Санкт-Петербургский государственный политехнический университет

Факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №4**

«Организация и исследование межпроцессорного обмена

по дисциплине "Микропроцессорные системы»

Вариант №5

Работу выполнил студент

Дорофеев Юрий Владимирович гр. 4081/12 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Работу принял преподаватель

Павловский Евгений Григорьевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_

г. Санкт-Петербург

2012

**Содержание**

[1. Цель работы 3](#_Toc343819055)

[2. Программа работы 3](#_Toc343819056)

[3. Теоретические сведения 3](#_Toc343819057)

[4.1. Программа простого обмена по последовательному порту 8](#_Toc343819058)

[4.2. Модифицированная программа send\_rec 9](#_Toc343819059)

[4.3. Программа вывода на экран ЖКИ текста, передаваемого с инструментальной ЭВМ ………………………………………………………………………………………….10](#_Toc343819060)

[4.4 Программа статической обработки данных 15](#_Toc343819061)

[4.5 Программа обмена информацией между соседними микроконтроллерными стендами (2n – 1) и 2n, где n = 1,2,3,… 27](#_Toc343819062)

[4.6 Программа «Master-Slave» 31](#_Toc343819063)

[4. Вывод 37](#_Toc343819064)

1. Цель работы

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

1. 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».

1. Теоретические сведения

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

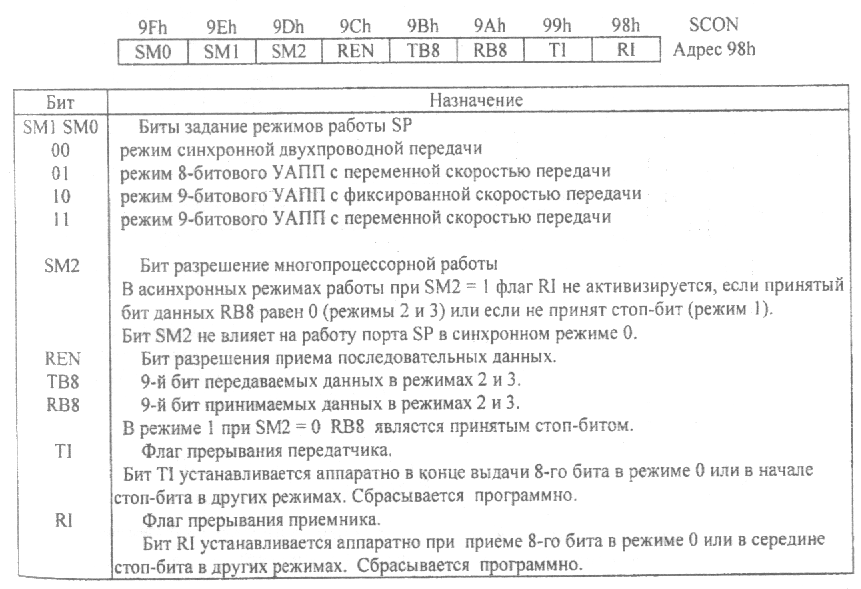
Основой данного порта является универсальный асинхронный приемопередатчик (УАПП), который осуществляет прием и передачу информации, представленной последовательным кодом, в полном дуплексном режиме обмена. В состав УАПП входят принимающий и передающий сдвигающие регистры, а также специальный буферный регистр (SBUF с адресом 9916) 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 (9816), битовая структура которого приведена ниже:

*Таблица 1*



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 (8716), 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); Fрез - 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 (SМ1 SМ0)** | **Константа перезагрузки** |
| 62,5 кГц | 1 | 10 | 0FF16 |
| 19,2 кГц | 1 | 10 | 0FD16 |
| 9,6 кГц | 0 | 10 | 0FD16 |
| 4,8 кГц | 0 | 10 | 0FA16 |
| 2,4 кГц | 0 | 10 | 0F416 |
| 1,2 кГц | 0 | 10 | 0E816 |

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:

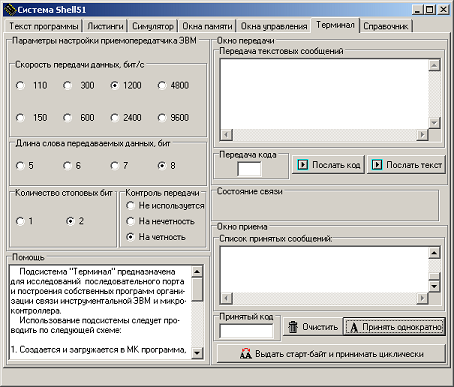
Таким образом, для частоты 19,2 кГц при Fрез = 12 МГц [ТН1] = 252,74510, возьмем приближенное значение [ТН1] = 25310, тогда формула для частоты резонатора имеет следующий вид:

В итоге для приведенных выше значений частота резонатора равняется Fрез = 11,0592 МГц. Однако если рассматривать частоту обмена равную 62,5 кГц, то значение [ТН1] = 25510 получается точно только при частоте резонатора равной 12 МГц.

В результате можно сделать вывод, что приведенные в таблице значения переменной перегрузки ТН1 являются ориентировочными и требуют дополнительной настойки под конкретную задачу. Смысл настройки складывается из последовательного перебора ближайших значений ТН1, 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 А516, 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 Т/С1

;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 Т/С1

anl tmod, #0Fh ;выбор режима работы таймера Т/С1

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 Т/С1

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.

* 1. 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

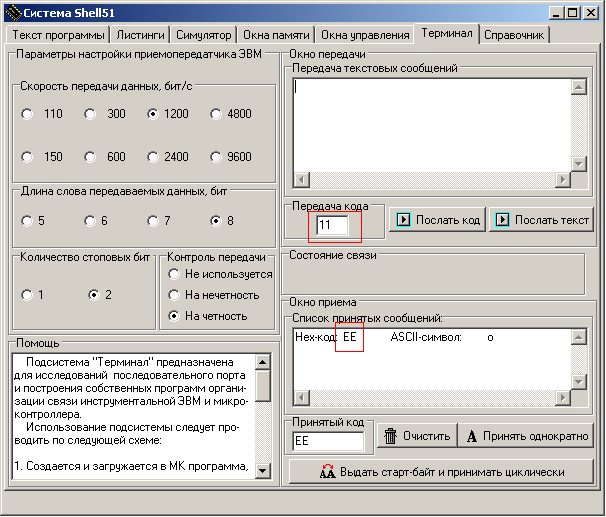
mov sbuf, a

clr ti

ret

***Результаты выполнения:***

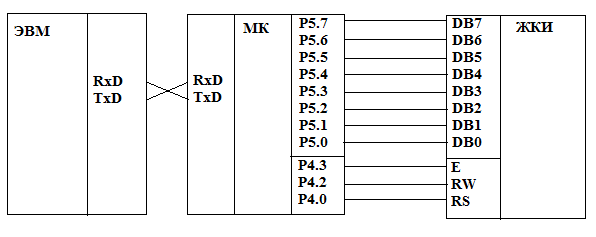
В ходе выполнения программы мы посылаем значения через «Терминал» в микроконтроллер с помощью кнопки «Послать код», а затем получаем инвертированные значения при нажатии на кнопку «Принять однократно».



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

* 1. 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 Т/С0

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 Т/С0

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 Т/С1

;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 Т/С1

anl tmod,#0Fh ;выбор режима работы таймера Т/С1

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 Т/С1

clr rxToFl

anl TMOD, #F0h

orl TMOD, #01h

mov TH0, #00h

mov TL0, #00h

clr tr0 ;останов таймера Т/С0

setb ea ;разрешение всех прерываний

setb et0 ; разрешение прерываний Т/С0

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\4081\_4\BS\5.4\p3\indic2.asm

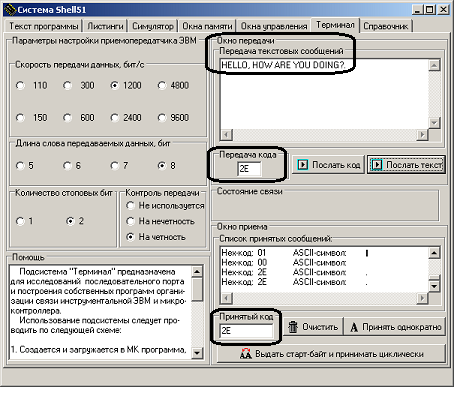
org 800Bh

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:

Терминал на инструментальной ЭВМ

Приемник

Передатчик

Приемник

Последовательный порт

P1.7

P1.6

P1.4

МК

 P4[7..0]

ЖКИ

ЖКИ

D[7..0]

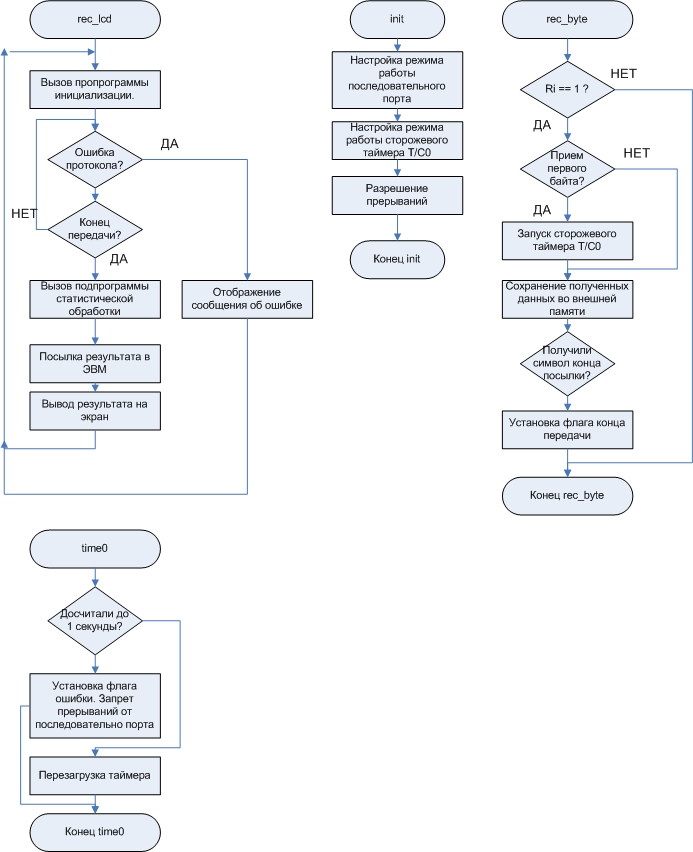
E

RW

RS

Передатчик

*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\4081\_4\static\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.

После-дова-тельный порт

P1.7

P1.6

P1.4

МК1

P4[7..0]

ЖКИ

ЖКИ

D[7..0]

E

RW

RS

После-дова-тельный порт

P1.7

P1.6

P1.4

МК2

P4[7..0]

ЖКИ

ЖКИ

D[7..0]

E

RW

RS

*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

;ИНАЧЕ

; если значение изменилось, отсылаем его другому контроллеру

setb CAN\_SEND

mov SND\_DATA, SCANCODE

mov OLD\_SCANCODE, SCANCODE

;ТО

$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\4081\_4\share\std.asm

include asms\4081\_4\share\indic4.asm

include asms\4081\_4\share\wr\_int2.asm

include asms\4081\_4\share\kbd\_read.asm

;======================================================

; Обработчики прерываний:

org 8023h

ljmp uart

**Пример работы программы:**

В результате выполнения программы получаем верные результаты:

(клавиши на стендах не нажаты)

MY NUMBER IS: 00

OTHER NUMBER IS: 00

(нажаты клавиши на обоих стендах)

MY NUMBER IS: 09

OTHER NUMBER IS: 14

(нажаты клавиши на обоих стендах)

MY NUMBER IS: 14

OTHER NUMBER IS: 09

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:***

*main.asm*

org 8100h

include asms\4081\_4\share\macro.asm

;======================================================

; Основная программа:

init: clr tr1 ; разрешаем счёт для 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

; разрешаем первую передачу

setb CAN\_SEND

mov STATE, #IDLE

; в состоянии IDLE бит SM2 должен быть сброшен

anl SCON, #11011111b

; 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

mov a, SCANCODE

subb a, OLD\_SCANCODE

jz $k1

; если значение изменилось, отсылаем его другому контроллеру

setb CAN\_SEND

mov SND\_BUF, SCANCODE

mov OLD\_SCANCODE, SCANCODE

$k1: mov a, STATE

cjne a, #IDLE, state\_master

; состояние IDLE:

mov a, RCV\_BUF ; записываем принятые данные

cjne a, #16, $l1

; получили код 16 - переходим в состояние SLAVE

mov STATE, #SLAVE\_WAITING

; обязательным атрибутом SLAVE контроллера является установленный бит SM2 SCON

orl SCON, #00100000b

putchar S\_STATE, #'S'

put2c S\_ADDRESS, #'?', #'?'

ljmp disp

$l1: mov a, SCANCODE

cjne a, #16, $j1

; нажата клавиша 16 - переходим в состояние MASTER

mov STATE, #MASTER

putchar S\_STATE, #'M'

$j1 ljmp disp

state\_master:

cjne a, #MASTER, state\_slave\_waiting

; состояние MASTER:

anl SCON, #11110111b

mov a, SCANCODE

cjne a, #13, $l4

; нажата клавиша 13 - переходим в состояние IDLE

mov STATE, #IDLE

anl SCON, #11011111b

; предварительно сообщив об этом \*всем\* контроллерам

orl SCON, #00001000b

setb CAN\_SEND

putchar S\_STATE, #'I'

ljmp disp

; адресам выставляется повышенный приоритет за счет установки бит TB8

$l4: cjne a, #14, $l5

orl SCON, #00001000b

setb CAN\_SEND

sjmp disp

$l5: cjne a, #15, disp

orl SCON, #00001000b

setb CAN\_SEND

sjmp disp

state\_slave\_waiting:

cjne a, #SLAVE\_WAITING, state\_slave\_receiving

; состояние SLAVE\_WAITING:

mov a, RCV\_BUF

cjne a, #15, $l2

mov STATE, #SLAVE\_RECEIVING

; сбрасываем бит SM2 для получения данных

anl SCON, #11011111b

put2c S\_ADDRESS, #'1', #'5'

sjmp disp

$l2: cjne a, #14, $l3

put2c S\_ADDRESS, #'1', #'4'

sjmp disp

$l3: cjne a, #13, disp

mov STATE, #IDLE

; в состоянии IDLE бит SM2 также не нужен

anl SCON, #11011111b

putchar S\_STATE, #'I'

put2c S\_ADDRESS, #'.', #'.'

sjmp disp

state\_slave\_receiving:

; состояние SLAVE\_RECEIVING:

mov a, RCV\_BUF

cjne a, #14, $l3

; получен неправильный адрес контроллера

; выставляем бит SM2 для фильтрации не предназначенных

; этому констроллеру данных

mov STATE, #SLAVE\_WAITING

orl SCON, #00100000b

put2c S\_ADDRESS, #'1', #'3'

sjmp disp

$l3: cjne a, #13, disp

; мастер завершил работу - переходим в состояние IDLE

mov STATE, #IDLE

anl SCON, #11011111b

putchar S\_STATE, #'I'

put2c S\_ADDRESS, #'.', #'.'

disp: mov dptr, #S\_SCAN1

mov CONV\_NUM, SND\_BUF

lcall wr\_int

mov dptr, #S\_SCAN2

mov CONV\_NUM, RCV\_BUF

lcall wr\_int

mov dptr, #STR1

lcall indic

ljmp 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

CONV\_NUM: equ 43h

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\_BUF equ 35h

RCV\_BUF equ 36h

OLD\_SCANCODE equ 37h

STATE equ 38h

IDLE equ 0

MASTER equ 1

SLAVE\_RECEIVING equ 2

SLAVE\_WAITING equ 3

; битовый флаг, отвечающий за отчылку данных

CAN\_SEND equ 0

; Внешняя память данных

STR1 db 'STATE: '

S\_STATE db 'I '

db 'A: '

S\_ADDRESS db '.. '

STR2 db 'SCAN1: 00'

S\_SCAN1 db ' SCAN2: 00'

S\_SCAN2 db ' '

;======================================================

; Функции:

uart: jnb ri, $u1

; получаю новые данные

mov RCV\_BUF, SBUF

clr ri

reti

$u1: setb CAN\_SEND

jnb CAN\_SEND, $u2

; если разрешено, отправляю подготовленные данные

mov SBUF, SND\_BUF

; сбрасываем флаг разрешения передачи

clr CAN\_SEND

$u2: clr ti

reti

sync\_send:

; ожидание освобождения буфера передатчика

jnb ti, sync\_send

jnb CAN\_SEND, $l1

mov SBUF, SND\_BUF

clr CAN\_SEND

$l1: clr ti

ret

;======================================================

; Подключение модулей:

include asms\4081\_4\share\std.asm

include asms\4081\_4\share\indic4.asm

include asms\4081\_4\share\wr\_int2.asm

;include asms\4081\_4\share\adc\_read.asm

include asms\4081\_4\share\kbd\_read.asm

;======================================================

; Обработчики прерываний:

org 8023h

ljmp uart

Результаты:

Начальное состояние

STATE: I A: ..

SCAN1: 00 SCAN2: 00

STATE: I A: ..

SCAN1: 00 SCAN2: 00

Первый в мастере

STATE: M A: .. STATE: I A: 15

SCAN1: 01 SCAN2: 00 SCAN1: 06 SCAN2: 01

1. 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.