|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  **МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**  (Национальный исследовательский университет) | | | | | |
|  | | | | |
|  | Институт №3 «Системы управления, информатика и электроэнергетика»  Кафедра 307 «Цифровые технологии и информационные системы» | | |  |
|  |  | | |  |
|  | ОТЧЁТ о выполнении лабораторной работы №1 «Измерение и отображение частоты и аплитуды сигнала на базе микроконтроллера ATmega328P»  по дисциплине «Программирование микроконтроллеров. Си» | | |  |
|  |  | | |  |
|  | Выполнил студент 3-го курса  группы М3О-314Б-22 | Подпись: | Бедретдинов Андрей Маратович |  |
|  | Проверил ст. преподаватель  каф. 307 | Подпись: | Коробков Максим Андреевич |  |
| Москва 2024 | | | | |

# РЕФЕРАТ

Отчёт 44 с., 15 рис..

ЧАСТОТА, АПЛИТУДА, МИКРОКОНТРОЛЛЕР AVR, АНАЛОГО-ЦИФРОВОЕ ПРЕОБРАЗОВАНИЕ, СЕМИСЕГМЕНТНЫЙ ИНДИКАТОР, ПРЕРЫВАНИЕ, СДВИГОВЫЙ РЕГИСТР

Цель работы — освоение методов измерения и отображения электрических сигналов (частоты и амплитуды) с использованием микроконтроллера ATmega328P, а также практическое изучение работы с аналогово-цифровым преобразователем (АЦП), сдвиговым регистром 74HC595 для управления анодным семисегментным индикатором, и использование прерываний для работы с кнопкой и таймерами для точного измерения параметров сигнала.

В процессе работы выполнены следующие задачи:

1. Настроить работу микроконтроллера ATmega328P с АЦП для измерения амплитуды входного сигнала.
2. Реализовать алгоритм измерения частоты сигнала с использованием пересечения "нуля" и подсчета периодов сигнала.
3. Настроить работу с сдвиговым регистром 74HC595 для управления анодным семисегментным индикатором.
4. Разработать алгоритм отображения частоты и амплитуды на семисегментном индикаторе с учетом дробной точности.
5. Реализовать работу с прерываниями для измерения частоты, получения данных с АЦП и работы с кнопкой на входе PD2/INT0.
6. Настроить работу кнопки для переключения режимов отображения частоты и амплитуды.
7. Реализовать вывод значений частоты, амплитуды и текущего напряжения через UART на виртуальный терминал.

Отчёт о проведении научно-исследовательской работы оформлен в соответствии с ГОСТ 7.32–2017.

**СОДЕРЖАНИЕ**

[РЕФЕРАТ 2](#_Toc181233687)

[ПОСТАНОВКА ЗАДАЧИ 7](#_Toc181233688)

[КРАТКОЕ ОПИСАНИЕ ТЕОРИИ 8](#_Toc181233689)

[1 Микроконтроллер ATmega328P 8](#_Toc181233690)

[1.1 Аналогово-цифровой преобразователь (АЦП) 8](#_Toc181233691)

[1.2 Таймеры и счетчики 9](#_Toc181233692)

[1.3 Внешние прерывания 9](#_Toc181233693)

[1.4 Интерфейс UART 9](#_Toc181233694)

[2 Сдвиговый регистр 74HC595 11](#_Toc181233695)

[2.1 Принцип работы сдвигового регистра 74HC595 11](#_Toc181233696)

[2.2 Основные выводы 74HC595 11](#_Toc181233697)

[2.3 Важные характеристики сдвигового регистра 74HC595 12](#_Toc181233698)

[ПРОЦЕСС СОЗДАНИЯ 13](#_Toc181233699)

[1 Выбор компонентов и подготовка схемы 13](#_Toc181233700)

[1.1 ATmega328P 13](#_Toc181233701)

[1.2 Сдвиговый регистр 74HC595 14](#_Toc181233702)

[1.3 Семисегментный индикатор 15](#_Toc181233703)

[1.4 Внешний источник сигнала 15](#_Toc181233704)

[1.5 Виртуальный терминал 15](#_Toc181233705)

[1.6 Общая схема подключения 17](#_Toc181233706)

[2 Инициализация микроконтроллера ATmega328P 20](#_Toc181233707)

[2.1 Настройка UART 20](#_Toc181233708)

[2.2 Настройка АЦП 20](#_Toc181233709)

[2.3 Настройка таймера 1 21](#_Toc181233710)

[2.4 Управление сдвиговым регистром 74HC595 21](#_Toc181233711)

[2.5 Инициализация кнопки и работы с режимами 22](#_Toc181233712)

[3 Реализация считывания и обработки сигналов 23](#_Toc181233713)

[3.1 Инициализация АЦП 23](#_Toc181233714)

[3.2 Обработчик прерываний АЦП 23](#_Toc181233715)

[3.3 Расчет амплитуды и частоты сигнала 24](#_Toc181233716)

[4 Работа с сдвиговым регистром 74HC595 26](#_Toc181233717)

[4.1 Инициализация сдвигового регистра 26](#_Toc181233718)

[4.2 Передача данных в сдвиговый регистр 26](#_Toc181233719)

[4.3 Управление разрядами семисегментного индикатора 27](#_Toc181233720)

[4.4 Отображение данных на семисегментном индикаторе 27](#_Toc181233721)

[5 Работа со сдвиговым регистром и семисегментным индикатором 29](#_Toc181233722)

[5.1 Инициализация сдвигового регистра 29](#_Toc181233723)

[5.2 Отправка данных в сдвиговый регистр 29](#_Toc181233724)

[5.3 Управление разрядами индикатора 30](#_Toc181233725)

[5.4 Отображение данных на индикаторе 31](#_Toc181233726)

[6 Работа с виртуальным терминалом 32](#_Toc181233727)

[6.1 Инициализация UART 32](#_Toc181233728)

[6.2 Отправка данных на виртуальный терминал 32](#_Toc181233729)

[6.3 Логирование данных 33](#_Toc181233730)

[ТЕСТИРОВАНИЕ 34](#_Toc181233731)

[1 Вывод значений на терминал 34](#_Toc181233732)

[2 Вывод частоты на семисегментный индикатор 36](#_Toc181233733)

[3 Вывод амплитуды на семисегментный индикатор 39](#_Toc181233734)

[4 Изменение частоты и амплитуды внешнего источника 42](#_Toc181233735)

[ЗАКЛЮЧЕНИЕ 44](#_Toc181233736)

# ПОСТАНОВКА ЗАДАЧИ

Целью лабораторной работы является настройка микроконтроллера **ATmega328P** для измерения и отображения параметров синусоидального сигнала с использованием внешних компонентов. Для этого требуется настроить встроенный аналогово-цифровой преобразователь (АЦП) для получения значений входного сигнала, определить алгоритм подсчета амплитуды и частоты сигнала, основываясь на пересечениях через "ноль", и выводить результаты на семисегментный индикатор через сдвиговый регистр **74HC595**.

Важной задачей является разработка алгоритма управления анодным семисегментным индикатором, чтобы корректно отображать значения с дробной частью. Значения должны выводиться с точностью до 0.01 без округления, а в случае превышения четырех разрядов — необходимо отбрасывать лишние цифры. Также требуется реализовать управление разрядами индикатора, используя сдвиговый регистр для экономии выводов микроконтроллера.

Отдельно необходимо настроить внешнее прерывание на PD2/INT0 для работы с кнопкой, которая позволяет переключаться между режимами отображения частоты и амплитуды. Управление режимами осуществляется через выводы PB6 и PB7, которые должны включать соответствующие индикаторы. Также необходимо разработать и настроить передачу данных измерений через интерфейс UART на виртуальный терминал для отладки и контроля за процессом измерения в режиме реального времени.

# КРАТКОЕ ОПИСАНИЕ ТЕОРИИ

### Микроконтроллер ATmega328P

ATmega328P — это 8-битный микроконтроллер семейства AVR, разработанный компанией Atmel (сейчас часть Microchip Technology). Он широко применяется в различных системах ввода-вывода и управления благодаря высокой производительности и низкому энергопотреблению. Микроконтроллер ATmega328P является сердцем таких популярных платформ, как Arduino, что делает его универсальным и доступным для разработки.

ATmega328P работает на частоте до 20 МГц при питании 5 В, но в рамках данной лабораторной работы микроконтроллер настроен на тактовую частоту 8 МГц. Это оптимальная частота для энергосберегающих приложений и позволяет добиться высокой точности измерений сигнала при одновременном снижении энергопотребления.

#### Аналогово-цифровой преобразователь (АЦП)

Один из ключевых модулей ATmega328P — это 10-битный АЦП. В данной работе АЦП используется для преобразования аналогового сигнала, поступающего с внешнего источника на вход ADC0, в цифровое значение. АЦП принимает сигналы с амплитудой от 0 до 5 В, и с разрешением в 10 бит может разделить это напряжение на 1024 уровня (от 0 до 1023). Это позволяет микроконтроллеру точно интерпретировать аналоговый сигнал для последующей обработки.

АЦП ATmega328P может работать в нескольких режимах, включая непрерывное преобразование и использование внешнего опорного напряжения. В данной работе используется режим с опорным напряжением AVcc, что означает, что входной сигнал на АЦП преобразуется относительно напряжения питания микроконтроллера (5 В).

АЦП запускается по команде и генерирует прерывание при завершении преобразования, что позволяет организовать непрерывный процесс измерения входного сигнала без блокировки основной программы.

#### Таймеры и счетчики

ATmega328P оснащен несколькими 8- и 16-битными таймерами/счетчиками. В данной лабораторной работе используется 16-битный таймер Timer1, который настраивается для создания прерываний с точной периодичностью (в данном случае 1 секунда). Это позволяет микроконтроллеру рассчитывать частоту сигнала, измеряя количество пересечений сигнала через "ноль" за 1 секунду. Таймер используется в режиме CTC (Clear Timer on Compare Match), что обеспечивает стабильное время срабатывания.

#### Внешние прерывания

Прерывания позволяют реагировать на внешние события (например, нажатие кнопки) без постоянной проверки состояния пинов. В данной работе используется внешний источник прерывания INT0, подключенный к PD2. Это прерывание генерируется при изменении состояния кнопки, что позволяет переключаться между режимами отображения частоты и амплитуды. Прерывание INT0 настроено на срабатывание по спаду сигнала (переход от высокого уровня к низкому).

#### Интерфейс UART

UART (Universal Asynchronous Receiver-Transmitter) — это интерфейс для передачи и приема данных. В ATmega328P UART реализован в виде модуля, который поддерживает асинхронную передачу данных по последовательному интерфейсу. В лабораторной работе UART используется для передачи данных о частоте, амплитуде и текущем напряжении на внешний терминал (например, через подключение к ПК). Настроенная скорость передачи данных составляет 9600 бод, что является стандартной скоростью для многих приложений.

UART работает с помощью двух регистров данных: UDR0 для записи передаваемых данных и UDRE0 для проверки готовности передатчика к следующей передаче. UART в ATmega328P также поддерживает работу с прерываниями, что делает его подходящим для работы в реальном времени.

### Сдвиговый регистр 74HC595

Сдвиговый регистр 74HC595 — это интегральная микросхема, которая используется для расширения числа выводов микроконтроллера. Это особенно полезно, когда необходимо управлять большим количеством выходов (например, для управления несколькими сегментами семисегментного индикатора), но количество выводов микроконтроллера ограничено. Сдвиговый регистр позволяет управлять восемью выходами, используя всего три управляющих вывода.

#### Принцип работы сдвигового регистра 74HC595

Сдвиговый регистр преобразует данные, передаваемые последовательно, в параллельные выходы. Это позволяет передавать несколько бит данных через один управляющий сигнал и затем использовать их для управления множеством устройств.

#### Основные выводы 74HC595

* DS (Data Pin) — входной пин данных. Через этот пин данные передаются в сдвиговый регистр последовательно.
* SH\_CP (Shift Register Clock Pin) — тактовый сигнал сдвигового регистра. На каждый импульс этого сигнала данные сдвигаются на один бит в регистре.
* ST\_CP (Storage Register Clock Pin) — тактовый сигнал регистра хранения. После передачи всех данных этот сигнал обновляет выходы сдвигового регистра, передавая сохраненные данные на выходные пины.
* MR (Master Reset Pin) — вход сброса. Устанавливая этот пин в логический 0, можно сбросить регистр, очистив все выходы.
* OE (Output Enable Pin) — вход разрешения выходов. При подаче логической единицы на этот пин выходы сдвигового регистра отключаются.

#### Важные характеристики сдвигового регистра 74HC595

* Разрядность: 74HC595 является 8-разрядным регистром, что означает, что с его помощью можно управлять восемью устройствами (например, сегментами индикатора).
* Последовательная передача данных: данные передаются последовательно, бит за битом через входной пин данных (DS). Каждый бит передвигается по регистру с каждым тактовым импульсом, поданным на пин SH\_CP.
* Параллельные выходы: после того как все восемь бит данных переданы, они одновременно записываются на выходы при подаче сигнала на ST\_CP. Это позволяет за один момент обновить состояние всех выходов, не влияя на текущие данные во время их передачи.

# ПРОЦЕСС СОЗДАНИЯ

### Выбор компонентов и подготовка схемы

Для выполнения лабораторной работы необходимо выбрать и подготовить компоненты, которые позволят реализовать измерение, обработку данных и их отображение на семисегментном индикаторе. Основные компоненты включают микроконтроллер ATmega328P, сдвиговый регистр 74HC595, анодный семисегментный индикатор и внешний источник сигнала. Каждый из этих компонентов играет важную роль в проекте.

#### ATmega328P

ATmega328P используется как основное управляющее устройство. Этот микроконтроллер отличается высокой функциональностью, интегрированным АЦП (аналого-цифровым преобразователем) и поддержкой различных периферийных интерфейсов. Его частота может составлять до 16 МГц, однако в данной работе используется частота 8 МГц с использованием встроенного генератора.

Микроконтроллер ATmega328P обеспечивает измерение амплитуды и частоты аналогового сигнала, поступающего на его АЦП. Встроенный 10-битный АЦП позволяет точно преобразовывать аналоговые сигналы в цифровые для дальнейшей обработки. Также микроконтроллер использует внешние прерывания для работы с кнопкой, которая позволяет переключаться между режимами отображения частоты и амплитуды. Для передачи данных на виртуальный терминал микроконтроллер использует встроенный UART.

Настройки микроконтроллера Atmega328P в Proteus изображены на рисунке 1.

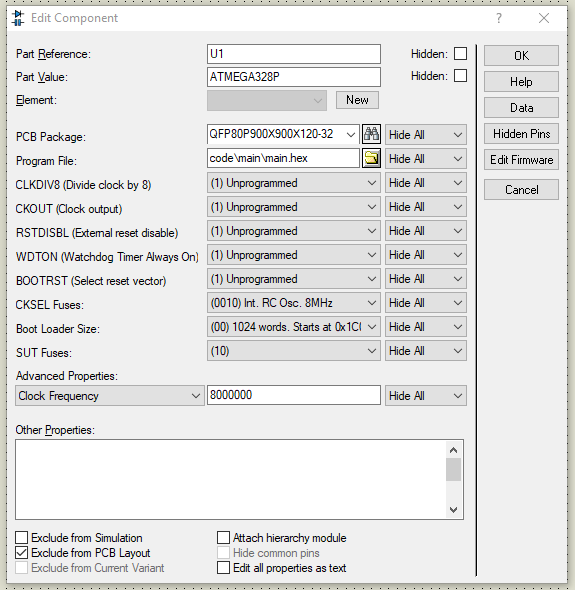


Рисунок 1 — Настройки микроконтроллера Atmega328P

#### Сдвиговый регистр 74HC595

Для управления семисегментным индикатором используется сдвиговый регистр 74HC595. Этот компонент выбран из-за его способности управлять несколькими выводами индикатора при минимальном количестве пинов микроконтроллера. 74HC595 позволяет подключить семисегментный индикатор через три линии управления: линия данных, тактовый сигнал и сигнал фиксации данных.

Сдвиговый регистр принимает последовательные биты от микроконтроллера и преобразует их в параллельные данные, что позволяет управлять сегментами семисегментного индикатора.

Основные сигналы для работы:

* DS (Data) — передача данных,
* SH\_CP (Shift Clock) — тактовый сигнал для сдвига данных,
* ST\_CP (Storage Clock) — фиксация данных на выходе,
* MR (Master Reset) — сброс регистра,
* OE (Output Enable) — включение/выключение выходов.

#### Семисегментный индикатор

Для вывода данных используется анодный семисегментный индикатор, способный отображать цифры от 0 до 9. Индикатор подключен к сдвиговому регистру, который управляет его сегментами. Важно настроить правильную передачу данных через регистр, чтобы отображать как целые, так и дробные числа.

#### Внешний источник сигнала

Для подачи синусоидального сигнала на АЦП микроконтроллера используется внешний источник сигнала. Этот сигнал позволяет микроконтроллеру измерять его амплитуду и частоту. АЦП преобразует аналоговый сигнал в цифровую форму для дальнейшей обработки.

#### Виртуальный терминал

Виртуальный терминал используется для отображения и логирования данных, таких как измеренные частоты, амплитуды и текущее напряжение. Виртуальный терминал эмулирует работу реального последовательного порта и взаимодействует с микроконтроллером через интерфейс UART (универсальный асинхронный приемо-передатчик).

ATmega328P имеет встроенный модуль UART, который позволяет передавать и принимать данные через последовательный интерфейс. Для работы UART необходимо настроить следующие параметры:

* Скорость передачи данных (обычно 9600 бод),
* Формат данных (8 бит данных, 1 стоп-бит),
* Режим работы (только передача или передача и прием).

В лабораторной работе UART настроен для передачи данных на виртуальный терминал, который подключен к пинам TXD (PD1) и RXD (PD0) микроконтроллера. В режиме передачи микроконтроллер отправляет измеренные данные на терминал в виде строк, используя функции передачи байтов и строк данных.

Настройки виртуального терминала представлены на рисунке 2.

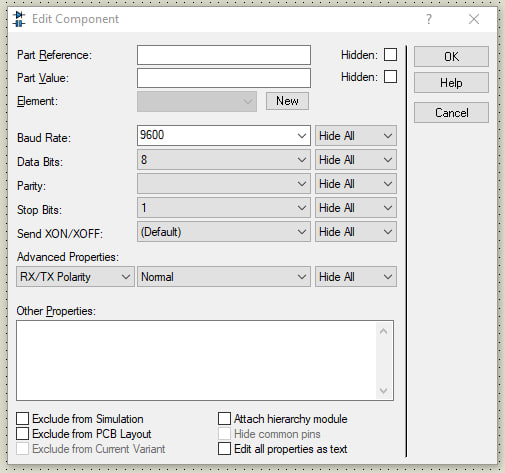


Рисунок 2 — Настройки виртуального терминала

#### Общая схема подключения

Для реализации схемы лабораторной работы необходимо выполнить последовательное подключение всех компонентов системы, включающих микроконтроллер ATmega328P, сдвиговый регистр 74HC595, семисегментный индикатор, кнопки и виртуальный терминал.

Общая схема подключения представлена на рисунке 3.

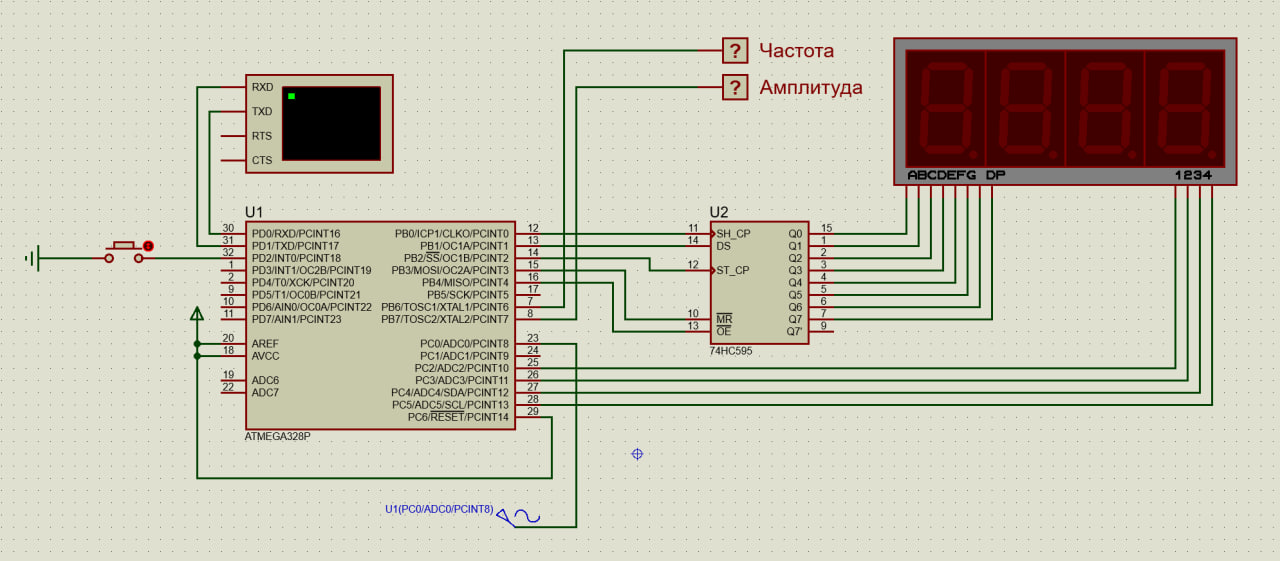


Рисунок 3 — Общая схема подключения

Микроконтроллер ATmega328P работает при частоте 8 МГц и подключен к различным периферийным устройствам. Кнопка подключается к выводу PD2 (INT0), и настроена на внешнее прерывание. Для этого вывод PD2 конфигурируется как вход, а для исключения ложных срабатываний включается встроенный подтягивающий резистор.

Для управления отображением на семисегментном индикаторе используются четыре вывода микроконтроллера, подключенные к разрядам дисплея через транзисторные ключи. Разряды индикатора подключаются следующим образом: первый разряд подключен к выводу PC5, второй разряд к PC4, третий к PC3 и четвертый к PC2. Эти выводы конфигурируются как выходы и позволяют выбирать активный разряд индикатора.

Для управления сегментами семисегментного индикатора применяется сдвиговый регистр 74HC595. Три управляющих вывода микроконтроллера подключены к сдвиговому регистру: PB0 (SH\_CP) для подачи тактового сигнала, PB2 (ST\_CP) для подачи сигнала записи данных в выходные регистры, и PB1 (DS) для передачи данных в сдвиговый регистр. Также используются выводы PB3 и PB4 для управления сигналами MR (Master Reset) и OE (Output Enable) сдвигового регистра. Вывод PB3 управляет сбросом регистра, а вывод PB4 включает и отключает выходы сдвигового регистра.

Вывод данных в виртуальный терминал осуществляется с помощью интерфейса UART. Выводы TXD (PD1) и RXD (PD0) микроконтроллера подключаются к виртуальному терминалу через последовательный интерфейс. PD1 работает как вывод для передачи данных, и его необходимо настроить для передачи информации о частоте и амплитуде сигнала.

Внешний источник сигнала подключается к выводу ADC0 (PC0), который используется для измерения аналогового сигнала с помощью встроенного АЦП. Сигнал, поступающий на PC0, измеряется, и на основе этих данных вычисляются частота и амплитуда сигнала.

### Инициализация микроконтроллера ATmega328P

Инициализация микроконтроллера ATmega328P является важным этапом в подготовке к работе с различными периферийными устройствами, такими как АЦП, таймеры, интерфейс UART, сдвиговый регистр 74HC595 и семисегментный индикатор. В ходе этой лабораторной работы требуется произвести настройку каждого из модулей микроконтроллера для обеспечения точной обработки входных сигналов и вывода данных на индикатор.

Первоначально необходимо настроить тактовую частоту микроконтроллера на 8 МГц. Для этого определяется макрос F\_CPU, который используется библиотеками, например, для точного расчета задержек. Также требуется инициализировать порты ввода-вывода, к которым подключены внешние компоненты.

#### Настройка UART

Для передачи данных через виртуальный терминал используется интерфейс UART (универсальный асинхронный приемопередатчик). Процесс инициализации включает задание скорости передачи данных (9600 бод), настройку длины фрейма (8 бит данных, 1 стоповый бит) и активацию передатчика:

void uart\_init**()** **{**

uint16\_t ubrr\_value **=** 51**;** // Для частоты 8 МГц и скорости 9600 бод

UBRR0H **=** **(**ubrr\_value **>>** 8**);** // Установка старшего байта UBRR

UBRR0L **=** ubrr\_value**;** // Установка младшего байта UBRR

UCSR0B **=** **(**1 **<<** TXEN0**);** // Включение передатчика UART

UCSR0C **=** **(**1 **<<** UCSZ01**)** **|** **(**1 **<<** UCSZ00**);** // Формат 8 бит данных, 1 стоповый бит

**}**

UART обеспечивает передачу данных о текущих измерениях (частота, амплитуда и напряжение) на виртуальный терминал.

#### Настройка АЦП

АЦП микроконтроллера используется для измерения аналоговых сигналов, поступающих на вход ADC0 (PC0). Для настройки АЦП активируются прерывания, указывается источник опорного напряжения (AVcc) и выбирается предделитель 64 для точного измерения при частоте 8 МГц:

void adc\_init**()** **{**

// Выбор опорного напряжения AVcc и входа ADC0

ADMUX **=** **(**1 **<<** REFS0**);**

// Включение АЦП, прерываний и предделителя 64

ADCSRA **=** **(**1 **<<** ADEN**)** **|** **(**1 **<<** ADIE**)** **|** **(**1 **<<** ADPS2**)** **|** **(**1 **<<** ADPS1**);**

**}**

Прерывания АЦП позволяют автоматически производить новое измерение, как только завершено текущее, а также контролировать минимальные и максимальные значения для расчета амплитуды сигнала.

#### Настройка таймера 1

Таймер 1 используется для создания периодических прерываний с интервалом в 1 секунду. Это необходимо для расчета частоты сигнала и обновления данных на индикаторе. Таймер настраивается на работу в режиме CTC (сравнение с OCR1A) с предделителем 1024:

void timer1\_init**()** **{**

// CTC режим

TCCR1B **|=** **(**1 **<<** WGM12**);**

// Задание сравниваемого значения для получения прерывания раз в секунду

OCR1A **=** 15624**;**

// Установка предделителя 1024

TCCR1B **|=** **(**1 **<<** CS12**)** **|** **(**1 **<<** CS10**);**

// Включение прерываний по совпадению с OCR1A

TIMSK1 **|=** **(**1 **<<** OCIE1A**);**

**}**

Прерывание по таймеру позволяет производить расчет частоты сигнала на основе числа пересечений синусоиды через ноль.

#### Управление сдвиговым регистром 74HC595

Для управления семисегментным индикатором используется сдвиговый регистр 74HC595, что позволяет существенно экономить выводы микроконтроллера. Для его инициализации необходимо настроить пины микроконтроллера, которые будут управлять передачей данных и тактовым сигналом:

void init\_shift\_register\_control**()** **{**

// Настройка пинов MR и OE как выходы

DDRB **|=** **(**1 **<<** MR\_PIN**)** **|** **(**1 **<<** OE\_PIN**);**

// Установка MR в логическую 1 (работа без сброса)

PORTB **|=** **(**1 **<<** MR\_PIN**);**

// Включение выходов сдвигового регистра

PORTB **&=** **~(**1 **<<** OE\_PIN**);**

**}**

Передача данных в сдвиговый регистр происходит через функции, использующие пины PB0, PB1 и PB2 для передачи данных и синхронизации сигналов.

#### Инициализация кнопки и работы с режимами

Кнопка, подключенная к выводу PD2, используется для переключения между режимами отображения частоты и амплитуды. Для работы с кнопкой настроено внешнее прерывание на выводе INT0, что позволяет быстро реагировать на нажатие кнопки и менять режимы отображения:

void button\_init**()** **{**

// PD2 как вход

DDRD **&=** **~(**1 **<<** BUTTON\_PIN**);**

// Включение подтягивающего резистора на PD2

PORTD **|=** **(**1 **<<** BUTTON\_PIN**);**

// Прерывание по спаду сигнала на INT0

EICRA **|=** **(**1 **<<** ISC01**);**

// Включение прерывания INT0

EIMSK **|=** **(**1 **<<** INT0**);**

**}**

Таким образом, нажатие кнопки приводит к смене режима отображения между частотой и амплитудой на индикаторе.

### Реализация считывания и обработки сигналов

Для реализации процесса считывания и обработки аналогового сигнала с внешнего источника используется встроенный в микроконтроллер ATmega328P АЦП (аналогово-цифровой преобразователь). Сначала необходимо настроить АЦП и запустить его, после чего значения сигнала с внешнего источника можно будет преобразовывать в цифровую форму и использовать для дальнейшей обработки.

#### Инициализация АЦП

Для работы с АЦП важно выбрать нужные настройки: выбрать источник опорного напряжения, канал для преобразования и установить предделитель для правильной скорости преобразования.

// Инициализация АЦП

void adc\_init**()** **{**

// Установка опорного напряжения как AVcc (5В) и выбор входного канала ADC0

ADMUX **=** **(**1 **<<** REFS0**);**

// Включение АЦП, разрешение прерываний, установка предделителя 64

ADCSRA **=** **(**1 **<<** ADEN**)** **|** **(**1 **<<** ADIE**)** **|** **(**1 **<<** ADPS2**)** **|** **(**1 **<<** ADPS1**);**

**}**

После инициализации необходимо запустить первое преобразование АЦП, чтобы начать считывание данных с внешнего источника. Для этого используется функция adc\_start\_conversion().

// Запуск следующего преобразования АЦП

void adc\_start\_conversion**()** **{**

// Установка бита ADSC для запуска преобразования

ADCSRA **|=** **(**1 **<<** ADSC**);**

**}**

#### Обработчик прерываний АЦП

Когда преобразование АЦП завершается, срабатывает прерывание, в котором считывается результат измерений. Также в этом прерывании происходит обновление максимальных и минимальных значений для расчета амплитуды сигнала, а также отслеживание пересечения через "ноль" для вычисления частоты.

// Обработчик прерывания АЦП

ISR**(**ADC\_vect**)** **{**

// Переменная для хранения предыдущего значения АЦП, используется для отслеживания пересечения через "ноль"

static uint16\_t last\_adc\_value **=** 0**;**

// Чтение текущего значения с АЦП (в регистре ADC хранится результат последнего преобразования)

adc\_value **=** ADC**;**

// Проверка, является ли текущее значение максимальным за цикл измерений

**if** **(**adc\_value **>** max\_value**)** **{**

max\_value **=** adc\_value**;** // Сохраняем новое максимальное значение

**}**

// Проверка, является ли текущее значение минимальным за цикл измерений

**if** **(**adc\_value **<** min\_value**)** **{**

min\_value **=** adc\_value**;** // Сохраняем новое минимальное значение

**}**

// Если текущее значение АЦП находится выше порога (середина диапазона) и предыдущее значение было ниже,

// то это означает пересечение через "ноль" (для синусоидального сигнала это ключевая точка для измерения частоты)

**if** **((**adc\_value **>=** ZERO\_CROSS\_THRESHOLD**)** **&&** **(**last\_adc\_value **<** ZERO\_CROSS\_THRESHOLD**))** **{**

zero\_cross\_count**++;** // Увеличиваем счетчик пересечений через "ноль"

**}**

// Сохраняем текущее значение как предыдущее для следующего сравнения

last\_adc\_value **=** adc\_value**;**

// Инициируем следующий цикл преобразования АЦП, чтобы продолжать измерения

ADCSRA **|=** **(**1 **<<** ADSC**);** // Устанавливаем бит ADSC (Start Conversion) для запуска нового преобразования

**}**

#### Расчет амплитуды и частоты сигнала

После завершения измерений за один период (1 секунда) данные о частоте и амплитуде вычисляются в обработчике прерывания таймера. Амплитуда определяется как разница между максимальным и минимальным значениями АЦП за цикл, а частота — как количество пересечений через "ноль", деленное на два (так как два пересечения означают один полный цикл синусоиды).

// Обработчик прерывания таймера 1 (срабатывает раз в 1 секунду)

ISR**(**TIMER1\_COMPA\_vect**)** **{**

// Расчет частоты сигнала:

// Два пересечения через "ноль" означают один полный цикл (синусоида проходит вверх и вниз).

// Поэтому делим количество пересечений на 2, чтобы получить частоту в герцах.

signal\_frequency **=** zero\_cross\_count **/** 2.0**;**

// Расчет амплитуды сигнала:

// Амплитуда — это разница между максимальным и минимальным значением АЦП, зафиксированным за последний цикл измерений.

signal\_amplitude **=** max\_value **-** min\_value**;**

// Сброс счетчика пересечений через "ноль" для нового цикла измерений в следующую секунду.

zero\_cross\_count **=** 0**;**

// Сброс максимального и минимального значений для нового цикла измерений.

max\_value **=** 0**;**

min\_value **=** 1023**;** // Для 10-битного АЦП максимальное значение — 1023 (это означает "0" для минимума в следующий цикл).

// Устанавливаем флаг, что измерения завершены и доступны для вывода или дальнейшей обработки.

measurement\_done **=** 1**;**

**}**

Таким образом, АЦП непрерывно проводит преобразования сигнала, данные о частоте и амплитуде обновляются раз в секунду, и после завершения измерений сигнал готов к отображению или дальнейшей обработке.

### Работа с сдвиговым регистром 74HC595

Сдвиговый регистр 74HC595 является важным компонентом для управления семисегментными индикаторами в проекте, так как позволяет минимизировать количество используемых выводов микроконтроллера. Основная задача сдвигового регистра — это последовательная передача данных на семисегментные индикаторы и преобразование их в параллельные сигналы.

#### Инициализация сдвигового регистра

Перед началом работы с 74HC595 необходимо настроить его ключевые пины: MR (Master Reset) и OE (Output Enable). Пин MR должен быть установлен в логическую единицу, чтобы сброс был неактивен, а OE в логический ноль для активации выходов.

// Инициализация MR и OE для 74HC595

void init\_shift\_register\_control**()** **{**

// Настройка пинов MR и OE как выходы

DDRB **|=** **(**1 **<<** MR\_PIN**)** **|** **(**1 **<<** OE\_PIN**);**

// Установка MR в логическую 1 (работа без сброса)

PORTB **|=** **(**1 **<<** MR\_PIN**);**

// Установка OE в логический 0 (включение выходов сдвигового регистра)

PORTB **&=** **~(**1 **<<** OE\_PIN**);**

**}**

#### Передача данных в сдвиговый регистр

Для передачи данных в 74HC595 используется функция shiftOut(). Эта функция отправляет 8 бит данных в регистр поочередно, начиная с самого старшего бита (MSB). Каждый бит передается через линию данных (DS), а сигнал синхронизации подается через пин SH\_CP (Shift Clock). После того как все биты переданы, необходимо подать сигнал обновления через ST\_CP (Storage Clock), чтобы данные появились на выходах регистра.

// Отправка байта в сдвиговый регистр 74HC595

void shiftOut**(**uint8\_t data**)** **{**

**for** **(**int i **=** 0**;** i **<** 8**;** i**++)** **{**

// Установка значения данных на пине DS

**if** **(**data **&** **(**1 **<<** **(**7 **-** i**)))** **{**

PORTB **|=** **(**1 **<<** DS\_PIN**);**

**}** **else** **{**

PORTB **&=** **~(**1 **<<** DS\_PIN**);**

**}**

// Подача тактового сигнала на SH\_CP для передачи одного бита

PORTB **|=** **(**1 **<<** SH\_CP\_PIN**);**

\_delay\_us**(**10**);**

PORTB **&=** **~(**1 **<<** SH\_CP\_PIN**);**

**}**

// Подача сигнала на ST\_CP для обновления выходов регистра

PORTB **|=** **(**1 **<<** ST\_CP\_PIN**);**

\_delay\_us**(**10**);**

PORTB **&=** **~(**1 **<<** ST\_CP\_PIN**);**

**}**

#### Управление разрядами семисегментного индикатора

Каждый разряд семисегментного индикатора управляется с помощью функции select\_digit(). Эта функция активирует нужный разряд путем установки соответствующего пина в логическую единицу. Все другие разряды при этом отключаются.

// Выбор разряда семисегментного индикатора

void select\_digit**(**uint8\_t digit**)** **{**

// Отключение всех разрядов

PORTC **&=** **~((**1 **<<** DIGIT\_1**)** **|** **(**1 **<<** DIGIT\_2**)** **|** **(**1 **<<** DIGIT\_3**)** **|** **(**1 **<<** DIGIT\_4**));**

// Включение нужного разряда

**switch** **(**digit**)** **{**

**case** 1**:** PORTC **|=** **(**1 **<<** DIGIT\_1**);** **break;**

**case** 2**:** PORTC **|=** **(**1 **<<** DIGIT\_2**);** **break;**

**case** 3**:** PORTC **|=** **(**1 **<<** DIGIT\_3**);** **break;**

**case** 4**:** PORTC **|=** **(**1 **<<** DIGIT\_4**);** **break;**

**default:** **break;**

**}**

**}**

#### Отображение данных на семисегментном индикаторе

Данные передаются на семисегментный индикатор через сдвиговый регистр 74HC595. Функция display\_value() используется для отображения значения на индикаторе. Для этого значение преобразуется в строку, затем поочередно отправляются данные для каждого разряда.

// Отображение числа на семисегментном индикаторе

void display\_value**(**float value**)** **{**

// Преобразуем значение в строку без округления

char buffer**[**8**];**

dtostrf**(**value**,** 4**,** 3**,** buffer**);** // Преобразуем значение float в строку формата "XX.XXX"

uint8\_t display\_index **=** 4**;** // Индекс для разрядов (начинаем с левого)

uint8\_t digit\_index **=** 0**;** // Индекс для символов строки

// Проходим по каждому символу в строке

**while** **(**buffer**[**digit\_index**]** **!=** '\0' **&&** display\_index **>** 0**)** **{**

// Отображение цифры на индикаторе

**if** **(**buffer**[**digit\_index**]** **>=** '0' **&&** buffer**[**digit\_index**]** **<=** '9'**)** **{**

select\_digit**(**display\_index**);** // Выбираем разряд

shiftOut**(**digits**[**buffer**[**digit\_index**]** **-** '0'**]);** // Отправляем цифру на индикатор

display\_index**--;** // Переходим к следующему разряду

**}**

// Обработка десятичной точки

**else** **if** **(**buffer**[**digit\_index**]** **==** '.'**)** **{**

select\_digit**(**display\_index **+** 1**);** // Добавляем точку к предыдущему символу

shiftOut**(**digits**[**buffer**[**digit\_index **-** 1**]** **-** '0'**]** **&** **~(**1 **<<** 7**));** // Включаем точку

**}**

digit\_index**++;** // Переходим к следующему символу строки

\_delay\_ms**(**segment\_delay**);** // Задержка для стабильного отображения

**}**

// Выключаем оставшиеся индикаторы, если они не используются

**while** **(**display\_index **>** 0**)** **{**

select\_digit**(**display\_index**);** // Выбираем разряд

shiftOut**(**0xFF**);** // Выключаем индикатор (0xFF выключает все сегменты)

display\_index**--;**

\_delay\_ms**(**segment\_delay**);**

**}**

**}**

С помощью сдвигового регистра 74HC595 можно передавать данные последовательно, экономя выводы микроконтроллера и облегчая управление семисегментными индикаторами. Этот механизм позволяет легко управлять несколькими разрядами индикаторов с помощью небольшого числа управляющих линий, что делает его особенно полезным в проектах с ограниченным количеством выводов.

### Работа со сдвиговым регистром и семисегментным индикатором

Сдвиговый регистр 74HC595 позволяет передавать данные последовательно и выводить их параллельно на нужные разряды семисегментного индикатора. Это позволяет экономить выводы микроконтроллера и использовать меньшее количество проводов для управления множеством индикаторов.

#### Инициализация сдвигового регистра

Для начала работы с сдвиговым регистром необходимо инициализировать его основные выводы. Это включает настройку пинов MR (Master Reset) и OE (Output Enable), чтобы регистрация данных могла происходить корректно.

// Инициализация MR и OE для 74HC595

void init\_shift\_register\_control**()** **{**

// Настройка пинов MR и OE как выходов

DDRB **|=** **(**1 **<<** MR\_PIN**)** **|** **(**1 **<<** OE\_PIN**);**

// Установка MR в логическую 1 (нормальная работа, сброс не активен)

PORTB **|=** **(**1 **<<** MR\_PIN**);**

// Установка OE в логический 0 (включение выходов сдвигового регистра)

PORTB **&=** **~(**1 **<<** OE\_PIN**);**

**}**

Этот код устанавливает начальные значения для управления сдвиговым регистром. Пин MR управляет сбросом регистра, который не должен быть активирован, а OE управляет включением или отключением выходов.

#### Отправка данных в сдвиговый регистр

Для того чтобы передать данные в сдвиговый регистр, необходимо последовательно отправить 8 бит через линию данных (DS), синхронизируя передачу с помощью тактового сигнала (SH\_CP). После передачи всех битов данные фиксируются на выходе с помощью ST\_CP.

// Отправка байта в сдвиговый регистр 74HC595

void shiftOut**(**uint8\_t data**)** **{**

// Передача 8 бит данных в сдвиговый регистр

**for** **(**int i **=** 0**;** i **<** 8**;** i**++)** **{**

// Передача каждого бита данных

**if** **(**data **&** **(**1 **<<** **(**7 **-** i**)))** **{**

PORTB **|=** **(**1 **<<** DS\_PIN**);**

**}** **else** **{**

PORTB **&=** **~(**1 **<<** DS\_PIN**);**

**}**

// Подача тактового сигнала на SH\_CP

PORTB **|=** **(**1 **<<** SH\_CP\_PIN**);**

\_delay\_us**(**10**);**

PORTB **&=** **~(**1 **<<** SH\_CP\_PIN**);**

**}**

// Подача сигнала на ST\_CP для фиксации данных на выходе

PORTB **|=** **(**1 **<<** ST\_CP\_PIN**);**

\_delay\_us**(**10**);**

PORTB **&=** **~(**1 **<<** ST\_CP\_PIN**);**

**}**

Функция shiftOut последовательно передает биты данных в сдвиговый регистр. Важным моментом является использование тактового сигнала (SH\_CP) для синхронизации передачи каждого бита, а ST\_CP применяется для фиксации данных на выходах после завершения передачи всех 8 бит.

#### Управление разрядами индикатора

Для отображения данных на семисегментных индикаторах используется метод управления разрядами. В каждом цикле мы активируем один из разрядов с помощью соответствующего пина, используя функцию select\_digit().

// Выбор разряда семисегментного индикатора

void select\_digit**(**uint8\_t digit**)** **{**

// Отключение всех разрядов

PORTC **&=** **~((**1 **<<** DIGIT\_1**)** **|** **(**1 **<<** DIGIT\_2**)** **|** **(**1 **<<** DIGIT\_3**)** **|** **(**1 **<<** DIGIT\_4**));**

// Включение нужного разряда

**switch** **(**digit**)** **{**

**case** 1**:** PORTC **|=** **(**1 **<<** DIGIT\_1**);** **break;**

**case** 2**:** PORTC **|=** **(**1 **<<** DIGIT\_2**);** **break;**

**case** 3**:** PORTC **|=** **(**1 **<<** DIGIT\_3**);** **break;**

**case** 4**:** PORTC **|=** **(**1 **<<** DIGIT\_4**);** **break;**

**default:** **break;**

**}**

**}**

Функция select\_digit() отключает все разряды и включает только нужный разряд на семисегментном индикаторе. Это позволяет обновлять данные на индикаторе поочередно для каждого разряда.

#### Отображение данных на индикаторе

Данные для отображения передаются через сдвиговый регистр, а затем с помощью функции display\_value() преобразуются в вид, подходящий для вывода на семисегментный индикатор.

// Отображение числа на семисегментном индикаторе

void display\_value**(**float value**)** **{**

// Преобразуем значение в строку с фиксированной точностью

char buffer**[**8**];**

dtostrf**(**value**,** 4**,** 3**,** buffer**);** // Преобразуем значение float в строку формата "XX.XXX"

uint8\_t display\_index **=** 4**;** // Индекс для разрядов (начинаем с самого левого)

uint8\_t digit\_index **=** 0**;** // Индекс для символов строки

// Проходим по каждому символу в строке

**while** **(**buffer**[**digit\_index**]** **!=** '\0' **&&** display\_index **>** 0**)** **{**

**if** **(**buffer**[**digit\_index**]** **>=** '0' **&&** buffer**[**digit\_index**]** **<=** '9'**)** **{**

// Отображение цифры на индикаторе

select\_digit**(**display\_index**);** // Выбираем разряд

shiftOut**(**digits**[**buffer**[**digit\_index**]** **-** '0'**]);** // Отправляем цифру на индикатор

display\_index**--;** // Переход к следующему разряду

**}** **else** **if** **(**buffer**[**digit\_index**]** **==** '.'**)** **{**

// Добавляем десятичную точку

select\_digit**(**display\_index **+** 1**);** // Устанавливаем точку к предыдущему разряду

shiftOut**(**digits**[**buffer**[**digit\_index **-** 1**]** **-** '0'**]** **&** **~(**1 **<<** 7**));** // Включаем точку

**}**

digit\_index**++;** // Переход к следующему символу

\_delay\_ms**(**segment\_delay**);** // Задержка для отображения

**}**

// Выключение оставшихся индикаторов

**while** **(**display\_index **>** 0**)** **{**

select\_digit**(**display\_index**);** // Отключение разряда

shiftOut**(**0xFF**);** // Выключение всех сегментов

display\_index**--;**

\_delay\_ms**(**segment\_delay**);**

**}**

**}**

Функция display\_value() отвечает за преобразование числа в строку с десятичной точкой и последовательное отображение символов на семисегментном индикаторе.

### Работа с виртуальным терминалом

Виртуальный терминал используется для визуализации данных, поступающих от микроконтроллера ATmega328P через последовательный интерфейс UART. В ходе лабораторной работы терминал служит средством для наблюдения за измеренными параметрами, такими как частота, амплитуда сигнала и текущее значение напряжения. В данном проекте передача данных на виртуальный терминал реализована через UART.

#### Инициализация UART

Для передачи данных по UART выполняется инициализация UART, где задается скорость передачи, формат данных и включается передатчик. В нашем случае скорость передачи составляет 9600 бод при частоте микроконтроллера 8 МГц.

// Инициализация UART

void uart\_init**()** **{**

// Установка значения предделителя для скорости 9600 бод при частоте 8 МГц

uint16\_t ubrr\_value **=** 51**;**

UBRR0H **=** **(**ubrr\_value **>>** 8**);**

UBRR0L **=** ubrr\_value**;**

// Включение передатчика

UCSR0B **=** **(**1 **<<** TXEN0**);**

// Установка формата данных - 8 бит данных, 1 стоповый бит

UCSR0C **=** **(**1 **<<** UCSZ01**)** **|** **(**1 **<<** UCSZ00**);**

**}**

Функция uart\_init() включает передатчик и настраивает UART на передачу данных длиной 8 бит с 1 стоп-битом.

#### Отправка данных на виртуальный терминал

После инициализации UART данные могут передаваться на виртуальный терминал в виде отдельных байтов или строк. Для отправки строк используется функция uart\_print().

// Отправка одного байта через UART

void uart\_transmit**(**uint8\_t data**)** **{**

// Ожидание готовности регистра передачи данных

**while** **(!(**UCSR0A **&** **(**1 **<<** UDRE0**)));**

// Передача байта данных

UDR0 **=** data**;**

**}**

// Отправка строки через UART

void uart\_print**(**const char**\*** str**)** **{**

// Передача каждого символа строки

**while** **(\***str**)** **{**

uart\_transmit**(\***str**++);**

**}**

**}**

Функция uart\_transmit() отправляет один байт данных, а uart\_print() последовательно передает каждый символ строки.

#### Логирование данных

При помощи функции uart\_print() данные о частоте, амплитуде и текущем значении напряжения отправляются на виртуальный терминал. Форматированные строки подготавливаются с использованием буфера и функции dtostrf() для преобразования числовых значений в строку.

// Логирование измеренных данных на виртуальный терминал

void log\_measurements**()** **{**

char buffer**[**32**];**

// Логирование частоты

uart\_print**(**"Frequency: "**);**

dtostrf**(**signal\_frequency**,** 6**,** 2**,** buffer**);**

uart\_print**(**buffer**);**

uart\_print**(**" Hz\r\n"**);**

// Логирование амплитуды

uart\_print**(**"Amplitude: "**);**

dtostrf**(**get\_amplitude\_in\_volts**(),** 4**,** 2**,** buffer**);**

uart\_print**(**buffer**);**

uart\_print**(**" V\r\n"**);**

// Логирование текущего напряжения

uart\_print**(**"Current Voltage: "**);**

dtostrf**(**get\_current\_voltage**(),** 4**,** 2**,** buffer**);**

uart\_print**(**buffer**);**

uart\_print**(**" V\r\n\r\n"**);**

**}**

Функция log\_measurements() отправляет на терминал текущие значения измеренных параметров, обновляя информацию каждую секунду после завершения цикла измерений.

# ТЕСТИРОВАНИЕ

### Вывод значений на терминал

Тестирование процесса вывода измеренных значений на виртуальный терминал проводилось для проверки корректной работы UART-передачи данных, а также точности расчета частоты, амплитуды и текущего значения напряжения. Для тестирования к аналоговому входу микроконтроллера ATmega328P был подключен внешний источник синусоидального сигнала, параметры которого, такие как частота и амплитуда, задавались в настройках источника, изображенных на рисунке 4.

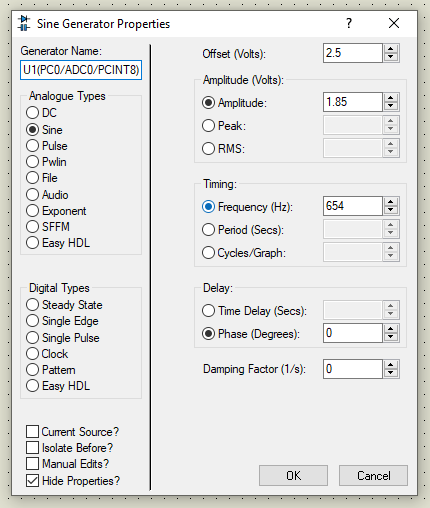


Рисунок 4 — Настройки внешнего синусоидального источника напряжения

При включении питания микроконтроллера и запуске программы в виртуальном терминале отображались данные, передаваемые через UART, включая частоту сигнала, амплитуду и текущее значение напряжения на входе АЦП. Значения обновлялись каждую секунду, что позволяло оперативно отслеживать изменения внешнего сигнала в реальном времени. На рисунке 5 представлен пример вывода терминала, демонстрирующий непрерывное обновление информации по мере изменения частоты и амплитуды сигнала.

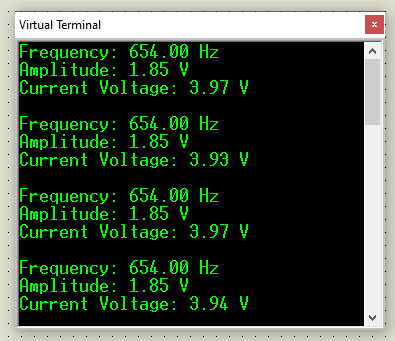


Рисунок 5 — Окно виртуального терминала

Корректное отображение данных на виртуальном терминале подтвердило правильную настройку UART и корректное выполнение всех операций по обработке и передаче значений на терминал.

### Вывод частоты на семисегментный индикатор

Для визуального отображения частоты измеряемого сигнала был настроен вывод значений на четырехразрядный семисегментный индикатор, управление которым выполнялось через сдвиговый регистр 74HC595. При запуске программы после подачи питания микроконтроллер отображает текущую частоту сигнала, измеряемую каждые секунду, на индикаторе в формате "XX.XX" герц. Последовательный вывод разрядов позволяет отображать каждую цифру поочередно, обеспечивая стабильное восприятие показаний.

На рисунках 6–9 показан процесс вывода значений частоты на семисегментный индикатор, где каждый рисунок иллюстрирует последовательное включение соответствующего разряда. На каждом этапе происходит подача данных для одного разряда, что позволяет выводить полное значение частоты в виде четырех символов. Эти последовательные обновления подтверждают корректность настройки сдвигового регистра и алгоритма вывода данных на семисегментный индикатор.

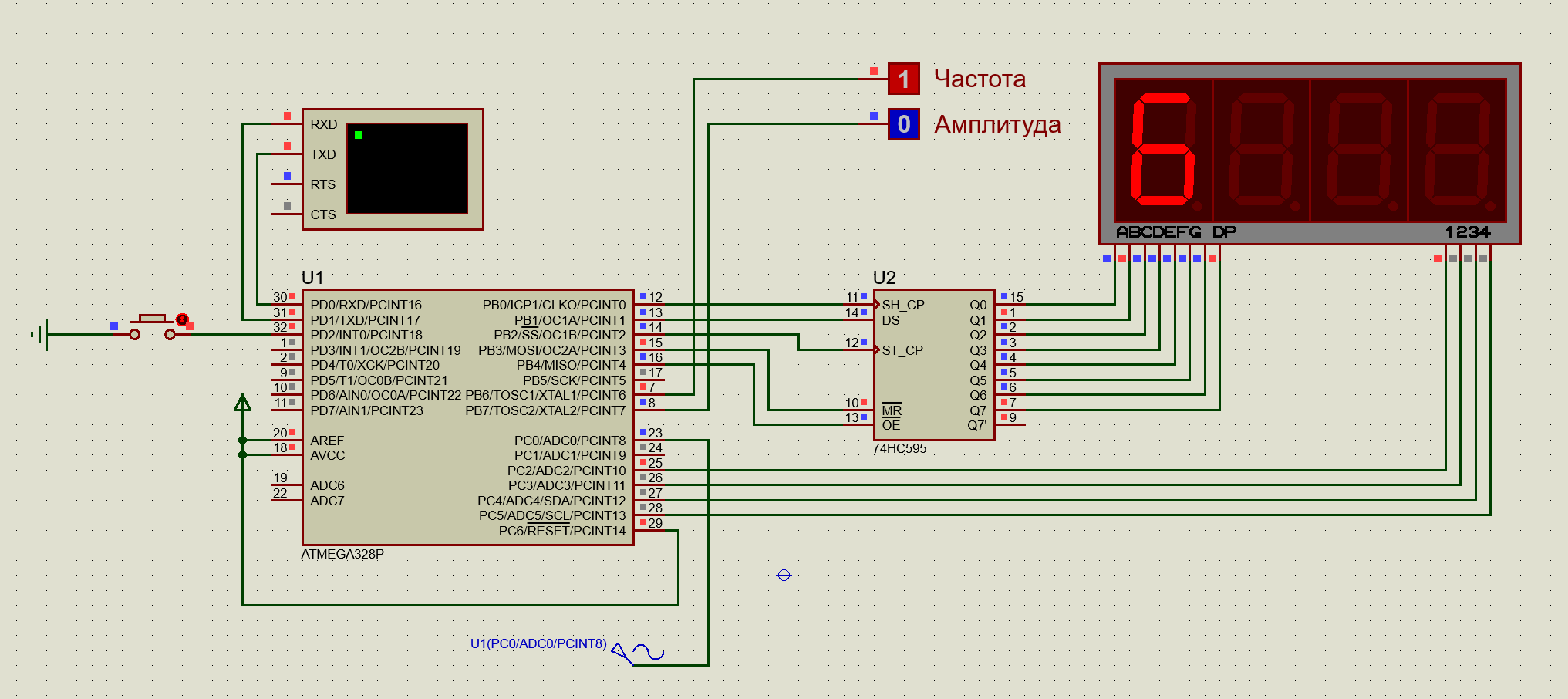


Рисунок 6 — Частота. Вывод старшего разряда на семисегментный индикатор

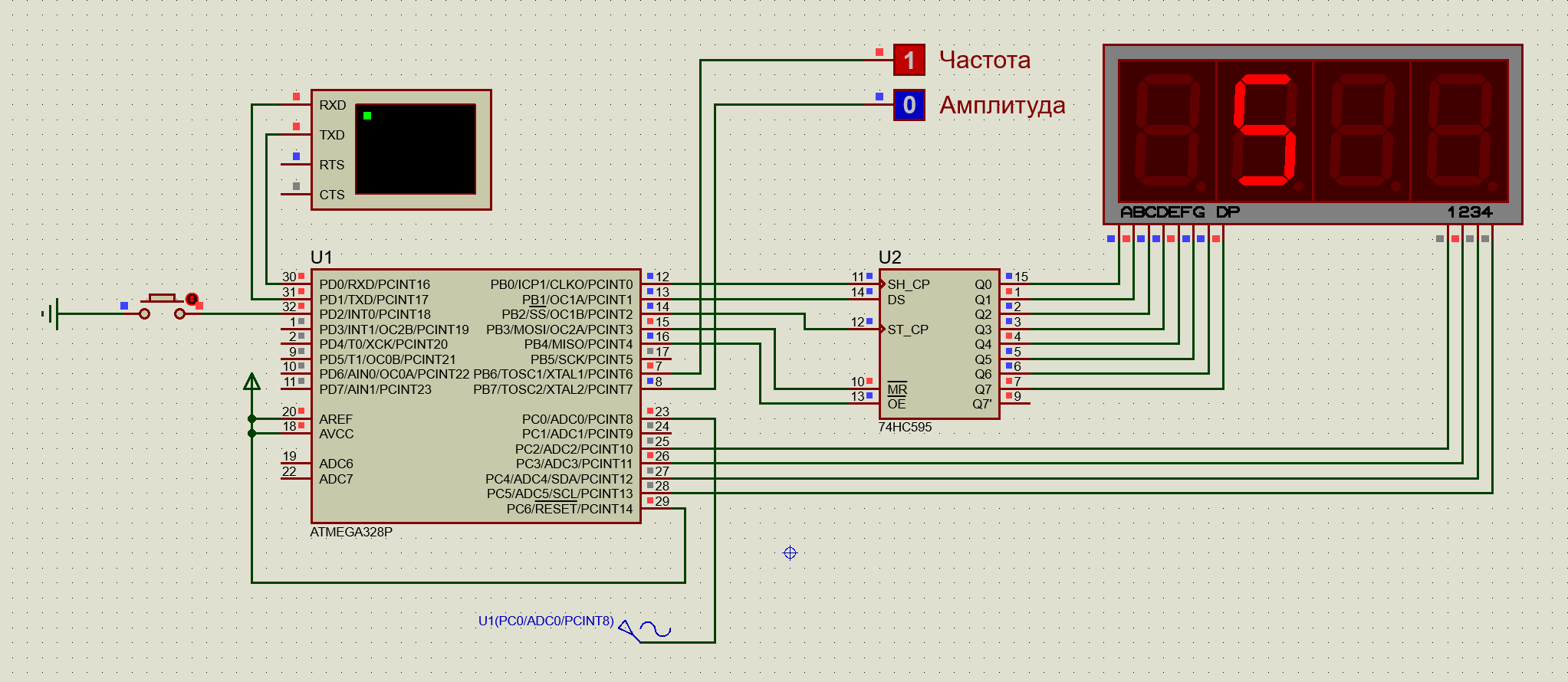


Рисунок 6 — Частота. Вывод второго разряда на семисегментный индикатор

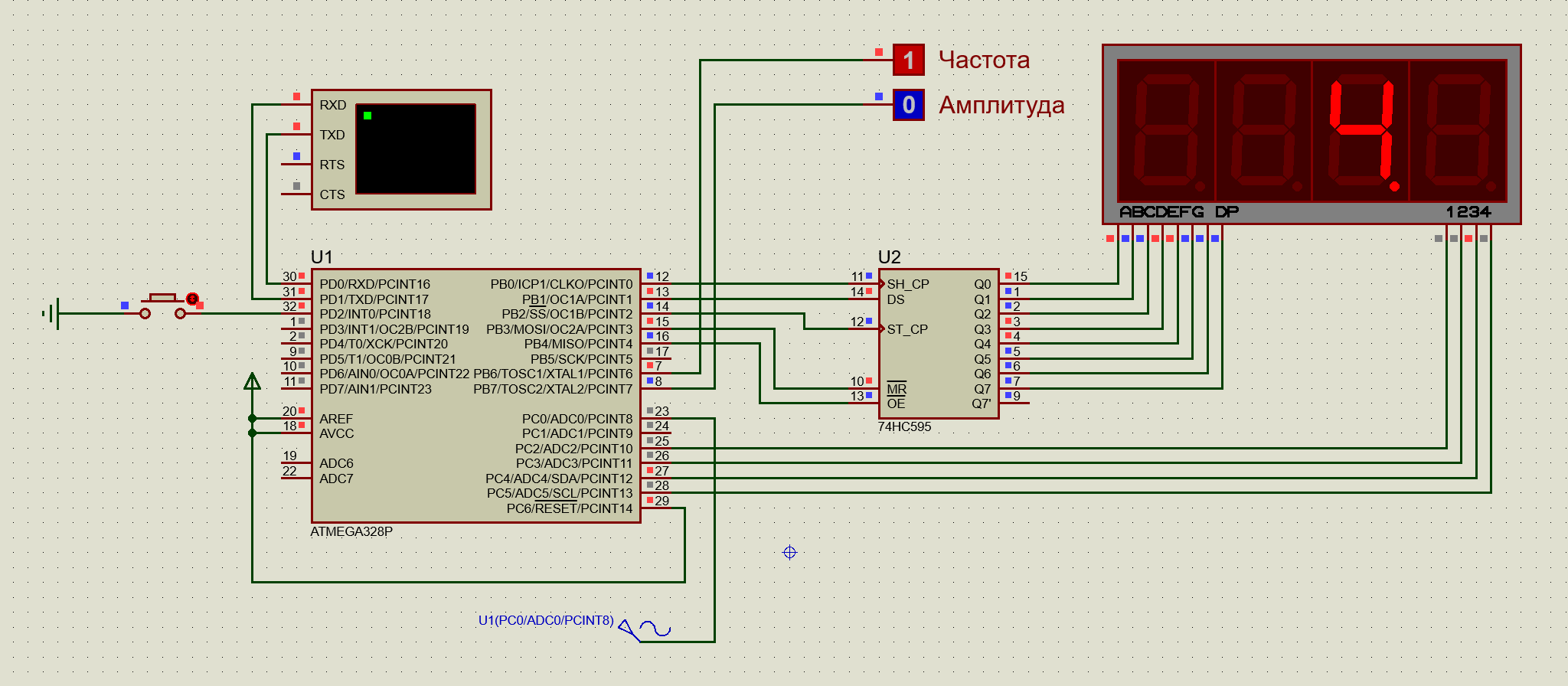


Рисунок 6 — Частота. Вывод третьего разряда на семисегментный индикатор

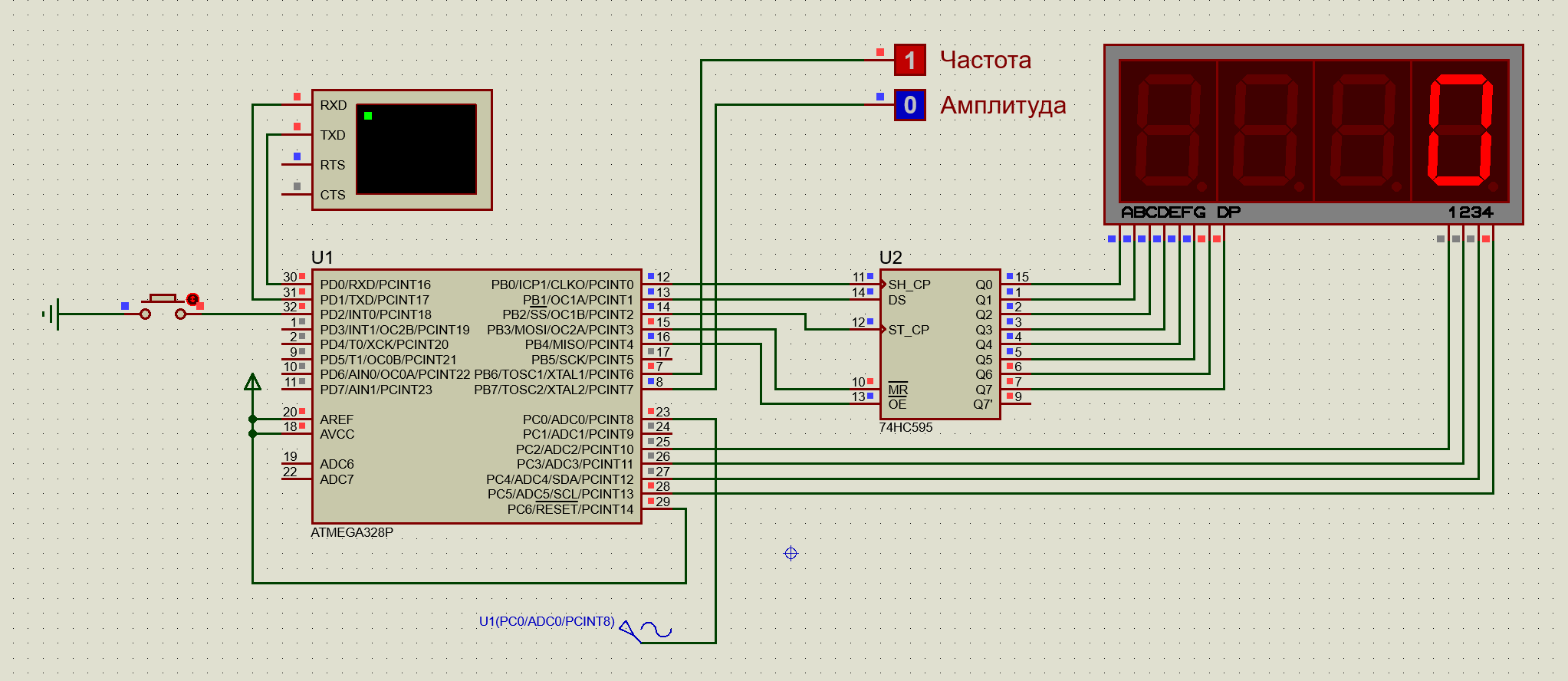


Рисунок 6 — Частота. Вывод младшего разряда на семисегментный индикатор

Тестирование показало, что индикатор стабильно и четко отображает измеренную частоту, которая обновляется каждую секунду, синхронно с данными на виртуальном терминале, обеспечивая пользователю возможность наблюдать за изменением частоты в реальном времени.

### Вывод амплитуды на семисегментный индикатор

Для отображения амплитуды сигнала на семисегментном индикаторе в данной системе используется однократное нажатие на кнопку, подключенную к PD2. После нажатия индикатор переключается в режим отображения амплитуды, что позволяет пользователю наблюдать значения амплитуды сигнала без необходимости дополнительных настроек.

Процесс вывода амплитуды на индикатор осуществляется поочередно для каждого разряда. На рисунках 10-13 показаны последовательные этапы отображения значений амплитуды, где каждый рисунок представляет включение одного разряда. Формат вывода сохраняется как "XX.XX" вольт, что позволяет отобразить амплитуду с точностью до сотых долей. В ходе тестирования было установлено, что значения амплитуды отображаются корректно и стабильно, что свидетельствует о надежности алгоритма переключения режима индикации и правильности настройки работы сдвигового регистра и семисегментного индикатора.

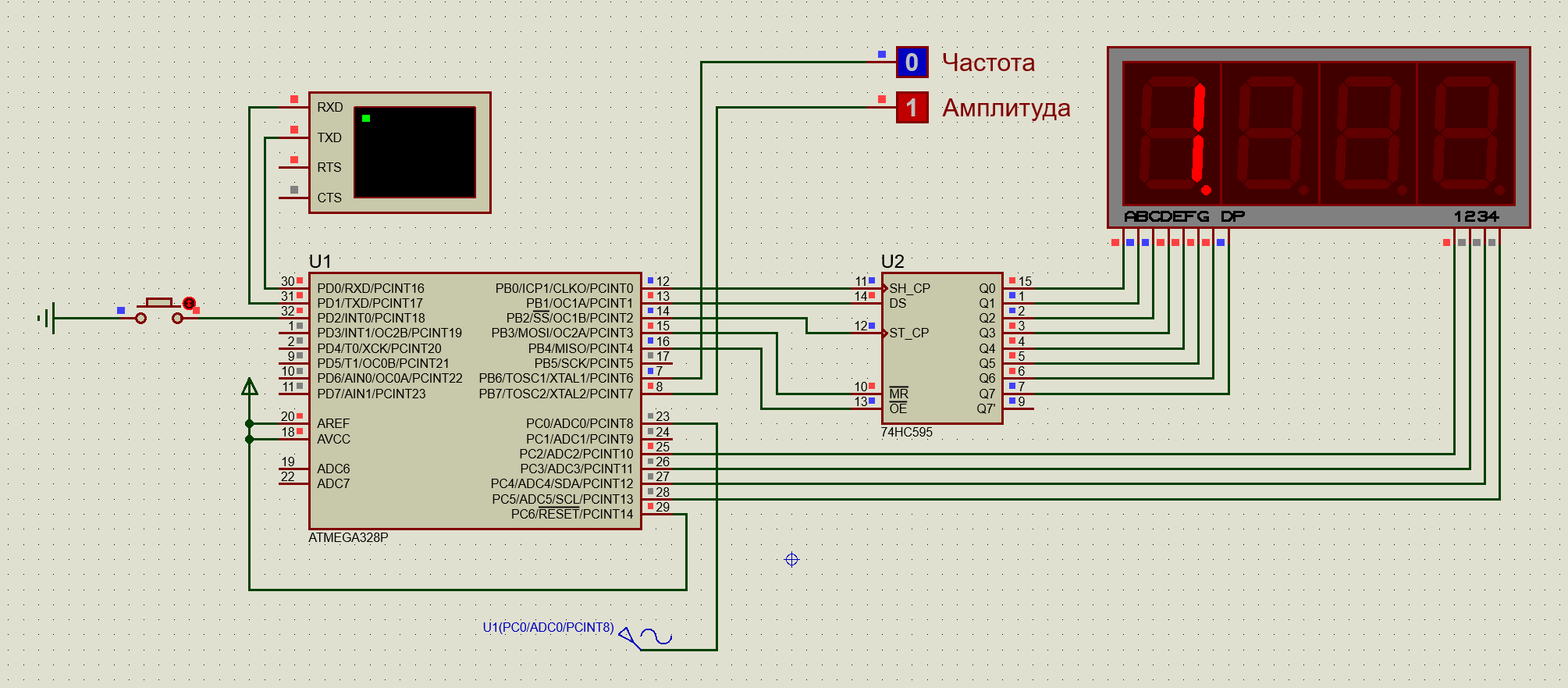


Рисунок 10 — Амплитуда. Вывод старшего разряда на семисегментный индикатор

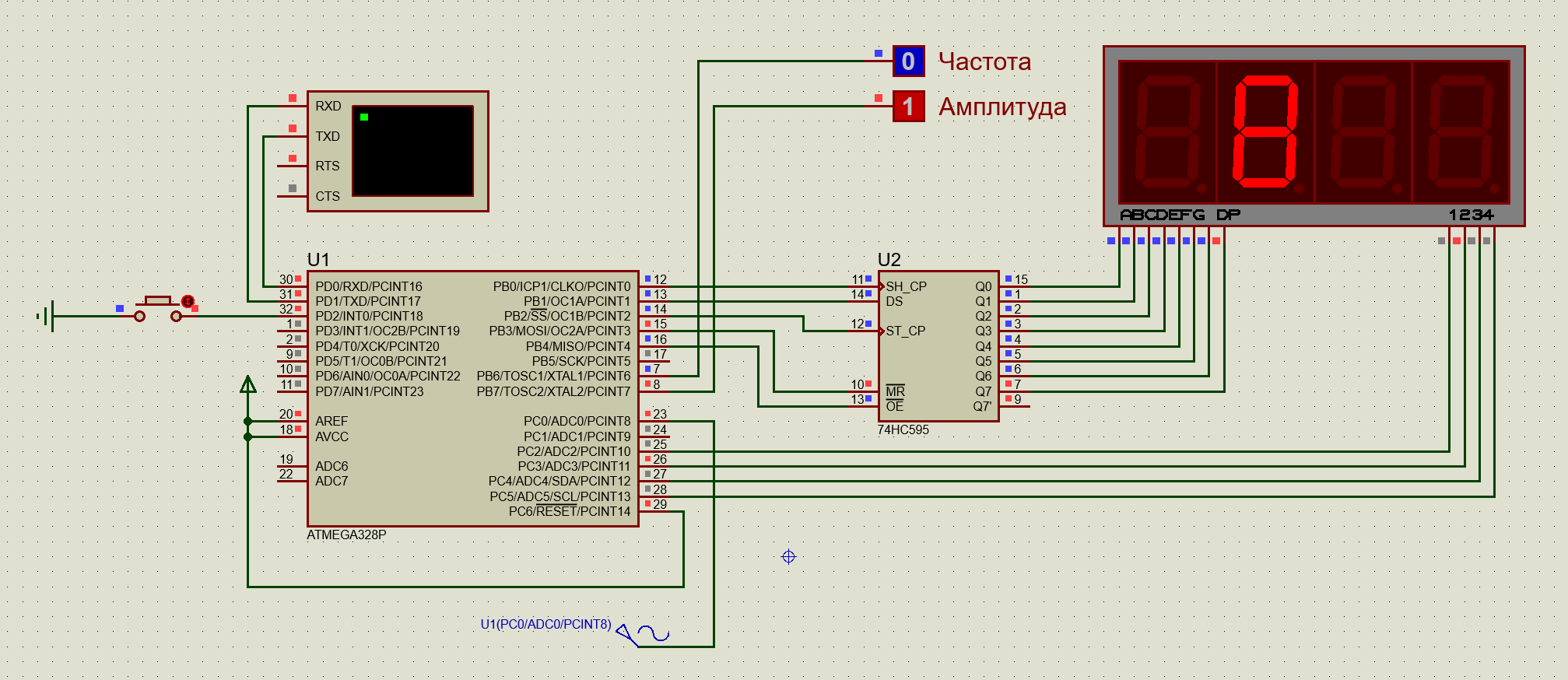


Рисунок 11 — Амплитуда. Вывод второго разряда на семисегментный индикатор

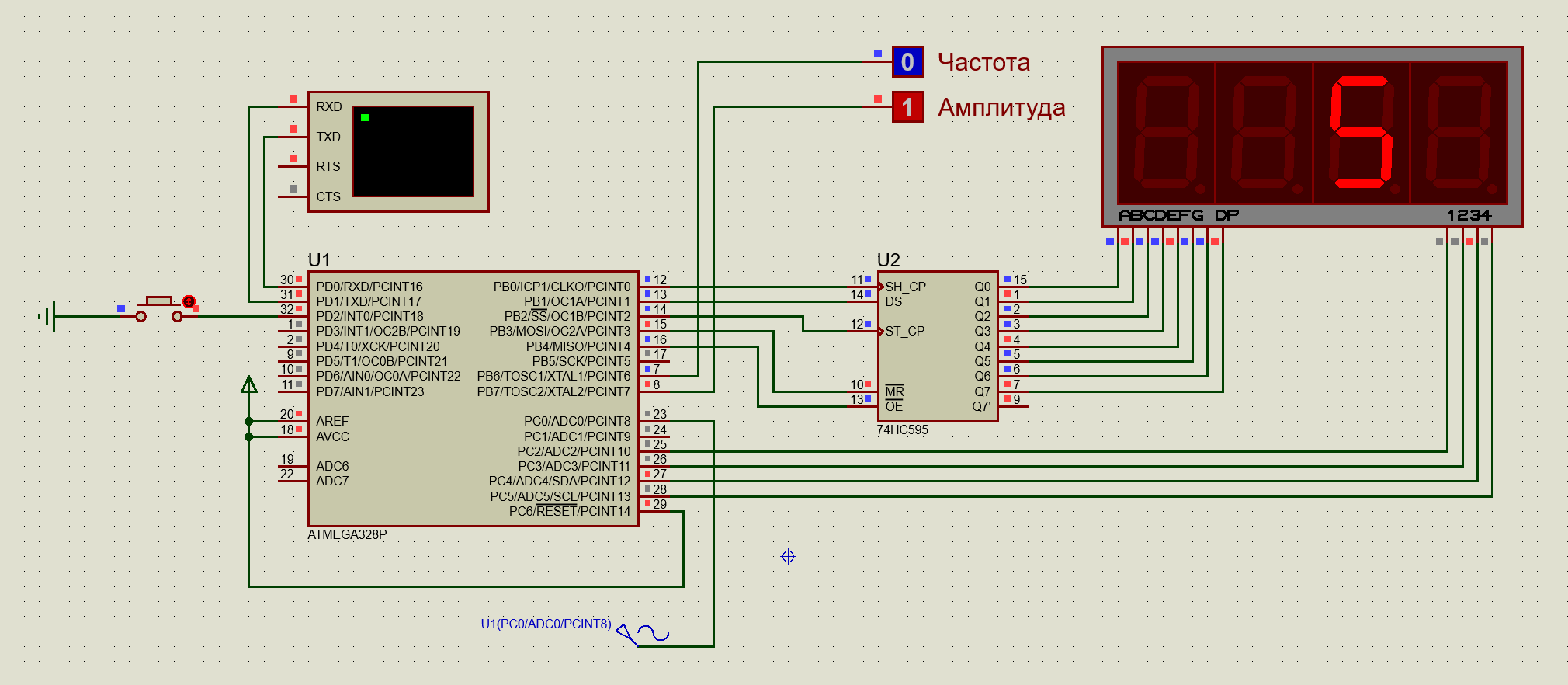


Рисунок 12 — Амплитуда. Вывод третьего разряда на семисегментный индикатор

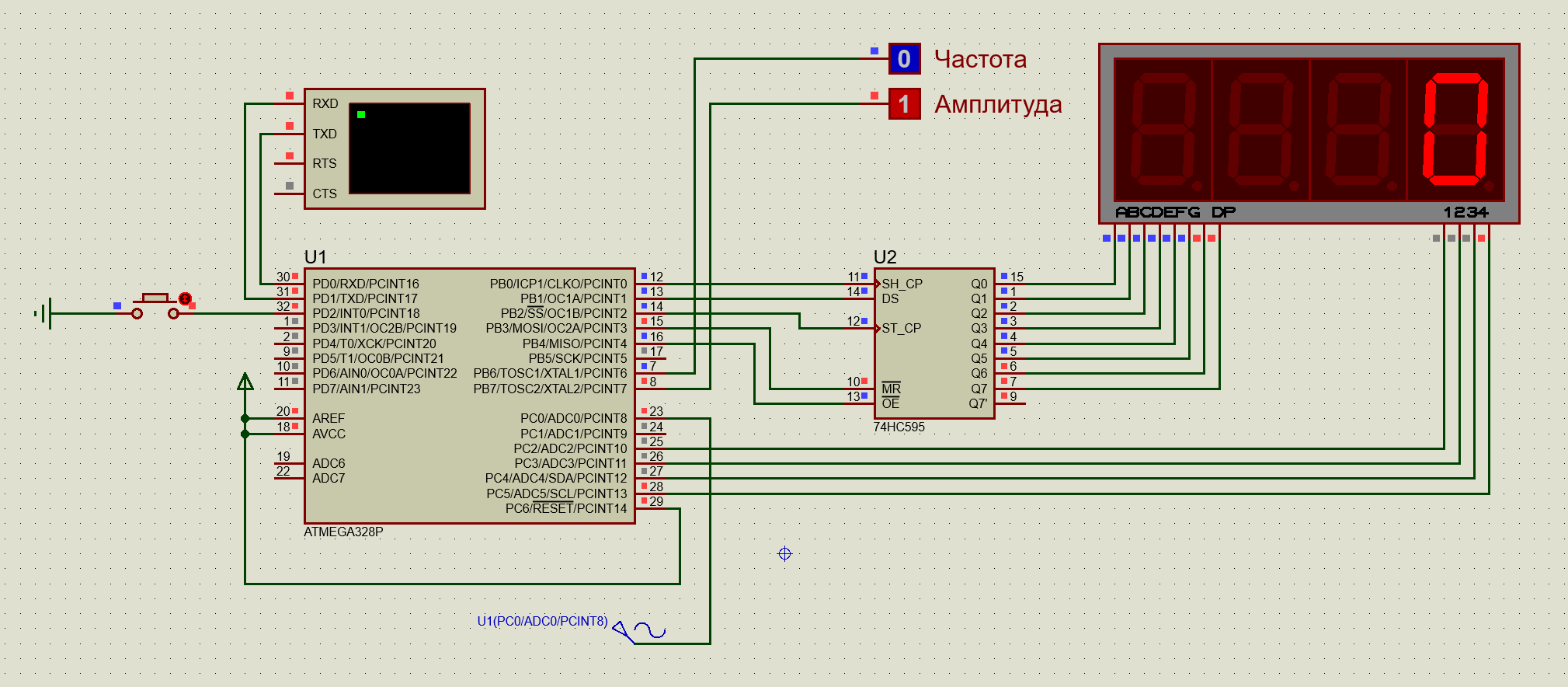


Рисунок 13 — Амплитуда. Вывод младшего разряда на семисегментный индикатор

Таким образом, система отображения амплитуды на индикаторе после нажатия на кнопку обеспечивает наблюдение за параметрами сигнала в реальном времени.

### Изменение частоты и амплитуды внешнего источника

В ходе тестирования устройства проводится проверка реакции системы на изменения параметров внешнего источника сигнала, таких как частота и амплитуда. На рисунке 14 показаны новые настройки внешнего источника напряжения, где изменены значения частоты и амплитуды. Эти параметры обновляются на микроконтроллере в режиме реального времени, что позволяет наблюдать их отображение как на виртуальном терминале, так и на семисегментном индикаторе.

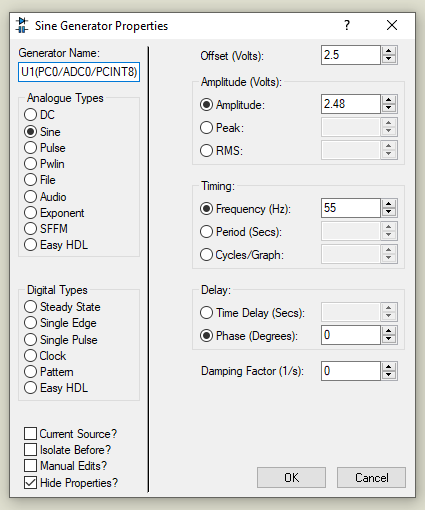


Рисунок 14 — Настройки внешнего синусоидального источника напряжения

После однократного нажатия на кнопку устройство переходит в режим отображения амплитуды, что также фиксируется на виртуальном терминале. На рисунке 15 приведено окно виртуального терминала, где отображены обновленные значения частоты и амплитуды после изменений параметров источника. Эти данные подтверждают правильную работу системы по измерению и отображению обновленных значений, что позволяет гибко контролировать и анализировать сигнал в режиме реального времени.

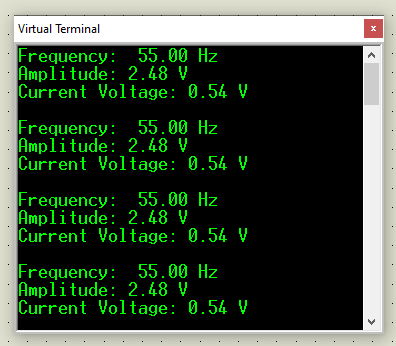


Рисунок 15 — Окно виртуального терминала

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была реализована и протестирована система измерения параметров синусоидального сигнала с помощью микроконтроллера ATmega328P. Основной целью было изучить процесс считывания аналоговых данных, их обработки и вывода результатов на различные интерфейсы, в том числе на семисегментный индикатор и виртуальный терминал. Для выполнения работы были выбраны компоненты, позволяющие наиболее полно раскрыть возможности микроконтроллера в условиях выполнения лабораторного эксперимента. С помощью периферийных модулей, таких как АЦП, таймеры и система прерываний, удалось организовать точное измерение частоты и амплитуды входного сигнала, что позволило на практике оценить методику работы с аналоговыми сигналами и изучить алгоритмы их цифровой обработки.

Одной из важнейших частей проекта стало использование сдвигового регистра 74HC595 для управления семисегментным индикатором. Сдвиговый регистр позволил эффективно распределить сигналы управления для многозначного отображения, обеспечив правильное переключение разрядов и последовательный вывод информации. Это не только освободило выводы микроконтроллера для других задач, но и продемонстрировало применение сдвиговых регистров в схеме управления индикацией. Настройка и тестирование регистра на передачу данных и контроль разрядов индикатора стало одним из ключевых этапов работы, требовавшим точной настройки времени переключения и синхронизации.

Для отображения информации на семисегментном индикаторе был разработан алгоритм, позволяющий выводить значения частоты и амплитуды в десятичном формате. Это дало возможность не только просматривать результаты измерений, но и анализировать динамику изменений, вызванных воздействием на источник сигнала. Благодаря четко продуманной реализации смены режимов отображения, управляемой внешней кнопкой, пользователь может переключаться между выводом частоты и амплитуды, что расширяет функциональные возможности устройства. Реализация данного механизма предоставила практическое понимание принципов работы с прерываниями, используемыми для обработки нажатий кнопки и выполнения требуемых действий по смене отображаемых данных.

Тестирование работы системы включало проверку вывода измеряемых параметров на виртуальный терминал, что позволило визуально контролировать точность получаемых данных и корректность алгоритмов расчета. Вывод данных на виртуальный терминал помог убедиться в стабильности работы системы и отсутствии сбоев при изменении значений частоты и амплитуды сигнала. Помимо этого, использование виртуального терминала для передачи данных открыло возможности для интеграции устройства с другими программными системами для дальнейшей обработки данных. Изменение частоты и амплитуды сигнала внешнего источника продемонстрировало устойчивость работы системы и правильность алгоритма обработки данных. При изменении этих параметров система корректно фиксировала новые значения и отображала их как на индикаторе, так и на терминале, что подтверждает работоспособность всей схемы.

В результате выполнения работы были получены знания и практические навыки по настройке и программированию микроконтроллера для задач измерения аналоговых сигналов, работы с периферийными модулями, а также создания интерфейса для вывода данных. Лабораторная работа позволила глубже понять принципы работы с микроконтроллерами, использования сдвигового регистра для управления индикаторами и применения прерываний для управления режимами работы системы. Эти знания закладывают прочную основу для создания более сложных систем, требующих точного измерения и отображения данных в реальном времени.

Все исходные файлы, коды и схемы, созданные в процессе выполнения лабораторной работы, находятся в репозитории на GitHub и доступны по адресу: «github.com/Andrey-Bedretdinov/PMC.LW.1». Репозиторий содержит полный комплект необходимых файлов для изучения и дальнейшего использования проекта, а также предоставляет возможность другим исследователям ознакомиться с методами и подходами, примененными в данной работе.