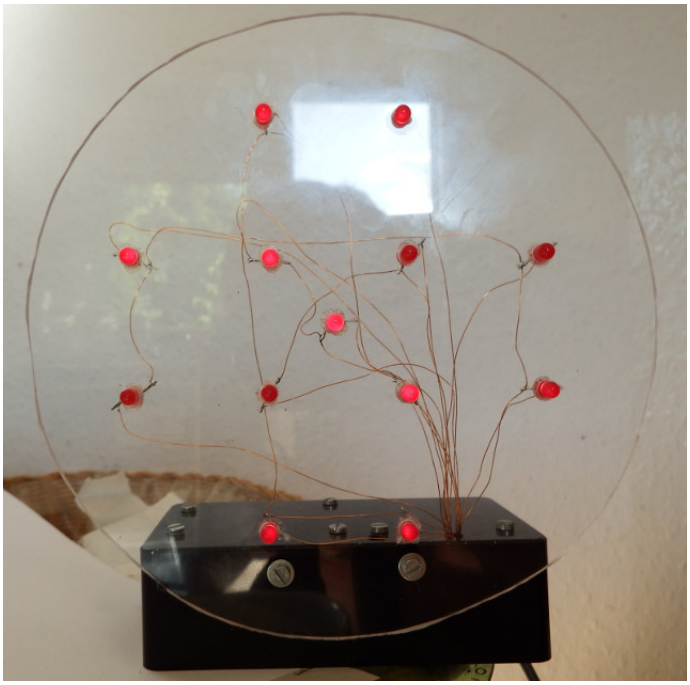
User manual for

Magic Circle



by

Schmidt Microsystems
Darmstadt/Germany
http://www.avr-asm-tutorial.net/avr_en/apps/magic_circle_tn2313/magic_circle.html

May 2017

Congratulations for buying Magic Circle!

This device does nothing but fun.

Look and feel

This device turns lamps on and off, but not immediately. The intensity starts at a low value, then increases until its top. Then it decreases again until darkness. No other device does that in that way, so you have an absolutely exclusive device.

Use of the device

Power supply:

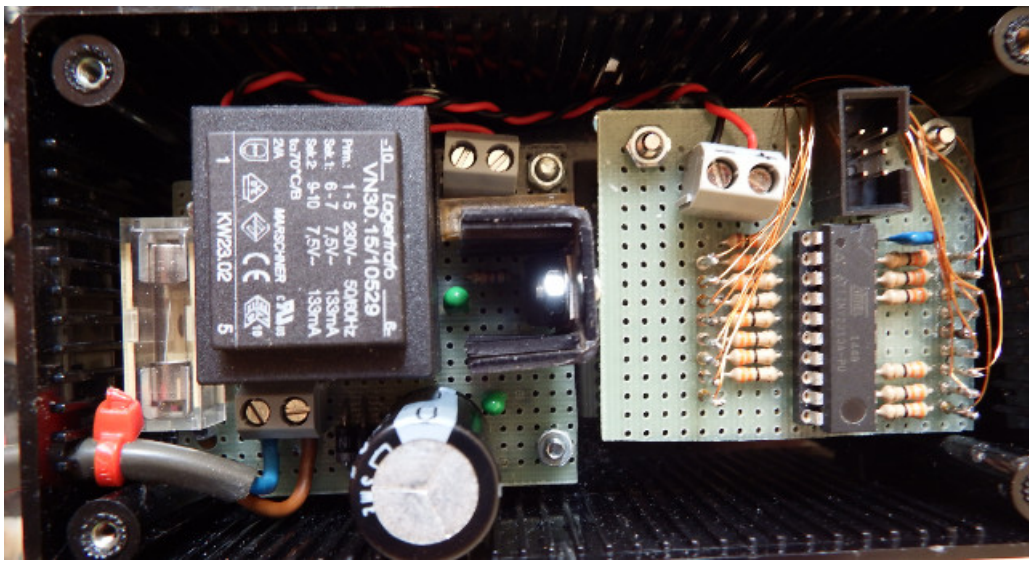
> Power comes from wall plug 110 or 230 Volts, depending from your device designers choice. Just plug in the plug, no switch has to be turned on. Consumption is less than 1 Watt, so less than 8.76 kWh are consumed over a whole year.

Funktioning:

> The device is constructed with a modern microprofessor ATtiny2313 from ATMEL. It runs on 3.3 Volt and consumes only 0.2 milliampere, even though it runs with one million instructions per second. But, due to the exclusive and extraordinary software design of Schmidt Microsystems, it is mostly sleeping, by that reducing its power consumption. The professor has 2,048 bytes program storage, which is filled roughly halve. The device is very longlived, except if you throw it into the toilet or from an airplane.

Repairs:

> You can open the box if you are curios about what is in there and if you believe that the fotos provided might be fake. But make sure that you remove the wall plug before opening the box, the inside may shock you to death otherwise. The device has a fuse inside, so it will not burn your house down if the transformer inside degrades. If you want to replace the fuse: buy a 50 mA delay fuse and remove the plastic cover from the fuseholder. After replacing the fuse re-install the plastic cover, please. No other repair parts are in the box. Keep your hands away from the professor, he is very sensitive to cold hands.



Quality:

The power supply was tested with 200 mA over two hours, but did not fail. The whole device was tested over two days and did not fail either. That is enough quality check for that kind of device.
ISO9000, Euro or any other quality seals as well as regular technical inspections are super-fluous inventions for that.

Warranty:

Your warranty is two years for correct functioning. If defectice call our service center in Ireland under 0900-1234567. They won't help you but they will book 99 Euro per minute from your phone bill. The repair is costfree but only if the device was not thrown into the toilet or from an airplane.

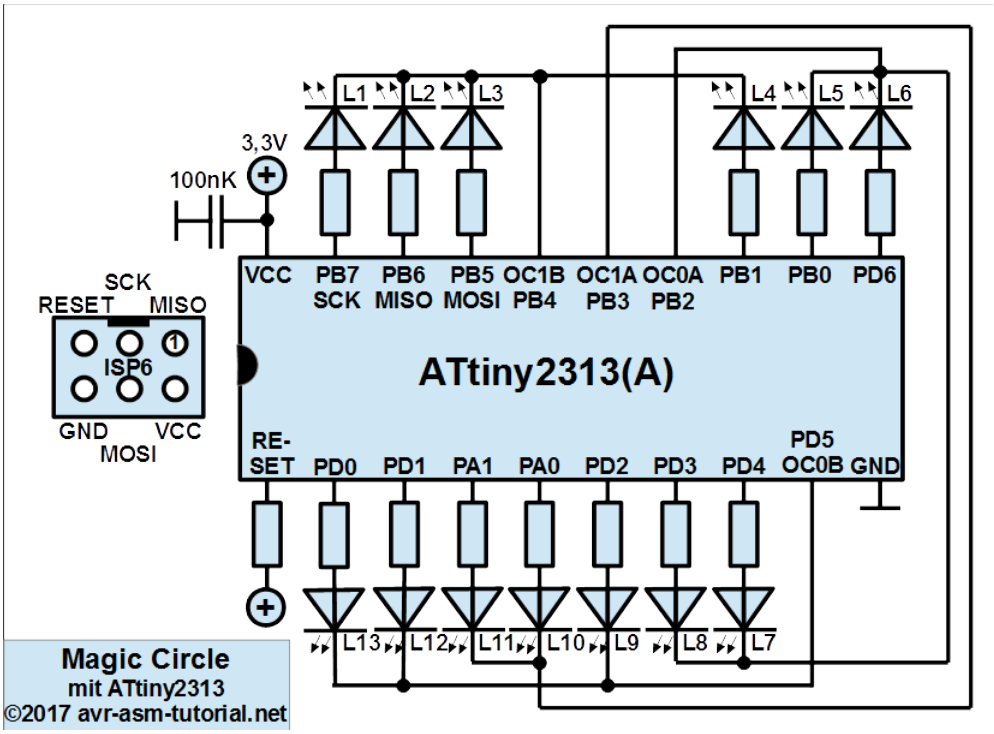
Updates:

You can design a new blink diagram and send it to the website of the producer. The producer then produces a new version and burns it into the professor's brain. For that the producer comes to your house with a small computer, a programming device and a screw driver, so make sure you are at home at any time. You can monitor the program update, if you are fearless.

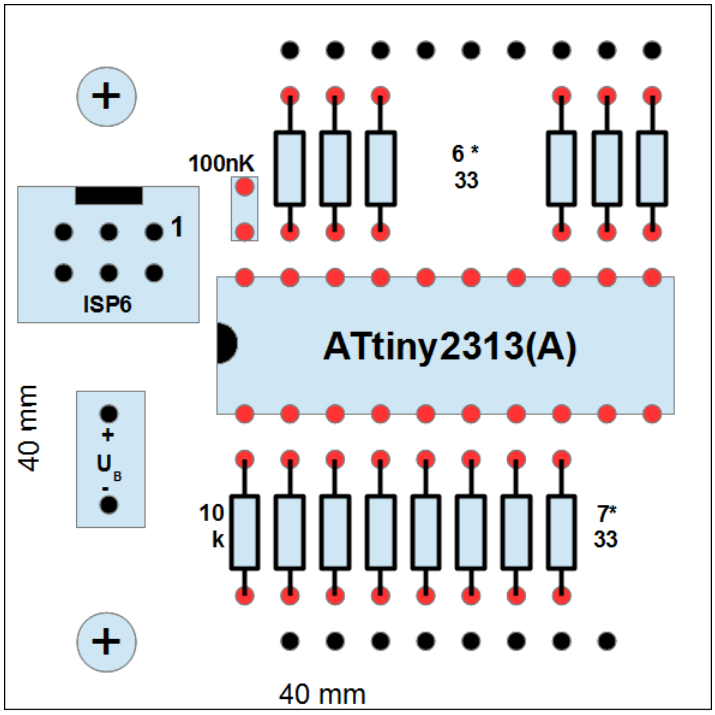
Attachments:

1. Schematic of the professor part
2. Schematic of LEDs and cables
3. Schematic of the power supply
4. Layout of professor device and power supply
5. Program of the mikroprofessor in AVR assembler

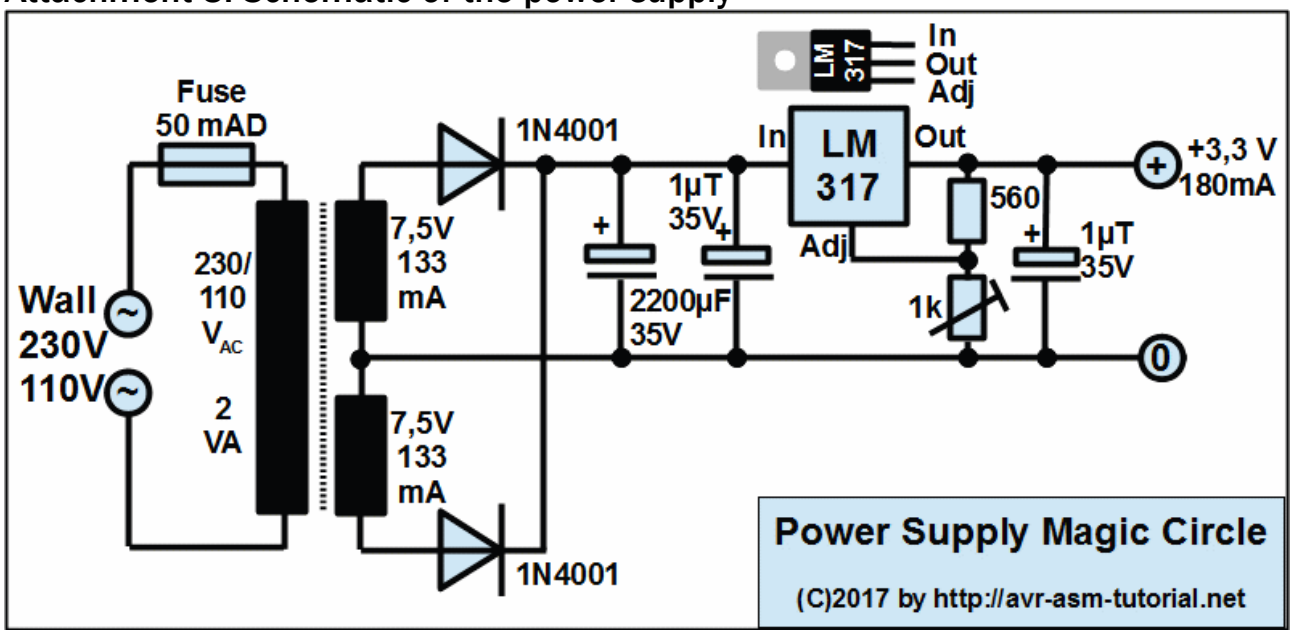
Attachment 1: Schematic of the microprofessor



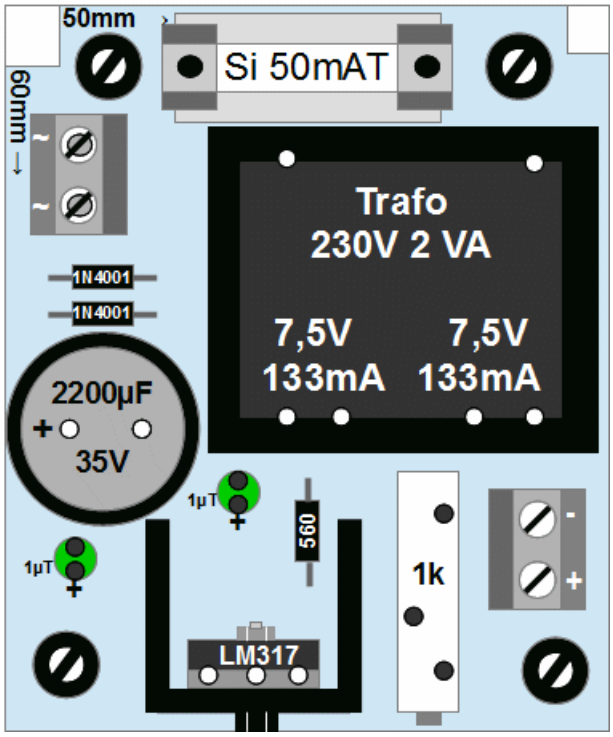
Attachment 2: Component placement of the microprofessor



Attachment 3: Schematic of the power supply



Attachment 4: Component placement of the power supply



Attachment 5: Program of the microprofessor in AVR assembler

```
;
; ****************************************
; * Magic circle with ATtiny2313 and 12 LEDs *
; * Version 1 of Juli 2017                 *
; * (C)2017 by www.avr-asm-tutorial.net    *
; ****************************************
;
; Include file for the AVR type
.NOLIST
.INCLUDE "tn2313def.inc" ; Header file for ATtiny2313
.LIST
;
; =============================================
;       D E B U G - S W I T C H E S
; =============================================
;
; Final: all debug switches = 0
.equ debug_lampcurrent = 0
.equ debug_back = 0
;
; =============================================
;   H A R D W A R E    I N F O R M A T I O N
; =============================================
;
; Devices = ATtiny2313/4313
;                 _________________
;          1 /                |20
;  Reset o--|reset          vcc|--o +3.3V (max 3.6V)
;          2|                  |19
;    L13 o--|pd0            pb7|--o L1
;          3|                  |18
;    L12 o--|pd1            pb6|--o L2
;          4|                  |17
;    L11 o--|pa1            pb5|--o L3
;          5|                  |16
;    L10 o--|pa0       oc1b/pb4|--o Cath L1+L2+L3+L4
;          6|                  |15
;     L9 o--|pd2       oc1a/pb3|--o Cath L10+L11
;          7|                  |14
;     L8 o--|pd3       oc0a/pb2|--o Cath L5+L6+L7+L8
;          8|                  |13
;     L7 o--|pd4            pb1|--o L4
; Cath     9|                  |12
; L9+L12 o--|pd5/oc0b       pb0|--o L5
; +L13    10|                  |11
;     0V o--|gnd            pd6|--o L6
;           |_________________|
;
; =============================================
;              T I M I N G
; =============================================
;
; Controller clock
;   The controller runs with the internal 8 MHz
;   RC oszillator and with the CKDIV8 fuse set
;   (without fuse change, default oscillator
;   settings, therefore with 1 MHz clock
;
; LED driving:
;   Timers TC0 and TC1 run synchroneous with a
;   prescaler of 8 (125 kHz) in 8-bit PWM mode
```

```
;   and drive the LEDs (488 Hz) via the four PWM
;   outputs OCR0A, OCR0B, OCR1A and OCR1B.
;   All A and B compare values are synchroneously
;   set to the same values.
;   All PWM outputs work in toggle mode with
;   244 Hz PWM frequency.
;   LEDs, that are switched off, have their
;   direction bit cleared. LEDs that are switched
;   on have their direction bit and their output
;   bit set.
;   The PWM value is increased during each PWM run.
;   If the PWM value flows over either the PWM
;   value is decreasing down to 0 or is immediately
;   restarting at zero (if flag bUpOnly is set).
;   At the end of each cycle the next bit combination
;   of the LEDs or a control byte is read and
;   executed.
;
; Cycle control:
;   This is done with a table in flash. Each data
;   entry consists of four bytes:
;   1. Flag control byte: The flags can be normal
;      (0), upwards only (bit 0 set), double speed
;      (bit 1 set) and fourfold speed (bits 2 and
;      3 set).
;   2. Direction bits for Port D (set = LED on),
;      bit 6 is is always set
;   3. Direction bits for Port B, with bits 2, 3
;      and 4 always set
;   4. Direction bits for Port A
;   With the following control bytes as first byte
;   the cycle execution is controlled. In that case
;   only one following byte is read:
;   0xFF: End of the cycle table, restart at the
;         beginning, the second byte is ignored
;   0xFE: Repeat the following entries (outer loop),
;         the second byte contains the number of
;         repetitions
;   0xFD: Repeat the following entries (inner loop),
;         the second byte contains the number of
;         repetitions
;   0xFC: End of the outer loop, decrease outstanding
;         repetitions, if not zero, restart at the
;         stored address of the outer loop
;   0xFB: End of the inner loop, decrease outstanding
;         repetitions, if not zero, restart at the
;         stored address of the inner loop
;
; ============================================
;   R E G I S T E R   D E F I N I T I O N S
; ============================================
;
; Free: R0 to R14
.DEF rSreg = R15 ; SREG save/restore
.DEF rmp = R16 ; Multi purpose register
.DEF rimp = R17 ; Multi purpose inside interrupts
.DEF rFlag = R18 ; Flag register
      .equ bUpOnly = 0 ; Up only
      .equ bFast = 1 ; Double speed
      .equ bVeryFast = 2 ; Fourfold speed
      .equ bFall = 6 ; Downwards cycle
      .equ bEndCycle = 7 ; End of cycle
.DEF rRep1 = R19 ; Outer loop repetitions
.DEF rRep2 = R20 ; Inner loop repetitions
```

```
.DEF rPwm = R21 ; PWM value
; Free: R22 to R25
; Used: R27:R26 = X for outer loop address
; Used: R29:R28 = Y for inner loop address
; Used: R31:R30 = Z for table read access address
;
; =============================================
;       S R A M   D E F I N I T I O N S
; =============================================
;
.DSEG
.ORG  0X0060
; (No SRAM use, used for stack only)
;
; =============================================
;   R E S E T   A N D   I N T   V E C T O R S
; =============================================
;
.CSEG
.ORG $0000
      rjmp Main ; Reset-Vektor
      reti ; INT0 Int Vektor 1
      reti ; INT1 Int Vektor 2
      reti ; TC1CAPT Int Vektor 3
      reti ; TC1COMPA Int Vektor 4
      reti ; TC1OVF Int Vektor 5
      rjmp Isr ; TC0OVF Int Vektor 6
      reti ; USART-RX Int Vektor 7
      reti ; USART UDRE Int Vektor 8
      reti ; USART TX Int Vektor 9
      reti ; ANACOMP Int Vektor 10
      reti ; PCINT Int Vektor 11
      reti ; TC1COMPB Int Vektor 12
      reti ; TC0COMPA Int Vektor 13
      reti ; TC0COMPB Int Vektor 14
      reti ; USI-START Int Vektor 15
      reti ; USI-OVERFLOW Int Vektor 16
      reti ; EEREADY Int Vektor 17
      reti ; WDT-OVERFLOW Int Vektor 18
;
; =============================================
;    I N T E R R U P T   S E R V I C E
; =============================================
;
Isr: ; TC0 overflow interrupt
      in rSreg,SREG ; Save SREG
      ldi rimp,0 ; clear all MSB compare values
      out OCR1AH,rimp
      out OCR1AL,rPwm ; Set PWM value A TC1
      out OCR1BH,rimp
      out OCR1BL,rPwm ; Set PWM value B TC1
      out OCR0A,rPwm ; Set PWM value A TC0
      out OCR0B,rPwm ; Set PWM value B TC0
      sbrs rFlag,bFall ; Skip next if falling PWM
      rjmp IsrUp ; PWM goes up only
      sbrc rFlag,bVeryFast ; Skip next if not very fast
      subi rPwm,2 ; two steps down
      sbrc rFlag,bFast ; Skip next if not fast
      subi rPwm,1 ; Add another step down
      subi rPwm,1 ; A single step down in any case
      brne IsrRet ; If not at zero: return
      cbr rFlag,1<<bFall ; Clear downwards flag
      sbr rFlag,1<<bEndCycle ; Signal cycle end
      rjmp IsrRet ; Return
```

```
IsrUp: ; PWM value upwards
      sbrc rFlag,bVeryFast ; Skip next if not very fast
      subi rPwm,-2 ; Add two
      sbrc rFlag,bFast ; Skip next if not fast
      inc rPwm ; Add another step up
      inc rPwm ; A single step up in any case
      brne IsrRet ; If not zero return
      sbrc rFlag,bUpOnly ; Skip next if up only
      rjmp IsrUpOnly ; Jump to up only
      sbrc rFlag,bVeryFast ; Skip next if not very fast
      subi rPwm,2 ; Two steps downward
      sbrc rFlag,bFast ; Skip next if not fast
      dec rPwm ; Add another step down
      dec rPwm ; Step down in any case
      sbr rFlag,1<<bFall ; Set downwards flag
      rjmp IsrRet ; Done, return
IsrUpOnly: ; Step up only
      clr rPwm ; Restart PWM at zero
      sbr rFlag,1<<bEndCycle ; Signal end of cycle
IsrRet:
      wdr ; Clear the watchdog
      out SREG,rSreg ; Restore SREG
      reti ; Return from interrupt
;
; =============================================
;      M A I N   P R O G R A M    I N I T
; =============================================
;
Main:
; Init stack
      ldi rmp, LOW(RAMEND) ; Init LSB stack
      out SPL,rmp
;
; Debug code
.if debug_back == 1
      rcall back
      .endif
; Add start-up delay
      clr ZH
      clr ZL
      ldi rmp,5
Delay:
      wdr
      sbiw ZL,1 ; 2
      brne Delay ; 2
      dec rmp
      brne Delay
;
; Init Port D
      ldi rmp,0b01011111 ; Output pins port D
      out PORTD,rmp
      ldi rmp,0b00100000 ; OC pin as output, LED pins as input
      out DDRD,rmp
; Init Port B
      ldi rmp,0b11100011 ; Output pins port B
      out PORTB,rmp
      ldi rmp,0b00011100 ; OC pins as output, LED pins as input
      out DDRB,rmp
; Init Port A
      ldi rmp,0x03 ; Output pins port A
      out PORTA,rmp
      clr rmp ; All pins in port A to input
      out DDRA,rmp
      rcall BackDelay
```

```
      rcall BackDelay
; Debug code: measuring the lamp currents
.if debug_lampcurrent == 1
      ldi rmp,0b00000011 ; All LEDs in port A on
      out DDRA,rmp
      ldi rmp,0b11111111 ; All LEDs in port B on
      out DDRB,rmp
      ldi rmp,0b01111111 ; All LEDs in port D on
      out DDRD,rmp
Infinite:
      rjmp Infinite
      .endif
; Table init
      clr rFlag ; Clear all flags
      rcall InitCycle ; Start the output cycle
; Init Timer TC0
      ldi rmp,0xFF
      out OCR0A,rmp ; Compare A
      out OCR0B,rmp ; Compare B
      ldi         rmp,(1<<COM0A1)|(1<<COM0A0)|(1<<COM0B1)|(1<<COM0B0)|(1<<WGM01)|
(1<<WGM00)
      out TCCR0A,rmp
; Init Timer TC1
      clr rmp
      out OCR1AH,rmp ; MSB Compare A
      ldi rmp,0xFF
      out OCR1AL,rmp ; LSB Compare A
      clr rmp
      out OCR1BH,rmp ; MSB Compare B
      ldi rmp,0xFF
      out OCR1BL,rmp ; LSB Compare B
      ldi rmp,(1<<COM1A1)|(1<<COM1A0)|(1<<COM1B1)|(1<<COM1B0)|(1<<WGM10)
      out TCCR1A,rmp
      ; Timer starten
      ldi rmp,(1<<WGM12)|(1<<CS11) ; Timer 1, Prescale = 8, 8-Bit-PWM
      out TCCR1B,rmp
      ldi rmp,1<<CS01 ; Timer 0, Prescale = 8
      out TCCR0B,rmp
      ldi rmp,1<<TOIE0 ; Timer 0 Interrupt
      out TIMSK,rmp
; Sleep and interrupt
      ldi rmp,1<<SE ; Enable sleep mode idle
      out MCUCR,rmp
      sei
;
; =============================================
;        P R O G R A M   L O O P
; =============================================
;
Loop:
      sleep ; Go to sleep
      nop ; Dummy for wake-up
      sbrc rFlag,bEndCycle ; Skip if not end of cycle
      rcall Cycle ; Cycle end
      rjmp loop ; Back to loop
;
; Next cycle
;
; Control codes
.equ cEnd = 0xFF
.equ cRepeat1 = 0xFE
.equ cRepeat2 = 0xFD
.equ cNext1 = 0xFC
.equ cNext2 = 0xFB
```

```
.equ cControl = 0xF0
;
Cycle:
      lpm rmp,Z+ ; Read next table value (flags)
      cpi rmp,cControl ; Check if that is a control byte
      brcs Cycle1 ; No, normal cycle
      cpi rmp,cRepeat1 ; Repetition outer loop?
      breq CycleRepeat1 ; Yes, start outer repetition loop
      brcc InitCycle ; 0xFF = Restart
      cpi rmp,cNext1 ; Next value outer repetition loop?
      breq CycleNext1 ; Yes, repeat outer loop
      cpi rmp,cRepeat2 ; Inner repetition loop?
      breq CycleRepeat2 ; Yes, start inner loop
      cpi rmp,cNext2 ; Next value inner repetition loop?
      breq CycleNext2 ; Yes, repeat inner loop
      lpm rmp,Z+ ; Read next byte as dummy
      rjmp Cycle ; Read next cycle values
CycleNext2: ; Next inner repetition
      lpm rmp,Z+ ; Read dummy value
      dec rRep2 ; Decrease repetition counter
      breq Cycle ; If zero, continue
      mov ZH,YH ; Copy inner repetition address to current address
      mov ZL,YL
      rjmp Cycle ; Read next cycle values
CycleNext1: ; Next outer repetition
      lpm rmp,Z+ ; Read dummy value
      dec rRep1 ; Decrease repetition counter
      breq Cycle ; If zero continue at address
      mov ZH,XH ; Move stored address outer loop
      mov ZL,XL
      rjmp Cycle ; Read next cycle value
CycleRepeat1: ; Start outer repetition loop
      lpm rRep1,Z+ ; Read repetition counter value
      mov XH,ZH ; Copy address to X
      mov XL,ZL
      rjmp Cycle ; Read next cycle value
CycleRepeat2: ; Start inner repetition loop
      lpm rRep2,Z+ ; Read repetition counter value
      mov YH,ZH ; Copy address to Y
      mov YL,ZL
      rjmp Cycle ; Read next cycle value
InitCycle: ; Start a cycle
      ldi ZH,High(2*Tab) ; Z to table start
      ldi ZL,Low(2*Tab)
      rjmp Cycle ; Read first cycle value
Cycle1: ; Read next data set from table
      mov rFlag,rmp ; First byte to flag
      lpm rmp,Z+ ; Next byte to port D direction
      out DDRD,rmp
      lpm rmp,Z+ ; Next byte to port B direction
      out DDRB,rmp
      lpm rmp,Z+ ; Next byte to port A direction
      out DDRA,rmp
      ret ; Done
;
; Test all LEDs backwards
;
back:
      cbi PORTD,PORTD5
      sbi DDRD,DDD5
      cbi PORTB,PORTB2
      sbi DDRB,DDB2
      cbi PORTB,PORTB3
      sbi DDRB,DDB3
```

```
cbi PORTB,PORTB4
sbi DDRB,DDB4
; LEDs on
sbi DDRD,DDD0 ; L13
sbi PORTD,PORTD0
rcall BackDelay
cbi DDRD,DDD0
cbi PORTD,PORTD0
sbi DDRA,DDA1 ; L11
sbi PORTA,PORTA1
rcall BackDelay
cbi DDRA,DDA1
cbi PORTA,PORTA1
sbi DDRD,DDD1 ; L12
sbi PORTD,PORTD1
rcall BackDelay
cbi DDRD,DDD1
cbi PORTD,PORTD1
sbi DDRA,DDA0 ; L10
sbi PORTA,PORTA0
rcall BackDelay
cbi DDRA,DDA0
cbi PORTA,PORTA0
sbi DDRD,DDD2 ; L9
sbi PORTD,PORTD2
rcall BackDelay
cbi DDRD,DDD2
cbi PORTD,PORTD2
sbi DDRB,DDB0 ; L8
sbi PORTB,PORTB0
rcall BackDelay
cbi DDRB,DDB0
cbi PORTB,PORTB0
sbi DDRB,DDB0 ; L5
sbi PORTB,PORTB0
rcall BackDelay
cbi DDRB,DDB0
cbi PORTB,PORTB0
sbi DDRD,DDD6 ; L6
sbi PORTD,PORTD6
rcall BackDelay
cbi DDRD,DDD6
cbi PORTD,PORTD6
sbi DDRB,DDB5 ; L3
sbi PORTB,PORTB5
rcall BackDelay
cbi DDRB,DDB5
cbi PORTB,PORTB5
sbi DDRB,DDB1 ; L4
sbi PORTB,PORTB1
rcall BackDelay
cbi DDRB,DDB1
cbi PORTB,PORTB1
sbi DDRD,DDD3 ; L8
sbi PORTD,PORTD3
rcall BackDelay
cbi DDRD,DDD3
cbi PORTD,PORTD3
sbi DDRD,DDD4 ; L7
sbi PORTD,PORTD4
rcall BackDelay
cbi DDRD,DDD4
cbi PORTD,PORTD4
sbi DDRB,DDB6 ; L2
```

```
      sbi PORTB,PORTB6
      rcall BackDelay
      cbi DDRB,DDB6
      cbi PORTB,PORTB6
      sbi DDRB,DDB7 ; L1
      sbi PORTB,PORTB7
      rcall BackDelay
      cbi DDRB,DDB7
      cbi PORTB,PORTB7
      ldi rmp,0b11100011
      out PORTB,rmp
      ldi rmp,0b01011111
      out PORTD,rmp
      ldi rmp,0b00000011
      out PORTA,rmp
      ldi rmp,0b11111111
      out DDRB,rmp
      ldi rmp,0b01111111
      out DDRD,rmp
      ldi rmp,0b00000011
      out DDRA,rmp
      rcall BackDelay
      rcall BackDelay
      rcall BackDelay
      rcall BackDelay
      rcall BackDelay
      clr rmp
      out PORTB,rmp
      out PORTD,rmp
      out PORTA,rmp
      rcall BackDelay
;
      cbi PORTD,PORTD5
      sbi DDRD,DDD5
      cbi PORTB,PORTB2
      sbi DDRB,DDB2
      cbi PORTB,PORTB3
      sbi DDRB,DDB3
      cbi PORTB,PORTB4
      sbi DDRB,DDB4
;
      sbi DDRD,DDD0 ; L13
      sbi PORTD,PORTD0
      rcall BackDelay
      cbi DDRD,DDD0
      cbi PORTD,PORTD0
back1:
      ret

BackDelay:
.equ cDelay = 0
      ldi r25,High(cDelay)
      ldi r24,Low(cDelay)
BackDelay1:
      sbiw r24,1
      brne BackDelay1
      ret
;
; Cycle table
;
;
; Load constants
;
.include "constants.inc"
```

```
;
; Flags
.equ cUpOnly = 1<<bUpOnly
.equ cFast = 1<<bFast
.equ cVeryFast  = (1<<bVeryFast)|(1<<bFast)
;
Tab:
; One round slow
.set cX = X1
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X2
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X7
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X8
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X4
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X3
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X6
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X5
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X9
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X10
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X12
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X11
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X0
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
; Backwards fast
.set cX = cVeryFast + X13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X11
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X12
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X10
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X9
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X5
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X6
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X3
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X4
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X8
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X7
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X2
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = cVeryFast + X1
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X0
```

```
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
; Rotate two LEDs
.set cX = X1_2
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X2_7
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X7_8
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X4_8
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X3_6
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X5_6
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X5_9
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X9_10
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X10_12
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X11_12
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X11_13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X0
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
; Explode
.set cX = X13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X9to13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = Xall
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
; Implode
.set cX = X9to13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.set cX = X13
.db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
.db cRepeat1,5 ; Long pause
      .set cX = X0
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .db cNext1,0
; Build an S
.db cRepeat1,3 ; Outer loop 3 times
      .set cX = XS1
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XS2
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XS3
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XS4
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XS5
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XS6
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XS7
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XS8
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .db cRepeat2,5 ; Inner loop 5 times
            .set cX = XS
```

```
            .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
            .db cNext2,0
      .db cNext1,0
; Long pause
.db cRepeat1,10
      .set cX = X0
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .db cNext1,0
; Random numbers
      .set cX = XR1
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR2
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR3
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR4
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR5
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR6
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR7
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR8
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR9
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR10
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR11
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR12
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR13
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR14
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR15
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR16
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR17
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR18
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR19
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR20
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR21
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR22
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR23
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR24
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR25
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR26
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR27
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR28
```

```
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR29
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR30
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR31
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)
      .set cX = XR32
      .db Byte1(cX),Byte2(cX),Byte3(cX),Byte4(cX)

.db 0xFF,0xFF ; End of table, restart
;
; End of source code
; Copyright
.db "(C)2017 by Gerhard Schmidt  " ; human readable
.db "C(2)17 0ybG reahdrS hcimtd  " ; wordwise
;
```

constants.inc

```
; Definitions
.equ X0=1843200 ; 00.1C.20.00
.equ X1=10231808 ; 00.9C.20.00
.equ X2=6037504 ; 00.5C.20.00
.equ X3=3940352 ; 00.3C.20.00
.equ X4=1974272 ; 00.1E.20.00
.equ X5=1908736 ; 00.1D.20.00
.equ X6=1859584 ; 00.1C.60.00
.equ X7=1847296 ; 00.1C.30.00
.equ X8=1845248 ; 00.1C.28.00
.equ X9=1844224 ; 00.1C.24.00
.equ X10=18620416 ; 01.1C.20.00
.equ X11=35397632 ; 02.1C.20.00
.equ X12=1843712 ; 00.1C.22.00
.equ X13=1843456 ; 00.1C.21.00
.equ X1_2=14426112 ; 00.DC.20.00
.equ X1_3=12328960 ; 00.BC.20.00
.equ X1_4=10362880 ; 00.9E.20.00
.equ X1_5=10297344 ; 00.9D.20.00
.equ X1_6=10248192 ; 00.9C.60.00
.equ X1_7=10235904 ; 00.9C.30.00
.equ X1_8=10233856 ; 00.9C.28.00
.equ X1_9=10232832 ; 00.9C.24.00
.equ X1_10=27009024 ; 01.9C.20.00
.equ X1_11=43786240 ; 02.9C.20.00
.equ X1_12=10232320 ; 00.9C.22.00
.equ X1_13=10232064 ; 00.9C.21.00
.equ X2_3=8134656 ; 00.7C.20.00
.equ X3_4=4071424 ; 00.3E.20.00
.equ X4_5=2039808 ; 00.1F.20.00
.equ X5_6=1925120 ; 00.1D.60.00
.equ X6_7=1863680 ; 00.1C.70.00
.equ X7_8=1849344 ; 00.1C.38.00
.equ X8_9=1846272 ; 00.1C.2C.00
.equ X9_10=18621440 ; 01.1C.24.00
.equ X10_11=52174848 ; 03.1C.20.00
.equ X11_12=35398144 ; 02.1C.22.00
.equ X12_13=1843968 ; 00.1C.23.00
.equ X1_2_3=16523264 ; 00.FC.20.00
.equ X2_3_4=8265728 ; 00.7E.20.00
.equ X3_4_5=4136960 ; 00.3F.20.00
.equ X4_5_6=2056192 ; 00.1F.60.00
.equ X5_6_7=1929216 ; 00.1D.70.00
.equ X6_7_8=1865728 ; 00.1C.78.00
.equ X7_8_9=1850368 ; 00.1C.3C.00
.equ X8_9_10=18623488 ; 01.1C.2C.00
.equ X9_10_11=52175872 ; 03.1C.24.00
.equ X10_11_12=52175360 ; 03.1C.22.00
.equ X11_12_13=35398400 ; 02.1C.23.00
.equ X2_7=6041600 ; 00.5C.30.00
.equ X4_8=1976320 ; 00.1E.28.00
.equ X3_6=3956736 ; 00.3C.60.00
.equ X5_9=1909760 ; 00.1D.24.00
.equ X10_12=18620928 ; 01.1C.22.00
.equ X11_13=35397888 ; 02.1C.21.00
.equ X9to13=52176640 ; 03.1C.27.00
.equ Xall=67075840 ; 03.FF.7F.00
.equ XS1=6037504 ; 00.5C.20.00
.equ XS2=14426112 ; 00.DC.20.00
.equ XS3=14491648 ; 00.DD.20.00
.equ XS4=14492672 ; 00.DD.24.00
.equ XS5=14492928 ; 00.DD.25.00
.equ XS6=14493440 ; 00.DD.27.00
.equ XS7=14495488 ; 00.DD.2F.00
.equ XS8=14626560 ; 00.DF.2F.00
.equ XS=16723712 ; 00.FF.2F.00
.equ XR1=10248192 ; 00.9C.60.00
.equ XR2=2056192 ; 00.1F.60.00
.equ XR3=10232832 ; 00.9C.24.00
.equ XR4=43917312 ; 02.9E.20.00
.equ XR5=6037504 ; 00.5C.20.00
.equ XR6=8151040 ; 00.7C.60.00
.equ XR7=27013376 ; 01.9C.31.00
.equ XR8=1859840 ; 00.1C.61.00
.equ XR9=1843456 ; 00.1C.21.00
.equ XR10=10231808 ; 00.9C.20.00
.equ XR11=35397888 ; 02.1C.21.00
.equ XR12=54272000 ; 03.3C.20.00
.equ XR13=1847296 ; 00.1C.30.00
.equ XR14=41689600 ; 02.7C.22.00
.equ XR15=18637824 ; 01.1C.64.00
.equ XR16=18621440 ; 01.1C.24.00
.equ XR17=1843200 ; 00.1C.20.00
.equ XR18=14426112 ; 00.DC.20.00
.equ XR19=18751488 ; 01.1E.20.00
.equ XR20=1848064 ; 00.1C.33.00
.equ XR21=22881536 ; 01.5D.25.00
.equ XR22=8152064 ; 00.7C.64.00
.equ XR23=22831104 ; 01.5C.60.00
.equ XR24=1908736 ; 00.1D.20.00
.equ XR25=18703872 ; 01.1D.66.00
```

```
.equ XR26=1848320 ; 00.1C.34.00
.equ XR27=18688000 ; 01.1D.28.00
.equ XR28=1844224 ; 00.1C.24.00
.equ XR29=10232832 ; 00.9C.24.00
.equ XR30=1844224 ; 00.1C.24.00
.equ XR31=20783104 ; 01.3D.20.00
.equ XR32=1843968 ; 00.1C.23.00
```