# Система дистанционного управления по электрической сети

Описанное здесь приложение является основой системы, которая позволяет дистанционно управлять различными электроприборами без использования других соединений, кроме обычной электрической сети дома или другого помещения. Конкретное предназначение этого проекта состоит в том, чтобы управлять всеми шторами в доме с одной точки; предполагается, что шторы оборудованы электродвигателями, имеющими рядом выключатели, которые позволяют их закрыть, открыть или остановиться в любом промежуточном положении.

Цели проекта состоят в иллюстрации реализации различных функций на ST7, таких как:

Интерфейс кнопок

Детектирование перехода напряжения через ноль

Детектирование работы двигателя

Бестрансформаторное питание и т.д.

Система имеет следующую архитектуру:

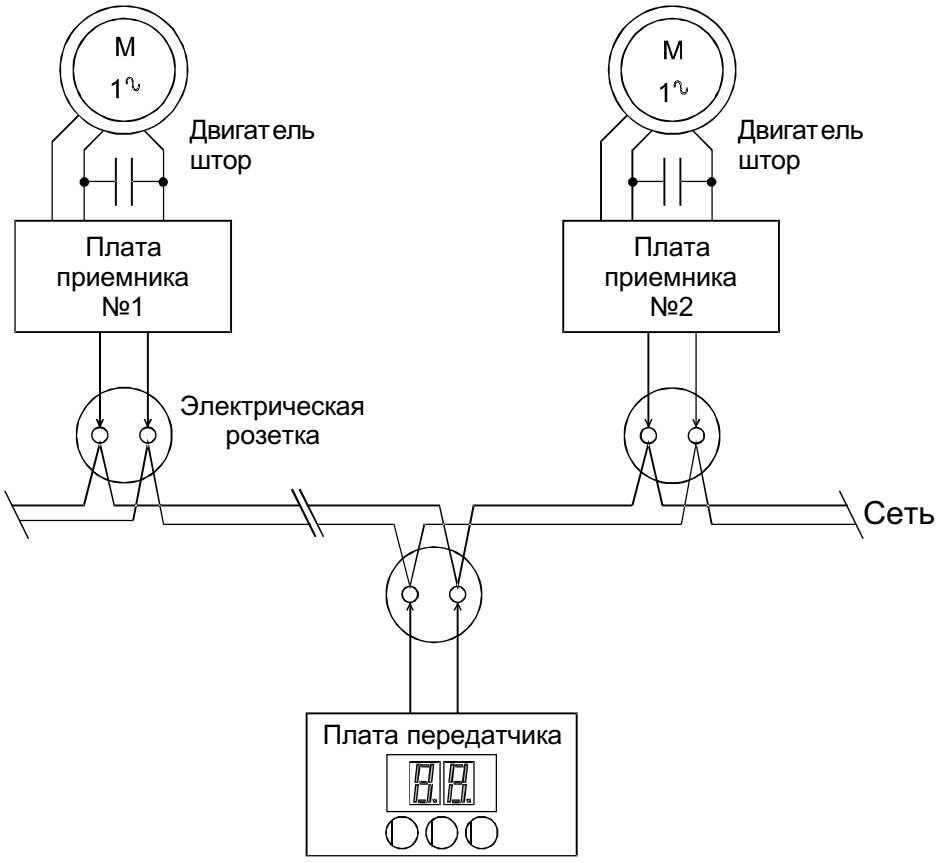



Схема системы дистанционного управления шторами

09-inst

Вначале мы объясним, что собой представляет управление по сети, взяв за основу стандарт X-10; затем мы опишем передатчик. Вторая часть главы будет посвящена приемнику.

# 1. Управление по сети и стандарт X-10

Управление по сети представляет собой беспроводную связь между двумя электронными устройствами. Термин «беспроводная» следует понимать как «не требующая соединения устройств с помощью специального кабеля», так как они уже соединены проводами питающей сети.

Основной принцип управления по сети основан на способности электропроводки передавать радиочастотные сигналы вместе с сетевым напряжением. Так как электропроводка изначально не была для этого предназначена, очевидно, что параметры линии в таком режиме не идеальны, поэтому должны иметься в виду следующие ограничения:

> Несущая частота должна выбираться таким образом, чтобы обеспечить минимальное затухание от передатчика до приемника.
>
> Мощность передаваемой несущей должна быть минимальной во избежание радиочастотных помех, так как общая длина электропроводки в доме насчитывает десятки или сотни метров, что образует протяженную антенну.

Это приводит к следующему:

> Наилучшее значение несущей частоты лежит в диапазоне от 100 до 150 КГц;
>
> Напряжение радиочастотного сигнала должно быть лишь немного больше одного вольта.

Отсюда следует:

> Выбранная частота почти полностью блокируется электросчетчиком, что ограничивает протяженность используемого участка электропроводки.
>
> Из-за наличия различных электроприборов, в частности, светорегуляторов и коллекторных двигателей, отношение сигнал/шум в посылке может быть очень плохим.

Как результат, можно рассчитывать только на малые скорости передачи данных, да и то со значительным потоком ошибок. Из-за этого разработано несколько разных стандартов, обеспечивающих разные возможности в плане скорости передачи данных, потока ошибок и стоимости.

Стандарт X-10, который описывается ниже, ориентирован в основном на снижение стоимости. Поэтому было умышленно пожертвовано скоростью передачи данных для обеспечения достаточно низкого потока ошибок в целях обеспечения надежности системы.

Следующее ниже описание теории X-10 заимствовано с Web[1].

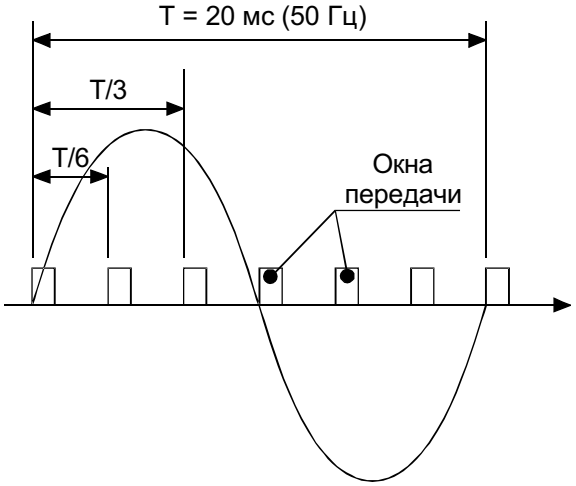
Стандарт X-10 позволяет связать между собой передатчики и приемники по проводам электрической сети. Сигналы передатчиков представляют собой короткие радиочастотные вспышки, которые несут цифровую информацию. Они синхронизированы с переходами сетевого напряжения через ноль. Передача должна осуществляться как можно ближе к моментам перехода через ноль, не дальше 200 мкс от этого перехода.

Логическая 1 представлена вспышкой длительностью 1 мс с частотой 120 КГц вблизи перехода через ноль, а логический 0 – отсутствием этой вспышки. Эти 1-миллисекундные вспышки должны передаваться три раза через равные промежутки времени, чтобы совпасть с моментами перехода через ноль всех трех фаз в трехфазных системах.

На рисунке ниже показаны временные соотношения для этих вспышек относительно перехода через ноль.

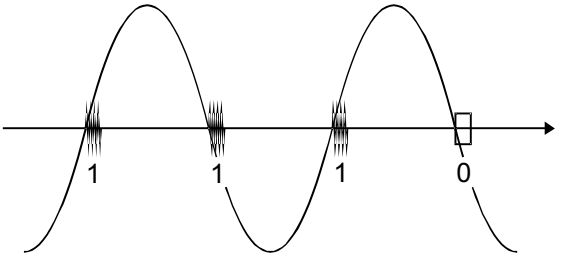
---

[1] X-10 Technology Transmission Theory. Автор: David Gaddis, August 14, 1995. http://www.hometeam.com




09-timing

Полная передача кода содержит одиннадцать периодов питающей сети. Два первых периода представляют стартовый код, как показано ниже:




Технология X-10: стартовый код

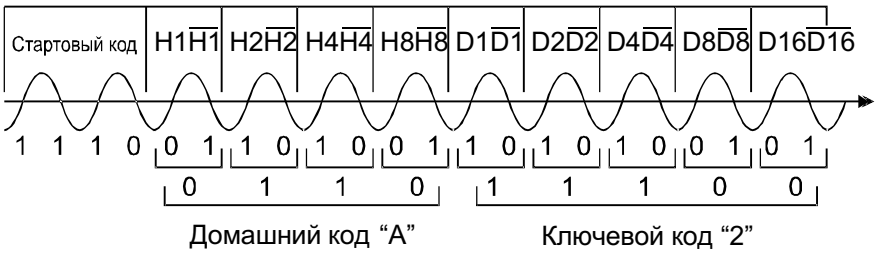
09-xst

Следующие четыре периода представляют домашний код, а четыре последних периода представляют числовой код (от 1 до 16) или код функции (Вкл., Выкл. и т.д.). Этот полный блок (стартовый код, домашний код, ключевой код) всегда должен передаваться группами по 2 (повторяться дважды) с промежутками 3 периода питающей сети между группами. Команды увеличения и уменьшения яркости света являются исключениями из правила и должны передаваться непрерывно (как минимум дважды) без промежутков между кодами. См. рисунок ниже.


Технология X-10: принцип передачи кода

09-xcode

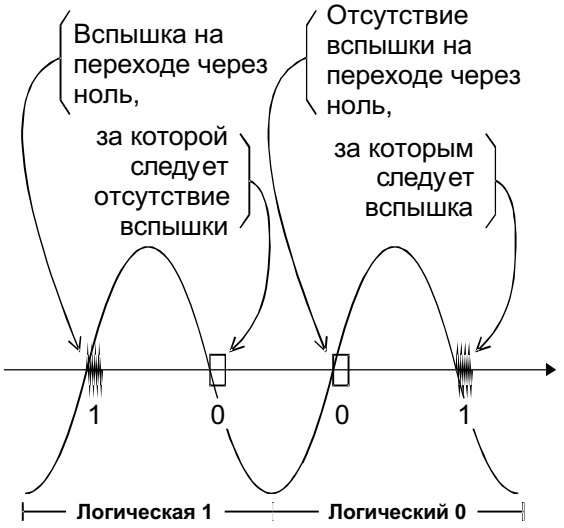
Пример: следующий рисунок показывает передачу ключевого кода 2 на домашний код A:




Технология X-10: пример передачи

09-xex

В любом блоке данных каждый бит 4-х или 5-битного кода должен передаваться в инверсном виде на противоположном полупериоде сетевого напряжения. Так, если на одном полупериоде передается 1-миллисекундная вспышка (логическая 1), то на следующем полупериоде вспышки быть не должно (логический 0). См. следующий рисунок.




Технология X-10: принцип передачи битов
(Домашний и ключевой коды,
исключая стартовый код)

09-xbit

Таблицы на следующем рисунке показывают двоичные коды для передачи каждого домашнего кода и функционального кода. Стартовый код всегда равен 1110, он является уникальным кодом и единственным кодом, который не подчиняется правилу инверсных бит в соседних полупериодах.

| Домашние коды | | | | | Ключевые коды | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | H1 | H2 | H4 | H8 | Цифровые коды | D1 | D2 | D4 | D8 | D16 |
| A | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| B | 1 | 1 | 1 | 0 | 2 | 1 | 1 | 1 | 0 | 0 |
| C | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 0 |
| D | 1 | 0 | 1 | 0 | 4 | 1 | 0 | 1 | 0 | 0 |
| E | 0 | 0 | 0 | 1 | 5 | 0 | 0 | 0 | 1 | 0 |
| F | 1 | 0 | 0 | 1 | 6 | 1 | 0 | 0 | 1 | 0 |
| G | 0 | 1 | 0 | 1 | 7 | 0 | 1 | 0 | 1 | 0 |
| H | 1 | 1 | 0 | 1 | 8 | 1 | 1 | 0 | 1 | 0 |
| I | 0 | 1 | 1 | 1 | 9 | 0 | 1 | 1 | 1 | 0 |
| J | 1 | 1 | 1 | 1 | 10 | 1 | 1 | 1 | 1 | 0 |
| K | 0 | 0 | 1 | 1 | 11 | 0 | 0 | 1 | 1 | 0 |
| L | 1 | 0 | 1 | 1 | 12 | 1 | 0 | 1 | 1 | 0 |
| M | 0 | 0 | 0 | 0 | 13 | 0 | 0 | 0 | 0 | 0 |
| N | 1 | 0 | 0 | 0 | 14 | 1 | 0 | 0 | 0 | 0 |
| O | 0 | 1 | 0 | 0 | 15 | 0 | 1 | 0 | 0 | 0 |
| P | 1 | 1 | 0 | 0 | 16 | 1 | 1 | 0 | 0 | 0 |
| Функциональные коды | | | | | | | | | | |
| Выключить все устройства | | | | | | 0 | 0 | 0 | 0 | 1 |
| Включить все освещение | | | | | | 0 | 0 | 0 | 1 | 1 |
| Включить | | | | | | 0 | 0 | 1 | 0 | 1 |
| Выключить | | | | | | 0 | 0 | 1 | 1 | 1 |
| Уменьшить яркость | | | | | | 0 | 1 | 0 | 0 | 1 |
| Увеличить яркость | | | | | | 0 | 1 | 0 | 1 | 1 |
| Выключить все освещение | | | | | | 0 | 1 | 1 | 0 | 1 |
| Расширенный код | | | | | | 0 | 1 | 1 | 1 | 1 |
| Запрос приветствия | | | | | | 1 | 0 | 0 | 0 | 1 |
| Ответ приветствия | | | | | | 1 | 0 | 0 | 1 | 1 |
| Предустановка яркости | | | | | | 1 | 0 | 1 | X | 1 |
| Расширенные данные (аналоговые) | | | | | | 1 | 1 | 0 | 1 | 1 |
| Статус = включено | | | | | | 1 | 1 | 0 | 1 | 1 |
| Статус = выключено | | | | | | 1 | 1 | 1 | 0 | 1 |
| Запрос статуса | | | | | | 1 | 1 | 1 | 1 | 1 |

**Технология X-10: Таблица домашних и ключевых кодов**

**Замечание 1:** «Запрос приветствия» передается для того, чтобы убедиться, присутствуют ли какие-либо передатчики X-10 в радиусе действия. Это позволяет пользователю установить другой домашний код, если получен «Ответ приветствия».

**Замечание 2:** В команде «Предустановка яркости» бит D8 представляет собой старший бит, а H1, H2, H4 и H8 – младшие биты уровня.

**Замечание 3:** За кодом «Расширенные данные» следуют 8-битные посылки, которые могут представлять аналоговые данные (после аналого-цифрового преобразования). Между кодом «Расширенные данные» и посылками данных промежутков быть не должно. Первый передаваемый байт может быть использован для указания количества байт данных. Если оставить промежутки между байтами данных, такие коды не будут приняты модулями X-10, что вызовет ошибку.

«Расширенный код» аналогичен «Расширенным данным»: байты, которые следуют за «Расширенным кодом» (без промежутков), могут представлять дополнительные коды. Это позволяет разработчику расширить количество кодов сверх 256, которые доступны сейчас.

Модули приемников X-10 требуют «тишины» как минимум в течение 3-х периодов сетевого напряжения между каждой парой 11-битных кодовых посылок (без промежутка между парами). Исключением из этого правила являются команды увеличения и уменьшения яркости. Они передаются непрерывно, без промежутков между 11-битными кодами. В этом случае промежуток в 3 периода сетевого напряжения необходим только между разными кодами, т.е. между увеличением и уменьшением яркости, или между включением и увеличением яркости и т.д.

# 2. Передатчик

### 2.1 Инструкция по использованию

Передатчик имеет панель управления со следующими органами управления:

Кнопка ОТКРЫТЬ (OPEN), которая сразу открывает шторы.

Кнопка ЗАКРЫТЬ (CLOSE), которая сразу закрывает шторы.

Кнопка ВРЕМЯ (TIME) устанавливает задержку, спустя которую шторы должны быть закрыты. При каждом нажатии счетчик часов инкрементируется. Количество часов индицируется на 2-х разрядном дисплее, максимальная возможная задержка составляет 15 часов. Поскольку количество часов всегда больше нуля, внутренний таймер считает время, декрементируя счетчик часов каждый час. Когда счетчик обнуляется, поступает команда ЗАКРЫТЬ. Нажатие кнопки ОТКРЫТЬ или ЗАКРЫТЬ сбрасывает время на ноль. Для сброса задержки, которая уже установлена, без закрытия штор, нажмите кнопку ОТКРЫТЬ.

### 2.2 Описание электрической схемы

Передатчик питается от сети через понижающий трансформатор; однако земляной проводник устройства непосредственно соединен с одним из сетевых проводов.

Вывод PB1 соединен с затвором транзистора, который включает или выключает генератор 120 КГц. Выход этого генератора усиливается и подается на другой сетевой провод через конденсатор.

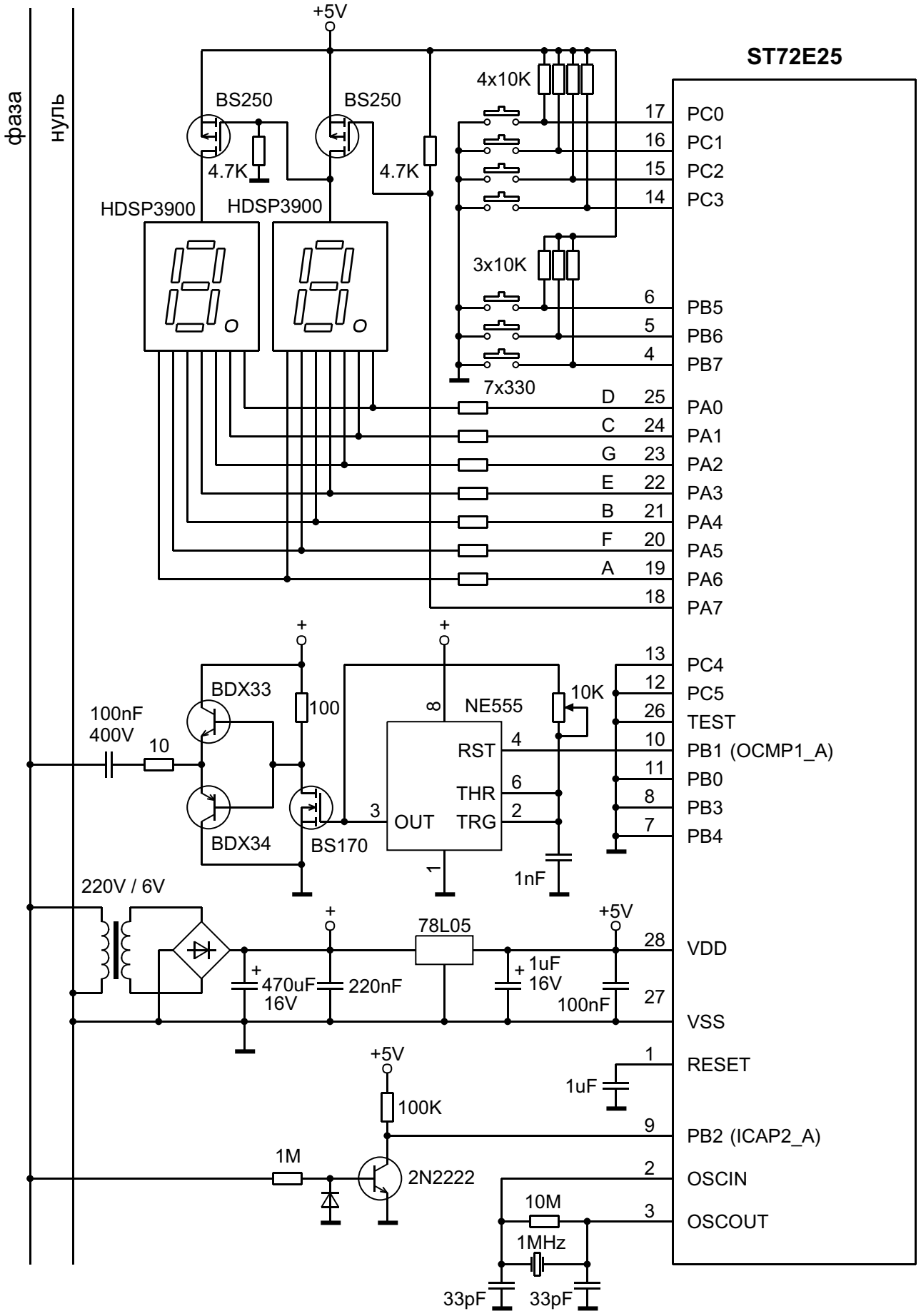
Этот же провод используется для детектирования перехода через ноль. Для этого он подключен к базе транзистора через резистор сопротивлением 1 Мом. Диод защищает базу от обратного напряжения. Коллектор нагружен на резистор 100 Ком, сигнал с него поступает на вывод PB2.

Три кнопки подключены к выводам PB5 – PB7 вместе с подтягивающими резисторами, при нажатии на кнопки выводы соединяются с землей.

| Функция | Номер входа |
|---|---|
| ОТКРЫТЬ | PB5 |
| ЗАКРЫТЬ | PB6 |
| ВРЕМЯ | PB7 |

Дисплей содержит два одноразрядных семисегментных светодиодных индикатора. Все сегменты обоих индикаторов соединены параллельно и подключены к выводам порта A. Общие электроды каждого индикатора управляются транзисторами, подключенными к порту PA7. Схема управления построена таким образом, что если на выводе PA7 высокий уровень, горит одна цифра. Когда низкий – другая.

Домашний код задается с помощью 4-разрядного DIP-переключателя, подключенного к выводам PC0 – PC3.




Система дистанционного управления: принципиальная схема передатчика

Используются следующие возможности ST72251:

Первый таймер работает в режиме ШИМ. Он сконфигурирован для генерации импульсов на выходе OCMP1_A, что является альтернативной функцией вывода PB1. Период повторения установлен равным точно 1/6 периода сетевого напряжения, а длительность импульса составляет 1 мс.

Как указано в описании ST72251, в режиме ШИМ таймер не может генерировать прерывания по совпадению. Однако таймер A имеет специальную возможность, которая заключается в том, что событие совпадение 2 заставляет возникать событие совпадение 1. Данное приложение использует эту возможность для получения одного прерывания на каждом периоде счетчика (т.е. шесть раз на каждом периоде сетевого напряжения). Для его разрешения бит маски прерывания для события совпадения установлен.

Синусоидальное напряжение со второго сетевого провода (первый уже подключен к $V_{SS}$) используется для получения прямоугольных импульсов, которые подаются на вход PB2. Этот вход используется как второй вход захвата таймера A (ICAP2_A). Захват значения независимо работающего счетчика выполняется один раз при изменении знака сетевого напряжения с положительного на отрицательный, т.е. один раз за период. Так как маска прерывания по захвату установлена, как было сказано выше, событие захват 2 также вызывает запрос прерывания. Какое из событий вызвало прерывание, определяется в подпрограмме обработки путем проверки двух битов, ICF1 и ICF2 в регистре состояния таймера (TSRA).

Внешняя схема, которая генерирует ВЧ колебания и посылает их в сеть, может включаться и выключаться для управления временем, когда присутствует несущая. Включение производится с помощью вывода PB1, который используется как выход совпадения таймера A (OCMP1_A), что сконфигурировано путем установки в единицу бита разрешения совпадения в регистре управления таймера 2 (OC1E регистра TACR2).

В такой конфигурации таймер будет генерировать непрерывную серию импульсов длительностью 1 мс. Для модуляции этих импульсов согласно стандарту X-10, бит OLVL1 регистра TACR1 устанавливается для передачи импульса, или сбрасывается, когда передача не нужна. Этот бит определяет уровень, который будет присутствовать на выходе при совпадении 1. Так как бит OLVL2 определяет выходной уровень для совпадения 2 (когда таймер находится в режиме ШИМ), и этот бит все время сброшен, то выход OCMP1_A или остается нулевым, или на нем присутствуют единичные импульсы, согласно состоянию OLVL1.

Импульсы должны быть привязаны к сетевому напряжению. Для достижения этого используется схема фазовой автоподстройки.

По прерыванию при переходе сетевого напряжения через ноль значение независимо работающего счетчика захватывается и генерируется запрос прерывания. Подпрограмма обслуживания прерывания проверяет это значение. Целью является равенство этого значения некоторой образцовой величине; на самом деле, значение может быть большим (опережение по времени) или меньшим (отставание по времени). Часть разницы между действительной и желаемой величиной добавляется к образцовому периоду, и сумма записывается в регистр сравнения 2, который изменяет период повторения таймера. Эта схема является системой с обратной связью, которая стабилизирует частоту на уровне сетевой частоты, умноженной на шесть, а также фазу таким образом, что начало импульса будет точно совпадать с моментом перехода сетевого напряжения через ноль, как того требует стандарт X-10.
В результате период будет оставаться в пределах допустимой величины. Отклонения периода таймера происходят в узком диапазоне, поэтому большие отклонения частоты невозможны. Это предотвращает выбросы и случайные импульсы в результате ошибочного

изменения положения 120 КГц импульсов относительно перехода сетевого напряжения через ноль.

Таймер B установлен в режим ШИМ и переполняется 64 раза в секунду. Он генерирует запросы прерывания.

### 2.3 Описание программного обеспечения

Программное обеспечение разделено на основную программу и три подпрограммы обслуживания прерываний.

Основная программа проверяет состояние кнопок и определяет командные последовательности для передачи. Она выполняется непрерывно.

Первая подпрограмма обслуживания прерывания выполняется после каждого переданного импульса. Один раз из трех (т.е. один раз за полупериод) она определяет, должен ли посылаться импульс согласно стандарту X-10 и тому сообщению, которое нужно передать, и соответствующим образом устанавливает бит OLVL1.

Вторая подпрограмма обслуживания прерывания выполняется при каждом переходе сетевого напряжения через ноль. Она синхронизирует таймер.

Третья подпрограмма обслуживания прерывания выполняется при каждом переполнении таймера B. Это происходит 64 раза в секунду. Подпрограмма обслуживания делает две вещи:

> Она обновляет дисплей, который мультиплексируется и поэтому должен быстро регенерироваться для предотвращения мерцаний.
>
> Она производит обратный отсчет времени, обеспечивая необходимую временную базу для индицируемой задержки.

Программное обеспечение целиком написано на Си, за исключением нескольких строк на ассемблере; необходимость этого будет объяснена ниже. Данный пример показывает, насколько легко писать (а в еще большей степени читать) исходный текст на Си, даже если задача выглядит простой и явно не требует большого количества программирования.

Здесь мы в деталях рассмотрим все части программного обеспечения.

#### 2.3.1 Основная программа

Основная программа включает участок инициализации и бесконечный цикл, который обрабатывает нажатия кнопок.

Участок инициализации настраивает порты, таймера и все требуемые регистры.

Бесконечный цикл считывает состояние кнопок, ожидает его установления в течение определенного промежутка времени (подавление дребезга) и производит соответствующие действия согласно нажатой кнопке:

Для кнопок OPEN и CLOSE вызывается функция `SendCommand`. Она устанавливает глобальные переменные для сообщения подпрограмме обработки прерывания, что необходимо передать сообщение:

```
/* Эта функция инициирует передачу одной команды */
void SendCommand ( Byte Command )
      {
      KeyCode = Codes[Command] ;          /* подготовка двух переменных, */
      HouseCode = Codes[HOUSE_CODE] ;     /* содержащих данные для передачи. */
      CycleNumber = 1 ;                   /* запуск процесса передачи */
      }
```

Эти глобальные переменные также используются прерыванием, которое посылает импульсы.

Как было сказано выше, передаваемые коды не соответствуют своему двоичному эквиваленту. Поэтому передаваемая величина извлекается из перекодировочной таблицы `Codes[]`, расположенной в ПЗУ, как будет объяснено дальше.

Кнопка TIME инкрементирует счетчик часов, если его значение меньше 15.

Подавление дребезга осуществляется следующей функцией. Она считывает состояние трех входов, соединенных с кнопками, и проверяет, совпадает ли считанное значение с предыдущим. Если это продолжает выполняться для некоторого количества считываний подряд, состояние кнопок принимается установившимся. После этого значения трех этих битов проверяются. Если нажата только одна кнопка, возвращается ее код. Если не нажата ни одна кнопка, или нажато более одной кнопки, возвращается нулевой код.

```
/* Эта функция ожидает, пока состояние кнопок установится, и */
/* возвращает код нажатой кнопки. Если их несколько, то 0. */
Byte Debounce ( void )
      {
      Byte PreviousButton = PUSH_BUTTONS ;
      int DebounceTime = DEBOUNCE_TIME ;

      /* Wait for state stable */
      while ( DebounceTime != 0 )
            {
            if ( PreviousButton != PUSH_BUTTONS )
                  {
                  DebounceTime = DEBOUNCE_TIME ; /* состояние изменилось:
                                                    перезапуск задержки */
                  PreviousButton = PUSH_BUTTONS ;
                  }
            else
                  DebounceTime-- ;
            }
      switch ( PreviousButton )
            {
            case CLOSE_BUTTON :
            case OPEN_BUTTON :
            case TIME_BUTTON :
                  return PreviousButton ;
            default:
                  return 0 ; /* Кнопка не нажата или нажато более одной
                                кнопки. */
            }
      }
```

Заметьте, что в тексте использовался макрос `PUSH_BUTTONS`. Этот макрос определен следующим образом:

```
#define PUSH_BUTTONS ( ( ~PBDR ) & 0xE0 ) /* три старших бита */
#define CLOSE_BUTTON 0x80
#define OPEN_BUTTON 0x40
#define TIME_BUTTON 0x20
```

Это выражение инвертирует биты, считанные с порта B, которые равны нулю, когда кнопка нажата; возвращаются только три бита, соответствующие кнопкам. Использование этого определения в начале исходного текста позволяет Вам легко изменить электрическую схему так, что кнопки будут подключены к другим выводам, в другом порядке, или будут устанавливать высокий уровень при нажатии. Только эти четыре строки должны быть изменены, что значительно легче, чем просматривать весь исходный текст в поиске того, как считываются кнопки.

Основная программа следующая:

```
void main ( void )
{
InitPorts () ;
InitTimerA() ;
InitTimerB() ;
MISCR = 0x00 ;    /* Нормальный режим (clock/2). */
EnableInterrupts ;

while (1)
      {
      switch ( Debounce () )
            {
            case CLOSE_BUTTON :
                  Delay = 0 ;       /* Отмена ожидания. */
                  SendCommand ( CLOSE_COMMAND ) ;
                  break ;

            case OPEN_BUTTON :
                  Delay = 0 ;       /* Отмена ожидания. */
                  SendCommand ( OPEN_COMMAND ) ;
                  break ;

            case TIME_BUTTON :
                  Hour = 3600 ;     /* Делает точным первый час. */
                  if ( Delay < 15 )
                  Delay++ ;
                  break ;
            }
      while ( CycleNumber != 0 ) ;  /* ожидание конца передачи */

      while ( Debounce () != 0 ) ;  /* ожидание отпускания всех кнопок */
      }
}
```

Прокомментированы только те строки, которые не являются очевидными.

### 2.3.2 Подпрограмма обработки прерывания по захвату таймера А

Существуют два разных события захвата, которые могут происходить для каждого таймера. Однако эти события разделяют один вектор прерывания, поэтому для любого события вызывается одна и та же подпрограмма обслуживания прерывания. Первой задачей подпрограммы обслуживания является определение события, которое вызвало прерывание.

**Общая подпрограмма обслуживания прерывания таймера А.**

Текст подпрограммы обслуживания прерывания:

```
#pragma TRAP_PROC SAVE_REGS
void TimerAInterrupt ( void )
      {
      WDGR = 0xFF ;     /* перезапуск сторожевого таймера */
      if ( TASR & ( 1 << ICF1 ) ) /* Определение, какое прерывание произошло. */
            {                     /* Это прерывание псевдо-захвата 1 */
            TAOC2HR = TimerAPeriod >> 8 ;       /* Установка нового периода, как
                                                   рассчитано в подпрграмме */
            TAOC2LR = TimerAPeriod & 0xff ;     /* обслуживания прерывания
                                                   захвата. Старший байт *ДОЛЖЕН*
                                                   быть записан первым. */
            asm
                  {
                  ld a, TAIC1LR ;   /* Пустое чтение для очистки
                                       запроса прерывания */
                  }
            if ( CycleNumber != 0 ) /* если не ноль, передача продолжается */
```

```
                {
                if ( Phase == 0 )
                SendOneFrameElement () ; /* Изменять элемент каждые 3
                                            прерывания. */
                Phase++ ;          /* инкремент фазы для следующего вызова */
                if ( Phase > 2 )
                Phase = 0 ;        /* 0, 1, 2, 0, 1, 2 ... */
                }
        }
else
        {     /* Прерывание по захвату 2. */
        PhaseLockedLoop () ;
                /* Запрос прерывания очищен чтением TAIC2LR */
        }
}
```

Первым проверяется флаг ICF1 для того, чтобы узнать, вызвано ли прерывание событием захвата 1. Если да, то выполняется первый блок; если нет, то второй.

В первом случае два байта регистра сравнения 2 загружаются величиной, которая хранится в глобальной переменной `TimerPeriod`. Так как это слово, его два байта выделяются с помощью двух простых выражений. Заметьте, что старший байт должен записываться первым, или функция сравнения будет запрещена, как описано в технической документации.

Для очистки запроса прерывания, как сказано в документации, должен быть прочитан регистр состояния, а затем должен быть произведен доступ к младшему байту регистра захвата. Здесь регистр состояния считывается всегда при определении, какое прерывание произошло. После этого мы должны считать регистр `TAIC1LR`. Для того чтобы это сделать на Си, его значение должно быть ассоциировано с переменной, которая используется позже. Иначе оптимизация, производимая компилятором, выбросит это чтение, результат которого не используется. Простейшим способом выполнить чтение этого регистра является ассемблерная вставка:

```
asm
        {
                ld a, TAIC1LR ;    /* Пустое чтение для очистки
                                   запроса прерывания */
        }
```

Величина, считанная в аккумулятор, игнорируется, но чтение регистра `TAIC1LR` осуществляться будет.

После этого счетчик инкрементируется для того, чтобы отсчитать три вызова подпрограммы обслуживания; функция `SendOneFrameElement` вызывается каждое третье прерывание. Это потому, что стандарт X-10 требует, чтобы каждый импульс передавался три раза в каждом полупериоде, так что приемники, подключенные к другим фазам питающей сети и рассинхронизированные на 120 или 240 градусов относительно передатчика, все равно будут находить импульсы.

Если прерывание вызвано событием захват 2, то вызывается функция `PhaseLockedLoop`.

**Функция `SendOneFrameElement`.**

Эта функция относительно длинная, хотя на самом деле очень простая. Ее поведение определяется глобальной переменной `CycleNumber`, которая выбирает один из 50 вариантов. Переменная инкрементируется каждый раз, так что каждый вариант выполняется один раз при каждом обслуживании прерывания. Так как прерывания возникают дважды за период сетевого напряжения, каждый передаваемый бит обрабатывается дважды, двумя последовательными прерываниями.

Первые четыре бита являются префиксом, или битами синхронизации. Они всегда должны быть 1110.

Следующие четыре пары битов несут домашний код. В каждой паре второй бит представляет собой инвертированный первый. Поэтому значения `CycleNumber` 5, 7, 9 и 11 обрабатываются одним участком кода, а значения 6, 8, 10 и 12, другим участком кода, который дает инверсный результат. Этот блок также сдвигает переменную `HouseCode` вправо на один бит, так что в следующем прерывании будет использоваться следующий бит.

Следующие пять пар битов несут ключевой код и обрабатываются точно так же, как и биты домашнего кода.

Так как весь описанный выше процесс должен повторяться, все варианты `case` дублируются со смещением 22.

Когда `CycleNumber` достигает 45, кадр заканчивается. Выход выключается, выключая передатчик. Так как требуются три периода «тишины» между двумя удачными посылками, обрабатываются еще шесть прерываний, которые ничего не делают. На 50-м прерывании `CycleNumber` сбрасывается в ноль, сообщая о том, что передача закончена. Это условие проверяется основной программой, чтобы предотвратить попытку передачи сообщения, когда предыдущее сообщение еще передается. Заметьте, что первым передается младший бит.

Исходный текст следующий:

```
/* Эта функция осуществляет передачу кадра. Она вызывается, когда должен */
/* передаваться элемент кадра, или 2 раза на период сетевого напряжения */
void SendOneFrameElement ( void )
      {
      static Byte TempKeyCode, TempHouseCode ;
      switch ( CycleNumber++ ) /* Инкремент счетчика сразу после проверки */
            {
            case 1 :    /* Первые 3 полупериода должны иметь импульсы
                        (условие старта) */
            case 23 :   /* Старт второй группы */
                  TempKeyCode = KeyCode ;
                  TempHouseCode = HouseCode ;
                  TACR1 |= ( 1 << OLVL1 ) ;     /* Разрешение импульса для
                                                   следующего прерывания */
                  break ;

            case 2 : /* Второй и третий импульсы кода старта (1-я группа) */
             case 3 :
            case 24 : /* Второй и третий импульсы кода старта (2-я группа) */
             case 25 :
                  break ;

            case 4 : /* 4-й полупериод не должен иметь импульса;
                        конец старта */
            case 26 :
                  TACR1 &= ~( 1 << OLVL1 ) ;    /* Запрещение импульса для
                                                   следующего прерывания */
                  break ;

            case 5 :
            case 7 :
            case 9 : /* Передача одного из четырех бит домашнего кода */
            case 11 :
            case 27 :
            case 29 :
            case 31 :
            case 33 :
                  if ( ( TempHouseCode & 1 ) != 0 )
                        TACR1 |= ( 1 << OLVL1 ) ;     /* Разрешение импульса для
                                                         следующего прерывания */
                  else
```

```
      TACR1 &= ~( 1 << OLVL1 ) ;          /* Запрещение импульса для
                                          следующего прерывания */
      break ;

case 6 :
case 8 :
case 10 :   /* Передача одного из четырех инверсных бит
            домашнего кода */
case 12 :
case 28 :
case 30 :
case 32 :
case 34 :
      if ( ( TempHouseCode & 1 ) == 0 )
            TACR1 |= ( 1 << OLVL1 ) ;     /* Разрешение импульса для
                                          следующего прерывания */
      else
      TACR1 &= ~( 1 << OLVL1 ) ;          /* Запрещение импульса для
                                          следующего прерывания */
      TempHouseCode >>= 1 ;   /* Для данного бита закончено.
                              Готовность для следующего бита */
      break ;

case 13 :
case 15 :
case 17 : /* Передача одного из пяти бит ключевого кода */
case 19 :
case 21 :
case 35 :
case 37 :
case 39 :
case 41 :
case 43 :
      if ( ( TempKeyCode & 1 ) != 0 )
            TACR1 |= ( 1 << OLVL1 ) ;     /* Разрешение импульса для
                                          следующего прерывания */
      else
            TACR1 &= ~( 1 << OLVL1 ) ;    /* Запрещение импульса для
                                          следующего прерывания */
      break ;

case 14 :
case 16 :
case 18 :   /* Передача одного из пяти инверсных бит
            ключевого кода */
case 20 :
case 22 :
case 36 :
case 38 :
case 40 :
case 42 :
case 44 :
      if ( ( TempKeyCode & 1 ) == 0 )
            TACR1 |= ( 1 << OLVL1 ) ;     /* Разрешение импульса для
                                          следующего прерывания */
      else
            TACR1 &= ~( 1 << OLVL1 ) ;    /* Запрещение импульса для
                                          следующего прерывания */
      TempKeyCode >>= 1 ;     /* Для данного бита закончено.
                              Готовность для следующего бита */
      break ;

case 45 :
      TACR1 &= ~( 1 << OLVL1 ) ;    /* Запрещение импульса для
                                    следующего прерывания */
      break ;
```

```
            case 46 :   /* Требуются три полных периода тишины */
            case 47 :   /* перед следующей передачей */
            case 48 :
            case 49 :
                  break ;

            case 50 :
                  CycleNumber = 0 ;       /* Сброс счетчика цикла для
                                             завершения потока и разрешения*/
                  break ;                 /* новой передачи. */
            }
      }
```

Основой структуры является оператор `switch`, который передает управление той части кода, которая соответствует текущему номеру передаваемого бита. Этот номер инкрементируется сразу после проверки, в результате он изменяется в диапазоне от 1 до 23. После этого номер устанавливается в ноль. Когда он нулевой, общая подпрограмма обслуживания прерывания больше не вызывает `SendFrameElement`, пока `CycleNumber` не будет снова установлен в 1 функцией `SendCommand`.

```
      switch ( CycleNumber++ )  /* Инкремент счетчика сразу после проверки */
            {
            case 1 :          /* Первые 3 полупериода должны иметь импульсы
                                 (условие старта) */
```

Каждое слово для передачи (домашний код и ключевой код) сдвигается вправо, когда самый правый бит будет проверен. В соответствие с состоянием этого бита, бит OLVL1 в регистре TACR1 устанавливается или сбрасывается для генерации импульса или для его отсутствия. Это делается следующим блоком операторов:

```
      if ( ( TempKeyCode & 1 ) == 0 )
            TACR1 |= ( 1 << OLVL1 ) ;     /* Разрешение импульса для
                                             следующего прерывания */
      else
            TACR1 &= ~( 1 << OLVL1 ) ;    /* Запрещение импульса для
                                             следующего прерывания */
      TempKeyCode >>= 1 ;     /* Для данного бита закончено.
                                 Готовность для следующего бита */
```

**Функция `PhaseLockedLoop`.**

Эта функция измеряет сдвиг фаз между номинальным положением перехода через ноль и началом 1-миллисекундного импульса. В соответствие с обнаруженной разницей, период таймера увеличивается или уменьшается, чтобы сделать эту разницу как можно меньшей. Реально, когда произошел захват, период совершает небольшие колебания вокруг номинального значения частоты сети.

Исходный текст следующий:

```
void PhaseLockedLoop ( void )
      {
      int Position, Deviation ;

      asm
            {
            ld a, TAIC2HR
            ld Position, a
            ld a, TAIC2LR
            ld Position:1, a
            }
      /* Этот ассемблерный текст логически эквивалентен следующему: */
```

```
/* Position = *((int *) &TAIC2HR)   ; считать знаковую величину
                                    из счетчика */
/* Однако порядок чтения двух байт является важным, в то время как */
/* Си не дает никаких гарантий относительно этого. Поэтому */
/* использован ассемблер. */

Deviation = Position - ( TIMER_A_PERIOD - PULSE_LENGTH ) ;
Deviation >>= 4 ; /* Деление на 16 (коэффициент обратной связи). */
if ( Deviation < -PERIOD_DEVIATION )
Deviation = -PERIOD_DEVIATION ; /* Значение не выходит за эти пределы. */
else
      if ( Deviation > PERIOD_DEVIATION )
            Deviation = PERIOD_DEVIATION ;
TimerAPeriod = TIMER_A_PERIOD + Deviation ; /* Изменение теоретического
                                               периода. */
}
```

Захваченная величина считывается с помощью ассемблерной вставки. Это понадобилось потому, что регистр захвата определен как два раздельных 8-разрядных регистра, и байты должны читаться в определенном порядке.

Как сказано в описании ST72251, механизм, предотвращающий рассинхронизацию данных, запрещает операцию захвата при чтении старшего байта до тех пор, пока младший байт тоже не будет считан. Поэтому мы должны быть уверены, что первым читается старший байт, а лишь затем – младший байт. Из-за оптимизации, производимой компилятором, единственным путем, который гарантирует правильный порядок, является использование ассемблера. Поэтому для чтения регистра захвата используется приведенный выше код. Соглашение, принятое для компилятора, определяет, что для 16-разрядной переменной старший байт размещается в адресном пространстве первым, поэтому запись `Position:1` означает младший байт.

Отклонение фазы вычисляется как разница между значением регистра захвата и положением фронта импульса. Это отклонение делится на 16 путем 4-х кратного сдвига вправо (на настоящее деление требуется намного больше времени) для уменьшения петлевого усиления, затем отклонение ограничивается на уровне менее 0.5 % от номинальной частоты. Конечное значение периода получается сложением этой знаковой величины со значением номинального периода. Это значение сохраняется в глобальной переменной `TimerAPeriod`, которая используется в начале подпрограммы обслуживания прерывания по совпадению для перезагрузки регистра сравнения. Это гарантирует, что регистр сравнения обновляется в то время, когда совпадения произойти не может, что исключает проблему возникновения события совпадения между загрузкой двух байтов регистра. Ведь величина периода вычисляется в прерывании по переходу через ноль, которое никак не связано с событиями совпадения, особенно когда система осуществляет поиск синхронизации. Соответственно, в этом прерывании делать перезагрузку регистра сравнения нельзя.

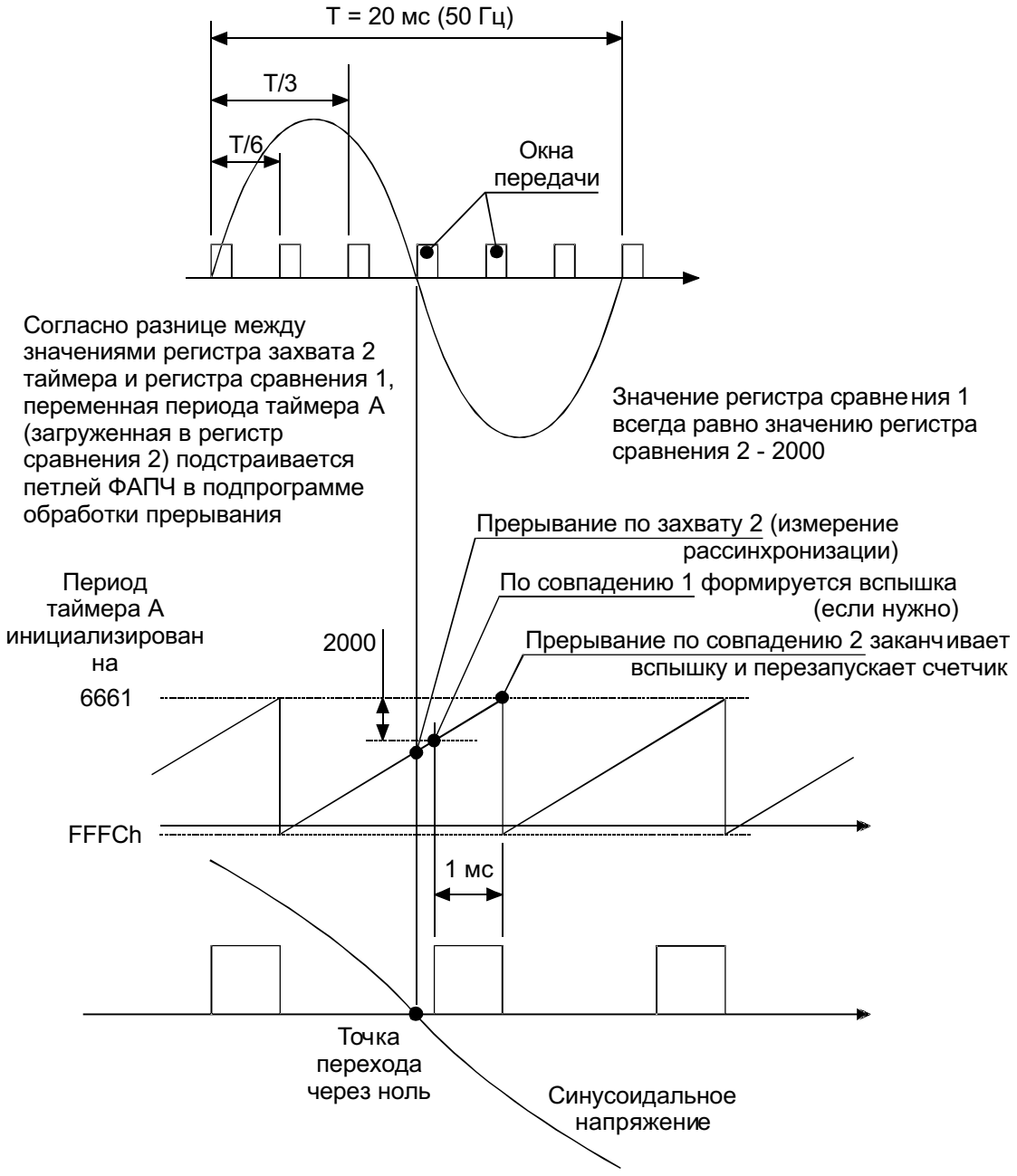
Номинальный период определен в файле заголовка `X10XMIT.H`, как показано ниже:

```
/* #define F60HZ */                 /* Удалите комментарии для 60 Гц. */
#ifdef F60HZ
#define TIMER_A_PERIOD 5555 - 5     /* Номинал для 60 Hz,
                                    в 1/2 микросекунды */
#else
#define TIMER_A_PERIOD 6666 - 5     /* Номинал для 50 Hz,
                                    в 1/2 микросекунды */
#endif
```

Этот исходный текст ориентирован на частоту сети 50 Гц. Для перехода на 60 Гц должен быть задействован первый `#define` путем удаления символов комментария (/* */). Для 50 Гц период таймера должен быть равен 1/6 периода сети, т.е. 3.333 мс. Так как таймер

работает на тактовой частоте 2 МГц, это составляет 6666 тиков таймера. Но когда происходит событие совпадение 2, счетчик сбрасывается на FFFCh, что составляет –5 в дополнительном коде. Поэтому номинальное значение регистра совпадения 2 составляет на самом деле 6666-5, или 6661.

Работа ФАПЧ пояснена на рисунке ниже:




Система дистанционного управления: петля ФАПЧ передатчика

09-pll

### 2.3.3 Подпрограмма обслуживания прерывания по переполнению таймера B

Эта подпрограмма играет две роли: она регенерирует дисплей и производит обратный отсчет времени.

Прерывание возникает при переполнении независимо работающего счетчика. При каждом прерывании состояние бита PA7 меняется на противоположное, поэтому каждая цифра горит в промежутке между двумя прерываниями, т.е. совершается 32 цикла регенерации в секунду.

Так как величина, которую нужно индицировать, лежит в пределах от 0 до 15, управление дисплеем может быть выполнено достаточно просто, используя следующую договоренность:

Все последовательности, которые могут индицироваться, сохранены в виде констант в двумерном массиве, который имеет 15 строк по 2 элемента. Каждая пара соответствует индицируемой величине; каждый элемент пары является одним из тех значений, которые должны попеременно записываться в PA. Таким образом, дисплей сканируется и обновляется, при этом он выглядит как непрерывно горящий дисплей из двух цифр.

Массив констант определен следующим образом:

```
const Byte SevenSegment[16][2] =
      {
      {0xff, 0x84}, {0xff, 0xed}, {0xff, 0xa2}, {0xff, 0xa8}, /* 0-3 */
      {0xff, 0xc9}, {0xff, 0x98}, {0xff, 0x90}, {0xff, 0xad}, /* 4-7 */
      {0xff, 0x80}, {0xff, 0x88}, {0x6d, 0x84}, {0x6d, 0xed}, /* 8-11 */
      {0x6d, 0xa2}, {0x6d, 0xa8}, {0x6d, 0xc9}, {0x6d, 0x98}  /* 12-15 */
      } ;
```

Эта таблица констант расположена в ПЗУ (или в программном сегменте) с помощью модификатора `const` и опции компилятора `-Cc`, добавленной в раздел `Compoptions` файла окружения. Таким же образом определена таблица `Codes[]`, упомянутая выше:

```
const Byte Codes[] =    {6, 7, 4, 5, 8, 9, 10, 11, 14, 15, 12, 13, 0, 1, 2, 3,
                         16, 24, 20, 28, 18, 26, 22, 30, 17, 25, 21, 19,
                         27, 23, 31} ;
```

Обновление дисплея осуществляется очень просто. Для этого служит следующий оператор, где `Delay` содержит число, которое должно индицироваться, а `Second` декрементируется в каждом прерывании:

```
PADR = SevenSegment[Delay][Second & 1] ;  /* Зажечь одну из цифр */
```

Когда величина задержки достигает нуля, дисплей выключается простой записью 0xff. Это же заставляет дисплей мигать раз в секунду:

```
if ( Second > 50 ) /* Дисплей мигает с частотой 1 Гц и малой скважностью */
            PADR = DISPLAY_OFF ;                /* Выключение дисплея. */
      else
            PADR = SevenSegment[Delay][Second & 1] ;  /* Зажечь одну из цифр. */
```

Так как значение Second меняется в диапазоне от 63 до 0, дисплей горит 51/64 времени каждую секунду.

Остаток кода обработчика прерывания просто поддерживает счетчики, которые подсчитывают 64 прерывания для получения секунды, затем 3600 секунд для получения часа, затем от 0 до 15 целых часов. Когда задержка истекает, передается команда CLOSE и дисплей выключается.

Текст является следующим:

```
#pragma TRAP_PROC SAVE_REGS
void DelayCounter ( void )
      {
      WDGR = 0xFF ; /* перезагрузка сторожевого таймера */
      asm
            {
            ld a, TBSR ;
            ld a, TBCLR ; /* пустое чтение для очистки запроса прерывания */
            }
      if ( Delay == 0 )
```

```
            PADR = DISPLAY_OFF ; /* Выключение дисплея. */
      else
            {
            if ( Second > 50 ) /* Дисплей мигает с частотой 1 Гц
                              и малой скважностью. */
                  PADR = DISPLAY_OFF ; /* Выключение дисплея. */
            else
                  PADR = SevenSegment[Delay][Second & 1] ; /* Зажечь
                                                      одну из цифр. */
            if ( Second == 0 )
                  {
                  Second = 63 ; /* Секунда истекла: перезагрузить счетчик. */
                  if ( Hour == 0 )
                        {
                        Hour = 3600 ; /* Час истек: перезагрузить счетчик. */
                        if ( Delay == 1 ) /* Окончился последний час ? */
                              SendCommand ( CLOSE_COMMAND ) ; /* Закрывать. */
                        Delay-- ; /* Декремент; если 1, то установить ноль
                                          и завершить задержку. */
                        }
                  else
                        Hour-- ;
                  }
            else
                  Second-- ;
            }
      }
```

# 3 Приемник

## 3.1 Инструкция по использованию

Несмотря на то, что в доме только один передатчик, приемников может быть несколько, каждый из которых расположен возле тех штор, которыми он управляет. Все приемники одинаковые, все они установлены на тот же домашний код, что и передатчик, и это обеспечивает одновременный прием одних и тех же команд. Поэтому, когда оператор нажимает кнопку ЗАКРЫТЬ на передатчике, все шторы начнут закрываться одновременно.

В дополнение дистанционному управлению, каждая штора имеет местное управление. Для этих целей каждый приемник имеет кнопку, которая позволяет шторе принимать следующие состояния: ОТКРЫТЬ, СТОП, ЗАКРЫТЬ, СТОП, ОТКРЫТЬ... и так далее по кругу.

## 3.2 Электрическая схема

Основными блоками электрической схемы являются:

Источник питания, который обеспечивает дежурное питание схемы

Микроконтроллер, который полностью управляет работой, декодирует кадры X-10 и координирует движение штор

Фильтр и детектор, которые выделяют несущую частоту, подавляя при этом помехи, и формируют логический сигнал, когда обнаруживается несущая

Реле и их ключи, которые включают и выключают двигатель, а также выбирают направление вращения (открывание или закрывание)

При проектировании приемника должны учитываться следующие ограничения:

Он должен быть маленьким, для того, чтобы облегчить его размещение в нужном месте, и иметь приемлемый внешний вид.

Он должен быть дешевым

Эти ограничения препятствуют использованию трансформатора в источнике питания, поскольку он дорогой и громоздкий. Из этих соображений питание обеспечивается прямо от

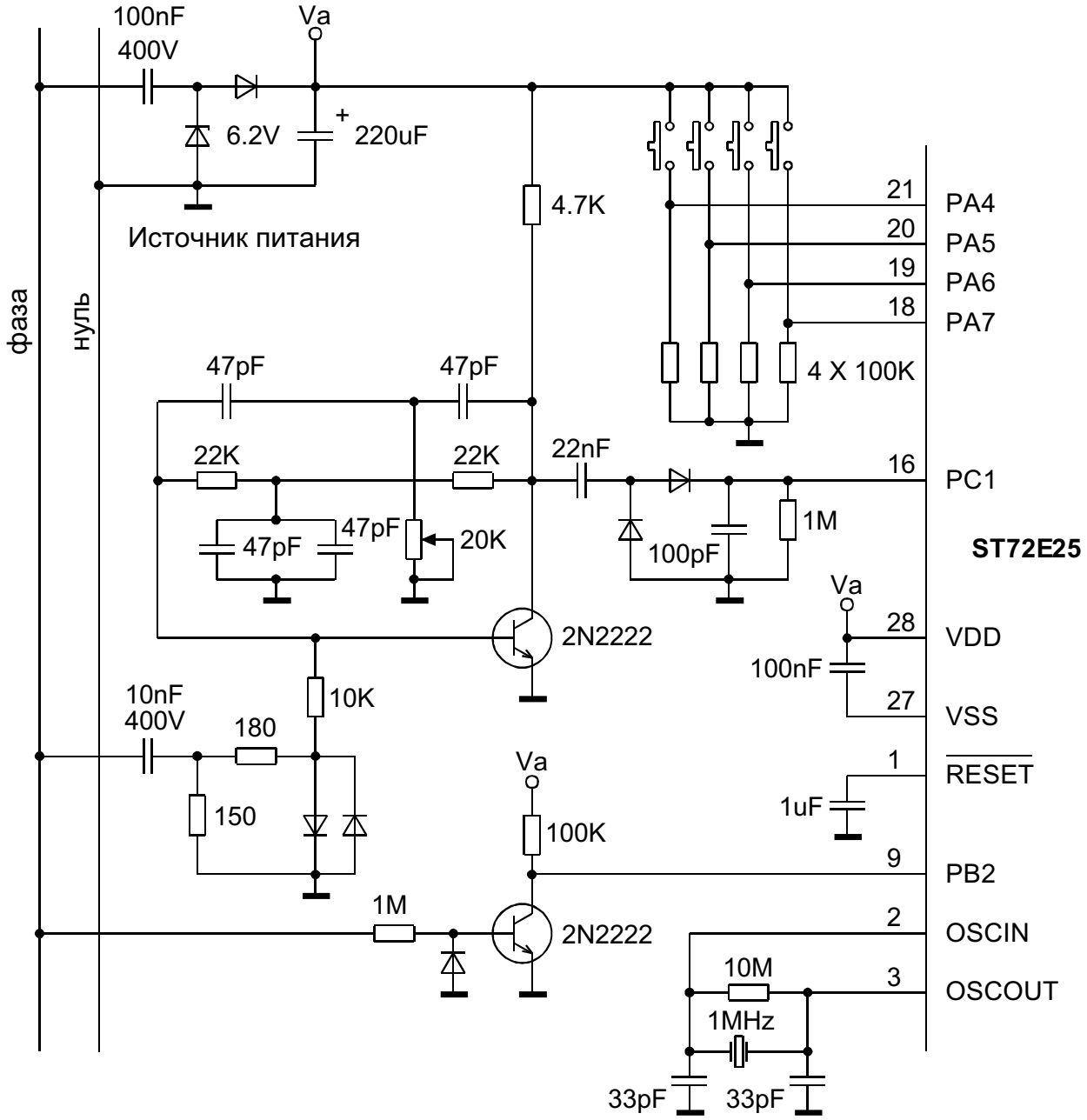
сети через конденсатор. Это дешевый метод, но он не обеспечивает большого тока при сохранении размеров и цены конденсатора в разумных пределах.

Поэтому требуется тщательное продумывание схемы на предмет минимизации потребляемой мощности. В фильтре используется один транзисторный каскад с высокоомным нагрузочным резистором для уменьшения потребления. Реле являются поляризованными, каждое из них имеет две обмотки: одна для включения реле в первое положение, другая – во второе. Во включенном состоянии такое реле не требует постоянного протекания тока в обмотке. Достаточно короткого импульса тока (40 мА в течение 5 мс) для переключения реле. Требуемая энергия запасается в конденсаторе фильтра источника питания, а частота переключений ограничена программно, чтобы позволить конденсатору зарядится между двумя импульсами. При емкости конденсатора 220 мкФ, учитывая, что в большинстве случаев два реле должны переключаться одновременно, это приводит к просадкам напряжения на конденсаторе порядка 1 В. ST72251 допускает достаточный диапазон напряжений питания, поэтому такие просадки вполне допустимы.

ST7 также должен быть использован экономичным образом. Для этого он тактируется кварцем 1 МГц, что обеспечивает внутреннюю тактовую частоту 500 КГц. Так как это КМОП-микросхема, потребление ST7 примерно пропорционально тактовой частоте. Поэтому уменьшение частоты приводит к уменьшению потребляемого тока.

В дополнение, питание АЦП включается лишь тогда, когда это необходимо, а ядро переводится в состояние ожидания каждый раз, когда ему нечего делать. Потребление в этом режиме снижено.

На рисунке ниже показана принципиальная схема приемника:



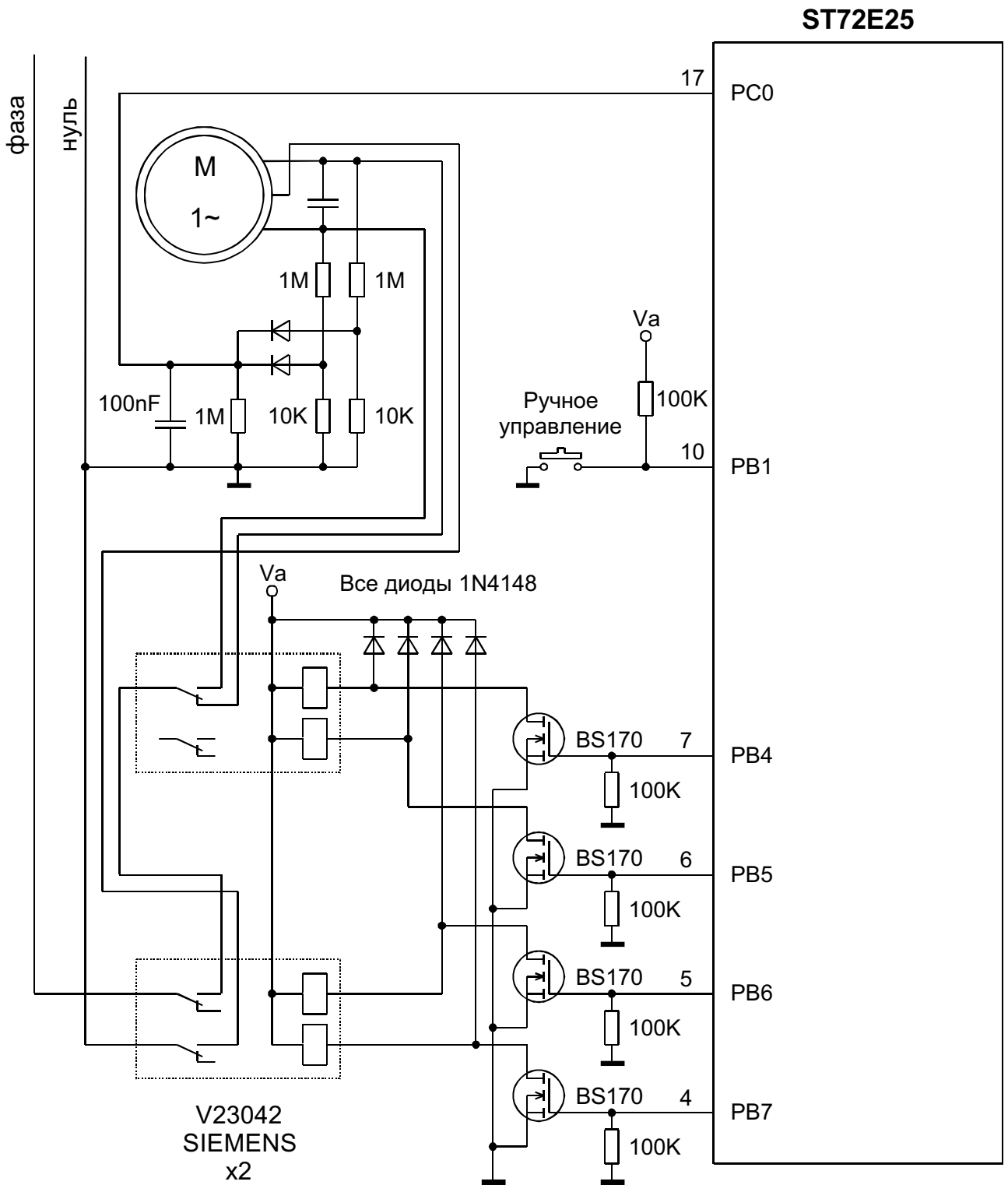

Система дистанционного управления: принципиальная схема приемника
часть 1: входные сигналы

09-xrx-1

Четыре переключателя вверху выбирают домашний код приемника. Приемник выполняет только те команды, которые переданы с его домашним кодом.

Источник питания выглядит как классический умножитель напряжения. На самом деле, входной конденсатор (0.1 мкФ) представляет собой реактивность, которая ограничивает ток на уровне примерно 7 мА (среднеквадратическое значение). Только половина этого тока выпрямляется и используется для заряда фильтрующего конденсатора (220 мкФ). Первый диод одновременно выпрямляет ток и ограничивает напряжение на уровне 6.2 В. Однако из этого напряжения нужно вычесть прямое падение на диоде, в результате получится напряжение порядка 5 В.

Фильтр имеет переменный резистор для регулировки центральной частоты, которую нужно установить равной частоте передатчика.




Система дистанционного управления: принципиальная схема приемника

09-xr-2

Обмотки реле управляются МОП-транзисторами, так как спецификация параллельных портов не позволяет подключить реле непосредственно. Резисторы, соединяющие затворы с землей, гарантируют, что во время включения питания катушки будут обесточены, что предотвращает ложное включение двигателя.

Двигатель имеет обмотку с тремя выводами, между двумя из которых включен фазосдвигающий конденсатор. Схема внизу выпрямляет напряжение, присутствующее на двух выводах двигателя. Получаемое постоянное напряжение находится в пределах 5 В, и

оно подается на вход АЦП. Когда двигатель остановлен, выпрямленное напряжение соответствует сетевому напряжению, которое равно 230 В (перем.). Это дает на выходе выпрямителя напряжение порядка 2.5 В. Когда двигатель вращается, неподключенный вывод имеет намного большее напряжение относительно общего вывода. Это обеспечивает большее напряжение на входе АЦП, поэтому используя соответствующий порог, микроконтроллер может продетектировать, когда двигатель вращается, а когда остановлен с помощью одного из встроенных концевых выключателей.

### 3.3 Программное обеспечение

#### 3.3.1 Функции прерывания

Приемник в основном работает так же, как и передатчик. Таймер А используется для генерации прерываний на каждом полупериоде сетевого напряжения (10 мс при 50 Гц), и он синхронизируется с сетевой частотой, используя фазовую автоподстройку. Параметры несколько отличаются от передатчика, поскольку внутренняя тактовая частота меньше и прерывания должны возникать один раз, а не три раза за полупериод.

Когда возникает прерывание, проверяется бит OCF1 регистра TASR, чтобы узнать, это прерывание по захвату или по переполнению таймера. В последнем случае вызывается функция `ReceiveOneFrameElement`. Текст этой функции приведен ниже:

```
void ReceiveOneFrameElement ( void )
      {
      static Byte TempHouseCode, TempKeyCode, TempHouseCode2, TempKeyCode2,
                                          CarrierDetected ;
      /* Вначале определяем, какое произошло прерывание */
      CarrierDetected = ( ReadADC ( CARRIER_ADC_CHANNEL ) >
                                          CARRIER_THRESHOLD ) ;
      switch ( CycleNumber++ ) /* Инкремент счетчика сразу после проверки */
            {
            case 23 :
                  TempHouseCode2 = TempHouseCode ; /* Сохранить команду,
                                                принятую в первом кадре. */
                  TempKeyCode2 = TempKeyCode ;

            case 1 :
            case 2 :
            case 3 :
            case 24 :
            case 25 :
                  /* 3 первых полупериода должны содержать
                  импульсы (условие старта) */
                  if ( ! CarrierDetected )
                  CycleNumber = 1 ; /* Сброс, если процесс имеет
                                    некорректный заголовок. */
                  break ;

            case 4 : /* 4 полупериода не должны иметь импульсов;
                              конец старта */
            case 26 :
                  if ( CarrierDetected )
                  CycleNumber = 1 ; /* Сброс, если процесс имеет
                                    некорректный заголовок. */
                  break ;

            case 5 : /* Прием одного из четырех бит домашнего кода */
            case 27 :
                  TempHouseCode = 0 ; /* Вначале инициализируем
                                          значение домашнего кода. */
            case 7 :
            case 9 :
            case 11 :
```

```
case 29 :
case 31 :
case 33 :
      TempHouseCode >>= 1 ; /* Сдвиг вправо на один бит */
      if ( CarrierDetected )
            TempHouseCode |= 8 ; /* установка старшего бита */
      break ;                 /* домашний код имеет 4 бита */

case 6 :
case 8 :
case 10 :
case 12 :   /* Проверка того , что следующий четный передаваемый
            бит является инверсным нечетному. */
case 28 :
case 30 :
case 32 :
case 34 :
      if ( ( ( TempHouseCode & 8 ) != 0 ) == CarrierDetected )
      CycleNumber = 1 ;       /* Сброс всего процесса, если
                              неправильный проверочный бит. */
      break ;

case 13 :   /* Прием одного из пяти бит ключевого кода */
case 35 :
      TempKeyCode = 0 ; /* Вначале инициализируем
                        значение ключевого кода. */
case 15 :
case 17 :
case 19 :
case 21 :
case 37 :
case 39 :
case 41 :
case 43 :
      TempKeyCode >>= 1 ;     /* Сдвиг вправо на один бит */
      if ( CarrierDetected )
      TempKeyCode |= 0x10 ;   /* Установка младшего бита */
      break ;                 /* Ключевой код имеет 5 бит */

case 14 :
case 16 :
case 18 :
case 20 :
case 22 :   /* Проверка, что следующий четный передаваемый
            бит является инверсным нечетному. */
case 36 :
case 38 :
case 40 :
case 42 :
case 44 :
      if ( ( ( TempKeyCode & 0x10 ) != 0 ) == CarrierDetected )
            CycleNumber = 1 ;       /* Сброс всего процесса, если
                                    неправильный проверочный бит. */
      break ;

case 45 :
      CycleNumber = 1 ; /* Поток завершен: сброс счетчика цикла */
      /* Проверка дважды принятой команды на идентичность */
      if ( ( TempHouseCode2 == TempHouseCode ) && ( TempKeyCode2
      == TempKeyCode ) )
            {
            HouseCode = TempHouseCode ; /* Сделать принятые данные
                              доступными для основной программы. */
            KeyCode = TempKeyCode ;
            }
      break ;
```

```
        default :
              CycleNumber = 1 ;        /* Сброс всего процесса, если
                                       неправильное количество бит. */
              break ;
        }
}
```

Первое, что делает эта функция, это чтение аналогового напряжения с выхода детектора. Для каждой позиции бита его значение записывается или проверяется на соответствие правилам стандарта X-10. Биты записываются в четвертую или пятую позицию для домашнего кода или ключевого кода соответственно, после сдвига кода на один бит вправо:

```
TempKeyCode >>= 1 ;                  /* Сдвиг вправо на один бит */
      if ( CarrierDetected )
            TempKeyCode |= 0x10 ;    /* Установка младшего бита */
```

В конце приема эти коды проверяются, чтобы выяснить, должен ли приемник что-то с ними делать. Если при проверке выясняется, что какой-то бит является ошибочным, процесс приема прерывается установкой `CycleNumber` обратно в 1:

```
if ( ( ( TempKeyCode & 0x10 ) != 0 ) == CarrierDetected )
      CycleNumber = 1 ;              /* Сброс всего процесса, если
                                     неправильный проверочный бит. */
```

Как постановляет стандарт X-10, каждая команда передается дважды. Приемник спроектирован так, что он должен успешно принять последовательно два кадра. Когда принимается второй кадр, принятые в первом кадре коды сравниваются с кодами, принятыми во втором кадре. Если они совпадают, коды считаются действительными, и они передаются в основную программу для выполнения необходимых действий.

### 3.3.2 Основная программа

Основная программа начинается с нескольких функций инициализации, которые очень схожи с функциями, использованными в передатчике. Потом она выполняет бесконечный цикл, который управляет последовательностью движения штор.

**Управление таймером B**

Таймер B используется для генерации прерывания после определенной задержки, для чего используется регистр сравнения 1. Когда требуется задержка, вызывается функция `WaitDelay`. Эта функция инициализирует таймер B вначале установкой независимо работающего счетчика на FFFCh, затем загрузкой регистра сравнения 1 величиной требуемой задержки (-5 для компенсации сброса на FFFCh). После этого все запросы прерываний очищаются, и разрешается прерывание по совпадению.

Эта функция частично написана на ассемблере для предотвращения побочных эффектов компилятора, связанных с порядком присвоений. Во время ожидания окончания задержки ядро с помощью команды `WFI` переходит в состояние ожидания. Это снижает потребляемую мощность. Так как существуют и другие прерывания, выход из состояния ожидания происходит не обязательно в конце задержки. Для проверки, закончилась ли задержка, команда `WFI` выполняется периодически в цикле, выход из которого осуществляется только тогда, когда условие будет выполнено. Это условие `TimeElapsed` определено в виде макроса:

```
#define TimeElapsed ( TBSR & ( 1 << OCF1 ) )
```

Это выражение действительно только тогда, когда бит OCF1 регистра TBSR установлен.

Бит разрешения прерывания по совпадению одновременно разрешает прерывания по двум событиям: совпадение 1 и совпадение 2. В данном случае необходимо очистить флаг совпадения 2 OCF2 перед разрешением прерываний, потому что каким бы ни было начальное значение TBOC2R, событие совпадение 2 может произойти до переполнения независимо работающего счетчика. Это вызовет возникновение запроса прерывания, так как бит маски прерывания установлен. Для предотвращения этого флаг совпадения 2 очищается, несмотря на то, что это событие не используется.

Полный текст функции приведен ниже:

```
void WaitDelay ( Word Time )
      {
      asm
            {
            clr TBCLR         /* Любая запись в независимо работающий
                              счетчик сбрасывает его на FFFCh. */
            ld a, TBSR        /* Очистка отложенного прерывания
                              по совпадению 2 */
            ld a, TBOC2LR
            ld a, Time        /* Установка времени, старший байт.
                              Здесь сравнения запрещаются. */
            ld TBOC1HR, a
            ld a, TBSR        /* Первый шаг для очистки
                              отложенного прерывания. */
            ld a, Time:1
            ld TBOC1LR, a     /* Запись младшего байта для
                              разрешеня сравнений. */
            bset TBCR1, #OCIE /* Разрешение прерывания по совпадению */
            }
      while ( ! TimeElapsed )
            WaitForInterrupt ; /* Оставаться в режиме ожидания,
                              пока не истечет промежуток времени. */
      }
```

Когда происходит совпадение, оно вызывает прерывание, которое обслуживается следующим кодом:

```
void CoilEndOfPulse ( void )
      {
      COILS = DE_ENERGIZE ;
      TBCR1 &= ~( 1 << OCIE ) ; /* Запретить следующие прерывания. */
      }
```

Эта функция маскирует запросы прерываний, генерируемые таймером B, а также выключает все обмотки реле. Так как эти две функции в основном используются для получения импульсов тока в обмотках реле, выключение обмоток производится прямо в подпрограмме обслуживания прерывания. Как утверждалось выше, для правильного поведения источника питания требуется строго ограничить длительность импульсов тока в обмотках реле необходимым временем, т.е. 5 мс.

**Управление АЦП**

АЦП используется для измерения двух разных напряжений: выходного напряжения детектора несущей и напряжения на обмотках двигателя. Последнее из измерений необходимо для проверки, действительно ли двигатель штор вращается, или он остановлен встроенными концевыми выключателями. Для этого напряжение двух фаз двигателя делится, затем выпрямляется, и результирующее напряжение будет представлять собой большее из двух. Когда двигатель остановлен, напряжение на его выводах равно сетевому (т.е. 230 В перем.); когда он вращается, один из выводов находится под большим напряжением из-за действия фазосдвигающего конденсатора. Эта разница напряжений говорит о том, вращается двигатель, или остановлен.

Для экономии энергии, питание АЦП включается только тогда, когда это необходимо. Поэтому следующая функция вначале загружает регистр управления АЦП номером канала, а также устанавливает бит ADON для включения преобразователя. Первый цикл ожидает, когда преобразование будет завершено; но, как указано в техническом описании, для получения точного результата преобразователю после включения необходимо некоторое время для установления. Поэтому первое значение отбрасывается, и второй цикл ожидает завершения второго преобразования. Полученная величина возвращается:

```
#pragma NO_OVERLAP      /* Эта функция также вызывается из подпрограммы
                        обслуживания прерывания */
Byte ReadADC ( Byte Channel )
      {
      TACR1 &= ~( 1 << ICIE ) ;     /* Запрещение прерываний таймера А для
                                    предотвращения повторного вызова. */
      ADCCSR = Channel | ( 1 << ADON ) ;
      while ( ! ( ADCCSR & ( 1 << COCO ) ) )
      ;           /* Осуществление одного преобразования для установления. */
      ADCCSR = Channel | ( 1 << ADON ) ;
      while ( ! ( ADCCSR & ( 1 << COCO ) ) )
      ;           /* Осуществление точного преобразования. */
      ADCCSR = 0 ; /* Turn off ADC. */
      TACR1 |= ( 1 << ICIE ) ;      /* Разрешение прерываний таймера А. */
      return ADCDR ;
      }
```

Есть специальное замечание, касающееся этой функции. Она вызывается как из основной программы, так и из подпрограммы обслуживания прерывания таймера А через функцию `ReceiveOneFrameElement`. Так как в Hiware Си функции являются нереентерабельными, мы должны позаботиться о том, чтобы она не вызывалась повторно, т.е. в то время, когда она уже вызвана основной программой. Для этих целей бит разрешения прерывания таймера А при входе сбрасывается и устанавливается снова при выходе.

Пожалуйста, обратите внимание на `pragma` перед заголовком функции. По умолчанию компилятор пытается разместить локальные переменные (явные, которые определены в исходном тексте на Си, или неявные, которые используются при преобразовании Си-ассемблер) в сегменте `_OVERLAP` для уменьшения расхода ОЗУ. Интеллектуальный механизм в порядке предотвращения коллизий определяет, какие функции вызываются какими функциями. Однако этот механизм не работает, если функция может быть вызвана подпрограммой обслуживания прерывания, так как прерывания образуют особое дерево вызовов. Коллизии устраняются путем явного указания того, что данная функция не может разделять память с другими функциями, отсюда и `pragma`.

**Управление обмотками реле**

Реле включены таким образом, что одно из них включает и выключает двигатель, а другое выбирает направление вращения (открывание или закрывание).

Как было сказано выше, реле являются поляризованными (или бистабильного типа). Это означает, что для переключения реле из одного состояния в другое, в обмотке требуется лишь короткий импульс тока, что экономит энергию.

Трудность работы с данным типом реле заключается в том, что мы не знаем, в каком положении оно находится, так как в состоянии с выключенными обмотками оно может находиться в обоих положениях. Поэтому для включения двигателя в одном из направлений мы должны сделать следующее:

Подать импульс на реле направления для задания необходимого направления вращения.

Подать импульс на реле включения для его переключения в положение «включено»

Для остановки двигателя мы должны сделать следующее:

Подать импульс на реле включения для его переключения в положение «выключено»

Функция для остановки двигателя очень проста:

```
void StopMotor ( void )
      {
      COILS = STOP ;
      WaitDelay ( COIL_PULSE ) ;
      }
```

Она включает соответствующую обмотку и запускает таймер, как было описано выше. Когда время истекает, ток в обмотке выключается.

Для включения двигателя в одном из направлений, требуется немного больше действий. Для упрощения написания исходного текста на Си и для его лучшей читаемости, была написана общая функция. Она принимает сообщение (всего их возможно четыре) и производит требуемые действия. Сообщение представляет собой значение, определенное с помощью перечисляемого типа:

```
      enum eStates { START_CLOSE, START_OPEN, STOP_CLOSE, STOP_OPEN } ;
```

Функция обрабатывает сообщения согласно их значению:

```
void SwitchMotor ( enum eStates Direction )
      {
      LastDirection = Direction ;
      switch ( Direction )
            {
            case STOP_OPEN:
            case STOP_CLOSE:
                  StopMotor () ;
                  break ;

            case START_OPEN:
                  COILS = OPEN ;
                  WaitDelay ( COIL_PULSE ) ;
                  COILS = START ;
                  WaitDelay ( COIL_PULSE ) ;
                  WaitDelay ( 5 * TENTH_SECOND ) ;
                  break ;

            case START_CLOSE:
                  COILS = CLOSE ;
                  WaitDelay ( COIL_PULSE ) ;
                  COILS = START ;
                  WaitDelay ( COIL_PULSE ) ;
                  WaitDelay ( 5 * TENTH_SECOND ) ;
                  break ;
            }
      }
```

Если сообщение означает «стоп», вызывается функция `StopMotor`. Иначе включается определенная обмотка, после этого запускается таймер на 5 мс. В конце добавляется дополнительная задержка, которая гарантирует, что цикл не будет повторяться слишком часто, предотвращая снижение напряжения питания до опасного предела. Это также позволяет двигателю разогнаться, а схеме детектора вращения двигателя работать правильно.

**Основная функция**

Теперь мы можем рассмотреть основную программу. Она содержит всего несколько строк, так как большая часть обработки выполняется в вызываемых функциях. Это уменьшает размер кода и делает код более понятным для последующих модификаций.

После инициализации программа входит в бесконечный цикл. Каждый раз, когда принимается команда, которая требует открывания или закрывания штор, двигатель начинает вращаться в определенном направлении. Если нажимается кнопка, текущее состояние заменяется другим по циклу: ОТКРЫТЬ, СТОП, ЗАКРЫТЬ, СТОП, ОТКРЫТЬ...

В конечном счете, если текущим состоянием является START_OPEN или START_CLOSE, и обнаруживается остановка двигателя, это означает, что движение закончено и сработал концевой выключатель. Тогда состояние изменяется на соответствующее состояние STOP_, и система готова начать движение в обратном направлении, когда в следующий раз будет нажата кнопка.

Текст основной программы следующий:

```
void main ( void )
      {
      InitPorts () ;
      InitTimerA() ;
      InitTimerB() ;
      MISCR = 0x0 ;     /* Нормальный режим (clock/2). */
      EnableInterrupts ;
      LastDirection = STOP_OPEN ; /* для правильного начала ручного цикла */
      StopMotor () ; /* Чтобы гарантировать выключенное состояние двигателя
                        при включении питания */
      while (1)
            {
            if ( HOUSE_CODE == Codes[HouseCode] )
                  {
                  switch ( Codes[KeyCode] )
                        {     /* Если код верный, запитать одну из обмоток. */
                              /* Она будет выключена в прерывании таймера. */
                        case CLOSE_COMMAND :
                        SwitchMotor ( START_CLOSE ) ;
                        break ;

                        case OPEN_COMMAND :
                        SwitchMotor ( START_OPEN ) ;
                        break ;
                        }
                  KeyCode = NO_COMMAND ;    /* Стереть последовательность. */
                  HouseCode = NO_COMMAND ;  /* Стереть последовательность. */
                  }
            if ( ButtonPressed () )
                        /* Проверить кнопку. Она имет приоритет перед
                        дистанционными командами. */
                  switch ( LastDirection )
                        {
                        case START_OPEN :
                        SwitchMotor ( STOP_OPEN ) ;
                        break ;

                        case STOP_OPEN :
                        SwitchMotor ( START_CLOSE ) ;
                        break ;

                        case START_CLOSE :
                        SwitchMotor ( STOP_CLOSE ) ;
                        break ;
```

```
                        case STOP_CLOSE :
                        SwitchMotor ( START_OPEN ) ;
                        break ;
                        }
            /* Обновить состояние, если сработал концевой выключатель. */
                  switch ( LastDirection )
                        {
                        case START_OPEN :
                        if ( ReadADC ( MOTOR_ADC_CHANNEL ) < RUNNING_VOLTAGE )
                        SwitchMotor ( STOP_OPEN ) ;
                        break ;

                        case START_CLOSE :
                        if ( ReadADC ( MOTOR_ADC_CHANNEL ) < RUNNING_VOLTAGE )
                        SwitchMotor ( STOP_CLOSE ) ;
                        break ;
                        }
            }
      }
```

## 4 Заключение

Это первое приложение было разработано для акцентирования следующих возможностей:

Использование языка Си в малых системах

Разнообразное использование таймеров

Использование АЦП

Использование возможностей, связанных с уменьшением энергопотребления, как аппаратных, так и программных

Программа на Си, использованная здесь, показывает, что Си не является тем языком, который требует мощной аппаратной части и больших вложений. Наоборот, имея небольшую практику, только небольшие фрагменты программы Вы будете писать на ассемблере, так как увидите, что писать на Си значительно легче. Если программа на Си адекватно откомментирована и структурирована, она значительно легче читается и поддается модификациям, чем программа на ассемблере. Вы также увидите, что любые части программы, которые все же требуют применения ассемблера, могут быть с легкостью написаны внутри исходного текста на Си, без необходимости создания отдельных файлов на языке ассемблера.

Таймера были использованы в режиме ШИМ с захватом и в независимом режиме со сравнением. Это только некоторые способы применения таймеров, но, тем не менее, заслуживающие внимания. На самом деле, приложение могло быть реализовано менее сложным путем, что главным образом относится к схеме фазовой автоподстройки; зато это приложение представляет собой хороший пример демонстрации возможностей, имеющихся даже в таком дешевом микроконтроллере, как 72251. Данная схема добавляет живучести в работу передатчика и приемника без увеличения стоимости, за исключением нескольких дополнительных строк программы. Это как раз может составлять разницу между хорошим и плохим изделием, тем более что стоимость аппаратной части не увеличивается.

АЦП используется полностью, так как он не имеет множества режимов; его возможность выключения способствует решению проблемы снижения потребляемой мощности.

Энергопотребление ST7, низкое изначально, может быть еще снижено путем уменьшения тактовой частоты, насколько это возможно с точки зрения необходимой вычислительной мощности. С помощью команды ожидания, которая останавливает ядро, и функции выключения АЦП, достигнуто общее среднее потребление менее 3 ма, даже вместе с реле, каждое из которых потребляют 40 ма в течение короткого промежутка времени.