**Цель работы**: изучение сдвиговых регистров, подключение их к контроллеру Arduino и практическое использование.

**Теоретические сведения**

1. **Сдвиговый регистр 74HC595**

Микросхема 74HC595 – один из самых распространённых сдвиговых регистров: синхронный, с регистром данных (latch). Он позволяет увеличивать количество выходов микроконтроллера. Чип преобразовывает входной последовательный сигнал на одном пине (DS) в выходной параллельный на 8 пинах (QX).

74HC595 – восьмиразрядный сдвиговый регистр с последовательным вводом, последовательным или параллельным выводом информации, с триггером-защелкой и тремя состояниями на выходе.

В данном регистре используется принцип синхронизированной последовательной передачи сигналов. Необходимые значения сигнала (биты HIGH или LOW) передаются в регистр один за другим, «проталкиваясь» по цепочке D-триггеров c каждым тактовым импульсом на пине SHCP. Одновременный вывод на пины параллельного интерфейса обеспечивается так называемой защёлкой (latch) STCP, которая «не пускает» переданные биты на выводы раньше времени. Когда байт считан, значения всех 8 бит распределены по выходам. То есть, на вход регистра сигналы подаются последовательно, на выходах регистра имеем параллельно 8 сигналов.

Другими словами: этот регистр позволяет контролировать 8 выходов, используя всего несколько выходов на самом контроллере. При этом несколько таких регистров можно объединять последовательно для каскадирования.

Выход регистра может иметь значения не только логического нуля или единицы (HIGH или LOW), но и быть в высокоомном (высокоимпедансном) состоянии – когда выход отключен от схемы. В высокоомное состояние не может быть переведен отдельный выход, а только все выходы регистра разом. Например, если мы говорим об управлении светодиодами, это может быть полезно в случае, когда мы хотим переключить управление ими на другой контроллер.

Для понимания работы регистра стоит взглянуть на функциональную схему (рис. 1). Она состоит из:

* 8-битного регистра сдвига,
* 8-битного регистра хранения,
* 8-битного выходного регистра.

Таблица 1 – Распиновка входов/выходов регистра 74HC595

|  |  |  |  |
| --- | --- | --- | --- |
|  | Пины 1-7, 15 | Q0-Q7 | Параллельные выходы |
| Пин 8 | GND | Земля |
| Пин 9 | Q7" | Выход для последовательного соединения регистров |
| Пин 10 | M̅͞R | Сброс значений регистра. Сброс происходит при получение LOW |
| Пин 11 | SHCP | Вход для тактовых импульсов SCLK (SPI) |
| Пин 12 | STCP | Синронизация ("защелкивание") выходов SS |
| Пин 13 | O̅E | Вход для переключения состояния выходов из высокоомного в рабочее |
| Пин 14 | DS | Вход для последовательных данных MOSI |
| Пин 16 | Vcc | Питание |

**Входы 74HC595:**

**O͞E (13)** – вход переводящий выходы из высокоимпедансного состояние в рабочее состояние. При логической единице на этом входе выходы будут отключены от остальной части схемы. Это нужно например для того чтобы другая микросхема могла управлять этими сигналами.

Если нужно включить в рабочее состояние микросхеме подайте логический ноль на этот вход. А если в принципе не нужно переводить выходы в высокоимпедансное состояние – смело заземляйте этот вывод.

**M͞R (10)** – сброс регистра. Переводить все выходы в состояние логического нуля. Чтобы сбросить регистр нужно подать логический ноль на этот вход и подать положительный импульс на вход STCP.

Подключаем этот выход через резистор к питанию микросхемы и при необходимости замыкаем на землю.

**DS (14)** – вход данных. Последовательно подаваемые сюда данные будут появляются на 8-ми выходах регистра в параллельной форме.

**SHCP (11)** – вход для тактовых импульсов. Когда на тактовом входе SHCP появляется логическая единица, бит находящийся на входе данных DS считывается и записывается в самый младший разряд сдвигового регистра. При поступлении на тактовый вход следующего импульса высокого уровня, в сдвиговый регистр записывается следующий бит со входа данных. Тот бит который был записан ранее сдвигается на один разряд (из Q0 в Q1) , а его место занимает вновь пришедший бит. И так далее по цепочке.

**STCP (12)** – вход «защёлкивающий» данные. Что бы данные появились на выходах Q0…Q7 нужно подать логическую единицу на вход STCP. Данные поступают в параллельный регистр который сохряняет их до следующего импульса STCP.

**Выходы 74HC595:**

Q0…Q7 – выходы, которыми будем управлять. Могут находится в трёх состояниях: логическая единица, логический ноль и высокоимпедансное состояние

Q7′ – выход предназначенный для последовательного соединения регистров.

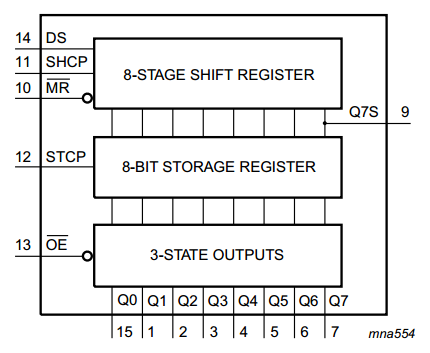


Рисунок 1 – Функциональная схема регистра 74HC595

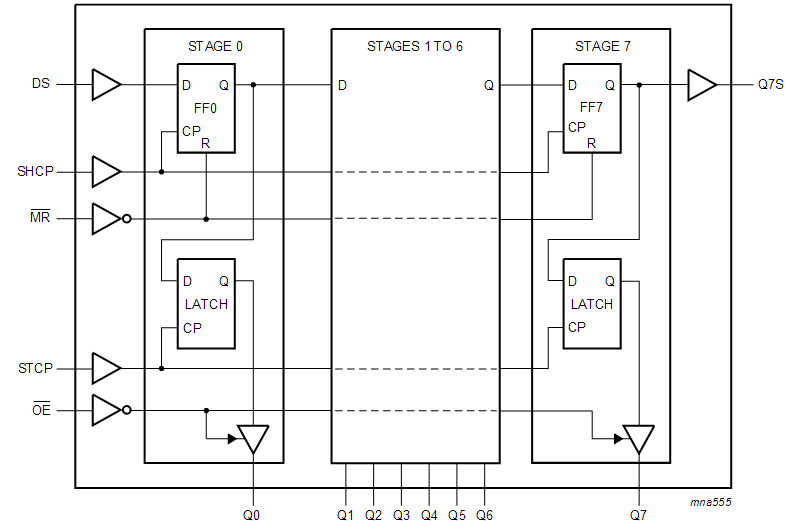


Рисунок 2 – Логическая схема регистра 74HC595

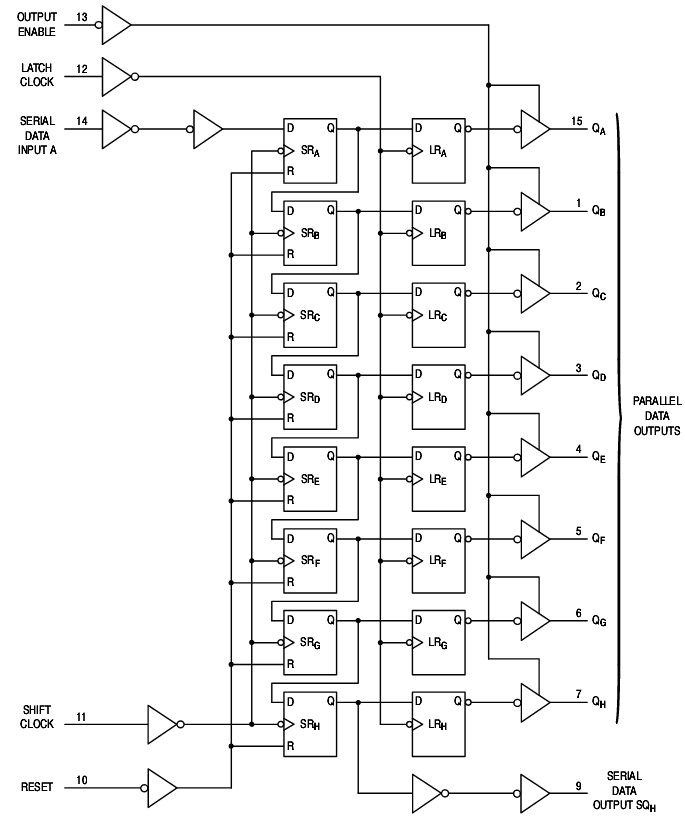


Рисунок 3 – Развёрнутая логическая схема регистра 74HC595

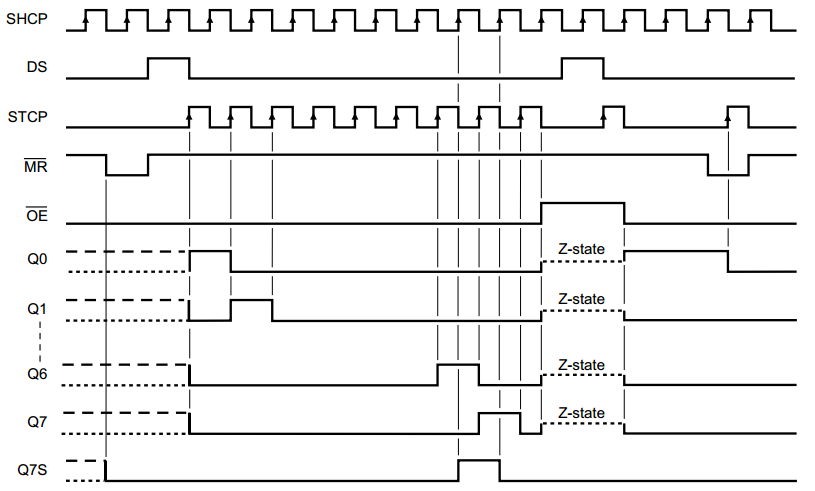


Рисунок 4 – Временная диаграмма, показывающая движение логической единицы по всем выходам регистра 74HC595.

1. **Сдвиговый регистр 74HC165**

Микросхема 74HC165 — сдвиговый регистр, преобразующий параллельный входной сигнал в последовательный выходной. Она позволяет увеличивать количество цифровых входов микроконтроллера.

Чип преобразовывает входящий параллельный сигнал на 8 пинах (Dx) в выходной последовательный сигнал на одном пине (Q7). Передача синхронна: для такта используется дополнительный пин (CP). Также отдельным пином управляется регистр данных (PL), что позволяет «загружать» параллельный сигнал для последовательного считывания с 8 выходов единовременно.

Вход P͞L управляет занесением состояний входов в триггеры — так называемой параллельной загрузкой: при наличии на нём логического нуля состояние входов регистра будет подано на S-входы триггеров. Чтобы отключить триггеры от входов регистра и иметь возможность читать из регистра, нужно установить P͞L в логическую единицу. Тактирование происходит при переходе CP из 0 в 1 при условии, что P͞L = 1 (параллельная загрузка отключена), а C͞E = 0 (тактирование включено). При каждом «тике» CP каждый триггер проталкивает бит в следующий триггер, захватывая бит со своего 1D-входа. Так как к 1D-входу первого триггера подключен вход SER, то подаваемые на него биты проталкиваются в регистр, позволяя соединять регистры в цепочки. В конечном итоге проталкиваемые биты достигают выхода последнего регистра, где подаются сразу на два вывода — на Q7 и, через инвертор, на Q͞7.

Таким образом из трёх пинов микроконтроллера можно получить 8 цифровых входов. Из регистров 74HC165 можно делать каскады, подключая один за другим, и из всё тех же 3 входящих линий получать 16, 24, 32 и т.д. цифровых входов.

Таблица 2 – Распиновка входов/выходов регистра 74HC165

|  |  |  |  |
| --- | --- | --- | --- |
|  | Пин 1 | P̅͞L | Защёлка SS(SPI) |
| Пин 2 | CP | Тактовый вход SCLK (SPI) |
| Пины  11-14,  3-6 | D0-D7 | Входы, состояние которых считывается в регистр |
| Пин 7 | Q̅͞7 | Инверсный вывод, на котором идут биты с QH, но инвертированные |
| Пин 8 | GND | Земля |
| Пин 9 | Q7 | Последовательный вывод MISO (SPI) |
| Пин 10 | DS | Последовательный ввод; к нему можно подсоединить вывод QH второго регистра, получив каскадное подключение |
| Пин 15 | C̅͞E | Инвертированный Clock Enable; когда на нём 1 – тактирование выключено |
| Пин 16 | Vcc | Питание |

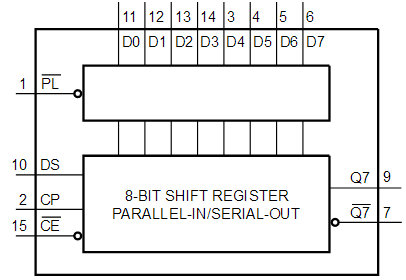


Рисунок 5 – Функциональная схема регистра 74HC165

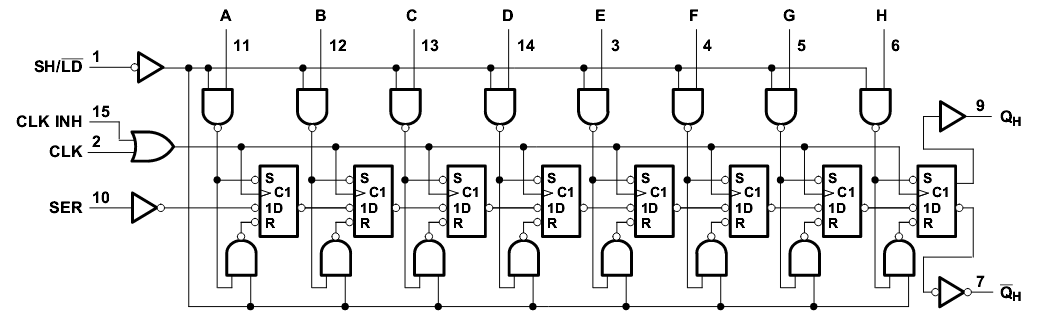


Рисунок 6 – Логическая схема регистра 74HC165

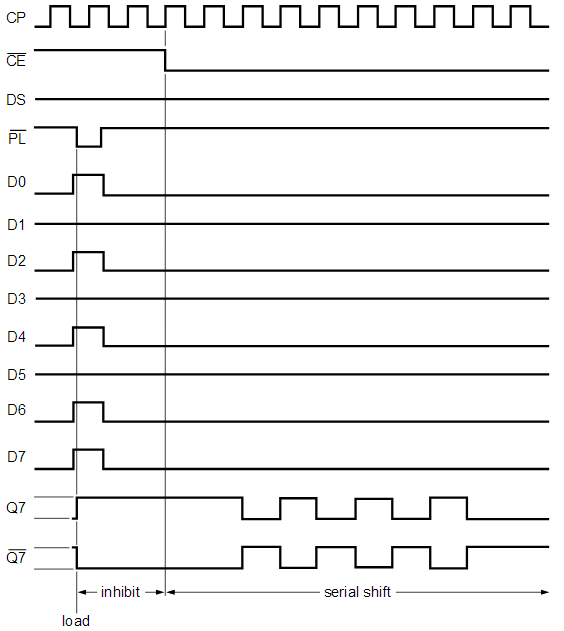


Рисунок 7 – Временная диаграмма работы регистра 74HC165

1. **SPI и Arduino**

SPI (Serial Peripheral Interface), или последовательный периферийный интерфейс, был разработан компанией Motorola для организации быстрого и простого в реализации обмена данными между компонентами системы — микроконтроллерами и периферийными устройствами. На шине может быть одно ведущее устройство (master) и несколько ведомых (slave). Интерфейс использует 4 линии для обмена данными:

* SCLK — Serial Clock: тактовые импульсы, генерируемые ведущим устройством для синхронизации процесса передачи данных.  
  Другие обозначения: SCK, CLK  
  Arduino: пин 13
* MOSI — Master Output, Slave Input: линия для передачи данных от ведущего к ведомому.  
  Другие обозначения: SDI, DI, SI  
  Arduino: пин 11
* MISO — Master Input, Slave Output: линия для передачи данных от ведомого к ведущему.  
  Другие обозначения: SDO, DO, SO  
  Arduino: пин 12
* SS — Slave Select: предназначена для активизации ведущим устройством того или иного периферийного устройства.  
  Другие обозначения: nCS, CS, CSB, CSN, nSS, STE  
  Arduino: по умолчанию пин 10

Линия SS обычно для каждого ведомого своя, но некоторых ведомых возможно подключить к одной SS — такой способ используется для каскадного подключения устройств. SPI является полнодуплексной шиной — данные передаются одновременно в обе стороны. Благодаря простоте алгоритма передачи SPI очень широко распространён.

Стандартный алгоритм работы SPI таков:

1. Ведущий устанавливает низкий уровень на той линии SS, к которой подключен нужный ведомый, инициируя цикл связи.
2. Ведущий задаёт такт уровнем на SCLK и, одновременно с каждым тактовым сигналом на SCLK, выставляет нужный уровень на MOSI, передавая ведомому по биту за такт.
3. Ведомый на каждый тактовый сигнал SCLK выставляет нужный уровень на MISO, передавая ведущему по биту за такт.
4. Для завершения передачи ведущий устанавливает высокий уровень на SS.

Для выполнения лабораторной работы будет использоваться упрощённая версия интерфейса.

**4. Семисегментный индикатор**

Светодиодный семисегментный индикатор представляет собой группу светодиодов, расположенных в определенном порядке и объединенных конструктивно. Светодиодные контакты промаркированы метками от A до G (и дополнительно DP – для отображения десятичной точки), и один общий вывод (рис.9), который определяет тип подключения индикатора (схема с общим анодом ОА, или общим катодом ОК). Зажигая одновременно несколько светодиодов, можно формировать на индикаторе символы цифр и некоторых букв. Регулировать яркость можно, подключив общий вывод индикатора к источнику смещающего напряжения 0..VDD, рассчитанного на втекающий ток, например к эмиттерному повторителю на транзисторе структуры p-n-p. Увеличивая смещение, будем уменьшать яркость свечения.

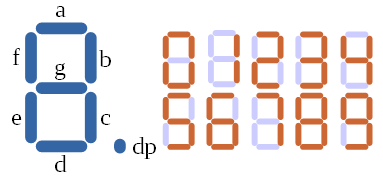


Рисунок 8 – Расположение сегментов индикатора

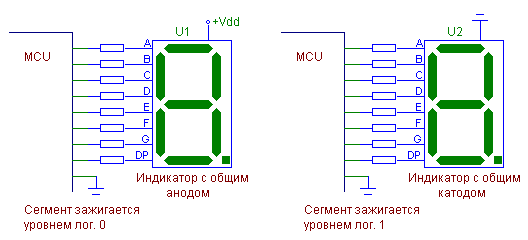


Рисунок 9 – Схема подключения индикаторов с общим анодом и общим катодом

Чтобы зажегся сегмент нужно подать напряжение между общей точкой и соответствующим выводом. Таким образом, если подать напряжение между выводами сегментов A, B и C и общей точкой, на индикаторе загорится "7". Сегменты, поскольку они являются отдельными светодиодами, следует подключать через токоограничивающие резисторы.

Светодиодный индикатор может быть одноразрядным (как на рисунке выше) и многоразрядным. У многоразрядных дисплеев есть восемь общих для всех разрядов выводов для определения индикации (A-G, DP – назовем их шиной управления сегментами), и несколько (по количеству разрядов) выводов для выбора разряда (шина адреса). Выводы для выбора разряда являются общими катодами или анодами.

Также у дисплея могут быть отдельные выводы A-G и DP для каждого разряда – или дисплей составляется из отдельных индикаторов. В таком случае вывод данных на него осуществляется по схеме, описанной выше. Проблема в том, что если каждым разрядом управлять индивидуально, то с увеличением их количества, число выводов микроконтроллера и количество проводников для подключения индикатора будет очень велико. Для управления N разрядами требуется 8\*N управляющих линий и 1 общий провод.

**Динамическая индикация** позволяет решить проблему, во много раз сократив требуемое для подключения количество выводов. Идея динамической индикации состоит в том, что информация отображается не во всех разрядах индикатора сразу, а поочерёдно, в каждый момент времени только в одном разряде. В связи с тем, что зрение инерционно, необязательно чтобы все элементы изображения светились непрерывно и одновременно. Если с достаточно высокой частотой последовательно переключаться от отображения одного разряда к следующему, а когда будет достигнут последний разряд индикатора, снова переходить к отображению первого и т.д., то глазом это будет восприниматься так, как если бы каждый разряд отображал информацию статично.

Так как требуется, чтобы в каждый момент времени работал только один разряд индикатора, то количество выводов можно существенно уменьшить: выводы одноимённых сегментов всех разрядов соединяются вместе, образуя общую шину для управления сегментами. Включение нужного разряда производится с помощью вывода общего анода (или катода, в зависимости от варианта исполнения индикатора) этого разряда.

Как правило, индикаторы, содержащие несколько разрядов, выпускаются именно в расчёте на динамическую индикацию и все необходимые соединения выполнены внутри устройства. N-разрядный индикатор в этом случае имеет 8 выводов для управления сегментами и N выводов для управления включением разрядов (общий анод или катод разряда). Всего требуется 8+N выводов, что намного лучше, чем 8\*N+1 при статической индикации.

В индикаторах, разряды которых выполнены по схеме с общим катодом, сегменты зажигаются высоким уровнем на выводах управления сегментами, а разряд включается низким уровнем на соответствующем выводе. В индикаторах с разрядами по схеме с общим анодом, наоборот, сегменты зажигаются низким уровнем, а разряд включается высоким уровнем на выводе.

Как уже было сказано, динамическая индикация предполагает последовательное поразрядное отображение информации с большой частотой переключения. Для этого в цикле выполняются следующие действия:

1. Гасятся все разряды индикатора – для предотвращения появления артефактов на выводимом изображении при смене состояния шины управления сегментами; если используется схема с общими катодами, для этого на общие катоды всех разрядов подаётся высокий уровень (лог. 1); в схеме с общими анодами, на аноды подаётся лог. 0.
2. На шину управления сегментами выдаются сигналы для отображения символа в соответствующем разряде.
3. Зажигается очередной разряд.

Либо можно погасить сегменты с помощью шины управления сегментами, переключиться на очередной разряд и выставить на шине сегментов индикатора сигналы для формирования символа. Возможны различные варианты, так что можно выбрать тот, который в данной ситуации проще реализовать.

Затем делается пауза, в течении которой происходит отображение информации на текущем разряде, после чего процесс повторяется. В результате происходит последовательное отображение от первого разряда до последнего, после чего вновь возвращаемся к первому и т.д.

Если разряды переключаются с частотой f, то время отображения одного разряда составит максимум 1/f. Максимум - потому что время горения разряда может быть и меньше периода переключения. Мы можем изменять время горения от 0 до 1/f, и тем самым регулировать яркость разряда за счёт эффекта от широтно-импульсной модуляции.

Количество выходов микроконтроллера ограничено, поэтому занимать 7 из них для вывода всего одного разряда накладно. Для того, чтобы снизить количество занятых выходов до одного, используют 7-сегментный драйвер или выходной сдвиговый регистр. Кроме того существуют индикаторы со встроенным сдвиговым регистром, но их стоимость выше и на практике они применяются достаточно редко.

Для выполнения работы используем четырёхразрядный индикатор F3461BH с общим анодом.

Характеристики:

– Четырёхразрядный семисегментный индикатор с общим плюсом.

– Цвет свечения: красный.

– Размер дисплея: 30х19х7.8 мм.

– Высота символа: 10 мм.

– Падение напряжения на сегменте: 1.8-2 В.

– Ток через сегмент: 20 мА.

– Падение напряжения на точке: 1.8-2 В.

– Ток через точку: 20 мА.

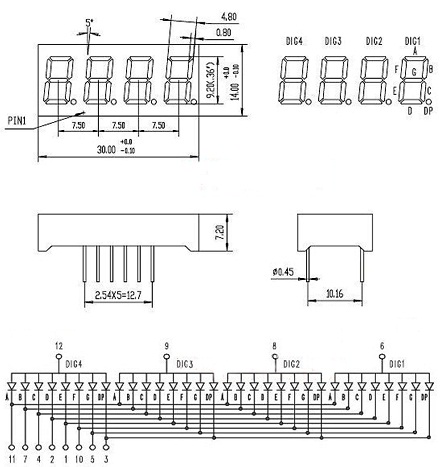


Рисунок 10 – Четырёхразрядный индикатор F3461BH с общим анодом

**Ход работы**

**Подключение семисегментного индикатора к микроконтроллеру Arduino через выходной сдвиговый регистр.**

Рассчитаем номинал резистора для каждого сегмента индикатора.

По закону Ома сопротивление резистора , где:

I – ток через сегмент индикатора,

V – напряжение через резистор. В нашем случае , где

VS – напряжение питания (напряжение на цифровом выводе Arduino – 5 В),

VL – падение напряжения на сегменте индикатора.

Таким образом .

Подключим семисегментный индикатор к микроконтроллеру через сдвиговый регистр по схеме на рисунке 11.

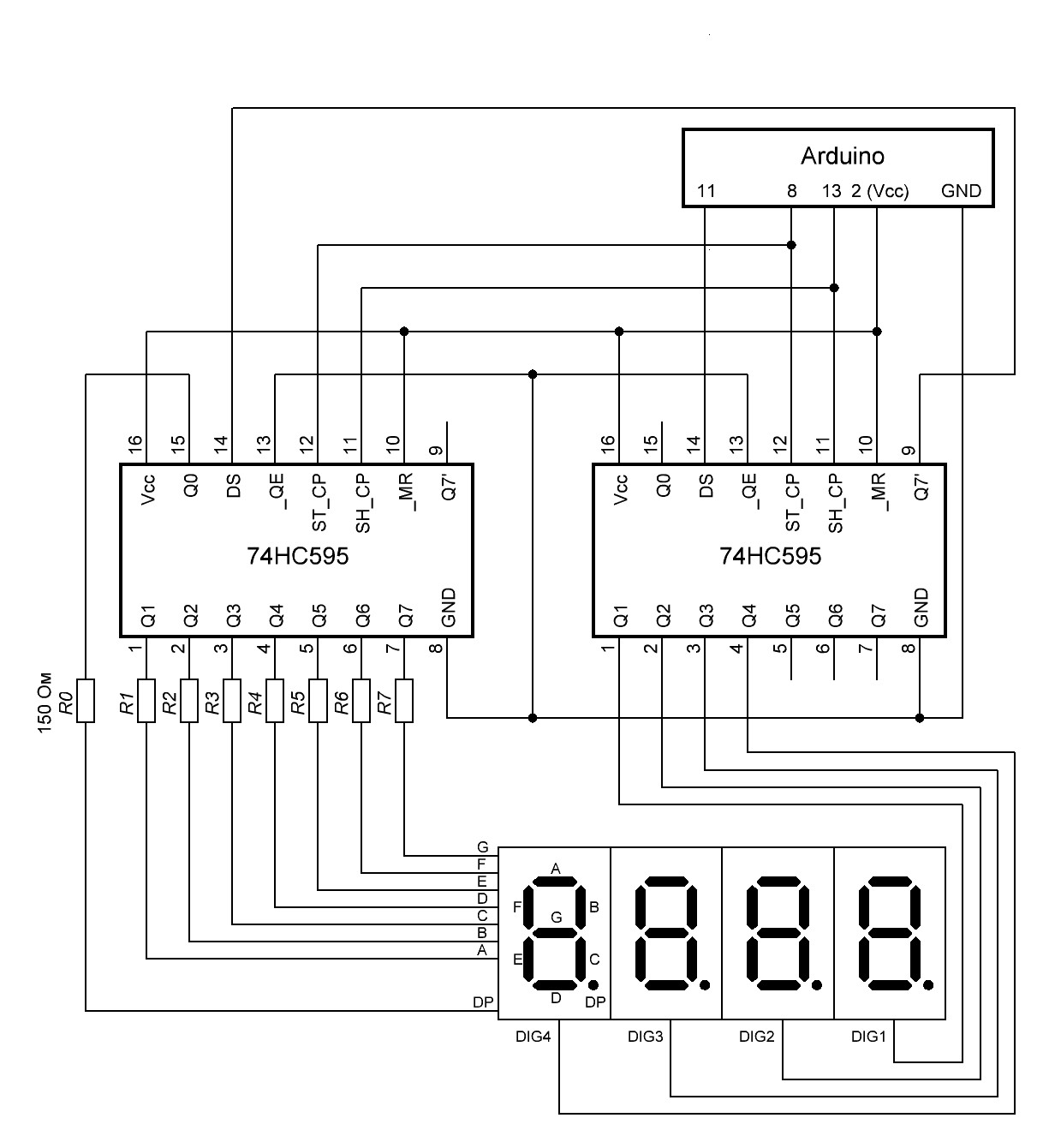


Рисунок 11 – Схема подключения семисегментного индикатора к МК через сдвиговый регистр

Напишем программу для Arduino.

Инициализируем глобальные переменные для нашей программы. Сначала определим назначение портов Arduino:

// настройка линии подачи напряжения

int POWR = 2; // питание +5В

// настройка пинов Arduino для выходного регистра

int LATCH\_PIN\_O = 8; // к 12 пину регистра - вход «защёлкивающий» данные

int DATA\_PIN\_O = 11; // к 14 пину регистра - вход данных

int CLOCK\_PIN\_O = 13; // к 11 пину регистра - вход для тактовых импульсов

С контроллера на регистр данные будут поступать по 2 байта. Значения первого байта будут подаваться на катоды и определять *символ*, который необходимо отобразить. Значения второго байта будут подаваться на аноды и определять на каком *разряде* будет отображён символ.

Создадим переменные, в которых будет содержаться код для отображения каждого отдельного сегмента индикатора. Назовём их так, чтобы в дальнейшем было легко определить какой сегмент используется. Используя такие переменные будет проще составить таблицу символов. Комбинация этих переменных будет представлять первый байт, подаваемый на катоды.

Так как сигналы для символа подаются на катоды, они должны быть низкого уровня (0В). Поэтому сначала, побитово сдвигая единицу влево, мы определяем пин на который нужно подать сигнал (сдвигаем 1 из нулевого разряда байта в соответствующий пину к которому подключен сегмент), затем инвертируем весь байт, чтобы получить 0 на месте 1.

// значения байта сигналов для соотв. сегментов индикатора

byte S\_TOP = ~(1 << 1); // A сегменты индикатора

byte S\_RTOP = ~(1 << 2); // B

byte S\_RBOT = ~(1 << 3); // C

byte S\_BOT = ~(1 << 4); // D

byte S\_LBOT = ~(1 << 5); // E

byte S\_LTOP = ~(1 << 6); // F

byte S\_MID = ~(1 << 7); // G

byte S\_DP = ~(1 << 0); // DP

byte S\_EMPT = ~0; // пустой

Теперь создадим «таблицу» символов. Так как каждый сегмент у нас представлен отдельной переменной, есть возможность включать необходимые сегменты путём объединения нужных переменных с помощью операции логического «И». Логическое «И» используется потому, что оно позволяет объединить нули, заменив ими единицы в разрядах байтов (1&0=0). Используя эту операцию составим массив символов цифр от 0 до 9, точки, букв E и r, также зададим комбинацию, которая будет соответствовать «пустому» символу. Соберём массив таким образом чтобы символы цифр от 0 до 9 соответствовали таким же индексам массива, т.е. ledChars[0] = код для отображения 0 и т.д.

Каждый элемент (байт) массива ledChars будет соответствовать коду, необходимому для отображения определённого символа.

// массив значений байта сигналов для отображения символа

// массив получается путём объединения значений для отдельных сегментов

byte ledChars[14] = {

(byte) (S\_TOP & S\_BOT & S\_LTOP & S\_LBOT & S\_RTOP & S\_RBOT), // 0

(byte) (S\_RTOP & S\_RBOT), // 1

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LBOT & S\_RTOP), // 2

(byte) (S\_TOP & S\_MID & S\_BOT & S\_RTOP & S\_RBOT), // 3

(byte) (S\_MID & S\_LTOP & S\_RTOP & S\_RBOT), // 4

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_RBOT), // 5

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_RBOT), // 6

(byte) (S\_TOP & S\_RTOP & S\_RBOT), // 7

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_RTOP & S\_RBOT), // 8

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_RTOP & S\_RBOT), // 9

(byte) (S\_DP), // .

(byte) (S\_EMPT), // пусто

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_DP), // E

(byte) (S\_MID & S\_LBOT), // r

};

Создадим такой же массив кодов (значений, подаваемых на аноды) для выбора разряда индикатора в котором будет отображён символ. В каждом элементе массива будет храниться значение кода, которое соответствует определённому разряду индикатора. Сигналы подаваемые на аноды должны быть высокого уровня (+5В), поэтому побитово сдвинем единицу влево, чтобы отметить соответствующий разряд. Нумерация разрядов начинается с нуля: 0-3. Разряды индикатора подключены к регистру начиная с первого цифрового вывода регистра (т.е 0 разряд к 1 выводу, 1 ко 2 и т.д.), а не с нулевого, поэтому поместим сигнал для первого разряда в первый бит байта, а не в нулевой.

// массив значений для байта управления выбором разряда

byte ledDigs [4] = { // разряды индикатора

1 << 1, // 0

1 << 2, // 1

1 << 3, // 2

1 << 4, // 3

};

На этом этапе мы имеем все данные, необходимые для программирования работы устройства. Настроим порты Arduino.

void setup() {

// настройка портов на выход

pinMode(LATCH\_PIN\_O, OUTPUT);

pinMode(DATA\_PIN\_O, OUTPUT);

pinMode(CLOCK\_PIN\_O, OUTPUT);

// настройка питания +5В на 6 пине Arduino

pinMode(POWR, OUTPUT);

digitalWrite(POWR, HIGH);

}

Теперь составим функции для работы с имеющимися данными. Для начала напишем функцию ledDisp(), которая будет отображать произвольный символ. Она будет принимать любые байты кодов для катодов и анодов и отправлять их последовательно на регистр. Сначала будет отправлен байт значений для катодов (символы), который «протолкнётся» во второй регистр, а затем байт значений для анодов (разряды). Эти байты, отправленные на регистр последовательно, поочерёдно выставятся параллельно на соответствующих его пинах.

Для передачи данных будем использовать встроенную функцию shiftOut(). Она представляет собой программно реализованный SPI для отправки данных.

Функция shiftOut() выводит байт информации на порт вход/выхода последовательно (побитно). Вывод может осуществляться как с первого (левого), так и с последнего (правого) бита. Каждый бит последовательно подается на заданный порт, после чего подается сигнал на синхронизирующий порт вход/выход, информируя о доступности к считыванию бита. Такой способ передачи данных называют последовательный протокол с синхронизацией.

|  |
| --- |
| Синтаксис:shiftOut(dataPin, clockPin, bitOrder, value). |

Параметры:

* dataPin: номер порта вход/выхода, на который выводятся биты (int);
* clockPin: номер порта по которому производится синхронизация (int);
* bitOrder: используемая последовательность вывода бит. MSBFIRST (Most Significant Bit First) — слева или LSBFIRST (Least Significant Bit First) — справа;
* value: значение (байт) для вывода (byte).

Возвращаемое значение: нет.

Текущая реализация функции shiftOut() может выводить только один байт (8 бит) информации, поэтому необходимо произвести несколько действий чтобы вывести значения больше 255.

// отображение произвольного символа

void ledDisp(byte symbCode, byte posCode) { // катоды, аноды

digitalWrite(LATCH\_PIN\_O, LOW);

shiftOut(DATA\_PIN\_O, CLOCK\_PIN\_O, MSBFIRST, symbCode); // байт для катодов

shiftOut(DATA\_PIN\_O, CLOCK\_PIN\_O, MSBFIRST, posCode); // байт для анодов

digitalWrite(LATCH\_PIN\_O, HIGH); // "защёлкиваем" данные

}

Теперь напишем функцию ledDispChar() для отображения символа из массива ledChars, определённого ранее. Она будет принимать значение индекса символа в массиве и номер разряда, на котором следует его отобразить. Данная функция будет использовать функцию ledDisp(), таким образом программа будет разбита на несколько простых частей, которые будут отвечать за конкретные действия и в дальнейшем объединятся в более сложные.

// отображение символа в соотв. с массивом

// (индекс символа в массиве, разряд на котором отображать)

void ledDispChar(int num, int dig) {

ledDisp(ledChars[num], ledDigs[dig]);

}

Создадим функцию ledDispPoint(), которая будет зажигать точку в указанном разряде. Для её работы будет использоваться функция ledDispChar().

// отображение точки в указанном разряде

void ledDispPoint(int dig) { // разряд для отображения точки

ledDisp(ledChars[10], ledDigs[dig]);

}

Напишем функцию ledDispErr(), которая будет выводить на индикатор сообщение об ошибке «Err». Она пригодится для обработки задач, которые по каким-либо причинам не могут быть правильно выполнены.

// отображение ошибки

void ledDispErr() {

ledDispChar(13, 1); // символ E

ledDispChar(13, 2); // символ r

ledDispChar(12, 3); // символ r

}

Все предыдущие функции предназначены для статической индикации, т.е. сами по себе, в таком виде они позволяют отобразить символ только в одном разряде индикатора.

Следующая функция – ledDisplayNumSet() – реализует динамическую индикацию, т.е. даёт возможность отображать разные символы на всех разрядах индикатора. Функция принимает целое число, которое необходимо отобразить, затем оно разбивается по разрядам. Каждый разряд этого числа кладётся в массив, который затем будет поэлементно отображён на соответствующих разрядах индикатора с помощью функции ledDispChar(). В один момент времени будет загораться только один разряд.

// отображение 4-х значного числа (динамическая индикация)

void ledDispNumSet(int x) { // 4-х значное число

if (x == 0) { // если число 0 - сразу выводим 0

ledDispChar(0, 0);

return;

}

if (x > 9999) { // если число > 4 знаков - выводим ошибку

ledDispErr();

return;

}

int nums[4] = {11, 11, 11, 11}; // массив пустых символов

int i = 0;

while (x > 0) { // разбиение по разрядам в массив nums

nums[i] = x % 10;

x = x / 10;

i++;

}

for (int i = 0; i < 4; i++) { // вывод получившегося массива nums на индикатор

ledDispChar(nums[i], i);

}

}

Добавим возможность отображать десятичную точку. Это будет выполнять функция ledDisplayNumSetWithPoint(). В данной реализации она будет как бы объединять в себе функции ledDisplayNumSet() и ledDispPoint().

// отображение числа с точкой в указанном разряде

// (число для отображения, разряд для точки)

void ledDispNumSetWithPoint(int num, int dig) {

if (dig < 4 && dig >= 0) { // соответствует ли номер существующему разряду

ledDispNumSet(num); // отображаем число

ledDispPoint(dig); // отображаем точку

} else { // если номер разряда отрицательный или превышает их кол-во

ledDispNumSet(num); // отображаем только число

}

}

Описанных выше функций достаточно для работы с индикатором. Используем их. Например, отобразим число «1234» с точкой в разряде десятков.

void loop() { // бесконечный цикл

ledDispNumSetWithPoint(1234, 1); // (число 0-9999, разряд для точки)

} // в номере разряда указываем 1, т.к. нумерация начинается с нуля

Напишем, на основе предыдущих, ещё одну функцию dispInc() для поочерёдного отображения на индикаторе всех допустимых для него чисел.

// Отображение инкрементирования

void dispInc(int t) { // период инкрементирования

for (int i = 0; i < 10000; i++) {

for (int j = 0; j < t; j++) {

ledDispNumSet(i);

}

}

}

Используем её:

void loop() { // бесконечный цикл

dispInc(120);

}

После загрузки данного кода в Arduino можно увидеть как поочерёдно выводятся на индикатор все числа от 0 до 9999.

На индикаторе можно отображать любую другую информацию, вызывая в бесконечном цикле другие описанные функции, а также составлять из них более сложные.

Таким образом, для подключения четырёхразрядного семисегментного индикатора используется всего 3 цифровых выхода микроконтроллера. На регистре, данные с которого подаются на разряды индикатора, остались незанятыми ещё 4 пина для вывода. Их можно использовать для подключения ещё 4 разрядов индикатора, например объединив 2 4-разрядных индикатора в один 8-разрядный. Также, при необходимости можно использовать ещё больше сдвиговых регистров, соединяя их каскадно.

**Подключение тактовых кнопок к микроконтроллеру Arduino через входной сдвиговый регистр.**

Подключим 4 тактовые кнопки к микроконтроллеру через входной сдвиговый регистр по схеме на рисунке 12. К первым разрядам регистра (D0-D3) подключим кнопки, оставшиеся разряды (D4-D7) притянем к земле, чтобы на них всегда подавался логический ноль. Кнопки подключим по схеме со стягивающим резистором (резисторы возьмём номиналом 10 кОм) для того, чтобы нажатие на кнопку индицировалось подачей высокого уровня (+5В) на порты регистра.

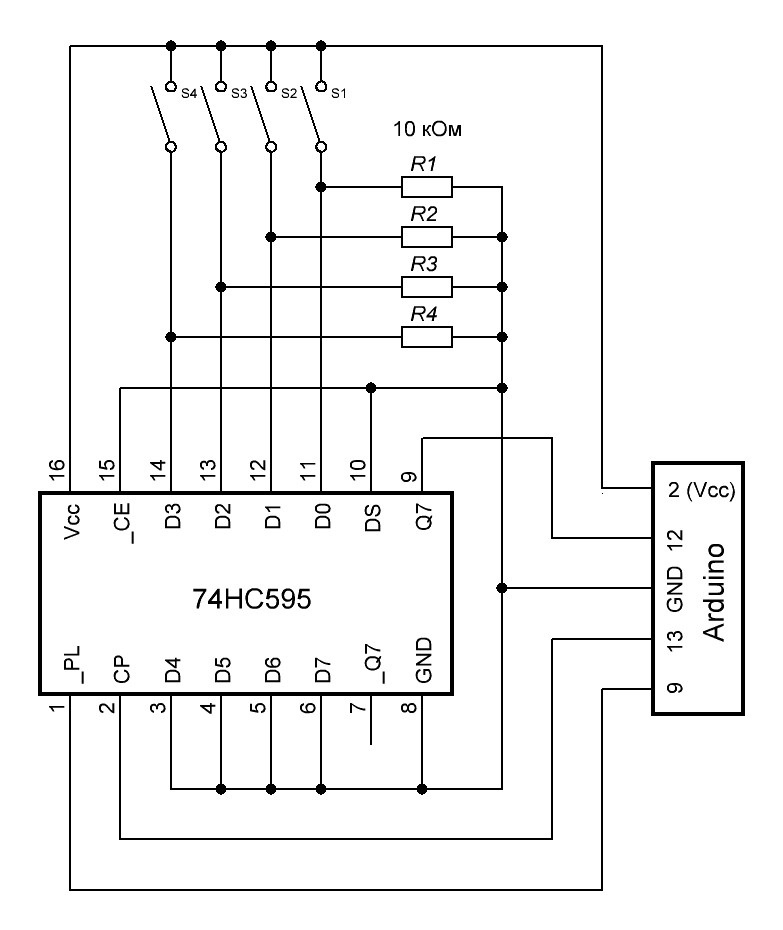


Рисунок 12 – Схема подключения тактовых кнопок к МК через входной сдвиговый регистр

Сдвиговый регистр будет преобразовывать параллельный сигнал с кнопок в последовательный, формируя байт и отправляя его в контроллер. Контроллер будет получать этот байт и при изменении его значений, (т.е. состояний кнопок) выводить байт в терминал.

Напишем программу для Arduino. Инициализируем глобальные переменные для нашей программы. Определим назначение портов Arduino.

// настройка линии подачи напряжения

int POWR = 2; // питание +5В

// настройка пинов Arduino для входного регистра

int LATCH\_PIN\_I = 9; // к 1 пину регистра - вход «защёлкивающий» данные

int DATA\_PIN\_I = 12; // к 9 пину регистра - вход данных

int CLOCK\_PIN\_I = 13; // к 2 пину регистра - вход для тактовых импульсов

Настроим порты Arduino соответствующим образом, включим «защёлку» регистра и инициируем последовательное соединение для обмена данными с терминалом на скорости 9600 бод.

void setup() {

// настройка портов на выход

pinMode(LATCH\_PIN\_I, OUTPUT);

pinMode(CLOCK\_PIN\_I, OUTPUT);

// настройка портов на вход

pinMode(DATA\_PIN\_I, INPUT);

// включаем защёлку

digitalWrite(LATCH\_PIN\_I, HIGH);

// открываем последовательный порт, устанавливаем скорость 9600 бод

Serial.begin(9600);

}

Теперь напишем функцию readByte(), которая будет возвращать переменную с байтом, полученным с входного регистра. Для чтения байта будем использовать встроенную функцию shiftIn(). Она представляет собой программно реализованный SPI для получения данных.

Функция shiftIn() осуществляет побитовый сдвиг и считывание байта данных, начиная с самого старшего (левого) или младшего (правого) значащего бита. Процесс считывания каждого бита заключается в следующем: тактовый вывод переводится в высокий уровень, считывается очередной бит из линии данных, после чего тактовый вывод сбрасывается в низкий уровень.

Синтаксис: byte incoming = shiftIn(dataPin, clockPin, bitOrder).

Параметры:

* dataPin: вывод, с которого будет считываться каждый бит (int);
* clockPin: тактовый вывод, который будет переключаться при считывании с dataPin (int);
* bitOrder: порядок, в котором будут сдвигаться и считываться биты; может принимать значения MSBFIRST или LSBFIRST. (Most Significant Bit First - старший значащий бит первым, или Least Significant Bit - младший значащий бит первым).

Возвращаемые значения: считанное значение (byte).

*Внимание!* В данной функции существует баг: функция теряет один бит, если в момент её вызова clockPin установлен в логический ноль. Для того, чтобы избежать потери бита: перед защёлкиванием данных и вызовом shiftIn() установим в clockPin логическую единицу.

Установим clockPin в HIGH, чтобы функция сработала правильно. После чего выставим на защёлке сначала низкий, потом высокий уровни. Сдвиговый регистр запомнит уровни сигналов на входах и сможет их отдать бит за битом. После чего применим встроенную функцию shiftIn() непосредственно для получения байта с регистра. Завершим функцию возвратом полученного значения.

byte readByte() {

digitalWrite(CLOCK\_PIN\_I,HIGH); // необходимо для корректной работы shiftIn()

digitalWrite(LATCH\_PIN\_I, LOW); // "дёргаем" защёлку для запоминания

digitalWrite(LATCH\_PIN\_I, HIGH); // уровней сигналов на входах

byte retByte = shiftIn(DATA\_PIN\_I,CLOCK\_PIN\_I,MSBFIRST); // читаем байт

return retByte;

}

Применим функцию readByte() в бесконечном цикле для индикации нажатия кнопок. Инициализируем переменную lastDataByte, типа byte, в которой будут хранится предыдущие состояния кнопок. Ключевое слово static предотвращает переинициализацию переменной внутри цикла. Оно препятствует повторной инициализации переменной. Сначала считаем состояния кнопок с регистра в переменную dataByte. Затем, если они изменились, выведем их в терминал и запомним для сравнения при считывании значений с регистра на следующей итерации цикла.

void loop() { // бесконечный цикл

static byte lastDataByte = 0; // предыдущие состояния входов регистра

byte dataByte = readByte(); // читаем текущие состояние входов регистра

if (dataByte != lastDataByte) { // если состояние любого входа изменилось

Serial.println(dataByte,BIN); // отправляем состояния входов в терминал

lastDataByte = dataByte; // запоминаем текущие состояния

}

}

При нажатии или отпускании любой кнопки, в терминале будут отображены состояния всех четырёх кнопок, подключенных к входам регистра. Например, если зажать самую правую из них, в терминале появится «0001». Как только она будет отпущена – отобразится «0000».

Таким образом, для подключения 4 тактовых кнопок было задействовано 3 цифровых пина микроконтроллера. На регистре осталось ещё 4 свободных пина, на них также можно подключить дополнительные кнопки и что-нибудь другое. Соединяя регистры каскадом можно увеличивать количество входов.

**Управление показаниями семисегментного индикатора через тактовые кнопки с подключением. Подключение индикатора и кнопок осуществляется через сдвиговые регистры.**

Объединим предыдущие схемы подключения в одну и реализуем управление семисегментным индикатором тактовыми кнопками. Нажатие на каждую из кнопок будет инкрементировать соответствующий ей разряд четырёхзначного числа, отображаемого на индикаторе. Тактовую линию можно сделать общей для обоих регистров, пользуясь ей по очереди.

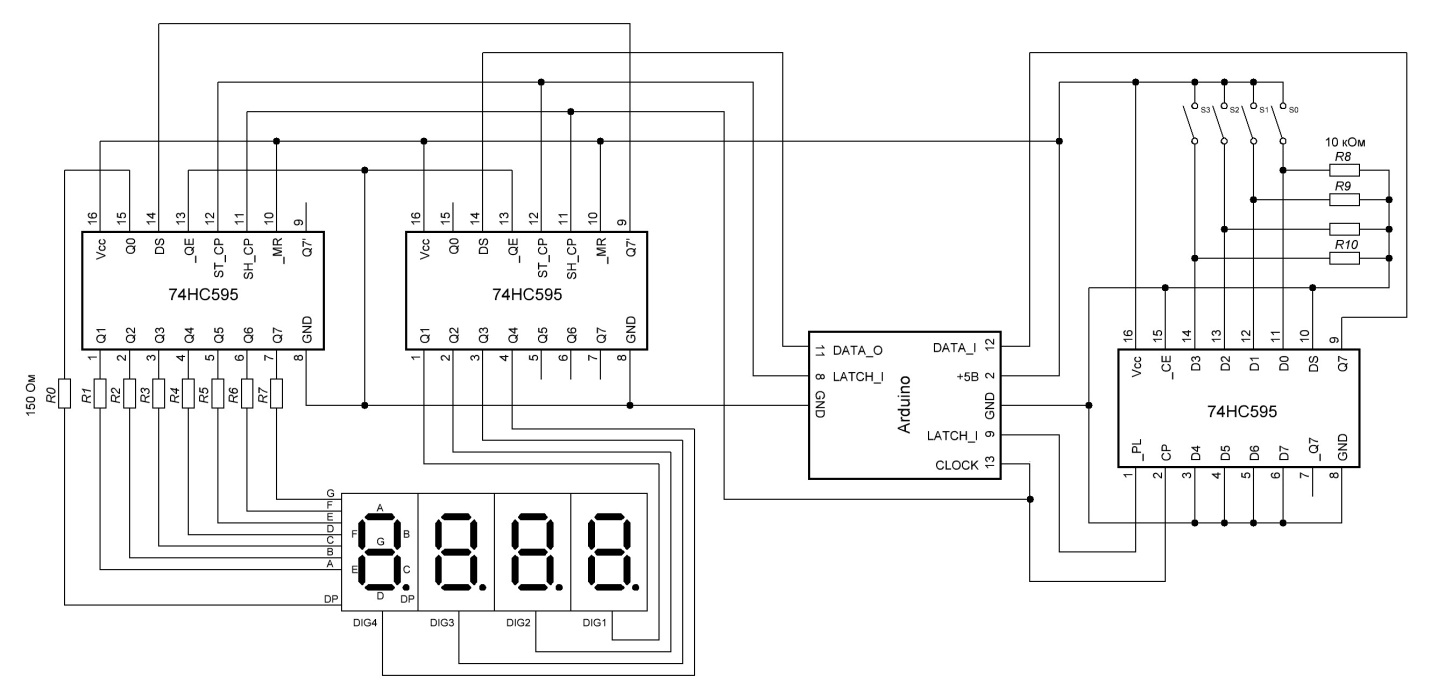


Рисунок 13 – Схема подключения семисегментного индикатора и тактовых кнопок к контроллеру Arduino

Напишем программу для Arduino.

Объединим коды двух предыдущих программ в одну. Скопируем в один файл из двух предыдущих всё, кроме бесконечного цикла void loop() {}. В нём, пользуясь функциями, написанными для обработки получаемых и отправляемых данных на регистры, мы напишем программу которая будет выполнять поставленную задачу. Полный код данной программы представлен в приложении А.

Инициализируем переменные для работы программы. Создадим переменную lastDataByte типа byte. В ней будем хранить предыдущие состояния кнопок (входов входного регистра).

В переменной dispNum будем хранить отображаемое на индикаторе число. Присвоим ей значение 0, чтобы при первом запуске программы отобразить его на дисплее.

Переменная addNum является слагаемым, добавляемым к отображаемому числу dispNum при каждом нажатии на одну из кнопок. Нажатие на самую правую кнопку прибавит 1 к dispNum на следующую за ней 10, далее 100 и 1000. Присвоим ей значение 0, т.к. при каждом изменении состояния кнопок оно будет разным, а состояния считываются каждую итерацию.

После этого считаем состояния кнопок в переменную dataByte. Если эти состояния изменились по сравнению с предыдущими (dataByte != lastDataByte), значит произошло нажатие либо отпускание одной из кнопок. Обработаем его.

Нам нужно реагировать только на изменение состояний при нажатии кнопок. Изменение состояний при отпускании нас не интересуют. Значение байта при нажатии на кнопку увеличивается (на множитель разряда байта, соответствующего кнопке), а при отпускании кнопки уменьшается (на него же). Таким образом, если введём условие (dataByte > lastDataByte), оно будет выполнятся только при нажатии на кнопку.

Теперь, когда состояние кнопок изменилось и мы определили, что они были нажаты, а не отпущены, нужно узнать какая кнопка была нажата последней, поскольку несколько кнопок могли быть нажаты одна за другой и не отпущены. При нажатии на последнюю значения с предыдущих, нажатых ранее и зажатых вместе с ней, не должны учитываться. Для этого используем операцию «ИСКЛЮЧАЮЩЕЕ ИЛИ». После применения её к считанному байту лишь один его бит будет равен единице. Он и соответствует кнопке, которая была нажата последней.

Далее нам нужно подобрать множитель разряда. Чтобы это сделать, узнаем в каком разряде стоит оставшийся бит. Возьмём единицу и будем побайтово сдвигать её влево до тех пор, пока она не будет на месте оставшегося бита в считанном байте. Последнее можно проверить применив логическое «И» к обработанному считанному байту и сдвигаемой единице. Единица будет на месте оставшегося бита, когда результат операции «И» будет равен 1, а не 0. Параллельно с этим, введём переменную m = 1, которая будет соответствовать десятичному множителю разряда. Каждый раз, применяя логический сдвиг будем умножать содержимое переменной m на 10 и результат возвращать обратно в неё (m = m\*10). Когда результат логического «И» будет 1, присвоим переменной addNum значение множителя разряда.

После этих манипуляций запомним состояния входов входного регистра для сравнения с полученными на следующей итерации бесконечного цикла.

Наконец, добавим дополнительное слагаемое addNum к отображаемому числу dispNum и выведем его на индикатор. Если отображаемое число превысит 9999 – сбросим его в ноль.

void loop() { // бесконечный цикл

static byte lastDataByte = 0; // предыдущие состояния входов

static int dispNum = 0;

int addNum = 0; // добавляемое к отображаемому числу при нажатии на кнопку

byte dataByte = readByte(); // читаем состояния входного регистра

if (dataByte != lastDataByte){ // если состояние любого входа изменилось

if (dataByte > lastDataByte){ // реагируем только на нажатия

byte changedDataByte = dataByte ^ lastDataByte; // изменившиеся биты

// подбираем множитель, соответствующий изменившемуся биту

int i = 0; // счётчик разрядов в байте

int m = 1; // множитель разряда (1 10 100 1000 соотв.)

while(!addNum){ // пока не подберём множитель

if (changedDataByte & (1<<i)){ // если натыкаемся на ненулевой бит

addNum = m; // присваиваем множитель разряда

}

i++; // увеличивем счётчик разрядов

m = m\*10; // увеличиваем степень множителя 10^i

}

}

lastDataByte = dataByte; // запоминаем текущие состояния

}

dispNum = dispNum + addNum; // добавляем слагаемое к отображаемому числу

if (dispNum > 9999){ // если число превышает максимальное - сбрасываем его

dispNum = 0;

}

ledDispNumSet(dispNum); // отображаем число на индикаторе

}

Таким образом, мы подключили к контроллеру 4 тактовые кнопки и через них реализовали управление семисегментным индикатором, используя для передачи данных всего 5 пинов контроллера.

**Вывод:** в ходе выполнения лабораторной работы была освоена работа с семисегментным индикатором через выходные сдвиговые регистры, управляемые контроллером Arduino. Собрана схема подключения. Реализованы и использованы для отображения символов статическая и динамическая индикации. Освоено и реализовано подключение и считывание состояний тактовых кнопок к контроллеру Arduino через входные сдвиговые регистры. Изучены стандартные функции языка Wiring для последовательной передачи данных. Реализовано управление семисегментным индикатором через тактовые кнопки.

**ПРИЛОЖЕНИЕ A**

**Листинг программ**

**outReg.ino:**

// Отображение информации на 4-разрядном, 7-сегментном LED-индикаторе, общий анод,

// подключение через 2 сдвиговых регистра 74HC595N.

//---------------------------------------------------------------------------

// настройка линии подачи напряжения

int POWR = 2; // питание +5В

// настройка пинов Arduino для выходного регистра

int LATCH\_PIN\_O = 8; // к 12 пину регистра - вход «защёлкивающий» данные

int DATA\_PIN\_O = 11; // к 14 пину регистра - вход данных

int CLOCK\_PIN\_O = 13; // к 11 пину регистра - вход для тактовых импульсов

// значения байта сигналов для соотв. сегментов индикатора

byte S\_TOP = ~(1 << 1); // A сегменты индикатора

byte S\_RTOP = ~(1 << 2); // B

byte S\_RBOT = ~(1 << 3); // C

byte S\_BOT = ~(1 << 4); // D

byte S\_LBOT = ~(1 << 5); // E

byte S\_LTOP = ~(1 << 6); // F

byte S\_MID = ~(1 << 7); // G

byte S\_DP = ~(1 << 0); // DP

byte S\_EMPT = ~0; // пустой

// массив значений байта сигналов для отображения символа

// массив получается путём объединения значений для отдельных сегментов

byte ledChars[14] = {

(byte) (S\_TOP & S\_BOT & S\_LTOP & S\_LBOT & S\_RTOP & S\_RBOT), // 0

(byte) (S\_RTOP & S\_RBOT), // 1

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LBOT & S\_RTOP), // 2

(byte) (S\_TOP & S\_MID & S\_BOT & S\_RTOP & S\_RBOT), // 3

(byte) (S\_MID & S\_LTOP & S\_RTOP & S\_RBOT), // 4

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_RBOT), // 5

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_RBOT), // 6

(byte) (S\_TOP & S\_RTOP & S\_RBOT), // 7

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_RTOP & S\_RBOT), // 8

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_RTOP & S\_RBOT), // 9

(byte) (S\_DP), // .

(byte) (S\_EMPT), // пусто

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_DP), // E

(byte) (S\_MID & S\_LBOT), // r

};

// массив значений для байта управления выбором разряда

byte ledDigs [4] = { // разряды индикатора

1 << 1, // 0

1 << 2, // 1

1 << 3, // 2

1 << 4, // 3

};

void setup() {

// настройка портов на выход

pinMode(LATCH\_PIN\_O, OUTPUT);

pinMode(DATA\_PIN\_O, OUTPUT);

pinMode(CLOCK\_PIN\_O, OUTPUT);

// настройка питания +5В на 6 пине Arduino

pinMode(POWR, OUTPUT);

digitalWrite(POWR, HIGH);

}

void loop() { // бесконечный цикл

dispInc(200);

}

//---------------------------------------------------------------------------

// ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

// Отображение инкрементирования

void dispInc(int t) { // период инкрементирования

for (int i = 0; i < 10000; i++) {

for (int j = 0; j < t; j++) {

ledDispNumSet(i);

}

}

}

//---------------------------------------------------------------------------

// ИНДИКАТОР

// отображение произвольного символа

void ledDisp(byte symbCode, byte posCode) { // катоды, аноды

digitalWrite(LATCH\_PIN\_O, LOW);

shiftOut(DATA\_PIN\_O, CLOCK\_PIN\_O, MSBFIRST, symbCode); // байт для катодов

shiftOut(DATA\_PIN\_O, CLOCK\_PIN\_O, MSBFIRST, posCode); // байт для анодов

digitalWrite(LATCH\_PIN\_O, HIGH); // "защёлкиваем" данные

}

// отображение символа в соотв. с массивом

// (индекс символа в массиве, разряд на котором отображать)

void ledDispChar(int num, int dig) {

ledDisp(ledChars[num], ledDigs[dig]);

}

// отображение точки в указанном разряде

void ledDispPoint(int dig) { // разряд для отображения точки

ledDisp(ledChars[10], ledDigs[dig]);

}

// отображение ошибки

void ledDispErr() {

ledDispChar(13, 1); // символ E

ledDispChar(13, 2); // символ r

ledDispChar(12, 3); // символ r

}

// отображение 4-х значного числа (динамическая индикация)

void ledDispNumSet(int x) { // 4-х значное число

if (x == 0) { // если число 0 - сразу выводим 0

ledDispChar(0, 0);

return;

}

if (x > 9999) { // если число > 4 знаков - выводим ошибку

ledDispErr();

return;

}

int nums[4] = {11, 11, 11, 11}; // массив пустых символов

int i = 0;

while (x > 0) { // разбиение по разрядам в массив nums

nums[i] = x % 10;

x = x / 10;

i++;

}

for (int i = 0; i < 4; i++) { // вывод массива nums на индикатор

ledDispChar(nums[i], i);

}

}

// отображение числа с точкой в указанном разряде

// число для отображения, разряд для точки

void ledDispNumSetWithPoint(int num, int dig) {

if (dig < 4 && dig >= 0) { // проверяем существует ли разряд с номером dig

ledDispNumSet(num); // отображаем число

ledDispPoint(dig); // отображаем точку

} else { // если номер разряда отрицательный или превышает их кол-во

ledDispNumSet(num); // отображаем только число

}

}

**inReg.ino:**

// Отправка в терминал состояний 4 тактовых кнопок, подключенных к

// входному сдвиговому регистру 74HC165.

//---------------------------------------------------------------------------

// настройка линии подачи напряжения

int POWR = 2; // питание +5В

// настройка пинов Arduino для входного регистра

int LATCH\_PIN\_I = 9; // к 1 пину регистра - вход «защёлкивающий» данные

int DATA\_PIN\_I = 12; // к 9 пину регистра - вход данных

int CLOCK\_PIN\_I = 13; // к 2 пину регистра - вход для тактовых импульсов

void setup() {

// настройка портов на выход

pinMode(LATCH\_PIN\_I, OUTPUT);

pinMode(CLOCK\_PIN\_I, OUTPUT);

// настройка портов на вход

pinMode(DATA\_PIN\_I, INPUT);

// включаем защёлку

digitalWrite(LATCH\_PIN\_I, HIGH);

// открываем последовательный порт, устанавливаем скорость 9600 бод

Serial.begin(9600);

}

void loop() { // бесконечный цикл

static byte lastDataByte = 0; // предыдущие состояния входов регистра

byte dataByte = readByte(); // читаем текущие состояние входов регистра

if (dataByte != lastDataByte) { // если состояние любого входа изменилось

Serial.println(dataByte,BIN); // отправляем состояния входов в терминал

lastDataByte = dataByte; // запоминаем текущие состояния

}

}

//---------------------------------------------------------------------------

// СДВИГОВЫЙ РЕГИСТР

// чтение байта с входного регистра

byte readByte() {

digitalWrite(CLOCK\_PIN\_I,HIGH); // необходимо для работы shiftIn()

digitalWrite(LATCH\_PIN\_I, LOW); // "дёргаем" защёлку

digitalWrite(LATCH\_PIN\_I, HIGH);

byte retByte = shiftIn(DATA\_PIN\_I,CLOCK\_PIN\_I,MSBFIRST); // читаем байт

return retByte;

}

**contr7seg.ino:**

// Изменение значений разрядов семисегментного индикатора по нажатию на тактовые кнопки

// Подключение кнопок и индикатора осуществляется через сдвиговые регистры.

//---------------------------------------------------------------------------

// настройка линии подачи напряжения

int POWR = 2; // питание +5В

// настройка пинов Arduino для выходного регистра

int LATCH\_PIN\_O = 8; // к 12 пину регистра - вход «защёлкивающий» данные

int DATA\_PIN\_O = 11; // к 14 пину регистра - вход данных

int CLOCK\_PIN\_O = 13; // к 11 пину регистра - вход для тактовых импульсов

// настройка пинов Arduino для входного регистра

int LATCH\_PIN\_I = 9; // к 1 пину регистра - вход «защёлкивающий» данные

int DATA\_PIN\_I = 12; // к 9 пину регистра - вход данных

int CLOCK\_PIN\_I = 13; // к 2 пину регистра - вход для тактовых импульсов

// описание кодов для сегментов индикатора

byte S\_TOP = ~(1 << 1); // A

byte S\_RTOP = ~(1 << 2); // B

byte S\_RBOT = ~(1 << 3); // C

byte S\_BOT = ~(1 << 4); // D

byte S\_LBOT = ~(1 << 5); // E

byte S\_LTOP = ~(1 << 6); // F

byte S\_MID = ~(1 << 7); // G

byte S\_DP = ~(1 << 0); // DP

byte S\_EMPT = ~0; // пустой

// массив значений кодов для отображения символов

byte ledChars[14] = {

(byte) (S\_TOP & S\_BOT & S\_LTOP & S\_LBOT & S\_RTOP & S\_RBOT), // 0

(byte) (S\_RTOP & S\_RBOT), // 1

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LBOT & S\_RTOP), // 2

(byte) (S\_TOP & S\_MID & S\_BOT & S\_RTOP & S\_RBOT), // 3

(byte) (S\_MID & S\_LTOP & S\_RTOP & S\_RBOT), // 4

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_RBOT), // 5

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_RBOT), // 6

(byte) (S\_TOP & S\_RTOP & S\_RBOT), // 7

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT & S\_RTOP & S\_RBOT), // 8

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_RTOP & S\_RBOT), // 9

(byte) (S\_DP), // .

(byte) (S\_EMPT), // пусто

(byte) (S\_TOP & S\_MID & S\_BOT & S\_LTOP & S\_LBOT), // E

(byte) (S\_MID & S\_LBOT), // r

};

// массив значений разрядов индикатора (0-3)

byte ledDigs [4] = {

1 << 1, // 0

1 << 2, // 1

1 << 3, // 2

1 << 4, // 3

};

void setup() {

// настройка питания +5В на 6 пине Arduino

pinMode(POWR, OUTPUT);

digitalWrite(POWR, HIGH);

// настройка портов для выходного регистра

pinMode(LATCH\_PIN\_O, OUTPUT);

pinMode(DATA\_PIN\_O, OUTPUT);

pinMode(CLOCK\_PIN\_O, OUTPUT);

// настройка портов для входного регистра

pinMode(LATCH\_PIN\_I, OUTPUT);

pinMode(CLOCK\_PIN\_I, OUTPUT);

pinMode(DATA\_PIN\_I, INPUT);

// включаем защёлку входного регистра

digitalWrite(LATCH\_PIN\_O, HIGH);

}

void loop() { // бесконечный цикл

static byte lastDataByte = 0; // предыдущие состояния входов регистра

static int dispNum = 0;

int addNum = 0; // слагаемое, добавляемое к отображаемому числу byte dataByte = readByte(); // читаем состояния входного регистра

if (dataByte != lastDataByte){ // если состояние любого входа изменилось

if (dataByte > lastDataByte){ // реагируем только на нажатия

byte changedDataByte = dataByte ^ lastDataByte; // изменившиеся биты

// подбираем множитель, соответствующий изменившемуся биту

int i = 0; // счётчик разрядов в байте

int m = 1; // множитель разряда (1 10 100 1000 соотв.)

while(!addNum){ // пока не подберём множитель

if (changedDataByte & (1<<i)){ // если натыкаемся на ненулевой бит

addNum = m; // присваиваем слагаемому множитель разряда

}

i++; // увеличивем счётчик разрядов

m = m\*10; // увеличиваем степень множителя 10^i

}

}

lastDataByte = dataByte; // запоминаем текущие состояния

}

dispNum = dispNum + addNum; // добавляем дополнительное слагаемое

if (dispNum > 9999){ // если число превышает максимальное - сбрасываем его

dispNum = 0;

}

ledDispNumSet(dispNum); // отображаем число на индикаторе

}

//---------------------------------------------------------------------------

// ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

// Отображение инкрементирования

void dispInc(int t) { // период инкрементирования

for (int i = 0; i < 10000; i++) {

for (int j = 0; j < t; j++) {

ledDispNumSet(i);

}

}

}

//---------------------------------------------------------------------------

// ИНДИКАТОР

// отображение произвольного символа

void ledDisp(byte symbCode, byte posCode) { // катоды, аноды

digitalWrite(LATCH\_PIN\_O, LOW);

shiftOut(DATA\_PIN\_O, CLOCK\_PIN\_O, MSBFIRST, symbCode); // байт для катодов

shiftOut(DATA\_PIN\_O, CLOCK\_PIN\_O, MSBFIRST, posCode); // байт для анодов

digitalWrite(LATCH\_PIN\_O, HIGH); // "защёлкиваем" данные

}

// отображение символа в соотв. с массивом

// (индекс символа в массиве, разряд на котором отображать)

void ledDispChar(int num, int dig) {

ledDisp(ledChars[num], ledDigs[dig]);

}

// отображение точки в указанном разряде

void ledDispPoint(int dig) { // разряд для отображения точки

ledDisp(ledChars[10], ledDigs[dig]);

}

// отображение ошибки

void ledDispErr() {

ledDispChar(13, 1); // символ E

ledDispChar(13, 2); // символ r

ledDispChar(12, 3); // символ r

}

// отображение 4-х значного числа (динамическая индикация)

void ledDispNumSet(int x) { // 4-х значное число

if (x == 0) { // если число 0 - сразу выводим 0

ledDispChar(0, 0);

return;

}

if (x > 9999) { // если число > 4 знаков - выводим ошибку

ledDispErr();

return;

}

int nums[4] = {11, 11, 11, 11}; // массив пустых символов

int i = 0;

while (x > 0) { // разбиение по разрядам в массив nums

nums[i] = x % 10;

x = x / 10;

i++;

}

for (int i = 0; i < 4; i++) { // вывод массива nums на индикатор

ledDispChar(nums[i], i);

}

}

// отображение числа с точкой в указанном разряде

// (число для отображения, разряд для точки)

void ledDispNumSetWithPoint(int num, int dig) { if (dig < 4 && dig >= 0) { // проверяем, соотв. ли номер существующему разряду

ledDispNumSet(num); // отображаем число

ledDispPoint(dig); // отображаем точку

} else { // если номер разряда отрицательный или превышает их кол-во

ledDispNumSet(num); // отображаем только число

}

}

//---------------------------------------------------------------------------

// ВХОДНОЙ РЕГИСТР

// чтение байта с входного регистра

byte readByte(){

byte retByte = 0; // возвращаемый байт

digitalWrite(CLOCK\_PIN\_I,HIGH); // необходимо для работы shiftIn()

digitalWrite(LATCH\_PIN\_I, LOW); // "дёргаем" защёлку

digitalWrite(LATCH\_PIN\_I, HIGH);

retByte = shiftIn(DATA\_PIN\_I,CLOCK\_PIN\_I,MSBFIRST); // читаем байт

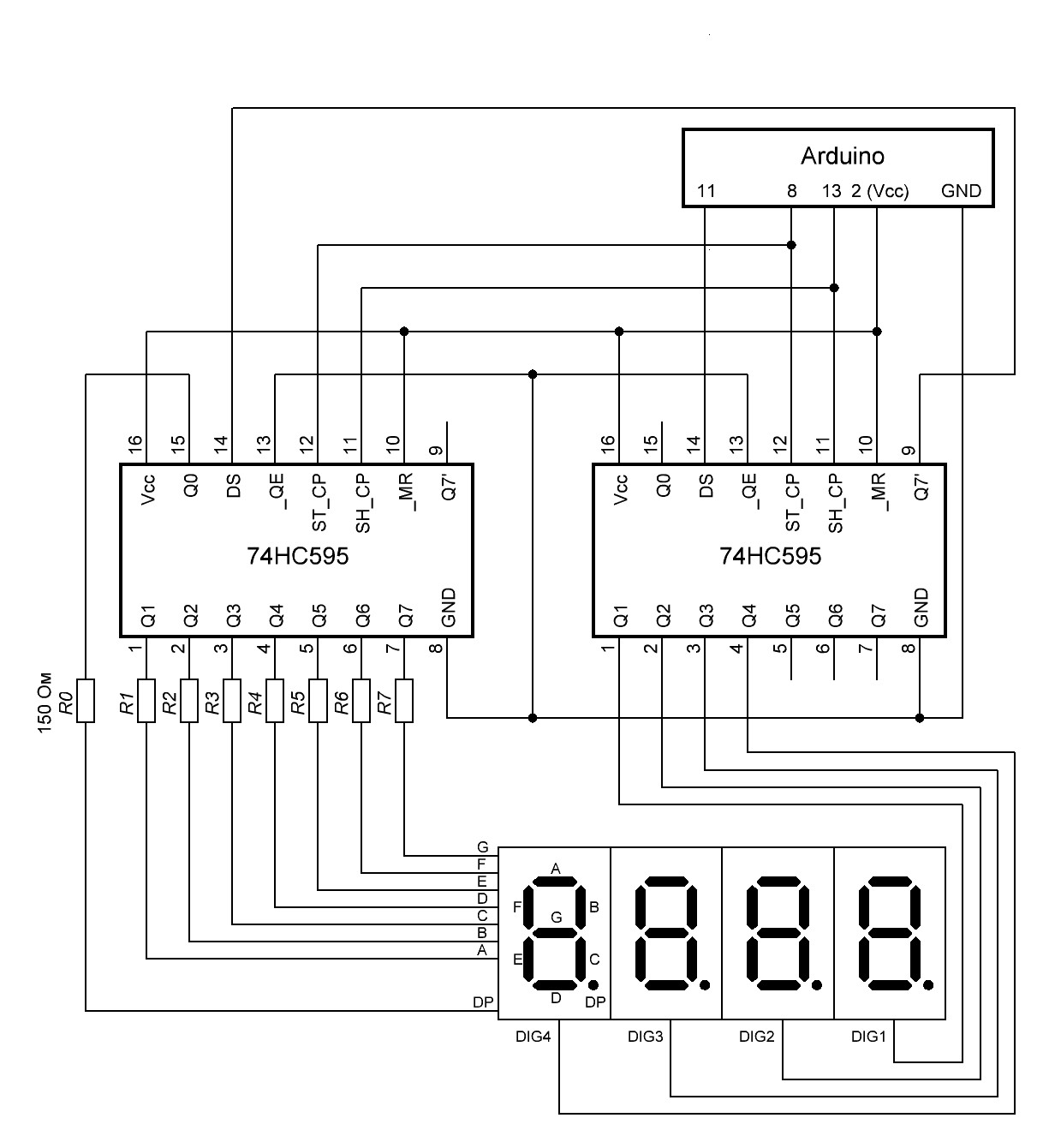
return retByte;

}

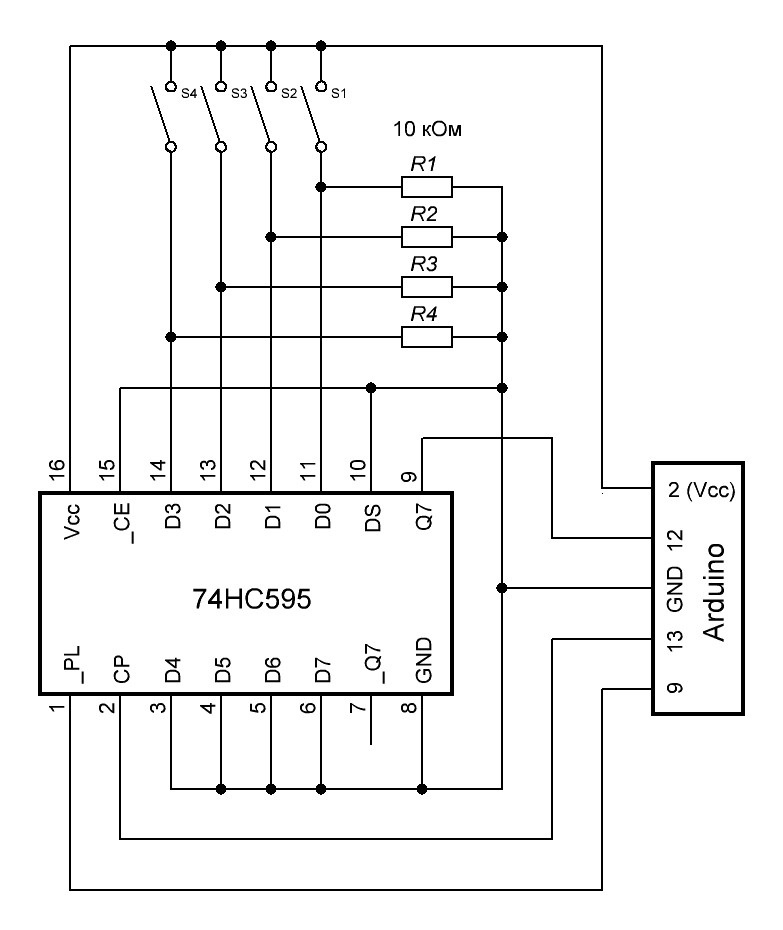
**ПРИЛОЖЕНИЕ Б**

**Принципиальные схемы**

**Схема подключения семисегментного индикатора к МК через сдвиговый регистр:**



**Схема подключения тактовых кнопок к МК через входной сдвиговый регистр:**



**Схема подключения семисегментного индикатора и тактовых кнопок к контроллеру Arduino:**

