Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт кибербезопасности и защиты информации

Отчёт
по лабораторной работе №4

ОРГАНИЗАЦИЯ АНАЛОГОВОГО ВВОДА-ВЫВОДА: АЦП И ШИМ

по дисциплине «Аппаратные средства вычислительной техники»

Выполнили: студенты группы
4831001/10003 _____ Г. А. Улановский
(подпись, дата)

_____ Г. Г. Фидаров
(подпись, дата)

Проверил: доцент, к.т.н. П. О. Семенов

_____
(подпись, дата)

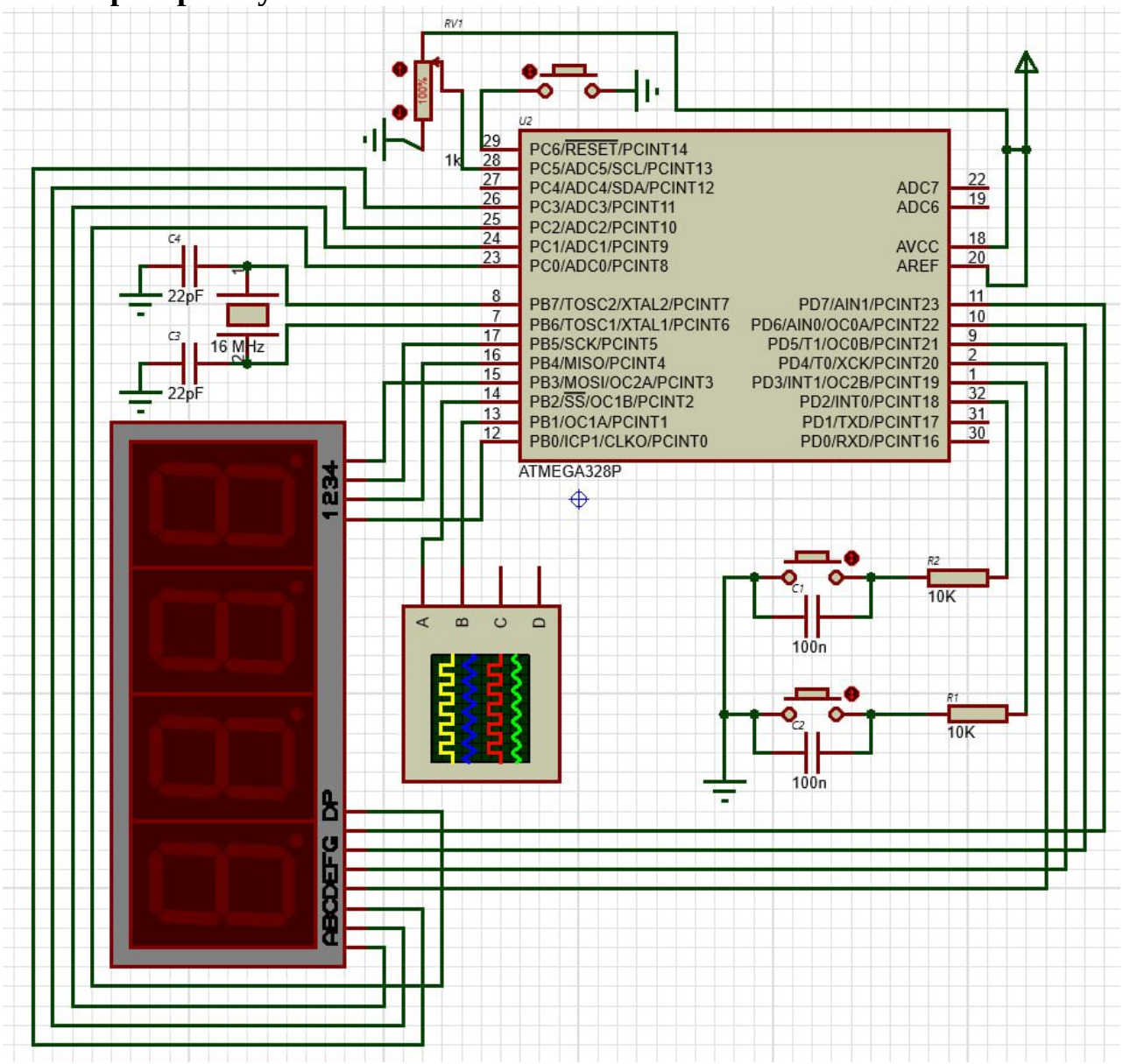
Алгоритм расчёта: MD5 Контрольная сумма:_____

Санкт-Петербург
2023

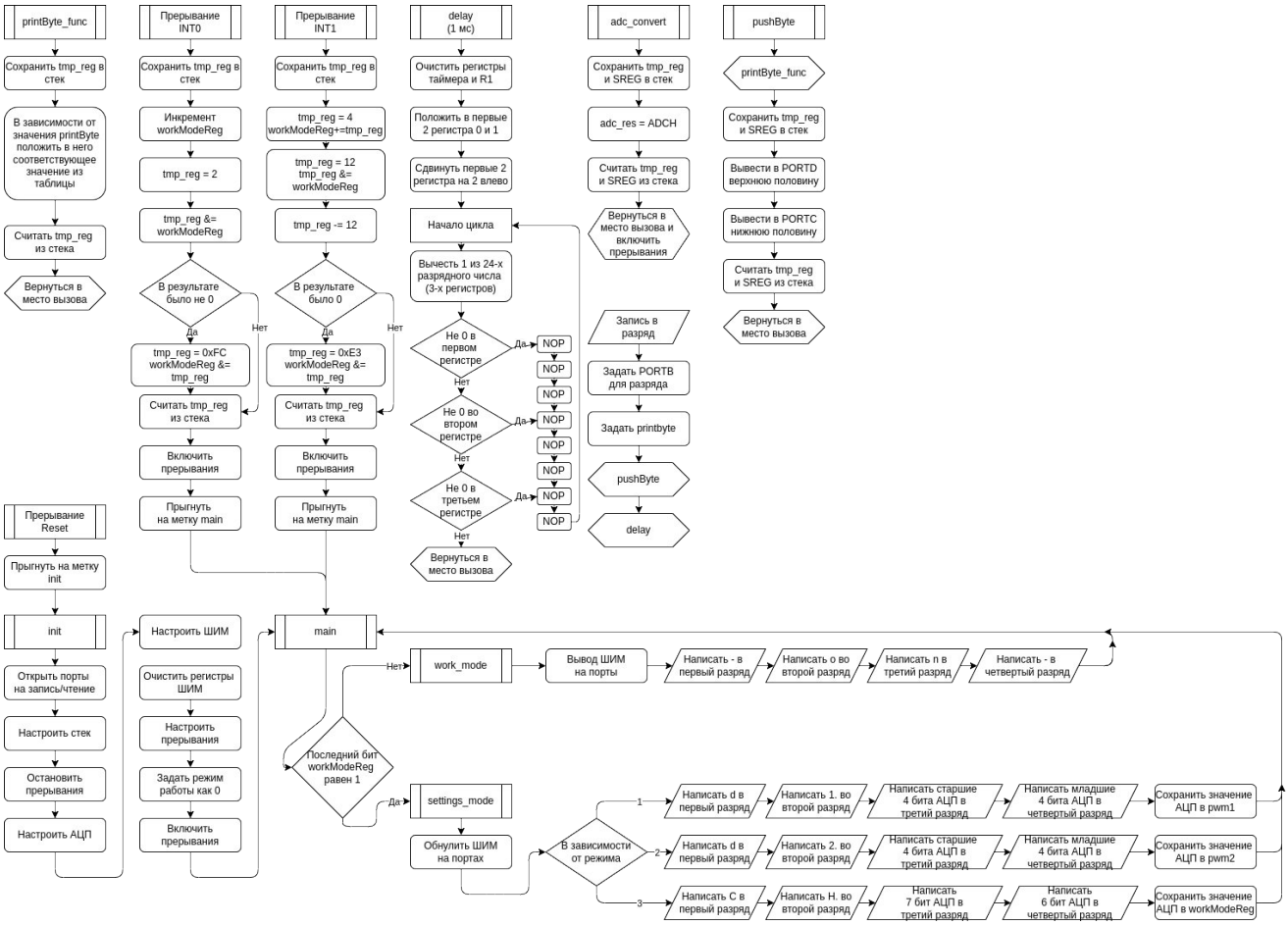
1 Формулировка задания

1. Программа должна осуществлять два режимы работы:
    a. демонстрация работы;
    b. настройка параметров работы.
2. Вывод PA5 должен быть настроен на ввод (DDRA=0xdf), для работы он игнорируется (изменения в логику работы не вносятся). Считывание значения аналогового сигнала должно производиться с помощью прерывания АЦП (ADC).
3. Переключение между режимами должно осуществляться циклически с помощью кнопки PD2 (прерывание INT0). Переключение между настраиваемыми параметрами должно осуществляться циклически с помощью кнопки PD3 (прерывание INT1). Имя изменяемого параметра должно отображаться на первых (одном, двух или трёх, в зависимости от параметра) семисегментных индикаторах, на последующих индикаторах должно отображаться значение соответствующего параметра в шестнадцатеричной системе счисления. На последнем индикаторе имени параметра в качестве разделителя должна гореть точка.
4. Изменение значений должно осуществляться с помощью потенциометра, подключённого к выводу PA5, следующим образом:
    a. крайнее левое положение соответствует нижней границе допустимого диапазона, крайнее правое – верхней;
    b. при повороте потенциометра значение на семисегментном индикаторе изменяется незамедлительно.

2 Схема лабораторной установки



3 Блок-схема алгоритма работы программы



4 Ответы на контрольные вопросы

1. Посредством каких регистров производится настройка АЦП?
    a. ADMUX – регистр настройки мультиплексора; В программе установлено на 0x65, т.е. 0b011*0101.
        i. 01 - Внешний источник питания на AVCC, с внешним конденсатором на AREF
        ii. 1 – Обратный порядок байт в регистрах ADCH и ADCL
        iii. 0101 – выбор ADC5 (A5 пин) на чтение
    b. ADCSRA – управляющий и статусный регистр; В программе установлено на 0x83, т.е. 0b10000011. В подпрограмме считывания АЦП происходит установка второго бита и ожидание его снятия
        i. 1 – включение/выключение АЦП.
        ii. 0 – запуск преобразования АЦП. В режиме одиночного преобразования нужно записать единицу в этот бит, чтобы начать преобразование. В режиме свободного запуска нужно записать единицу в этот бит, чтобы начать первое преобразование.
        iii. 0 – включение автоматического запуска АЦП. Когда этот бит записывается в единицу, включается автоматическая работа АЦП, то есть значения будут считываться постоянно.
        iv. 0 – флаг прерывания АЦП. Этот бит устанавливается, когда преобразование АЦП завершается и регистры данных обновляются.

v. 0 – активация прерывания АЦП.
vi. 011 – выбор предделителя АЦП, в данном случае – 8

c. ADCSRB – управляющий регистр; В данном регистре настраивается автоматическое считывание АЦП. Например, от прерываний таймеров или внешнего INTR0
d. ADCH и ADCL – регистры, в которых записывается значение АЦП.

2. В каких режимах может работать АЦП?
   a. Режим одиночного преобразования – в этом режиме АЦП может преобразовывать аналоговый сигнал с одного входа, после чего прекращает работу.
   b. Режим автозапуска – в этом режиме АЦП постоянно преобразует сигналы с заданного входа и выводит результаты в соответствующий регистр.
   c. Режим захвата – в этом режиме АЦП используется для захвата сигнала, если он превышает или находится ниже определенного уровня. Результаты преобразования выводятся в регистры ADCH и ADCL.

3. Какие порты и разряды портов микроконтроллера ATmega328p могут обрабатывать входящие аналоговые сигналы?
   a. По datasheet на микроконтроллер ATmega328P, входы для аналоговых сигналов находятся на портах C (ADC0-ADC5). Кроме того, порты C могут использоваться для работы с внешними прерываниями, включая возможность настройки на работу с изменениями уровня аналоговых сигналов.
   b. Компаратор доступен только на пинах PD6 (AIN0) и PD7 (AIN1).

4. Какими способами реализуется ШИМ?
   a. Аппаратный ШИМ (PWM):
      ATmega328P имеет три 8-битных таймера/счетчика (Timer0, Timer1, Timer2), которые могут генерировать широтно-импульсную модуляцию (PWM).
      Timer0 может быть использован для генерации ШИМ на пине OC0A (PD6) и OC0B (PD5), Timer2 - на пине OC2A (PB3) и OC2B (PD3), а Timer1 - на пинах OC1A (PB1) и OC1B (PB2). Для генерации ШИМ на каждом из этих пинов есть специальные регистры TCCR0A, TCCR2A и TCCR1A.
   b. Фазово-корректирующий режим (PWM Phase Correct):
      ATmega328P также поддерживает фазово-корректирующий режим ШИМ, который может быть использован для более точного управления сервоприводами и другими устройствами. В этом режиме ШИМ может быть генерируем на пинах OC1A и OC1B (Timer1), и на пинах OC2A и OC2B (Timer2). Для настройки этого режима есть специальные регистры TCCR1A и TCCR2A.
   c. ШИМ на программном уровне (Software PWM):

Также возможна генерация ШИМ на программном уровне при помощи использования обычных digital-пинов с помощью прерываний или искусственных задержек.

5. Как настроить ШИМ с помощью таймера-счётчика?

Настройка аппаратного ШИМ на ATmega328P с помощью таймера-счетчика производится путем настройки соответствующих регистров TCCRnA, TCCRnB, а также регистров сравнения OCRnA, OCRnB (где n – номер таймера)

5 Выводы по лабораторной работе

В ходе лабораторной работы были получены навыки работы с широтно-импульсной модуляцией, аналого-цифровым преобразователем ATMega328p, аппаратными прерываниями. Изучены принципы работы с 7-сегментным индикатором, кнопками и потенциометром.

Приложение 1

Комментированный листинг программы для МК на языке ассемблера

```
;
; AssemblerApplication1.asm
;
; Created: 06.02.2023 18:38:51
; Author : Georgul
;
.device atmega328p
.def    tmp_reg     = R16
.def    PORTB_out   = R17
.def    PORTC_out   = R18
.def    PORTD_out   = R19
.def    workModeReg = R20
.def    timer100Byte= R21
.def    timer010Byte= R22
.def    timer001Byte= R23
.def    printByte   = R24
.def    convertReg  = R25
.def    pwmOutReg   = R26
.def    pwm_reg_1   = R27
.def    pwm_reg_2   = R28
.def    adc_res     = R29
.org $000                   ; Reset Interrupt ADDR
    RJMP init               ; Reset Interrupt
.org 0x0002             ; Interrupt INT0 ADDR
    RJMP changeWorkMode     ; Interrupt INT0
.org 0x0004             ; Interrupt INT1 ADDR
    RJMP changeSettMode     ; Interrupt INT1
.org 0x002A                 ; Interrupt vector for ADC conversion complete
    RJMP ADC_interrupt
.org 0x0033
ADC_interrupt:
    PUSH tmp_reg            ; Save tmp to stack
    LDS tmp_reg, SREG       ; Save SREG
    PUSH tmp_reg            ; to stack
    LDS adc_res, ADCL       ; get value (ignore high byte)
    LDS adc_res, ADCH       ; Save ADC result in R16
    POP tmp_reg             ; load SREG
    STS SREG, tmp_reg       ; from stack
    POP tmp_reg             ; load tmp from stack
RETI                        ; Return from interrupt
init:
    LDI tmp_reg, 0xFF       ; Set B0-B5 (D8-D13) enable to write
    OUT DDRB, tmp_reg       ; Common cathode (D8, D11-D13)
                            ; D9-D10 for PWM
    LDI tmp_reg, 0xF0       ; Set D4-D7 ports enable to write
    OUT DDRD, tmp_reg       ; high byte of 7-segment data
    LDI tmp_reg, 0x0F       ; Set A0-A3 ports enable to write
    OUT DDRC, tmp_reg       ; low  byte of 7-segment data
    LDI tmp_reg, 0xC6       ; Set 0b**000110 to B port. 11 for PWM
    OUT PORTB, tmp_reg      ;
    LDI tmp_reg,low(RAMEND) ; Setup stack
    out SPL,tmp_reg
```

```
    LDI tmp_reg,high(RAMEND)
    out SPH,tmp_reg
    CLI                     ; Stop interrupts
    LDI tmp_reg, 0xEF       ; 0b10000011. Enable ADC, 011 - division by 8 (125 khz?)
    STS ADCSRA, tmp_reg
    LDI tmp_reg, 0x00       ; 0b00001000. Select ADC5 as interrupt source
    STS ADCSRB, tmp_reg     ; Set ADC5 as interrupt source
   LDI tmp_reg, 0x65      ; 0b01100101. 01 - AVcc with external capacitor at AREF pin
    STS ADMUX, tmp_reg      ; 1 - ADC Left Adjust Result
                            ; 0101 - ADC5 pin
    ;RCALL adc_convert      ; read ADC

    LDI tmp_reg, 0xA1       ; 0b10100001, FAST PWM 8-bit
    STS TCCR1A, tmp_reg     ; Clear OC1A/OC1B on compare match,
                            ; set OC1A/OC1B at BOTTOM (non-inverting mode)
    LDI tmp_reg, 0x09       ; 0b00001001 No clock prescaling
    STS TCCR1B, tmp_reg
    CLR pwm_reg_1           ; clear
    CLR pwm_reg_2           ; pwm
    CLR pwmOutReg           ; regs
; в регистре MCUCR для ATmega328P на самом деле находятся следующие биты:
; IVSEL: Выбор вектора прерывания. Если этот бит установлен, вектор прерывания будет
расположен на старшем адресе Flash,
; если сброшен - на начальном адресе Flash.
; IVCE: Разрешение доступа к IVSEL. Если этот бит установлен, доступ к IVSEL разрешен.
; BODS: Управление функцией Brown-out Detection (BOD) - выбор источника опорного
напряжения для BOD.
; BODSE: Разрешение изменения значения BODS. Если этот бит установлен вместе с BODS,
то значение BODS может быть изменено.
; PUD: Управление внутренним подтягивающим резистором для пинов ввода/вывода порта B.
; Таким образом, регистр MCUCR в контексте прерываний на микроконтроллере ATmega328P
отвечает за выбор вектора прерывания,
; настройку функции BOD, а также за управление подтягивающими резисторами для пинов
ввода/вывода порта B.
; Он не отвечает за настройку внешних или внутренних прерываний, как было указано ранее.

    LDI tmp_reg, 0x00       ; Set MCUCR (???)
    OUT MCUCR, tmp_reg      ;
    LDI tmp_reg, 0x03       ; Enable interrupts on INT0 and INT1
    OUT EIMSK, tmp_reg      ;
    OUT EIFR, tmp_reg       ; Avoid interrupt on awake (SEI)
    LDI tmp_reg, 0x0A       ;
    STS EICRA, tmp_reg      ; FALLEN to intr setup

    LDI workModeReg, 0      ; Set workModeReg to 0 by default
    SEI                     ; Start interrupts
;                           ; End init
main:
    LDI tmp_reg, 1          ; if mode is 1
    AND tmp_reg, workModeReg
    BREQ work_mode          ; goto WorkMode
    RJMP settings_mode      ; else goto SettingsMode
work_mode:
```

```
    CLR R1                      ; Clear R1 (zero register)
    STS OCR1AH, R1              ; clear high D10
    STS OCR1AL, R1              ; clear low  D10
    SBRC pwmOutReg, 0
    STS OCR1AL, pwm_reg_1       ; push D10 pwm

    STS OCR1BH, R1              ; clear high D11
    STS OCR1BL, R1              ; clear low  D11
    SBRC pwmOutReg, 1
    STS OCR1BL, pwm_reg_2       ; push D11 pwm
    ; 1-s digit for PWM
    LDI PORTB_out, 0b00000111   ; Open 1-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 18           ; print -
    RCALL pushByte              ; push  -
    RCALL delay_setup           ; delay
    ; 2-d digit for PWM
    LDI PORTB_out, 0b00001110   ; Open 2-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 16           ; print O
    RCALL pushByte              ; push  O
    RCALL delay_setup           ; delay
    ; 3-d digit for PWM
    LDI PORTB_out, 0b00010110   ; Open 3-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 17           ; print N
    RCALL pushByte              ; push  N
    RCALL delay_setup           ; delay
    ; 4-d digit for PWM
    LDI PORTB_out, 0b00100110   ; Open 4-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 18           ; print -
    RCALL pushByte              ; push  -
    RCALL delay_setup           ; delay
RJMP main
settings_mode:
    CLR R1                      ; Clear R1 (zero register)
    STS OCR1AH, R1              ; clear D10
    STS OCR1AL, R1              ;
    STS OCR1BH, R1              ; clear D11
    STS OCR1BL, R1              ;
    OUT PORTC, R1               ; clear C
    LDI tmp_reg, 0x0F           ; clear D
    OUT PORTD, tmp_reg          ; (without intr)
    LDI tmp_reg, 9              ; 0b****1001
    CP workModeReg, tmp_reg
    BRSH mode_2
    LDI tmp_reg, 8              ; 0b****1000
    CP workModeReg, tmp_reg
    BRSH mode_2
    LDI tmp_reg, 5              ; 0b****0101
    CP workModeReg, tmp_reg
    BRSH mode_1
    LDI tmp_reg, 4              ; 0b****0100
```

```
    CP workModeReg, tmp_reg
    BRSH mode_1
    LDI tmp_reg, 1              ; 0b****0001
    CP workModeReg, tmp_reg
    BRSH mode_3
    LDI tmp_reg, 0              ; 0b****0000
    CP workModeReg, tmp_reg
    BRSH mode_3
mode_1:
    ;RCALL adc_convert          ; get ADC
    MOV pwm_reg_1, adc_res
    ;MOV tmp_reg, adc_res
    ;STS OCR1AL, tmp_reg        ; push D10 pwm
    ;LDI tmp_reg, 0xFF
    ;EOR tmp_reg, adc_res
    ;STS OCR1BL, tmp_reg        ; push D11 pwm
    ; 1-s digit for PWM
    LDI PORTB_out, 0b00000111   ; Open 1-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 13           ; print d
    RCALL pushByte              ; push  d
    RCALL delay_setup           ; delay
    ; 2-d digit for PWM
    LDI PORTB_out, 0b00001110   ; Open 2-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 19           ; print 1.
    RCALL pushByte              ; push  1.
    RCALL delay_setup           ; delay
    RJMP mode_12_end

mode_2:
    ;RCALL adc_convert          ; get ADC
    MOV pwm_reg_2, adc_res

    ;MOV tmp_reg, adc_res
    ;STS OCR1AL, tmp_reg        ; push D10 pwm
    ;LDI tmp_reg, 0xFF
    ;EOR tmp_reg, adc_res
    ;STS OCR1BL, tmp_reg        ; push D11 pwm
    ; 1-s digit for PWM
    LDI PORTB_out, 0b00000001   ; Open 1-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 13           ; print d
    RCALL pushByte              ; push  d
    RCALL delay_setup           ; delay
    ; 2-d digit for PWM
    LDI PORTB_out, 0b00001000   ; Open 2-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 20           ; print 2.
    RCALL pushByte              ; push  2.
    RCALL delay_setup           ; delay

mode_12_end:
```

```
    ; setup out
    MOV tmp_reg, adc_res        ; convert to 0-15
    ANDI tmp_reg, 0xF0
    LSR tmp_reg
    LSR tmp_reg
    LSR tmp_reg
    LSR tmp_reg
    ; 3-d digit for PWM
    LDI PORTB_out, 0b00010000   ; Open 3-d digit
    OUT PORTB, PORTB_out        ;
    MOV printByte, tmp_reg      ; print digit
    RCALL pushByte              ; push  digit
    RCALL delay_setup           ; delay
    ; setup out
    MOV tmp_reg, adc_res        ; convert to 0-15
    ANDI tmp_reg, 0x0F
    ; 4-d digit for PWM
    LDI PORTB_out, 0b00100000   ; Open 4-d digit
    OUT PORTB, PORTB_out        ;
    MOV printByte, tmp_reg      ; print digit
    RCALL pushByte              ; push  digit
    RCALL delay_setup           ; delay
    RJMP main
mode_3:
    ;RCALL adc_convert          ; get ADC
    ;MOV tmp_reg, adc_res
    ;STS OCR1AL, tmp_reg        ; push D10 pwm
    ;LDI tmp_reg, 0xFF
    ;EOR tmp_reg, adc_res
    ;STS OCR1BL, tmp_reg        ; push D11 pwm
    ; 1-s digit for PWM
    LDI PORTB_out, 0b00000001   ; Open 1-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 12           ; print C
    RCALL pushByte              ; push  C
    RCALL delay_setup           ; delay
    ; 2-d digit for PWM
    LDI PORTB_out, 0b00001000   ; Open 2-d digit
    OUT PORTB, PORTB_out        ;
    LDI printByte, 21           ; print H.
    RCALL pushByte              ; push  H.
    RCALL delay_setup           ; delay

    ; setup out
    CLR printByte
    BST adc_res, 7              ; look for 7-bit (128-255)
    ; 3-d digit for PWM
    LDI PORTB_out, 0b00010000   ; Open 3-d digit
    OUT PORTB, PORTB_out        ;
    BLD printByte, 0            ; print bit
    RCALL pushByte              ; push  bit
    RCALL delay_setup
    ; setup out
    CLR printByte
```

```
    BST adc_res, 6              ; look for 6-bit (64-127 and 192-255)
    ; 4-d digit for PWM
    LDI PORTB_out, 0b00100000   ; Open 4-d digit
    OUT PORTB, PORTB_out        ;
    BLD printByte, 0            ; print bit
    RCALL pushByte              ; push  bit
    RCALL delay_setup
    MOV pwmOutReg, adc_res      ; set 6-7 bit to reg
    LSR pwmOutReg
    LSR pwmOutReg
    LSR pwmOutReg
    LSR pwmOutReg
    LSR pwmOutReg
    LSR pwmOutReg
    RJMP main
pushByte:
    RCALL printByte_func        ; convert print
    PUSH tmp_reg                ; Save tmp to stack
    LDS tmp_reg, SREG           ; Save SREG
    PUSH tmp_reg                ; to stack

    MOV PORTD_out, printByte    ; copy
    ANDI PORTD_out, 0xF0        ; get highest half
    LDI tmp_reg, 0x0F           ; add const
    ADD PORTD_out, tmp_reg      ; 0b****1111 for intr
    OUT PORTD, PORTD_out        ; push D
    MOV PORTC_out, printByte    ; copy
    ANDI PORTC_out, 0x0F        ; get lowest half
    OUT PORTC, PORTC_out        ; push C
    POP tmp_reg                 ; load SREG
    STS SREG, tmp_reg           ; from stack
    POP tmp_reg                 ; load tmp from stack
RET
changeWorkMode:
    PUSH tmp_reg            ; Send tmp to Stack
    inc workModeReg         ; wm++
    LDI tmp_reg, 0x02       ; compare with 0b00000010
    AND tmp_reg, workModeReg; if second bit is down
    BREQ changeWorkMode_exit; goto and
    LDI tmp_reg, 0xFC       ; else erase last 2 bits
    AND workModeReg, tmp_reg
changeWorkMode_exit:
    POP tmp_reg             ; Read tmp from Stack
    SEI                     ; Attach intr
RJMP main                   ; goto main
changeSettMode:
    PUSH tmp_reg            ; Send tmp to Stack
    LDI tmp_reg, 0b00000100 ; set 4 for adding
    ADD workModeReg, tmp_reg; wm+=4
    LDI tmp_reg, 0x0C       ; compare with 0b0000_11_00
    AND tmp_reg, workModeReg; get bits of settmode
    SUBI tmp_reg, 0x0C      ; check if not 0b****1100
    BRNE changeSettMode_exit; goto and
    LDI tmp_reg, 0xE3       ; else erase 3-5 bits
```

```
    AND workModeReg, tmp_reg
changeSettMode_exit:
    POP tmp_reg             ; Read tmp from Stack
    SEI                     ; Attach intr
RJMP main                   ; goto main
printByte_func:             ; many IFs, no to comment
    PUSH tmp_reg
    LDI tmp_reg, 21
    CP printByte, tmp_reg
    BRSH print_H
    LDI tmp_reg, 20
    CP printByte, tmp_reg
    BRSH print_2dot
    LDI tmp_reg, 19
    CP printByte, tmp_reg
    BRSH print_1dot
    LDI tmp_reg, 18
    CP printByte, tmp_reg
    BRSH print_minus
    LDI tmp_reg, 17
    CP printByte, tmp_reg
    BRSH print_N
    LDI tmp_reg, 16
    CP printByte, tmp_reg
    BRSH print_O
    LDI tmp_reg, 15
    CP printByte, tmp_reg
    BRSH print_F
    LDI tmp_reg, 14
    CP printByte, tmp_reg
    BRSH print_E
    LDI tmp_reg, 13
    CP printByte, tmp_reg
    BRSH print_D
    LDI tmp_reg, 12
    CP printByte, tmp_reg
    BRSH print_C
    LDI tmp_reg, 11
    CP printByte, tmp_reg
    BRSH print_B
    LDI tmp_reg, 10
    CP printByte, tmp_reg
    BRSH print_A
    LDI tmp_reg, 9
    CP printByte, tmp_reg
    BRSH print_9
    LDI tmp_reg, 8
    CP printByte, tmp_reg
    BRSH print_8
    LDI tmp_reg, 7
    CP printByte, tmp_reg
    BRSH print_7
    LDI tmp_reg, 6
    CP printByte, tmp_reg
```

```
    BRSH print_6
    LDI tmp_reg, 5
    CP printByte, tmp_reg
    BRSH print_5
    LDI tmp_reg, 4
    CP printByte, tmp_reg
    BRSH print_4
    LDI tmp_reg, 3
    CP printByte, tmp_reg
    BRSH print_3
    LDI tmp_reg, 2
    CP printByte, tmp_reg
    BRSH print_2
    LDI tmp_reg, 1
    CP printByte, tmp_reg
    BRSH print_1
    LDI tmp_reg, 0
    CP printByte, tmp_reg
    BRSH print_0
print_H:
    LDI printByte, 0xED
    RJMP label_ret
print_2dot:
    LDI printByte, 0xB7
    RJMP label_ret
print_1dot:
    LDI printByte, 0x0D
    RJMP label_ret
print_minus:
    LDI printByte, 0x80
    RJMP label_ret
print_N:
    LDI printByte, 0xA8
    RJMP label_ret
print_O:
    LDI printByte, 0xB8
    RJMP label_ret
print_F:
    LDI printByte, 0xE2
    RJMP label_ret
print_E:
    LDI printByte, 0xF2
    RJMP label_ret
print_D:
    LDI printByte, 0xBC
    RJMP label_ret
print_C:
    LDI printByte, 0x72
    RJMP label_ret
print_B:
    LDI printByte, 0xF8
    RJMP label_ret
print_A:
    LDI printByte, 0xEE
```

```
    RJMP label_ret
print_9:
    LDI printByte, 0xDE
    RJMP label_ret
print_8:
    LDI printByte, 0xFE
    RJMP label_ret
print_7:
    LDI printByte, 0x0E
    RJMP label_ret
print_6:
    LDI printByte, 0xFA
    RJMP label_ret
print_5:
    LDI printByte, 0xDA
    RJMP label_ret
print_4:
    LDI printByte, 0xCC
    RJMP label_ret
print_3:
    LDI printByte, 0x9E
    RJMP label_ret
print_2:
    LDI printByte, 0xB6
    RJMP label_ret
print_1:
    LDI printByte, 0x0C
    RJMP label_ret
print_0:
    LDI printByte, 0x7E
    RJMP label_ret
label_ret:
    POP tmp_reg
RET

delay_setup:
   CLR timer100Byte
   CLR timer010Byte
   CLR timer001Byte
   CLR R1
   LDI timer100Byte, 0          ; here is the hihest byte
   LDI timer010Byte, 1          ; here is the lowest byte
   LSL timer100Byte             ; << 2 left to have 10 clear bits at right
   BST timer010Byte, 7          ; 14 left to set time
   BLD timer100Byte, 0          ;
   LSL timer010Byte
   LSL timer100Byte
   BST timer010Byte, 7
   BLD timer100Byte, 0
   LSL timer010Byte
delay_cycle:
   SUBI timer001Byte, 1     ; 1 tick
   SBCI timer010Byte, 0     ; 1 tick
   SBCI timer100Byte, 0     ; 1 tick
```

```
    CPSE timer100Byte, R1     ; 1 ticks, if equal then skip (2 ticks)
    RJMP wait_nop_8           ; 2 ticks
    CPSE timer010Byte, R1     ; 1 ticks, if equal then skip (2 ticks)
    RJMP wait_nop_5           ; 2 ticks
    CPSE timer001Byte, R1     ; 1 ticks, if equal then skip (2 ticks)
    RJMP wait_nop_2           ; 2 ticks
    NOP
    NOP
    NOP
RET                           ; go back, 4 ticks
wait_nop_8:
    NOP
    NOP
    NOP
wait_nop_5:
    NOP
    NOP
    NOP
wait_nop_2:
    NOP
    NOP
RJMP   delay_cycle
```