Университет ИТМО

**Лабораторная работа №4 по дисциплине**

**«Системы ввода/вывода и**

**периферийные устройства»**

Выполнили:

студенты 4-го курса

группы P3415

Припадчев Артём

Кунцова Анастасия

Григорьева Екатерина

Санкт-Петербург

2015

**Задание**

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

**Вариант 8**

 Скорость последовательного канала – 2400 бит/с.

 Целочисленный делитель десятичных чисел. Диапазон значений делимого и делителя – от 010 до 9910 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.

**Теоретическая часть**

*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.



**Рисунок 1 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 0х080000): 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. 1, рис. 2). 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.



**Рисунок 2 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 3). Такое многократное замыкание/размыкание контактов вызвано тем, что контакты пружинят, обгорают и тому подобное.



Рисунок 3 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. 4 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.



Рисунок 4 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. 5).



Рисунок 5 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. 6). Таким образом, можно заметно увеличить количество кнопок, только хватило бы разрядности дешифратора. Главное – учитывать на каком такте счетчика будет какой столбец.



Рисунок 6 Расширение разрядности клавиатурной матрицы счетчиком и дешифратором

**Блок-схема**

****

**Исходный код**

#include "aduc812.h"

#include "sio.h"

#include "led.h"

#include "max.h"

#define NEWLINE 0x0A

#define DEFAULT\_VAL 0xFF

#define MAX\_RESULT 0x63

#define MAX\_KEYS\_PRESSED 3

unsigned char mode, result, d1, d2;

char drebezg[4][4];

int time\_count[4][4];

unsigned char temp;

unsigned char temo = 0x00;

char queue[20];

char frontQ = -1;

char endQ = -1;

char dipmod=0;

unsigned char colnum=3;

char KBTable[]="147\*2580369#ABCD";

void insertQ(unsigned char a) {

 if (frontQ == 19)

 frontQ = -1;

 queue[++frontQ] = a;

}

unsigned char getQ() {

 unsigned char symbol;

 if (endQ == 19)

 endQ = -1;

 symbol = queue[++endQ];

 return symbol;

}

unsigned char ScanKBOnce()

{

unsigned char row,col,rownum;

unsigned int i;

 if (colnum==3)

 colnum=0;

 else

 colnum++;

 col = 0x1 << colnum; //0001,0010,0100,1000,0001,...

write\_max(KB, ~col); //11111110,11111101,11111011,11110111,11111110,...

row=read\_max(KB) & (0xF0);

 row=~row>>4;

 return row;

}

void moderes() {

 mode = 1;

 d1 = 0xFF;

 d2 = 0xFF;

 result = 0xFF;

}

void mistake() {

 EA = 0; //блокируем прерывания

 ES = 0;

 type ("Mistake has happend\n");

 leds(0xFF);

 frontQ=-1;

 endQ=-1;

 moderes();

 EA = 1;

 ES = 1;// разрешаем прерывания от таймера для опроса

}

void fantom\_mis(){

 unsigned char i,j;

 EA=0;

 type("fantom mistake\n");

 //leds(0xAA);

 for(i=0;i<4;i++)

 for(j=0;j<4;j++){

 drebezg[i][j]=0;

 time\_count[i][j]=0;

 }

 EA=1;

}

void TIMER\_T2(void) \_\_interrupt( 3 ){

 EA = 0;

 TH2 = 0xFE;

 TL2 = 0x00;

 leds(0xFF);

 EA = 1;

}

void TIMER\_KB(void) \_\_interrupt( 2 ){

 unsigned char i,j,count;

 unsigned char input;

 unsigned char rownum;

 input=ScanKBOnce();//&(0x0f)

 for(rownum=0;rownum<4;rownum++)

 {

 count=0;

 if(input&(0x01<<rownum))

 {

 if(drebezg[colnum][rownum]==5)

 time\_count[colnum][rownum]++;

 else

 drebezg[colnum][rownum]++;

 }

 else

 {

 if(drebezg[colnum][rownum]>0)

 drebezg[colnum][rownum]--;

 else if (drebezg[colnum][rownum]==0)

 if(time\_count[colnum][rownum]>0)

 time\_count[colnum][rownum]=0;

 }

 for(i=0;i<4;i++)

 for(j=0;j<4;j++)

 if(drebezg[i][j] == 5)

 count = count + 1; //for multyple pushed buttons\*/

 if(count > MAX\_KEYS\_PRESSED)

 fantom\_mis();

 else

 {

 if(time\_count[colnum][rownum]==1 && drebezg[colnum][rownum]==5) //first enterance

 {

 insertQ(KBTable[colnum\*4+rownum]);

 }

 if(time\_count[colnum][rownum]==100)

 { //after 1 sec 300

 insertQ(KBTable[colnum\*4+rownum]);

 time\_count[colnum][rownum]=80; //once 250

 }

 }

 }

 TH0 = 0xFE; // Инициализация таймера 0 d8 fe

 TL0 = 0xB0; //частота=10кГц(1 в 10млсек)

 //ES=1;

 //TI=1;

}

void SIO\_ISR( void ) \_\_interrupt ( 4 )

{

 if(TI)

 {

 if(dipmod==0){ //just send

 temp=getQ();

 leds(temp);

 SBUF=temp;

 TI=0;

 while(!TI);

 ES=0;

 ET0=1;

 }

 else if (dipmod==1){

 if ( result != DEFAULT\_VAL )

 {

 //0x63 - это максимальный возможный результат деления.

 //Этот блок нужен для отправки \n после числа

 if ( result > MAX\_RESULT )

 {

 SBUF = NEWLINE;

 moderes();

 TI=0;

 while(!TI);

 ES=0;

 ET0=1;

 }

 //Данный блок отправляет цифру - десятки

 else if ( result >= 10 )

 {

 SBUF = ( result / 10 ) + 0x30;

 result = result % 10;

 TI = 0;

 while(!TI);

 }

 //Данный блок отправялцет цифру - единицы

 else if (result <10)

 {

 SBUF = result + 0x30;

 result = 0xAA;

 TI = 0;

 while(!TI);

 }

 }

 else TI=0;

 }

 }

 if(RI)

 {

 leds(0xAA);

 }

}

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

}

void main( void )

{

 int c;

 unsigned char i;

 unsigned char dipval;

 SetVector( 0x2023, (void \*)SIO\_ISR );

 SetVector( 0x200B, (void \*)TIMER\_KB);

 SetVector( 0x202B, (void \*)TIMER\_T2);

 init\_sio\_timer( S2400 );

 moderes();

 while(1){

 if(readdip()==0x01){ //CALC

 dipmod=1;

 if(frontQ!=endQ){

 //ES=1; //разрешаем uart

 ET0=0; //запрещем прерывания таймера.

 c = getQ(); //считываем значения из буфера

 leds(c);

 if (c>=0x30 &&c<=0x39 ) {

 switch (mode) {

 case 1:

 if (d1 == 0xFF)

 d1 = (c-0x30);

 else if (d1 < 10) {

 d1 \*= 10;

 d1 += (c - 0x30);

 }

 else

 mistake();

 break;

 case 2:

 if (d2 == 0xFF)

 d2 = (c-0x30);

 else if (d2 < 10){

 d2 \*= 10;

 d2 += (c - 0x30);

 }

 else

 mistake();

 break;

 default:

 mistake();

 break;

 }

 }

 else if (c == 'C') { //replace to D

 if(mode!=2)

 mode = 2;

 else

 mistake();

 }

 else if (c == '#') {

 if ( d1 < 100 && d2 > 0 && d2 < 100 ){

 result = d1/d2;

 leds(result);

 ES=1;

 TI=1;

 }

 else

 {

 mistake();

 }

 }

 else {

 //mistake();

 moderes();

 }

 //ET0=1; //разрешаем прерывания от таймера

 //ES=0;

 }

 else{

 ES=0;

 ET0=1;

 }

 }

 else { //just read from keyboard

 dipmod=0;

 if(frontQ!=endQ){

 ES=1;

 ET0=0;

 TI=1;

 }

 else{

 ES=0;

 ET0=1;

 }

 }

}

}