Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет» Кафедра «Вычислительная техника»

Дисциплина «Автоматизация проектирования микропроцессорных систем»

Лабораторная работа №4

Вариант № 5

Выполнил студент группы ИВТАПбд-41:

Галацков И.А.

Проверил:

Игонин А.Г.

Ульяновск, 2024

**Содержание**

[Постановка задачи 3](#_Toc185422508)

[Краткие теоретические сведения 3](#_Toc185422509)

[Ход выполнения работы 5](#_Toc185422510)

[Тестирование 8](#_Toc185422511)

[Вывод 9](#_Toc185422512)

[Список используемой литературы 10](#_Toc185422513)

[Приложение 11](#_Toc185422514)

# **Постановка задачи**

Разобраться с поддерживаемыми прерываниями. Реализовать 3 обработчика прерывания для микроконтроллера ATmega16. В качестве системы моделирования используется программа Proteus. Типы прерываний:

1. USART, TXC (USART, Tx Complete)
2. SPI, STC (Serial Transfer Complete)
3. INT2 (External interrupt Request)

# **Краткие теоретические сведения**

Proteus – пакет программ для автоматизированного проектирования (САПР) электронных схем. Пакет представляет собой систему моделирования, базирующуюся на основе моделей электронных компонентов.

ATmega16 – это 8-битный высокопроизводительный микроконтроллер из семейства Atmel Mega AVR.

ATmega16 – это 40-контактный микроконтроллер, основанный на усовершенствованной архитектуре RISC (Reduced Instruction Set Computing) с 131 мощной инструкцией.

Прерывание представляет собой событие, при наступлении которого выполнение основной программы приостанавливается и вызывается функция, обрабатывающая прерывание определённого типа.

Всего существует 21 вектор прерываний (рисунок 1).

В микроконтроллере ATmega16 каждому прерыванию соответствует свой вектор прерывания (адрес в начале области памяти программ, в которой хранится команда для перехода к заданной подпрограмме обработки прерывания)

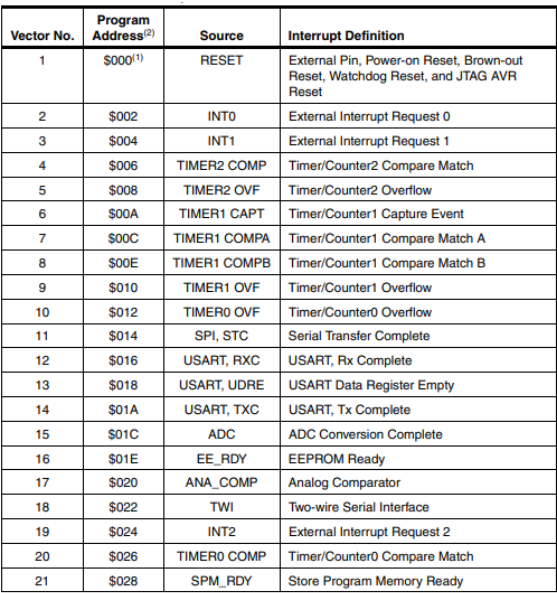


Рис. 1. Таблица прерываний

Microchip Studio- (ранее Atmel Studio и AVR Studio) — основанная на Visual Studio бесплатная проприетарная интегрированная среда разработки (IDE) для разработки приложений для 8- и 32-битных микроконтроллеров семейства AVR и 32-битных микроконтроллеров семейства ARM от компании Atmel, работающая в операционных системах Windows. Atmel Studio содержит компилятор GNU C/C++ и эмулятор, позволяющий отладить выполнение программы.

# **Ход выполнения работы**

Для начала нудно создать новый проект в программе Microchip Studio, для написания программного кода. Также необходимо создать проект в Proteus и подготовить схему для дальнейшей реализации прерываний.

Модель схемы была реализована следующим образом:

Помещаем на схему микроконтроллер ATmega16, и выставляем настройки устанавливая частоту микроконтроллера 8 МГц. Далее подключаем к микроконтроллеру 3 лампочки, которые будет отвечать за индикацию срабатывания прерывания. Также дополнительно подключаем виртуальный терминал для отображения вывода данных, и кнопку для активации прерывания. Каждый элемент подключаем в соответствующие пины на микроконтроллере.

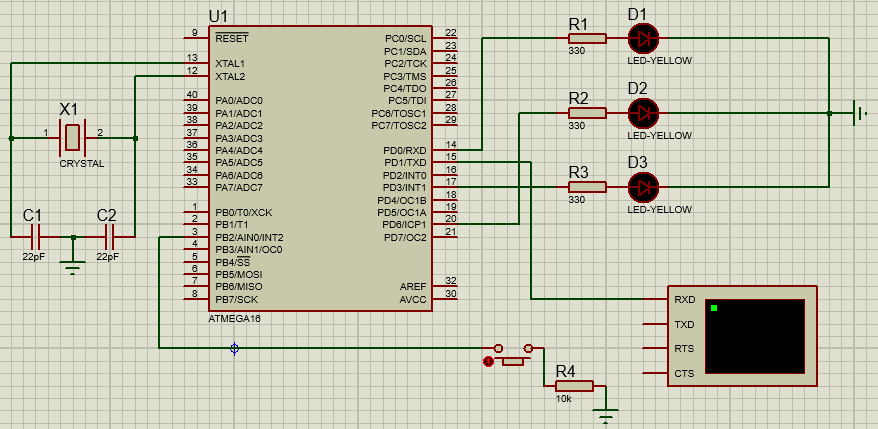


Рис. 2. Итоговая схема

После создания схемы приступаем к написанию кода. Для начала подключаем все необходимы библиотеки для работы и устанавливаем макросы (листинг 1).

Листинг 1. Библиотеки и макросы

|  |
| --- |
| #include <avr/io.h>  #include <avr/interrupt.h>  #define *F\_CPU* 8000000UL  #define BAUD 9600  #define MYUBRR *F\_CPU*/16/BAUD-1 |

Далее настраиваем соответствующие пины, программируем порты PD0, PD3, PD6 как выход, к ним подключены лампочки, а порт PB2 программируем как вход и включаем подтягивающий резистор, так как к этому порту подключена кнопка, активирующая прерывание INT2 (листинг 2).

Листинг 2. Настройка пинов

|  |
| --- |
| // Настройка выводов для светодиодов  DDRB &= ~(1 << PB2); // Настроить PB2 как вход  PORTB |= (1 << PB2); // Включить подтягивающий резистор на PB2  DDRD |= (1 << PD6); // Настроить PD6 как выход (для светодиода)  PORTD &= ~((1 << PD0) | (1 << PD6) | (1 << PD3)); // Все светодиоды выключены |

Далее необходимо инициализировать каждое прерывание. Начнем с прерывания USART. Прерывание USART отвечает за передачу данных. Для наглядности был использован виртуальный терминал, на который выводятся передаваемые данные. Терминал подключен к порту PD1/TXD, отвечающее за чтение данных. Инициализация происходит в функции USART\_Init, в которой устанавливается скорость передачи данных, включается само прерывание, а также возможность прерывания по завершению передачи. В функции USART\_Transmit реализована передача одного байта. В цикле ожидается пока буфер обмена будет пуст, значит готов к записи, отправляем данные в соответствующий регистр, который автоматически отправляет данные через USART (листинг 3).

Листинг 3. Инициализация USART

|  |
| --- |
| void USART\_Init(unsigned int ubrr) {  UBRRH = (unsigned char)(ubrr >> 8);  UBRRL = (unsigned char)ubrr;  UCSRB = (1 << TXEN) | (1 << TXCIE); // Включить передатчик и прерывание TXC  UCSRC = (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0); // Формат 8N1  }  // Передача одного байта по USART  void USART\_Transmit(unsigned char data) {  while (!(UCSRA & (1 << UDRE))); // Ожидание готовности буфера  UDR = data; // Отправить данные  } |

Инициализируем прерывание SPI. Данное прерывание является одним из способов передачи данных. Для инициализации изменяем значение регистров, указываем что SPI включен, устанавливаем режим мастер, в котором и происходит передача данных, а также разрешаем прерывания. (листинг 3). В качестве индикации срабатывания прерывания используем лампочку, которая моргает каждый раз, когда происходит передача данных.

Листинг 3. Инициализация SPI

|  |
| --- |
| / SPI Initialization (Master)  void SPI\_MasterInit(void) {  DDRB = (1 << PB5) | (1 << PB7); // MOSI, SCK как выходы  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPIE) | (1 << SPR0); // SPI включён, Master, прерывания  } |

Инициализируем прерывание INT2, данное внешнее прерывание настраивается на срабатывание по определенному событию, в данном случае таким событием будет выступать нажатие кнопки, индикацией срабатывания прерывания будет служить лампочка, которая будет загораться и гаснуть по нажатию на кнопку. Настраиваем прерывание на срабатывание по спаду сигнала, а также разрешаем само прерывание (листинг 4).

Листинг 4. Инициализация INT2

|  |
| --- |
| // INT2 Initialization  void INT2\_Init(void) {  MCUCSR &= ~(1 << ISC2); // Срабатывание по спаду сигнала  GICR |= (1 << INT2); // Разрешить прерывание INT2  } |

После срабатывание любого из прерываний вызывается соответствующий обработчик прерывания, в котором происходит переключение лампочки.

Далее функции для срабатывания прерываний вызываются в основном цикле, все кроме прерывания INT2, данное прерывание срабатывает по нажатию на кнопку.

# **Тестирование**

Прерывание USART и SPI срабатывают сразу после запуска программы. В случае USART, в виртуальный терминал выводится символ ‘A’ и моргает лампочка, каждый раз, когда срабатывает прерывание. В случае с SPI индикатором является лампочка, которая моргает, когда происходит передача данных (рисунок 3).

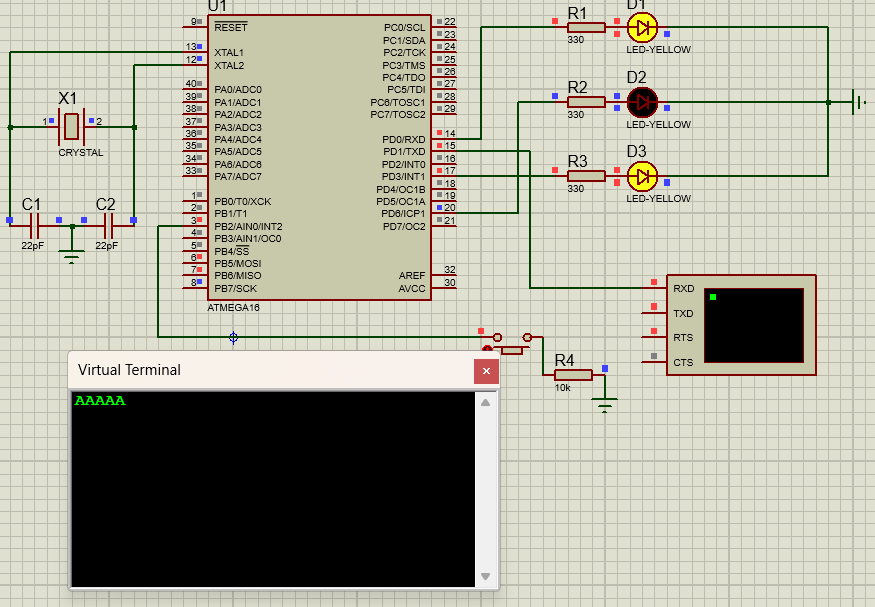


Рис. 3. Работа прерываний USART и SPI

В случае с прерыванием INT2, срабатывание происходит по нажатию на кнопку, соответственно загорается лампочка (рисунок 4).

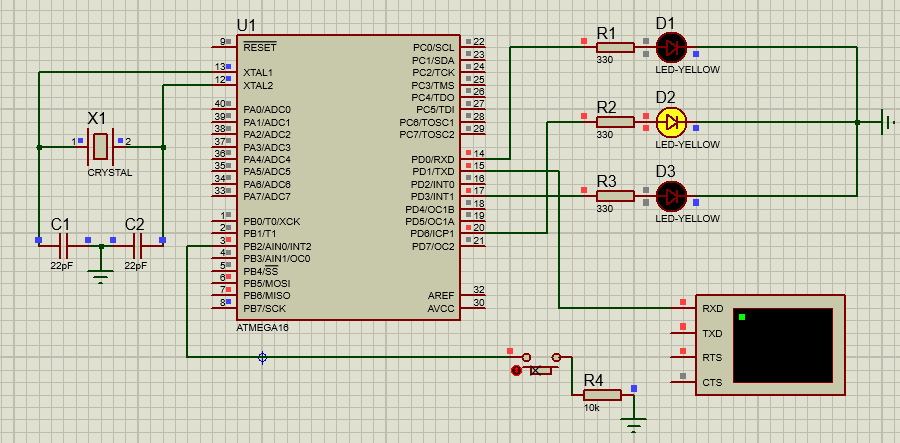


Рис. 4. Работа прерывания INT2

# **Вывод**

В ходе выполнения данной лабораторной работы были реализованы три прерывания для микроконтроллера ATmega16, типа AVR. Программа работает исправно в штатном режиме.

# **Список используемой литературы**

1. Руководство пользователя по PROTEUS. Сайт о микроконтроллерах AVR.ru [Электронный ресурс] - https://avr.ru/tools/proteus/guide
2. Программирование микроконтроллеров AVR. [Электронный ресурс] - <https://www.youtube.com/watch?v=Cz4UGSPbj98>
3. Программирование на языке Си для AVR и PIC микроконтроллеров. Шпак Ю.А. 2-е издание. 2011-г. [Электронный ресурс] - <https://radioparty.ru/literatura/275-programmirovanie-na-c-dlya-avr-i-pic-2-izdanie?ysclid=ld2veu85s7512645242>

# **Приложение**

#include <avr/io.h>

#include <avr/interrupt.h>

#define *F\_CPU* 8000000UL

#define BAUD 9600

#define MYUBRR *F\_CPU*/16/BAUD-1

volatile *uint16\_t* timer\_count = 0; // Счётчик для формирования 1 секунды

volatile *uint8\_t* usart\_flag = 0; // Флаг для отправки данных

volatile *uint8\_t* spi\_flag = 0; // Флаг для передачи SPI

// USART Initialization

void USART\_Init(unsigned int ubrr) {

UBRRH = (unsigned char)(ubrr >> 8);

UBRRL = (unsigned char)ubrr;

UCSRB = (1 << TXEN) | (1 << TXCIE); // Включить передатчик и прерывание TXC

UCSRC = (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0); // Формат 8N1

}

// Передача одного байта по USART

void USART\_Transmit(unsigned char data) {

while (!(UCSRA & (1 << UDRE))); // Ожидание готовности буфера

UDR = data; // Отправить данные

}

// SPI Initialization (Master)

void SPI\_MasterInit(void) {

DDRB = (1 << PB5) | (1 << PB7); // MOSI, SCK как выходы

SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPIE) | (1 << SPR0); // SPI включён, Master, прерывания

}

// Передача данных по SPI

void SPI\_MasterTransmit(unsigned char data) {

SPDR = data; // Записать данные в SPI Data Register

// Ожидание завершения передачи можно сделать с помощью прерывания SPI STC

}

// INT2 Initialization

void INT2\_Init(void) {

MCUCSR &= ~(1 << ISC2); // Срабатывание по спаду сигнала

GICR |= (1 << INT2); // Разрешить прерывание INT2

}

// Timer0 Initialization

void Timer0\_Init(void) {

TCCR0 = (1 << WGM01) | (1 << CS02) | (1 << CS00); // CTC, делитель 1024

OCR0 = 78; // Прерывание каждые 10 мс (F\_CPU/1024/100)

TIMSK = (1 << OCIE0); // Разрешить прерывания сравнения

}

// USART TXC Interrupt Handler

ISR(USART\_TXC\_vect) {

PORTD ^= (1 << PD0); // Переключить светодиод PD0

}

// SPI STC Interrupt Handler

ISR(SPI\_STC\_vect) {

PORTD ^= (1 << PD3); // Переключить светодиод PD3 (индикация завершения передачи)

}

// INT2 Interrupt Handler

ISR(INT2\_vect) {

PORTD ^= (1 << PD6); // Переключить светодиод PD2

}

// Timer0 Interrupt Handler

ISR(TIMER0\_COMP\_vect) {

timer\_count++;

if (timer\_count >= 100) { // 100 \* 10 мс = 1 секунда

timer\_count = 0;

usart\_flag = 1; // Установить флаг для отправки данных по USART

spi\_flag = 1; // Установить флаг для передачи SPI

}

}

int main(void) {

// Настройка выводов для светодиодов

DDRB &= ~(1 << PB2); // Настроить PB2 как вход

PORTB |= (1 << PB2); // Включить подтягивающий резистор на PB2

DDRD |= (1 << PD6); // Настроить PD6 как выход (для светодиода)

PORTD &= ~((1 << PD0) | (1 << PD6) | (1 << PD3)); // Все светодиоды выключены

// Инициализация всех модулей

USART\_Init(MYUBRR); // USART

SPI\_MasterInit(); // SPI

INT2\_Init(); // INT2

Timer0\_Init(); // Timer0

sei(); // Включение глобальных прерываний

while (1) {

if (usart\_flag) {

usart\_flag = 0; // Сбрасываем флаг

USART\_Transmit('A'); // Отправить символ 'A'

}

if (spi\_flag) {

spi\_flag = 0; // Сбрасываем флаг

SPI\_MasterTransmit(0xAA); // Передать байт 0xAA по SPI

}

}

}