Университет ИТМО

Факультет программной инженерии
и компьютерной техники

Кафедра вычислительной техники

Информационно-управляющие системы
Лабораторная работа №3
Вариант №8

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

Преподаватель:
Ключев А. О.

Санкт-Петербург
2017 г.

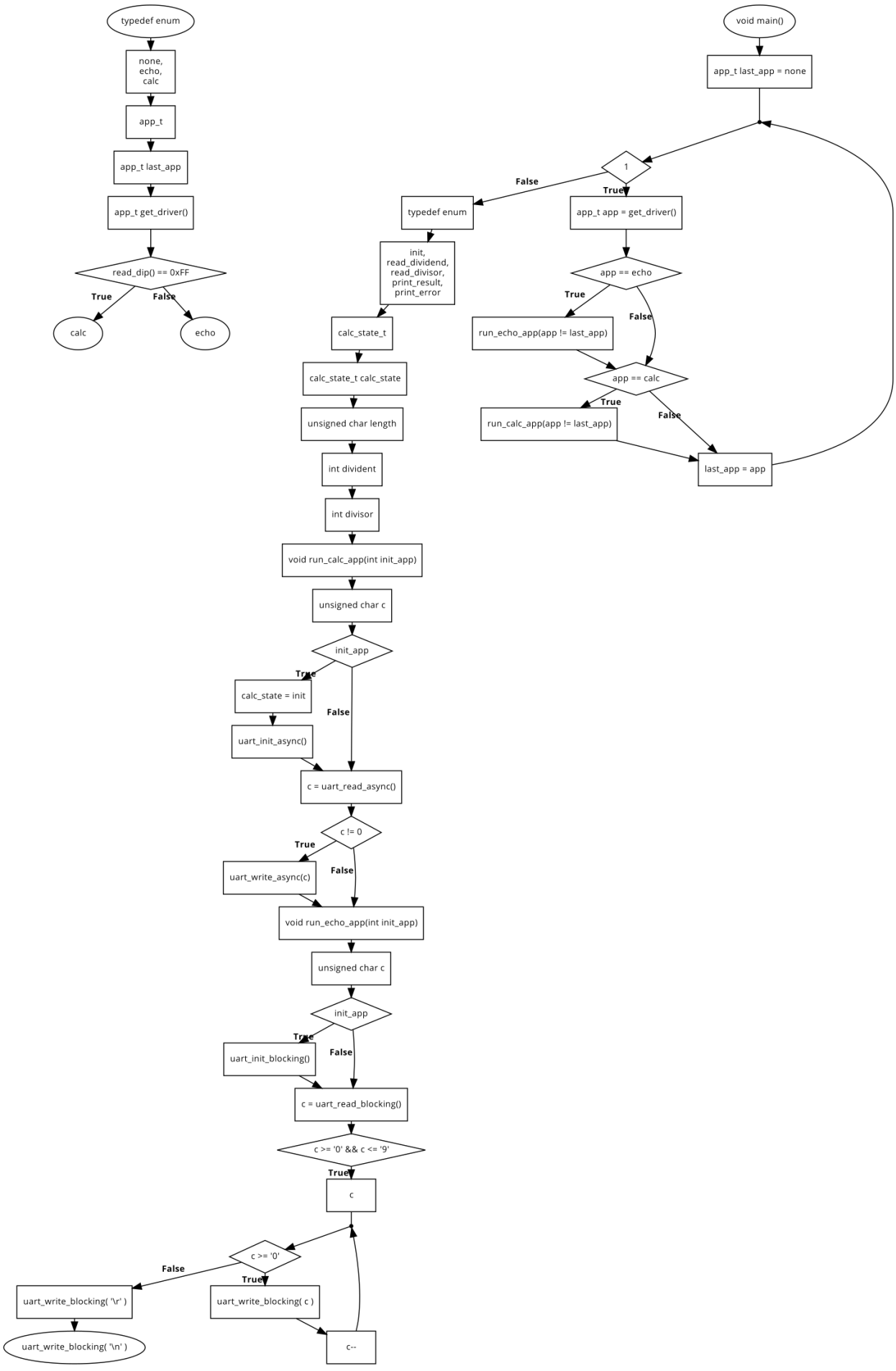
Описание задания

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:

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 010 до 9910 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).


typedef enum
none,
echo,
calc
app_t
app_t last_app
app_t get_driver()
read_dip() == 0xFF
True
False
calc
echo
void main()
app_t last_app = none
1
False
True
typedef enum
app_t app = get_driver()
init,
read_dividend,
read_divisor,
print_result,
print_error
app == echo
True
False
calc_state_t
run_echo_app(app != last_app)
calc_state_t calc_state
app == calc
True
False
unsigned char length
run_calc_app(app != last_app)
last_app = app
int divident
int divisor
void run_calc_app(int init_app)
unsigned char c
init_app
True
False
calc_state = init
uart_init_async()
c = uart_read_async()
c != 0
True
False
uart_write_async(c)
void run_echo_app(int init_app)
unsigned char c
init_app
True
False
uart_init_blocking()
c = uart_read_blocking()
c >= '0' && c <= '9'
True
c
c >= '0'
False
True
uart_write_blocking('\r')
uart_write_blocking(c)
uart_write_blocking('\n')
c--

Листинги

main.c

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

typedef enum {
    none,
    echo,
    calc
} app_t;

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

app_calc.c

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

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

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

```

```
void run_calc_app(int init_app) {
    unsigned char c;
    if (init_app) {
        calc_state = init;
        uart_init_async();
    }
    c = uart_read_async();
    if (c != 0)
        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' ) {
                leds (0x00);
                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:
            leds( 0xFF );
            uart_write_str_async("\r\nInput error!\r\n");
            calc_state = init;
            break;
    }
}
```

app_calc.h

```
#pragma once

void run_calc_app(int init_app);
```

app_echo.c

```
#include "app_echo.h"
#include "uart_blocking.h"

void run_echo_app(int init_app) {
    unsigned char c;
    if (init_app) {
        uart_init_blocking();
    }
    c = uart_read_blocking();
    if( c >= '0' && c <= '9' ) {
        for( c; c >= '0'; c-- ) {
            uart_write_blocking( c );
        }
        uart_write_blocking( '\r' );
        uart_write_blocking( '\n' );
    }
}
```

app_echo.h

```
#pragma once

void run_echo_app(int init_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 <stdlib.h>
#include "fifo.h"

void fifo_init( fifo_t* fifo ) {
    fifo->front = NULL;
    fifo->back = NULL;
}

void fifo_enqueue( fifo_t* fifo, unsigned char val ) {
    fifo_node_t* node = malloc( sizeof(fifo_node_t) );

    node->val = val;
    node->next = NULL;

    if( fifo->front == NULL ) {
        fifo->front = node;
    } else {
        fifo->back->next = node;
    }

    fifo->back = node;
}

unsigned char fifo_dequeue( fifo_t* fifo ) {
    unsigned char ret;

    fifo_node_t* front = fifo->front;

    if( front == NULL ) return '!';

    ret = front->val;
    fifo->front = front->next;

    free( front );
```

```
    return ret;
}

unsigned char fifo_peek( fifo_t* fifo ) {
    if( fifo->front != NULL ) {
        return fifo->front->val;
    }
    return '!';
}

unsigned int fifo_get_size( fifo_t* fifo ) {
    unsigned int i = 0;
    fifo_node_t* tmp = fifo->front;
    if( tmp == NULL ) return 0;

    i++;
    while( tmp->next != NULL ) {
        i++;
        tmp = tmp->next;
    }

    return i;
}
```

fifo.h

```
#pragma once

typedef struct fifo_node_t {
    unsigned char val;
    struct fifo_node_t* next;
} fifo_node_t;

typedef struct {
    fifo_node_t* front;
    fifo_node_t* back;
} fifo_t;

void fifo_init( fifo_t* fifo );

void fifo_enqueue( fifo_t* fifo, unsigned char val );

unsigned char fifo_dequeue( fifo_t* fifo );

unsigned char fifo_peek( fifo_t* fifo );

unsigned int fifo_get_size( fifo_t* 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 );
```

list

```
app_calc.c
app_calc.h
app_echo.c
app_echo.h
dip.c
dip.h
fifo.c
fifo.h
isr_utils.c
isr_utils.h
list
main.c
uart_async.c
uart_async.h
uart_blocking.c
uart_blocking.h
util.c
util.h
```

uart_async.c

```
#include "uart_async.h"
#include "uart_blocking.h"
#include "isr_utils.h"
#include "fifo.h"
#include "aduc812.h"

fifo_t read_queue;
fifo_t write_queue;

void SIO_ISR() __interrupt
{
    if(TI) {
        TI = 0;
        // remove transmitted byte from queue
        fifo_dequeue( &write_queue );
        // transmit next byte in queue, if any
        if( fifo_get_size( &write_queue ) > 0 ){
            // start transmitting byte
            // but keep it in queue until transmission is complete
```

```
            SBUF = fifo_peek( &write_queue );
        }
    }

    if(RI) {
        RI = 0;
        fifo_enqueue( &read_queue, SBUF );
    }
}


void uart_init_async() {
    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( unsigned 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 ) {
        // manually initiate transmission
        TI = 1;
    }
}

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

unsigned char uart_read_async() {
    unsigned 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( unsigned char c );
```

```
void uart_write_str_async( unsigned char * str );
unsigned 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( unsigned char c ) {
    TI = 0;
    SBUF = c;
    while( !TI );
}

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

uart_blocking.h

```
#pragma once

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

util.c

```
#include "util.h"
#include <string.h>

void reverse(char s[])
{
    int i, j;
    char c;

    for (i = 0, j = strlen(s)-1; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

void itoa(int n, char s[])
```

```
{
    int i, sign;

    if ((sign = n) < 0)
        n = -n;
    i = 0;
    do {
        s[i++] = n % 10 + '0';
    } while ((n /= 10) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

util.h

```
#pragma once

void itoa(int n, char s[]);
```

Вывод

В результате выполнения лабораторной работы были разработаны и драйверы последовательного канала для учебно-лабораторного стенда SDK-1.1 с использованием и без использования прерываний и тестовая программа для разработанных драйверов.