1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. **Кафедра «Высшая школа кибербезопасности»**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «ОРГАНИЗАЦИЯ ЦИФРОВОГО ВВОДА-ВЫВОДА»
2. по дисциплине «Аппаратные средства вычислительной техники»
3. Вариант 9а
4. Выполнил
5. студент гр. 5131001/30003 Шевчук Н.Е.

<*подпись*>

Преподаватель Макаров А.С.

<*подпись*>

1. Санкт-Петербург
2. 2025

**Оглавление**

[1 Цель работы 3](#_Toc194568209)

[2 Ход Работы 3](#_Toc194568244)

[2.1 Схема установки 3](#_Toc194568245)

[2.2 Блок-схема реализованного алгоритма 4](#_Toc194568246)

[3 Ответы на контрольные вопросы 4](#_Toc194568247)

[4 Вывод 7](#_Toc194568248)

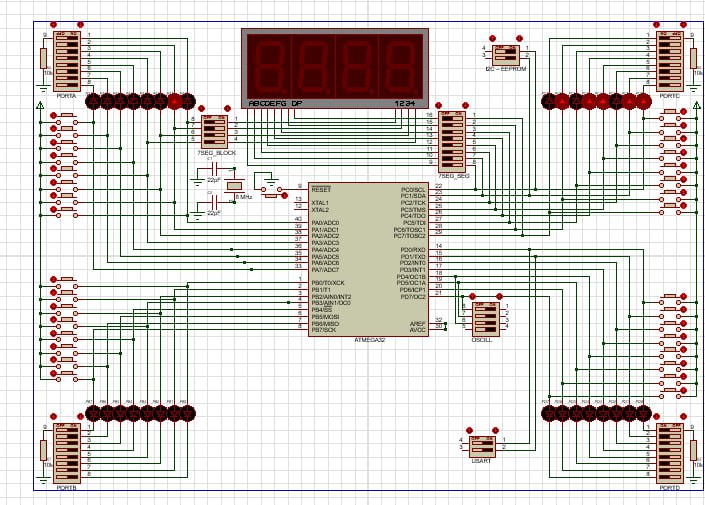
[Приложение 1 7](#_Toc194568249)

# Цель работы

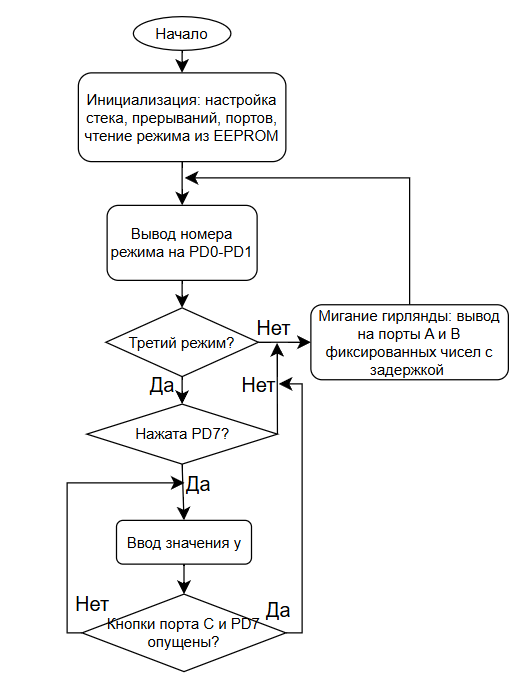
Изучить основы работы с цифровыми портами ввода-вывода микроконтроллера ATmega32. Получить практические навыки по обработке внешних прерываний и организации ввода-вывода с помощью механизма прерываний.

# Ход Работы

## Схема установки



## Блок-схема реализованного алгоритма



На рисунке выше представлена блок-схема реализованного алгоритма. Для циклического переключения между режимами использовались прерывания INT0 и INT1. Считывание и запись данных происходит по адресу 0x0F в EEPROM. С листингом реализованного кода можно ознакомиться в приложении 1.

# Ответы на контрольные вопросы

1. *Какими способами можно подключить внешние устройства (светодиод, кнопку) к микроконтроллеру?*

Светодиоды подключаются с помощью токоограничивающего резистра, чтобы избежать сгорания светодиода, двумя способами: подключение на землю и подключение на питание.

При подключении светодиода через землю – светодиод будет гореть в том случае, если на выходе порта записана логическая единица. При подключении светодиода на питание – светодиод будет гореть в том случае, если на входе порта будет записан логический ноль.

Кнопка подключается с помощью подтягивающего резистора. Тогда, когда кнопка отпущена, с входа микроконтроллера будет считываться логическая единица, т.к. вход подтянут резистором к линии питания. Когда кнопка нажата, то линия питания соединяется с землей через резистор.

1. *Как реализуется подсистема прерываний в микроконтроллере AVR?*

Все микроконтроллеры AVR имеют многоуровневую системупрерываний. У каждого периферийного устройства, что входит в состав AVR микроконтроллеров, есть как минимум один источник прерывания (Interrupt source). За каждым прерыванием, строго закреплён вектор (ссылка) указывающий на процедуру обработки прерывания (Interrupt service routine). Все векторы прерываний, располагаются в самом начале памяти программ и вместе формируют «таблицу векторов прерываний» (Interrupt vectors table).

1. *Как программно разрешить или запретить выполнение конкретного прерывания?*

Каждому прерыванию соответствует определённый «бит активации прерывания» (Interrupt Enable bit). Таким образом, чтобы использовать определённое прерывание, следует записать в его «бит активации прерывания» – логическую единицу. Далее, независимо от того активированы ли определённые прерывания, микроконтроллер не начнёт обработку этих прерываний, пока в «бит всеобщего разрешения прерываний» (Global Interrupt Enable bit в регистре состояния SREG) не будет записана логическая единица. Также, чтобы запретить все прерывания (на неопределённое время), в бит всеобщего разрешения прерываний следует записать – логический ноль.

1. *Какие источники прерываний есть в микроконтроллерах AVR?*

Ниже приведён перечень источников прерываний для МК ATmega32:

* RESET – Сигнал сброса.
* INT0 – Внешний запрос на прерывание по входу INT0
* INT1 – Внешний запрос на прерывание по входу INT1
* INT2 – Внешний запрос на прерывание по входу INT2
* Прерывания для работы с таймером:
* TIMER2\_COMP – Совпадение с регистром сравнения таймера T/C2
* TIMER2\_OVF ­– Переполнение счётчика T/C2
* TIMER1\_CAPT – Захват по таймеру T/C1
* TIMER1\_COMPA – Совпадение с регистром сравнения A таймера T/C1
* TIMER1\_COMPB – Совпадение с регистром сравнения B таймера T/C1
* TIMER1\_OVF – Переполнение счётчика T/C1
* TIMER0\_COMP – Совпадение с регистром сравнения таймера T/C0
* TIMER0\_OVF – Переполнение счётчика T/C0
* Прерывания для работы с интерфейсами:
* SPI\_STC – Передача данных по интерфейсу SPI завершена
* UART\_RXC – Приём данных приёмопередатчиком UART завершён
* UART\_UDRE – Регистр данных UART пуст
* UART\_TXC – Передача данных приёмопередатчиком UART завершена
* TWI – Прерывание от интерфейса I2C
* ADC – Завершено преобразование АЦП
* EE\_RDY – EEPROM готов
* ANA\_COMP – Прерывание от аналогового компаратора
* SPM\_RDY – Запись программной памяти (Flash) готова

1. *Как настраиваются внешние прерывания*?

Управление внешними прерывания реализуется при помощи регистров:

* GICR (GIMSK) – запрет/разрешение прерываний по сигналам на входах INT0, INT1;
* MCUCR – выбор условия срабатывания прерываний int0 и int1;
* GIFR – управление внешними прерываниями.

# Вывод

В ходе лабораторной работы были изучены основы работы с цифровыми портами ввода-вывода микроконтроллера ATmega32. Также были получены практические навыки по обработке внешних прерываний и организации ввода-вывода с помощью механизма прерываний. Была реализована и протестирована программа на ассемблере, осуществляющая мигание “новогодней гирлянды” с помощью выводов портов платы.

# Приложение 1

Листинг реализованного кода:

; Определения регистров

.def ZERO = R18 ; Нулевое значение

.def PIN\_D = R19 ; Хранит значение PIND для чтения состояния порта D

.def TMP = R20 ; Временный регистр для общих операций

.def MODE = R21 ; Текущий режим работы (1, 2 или 3)

.def STATE1 = R23 ; Первое состояние гирлянды (0xFF, 0xAA или y)

.def STATE2 = R24 ; Второе состояние гирлянды (0x00, 0x55 или -y)

.def YVAL = R25 ; Хранит значение y

; Векторы прерываний

.org $000

JMP reset

.org INT0addr

JMP INT0\_HANDLER

.org INT1addr

JMP INT1\_HANDLER

; Инициализация микроконтроллера

RESET:

; Инициализация регистров

LDI ZERO, 0x00 ; Устанавливаем нулевое значение

LDI TMP, 0x01 ; Начальное значение для R0

LDI R19, 0x03

MOV R0, TMP

CLR TMP

MOV R1, TMP ; Обнуляем R1-R3

MOV R2, TMP

MOV R3, TMP

; Настройка портов

LDI MODE, 1 ; Начальный режим = 1

SER TMP ; Устанавливаем все биты в 1 (0xFF)

OUT DDRA, TMP ; PORTA как выход (светодиоды)

OUT DDRB, TMP ; PORTB как выход (светодиоды)

CLR TMP

OUT DDRC, TMP ; PORTC как вход (для ввода y)

LDI R16, 0x0F ; Адрес EEPROM (0x0F)

LDI R17, 0x0F

LDI TMP, 0b00000011

OUT DDRD, TMP ; PD0-PD1 как выходы (для номера режима), остальные — входы

; Настройка стека

LDI TMP,HIGH(RAMEND)

OUT SPH, TMP

LDI TMP, LOW(RAMEND)

OUT SPL, TMP

; Настройка прерываний

LDI TMP, 0x0F ; INT0 и INT1 срабатывают по нарастающему фронту

OUT MCUCR, TMP

LDI TMP, 0xC0 ; Разрешаем INT0 и INT1

OUT GICR, TMP

OUT GIFR, TMP ; Сбрасываем флаги прерываний

; Начальные значения

LDI YVAL, 0x71 ; Начальное значение y = 0x71

SEI ; Разрешаем глобальные прерывания

CALL load\_mode ; Читаем сохраненный режим из EEPROM

CALL update\_mode ; Применяем настройки режима

CALL run\_lights ; Запускаем основной цикл

; Основные функции программы

; Задержка ~500 мс (частота 2 Гц)

wait:

LDI R31, 20

LDI R30, 20

LDI R29, 20

wait\_loop:

DEC R29

BRNE wait\_loop

DEC R30

BRNE wait\_loop

DEC R31

BRNE wait\_loop

NOP

NOP

RET

; Основной цикл работы гирлянды

run\_lights:

OUT PORTD, MODE ; Выводим номер режима на PD0-PD1

CPI MODE, 3 ; Если режим 3, проверяем ввод y

BREQ check\_input

blink\_loop:

OUT PORTB, STATE1 ; Первое состояние на PORTB

OUT PORTA, STATE2 ; Второе состояние на PORTA

CALL wait ; Ждем 500 мс

OUT PORTB, STATE2 ; Меняем состояния местами

OUT PORTA, STATE1

CALL wait ; Ждем 500 мс

RJMP run\_lights ; Повторяем цикл

; Проверка нажатия PD7 в режиме 3

check\_input:

IN PIN\_D, PIND ; Читаем состояние порта D

CPI PIN\_D, 131 ; Проверяем, нажата ли PD7

BRNE blink\_loop ; Если не нажата, продолжаем мигание

CALL wait ; Антидребезг: ждем 500 мс

IN PIN\_D, PIND ; Перепроверяем

CPI PIN\_D, 131

BREQ get\_y ; Если PD7 все еще нажата, переходим к вводу y

RJMP blink\_loop

; Ввод числа y через PORTC

get\_y:

CLI ; Запрещаем прерывания на время ввода

LDI YVAL, 0x00

OUT PORTA, YVAL ; Гасим светодиоды

OUT PORTB, YVAL

CALL wait ; Ждем 500 мс

LDI ZERO, 0

IN YVAL, PINC ; Читаем значение с PORTC

CP YVAL, ZERO ; Если ничего не нажато, повторяем

BREQ get\_y

MOV STATE1, YVAL ; Сохраняем y

MOV TMP, YVAL ; Копируем y во временный регистр

LDI R16, 0x80 ; Маска для старшего бита

EOR TMP, R16 ; Инвертируем старший бит для -y (прямой код) в TMP

MOV STATE2, TMP ; Сохраняем -y

wait\_release:

LDI R27, 100 ; Счетчик для таймаута

check\_timeout:

IN YVAL, PINC

CP YVAL, ZERO ; Если кнопки отпущены, выходим

BREQ end\_input

DEC R27

BRNE check\_timeout ; Если таймаут, все равно выходим

end\_input:

IN PIN\_D, PIND ; Читаем PIND в PIN\_D, а не в YVAL

CPI PIN\_D, 131 ; Проверяем, отпущена ли PD7

BRNE get\_y

SEI ; Разрешаем прерывания

RJMP blink\_loop

; Настройки режимов

set\_mode1:

LDI STATE1, 0xFF ; Первое состояние (0xFF)

LDI STATE2, 0x00 ; Второе состояние (0x00)

RET

set\_mode2:

LDI STATE1, 0xAA ; Первое состояние (0xAA)

LDI STATE2, 0x55 ; Второе состояние (0x55)

RET

set\_mode3:

MOV STATE1, YVAL ; Первое состояние = y

MOV TMP, YVAL ; Копируем y во временный регистр

LDI R16, 0x80 ; Маска для старшего бита

EOR TMP, R16 ; Инвертируем старший бит для -y (прямой код) в TMP

MOV STATE2, TMP ; Сохраняем -y в STATE2

RET

; Работа с EEPROM

save\_mode:

SBIC EECR, EEWE ; Проверяем, не идет ли уже запись

RJMP save\_mode

OUT EEARH, ZERO ; Старший байт адреса (0x00)

OUT EEARL, R17 ; Младший байт адреса (0x0F)

OUT EEDR, MODE ; Записываем MODE в регистр данных

SBI EECR, EEMWE ; Разрешаем запись

SBI EECR, EEWE ; Запускаем запись

RET

load\_mode:

SBIC EECR, EEWE ; Ждем окончания предыдущей записи

RJMP load\_mode

OUT EEARH, ZERO ; Адрес EEPROM (0x0F)

OUT EEARL, R17

SBI EECR, EERE ; Запускаем чтение

IN MODE, EEDR ; Загружаем данные в MODE

RET

; Смена режима и сохранение в EEPROM

update\_mode:

CALL save\_mode ; Сохраняем новый режим

CPI MODE, 1 ; Проверяем режим и вызываем

BREQ set\_mode1 ; соответствующую функцию

CPI MODE, 2

BREQ set\_mode2

CPI MODE, 3

BREQ set\_mode3

RET

; Обработчики прерываний

; Обработчик прерывания INT0 (увеличение режима)

INT0\_HANDLER:

IN TMP, SREG ; Сохраняем флаги в R28

CALL load\_mode ; Читаем текущий режим из EEPROM

INC MODE ; Увеличиваем режим на 1

CPI MODE, 4 ; Если режим > 3, сбрасываем до 1

BREQ reset\_to\_1

CALL update\_mode ; Применяем новый режим

OUT SREG, TMP ; Восстанавливаем флаги

RETI ; Выходим из прерывания

; Обработчик прерывания INT1 (уменьшение режима)

INT1\_HANDLER:

IN TMP, SREG ; Сохраняем флаги

CALL load\_mode

DEC MODE ; Уменьшаем режим

CPI MODE, 0 ; Если режим < 1, устанавливаем 3

BREQ reset\_to\_3

CALL update\_mode

OUT SREG, TMP ; Восстанавливаем флаги

RETI

; Обработка переполнения режима (сверху)

reset\_to\_1:

LDI MODE, 1 ; Сбрасываем режим до 1

CALL update\_mode

OUT SREG, R28 ; Восстанавливаем флаги

RETI

; Обработка переполнения режима (снизу)

reset\_to\_3:

LDI MODE, 3 ; Устанавливаем режим 3

CALL update\_mode

OUT SREG, R28

RETI