 4f         sbci    r31, 0xFE       ; 254
 56c:   80 81         ld      r24, Z
 56e:   91 81         ldd     r25, Z+1        ; 0x01
 570:   9b 83         std     Y+3, r25        ; 0x03
 572:   8a 83         std     Y+2, r24        ; 0x02
     *ledChannelAndColorPointer++ = *(gammaTable + *incomingValuePointer);   // blue
 574:   d9 01         movw    r26, r18
 576:   11 96         adiw    r26, 0x01       ; 1
 578:   ec 91         ld      r30, X
 57a:   f0 e0         ldi     r31, 0x00       ; 0
 57c:   ee 0f         add     r30, r30
 57e:   ff 1f         adc     r31, r31
 580:   e8 5f         subi    r30, 0xF8       ; 248
 582:   fe 4f         sbci    r31, 0xFE       ; 254
 584:   80 81         ld      r24, Z
 586:   91 81         ldd     r25, Z+1        ; 0x01
 588:   9d 83         std     Y+5, r25        ; 0x05
 58a:   8c 83         std     Y+4, r24        ; 0x04
  initSerialCommunication();
  initLeds();
  interrupts();             // Enable global interrupts
```

```
}
//----------------------------------------------------------------------------------------------------------
void loop()
 58c:   26 96          adiw    r28, 0x06       ; 6
  byte channel = 0;

  noInterrupts();
  if( *useGammaTable )
  {
    while( channel < NUM_OF_RGB_LEDS )
 58e:   b1 e0          ldi     r27, 0x01       ; 1
 590:   48 30          cpi     r20, 0x08       ; 8
 592:   5b 07          cpc     r21, r27
 594:   79 f6          brne    .-98            ; 0x534 <__stack+0x35>
 596:   2a c0          rjmp    .+84            ; 0x5ec <__stack+0xed>
 598:   23 e0          ldi     r18, 0x03       ; 3
 59a:   31 e0          ldi     r19, 0x01       ; 1
  }
  else
  {
    while( channel < NUM_OF_RGB_LEDS )
    {
      byte *incomingValuePointer = incomingData + *(channelOrder + channel++) * 3;
 59c:   43 e0          ldi     r20, 0x03       ; 3
 59e:   d9 01          movw    r26, r18
 5a0:   ed 91          ld      r30, X+
 5a2:   9d 01          movw    r18, r26
 5a4:   e4 9f          mul     r30, r20
 5a6:   f0 01          movw    r30, r0
 5a8:   11 24          eor     r1, r1
 5aa:   e3 59          subi    r30, 0x93       ; 147
 5ac:   fc 4f          sbci    r31, 0xFC       ; 252

      *ledChannelAndColorPointer++ = *incomingValuePointer++ * 4; // red
 5ae:   df 01          movw    r26, r30
 5b0:   8d 91          ld      r24, X+
 5b2:   90 e0          ldi     r25, 0x00       ; 0
 5b4:   88 0f          add     r24, r24
 5b6:   99 1f          adc     r25, r25
 5b8:   88 0f          add     r24, r24
 5ba:   99 1f          adc     r25, r25
 5bc:   99 83          std     Y+1, r25        ; 0x01
 5be:   88 83          st      Y, r24
      *ledChannelAndColorPointer++ = *incomingValuePointer++ * 4; // green
 5c0:   81 81          ldd     r24, Z+1        ; 0x01
 5c2:   90 e0          ldi     r25, 0x00       ; 0
 5c4:   88 0f          add     r24, r24
 5c6:   99 1f          adc     r25, r25
 5c8:   88 0f          add     r24, r24
 5ca:   99 1f          adc     r25, r25
 5cc:   9b 83          std     Y+3, r25        ; 0x03
 5ce:   8a 83          std     Y+2, r24        ; 0x02
      *ledChannelAndColorPointer++ = *incomingValuePointer * 4;   // blue
 5d0:   11 96          adiw    r26, 0x01       ; 1
 5d2:   8c 91           ld      r24, X
 5d4:   90 e0           ldi     r25, 0x00       ; 0
 5d6:   88 0f           add     r24, r24
 5d8:   99 1f           adc     r25, r25
 5da:   88 0f           add     r24, r24
 5dc:   99 1f           adc     r25, r25
 5de:   9d 83           std     Y+5, r25        ; 0x05
 5e0:   8c 83           std     Y+4, r24        ; 0x04
  initSerialCommunication();
  initLeds();
  interrupts();               // Enable global interrupts
}
//----------------------------------------------------------------------------------------------------
void loop()
 5e2:   26 96           adiw    r28, 0x06       ; 6
      *ledChannelAndColorPointer++ = *(gammaTable + *incomingValuePointer);   // blue
    }
  }
  else
  {
    while( channel < NUM_OF_RGB_LEDS )
 5e4:   b1 e0           ldi     r27, 0x01       ; 1
 5e6:   28 30           cpi     r18, 0x08       ; 8
 5e8:   3b 07           cpc     r19, r27
 5ea:   c9 f6           brne    .-78            ; 0x59e <__stack+0x9f>
      *ledChannelAndColorPointer++ = *incomingValuePointer++ * 4; // red
      *ledChannelAndColorPointer++ = *incomingValuePointer++ * 4; // green
      *ledChannelAndColorPointer++ = *incomingValuePointer * 4;   // blue
    }
  }
  interrupts();
 5ec:   78 94           sei
        case C_END:
          // End Flag
          // For each channel, we should have received 3 values. If so, we have received a valid packet
          if (byteCount == channelMode * 3) {
            loadNewLedValues(byteCount);
            state = S_WAIT_FOR_SF;
 5ee:   10 92 49 03     sts     0x0349, r1
            byteCount = 0;
 5f2:   10 92 4c 03     sts     0x034C, r1
 5f6:   aa c0           rjmp    .+340           ; 0x74c <__stack+0x24d>
            return true; // <------------------------ TRUE IS RETURNED
          }
          else {
            // Something's gone wrong: restart
            state = S_WAIT_FOR_SF;
 5f8:   10 92 49 03     sts     0x0349, r1
            byteCount = 0;
 5fc:   10 92 4c 03     sts     0x034C, r1
 600:   1a c0           rjmp    .+52            ; 0x636 <__stack+0x137>
            return false;
          }
        case C_ESC:
```

```
        // Escape character
        state = S_RECV_RGB_ESC;
 602:   82 e0           ldi     r24, 0x02       ; 2
 604:   16 c0           rjmp    .+44            ; 0x632 <__stack+0x133>
        return false;
      default:
        // The character received wasn't a flag, so store it as an RGB value
        *(payload + byteCount++) = recv;
 606:   80 91 4c 03     lds     r24, 0x034C
 60a:   e8 2f           mov     r30, r24
 60c:   f0 e0           ldi     r31, 0x00       ; 0
 60e:   e3 5b           subi    r30, 0xB3       ; 179
 610:   fc 4f           sbci    r31, 0xFC       ; 252
 612:   90 83           st      Z, r25
 614:   8f 5f           subi    r24, 0xFF       ; 255
 616:   80 93 4c 03     sts     0x034C, r24
 61a:   0d c0           rjmp    .+26            ; 0x636 <__stack+0x137>
        return false;
    }
    case S_RECV_RGB_ESC:
      // ================================== RGB Escaped data reception state
      // Store the value in the payload, no matter what it is
      *(payload + byteCount++) = recv;
 61c:   80 91 4c 03     lds     r24, 0x034C
 620:   e8 2f           mov     r30, r24
 622:   f0 e0           ldi     r31, 0x00       ; 0
 624:   e3 5b           subi    r30, 0xB3       ; 179
 626:   fc 4f           sbci    r31, 0xFC       ; 252
 628:   90 83           st      Z, r25
 62a:   8f 5f           subi    r24, 0xFF       ; 255
 62c:   80 93 4c 03     sts     0x034C, r24
      state = S_RECV_RGB;
 630:   81 e0           ldi     r24, 0x01       ; 1
 632:   80 93 49 03     sts     0x0349, r24
   if(ledsOff)
     ledsOff = false;
  }
  else
  {
   if(!ledsOff)
 636:   80 91 02 01     lds     r24, 0x0102
 63a:   88 23           and     r24, r24
 63c:   71 f5           brne    .+92            ; 0x69a <__stack+0x19b>
 63e:   03 c0           rjmp    .+6             ; 0x646 <__stack+0x147>
void loop()
{
  if( getCommand() )
  {
   if(ledsOff)
     ledsOff = false;
 640:   10 92 02 01     sts     0x0102, r1
 644:   2a c0           rjmp    .+84            ; 0x69a <__stack+0x19b>
  return valueToReturn;
}
```

```
//-------------------------------------------------------------------------------------------------------------
inline word timeElapsedSinceLastCommand()
{
  return ((millis() - timeOfLastTransmition) / 1000);
 646:   0e 94 cf 04     call    0x99e   ; 0x99e <millis>
}
//-------------------------------------------------------------------------------------------------------------
static inline void turnOffLedsIfNeeded()
{
   // Turn off LEDs if no data for defined period of time
   if( (timeElapsedSinceLastCommand() > (*idleTimeLimit + 1)) && !ledsOff )
 64a:   20 91 18 03     lds     r18, 0x0318
 64e:   30 91 19 03     lds     r19, 0x0319
 652:   40 91 1a 03     lds     r20, 0x031A
 656:   50 91 1b 03     lds     r21, 0x031B
 65a:   62 1b           sub     r22, r18
 65c:   73 0b           sbc     r23, r19
 65e:   84 0b           sbc     r24, r20
 660:   95 0b           sbc     r25, r21
 662:   28 ee           ldi     r18, 0xE8       ; 232
 664:   33 e0           ldi     r19, 0x03       ; 3
 666:   40 e0           ldi     r20, 0x00       ; 0
 668:   50 e0           ldi     r21, 0x00       ; 0
 66a:   0e 94 ad 07     call    0xf5a   ; 0xf5a <__udivmodsi4>
 66e:   80 91 3d 03     lds     r24, 0x033D
 672:   90 e0           ldi     r25, 0x00       ; 0
 674:   01 96           adiw    r24, 0x01       ; 1
 676:   82 17           cp      r24, r18
 678:   93 07           cpc     r25, r19
 67a:   78 f4           brcc    .+30            ; 0x69a <__stack+0x19b>
 67c:   80 91 02 01     lds     r24, 0x0102
 680:   88 23           and     r24, r24
 682:   59 f4           brne    .+22            ; 0x69a <__stack+0x19b>
 684:   ed e7           ldi     r30, 0x7D       ; 125
 686:   f3 e0           ldi     r31, 0x03       ; 3
static inline void fadeOutLEDs()
{
   word * const ledChannelsNewPtr = &ledChannelsNew[0][0];

   for(byte led = 0; led < NUM_OF_LEDS; led++)
     *(ledChannelsNewPtr + led ) = 0;
 688:   11 92           st      Z+, r1
 68a:   11 92           st      Z+, r1
//-------------------------------------------------------------------------------------------------------------
static inline void fadeOutLEDs()
{
   word * const ledChannelsNewPtr = &ledChannelsNew[0][0];

   for(byte led = 0; led < NUM_OF_LEDS; led++)
 68c:   83 e0           ldi     r24, 0x03       ; 3
 68e:   eb 39           cpi     r30, 0x9B       ; 155
 690:   f8 07           cpc     r31, r24
 692:   d1 f7           brne    .-12            ; 0x688 <__stack+0x189>
     *(ledChannelsNewPtr + led ) = 0;
  ledsOff = true;
694:   81 e0          ldi     r24, 0x01       ; 1
696:   80 93 02 01    sts     0x0102, r24
 {
   if(!ledsOff)
     turnOffLedsIfNeeded();
 }

 if( *isSmoothEnabled || (!*isSmoothEnabled && ledsOff) )
69a:   80 91 3e 03    lds     r24, 0x033E
69e:   88 23          and     r24, r24
6a0:   29 f4          brne    .+10            ; 0x6ac <__stack+0x1ad>
6a2:   80 91 02 01    lds     r24, 0x0102
6a6:   88 23          and     r24, r24
6a8:   09 f4          brne    .+2             ; 0x6ac <__stack+0x1ad>
6aa:   60 c0          rjmp    .+192           ; 0x76c <__stack+0x26d>
   smooth(*smoothAmount);
6ac:   40 91 3f 03    lds     r20, 0x033F
  {
    difference = abs((int)(*ledChannelsNewPtr - *ledChannelsPtr));

    if( difference )
    {
      epsilon = ( (*ledChannelsPtr > 128) && (difference > (smoothAmount * 2)) ) ? (difference / smoothAmount) : 1;
6b0:   c4 2e          mov     r12, r20
6b2:   dd 24          eor     r13, r13
6b4:   cc 0c          add     r12, r12
6b6:   dd 1c          adc     r13, r13
6b8:   0c e1          ldi     r16, 0x1C       ; 28
6ba:   13 e0          ldi     r17, 0x03       ; 3
6bc:   82 e0          ldi     r24, 0x02       ; 2
6be:   e8 2e          mov     r14, r24
6c0:   f1 2c          mov     r15, r1
6c2:   e0 0e          add     r14, r16
6c4:   f1 1e          adc     r15, r17
6c6:   cf e7          ldi     r28, 0x7F       ; 127
6c8:   d3 e0          ldi     r29, 0x03       ; 3
  byte i = NUM_OF_LEDS - 1;
  word difference;

  do
  {
    difference = abs((int)(*ledChannelsNewPtr - *ledChannelsPtr));
6ca:   d8 01          movw    r26, r16
6cc:   ed 91          ld      r30, X+
6ce:   fc 91          ld      r31, X
6d0:   3a 91          ld      r19, -Y
6d2:   2a 91          ld      r18, -Y
6d4:   22 96          adiw    r28, 0x02       ; 2
6d6:   2e 1b          sub     r18, r30
6d8:   3f 0b          sbc     r19, r31
6da:   c9 01          movw    r24, r18
6dc:   37 ff          sbrs    r19, 7
```

```
6de:   04 c0          rjmp    .+8            ; 0x6e8 <__stack+0x1e9>
6e0:   88 27          eor     r24, r24
6e2:   99 27          eor     r25, r25
6e4:   82 1b          sub     r24, r18
6e6:   93 0b          sbc     r25, r19

    if( difference )
6e8:   00 97          sbiw    r24, 0x00      ; 0
6ea:   d1 f0          breq    .+52           ; 0x720 <__stack+0x221>
    {
      epsilon = ( (*ledChannelsPtr > 128) && (difference > (smoothAmount * 2)) ) ? (difference / smoothAmount) : 1;
6ec:   e1 38          cpi     r30, 0x81      ; 129
6ee:   f1 05          cpc     r31, r1
6f0:   40 f0          brcs    .+16           ; 0x702 <__stack+0x203>
6f2:   c8 16          cp      r12, r24
6f4:   d9 06          cpc     r13, r25
6f6:   28 f4          brcc    .+10           ; 0x702 <__stack+0x203>
6f8:   64 2f          mov     r22, r20
6fa:   70 e0          ldi     r23, 0x00      ; 0
6fc:   0e 94 86 07     call    0xf0c   ; 0xf0c <__udivmodhi4>
700:   01 c0          rjmp    .+2            ; 0x704 <__stack+0x205>
702:   61 e0          ldi     r22, 0x01      ; 1
704:   70 e0          ldi     r23, 0x00      ; 0

      if( (int)(*ledChannelsNewPtr++ - *ledChannelsPtr) < 0  )
706:   37 ff          sbrs    r19, 7
708:   06 c0          rjmp    .+12           ; 0x716 <__stack+0x217>
        *ledChannelsPtr++ -= epsilon;
70a:   e6 1b          sub     r30, r22
70c:   f7 0b          sbc     r31, r23
70e:   d8 01          movw    r26, r16
710:   ed 93          st      X+, r30
712:   fc 93          st      X, r31
714:   05 c0          rjmp    .+10           ; 0x720 <__stack+0x221>
      else
        *ledChannelsPtr++ += epsilon;
716:   6e 0f          add     r22, r30
718:   7f 1f          adc     r23, r31
71a:   f8 01          movw    r30, r16
71c:   71 83          std     Z+1, r23       ; 0x01
71e:   60 83          st      Z, r22
720:   0e 5f          subi    r16, 0xFE      ; 254
722:   1f 4f          sbci    r17, 0xFF      ; 255
724:   82 e0          ldi     r24, 0x02      ; 2
726:   90 e0          ldi     r25, 0x00      ; 0
728:   e8 0e          add     r14, r24
72a:   f9 1e          adc     r15, r25
72c:   22 96          adiw    r28, 0x02      ; 2
  word *ledChannelsPtr  = &ledChannels[0][0];
  word *ledChannelsNewPtr = &ledChannelsNew[0][0];
  byte i = NUM_OF_LEDS - 1;
  word difference;

  do
```

```
 72e:   9c e3           ldi     r25, 0x3C       ; 60
 730:   e9 16           cp      r14, r25
 732:   93 e0           ldi     r25, 0x03       ; 3
 734:   f9 06           cpc     r15, r25
 736:   49 f6           brne    .-110           ; 0x6ca <__stack+0x1cb>
 738:   19 c0           rjmp    .+50            ; 0x76c <__stack+0x26d>
    else
      valueToReturn = false;
  }
  else
  {
     valueToReturn = ( Serial.available() > 0 ) ? getAmbloneCommand() : false;
 73a:   8d e2           ldi     r24, 0x2D       ; 45
 73c:   94 e0           ldi     r25, 0x04       ; 4
 73e:   0e 94 55 06     call    0xcaa   ; 0xcaa <_ZN14HardwareSerial9availableEv>
 742:   18 16           cp      r1, r24
 744:   19 06           cpc     r1, r25
 746:   0c f4           brge    .+2             ; 0x74a <__stack+0x24b>
 748:   77 ce           rjmp    .-786           ; 0x438 <loop+0xea>
 74a:   75 cf           rjmp    .-278           ; 0x636 <__stack+0x137>
#else
#error : You need to define COMMUNICATION_PROTOCOL as equal to AMBLONE_PROTOCOL or
ATMOLIGHT_PROTOCOL
#endif

  if( valueToReturn == true )  // if correct command recived
    timeOfLastTransmition = millis();
 74c:   0e 94 cf 04     call    0x99e   ; 0x99e <millis>
 750:   60 93 18 03     sts     0x0318, r22
 754:   70 93 19 03     sts     0x0319, r23
 758:   80 93 1a 03     sts     0x031A, r24
 75c:   90 93 1b 03     sts     0x031B, r25
//-----------------------------------------------------------------------------------------------
void loop()
{
  if( getCommand() )
  {
    if(ledsOff)
 760:   80 91 02 01     lds     r24, 0x0102
 764:   88 23           and     r24, r24
 766:   09 f0           breq    .+2             ; 0x76a <__stack+0x26b>
 768:   6b cf           rjmp    .-298           ; 0x640 <__stack+0x141>
 76a:   97 cf           rjmp    .-210           ; 0x69a <__stack+0x19b>
      turnOffLedsIfNeeded();
  }

  if( *isSmoothEnabled || (!*isSmoothEnabled && ledsOff) )
    smooth(*smoothAmount);
}
 76c:   df 91           pop     r29
 76e:   cf 91           pop     r28
 770:   1f 91           pop     r17
 772:   0f 91           pop     r16
 774:   ff 90           pop     r15
 776:   ef 90           pop     r14
 778:   df 90           pop     r13
 77a:   cf 90           pop     r12
 77c:   08 95           ret

0000077e <setup>:
static inline boolean sendInfo();
inline boolean getCommand();
inline word timeElapsedSinceLastCommand();
inline void initSettings();
inline boolean getUpdateSettingsCommand();
void setup()
 77e:   cf 93           push    r28
 780:   df 93           push    r29
{
  noInterrupts();             // Disable global interrupts
 782:   f8 94           cli
{
  byte settingValue;
  byte i = 0;

  // If no settings in EEPROM for this version of UnoLight load defaults.
  if( EEPROM.read(NUMBER_OF_SETTINGS) != UNOLIGHT_VERSION )
 784:   8b e9           ldi     r24, 0x9B       ; 155
 786:   93 e0           ldi     r25, 0x03       ; 3
 788:   6f e0           ldi     r22, 0x0F       ; 15
 78a:   70 e0           ldi     r23, 0x00       ; 0
 78c:   0e 94 74 04     call    0x8e8   ; 0x8e8 <_ZN11EEPROMClass4readEi>
 790:   82 30           cpi     r24, 0x02       ; 2
 792:   49 f1           breq    .+82            ; 0x7e6 <setup+0x68>
 794:   c0 e0           ldi     r28, 0x00       ; 0
 796:   d0 e0           ldi     r29, 0x00       ; 0
  {
    while( i < NUMBER_OF_SETTINGS )
    {
      switch( i )
 798:   c3 30           cpi     r28, 0x03       ; 3
 79a:   91 f0           breq    .+36            ; 0x7c0 <setup+0x42>
 79c:   c4 30           cpi     r28, 0x04       ; 4
 79e:   28 f4           brcc    .+10            ; 0x7aa <setup+0x2c>
 7a0:   cc 23           and     r28, r28
 7a2:   41 f0           breq    .+16            ; 0x7b4 <setup+0x36>
 7a4:   c2 30           cpi     r28, 0x02       ; 2
 7a6:   41 f4           brne    .+16            ; 0x7b8 <setup+0x3a>
 7a8:   09 c0           rjmp    .+18            ; 0x7bc <setup+0x3e>
 7aa:   c4 30           cpi     r28, 0x04       ; 4
 7ac:   19 f0           breq    .+6             ; 0x7b4 <setup+0x36>
 7ae:   c5 30           cpi     r28, 0x05       ; 5
 7b0:   19 f4           brne    .+6             ; 0x7b8 <setup+0x3a>
 7b2:   08 c0           rjmp    .+16            ; 0x7c4 <setup+0x46>
 7b4:   41 e0           ldi     r20, 0x01       ; 1
 7b6:   07 c0           rjmp    .+14            ; 0x7c6 <setup+0x48>
 7b8:   40 e0           ldi     r20, 0x00       ; 0
 7ba:   05 c0           rjmp    .+10            ; 0x7c6 <setup+0x48>
 7bc:   40 e8           ldi     r20, 0x80       ; 128
 7be:   03 c0           rjmp    .+6             ; 0x7c6 <setup+0x48>
 7c0:   4a e0           ldi     r20, 0x0A       ; 10
 7c2:   01 c0           rjmp    .+2             ; 0x7c6 <setup+0x48>
 7c4:   44 e1           ldi     r20, 0x14       ; 20
          settingValue = 20;
          break;
        default:
          settingValue = 0;
      }
      EEPROM.write(i++, settingValue);
 7c6:   8b e9           ldi     r24, 0x9B       ; 155
 7c8:   93 e0           ldi     r25, 0x03       ; 3
 7ca:   be 01           movw    r22, r28
 7cc:   0e 94 7b 04     call    0x8f6   ; 0x8f6 <_ZN11EEPROMClass5writeEih>
 7d0:   21 96           adiw    r28, 0x01       ; 1
  byte i = 0;

  // If no settings in EEPROM for this version of UnoLight load defaults.
  if( EEPROM.read(NUMBER_OF_SETTINGS) != UNOLIGHT_VERSION )
  {
    while( i < NUMBER_OF_SETTINGS )
 7d2:   cf 30           cpi     r28, 0x0F       ; 15
 7d4:   d1 05           cpc     r29, r1
 7d6:   01 f7           brne    .-64            ; 0x798 <setup+0x1a>
        default:
          settingValue = 0;
      }
      EEPROM.write(i++, settingValue);
    }
    EEPROM.write(i, UNOLIGHT_VERSION);
 7d8:   8b e9           ldi     r24, 0x9B       ; 155
 7da:   93 e0           ldi     r25, 0x03       ; 3
 7dc:   6f e0           ldi     r22, 0x0F       ; 15
 7de:   70 e0           ldi     r23, 0x00       ; 0
 7e0:   42 e0           ldi     r20, 0x02       ; 2
 7e2:   0e 94 7b 04     call    0x8f6   ; 0x8f6 <_ZN11EEPROMClass5writeEih>
 7e6:   c0 e0           ldi     r28, 0x00       ; 0
 7e8:   d0 e0           ldi     r29, 0x00       ; 0
  }

  i = 0;
  while( i < NUMBER_OF_SETTINGS )
    *(settings + i++) = EEPROM.read(i);
 7ea:   8b e9           ldi     r24, 0x9B       ; 155
 7ec:   93 e0           ldi     r25, 0x03       ; 3
 7ee:   be 01           movw    r22, r28
 7f0:   0e 94 74 04     call    0x8e8   ; 0x8e8 <_ZN11EEPROMClass4readEi>
 7f4:   fe 01           movw    r30, r28
 7f6:   e6 5c           subi    r30, 0xC6       ; 198
 7f8:   fc 4f           sbci    r31, 0xFC       ; 252
 7fa:   80 83           st      Z, r24
 7fc:   21 96           adiw    r28, 0x01       ; 1
    }
    EEPROM.write(i, UNOLIGHT_VERSION);
   }

  i = 0;
  while( i < NUMBER_OF_SETTINGS )
 7fe:   cf 30           cpi     r28, 0x0F       ; 15
 800:   d1 05           cpc     r29, r1
 802:   99 f7           brne    .-26            ; 0x7ea <setup+0x6c>
*/

//-------------------------------------------------------------------------------------------------
static inline void initSerialCommunication()
{
  Serial.begin(SERIAL_BAUD_RATE);  // Setting serial speed correct for used protocol
 804:   8d e2           ldi     r24, 0x2D       ; 45
 806:   94 e0           ldi     r25, 0x04       ; 4
 808:   40 e0           ldi     r20, 0x00       ; 0
 80a:   58 ee           ldi     r21, 0xE8       ; 232
 80c:   63 e0           ldi     r22, 0x03       ; 3
 80e:   70 e0           ldi     r23, 0x00       ; 0
 810:   0e 94 cf 05     call    0xb9e   ; 0xb9e <_ZN14HardwareSerial5beginEm>
   but it is easy to understend
   for everybody.
*/
static inline void initLeds()
{
  pinMode(2, OUTPUT);
 814:   82 e0           ldi     r24, 0x02       ; 2
 816:   61 e0           ldi     r22, 0x01       ; 1
 818:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(3, OUTPUT);
 81c:   83 e0           ldi     r24, 0x03       ; 3
 81e:   61 e0           ldi     r22, 0x01       ; 1
 820:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(4, OUTPUT);
 824:   84 e0           ldi     r24, 0x04       ; 4
 826:   61 e0           ldi     r22, 0x01       ; 1
 828:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(5, OUTPUT);
 82c:   85 e0           ldi     r24, 0x05       ; 5
 82e:   61 e0           ldi     r22, 0x01       ; 1
 830:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(6, OUTPUT);
 834:   86 e0           ldi     r24, 0x06       ; 6
 836:   61 e0           ldi     r22, 0x01       ; 1
 838:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(7, OUTPUT);
 83c:   87 e0           ldi     r24, 0x07       ; 7
 83e:   61 e0           ldi     r22, 0x01       ; 1
 840:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(8, OUTPUT);
 844:   88 e0           ldi     r24, 0x08       ; 8
 846:   61 e0           ldi     r22, 0x01       ; 1
 848:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
```

```
  pinMode(9, OUTPUT);
 84c:   89 e0           ldi     r24, 0x09       ; 9
 84e:   61 e0           ldi     r22, 0x01       ; 1
 850:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(10, OUTPUT);
 854:   8a e0           ldi     r24, 0x0A       ; 10
 856:   61 e0           ldi     r22, 0x01       ; 1
 858:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(11, OUTPUT);
 85c:   8b e0           ldi     r24, 0x0B       ; 11
 85e:   61 e0           ldi     r22, 0x01       ; 1
 860:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(12, OUTPUT);
 864:   8c e0           ldi     r24, 0x0C       ; 12
 866:   61 e0           ldi     r22, 0x01       ; 1
 868:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(13, OUTPUT);
 86c:   8d e0           ldi     r24, 0x0D       ; 13
 86e:   61 e0           ldi     r22, 0x01       ; 1
 870:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
#if USE_ANALOG_PINS
  pinMode(A0, OUTPUT);
 874:   8e e0           ldi     r24, 0x0E       ; 14
 876:   61 e0           ldi     r22, 0x01       ; 1
 878:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(A1, OUTPUT);
 87c:   8f e0           ldi     r24, 0x0F       ; 15
 87e:   61 e0           ldi     r22, 0x01       ; 1
 880:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
  pinMode(A2, OUTPUT);
 884:   80 e1           ldi     r24, 0x10       ; 16
 886:   61 e0           ldi     r22, 0x01       ; 1
 888:   0e 94 18 05     call    0xa30   ; 0xa30 <pinMode>
//---------------------------------------------------------------------------------------------------
static inline void initTimer2()
{
#if (defined __AVR_ATmega168__ || defined __AVR_ATmega328P__)
  // Initialize TIMER2
  BIT_CLR(TCCR2B,WGM22);
 88c:   80 91 b1 00     lds     r24, 0x00B1
 890:   87 7f           andi    r24, 0xF7       ; 247
 892:   80 93 b1 00     sts     0x00B1, r24
  BIT_SET(TCCR2A,WGM21);
 896:   80 91 b0 00     lds     r24, 0x00B0
 89a:   82 60           ori     r24, 0x02       ; 2
 89c:   80 93 b0 00     sts     0x00B0, r24
  BIT_CLR(TCCR2A,WGM20);   // CTC PWM
 8a0:   80 91 b0 00     lds     r24, 0x00B0
 8a4:   8e 7f           andi    r24, 0xFE       ; 254
 8a6:   80 93 b0 00     sts     0x00B0, r24

  BIT_CLR(TCCR2B,CS22);
 8aa:   80 91 b1 00     lds     r24, 0x00B1
 8ae:   8b 7f           andi    r24, 0xFB       ; 251
 8b0:   80 93 b1 00     sts     0x00B1, r24
  BIT_SET(TCCR2B,CS21);
 8b4:   80 91 b1 00     lds     r24, 0x00B1
 8b8:   82 60           ori     r24, 0x02       ; 2
 8ba:   80 93 b1 00     sts     0x00B1, r24
  BIT_SET(TCCR2B,CS20);    // Timer Prescaler 32, results in 488 Hz led frequency
 8be:   80 91 b1 00     lds     r24, 0x00B1
 8c2:   81 60           ori     r24, 0x01       ; 1
 8c4:   80 93 b1 00     sts     0x00B1, r24

  BIT_SET(TIMSK2,TOIE2);  // Enable Overflow Interrupt
 8c8:   80 91 70 00     lds     r24, 0x0070
 8cc:   81 60           ori     r24, 0x01       ; 1
 8ce:   80 93 70 00     sts     0x0070, r24
  TCNT2 = 0x00;
 8d2:   10 92 b2 00     sts     0x00B2, r1

  BIT_CLR(TIMSK1,TOIE1);  // turn off interrupt for Timer1 (not used interrupt)
 8d6:   80 91 6f 00     lds     r24, 0x006F
 8da:   8e 7f           andi    r24, 0xFE       ; 254
 8dc:   80 93 6f 00     sts     0x006F, r24
{
  noInterrupts();             // Disable global interrupts
  initSettings();
  initSerialCommunication();
  initLeds();
  interrupts();               // Enable global interrupts
 8e0:   78 94           sei
}
 8e2:   df 91           pop     r29
 8e4:   cf 91           pop     r28
 8e6:   08 95           ret

000008e8 <_ZN11EEPROMClass4readEi>:
/** \ingroup avr_eeprom
    Read one byte from EEPROM address \a __p.
 */
__ATTR_PURE__ static __inline__ uint8_t eeprom_read_byte (const uint8_t *__p)
{
    do {} while (!eeprom_is_ready ());
 8e8:   f9 99           sbic    0x1f, 1 ; 31
 8ea:   fe cf           rjmp    .-4             ; 0x8e8 <_ZN11EEPROMClass4readEi>
#if E2END <= 0xFF
    EEARL = (uint8_t)__p;
#else
    EEAR = (uint16_t)__p;
 8ec:   72 bd           out     0x22, r23       ; 34
 8ee:   61 bd           out     0x21, r22       ; 33
        "/* END EEPROM READ CRITICAL SECTION */ \n\t"
        : "=r" (__result)
        : "i" (_SFR_IO_ADDR(EECR)),
          "i" (EERE),
          "i" (_SFR_IO_ADDR(EEDR))
    );
 8f0:   f8 9a           sbi     0x1f, 0 ; 31
 8f2:   80 b5           in      r24, 0x20       ; 32
 ****************************************************************************/

uint8_t EEPROMClass::read(int address)
{
        return eeprom_read_byte((unsigned char *) address);
}
 8f4:   08 95           ret

000008f6 <_ZN11EEPROMClass5writeEih>:
/** \ingroup avr_eeprom
    Write a byte \a __value to EEPROM address \a __p.
 */
static __inline__ void eeprom_write_byte (uint8_t *__p, uint8_t __value)
{
    do {} while (!eeprom_is_ready ());
 8f6:   f9 99           sbic    0x1f, 1 ; 31
 8f8:   fe cf           rjmp    .-4             ; 0x8f6 <_ZN11EEPROMClass5writeEih>

#if     defined(EEPM0) && defined(EEPM1)
    EECR = 0;           /* Set programming mode: erase and write.       */
 8fa:   1f ba           out     0x1f, r1        ; 31
#endif

#if     E2END <= 0xFF
    EEARL = (unsigned)__p;
#else
    EEAR = (unsigned)__p;
 8fc:   72 bd           out     0x22, r23       ; 34
 8fe:   61 bd           out     0x21, r22       ; 33
#endif
    EEDR = __value;
 900:   40 bd           out     0x20, r20       ; 32
        : [__eecr]  "i" (_SFR_IO_ADDR(EECR)),
          [__sreg]  "i" (_SFR_IO_ADDR(SREG)),
          [__eemwe] "i" (EEMWE),
          [__eewe]  "i" (EEWE)
        : "r0"
    );
 902:   0f b6           in      r0, 0x3f        ; 63
 904:   f8 94           cli
 906:   fa 9a           sbi     0x1f, 2 ; 31
 908:   f9 9a           sbi     0x1f, 1 ; 31
 90a:   0f be           out     0x3f, r0        ; 63

void EEPROMClass::write(int address, uint8_t value)
{
        eeprom_write_byte((unsigned char *) address, value);
}
 90c:   08 95           ret

0000090e <__vector_16>:
volatile unsigned long timer0_overflow_count = 0;
volatile unsigned long timer0_millis = 0;
static unsigned char timer0_fract = 0;

SIGNAL(TIMER0_OVF_vect)
{
 90e:   1f 92           push    r1
 910:   0f 92           push    r0
 912:   0f b6           in      r0, 0x3f        ; 63
 914:   0f 92           push    r0
 916:   11 24           eor     r1, r1
 918:   2f 93           push    r18
 91a:   3f 93           push    r19
 91c:   8f 93           push    r24
 91e:   9f 93               push    r25
 920:   af 93           push    r26
 922:   bf 93           push    r27
        // copy these to local variables so they can be stored in registers
        // (volatile variables must be read from memory on every access)
        unsigned long m = timer0_millis;
 924:   80 91 a0 03     lds     r24, 0x03A0
 928:   90 91 a1 03     lds     r25, 0x03A1
 92c:   a0 91 a2 03     lds     r26, 0x03A2
 930:   b0 91 a3 03     lds     r27, 0x03A3
        unsigned char f = timer0_fract;
 934:   30 91 a4 03     lds     r19, 0x03A4

        m += MILLIS_INC;
 938:   01 96           adiw    r24, 0x01       ; 1
 93a:   a1 1d           adc     r26, r1
 93c:   b1 1d           adc     r27, r1
        f += FRACT_INC;
 93e:   23 2f           mov     r18, r19
 940:   2d 5f           subi    r18, 0xFD       ; 253
        if (f >= FRACT_MAX) {
 942:   2d 37           cpi     r18, 0x7D       ; 125
 944:   20 f0           brcs    .+8             ; 0x94e <__vector_16+0x40>
                f -= FRACT_MAX;
 946:   2d 57           subi    r18, 0x7D       ; 125
                m += 1;
 948:   01 96           adiw    r24, 0x01       ; 1
 94a:   a1 1d           adc     r26, r1
 94c:   b1 1d           adc     r27, r1
        }

        timer0_fract = f;
 94e:   20 93 a4 03     sts     0x03A4, r18
        timer0_millis = m;
 952:   80 93 a0 03     sts     0x03A0, r24
 956:   90 93 a1 03     sts     0x03A1, r25
 95a:   a0 93 a2 03     sts     0x03A2, r26
 95e:   b0 93 a3 03     sts     0x03A3, r27
        timer0_overflow_count++;
 962:   80 91 9c 03     lds     r24, 0x039C
 966:   90 91 9d 03     lds     r25, 0x039D
 96a:   a0 91 9e 03     lds     r26, 0x039E
 96e:   b0 91 9f 03     lds     r27, 0x039F
 972:   01 96           adiw    r24, 0x01       ; 1
 974:   a1 1d           adc     r26, r1
 976:   b1 1d           adc     r27, r1
 978:   80 93 9c 03     sts     0x039C, r24
 97c:   90 93 9d 03     sts     0x039D, r25
 980:   a0 93 9e 03     sts     0x039E, r26
 984:   b0 93 9f 03     sts     0x039F, r27
}
 988:   bf 91           pop     r27
 98a:   af 91           pop     r26
 98c:   9f 91           pop     r25
 98e:   8f 91           pop     r24
 990:   3f 91           pop     r19
 992:   2f 91           pop     r18
 994:   0f 90           pop     r0
 996:   0f be           out     0x3f, r0        ; 63
 998:   0f 90           pop     r0
 99a:   1f 90           pop     r1
 99c:   18 95           reti

0000099e <millis>:

unsigned long millis()
{
	unsigned long m;
	uint8_t oldSREG = SREG;
 99e:   8f b7           in      r24, 0x3f       ; 63

	// disable interrupts while we read timer0_millis or we might get an
	// inconsistent value (e.g. in the middle of a write to timer0_millis)
	cli();
 9a0:   f8 94           cli
	m = timer0_millis;
 9a2:   20 91 a0 03     lds     r18, 0x03A0
 9a6:   30 91 a1 03     lds     r19, 0x03A1
 9aa:   40 91 a2 03     lds     r20, 0x03A2
 9ae:   50 91 a3 03     lds     r21, 0x03A3
	SREG = oldSREG;
 9b2:   8f bf           out     0x3f, r24       ; 63

	return m;
}
 9b4:   b9 01           movw    r22, r18
 9b6:   ca 01           movw    r24, r20
 9b8:   08 95           ret

000009ba <init>:

void init()
{
	// this needs to be called before setup() or some functions won't
	// work there
	sei();
 9ba:	78 94       	sei

	// on the ATmega168, timer 0 is also used for fast hardware pwm
	// (using phase-correct PWM would mean that timer 0 overflowed half as often
	// resulting in different millis() behavior on the ATmega8 and ATmega168)
#if defined(TCCR0A) && defined(WGM01)
	sbi(TCCR0A, WGM01);
 9bc:	84 b5       	in	r24, 0x24	; 36
 9be:	82 60       	ori	r24, 0x02	; 2
 9c0:	84 bd       	out	0x24, r24	; 36
	sbi(TCCR0A, WGM00);
 9c2:	84 b5       	in	r24, 0x24	; 36
 9c4:	81 60       	ori	r24, 0x01	; 1
 9c6:	84 bd       	out	0x24, r24	; 36
	// this combination is for the standard atmega8
	sbi(TCCR0, CS01);
	sbi(TCCR0, CS00);
#elif defined(TCCR0B) && defined(CS01) && defined(CS00)
	// this combination is for the standard 168/328/1280/2560
	sbi(TCCR0B, CS01);
 9c8:	85 b5       	in	r24, 0x25	; 37
 9ca:	82 60       	ori	r24, 0x02	; 2
 9cc:	85 bd       	out	0x25, r24	; 37
	sbi(TCCR0B, CS00);
 9ce:	85 b5       	in	r24, 0x25	; 37
 9d0:	81 60       	ori	r24, 0x01	; 1
 9d2:	85 bd       	out	0x25, r24	; 37

	// enable timer 0 overflow interrupt
#if defined(TIMSK) && defined(TOIE0)
	sbi(TIMSK, TOIE0);
#elif defined(TIMSK0) && defined(TOIE0)
	sbi(TIMSK0, TOIE0);
 9d4:	ee e6       	ldi	r30, 0x6E	; 110
 9d6:	f0 e0       	ldi	r31, 0x00	; 0
 9d8:	80 81       	ld	r24, Z
 9da:	81 60       	ori	r24, 0x01	; 1
 9dc:	80 83       	st	Z, r24
	// this is better for motors as it ensures an even waveform
	// note, however, that fast pwm mode can achieve a frequency of up
	// 8 MHz (with a 16 MHz clock) at 50% duty cycle

#if defined(TCCR1B) && defined(CS11) && defined(CS10)
	TCCR1B = 0;
 9de:	e1 e8       	ldi	r30, 0x81	; 129
 9e0:	f0 e0       	ldi	r31, 0x00	; 0
 9e2:	10 82       	st	Z, r1

	// set timer 1 prescale factor to 64
	sbi(TCCR1B, CS11);
 9e4:	80 81       	ld	r24, Z
 9e6:	82 60       	ori	r24, 0x02	; 2
 9e8:	80 83       	st	Z, r24
```

```
        sbi(TCCR1B, CS10);
 9ea:   80 81           ld      r24, Z
 9ec:   81 60           ori     r24, 0x01       ; 1
 9ee:   80 83           st      Z, r24
        sbi(TCCR1, CS11);
        sbi(TCCR1, CS10);
#endif
        // put timer 1 in 8-bit phase correct pwm mode
#if defined(TCCR1A) && defined(WGM10)
        sbi(TCCR1A, WGM10);
 9f0:   e0 e8           ldi     r30, 0x80       ; 128
 9f2:   f0 e0           ldi     r31, 0x00       ; 0
 9f4:   80 81           ld      r24, Z
 9f6:   81 60           ori     r24, 0x01       ; 1
 9f8:   80 83           st      Z, r24

        // set timer 2 prescale factor to 64
#if defined(TCCR2) && defined(CS22)
        sbi(TCCR2, CS22);
#elif defined(TCCR2B) && defined(CS22)
        sbi(TCCR2B, CS22);
 9fa:   e1 eb           ldi     r30, 0xB1       ; 177
 9fc:   f0 e0           ldi     r31, 0x00       ; 0
 9fe:   80 81           ld      r24, Z
 a00:   84 60           ori     r24, 0x04       ; 4
 a02:   80 83           st      Z, r24

        // configure timer 2 for phase correct pwm (8-bit)
#if defined(TCCR2) && defined(WGM20)
        sbi(TCCR2, WGM20);
#elif defined(TCCR2A) && defined(WGM20)
        sbi(TCCR2A, WGM20);
 a04:   e0 eb           ldi     r30, 0xB0       ; 176
 a06:   f0 e0           ldi     r31, 0x00       ; 0
 a08:   80 81           ld      r24, Z
 a0a:   81 60           ori     r24, 0x01       ; 1
 a0c:   80 83           st      Z, r24
#if defined(ADCSRA)
        // set a2d prescale factor to 128
        // 16 MHz / 128 = 125 KHz, inside the desired 50-200 KHz range.
        // XXX: this will not work properly for other clock speeds, and
        // this code should use F_CPU to determine the prescale factor.
        sbi(ADCSRA, ADPS2);
 a0e:   ea e7           ldi     r30, 0x7A       ; 122
 a10:   f0 e0           ldi     r31, 0x00       ; 0
 a12:   80 81           ld      r24, Z
 a14:   84 60           ori     r24, 0x04       ; 4
 a16:   80 83           st      Z, r24
        sbi(ADCSRA, ADPS1);
 a18:   80 81           ld      r24, Z
 a1a:   82 60           ori     r24, 0x02       ; 2
 a1c:   80 83           st      Z, r24
        sbi(ADCSRA, ADPS0);
 a1e:   80 81           ld      r24, Z
 a20:   81 60           ori     r24, 0x01       ; 1
 a22:   80 83           st      Z, r24

        // enable a2d conversions
        sbi(ADCSRA, ADEN);
 a24:   80 81           ld      r24, Z
 a26:   80 68           ori     r24, 0x80       ; 128
 a28:   80 83           st      Z, r24
        // here so they can be used as normal digital i/o; they will be
        // reconnected in Serial.begin()
#if defined(UCSRB)
        UCSRB = 0;
#elif defined(UCSR0B)
        UCSR0B = 0;
 a2a:   10 92 c1 00     sts     0x00C1, r1
#endif
}
 a2e:   08 95           ret

00000a30 <pinMode>:
#include "wiring_private.h"
#include "pins_arduino.h"

void pinMode(uint8_t pin, uint8_t mode)
{
        uint8_t bit = digitalPinToBitMask(pin);
 a30:   48 2f           mov     r20, r24
 a32:   50 e0           ldi     r21, 0x00       ; 0
 a34:   ca 01           movw    r24, r20
 a36:   86 56           subi    r24, 0x66       ; 102
 a38:   9f 4f           sbci    r25, 0xFF       ; 255
 a3a:   fc 01           movw    r30, r24
 a3c:   24 91           lpm     r18, Z+
        uint8_t port = digitalPinToPort(pin);
 a3e:   4a 57           subi    r20, 0x7A       ; 122
 a40:   5f 4f           sbci    r21, 0xFF       ; 255
 a42:   fa 01           movw    r30, r20
 a44:   84 91           lpm     r24, Z+
        volatile uint8_t *reg;

        if (port == NOT_A_PIN) return;
 a46:   88 23           and     r24, r24
 a48:   c1 f0           breq    .+48            ; 0xa7a <pinMode+0x4a>

        // JWS: can I let the optimizer do this?
        reg = portModeRegister(port);
 a4a:   e8 2f           mov     r30, r24
 a4c:   f0 e0           ldi     r31, 0x00       ; 0
 a4e:   ee 0f           add     r30, r30
 a50:   ff 1f           adc     r31, r31
 a52:   e8 59           subi    r30, 0x98       ; 152
 a54:   ff 4f           sbci    r31, 0xFF       ; 255
 a56:   a5 91           lpm     r26, Z+
 a58:   b4 91           lpm     r27, Z+
        if (mode == INPUT) {
 a5a:   66 23           and     r22, r22
 a5c:   41 f4           brne    .+16            ; 0xa6e <pinMode+0x3e>
                uint8_t oldSREG = SREG;
 a5e:   9f b7           in      r25, 0x3f       ; 63
                cli();
 a60:   f8 94           cli
                *reg &= ~bit;
 a62:   8c 91           ld      r24, X
 a64:   20 95           com     r18
 a66:   82 23           and     r24, r18
 a68:   8c 93           st      X, r24
                SREG = oldSREG;
 a6a:   9f bf           out     0x3f, r25       ; 63
 a6c:   08 95           ret
        } else {
                uint8_t oldSREG = SREG;
 a6e:   9f b7           in      r25, 0x3f       ; 63
                cli();
 a70:   f8 94           cli
                *reg |= bit;
 a72:   8c 91           ld      r24, X
 a74:   82 2b           or      r24, r18
 a76:   8c 93           st      X, r24
                SREG = oldSREG;
 a78:   9f bf           out     0x3f, r25       ; 63
 a7a:   08 95           ret

00000a7c <_Z11serialEventv>:
    !defined(SIG_UART0_RECV) && !defined(USART0_RX_vect) && \
        !defined(SIG_UART_RECV)
  #error Don't know what the Data Received vector is called for the first UART
#else
  void serialEvent() __attribute__((weak));
  void serialEvent() {}
 a7c:   08 95           ret

00000a7e <__vector_18>:
  #define serialEvent_implemented
#if defined(USART_RX_vect)
  SIGNAL(USART_RX_vect)
 a7e:   1f 92           push    r1
 a80:   0f 92           push    r0
 a82:   0f b6           in      r0, 0x3f        ; 63
 a84:   0f 92           push    r0
 a86:   11 24           eor     r1, r1
 a88:   2f 93           push    r18
 a8a:   3f 93           push    r19
 a8c:   4f 93           push    r20
 a8e:   8f 93           push    r24
 a90:   9f 93           push    r25
 a92:   ef 93           push    r30
 a94:   ff 93           push    r31
```

```
#elif defined(SIG_UART_RECV)
  SIGNAL(SIG_UART_RECV)
#endif
  {
  #if defined(UDR0)
    unsigned char c  =  UDR0;
 a96:   40 91 c6 00     lds     r20, 0x00C6
  ring_buffer tx_buffer3  =  { { 0 }, 0, 0 };
#endif

inline void store_char(unsigned char c, ring_buffer *buffer)
{
  int i = (unsigned int)(buffer->head + 1) % SERIAL_BUFFER_SIZE;
 a9a:   20 91 e5 03     lds     r18, 0x03E5
 a9e:   30 91 e6 03     lds     r19, 0x03E6
 aa2:   2f 5f           subi    r18, 0xFF       ; 255
 aa4:   3f 4f           sbci    r19, 0xFF       ; 255
 aa6:   2f 73           andi    r18, 0x3F       ; 63
 aa8:   30 70           andi    r19, 0x00       ; 0

  // if we should be storing the received character into the location
  // just before the tail (meaning that the head would advance to the
  // current location of the tail), we're about to overflow the buffer
  // and so we don't write the character or advance the head.
  if (i != buffer->tail) {
 aaa:   80 91 e7 03     lds     r24, 0x03E7
 aae:   90 91 e8 03     lds     r25, 0x03E8
 ab2:   28 17           cp      r18, r24
 ab4:   39 07           cpc     r19, r25
 ab6:   59 f0           breq    .+22            ; 0xace <__vector_18+0x50>
    buffer->buffer[buffer->head] = c;
 ab8:   e0 91 e5 03     lds     r30, 0x03E5
 abc:   f0 91 e6 03     lds     r31, 0x03E6
 ac0:   eb 55           subi    r30, 0x5B       ; 91
 ac2:   fc 4f           sbci    r31, 0xFC       ; 252
 ac4:   40 83           st      Z, r20
    buffer->head = i;
 ac6:   30 93 e6 03     sts     0x03E6, r19
 aca:   20 93 e5 03     sts     0x03E5, r18
    unsigned char c  =  UDR;
  #else
    #error UDR not defined
  #endif
    store_char(c, &rx_buffer);
  }
 ace:   ff 91           pop     r31
 ad0:   ef 91           pop     r30
 ad2:   9f 91           pop     r25
 ad4:   8f 91           pop     r24
 ad6:   4f 91           pop     r20
 ad8:   3f 91           pop     r19
 ada:   2f 91           pop     r18
 adc:   0f 90           pop     r0
 ade:   0f be           out     0x3f, r0        ; 63
 ae0:   0f 90           pop     r0
 ae2:   1f 90           pop     r1
 ae4:   18 95           reti

00000ae6 <_Z14serialEventRunv>:
  _rx_buffer->head = _rx_buffer->tail;
}

int HardwareSerial::available(void)
{
  return (unsigned int)(SERIAL_BUFFER_SIZE + _rx_buffer->head - _rx_buffer->tail) % SERIAL_BUFFER_SIZE;
 ae6:   e0 91 39 04     lds     r30, 0x0439
 aea:   f0 91 3a 04     lds     r31, 0x043A
 aee:   e0 5c           subi    r30, 0xC0       ; 192
 af0:   ff 4f           sbci    r31, 0xFF       ; 255
 af2:   81 91           ld      r24, Z+
 af4:   91 91           ld      r25, Z+
 af6:   20 81           ld      r18, Z
 af8:   31 81           ldd     r19, Z+1        ; 0x01
#endif

void serialEventRun(void)
{
#ifdef serialEvent_implemented
  if (Serial.available()) serialEvent();
 afa:   82 1b           sub     r24, r18
 afc:   93 0b           sbc     r25, r19
 afe:   8f 73           andi    r24, 0x3F       ; 63
 b00:   90 70           andi    r25, 0x00       ; 0
 b02:   89 2b           or      r24, r25
 b04:   11 f0           breq    .+4             ; 0xb0a <_Z14serialEventRunv+0x24>
 b06:   0e 94 3e 05     call    0xa7c   ; 0xa7c <_Z11serialEventv>
 b0a:   08 95           ret

00000b0c <__vector_19>:
#elif defined(UART_UDRE_vect)
ISR(UART_UDRE_vect)
#elif defined(USART0_UDRE_vect)
ISR(USART0_UDRE_vect)
#elif defined(USART_UDRE_vect)
ISR(USART_UDRE_vect)
 b0c:   1f 92           push    r1
 b0e:   0f 92           push    r0
 b10:   0f b6           in      r0, 0x3f        ; 63
 b12:   0f 92           push    r0
 b14:   11 24           eor     r1, r1
 b16:   2f 93           push    r18
 b18:   3f 93           push    r19
 b1a:   4f 93           push    r20
 b1c:   5f 93           push    r21
 b1e:   6f 93           push    r22
 b20:   7f 93           push    r23
 b22:   8f 93           push    r24
 b24:   9f 93           push    r25
 b26:   af 93           push    r26
 b28:   bf 93           push    r27
 b2a:   ef 93           push    r30
 b2c:   ff 93           push    r31
#endif
{
  if (tx_buffer.head == tx_buffer.tail) {
 b2e:   20 91 29 04     lds     r18, 0x0429
 b32:   30 91 2a 04     lds     r19, 0x042A
 b36:   80 91 2b 04     lds     r24, 0x042B
 b3a:   90 91 2c 04     lds     r25, 0x042C
 b3e:   28 17           cp      r18, r24
 b40:   39 07           cpc     r19, r25
 b42:   31 f4           brne    .+12            ; 0xb50 <__vector_19+0x44>
        // Buffer empty, so disable interrupts
#if defined(UCSR0B)
    cbi(UCSR0B, UDRIE0);
 b44:   80 91 c1 00     lds     r24, 0x00C1
 b48:   8f 7d           andi    r24, 0xDF       ; 223
 b4a:   80 93 c1 00     sts     0x00C1, r24
 b4e:   16 c0           rjmp    .+44            ; 0xb7c <__vector_19+0x70>
    cbi(UCSRB, UDRIE);
#endif
  }
  else {
    // There is more data in the output buffer. Send the next byte
    unsigned char c = tx_buffer.buffer[tx_buffer.tail];
 b50:   e0 91 2b 04     lds     r30, 0x042B
 b54:   f0 91 2c 04     lds     r31, 0x042C
 b58:   e7 51           subi    r30, 0x17       ; 23
 b5a:   fc 4f           sbci    r31, 0xFC       ; 252
 b5c:   40 81           ld      r20, Z
    tx_buffer.tail = (tx_buffer.tail + 1) % SERIAL_BUFFER_SIZE;
 b5e:   80 91 2b 04     lds     r24, 0x042B
 b62:   90 91 2c 04     lds     r25, 0x042C
 b66:   01 96           adiw    r24, 0x01       ; 1
 b68:   60 e4           ldi     r22, 0x40       ; 64
 b6a:   70 e0           ldi     r23, 0x00       ; 0
 b6c:   0e 94 9a 07     call    0xf34   ; 0xf34 <__divmodhi4>
 b70:   90 93 2c 04     sts     0x042C, r25
 b74:   80 93 2b 04     sts     0x042B, r24

  #if defined(UDR0)
    UDR0 = c;
 b78:   40 93 c6 00     sts     0x00C6, r20
    UDR = c;
  #else
    #error UDR not defined
  #endif
  }
}
 b7c:   ff 91           pop     r31
 b7e:   ef 91           pop     r30
 b80:   bf 91           pop     r27
 b82:   af 91           pop     r26
 b84:   9f 91           pop     r25
 b86:   8f 91           pop     r24
 b88:   7f 91           pop     r23
 b8a:   6f 91           pop     r22
 b8c:   5f 91           pop     r21
 b8e:   4f 91           pop     r20
 b90:   3f 91           pop     r19
 b92:   2f 91           pop     r18
 b94:   0f 90           pop     r0
 b96:   0f be           out     0x3f, r0        ; 63
 b98:   0f 90           pop     r0
 b9a:   1f 90           pop     r1
 b9c:   18 95           reti

00000b9e <_ZN14HardwareSerial5beginEm>:
  _u2x = u2x;
}

// Public Methods //////////////////////////////////////////////////////////////

void HardwareSerial::begin(unsigned long baud)
 b9e:   af 92           push    r10
 ba0:   bf 92           push    r11
 ba2:   df 92           push    r13
 ba4:   ef 92           push    r14
 ba6:   ff 92           push    r15
 ba8:   0f 93           push    r16
 baa:   1f 93           push    r17
 bac:   cf 93           push    r28
 bae:   df 93           push    r29
 bb0:   ec 01           movw    r28, r24
 bb2:   7a 01           movw    r14, r20
 bb4:   8b 01           movw    r16, r22
 bb6:   dd 24           eor     r13, r13
 bb8:   40 30           cpi     r20, 0x00       ; 0
 bba:   81 ee           ldi     r24, 0xE1       ; 225
 bbc:   58 07           cpc     r21, r24
 bbe:   80 e0           ldi     r24, 0x00       ; 0
 bc0:   68 07           cpc     r22, r24
 bc2:   80 e0           ldi     r24, 0x00       ; 0
 bc4:   78 07           cpc     r23, r24
 bc6:   11 f0           breq    .+4             ; 0xbcc <_ZN14HardwareSerial5beginEm+0x2e>
 bc8:   dd 24           eor     r13, r13
 bca:   d3 94           inc     r13
#endif

try_again:

  if (use_u2x) {
    *_ucsra = 1 << _u2x;
 bcc:   91 e0           ldi     r25, 0x01       ; 1
 bce:   a9 2e           mov     r10, r25
 bd0:   b1 2c           mov     r11, r1
 bd2:   ec 89           ldd     r30, Y+20       ; 0x14
 bd4:   fd 89           ldd     r31, Y+21       ; 0x15
  }
#endif

try_again:

  if (use_u2x) {
 bd6:   dd 20           and     r13, r13
 bd8:   69 f0           breq    .+26            ; 0xbf4 <_ZN14HardwareSerial5beginEm+0x56>
    *_ucsra = 1 << _u2x;
 bda:   c5 01           movw    r24, r10
 bdc:   0e 8c           ldd     r0, Y+30        ; 0x1e
 bde:   02 c0           rjmp    .+4             ; 0xbe4 <_ZN14HardwareSerial5beginEm+0x46>
 be0:   88 0f           add     r24, r24
 be2:   99 1f           adc     r25, r25
 be4:   0a 94           dec     r0
 be6:   e2 f7           brpl    .-8             ; 0xbe0 <_ZN14HardwareSerial5beginEm+0x42>
 be8:   80 83           st      Z, r24
    baud_setting = (F_CPU / 4 / baud - 1) / 2;
 bea:   60 e0           ldi     r22, 0x00       ; 0
 bec:   79 e0           ldi     r23, 0x09       ; 9
 bee:   8d e3           ldi     r24, 0x3D       ; 61
 bf0:   90 e0           ldi     r25, 0x00       ; 0
 bf2:   05 c0           rjmp    .+10            ; 0xbfe <_ZN14HardwareSerial5beginEm+0x60>
  } else {
    *_ucsra = 0;
 bf4:   10 82           st      Z, r1
    baud_setting = (F_CPU / 8 / baud - 1) / 2;
 bf6:   60 e8           ldi     r22, 0x80       ; 128
 bf8:   74 e8           ldi     r23, 0x84       ; 132
 bfa:   8e e1           ldi     r24, 0x1E       ; 30
 bfc:   90 e0           ldi     r25, 0x00       ; 0
 bfe:   a8 01           movw    r20, r16
 c00:   97 01           movw    r18, r14
 c02:   0e 94 ad 07     call    0xf5a   ; 0xf5a <__udivmodsi4>
 c06:   21 50           subi    r18, 0x01       ; 1
 c08:   30 40           sbci    r19, 0x00       ; 0
 c0a:   40 40           sbci    r20, 0x00       ; 0
 c0c:   50 40           sbci    r21, 0x00       ; 0
 c0e:   56 95           lsr     r21
 c10:   47 95           ror     r20
 c12:   37 95           ror     r19
 c14:   27 95           ror     r18
  }

  if ((baud_setting > 4095) && use_u2x)
 c16:   80 e1           ldi     r24, 0x10       ; 16
 c18:   20 30           cpi     r18, 0x00       ; 0
 c1a:   38 07           cpc     r19, r24
 c1c:   20 f0           brcs    .+8             ; 0xc26 <_ZN14HardwareSerial5beginEm+0x88>
 c1e:   dd 20           and     r13, r13
 c20:   11 f0           breq    .+4             ; 0xc26 <_ZN14HardwareSerial5beginEm+0x88>
 c22:   dd 24           eor     r13, r13
 c24:   d6 cf           rjmp    .-84            ; 0xbd2 <_ZN14HardwareSerial5beginEm+0x34>
    use_u2x = false;
    goto try_again;
   }

   // assign the baud_setting, a.k.a. ubbr (USART Baud Rate Register)
   *_ubrrh = baud_setting >> 8;
 c26:   e8 89           ldd     r30, Y+16       ; 0x10
 c28:   f9 89           ldd     r31, Y+17       ; 0x11
 c2a:   30 83           st      Z, r19
   *_ubrrl = baud_setting;
 c2c:   ea 89           ldd     r30, Y+18       ; 0x12
 c2e:   fb 89           ldd     r31, Y+19       ; 0x13
 c30:   20 83           st      Z, r18

   sbi(*_ucsrb, _rxen);
 c32:   ee 89           ldd     r30, Y+22       ; 0x16
 c34:   ff 89           ldd     r31, Y+23       ; 0x17
 c36:   40 81           ld      r20, Z
 c38:   21 e0           ldi     r18, 0x01       ; 1
 c3a:   30 e0           ldi     r19, 0x00       ; 0
 c3c:   c9 01           movw    r24, r18
 c3e:   0a 8c           ldd     r0, Y+26        ; 0x1a
 c40:   02 c0           rjmp    .+4             ; 0xc46 <_ZN14HardwareSerial5beginEm+0xa8>
 c42:   88 0f           add     r24, r24
 c44:   99 1f           adc     r25, r25
 c46:   0a 94           dec     r0
 c48:   e2 f7           brpl    .-8             ; 0xc42 <_ZN14HardwareSerial5beginEm+0xa4>
 c4a:   48 2b           or      r20, r24
 c4c:   40 83           st      Z, r20
   sbi(*_ucsrb, _txen);
 c4e:   ee 89           ldd     r30, Y+22       ; 0x16
 c50:   ff 89           ldd     r31, Y+23       ; 0x17
 c52:   40 81           ld      r20, Z
 c54:   c9 01           movw    r24, r18
 c56:   0b 8c           ldd     r0, Y+27        ; 0x1b
 c58:   02 c0           rjmp    .+4             ; 0xc5e <_ZN14HardwareSerial5beginEm+0xc0>
 c5a:   88 0f           add     r24, r24
 c5c:   99 1f           adc     r25, r25
 c5e:   0a 94           dec     r0
 c60:   e2 f7           brpl    .-8             ; 0xc5a <_ZN14HardwareSerial5beginEm+0xbc>
 c62:   48 2b           or      r20, r24
 c64:   40 83           st      Z, r20
   sbi(*_ucsrb, _rxcie);
 c66:   ee 89           ldd     r30, Y+22       ; 0x16
 c68:   ff 89           ldd     r31, Y+23       ; 0x17
 c6a:   40 81           ld      r20, Z
 c6c:   c9 01           movw    r24, r18
 c6e:   0c 8c           ldd     r0, Y+28        ; 0x1c
 c70:   02 c0           rjmp    .+4             ; 0xc76 <_ZN14HardwareSerial5beginEm+0xd8>
 c72:   88 0f           add     r24, r24
 c74:   99 1f           adc     r25, r25
 c76:   0a 94           dec     r0
 c78:   e2 f7           brpl    .-8             ; 0xc72 <_ZN14HardwareSerial5beginEm+0xd4>
 c7a:   48 2b           or      r20, r24
 c7c:   40 83           st      Z, r20
  cbi(*_ucsrb, _udrie);
 c7e:   ee 89           ldd     r30, Y+22       ; 0x16
 c80:   ff 89           ldd     r31, Y+23       ; 0x17
 c82:   80 81           ld      r24, Z
 c84:   0d 8c           ldd     r0, Y+29        ; 0x1d
 c86:   02 c0           rjmp    .+4             ; 0xc8c <_ZN14HardwareSerial5beginEm+0xee>
 c88:   22 0f           add     r18, r18
 c8a:   33 1f           adc     r19, r19
 c8c:   0a 94           dec     r0
 c8e:   e2 f7           brpl    .-8             ; 0xc88 <_ZN14HardwareSerial5beginEm+0xea>
 c90:   20 95           com     r18
 c92:   28 23           and     r18, r24
 c94:   20 83           st      Z, r18
}
 c96:   df 91           pop     r29
 c98:   cf 91           pop     r28
 c9a:   1f 91           pop     r17
 c9c:   0f 91           pop     r16
 c9e:   ff 90           pop     r15
 ca0:   ef 90           pop     r14
 ca2:   df 90           pop     r13
 ca4:   bf 90           pop     r11
 ca6:   af 90           pop     r10
 ca8:   08 95           ret

00000caa <_ZN14HardwareSerial9availableEv>:
  _rx_buffer->head = _rx_buffer->tail;
}

int HardwareSerial::available(void)
{
  return (unsigned int)(SERIAL_BUFFER_SIZE + _rx_buffer->head - _rx_buffer->tail) % SERIAL_BUFFER_SIZE;
 caa:   dc 01           movw    r26, r24
 cac:   1c 96           adiw    r26, 0x0c       ; 12
 cae:   ed 91           ld      r30, X+
 cb0:   fc 91           ld      r31, X
 cb2:   1d 97           sbiw    r26, 0x0d       ; 13
 cb4:   e0 5c           subi    r30, 0xC0       ; 192
 cb6:   ff 4f           sbci    r31, 0xFF       ; 255
 cb8:   21 91           ld      r18, Z+
 cba:   31 91           ld      r19, Z+
 cbc:   80 81           ld      r24, Z
 cbe:   91 81           ldd     r25, Z+1        ; 0x01
 cc0:   28 1b           sub     r18, r24
 cc2:   39 0b           sbc     r19, r25
 cc4:   2f 73           andi    r18, 0x3F       ; 63
 cc6:   30 70           andi    r19, 0x00       ; 0
}
 cc8:   c9 01           movw    r24, r18
 cca:   08 95           ret

00000ccc <_ZN14HardwareSerial4peekEv>:
```

```
int HardwareSerial::peek(void)
{
  if (_rx_buffer->head == _rx_buffer->tail) {
 ccc:   dc 01           movw    r26, r24
 cce:   1c 96           adiw    r26, 0x0c       ; 12
 cd0:   ed 91           ld      r30, X+
 cd2:   fc 91           ld      r31, X
 cd4:   1d 97           sbiw    r26, 0x0d       ; 13
 cd6:   e0 5c           subi    r30, 0xC0       ; 192
 cd8:   ff 4f           sbci    r31, 0xFF       ; 255
 cda:   20 81           ld      r18, Z
 cdc:   31 81           ldd     r19, Z+1        ; 0x01
 cde:   e0 54           subi    r30, 0x40       ; 64
 ce0:   f0 40           sbci    r31, 0x00       ; 0
 ce2:   df 01           movw    r26, r30
 ce4:   ae 5b           subi    r26, 0xBE       ; 190
 ce6:   bf 4f           sbci    r27, 0xFF       ; 255
 ce8:   8d 91           ld      r24, X+
 cea:   9c 91           ld      r25, X
 cec:   11 97           sbiw    r26, 0x01       ; 1
 cee:   28 17           cp      r18, r24
 cf0:   39 07           cpc     r19, r25
 cf2:   19 f4           brne    .+6             ; 0xcfa <_ZN14HardwareSerial4peekEv+0x2e>
 cf4:   2f ef           ldi     r18, 0xFF       ; 255
 cf6:   3f ef           ldi     r19, 0xFF       ; 255
 cf8:   07 c0           rjmp    .+14            ; 0xd08 <_ZN14HardwareSerial4peekEv+0x3c>
    return -1;
  } else {
    return _rx_buffer->buffer[_rx_buffer->tail];
 cfa:   8d 91           ld      r24, X+
 cfc:   9c 91           ld      r25, X
 cfe:   e8 0f           add     r30, r24
 d00:   f9 1f           adc     r31, r25
 d02:   80 81           ld      r24, Z
 d04:   28 2f           mov     r18, r24
 d06:   30 e0           ldi     r19, 0x00       ; 0
  }
}
 d08:   c9 01           movw    r24, r18
 d0a:   08 95           ret

00000d0c <_ZN14HardwareSerial4readEv>:

int HardwareSerial::read(void)
{
  // if the head isn't ahead of the tail, we don't have any characters
  if (_rx_buffer->head == _rx_buffer->tail) {
 d0c:   dc 01           movw    r26, r24
 d0e:   1c 96           adiw    r26, 0x0c       ; 12
 d10:   ed 91           ld      r30, X+
 d12:   fc 91           ld      r31, X
 d14:   1d 97           sbiw    r26, 0x0d       ; 13
 d16:   e0 5c           subi    r30, 0xC0       ; 192
 d18:   ff 4f           sbci    r31, 0xFF       ; 255
 d1a:   20 81           ld      r18, Z
 d1c:   31 81           ldd     r19, Z+1        ; 0x01
 d1e:   e0 54           subi    r30, 0x40       ; 64
 d20:   f0 40           sbci    r31, 0x00       ; 0
 d22:   df 01           movw    r26, r30
 d24:   ae 5b           subi    r26, 0xBE       ; 190
 d26:   bf 4f           sbci    r27, 0xFF       ; 255
 d28:   8d 91           ld      r24, X+
 d2a:   9c 91           ld      r25, X
 d2c:   11 97           sbiw    r26, 0x01       ; 1
 d2e:   28 17           cp      r18, r24
 d30:   39 07           cpc     r19, r25
 d32:   19 f4           brne    .+6             ; 0xd3a <_ZN14HardwareSerial4readEv+0x2e>
 d34:   2f ef           ldi     r18, 0xFF       ; 255
 d36:   3f ef           ldi     r19, 0xFF       ; 255
 d38:   10 c0           rjmp    .+32            ; 0xd5a <_ZN14HardwareSerial4readEv+0x4e>
    return -1;
  } else {
    unsigned char c = _rx_buffer->buffer[_rx_buffer->tail];
 d3a:   8d 91           ld      r24, X+
 d3c:   9c 91           ld      r25, X
 d3e:   11 97           sbiw    r26, 0x01       ; 1
 d40:   e8 0f           add     r30, r24
 d42:   f9 1f           adc     r31, r25
 d44:   20 81           ld      r18, Z
    _rx_buffer->tail = (unsigned int)(_rx_buffer->tail + 1) % SERIAL_BUFFER_SIZE;
 d46:   8d 91           ld      r24, X+
 d48:   9c 91           ld      r25, X
 d4a:   11 97           sbiw    r26, 0x01       ; 1
 d4c:   01 96           adiw    r24, 0x01       ; 1
 d4e:   8f 73           andi    r24, 0x3F       ; 63
 d50:   90 70           andi    r25, 0x00       ; 0
 d52:   11 96           adiw    r26, 0x01       ; 1
 d54:   9c 93           st      X, r25
 d56:   8e 93           st      -X, r24
    return c;
 d58:   30 e0           ldi     r19, 0x00       ; 0
  }
}
 d5a:   c9 01           movw    r24, r18
 d5c:   08 95           ret

00000d5e <_ZN14HardwareSerial5flushEv>:

void HardwareSerial::flush()
{
  while (_tx_buffer->head != _tx_buffer->tail)
 d5e:   fc 01           movw    r30, r24
 d60:   86 85           ldd     r24, Z+14       ; 0x0e
 d62:   97 85           ldd     r25, Z+15       ; 0x0f
 d64:   dc 01           movw    r26, r24
 d66:   a0 5c           subi    r26, 0xC0       ; 192
 d68:   bf 4f           sbci    r27, 0xFF       ; 255
 d6a:   fc 01           movw    r30, r24
 d6c:   ee 5b           subi    r30, 0xBE       ; 190
 d6e:   ff 4f           sbci    r31, 0xFF       ; 255
 d70:   2d 91           ld      r18, X+
 d72:   3c 91           ld      r19, X
 d74:   11 97           sbiw    r26, 0x01       ; 1
 d76:   80 81           ld      r24, Z
 d78:   91 81           ldd     r25, Z+1        ; 0x01
 d7a:   28 17           cp      r18, r24
 d7c:   39 07           cpc     r19, r25
 d7e:   c1 f7           brne    .-16            ; 0xd70 <_ZN14HardwareSerial5flushEv+0x12>
    ;
}
 d80:   08 95           ret

00000d82 <_ZN14HardwareSerial5writeEh>:

size_t HardwareSerial::write(uint8_t c)
 d82:   cf 93           push    r28
 d84:   df 93           push    r29
 d86:   ec 01           movw    r28, r24
 d88:   46 2f           mov     r20, r22
{
  int i = (_tx_buffer->head + 1) % SERIAL_BUFFER_SIZE;
 d8a:   ee 85           ldd     r30, Y+14       ; 0x0e
 d8c:   ff 85           ldd     r31, Y+15       ; 0x0f
 d8e:   e0 5c           subi    r30, 0xC0       ; 192
 d90:   ff 4f           sbci    r31, 0xFF       ; 255
 d92:   80 81           ld      r24, Z
 d94:   91 81           ldd     r25, Z+1        ; 0x01
 d96:   e0 54           subi    r30, 0x40       ; 64
 d98:   f0 40           sbci    r31, 0x00       ; 0
 d9a:   01 96           adiw    r24, 0x01       ; 1
 d9c:   60 e4           ldi     r22, 0x40       ; 64
 d9e:   70 e0           ldi     r23, 0x00    ; 0
 da0:   0e 94 9a 07     call    0xf34   ; 0xf34 <__divmodhi4>
 da4:   9c 01           movw    r18, r24

  // If the output buffer is full, there's nothing for it other than to
  // wait for the interrupt handler to empty it a bit
  // ???: return 0 here instead?
  while (i == _tx_buffer->tail)
 da6:   df 01           movw    r26, r30
 da8:   ae 5b           subi    r26, 0xBE       ; 190
 daa:   bf 4f           sbci    r27, 0xFF       ; 255
 dac:   8d 91           ld      r24, X+
 dae:   9c 91           ld      r25, X
 db0:   11 97           sbiw    r26, 0x01       ; 1
 db2:   28 17           cp      r18, r24
 db4:   39 07           cpc     r19, r25
 db6:   d1 f3           breq    .-12            ; 0xdac <_ZN14HardwareSerial5writeEh+0x2a>
    ;

  _tx_buffer->buffer[_tx_buffer->head] = c;
 db8:   e0 5c           subi    r30, 0xC0       ; 192
 dba:   ff 4f           sbci    r31, 0xFF       ; 255
 dbc:   80 81           ld      r24, Z
 dbe:   91 81           ldd     r25, Z+1        ; 0x01
 dc0:   e0 54           subi    r30, 0x40       ; 64
 dc2:   f0 40           sbci    r31, 0x00       ; 0
 dc4:   e8 0f           add     r30, r24
 dc6:   f9 1f           adc     r31, r25
 dc8:   40 83           st      Z, r20
  _tx_buffer->head = i;
 dca:   ee 85           ldd     r30, Y+14       ; 0x0e
 dcc:   ff 85           ldd     r31, Y+15       ; 0x0f
 dce:   e0 5c           subi    r30, 0xC0       ; 192
 dd0:   ff 4f           sbci    r31, 0xFF       ; 255
 dd2:   31 83           std     Z+1, r19        ; 0x01
 dd4:   20 83           st      Z, r18

  sbi(*_ucsrb, _udrie);
 dd6:   ee 89           ldd     r30, Y+22       ; 0x16
 dd8:   ff 89           ldd     r31, Y+23       ; 0x17
 dda:   20 81           ld      r18, Z
 ddc:   81 e0           ldi     r24, 0x01       ; 1
 dde:   90 e0           ldi     r25, 0x00       ; 0
 de0:   0d 8c           ldd     r0, Y+29        ; 0x1d
 de2:   02 c0           rjmp    .+4             ; 0xde8 <_ZN14HardwareSerial5writeEh+0x66>
 de4:   88 0f           add     r24, r24
 de6:   99 1f           adc     r25, r25
 de8:   0a 94           dec     r0
 dea:   e2 f7           brpl    .-8             ; 0xde4 <_ZN14HardwareSerial5writeEh+0x62>
 dec:   28 2b           or      r18, r24
 dee:   20 83           st      Z, r18

  return 1;
}
 df0:   81 e0           ldi     r24, 0x01       ; 1
 df2:   90 e0           ldi     r25, 0x00       ; 0
 df4:   df 91           pop     r29
 df6:   cf 91           pop     r28
 df8:   08 95           ret

00000dfa <_GLOBAL__I_rx_buffer>:
 dfa:   10 92 30 04     sts     0x0430, r1
 dfe:   10 92 2f 04     sts     0x042F, r1
 e02:   88 ee           ldi     r24, 0xE8       ; 232
 e04:   93 e0           ldi     r25, 0x03       ; 3
 e06:   a0 e0           ldi     r26, 0x00       ; 0
 e08:   b0 e0           ldi     r27, 0x00       ; 0
 e0a:   80 93 31 04     sts     0x0431, r24
 e0e:   90 93 32 04     sts     0x0432, r25
 e12:   a0 93 33 04     sts     0x0433, r26
 e16:   b0 93 34 04     sts     0x0434, r27

HardwareSerial::HardwareSerial(ring_buffer *rx_buffer, ring_buffer *tx_buffer,
  volatile uint8_t *ubrrh, volatile uint8_t *ubrrl,
  volatile uint8_t *ucsra, volatile uint8_t *ucsrb,
  volatile uint8_t *udr,
  uint8_t rxen, uint8_t txen, uint8_t rxcie, uint8_t udrie, uint8_t u2x)
 e1a:   8c e0           ldi     r24, 0x0C       ; 12
 e1c:   93 e0           ldi     r25, 0x03       ; 3
 e1e:   90 93 2e 04     sts     0x042E, r25
 e22:   80 93 2d 04     sts     0x042D, r24
{
  _rx_buffer = rx_buffer;
 e26:   85 ea           ldi     r24, 0xA5       ; 165
 e28:   93 e0           ldi     r25, 0x03       ; 3
 e2a:   90 93 3a 04     sts     0x043A, r25
 e2e:   80 93 39 04     sts     0x0439, r24
  _tx_buffer = tx_buffer;
 e32:   89 ee           ldi     r24, 0xE9       ; 233
 e34:   93 e0           ldi     r25, 0x03       ; 3
 e36:   90 93 3c 04     sts     0x043C, r25
 e3a:   80 93 3b 04     sts     0x043B, r24
  _ubrrh = ubrrh;
 e3e:   85 ec           ldi     r24, 0xC5       ; 197
 e40:   90 e0           ldi     r25, 0x00       ; 0
 e42:   90 93 3e 04     sts     0x043E, r25
 e46:   80 93 3d 04     sts     0x043D, r24
  _ubrrl = ubrrl;
 e4a:   84 ec           ldi     r24, 0xC4       ; 196
 e4c:   90 e0           ldi     r25, 0x00       ; 0
 e4e:   90 93 40 04     sts     0x0440, r25
 e52:   80 93 3f 04     sts     0x043F, r24
  _ucsra = ucsra;
 e56:   80 ec           ldi     r24, 0xC0       ; 192
 e58:   90 e0           ldi     r25, 0x00       ; 0
 e5a:   90 93 42 04     sts     0x0442, r25
 e5e:   80 93 41 04     sts     0x0441, r24
  _ucsrb = ucsrb;
 e62:   81 ec           ldi     r24, 0xC1       ; 193
 e64:   90 e0           ldi     r25, 0x00       ; 0
 e66:   90 93 44 04     sts     0x0444, r25
 e6a:   80 93 43 04     sts     0x0443, r24
  _udr = udr;
 e6e:   86 ec           ldi     r24, 0xC6       ; 198
 e70:   90 e0           ldi     r25, 0x00       ; 0
 e72:   90 93 46 04     sts     0x0446, r25
 e76:   80 93 45 04     sts     0x0445, r24
  _rxen = rxen;
 e7a:   84 e0           ldi     r24, 0x04       ; 4
 e7c:   80 93 47 04     sts     0x0447, r24
  _txen = txen;
 e80:   83 e0           ldi     r24, 0x03       ; 3
 e82:   80 93 48 04     sts     0x0448, r24
  _rxcie = rxcie;
 e86:   87 e0           ldi     r24, 0x07       ; 7
 e88:   80 93 49 04     sts     0x0449, r24
  _udrie = udrie;
 e8c:   85 e0           ldi     r24, 0x05       ; 5
 e8e:   80 93 4a 04     sts     0x044A, r24
  _u2x = u2x;
 e92:   81 e0           ldi     r24, 0x01       ; 1
 e94:   80 93 4b 04     sts     0x044B, r24
// Preinstantiate Objects //////////////////////////////////////////////////////

#if defined(UBRRH) && defined(UBRRL)
  HardwareSerial Serial(&rx_buffer, &tx_buffer, &UBRRH, &UBRRL, &UCSRA, &UCSRB, &UDR, RXEN, TXEN,
RXCIE, UDRIE, U2X);
#elif defined(UBRR0H) && defined(UBRR0L)
  HardwareSerial Serial(&rx_buffer, &tx_buffer, &UBRR0H, &UBRR0L, &UCSR0A, &UCSR0B, &UDR0, RXEN0,
TXEN0, RXCIE0, UDRIE0, U2X0);
 e98:   08 95           ret

00000e9a <main>:
#define ARDUINO_MAIN
#include <Arduino.h>

int main(void)
 e9a:   cf 93           push    r28
 e9c:   df 93           push    r29
{
        init();
 e9e:   0e 94 dd 04     call    0x9ba   ; 0x9ba <init>

#if defined(USBCON)
        USB.attach();
#endif

        setup();
 ea2:   0e 94 bf 03     call    0x77e   ; 0x77e <setup>

        for (;;) {
                loop();
                if (serialEventRun) serialEventRun();
 ea6:   c3 e7           ldi     r28, 0x73       ; 115
 ea8:   d5 e0           ldi     r29, 0x05       ; 5
#endif

        setup();

        for (;;) {
                loop();
 eaa:   0e 94 a7 01     call    0x34e   ; 0x34e <loop>
                if (serialEventRun) serialEventRun();
 eae:   20 97           sbiw    r28, 0x00       ; 0
 eb0:   e1 f3           breq    .-8             ; 0xeaa <main+0x10>
 eb2:   0e 94 73 05     call    0xae6   ; 0xae6 <_Z14serialEventRunv>
 eb6:   f9 cf           rjmp    .-14            ; 0xeaa <main+0x10>

00000eb8 <_ZN5Print5writeEPKhj>:
#include "Print.h"

// Public Methods //////////////////////////////////////////////////////////////
```

```
/* default implementation: may be overridden */
size_t Print::write(const uint8_t *buffer, size_t size)
 eb8:   cf 92           push    r12
 eba:   df 92           push    r13
 ebc:   ef 92           push    r14
 ebe:   ff 92           push    r15
 ec0:   0f 93           push    r16
 ec2:   1f 93           push    r17
 ec4:   cf 93           push    r28
 ec6:   df 93           push    r29
 ec8:   7c 01           movw    r14, r24
 eca:   6b 01           movw    r12, r22
 ecc:   8a 01           movw    r16, r20
 ece:   c0 e0           ldi     r28, 0x00       ; 0
 ed0:   d0 e0           ldi     r29, 0x00       ; 0
 ed2:   0f c0           rjmp    .+30            ; 0xef2 <_ZN5Print5writeEPKhj+0x3a>
{
  size_t n = 0;
  while (size--) {
    n += write(*buffer++);
 ed4:   d6 01           movw    r26, r12
 ed6:   6d 91           ld      r22, X+
 ed8:   6d 01           movw    r12, r26
 eda:   d7 01           movw    r26, r14
 edc:   ed 91           ld      r30, X+
 ede:   fc 91           ld      r31, X
 ee0:   01 90           ld      r0, Z+
 ee2:   f0 81           ld      r31, Z
 ee4:   e0 2d           mov     r30, r0
 ee6:   c7 01           movw    r24, r14
 ee8:   09 95           icall
 eea:   c8 0f           add     r28, r24
 eec:   d9 1f           adc     r29, r25
 eee:   01 50           subi    r16, 0x01       ; 1
 ef0:   10 40           sbci    r17, 0x00       ; 0

/* default implementation: may be overridden */
size_t Print::write(const uint8_t *buffer, size_t size)
{
  size_t n = 0;
  while (size--) {
 ef2:   01 15           cp      r16, r1
 ef4:   11 05           cpc     r17, r1
 ef6:   71 f7           brne    .-36            ; 0xed4 <_ZN5Print5writeEPKhj+0x1c>
    n += write(*buffer++);
  }
  return n;
}
 ef8:   ce 01           movw    r24, r28
 efa:   df 91           pop     r29
 efc:   cf 91           pop     r28
 efe:   1f 91           pop     r17
 f00:   0f 91           pop     r16
 f02:	ff 90       	pop	r15
 f04:	ef 90       	pop	r14
 f06:	df 90       	pop	r13
 f08:	cf 90       	pop	r12
 f0a:	08 95       	ret

00000f0c <__udivmodhi4>:
 f0c:	aa 1b       	sub	r26, r26
 f0e:	bb 1b       	sub	r27, r27
 f10:	51 e1       	ldi	r21, 0x11	; 17
 f12:	07 c0       	rjmp	.+14     	; 0xf22 <__udivmodhi4_ep>

00000f14 <__udivmodhi4_loop>:
 f14:	aa 1f       	adc	r26, r26
 f16:	bb 1f       	adc	r27, r27
 f18:	a6 17       	cp	r26, r22
 f1a:	b7 07       	cpc	r27, r23
 f1c:	10 f0       	brcs	.+4      	; 0xf22 <__udivmodhi4_ep>
 f1e:	a6 1b       	sub	r26, r22
 f20:	b7 0b       	sbc	r27, r23

00000f22 <__udivmodhi4_ep>:
 f22:	88 1f       	adc	r24, r24
 f24:	99 1f       	adc	r25, r25
 f26:	5a 95       	dec	r21
 f28:	a9 f7       	brne	.-22     	; 0xf14 <__udivmodhi4_loop>
 f2a:	80 95       	com	r24
 f2c:	90 95       	com	r25
 f2e:	bc 01       	movw	r22, r24
 f30:	cd 01       	movw	r24, r26
 f32:	08 95       	ret

00000f34 <__divmodhi4>:
 f34:	97 fb       	bst	r25, 7
 f36:	09 2e       	mov	r0, r25
 f38:	07 26       	eor	r0, r23
 f3a:	0a d0       	rcall	.+20     	; 0xf50 <__divmodhi4_neg1>
 f3c:	77 fd       	sbrc	r23, 7
 f3e:	04 d0       	rcall	.+8      	; 0xf48 <__divmodhi4_neg2>
 f40:	e5 df       	rcall	.-54     	; 0xf0c <__udivmodhi4>
 f42:	06 d0       	rcall	.+12     	; 0xf50 <__divmodhi4_neg1>
 f44:	00 20       	and	r0, r0
 f46:	1a f4       	brpl	.+6      	; 0xf4e <__divmodhi4_exit>

00000f48 <__divmodhi4_neg2>:
 f48:	70 95       	com	r23
 f4a:	61 95       	neg	r22
 f4c:	7f 4f       	sbci	r23, 0xFF	; 255

00000f4e <__divmodhi4_exit>:
 f4e:	08 95       	ret

00000f50 <__divmodhi4_neg1>:
 f50:	f6 f7       	brtc	.-4      	; 0xf4e <__divmodhi4_exit>
 f52:   90 95       com     r25
 f54:   81 95       neg     r24
 f56:   9f 4f       sbci    r25, 0xFF       ; 255
 f58:   08 95       ret

00000f5a <__udivmodsi4>:
 f5a:   a1 e2       ldi     r26, 0x21       ; 33
 f5c:   1a 2e       mov     r1, r26
 f5e:   aa 1b       sub     r26, r26
 f60:   bb 1b       sub     r27, r27
 f62:   fd 01       movw    r30, r26
 f64:   0d c0       rjmp    .+26            ; 0xf80 <__udivmodsi4_ep>

00000f66 <__udivmodsi4_loop>:
 f66:   aa 1f       adc     r26, r26
 f68:   bb 1f       adc     r27, r27
 f6a:   ee 1f       adc     r30, r30
 f6c:   ff 1f       adc     r31, r31
 f6e:   a2 17       cp      r26, r18
 f70:   b3 07       cpc     r27, r19
 f72:   e4 07       cpc     r30, r20
 f74:   f5 07       cpc     r31, r21
 f76:   20 f0       brcs    .+8             ; 0xf80 <__udivmodsi4_ep>
 f78:   a2 1b       sub     r26, r18
 f7a:   b3 0b       sbc     r27, r19
 f7c:   e4 0b       sbc     r30, r20
 f7e:   f5 0b       sbc     r31, r21

00000f80 <__udivmodsi4_ep>:
 f80:   66 1f       adc     r22, r22
 f82:   77 1f       adc     r23, r23
 f84:     88 1f       adc     r24, r24
 f86:   99 1f       adc     r25, r25
 f88:   1a 94       dec     r1
 f8a:   69 f7       brne    .-38            ; 0xf66 <__udivmodsi4_loop>
 f8c:   60 95       com     r22
 f8e:   70 95       com     r23
 f90:   80 95       com     r24
 f92:   90 95       com     r25
 f94:   9b 01       movw    r18, r22
 f96:   ac 01       movw    r20, r24
 f98:   bd 01       movw    r22, r26
 f9a:   cf 01       movw    r24, r30
 f9c:   08 95       ret

00000f9e <__tablejump2__>:
 f9e:   ee 0f       add     r30, r30
 fa0:   ff 1f       adc     r31, r31

00000fa2 <__tablejump__>:
 fa2:   05 90       lpm     r0, Z+
 fa4:   f4 91       lpm     r31, Z+
 fa6:   e0 2d       mov     r30, r0
 fa8:   09 94       ijmp
```

```
00000faa <_exit>:
 faa:   f8 94           cli

00000fac <__stop_program>:
 fac:   ff cf           rjmp    .-2             ; 0xfac <__stop_program>
```