Университет ИТМО

Факультет программной инженерии
и компьютерной техники

Кафедра вычислительной техники

Информационно-управляющие системы
Лабораторная работа №4“
Вариант №8

Выполнили:
Милосердов А. О.
Калугин Ф. И.
Группа P3410

Преподаватель:
Ключев А. О.

Санкт-Петербург
2017 г.

Описание задания

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

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.

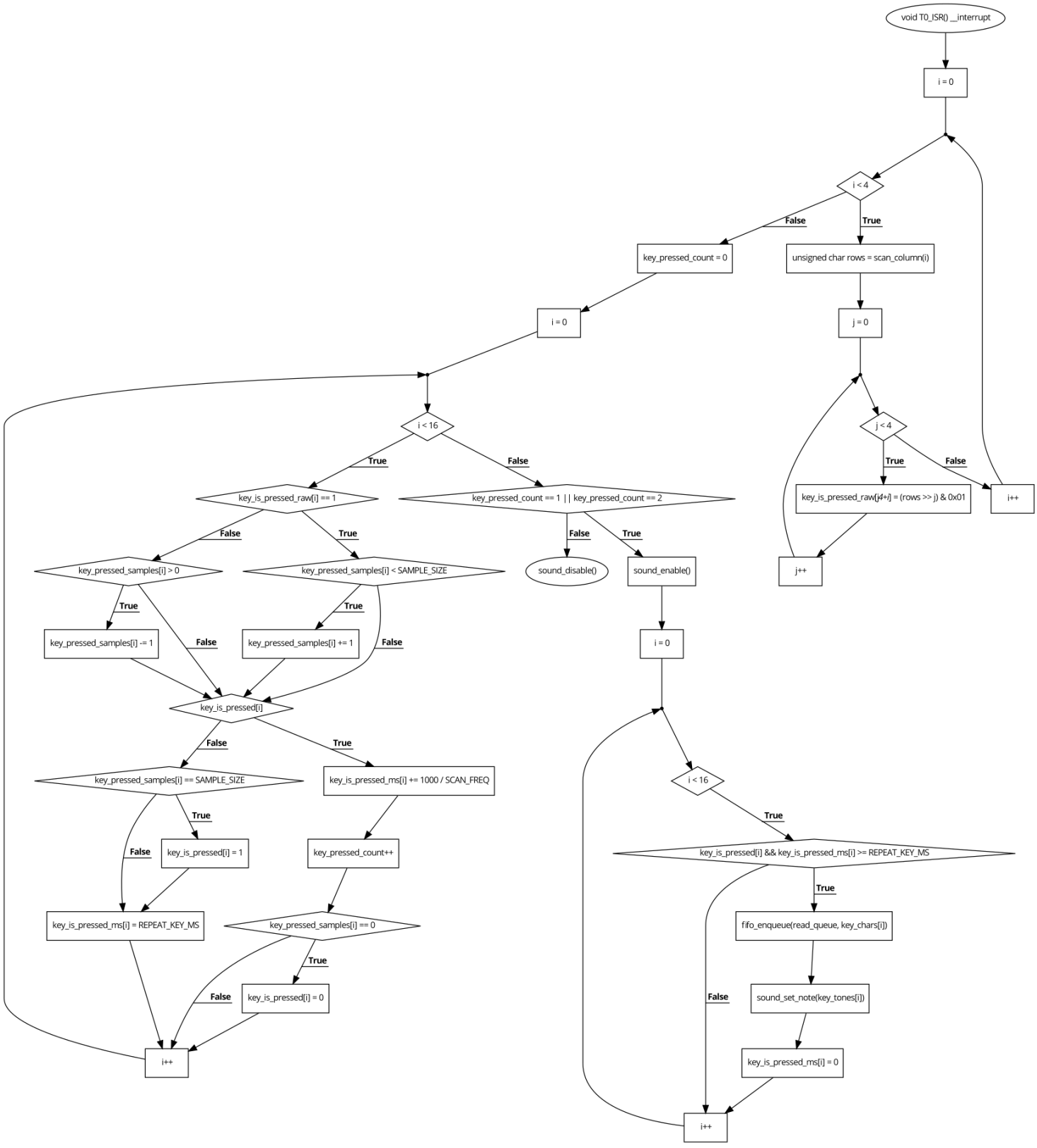

Рис. 1: Блок схема функции сканирования нажатия кнопки на клавиатуре

Листинги

app_calc.c

```
#include "app_calc.h"
#include "uart_async.h"
#include "keypad.h"
#include "util.h"

typedef enum {
    init,
    read_dividend,
    read_divisor,
    print_result,
    print_error
} calc_state_t;

static calc_state_t calc_state;
static unsigned char length;
static int divident;
static int divisor;

void init_calc_app() {
    uart_init_async();
    keypad_init();
    calc_state = init;
}

void run_calc_app() {
    char c = keypad_read();
    if ( c ) uart_write_async(c);
    switch (calc_state) {
        case init:
            calc_state = read_dividend;
            length = 0;
            divident = 0;
            divisor = 0;
            break;
        case read_dividend:
            if ( c >= '0' && c <= '9' ) {
                if (length == 2) {
                    calc_state = print_error;
                }
                if (length == 1) {
                    divident *= 10;
                }
                divident += c - '0';
                length++;
                break;
            }
            if ( c == '/' ) {
                if (length == 0){
                    calc_state = print_error;
                }
                else {
                    length = 0;
```

```
                    calc_state = read_divisor;
                }
                break;
            }
            if ( c == 0 ) {
                break;
            }
            calc_state = print_error;
            break;
        case read_divisor:
            if ( c >= '0' && c <= '9' ) {
                if (length == 2) {
                    calc_state = print_error;
                }
                if (length == 1) {
                    divisor *= 10;
                }
                divisor += c - '0';
                length++;
                break;
            }
            if ( c == '=' ) {
                if (length == 0)
                    calc_state = print_error;
                else
                    calc_state = print_result;
                break;
            }
            if ( c == 0 ) {
                break;
            }
            calc_state = print_error;
            break;
        case print_result:
            if (divisor == 0){
                uart_write_str_async("\r\nCannot divide by 0!\r\n");
            } else {
                char str[3];
                itoa(divident / divisor, str);
                uart_write_str_async(str);
                uart_write_str_async("\r\n");
            }
            calc_state = init;
            break;
        case print_error:
            uart_write_str_async("\r\nInput error!\r\n");
            calc_state = init;
            break;
    }
}
```

app_calc.h

```
#pragma once

void init_calc_app();
```

```
void run_calc_app();
```

app_echo.c

```
#include "app_echo.h"
#include "uart_async.h"
#include "keypad.h"

void init_echo_app() {
    uart_init_async();
    keypad_init();
}

void run_echo_app() {
    char c = keypad_read();
    if( c ) {
       uart_write_async( c );
       uart_write_str_async( "\r\n" );
    }
}
```

app_echo.h

```
#pragma once

void init_echo_app();
void run_echo_app();
```

dip.c

```
#include "max.h"
#include "dip.h"

unsigned char read_dip() {
    return read_max( EXT_LO );
}
```

dip.h

```
#pragma once

unsigned char read_dip();
```

fifo.c

```
#include "fifo.h"

void fifo_init( char* fifo ) {
    unsigned int i;
    for (i = 0; i < FIFO_SIZE; i++) {
        fifo[i]='\0';
    }
}

void fifo_enqueue( char* fifo, char c ) {
    unsigned int count = 0;
    while (*fifo) {
```

```
        count++;
        fifo++;
    }
    if (count < FIFO_SIZE - 1) {
        *fifo = c;
    }
}

char fifo_dequeue( char* fifo ) {
    unsigned int i;
    char c = fifo[0];
    for (i = 0; i < FIFO_SIZE - 1; i++) {
        fifo[i] = fifo[i+1];
    }
    return c;
}

char fifo_peek( char* fifo ) {
    return fifo[0];
}

unsigned int fifo_get_size( char* fifo ) {
    unsigned int count = 0;
    while (*fifo) {
        count++;
        fifo++;
    }
    return count;
}
```

fifo.h

```
#pragma once

#define FIFO_SIZE 20

void fifo_init( char* fifo );

void fifo_enqueue( char* fifo, char c );

char fifo_dequeue( char* fifo );

char fifo_peek( char* fifo );

unsigned int fifo_get_size( char* fifo );
```

isr_utils.c

```
#include "isr_utils.h"

void set_vector( 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;

}
```

isr_utils.h

```
#pragma once

void set_vector( unsigned char __xdata * Address, void * Vector );
```

keypad.c

```
#include "keypad.h"
#include "aduc812.h"
#include "isr_utils.h"
#include "max.h"
#include "fifo.h"
#include "sound.h"
#include "notes.h"

#define SCAN_FREQ 100
#define SAMPLE_SIZE 10
#define REPEAT_KEY_MS 1000

static char read_queue[FIFO_SIZE];

__code static char key_chars[] = "123A"
                                 "456B"
                                 "789C"
                                 "*0=/";

__code static short key_tones[] = {
    C_NOTE_3, D_NOTE_3, F_NOTE_3, G_NOTE_3,
    C_SHARP_NOTE_3, D_SHARP_NOTE_3, F_SHARP_NOTE_3, G_SHARP_NOTE_3,
    C_NOTE_4, D_NOTE_4, F_NOTE_4, G_NOTE_4,
    C_SHARP_NOTE_4, D_SHARP_NOTE_4, F_SHARP_NOTE_4, G_SHARP_NOTE_4
};

static unsigned char key_pressed_samples[16];
static unsigned char key_is_pressed[16];
__pdata static unsigned short key_is_pressed_ms[16];

static unsigned char autoreload_h;
static unsigned char autoreload_l;

unsigned char scan_column(unsigned char column) {
    // set column as 0, for example 11110111
    write_max(0x0, ~ (0b1 << column));
    // return selected rows as bit array (lower 4 bits)
    return ~(read_max(0x0) >> 4);
}

void T0_ISR() __interrupt {
    unsigned int key_is_pressed_raw[16];
```

```
    unsigned int i, j, key_pressed_count;
    ET0 = 0;

    for (i = 0; i < 4; i++) {
        unsigned char rows = scan_column(i);
        for (j = 0; j < 4; j++) {
            key_is_pressed_raw[j*4+i] = (rows >> j) & 0x01;
        }
    }

    key_pressed_count = 0;
    for (i = 0; i < 16; i++) {
        if (key_is_pressed_raw[i] == 1) {
            if (key_pressed_samples[i] < SAMPLE_SIZE)  key_pressed_samples[i] += 1;
        } else {
            if (key_pressed_samples[i] > 0)            key_pressed_samples[i] -= 1;
        }
        if (key_is_pressed[i]) {
            key_is_pressed_ms[i] += 1000 / SCAN_FREQ;
            key_pressed_count++;
            if (key_pressed_samples[i] == 0)           key_is_pressed[i] = 0;
        } else {
            if (key_pressed_samples[i] == SAMPLE_SIZE) key_is_pressed[i] = 1;
            key_is_pressed_ms[i] = REPEAT_KEY_MS;
        }
    }

    if (key_pressed_count == 1 || key_pressed_count == 2) {
        sound_enable();
        for (i = 0; i < 16; i++) {
            if (key_is_pressed[i] && key_is_pressed_ms[i] >= REPEAT_KEY_MS) {
                fifo_enqueue(read_queue, key_chars[i]);
                sound_set_note(key_tones[i]);
                key_is_pressed_ms[i] = 0;
            }
        }
    } else {
        sound_disable();
    }

    TH0 = autoreload_h;
    TL0 = autoreload_l;
    ET0 = 1;
}

void keypad_init() {
    unsigned int i;
    unsigned short autoreload = 0xFFFF - MCLKIN / ( SCAN_FREQ * 12 );
    autoreload_h = (autoreload >> 8 ) & 0xFF;
    autoreload_l = (autoreload & 0xFF);
    for (i = 0; i < 16; i++) {
        key_is_pressed[i] = 0;
        key_is_pressed_ms[i] = 0;
        key_pressed_samples[i] = 0;
    }
    sound_init();
```

```
    fifo_init( read_queue );              // init read queue
    set_vector( 0x200B, (void *)T0_ISR ); // register interrupt handler
    EA   = 1;                             // enable interrupts
    ET0  = 1;                             // enable timer 0 interrupt
    TMOD = (TMOD & 0xF0) | 0x01;          // set timer 0 to mode 1 (16 bit counter)
    TR0  = 1;                             // enable timer 0
}

char keypad_read() {
    char res = '\0';
    ET0 = 0;
    if( fifo_get_size( read_queue ) != 0 ) {
        res = fifo_dequeue( read_queue );
    }
    ET0 = 1;
    return res;
}
```

keypad.h

```
#pragma once

void keypad_init();
char keypad_read();
```

main.c

```
#include "dip.h"
#include "app_echo.h"
#include "app_calc.h"

typedef enum {
    none,
    echo,
    calc
} app_t;

static app_t last_app;

app_t get_driver() {
    if (read_dip() == 0xFF){
        return calc;
    }
    return echo;
}

void main() {
    app_t last_app = none;
    while( 1 ) {
        app_t app = get_driver();

        if (app == echo) {
            if (app != last_app) {
                init_echo_app();
            }
            run_echo_app();
```

```
        }

        if (app == calc) {
            if (app != last_app) {
                init_calc_app();
            }
            run_calc_app();
        }

        last_app = app;
    }
}
```

notes.h

```
#pragma once

#define C_NOTE_0        16.4
#define C_SHARP_NOTE_0  17.3
#define D_NOTE_0        18.4
#define D_SHARP_NOTE_0  19.5
#define E_NOTE_0        20.6
#define F_NOTE_0        21.8
#define F_SHARP_NOTE_0  23.1
#define G_NOTE_0        24.5
#define G_SHARP_NOTE_0  26
#define A_NOTE_0        27.5
#define A_SHARP_NOTE_0  29.1
#define B_NOTE_0        30.9

#define C_NOTE_3        130.81
#define C_SHARP_NOTE_3  138.59
#define D_NOTE_3        146.83
#define D_SHARP_NOTE_3  155.56
#define E_NOTE_3        164.81
#define F_NOTE_3        174.61
#define F_SHARP_NOTE_3  185.00
#define G_NOTE_3        196.00
#define G_SHARP_NOTE_3  207.65
#define A_NOTE_3        220.00
#define A_SHARP_NOTE_3  233.08
#define B_NOTE_3        246.94

#define C_NOTE_4        261.63
#define C_SHARP_NOTE_4  277.18
#define D_NOTE_4        293.66
#define D_SHARP_NOTE_4  311.13
#define E_NOTE_4        329.63
#define F_NOTE_4        349.23
#define F_SHARP_NOTE_4  369.99
#define G_NOTE_4        392.00
#define G_SHARP_NOTE_4  415.30
#define A_NOTE_4        440.00
#define A_SHARP_NOTE_4  466.16
#define B_NOTE_4        493.88

```

```
#define C_NOTE_5        523.25
#define C_SHARP_NOTE_5  554.37
#define D_NOTE_5        587.33
#define D_SHARP_NOTE_5  622.25
#define E_NOTE_5        659.25
#define F_NOTE_5        698.46
#define F_SHARP_NOTE_5  739.99
#define G_NOTE_5        783.99
#define G_SHARP_NOTE_5  830.61
#define A_NOTE_5        880
#define A_SHARP_NOTE_5  932.33
#define B_NOTE_5        987.77
```

sound.c

```
#include "sound.h"
#include "max.h"
#include "aduc812.h"
#include "isr_utils.h"

#define DIAPHRAGM_MIN 0
#define DIAPHRAGM_MAX 0b00011100

static int diaphragm_is_min;

void move_diaphragm(unsigned char level) {
    write_max(ENA, level);
}

void T2_ISR() __interrupt {
    if( diaphragm_is_min ) {
        move_diaphragm(DIAPHRAGM_MAX);
        diaphragm_is_min = 0;
    } else {
        move_diaphragm(DIAPHRAGM_MIN);
        diaphragm_is_min = 1;
    }
    TF2 = 0;
}

void sound_set_note( int hz ) {
    unsigned short autoreload = 0xFFFF - MCLKIN / ( hz * 2 * 12 );
    RCAP2H = (autoreload >> 8 ) & 0xFF;
    RCAP2L = (autoreload & 0xFF);
}

void sound_init(){
    diaphragm_is_min = 0;
    set_vector( 0x202B, (void *)T2_ISR ); // register interrupt handler
    EA     = 1;                           // enable interrupts
    ET2    = 1;                           // enable timer 2 interrupt
    PT2    = 1;                           // set timer 2 high priority
    T2CON  = 0x00;                        // reset timer 2 settings
    RCAP2L = 0;                           // reset autoreload registers
    RCAP2H = 0;                           // reset autoreload registers
}
```

```
42
43  void sound_enable() {
44      TR2    = 1;                          // enable timer 2
45  }
46
47  void sound_disable() {
48      TR2    = 0;                          // disable timer 2
49  }
```

sound.h

```
1   #pragma once
2
3   void sound_init();
4
5   // enable/disable all sound effects
6   void sound_enable();
7   void sound_disable();
8
9   // Set sound frequency
10  void sound_set_note( int hz );
```

uart_async.c

```
1   #include "uart_async.h"
2   #include "uart_blocking.h"
3   #include "isr_utils.h"
4   #include "fifo.h"
5   #include "aduc812.h"
6
7   static char read_queue[FIFO_SIZE];
8   static char write_queue[FIFO_SIZE];
9
10  void SIO_ISR() __interrupt {
11      ES = 0;
12      if(TI) {
13          TI = 0;
14          // remove transmitted byte from queue
15          fifo_dequeue( write_queue );
16          // transmit next byte in queue, if any
17          if( fifo_get_size( write_queue ) > 0 ){
18              // start transmitting byte
19              // but keep it in queue until transmission is complete
20              SBUF = fifo_peek( write_queue );
21          }
22      }
23
24      if(RI) {
25          RI = 0;
26          fifo_enqueue( read_queue, SBUF );
27      }
28      ES = 1;
29  }
30
31  void uart_init_async() {
32      fifo_init( read_queue );               // init read queue
```

```
    fifo_init( write_queue );              // init write queue
    uart_init_blocking();                  // init core uart features
    set_vector( 0x2023, (void *)SIO_ISR ); // register interrupt handler
    EA = 1;                                // enable interrupts
    ES = 1;                                // enable uart interrupt
}

void uart_write_async( char c ) {
    int first_write = 0;
    ES = 0;
    if( fifo_get_size( write_queue ) == 0 ) {
        // enqueue dummy byte which will be removed
        fifo_enqueue( write_queue, '?' );
        first_write = 1;
    }
    fifo_enqueue( write_queue, c );
    ES = 1;
    if( first_write == 1 ) {
        TI = 1; // manually initiate transmission
    }
}

void uart_write_str_async( char * str ) {
    while( *str ) uart_write_async( *str++ );
}

char uart_read_async() {
    char res = '\0';
    ES = 0;
    if( fifo_get_size( read_queue ) != 0 ) {
        res = fifo_dequeue( read_queue );
    }
    ES = 1;
    return res;
}
```

uart_async.h

```
#pragma once

void uart_init_async();
void uart_write_async( char c );
void uart_write_str_async( char * str );
char uart_read_async();
```

uart_blocking.c

```
#include "uart_blocking.h"
#include "aduc812.h"

void uart_init_blocking() {
    TMOD = (TMOD & 0x0F) | 0x20; // set timer 1 to mode 2 (8 bit autoreload)
    PCON = PCON & 0x7F; // disable double UART baud rate boost (2400 baud rate)
    SCON = 0x50; // set UART to mode 1 and enable reception
    ES   = 0;    // disable uart interrupt
    ET1  = 0;    // disable timer 1 interrupt
```

```
    TH1  = 0xF4; // autorealod timer 1 with preset value (2400 baud rate)
    TR1  = 1;    // enable timer 1
}

void uart_write_blocking( char c ) {
    TI = 0;
    SBUF = c;
    while( !TI );
}

char uart_read_blocking() {
    RI = 0;
    while( !RI );
    return SBUF;
}
```

uart_blocking.h

```
#pragma once

void uart_init_blocking();
void uart_write_blocking( char c );
char uart_read_blocking();
```

Вывод

В результате выполнения лабораторной работы был разработан драйвер клавиатуры для учебно-лабораторного стенда SDK-1.1 и написана тестовая программа для разработанного драйвера.