Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ОТЧЕТ
по лабораторной работе №4

Организация межпроцессорного обмена

(тема работы)

Микропроцессорные системы

(наименование дисциплины)

Работу выполнили:

Волкова М.Д.
Ерниязов Т.Е.

подпись Ф.И.О.

Преподаватель:

Кузьмин А.А.

подпись Ф.И.О.

Санкт-Петербург
2017

1. Цель работы

- Ознакомление с принципами организации обменов по последовательному каналу;
- Приобретение навыков создания коммуникационных протоколов последовательной связи;
- Знакомство с организацией межпроцессорных обменов.

2. Программа работы

- Изучить программу send_rec.asm, реализующую приём-передачу данных по последовательному каналу между инструментальной ЭВМ и МК.
- Модифицировать программу send_rec.asm, дополнив её командой cpl a, выполняемой после команды lcall receive.
- Разработать и выполнить программу rec_lcd, реализующую вывод на экран ЖКИ текста, передаваемого с инструментальной ЭВМ (используя вкладку «Терминал» среды Shell51).
- Разработать и выполнить программу статистической обработки данных.
- Разработать и выполнить программу обмена информацией между соседними микроконтроллерными стендами (2n-1) и 2n, где n=1,2,3,4.
- Модифицировать программу обмена информацией между соседними микроконтроллерными стендами для реализации межконтроллерного обмена информацией по последовательному каналу в режиме «Master-Slave».

3. Теоретические сведения

Данная работа направлена на изучение принципов построения межпроцессорного взаимодействия на основе последовательного порта.

Основой данного порта является универсальный асинхронный приемопередатчик (УАПП), который осуществляет прием и передачу информации, представленной последовательным кодом, в полном дуплексном режиме обмена. В состав УАПП входят принимающий и передающий сдвигающие регистры, а также специальный буферный регистр (SBUF с адресом 99_{16}) для хранения передаваемой или получаемой информации. Запись байта в буферный регистр со стороны МК предполагает автоматическую перегрузку данного байта в сдвигающий регистр передатчика и инициирует начало передачи байта. Также буферный регистр используется и приемником, в этом случае его задачей является обеспечение независимой операции считывания принятого байта с операцией приема очередного данного на сдвигающий регистр, то есть, по сути, освобождает сдвигающий регистр от функций хранения информации, оставляя за ним только функцию компоновки битов одного переданного байта. Такой механизм хранения информации рассчитан только на один цикл, что означает, если по каким-нибудь причинам байт из буферного регистра не был считан до окончания приема очередного данного, то он

перезаписывается новым значением, то есть теряется. Таким образом, необходимо четкая настройка системы управления последовательным портом внутри самого МК.

Помимо внутренних настроек, существуют еще и внешние, необходимые для корректной работы двух (или более) устройств обменивающихся по данному последовательному каналу связи. Они заключаются в необходимости согласования скоростей передачи информации как в ту, так и в другую сторону. Основным условием корректной работы является синхронизация передаваемых данных. Так как имеющийся в МК-51 последовательный порт может работать в четырех режимах, где три из них асинхронные, то система выбора синхронизации будет различна для этих случаев. При синхронной передаче одна из линий приемопередатчика (для МК-51 это линия TxD) выделяется под синхроимпульсы, а по второй передаются данные, причем направление передачи синхроимпульсов не зависит от направления передачи данных (синхроимпульсы постоянно передаются только в выбранную сторону).

При асинхронном обмене, естественно никаких синхроимпульсов не подается, но синхронизация необходима, для этого используется старт-стопный механизм. То есть в начале посылки передается запрос на передачу, после чего ожидается ответ от приемника на разрешение данного процесса. При получении ответа осуществляется непосредственная передача информации, причем первым передается стартовый импульс, а последним - стоповый. В промежутке между ними передаются биты одного байта, при этом возможен примитивный механизм обнаружения ошибок добавление бита четности/ нечетности. Введение данного бита уменьшает скоростные параметры канала связи, так как передается дополнительный бит, но улучшает точность получаемой информации, поскольку при нарушении бита четности выставляется сигнал ошибки и передача повторяется.

Для настройки выбранного режима последовательного порта в МК-51 используется специальный регистр управления SCON (98_{16}), битовая структура которого приведена ниже:

Таблица 1

9Fh	9Eh	9Dh	9Ch	9Bh	9Ah	99h	98h	SCON
SM0	SM1	SM2	REN	TB8	RB8	TI	RI	Адрес 98h

Бит	Назначение
SM1 SM0	Биты задание режимов работы SP
00	режим синхронной двухпроводной передачи
01	режим 8-битового УАПП с переменной скоростью передачи
10	режим 9-битового УАПП с фиксированной скоростью передачи
11	режим 9-битового УАПП с переменной скоростью передачи
SM2	Бит разрешение многопроцессорной работы В асинхронных режимах работы при SM2 = 1 флаг RI не активизируется, если принятый бит данных RB8 равен 0 (режимы 2 и 3) или если не принят стоп-бит (режим 1). Бит SM2 не влияет на работу порта SP в синхронном режиме 0.
REN	Бит разрешения приема последовательных данных.
TB8	9-й бит передаваемых данных в режимах 2 и 3.
RB8	9-й бит принимаемых данных в режимах 2 и 3. В режиме 1 при SM2 = 0 RB8 является принятым стоп-битом.
TI	Флаг прерывания передатчика. Бит TI устанавливается аппаратно в конце выдачи 8-го бита в режиме 0 или в начале стоп-бита в других режимах. Сбрасывается программно.
RI	Флаг прерывания приемника. Бит RI устанавливается аппаратно при приеме 8-го бита в режиме 0 или в середине стоп-бита в других режимах. Сбрасывается программно.

Как видно, регистр SCON содержит не только управляющие биты, определяющие режим работы последовательного порта, но и девятый (контрольный) бит принимаемых или передаваемых данных, а также биты прерывания приемопередатчика.

Настройка определенной скорости передачи данных должна быть выполнена точно на всех устройствах участвующих в обмене через заданный последовательный порт, так как при асинхронной работе нет синхросигнала, а биты старта и стопа должны быть своевременно распознаны приемным устройством. Поскольку в противном случае возможен пропуск битов или двойное считывание одного и того же бита информации, то есть растянутая или сжатая передача, из-за несоответствия длительности битовых импульсов. Настройка скорости передачи не возможна без точного соответствия внутренних тактовых генераторов устройств («часов» системы), участвующих в обмене. Это еще одна составная часть процедуры обеспечения синхронизации приемопередачи, необходимой для корректного обмена устройств, при работе с последовательным портом в асинхронном режиме.

Для микроконтроллера МК-51 скорость передачи данных задается для каждого режима по своим определенным правилам. Для синхронного режима частота выдачи бит в последовательный порт равна тактовой частоте МК (1 МГц), то есть скорость постоянна и составляет 1 Мбит/с. Для асинхронного режима с фиксированной скоростью, частота выдачи/приема битов информации имеет два предопределенных значения 187,5 кГц и 375 кГц. Выбор одной из этих частот осуществляется заданием значения бита SMOD, находящегося в 7 разряде регистра управления питанием PCON (87_{16}), при этом если SMOD = 1, то скорость составляет 375 Кбит/с, а при SMOD = 0 - 187,5 Кбит/с.

В режимах с изменяемой скоростью обмена частота высчитывается в соответствии с формулой:

Здесь F - фактическая частота обмена; SMOD - значение 7 бита PCON (см. выше); $F_{рез}$ - частота резонатора; [TH1] - значение регистра старшего бита таймера/счетчика 1. Из формулы понятно, каким образом производится изменение скорости обмена - путем задания необходимой переменной перегрузки в таймер/счетчик 1 контроллера. Для большинства стандартных скоростей обмена таймер 1 устанавливается в режим 8-битного автоперегружателя, поскольку 8-ми бит для задания нужного периода вполне хватает. Ниже приведены значения переменных перезагрузки соответствующих стандартным скоростям обмена (частотам обмена) информацией по последовательным портам:

Таблица 2

Частота обмена	**Значение SMOD**	**Настройки таймера 1**	
		Режим (SM1 SM0)	**Константа перезагрузки**
62,5 кГц	1	10	$0FF_{16}$
19,2 кГц	1	10	$0FD_{16}$
9,6 кГц	0	10	$0FD_{16}$
4,8 кГц	0	10	$0FA_{16}$
2,4 кГц	0	10	$0F4_{16}$
1,2 кГц	0	10	$0E8_{16}$

Приведенные выше расчеты справедливы для частоты резонатора равной не 12 МГц (как в стандартном МК-51), а 11,0592 МГц, это обусловлено тем, фактом, что точные значения стандартных частот обмена для последовательного порта могут быть получены только при таком значении частоты резонатора. Этот факт подтверждают простейшие математические выкладки:

Таким образом, для частоты 19,2 кГц при $F_{рез}$ = 12 МГц [TH1] = $252{,}745_{10}$, возьмем приближенное значение [TH1] = 253_{10}, тогда формула для частоты резонатора имеет следующий вид:

В итоге для приведенных выше значений частота резонатора равняется $F_{рез}$ = 11,0592 МГц. Однако если рассматривать частоту обмена равную 62,5 кГц, то значение [TH1] = 255_{10} получается точно только при частоте резонатора равной 12 МГц.

В результате можно сделать вывод, что приведенные в таблице значения переменной перегрузки TH1 являются ориентировочными и требуют дополнительной настойки под конкретную задачу. Смысл настройки складывается из последовательного перебора ближайших значений TH1, взятых из таблицы для интересующей скорости обмена, и наблюдения корректности приема/передачи данных между устройствами, причем необходимо проделать испытание данного состоящего из всех 1, то есть максимального для заданной разрядности (проверка запаздывания приема, первый байт (старший) может быть не считан) и числа отличающегося от него на единицу младшего разряда (проверка опережения приема данных, последний байт (младший) не считан).

Вкладка «Терминал»:

Предназначена для исследования последовательного интерфейса, стыкующего МК и ЭВМ, без участия связных компонент программы-монитора, и построения пользовательских связных компонент (рис. 2).

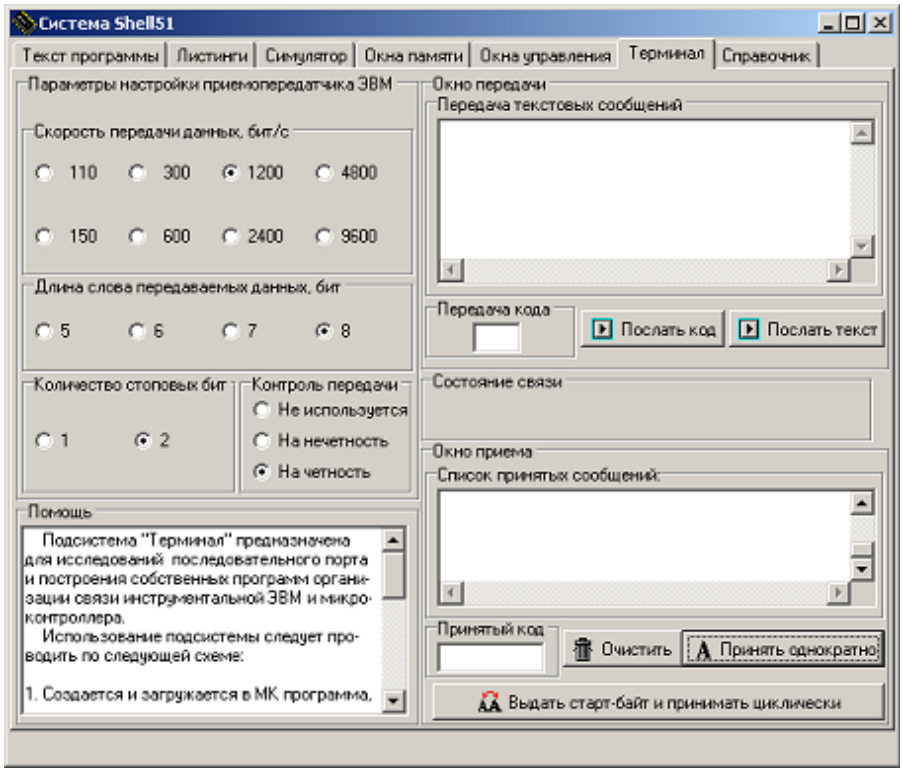

Рис. 2. Вкладка "Терминал"

Вкладка содержит: поле настроек последовательного приемопередатчика инструментальной ЭВМ (слева вверху), окно справки (слева внизу), поле окна передачи (справа вверху), поле окна приема (справа внизу) и поле состояния связи (справа в центре). Параметры протокола представлены на рис. 2.

Используя переключатели поля настроек приемопередатчика ЭВМ, пользователь задает необходимые режимы его работы (аналогичные значения скорости передачи, длины посылки и т.д., должны быть заданы и для приемопередатчика МК).

Следуя логике программы, функционирующей на МК (созданной и загруженной в МК штатными средствами системы), пользователь применяет следующие варианты работы с подсистемой:

1. Посылка информации в порт:

- если посылка однобайтная, используется окно передачи шестнадцатеричных кодов и панель-кнопка "Послать код", активизирующая выдачу данного значения в последовательный порт на МК;
- если посылка многобайтная, целесообразно использовать окно передачи текста, предварительно введенного с клавиатуры ЭВМ, и панель-кнопка "Послать текст", при этом производится выдача кодов символов, составляющих текст.

2. Прием информации из порта:

- при активизации кнопки-панели "Принять однократно", код, содержащийся в буфере приемника ЭВМ, будет отображен в окне "Принятый код", а также отображен как очередной эквивалентный символ в окне "Принятый текст". Кнопка-панель "Очистить" позволяет освободить окна приема от содержащейся в них информации;
- при активизации кнопки-панели "Выдать старт-байт и принимать циклически" система вышлет в последовательный порт код A516, а затем перейдет к циклу ожидания поступления входной информации из последовательного порта. Указанный старт-байт может быть использован программистом микроконтроллера как уведомление о необходимости передавать результаты в ЭВМ.

3. Комбинация вышеуказанных вариантов.

При наличии ошибок в логике взаимодействия пользователя и программы МК возможно нарушение обмена, обозначаемое сообщением "Нет связи" в поле статуса линии. Необходимо выявить и устранить причину, после чего разблокировать приемопередатчик путем двойного щелчка манипулятором "мышь" на указанном сообщении.

1. Программа простого обмена по последовательному порту

Переде посылкой байтов нужно было настроить параметры обмена: скорость 1200 бит/с, длина слова по 8 бит, контроль передачи на чётность, 2 стоповых бита. Далее можно послать байт с терминала, который будет отправлен микроконтроллером обратно. Таким образом, осуществлялась пересылка байта информации от терминала к МК и обратно.

Алгоритм:

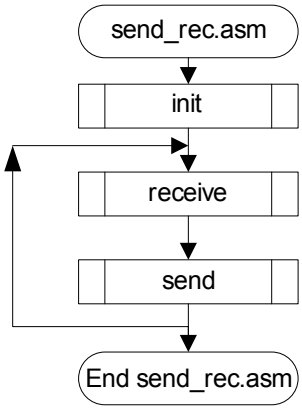

Рис. 1. Алгоритм программы send_rec.asm

Код программы:

send_rec.asm

```
      org 8100h
      lcall init
work:       lcall receive
      lcall send
      sjmp work

init: clr tr1                 ;останов таймера T/C1
;задание режима работы SP
      mov scon, #11010010b
      anl tmod, #0Fh          ;выбор режима работы таймера T/C1
      orl tmod, #00100000b    ;8-битный автоперезагружаемый счетчик

      anl D8h, #7Fh           ;сброс бита BD в регистре ADCON
      anl 87h, #7Fh           ; сброс бита SMOD в регистре PCON
      mov th1, #e6h           ;скорость обмена 1200 бит/с
      setb tr1                ;разрешение работы таймера T/C1
      ret

receive:    jnb ri, receive         ;ожидание завершения приема
(установка флага RI)
      mov a, sbuf       ;перепись принятого байта в аккумуляторе
      clr ri                  ;сброс флага RI
      ret

send: jnb ti, send            ;ожидания завершения передачи (установка
флага TI)
      mov sbuf, a
      clr ti
ret
```

Результаты выполнения:

В результате выполнения программы была успешно осуществлена передача кода из ЭВМ в МК и обратно.

2. Модифицированная программа send_rec

По заданию необходимо дополнить программу приёма-передачи данных по последовательному каналу между инструментальной ЭВМ и МК командой cpl A, выполняемой после команды lcall receive (перед командой lcall send).
Данное дополнение приведёт к тому, что при посылке данных через «Терминал» мы будем наблюдать инверсные значения при приёме этих данных.

Код программы:

send_rec.asm

```
        org 8100h
        lcall init
work:       lcall receive
        cpl A
        lcall send
        sjmp work

init: clr tr1

        mov scon, #11010010b
        anl tmod, #0Fh
        orl tmod, #00100000b

        anl D8h, #7Fh
        anl 87h, #7Fh
        mov th1, #e6h
        setb tr1
        ret

receive:    jnb ri, receive
        mov a, sbuf
        clr ri
        ret

send: jnb ti, send
        mov sbuf, a
        clr ti
        ret
```

Результаты выполнения:

В ходе выполнения программы мы посылаем значения через «Терминал» в микроконтроллер с помощью кнопки «Послать код», а затем получаем инвертированные значения при нажатии на кнопку «Принять однократно».

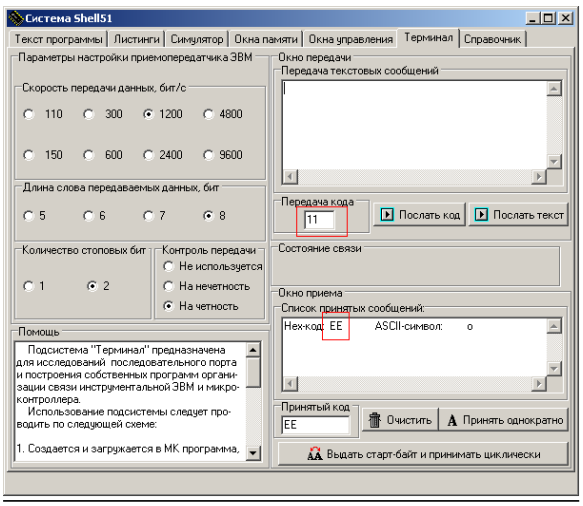

Рис. 2. Передачи и приём значений через «Терминал»

3. Программа вывода на экран ЖКИ текста, передаваемого с инструментальной ЭВМ

Схема соединения МК с ЭВМ и блоком ЖКИ показана на рис. 3.

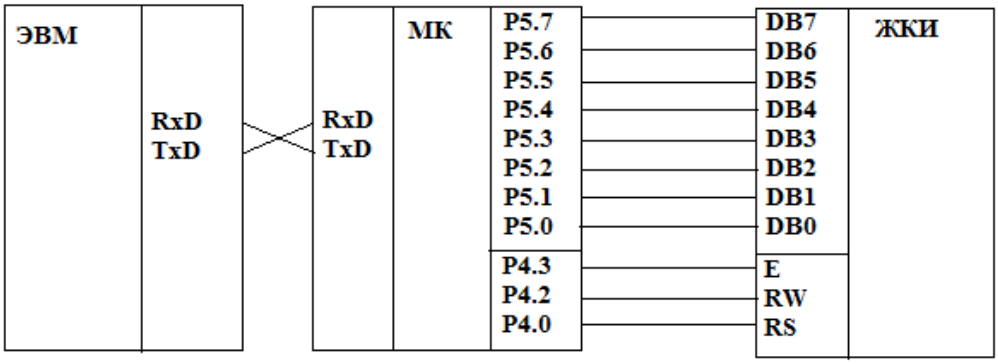

Рис. 3. Схема информационных связей

Передаваемый текст вводится в окно «Передача текстовых сообщений». Строка символов заканчивается символом конца посылки – ‘.’. При нажатии кнопки «послать текст» подготовленное сообщение по последовательному каналу передается в МК и записывается в видеобуфер, при этом контролируется символ конца посылки и число отображенных символов. Ограничений на число символов в посылке нет. По окончанию приема посылки содержимое принятой посылки отображается на экране ЖКИ, начиная с первого знакоместа экрана. При завершении вывода на экран ЖКИ управление передается подпрограмме приема последовательных данных, которая ожидает поступления следующей посылки.

В отличие от предыдущих двух программ, в программе rec_lcd получение данных и их сохранение в памяти выполняется по прерыванию от последовательного порта. Если получен конец посылки, то обработчик устанавливает флаг завершения приема, основная программа анализирует его и отображает на экран полученные данные. Таким образом основную программу очень проста, но время выполнения обработчика прерываний от последовательного порта довольно большое. Так как в программе используется также таймер T/C0, который играет роль сторожевого таймера, запросам прерываний от него был присвоен высший приоритет.

Работа сторожевого таймера заключается в следующем: таймер запускается при поступлении первого байта посылки. Его сброс и запрет счета выполняется при обнаружении символа конца посылки. Если в течение одной секунды после начала приема символ конца посылки не был принят, то таймер формирует запрос прерывания, что сигнализирует о нарушении протокола обмена. Обработчик прерывания от таймера устанавливает флаг ошибки, который анализирует основная программа и выводит на экран сообщение «Protocol Error».

Алгоритм работы программы представлен на рис. 4:

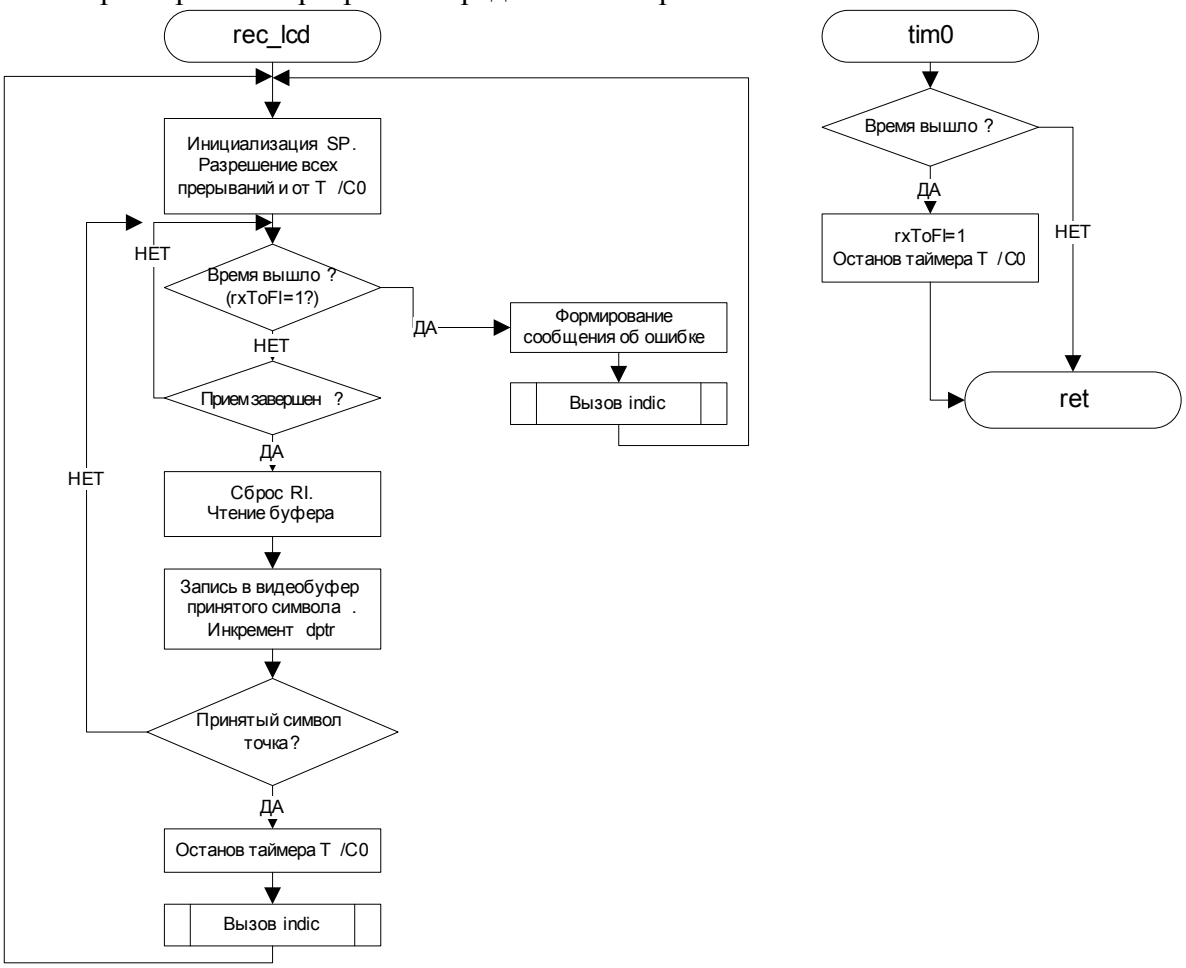

Рис. 4. Алгоритм программы вывода на экран ЖКИ принятого текста

Код программы:

```
org 8800h
sjmp main
rxBuf:      equ FFD0h
rxToFl:     equ  0
p4:   equ E8h
p5:   equ F8h

main:
      lcall init
rxStart:
      mov dptr,#rxBuf         ;задание начальн. адреса видеобуфера
      mov r6,#40h
      mov a,#20h
hole1:      movx @dptr,a            ;заполнение пробелами
      inc dptr
      djnz r6,hole1
wait1:
      jnb ri,wait1
      mov a,sbuf
      clr ri
      mov r5,#3
      setb tr0          ;разрешение работы T/C0
      mov dptr,#rxBuf
wr2Vbuf:
      movx @dptr,a      ;запись в видеобуфер принятого символа
      inc dptr
      clr c
      subb a, #2Eh      ;проверка является ли символ точкой
      jnz receive       ;если не точка, то продолжить прием
      dec dptr
      mov a,#20h
      movx @dptr,a      ;вместо принятой точки записать пробел
      clr tr0           ; останов таймера T/C0
      lcall indic       ;вызов подпрограммы индикации
      ljmp main
receive:
      jb rxToFl,rxTout        ;если время вышло, то вывести ошибку
      jnb ri,receive          ; ожидание завершения приема (установка
флага RI)
      clr ri
      mov a,sbuf        ;перепись принятого байта в аккумуляторе
      mov sbuf, a
      ljmp wr2Vbuf;

init:
      clr tr1                 ;останов таймера T/C1
      ;задание режима работы SP
      mov scon,#11010010b     ;9-битный асинхронный режим  с переменной
скоростью и устан. Флагом TI

      ;задание скорости работы SP осуществляется при программировании
таймера T/C1
      anl tmod,#0Fh           ;выбор режима работы таймера T/C1
      orl tmod, #00100000b    ; 8-битный автоперезагружаемый счетчик

      anl D8h,#7Fh            ;сброс бита BD в регистре ADCON
      anl 87h,#7Fh            ; сброс бита SMOD в регистре PCON
      mov th1,#e6h            ;скорость обмена 1200 бит/с
      setb tr1                ;разрешение работы таймера T/C1
      clr rxToFl

      anl TMOD, #F0h
```

```
        orl TMOD, #01h
        mov TH0, #00h
        mov TL0, #00h
        clr tr0                 ;останов таймера T/C0
        setb ea                 ;разрешение всех прерываний
        setb et0                ; разрешение прерываний T/C0
        ret

send:   jnb ti,send
        mov sbuf,a
        clr ti
        ret

tim0:
        cpl p1.1
        djnz r5,tim0ret
        setb rxToFl       ;время вышло, установка бита
        clr tr0                 ;останов таймера
tim0ret:
        reti

;Сообщение об ошибке
rxTout:
        mov dptr,#str1
        mov a,#4fh        ; О
        movx @dptr,a
        inc dptr
        mov a,#ach        ;Ш
        movx @dptr,a
        inc dptr
        mov a,#a5h        ;И
        movx @dptr,a
        inc dptr
        mov a,#a0h        ;Б
        movx @dptr,a
        inc dptr
        mov a,#4bh        ;К
        movx @dptr,a
        inc dptr
        mov a,#41h        ;А
        movx @dptr,a
        inc dptr
        mov r6,#34
        mov a,#20h
hole2:
        movx @dptr,a
        inc dptr
        djnz r6,hole2           ;заполнение пробелами оставшихся символов
        lcall indic

        ljmp main
        ret

        include asms\43501_3\5.4\p3\indic2.asm

        org 800Bh
        ljmp tim0
```

Испытание программы и выводы по заданию:

В результате запуска программы через вкладку «Терминал» стало возможным осуществлять посылку информации в МК. Посылки принимаются микроконтроллером побайтно. Символы выводятся на ЖКИ последовательно друг за другом. Таким образом, здесь осуществляется односторонняя передача от инструментальной ЭВМ в микроконтроллер.

Отправляемое сообщение во вкладке «Терминал» представлено на рисунке 5.

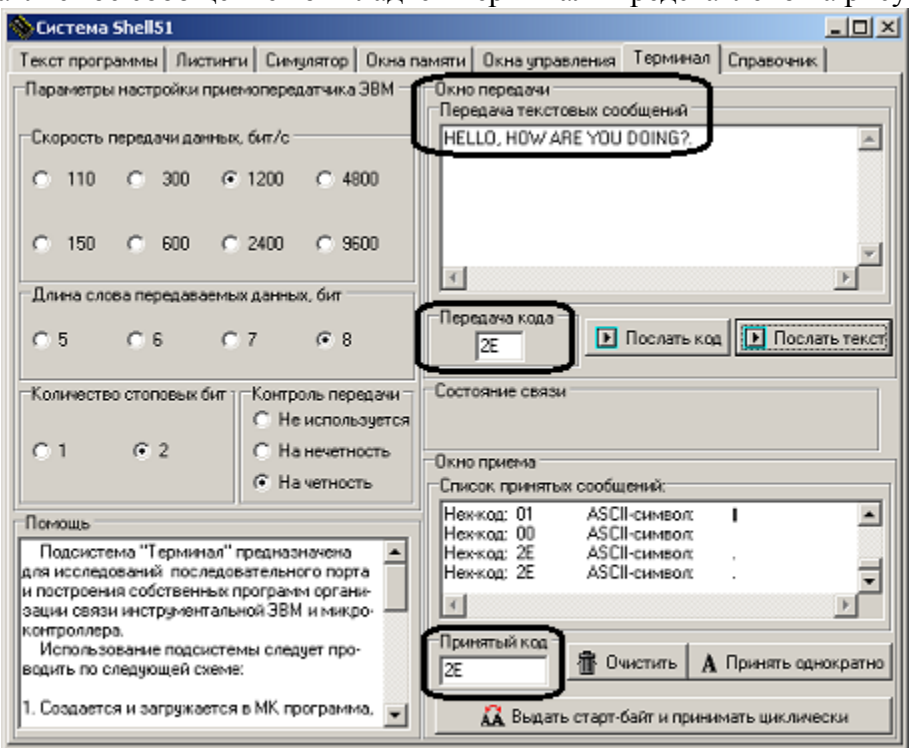

Рис. 5. Вкладка «Терминал» во время работы программы rec_lcd.asm

Передаётся сообщение “HELLO, HOW ARE YOU DOING?.”. В результате работы программы это сообщение было выведено на экран ЖКИ. Так же пытались передавать сообщения, длина которых составляет более 39 символов. Сообщение передавалось не полностью, на выход поступало сообщение состоящее из символов, находящихся после 39 символа.

Если мы попробуем передать сообщение без точки в конце, увидим во втором окне сообщение– «ОШИБКА».

4.4 Программа статической обработки данных

Разработать и выполнить сервисную программу статистической обработки данных.

Для решения поставленной задачи необходимо составить протокол обработки поступающих кодов. Первый посылаемый символ задаёт код операции. Примем следующие обозначения кодов операций:

‘+’ – поиск максимума;

‘–‘ – поиск минимума;

‘=’ – подсчёт среднего арифметического;

Для каждой операции должна быть определена своя процедура дальнейшего функционирования микроконтроллера.

Стоит отметить, что посылаемые с терминала коды соответствуют ASCII кодам введенных в окно символов, а не их значению в какой-либо системе счисления. Этот фактор важен и будет учтён при обработке поступающей информации.

В соответствии с протоколом передачи данных за кодом операции должны следовать числа через пробел. В конце сообщения необходимо поставить точку. Нарушение этих правил приведёт к выводу байта, символизирующего об ошибке и завершения выполнения текущей операции.

Так как цифры принимаются побайтно, то число фиксируется только по приёму пробела или точки. Поскольку работа ведётся только с числами, ограниченными рамками байта (возможны числа в пределах 0-255), то нарушение этого правила также вызовет сообщение об ошибке и завершение операции.

После выполнения операции на экран ЖКИ выводится её результат. Кроме того, ответ передаётся в терминал.

Алгоритм поиска среднего арифметического заключается в складывании всех поступивших в микроконтроллер чисел, а последовательном вычитании из 16разрядного результата 8разрядного число, обозначающего количество поступивших значений.

Поиск минимума прост: изначально в качестве минимума записано максимально возможное значение (255 или FF в шестнадцатеричной системе счисления). Если в результате вычитания нового числа из текущего минимума установился флаг переноса, то это число сохраняется в качестве нового минимума.

Поиск максимума аналогичен поиску минимума. Здесь в качестве максимума изначально записано минимально возможное значение – нуль. Каждое следующее большее него число записывается на его место.

На рисунке 5 представлена схема соединений:

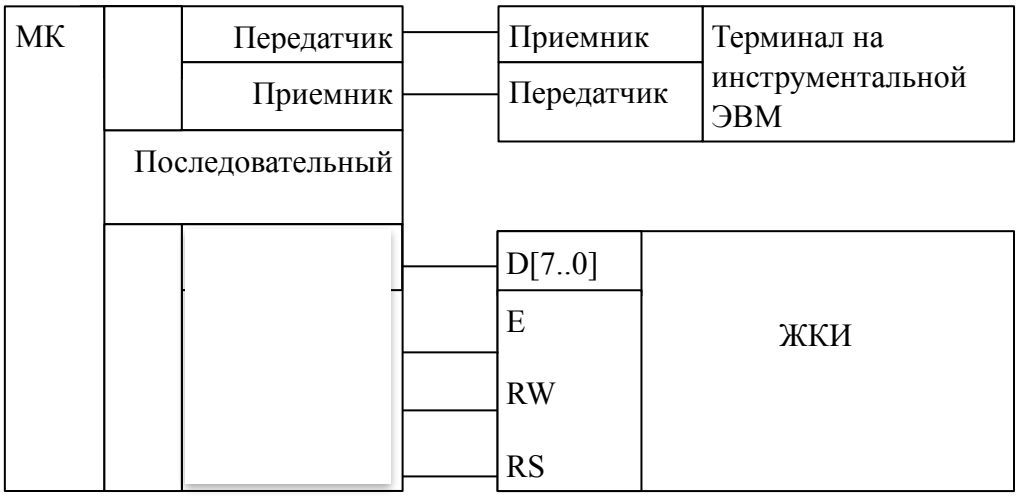

Рис. 5. Схема соединений модуля ЖКИ к МК

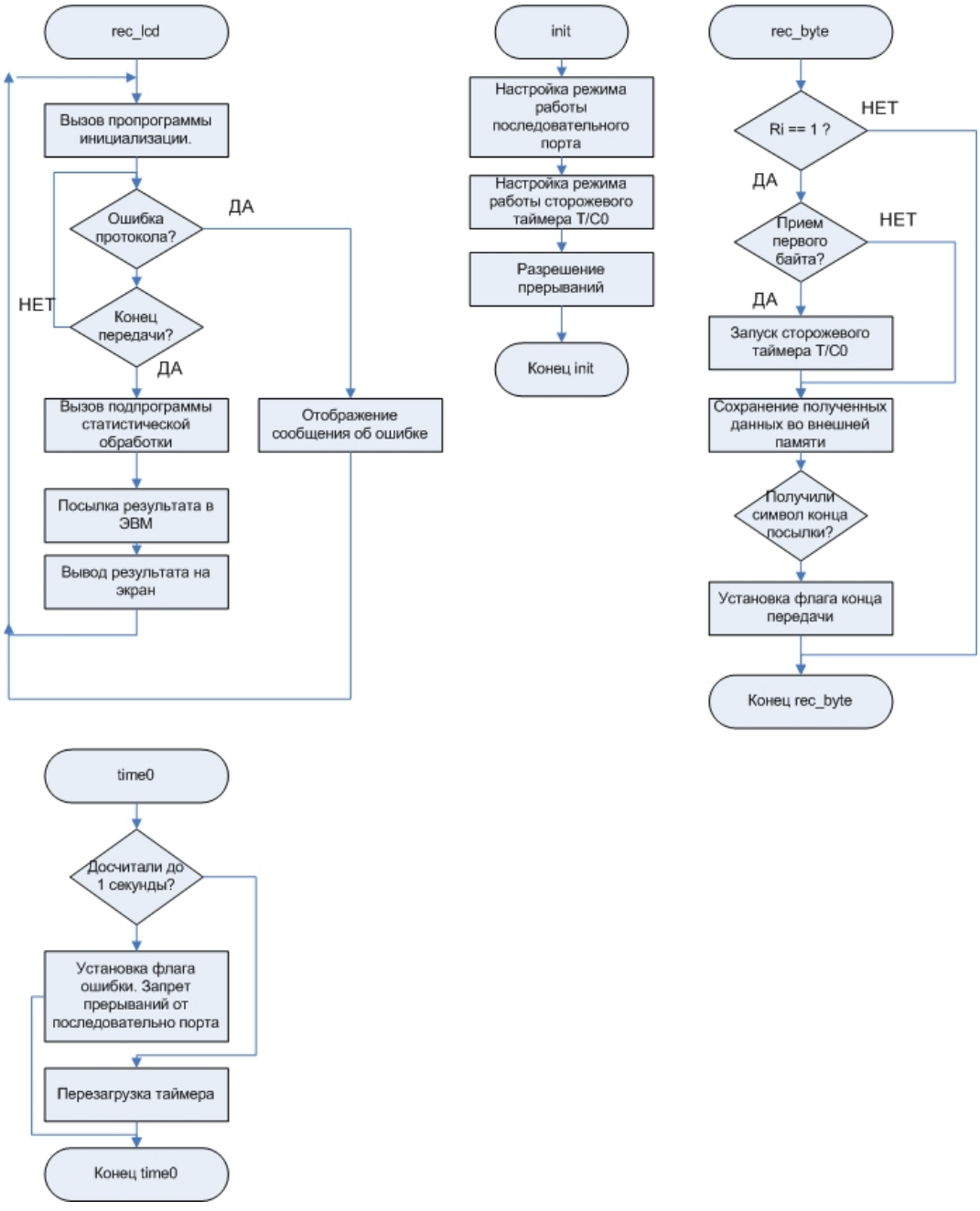

Рис. 6. Схема программы статистической обработки данных

Код программы:

Static.asm

```
;Программа 4.3. a=среднее. s=max. i=min
      org 8400h
      sjmp st_init

st_init:
      lcall init

st_loop:
      lcall receive
      lcall indic
      sjmp st_loop


init:
      clr tr1
      clr tr0
      mov 31h, #D8h           ; th0
      mov 30h, #EFh           ; tl0
      mov 40h, #0

      mov scon, #11010010b    ; режим работы SP

      anl tmod, #0            ; задание скорости SP
      orl tmod, #00100001b; программированием T/C1,
                  ; а также настройка T/C0

      anl D8h, #7Fh     ; сброс BD
      anl 87h, #7Fh     ; сброс SMOD
      mov th1, #E6h     ; скор. обмена 1.2 Кбит/с

      setb tr1
      setb ea
      setb et0

      ret

receive:
      jnb ri, receive

      mov r0, sbuf
      clr ri
      mov a, r0
      xrl a, #'.'
      jz receive

      mov a, r0
      xrl a, #' '
      jz receive

rcv_nempty:
      mov th0, 31h
      mov tl0, 30h
      mov 40h, #0
      lcall clr_scr
      setb tr0

      mov a, r0
      xrl a, #'s'
      jz rcv_max

      mov a, r0
```

```
        xrl a, #'i'
        jnz rcv_opt_eq
        ajmp rcv_min

rcv_opt_eq:
        mov a, r0
        xrl a, #'a'
        jnz rcv_opt_deflt
        ajmp rcv_mid

rcv_opt_deflt:
        mov a, r0
        clr c
        subb a, #30h
        jc receive

        subb a, #Ah
        jnc receive

rcv_max:
        ; запись заглавия в видеопамять
        mov dptr, #FFD0h
        mov a, #'m'
        movx @dptr, a
        inc dpl
        mov a, #'a'
        movx @dptr, a
        inc dpl
        mov a, #'x'
        movx @dptr, a

        clr 0
        clr 2
        mov 32h, #0
        mov r1, #0
rcv_max_loop:
        jb tr0, rcv_max_chck
        ajmp rcv_end
rcv_max_chck:
        jnb ri, rcv_max_loop

        mov r0, sbuf
        clr ri

        mov a, r0
        xrl a, #'.'
        jnz rcv_max_nend
        clr tr0
        jb 0, rcv_max_last

        ajmp rcv_showres

rcv_max_last:
; прием чисел окончен, сравнение последнего с max

        clr 0
        setb 2
        mov a, 32h
        clr c
        subb a, r1
        jnc rcv_max_last0

        mov 32h, r1
rcv_max_last0:
        ajmp rcv_showres
```

```
rcv_max_nend:
      mov a, r0
      xrl a, #' '
      jnz rcv_max_nspace
      jnb 0, rcv_max_loop

      ; прием числа окончен, сравнение с max
      clr 0
      setb 2
      mov 33h, r1
      mov r1, #0
      mov a, 32h
      clr c
      subb a, 33h
      jnc rcv_max_loop

      mov 32h, 33h
      ajmp rcv_max_loop

rcv_max_nspace:
      mov a, r0
      clr c
      subb a, #30h
      jnc rcv_max_ae30

      clr 0       ; принят символ с кодом <30
      mov r1, #0  ; сброс приема числа
      ajmp rcv_max_loop       ;

rcv_max_ae30:
      mov r0, a
      subb a, #Ah
      jc rcv_max_corrnum

      clr 0       ; принят символ с кодом >39
      mov r1, #0  ; сброс приема числа
      ajmp rcv_max_loop

rcv_max_corrnum:
      ; прием и преобразование очередной цифры числа
      setb 0
      mov a, r1
      mov b, #10
      mul ab
      add a, r0
      mov r1, a

      ajmp rcv_max_loop

;--------------------------
rcv_min:
      ; запись заглавия в видеопамять
      mov dptr, #FFD0h
      mov a, #'m'
      movx @dptr, a
      inc dpl
      mov a, #'i'
      movx @dptr, a
      inc dpl
      mov a, #'n'
      movx @dptr, a

      clr 0
      clr 2
      mov 32h, #255
      mov r1, #0
```

```
rcv_min_loop:
      jb tr0, rcv_min_chck
      ajmp rcv_end
rcv_min_chck:
      jnb ri, rcv_min_loop

      mov r0, sbuf
      clr ri

      mov a, r0
      xrl a, #'.'
      jnz rcv_min_nend
      clr tr0
      jb 0, rcv_min_last

      ajmp rcv_showres

rcv_min_last:
      ; прием чисел окончен, сравнение последнего с min
      clr 0
      setb 2
      mov a, r1
      clr c
      subb a, 32h
      jnc rcv_min_last0

      mov 32h, r1
rcv_min_last0:
      ajmp rcv_showres

rcv_min_nend:
      mov a, r0
      xrl a, #' '
      jnz rcv_min_nspace
      jnb 0, rcv_min_loop

      ; прием числа окончен, сравнение с min
      clr 0
      setb 2
      mov 33h, r1
      mov r1, #0
      mov a, 33h
      clr c
      subb a, 32h
      jnc rcv_min_loop

      mov 32h, 33h
      ajmp rcv_min_loop

rcv_min_nspace:
      mov a, r0
      clr c
      subb a, #30h
      jnc rcv_min_ae30

      clr 0       ; принят символ с кодом <30
      mov r1, #0  ; сброс приема числа
      ajmp rcv_min_loop

rcv_min_ae30
      mov r0, a
      subb a, #Ah
      jc rcv_min_corrnum

      clr 0       ; принят символ с кодом >39
```

```
            mov r1, #0  ; сброс приема числа
            ajmp rcv_max_loop       ;

      rcv_min_corrnum:
            ; прием и преобразование очередной цифры числа
            setb 0
            mov a, r1
            mov b, #10
            mul ab
            add a, r0
            mov r1, a

            ajmp rcv_min_loop

      ;-------------------------
      rcv_mid:
            ; запись заглавия в видеопамять
            mov dptr, #FFD0h
            mov a, #'m'
            movx @dptr, a
            inc dpl
            mov a, #'i'
            movx @dptr, a
            inc dpl
            mov a, #'d'
            movx @dptr, a

            clr 0
            clr 2
            mov 32h, #0
            mov 33h, #0
            mov r1, #0
            mov r2, #0

      rcv_mid_loop:
            jb tr0, rcv_mid_chck
            ajmp rcv_end
      rcv_mid_chck:
            jnb ri, rcv_mid_loop

            mov r0, sbuf
            clr ri

            mov a, r0
            xrl a, #'.'
            jnz rcv_mid_nend
            clr tr0
            jnb 0, rcv_mid_calc

            ; прием чисел окончен
            clr 0
            setb 2
            mov a, r1
            add a, 32h
            mov 32h, a
            mov a, 33h
            addc a, #0
            mov 33h, a
            inc r2

            ajmp rcv_mid_calc

      rcv_mid_nend:
            mov a, r0
            xrl a, #' '
            jnz rcv_mid_nspace
```

```
        jnb 0, rcv_mid_loop

        ; прием числа окончен, добавление к общей сумме
        ; и инкремент счетчика чисел r2
        clr 0
        setb 2
        mov a, r1
        add a, 32h
        mov 32h, a
        mov a, 33h
        addc a, #0
        mov 33h, a
        inc r2
        mov r1, #0

        ajmp rcv_mid_loop

rcv_mid_nspace:
        mov a, r0
        clr c
        subb a, #30h
        jnc rcv_mid_ae30

        clr 0       ; принят символ с кодом <30
        mov r1, #0  ; сброс приема числа
        ajmp rcv_mid_loop

rcv_mid_ae30:
        mov r0, a
        subb a, #Ah
        jc rcv_mid_corrnum

        clr 0       ; принят символ с кодом >39
        mov r1, #0  ; сброс приема числа
        ajmp rcv_mid_loop       ;

rcv_mid_corrnum:
        ; прием и преобразование очередной цифры числа
        setb 0
        mov a, r1
        mov b, #10
        mul ab
        add a, r0
        mov r1, a

        ajmp rcv_mid_loop

rcv_mid_calc:
        ; вычисление ср. арифм.
        mov a, r2
        xrl a, #1
        jnz rcv_mc0

        ajmp rcv_showres

rcv_mc0:
        mov a, #128
        mov b, r2
        div ab
        rl a
        mov r3, a

        mov a, b
        rl a
        mov b, r2
        div ab
```

```
            add a, r3
            mov r4, b

            mov b, 33h
            mul ab
            mov 34h, a

            mov a, r4
            mov b, 33h
            mul ab
            mov b, r2
            div ab
            add a, 34h
            mov 33h, a
            mov r4, b

            mov a, 32h
            mov b, r2
            div ab
            add a, 33h
            mov 33h, a

            mov a, b
            add a, r4
            mov b, r2
            div ab
            add a, 33h

            mov 32h, a
      rcv_showres:
            jb 2, rcv_showres0

            mov dph, #FFh
            mov r0, #D0h
            mov r1, #70h
      rs_wr_loop:
            mov dpl, r1
            movx a, @dptr
            mov dpl, r0
            movx @dptr, a
            inc r0
            inc r1
            cjne r0, #F8h, rs_wr_loop

            sjmp rcv_end

      rcv_showres0:
            ; преобразование числа из ячейки 32h
            ; и запись в видеопамять
            lcall conv_res

      rcv_end:
            ret
      conv_res:
            clr 1
            mov a, 32h
            jnz convr_nz

            mov dptr, #FFE3h
            mov a, #30h
            movx @dptr, a
            sjmp convr_end

      convr_nz:
            mov dptr, #FFE1h
            mov b, #100
```

```
            div ab
            mov 32h, b
            jz convr0

            setb 1
            add a, #30h
            movx @dptr, a

      convr0:
            inc dpl

            mov a, 32h
            mov b, #10
            div ab
            jz convr1

            add a, #30h
            movx @dptr, a
            sjmp convr2

      convr1:
            jnb 1, convr2

            mov a, #30h
            movx @dptr, a

      convr2:
            inc dpl
            mov a, b
            add a, #30h
            movx @dptr, a

      convr_end:
            ret

      clr_scr:
            push 0
            mov a, #' '
            mov dptr, #FFD0h
      clr_loop:
            movx @dptr, a
            inc dpl
            mov r0, dpl
            cjne r0, #F8h, clr_loop
            pop 0
            ret
      tim0:
            push a
            push 0
            push 1
            push psw
            mov th0, 31h
            mov tl0, 30h

            inc 40h
            mov a, 40h
            clr c
            subb a, #FFh
            jc tim0_end

            clr tr0
            mov 40h, #0
            mov dph, #FFh
            mov r0, #D0h
            mov r1, #A0h
      err_wr_loop:
```

```
        mov dpl, r1
        movx a, @dptr
        mov dpl, r0
        movx @dptr, a
        inc r0
        inc r1
        cjne r0, #F8h, err_wr_loop

        lcall indic
tim0_end:
        pop psw
        pop 1
        pop 0
        pop a
        reti
        org 800Bh
        ljmp tim0
        org FF70h
non1: db '    *data errore*   '
non2: db 'no correct nums rcvd'

        org FFA0h
err1: db '  *protocol error*  '
err2: db ' use . as end symbol'

        org FFD0h
str1: db '   :               0'
str2: db '                     '
        include C:\SHELL51\ASMS\43501_3\5.4\p3\indic.asm
```

indic.asm

```
        org 8100h
        w1: equ 20h
        w0: equ 21h

indic:      mov w1,#0
        mov w0,#38h
        lcall ind_wr
        mov w0,#0Ch
        lcall ind_wr
        mov w0,#80h
        lcall ind_wr
        mov w1,#1
        mov dptr,#FFD0h

wr_str1:
        movx a,@dptr
        mov w0,a
        lcall ind_wr
        inc dptr
        mov a,dpl
        cjne a,#0E4h,wr_str1
        mov w1,0
        mov w0,#C0h
        lcall ind_wr
        mov w1,#1

wr_str2:
        movx a,@dptr
        mov w0,a
        lcall ind_wr
        inc dptr
        mov a,dpl
        cjne a,#0F8h,wr_str2

        ret
ind_wr:
        mov F8h,w0
        setb P1.7
        clr P1.6
        mov a,w1
        mov c,acc.0
        mov P1.4,c
        lcall delay
        clr P1.7
        lcall delay
        setb P1.7
        ret
delay:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        ret
```

Результаты программы:

В результате выполнения программы получаем верные результаты:

Испытание программы:

В результате запуска и тестирования программы на микроконтроллере на ЖКИ были получены результаты различной обработки введённых данных.

Если послать с терминала строчку '+88 1 167 25 202 123 8 16 207 47 245 59.', через некоторое время на ЖКИ появится результат: «245» (максимальное значение).

При посылке той же строки, но с первым символом ‘-’ (минимальное значение) результат на ЖКИ - «001».

При посылке той же строки, но с первым символом ‘=’ (среднее арифметическое) результат на ЖКИ – «100».

При посылке строки без точки на месте последнего символа, на экран ЖКИ выводится символ сигнализирования об ошибке ввода данных – ‘¿’.

Наборы на которых тестировалась программа:

«+1 1 1 5 1 5 2 0» - на ЖКИ результат 5

«-1 1 1 5 1 5 2 0» - на ЖКИ результат 0

«=1 1 1 5 1 5 2 0» - на ЖКИ результат 2

4.5 Программа обмена информацией между соседними микроконтроллерными стендами (2n – 1) и 2n, где n = 1,2,3,…

Разработать и выполнить программу обмена информацией между соседними микроконтроллерными стендами 5 и 6.

Для решения данной задачи необходимо осуществить правильное взаимодействие микроконтроллеров через последовательный порт. Для этого необходимо в каждом из них задать одну и ту же скорость передачи, один и тот же режим. Это легко выполнить при запуске одной и той же программы на обоих стендах. Также необходимо выполнять обработку нажатой клавиши и посылку её номера через последовательный порт в основной программе, а приём данных от соседнего стенда организовать в режиме прерывания, обрабатывая прерывания от приёмопередатчика. Нажатую на текущем стенде клавишу надо помещать в первую строку видеопамяти, а принятые байты – во вторую.

Структура информационных связей обоих стендов представлена на рис. 6.

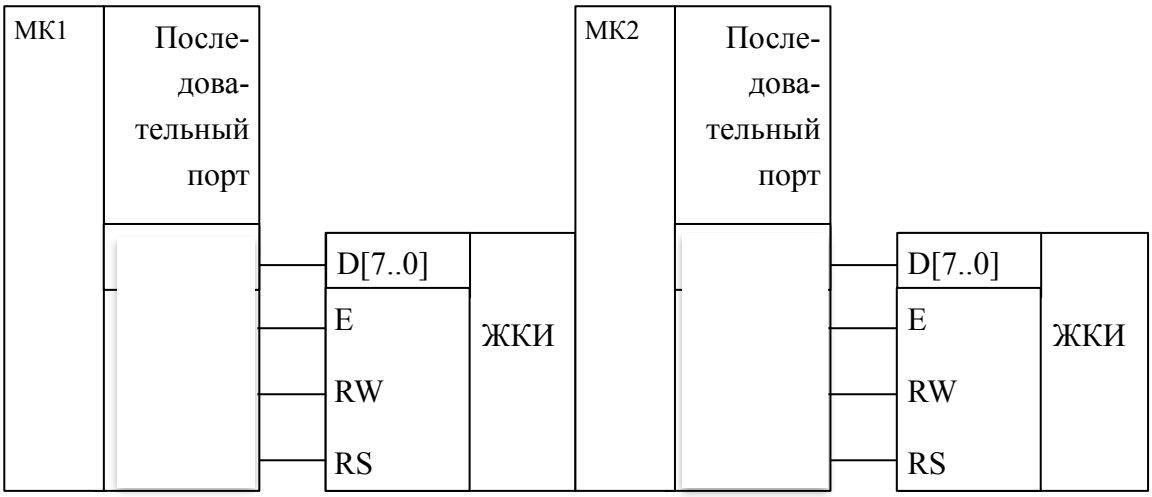

Рис. 6. Схема соединения МК с периферией и между собой.

Программа communication использует комбинированный способ взаимодействия с последовательным портом. Так, получение новых данных выполняется по прерыванию, а посылка из основной программы. Кроме того с целью уменьшения объема пересылок, данные пересылаются только в случае их изменения.

Алгоритм работы программы (рис. 7):

Алгоритм:

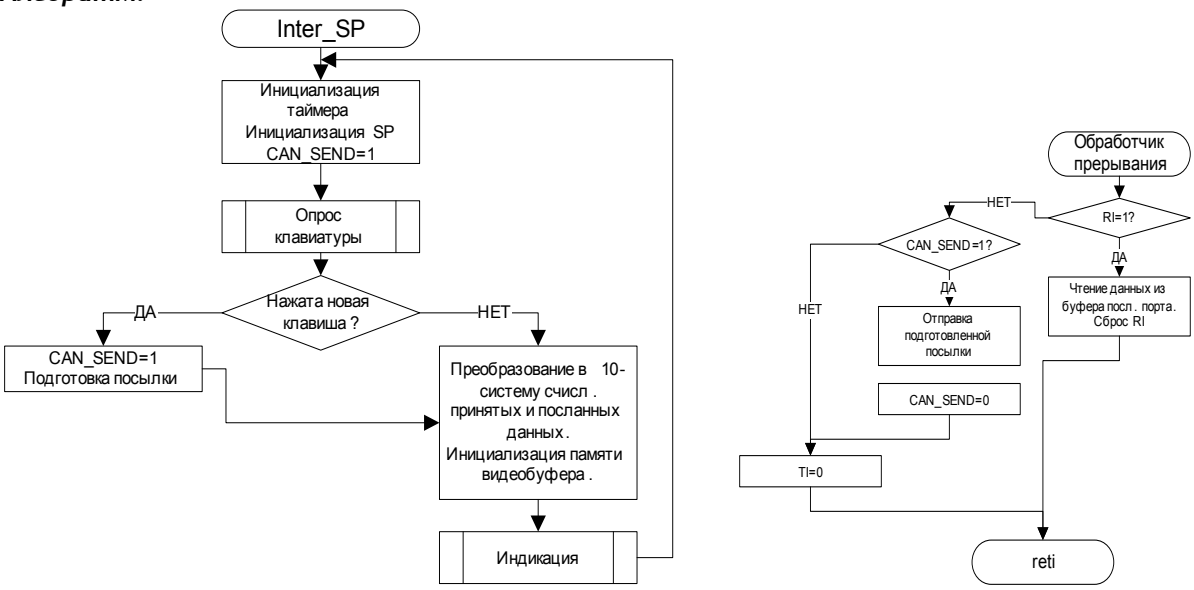

Рис. 7. Алгоритм программы обмена информацией между соседними стендами

Код программы:

main.asm

```
; Основная программа:

init: clr   tr1   ; запрет счёта таймера T/C1
      ; счетчик T/C1 работает в режиме 8-разрядного автогенератора
      anl   TMOD, #0Fh
      orl   TMOD, #20h
      ; настраиваем  скорость обмена 1200 б/с
      mov   TH1, #E6h

      ; Настойка последовательного порта
```

```
        mov   SCON, #11010010b
        anl   ADCON, #7Fh
        anl   PCON, #7Fh

        ; Настройка прерываний
        setb  ea
        ; разрешаем прерывания от последовательного порта
        setb  es
        ; запускаем счетчик последовательного порта
        setb  tr1

        ; Установка начальных значений
        mov   sp, #07h
        ; устанавливаем начальное состояние программы
        mov   OLD_SCANCODE, #0  ; скан-код предыдущей нажатой кнопки
        ; разрешаем первую передачу
        setb  CAN_SEND    ; бит разрешения передачи
        ; параметры преобразования числа в ASCII
        mov   CONV_BASE, #10
        mov   N_DIGITS, #2


        ; главный цикл приема информации
loop:   ; опрашиваем клавиатуру, формируем номер нажатой клавиши
        lcall kbd_read
        clr   c
        ; сравниваем новое значение и предыдущее
        mov   a, SCANCODE
        subb  a, OLD_SCANCODE
        ;ЕСЛИ скан-код новой клавиши тот же, что и старый
        jz    $l1

        ; если значение изменилось, отсылаем его другому контроллеру
        setb  CAN_SEND
        mov   SND_DATA, SCANCODE
        mov   OLD_SCANCODE, SCANCODE

        ;TO
$l1:    ; формируем видеобуфер для отображения на ЖКИ
        mov   CONV_NUM, SCANCODE
        mov   dptr, #S_SCANCODE       ; инициализируем область памяти для
видеобуфера
        lcall wr_int                  ; преобразование скан-кода в 10-ю
систему счисления
        mov   CONV_NUM, RCV_DATA
        mov   dptr, #S_OTHER_SCAN
        lcall wr_int
        mov   dptr, #STR1
        lcall indic
        sjmp  loop
        ret

; Глобальные переменные и константы:
; Порты:
p4:         equ   E8h
p5:         equ   F8h

; Внутренняя память данных
; Для работы с клавиатурой:
KBD_BUF:          equ   30h
SCANCODE:   equ   34h

; Для преобразования числа в ascii:
CONV_BASE:  equ   42h   ;  число для конвертирования
CONV_NUM:   equ   43h   ; основание системы счисления используемой для
преобразования
```

```
N_DIGITS:         equ   44h

; Для работы с АЦП:
ADCON_SETUP:      equ   45h
U:          equ   46h
ADCON:            equ   D8h
ADDAT:            equ   D9h
DARP:       equ   Dah

; Для работы с последовательным портом
PCON        equ   87h
SND_DATA    equ   35h
RCV_DATA    equ   36h
OLD_SCANCODE      equ   37h
; битовый флаг, отвечающий за отсылку данных
CAN_SEND    equ   0

; Внешняя память данных
STR1        db    'MY NUMBER IS:     00'
S_SCANCODE  db    'OTHER NUMBER IS:  00'
S_OTHER_SCAN      db    ''


; Функции:
uart: jnb   ri, $u1
      ; получаю новые данные
      mov   RCV_DATA, SBUF
      clr   ri
      sjmp  $exit
$u1:  jnb   CAN_SEND, $u2
      ; если разрешено, отправляю подготовленные данные
      mov   SBUF, SND_DATA
      ; сбрасываем флаг разрешения передачи
      clr   CAN_SEND
$u2:  clr   ti
$exit:      reti


; Подключение модулей:

      include asms\43501_3\share\std.asm
      include asms\43501_3\share\indic4.asm
      include asms\43501_3\share\wr_int2.asm
      include asms\43501_3\share\kbd_read.asm

; Обработчики прерываний:
      org   8023h
      ljmp  uart
```

Пример работы программы:

В результате выполнения программы получаем верные результаты, которые были продемонстрированы преподавателю.

4.6. Программа «Master-Slave»

Структура информационных связей обоих стендов соответствует предыдущей задаче.

Протокол обмена:

Будем считать, что кнопка 15 предназначена для перевода МК в режим “Master”. При ее нажатии МК присваивается статус “Master” с отображением символа “M” на табло ЖКИ, при этом код кнопки передается в канал связи. Ведомый МК принимает посылку и при декодировании кода 15 переводится в режим “Slave”.

Последовательные порты микроконтроллеров на передающей (после отправки кода 15) и принимающей стороне (после получения кода 15) перепрограммируется: SM2 = 1 и, поскольку принимающему МК присваивается статус “Slave”, передача от “Slave” к “Master” запрещается. При переводе МК в режим “Master-Slave” изменяется протокол обмена. В этом режиме “Master” сначала передает адресную информацию, после чего разрешается передача данных. Код адресной информации имеет установленный бит TB8, а код данных – сброшенный бит TB8. При SM2 = 1 МК может принять только адресную информацию, то есть посылку с установленным битом RB8.

Кнопки 13 и 14 кодируют адреса ведомых МК, при этом код кнопки 14 соответствует адресу соседнего стенда. При нажатии кнопки 14 (или 13) на экране ЖКИ ведущего МК высвечивается информация “aN” (N = 14 или 13), формируется и отправляется адресная посылка с установленным битом TB8. Завершающим этапом посылки адреса является запрет установки бита TB8 в последующих посылках. Поскольку принимаемая посылка является адресом, ведомые МК получат и проанализируют эту посылку. Если адрес 14, ведомый МК идентифицирует себя как приемник. Он программирует свой последовательный порт (сбрасывает бит SM2) и отображает на экране ЖКИ номер “a14”. Если адрес не равен 14, ведомый МК указанные действия не выполняет. После отправки адреса последующие посылки являются посылками данных.

Кнопка 12 кодирует команду «конец посылки данных». При декодировании кода 12 оба МК переводятся в режим межконтроллерного взаимодействия. В этом режиме при нажатии кнопки 15 любой из МК может быть переведен в режим “Master”.

Кнопки 0-11 – это обычные информационные кнопки. При их нажатии на передающей стороне осуществляется процедура определения номера нажатой клавиши и отображения этого номера на экране ЖКИ стенда. Код номера нажатой клавиши передается ведомому. Значение бита TB8 безразлично. На приемной стороне (при SM2 = 0) данные принимаются и отображаются, поскольку при SM2 = 0 прием не зависит от значения бита RB8 в посылке.

Описание программы:

Для реализации режима «master-slave» задействованы специально предназначенные для этого ресурсы МК, а именно биты SM2, RB8 и TB8 регистра SCON.

Алгоритм работы представлен на рисунке 8

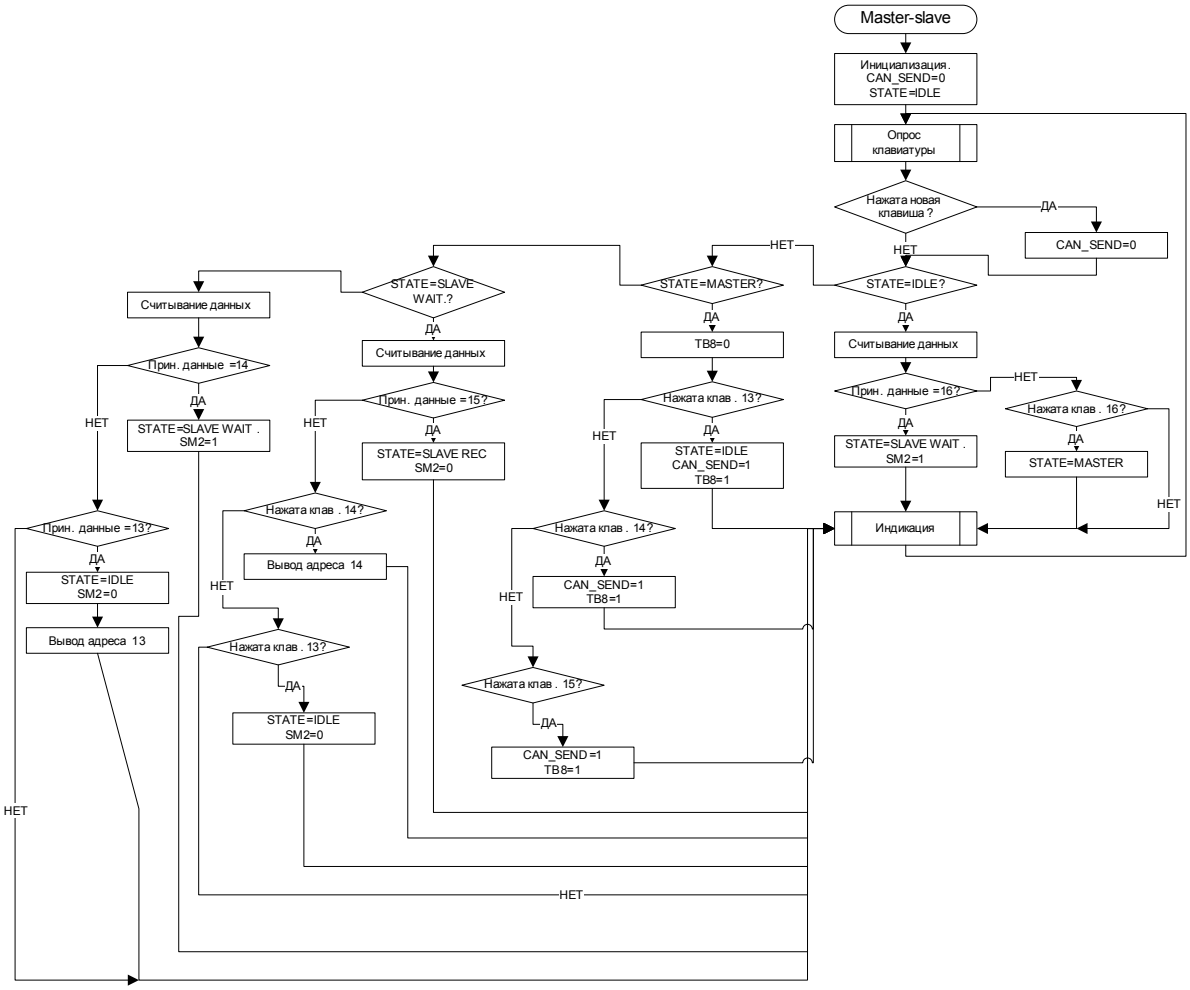

Рис. 8. Алгоритм программы «Master-Slave»

Программа может пребывать в одном из 5 состояний. Переход из состояния в состояние происходит по приходу двух типов сигналов: сигнал типа self означает, что нажата собственная клавиша, сигнал типа other, означает, что был получен номер нажатой клавиши от соседа.

Описание состояний:

В начале работы, программа находится в состоянии EQUAL, это означает, что в сети нет «мастера» и оба контроллера равноправны, т.е. нажатие клавиши на одном мк, отображается на экране ЖКИ, подключенном к другому мк. При нажатии на клавишу 15 программа переходит в состояние MASTER_ADDR, при этом она ждет ввода адреса мк, которому планируется слать данные. В состоянии MASTER_DATA программа выполняет посылку номера нажатой клавиши по последовательному порту, при нажатии клавиши 12, программа снимает с себя полномочия мастера и переходит в режим равноправия. В состояние SLAVE_ADR программа переходит если соседний контроллер прислал номер клавиши 15. В этом состоянии она ожидает когда мастер обратится к ней по ее адресу (клавиша 14). После того как мастер прислал номер 14, программа переходит в состояние ожидания данных SLAVE_DATA, при этом она принимает номер нажатой клавиши у мастера и отображает его на экране. Если мастер прислал номер 12, программа переходит в состояние равноправия EQUALS и сама может стать мастером.

Код программы:

indic.asm

```
P4:   equ E8h
P5:   equ F8h

indic:      clr P5.0
      mov r4, #38h
      lcall ind_wr
      mov r4, #0Ch
      lcall ind_wr
      mov r4, #80h
      lcall ind_wr

      mov dptr, #FFD0h
      setb P5.0

wr_str1:    movx a, @dptr
      mov r4, a
      lcall ind_wr
      inc dptr
      mov a, dpl
      cjne a, #E4h, wr_str1

      clr P5.0
      mov r4, #C0h
      lcall ind_wr
      setb P5.0

wr_str2:    movx a, @dptr ;
      mov r4, a
      lcall ind_wr
      inc dptr
      mov a, dpl
      cjne a, #0F8h, wr_str2
      ret

ind_wr:     mov P4, r4
      setb p5.3
      clr p5.2
      lcall delay
      clr p5.3
      lcall delay
      setb p5.3
      ret

delay:      mov r3, #7
m2:   djnz r3, m2
      ret
```

p_thrd.asm

```
            org 8400h

p5:   equ   f8h
p4:   equ   e8h
intPt:      equ   41h
retPt:      equ   42h
status:     equ   44h
tmp:  equ   45h
button:     equ   34h
button2:    equ   35h
IP0:        equ   A9h
IP1:        equ   B9h
nowTask:    equ   20h

      mov nowTask,#0h
      mov button, #0h
      mov status, #20h
      mov dptr, #task1
      mov r0,dpl
      mov r1,dph
      mov dptr, #tsk1_d
      mov a,dpl
      add a,#8h
      mov dpl,a
      mov a,r0
      movx @dptr,a
      inc dptr
      mov a,r1
      movx @dptr,a
      mov dptr, #task2
      mov r0,dpl
      mov r1,dph
      mov dptr, #tsk2_d
      mov a,dpl
      add a,#8h
      mov dpl,a
      mov a,r0
      movx @dptr,a
      inc dptr
      mov a,r1
      movx @dptr,a
      mov dptr, #task3
      mov r0,dpl
      mov r1,dph
      mov dptr, #tsk3_d
      mov a,dpl
      add a,#8h
      mov dpl,a
      mov a,r0
      movx @dptr,a
      inc dptr
      mov a,r1
      movx @dptr,a

      setb ea
      lcall TC0Init
      lcall initSendRec
task1:      lcall memklav
      sjmp task1

      ; ËÌ‰ËÍ‡ËˆËˇ,
task2:      lcall indic
      mov a, button
      jz task2
```

```
      lcall klavIn
      mov a, status
      clr C
      subb a, #53h      ; S-Slave
      jz task2

      mov a, button
      lcall send
      sjmp task2

task3:      lcall receive
      mov button2, a
      lcall klavIn2
      sjmp task3



delay31:
      mov r4, #04h
d_m11:      mov r3, #FFh
d_m21:      djnz r3, d_m21
      djnz r4, d_m11
      ret

delay32:
      mov r4, #01h
d_m12:      mov r3, #FFh
d_m22:      djnz r3, d_m22
      djnz r4, d_m12
      ret

delay33:
      mov r4, #01h
d_m13:      mov r3, #FFh
d_m23:      djnz r3, d_m23
      djnz r4, d_m13
      ret

time0:
      cpl P1.0

      push dph
      push dpl
      push psw
      push b
      push a
      push 0
      push 1

      mov r0,sp
      inc r0
      lcall getTskDsc
      mov r1,#0h
saveM1:
      mov a,@r1
      movx @dptr,a
      inc r1
      inc dptr
      djnz r0,saveM1

      lcall nextTask
      lcall getTskDsc

      movx a,@dptr
      mov r0,a
```

```
      dec r0
      mov sp,r0

      mov r1,#0h
restM2:     movx a,@dptr
      mov @r1,a
      inc r1
      inc dptr
      djnz r0,restM2

      pop 1
      pop 0
      pop a
      pop b
      pop psw
      pop dpl
      pop dph

      mov TH1,#ech
      mov TL1,#77h
      reti

nextTask:
      inc nowTask
      mov a,nowTask
      cjne a,#3h,skip
      mov nowTask,#0h
skip: ret

getTskDsc:
      mov a,nowTask
      cjne a, #0h,s1
      mov dptr, #tsk1_d
      ret
s1:   cjne a, #1h,s2
      mov dptr, #tsk2_d
      ret
s2:   mov dptr, #tsk3_d
      ret

;str1:   db 'x  Button number Nx' ; 17-18 bytes
;str2:   db '   Button number Nx'
          ;'0123456789_123456789'

tsk1_d: db 11h,
1,0,0,0,0,0,0,00,00,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
tsk2_d: db 11h,
1,0,0,0,0,0,0,00,00,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
tsk3_d: db 11h,
1,0,0,0,0,0,0,00,00,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0


  include C:\SHELL51\ASMS\43501_3\bk\4\p46\indic.asm
  include C:\SHELL51\ASMS\43501_3\bk\4\p46\timer.asm
  include C:\SHELL51\ASMS\43501_3\bk\4\p46\sklav.asm
  include C:\SHELL51\ASMS\43501_3\bk\4\p46\klavIn.asm
  include C:\SHELL51\ASMS\43501_3\bk\4\p46\sendRec.asm


      org FFD0h

str1:   db 'X          send: xx ' ; 17-18 bytes
str2:   db 'addr: xx   recv: xx '
```

klav1.asm

```
klavIn:
      clr TB8
      mov a, status
      clr C
      subb a, #53h      ; S-Slave
      jz s_end

      mov a, button
      clr C
      subb a, #0Fh
      jnz s_m1

      mov a, #4Dh
      mov status, a
      mov dptr, #str1
      movx @dptr, a
      jmp s_next


s_m1: mov a, status
      clr C
      subb a, #20h
      jz s_next


      mov a, button
      clr C
      subb a, #0Eh
      jnz s_m2

      setb TB8

      mov a, #31h ; #31h="1"
      mov dptr, #str2+6
      movx @dptr, a

      mov a, #34h ; #34h="4"
      inc dptr
      movx @dptr, a
      jmp s_next

s_m2: mov a, button
      clr C
      subb a, #0Dh
      jnz s_m3

      setb TB8
      mov a, #31h
      mov dptr, #str2+6
      movx @dptr, a

      mov a, #33h ; #33h="3"
      inc dptr
      movx @dptr, a

      jmp s_next

s_m3: mov a, button
      clr C
      subb a, #0Ch
      jnz s_next

      setb TB8
      clr SM2
```

```
        mov status, #20h
        mov a, #78h ; #78h="x"
        mov dptr, #str1
        movx @dptr, a

        mov dptr, #str2+6
        movx @dptr, a
        inc dptr
        movx @dptr, a
        jmp s_next

s_next:
        mov r0, #34h
        mov a, @r0
        mov b, #0Ah
        div ab
        mov dptr, #str1+17  ; address "N" in str1
        add a, #30h  ; translate to ASCII-code
        movx @dptr, a ; "N" <=> 0-1
        mov a, b  ; b=0-6
        add a, #30h  ; ASCII-code
        inc dptr  ; address "x" in str2
        movx @dptr, a ; "N" <=> 0-1
s_end:
        ret

klavIn2:
        mov a, status
        clr C
        subb a, #4Dh      ; M-Master
        jz r_end

        mov a, button2
        clr C
        subb a, #0Fh
        jnz r_m1

        setb SM2
        mov a, #53h ; #53h="S"(Slave)
        mov status, a     ; status=Slave
        mov dptr, #str1
        movx @dptr, a
        jmp r_next

r_m1: mov a, status
        clr C
        subb a, #20h
        jz r_next

        mov a, button2
        clr C
        subb a, #0Dh
        jnz r_m2

        clr SM2

        mov a, #31h ; #31h="1"
        mov dptr, #str2+6
        movx @dptr, a

        mov a, #33h ; #34h="3"
        inc dptr
        movx @dptr, a
        jmp r_next
```

```
r_m2: mov a, button2
      clr C
      subb a, #0Dh
      jz r_m3

      clr SM2

      mov a, #31h ; #31h="1"
      mov dptr, #str2+6
      movx @dptr, a

      mov a, #34h ; #34h="4"
      inc dptr
      movx @dptr, a
      jmp r_next

r_m3: mov a, button2
      clr C
      subb a, #0Ch
      jnz r_next
      clr SM2
      mov status, #20h

      mov a, #78h ; #78h="x"
      mov dptr, #str1
      movx @dptr, a
      mov dptr, #str2+6
      movx @dptr, a
      inc dptr
      movx @dptr, a
      jmp r_next

r_next:
      mov r0, #35h  ; address of inside memmory
      mov a, @r0  ; number of button 1-16
      mov b, #0Ah ; b = 10
      div ab  ; a/b  => a=0-1,  b=0-6
      mov dptr, #str2+17  ; address "N" in str1
      add a, #30h  ; translate to ASCII-code
      movx @dptr, a ; "N" <=> 0-1
      mov a, b  ; b=0-6
      add a, #30h  ; ASCII-code
      inc dptr  ; address "x" in str2
      movx @dptr, a ; "N" <=> 0-1
r_end:
      ret
```

sendRec.asm

```
initSendRec:
      clr tr1
      mov scon, #11010010b
      anl tmod, #0Fh
      orl tmod, #00100000b
      mov th1, #e6h
      anl D8h, #7Fh
      anl 87h, #7Fh
      setb tr1
      ret


 ;********************receive
receive:    jnb ri, receive
      mov a, sbuf
      clr ri
      ret



 ;************************send
send: jnb ti, send
      mov sbuf, a
      clr ti
      ret
```

timer.asm

```
TC0Init:
  anl TMOD,#11110000b
  orl TMOD,#00000001b
  mov TH0,#ech
  mov TL0,#77h
  setb ea
  setb et0
  setb tr0
  ret


  ; TC1 interruption
  org 800bh
  ljmp time0
```

skills.asm

```
memklav:
      mov 20h, #0h
      mov R1, #33h
      mov R3, #3h
      mov 35h, #0h
      mov 37h, #0h
      mov 38h, #0h
      mov 39h, #0h
      lcall klav

zero_chk:
      mov C, 0h
      mov A, @R1
      ;mov 56h, R1
      subb A, #f0h
      jz skip_cntr
      inc 35h
      mov A,@R1
      mov 37h,A
      mov 38h,R3

skip_cntr:
      dec R1
      dec R3
      mov C, 0h
      cjne R1, #2Fh, zero_chk

      mov A, 35h
      jz wr_0
      mov C, 0h
      cjne A, #01h, wr_FF
      mov dptr, #cdMask
mov R3,#0h
find_column:
      inc R3;
      mov 39h,R3
      mov A,R3;
      mov C, 0h         ; clear C
      subb A,#5h
      jz wr_FF

      movx A, @dptr
      inc dptr
      mov C, 0h
      cjne A, 37h, find_column

get_num:
      mov A, 38h
      mov C, 0h
      rl A
      rl A
      ;add A, 39h
      add A, #5h
      subb A, 39h
      mov 34h, A
      sjmp ext
wr_0: mov 34h, #0h
      sjmp ext
wr_FF:      mov 34h, #FFh
      sjmp ext

cdMask: db E0h, D0h, B0h, 70h

ext:  ret
```

```
klav: mov r0, #30h
      orl p5, #f0h
      mov a, #7fh

mb:   mov r2, a

      rlc a
      mov p1.7, c
      rlc a
      mov p1.6, c
      rlc a
      mov p1.5, c
      rlc a
      mov p1.4, c

      mov a, p5
      anl a,#f0h
      mov @r0, a
      inc r0
      mov a, r2
      rr a
      cjne a, #f7h, mb
      ret
```

4. Вывод

В ходе лабораторной работы были исследованы возможности обмена данными по последовательному порту.

При организации обмена следует обратить особое внимане на установку одних и тех же параметров передачи как на приемной, так и на передающей стороне. Параметрами передачи являются скорость передачи данных, длинна слова передаваемых данных, контроль передачи на четность, количество стоповых бит. Если параметры передачи различны, то корректная передача данных становится невозможной.

Информацию о принятых/отправленных данных можно получить анализируя биты ri/si регистра SCON. Если бит ri установлен, значит получены новые данные, если установлен бит si, значит данные были отправленны, и передатчик готов к работе. Опрашивать биты ri и ti можно в основной программе, однако зачастую это нерациональное использование процессорного времени, так как последовательный порт работает с гораздо меньшей скоростью, чем процессор. Другим вариантом организации приема/ передачи данных является использование прерываний от последовательного порта. В этом случае нужно помнить, что обработчик прерываний вызывается как при получении данных, так и при окончании посылки, поэтому в нем следует предусмотреть ветвтление, в зависимости от того какое событие наступило.

Кроме установки одинаковых параметров передачи, устройства, обменивающиеся данными должны согласовать протокол передачи данных. Т.е. кто и в каком порядке шлет данные, какие символы являются допустимыми, как распознать окончание передачи и т. д. Если протокол включает в себя ограничения на максимальное время передачи (как в программах п.4.3,4.4. лабораторной работы), то для его контроля удобно использовать один из таймером, к примеру T/C0 в качестве сторожевого. Если при этом работа с последовательным портом ведется через прерывания, то для обеспечения точного контроля времени нужно присвоить прерываниям от сторожевого таймера более высокий приоритет.

Иногда требуется организовать мультипроцессорною систему с одним ведущим и несколькими ведомыми МК. Поддержку такого взаимодействия обеспечивает специальный бит разрешения мутльипроцессорной работы SM2, регистра SCON. Фактически он управляет разрешением прерываний от последовательного порта: если бит SM2 установлен, то прерывания от ПП активизируются только при поступлении «единичного» девятого бита входной информации (RB8), а при SM2 = 0, прервыния от ПП активизируются независимо от состояния бита RB8.

Подводя итоги, можно сказать что микроконтроллер предоставляет широкие возможности для обмена данными по последовательному порту, с помощью которых можно реализовать различные протоколы передачи.