Севастопольский национальный технический
университет

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ

**Методические указания
к выполнению лабораторных работ по дисциплине
«Проектирование микропроцессорных систем» для студентов
дневной и заочной формы обучения направления 6.050102—
“Компьютерная инженерия”**

Часть 2

**Севастополь
2014**

УДК 004.2:004.3:004.4'23

Микропроцессорные системы: Методические указания к выполнению лабораторных работ по дисциплине «Проектирование микропроцессорных систем» для студентов дневной и заочной формы обучения направления 6.050102— “Компьютерная инженерия”. Часть 2./ Сост. Тарасова А.В., Бобылев С.Н., Волкова Т.В. — Севастополь: Изд-во СевНТУ, 2014. — 48с.

Целью методических указаний является оказание помощи студентам в подготовке и выполнению цикла лабораторных работ по дисциплине «Проектирование микропроцессорных систем». Во второй части методических указаний рассматриваются лабораторные работы, целью которых является изучение принципов реализации временных интервалов на базе микроконтроллеров ATMEL AVR, в том числе, с использованием таймеров-счетчиков, а также изучение системы встроенных интерфейсов на примере работы с интерфейсами SPI, USART (RS-232) и TWI. Указания предназначены для студентов дневной и заочной форм обучения направления 6.050102 — «Компьютерная инженерия».

Методические указания рассмотрены и утверждены на заседании кафедры Кибернетики и вычислительной техники (протокол № 11 от 25 июня 2014 г.).

Допущено учебно-методическим центром и научно-методическим советом СевНТУ в качестве методических указаний.

Рецензент: В.С. Чернега, канд. техн. наук, доцент

СОДЕРЖАНИЕ

1. ЛАБОРАТОРНАЯ РАБОТА №5

Исследование способов реализации временных интервалов в микроконтроллерах семейства AVR

1. Цель работы

Исследовать методы реализации временных интервалов в микроконтроллерах семейства AVR, ознакомиться с системой прерываний микроконтроллеров семейства AVR, рассмотреть наиболее распространенные задачи, требующие реализации функций времени и обработки прерываний, получить навыки программирования микроконтроллеров. Изучить алгоритмы выполнения простых преобразований (масштабирование чисел, преобразование чисел из двоичной системы в десятичную).

2. Теоретическое введение

2.1. Метод программных циклов. Временная задержка малой длительности

Предположим, что управляющая программа должна инвертировать слово на порте B через интервалы времени, кратные 100 мкс. Фрагмент программы, реализующий временную задержку заданной длительности (100 мкс), требуется оформить в виде подпрограммы. Предполагается, что основная программа будет производить к этой подпрограмме многократные обращения для формирования на выводах порта B импульсных сигналов, длительность которых кратна 100 мкс.

Блок-схема реализации временной задержки приведена на рисунке 1.1.

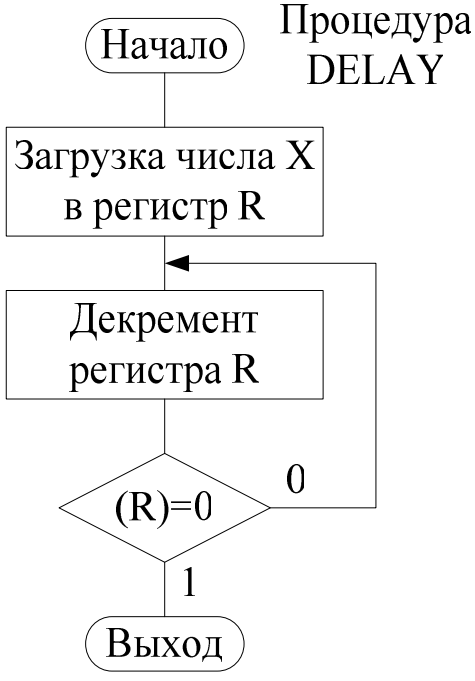


Рисунок 1.1 - Блок-схема алгоритма реализации временной задержки малой длительности

Фрагмент программы для микроконтроллера AT90CAN128:

```
;-----------------------------------
LDI  R17,0x02  ;инициализация
OUT  SPH,R17   ;старшего и
CLR  R17       ;младшего байтов
OUT  SPL, R17  ;указателя стека
SER  R17
OUT  DDRB, R17
M1:       COM  R17
          OUT  PORTB
          RCALL DELAY ; 3 машинных цикла
          RJMP M1
;-----------------------------------
DELAY:    LDI  R01, X; 1 машинный цикл
COUNT:    DEC  R01; 1 машинный цикл
          BRNE COUNT; 1 м.ц., если Z=1
                    ; 2 м.ц., если Z=0
```

```
                    ; Z – флаг нуля
          RET; 4 машинных цикла
;-----------------------------------
```

Произведем расчет числа X — начального значения регистра R1.

Реализуемое время задержки в соответствии с данными о времени (количестве машинных циклов) выполнения команд вычисляется по формуле (1.1).

$$Тз = (3 + 1 + (X - 1)(1 + 2) + (1 + 1) + 4) \cdot t_{мц}, \qquad (1.1)$$

где 3 — время выполнения команды RCALL (в машинных циклах), 1 — время выполнения команды LDI , (X-1)(1+2) — время длительности (X – 1) проходов цикла реализации задержки, (1+1) — время последнего прохода цикла реализации задержки, 4 — время выполнения команды RET.

Время машинного цикла для рассматриваемого микроконтроллера равно одному периоду базовой тактовой частоты. Если микроконтроллер работает на частоте 8 МГц, то время машинного цикла соответствует (1.2).

$$t_{мц} = 1/(8 \cdot 10^6) = 0{,}125 \cdot 10^{-6} = 0{,}125(мкс). \qquad (1.2)$$

Тогда для реализации задержки 100 мкс необходимо решить следующее уравнение: $0{,}125 \cdot (3X + 7) = 100$.

Находим X=264,333. Получили дробное число, следовательно, значение задержки будет приблизительным (1.3).

$$0{,}125 \cdot (3 \cdot 264 + 7) = 99{,}875(мкс). \qquad (1.3)$$

Для уточнения задержки можно ввести в подпрограмму требуемое количество команд NOP (данная команда выполняется 1 машинный цикл, т.е. 0,125 мкс). Тогда окончательно фрагмент программы будет выглядеть так:

```
;-----------------------------------
RCALL     DELAY
          …
;-----------------------------------
DELAY:    LDI  R01, 0x78 ; 120 в шестнадцатеричной системе
                         ;счисления
COUNT:    DEC  R01
          BRNE COUNT
          NOP
          RET
;-----------------------------------
```

Реализуемое значение задержки при этом будет равно (1.4).

$$Тз = 99{,}875 + 0{,}125 = 100(мкс). \qquad (1.4)$$

Для более точной реализации заданного временного интервала в приведенной выше программе желательно учесть длительность команд <COM R17> и <OUT PORTB> (в приведенном расчете она не учитывалась).

Минимальное время задержки, реализуемой приведенной выше подпрограммой DELAY, составляет 1,375 мкс (X=1). Максимальная длительность задержки, реализуемой подпрограммой DELAY составляет 96,625 мкс (X=255).

Для увеличения длительности задержки можно увеличить тело цикла за счет введения дополнительных команд (NOP) или использовать метод вложенный циклов.

2.2. Увеличение временной задержки методом вложенных циклов

Блок-схема алгоритма реализации задержки приведена на рисунке 1.2.

Соответствующий фрагмент программы:

```
     ...
RCALL     DELAY
     ...
;-----------------------------------
DELAY:    LDI  R01, X
M1:       LDI  R02, Y
M2:       DEC  R02
          BRNE M2
          DEC  R01
     BRNE M1
          RET
;-----------------------------------
```

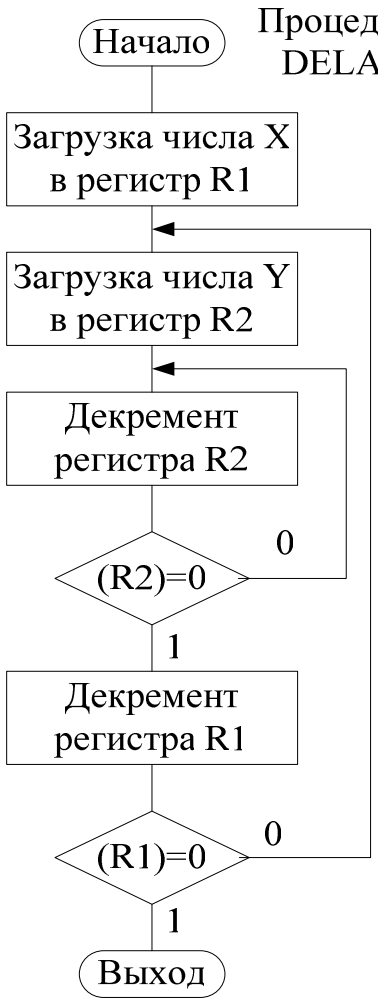


Рисунок 1.2 - Блок-схема алгоритма реализации временной задержки методом вложенных циклов

Пусть необходимо реализовать задержку 2мс=2000мкс (1.5).

Тз=0,125(3+1+X(1+Y(1+2)-1*+1+2)-1*+4)=2000, (1.5)

где 1* — машинный цикл, вычитаемый из-за того, что время последнего исполнения команды BRNE равно одному машинному циклу вместо двух, так как в случае равенства счетчика цикла нулю осуществляется просто переход на следующий адрес (без загрузки в счетчик команд адреса перехода).

Теперь нужно подобрать такие значения X и Y, чтобы равенство (1.5) выполнялось (1.6).

$$0{,}125(7+X(3Y+3))=2000. \qquad (1.6)$$

Пусть Y=100, тогда 0,125(7+303X)=2000; X=52.

Проверяем значение реализуемой задержки (1.7).

$$0{,}125 \cdot (7 + 303 \cdot 52) = 1970{,}375(мкс) \qquad (1.7)$$

2000 - 1972,609=29,625 (мкс) — необходима точная подстройка, например:

```
;-----------------------------------
RCALL     DELAY
;-----------------------------------
DELAY:    LDI  R01, 0x18     ; 24 в шестнадцатеричной системе
M1:  LDI  R02, 0x64          ; 100 в шестнадцатеричной системе
M2:  DEC  R02
     BRNE M2
     DEC  R01
     BRNE M1
; начало участка точной подстройки
LDI  R03,0x4E; 78 в шестнадцатеричной системе
M3:  DEC  R03
```

```
     BRNE M3
     NOP
     NOP
     NOP
; конец участка точной подстройки
     RET
;-----------------------------------
```

Длительность участка точной подстройки равна (1.8).

$$0{,}125(1+78(1+2)-1+1+1+1)=29{,}625 \text{ (мкс)}. \quad (1.8)$$

Естественно, данное решение не является единственным. Возможно, при выборе других начальных значений счетчиков цикла можно получить более точное решение, если это необходимо.

Задержки большей длительности можно реализовать многократным вызовом подпрограммы DELAY. Например, задержку в 1с можно реализовать десятикратным вызовом подпрограммы, реализующей задержку в 100мс:

```
ONESEQ:  LDI  R04, 0x0A     ; 0,125мкс
LOOP:    RCALL DELAY        ; 100мс
         DEC  R04           ; 0,125мкс
         BRNE LOOP          ; 0,542/0,125мкс
```

Погрешность: 0,125(1+10(1+2)-1)=3,75мкс на 1 с (например, за каждую секунду часы будут убегать вперед всего на 3,75 мкс, т.е. меньше, чем на 1с в сутки, что вполне допустимо).

2.3. Формирование временной задержки с использованием таймеров и запретом прерываний

Недостаток метода программных циклов — нерациональное использование ресурсов микроконтроллера: во время формирования временной задержки микроконтроллер не может выполнять никаких других действий, т.е. фактически простаивает.

Можно реализовать задержки с использованием таймеров, т.е. программно-аппаратным методом. Для формирования временной задержки будем использовать восьмиразрядный таймер T0, имеющийся во всех моделях микроконтроллеров AVR и специально предназначенный для подсчета и измерения временных интервалов или для подсчета внешних событий. На другие таймеры в микропроцессорных системах могут быть возложены альтернативные функции, например, генерация сигнала с ШИМ. Описание таймеров приведено в [1, 2, 3, 4].

Если использовать таймер/счетчик T0 в режиме СК/1024, который задается установкой комбинации 101 в битах CS02, CS01, CS00 регистра управления таймером TCCR0, то частота синхросерии, инкрементирующей счетчик, будет в 1024 раза меньше тактовой частоты микроконтроллера СК.

Если базовая частота микроконтроллера 8 МГц, то счетчик будет инкрементироваться через каждые $1/(8*10^6/1024)=128*10^{-6}$ (с)=128 мкс.

Следовательно, в описанном случае можно получить задержки в диапазоне от 128 мкс до 128*255=32640 мкс.

Изменяя базовую частоту микроконтроллера и источник тактового сигнала для T0 (СК, СК/8, СК/64, СК/256) этот диапазон можно варьировать.

Таймер программируется на формирование заданной временной задержки

меньшей, чем максимальная, путем занесения в его счетный регистр TCNT0 соответствующего начального значения. Появление сигнала прерывания от таймера соответствует истечению временного интервала.

Этот вариант реализации основан на установке запрета прерываний от таймера и анализе флага прерывания от таймера. В этом случае указанный выше недостаток не снимается.

Приведенная ниже программа осуществляет инкремент регистра R1 и выдачу его значения в порт B через заданный промежуток времени.

```
.include "can128def.inc"
jmp RESET
.equ X0=0xF9  ;начальное значение для регистра TCNT0
.cseg
.org 0x32
RESET:
; инициализация стека
     ldi       r16,low(ramend)
     out       SPL,r16
     ldi       r16,high(ramend)
     out       SPH,r16
;-----------------------------------
     ser       R17
     out       DDRB, R17

;запретить прерывания от таймера T0 (TIMSK0(1):=0)

     lds       R16, TIMSK0
     andi      R16, 0xFD
     sts       TIMSK0, R16

; старт T0 в режиме СК/1024 (TCCR0(2:0):=101)

     in        R16, TCCR0A
     ori       R16, 0x05
     andi      R16, 0xFD
     out       TCCR0A, R16

; загрузка начального значения в счетный регистр таймера T0
M1:
     ldi       R16, X0
     out       TCNT0, R16

; ожидание переполнения от таймера T0 (цикл пока TIFR0(1)=0)
M2:
     in        R16, TIFR0
     andi      R16, 0x02
     brbs      1, M2     ; переход на M2, если флаг нуля Z
                         ;установлен (SREG(1)=1)
; флаг прерывания от таймера установился в 1
; сброс флага прерывания от таймера T0 (TIFR(1):=0)
; осуществляется путем записи 1 в бит TIFR(1)
```

```
in        R16, TIFR0
ori       R16, 0x02
out       TIFR0, R16
com       r17             ; инвертирование регистра R17
out       PORTB, r17      ; и вывод в порт B
rjmp M1
```

Допустим, необходимо реализовать задержку 30000мкс. Тогда значение X=(32640-30000)/128≈21
Проверка: (255-21)*128=29952 (мкс)
(255-20)*128=30080 (мкс)

2.4. Формирование временной задержки с использованием таймеров и разрешением прерываний

Другой вариант реализации программы основан на обработке прерывания от таймера. Микроконтроллер последовательно выполняет команды основной программы, но через заданный промежуток времени управление автоматически передается процедуре обработки прерывания от таймера T0. Достоинством метода является реализация задержки аппаратно (в фоновом режиме), в то время как микроконтроллер выполняет другие действия.

2.4.1. Система прерываний микроконтроллеров семейства AVR

Прерывание прекращает нормальный ход программы для выполнения приоритетной задачи, определяемой внутренним или внешним событием микроконтроллера. При возникновении прерывания микроконтроллер сохраняет в стеке содержимое счетчика команд PC и загружает в него адрес соответствующего вектора прерывания. По этому адресу должна находиться команда относительного перехода к подпрограмме обработки прерывания. Кроме того, последней командой подпрограммы обработки прерывания должна быть команда RETI, которая обеспечивает возврат в основную программу и восстановление предварительно сохраненного счетчика команд.

Поскольку источниками прерываний являются различные периферийные устройства микроконтроллеров, количество прерываний зависит от конкретной модели [1, 2, 3, 4].

Микроконтроллеры AVR имеют многоуровневую систему приоритетных прерываний. Младшие адреса памяти программ отведены под таблицу векторов прерывания. Каждому прерыванию соответствует свой адрес в этой таблице, и именно этот адрес загружается в счетчик команд при возникновении прерывания. Положение вектора в таблице определяет также и приоритет соответствующего прерывания: чем меньше адрес, тем выше приоритет прерывания. Размер таблицы зависит от модели микроконтроллера. Если в программе прерывания никогда не используются, то на месте таблицы векторов прерываний может быть размещена основная программа.

Для разрешения прерываний флаг I регистра SREG должен быть установлен в «1». Разрешение или запрещение (маскирование) отдельных прерываний произво-

дится установкой или сбросом соответствующих разрядов регистров масок прерываний.

Обработка прерываний осуществляется следующим образом:

1. При выполнении условий, необходимых для генерации прерывания, соответствующий этому прерыванию флаг устанавливается в «1», а флаг I аппаратно сбрасывается, запрещая тем самым обработку следующих прерываний. Однако в подпрограмме обработки прерывания этот флаг можно будет установить в «1» для разрешения вложенных прерываний.

2. Если прерывание разрешено (флаг разрешения прерывания установлен), в счетчик команд загружается адрес вектора соответствующего прерывания. При этом флаг прерывания аппаратно сбрасывается. Ряд флагов прерываний может быть также сброшен записью лог. «1» в I разряд регистра, соответствующий флагу. Если же прерывание запрещено (флаг разрешения прерывания сброшен), флаг прерывания остается в состоянии лог. «1» до разрешения прерывания (в этом случае он будет сброшен аппаратно), либо до программного сброса этого флага.

3. Выполняется подпрограмма обработки прерывания.

4. Выполняется команда возврата из прерывания RETI, при этом флаг I аппаратно устанавливается в «1», разрешая обработку последующих прерываний.

5. Центральный процессор автоматически восстанавливает содержимое счетчика команд. Затем основная программа продолжает свое выполнение с того места, где она была прервана.

При вызове подпрограмм обработки прерываний содержимое регистра состояния SREG не сохраняется. Поэтому пользователь должен самостоятельно запоминать содержимое этого регистра при входе в подпрограмму обработки прерывания (если это необходимо) и восстанавливать его значение перед вызовом команды RETI.

Следует помнить, что для прерываний, вызванных статическими событиями (например, для прерывания, генерируемого при равенстве содержимого счетного регистра и регистра сравнения таймера), флаг прерывания устанавливается только в момент возникновения события. Если флаг прерывания сброшен, а условия генерации прерывания присутствуют, флаг будет установлен только в момент возникновения следующего события.

С другой стороны, для внешних прерываний, генерируемых по уровню, флаги не предусмотрены, поэтому информация об этих прерываниях будет храниться до тех пор, пока присутствует событие, вызывающее прерывание.

Микроконтроллеры AVR поддерживают очередь прерываний. Она работает следующим образом: если условия генерации одного или более прерываний возникают в то время, когда флаг общего разрешения прерываний сброшен (все прерывания запрещены), соответствующие флаги устанавливаются в «1» и остаются в этом состоянии до установки флага общего разрешения прерываний. После разрешения прерываний выполняется их обработка в соответствии с приоритетом.

Время отклика для любого прерывания составляет не менее 4 машинных циклов. В течение первых двух машинных циклов происходит сохранение счетчика команд в стеке, а в течение следующих двух циклов выполняется команда перехода к подпрограмме обработки прерывания. Причем если прерывание произойдет во время выполнения команды, длящейся несколько циклов, то генерация прерывания

произойдет только после выполнения этой команды. Под временем отклика здесь понимается время, прошедшее от наступления события (от установки флага прерывания) до выполнения первой команды подпрограммы обработки прерывания.

Возврат в основную программу также занимает 4 машинных цикла. После выхода из прерывания процессор всегда выполняет одну команду основной программы, прежде чем обслужить любое отложенное прерывание.

2.4.2. Обработка прерываний от таймера T0

В качестве примера приводится программа, реализующая заданный временной интервал с помощью обработки прерывания от таймера T0.

```
.include "can128def.inc"
jmp RESET
.equ X0=0xF9
.cseg
.org 0x32
RESET:
; Инициализация программного стека AT90CAN128 – занесение
; в указатель стека значения верхнего адреса ОЗУ (после подачи
; напряжения питания или сброса указатель стека  равен нулю).
; В данной программе указатель стека инициализируется значением
; адреса внутреннего ОЗУ – $025F
    ldi     R16,0x00
    out     SPL,R16
    ldi     R16,0x5F
    out     SPH,R16
;---------------------------------
    ser     R17
    out     DDRB, R17
; общее разрешение прерываний (SREG(7):=1)
    in      R16, SREG
    ori     R16, 0x80
    out     SREG,R16
; загрузка начального значения в счетный регистр таймера T0
    ldi     R16, X0
    out     TCNT0, r16
; разрешение прерываний от таймера T0 (TIMSK(1):=1)
    lds     R16, TIMSK0
    ori     R16, 0x02
    sts     TIMSK0, R16
; старт T0 в режиме CK/1024 (TCCR0(2:0):=101)
    in      R16, TCCR0A
    ori     R16, 0x01
    andi R16, 0xFD
    out     TCCR0A, R16
; следующие операторы основной программы
; в этой части должен присутствовать бесконечный цикл
; конец основной программы
M1:
   nop
```

```
      jmp    M1
; End of program
;-----------------------------------
.org 0x0020       ; вектор прерывания от таймера
     jmp    TIMER0 ; переход на процедуру обработки прерывания

.org 0x0022       ;
     jmp    TIMER0 ;
;-----------------------------------
.org 0x200
     TIMER0:
               in   R1, SREG ; сохранение регистра состояния
               com  R17          ; инвертирование регистра R17
               out PORTB, R17    ; и вывод в порт B
; загрузка начального значения в счетный регистр таймера T0
               ldi R16, X0
               out TCNT0, R16
               out  SREG, R1 ; восстановление регистра состояния
     RETI                    ; возврат из прерывания
```

2.5. Алгоритмы выполнения простых преобразований. Масштабирование

При вводе и выводе информации часто возникает задача согласования диапазонов представления чисел в микроконтроллере (МК) и во внешней аппаратуре, обеспечивающей связь МК с объектом управления. Эта задача называется масштабированием и сводится к операции умножения/деления числа на некоторую константу k (не обязательно целую).

Если в системе команд МК отсутствуют команды умножения и деления для масштабирования можно использовать способ умножения числа на константу с помощью операции сдвига.

Например, нужно получить произведение числа X на константу k=3,75.

$$X \cdot k = 2X + X + \frac{1}{2}X + \frac{1}{2^2}X,$$

где 2X – сдвиг X влево на 1 разряд,

$\frac{1}{2}X$ - сдвиг вправо на 1 разряд,

$\frac{1}{2^2}X$ - сдвиг вправо на два разряда.

Если константу k нельзя точно представить в виде суммы степеней двойки (например, k=0,66), то она может быть переведена в двоичную систему с округлением.

При переводе числа в другую систему счисления целая часть числа делится на основание системы счисления, а дробная – умножается.

Если необходимо выполнить деление на константу k, то можно заменить эту операцию умножением на константу m=1/k.

2.6. Алгоритмы выполнения простых преобразований. Перевод чисел из двоичной в десятичную систему счисления.

Подразумевается, что десятичные числа представляются в двоично-десятичном коде. Осуществить указанную операцию можно путем деления исходного числа, представленного в двоичной системе счисления на 1010 (10 в двоичной системе).

Если в системе команд МК отсутствует команда деления, то используется метод двух счетчиков (более простой, но и более медленный): из исходного кода вычитается, а к результирующему коду прибавляется по единице до обнуления исходного кода. Вычитание осуществляется в «старой», а прибавление в «новой» системе счисления.

Алгоритм метода представлен на рисунке 1.3. В регистр R1 заносится код двоичного числа X (предполагается, что X≥99), которое необходимо перевести в двоично-десятичный код. Результат перевода формируется в регистре R2.

3. Описание лабораторной установки

В ходе выполнения лабораторной работы текст программы необходимо отладить вначале в среде AVR-Studio, а затем — на стенде CAN128. Проверка соответствия получаемой задержки заданной по варианту задания осуществляется путем вывода получаемого сигнала на осциллограф. Для этого измерительный щуп осциллографа подключаются к тому порту на стенде CAN128, на который выводится, согласно программе, получаемая задержка, а общий – к любому выводу «земля» на стенде. После настройки осциллографа на экране должен отобразиться сигнал прямоугольной формы, параметры которого можно измерить с помощью координатной сетки.

4. Задание исследование

1. Разработать 4 варианта программы, выдающей на один из выводов порта A синхросерию с заданными параметрами:

1.1. Программа на основе метода программных циклов на языке ассемблера AVR.

1.2. Программа, построенная на ожидании изменения флага переполнения таймера на языке ассемблера AVR.

1.3 Программа на языке C [5], использующая библиотеку задержек avr/delay.h [5].

1.4. Программа, построенная на ожидании изменения флага переполнения таймера на языке C.

Вид синхросерии представлен на рисунке 1.4. Значения параметров для конкретного варианта представлены в таблице 1.1.

2. Скомпилировать и проверить правильность работы программы в режиме эмулятора AVR-Studio (для каждой из четырех разработанных версий программы).

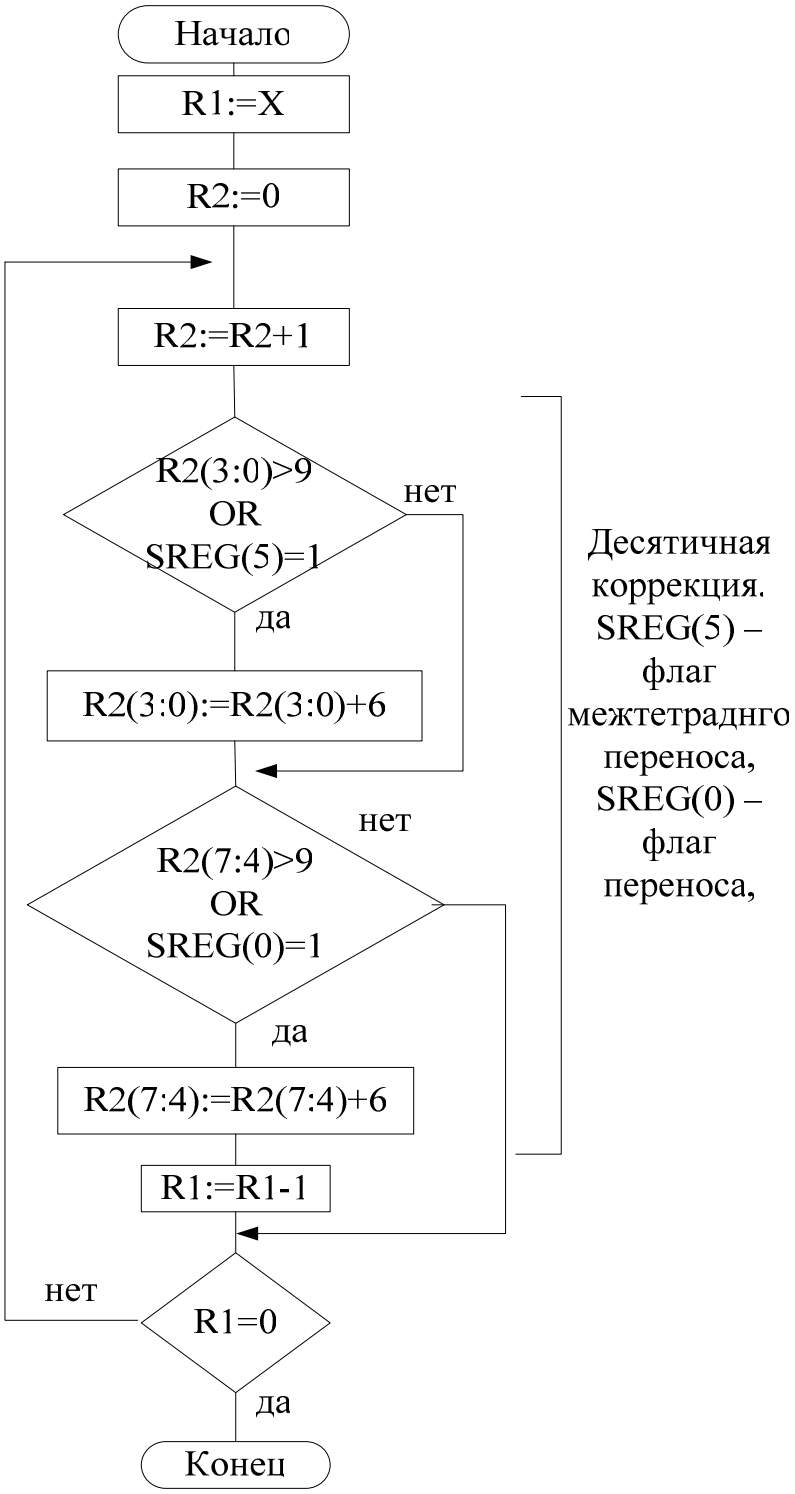


Рисунок 1.3 — Схема перевода чисел из двоичной в десятичную систему счисления

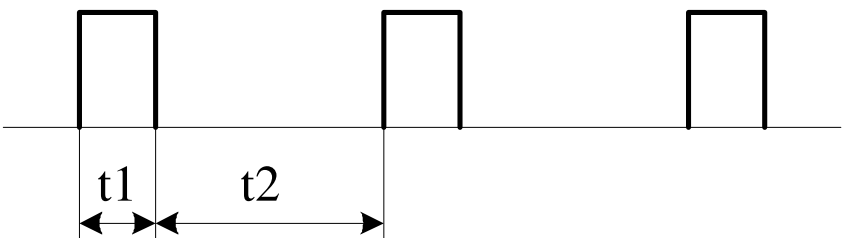


Рисунок 1.4 — Форма импульсной последовательности

Таблица 1.1 — Варианты задания на лабораторную работу №4

№ варианта	Задержка t1=t2, мкс	Масштабный коэффициент	Операция
1	500	3,25	умножение
2	800	0,36	деление
3	1500	2,5	умножение
4	800	0,44	деление
5	2000	1,75	умножение
6	3000	0,57	деление
7	1500	2,25	умножение
8	3000	0,31	деление
9	2000	2,75	умножение
10	5000	0,4	деление
11	4000	1,5	умножение
12	5000	0,35	деление
13	500	0,5	умножение
14	800	2,3	деление
15	1500	3,2	умножение
16	800	0,9	деление
17	2000	1,25	умножение
18	3000	3,75	деление
19	1500	3,5	умножение
20	3000	0,2	деление
21	2000	0,4	умножение
22	5000	1,5	деление
23	4000	2,7	умножение
24	5000	2,25	деление
25	500	3,75	умножение

3. Занести программу в память микроконтроллера (для каждой из четырех разработанных версий программы).

4. Исследовать с помощью осциллографа получаемые временные задержки, убедится, что они совпадает с заданной по варианту.

6. Провести сравнительный анализ разработанных версий программ.

7. Разработать на языке ассемблера AVR программу, которая масштабирует двоичное число (без знака), вводимое пользователем с клавиатуры (порт D), переводит его в десятичную систему счисления и выдает через порт B на индикацию (программа должна реагировать на любое изменение информации, вводимой через порт D). Параллельно с этим на одном из выводов порта A должна формироваться импульсная последовательность, рассчитанная в предыдущей лабораторной работе. Для формирования импульсной последовательности использовать метод обработки прерываний от таймера, рассмотренный в данной лабораторной работе. Масштабируемые числа и масштабные коэффициенты приведены в таблице 1.1. Программу разрабатывать в предположении, что диапазон десятичных чисел, получаемых после масштабирования, составляет 0...99.

8. Скомпилировать и проверить правильность работы программы в режиме эмулятора AVR-Studio.

9. Записать программу в память микроконтроллера.

10. Исследовать получаемую временную задержку с помощью осциллографа, убедиться, что она совпадает с заданной по варианту.

11. Исследовать выполнения масштабирования и перевода в десятичную систему при помощи индикаторов макета CAN128.

12. Разработать и исследовать версию той же программы на языке С.

4. Содержание отчёта

1. Цель работы.

2. Постановка задачи (с указанием данных конкретного варианта).

3. Расчет необходимых параметров (в частности, расчет значений параметров для реализации заданной импульсной последовательности, расчет необходимого для масштабирования количества операций сдвига исходного кода вправо и влево).

4. Блок-схема алгоритма.

5. Текст программы с комментариями.

6. Описание программ.

7. Описание результатов тестирования (входных и соответствующих им выходных значений).

8. Полученные осциллограммы.

9. Результаты исследований и сравнительный анализ программ.

10. Выводы.

5. Контрольные вопросы

1. Опишите метод программных циклов для реализации временных интервалов. Как рассчитываются параметры циклов?

2. Как организовать временную задержку заданной величины при помощи таймера? Опишите метод ожидания переполнения таймера.

3. Укажите основной недостаток рассмотренных методов.

4. Какие служебные биты определяют время, через которое инкрементируется счетный регистр таймера T0 (для микроконтроллеров AVR)? Как рассчитать это время?

5. Какие действия выполняет микроконтроллер при возникновении сигнала прерывания?

6. Назовите виды прерываний, поддерживаемых микроконтроллером AT90CAN128.

7. Что представляет из себя таблица векторов прерывания? Назовите минимальный и максимальный размер таблицы прерываний для микроконтроллеров AVR .

8. Что такое вектор прерывания (местонахождение, размер, содержимое)?

9. Чем определяется приоритет прерывания?

10. Какие действия выполняются микроконтроллером по команде возврата из прерывания RETI?

11. Какие служебные регистры (регистры ввода-вывода) используются для

управления работой системы прерываний (общее разрешение прерываний, разрешение конкретного прерывания, например, от таймера T0, флаг прерывания)?

12. Для чего в программе обработки прерывания необходимо сохранять старое состояние регистра SREG? Объясните на примере своей программы.

13. Опишите метод реализации временной задержки заданной величины на основе обработки прерывания от таймера. Укажите его преимущество.

2. ЛАБОРАТОРНАЯ РАБОТА №6

Исследование передачи данных по интерфейсу SPI для микроконтроллеров семейства AVR

1. Цель работы

Ознакомится с последовательным периферийным интерфейсом SPI и форматом передаваемых им данных. Исследовать способ обмена данными между микроконтроллером и ЦАП посредством интерфейса SPI.

2. Теоретическое введение

2.1. Последовательный периферийный интерфейс SPI

Последовательный интерфейс периферии SPI (Serial Peripheral Interface) имеет несколько назначений. Во-первых, с его помощью может осуществляться обмен данными между микроконтроллером и различными периферийными устройствами, такими, как ЦАП/АЦП, FLASH-ПЗУ, цифровыми потенциометрами и т.д. Во-вторых, посредством этого интерфейса можно производить обмен данными между несколькими микроконтроллерами. При обмене данными по интерфейсу SPI микроконтроллер AVR может работать как ведущий (режим Master) либо как ведомый (режим Slave). Также через интерфейс SPI может быть произведено программирование микроконтроллера — так называемый режим последовательного программирования. Структурная схема модуля SPI приведена на рисунке 2.1.Модуль SPI использует четыре вывода микроконтроллера. Как и в большинстве прочих периферийных устройств, эти выводы являются линиями портов ввода/вывода общего назначения, при этом для разных типов микроконтроллеров порты и номера их линий могут быть различны.

Если работа SPI разрешена, то разрешается альтернативное направление выводов MOSI, MISO, SCK и SS (таблица 2.1).

Таблица 2.1 — Направление выводов SPI

Вывод	Направление для ведущего SPI	Направление для подчиненного SPI
MOSI	Определяется пользователем	Вход
MISO	Вход	Определяется пользователем
SCK	Определяется пользователем	Вход
SS	Определяется пользователем	Вход

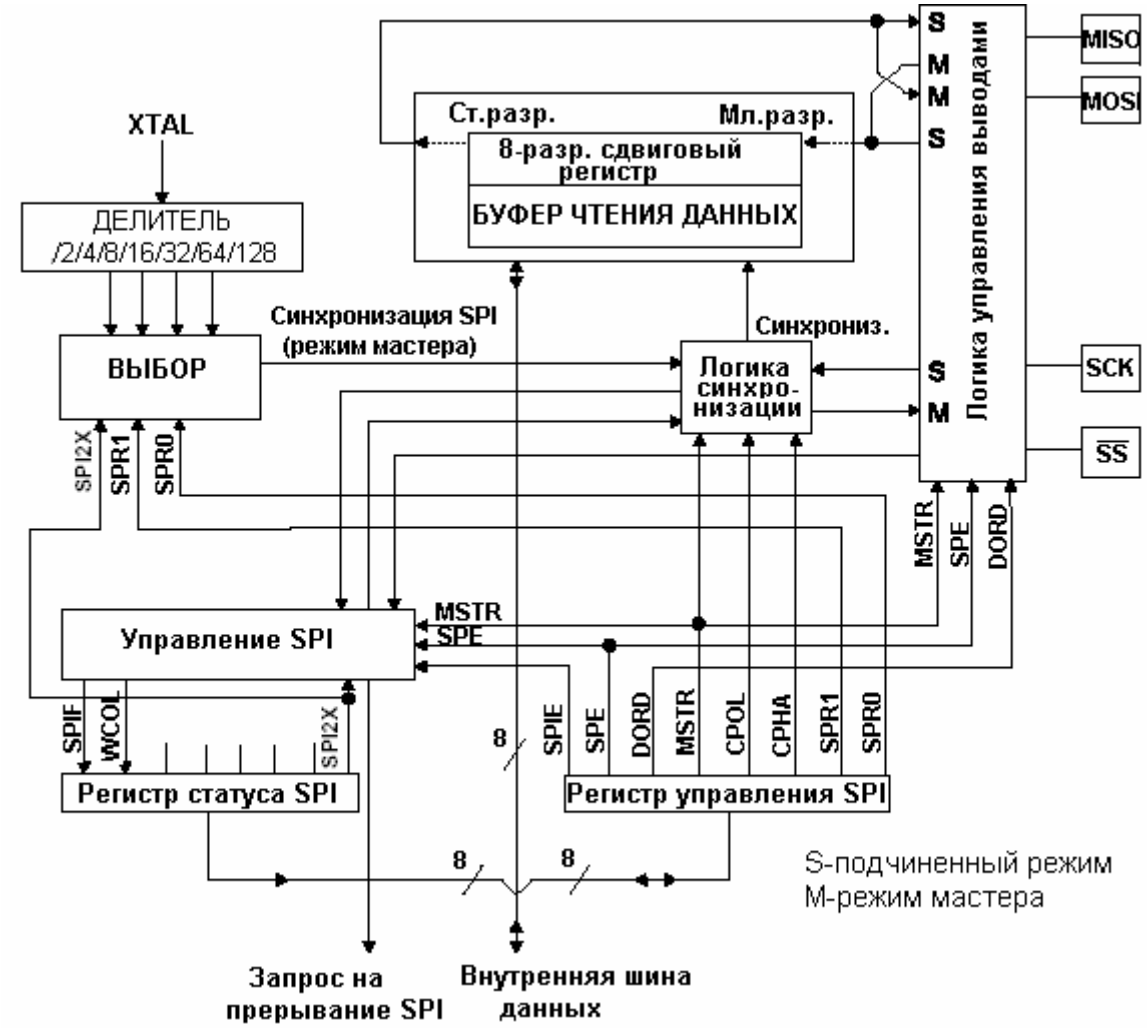


Рисунок 2.1 — Структурная схема модуля SPI

В таблице 2.2 приведено назначение битов регистра управления SPCR.

Таблица 2.2 — Регистр управления SPCR

Разряд	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Чтение/запись	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	
Исх. значение	0	0	0	0	0	0	0	0	

Если установлен флаг SPIF в регистре SPSR и установлен бит общего разрешения прерываний I в регистре SREG, то установка данного бита приведет к исполнению процедуры обработки прерывания по SPI. Если в SPE записать лог. 1, то разрешается работа SPI. Данный бит должен быть установлен, если необходимо использовать SPI независимо от того в каком режиме он будет работать. Если DORD=1, то при передаче слова данных первым передается младший разряд. Если же DORD=0, то первым передается старший разряд. Если в MSTR записана лог. 1, то SPI работает как ведущий (мастер), иначе (MSTR=0) как подчиненный. Если SS настроен как вход и к нему приложен низкий уровень, когда MSTR был равен 1, то бит MSTR автоматически сбрасывается и устанавливается флаг прерывания SPIF в регистре SPSR. Для возобновления ведущего режима SPI пользователь должен предусмотреть программную установку бита MSTR. Если CPOL равен лог. 1, то SCK

имеет высокий уровень в состоянии ожидания. Если CPOL=0, то SCK имеет низкий уровень в состоянии ожидания. Значение бита фазы синхронизации (CPHA) определяет по какому фронту SCK происходит выборка данных: по переднему или заднему. SPR1, SPR0 задают частоту синхронизации на выводе SCK в режиме мастера. SPR1 и SPR0 не оказывают никакого влияния в режиме подчиненного.

В таблице 2.3 приведено назначение битов регистра статуса SPSR.

Таблица 2.3 — Регистр статуса SPSR

Разряд	7	6	5	4	3	2	1	0	
	SPIF	WCOL	—	—	—	—	—	SPI2X	SPSR
Чтение/запись	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт./Зп.	
Исх. значение	0	0	0	0	0	0	0	0	

Флаг SPIF устанавливается по завершении последовательной передачи. Прерывание генерируется в том случае, если установлен бит SPIE в регистре SPCR и разрешены общие прерывания. Если SS настроен как вход и к нему приложен низкий уровень, то, если SPI находился в режиме мастера, также установится флаг SPIF. SPIF сбрасывается аппаратно при переходе на соответствующий вектор прерывания. Альтернативно, бит SPIF сбрасывается при первом чтении регистра статуса SPI с установленным флагом SPIF, а также во время доступа к регистру данных SPI (SPDR). Бит WCOL устанавливается, если выполнена запись в регистр данных SPI (SPDR) во время передачи данных. Бит WCOL (а также бит SPIF) сбрасывается при первом чтении регистра статуса SPI с установленным WCOL, а также во время доступа к регистру данных SPI. Если в SPI2X записать лог. 1 то скорость работы SPI (частота SCK) удвоится, если SPI находится в режиме мастера. Это означает, что минимальный период SCK будет равен двум периодам синхронизации ЦПУ. Если SPI работает как подчиненный, то работа SPI гарантирована только на частоте fosc /4 или менее

В таблице 2.4 приведено назначение битов регистра данных SPDR.

Таблица 2.4 — Регистр данных SPDR

Разряд	7	6	5	4	3	2	1	0	
	Ст.разр.							Мл.разр.	SPDR
Чтение/запись	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	Чт/Зп	
Исх. значение	x	x	x	x	x	x	x	x	Неопред.

Регистр данных SPI имеет доступ на чтение и запись и предназначен для обмена данными между файлом регистров (r0…r31) и сдвиговым регистром SPI. Запись в данный регистр инициирует передачу данных. При чтении данного регистра фактически считывается содержимое приемного буфера сдвигового регистра.

2.2. Цифро-аналоговое преобразование

Цифро-аналоговый преобразователь (ЦАП) — устройство для преобразования цифрового (обычно двоичного) кода в аналоговый сигнал (ток, напряжение или за-

ряд). Наиболее общие типы электронных ЦАП: широтно-импульсный модулятор, ЦАП передискретизации, цепная R-2R схема, взвешивающий, сегментный ЦАП, гибридные ЦАП [6].

При разработке микропроцессорной системы, содержащей ЦАП, необходимо произвести расчет параметров: необходимые разрядности по погрешности (p[%]), по амплитуде (Umax[B]).

Для обеспечения точности, число разрядов преобразователя вычисляется по формуле (2.1).

$$N = \lceil \log_2 (1 / p) \rceil . \quad (2.1)$$

Например, p = 0,5 %: $N = \lceil \log_2 (1 / 0{,}005) \rceil = \lceil \log_2 200 \rceil = 7{,}64 \approx 8$ (бит).

При этом цена деления будет равна *Umax/2N* , где *Umax* – диапазон изменения амплитуды входного (выходного) аналогового сигнала.

Если входной сигнал изменяется в диапазоне от *0* до *Fmax Гц* , то по теореме Котельникова (Шеннона), максимальный интервал между отсчетами (записью) вычисляется по формуле (2.2).

$$\Delta t = \frac{1}{2 * F \max} . \quad (2.2)$$

Максимальное время преобразования ЦАП должно быть не больше Δt . Тогда сигнал можно восстановить без ощутимых потерь.

2.3. Схема цифро-аналогового преобразователя макета CAN128

Микросхема MCP4921 – экономичный 12-разрядный цифро-аналоговый преобразователь с возможностью изменения коэффициента передачи выходного буфера и SPI-интерфейсом. Преобразователь обеспечивает высокую точность и малый уровень шумов во всем расширенном диапазоне температур.

Схема электрическая принципиальная подключения ЦАП на макете CAN128 представлены на рисунке 2.2.

Для связи с микросхемой ЦАП используется интерфейс SPI. Передача данных осуществляется по временной диаграмме, изображенной на рисунке 2.3. Передаваемое слово имеет следующий формат: 4 конфигурационных бита и 12 информационных бит.

Описание конфигурационных бит:

- A/B – выбор канала A или B, 0 или 1 соответственно.
- BUF – бит входного буфера усилителя, 1 – буферизировано, 0 – не буферизировано.
- GA – выбор напряжения увеличенного в двое, при 0 увеличенное вдвое, при 1 не увеличенное.
- SHDN – выбор режима отключения выходного сигнала, 0 – отключён, 1 - включён.

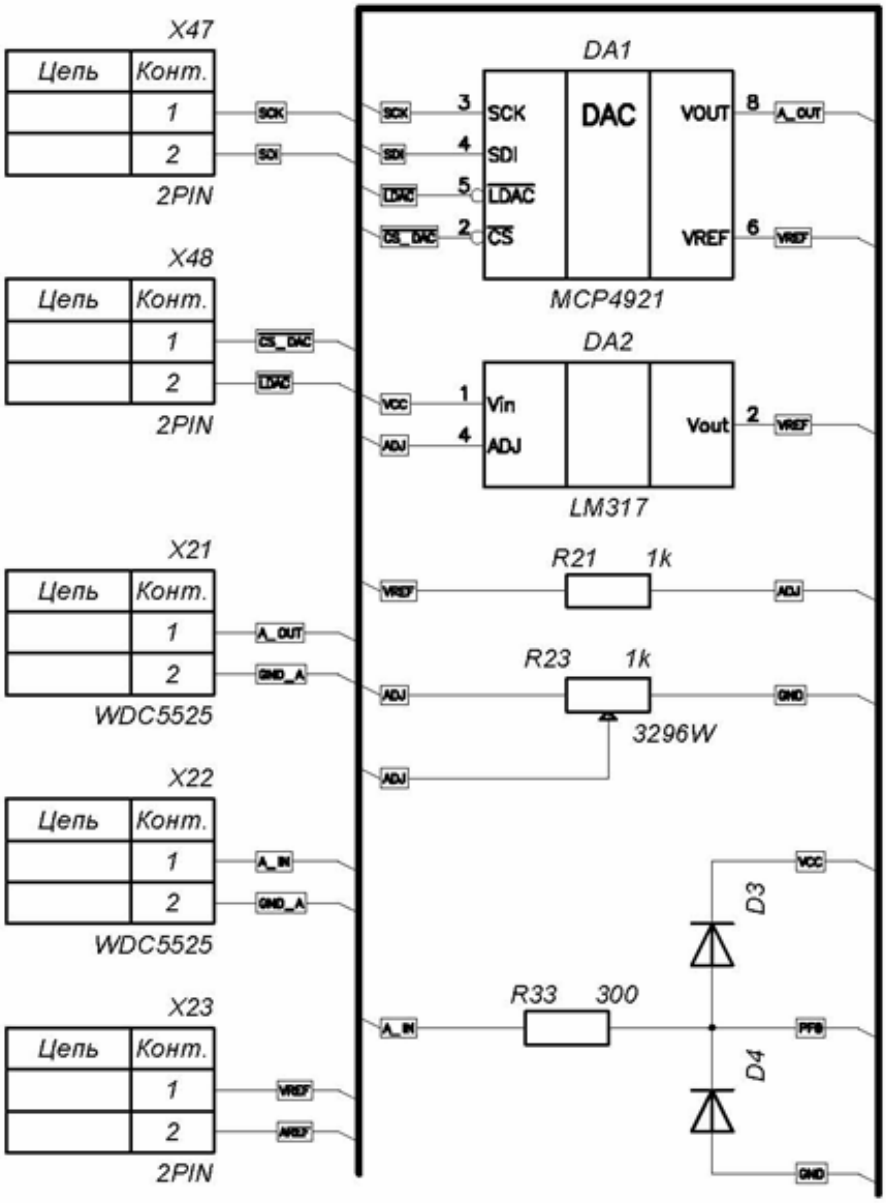


Рисунок 2.2 – Схема подключения ЦАП на макете CAN128

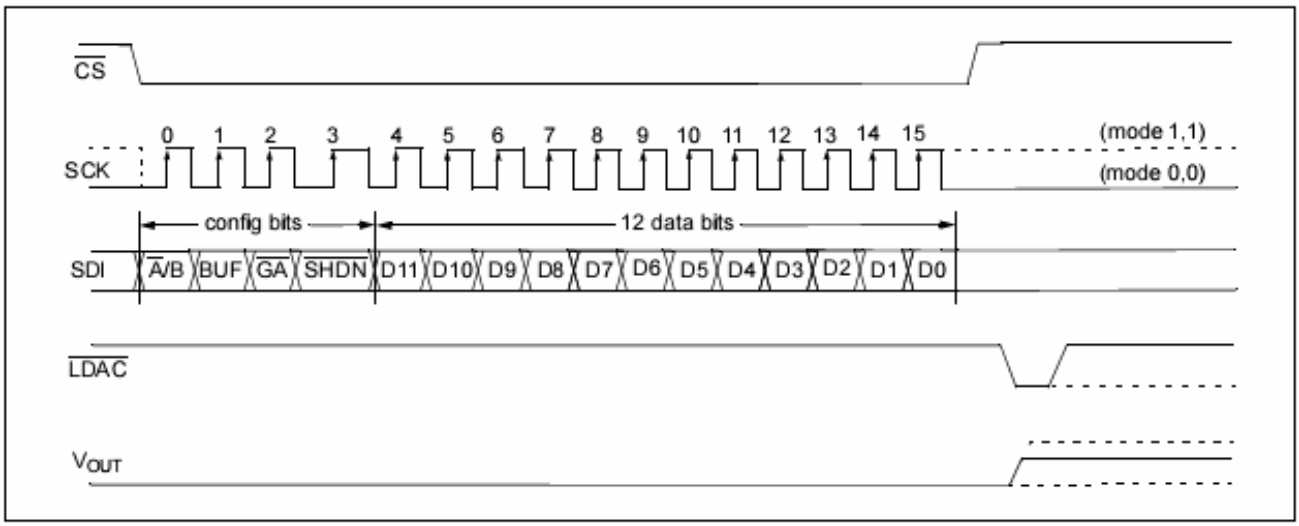


Рисунок 2.3 – Временная диаграмма передачи данных по SPI

2.4. Коммутация ЦАП на макете CAN128

Для подключения ЦАП через SPI на макете CAN128 необходимо соединить выводы в следующим образом: вывод CS с выводом PB0, SCK – с PB1, SDI – с PB2, LDAC – с PB4. На рисунке 2.4 указано обозначение и местонахождение перечисленных выводов и показано данное подключение непосредственно на макете. Также описание коммутации сведено в таблицу 2.5.

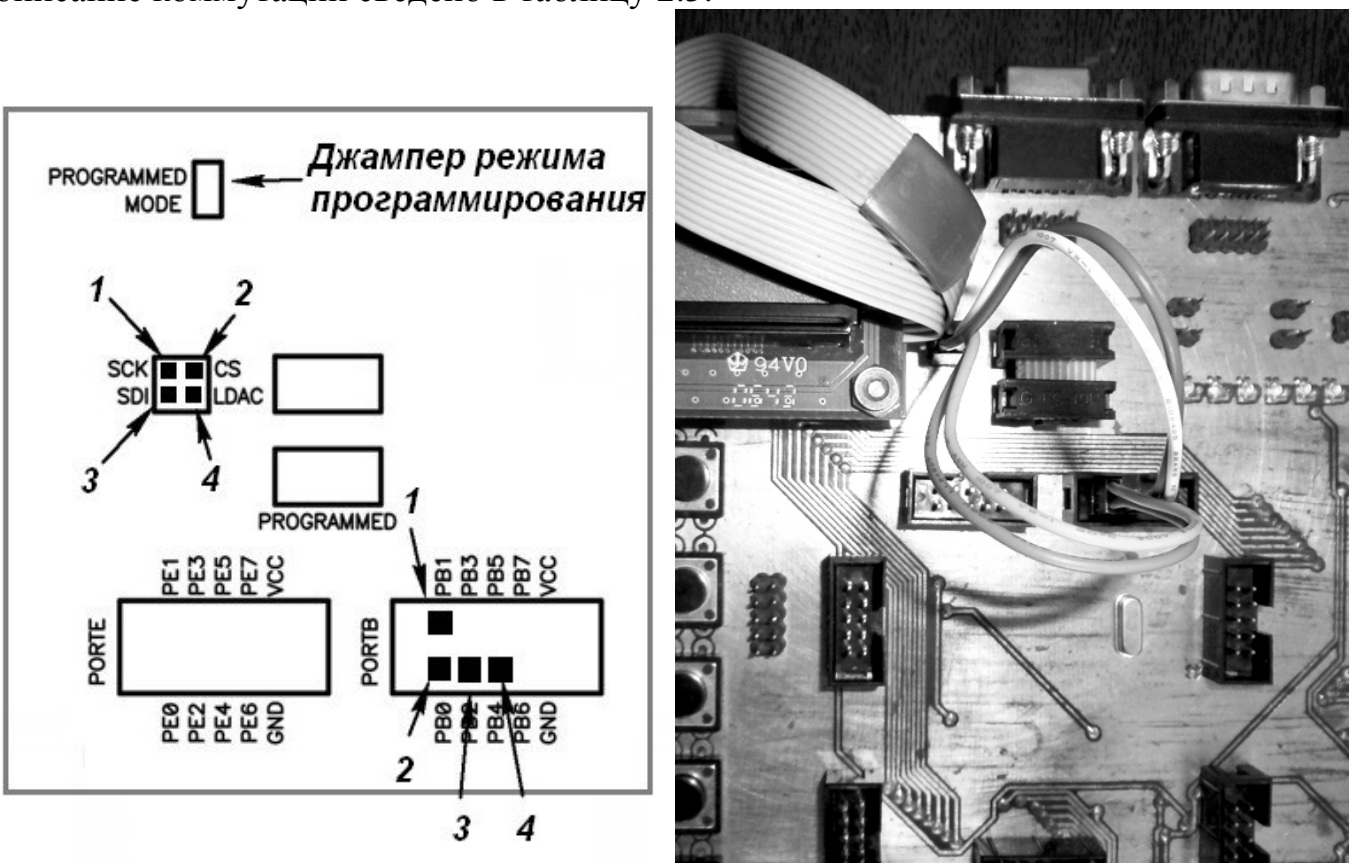


Рисунок 2.4 – Размещение и обозначение выводов для подключения ЦАП через SPI

Таблица 2.5 – Описание коммутации соединений для тестирования интерфейса SPI (с микросхемы MCP4921 на порт B микроконтроллера)

№ 2PIN	№ Контакта 2PIN, выходы MCP4921	Цепь, обозначения выходов MCP4921	№ 10PIN	№Конт. 10PIN	Цепь, обозначения контактов МК
X48	1	CS	X2	1	PB0(CS)
	2	LDAC		5	PB4(LDAC)
X47	1	SCK		2	PB1(SCK)
	2	SDI		3	PB2(SDI)

Необходимо помнить, что интерфейс SPI используется в макете CAN128 и для организации программирования микроконтроллера, поэтому для работы с ЦАП необходимо отключить программирование (сделать это нужно после прошивки программы в микроконтроллер). Для этого необходимо вытащить джампер режима программирования. Его расположение указано на схеме на рисунке 2.4 и на фотографии макета (рисунок 2.5).

Рисунок 2.5 – Расположение джампера режима программирования

3. Задание на исследование

Исходными данными к лабораторной работе являются параметры сигнала: форма, амплитуда, период сигнала и погрешность преобразования. Также задан язык программирования. Необходимо:

1. Проанализировать параметры сигнала согласно варианту.
2. Сделать необходимые расчеты.
3. Написать программу (язык программирования – согласно варианту), которая генерирует заданный сигнал.
4. Исследовать полученную осциллограмму сгенерированного сигнала.

Варианты задания представлены в таблице 2.6.

Таблица 2.6 – Варианты заданий на лабораторную работу №6

№ варианта	Форма сигнала	Амплитуда сигнала U, В			Период сигнала T, с			Погр. преоб. P, %
		U1	U2	U3	T1	T2	T3	
1	Прямоугольная	0.50	0.40	0.30	0.050	0.040	0.030	—
2	Треугольная	0.50	1	—	0.025	0.050	—	10%
3	Пила обр. справа	0.30	0.40	0.50	0.050	0.040	0.030	5%
4	Пила обр. слева	0.50	0.40	0.30	0.030	0.040	0.050	5%
5	Синусоида	4.09	—	—	0.005	—	—	10%
6	Прямоугольная	1	2	3	0.010	0.010	0.010	—
7	Треугольная	2	2	2	0.010	0.020	0.030	5%
8	Пила обр. справа	3	1	2	0.010	0.020	0.030	20%
9	Пила обр. слева	2	4	—	0.020	0.010	—	10%
10	Синусоида	1	—	—	0.010	—	—	15%
11	Прямоугольная	1	1.40	—	0.050	0.040	—	—
12	Треугольная	2	0.5	1.8	0.005	0.005	0.005	15%
13	Пила обр. справа	4	2	0.50	0.050	0.040	0.030	10%
14	Пила обр. слева	1.50	2.40	3.30	0.030	0.040	0.050	5%
15	Синусоида	1	—	—	0.050	—	—	1%

Сигнал может иметь как простую, так и сложную форму, которая задается двумя или тремя составляющими сигнала. Язык программирования выбирается следующим образом: для нечётного номера варианта – Ассемблер, для чётного номера варианта – Си.

В приложении А приведен пример программы, реализующей вывод данных на ЦАП.

4. Содержание отчёта

1. Цель работы.
2. Постановка задачи (с указанием данных конкретного варианта).
3. Расчет необходимых параметров
4. Текст программы.
5. Описание программы.
6. Полученные осциллограммы.
7. Результаты исследования.
8. Выводы.

5. Контрольные вопросы

1. Перечислите последовательные интерфейсы, реализованные в микроконтроллерах AVR. Приведите их сравнительный анализ.
2. Каковы принципы последовательной передачи данных?
3. Какие особенности имеет интерфейс SPI?
4. приведите временные диаграммы передачи данных по интерфейсу SPI.
5. Какие режимы работы SPI Вам известны?
6. Каковы виды и форматы посылок SPI?
7. Приведите схему подключения устройств по интерфейсу SPI.
8. Какова структура блока SPI?
9. Каково назначение интерфейса SPI?

3. ЛАБОРАТОРНАЯ РАБОТА №7

Исследование передачи данных по интерфейсу УСАПП в микроконтроллерах семейства AVR

1. Цель работы

Ознакомится с универсальным синхронным и асихронным последовательным приемопередатчиком (УАПП/УСАПП) и форматом передаваемых им данных. Изучит способ обмена данными между микроконтроллерами посредством интерфейса RS-323.

2. Теоретическое введение

2.1. Универсальный синхронный и асинхронный последовательный приемопередатчик (УСАПП)

Универсальный синхронный и асинхронный последовательный приемопере-

датчик (УСАПП, англ.: USART) предназначен для организации гибкой последовательной связи.

На рисунке 3.1 представлена упрощенная функциональная схема УСАПП, жирным шрифтом выделены регистры и выводы УСАПП.

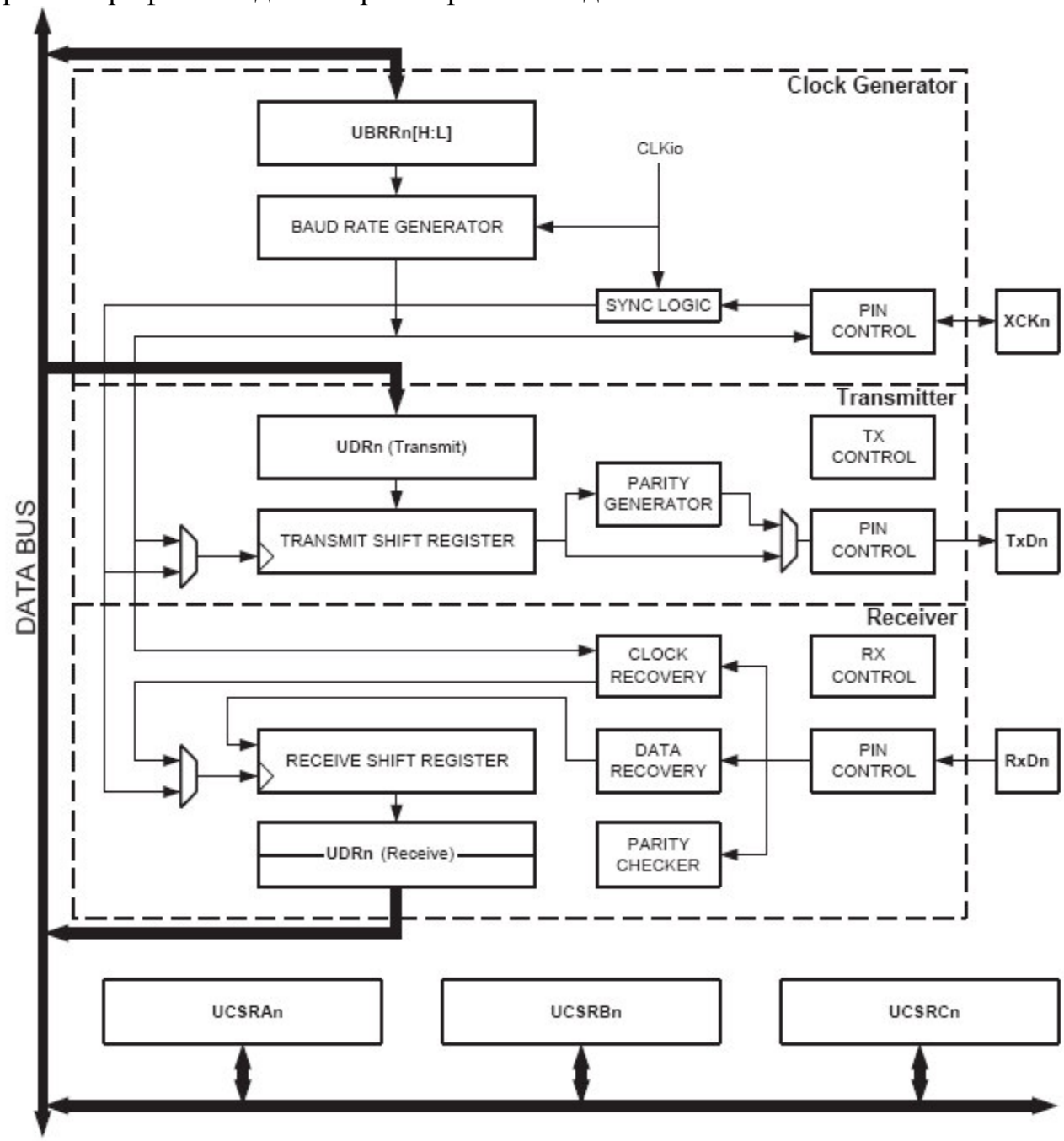


Рисунок 3.1 — Функциональная схема УСАПП

Пунктирной линией выделены три основных блока УСАПП: тактовый генератор, передатчик и приемник. Регистры управления используются всеми блоками. Логика тактового генератора состоит из логики синхронизации, связанной с внешним тактовым входом (используется в подчиненном режиме) и генератора скорости связи. Вывод XCK (синхронизация передачи) используется только в режиме синхронной передачи.

Передатчик состоит из одного буфера записи, последовательного сдвигового регистра, генератора паритета и управляющей логики, которая поддерживает раз-

личные форматы последовательной посылки. Буфер записи позволяет непрерывно передавать данные без каких-либо задержек между передачей посылок. Приемник является более сложным блоком УСАПП, т.к. в его состав входят модули обнаружения данных и синхронизации. Модули обнаружения необходимы для асинхронного приема данных. Помимо модулей обнаружения в приемник входит устройство проверки паритета, сдвиговый регистр, и двухуровневый приемный буфер (UDR). Приемник поддерживает те же последовательные форматы, что и передатчик, и может определить ошибку в посылке (кадре), переполнение данных и ошибку паритета.

Последовательная посылка состоит из бит данных, бит синхронизации (старт и стоп-биты), а также опционального бита паритета для поиска ошибок. УСАПП поддерживает все 30 комбинаций следующих форматов посылок:

- 1 старт-бит
- 5, 6, 7, 8 или 9 бит данных
- без паритета, с битом четности, с битом нечетности
- 1 или 2 стоп-бита

Посылка начинается со старт-бита, а за ним следует передача бит данных, начиная с самого младшего разряда. Затем следует передача остальных бит данных (максимальное число бит данных 9), которая заканчивается передачей старшего разряда данных. Если разрешена функция контроля паритета, то сразу после бит данных передается бит паритета, а затем стоп-биты. После завершения передачи посылки имеется возможность либо передавать следующую посылку, либо перевести линию связи в состояние ожидания (высокий уровень). На рисунке 3.2 проиллюстрирована возможность сочетания форматов посылки. Наличие прямоугольной скобки указывает на опциональность данного формата посылки.

Рисунок 3.2 — Форматы посылки

- St - Старт-бит имеет всегда низкий уровень.
- 0…8 - Номер бита данных.
- P - бит паритета: четность или нечетность.
- Sp1,Sp2 - Стоп-бит имеет всегда высокий уровень.
- IDLE - состояние ожидания, в котором приостановлена передача на RxD или TxD. В состоянии ожидания на линии должен быть высокий уровень.

Формат посылки, который используется УСАПП, задается битами UCSZ2:0, UPM1:0 и USBS в регистрах UCSRB и UCSRC. Приемник и передатчик используют одни и те же установки форматов. Обратите внимание, что изменение установок любого из этих бит может привести к повреждению текущего сеанса связи, как для приемника, так и для передатчика.

Биты выбора длины передаваемых данных (UCSZ2:0) определяют, из скольки бит данных состоит посылка. Биты режима паритета УСАПП (UPM1:0) разрешают

передачу/контроля бита паритета и устанавливают тип паритета: четность, нечетность. Выбрать один или два стоп-бита позволяет бит выбора стоп-бита УСАПП (USBS). Приемник игнорирует второй стоп-бит. Флаг ошибки посылки FE позволяет выявить ситуацию, когда первый стоп-бит равен 0.

Бит паритета вычисляется путем выполнения логической операции исключающего ИЛИ над всеми битами данных. Если используется нечетность, то результат этой операции инвертируется:

$$P_{ЧЕТН} = d_{n-1} \oplus ... \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0,$$

$$P_{НЕЧЕТН} = d_{n-1} \oplus ... \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1,$$

где

$P_{ЧЕТН}$ — бит четного паритета;

$P_{НЕЧЕТН}$ — бит нечетного паритета;

d_n — n-ый бит данных в посылке.

После разрешения, бит паритета передается между последним битом данных и первым стоп-битом.

Логика генерации тактовых импульсов формирует основную синхронизацию приемника и передатчика. УСАПП поддерживает четыре режима работы синхронизации: нормальная асинхронная, асинхронная с удвоением скорости, ведущая синхронная и подчиненная синхронная. Бит UMSEL в регистре С управления и статуса (UCSRC) позволяют выбрать асинхронную или синхронную работу. Удвоение скорости (только в асинхронном режиме) управляется битом U2X в регистре UCSRA. При использовании синхронного режима (UMSEL = 1) соответствующий бит в регистре направления данных для вывода XCK (DDR_XCK) задает будет ли синхронизация внутренней (ведущий режим) или внешней (подчиненный режим). Вывод XCK активен только при использовании синхронного режима (рисунок 3.3).

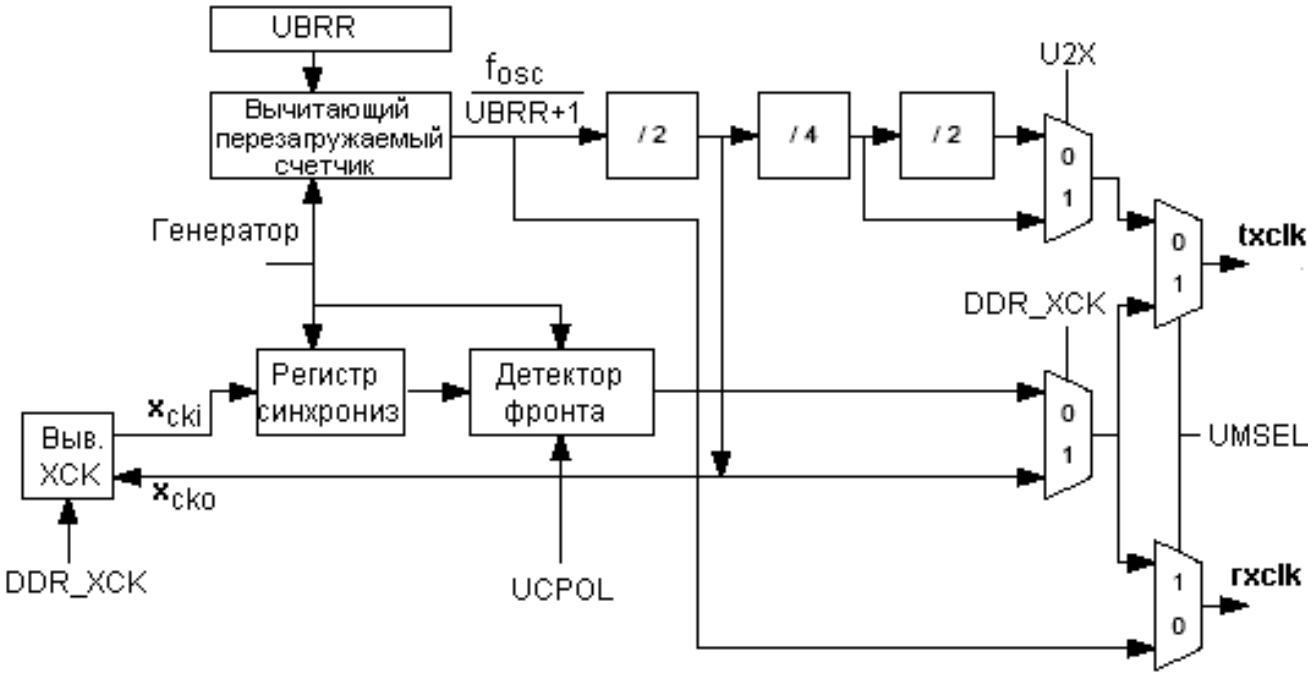


Рисунок 3.3 — Функциональная схема логики синхронизации УСАПП

Описание сигналов:

- txclk - синхронизация передатчика (внутренний сигнал)
- rxclk - основная синхронизация приемника (внутренний сигнал)
- xcki - вход от вывода XCK (внутренний сигнал). Используется для синхронной подчиненной работы.
- xcko - выход синхронизации к выводу XCK (внутренний сигнал). Используется в ведущем синхронном режиме.
- fosc - вывод частоты XTAL (системная синхронизация).

Внутренняя синхронизация используется для асинхронного и ведущего синхронного режимов работы. Регистр генератора скорости связи (UBRR) и связанный с ним вычитающий счетчик функционируют как программируемый предделитель или генератор скорости связи. Вычитающий счетчик тактируется системной синхронизацией (fosc) и перезагружается значением из регистра UBRR всякий раз при достижении нулевого значения или после записи регистра UBRRL. Тактовый сигнал генерируется всякий раз при достижении счетчиком нулевого значения. Данный тактовый сигнал является тактовым выходом генератора скорости связи (= fosc/(UBRR+1)). Передатчик делит частоту генератора скорости связи на 2, 8 или 16 в зависимости от режима работы. Модули обнаружения синхронизации и данных приемника подключены непосредственно к тактовому выходу генератора скорости связи. Однако цифровой автомат модулей обнаружения используют 2, 8 или 16 состояний в зависимости от режима, задаваемого битами UMSEL, U2X и DDR_XCK.

Таблица 3.1 содержит выражения для вычисления скорости связи (в битах в секунду) и вычисления значений UBRR для каждого из рабочих режимов при использовании внутренне генерируемого тактового источника. Скорость связи представлена в битах в секунду (бод).

- BAUD - скорость связи (в битах в секунду, бод);
- fOSC - частота синхронизации системного генератора;
- UBRR - Содержимое регистров UBRRH и UBRRL, (0 … 4095).

Таблица 3.1 — Выражения для вычисления установок регистра скорости связи

Режим работы	**Выражение для вычисления скорости связи**	**Выражения для вычисления значения UBRR**
Нормальный асинхронный режим (U2X=0)	$BAUD = \frac{f_{osc}}{16(UBRR+1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Асинхронный режим с удвоением скорости (U2X=1)	$BAUD = \frac{f_{osc}}{8(UBRR+1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Синхронный ведущий режим	$BAUD = \frac{f_{osc}}{2(UBRR+1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

Подробно форматы регистров УСАПП рассмотрены в [1,2,3,4].

3. Описание лабораторной установки

Микроконтроллер AT90CAN128 содержит два УСАПП: УСАПП0 и УСАПП1.

УСАПП0 и УСАПП1 имеют раздельные регистры ввода-вывода. На стенде CAN128 реализован интерфейс RS-232 и преобразователь ST-232.

В приложении Б приведены примеры программ обмена по интерфейсу УСАПП на языке Ассемблер.

Для занесения программы в память микроконтроллера необходимо подключить стенды, как это описано в предыдущей лабораторной работе. В данной работе обмен информацией осуществляется между двумя стендами, поэтому занесение программ в память можно осуществить или по очереди подключая стенды к одному компьютеру, или же подключить их разным компьютерам.

Программа обмена, приведенная в приложении, рассчитана на частоту работы микроконтроллера в 8МГц, поэтому необходимо убедится, что на обоих макетах выставлена соответствующая частота. Для этого в PonyProg2000 необходимо прочитать FuseBits и сверить, с тем, что представлено на рисунке 3.4. Если же прочитанные FuseBits с данного макета не совпадают с рисунком 3.4, то необходимо установить указанные значения и записать их в макет. Внимание, при неверных установленных значениях FuseBits макет будет заблокирован, что приведет к невозможности его дальнейшего использования!

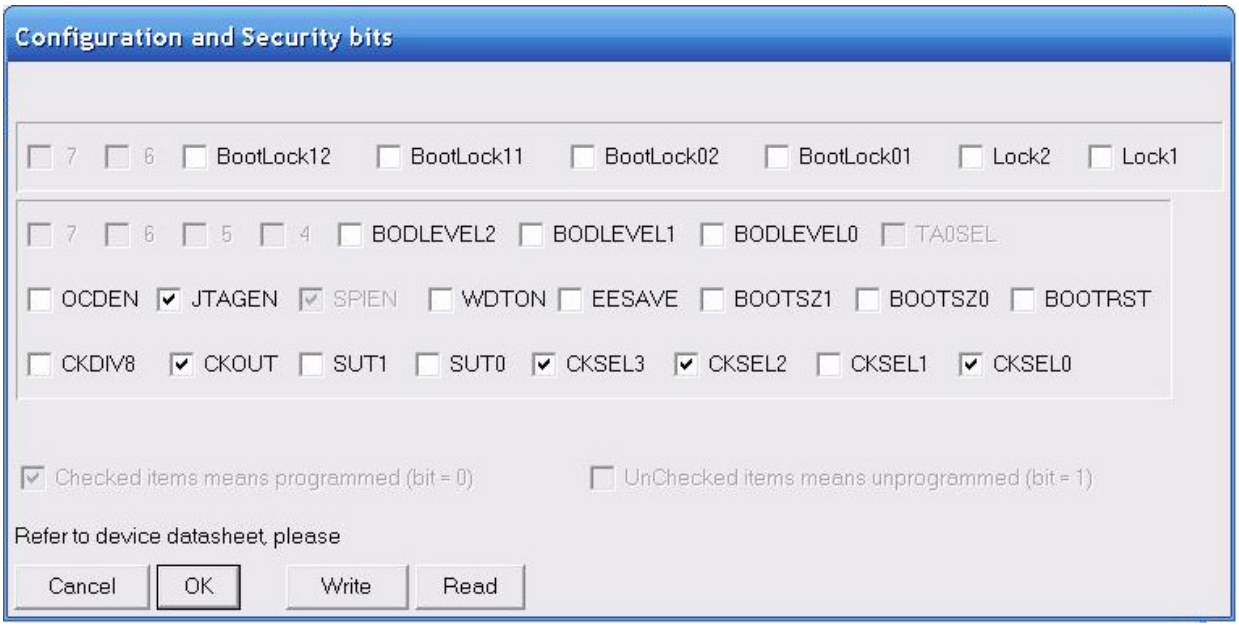


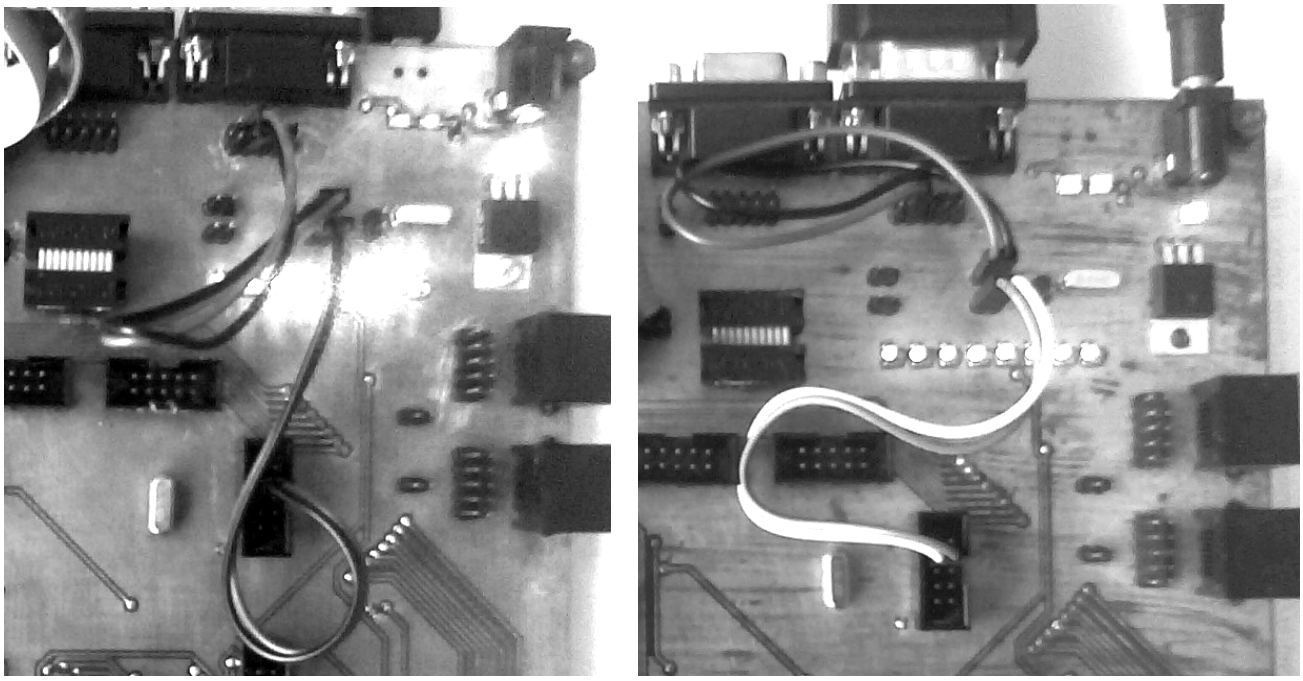
Рисунок 3.4 – Значения FuseBits для 8МГц

После проверки и установки рабочей частоты макетов, можно переходить непосредственно к занесению программ в память микроконтроллеров. Пусть макет1 будет передатчиком (transmitter), а макет2 — приёмником (receiver). После того, как программы занесены в память микроконтроллеров, необходимо отключить питание от обоих стендов и произвести коммутацию между макетами. Схема соединений между макетами через интерфейс УСАПП сведена в таблицу 3.2, а также приведена на рисункее 3.5 (а – передатчик, б – приемник).

После коммутации можно подать питание на оба макета и убедится в работоспособности тестовой программу. Если все правильно, то значения на линейке светодиодов обоих макетов совпадают.

Таблица 3.2 – Описание коммутации соединений для интерфейса UART

№ 2PIN	№Контакта 2PIN, выходы ST232	Цепь, обозначения выходов ST232	№IDC10 (10PIN)	№Конт. IDC10, Порты МК	Цепь, обозначения контактов МК и IDC10
Описание соединений для тестирования UART_1					
С микросхемы ST232 на порт E микроконтроллера					
X43	1	T1IN	X5	1	PE0(RXD0)
X43	2	R1OUT	X5	2	PE1(TXD0)
С микросхемы ST232 на IDC10(10PIN)					
X41	1	T1OUT	X12	2	RXD
X41	2	R1IN	X12	3	TXD
Описание соединений для тестирования UART_2					
С микросхемы ST232 на порт D микроконтроллера					
X44	1	T2IN	X4	3	PD2(RXD1)
X44	2	R2OUT	X4	4	PD3(TXD1)
С микросхемы ST232 на IDC10(10PIN)					
X42	1	T2OUT	X12	2	RXD
X42	2	R2IN	X12	3	TXD



а) б)

Рисунок 3.5 – Коммутация соединения: а) для передатчика; б) для приёмника

4. Задание на исследование

В рамках выполнения лабораторной работы необходимо:

1. Изучить теоретические сведения о работе УСАПП.

2. Разработать программы для реализации обмена данными по интерфейсу УСАПП: одна программа для передатчика, вторая — для приемника. Программы должны обеспечивать при нажатии на заданную кнопку первого макета вывод на ЖКИ второго макета требуемой информации (на английском языке). Дополнитель-

ные параметры передачи, передаваемые данные и язык программирования приведены в таблице 3.2.

Таблица 3.2 – Варианты заданий на лабораторную работу №7

№ варианта	Скорость передачи (бит/с)	Язык программирования	№ нажимаемой кнопки	Передаваемая информация
1	9600	Ассемблер	1	Фамилия
			2	Номер группы
2	19200	Ассемблер	3	Фамилия
			4	Дата рождения
3	38400	Ассемблер	5	Фамилия
			6	Номер группы
4	57600	Ассемблер	7	Фамилия
			8	Дата рождения
5	9600	Си	9	Фамилия
			10	Номер группы
6	19200	Си	11	Фамилия
			12	Дата рождения
7	38400	Си	13	Фамилия
			14	Номер группы
8	57600	Си	15	Фамилия
			16	Дата рождения
9	9600	Ассемблер	1	Фамилия
			2	Дата рождения
10	19200	Ассемблер	3	Фамилия
			4	Номер группы
11	38400	Ассемблер	5	Фамилия
			6	Дата рождения
12	57600	Ассемблер	7	Фамилия
			8	Номер группы
13	9600	Си	9	Фамилия
			10	Дата рождения
14	19200	Си	11	Фамилия
			12	Номер группы
15	38400	Си	13	Фамилия
			14	Дата рождения
16	57600	Си	15	Фамилия
			16	Номер группы
17	9600	Ассемблер	1	Фамилия
			2	Дата рождения
18	19200	Си	3	Фамилия
			4	Номер группы
19	38400	Ассемблер	5	Фамилия
			6	Дата рождения
20	57600	Си	7	Фамилия
			8	Номер группы

3. Произвести отладку программ в AVR-Studio и на макетах CAN128.

4. Исследовать с помощью разработанных программ принципы организации передачи данных по интерфейсу УСАПП.

5. Содержание отчёта

1. Цель работы.
2. Постановка задачи (с указанием данных конкретного варианта).
3. Расчет необходимых параметров.
4. Текст программы.
5. Описание программы.
6. Результаты проведенных исследований.
7. Выводы.

6. Контрольные вопросы

1. Перечислите последовательные интерфейсы, реализованные в микроконтроллерах AVR. Приведите их сравнительный анализ.
2. Каковы принципы последовательной передачи данных?
3. Перечислите отличительные особенности интерфейса УСАПП.
4. Приведите временные диаграммы передачи данных по интерфейсу УСАПП.
5. Проведите сравнение УСАПП и УАПП.
6. Каков формат посылки УСАПП?
7. Приведите схему подключения устройств по интерфейсу УСАПП.
8. Какова структура блока УСАПП?
9. Каково назначение интерфейса УСАПП?
10. Приведите описание модуля преобразования сигналов УСАПП в RS-232.
11. Какова реализация интерфейса RS-232 на макете CAN128?

4. ЛАБОРАТОРНАЯ РАБОТА №8

Исследование передачи данных по интерфейсу TWI для микроконтроллеров семейства AVR

1. Цель работы

Ознакомится с последовательным интерфейсом TWI и форматом передаваемых им данных. Изучит способ обмена данными между микроконтроллерами посредством интерфейса TWI.

2. Теоретическое введение

2.1. Последовательный интерфейс TWI

Двухпроводной последовательный интерфейс TWI идеально подходит для типичных применений микроконтроллера. Протокол TWI позволяет проектировщику системы внешне связать до 128 различных устройств через одну двухпроводную двунаправленную шину, где одна линия - линия синхронизации SCL и одна - линия данных SDA. В качестве внешних аппаратных компонентов, которые требуются для

реализации шины, необходимы только подтягивающий к плюсу питания резистор на каждой линии шины. Все устройства, которые подключены к шине, имеют свой индивидуальный адрес, а механизм определения содержимого шины поддерживается протоколом TWI. Боле подробно принцип подключения устройств к шине TWI рассмотрены в [1,2,3,4].

Каждый передаваемый бит данных по шине TWI сопровождается импульсом на линии синхронизации. Уровень данных должен быть стабильным, когда на линии синхронизации присутствует логическая 1. Исключением для этого правила является генерация условий старта и останова сеанса связи.

Ведущее устройство инициирует и заканчивает передачу данных. Передача инициируется, когда ведущий формирует условие СТАРТа на шине, и прекращается, когда ведущий формирует на шине условие ОСТАНОВа. Между условиями СТАРТа и ОСТАНОВа шина считается занятой и в этом случае никакой другой мастер не может осуществлять управляющие воздействия на шине. Существуют особые случаи, когда новое условие СТАРТа возникает между условиями СТАРТа и ОСТАНОВа. Данный случай именуется как условие "Повторного старта" и используется при необходимости инициировать мастером новый сеанс связи, не теряя при этом управление шиной. После "Повторного старта" шина считается занятой до следующего ОСТАНОВа. Это идентично поведению после СТАРТа, следовательно, при описании ссылка на условие СТАРТа распространяется и на "Повторный старт", если, конечно же, нет специального примечания. Условия СТАРТа и ОСТАНОВа являются изменение логического уровня на линии SDA, когда на линии SCL присутствует логическая 1 (рисунок 4.1).

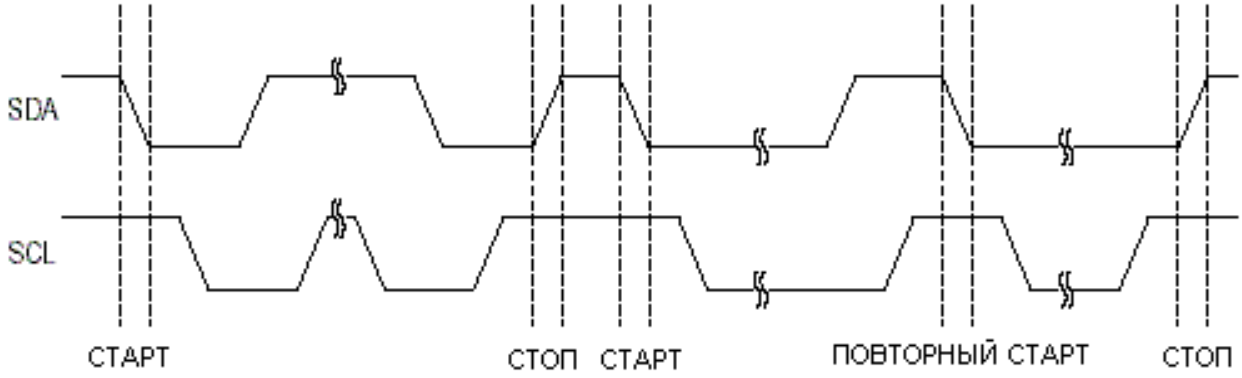


Рисунок 4.1 – Условия СТАРТа, ПОВТОРНОГО СТАРТА и ОСТАНОВа

Все передаваемые адресные пакеты по шине TWI состоят из 9 бит, в т.ч. 7 бит адреса, один бит управления для задания типа операции ЧТЕНИЕ/ЗАПИСЬ и один бит подтверждения. Если бит ЧТЕНИЕ/ЗАПИСЬ = 1, то будет выполнена операция чтения, иначе - запись. Если подчиненный распознает, что к нему происходит адресация, то он должен сформировать низкий уровень на линии SDA на 9-ом цикле SCL (формирование бита подтверждения). Если адресуемое подчиненное устройство занято или по каким-либо другим причинам не может обслужить ведущее устройство, то на линии SDA необходимо оставить высокий уровень во время цикла подтверждения. Ведущий после этого может передать условие ОСТАНОВа или "Повторного старта" для инициации новой передачи. Адресный пакет, состоящий из адреса подчиненного устройства и бита ЧТЕНИЕ или ЗАПИСЬ, обозначим как

ПОДЧИН_АДР+ЧТЕНИЕ или ПОДЧИН_АДР+ЗАПИСЬ, соответственно.

Старший разряд адресного байта передается первым (рисунок 4.2). Нет никаких ограничений на выбор адреса подчиненного устройства, за исключением адреса 0000 000, который зарезервирован для общего вызова.

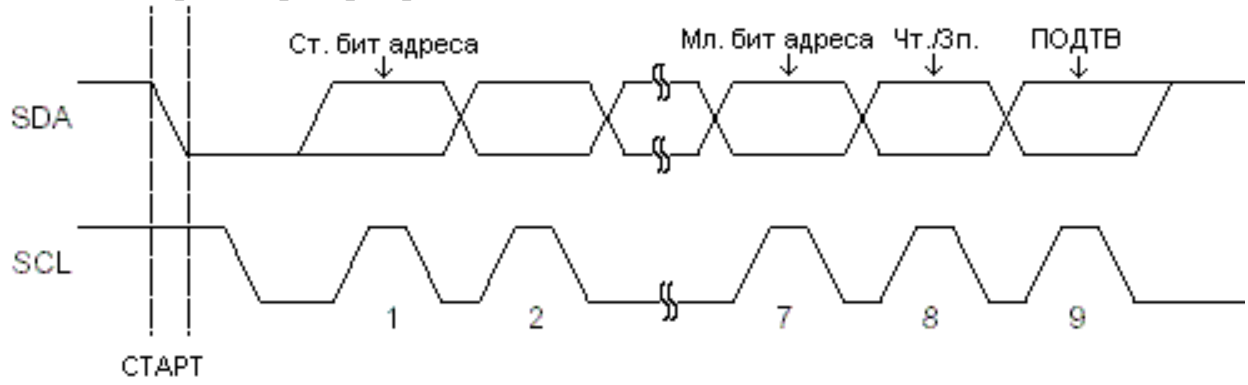


Рисунок 4.2 – Формат адресного пакета

При определении общего вызова все подчиненные устройства должны ответить низким уровнем на линии SDA во время цикла подтверждения (ACK). Общий вызов необходимо использовать, если одно и тоже сообщение необходимо передать от ведущего к нескольким подчиненным устройствам. Если вслед за битом ЗАПИСИ передан адрес общего вызова, то все подчиненные устройства устанавливают низкий уровень на линии SDA для подтверждения общего вызова во время цикла подтверждения. Следующие пакеты данных будут приниматься всеми подчиненными устройствами, которые подтвердили общий вызов. Обратите внимание, что передача адреса общего вызова вслед за битом ЧТЕНИЕ бессмысленна, т.к. одновременное чтение нескольких подчиненных устройств одним ведущим не возможно.

Все адреса с форматом 1111 xx необходимо зарезервировать для будущего использования.

Все пакеты данных, передаваемые по шине TWI, состоят из 9 бит, в т.ч. 1 байт данных и бит подтверждения. Во время передачи данных ведущее устройство генерирует синхронизацию, а также условия СТАРТа и ОСТАНОВа, при этом на приемник возлагается подтверждение приема. Подтверждение (ПОДТВ) сигнализируется приемником выводом низкого уровня на линию SDA во время 9-го такта сигнала SCL. Если приемник оставляет линию SDA в высоком состоянии, то этот сигнализирует о том, что подтверждения не было (НЕТ ПОДТВ). После получения приемником последнего байта или если по каким-либо причинам не имеется возможности далее принимать данные он должен информировать передатчика отправкой бита НЕТ ПОДТВ (нет подтверждения) после последнего байта. Старший бит данных передается первым (рисунок 4.3).

Сеанс связи обычно состоит из условия СТАРТа, ПОДЧИН_АДР+ЧТЕНИЕ/ЗАПИСЬ, одного или более пакетов данных и условия ОСТАНОВа. Передача пустого сообщения, которое состоит из условия СТАРТа, переданного вслед за условием ОСТАНОВа, является недопустимым. Обратите внимание, что монтажное "И" на линии SCL может использоваться для реализации подтверждения связи между ведущим и подчиненным. Подчиненный может продлить низкое состояние на линии SCL путем установки лог. 0 на выводе SCL. Данный способ полезно использовать, если установленная скорость связи мастером является повы-

шенной по отношению к подчиненному или если подчиненному требуется дополнительное время на обработку между приемами данных. Починенный, продлевающий низкое состояние на линии SCL, не будет оказывать влияние на длительность высокого состояния SCL, которая определяется мастером. Как следствие, подчиненный может снизить скорость передачи данных, продлевая рабочий цикл SCL.

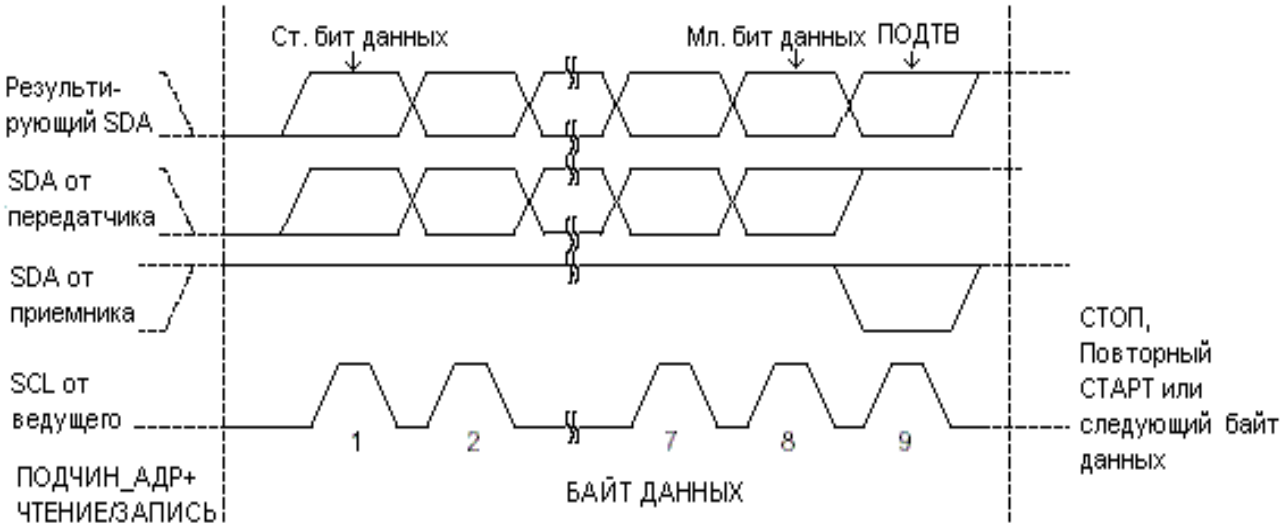


Рисунок 4.3 – Формат пакета данных

На рисунке 4.4 показана типичная передача данных. Обратите внимание, что несколько байт данных могут быть переданы между условиями ПОДЧИН_АДР+ЧТЕНИЕ/ЗАПИСЬ и СТОП в зависимости от программного протокола, реализованного в прикладной программе.

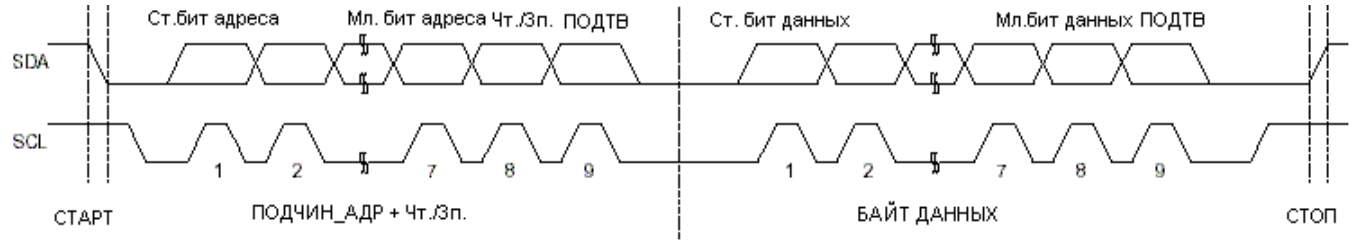


Рисунок 4.4 – Типичная передача данных

2.1. Обзор модуля TWI

Модуль TWI состоит из нескольких подмодулей (рисунок 4.5). Все регистры, выделенные жирной линией, доступны через шину данных микроконтроллера.

Выводы SCL и SDA связывают двухпроводной интерфейс микроконтроллера с остальными микроконтроллерами в системе. Драйверы выходов содержат ограничитель скорости изменения фронтов для выполнения требований к TWI. Входные каскады содержат блок подавления помех, задача которого состоит в игнорировании импульсов длительностью менее 50 нс. Обратите внимание, что к каждой из этих линий можно подключить внутренний подтягивающий резистор путем установки разрядов PORTD.0 (SCL), PORTD.1 (SDA) (см. также "Порты ввода-вывода"). Использование встроенных подтягивающих резисторов в ряде случаев позволяет отказаться от применения внешних.

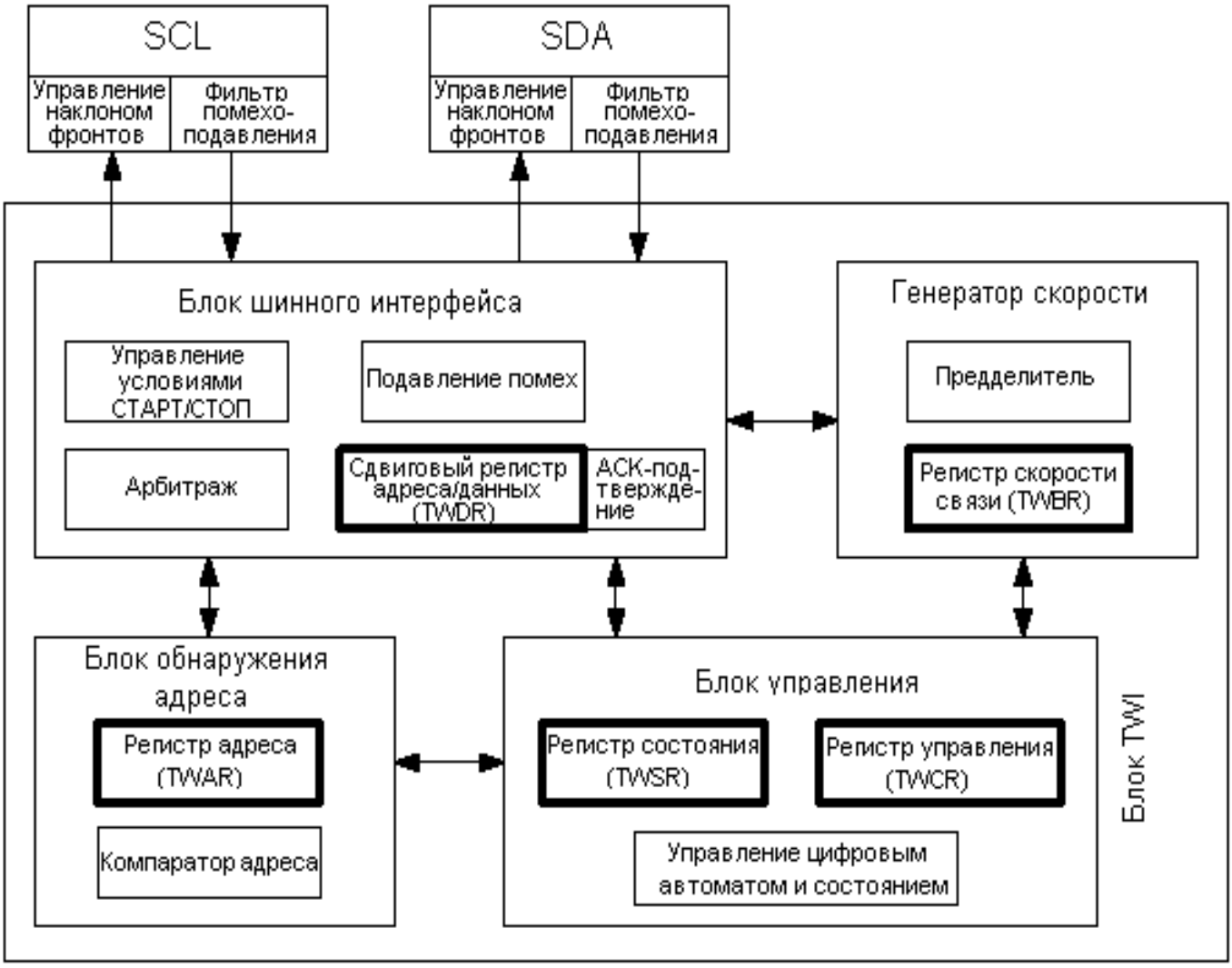


Рисунок 4.5 – Функциональная схема модуля TWI

Блок генератора скорости связи управляет периодом импульсов SCL в режиме ведущего устройства. Период SCL задается регистром скорости TWI (TWBR) и значением бит управления предделителем в регистре состояния TWI (TWSR). В подчиненном режиме значения скорости или установки предделителя не оказывают влияния на работу, но частота синхронизации ЦПУ подчиненного устройства должна быть минимум в 16 раз выше частоты SCL. Обратите внимание, что подчиненные могут продлевать длительность низкого уровня на линии SCL, тем самым уменьшая среднюю частоту синхронизации шины TWI. Частота SCL генерируется в соответствии с выражением (4.1).

$$f_{SCL} = \frac{f_{ЦПУ}}{16 + 2 \cdot (TWBR) \cdot 4^{TWPS}}, \qquad (4.1)$$

где TWBR – значение регистра скорости TWI; TWPS – значение бит предделителя в регистре состояния TWI. TWBR должен быть равен не менее 10, если TWI работает в ведущем режиме. Если TWBR меньше 10, то ведущий может генерировать некорректное состояние на линиях SDA и SCL. Проблема возникает при работе в ведущем режиме при передаче условий СТАРТ+ПОДЧИН_АДР+ ЧТЕНИЕ/ЗАПИСЬ подчиненному.

Блок шинного интерфейса содержит сдвиговый регистр адреса и данных

(TWDR), контроллер СТАРТа/СТОПа и схему арбитрации. TWDR содержит передаваемый байт адреса или данных, или принятый байт адреса или данных. Помимо 8-разр. регистра TWDR в состав блока шинного интерфейса также входит регистр, хранящий значение передаваемого или принятого бита (НЕТ) ПОДТВ. К данному регистру нет прямого доступа со стороны программного обеспечения. Однако во время приема он может устанавливаться или сбрасываться путем манипуляций с регистром управления TWI (TWCR). В режиме передатчика значение принятого бита (НЕТ) ПОДТВ можно определить по значению регистра TWSR.

Контроллер СТАРТа/СТОПа отвечает за генерацию и детекцию условий СТАРТ, ПОВТОРНЫЙ СТАРТ и СТОП. Контроллер СТАРТа/СТОПа позволяет обнаружить условия СТАРТ и СТОП, даже если микроконтроллер находится в одном из режимов сна. Этим обеспечивается возможность пробуждения микроконтроллера по запросу ведущего шины.

Если TWI инициировал передачу в качестве ведущего, то схема арбитрации непрерывно контролирует передачу, определяя возможность дальнейшей передачи. Если TWI теряет арбитрацию, то блок формирует соответствующий сигнал блоку управления, который выполняет адекватные действия и генерирует соответствующий код состояния.

Блок обнаружения адреса проверяет, равен ли принятый адрес значению 7-разр. адреса из регистра TWAR. Если установлен бит разрешения обнаружения общего вызова TWGCE в регистре TWAR, то все входящие адресные биты будут дополнительно сравниваться с адресом общего вызова. При адресном совпадении подается сигнал блоку управления, что позволяет выполнить ему необходимые действия. В зависимости от установки регистра TWCR подтверждение адреса TWI может происходит, а может и нет. Блок обнаружения адреса способен функционировать даже, когда микроконтроллер переведен в режим сна, тем самым позволяя возобновить нормальную работу микроконтроллера по запросу мастера шины. Если при адресном совпадении TWI в экономичном режиме микроконтроллера, т.е. когда инициируется возобновление работы микроконтроллера, возникает другое прерывание (например, INT0), то TWI прекращает работу и возвращается к состоянию холостого хода (Idle). Если возникновение данного эффекта нежелательно, то необходимо следить, чтобы во время обнаружения адресования, когда микроконтроллер находится в режиме выключения (Power-down), было разрешено только одно прерывание.

Блок управления наблюдает за шиной TWI и генерирует отклики в соответствии с установками регистра управления TWI (TWCR). Если на шине TWI возникает событие, которое требует внимания со стороны программы, то устанавливается флаг прерывания TWINT. Следующим тактом обновляется содержимое регистра статуса TWI - TWSR, в котором будет записан код, идентифицирующий возникшее событие. Даная информация хранится в TWSR только тогда, когда установлен флаг прерывания TWI. Остальное время в регистре TWSR содержится специальный код состояния, который информирует о том, что нет информации о состоянии TWI. До тех пор пока установлен флаг TWINT линия SCL остается в низком состоянии. Этим обеспечивается возможность завершить программе все задачи перед продолжением сеанса связи.

3. Задание на исследование

В рамках выполнения лабораторной работы необходимо:

1. Изучить теоретические сведения о работе TWI.

2. Разработать программы для реализации обмена данными по интерфейсу TWI: одна программа для передатчика, вторая — для приемника. Программы должны обеспечивать при нажатии на заданную кнопку первого макета вывод на ЖКИ второго макета требуемой информации (на английском языке). Дополнительные параметры передачи, передаваемые данные и язык программирования приведены в таблице 4.2.

Таблица 4.2 – Варианты заданий на лабораторную работу №8

№ варианта	Режим работы	Язык программирования	№ нажимаемой кнопки	Передаваемая информация
1	2	3	4	5
1	Master-передатчик	Ассемблер	1	Фамилия
	Slave-приемник		2	Номер группы
2	Master-приемник	Ассемблер	3	Фамилия
	Slave-передатчик		4	Дата рождения
3	Master-передатчик	Си	5	Фамилия
	Slave-приемник		6	Номер группы
4	Master-приемник	Си	7	Фамилия
	Slave-передатчик		8	Дата рождения
5	Master-передатчик	Ассемблер	9	Фамилия
	Slave-приемник		10	Номер группы
6	Master-приемник	Ассемблер	11	Фамилия
	Slave-передатчик		12	Дата рождения
7	Master-передатчик	Си	13	Фамилия
	Slave-приемник		14	Номер группы
8	Master-приемник	Си	15	Фамилия
	Slave-передатчик		16	Дата рождения
9	Master-передатчик	Ассемблер	1	Фамилия
	Slave-приемник		2	Дата рождения
10	Master-приемник	Ассемблер	3	Фамилия
	Slave-передатчик		4	Номер группы
11	Master-передатчик	Си	5	Фамилия
	Slave-приемник		6	Дата рождения
12	Master-приемник	Си	7	Фамилия
	Slave-передатчик		8	Номер группы
13	Master-передатчик	Ассемблер	9	Фамилия
	Slave-приемник		10	Дата рождения
14	Master-приемник	Ассемблер	11	Фамилия
	Slave-передатчик		12	Номер группы
15	Master-передатчик	Си	13	Фамилия
	Slave-приемник		14	Дата рождения

Продолжение таблицы 4.2

1	2	3	4	5
16	Master-приемник	Си	15	Фамилия
	Slave-передатчик		16	Номер группы
17	Master-передатчик	Ассемблер	1	Фамилия
	Slave-приемник		2	Дата рождения
18	Master-приемник	Ассемблер	3	Фамилия
	Slave-передатчик		4	Номер группы
19	Master-передатчик	Си	5	Фамилия
	Slave-приемник		6	Дата рождения
20	Master-приемник	Си	7	Фамилия
	Slave-передатчик		8	Номер группы

3. Произвести отладку программ в AVR-Studio и на макетах CAN128.

4. Исследовать на основе полученных программ принципы передачи данных по TWI.

4. Содержание отчёта

1. Цель работы.
2. Постановка задачи (с указанием данных конкретного варианта).
3. Расчет необходимых параметров.
4. Текст программы.
5. Описание программы.
6. Полученные осциллограммы.
7. Выводы.

5. Контрольные вопросы

1. Перечислите последовательные интерфейсы, реализованные в микроконтроллерах AVR. Приведите их сравнительный анализ.
2. Каковы принципы последовательной передачи данных?
3. Каковы отличительные особенности интерфейса TWI?
4. Приведите временные диаграммы передачи данных по интерфейсу TWI.
5. Какие существуют режимы работы TWI?
6. Каков формат посылки TWI?
7. Приведите схему подключения устройств по интерфейсу TWI.
8. Какова структура блока TWI?
9. Каково назначение интерфейса TWI?
10. Перечислите особенности реализации TWI на макете CAN128.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Евстифеев А.В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. — М.: Издательский дом «Додэка-XXI», 2007. — 592с.

2. Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы «ATMEL» — 3-е изд., стер. — М.: Издательский дом «Додэка-XXI», 2006. — 288с.

3. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «ATMEL» — М.: Издательский дом «Додэка-XXI», 2004. — 560с.

4. 8-bit Microcontroller with 128K Bytes of ISP Flash and CAN Controller AT90CAN128 [Электронный ресурс]. — Электрон. текстовые данные (5 742 631 bytes). — Режим доступа: http://www.atmel.com/dyn/resources/prod_docments/doc7522.pdf Tuesday, 17 March 2009, 12:06:48.

5. Программирование на языке С для AVR и PIC микроконтроллеров./Сост. Ю.А.Шпак — К.: «МК-Пресс», 2006. — 400 с.

6. Федорков Б.Г. Микросхемы ЦАП и АЦП: функционирование, параметры, применение. / Б.Г. Федорков, В.А. Телец – М.: Энергоатомиздат, 1990. – 320 с.

ПРИЛОЖЕНИЕ А
(справочное)
Пример программы для ЦАП

Данная программа генерирует пилообразный сигнал на частоте 8 МГц с периодом 0,65 мс. Для адекватной работы программы на макете CAN128 необходимо выполнить коммутацию PB0 с CS, PB1 с SCK, PB2 с SDI, PB4 с LDAC.

```
.include "can128def.inc"

start:
; Инициализация стека
  ldi    16,low(ramend)
  out    PL,r16
  ldi    16,high(ramend)
  out    PH,r16
// ----------------
  ldi    XL,0xFF
  ldi    XH,0x0F

loop:
  call   DAC
  adiw   XL,63
  rjmp   loop
  jmp    start
// данные в регистре r18
  led_out:
  push   r16
  // На вывод
  ldi    r16,0xFF
  out    DDRA,r16
  out    PORTA,r18
  // Защёлка
  sbi    DDRG,3
  sbi    PORTG,3
  cbi    PORTG,3
// ----------------
  pop    r16
  ret
// Функия задержки
  delay:
  push   r18
  push   r19
  push   r20
  ldi    r18,0xFF
  ldi    r19,0xFF
  ldi    r20,0x03
dd:
  dec    r18
  brne   dd
  dec    r19
  brne   dd
  dec    r20
  brne   dd
  pop    r20
  pop    r19
  pop    r18
  ret
//-------------------
.equ CS       = 0
.equ SCK      = 1
.equ SDI      = 2
.equ LDAC     = 4
// Функция вывода на ЦАП
DAC:
  push   r16
  ldi 16,(1<<CS)+(1<<SDI)+
  (1<<LDAC)+(1<<SCK)
  out    DDRB,r16
  ldi    r16,(1<<SPE)+(1<<MSTR)
  out    SPCR,r16
  ldi    r16,(1<<SPI2X)
  out    SPSR,r16
  ori    XH,0b00110000
  andi   XH,0b00111111
  sbi    PORTB,LDAC
  cbi    PORTB,CS

  out    SPDR,XH
waiting1:
  inr16,SPSR
  sbrs   r16,SPIF
  rjmp   waiting1

  out    SPDR,XL
waiting2:
  inr16,SPSR
  sbrs   r16,SPIF
  rjmp   waiting2
  sbi    PORTB,CS
  cbi    PORTB,LDAC
  pop    r16

  ret
```

ПРИЛОЖЕНИЕ Б
(справочное)
Примеры реализации программ обмена по интерфейсу УСАПП

Передатчик

```
.include "can128def.inc"
start:
     ; Инициализация стека
     ldi       r16,low(ramend)
     out       SPL,r16
     ldi       r16,high(ramend)
     out       SPH,r16

     ldi       r16,0xFF
     out       DDRA,r16
loop:
     call Master
     out  PORTA,r18
     // Защёлка
     sbi       DDRG,3
     sbi       PORTG,3
     cbi       PORTG,3
     inc       r18
     call delay
     rjmp loop
     jmp       start
// Функия задержки
delay:
     push r18
     push r19
     ldi       r18,0xFF
     ldi       r19,0x22
dd:
     dec       r18
     brne dd
     dec       r19
     brne dd
     pop       r19
     pop       r18
     ret
Master:
     // Задаём частоту работы
     ldi       r16,0x00
     sts       UBRR1H,r16
    ldi        r16,51
     sts       UBRR1L,r16
     // Разрешаем передатчик
     ldi
     r16,(1<<TXEN1)//(1<<RXEN1)+
     sts       UCSR1B,r16
// Режим 8 бит
     ldi       r16,6
     sts       UCSR1C,r16
wait:
     lds       r16,UCSR1A
     sbrs r16,UDRE1
     rjmp wait
     sts       UDR1,r18
     ret
```

Приемник

```
.include "can128def.inc"
start:
     ; Инициализация стека
     ldi       r16,low(ramend)
     out       SPL,r16
     ldi       r16,high(ramend)
     out       SPH,r16
     ldi       r16,0xFF
     out       DDRA,r16
loop:
     call Slave
     out       PORTA,r18
// Защёлка
     sbi       DDRG,3
     sbi       PORTG,3
     cbi       PORTG,3
     rjmp loop
     jmp       start
Slave:
     // Задаём частоту работы
     ldi       r16,0x00
     sts       UBRR1H,r16
     ldi       r16,51
     sts       UBRR1L,r16
     // Разрешаем передатчик
     ldi       r16,(1<<RXEN1)
     sts       UCSR1B,r16
     // Режим 8 бит
     ldi       r16,6
     sts       UCSR1C,r16
wait:
     lds       r16,UCSR1A
     sbrs       r16,RXC1
     rjmp       wait
     lds       r18,UDR1
     ret
```

ПРИЛОЖЕНИЕ В
(справочное)
Примеры реализации программ обмена по интерфейсу TWI

Режим «Master – передатчик»

```
.include "can128def.inc"

start:
     ; Инициализация стека
     ldi  r16,low(ramend)
     out  SPL,r16
     ldi  r16,high(ramend)
     out  SPH,r16
;====================
     ldi  r18,0x01

     ldi  r16,0xFF
     out  DDRA,r16
loop:
     call delay
     call Write
     out  PORTA,r18
     // Защёлка
     sbi  DDRG,3
     sbi  PORTG,3
     cbi  PORTG,3
     inc  r18
     rjmp loop

     jmp  start
// Функия задержки
delay:
     push r18
     push r19
     ldi  r18,0xFF
     ldi  r19,0x8F
dd:  dec  r18
     brne dd
     dec  r19
     brne dd
     pop  r19
     pop  r18
     ret
//-------------------
Write:
     ldi  r16,(1<<PUD)
     out  MCUCR,r16
     ldi  r16,0xFF
     out  DDRD,r16
; Разрешение TWI и СТАРТ
     ldi  r16,(1<<TWEN)+
     (1<<TWSTA)+(1<<TWINT)
     sts  TWCR,r16
wait1:   // Ждём
     lds  r16,TWCR
     sbrs r16,TWINT
     rjmp wait1
// load SLA_W
     ldi  r16,0x00
     sts  TWDR,r16
     ldi  r16,(1<<TWEN)+
     (1<<TWINT)
     sts  TWCR,r16
wait2:   // Ждём
     lds  r16,TWCR
     sbrs r16,TWINT
     rjmp wait2
// load DATA
     sts  TWDR,r18
     ldi  r16,(1<<TWINT)+
     (1<<TWEN)
     sts  TWCR,r16
wait3:   // Ждём
     lds  r16,TWCR
     sbrs r16,TWINT
     rjmp wait3
; СТОП
     ldi  r16,(1<<TWEN)+
     (1<<TWSTO)+ (1<<TWINT)
     sts  TWCR,r16

     ret
```

Режим «Master – приемник»

```
.include "can128def.inc"

start:
     ; Инициализация стека
     ldi  r16,low(ramend)
     out  SPL,r16
     ldi  r16,high(ramend)
     out  SPH,r16
     ;====================
     ldi  r16,0xFF
     out  DDRA,r16
loop:
     call delay
     call Write
     out  PORTA,r18
     // Защёлка
     sbi  DDRG,3
     sbi  PORTG,3
     cbi  PORTG,3
     rjmp loop
     jmp  start
// Функия задержки
delay:
     push r18
     push r19
     ldi  r18,0xFF
     ldi  r19,0x8F
dd:
     dec  r18
     brne dd
     dec  r19
     brne dd
     pop  r19
     pop  r18
     ret
//--------------------
Write:
     ldi  r16,(1<<PUD)
     out  MCUCR,r16
     ldi  r16,0xFF
     out  DDRD,r16
; Разрешение TWI и СТАРТ
     ldi       r16,(1<<TWEN)+
    (1<<TWSTA)+1<<TWINT)
     sts  TWCR,r16
wait1:   // Ждём
     lds  r16,TWCR
     sbrs r16,TWINT
    rjmp wait1
// load SLA_R
     ldi  r16,0x01
      sts TWDR,r16
     ldi  r16,(1<<TWEN)+
    (1<<TWINT)
     sts  TWCR,r16
wait2:   // Ждём
     lds  r16,TWCR
     sbrs r16,TWINT
     rjmp wait2
     ldi  r16,(1<<TWEN)+
    (1<<TWINT)
     sts  TWCR,r16
wait3:   // Ждём
     lds  r16,TWCR
     sbrs r16,TWINT
     rjmp wait3
// load DATA
     lds  r18,TWDR
     ; СТОП
     ldi  r16,(1<<TWEN)+
(1<<TWSTO)+(1<<TWINT)
     sts  TWCR,r16


     ret
```

Режим «Slave – передатчик»

```
.include "can128def.inc"

start:
     ; Инициализация стека
     ldi       r16,low(ramend)
     out       SPL,r16
     ldi       r16,high(ramend)
     out       SPH,r16

     ;====================
     ldi       r16,0xFF
     out       DDRA,r16

loop:
     call Read
     out       PORTA,r18
     // Защёлка
     sbi       DDRG,3
     sbi       PORTG,3
     cbi       PORTG,3
     rjmp loop

     jmp       start

Read:
     ldi       r16,(1<<PUD)
     out       MCUCR,r16
     ldi       r16,0xFF
     out       DDRD,r16

;Бит разрешения обнаружения об-
щего вызова по шине TWI
     ldi       r16,(1<<TWGCE)
     sts       TWAR,r16

; Разрешение TWI
     ldi       r16,(1<<TWEN)+
    (1<<TWEA)+(1<<TWINT)
     sts       TWCR,r16
wait1:   // Ждём
     lds       r16,TWCR
     sbrs r16,TWINT
     rjmp      wait1

; Разрешение TWI
     ldi       r16,(1<<TWEN)+
    (1<<TWEA)+(1<<TWINT)
     sts       TWCR,r16
wait2:   // Ждём
     lds       r16,TWCR
     sbrs      r16,TWINT
     rjmp      wait2
     lds       r18,TWDR

     ldi       r16,(1<<TWINT)
     sts       TWCR,r16

     ret
```

Режим «Slave – приемник»

```
.include "can128def.inc"

start:

; Инициализация стека
     ldi       r16,low(ramend)
     out       SPL,r16
     ldi       r16,high(ramend)
     out       SPH,r16
     ;====================
     ldi       r16,0xFF
     out       DDRA,r16

loop:
     call Read
     out       PORTA,r18

// Защёлка
     sbi       DDRG,3
     sbi       PORTG,3
     cbi       PORTG,3
     inc       r18
     rjmp loop

     jmp       start

Read:
     ldi       r16,(1<<PUD)
     out       MCUCR,r16
     ldi       r16,0xFF
     out       DDRD,r16

;Бит разрешения обнаружения об-
щего вызова по шине TWI
     ldi       r16,(1<<TWGCE)
     sts       TWAR,r16

; Разрешение TWI
     ldi       r16,(1<<TWEN)+
     (1<<TWEA)+(1<<TWINT)
     sts       TWCR,r16
wait1:   // Ждём
     lds       r16,TWCR
     sbrs r16,TWINT
     rjmp      wait1

     sts       TWDR,r18

     ;  TWI
      ldi      r16,(1<<TWEN)+
     (1<<TWEA)+(1<<TWINT)
     sts       TWCR,r16

wait3:   // Ждём
     lds       r16,TWCR
     sbrs r16,TWINT
     rjmp      wait3

     ldi       r16,(1<<TWINT)
     sts       TWCR,r16

     ret
```

Для заметок

Заказ № _____ от «_____» _____ 20___ г. Тираж _____ экз.
Изд-во СевНТУ