Комплекс лабораторных работ для учебного лабораторного стенда SDK-1.1

Версия 1.1.0

Ковязина Д. Р.,
Петров Е. В.

Санкт-Петербург
2010

Содержание

Лабораторная работа № 1

«Дискретные порты ввода-вывода»

Задание

Разработать и реализовать драйверы светодиодных индикаторов и DIP-переключателей контроллера SDK-1.1. Написать тестовую программу с использованием разработанных драйверов по алгоритму, соответствующему варианту задания.

Порты ввода-вывода

Каждый процессор для встраиваемых применений имеет некоторое количество внешних линий ввода-вывода, подключенных к внешним выводам микросхемы и называемых внешними портами. Одиночные (одноразрядные, состоящие из одной линии) порты ввода-вывода объединяются в группы обычно по 4, 8 или 16 линий, которые называются параллельными портами. Через порты процессорное ядро взаимодействует с различными внешними устройствами: считывает значения входных сигналов и устанавливает значения выходных сигналов. Во встраиваемых системах в качестве внешних устройств чаще всего рассматриваются датчики, исполнительные устройства, устройства ввода-вывода данных оператором, устройства внешней памяти.

По типу сигнала различают порты: дискретные (цифровые), аналоговые и перестраиваемые.

Дискретные (цифровые) порты используются для ввода-вывода дискретных значений логического «0» или «1». В большинстве современных процессоров для встраиваемых применений поддерживается как независимое управление каждой линией параллельного порта, так и групповое управление всеми разрядами.

Аналоговый порт ввода-вывода предназначен для работы с аналоговым (непрерывным) сигналом. В отличие от дискретного сигнала, принимающего всего два значения, напряжение аналогового сигнала может иметь любое значение (в определенных пределах) и меняться во времени. Пример аналогового сигнала – синусоида. Такую форму, например, имеет электрическое напряжение в сети 220 В 50 Гц.

Перестраиваемые порты ввода-вывода настраиваются на аналоговый или цифровой режим работы.

Описание работы

В контроллере SDK-1.1 изучение дискретных портов ввода-вывода будет проводиться на примере работы с набором светодиодных индикаторов и DIP[1]-переключателей (см. рис. 1). Данные устройства вывода и ввода, соответственно, подключены к расширителю портов ввода-вывода, который в стенде SDK-1.1 выполнен на базе ПЛИС (программируемая логическая интегральная схема). Для программиста расширитель портов представлен в виде нескольких однобайтовых регистров, находящихся в начале восьмой страницы внешней памяти данных.

[1] DIP (Dual In-line Package) – тип корпуса микросхем, микросборок и некоторых других электронных компонентов. Имеет прямоугольную форму с двумя рядами выводов по длинным сторонам. Может быть выполнен из пластика (PDIP) или керамики (CDIP). В корпусе DIP могут выпускаться различные полупроводниковые или пассивные компоненты: микросхемы, сборки диодов, транзисторов, резисторов, малогабаритные переключатели.

По умолчанию линии 0-7 дискретного параллельного порта ПЛИС притянуты к логической «1» (через резисторы на +5В). Для их принудительного «обнуления» в схему введен набор DIP-переключателей (на рис. 2 обозначение «SW3-2»), замыкающих соответствующие линии через резисторы 100 Ом на корпус. Для того чтобы принудительно «обнулить» соответствующую линию, необходимо установить соответствующий DIP-переключатель в положение «ON».

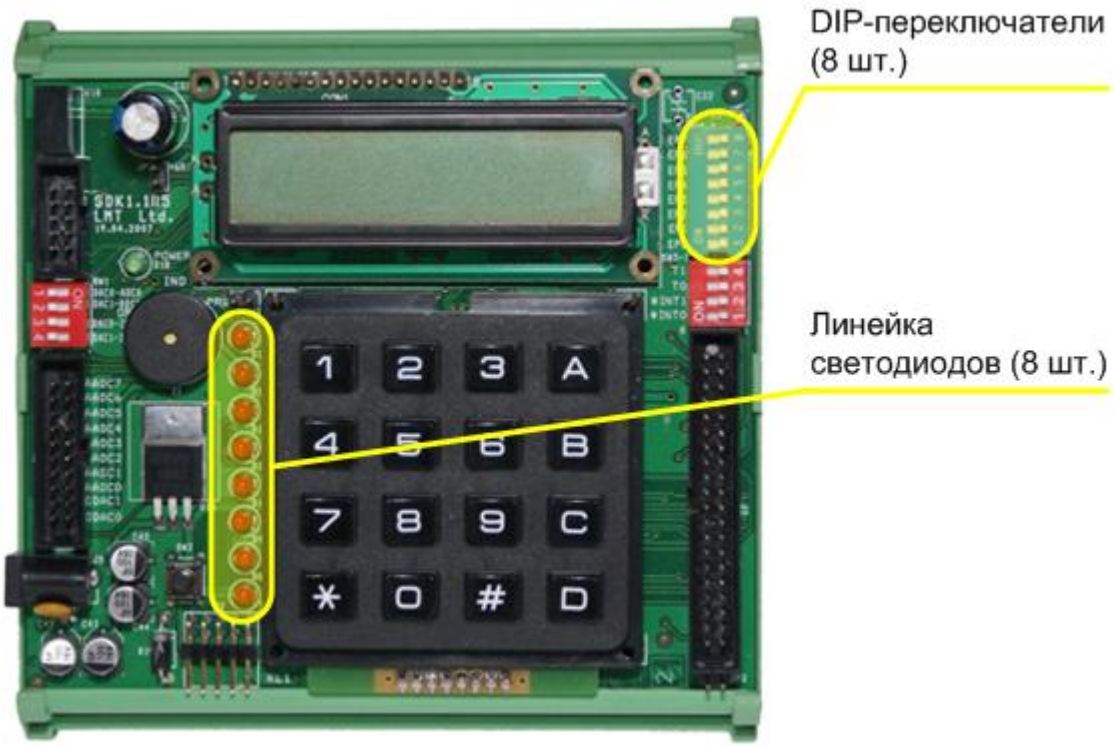
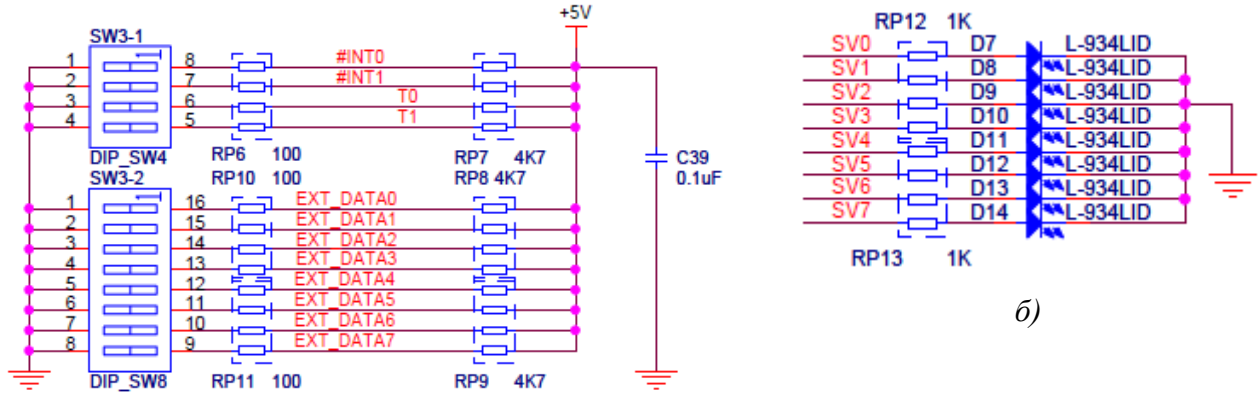


Рис. 1. Расположение программируемых в лабораторной работе светодиодных индикаторов и DIP-переключателей контроллера SDK-1.1

Для удобства, двоичный (шестнадцатеричный) код, задаваемый набором DIP-переключателей, выставляется следующим образом: первый переключатель соответствует младшему (0-му) разряду двоичного кода; восьмой переключатель соответствует старшему (7-му) разряду. При этом единичный разряд в коде – это соответствующий DIP-переключатель в положении «ON», нулевой разряд – это соответствующий DIP-переключатель в положении «OFF».



а) *б)*

Рис. 2. Обозначение DIP-переключателей (а) и светодиодных индикаторов (б) на принципиальной электрической схеме контроллера SDK-1.1

При установке DIP-переключателя в положение «ON» напряжение не сразу устанавливается на уровне 0В, а «скачет» в течение некоторого времени (1-10 мс), пока цепь надежно не замкнется. После того, как DIP-переключатель будет установлен в положение «OFF», напряжение также «скачет», пока не установится на уровне +5В. Такого рода переходные процессы называются дребезгом. Таким образом, при изменении положения DIP-переключателя («ON» ↔ «OFF») возникает эффект дребезга, отрицательное влияние которого в данной работе никак не устраняется.

Подробнее эта проблема будет исследоваться в лабораторной работе № 4 «Клавиатура».

Требования к выполнению работы

1. Все программы должны быть написаны на языке Си.
2. Разрабатываемые драйверы устройств должны быть выполнены в виде отдельных программных модулей (файлов), содержащих функции по работе с заданным одним устройством. Например, в этой лабораторной работе драйвер светодиодных индикаторов должен содержать функцию установки состояния светодиодов, а драйвер DIP-переключателей – функцию чтения их состояния. В главном программном модуле должна решаться предлагаемая вариантом задания прикладная задача с использованием разработанных драйверов.
3. В тестовой программе для осуществления анимации запрещается использовать «покадровое» формирование картинки. Для реализации алгоритма анимации должны быть использованы логические, арифметические и бинарные операции, а также операции сдвигов.
4. Для задержек в программе следует использовать пустые циклы (тысячи – десятки тысяч итераций, в зависимости от длительности задержки).
5. **В варианте задания представлен лишь фрагмент анимации, при этом все анимации являются циклическими. По приведенному фрагменту требуется определить алгоритм анимации и реализовать его.**
6. Текст программы должен соответствовать правилам оформления программ на языке Си, приведенным в приложении (Приложение Б. Требования к оформлению программ на языке Си, [28]).

Содержание отчета

1. Титульный лист.
2. Задание.
3. Блок-схема программы.
4. Исходный текст программы с комментариями.
5. Основные результаты.

Литература

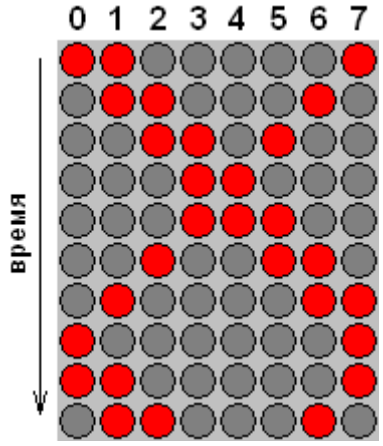
Литература к лабораторной работе: [9], [11], [12], [13], [14], [16], [21], [26], [27], [30], [32], [33], [34], [38], [42].

Варианты заданий

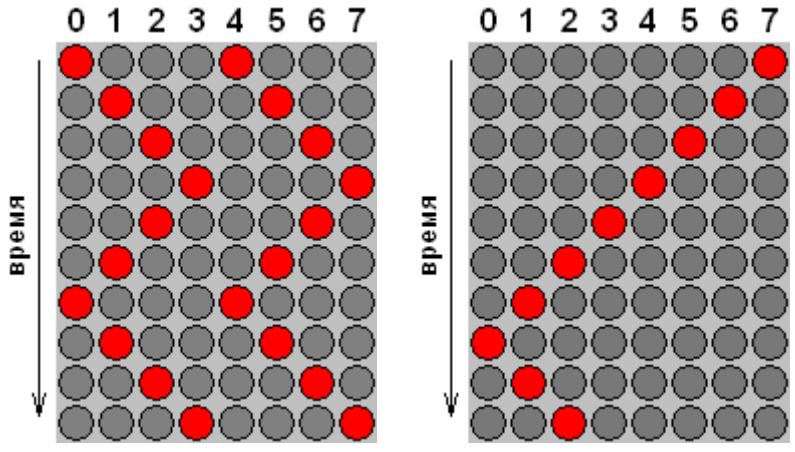
Во всех вариантах задания единичный разряд в шестнадцатеричном коде – это соответствующий DIP-переключатель в положении «ON», нулевой разряд – это соответствующий DIP-переключатель в положении «OFF».

В варианте задания представлен лишь фрагмент анимации, при этом все анимации являются циклическими. По приведенному фрагменту требуется определить алгоритм анимации и реализовать его.

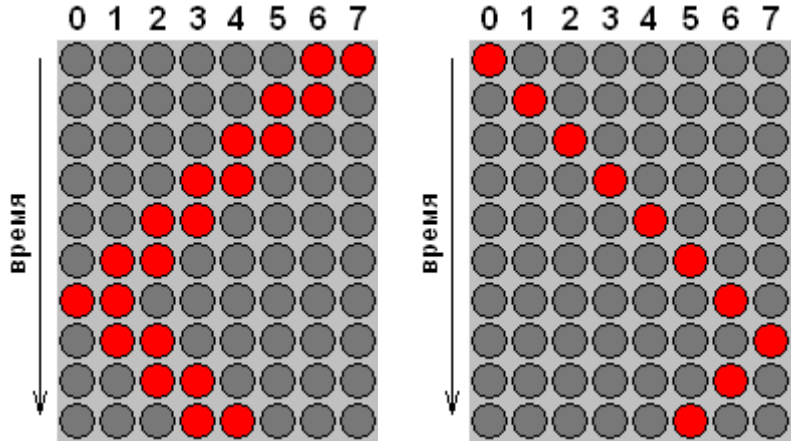
1. В случае установки на DIP-переключателях кода 0x11 (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться анимация, показанная ниже. Во всех остальных случаях светодиодные индикаторы отражают значение, выставленное на DIP-переключателях.



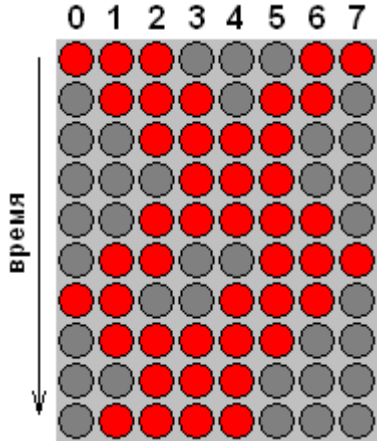
2. В случае установки на DIP-переключателях кода 0x22 (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться первая анимация, в случае установки кода 0xDD (шестнадцатеричное значение) – вторая анимация. Во всех остальных случаях светодиодные индикаторы отражают инвертированное значение, выставленное на DIP-переключателях.



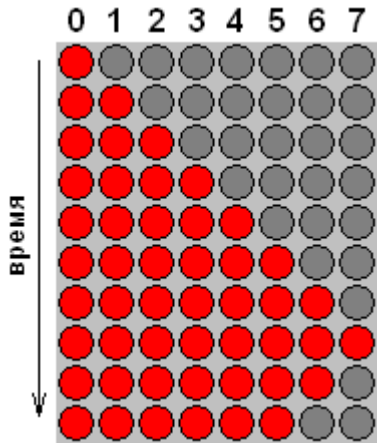
3. В случае установки на DIP-переключателях кода 0x33 (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться первая анимация, в случае установки кода 0xCC (шестнадцатеричное значение) – вторая анимация. Во всех остальных случаях светодиодные индикаторы отражают значение, выставленное на DIP-переключателях.



4. В случае установки на DIP-переключателях кода 0x44 (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться анимация, показанная ниже. Во всех остальных случаях светодиодные индикаторы отражают инвертированное значение, выставленное на DIP-переключателях.

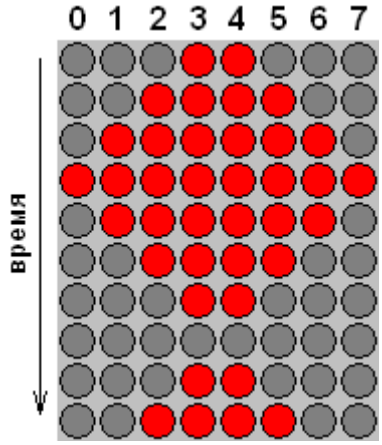


5. В случае установки на DIP-переключателях кода 0x55 (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться анимация, показанная ниже. Во всех остальных случаях светодиодные индикаторы отражают значение, выставленное на DIP-переключателях.

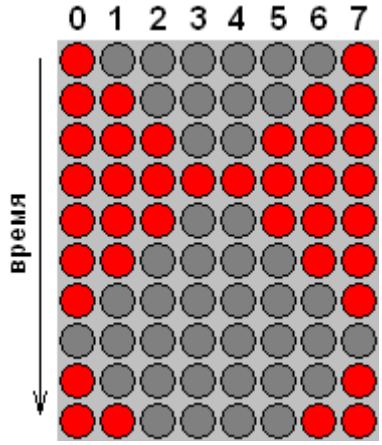


6. В случае установки на DIP-переключателях кода 0x66 (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться анимация, показанная ниже. Во всех

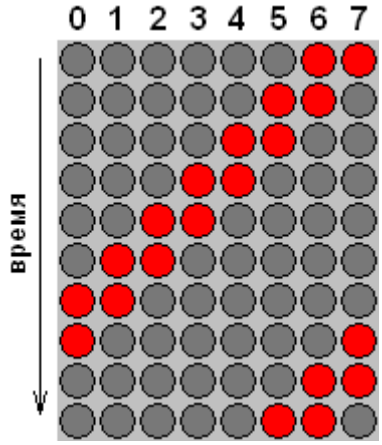
остальных случаях светодиодные индикаторы отражают инвертированное значение, выставленное на DIP-переключателях.



7. В случае установки на DIP-переключателях кода 0x77 (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться анимация, показанная ниже. Во всех остальных случаях светодиодные индикаторы отражают значение, выставленное на DIP-переключателях.

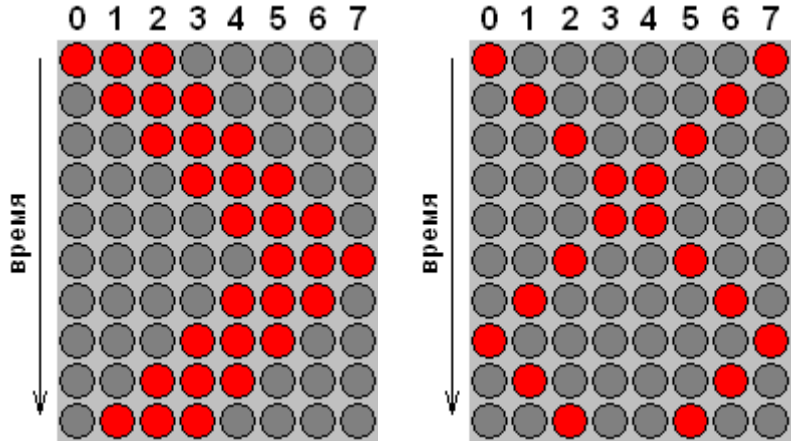


8. В случае установки на DIP-переключателях кода 0xAA (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться анимация, показанная ниже. Во всех остальных случаях светодиодные индикаторы отражают инвертированное значение, выставленное на DIP-переключателях.

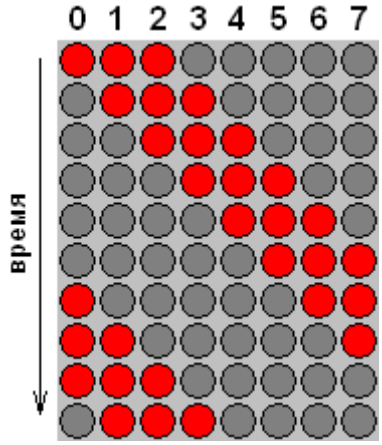


9. В случае установки на DIP-переключателях кода 0xBB (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться первая анимация, в случае установки кода

0x44 (шестнадцатеричное значение) – вторая анимация. Во всех остальных случаях светодиодные индикаторы отражают значение, выставленное на DIP-переключателях.



10. В случае установки на DIP-переключателях кода 0xCC (шестнадцатеричное значение) на светодиодные индикаторы должна выводиться анимация, показанная ниже. Во всех остальных случаях светодиодные индикаторы отражают инвертированное значение, выставленное на DIP-переключателях.



Лабораторная работа № 2

«Таймеры. Система прерываний»

Задание

Разработать и реализовать драйвер системного таймера микроконтроллера ADuC812. Написать тестовую программу с использованием разработанного драйвера по алгоритму, соответствующему варианту задания.

Таймеры-счетчики

Таймер позволяет производить отсчет временных интервалов заданной продолжительности. Принцип действия таймера основан на двоичном счетчике с возможностью предварительной записи исходного значения. После каждого такта синхросигнала счетчик прибавляет или отнимает единицу от имеющегося у него значения. При достижении нуля (т.е. при переполнении), счетчик вырабатывает активный уровень на выходе. Как правило, выходной сигнал таймера заводят на вход запроса прерывания микропроцессора или контроллера прерываний.

В большинстве современных встроенных систем таймеры используются в качестве основы для организации системы разделения времени на базе переключателя задач. В данном случае таймер используется в паре с механизмом прерываний.

Микроконтроллер ADuC812 имеет три программируемых 16-битных таймера/счетчика: таймер 0, таймер 1, таймер 2. Каждый таймер состоит из двух 8-битных регистров THX и TLX. Все три таймера могут быть настроены на работу в режимах «таймер» или «счетчик».

В режиме «таймер» регистр инкрементируется каждый машинный цикл, т.е. можно рассматривать это как подсчет машинных циклов. Так как машинный цикл состоит из 12 перепадов напряжения на тактовом входе микроконтроллера, частота инкрементирования таймера в 12 раз меньше тактовой частоты микроконтроллера (соответствующего кварцевого резонатора).

В режиме «счетчик» регистр инкрементируется по перепаду из «1» в «0» внешнего входного сигнала, подаваемого на вывод микроконтроллера T0, T1 или T2 (см. принципиальную электрическую схему контроллера SDK-1.1). Когда опрос показывает высокий уровень в текущем машинном цикле и низкий уровень в следующем, счетчик увеличивается на 1. Таким образом, на распознавание периода требуются два машинных цикла, максимальная частота подсчета входных сигналов равна 1/24 частоты кварцевого резонатора. На длительность периода входных сигналов ограничений сверху нет. Для гарантированного прочтения входной сигнал должен удерживать значение «1», как минимум, в течение одного машинного цикла микроконтроллера.

Таким образом, отличие этих двух режимов работы таймера заключается в типе источника инкрементирования. В остальном функционирование в режимах счетчика и таймера аналогично.

Описание работы

Данная лабораторная работа посвящена изучению таймера и системы прерываний микроконтроллера ADuC812. Основными функциями системного таймера являются: измерение интервалов времени и выполнение периодических задач. В данной работе с помощью таймеров требуется управлять светодиодными индикаторами (динамическая индикация) или звуковым излучателем (проигрывание мелодии), входящими в состав контроллера SDK-1.1.

Драйвер системного таймера должен состоять из двух частей: основной части, находящейся в обработчике прерывания и прикладной части – API-функций, используемых в программе:

Функция	Описание
`void InitTimer(void)`	Инициализация таймера.
`unsigned long GetMsCounter(void)`	Получение текущей метки времени в миллисекундах.
`unsigned long DTimeMs(unsigned long t0)`	Измерение количества миллисекунд, прошедших с временной метки `t0` и до текущего времени.
`void DelayMs(unsigned long t)`	Задержка на `t` миллисекунд.

Кроме того, могут быть реализованы функции работы с таймером в режиме «счетчик» (например, чтение счетчика).

Драйвер светодиодных индикаторов/звукового излучателя (зависит от варианта задания) должен быть реализован по тому же принципу, что и драйвер системного таймера. А именно: в обработчике прерывания от таймера должна выполняется сама динамическая индикация/проигрывание мелодии – примеры периодических задач, а через API-функции осуществляется настройка отображения анимации/звука и управление этими процессами.

Пояснения для вариантов заданий с использованием светодиодов

Написать драйвер, позволяющий управлять яркостью свечения линейки светодиодов. Драйвер должен обеспечивать независимое управление яркостью каждого светодиода. Яркость свечения светодиодов должна задаваться в процентах.

Яркость свечения светодиода можно регулировать, меняя скважность сигнала управляющего питанием светодиода. **Скважность** – это отношение периода следования (повторения) импульсов одной последовательности к их длительности. Величина, обратная скважности, называется **коэффициентом заполнения.** Сигнал со скважностью, равной двум, то есть длительностью импульсов равной длительности пауз между ними, называется **меандром**. Внешне меандр представляет собой прямоугольное колебание.

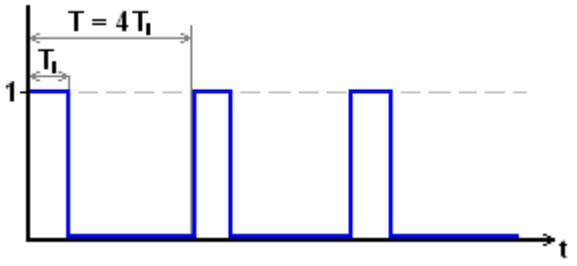


Рис. 3. Пример сигнала со скважностью 4

Таким образом, при скважности питающего сигнала равной 1, светодиод будет светиться с максимальной яркостью. При скважности сигнала управления равной 4 (см. рис. 3), светодиод будет светиться с яркостью 25% от максимальной и т.д. Кроме того, нужно стремиться уменьшить период и увеличить частоту сигнала управления, чтобы избежать эффекта мерцания светодиодов.

Пояснения для вариантов заданий с использованием звукового излучателя

Требуется написать драйвер звукового излучателя, позволяющий задавать частоту (в герцах) и длительность звука (в миллисекундах).

Для управления звуковым излучателем используется регистр ПЛИС ENA (адрес 080004h). 2-4 биты регистра ENA управляют величиной напряжения на динамике, т.е. позволяют задавать громкость звука[2]: чем больше «единиц» выставлено в этих битах, тем громче звук. Частота звука задается частотой смены нулей и единиц в управляющих битах регистра ENA. Ниже, в табл. 1, приведены частоты нот первой октавы.

Таблица 1. Частоты музыкальных нот

Нота	Частота, Гц
До	261,63
Ре	293,67
Ми	329,63
Фа	349,22
Соль	391,99
Ля	440,00
Си	493,88

Частоты нот каждой соседней октавы отличаются в 2 раза. Например, частота ноты «ля» второй октавы 880 Гц.

Особенности обработки прерываний в стенде SDK-1.1

В микроконтроллере ADuC812 девять источников прерываний с двумя уровнями приоритетов (см. табл. 2). Когда происходит прерывание, значение программного счетчика помещается в стек, а в сам счетчик загружается адрес соответствующего вектора прерывания.

Таблица 2. Адреса векторов прерывания микроконтроллера ADuC812

Прерывание	Наименование	Адрес вектора	Приоритет
PSMI	Монитор источника питания ADuC812	43H	1
IE0	Внешнее прерывание INT0	03H	2
ADCI	Конец преобразования АЦП	33H	3
TF0	Переполнение таймера/счетчика 0	0BH	4
IE1	Внешнее прерывание INT1	13H	5
TF1	Переполнение таймера/счетчика 1	1BH	6
I2CI/ISPI	Прерывание от I^2C/SPI	3BH	7
RI/TI	Прерывание от UART	23H	8
TF2/EXF2	Переполнение таймера/счетчика 2	2BH	9

Для каждого источника прерывания программист может задать один из двух уровней приоритета: высокий и низкий. Прерывание с высоким уровнем приоритета может прерывать обслуживание прерывания с низким уровнем приоритета, а если прерывания с разными уровнями происходят одновременно, то прерывание с высоким приоритетом будет обслужено первым. Обслуживание прерывания не может быть прервано прерыванием с таким же уровнем приоритета. Если два прерывания с одинаковым уровнем приоритета

[2] Такая возможность есть только в стендах SDK-1.1 ревизии R1 и R2.

происходят одновременно, то порядок их обработки определяется с помощью следующей табл. 3:

Таблица 3. Порядок обработки прерываний с одинаковым уровнем приоритета

Источник	Приоритет	Описание
PSMI	1 (Наивысший)	Монитор источника питания ADuC812
IE0	2	Внешнее прерывание INT0
ADCI	3	Конец преобразования АЦП
TF0	4	Переполнение таймера/счетчика 0
IE1	5	Внешнее прерывание INT1
TF1	6	Переполнение таймера/счетчика 1
I2CI+ISPI	7	Прерывание от I^2C/SPI
RI+TI	8	Прерывание от UART
TF2+EXF2	9 (Низший)	Переполнение таймера/счетчика 2

Прерывания ADuC812 имеют вектора в диапазоне 0003h-0043h, которые попадают в область младших адресов памяти программ. Это пространство соответствует 8Кб (0000h-2000h) FLASH-памяти.

В стенде SDK-1.1 пользователь не имеет возможности записи во FLASH-память (запись программ осуществляется во внешнюю память программ и данных), следовательно, не может подставить свои процедуры обработки прерываний (точнее, команды перехода к процедурам) по адресам, соответствующим векторам прерываний.

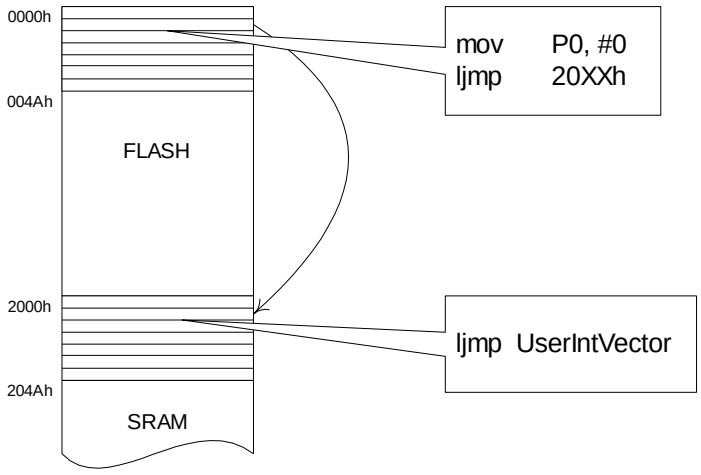


Рис. 4. Использование прерываний в SDK-1.1

Проблема использования прерываний в пользовательских программах решается следующим образом (см. рис. 4):

1. По адресам (0003h-0043h) векторов прерываний во FLASH-памяти SDK-1.1 располагаются команды переходов на вектора пользовательской таблицы, размещенной в адресах 2003h-2043h.
2. По адресам векторов пользовательской таблицы пользователем указываются команды переходов на процедуры обработки прерываний.

Приведем пример помещения собственного вектора в пользовательскую таблицу. Программа использует прерывание от таймера 0 (прерывание 0Bh) для зажигания линейки светодиодов. Данный пример адаптирован для компилятора SDCC. По сравнению с компилятором Keil C51 изменения коснулись функции SetVector (в связи с тем, что компилятор SDCC и микроконтроллер ADuC812 используют различные способы записи

многобайтовых чисел в памяти; SDCC – little-endian, ADuC812 – big-endian) и заголовка функции обработчика прерывания (T0_ISR).

```
#include "aduc812.h"
#define MAXBASE         8

////////////////////////// WriteMax /////////////////////////////
// Запись байта в регистр ПЛИС
// Вход:
//   regnum – адрес регистра ПЛИС,
//   val – записываемое значение.
// Выход: нет.
// Результат: нет.
/////////////////////////////////////////////////////////////////
void WriteMax (unsigned char xdata *regnum, unsigned char val)
{
      // Сохранение текущего значения регистра страниц
      unsigned char oldDPP = DPP;

      DPP = MAXBASE;    // Установка адреса страницы ПЛИС
      *regnum = val;    // Запись значения в регистр ПЛИС
      DPP = oldDPP;     // Восстановление сохраненного значения
                        // регистра страниц
}

////////////////////////// WriteLED  /////////////////////////////
// Функция установки состояния линейки светодиодов.
// Вход:
//   value – состояния светодиодов.
// Выход: нет.
// Результат: нет.
/////////////////////////////////////////////////////////////////
void WriteLED(unsigned char value)
{
      // Запись состояния светодиодов в регистр 7-й регистр ПЛИС
      WriteMax( 7, value );
}

////////////////////////// T0_ISR ///////////////////////////////
// Обработчик прерывания от таймера 0.
// Вход: нет.
// Выход: нет.
// Результат: нет.
/////////////////////////////////////////////////////////////////
void T0_ISR( void ) __interrupt ( 1 )
{
      WriteLED( 0x55 ); // Зажигание светодиодов (через один)
}

////////////////////////// SetVector  ////////////////////////////
// Функция, устанавливающая вектор прерывания в
// пользовательской таблице прерываний.
// Вход:
//   Vector – адрес обработчика прерывания,
//   Address – вектор пользовательской таблицы прерываний.
// Выход: нет.
// Результат: нет.
/////////////////////////////////////////////////////////////////
void SetVector(unsigned char xdata * Address, void * Vector)
{
      unsigned char xdata * TmpVector;    // Временная переменная

      // Первым байтом по указанному адресу записывается
```

```
        // код команды передачи управления ljmp, равный 02h
        *Address = 0x02;

        // Далее записывается адрес перехода Vector
        TmpVector = (unsigned char xdata *) (Address + 1);
        *TmpVector = (unsigned char) ((unsigned short)Vector >> 8);
        ++TmpVector;
        *TmpVector = (unsigned char) Vector;
        // Таким образом, по адресу Address теперь
        // располагается инструкция ljmp Vector
}

////////////////////////// Main ////////////////////////////////
// Главная функция
////////////////////////////////////////////////////////////////
void main( void )
{
        TH0 = 0xFF;         // Инициализация таймера 0
        TL0 = 0xF0;         //
        TMOD = 0x01;        //
        TCON = 0x10;        //

        // Установка вектора в пользовательской таблице
        SetVector( 0x200B, (void *)T0_ISR );
        // Разрешение прерываний от таймера 0
        ET0 = 1; EA = 1;

        while( 1 );
}
```

В ходе выполнения лабораторной работы производится ознакомление с организацией и принципом работы не только таймеров/счетчиков по прерываниям, но и внешних прерываний INT0/INT1. Необходимо отметить, что в случае последних возможны следующие настройки: по перепаду (фронт или спад напряжения) или по уровню напряжения на внешнем входе. В контроллере SDK-1.1 линии счетного входа таймера/счетчика 0/1 (T0/1) и внешних прерываний INT0/INT1 выведены на DIP-переключатели SW3-1 (см. рис. 5).

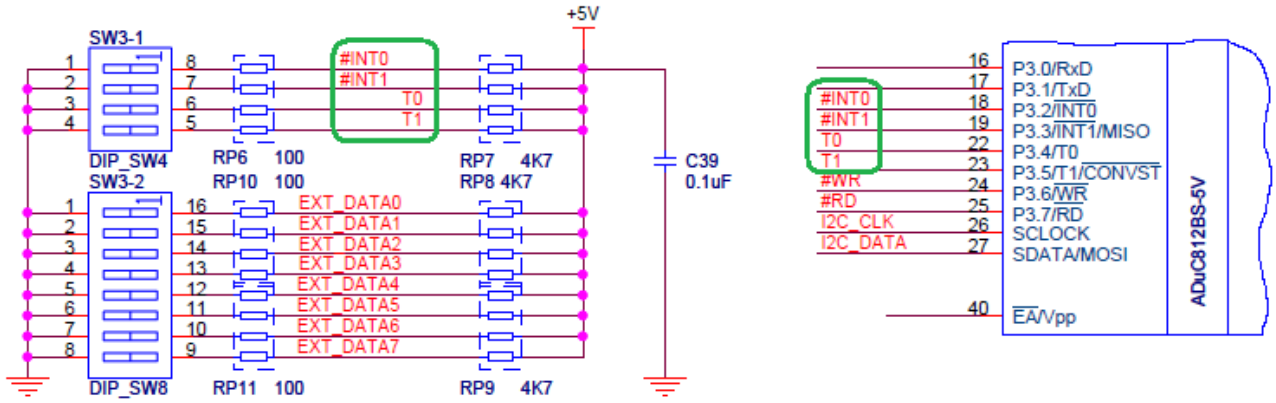


Рис. 5. Расположение линий T0/1 и INT0/1 МК ADuC812 (справа) и их вывод на DIP-переключатели (слева) на принципиальной электрической схеме контроллера SDK-1.1

Необходимо настроить внешнее прерывание INT0/1 так, чтобы оно работало по спаду, а не по уровню (регистр специального назначения TCON). Для корректной работы внешнего прерывания INT0 5-й бит регистра управления ENA (адрес 0x080004 ПЛИС) должен быть равен 1.

Требования к выполнению работы

1. Все программы должны быть написаны на языке Си.
2. Разрабатываемые драйверы устройств должны быть выполнены в виде отдельных программных модулей (файлов), содержащих функции по работе с заданным одним устройством.
3. В программе должны быть использованы механизмы взаимного исключения (см. [14], IOS2003_lab4.pdf).
4. В варианте задания с использованием светодиодов представлен лишь фрагмент анимации, при этом все анимации являются циклическими. По приведенному фрагменту требуется определить алгоритм анимации и реализовать его. Смена яркости свечения светодиодов должна быть плавной, без видимых «рывков» и эффекта «мерцания».
5. Текст программы должен соответствовать правилам оформления программ на языке Си, приведенным в приложении (Приложение Б. Требования к оформлению программ на языке Си, [28]).

Содержание отчета

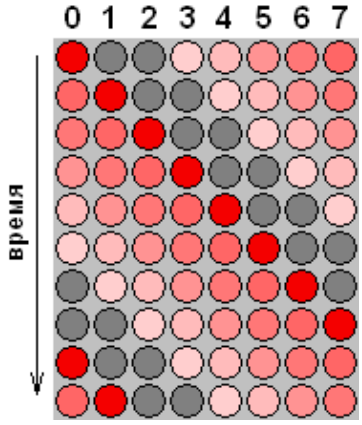
1. Титульный лист.
2. Задание.
3. Модель взаимодействия прикладной программы, прикладной части драйверов и системной части драйверов.
4. Исходный текст программы с комментариями.
5. Основные результаты.

Литература

Литература к лабораторной работе: [9], [11], [12], [13], [14], [16], [21], [26], [27], [30], [32], [33], [34], [38], [42].

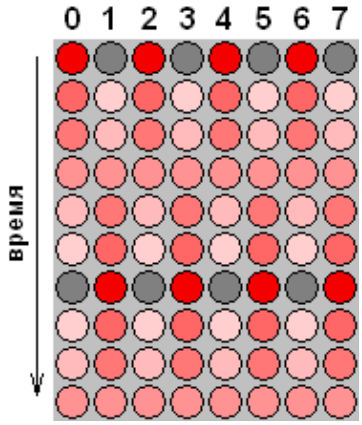
Варианты заданий

1. Контроллер SDK-1.1 на линейку светодиодов циклически отображает приведенную ниже анимацию или выводит количество замыканий входа T0 (счетный вход таймера 0 на рис. 5). Подсчет количества замыканий входа должен быть реализован с помощью таймера-счетчика 0. Смена режима отображения производится по поступлению сигнала внешнего прерывания INT0 (рис. 5). В анимации должны использоваться не менее 6 градаций яркости свечения светодиодов (например, 0%, 20%, 40%, 60%, 80% и 100%). В результате выполнения работы должны быть разработаны драйверы системного таймера, таймера-счетчика, светодиодных индикаторов, внешнего прерывания.



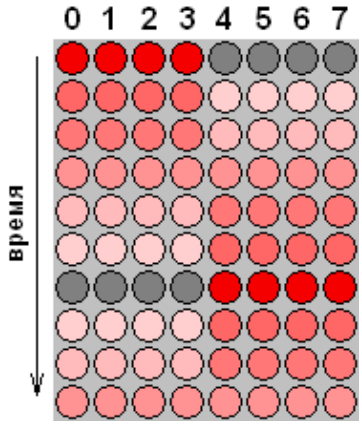
2. Контроллер SDK-1.1 циклически проигрывает восходящую гамму нот первой октавы (длительность каждой ноты – 1 секунда) и на линейку светодиодов выводит количество замыканий входа T0 (счетный вход таймера 0 на рис. 5). Подсчет количества замыканий входа должен быть реализован с помощью таймера-счетчика 0. В результате выполнения работы должны быть разработаны драйверы системного таймера, таймера-счетчика, звукового излучателя, светодиодных индикаторов.

3. Контроллер SDK-1.1 на линейку светодиодов циклически отображает приведенную ниже анимацию или выводит количество замыканий входа T1 (счетный вход таймера 1 на рис. 5). Подсчет количества замыканий входа должен быть реализован с помощью таймера-счетчика 1. Смена режима отображения производится по поступлению сигнала внешнего прерывания INT1 (рис. 5). В анимации должны использоваться не менее 6 градаций яркости свечения светодиодов (например, 0%, 20%, 40%, 60%, 80% и 100%). В результате выполнения работы должны быть разработаны драйверы системного таймера, таймера-счетчика, светодиодных индикаторов, внешнего прерывания.



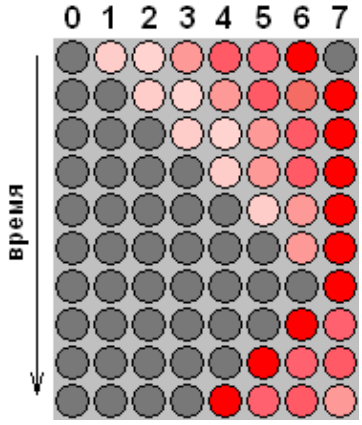
4. Контроллер SDK-1.1 циклически проигрывает нисходящую гамму нот второй октавы (длительность каждой ноты – 1 секунда) и на линейку светодиодов выводит количество замыканий входа T1 (счетный вход таймера 1 на рис. 5). Подсчет количества замыканий входа должен быть реализован с помощью таймера-счетчика 1. В результате выполнения работы должны быть разработаны драйверы системного таймера, таймера-счетчика, звукового излучателя, светодиодных индикаторов.

5. Контроллер SDK-1.1 на линейку светодиодов циклически отображает приведенную ниже анимацию или выводит количество замыканий входа T0 (счетный вход таймера 0 на рис. 5). Подсчет количества замыканий входа должен быть реализован с помощью таймера-счетчика 0. Смена режима отображения производится по поступлению сигнала внешнего прерывания INT1 (рис. 5). В анимации должны использоваться не менее 6 градаций яркости свечения светодиодов (например, 0%, 20%, 40%, 60%, 80% и 100%). В результате выполнения работы должны быть разработаны драйверы системного таймера, таймера-счетчика, светодиодных индикаторов, внешнего прерывания.



6. Контроллер SDK-1.1 циклически проигрывает восходящую гамму нот первой октавы (длительность каждой ноты – 0,5 секунды) и на линейку светодиодов выводит количество замыканий входа INT0 (линия внешнего прерывания INT0 на рис. 5). В результате выполнения работы должны быть разработаны драйверы системного таймера, звукового излучателя, светодиодных индикаторов, счетчика срабатываний внешнего прерывания.

7. Контроллер SDK-1.1 на линейку светодиодов циклически отображает приведенную ниже анимацию или выводит количество замыканий входа T1 (счетный вход таймера 1 на рис. 5). Подсчет количества замыканий входа должен быть реализован с помощью таймера-счетчика 1. Смена режима отображения производится по поступлению сигнала внешнего прерывания INT0 (рис. 5). В анимации должны использоваться не менее 6 градаций яркости свечения светодиодов (например, 0%, 20%, 40%, 60%, 80% и 100%). В результате выполнения работы должны быть разработаны драйверы системного таймера, таймера-счетчика, светодиодных индикаторов, внешнего прерывания.



8. Контроллер SDK-1.1 циклически проигрывает нисходящую гамму нот второй октавы (длительность каждой ноты – 0,5 секунды) и на линейку светодиодов выводит количество замыканий входа INT1 (линия внешнего прерывания INT1 на рис. 5). В результате

выполнения работы должны быть разработаны драйверы системного таймера, звукового излучателя, светодиодных индикаторов, счетчика срабатываний внешнего прерывания.

Лабораторная работа № 3

«Последовательный интерфейс RS-232. UART»

Задание

Разработать и написать драйверы последовательного канала для учебно-лабораторного стенда SDK-1.1 с использованием и без использования прерываний. Написать тестовую программу для разработанных драйверов, которая выполняет определенную вариантом задачу.

Контроллер последовательного интерфейса RS-232

Интерфейс RS-232 – стандартный интерфейс, предназначенный для последовательной двоичной передачи данных между терминальным (DTE, Data Terminal Equipment) и связным (DCE, Data Communications Equipment) оборудованием (рис. 6).

Рис. 6. Соединение двух удаленных терминалов

Чтобы не составить неправильного представления об интерфейсе RS-232, необходимо отчетливо понимать различие между этими видами оборудования. Терминальное оборудование, например микрокомпьютер, может посылать и (или) принимать данные по последовательному интерфейсу. Оно как бы оканчивает (terminate) последовательную линию. Связное оборудование — устройства, которые могут упростить передачу данных совместно с терминальным оборудованием. Наглядным примером связного оборудования служит модем (модулятор–демодулятор). Он оказывается соединительным звеном в последовательной цепочке между компьютером и телефонной линией.

Различие между терминальными и связными устройствами довольно расплывчато, поэтому возникают некоторые сложности в понимании того, к какому типу оборудования относится то или иное устройство. Рассмотрим ситуацию с принтером. К какому оборудованию его отнести? Как связать два компьютера, когда они оба действуют как терминальное оборудование.

Информация передается по проводам с уровнями сигналов, отличающимися от обычных уровней цифрового сигнала (5В, 3.3В и т.п.), для обеспечения большей устойчивости к помехам. Асинхронная передача данных осуществляется с установленной скоростью при синхронизации уровнем сигнала стартового импульса. RS-232 используется для передачи данных на небольшое расстояние (единицы - десятки метров) с небольшой скоростью (обычно, не быстрее 115200 бит/сек). Для формирования уровня сигнала используются микросхемы приёмопередатчиков, а для формирования и распознавания посылок – микросхемы UART.

Модуль универсального синхронно-асинхронного приемопередатчика (Universal Synchronous/Asynchronous Receiver and Transmitter, USART) стал стандартом «де-факто» среди контроллеров последовательного обмена. В названии часто опускают слово «синхронный», и модуль не совсем корректно именуется UART (чисто асинхронные приемопередатчики сейчас встречаются достаточно редко). Характеристики

последовательного порта UART не позволяют производить приём и передачу данных за пределы печатной платы. Для связи с другими устройствами сигнал от UART необходимо пропустить через приёмопередатчик, работающий в одном из стандартов:

- RS-232;
- RS-485;
- RS-422.

Обычно модули UART в асинхронном режиме поддерживают протокол обмена для интерфейса RS-232 (8N1 или 9N1); в синхронном режиме – нестандартные синхронные протоколы, в некоторых случаях – протокол SPI.

Приемопередатчик – преобразователь уровня, как правило, в интегральном исполнении. Предназначен для преобразования электрических сигналов из уровня ТТЛ в уровень, соответствующий физическому уровню определенного стандарта.

Контроллер UART обычно содержит:

1. Источник тактирования (обычно с увеличенной частотой тактирования по сравнению со скоростью обмена, чтобы иметь возможность отслеживать состояние линии передачи данных в середине передачи бита).
2. Входные и выходные сдвиговые регистры.
3. Регистры управления приемом/передачей данных, чтением/записью.
4. Буферы приема/передачи.
5. Параллельная шина данных для буферов приема/передачи.
6. FIFO-буферы памяти (опционально).

Особенности последовательного интерфейса в микроконтроллере с ядром MCS-51

Последовательный порт в микроконтроллерах MCS-51 позволяет осуществлять последовательный дуплексный ввод-вывод в синхронном и асинхронном режимах с разными скоростями обмена. Помимо обычного ввода-вывода, в нем предусмотрена аппаратная поддержка взаимодействия нескольких микроконтроллеров.

Схематически контроллер последовательного порта представлен на рис. 7. Как видно из рисунка, регистр SBUF, доступный пользователю для чтения и записи, есть на самом деле два регистра, в один из которых можно только записывать, а из другого – читать. Контроллер позволяет дуплексный обмен данными, т.е. одновременно может передавать и принимать информацию по линиям TxD (P3.1) и RxD (P3.0) соответственно.

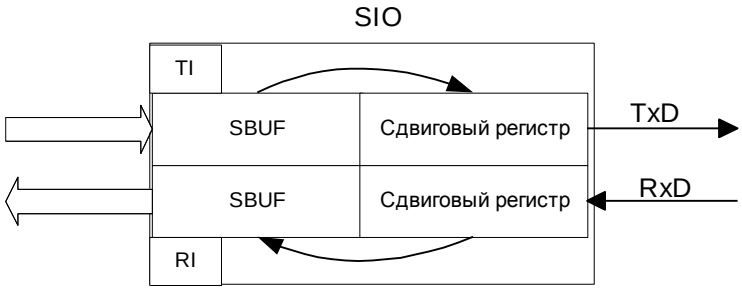


Рис. 7. Схема контроллера UART МК ADuC812

Передача инициируется записью в SBUF байта данных. Этот байт переписывается в сдвиговый регистр, из которого пересылается бит за битом. Как только байт будет переслан целиком, устанавливается флаг TI, сигнализирующий о том, что контроллер готов к передаче очередного байта.

Прием осуществляется в обратном порядке: после начала процесса приема принятые биты данных последовательно сдвигаются в сдвиговом регистре, пока не будет принято их установленное количество, затем содержимое сдвигового регистра переписывается в SBUF и

устанавливается флаг RI. После этого возможен прием следующей последовательности бит. Необходимо отметить, что запись принятого байта из сдвигового регистра в SBUF происходит только при условии, что RI=0, поэтому при заборе пользовательской программой очередного байта из SBUF ей необходимо сбрасывать этот флаг, иначе принятый в сдвиговый регистр, но не записанный в SBUF, следующий байт будет безвозвратно утерян.

Контроллер последовательного порта может работать в одном из четырех режимов (один синхронный и три асинхронных), различающихся скоростями обмена (бод) и количеством передаваемых/принимаемых бит:

- Режим 0 (синхронный): данные передаются и принимаются через RxD, TxD является синхронизирующим (выдает импульсы сдвига). Скорость в этом режиме фиксирована (1/12 частоты тактового генератора).
- Режим 1 (8 бит данных, асинхронный, переменная скорость). Передаются 10 бит: старт-бит, 8 бит данных (SBUF) и стоп-бит.
- Режим 2 (9 бит данных, асинхронный, фиксированная скорость). Передаются 11 бит: старт-бит, 8 бит (SBUF), бит TB8/RB8 (посылка/прием соответственно) и 1 стоп-бит. Биты TB8 и RB8 содержатся в регистре SCON (биты 3 и 2 соответственно), первый устанавливается программно, а второй содержит 9-й бит принятой комбинации. С помощью них можно организовать, например, контроль четности или обмен с двумя стоп-битами.
- Режим 3 (9 бит данных, асинхронный, переменная скорость). То же, что и режим 2, только скорость обмена переменная.

В режиме 1 и 3 используются Таймер 0 и/или Таймер 1 для настройки скорости обмена.

Организация буферизированного последовательного ввода-вывода в стенде SDK-1.1

В микроконтроллере ADuC812 прерывание от последовательного канала имеет номер 4, адрес-вектор 0x23 и генерируется контроллером последовательного порта во время установки флага TI или RI. Чтобы разрешить это прерывание, нужно установить биты ES и EA регистра IEN0.

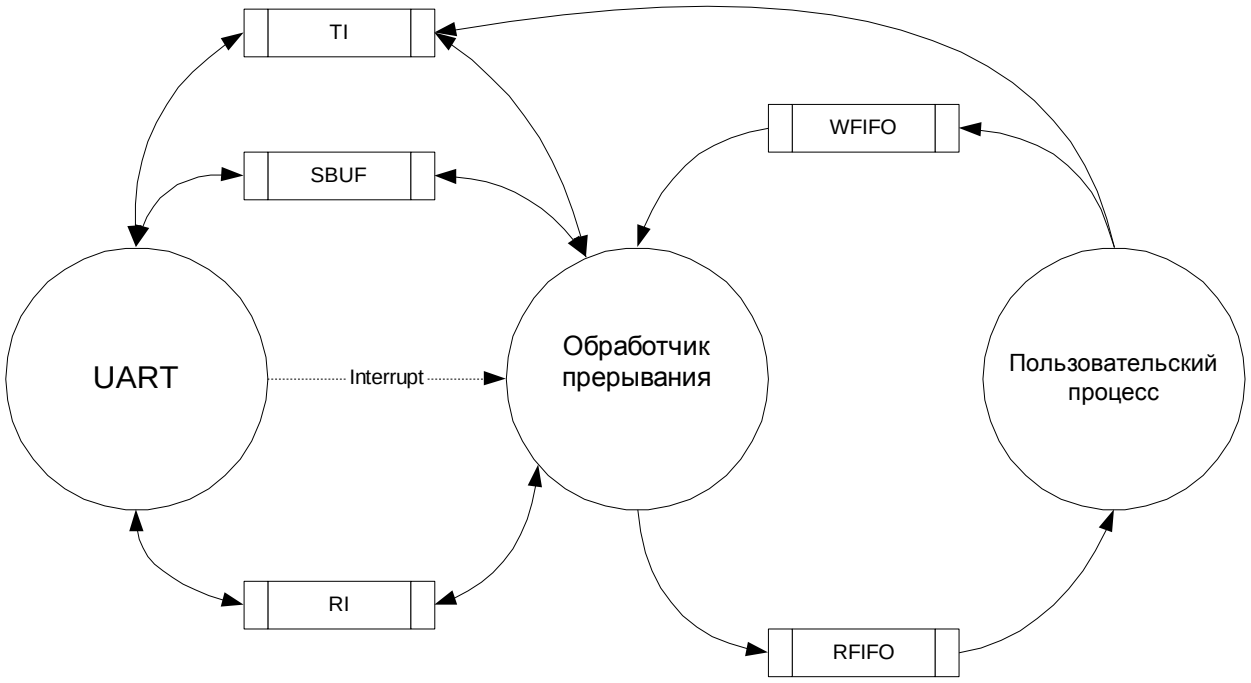


Рис. 8. Драйвер UART (диаграмма DFD)

Управление обработчику прерывания передается тогда, когда UART либо готов к передаче очередного байта (TI=1), либо принял байт из канала (RI=1). При использовании этой схемы появляется проблема посылки первого байта.

Пусть прикладная программа желает послать байт, когда очередь WFIFO пуста и в порт данные не записывались длительное время (TI=0). Так как обработчик прерываний, обслуживающий очередь, вызывается только при установленных флагах RI или TI, прикладной программе придется либо самой записать первый байт в SBUF (начав тем самым процесс передачи), либо, записав байты для передачи в очередь WFIFO, искусственно установить флаг TI (вызвав тем самым обработчик, который обслужит очередь). Проблема разрешима, однако при ее решении необходимо учитывать, что TI также сброшен, когда контроллер занят посылкой байта, даже если очередь (WFIFO) пуста (посылка последнего байта в очереди).

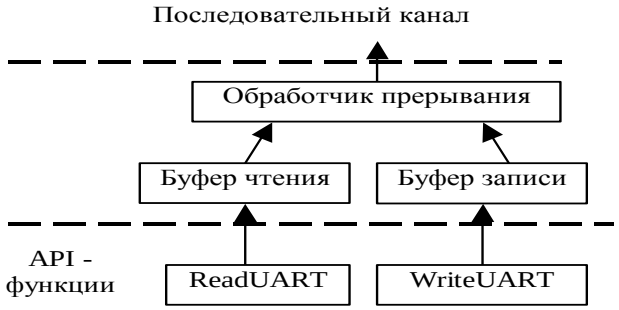
Организация программы

На рис. 9 изображены основные компоненты программы, которая должна быть результатом выполнения задания: драйвер последовательного канала; разборщик/преобразователь данных, переданных (принятых) по UART; пользовательский процесс, в котором выполняется прикладная задача.

Рис. 9. Основные составляющие программы

Драйвер последовательного канала, организованного по опросу, должен содержать три функции: функция инициализации UART, функция передачи и приема байта данных по UART.

Драйвер последовательного канала по прерыванию состоит из функции инициализации UART, обработчика прерывания от UART, циклических буферов чтения и записи, API-функций: чтения байта из последовательного канала и записи байта в последовательный канал. Взаимодействие обработчика и API-функций осуществляется только через буфер.



Обработчик взаимодействует с последовательным каналом напрямую. Если прерывание вызвано приходом байта по последовательному каналу, то обработчик записывает байт в буфер чтения RFIFO. Если прерывание вызвано окончанием передачи, то обработчик проверяет состояние буфера записи WFIFO, и, если он не пуст, считывает и пересылает следующий байт.

Функция чтения байта ReadUART() проверяет состояние буфера RFIFO, и если он не пуст, считывает из него байт. Функция WriteUART() записывает байт в буфер записи последовательного канала WFIFO. Если до записи буфер был пуст, необходимо аппаратно вызвать обработчик прерывания, чтобы инициировать передачу данных. В случае, когда к моменту записи в буфере уже находятся какие-либо данные, обработчик вызывать не надо, так как запись была инициирована раньше

и в данный момент последовательный канал занят передачей данных, записанных в буфер ранее.

Принятые при помощи драйвера UART данные требуют обработки в соответствии с вариантом задания: смена нижнего регистра на верхний, фильтрация только по числовым или буквенным значениям, по количеству вводимых значений и др. Таким образом, требуется написать функцию (разборщик/преобразователь), которая будет отфильтровывать значения, неудовлетворяющие заданному алфавиту корректных символов. Далее над отфильтрованными данными производится указанная в варианте задания операция: арифметическое действие или конвертирование. Обработка ошибок в разборщике/преобразователе и прикладном алгоритме тоже должна быть предусмотрена.

Кроме того, для выполнения задания требуется использовать драйвер DIP-переключателей и светодиодных индикаторов.

Организация обработчика прерывания UART

Окончание передачи байта и окончание приема байта по последовательному каналу вызывают одно и тоже прерывание и в обработчике два этих прерывания различаются программно. Обработчик логически разделен на две части: одна работает при установленном флаге RI (окончание приема), а другая при установленном флаге TI (окончание передачи). Ниже приведен шаблон обработчика прерывания последовательного канала:

```
////////////////////////// SIO_ISR ////////////////////////////////
// Обработчик прерывания UART.
// Вход: нет.
// Выход: нет.
// Результат: нет.
//////////////////////////////////////////////////////////////////
void SIO_ISR( void ) __interrupt ( 4 )
{
      if(TI)
      {
          // Передача байта
          // Читаем WFIFO (буфер передачи) и записываем его в SBUF
      }

      if(RI)
      {
          // Прием байта
          // Читаем SBUF и записываем его в RFIFO (буфер приема)
      }
}
```

Кроме того, функция SetVector должна быть такой (см. лабораторную работу № 2):

```
////////////////////////// SetVector  ////////////////////////////
// Функция, устанавливающая вектор прерывания в
// пользовательской таблице прерываний.
// Вход:
//   Vector – адрес обработчика прерывания,
//   Address – вектор пользовательской таблицы прерываний.
// Выход: нет.
// Результат: нет.
//////////////////////////////////////////////////////////////////
void SetVector(unsigned char xdata * Address, void * Vector)
{
      unsigned char xdata * TmpVector;

      *Address = 0x02;
```

```
      TmpVector = (unsigned char xdata *) (Address + 1);
      *TmpVector = (unsigned char) ((unsigned short)Vector >> 8);
      ++TmpVector;
      *TmpVector = (unsigned char) Vector;
}
```

Описание работы

Данная лабораторная работа посвящена изучению организации и принципов работы контроллера последовательного канала UART микроконтроллера ADuC812 стенда SDK-1.1.

В рамках лабораторной работы необходимо разработать программу для контроллера SDK-1.1 (ведомый), который обменивается данными с персональным компьютером (ведущий). В качестве канала связи используется последовательный канал RS-232. На стороне персонального компьютера имеется инструментальное средство для обеспечения взаимодействия с SDK-1.1 – это терминальная программа M3P. С описанием инструментальной системы (G)M3P можно ознакомиться в соответствующем руководстве пользователя.

Программа должна выполнять две задачи в соответствии с вариантом задания. Первая задача выполняется при помощи драйвера последовательного канала, работающего по опросу, – это реализация так называемого “эха”: со стороны персонального компьютера передаются символы контроллеру SDK-1.1, на которые контроллер отвечает определенным образом (см. вариант задания). Вторая задача выполняется при помощи драйвера последовательного канала, реализованного в режиме прерываний с буферизацией байтов в буфере FIFO (см. [14], IOS2003_lab4.pdf). Такой задачей является реализация устройства, которое выполняет одну арифметическую операцию над десятичными числами, или конвертора из одной системы счисления в другую. Переключение между двумя задачами в программе должно быть выполнено с использованием DIP-переключателей (лабораторная работа № 1).

Рекомендации к выполнению работы:

1. Написать драйвер последовательного канала по опросу и простую тестовую программу для него.
2. Написать драйвер последовательного канала, осуществляющий асинхронный обмен данными по прерыванию, и отладить при помощи уже написанной тестовой программы (см. п. 1).
3. Написать тестовую программу для разработанных драйверов, которая выполняет две задачи в соответствии с вариантом задания.

Требования к выполнению работы

1. В тестовой программе должна быть продемонстрирована работа с последовательным интерфейсом и по опросу, и асинхронно по прерыванию, т.е. должны быть реализованы и использованы драйверы UART двух видов для решения двух разных задач, указанных в варианте задания.
2. Разрабатываемые драйверы устройств должны быть выполнены в виде отдельных программных модулей (файлов), содержащих функции по работе с заданным одним устройством.
3. Должен быть предусмотрен контроль ввода корректных значений, выполнения арифметических действий в рамках второй задачи программы.
4. В программе должны быть использованы механизмы взаимного исключения (см. [14], IOS2003_lab4.pdf).

5. Текст программы должен соответствовать правилам оформления программ на языке Си, приведенным в приложении (Приложение Б. Требования к оформлению программ на языке Си, [28]).

Содержание отчета

1. Титульный лист.
2. Номер варианта, задание.
3. Иллюстрация организации и функционирования разработанного программного обеспечения (драйверы, тестовая программа) в виде блок-схемы, диаграммы процессов, потоков данных, диаграммы состояний автоматов и других способов структурного и поведенческого описания программы (по выбору студента).
4. Исходный текст программы с комментариями.
5. Основные результаты.

Литература

Литература к лабораторной работе: [3], [4], [13], [14], [15], [21], [26], [30].

Варианты заданий

Каждый вариант состоит из двух частей: первая выполняется с использованием драйвера последовательного канала по опросу, вторая – по прерыванию. Переключение между двумя частями в тестовой программе должно быть выполнено с использованием DIP-переключателей (лабораторная работа № 1).

1. Скорость последовательного канала – 9600 бит/с.
    - Каждый принятый по последовательному каналу символ (от персонального компьютера к SDK-1.1) передается в утроенном виде в обратную сторону (от SDK-1.1 к персональному компьютеру) и отображается в терминальной программе. Причем все символы русского алфавита отображаются в нижнем регистре, все символы английского алфавита – в верхнем регистре. Например, на символ ‘л’ (‘Л’) ответом является ‘ллл’, ‘1’ – ‘111’, ‘i’ (‘I’) – ‘III’ и т.д.
    - Конвертор из десятичной в двоичную систему счисления. Диапазон преобразуемых значений – от 0_{10} до 255_{10} включительно. 8-разрядная сетка для отображения двоичных чисел. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы передается десятичное число для конвертирования, причем число это отображается в терминале, а концом ввода является перевод на следующую строку (<CR><LF>). После чего контроллер возвращает результат преобразования числа в двоичную систему счисления, который отображается в терминале персонального компьютера и на светодиодных индикаторах стенда SDK-1.1 (лабораторная работа № 1). Каждое новое преобразование начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал.

2. Скорость последовательного канала – 9600 бит/с.
    - Со стороны персонального компьютера с использованием терминальной программы контроллеру SDK-1.1 по последовательному каналу передается любой символ русского алфавита в любом регистре (‘а’, ‘Б’, ‘в’,…, ‘Я’). В ответ

контроллер SDK-1.1 передает принятый символ в верхнем регистре и 5 символов русского алфавита, следующих за введенным, в нижнем регистре. Все остальные вводимые символы игнорируются контроллером SDK-1.1. Например, на символ ‘к’ (или ‘К’) ответом является ‘Клмноп’, ‘у’ (‘У’) – ‘Уфхцчш’, ‘э’ (‘Э’) – ‘Эюя’ и т.д. Каждому обмену данными между персональным компьютером и стендом SDK-1.1 назначается отдельная строка.

- Сумматор десятичных чисел. Диапазон значений слагаемых – от 0_{10} до 99_{10} включительно. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы передаются два слагаемых (десятичные числа), причем разделителем слагаемых является символ сложения (‘+’), концом ввода является символ равенства (‘=‘), получившееся выражение отображается в терминале персонального компьютера. После чего контроллер возвращает результат сложения, который отображается в терминале. Каждое новое выражение начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал.

3. Скорость последовательного канала – 4800 бит/с.
   - Со стороны персонального компьютера с использованием терминальной программы контроллеру SDK-1.1 по последовательному каналу передается любой символ английского алфавита в нижнем регистре (‘a’, ‘b’, ‘c’,…, ‘z’). В ответ контроллер SDK-1.1 передает принятый символ в нижнем регистре и 5 символов английского алфавита, следующих за введенным, в верхнем регистре. Все остальные вводимые символы игнорируются контроллером SDK-1.1. Например, на символ ‘u’ ответом является ‘uVWXYZ’, ‘m’ – ‘mNOPQRS’, ‘w’ – ‘wXYZ’ и т.д. Каждому обмену данными между персональным компьютером и стендом SDK-1.1 назначается отдельная строка.
   - Конвертор из шестнадцатеричной в двоичную систему счисления. Диапазон преобразуемых значений – от 0_{16} до FF_{16} включительно. 8-разрядная сетка для отображения двоичных чисел. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы передается шестнадцатеричное число для конвертирования, причем число это отображается в терминале, а концом ввода является перевод на следующую строку (<CR><LF>). После чего контроллер возвращает результат преобразования числа в двоичную систему счисления, который отображается в терминале персонального компьютера и на светодиодных индикаторах стенда SDK-1.1 (лабораторная работа № 1). Каждое новое преобразование начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал.

4. Скорость последовательного канала – 4800 бит/с.
   - На каждый принятый по последовательному каналу символ (от персонального компьютера к SDK-1.1) в ответ передается этот же символ и 2 следующих за ним символа согласно таблице ASCII (от SDK-1.1 к персональному компьютеру) и отображается в терминальной программе. Причем все символы русского алфавита отображаются в верхнем регистре, все символы английского алфавита – в нижнем регистре. Например, на символ ‘л’ (‘Л’) ответом является ‘ЛМН’, ‘1’ – ‘123’, ‘i’ (‘I’) – ‘ijk’ и т.д.
   - Вычитатель десятичных чисел. Диапазон значений уменьшаемого и вычитаемого – от 0_{10} до 99_{10} включительно. Разность может быть как положительной, так и отрицательной. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы

передаются уменьшаемое и вычитаемое (десятичные числа), причем разделителем введенных значений является символ вычитания (‘-’), концом ввода является символ равенства (‘=‘), получившееся выражение отображается в терминале персонального компьютера. После чего контроллер возвращает результат операции, который отображается в терминале. Каждое новое выражение начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал.

5. Скорость последовательного канала – 1200 бит/с.
   - Со стороны персонального компьютера с использованием терминальной программы контроллеру SDK-1.1 по последовательному каналу передается любой числовой символ (‘0’, ‘1’, ‘2’,…, ‘9’). В ответ контроллер SDK-1.1 передает принятый символ и все остальные числовые символы, следующие за введенным. Все остальные вводимые символы игнорируются контроллером SDK-1.1. Например, на символ ‘4’ ответом является ‘456789’, ‘8’ – ‘89’, ‘1’ – ‘123456789’ и т.д. Каждому обмену данными между персональным компьютером и стендом SDK-1.1 назначается отдельная строка.
   - Конвертор из десятичной в шестнадцатеричную систему счисления. Диапазон преобразуемых значений – от 0_{10} до 255_{10} включительно. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы передается десятичное число для конвертирования, причем число это отображается в терминале, а концом ввода является перевод на следующую строку (<CR><LF>). После чего контроллер возвращает результат преобразования числа в шестнадцатеричную систему счисления, который отображается в терминале персонального компьютера. Каждое новое преобразование начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал и зажигание светодиодов (лабораторная работа № 1).

6. Скорость последовательного канала – 1200 бит/с.
   - На каждый принятый по последовательному каналу символ (от персонального компьютера к SDK-1.1) в ответ передается этот же символ и 2 предшествующих ему символа согласно таблице ASCII (от SDK-1.1 к персональному компьютеру) и отображается в терминальной программе. Причем все символы русского алфавита отображаются в верхнем регистре, все символы английского алфавита – в нижнем регистре. Например, на символ ‘л’ (‘Л’) ответом является ‘ЛКЙ’, ‘5’ – ‘543’, ‘i’ (‘I’) – ‘ihg’ и т.д.
   - Умножитель десятичных чисел. Диапазон значений множителей – от 0_{10} до 99_{10} включительно. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы передаются множители (десятичные числа), причем разделителем введенных значений является символ умножения (‘*’), концом ввода является символ равенства (‘=‘), получившееся выражение отображается в терминале персонального компьютера. После чего контроллер возвращает результат операции, который отображается в терминале. Каждое новое выражение начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал и зажигание светодиодов (лабораторная работа № 1).

7. Скорость последовательного канала – 2400 бит/с.
   - Со стороны персонального компьютера с использованием терминальной программы контроллеру SDK-1.1 по последовательному каналу передается

любой символ английского алфавита в нижнем регистре (‘a’, ‘b’, ‘c’,…, ‘z’). В ответ контроллер SDK-1.1 передает принятый символ в нижнем регистре и все символы английского алфавита, предшествующие введенному, в верхнем регистре. Все остальные вводимые символы игнорируются контроллером SDK-1.1. Например, на символ ‘f’ ответом является ‘fEDCBA’, ‘j’ – ‘jIHGFEDCBA’, ‘p’ – ‘pONMLKJIHGFEDCBA’ и т.д. Каждому обмену данными между персональным компьютером и стендом SDK-1.1 назначается отдельная строка.

- Конвертор из шестнадцатеричной в десятичную систему счисления. Диапазон преобразуемых значений – от 0_{16} до FF_{16} включительно. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы передается шестнадцатеричное число для конвертирования, причем число это отображается в терминале, а концом ввода является перевод на следующую строку (<CR><LF>). После чего контроллер возвращает результат преобразования числа в десятичную систему счисления, который отображается в терминале персонального компьютера. Каждое новое преобразование начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал и зажигание светодиодов (лабораторная работа № 1).

8. Скорость последовательного канала – 2400 бит/с.
    - Со стороны персонального компьютера с использованием терминальной программы контроллеру SDK-1.1 по последовательному каналу передается любой числовой символ (‘0’, ‘1’, ‘2’,…, ‘9’). В ответ контроллер SDK-1.1 передает принятый символ и все остальные числовые символы, предшествующие введенному. Все остальные вводимые символы игнорируются контроллером SDK-1.1. Например, на символ ‘4’ ответом является ‘43210’, ‘8’ – ‘876543210’, ‘1’ – ‘10’ и т.д. Каждому обмену данными между персональным компьютером и стендом SDK-1.1 назначается отдельная строка.
    - Целочисленный делитель десятичных чисел. Диапазон значений делимого и делителя – от 0_{10} до 99_{10} включительно. Контроллеру SDK-1.1 по последовательному каналу со стороны персонального компьютера с использованием терминальной программы передаются делимое и делитель (десятичные числа), причем разделителем введенных значений является символ деления (‘/’), концом ввода является символ равенства (‘=‘), получившееся выражение отображается в терминале персонального компьютера. После чего контроллер возвращает частное, которое отображается в терминале. Каждое новое выражение начинается с новой строки. Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал и зажигание светодиодов (лабораторная работа № 1).

Лабораторная работа № 4 «Клавиатура»

Задание

Разработать и написать драйвер клавиатуры для учебно-лабораторного стенда SDK-1.1. Написать тестовую программу для разработанного драйвера, которая выполняет определенную вариантом задачу.

Матричная клавиатура

Нередко во встраиваемой технике предусмотрен ввод данных с использованием кнопок, переключателей или других контактных групп. Но простое подключение контактных групп к линиям ввода/вывода микроконтроллера может породить проблему нехватки этих самых линий, если таких контактных групп много. Решение проблемы есть – это использование матричной клавиатуры.

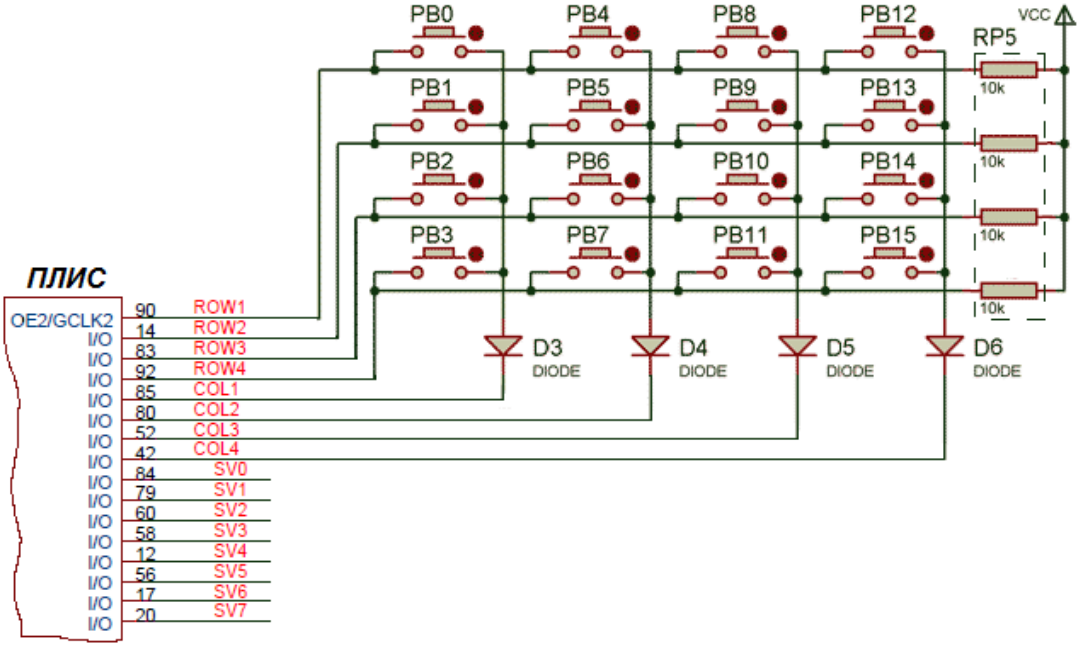


Рис. 10. Схема матричной клавиатуры контроллера SDK-1.1

Клавиатура в контроллере SDK-1.1 подключена к микроконтроллеру ADuC812 через расширитель портов ввода-вывода (ПЛИС). Схема клавиатурной матрицы представлена на рис. 10. Кнопки включены таким образом, что при нажатии кнопка замыкает строку на столбец. Из схемы видно, что часть линий ПЛИС используется в качестве сканирующих (столбцы), а часть в качестве считывающих (строки). Количество кнопок, подключенных таким образом, определяется как количество сканирующих линий, умноженное на количество считывающих. Отсюда следует, что использование матричной клавиатуры для случая, когда кнопок меньше или равно четырем, не имеет смысла, так как понадобятся те же четыре линии, а схема и программа усложнятся. Доступ к столбцам и строкам клавиатуры организован как чтение/запись регистра ПЛИС (адрес 0x080000): младшие 4 бита соответствуют 4 столбцам (COL1..COL4), старшие 4 бита – строкам (ROW1..ROW4).

Как это работает? Линии сканирующего порта (столбцы) по умолчанию находятся в состоянии, когда на всех линиях, кроме одной, установлен высокий логический уровень. Линия, на которой установлен низкий логический уровень, является опрашиваемой в текущий момент, т.е. определяет опрашиваемый столбец. Если какая-либо кнопка этого столбца будет нажата, на соответствующей линии считывающего порта (строке) также будет

низкий логический уровень, потому что замкнутая кнопка подтянет строку к потенциалу столбца, т.е. к земле. Если известен номер опрашиваемого столбца и номера линий считывающего порта, на которых установлен логический «0», можно однозначно определить, какие кнопки этого столбца нажаты.

Далее выбирается следующий опрашиваемый столбец путем установки логического «0» на соответствующей линии сканирующего порта и со считывающего порта снова снимаются данные. Цикл сканирования продолжается до тех пор, пока не будут перебраны таким образом все сканирующие линии. Традиционным решением является помещение процедуры опроса клавиатуры в обработчик прерывания от таймера. В контроллере SDK-1.1 есть еще другая возможность. В полной конфигурации ПЛИС работа с клавиатурой может быть организована по прерыванию: 6-й бит (KB) регистра ENA заведен на вход внешнего прерывания INT0 МК ADuC812. Внешнее прерывание будет возникать, как только на клавиатуре будет нажата хоть одна клавиша (любая).

Для случая, когда одновременно нажато несколько кнопок одного столбца все понятно. Будет установлено в логический «0» несколько битов считывающего порта одновременно. Но что произойдет, если будут замкнуты контакты нескольких кнопок из разных столбцов одной строки? Ведь в разных столбцах могут оказаться разные напряжения, так как на всех столбцах, кроме одного, логическая «1». Одновременное нажатие двух кнопок в одной строке привело бы к короткому замыканию и выжженным портам, если бы не диоды D3-D6 (рис. 10, рис. 11). Именно они защищают порты от короткого замыкания.

Резисторы RP5 на схеме являются подтягивающими. Если используемый микроконтроллер (в нашем случае используется ПЛИС) имеет в своем составе подтягивающие резисторы, от них можно отказаться.

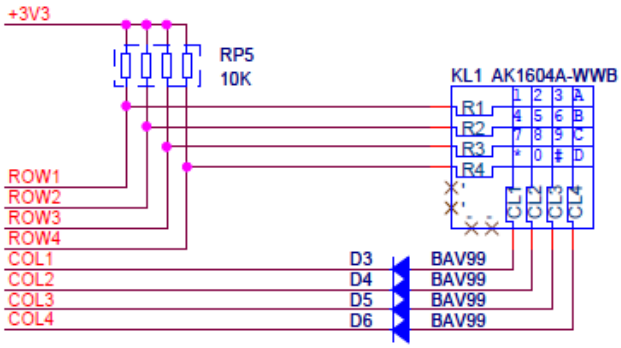


Рис. 11. Матричная клавиатура на принципиальной электрической схеме SDK-1.1

При работе с механическими кнопками возникает одна проблема – дребезг контактов. Суть его в том, что при нажатии на кнопку напряжение не сразу устанавливается на уровне 0В, а «скачет» в течение некоторого времени (десятки миллисекунд), пока цепь надежно не замкнется. После того, как кнопка будет отпущена, напряжение также «скачет», пока не установится на уровне логической «1» (рис. 12). Такое многократное замыкание/размыкание контактов вызвано тем, что контакты пружинят, обгорают и тому подобное.

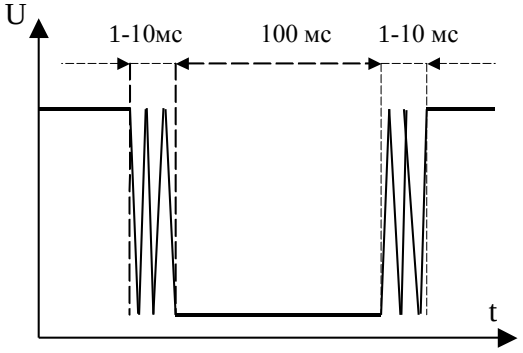


Рис. 12. Эффект дребезга контактов

Поскольку процессор обладает высоким быстродействием, то он может воспринять эти скачки напряжения за несколько нажатий. Решить эту проблему можно как аппаратно с помощью RS-триггера, триггера Шмитта, так и программно посредством небольшой задержки перед следующим опросом кнопки. Задержка подбирается такой, чтобы дребезг успел прекратиться к ее окончанию.

У приведенной схемы клавиатурной матрицы есть недостаток – эффект фантомного нажатия кнопки, который проявляется следующим образом: при нажатии определенной группы кнопок ПЛИС (микроконтроллер) считает еще несколько кнопок нажатыми, хотя никто этого не делал. На рис. 13 синими квадратами обозначены линии, на которых присутствует низкий логический уровень. Соответственно, красными квадратами обозначены линии с высоким логическим уровнем. В замкнутом состоянии находятся кнопки PB1, PB5 и PB6. В текущий момент сканируется столбец COL1, так как на выводе 85 ПЛИС присутствует низкий логический уровень, а на всех остальных сканирующих линиях высокий. В столбце COL1 в замкнутом состоянии находится только одна кнопка – PB1, и она подтягивает всю строку ROW2 к земле. Прочитав низкий логический уровень на линии ROW2, микроконтроллер через ПЛИС определит, что кнопка PB1 нажата, и это нормально. Однако в строке ROW2 есть еще одна нажатая кнопка – PB5. Она расположена в столбце COL2, а это значит, что и весь этот столбец окажется подтянутым к нулю. Сам по себе этот факт был бы не страшен, если бы в столбце COL2 не оказалась нажатой еще одна кнопка – PB6. Вот тут-то и начинаются неприятности. Эта кнопка подтянет к земле строку ROW3. Микроконтроллер через ПЛИС, прочитав со строки ROW3 низкий логический уровень, распознает кнопку PB2 нажатой (напомню, идет сканирование столбца COL1), а ее никто не нажимал.

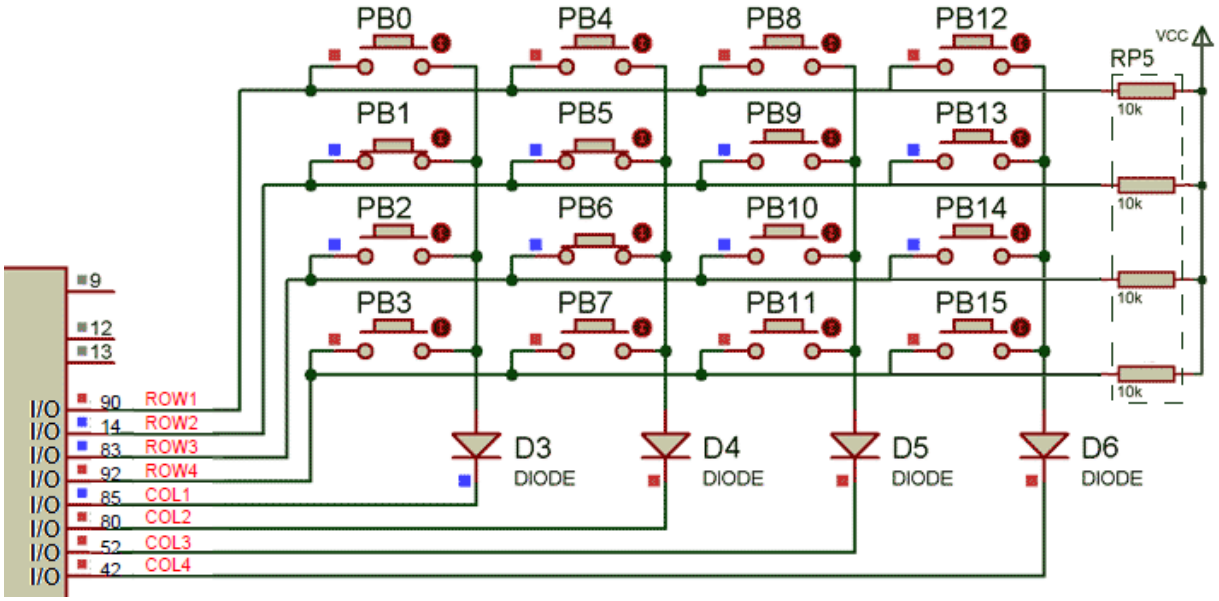


Рис. 13. Эффект фантомного нажатия кнопки

Итак, проблема проявляется при следующем условии: нажатыми должны быть минимум три кнопки, первая из которых находится в сканируемом столбце, вторая в том же ряду, что и первая, а третья в том же столбце, что и вторая.

Основным решением проблемы представляется ограничение на количество одновременно нажатых кнопок. Безопасным количеством являются две кнопки. Если нажато более двух кнопок, можно всячески предупреждать об этом пользователя (визуальная и/или звуковая сигнализация).

Второе решение проблемы вытекает из условия проявления проблемы. Ведь не любое сочетание клавиш приводит к фантомному нажатию. Можно реализовать алгоритм, проверяющий условие, при котором эта проблема проявляется, и в этом случае игнорировать

ввод. Но вот нужна ли такая клавиатура, в которой не все комбинации клавиш допустимы, это вопрос конкретной разработки.

Если у нас кнопок не просто много, а очень много, то даже матрицирование не спасает от огромного расхода линий порта микроконтроллера. Тут приходится либо жертвовать несколькими портами, либо вводить дополнительную логику, например, дешифратор с инверсным выходом. Дешифратор принимает на вход двоичный код, а на выходе выдает «1» в выбранный разряд, а у инверсного дешифратора будет «0» (рис. 14).

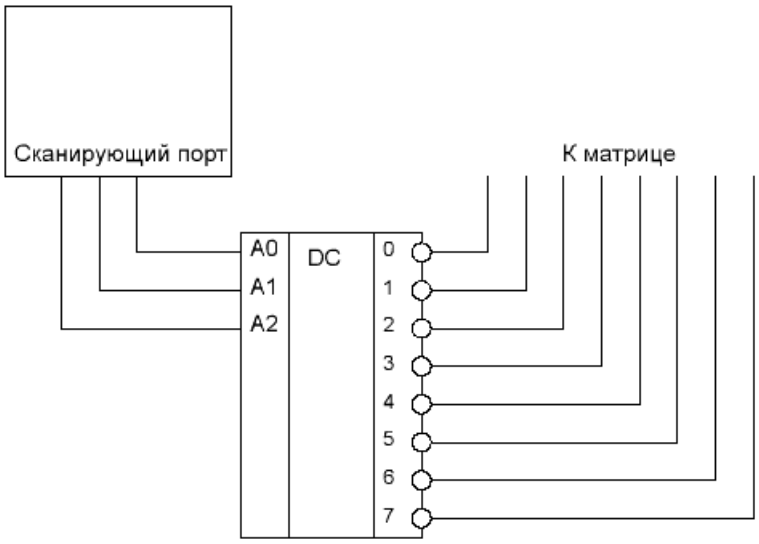


Рис. 14. Расширение разрядности клавиатурной матрицы дешифратором

Можно пойти еще дальше и поставить микросхему счетчика, который инкрементируется от импульсов с порта микроконтроллера. Значение же со счетчика прогоняется через дешифратор (рис. 15). Таким образом, можно заметно увеличить количество кнопок, только хватило бы разрядности дешифратора. Главное – учитывать на каком такте счетчика будет какой столбец.

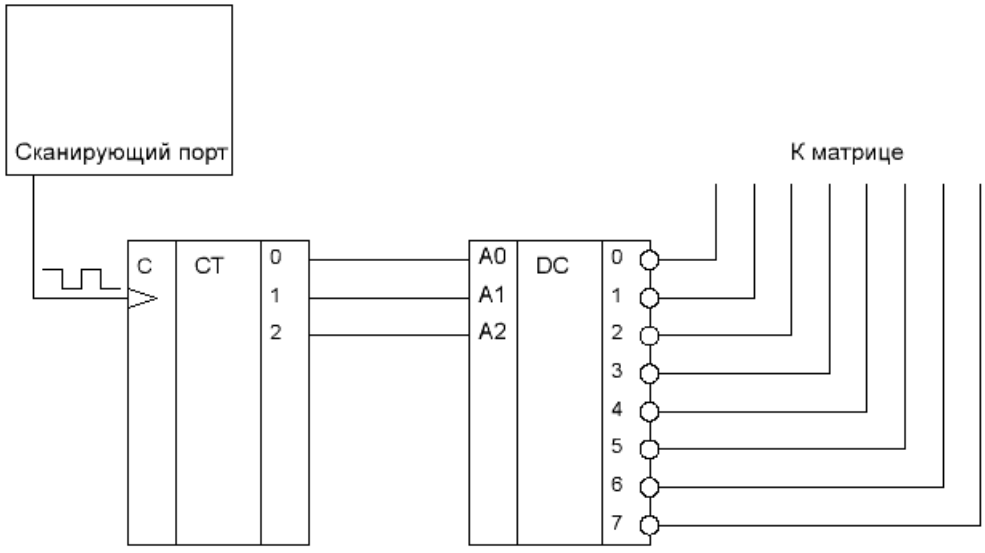


Рис. 15. Расширение разрядности клавиатурной матрицы счетчиком и дешифратором

Описание работы

Данная лабораторная работа посвящена изучению клавиатуры в составе контроллера SDK-1.1.

В рамках лабораторной работы необходимо разработать программу для контроллера SDK-1.1 (ведомый), который обменивается данными с персональным компьютером (ведущий). В качестве канала связи используется последовательный канал RS-232. На стороне персонального компьютера имеется инструментальное средство для обеспечения взаимодействия с SDK-1.1 – это терминальная программа МЗР. Вариант задания такой же, как и в лабораторной работе № 3, с некоторыми изменениями, главным из которых является замена устройства ввода данных: вместо клавиатуры персонального компьютера используется клавиатура SDK-1.1. Устройством вывода является терминал персонального компьютера и светодиодные индикаторы SDK-1.1.

Программа должна выполнять две задачи. Первая задача – это тестирование клавиатуры (написанного драйвера): символы всех нажимаемых на клавиатуре SDK-1.1 кнопок выводятся в последовательный канал и отображаются в терминале. Вторая задача – выполнение варианта задания. Переключение между двумя задачами в программе должно быть выполнено с использованием DIP-переключателей (лабораторная работа № 1).

Драйвер клавиатуры должен работать по прерыванию от таймера, т.е. сканирование должно осуществляться в обработчике прерывания от таймера. В связи с этим необходимо следить, чтобы время выполнения обработчика не превышало времени между соседними прерываниями. Иначе это приведет к повторному входу в обработчик прерывания – последствия могут быть непредсказуемыми. Поэтому рекомендуется в каждом вызове обработчика опрашивать только один столбец клавиатуры, а не все сразу, так как увеличение времени выполнения обработчика приведет к уменьшению времени выполнения основной программы: может оказаться, что процессор большую часть времени будет занят обработкой прерываний от таймера со сканированием клавиатуры.

Драйвер клавиатуры должен содержать: функцию инициализации, функции сканирования клавиатуры и обработки нажатия кнопок (вызываются в обработчике прерывания от таймера), циклический буфер клавиатуры (нажатые кнопки), API-функцию чтения символа из буфера клавиатуры (см. [14], IOS2003_lab5.pdf). Взаимодействие обработчика и API-функции осуществляется только через буфер. Кроме того, работа с клавиатурой должна быть организована с переповторами, т.е. с отслеживанием длительного нажатия кнопки (как на клавиатуре персонального компьютера). Что это значит? Если после фиксирования нажатой кнопки (соответствующий символ занесен в буфер клавиатуры) проходит время, равное задержке перед повтором символа, а она все еще нажата, то в буфер клавиатуры повторно заносится этот же символ. После этого через промежутки времени, определяемые скоростью повтора символа, код кнопки заносится в буфер до тех пор, пока не будет зафиксировано отпускание кнопки. При инициализации необходимо указать задержку перед повтором символа (первый параметр) и скорость повтора символа (второй параметр). Рекомендуемая величина задержки перед повтором – 1 секунда (не меньше).

Алгоритм опроса матричной клавиатуры рекомендуется реализовать в виде конечного автомата (Finite State Machine) – функции, которая в зависимости от своего состояния (значения определенной переменной) и входного воздействия, выполняет разную работу [2, 25, 29].

Драйвер клавиатуры должен адекватно обрабатывать одновременное нажатие нескольких кнопок.

Каждое нажатие кнопки на клавиатуре должно сопровождаться коротким звуковым сигналом, что требует использования драйвера звукового излучателя (генерация звука реализуется после попадания символа в буфер клавиатуры). Драйвер звукового излучателя должен работать по прерыванию от таймера. Длительность генерации звука – десятки миллисекунд. Таким образом, получается «музыкальная клавиатура».

Кроме того, должен быть предусмотрен контроль ввода корректных значений в рамках второй задачи программы (лабораторная работа № 3). Сигнализация в случае ввода некорректных значений – сообщение об ошибке в последовательный канал и зажигание светодиодов.

Рассмотрим применение конечных автоматов на более простом примере работы с одной клавишей без переповторов. Драйвер такой клавиши содержит:

- Функцию инициализации, в которой производится инициализация портов ввода-вывода микроконтроллера, к которым подключена клавиша; определение начальных значений буфера клавиатуры, счетчика нажатий клавиши и т.д.;
- Функции опроса клавиши и определения ее нажатия/отпускания, которые вызываются в обработчике прерывания от таймера (рис. 16, а);
- Циклический буфер клавиатуры, в котором сохраняются сообщения о том, что клавиша нажата или не нажата;
- API-функцию чтения символа из буфера клавиатуры.

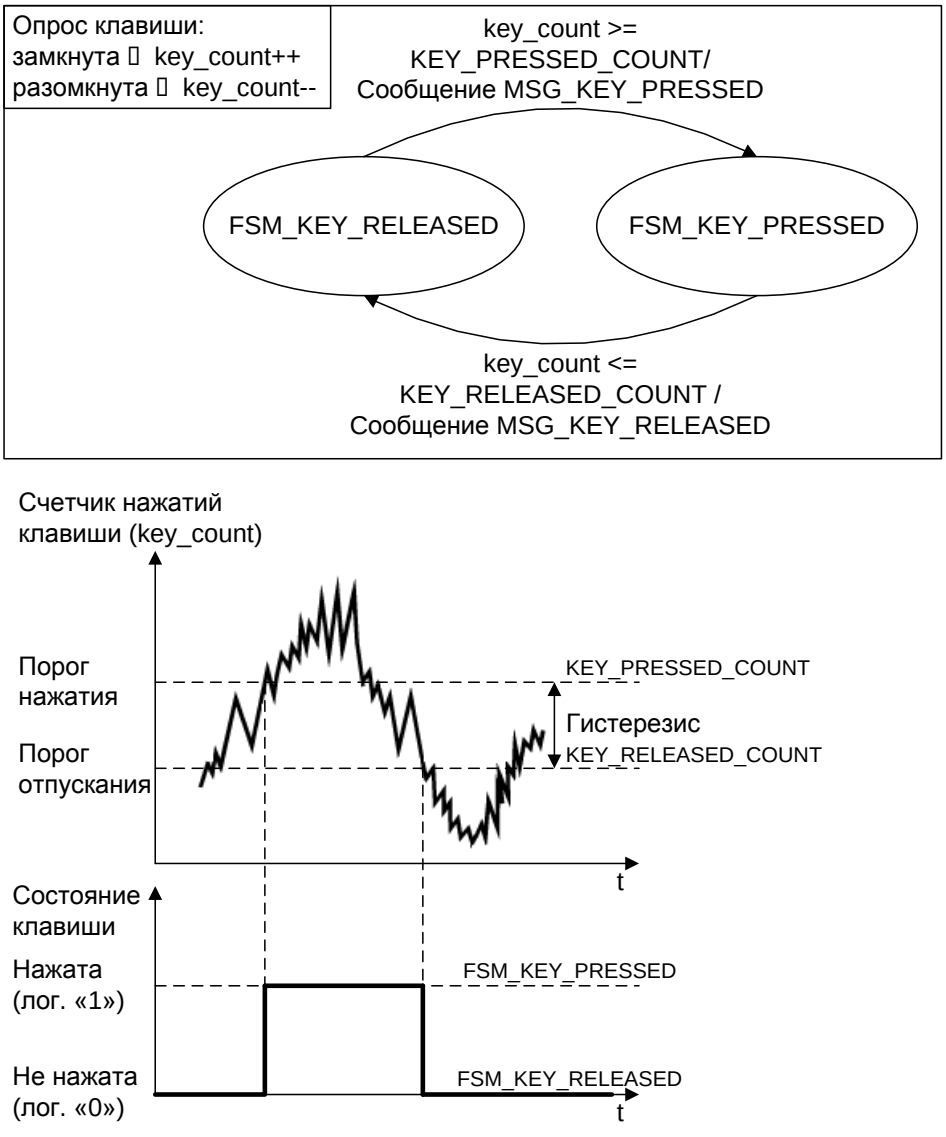


Рис. 16. Граф конечного автомата определения: нажата или отпущена клавиша? (а), временная диаграмма работы автомата (б)

Клавиша может находиться только в двух состояниях (рис. 16, а): клавиша нажата (FSM_KEY_PRESSED) и клавиша не нажата (FSM_KEY_RELEASED). Сначала в обработчике прерывания от таймера выполняется опрос клавиши на предмет ее

замкнутости/разомкнутости. В процессе этого опроса счетчик нажатий клавиши key_count инкрементируется или декрементируется соответственно. Далее в виде конечного автомата реализуется алгоритм определения нажатия/отпускания клавиши на основе значения счетчика нажатий клавиши: если значение key_count превышает порог нажатия клавиши KEY_PRESSED_COUNT, то фиксируется ее нажатие (записывается сообщение MSG_KEY_PRESSED в буфер клавиатуры); если значение key_count падает ниже порога отпускания клавиши KEY_RELEASED_COUNT, то фиксируется ее отпускание (записывается сообщение MSG_KEY_RELEASED в буфер клавиатуры). Таким образом используется свойство гистерезиса: любой шум (дребезг), любые помехи с амплитудой, меньшей величины (KEY_PRESSED_COUNT − KEY_RELEASED_COUNT), отсекаются, а состояние нажатой клавиши резко отличается от состояния отпущенной клавиши в плане определения и фиксации (рис. 16, б).

Данный принцип организации работы с одной клавишей можно перенести на набор клавиш, и в том числе, матричную клавиатуру. Особенностью такой схемы работы является то, что пороговые значения счетчика нажатий клавиши (KEY_PRESSED_COUNT и KEY_RELEASED_COUNT) прямо пропорциональны частоте прерываний от таймера, поэтому могут легко настраиваться при повторном использовании драйвера клавиатуры в других программных продуктах. Кроме того, данная схема абсолютно симметрично обрабатывает дребезг при нажатии и отпускании клавиши, а свойство гистерезиса позволяет регулировать чувствительность клавиатуры. Если ввести новые пороговые значения счетчика нажатий клавиши key_count, то можно организовать работу с клавиатурой с переповторами.

Требования к выполнению работы

1. Разрабатываемые драйверы устройств должны быть выполнены в виде отдельных программных модулей (файлов), содержащих функции по работе с заданным одним устройством.
2. Переключение между двумя задачами в тестовой программе должно быть выполнено с использованием DIP-переключателей.
3. В тестовой программе должна быть продемонстрирована работа с клавиатурой и последовательным интерфейсом по прерыванию.
4. Работа с клавиатурой должна быть организована с переповторами, т.е. с отслеживанием длительного нажатия кнопки (как на клавиатуре персонального компьютера).
5. Драйвер клавиатуры должен адекватно обрабатывать одновременное нажатие нескольких кнопок.
6. Должен быть предусмотрен контроль ввода корректных значений в рамках выполнения прикладной задачи.
7. В программе должны быть использованы механизмы взаимного исключения (см. [14], IOS2003_lab4.pdf).
8. Текст программы должен соответствовать правилам оформления программ на языке Си, приведенным в приложении (Приложение Б. Требования к оформлению программ на языке Си, [28]).

Содержание отчета

1. Титульный лист.
2. Номер варианта, задание.
3. Иллюстрация организации и функционирования разработанного программного обеспечения (драйверы, тестовая программа) в виде блок-схемы, диаграммы процессов, потоков данных, диаграммы состояний автоматов и других способов структурного и поведенческого описания программы (по выбору студента).

4. Исходный текст программы с комментариями.
5. Основные результаты.

Литература

Литература к лабораторной работе: [4], [13], [14], [16], [17], [18], [21], [25], [26], [29], [30], [32], [33], [41].

Варианты заданий

В случае установки на DIP-переключателях заданной комбинации (определяется студентом) контроллер SDK-1.1 входит в режим тестирования клавиатуры. В остальных случаях выполняется задача в соответствии с вариантом задания.

1. Скорость последовательного канала – 9600 бит/с.

Конвертор из десятичной в двоичную систему счисления. Диапазон преобразуемых значений – от 0_{10} до 255_{10} включительно. 8-разрядная сетка для отображения двоичных чисел. При помощи клавиатуры SDK-1.1 вводится десятичное число для конвертирования. Кнопка перевода в двоичную систему счисления – «*». Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое преобразование начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

2. Скорость последовательного канала – 9600 бит/с.

Сумматор десятичных чисел. Диапазон значений слагаемых – от 0_{10} до 99_{10} включительно. При помощи клавиатуры SDK-1.1 вводятся два слагаемых (десятичные числа), причем разделителем слагаемых является символ «+» (кнопка «A» на клавиатуре), символом начала вычисления – «=» (кнопка «#»). Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое выражение начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

3. Скорость последовательного канала – 4800 бит/с.

Конвертор из шестнадцатеричной в двоичную систему счисления. Диапазон преобразуемых значений – от 0_{16} до FF_{16} включительно. 8-разрядная сетка для отображения двоичных чисел. При помощи клавиатуры SDK-1.1 вводится шестнадцатеричное число для конвертирования. Кнопка перевода в двоичную систему счисления – «#». Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое преобразование начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

4. Скорость последовательного канала – 4800 бит/с.

Вычитатель десятичных чисел. Диапазон значений уменьшаемого и вычитаемого – от 0_{10} до 99_{10} включительно. Разность может быть как положительной, так и отрицательной. При помощи клавиатуры SDK-1.1 вводятся уменьшаемое и вычитаемое (десятичные числа), причем разделителем введенных значений является символ «-» (кнопка «B» на клавиатуре), символом начала вычисления – «=» (кнопка «#»). Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое выражение начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

5. Скорость последовательного канала – 1200 бит/с.

Конвертор из десятичной в шестнадцатеричную систему счисления. Диапазон преобразуемых значений – от 0_{10} до 255_{10} включительно. При помощи клавиатуры SDK-1.1 вводится десятичное число для конвертирования. Кнопка перевода в шестнадцатеричную систему счисления – «*». Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое преобразование начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

6. Скорость последовательного канала – 1200 бит/с.

Умножитель десятичных чисел. Диапазон значений множителей – от 0_{10} до 99_{10} включительно. При помощи клавиатуры SDK-1.1 вводятся множители (десятичные числа), причем разделителем введенных значений является символ «*» (кнопка «C» на клавиатуре), символом начала вычисления – «=» (кнопка «#»). Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое выражение начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

7. Скорость последовательного канала – 2400 бит/с.

Конвертор из шестнадцатеричной в десятичную систему счисления. Диапазон преобразуемых значений – от 0_{16} до FF_{16} включительно. При помощи клавиатуры SDK-1.1 вводится шестнадцатеричное число для конвертирования. Кнопка перевода в десятичную систему счисления – «#». Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое преобразование начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

8. Скорость последовательного канала – 2400 бит/с.

Целочисленный делитель десятичных чисел. Диапазон значений делимого и делителя – от 0_{10} до 99_{10} включительно. При помощи клавиатуры SDK-1.1 вводятся делимое и делитель (десятичные числа), причем разделителем введенных значений является символ «/» (кнопка «D» на клавиатуре), символом начала вычисления – «=» (кнопка «#»). Вводимые с клавиатуры числа и результат должны выводиться в последовательный канал и отображаться в терминале персонального компьютера. Каждое новое выражение начинается с новой строки. Должен быть предусмотрен контроль ввода корректных значений.

Лабораторная работа № 5

«Жидкокристаллический индикатор»

Задание

Разработать и написать драйвер жидкокристаллического индикатора (ЖКИ) для учебно-лабораторного стенда SDK-1.1. Написать программу для разработанного драйвера, которая выполняет определенную вариантом прикладную задачу.

Описание работы

ЖКИ в контроллере SDK-1.1 подключен не напрямую к микроконтроллеру ADuC812, а через расширитель портов ввода-вывода, выполненный на базе ПЛИС. В ЖКИ есть специальный контроллер, формирующий необходимые напряжения на входах матрицы и осуществляющий динамическую индикацию. Для работы с этим контроллером реализован простейший интерфейс, описанный ниже (рис. 17, табл. 4). Матрица имеет 80 входов по горизонтали и 16 входов по вертикали[3].

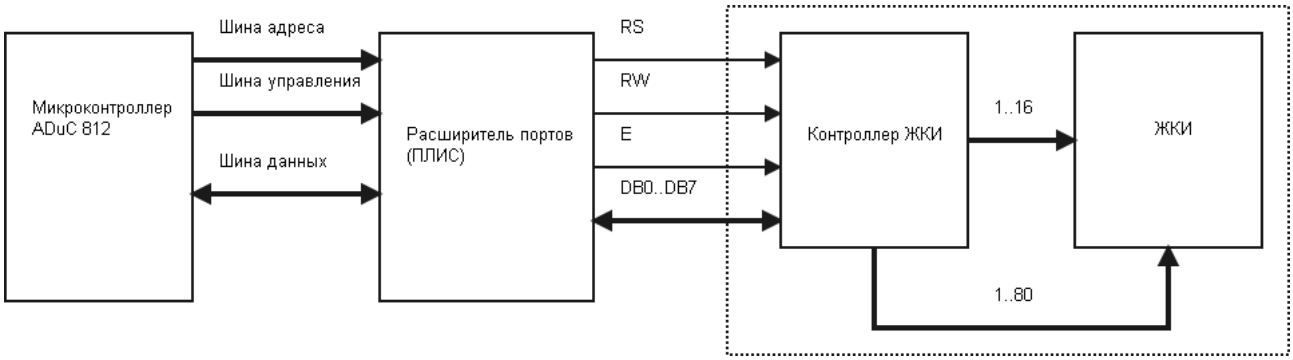


Рис. 17. Схема подключения контроллера ЖКИ к МК ADuC812

Матрица ЖКИ состоит из 32 знакомест (2 строки по 16 символов) размером 5 точек по горизонтали и 8 точек по вертикали. Для отображения различных символов внутри контроллера ЖКИ есть знакогенератор[4].

Таблица 4. Интерфейс ЖКИ

Обозначение	Описание
RS	Переключение между регистрами команд и данных: 1 – данные, 0 – команды
R/W	1– чтение (из контроллера ЖКИ), 0 – запись (в контроллер ЖКИ)
E	Разрешающий сигнал (1 – активный уровень). Если сигнал E = 0, то контроллер ЖКИ игнорирует все остальные сигналы
DB0	Бит данных 0
DB1	Бит данных 1
DB2	Бит данных 2
DB3	Бит данных 3
DB4	Бит данных 4
DB5	Бит данных 5

[3] Чтобы уменьшить количество проводников используется динамическая индикация. Принцип работы динамической индикации аналогичен принципу, используемому при опросе клавиатуры учебного стенда.

[4] Знакогенератор – специальное устройство, содержащее в себе ПЗУ (или ОЗУ) с битовыми картами с изображениями различных символов. Каждому изображению символа ставится в соответствие его код.

DB6	Бит данных 6
DB7	Бит данных 7

Основными компонентами контроллера ЖКИ являются память DDRAM (Data Display RAM), память CGRAM (Character Generator RAM), память CGROM (Character Generator ROM), счетчик адреса, регистр команд IR (Instruction Register), регистр данных DR (Data Register). Регистр команд предназначен для записи в него таких операций, как очистка дисплея, перемещение курсора, включение/выключение дисплея, а также установка адреса памяти DDRAM и CGRAM для последующего их выполнения. Регистр данных временно хранит данные, предназначенные для записи или чтения из DDRAM или CGRAM (символы). Эти два регистра можно выбрать с помощью регистрового переключателя RS (Register Select).

Работать с ЖКИ достаточно просто. За связь с ЖКИ в расширителе портов ввода-вывода отвечают два регистра:

1. DATA_IND (адрес 0x080001) отвечает за выдачу информации на шину данных (через этот регистр можно передавать команды контроллеру и данные).
2. C_IND (адрес 0x080006) отвечает за формирование сигналов E, R/W и RS, позволяющих регулировать обмен на шине между расширителем портов и контроллером ЖКИ.

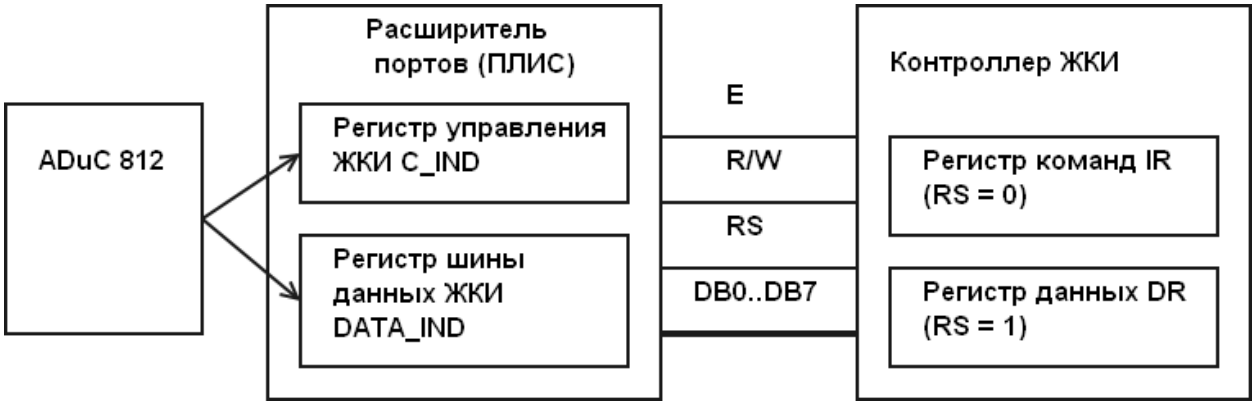


Рис. 18. Регистры, необходимые для работы с ЖКИ

На рис. 18 регистры расширителя портов изображены слева, регистры контроллера ЖКИ справа. Для доступа к регистрам контроллера ЖКИ вы должны сформировать на шине сигналы с помощью регистров расширителя портов.

Вся работа с индикатором сводится к нескольким простым вещам:

1. Первым шагом вы записываете команду или данные (коды выводимых символов) в регистр DATA_IND расширителя портов. После этого, содержимое этого регистра появляется на шине данных контроллера ЖКИ (DB0-DB7). Контроллер на эти данные, естественно, не реагирует, так как сигнал E (Enable) нами еще не выставлен в активный уровень (логическая «1»).
2. Вторым шагом вы должны разрешить работу с шиной с помощью сигнала E (логическая «1»), выставить сигнал записи (логический «0» на линии W) и указать тип регистра, с которым вы будете работать в контроллере ЖКИ на линии RS. Если вы передаёте данные, то на сигнал RS нужно подать «1», если команду, то «0».

Необходимо помнить, что в учебном стенде SDK-1.1 начальная инициализация контроллера ЖКИ уже выполнена в загрузчике. В реальной системе вам придется программировать ее самостоятельно.

Время выполнения команд контроллером ЖКИ не равно нулю, и это нужно учитывать в своем драйвере: опрашивать флаг готовности ЖКИ (BF). Таким образом, функции работы с

ЖКИ НЕ нужно вызывать в обработчиках прерывания других периферийных устройств (например, таймера).

Данная лабораторная работа посвящена изучению жидкокристаллического индикатора (ЖКИ) стенда SDK-1.1.

В рамках лабораторной работы необходимо разработать программу для контроллера SDK-1.1, которая выполняет конкретную прикладную задачу (см. варианты задания). Реализация задачи требует знания материалов предыдущих лабораторных работ: таймеры микроконтроллера ADuC812, последовательный канал, светодиодные индикаторы, клавиатура и др.

Драйвер ЖКИ должен включать следующие функции:

Функция	Описание
`void InitLCD(void)`	Инициализация ЖКИ.
`void WriteControlLCD( unsigned char ch)`	Запись значения в регистр управления ЖКИ C_IND (ПЛИС): `ch` – значение, записываемое в C_IND.
`bit ReadBFLCD(void)`	Чтение флага BF (флаг занятости контроллера ЖКИ).
`unsigned char ClearLCD(void)`	Очистка дисплея с возвратом результата выполнения операции.
`unsigned char GotoXYLCD (unsigned char x, bit y)`	Переход в заданную позицию дисплея с возвратом результата выполнения операции: `x, y` – координаты позиции.
`unsigned char PrintCharLCD (unsigned char symbol)`	Вывод символа на дисплей с возвратом результата выполнения операции: `symbol` – выводимый символ.

Кроме того, может быть реализована функция вывода строки на ЖКИ, функция дополнительной настройки ЖКИ (отображение, мерцание курсора).

Драйвер клавиатуры использует прерывание таймера, в котором производится опрос состояния кнопок (сканирование клавиатуры). В данном случае можно применить автоматное программирование. В драйвер клавиатуры рекомендуется включить функцию чтения нажатых кнопок из буфера, связывающего ее с обработчиком прерывания таймера. Реакции на нажатия кнопок клавиатуры должны формироваться в главной программе. Вся обработка нажатий кнопок не должна быть локализована в обработчике прерываний таймера.

Драйвер таймера должен включать следующие функции (помимо обработчика прерывания):

Функция	Описание
`void InitTimer(void)`	Инициализация таймера.
`unsigned long GetMsCounter(void)`	Получение текущей метки времени в миллисекундах.
`unsigned long DTimeMs(unsigned long t0)`	Измерение количества миллисекунд, прошедших с временной метки `t0` и до текущего времени.
`void DelayMs(unsigned long t)`	Задержка на `t` миллисекунд.

Кроме того, могут быть реализованы функции работы с таймером в режиме «счетчик» (например, чтение счетчика).

Работа с последовательным каналом (приемопередатчиком UART) должна быть организована асинхронно по прерыванию. Драйвер последовательного канала включает следующие функции (помимо обработчика прерывания):

Функция	Описание
`void InitSerial(void)`	Инициализация последовательного канала.
`unsigned char WriteSerial(unsigned char data_buf)`	Передача байта данных `data_buf` с возвратом результата выполнения операции.
`unsigned char ReadSerial(unsigned char* data_buf)`	Прием байта данных `*data_buf` с возвратом результата выполнения операции.
`unsigned char StatusSerial(void)`	Чтение признака наличия байта в буфере приема.

Кроме того, может быть реализована функция вывода строки в последовательный канал.

Требования к выполнению работы

1. Разрабатываемые драйверы устройств должны быть выполнены в виде отдельных программных модулей (файлов), содержащих функции по работе с заданным одним устройством.
2. На уровне драйверов (особенно обработчиков прерываний) НЕ рекомендуется смешивать работу с несколькими периферийными устройствами (например, в обработчике прерывания таймера выводить строку на ЖКИ, опрашивать DIP-переключатели и т.д.). Взаимодействие устройств ввода-вывода следует организовать на прикладном уровне с использованием API-функций их драйверов.
3. Должен быть предусмотрен контроль ввода корректных значений в рамках выполнения прикладной задачи.
4. В программе должны быть использованы механизмы взаимного исключения (см. [14], IOS2003_lab4.pdf).
5. Текст программы должен соответствовать правилам оформления программ на языке Си, приведенным в приложении (Приложение Б. Требования к оформлению программ на языке Си, [28]).

Содержание отчета

1. Титульный лист.
2. Номер варианта, задание.
3. Модель написанной программы (см. Приложение А. Проектирование и разработка программы).
4. Разработанные протоколы, форматы данных и др.
5. Исходный текст программы с комментариями (можно не весь, но обязательно главная программа и полностью драйвер периферийного устройства, изучению которого была посвящена лабораторная работа).
6. Основные результаты.

Литература

Литература к лабораторной работе: [13], [14], [16], [21], [22], [24], [26], [33], [41], [45].

Варианты заданий

1. Секундомер.

Написать программу, реализующую функции электронного секундомера. Точность измерения времени – сотые доли секунды. В качестве устройства, измеряющего время, следует использовать один из внутренних таймеров микроконтроллера ADuC812. Управление секундомером должно осуществляться с клавиатуры стенда SDK-1.1:

- кнопка «*» («старт/пауза») – запускает процесс измерения времени либо приостанавливает его, не сбрасывая;
- кнопка «#» («сброс») – сбрасывает замеряемое время в ноль.

На ЖКИ должна отображаться четко следующая информация: замеряемое время (слева в верхней строке); минимальное из всех замеренных времен (слева в нижней строке); максимальное из всех замеренных времен (справа в нижней строке). Формат отображения времени: «SS:CC», где SS – секунды, CC – сотые доли секунды. После переполнения секундомер начинает отсчет с нуля, т.е. 99,99с → 0с. Замеренный интервал времени по нажатию кнопки «пауза» должен выводиться в последовательный канал в формате, описанном ранее. Каждый интервал с новой строки. Сообщения о сбросе и переполнении секундомера тоже должны выводиться в последовательный канал (формат этих сообщений определяется студентом).

В рамках задания необходимо реализовать:

- драйвер таймера;
- драйвер последовательного канала;
- драйвер клавиатуры;
- драйвер ЖКИ.

2. Калькулятор.

Написать программу, реализующую функции простого калькулятора. Управление калькулятором должно осуществляться с клавиатуры стенда SDK-1.1:

- кнопки «0 – 9» – ввод операндов;
- кнопка «A» – операция сложения;
- кнопка «B» – операция вычитания;
- кнопка «C» – операция умножения;
- кнопка «D» – операция деления;
- кнопка «*» – начало процесса вычисления;
- кнопка «#» – сброс. На ЖКИ должна выводиться следующая информация: вычисляемое выражение (например, «`25+75`», затем, после нажатия кнопки «*» – «`25+75=100`»).

Разрядная сетка операндов и результатов определяется студентом, однако должна содержать не меньше 2 десятичных разрядов и обладать возможностью легкого увеличения/уменьшения. После нажатия кнопки сброса, а также в начале работы калькулятора на индикаторе должен отображаться ноль. Каждое вычисленное выражение (по нажатию кнопки «*») должно выводиться в последовательный канал в формате, описанном ранее, и начинаться с новой строки.

В рамках задания необходимо реализовать:

- драйвер последовательного канала;
- драйвер клавиатуры;
- драйвер ЖКИ.

3. Текстовый редактор.

Написать программу, реализующую функции простого текстового редактора. Ввод текста должен осуществляться с клавиатуры стенда SDK-1.1 кнопками «0 – 9». Таким образом, текст может состоять только из цифровых символов. Объем вводимого текста не должен превышать 8 строк по 16 символов, при этом ЖКИ отображает только две соседние строки. Номер текущей строки должен отображаться на светодиодном индикаторе (первая строка – первый светодиод, вторая строка – второй светодиод и т.д.). Перемещение по тексту осуществляется с помощью курсора, управляемого клавишами «A» (влево), «B» (вправо), «C» (вверх) и «D» (вниз). Ввод текста производится в позицию, указываемую курсором, при этом все символы находящиеся правее введенного символа сдвигаются, курсор перемещается (как в обычном текстовом редакторе). В качестве клавиши перевода строки выступает кнопка «*», удаление символов должно производиться кнопкой «#».

В рамках задания необходимо реализовать:

- драйвер светодиодных индикаторов;
- драйвер клавиатуры;
- драйвер ЖКИ.

4. Бегущая строка.

Написать программу, реализующую эффект «бегущей строки». Эффект заключается в постепенном смещении отображаемой на ЖКИ строки по кругу (вправо или влево). Управление «бегущей строкой» должно осуществляться с клавиатуры стенда SDK-1.1:

- кнопка «A» – смена направления движения текста;
- кнопка «B» – смена строки ЖКИ, по которой двигается («бежит») текст, причем смена должна производиться немедленно;
- кнопки «C» и «D» – изменяют скорость движения текста (диапазон значений: 1 – 10 позиций в секунду).

Для формирования скорости движения строки обязательным является использование таймера микроконтроллера ADuC812. Движение текстовой строки реализуется таким образом, чтобы исчезающая из области видимости часть текста появлялась постепенно с противоположной стороны строки ЖКИ. Кроме того, отображение «бегущей строки» должно быть плавным, четким, без образованных движением текста «хвостов».

По нажатию кнопки «*» программа переходит в режим ввода новой текстовой строки. Ввод производится с помощью кнопок «0 – 9». Таким образом, вводимый текст может состоять только из цифровых символов. При вводе должна контролироваться длина текста (не менее 1 и не более 16 символов). Завершение ввода строки производится кнопкой «*». Отмена ввода – кнопка «#». Исполнение программы должно начинаться с прокручивания в верхней строке ЖКИ слева направо текста «SDK-1.1».

В рамках задания необходимо реализовать:

- драйвер таймера;
- драйвер клавиатуры;
- драйвер ЖКИ.

5. Монитор состояний устройств.

Написать программу, реализующую монитор состояний устройств стенда SDK-1.1. Программа должна отражать состояние следующих трех устройств:

- DIP-переключатели (линии 0-7 дискретного параллельного порта ПЛИС): отображается состояние DIP-переключателей в двоичной системе счисления;
- таймер-счетчик: отображается количество перепадов на счетном входе T0 или T1, вызываемых замыканием соответствующих DIP-переключателей (см. рис. 5);
- таймер: отображается системное время, прошедшее с момента старта программы в миллисекундах.

Программа должна работать в двух режимах: автоматическом и ручном. Смена режима работы должна производиться нажатием кнопки «*» на клавиатуре стенда SDK-1.1. В автоматическом режиме отображение состояний устройств производится циклически с интервалом 3 секунды по умолчанию. Данный интервал должен изменяться с помощью кнопок «C» и «D». На ЖКИ состояние каждого устройства должно выглядеть следующим образом: в верхней строке – название устройства, в нижней строке – текущее состояние устройства.

В ручном режиме смена отображения состояния устройств должна производиться нажатием кнопки «#».

Кроме того, каждые 2 секунды (может быть другой приемлемый интервал времени) в последовательный канал должно выводиться текущее состояние всех устройств в виде одной строки, формат строки – свободный.

В рамках задания необходимо реализовать:

- драйвер DIP-переключателей;
- драйвер таймера/счетчика;
- драйвер последовательного канала;
- драйвер клавиатуры (может быть реализован без переповторов, т.е. по принципу «кнопка нажата или нет»);
- драйвер ЖКИ.

Лабораторная работа № 6

«Последовательный интерфейс I²C»

Задание

Разработать и написать драйверы интерфейса I²C и I²C-устройств учебно-лабораторного стенда SDK-1.1. Написать программу для разработанных драйверов, которая выполняет определенную вариантом прикладную задачу.

Описание работы

В бытовой технике, телекоммуникационном оборудовании и промышленной электронике часто встречаются похожие решения в, казалось бы, никак не связанных изделиях. Например, практически каждая система включает в себя:

- некоторый «умный» узел управления, обычно однокристальная микроЭВМ;
- узлы общего назначения, такие как буферы ЖКИ, порты ввода-вывода, RAM, E²PROM или преобразователи данных;
- специфические узлы, такие как схемы цифровой настройки и обработки сигнала для радио- и видеосистем, или генераторы тонального набора для телефонии.

Для того чтобы использовать эти общие решения с выгодой для конструкторов и производителей (технологов), а также увеличить эффективность аппаратуры и упростить схемотехнические решения, компания Philips в 1980 году разработала простую двунаправленную двухпроводную шину для эффективного «межмикросхемного» (inter-IC) управления. Шина так и называется – Inter-Integrated Circuit, или IIC (I²C) шина. В настоящее время ассортимент продукции Philips включает более 150 КМОП и биполярных I²C-совместимых устройств, функционально предназначенных для работы во всех трех вышеперечисленных категориях электронного оборудования. Все I²C-совместимые устройства имеют встроенный интерфейс, который позволяет им связываться друг с другом по шине I²C. Это конструкторское решение разрешает множество проблем сопряжения различных устройств, которые обычно возникают при разработке цифровых систем.

Основной режим работы шины I²C – 100 кбит/с; 10 кбит/с в режиме работы с пониженной скоростью. Заметим, что стандарт допускает тактирование с частотой вплоть до нулевой. Для адресации I²C-устройств используется 7 бит (1980 год).

Список возможных применений I²C:

- доступ к модулям памяти (RAM, E²PROM, FLASH и др.);
- доступ к низкоскоростным ЦАП/АЦП;
- работа с часами реального времени (RTC);
- регулировка контрастности, насыщенности и цветового баланса мониторов;
- управление интеллектуальными звукоизлучателями (динамиками);
- управление ЖКИ, в том числе в мобильных телефонах;
- чтение информации с датчиков мониторинга и диагностики оборудования, например, термостат центрального процессора или датчик скорости вращения вентилятора охлаждения процессора;
- информационный обмен между микроконтроллерами.

I²C использует две двунаправленные линии с открытым стоком: последовательная линия данных (SDA, англ. Serial DAta) и последовательная линия тактирования (SCL, англ. Serial CLock), обе нагруженные резисторами (см. рис. 19). Максимальное напряжение +5В, часто используется +3,3В, однако допускаются и другие напряжения (не менее +2В). Шина

I²C поддерживает любую технологию изготовления микросхем (НМОП, КМОП, биполярную).

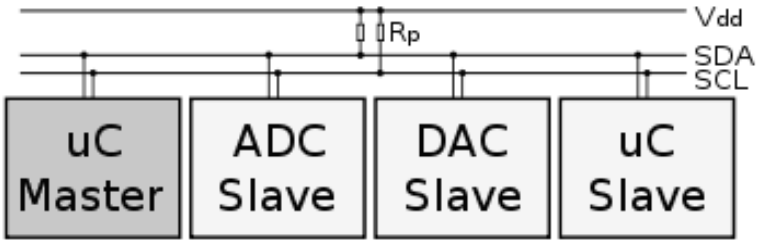


Рис. 19. Пример соединения устройств на шине I²C

Каждое устройство распознается по уникальному адресу, будь то микроконтроллер, ЖКИ-буфер, память или интерфейс клавиатуры, и может работать как передатчик или приёмник, в зависимости от назначения устройства. Обычно ЖКИ-буфер – только приёмник, а память может как принимать, так и передавать данные. Кроме того, устройства могут быть классифицированы как ведущие и ведомые при передаче данных. Ведущий – это устройство, которое инициирует передачу данных и вырабатывает сигналы синхронизации. При этом любое адресуемое устройство считается ведомым по отношению к ведущему. Классическая адресация включает 7-битное адресное пространство с 16 зарезервированными адресами (шина I²C 1980 года). Это означает до 112 свободных адресов для подключения периферии на одну шину.

Возможность подключения более одного микроконтроллера к шине означает, что более чем один ведущий может попытаться начать пересылку в один и тот же момент времени. Для устранения хаоса, который может возникнуть в данном случае, разработана процедура арбитража. Эта процедура основана на том, что все I²C-устройства подключаются к шине по правилу монтажного И.

Генерация синхросигнала – это всегда обязанность ведущего: каждый ведущий генерирует свой собственный сигнал синхронизации при пересылке данных по шине. Сигнал синхронизации может быть изменен, только если он «вытягивается» медленным ведомым устройством (путем удержания линии в низком состоянии), или другим ведущим в случае столкновения.

Данная лабораторная работа посвящена изучению последовательного интерфейса I^2C и устройств, подключенных по этому интерфейсу к микроконтроллеру ADuC812 стенда SDK-1.1, – энергонезависимая память EEPROM и часы реального времени (RTC).

В рамках лабораторной работы необходимо разработать программу для контроллера SDK-1.1, которая выполняет конкретную прикладную задачу (см. варианты задания). Реализация задачи требует знания материалов предыдущих лабораторных работ: таймеры микроконтроллера ADuC812, последовательный канал, светодиодные индикаторы, клавиатура, ЖКИ, звуковой излучатель и др.

В данной работе МК ADuC812 – ведущий, а EEPROM и RTC – ведомые на шине I^2C. К особенностям реализации контроллера последовательного двухпроводного интерфейса в ADuC812 можно отнести следующие: в режиме ведущего генерация сигналов на линиях данных и синхронизации является программной (через биты регистров специального назначения); в режиме ведущего (в отличие от ведомого) НЕ генерируются прерывания по приему/передаче данных. С учетом сказанного ниже приведены варианты реализации драйвера I^2C:

1. Простой вариант. В драйвере I^2C реализуется синхронный обмен данными, который предполагает отсутствие ситуации неготовности I^2C-устройств. Однако EEPROM и RTC не порты ввода-вывода и все-таки требуют определенное количество времени на выполнение циклов чтения/записи/стирания данных. При

синхронной организации обмена эти задержки должна учитывать программа, т.е. драйвер I²C.

2. Усложненный вариант. Драйвер I²C должен использовать прерывание таймера, в котором работа с интерфейсом представляет собой периодический процесс генерации сигналов на линиях данных (SDA) и синхронизации (SCL) и организуется в виде конечного автомата. Состояниями данного автомата являются состояния шины I²C по передаче и приему данных (например, передача старт- или стоп-состояния, передача адреса ведомого устройства, передача или прием байта данных, передача или прием подтверждения и др.). Кроме того, необходимо отслеживать ошибочные состояния I²C и обрабатывать их. Частота настройки таймера определяет невысокую скорость передачи данных по шине I²C. Вся работа с интерфейсом НЕ должна быть локализована в обработчике прерываний таймера, так как процессом обмена данными необходимо управлять при помощи API-функций, которые взаимодействуют с обработчиком прерывания через буферы.

Независимо от варианта реализации драйвера I²C он должен содержать такие API-функции, как инициализация I²C, прием и передача блока данных и т.д. Взаимодействие с I²C-устройствами (EEPROM, RTC) при помощи такого драйвера может быть выполнено по опросу (проверка готовности ведомых устройств к обмену).

Драйвер EEPROM должен включать следующие функции:

Функция	Описание
`unsigned char ReadEEPROM (unsigned long addr, unsigned long size, unsigned char *buf)`	Чтение данных из EEPROM с возвратом результата выполнения операции: `addr` – адрес ячейки памяти, `size` – размер буфера для чтения, `buf` – буфер.
`unsigned char WriteEEPROM (unsigned long addr, unsigned long size, unsigned char *buf)`	Запись данных в EEPROM с возвратом результата выполнения операции: `addr` – адрес ячейки памяти, `size` – размер буфера записи, `buf` – буфер.

Драйвер часов реального времени должен включать следующие функции:

Функция	Описание
`void InitRTC(void)`	Инициализация часов реального времени.
`unsigned char ReadRTC (TimeDate *td)`	Чтение даты и времени из RTC с возвратом результата выполнения операции: `td` – буфер для даты и/или времени в виде структуры специального формата.
`unsigned char WriteRTC (TimeDate *td)`	Запись даты и времени в RTC с возвратом результата выполнения операции: `td` – буфер для даты и/или времени в виде структуры специального формата.

Кроме того, в драйвер могут входить функции для работы с будильником и для дополнительной настройки RTC.

Драйвер клавиатуры тоже использует прерывание таймера, в котором производится опрос состояния кнопок (лабораторная работа № 4 «Клавиатура»). В данном случае можно применить автоматное программирование. В драйвер клавиатуры рекомендуется включить функцию чтения нажатых кнопок из буфера, связывающего ее с обработчиком прерывания.

Реакции на нажатия кнопок клавиатуры должны формироваться в главной программе. Вся обработка нажатий кнопок НЕ должна быть локализована в обработчике прерываний таймера.

Драйвер ЖКИ должен включать следующие функции (лабораторная работа № 5):

Функция	Описание
`void InitLCD(void)`	Инициализация ЖКИ.
`void WriteControlLCD( unsigned char ch)`	Запись значения в регистр управления ЖКИ C_IND (ПЛИС): `ch` – значение, записываемое в C_IND.
`bit ReadBFLCD(void)`	Чтение флага BF (флаг занятости контроллера ЖКИ).
`unsigned char ClearLCD(void)`	Очистка дисплея с возвратом результата выполнения операции.
`unsigned char GotoXYLCD (unsigned char x, bit y)`	Переход в заданную позицию дисплея с возвратом результата выполнения операции: `x, y` – координаты позиции.
`unsigned char PrintCharLCD (unsigned char symbol)`	Вывод символа на дисплей с возвратом результата выполнения операции: `symbol` – выводимый символ.

Кроме того, может быть реализована функция вывода строки на ЖКИ, функция дополнительной настройки ЖКИ (отображение, мерцание курсора).

Драйвер таймера должен включать следующие функции помимо обработчика прерывания (лабораторная работа № 2):

Функция	Описание
`void InitTimer(void)`	Инициализация таймера.
`unsigned long GetMsCounter(void)`	Получение текущей метки времени в миллисекундах.
`unsigned long DTimeMs(unsigned long t0)`	Измерение количества миллисекунд, прошедших с временной метки `t0` и до текущего времени.
`void DelayMs(unsigned long t)`	Задержка на `t` миллисекунд.

Работа с последовательным каналом (приемопередатчиком UART) должна быть организована по прерыванию. Драйвер последовательного канал включает следующие функции помимо обработчика прерывания (лабораторная работа № 3):

Функция	Описание
`void InitSerial(void)`	Инициализация последовательного канала.
`unsigned char WriteSerial(unsigned char data_buf)`	Передача байта данных `data_buf` с возвратом результата выполнения операции.
`unsigned char ReadSerial(unsigned char* data_buf)`	Прием байта данных `*data_buf` с возвратом результата выполнения операции.
`unsigned char`	Чтение признака наличия байта в буфере приема.

`StatusSerial (void)`	

Кроме того, может быть реализована функция вывода строки в последовательный канал.

Требования к выполнению работы

1. Разрабатываемые драйверы устройств должны быть выполнены в виде отдельных программных модулей (файлов), содержащих функции по работе с заданным одним устройством.
2. На уровне драйверов (особенно обработчиков прерываний) НЕ рекомендуется смешивать работу с несколькими периферийными устройствами (например, в обработчике прерывания таймера выводить строку на ЖКИ, опрашивать DIP-переключатели и т.д.). Взаимодействие устройств ввода-вывода следует организовать на прикладном уровне с использованием API-функций их драйверов.
3. Должен быть предусмотрен контроль ввода корректных значений в рамках выполнения прикладной задачи.
4. В программе должны быть использованы механизмы взаимного исключения (см. [14], IOS2003_lab4.pdf).
5. Текст программы должен соответствовать правилам оформления программ на языке Си, приведенным в приложении (Приложение Б. Требования к оформлению программ на языке Си, [28]).

Содержание отчета

1. Титульный лист.
2. Номер варианта, задание.
3. Модель написанной программы (см. Приложение А. Проектирование и разработка программы).
4. Разработанные протоколы, форматы данных и др.
5. Исходный текст программы с комментариями (можно не весь, но обязательно главная программа и полностью драйвер периферийного устройства, изучению которого была посвящена лабораторная работа).
6. Основные результаты.

Литература

Литература к лабораторной работе: [10], [13], [14], [26], [30], [35], [39], [43], [44], [45].

Варианты заданий

1. Тестирование EEPROM.

Контроллер SDK-1.1 организует по последовательному каналу систему меню, по которому можно перемещаться с помощью символов, передаваемых со стороны персонального компьютера с использованием терминальной программы. Прием неправильного символа по последовательному каналу приводит к перерисовке меню. Предлагается следующий вариант меню (как оно может выглядеть):

```
Тест EEPROM
------------------------------------------------------
1 – Запись данных
```

```
2 – Чтение данных
3 – Очистка памяти
4 – Автоматический тест
-----------------------------------------------------
```

Кроме того, аналогичная система меню должна быть организована и на ЖКИ стенда SDK-1.1. Перемещение по этому меню реализуется при помощи клавиатуры SDK-1.1 (кнопки управления по выбору студента). Указанные 4 пункта меню должны быть выполнены обязательно, оформление может быть иным и разным для терминала и ЖКИ, но главное – понятным и удобным для использования.

При запуске теста в терминал и на ЖКИ SDK-1.1 должно выводиться меню.

По выбору пункта меню «Запись данных» EEPROM заполняется последовательностью случайных чисел (127/255 байт[5]) и CRC8, рассчитанного по этой последовательности. Результат выполнения операции (OK/ERR_I2C/ERR_CRC), записанная последовательность и CRC8 в шестнадцатеричном формате выводятся в терминал. Пример вывода:

```
Запись данных [OK]
[FA 11 45 67 23 21 CC E2 99 23 CB B5 5A 2F 25 81
 1F 44 …
 …          …          …          …          …          ]
CRC8 [0x28]
```

Вывод записанных данных для удобочитаемости может быть выполнен в виде 8/16 строк по 16 значений.

На ЖКИ SDK-1.1 отображается результат выполнения операции («`Test EEPROM Write OK`»). По нажатию специальной кнопки на клавиатуре SDK-1.1 на ЖКИ снова выводится меню.

По выбору пункта меню «Чтение данных» из EEPROM считывается записанная последовательность и рассчитывается CRC8 (все 128/256 байт). Результат выполнения операции (OK/ERR_I2C/ERR_CRC), прочитанная последовательность и CRC8 в шестнадцатеричном формате выводятся в терминал. Пример вывода:

```
Чтение данных [OK]
[FA 11 45 67 23 21 CC E2 99 23 CB B5 5A 2F 25 81
 1F 44 …
 …          …          …          …          …          ]
CRC8 [0x28]
```

На ЖКИ SDK-1.1 отображается результат выполнения операции («`Test EEPROM Read OK`»). По нажатию специальной кнопки на клавиатуре SDK-1.1 на ЖКИ снова выводится меню.

По выбору пункта меню «Очистка памяти» выполняется стирание всей памяти EEPROM (заполнение 0xFF). Результат выполнения операции (OK/ERR_I2C/ERR_CRC) выводится в терминал. Пример вывода:

```
Очистка памяти [OK]
```

На ЖКИ SDK-1.1 отображается результат выполнения операции («`Test EEPROM Erase all OK`»). По нажатию специальной кнопки на клавиатуре SDK-1.1 на ЖКИ снова выводится меню.

По выбору пункта меню «Автоматический тест» выполняется запись и чтение данных EEPROM как в пунктах меню 1 и 2. Результат выполнения операции (OK/ERR_I2C/ERR_CRC) выводится в терминал. Пример вывода:

[5] Емкость EEPROM определяется установленной микросхемой в стенде SDK-1.1.

```
Запись данных [OK]
Чтение данных [OK]
```

На ЖКИ SDK-1.1 отображается результат выполнения операции («`Test EEPROM Write & Read OK`»). По нажатию специальной кнопки на клавиатуре SDK-1.1 на ЖКИ снова выводится меню.

Необходимо отметить, что ошибка в качестве результата выполнения операции может быть двух видов: ошибка обмена по каналу I^2C (ERR_I2C) и ошибка в циклическом коде (ERR_CRC). Первая должна отслеживаться на уровне драйвера I^2C, вторая – на уровне расчета CRC8.

В рамках задания необходимо реализовать:

- драйвер последовательного канала;
- драйвер клавиатуры (может быть реализован без переповторов, т.е. по принципу «кнопка нажата или нет»);
- драйвер ЖКИ;
- драйвер I^2C;
- драйвер EEPROM;
- функцию генерации случайного (псевдослучайного) числа в диапазоне от 0 до 255 на основе таймера ADuC812;
- функцию расчета CRC8 с использованием полинома $x^8 + x^5 + x^4 + 1$.

2. Журнал событий.

Журнал событий реализуется на основе EEPROM: в нем сохраняется хронологическая информация о действиях пользователя, производимых со стендом SDK-1.1. Таким действием является:

- Изменение состояния одного из 12 DIP-переключателей (SW3-1 и SW3-2 на рис. 2) с указанием номера и положения вкл./выкл.;
- Нажатие кнопки клавиатуры (одной из 16) с указанием символа.

На запись о произведенном действии в EEPROM выделяется 4 байта, причем формат записи следующий:

- 1 и 2 байт – временная (секундная) метка события, выставляемая по таймеру;
- 3 байт – код события;
- 4 байт – CRC8 по предыдущим трем байтам.

При запуске системы таймер (переменная, отвечающая за системное время) инициализируется значением последней временной метки, присутствующей в журнале событий, если таковая есть. Должны быть реализованы механизмы по определению начала журнала событий (для его вычитывания) и конца (для дополнения новой записью). Необходимо предусмотреть возможность изменения размера журнала событий, т.е. журнал может занимать не всю память EEPROM. Журнал (буфер) является циклическим, то есть в случае записи события в последние 4 байта журнала, следующая запись производится в его первые 4 байта. Кроме того, нужно предусмотреть возможность стирания всей памяти EEPROM.

Реализация системы меню из двух пунктов – чтение журнала событий и очистка EEPROM – на стороне персонального компьютера (терминальная программа). Перемещение по этому меню реализуется при помощи клавиатуры ПК.

Необходимо реализовать возможность передачи содержимого журнала событий по последовательному каналу в персональный компьютер в любой момент. Отображение считанной информации в терминальной программе должно быть выполнено в формате: временная метка – событие. Каждому событию – новая строка. При чтении журнала событий

проверка CRC8 для каждой записи является обязательной. В случае ошибки обмена по каналу I^2C, несовпадения циклического кода прочитанной из журнала записи и др. некорректных ситуаций сообщение об этом должно выводиться в терминал во время чтения журнала.

Последнее произведенное действие (в формате записи в журнале событий), сообщения о выполненных операциях (чтение журнала и очистка памяти), об ошибках в работе должны отображаться на ЖКИ контроллера SDK-1.1. После запуска системы на ЖКИ выводится последняя запись журнала событий.

В рамках задания необходимо реализовать:

- драйвер DIP-переключателей;
- драйвер таймера;
- драйвер последовательного канала;
- драйвер клавиатуры (может быть реализован без переповторов, т.е. по принципу «кнопка нажата или нет»);
- драйвер ЖКИ;
- драйвер I^2C;
- драйвер EEPROM;
- функцию расчета CRC8 с использованием полинома $x^8 + x^5 + x^4 + 1$.

3. Часы.

В задании используются часы реального времени. Необходимо на ЖКИ контроллера SDK-1.1 отображать текущую дату и время в формате «dd.mm.yyyy» на первой строке и «hh:mm:ss» – на второй строке. Введение новой даты и времени организовать с помощью клавиатуры стенда SDK-1.1. Во время редактирования текущее редактируемое знакоместо отображать курсором. Некорректный ввод должен быть блокирован. Для входа/выхода в/из режим(а) редактирования необходимо предусмотреть специальную клавишу (или комбинацию клавиш).

Кроме того, введение новой даты и времени нужно организовать и по последовательному каналу со стороны персонального компьютера с помощью терминальной программы. Формат даты и времени такой же, как и на ЖКИ. Дата и время вводятся вместе (в одной строке). Сообщение о некорректном вводе должно отображаться в терминале.

В рамках задания необходимо реализовать:

- драйвер последовательного канала;
- драйвер клавиатуры (может быть реализован без переповторов, т.е. по принципу «кнопка нажата или нет»);
- драйвер ЖКИ;
- драйвер I^2C;
- драйвер часов реального времени (RTC).

4. Будильник.

На одной из строк ЖКИ контроллера SDK-1.1 отображается текущее время в формате «hh:mm:ss». На другой строке ЖКИ отображается время срабатывания будильника в таком же формате. Чтобы отличать одну строку от другой, можно использовать какой-нибудь специальный символ. Введение текущего времени и нового времени срабатывания будильника организовано по последовательному каналу со стороны персонального компьютера с помощью терминальной программы. Для этого нужно реализовать систему меню из соответствующих двух пунктов. Формат ввода времени такой же, как и на ЖКИ. Некорректный ввод должен быть блокирован.

При наступлении времени срабатывания будильника запускается проигрывание мелодии с помощью звукового пьезоизлучателя контроллера SDK-1.1. После нажатия кнопки клавиатуры контроллера SDK-1.1 сигнал будильника перестает звучать.

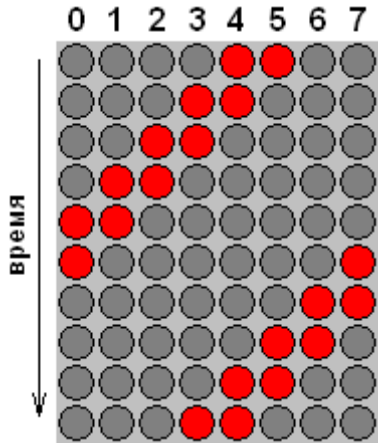
Следует уделить особое внимание тому, что функция будильника реализуется на основе часов реального времени.

В рамках задания необходимо реализовать:

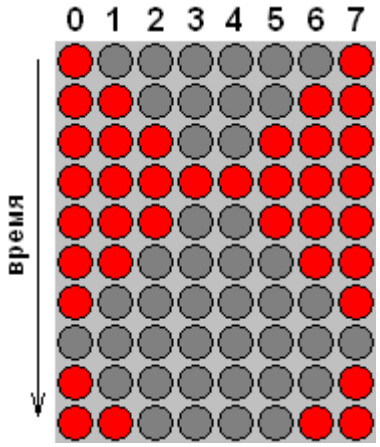
- драйвер звукового излучателя на основе таймера (лабораторная работа № 2);
- драйвер последовательного канала;
- драйвер клавиатуры (может быть реализован без переповторов, т.е. по принципу «кнопка нажата или нет»);
- драйвер ЖКИ;
- драйвер I^2C;
- драйвер часов реального времени (RTC).

5. Сохранение контекста.

На светодиодные индикаторы контроллера SDK-1.1 выводится либо одна, либо другая анимация.



A=1



A=2

Выбор анимации осуществляется при помощи специального параметра (например, A). Отображение первой анимации (A=1) задается с использованием трех параметров: количество зажженных светодиодов (например, N), направление движения зажженных светодиодов (например, D) и скорость движения (например, S). Отображение второй анимации (A=2) задается с использованием двух параметров: количество зажженных светодиодов (N) и скорость движения (S). Параметр S выражается в условных единицах (диапазон значений: 1 – 9), причем 1 – самая низкая скорость, 9 – самая высокая. Для формирования скорости движения анимации обязательным является использование таймера микроконтроллера ADuC812.

Все перечисленные параметры (A, N, D, S) являются контекстными, то есть сохраняются в EEPROM и защищаются CRC8. Таким образом, после перезапуска системы эти параметры считываются и задают режим отображения анимаций. Если при чтении параметров CRC8 не сошелся, то они считаются некорректными и принимаются значения по умолчанию (табл. 5). Необходимо заметить, что параметры N и S имеют различные значения для одной и другой анимации (для каждой анимации свой контекст).

Таблица 5. Допустимые значения параметров анимаций

	A = 1			**A = 2**	
	N	D	S	N	S
Диапазон значений	1 – 7	Налево, направо	1 – 9	1 – 4 (относится к половинкам линейки светодиодов)	1 – 9
Значение по умолчанию (см. рисунки)	2	Налево	1	4	1

Кроме того, все перечисленные параметры отображаются на ЖКИ стенда SDK-1.1 и могут редактироваться при помощи клавиатуры SDK-1.1. На первой строке ЖКИ – выбранный тип анимации, на второй строке – параметры отображаемой анимации. Пример вывода на ЖКИ:

```
     A=1
N=3  D=←  S=5
```

Изменение параметра контекста должно тут же сохраняться в EEPROM и оказывать влияние на выводимую анимацию.

Необходимо реализовать функцию очистки EEPROM по нажатию кнопки клавиатуры SDK-1.1 (например, «#»).

Перечисленные функции по изменению параметров анимаций и очистке памяти EEPROM выполняются и при помощи системы меню, организованной в терминальной программе персонального компьютера (обмен данными по последовательному каналу).

В рамках задания необходимо реализовать:

- драйвер светодиодных индикаторов;
- драйвер таймера;
- драйвер последовательного канала;
- драйвер клавиатуры (может быть реализован без переповторов, т.е. по принципу «кнопка нажата или нет»);
- драйвер ЖКИ;
- драйвер I^2C;
- драйвер EEPROM;
- функцию расчета CRC8 с использованием полинома $x^8 + x^5 + x^4 + 1$.

6. Текстовый редактор с памятью.

Написать программу, реализующую функции простого текстового редактора на ЖКИ стенда SDK-1.1. Ввод текста должен осуществляться с клавиатуры контроллера SDK-1.1 и персонального компьютера (при помощи терминальной программы). Стенд SDK-1.1 подключен к ПК через коммуникационный кабель RS-232. На клавиатуре SDK-1.1 для ввода текста используются только кнопки «0 – 9», таким образом выводимая на ЖКИ информация является исключительно цифровой. Объем выводимого текста не должен превышать 8 строк по 16 символов, при этом ЖКИ отображает только две соседние строки. Номер текущей строки должен отображаться на светодиодном индикаторе (первая строка – первый светодиод, вторая строка – второй светодиод и т.д.).

На клавиатуре SDK-1.1 перемещение по тексту осуществляется с помощью кнопок «A» (влево), «B» (вправо), «C» (вверх) и «D» (вниз), текущая позиция выделяется курсором. Ввод

текста производится в позицию, указываемую курсором, при этом все символы находящиеся правее введенного символа сдвигаются, курсор перемещается (как в обычном текстовом редакторе). В качестве клавиши перевода строки выступает кнопка «*», удаление символов должно производиться кнопкой «#».

На клавиатуре ПК реализуются аналогичные функции редактирования: перемещение вправо-влево и вверх-вниз (клавиши навигации), в начало и в конец текста (Home, End), удаление символа (Delete, Backspace).

Вводимый текст сохраняется в EEPROM и защищается CRC8. Таким образом, после перезапуска системы этот текст считывается и отображается на ЖКИ, готовый для редактирования. Если при чтении сохраненного текста из EEPROM CRC8 не сошелся, то он считается некорректным и принимается значение по умолчанию (на усмотрение студента).

В рамках задания необходимо реализовать:

- драйвер светодиодных индикаторов;
- драйвер последовательного канала;
- драйвер клавиатуры (может быть реализован без переповторов, т.е. по принципу «кнопка нажата или нет»);
- драйвер ЖКИ;
- драйвер I^2C;
- драйвер EEPROM;
- функцию расчета CRC8 с использованием полинома $x^8 + x^5 + x^4 + 1$.

Приложение А. Проектирование и разработка программы

Одним из основных компонентов отчета в лабораторных работах №№5-6 является модель написанной программы. Что под этим понимается?

Сложность задачи, которую необходимо решить в этих лабораторных работах, требует ознакомления с понятием процесса проектирования программного обеспечения. В упрощенном виде к основным этапам традиционного потока проектирования ПО можно отнести:

1. Разработку технического задания и спецификаций (в данном случае они уже представлены и предложены в виде варианта задания).
2. Архитектурное проектирование (по-другому, системное или концептуальное проектирование).
3. Разработку программного обеспечения (непосредственная реализация, т.е. кодирование).
4. Отладку и тестирование.
5. Создание рабочей документации (в данном случае это отчет).
6. Ввод в эксплуатацию, сопровождение (в данном случае это демонстрация преподавателю выполненной работы и ее защита).

Вопросам технологий проектирования программного обеспечения в частности и вычислительных систем в целом, проблемам в различных подходах к проектированию и разработке систем, так называемому «кризису сложности» в вычислительной технике посвящено множество статей и книг [1, 2, 5, 6, 7, 8, 12, 19, 20, 36, 37, 40, 45]. Однако недостаточное качество, многократно заваленные сроки выполнения, высокая стоимость до сих пор являются характерными чертами систем такого рода.

В данной работе Вам настоятельно рекомендуется не пропускать этап архитектурного проектирования, т.е. продумать, как Вы будете решать поставленную задачу, а потом уж приступать к кодированию придуманного. Что это Вам даст? Грамотное системное проектирование

- существенно сократит время реализации ПО в частности и срок выполнения работы в целом (примерное временно*е* соотношение этапа №2 и этапа №3 – «80% к 20%»);
- сделает Вашу программу прозрачной, простой для понимания, предсказуемой, значит, легко отлаживаемой;
- повысит качество;
- сократит ресурсоемкость;
- добавит в Вашу программу возможность наращивания функциональности, модифицируемости и дальнейшего развития (если, конечно, критерий повторного использования учитывался на этапе проектирования);
- сделает более доступным анализ разработанного ПО;
- и т.д.

Этап архитектурного проектирования обычно тесно связан с вопросом описания разрабатываемой системы. Недостатком вербального описания программного обеспечения является низкая степень формализации, т.е. практическая невозможность математического доказательства правильности тех или иных утверждений, выраженных таким способом. Блок-схема алгоритма, применяющаяся для описания сравнительно несложных программ, уже не дает никакого эффекта в крупных проектах. Еще в работах Дейкстры и Вирта, а далее в работах Йордона, Росса и др. было предложено описывать программные системы в виде совокупности структурных и поведенческих составляющих. Система разрабатывалась на основе декомпозиции (разделения) общих сущностей на более частные. Была предложена так называемая «абстракция», т.е. выделение существенных для проектировщика деталей проекта и сокрытие второстепенных.

Что такое структура и поведение? Структура системы – это совокупность частей (элементов и подсистем) и связи между ними. Поведение системы – это изменение структурных составляющих (подсистем и элементов), а также связей между ними во времени.

Для описания двух этих понятий, люди издавна используют графические изображения. К сожалению, пока не существует способа изображения на одной картинке всей структуры или поведения вычислительной системы. Приходится рассматривать и структуру, и поведение с разных позиций.

Структура может быть представлена следующими способами:

- совокупность блоков системы и интерфейсов (объект A соединен с объектом B с помощью интерфейса I^2C);
- совокупность объектов и зависимостей (объект «дом» зависит от объекта «электростанция»);
- схема наследования классов (класс X происходит от классов Y и Z);
- схема включения классов (агрегация, класс Q включает в себя объекты S и D);

Поведение можно представить в виде:

- конечного автомата [29];
- временной диаграммы (процессограммы, диаграммы взаимодействий и т.п.);
- потоковой диаграммы (DFD, CFD и т.п.).

Эти перечни не претендуют на полноту. В книгах по проектированию вычислительных систем можно найти другие способы описания структуры и поведения или, так называемых, нотаций [2].

Требуемая в отчете модель программы – это и есть ее архитектурное описание. В случае прохождения этапа архитектурного проектирования представить эту модель не составляет труда.

Чтобы проанализировать и оценить логику работы программы (решения поставленной задачи), организацию работы с периферийными устройствами, алгоритмы управления в прикладной и системной части необходимо оторваться от строчек кода (абстрагироваться) и посмотреть на программу в целом, панорамно, по уровням. Поэтому Вам нужно выделить основные сущности Вашей программы и способы (каналы) их взаимодействия – соответственно получается модель первого уровня. Потом (на втором уровне) необходимо представить модели каждой из выделенных сущностей: чаще всего это может быть модель функционирования драйвера клавиатуры, последовательного канала, звукового излучателя, шины I^2C, основного прикладного алгоритма и т.д. с выделением потока данных и/или управления (команд). Например, в лабораторной работе № 4 «Клавиатура» для представления задачи сканирования клавиатуры (это лишь часть драйвера клавиатуры) прекрасно подходят конечные автоматы (FSM) [2, 25, 29]. Также могут быть применены сети процессов Кана, DFD и другие перечисленные ранее способы описания структуры и поведения системы. Рекомендуется ознакомиться с этими нотациями, но можно использовать и свой способ описания, – главное, чтобы он удовлетворял перечисленным выше требованиям.

Приложение Б. Требования к оформлению программ на языке Си

Соглашения по идентификаторам

Подбор идентификаторов

А. Все идентификаторы должны выбираться из соображений читаемости и максимальной семантической нагрузки.

Например:

```
const float Eps = 0.0001;      // точность
unsigned short Sum;            // сумма
unsigned char Message[ 20 ];   // сообщение
```

Неудачными можно считать идентификаторы:

```
const float UU = 0.0001;       // точность
unsigned short Kk;             // сумма
unsigned char Zz[ 20 ];        // сообщение
```

Б. Идентификаторы рекомендуется подбирать из слов английского языка.

Например:

```
// выдает звуковой сигнал заданной частоты и длительности
void Beep( unsigned short Hertz, unsigned short MSec );
// выдает True (1), если файл с именем FName существует
unsigned char ExistFile( unsigned char* FName );
// признак окончания работы с программой
unsigned char Done;
// размеры изделия (ширина, высота)
unsigned short Width, Height;
```

Не очень удачными можно считать идентификаторы:

```
// выдает звуковой сигнал заданной частоты и длительности
void Zvuk(unsigned short Chast, unsigned short Dlit );
// выдает True (1), если файл с именем Im существует
unsigned char EstFile(unsigned char* Im );
// признак окончания работы с программой
unsigned char Konec;
// размеры изделия (ширина, высота)
unsigned short Shirina, Vysota;
```

Написание идентификаторов

Существует два основных способа написания идентификаторов.

А. В любых идентификаторах каждое слово, входящее в идентификатор, писать, начиная с большой буквы, остальные буквы – маленькие.

Например:

```
float NextX, LastX;      // следующая и предыдущая итерация
char BeepOnError;        // подавать ли звуковой сигнал при
                         // неправильном вводе пользователя?
unsigned char FileName[ 20 ];

// стандартная функция модуля Graph; выдает описание
// ошибки использования графики по ее коду
unsigned char* GraphErrorMsg( short ErrCode );
```

Б. В любых идентификаторах каждое слово, входящее в идентификатор, разделять символом “_”, при этом все буквы – маленькие.

Например:

```
float next_x, last_x;    // следующая и предыдущая итерация
char beep_on_error;      // подавать ли звуковой сигнал при
                         // неправильном вводе пользователя?
unsigned char file_name[ 20 ];

// стандартная функция модуля Graph; выдает описание
// ошибки использования графики по ее коду
unsigned char* graph_error_msg( short err_code );
```

Соглашения по самодокументируемости программ

Комментарии

А. Комментарии в теле программы следует писать на русском языке и по существу так, чтобы программист, не участвовавший в разработке программы (но имеющий опыт работы на языке Си), мог без особого труда разобраться в логике программы, и, при необходимости, сопровождать данный программный продукт.

Б. Рекомендуется комментарии к программе писать после символов //, а /* и */ использовать при отладке программы как "заглушки" участков программного кода.

Спецификация функций

Для каждой пользовательской функции должна быть описана в виде комментария спецификация, содержащая следующую информацию [28]:

- назначение функции;
- описание семантики параметров-значений (параметров, передаваемых по значению);
- описание семантики параметров-переменных (параметров, передаваемых по ссылке);
- описание семантики возвращаемого значения.

Например:

```
//////////////////////////Gauss//////////////////////////////
// Решение системы линейных алгебраических уравнений
// методом Гаусса.
// Вход:
//          A   - матрица коэффициентов системы;
//          B   - столбец свободных членов системы;
//          Eps - точность вычислений.
// Выход:
//          X                - вектор решения;
//          HasSolution      - флаг, устанавливаемый в True, если
//                             решение системы существует, и в False
//                             во всех остальных случаях;
//          NumOfRoots       - число корней в решении системы, может
//                             принимать значения:
//                             0         - если решение системы не
//                                         существует,
//                             MaxN      - если решение системы
//                                         существует и единственно,
//                             MaxInt    - если существует бесконечное
//                                         множество решений;
//          Det              - значение определителя матрицы A;
//          AForReverse      - нижняя треугольная матрица,
//                             полученная из A в результате
//                             выполнения прямого хода алгоритма
```

```
//                              Гаусса;
//          BForReverse       - столбец свободных членов, полученный
//                              из B в результате выполнения
//                              прямого хода алгоритма Гаусса.
// Результат: 0 - успешно,1 – ошибка.
////////////////////////////////////////////////////////////

unsigned char Gauss( Matrix A, Vector B, float Eps, Vector* X,
 char* HasSolution, short* NumOfRoots,
  float* Det, Matrix* AForReverse,
  Vector* BForReverse )
{
…
return 0;
…
return 1;
}
```

Замечание:

Если функция реализует какой-либо вычислительный метод (например: нахождение площади фигуры методом трапеций, поиск минимума функции методом Ньютона и т.п.), рекомендуется в теле функции поместить комментарий с кратким описанием метода, либо ссылку на источник, где описан метод [28].

Спецификация программного файла или модуля

Программный файл или модуль должен начинаться со спецификации в виде комментария, содержащего следующую информацию [28]:

- идентификация проекта, к которому принадлежит файл;
- назначение (название) и имя файла;
- версия файла;
- фамилия автора;
- описание модуля;
- история изменений модуля.

Например:

```
/*-------------------------------------------------------------
Проект:     ABC-1.0
Название:   Математические расчеты
Файл:       primes.c
Версия:     1.0.2
Автор:      Иванов И.И.
Описание:   Подсчет количества простых чисел.
Изменения:
-------------------------------------------------------------
N  Дата       Версия   Автор        Описание
-------------------------------------------------------------
1  01.03.07   1.0.1    Иванов И.И.  Расчет в промежутке [1..200].
2  05.07.07   1.0.2    Иванов И.И.  Расчет в заданном пользователем промежутке.
-------------------------------------------------------------*/
```

Замечание:

После спецификации программного файла рекомендуется поместить комментарий с указаниями по запуску программы и работе с ней (указаниями по использованию модуля другими программистами) или ссылку на источник, который использован при составлении программы (модуля).

Соглашения по читаемости программ

Лесенка

"Лесенка" должна отражать структурную вложенность языковых конструкций. Рекомендуется отступ не менее 2-х и не более 8-и пробелов. Принятого отступа нужно придерживаться во всем тексте программы. Правила написания некоторых конструкций [28]:

```
if ( <условие> )
{
    <операторы>
{

if ( <условие> )
    <оператор>;

if ( <условие> )
{
    <операторы>
}
else
{
    <операторы>
}

while ( <условие> )
{
    <операторы>
}

while ( <условие> )
    <оператор>;

for ( <иниц. счетчика>; <условие>; <изменение счетчика> )
{
    <операторы>
}

for ( <иниц. счетчика>; <условие>; <изменение счетчика> )
    <оператор>;

switch ( <выражение> )
{
    case <выражение>:
        <операторы>;
        break;
    ........
    default:
        <операторы>;
}

short Sign( float X )
{
    // выдает знак числа X
    if ( X > 0 ) return 1;
    else
        if ( X < 0 ) return -1;
        else
            return 0;
}

void Equation( float A, float B, float C,
               float* X1, float* X2, char* Num )
{
```

```
    // нахождение действительных корней квадратного уравнения;
    // A, B, C -- коэффициенты
    // X1, X2  -- корни (если действительного решения нет, то
    //             полагаются равными 0);
    // Num     -- число корней (0, 1, или 2)

    float D;

    D = sqr( B ) -4 * A * C;
    if ( D < 0 )
    {
        *Num = 0;
        *X1 = 0;
        *X2 = 0;
    }
    else
    {
        *X1 = ( -B + sqrt( D ) ) / ( 2 * A );
        *X2 = ( -B - sqrt( D ) ) / ( 2 * A );
        if ( *X1 == *X2 ) *Num = 1;
        else *Num = 2;
    }
}
```

Длина строк программного текста

Длина строк программы не должна превышать ширины экрана (80 символов).

Прочие рекомендации

А. Рекомендуется операнды бинарных операций (+, = и т.п.) отделять от знака операции одним пробелом " ".

Например:

```
Sum = A + B;
```

Б. Рекомендуется при перечислении идентификаторов после запятой "," ставить один пробел " ".

Например:

```
printf( "Сумма: %d; Разность: %d.", A + B, A – B );
unsigned short Day, Month, Year;
unsigned char i, j, k, l, m, n;
```

В. Рекомендуется всегда писать символ-разделитель операторов ";" (непосредственно после оператора).

Например:

```
switch ( Num )
{
    case 1: printf( "один..." ); break;
    case 2: printf( "два..." ); break;
    case 3: printf( "три..." ); break;
    default: printf( "много!" ); break;
}
```

Г. Рекомендуется 16-ричные числа писать большими буквами.

Например:

```
#define BadDate 0xFFFF
#define kbEnter 0x0D // код клавиши <Enter>
```

Литература

1. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. СПб.: Символ-Плюс, 2010. 304 с. ISBN 5-93286-005-7, ISBN 0-201-83595-9
2. Буч Г. Объектно-ориентированный анализ и проектирование с примерами на языке C++. СПб.: Бином, Невский диалект, 1998. 560 с. ISBN 0-8053-5340-2, ISBN 5-7989-0067-3
3. Гук М.Ю. Аппаратные интерфейсы ПК. Энциклопедия. СПб.: Питер, 2002. 528 с.: ил. ISBN 5-94723-180-8
4. Гук М.Ю. Аппаратные средства IBM PC. Энциклопедия. 3-е изд. СПб.: Питер, 2006. 1072 с.: ил. ISBN 5-469-01182-8
5. Дейкстра Э.В. Два взгляда на программирование // Клуб программистов «Весельчак У» URL: http://club.shelek.ru/viewart.php?id=211 (дата обращения: 01.09.2010).
6. Дейкстра Э.В. Конец информатики? // Communications of the ACM, Ноябрь, 19, 2000. 44(3). с. 92
7. Дейкстра Э.В. Почему программное обеспечение такое дорогое? Пояснение для разработчиков аппаратуры // Springer-Verlag, 1982. с. 338-348
8. Дейкстра Э.В. Программирование как вид человеческой деятельности // Клуб программистов «Весельчак У» URL: http://club.shelek.ru/viewart.php?id=137 (дата обращения: 01.09.2010).
9. Игнатов В. Эффективное использование GNU Make // Центр Информационных Технологий. 1997. URL: http://www.citforum.ru/operating_systems/gnumake/index.shtml (дата обращения: 01.09.2010).
10. Интерфейсная шина IIC (I2C) // Easy Elecronics – Электроника для всех. 2008. URL: http://easyelectronics.ru/interface-bus-iic-i2c.html (дата обращения: 01.09.2010).
11. Керниган Б., Ритчи Д. Язык программирования C. 5-е изд. М.: Вильямс, 2009. 304 с. ISBN 978-5-8459-0891-9, ISBN 5-8459-0891-4, ISBN 0-13-110362-8
12. Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительных систем. СПб.: СПбГУ ИТМО, 2009. 212 с.
13. Ключев, А.О., Ковязина, Д.Р., Кустарев, П.В., Платунов, А.Е. Аппаратные и программные средства встраиваемых систем. Учебное пособие. СПб.: СПбГУ ИТМО, 2010. 287 с.: ил.
14. Комплекс лабораторных работ для учебного лабораторного стенда SDK-1.1 // Интернет-портал «Встроенные вычислительные системы и системы на кристалле». 2009. URL: http://embedded.ifmo.ru/sdk/sdk11/labs/2003 (дата обращения: 01.09.2010).
15. Кузьминов А.Ю. Интерфейс RS232: Связь между компьютером и микроконтроллером: От DOS к WINDOWS98/XP. М.: Издательский дом «ДМКпресс», 2006. 320 с. ISBN 5-9706-0029-6
16. Логическая схема расширителя портов ввода-вывода стенда SDK-1.1 (Rev. 3) // Интернет-портал «Встроенные вычислительные системы и системы на кристалле». 2009. URL: http://embedded.ifmo.ru/sdk/sdk11/sch/sdk11r3_pld_ext.pdf (дата обращения: 01.09.2010).
17. Матричная клавиатура // Easy Elecronics – Электроника для всех. 2008. URL: http://easyelectronics.ru/matrichnaya-klaviatura.html (дата обращения: 01.09.2010).
18. Матричная клавиатура // SKF development. 2009. URL: http://www.microcontrollerov.net/index.php/ru/microcontrollers/articles/23-matrixkeyboard (дата обращения: 01.09.2010).
19. Непейвода Н.Н., Скопин И.Н. Основания программирования. М.-Ижевск: Институт компьютерных исследований, 2003. 868 с. ISBN 5-93972-299-7
20. Платунов А.Е., Постников Н.П. Перспективы формализации методов проектирования встроенных систем // «Электронные компоненты», 2005, №1. с.1-6
21. Принципиальная электрическая схема стенда SDK-1.1 (Rev. 4) // Интернет-портал «Встроенные вычислительные системы и системы на кристалле». 2009. URL: http://embedded.ifmo.ru/sdk/sdk11/sch/sdk1_1_sch_rev4.pdf (дата обращения: 01.09.2010).

22. Спецификация ЖКИ WH1602B-YGK-CP (Winstar Display Co.) // Интернет-портал «Встроенные вычислительные системы и системы на кристалле». 2009. URL: http://embedded.ifmo.ru/sdk/sdk11/components/lcd/WH1602B-YGK-CP.pdf (дата обращения: 01.09.2010).
23. Спецификация интерфейса I²C (рус.) // KAZUS.RU «Электронный портал». 2003. URL: http://kazus.ru/articles/343.html (дата обращения: 01.09.2010).
24. Спецификация контроллера ЖКИ HD44780U (HITACHI) // Интернет-портал «Встроенные вычислительные системы и системы на кристалле». 2009. URL: http://embedded.ifmo.ru/sdk/sdk11/components/lcd/hd44780.pdf (дата обращения: 01.09.2010).
25. Татарчевский В. Применение SWITCH-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 8 // «Компоненты и технологии», 2007, №8. с. 170-172
26. Учебный стенд SDK-1.1. Руководство пользователя (Версия 1.0.11) // Интернет-портал «Встроенные вычислительные системы и системы на кристалле». 2009. URL: http://embedded.ifmo.ru/sdk/sdk11/doc/sdk11_userm_v1_0_11.pdf (дата обращения: 01.09.2010).
27. Цикл лекций по микроконтроллерам семейства MCS-51 // Цифровая техника в радиосвязи. 2009. URL: http://digital.sibsutis.ru/content.htm (дата обращения: 01.09.2010).
28. Цымблер М.Л. Требования к оформлению программ на языке Turbo Pascal // М.Л. Цымблер. 2010. URL: http://www.mzym.susu.ru/papers/coderule.html (дата обращения: 01.09.2010).
29. Шалыто А.А. Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.
30. ADuC812: MicroConverter, Multichannel 12-Bit ADC with Embedded Flash MCU Data Sheet (Rev E, 04/2003) // Norwood: Analog Devices Inc. 2003. URL: http://www.analog.com/static/imported-files/data_sheets/ADUC812.pdf (дата обращения: 01.09.2010).
31. Application Note #1 (uC001): MicroConverter I2C Compatible Interface (Rev. 2.1, 2002/11) // Norwood: Analog Devices Inc. 2003. URL: http://www.analog.com/static/imported-files/application_notes/uC001_-_MicroConverter_I2C_Compatible_Interface.pdf (дата обращения: 01.09.2010).
32. Ayala K. The 8051 Microcontoller, 3rd ed. Delmar Cengage Learning, 2004. 448 p.
33. Ayala K. The 8051 microcontroller: architecture, programming and applications; 2nd ed. NY: Thomson Delmar Leaning, 1996. 367 p.
34. GNU make manual // The GNU Operating System. 1996. URL: http://www.gnu.org/software/make/manual/html_node/index.html (дата обращения: 01.09.2010).
35. I²C-bus specification and user manual (Rev. 03, 06/2007) // NXP Semiconductors. 2006. URL: http://www.nxp.com/documents/user_manual/UM10204.pdf (дата обращения: 01.09.2010).
36. Lee A. Edward. Cyber-Physical Systems – Are Computing Foundations Adequate? // Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap. October 16-17, 2006, Austin, TX. 9 p.
37. Lee E.A. Model-Driven Development – From Object-Oriented Design to Actor-Oriented Design // In Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (a.k.a. The Monterey Workshop). Chicago. September 24, 2003. 7 p.
38. Mazidi M.A., McKinlay R. 8051 Microcontroller and Embedded Systems. NJ: Prentice Hall, 2005. 640 p.
39. PCF8583 – Clock/calendar with 240 x 8-bit RAM // NXP Semiconductors. 2006. URL: http://www.nxp.com/documents/data_sheet/PCF8583.pdf (дата обращения: 01.09.2010).
40. Sangiovanni-Vincentelli A. Quo Vadis SLD: Reasoning About the Trends and Challenges of System Level Design // Proceedings of the IEEE. 95(3), 2007. P. 467-506.

41. Schultz T.W. C and the 8051. Otsego: PageFree Publishing Inc., 2004. 3rd ed. 412 p. ISBN 1-58961-237-X
42. SDCC Compiler User Guide // SDCC – Small Device C Compiler. 2009. URL: http://sdcc.sourceforge.net/doc/sdccman.html/ (дата обращения: 01.09.2010).
43. The I²C-bus specification (version 2.1, 2000) // NXP Semiconductors. 2006. URL: http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf (дата обращения: 01.09.2010).
44. Two-wire Serial EEPROM AT24C02A/AT24C04A // Atmel Corporation. 1998. URL: http://www.atmel.com/dyn/resources/prod_documents/doc5083.pdf (дата обращения: 01.09.2010).
45. Wolf W.H. Computers as Components: Principles of Embedded Computing Systems Design. San Francisco: Morgan Kaufmann, 2005. 656 p. – ISBN 978-0-12-369459-1