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**

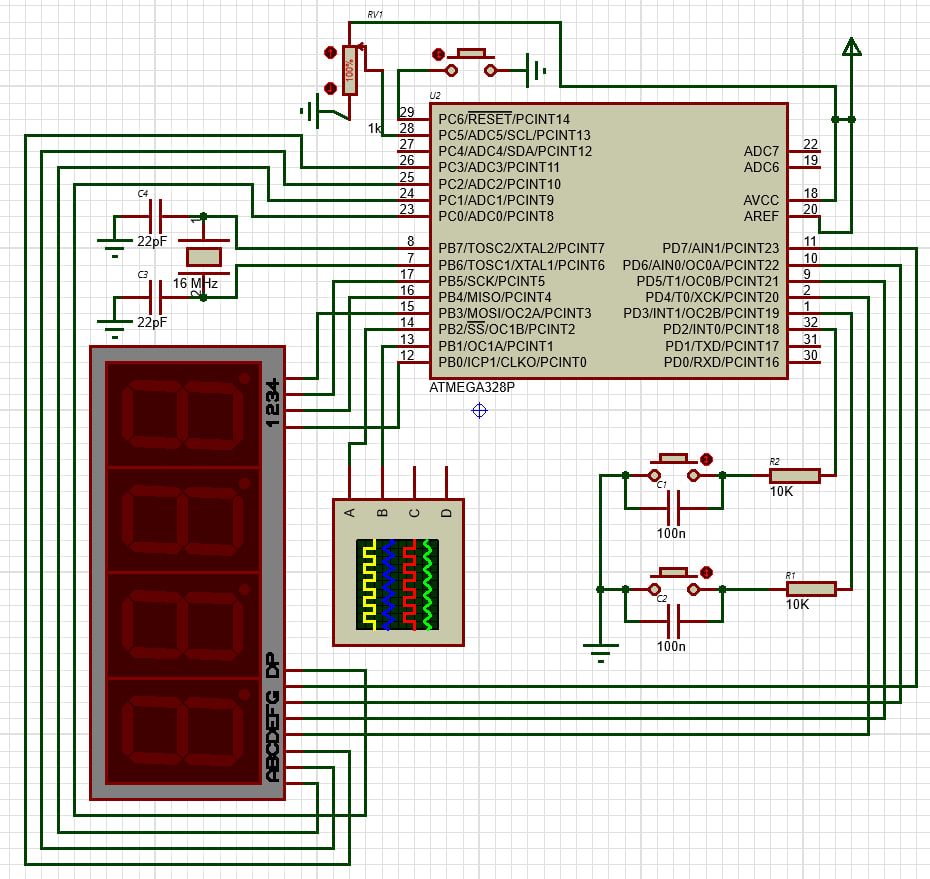
по дисциплине «Аппаратные средства вычислительной техники»

|  |  |  |  |
| --- | --- | --- | --- |
| 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) |  |

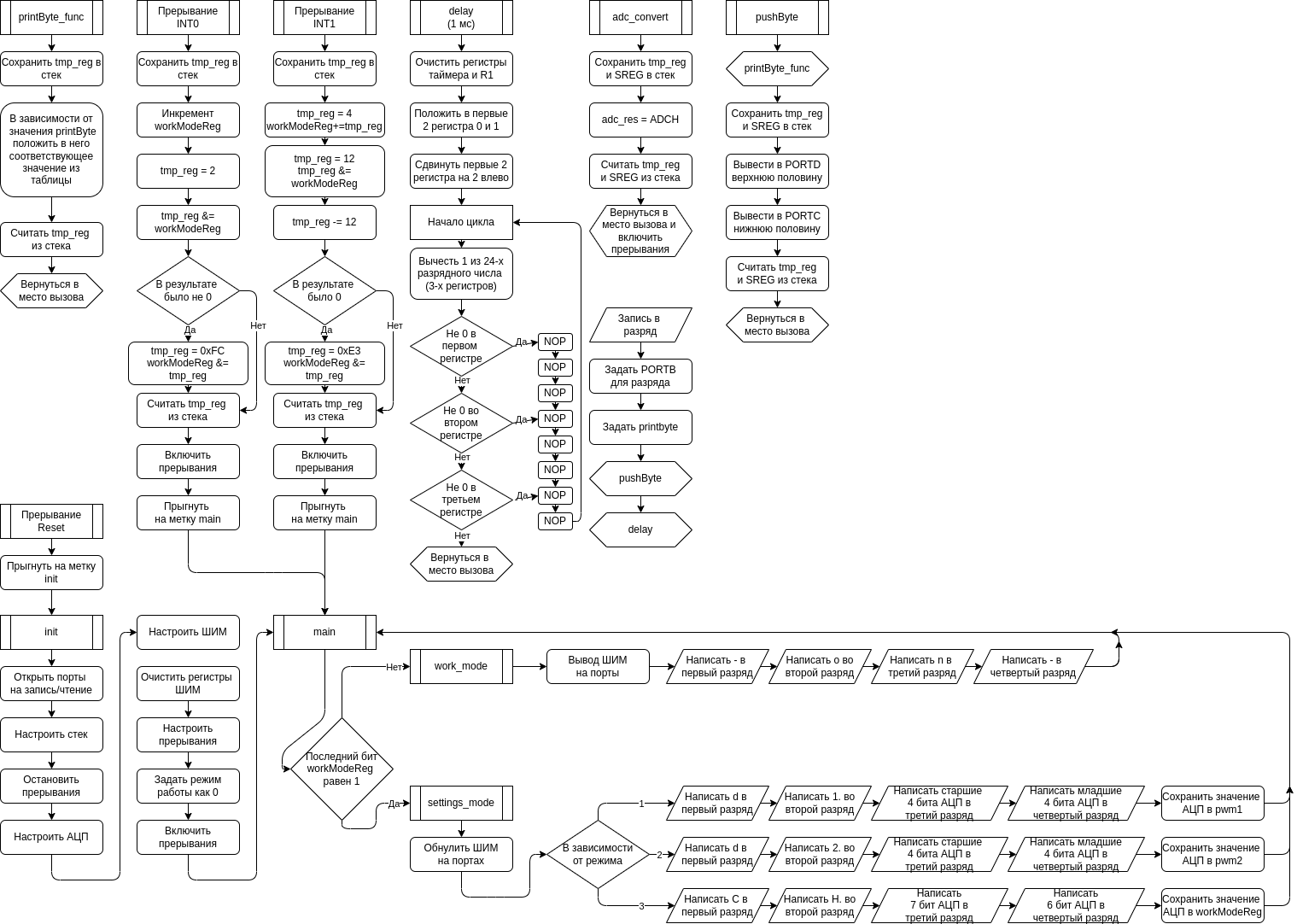
# Формулировка задания

1. Программа должна осуществлять два режимы работы:
   1. демонстрация работы;
   2. 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:
   1. крайнее левое положение соответствует нижней границе допустимого диапазона, крайнее правое – верхней;
   2. при повороте потенциометра значение на семисегментном индикаторе изменяется незамедлительно.

# Схема лабораторной установки



# Блок-схема алгоритма работы программы



# Ответы на контрольные вопросы

1. Посредством каких регистров производится настройка АЦП?
   1. ADMUX – регистр настройки мультиплексора; В программе установлено на 0x65, т.е. 0b011\*0101.
      1. 01 - Внешний источник питания на AVCC, с внешним конденсатором на AREF
      2. 1 – Обратный порядок байт в регистрах ADCH и ADCL
      3. 0101 – выбор ADC5 (A5 пин) на чтение
   2. 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
      1. 1 – включение/выключение АЦП.
      2. 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.
      3. 0 – включение автоматического запуска АЦП. Когда этот бит записывается в единицу, включается автоматическая работа АЦП, то есть значения будут считываться постоянно.
      4. 0 – флаг прерывания АЦП. Этот бит устанавливается, когда преобразование АЦП завершается и регистры данных обновляются.
      5. 0 – активация прерывания АЦП.
      6. 011 – выбор предделителя АЦП, в данном случае – 8
   3. ADCSRB – управляющий регистр; В данном регистре настраивается автоматическое считывание АЦП. Например, от прерываний таймеров или внешнего INTR0
   4. ADCH и ADCL – регистры, в которых записывается значение АЦП.
2. В каких режимах может работать АЦП?
   1. Режим одиночного преобразования – в этом режиме АЦП может преобразовывать аналоговый сигнал с одного входа, после чего прекращает работу.
   2. Режим автозапуска – в этом режиме АЦП постоянно преобразует сигналы с заданного входа и выводит результаты в соответствующий регистр.
   3. 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?
   1. По datasheet на микроконтроллер ATmega328P, входы для аналоговых сигналов находятся на портах С (ADC0-ADC5). Кроме того, порты С могут использоваться для работы с внешними прерываниями, включая возможность настройки на работу с изменениями уровня аналоговых сигналов.
   2. Компаратор доступен только на пинах PD6 (AIN0) и PD7 (AIN1).
4. Какими способами реализуется ШИМ?
   1. 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.

* 1. 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.

* 1. 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.

1. 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)

# 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