

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 Р3410

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.

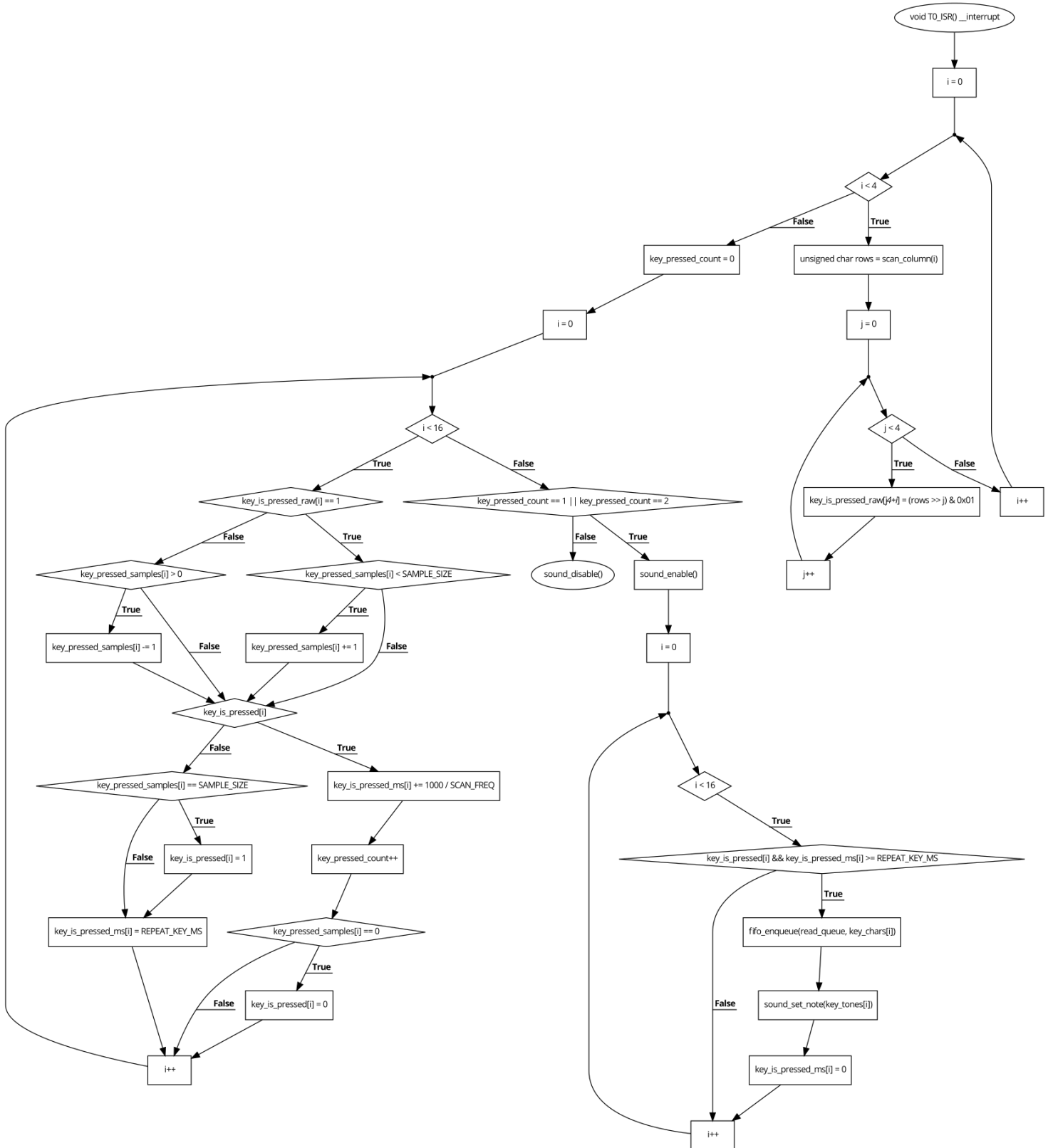


Рис. 1: Блок схема функции сканирования нажатия кнопки на клавиатуре

Листинги

app_calc.c

```
1  #include "app_calc.h"
2  #include "uart_async.h"
3  #include "keypad.h"
4  #include "util.h"
5
6  typedef enum {
7      init,
8      read_dividend,
9      read_divisor,
10     print_result,
11     print_error
12 } calc_state_t;
13
14 static calc_state_t calc_state;
15 static unsigned char length;
16 static int dividend;
17 static int divisor;
18
19 void init_calc_app() {
20     uart_init_async();
21     keypad_init();
22     calc_state = init;
23 }
24
25 void run_calc_app() {
26     char c = keypad_read();
27     if ( c ) uart_write_async(c);
28     switch (calc_state) {
29         case init:
30             calc_state = read_dividend;
31             length = 0;
32             dividend = 0;
33             divisor = 0;
34             break;
35         case read_dividend:
36             if ( c >= '0' && c <= '9' ) {
37                 if (length == 2) {
38                     calc_state = print_error;
39                 }
40                 if (length == 1) {
41                     dividend *= 10;
42                 }
43                 dividend += c - '0';
44                 length++;
45                 break;
46             }
47             if ( c == '/' ) {
48                 if (length == 0){
49                     calc_state = print_error;
50                 }
51                 else {
52                     length = 0;
```

```

53         calc_state = read_divisor;
54     }
55     break;
56 }
57 if ( c == 0 ) {
58     break;
59 }
60 calc_state = print_error;
61 break;
62 case read_divisor:
63     if ( c >= '0' && c <= '9' ) {
64         if (length == 2) {
65             calc_state = print_error;
66         }
67         if (length == 1) {
68             divisor *= 10;
69         }
70         divisor += c - '0';
71         length++;
72         break;
73     }
74     if ( c == '=' ) {
75         if (length == 0)
76             calc_state = print_error;
77         else
78             calc_state = print_result;
79         break;
80     }
81     if ( c == 0 ) {
82         break;
83     }
84     calc_state = print_error;
85     break;
86 case print_result:
87     if (divisor == 0){
88         uart_write_str_async("\r\nCannot divide by 0!\r\n");
89     } else {
90         char str[3];
91         itoa(divident / divisor, str);
92         uart_write_str_async(str);
93         uart_write_str_async("\r\n");
94     }
95     calc_state = init;
96     break;
97 case print_error:
98     uart_write_str_async("\r\nInput error!\r\n");
99     calc_state = init;
100    break;
101 }
102 }

```

app_calc.h

```

1  #pragma once
2
3  void init_calc_app();

```

```
4 void run_calc_app();
```

app_echo.c

```
1 #include "app_echo.h"
2 #include "uart_async.h"
3 #include "keypad.h"
4
5 void init_echo_app() {
6     uart_init_async();
7     keypad_init();
8 }
9
10 void run_echo_app() {
11     char c = keypad_read();
12     if( c ) {
13         uart_write_async( c );
14         uart_write_str_async( "\r\n" );
15     }
16 }
```

app_echo.h

```
1 #pragma once
2
3 void init_echo_app();
4 void run_echo_app();
```

dip.c

```
1 #include "max.h"
2 #include "dip.h"
3
4 unsigned char read_dip() {
5     return read_max( EXT_L0 );
6 }
```

dip.h

```
1 #pragma once
2
3 unsigned char read_dip();
```

fifo.c

```
1 #include "fifo.h"
2
3 void fifo_init( char* fifo ) {
4     unsigned int i;
5     for (i = 0; i < FIFO_SIZE; i++) {
6         fifo[i]='\0';
7     }
8 }
9
10 void fifo_enqueue( char* fifo, char c ) {
11     unsigned int count = 0;
12     while (*fifo) {
```

```

13         count++;
14         fifo++;
15     }
16     if (count < FIFO_SIZE - 1) {
17         *fifo = c;
18     }
19 }
20
21 char fifo_dequeue( char* fifo ) {
22     unsigned int i;
23     char c = fifo[0];
24     for (i = 0; i < FIFO_SIZE - 1; i++) {
25         fifo[i] = fifo[i+1];
26     }
27     return c;
28 }
29
30 char fifo_peek( char* fifo ) {
31     return fifo[0];
32 }
33
34 unsigned int fifo_get_size( char* fifo ) {
35     unsigned int count = 0;
36     while (*fifo) {
37         count++;
38         fifo++;
39     }
40     return count;
41 }

```

fifo.h

```

1  #pragma once
2
3  #define FIFO_SIZE 20
4
5  void fifo_init( char* fifo );
6
7  void fifo_enqueue( char* fifo, char c );
8
9  char fifo_dequeue( char* fifo );
10
11 char fifo_peek( char* fifo );
12
13 unsigned int fifo_get_size( char* fifo );

```

isr_utils.c

```

1  #include "isr_utils.h"
2
3  void set_vector( unsigned char __xdata * Address, void * Vector ) {
4      unsigned char __xdata * TmpVector;
5
6      *Address = 0x02;
7
8      TmpVector = (unsigned char __xdata *) (Address + 1);

```

```

9      *TmpVector = (unsigned char) ((unsigned short)Vector >> 8);
10     ++TmpVector;
11     *TmpVector = (unsigned char) Vector;
12
13 }

```

isr_utils.h

```

1  #pragma once
2
3  void set_vector( unsigned char __xdata * Address, void * Vector );

```

keypad.c

```

1  #include "keypad.h"
2  #include "aduc812.h"
3  #include "isr_utils.h"
4  #include "max.h"
5  #include "fifo.h"
6  #include "sound.h"
7  #include "notes.h"
8
9  #define SCAN_FREQ 100
10 #define SAMPLE_SIZE 10
11 #define REPEAT_KEY_MS 1000
12
13 static char read_queue[FIFO_SIZE];
14
15 __code static char key_chars[] = "123A"
16                                "456B"
17                                "789C"
18                                "**0=/";
19
20 __code static short key_tones[] = {
21     C_NOTE_3, D_NOTE_3, F_NOTE_3, G_NOTE_3,
22     C_SHARP_NOTE_3, D_SHARP_NOTE_3, F_SHARP_NOTE_3, G_SHARP_NOTE_3,
23     C_NOTE_4, D_NOTE_4, F_NOTE_4, G_NOTE_4,
24     C_SHARP_NOTE_4, D_SHARP_NOTE_4, F_SHARP_NOTE_4, G_SHARP_NOTE_4
25 };
26
27 static unsigned char key_pressed_samples[16];
28 static unsigned char key_is_pressed[16];
29 __pdata static unsigned short key_is_pressed_ms[16];
30
31 static unsigned char autoreload_h;
32 static unsigned char autoreload_l;
33
34 unsigned char scan_column(unsigned char column) {
35     // set column as 0, for example 11110111
36     write_max(0x0, ~(0b1 << column));
37     // return selected rows as bit array (lower 4 bits)
38     return ~(read_max(0x0) >> 4);
39 }
40
41 void T0_ISR() __interrupt {
42     unsigned int key_is_pressed_raw[16];

```



```

43 unsigned int i, j, key_pressed_count;
44 ET0 = 0;
45
46 for (i = 0; i < 4; i++) {
47     unsigned char rows = scan_column(i);
48     for (j = 0; j < 4; j++) {
49         key_is_pressed_raw[j*4+i] = (rows >> j) & 0x01;
50     }
51 }
52
53 key_pressed_count = 0;
54 for (i = 0; i < 16; i++) {
55     if (key_is_pressed_raw[i] == 1) {
56         if (key_pressed_samples[i] < SAMPLE_SIZE) key_pressed_samples[i] += 1;
57     } else {
58         if (key_pressed_samples[i] > 0) key_pressed_samples[i] -= 1;
59     }
60     if (key_is_pressed[i]) {
61         key_is_pressed_ms[i] += 1000 / SCAN_FREQ;
62         key_pressed_count++;
63         if (key_pressed_samples[i] == 0) key_is_pressed[i] = 0;
64     } else {
65         if (key_pressed_samples[i] == SAMPLE_SIZE) key_is_pressed[i] = 1;
66         key_is_pressed_ms[i] = REPEAT_KEY_MS;
67     }
68 }
69
70 if (key_pressed_count == 1 || key_pressed_count == 2) {
71     sound_enable();
72     for (i = 0; i < 16; i++) {
73         if (key_is_pressed[i] && key_is_pressed_ms[i] >= REPEAT_KEY_MS) {
74             fifo_enqueue(read_queue, key_chars[i]);
75             sound_set_note(key_tones[i]);
76             key_is_pressed_ms[i] = 0;
77         }
78     }
79 } else {
80     sound_disable();
81 }
82
83 TH0 = autoreload_h;
84 TL0 = autoreload_l;
85 ET0 = 1;
86 }
87
88 void keypad_init() {
89     unsigned int i;
90     unsigned short autoreload = 0xFFFF - MCLKIN / ( SCAN_FREQ * 12 );
91     autoreload_h = (autoreload >> 8 ) & 0xFF;
92     autoreload_l = (autoreload & 0xFF);
93     for (i = 0; i < 16; i++) {
94         key_is_pressed[i] = 0;
95         key_is_pressed_ms[i] = 0;
96         key_pressed_samples[i] = 0;
97     }
98     sound_init();

```

```

99     fifo_init( read_queue );           // init read queue
100     set_vector( 0x200B, (void *)T0_ISR ); // register interrupt handler
101     EA  = 1;                           // enable interrupts
102     ET0 = 1;                           // enable timer 0 interrupt
103     TMOD = (TMOD & 0xF0) | 0x01;       // set timer 0 to mode 1 (16 bit counter)
104     TR0  = 1;                           // enable timer 0
105 }
106
107 char keypad_read() {
108     char res = '\0';
109     ET0 = 0;
110     if( fifo_get_size( read_queue ) != 0 ) {
111         res = fifo_dequeue( read_queue );
112     }
113     ET0 = 1;
114     return res;
115 }

```

keypad.h

```

1  #pragma once
2
3  void keypad_init();
4  char keypad_read();

```

main.c

```

1  #include "dip.h"
2  #include "app_echo.h"
3  #include "app_calc.h"
4
5  typedef enum {
6      none,
7      echo,
8      calc
9  } app_t;
10
11  static app_t last_app;
12
13  app_t get_driver() {
14      if (read_dip() == 0xFF){
15          return calc;
16      }
17      return echo;
18  }
19
20  void main() {
21      app_t last_app = none;
22      while( 1 ) {
23          app_t app = get_driver();
24
25          if (app == echo) {
26              if (app != last_app) {
27                  init_echo_app();
28              }
29              run_echo_app();

```

```

30     }
31
32     if (app == calc) {
33         if (app != last_app) {
34             init_calc_app();
35         }
36         run_calc_app();
37     }
38
39     last_app = app;
40 }
41 }

```

notes.h

```

1  #pragma once
2
3  #define C_NOTE_0      16.4
4  #define C_SHARP_NOTE_0 17.3
5  #define D_NOTE_0      18.4
6  #define D_SHARP_NOTE_0 19.5
7  #define E_NOTE_0      20.6
8  #define F_NOTE_0      21.8
9  #define F_SHARP_NOTE_0 23.1
10 #define G_NOTE_0      24.5
11 #define G_SHARP_NOTE_0 26
12 #define A_NOTE_0      27.5
13 #define A_SHARP_NOTE_0 29.1
14 #define B_NOTE_0      30.9
15
16 #define C_NOTE_3      130.81
17 #define C_SHARP_NOTE_3 138.59
18 #define D_NOTE_3      146.83
19 #define D_SHARP_NOTE_3 155.56
20 #define E_NOTE_3      164.81
21 #define F_NOTE_3      174.61
22 #define F_SHARP_NOTE_3 185.00
23 #define G_NOTE_3      196.00
24 #define G_SHARP_NOTE_3 207.65
25 #define A_NOTE_3      220.00
26 #define A_SHARP_NOTE_3 233.08
27 #define B_NOTE_3      246.94
28
29 #define C_NOTE_4      261.63
30 #define C_SHARP_NOTE_4 277.18
31 #define D_NOTE_4      293.66
32 #define D_SHARP_NOTE_4 311.13
33 #define E_NOTE_4      329.63
34 #define F_NOTE_4      349.23
35 #define F_SHARP_NOTE_4 369.99
36 #define G_NOTE_4      392.00
37 #define G_SHARP_NOTE_4 415.30
38 #define A_NOTE_4      440.00
39 #define A_SHARP_NOTE_4 466.16
40 #define B_NOTE_4      493.88
41

```

```

42 #define C_NOTE_5      523.25
43 #define C_SHARP_NOTE_5 554.37
44 #define D_NOTE_5      587.33
45 #define D_SHARP_NOTE_5 622.25
46 #define E_NOTE_5      659.25
47 #define F_NOTE_5      698.46
48 #define F_SHARP_NOTE_5 739.99
49 #define G_NOTE_5      783.99
50 #define G_SHARP_NOTE_5 830.61
51 #define A_NOTE_5      880
52 #define A_SHARP_NOTE_5 932.33
53 #define B_NOTE_5      987.77

```

sound.c

```

1  #include "sound.h"
2  #include "max.h"
3  #include "aduc812.h"
4  #include "isr_utils.h"
5
6  #define DIAPHRAGM_MIN 0
7  #define DIAPHRAGM_MAX 0b00011100
8
9  static int diaphragm_is_min;
10
11 void move_diaphragm(unsigned char level) {
12     write_max(ENA, level);
13 }
14
15 void T2_ISR() __interrupt {
16     if( diaphragm_is_min ) {
17         move_diaphragm(DIAPHRAGM_MAX);
18         diaphragm_is_min = 0;
19     } else {
20         move_diaphragm(DIAPHRAGM_MIN);
21         diaphragm_is_min = 1;
22     }
23     TF2 = 0;
24 }
25
26 void sound_set_note( int hz ) {
27     unsigned short autoreload = 0xFFFF - MCLKIN / ( hz * 2 * 12 );
28     RCAP2H = (autoreload >> 8 ) & 0xFF;
29     RCAP2L = (autoreload & 0xFF);
30 }
31
32 void sound_init(){
33     diaphragm_is_min = 0;
34     set_vector( 0x202B, (void *)T2_ISR ); // register interrupt handler
35     EA      = 1;                          // enable interrupts
36     ET2     = 1;                          // enable timer 2 interrupt
37     PT2     = 1;                          // set timer 2 high priority
38     T2CON   = 0x00;                       // reset timer 2 settings
39     RCAP2L  = 0;                          // reset autoreload registers
40     RCAP2H  = 0;                          // reset autoreload registers
41 }

```

```

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

```

```

33     fifo_init( write_queue );           // init write queue
34     uart_init_blocking();               // init core uart features
35     set_vector( 0x2023, (void *)SIO_ISR ); // register interrupt handler
36     EA = 1;                             // enable interrupts
37     ES = 1;                             // enable uart interrupt
38 }
39
40 void uart_write_async( char c ) {
41     int first_write = 0;
42     ES = 0;
43     if( fifo_get_size( write_queue ) == 0 ) {
44         // enqueue dummy byte which will be removed
45         fifo_enqueue( write_queue, '?' );
46         first_write = 1;
47     }
48     fifo_enqueue( write_queue, c );
49     ES = 1;
50     if( first_write == 1 ) {
51         TI = 1; // manually initiate transmission
52     }
53 }
54
55 void uart_write_str_async( char * str ) {
56     while( *str ) uart_write_async( *str++ );
57 }
58
59 char uart_read_async() {
60     char res = '\0';
61     ES = 0;
62     if( fifo_get_size( read_queue ) != 0 ) {
63         res = fifo_dequeue( read_queue );
64     }
65     ES = 1;
66     return res;
67 }

```

uart_async.h

```

1  #pragma once
2
3  void uart_init_async();
4  void uart_write_async( char c );
5  void uart_write_str_async( char * str );
6  char uart_read_async();

```

uart_blocking.c

```

1  #include "uart_blocking.h"
2  #include "aduc812.h"
3
4  void uart_init_blocking() {
5      TMOD = (TMOD & 0x0F) | 0x20; // set timer 1 to mode 2 (8 bit autoreload)
6      PCON = PCON & 0x7F; // disable double UART baud rate boost (2400 baud rate)
7      SCON = 0x50; // set UART to mode 1 and enable reception
8      ES = 0; // disable uart interrupt
9      ET1 = 0; // disable timer 1 interrupt

```

```

10     TH1  = 0xF4; // autoreload timer 1 with preset value (2400 baud rate)
11     TR1  = 1;    // enable timer 1
12 }
13
14 void uart_write_blocking( char c ) {
15     TI = 0;
16     SBUF = c;
17     while( !TI );
18 }
19
20 char uart_read_blocking() {
21     RI = 0;
22     while( !RI );
23     return SBUF;
24 }

```

uart_blocking.h

```

1  #pragma once
2
3  void uart_init_blocking();
4  void uart_write_blocking( char c );
5  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.