

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Кафедра вычислительной техники

Лабораторная работа № 3 по дисциплине
”Информационно-управляющие системы”

Вариант 4

Выполнили:

Айтуганов Д. А.

Чебыкин И. Б.

Группа: Р3401

Проверяющий: Пинкевич В. Ю.

Санкт-Петербург, 2017

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.

Скорость последовательного канала – 4800 бит/с.

1. 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', і (і) – іjk' и т.д.
2. 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.

Модель взаимодействия

Исходный код

```

#include "aduc812.h"
#include "async.h"
#include "sio.h"
#include "sync.h"
#include "system.h"

//Определяет, является ли символ десятичной цифрой
//Вход: проверяемый символ
//Выход: 0 - не является
// 1 - является
bit IsDigit(unsigned char symb) {
    if ((symb >= 48) && (symb <= 57)) {
        return 1;
    } else {
        return 0;
    }
}

//Преобразует символ русского алфавита в верхний регистр
//Вход: символ для преобразования
//Выход: нет
unsigned char RussianToUpper(unsigned char symb) {
    if ((symb >= 160) && (symb <= 175)) {
        symb -= 32;
    } else if ((symb >= 224) && (symb <= 239)) {
        symb -= 80;
    }
    return symb;
}

//Преобразует символ русского алфавита в нижний регистр
//Вход: символ для преобразования
//Выход: нет
unsigned char RussianToLower(unsigned char symb) {
    if ((symb >= 128) && (symb <= 143)) {
        symb += 32;
    } else if ((symb >= 144) && (symb <= 159)) {
        symb += 80;
    }
    return symb;
}

//Определяет, является ли символ русской буквой в верхнем регистре
//Вход: проверяемый символ
//Выход: 0 - не является
// 1 - является
bit IsRussianBig(unsigned char symb) {
    if ((symb >= 128) && (symb <= 159)) {
        return 1;
    } else {
        return 0;
    }
}

//Определяет, является ли символ русской буквой в нижнем регистре
//Вход: проверяемый символ
//Выход: 0 - не является
// 1 - является
bit IsRussianSmall(unsigned char symb) {
    if (((symb >= 160) && (symb <= 175)) || ((symb >= 224) && (symb <= 239))) {
        return 1;
    } else {
        return 0;
    }
}

//Определяет является ли символ буквой латинского алфавита
//Вход: проверяемый символ

```

```

//Выход: 0 - не является
// 1 - является
bit IsEnglish(unsigned char symb) {
if (((symb >= 65) && (symb <= 90)) || ((symb >= 97) && (symb <= 122))) {
return 1;
} else {
return 0;
}
}

#define ERR_OUT_OF_RANGE "\n\rnumber not 0-99\n\r\0"
#define ERR_INVALID_CHAR "\n\rinvalid operator\n\r\0"
#define READ_OK 0
#define READ_OUT_OF_RANGE_ERROR 1
#define READ_INVALID_CHAR_ERROR 2

//Функция последовательно считывает до 2х символов из последовательного канала,
//пока не встретится оператор и преобразует их в 2х значное десятичное число
//Вход: signed char* num - указатель на число (используется для возврата
//результата)
// signed char operator - оператор
//Выход: результат выполнения операции
signed char ReadNumber(signed char *num, signed char operator) {
signed char c = 0, i = 0;

*num = 0;
for (i = 0; i < 3; i++) {
c = 0;
while (!c) {
if (GetDIP() != 128)
break;
c = ReadUART();
}
if (GetDIP() != 128)
break;
WriteUART(c);
if (c == operator) {
return READ_OK;
}
if (!IsDigit(c)) {
return READ_INVALID_CHAR_ERROR;
}
if (i > 1) {
return READ_OUT_OF_RANGE_ERROR;
}
*num *= 10;
*num += c & 0x0F;
}
return READ_OK;
}

//Функция преобразует десятичную строку в массив ASCII - символов,
//и последовательно отправляет элементы массива в последовательный канал
//Вход: unsigned char res - десятичное число
//Выход: нет
void ResultOutput(unsigned char res) {
unsigned char rr[4], c;
char i = 0;
// bit isNegative = 0;
// if ((res & 0x80) == 0x80)
//{
// isNegative = 1;
// res = ~res;
// res += 1;
//}

for (i = 0; i < 3; i++) {
c = res % 10;
c |= 0x30;
rr[i] = c;
res /= 10;
}
}

```

```

if (res == 0)
break;
}
// if (isNegative)
//{
// i++;
// rr[i] = '-';
//}
for (; i >= 0; i--)
WriteUART(rr[i]);
}

//Вычитатель
void Calc(void) {
signed char a = 0, b = 0, res = 0, last_operation = 0;
if (GetDIP() != 128)
return;
last_operation = ReadNumber(&a, '-');
if (last_operation != READ_OK) {
// if(last_operation ==
// READ_INVALID_CHAR_ERROR){APIString(ERR_INVALID_CHAR); return;}
// if(last_operation ==
// READ_OUT_OF_RANGE_ERROR){APIString(ERR_OUT_OF_RANGE); return;}
if (last_operation == READ_INVALID_CHAR_ERROR) {
SendString(ERR_INVALID_CHAR);
return;
}
if (last_operation == READ_OUT_OF_RANGE_ERROR) {
SendString(ERR_OUT_OF_RANGE);
return;
}
}
if (GetDIP() != 128)
return;

last_operation = ReadNumber(&b, '=');
if (last_operation != READ_OK) {
// if(last_operation ==
// READ_INVALID_CHAR_ERROR){APIString(ERR_INVALID_CHAR); return;}
// if(last_operation ==
// READ_OUT_OF_RANGE_ERROR){APIString(ERR_OUT_OF_RANGE); return;}
if (last_operation == READ_INVALID_CHAR_ERROR) {
SendString(ERR_INVALID_CHAR);
return;
}
if (last_operation == READ_OUT_OF_RANGE_ERROR) {
SendString(ERR_OUT_OF_RANGE);
return;
}
}
if (GetDIP() != 128)
return;

res = a - b;
ResultOutput(res);
WriteUART('\n');
WriteUART('\r');
}

void main(void) {
unsigned char c;
unsigned char i = 0;

init_sio(S4800);
while (1) {
if (GetDIP() == 128) {
ES = 1;
EA = 1;
while (1) {
Calc();
if (GetDIP() != 128) {

```

```

ES = 0;
break;
}
}
if (GetDIP() == 1) {
ES = 0;
EA = 0;
while (1) {
if (rsiostat()) {
c = rsio();
wsio(c);
wsio('\n');
if (GetDIP() != 1)
break;
// new_c = Convert(c);
if (IsRussianBig(c)) {
wsio(c);
c = RussianToLower(c);
} else if (IsRussianSmall(c)) {
wsio(RussianToUpper(c));
} else {
continue;
}
c++;
for (i = 0; i < 5; i++, c++) {
if (c == 176)
c = 224;
if (c > 239)
break;
wsio(c);
}
wsio('\n');
if (GetDIP() != 1)
break;
}
}
}
}

#include "aduc812.h"

#define FIFOSize 16

void SetVector(unsigned char xdata *Address, void *Vector);
void SIO_ISR(void) __interrupt(4);

struct FIFOb {
unsigned char buf[FIFOSize];
char RP;
char WP;
};

struct FIFOb wFIFO, rFIFO;
bit TRANSFER_NOW; //Флаг для разрешения проблемы начальной передачи

//Функция, устанавливающая вектор прерывания в пользовательской таблице
//прерываний Вход: Vector - адрес обработчика прерываний Address - вектор
// пользовательской таблицы прерываний
//Выход: 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;
}

/*-----

```

```

init_sio()
-----
Инициализирует последовательный канал на заданной скорости. Использует таймер 1

Вход:      char speed - скорость. Задается константами, описанными в
            заголовочном файле sio.h
            bit sdouble - дублирование скорости: 0 - не дублировать скорость,
            заданную аргументом speed; 1 - дублировать.

Выход:      нет
Результат:  нет
----- */
void init_sio(unsigned char speed) {
    TH1 = speed;
    TMOD |= 0x20; //Таймер 1 будет работать в режиме autoreload
    TCON |= 0x40; //Запуск таймера 1
    PCON |= 0x00; // SMOD = 0
    SCON = 0x50; //Настройки последовательного канала: Режим 1(8 бит данных,
                //асинхронный, переменная скорость)
    ES = 0; //Запрещение прерываний от приемопередатчика
    wFIFO.RP = wFIFO.WP = rFIFO.RP = rFIFO.WP = 0;
    TRANSFER_NOW = 0;
    SetVector(0x2023, (void *)SIO_ISR);
}

//Функция записи элемента в буфер
//Вход: FIFOb* a - буфер
//Выход: 1 - успешно записан
// 0 - буфер полон
bit PushFIFO(struct FIFOb *a, unsigned char c) {
    if (!(a->RP == 0 && a->WP == FIFOSize - 1) ||
        ((a->RP - a->WP) == 1))) //если буфер не полон
    {
        a->buf[a->WP] = c; //записываем в буфер элемент с индексом WP
        if (++a->WP > FIFOSize - 1)
            a->WP = 0; //проверка выхода указателя за границы буфера
        return 1;
    } else
        return 0;
}

//Функция извлечения элемента из буфера
//Вход: FIFOb* a - буфер
//Выход: элемент буфера при удачном извлечении
// 0 - если буфер пуст
unsigned char PopFIFO(struct FIFOb *a) {
    unsigned char c;
    if (a->WP == a->RP)
        return 0; //если буфер пуст, возвращаем 0
    c = a->buf[a->RP]; //извлекаем элемент с индексом RP
    if (++(a->RP) > FIFOSize - 1)
        a->RP = 0; //проверка выхода указателя за границы буфера
    return c;
}

//Обработчик прерывания от последовательного канала
//Вход: нет
//Выход: нет
void SIO_ISR(void) __interrupt(4) {
    unsigned char c;
    if (TI) {
        c = PopFIFO(&wFIFO);
        TRANSFER_NOW = 1;
        if (c) { //если буфер непуст
            SBUF = c;
            TI = 0;
        } else
            TRANSFER_NOW =
                0; //завершаем цикл передачи - больше нечего передавать
    }

    if (RI) {

```



```

PushFIFO(&rFIFO, SBUF);
RI = 0;
}
}

// API - функция отправки символа в последовательный канал
//Вход: нет
//Выход: принятый символ или 0 в случае пустого буфера
bit WriteUART(unsigned char c) {
ES = 0; //отключаем прерывание от UART - работа с разделяемыми ресурсами
if (PushFIFO(&wFIFO, c)) {
ES = 1;
if (!TRANSFER_NOW)
TI = 1; //если цикл передачи не начат, искусственно вызываем
//прерывание установкой флага
return 1;
} else {
ES = 1;
return 0; //если буфер полон
}
}

// API - функция отправки ASCIIZ строки в последовательный канал
//Вход: нет
//Выход: принятый символ или 0 в случае пустого буфера
void APIString(const unsigned char *str) {
unsigned char i = 0;
while (i < 80) {
if (str[i] == '\0')
break;
WriteUART(str[i++]);
}
}

// API - функция приема символа из последовательного канала
//Вход: нет
//Выход: принятый символ или 0 в случае пустого буфера
unsigned char ReadUART(void) {
unsigned char c;
ES = 0;
c = PopFIFO(&rFIFO);
ES = 1;
return c;
}

#include "aduc812.h"
#include "system.h"

/*-----
                        RSioStat()
-----
Возвращает ненулевое значение, если буфер приема не пуст

Вход:      нет
Выход:     нет
Результат: 0 - буфер приема пуст;
           1 - был принят символ
----- */
unsigned char rsiostat(void) { return RI; }

/*-----
                        wsio
-----
Отправляет символ по последовательному каналу

Вход:      unsigned char c - символ, который нужно отправить
Выход:     нет
Результат: нет
----- */
void wsio(unsigned char c) {
SBUF = c;
TI = 0;
}

```

```

while (!TI)
;
}

/*
-----
                rsio()
-----
Дождается приема символа из последовательного канала и возвращает его.

Вход:      нет
Выход:     нет
Результат: принятый символ
----- */
unsigned char rsio(void) {
while (!RI) {
if (GetDIP() != 1)
break;
}
RI = 0;
return SBUF;
}

/*-----
                SendString()
-----
Выводит ASCIIZ-строку в последовательный канал

Вход:      char *str - указатель на строку
Выход:     нет
Результат: нет
----- */
void SendString(const char *str) {
while (*str)
wsio(*str++);
}

```

Вывод

В результате выполнения лабораторной работы был получен опыт с последовательным каналом SDK 1.1.