Санкт-петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

С.Э. Вельдер, Ю.Д. Бедный

Универсальный инфракрасный пульт для бытовой техники

Программирование с явным выделением состояний
Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2005

Оглавление

Введение

Для алгоритмизации и программирования эмулятора инфракрасного пульта для бытовой техники была предложена SWITCH-технология, которая развивается применительно к разработке программного обеспечения событийных систем и устройств [1] (http://is.ifmo.ru).

В настоящей работе используется автоматный стиль программирования. Такой подход был назван Н. И. Туккелем и А. А. Шалыто "программирование с явным выделением состоянием".

Технология демонстрируется на примере создания **универсального** инфракрасного (ИК) пульта дистанционного управления (ДУ), реализующего следующие функции:

- запись ИК-сигнала от внешнего источника (например, от ИК-пульта телевизора), его декодирование, преобразование в битовую строку и сопоставление ему одной из кнопок клавиатуры;
- излучение ИК-сигнала при нажатии соответствующей ему кнопки;
- отображение на жидкокристаллическом (ЖК) экране режима работы;
- переключение между различными раскладками клавиатуры.

Работа содержит документацию, включающую в себя техническое описание, структурную схему автомата и его граф переходов, по которым формально и изоморфно реализована программа.

Целью создания этого проекта является демонстрация областей применимости технологии программирования с явным выделением состояний. В дальнейшем будет показано, что использование автоматного подхода является эффективным для реализации различных систем управления, в том числе и для компактных электронных устройств.

Эмуляция пульта была также реализована в прототипе устройства на базе стенда *SDK-1.1* [4, 5], позволяющем конструировать и отлаживать программы для микроконтроллера *ADuC812/83X/84X* фирмы *Analog Device*.

1. Постановка задачи

Пульт ДУ состоит из следующих основных модулей: ИК-приемник, обеспечивающий прием и усиление внешнего ИК-сигнала, ИК-передатчик, клавиатура с управляющими и программируемыми кнопками, жидкокристаллический индикатор (ЖКИ) для отображения состояния устройства.

На клавиатуре расположены программируемые клавиши цифр от 0 до 9, клавиши «запись» и «смена раскладки», а так же 18 дополнительных программируемых клавиш, некоторые из которых для удобства использования помечены специальными символами, наиболее часто используемыми в бытовой технике (рис. 1).

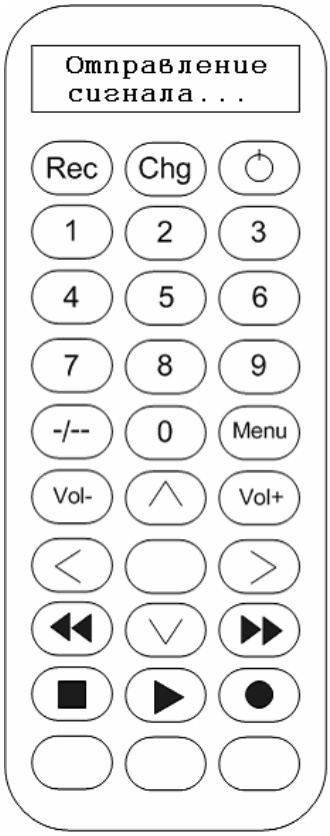

Рис. 1. Универсальный пульт дистанционного управления

Опишем поведение пульта.

Пульт работает в одном из трех режимов: рабочем, энергосбережения, обучение (прием сигнала и сопоставление его клавише).

При нажатии одной из цифровых клавиш в рабочем режиме пульт излучает сигнал, соответствующий нажатой клавише.

После простоя в течение некоторого времени пульт переходит в режим энергосбережения и остается в нем до нажатия любой клавиши.

Для обучения пульта нажимается последовательно клавиша «запись» и одна из программируемых клавиш. Пульт при этом переходит в режим записи сигнала с внешнего устройства. Как только сигнал принят, он запоминается и сопоставляется данной программируемой клавише, и при ее нажатии в рабочем режиме пульт воспроизводит соответствующий инфракрасный сигнал.

Сигнал, посылаемый внешним устройством, представляет собой двоичную последовательность, удовлетворяющую стандарту *RC-5*, который поддерживается большинством производителей бытовой техники [7, 8]. Последовательность пересылается неделимыми блоками, каждый из которых пульт сохраняет во внутреннем буфере до тех пор, пока не будет принят управляющий стоп-сигнал, либо пока пользователь не остановит запись.

Также пульт контролирует переполнение внутреннего буфера, выдавая на ЖКИ сообщения о попытке передать слишком много сигналов.

При помощи кнопки «раскладка» выполняется переключение раскладки клавиатуры, это дает возможность назначать одним и тем же клавишам различные сигналы для соответствующих устройств (в этом заключается универсальность пульта).

Нажатие клавиши «сброс» в рабочем режиме приводит к удалению информации о назначениях клавиш и установки раскладки по умолчанию.

Из любого активного состояния пульт по определенному интервалу бездействия переходит в рабочий режим. Этот интервал контролируется с помощью таймера, встроенного в пульт.

В состав аппаратная части системы входят:

- микроконтроллер со встроенным ОЗУ и энергонезависимой флэш-памятью;
- клавиатура;
- ЖК-дисплей;
- инфракрасный приемопередатчик.

Управление пультом осуществляет автомат *ARemote*, который описывается ниже.

2. Схема связей и граф переходов автомата

Схема связей автомата *ARemote* представлена на рис. 2.

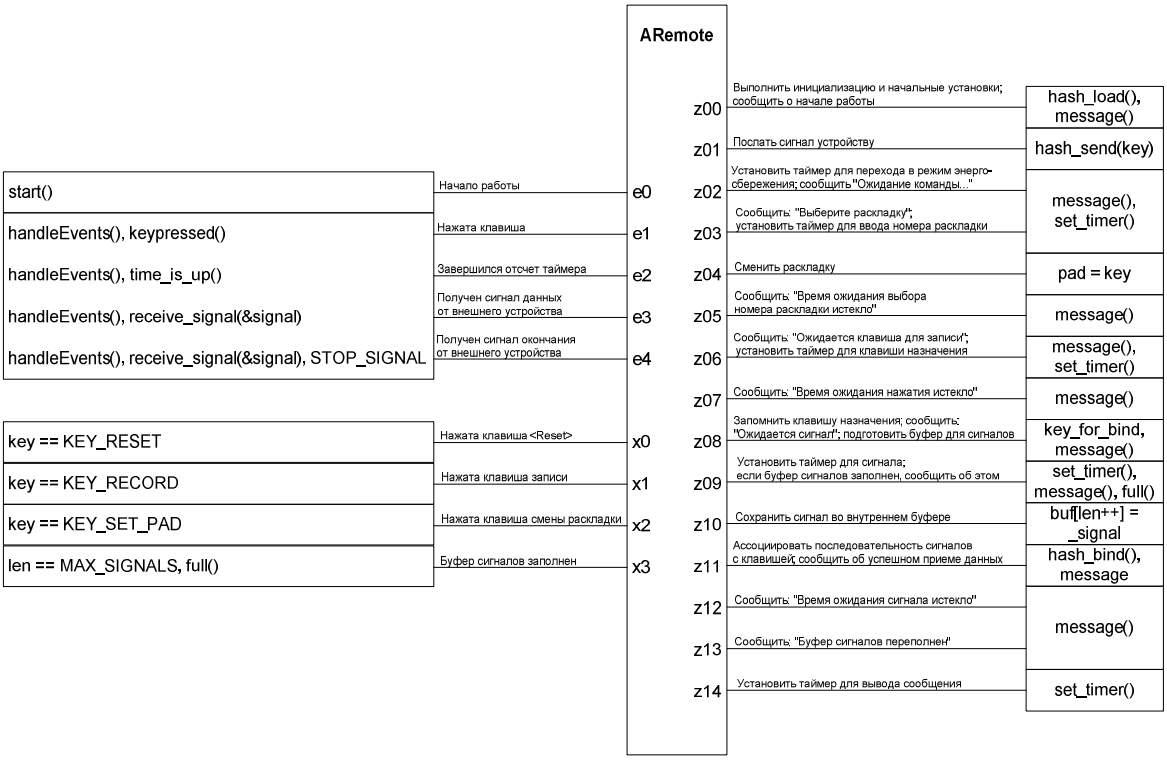

Рис. 2. Схема связей автомата *ARemote*

Граф переходов автомата приведен на рис. 3.

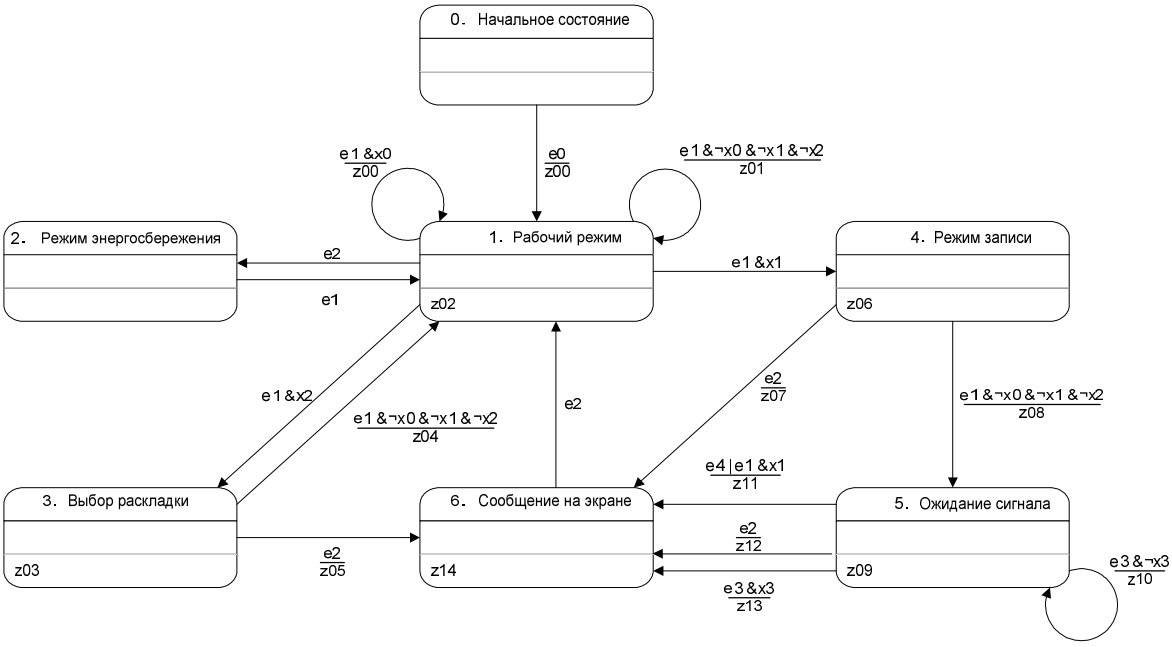

Рис. 3. Граф переходов автомата *ARemote*

3. Реализация модели устройства на персональном компьютере

Реализация модели устройства выполнена на языке *C* в среде *Borland C++ 3.1*. Основная ее часть изоморфна графу переходов автомата *ARemote*. Пример отладочного протокола работы этого автомата приведен в Приложении 1.

В приложении 2 приведен исходный текст программы, моделирующей устройство, которая работает в консольном режиме. В этой программе функция `ARemote()` реализует автомат, а функции `x0()`, …, `x3()` и `z00()`, …, `z14()` — его входные и выходные переменные соответственно.

Программа в бесконечном цикле опрашивает состояние клавиатуры с помощью процедуры `checkKbdBuf()`, моделирующей системный буфер клавиатуры в виде очереди символов. Кроме того, программа с помощью процедуры `handleEvents()` обрабатывает события нажатия клавиши (функция `keypressed(int *key)`), срабатывания таймера (функция `time_is_up()`) и приема сигнала, в том числе и стоп-сигнала (функция `receive_signal(int *signal)`).

Переменные `buf` и `len` отвечают за внутренний буфер пульта, который используется во время приема сигнала.

Функции `hash_load()`, `hash_bind()` и `hash_send(int key)` предназначены для работы с памятью пульта. Они отвечают за ее инициализацию, сопоставление сигнала клавише и излучение сигнала.

Функция `storeLog(int event, int y)` обеспечивает ведение протокола работы автомата, который сохраняется в файл.

4. Реализация модели устройства на микроконтроллере

В рамках проекта по дисциплине «Компьютерные информационно-управляющие системы» на базе стенда *SDK-1.1* [5] была реализована программа, эмулирующая работу пульта применительно к микроконтроллеру *ADuC812BS*. Различные элементы аппаратной части стенда использованы в качестве аналогичных деталей пульта. Соответствие этих элементов указано в таблице.

№	**Пульт + внешнее устройство**	***SDK-1.1* + терминал**
1	Микроконтроллер со встроенным ОЗУ и энергонезависимой флэш-памятью	Микроконтроллер *ADuC812BS*
2	ЖКИ пульта	ЖКИ *SDK*
3	Клавиатура пульта	Клавиатура *SDK*
4	Инфракрасный приемопередатчик пульта	Последовательный приемопередатчик *USART*
5	Инфракрасный приемник внешнего устройства	*COM*-порт и дисплей терминала
6	Инфракрасный передатчик внешнего устройства	*COM*-порт и клавиатура терминала
7	Таймер	Часы/календарь *PCF8583*

Программа (приложение 3) написана на языке *C* при помощи инструментальных средств фирмы *Keil Software* [6]. Часть программы, реализующая автомат, совпадает с аналогичной частью рассмотренной выше программы для компьютера. Опишем другую ее часть, связанную с использованием микроконтроллера.

Функция `init_all()` выполняет инициализацию аппаратных компонент стенда (ЖКИ и последовательного приемопередатчика *USART*) для того, чтобы их в дальнейшем можно было использовать для эмуляции соответствующих компонент пульта. Функция `message(const char *s)` очищает ЖКИ и отображает на нем сообщение, указанное в строке аргумента. Функция `send(const int *signals)` пересылает входной массив сигналов по последовательному порту на терминал. После выполнения этой функции на терминале отобразятся символы, соответствующие этим сигналам. Функция `keypressed(int *key)` проверяет, нажата ли хотя бы одна клавиша на клавиатуре. В случае положительного результата номер клавиши возвращается через выходной параметр `key`. После этого функция вызывает небольшую задержку с целью предотвратить «залипание» клавиш. Функция `receive_signal(int *signal)` проверяет, пришел ли сигнал (любой) с последовательного порта и, если пришел, то помещает его в параметр `signal`.

Заключение

Данная работа является эффективной демонстрацией применения технологии программирования с явным выделением состояний к системам управления электронными приборами. В ней приведены реализации модели пульта для двух аппаратных платформ: персонального компьютера и микроконтроллерного стенда. При этом первая реализация менее трудоемка и использовалась для проверки правильности поведенческой составляющей модели, описываемой автоматом, которая не зависит от платформы. Это явное преимущество автоматного подхода.

Источники

1. *Шалыто А.А*. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. *Добрицкий И.О., Куликов А.А., Шалыто А.А*. Игра "Lines". http://is.ifmo.ru, раздел "Проекты".
3. *Пенев В.П., Степаненков В.В., Сучкоусов Е.А., Шалыто А.А*. Компьютерная игра "Automatic Bomber". http://is.ifmo.ru, раздел "Проекты".
4. *Сайт «Лаборатория микропроцессорной техники кафедры вычислительной техники СПбГУИТМО»*. http://embedded.ifmo.ru.
5. *Сайт «Научно-производственная фирма "ЛМТ"»*. http://lmt.cs.ifmo.ru.
6. *Сайт «Embedded Development Tools» компании «KEIL Software»*. http://www.keil.com.
7. *Сайт "Universal Infrared Receiver"*. http://infrared.h1.ru.
8. *Сайт "SB-Projects"*. http://www.sbprojects.com.

Приложение 1. Пример отладочного протокола

```
Переход по событию e0 (начало работы) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 1 (рабочий режим).
Переход по событию e4 (получен сигнал окончания) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 3 (выбор раскладки).
Переход по событию e1 (нажата клавиша) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 4 (режим записи).
Переход по событию e1 (нажата клавиша) в состояние 5 (ожидание сигнала).
Переход по событию e3 (получен сигнал данных) в состояние 5 (ожидание сигнала).
Переход по событию e3 (получен сигнал данных) в состояние 5 (ожидание сигнала).
Переход по событию e3 (получен сигнал данных) в состояние 5 (ожидание сигнала).
Переход по событию e4 (получен сигнал окончания) в состояние 6 (сообщение на экране).
Переход по событию e2 (завершился отсчет таймера) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 4 (режим записи).
Переход по событию e1 (нажата клавиша) в состояние 5 (ожидание сигнала).
Переход по событию e3 (получен сигнал данных) в состояние 5 (ожидание сигнала).
Переход по событию e1 (нажата клавиша) в состояние 6 (сообщение на экране).
Переход по событию e2 (завершился отсчет таймера) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 4 (режим записи).
Переход по событию e1 (нажата клавиша) в состояние 5 (ожидание сигнала).
Переход по событию e3 (получен сигнал данных) в состояние 5 (ожидание сигнала).
Переход по событию e2 (завершился отсчет таймера) в состояние 6 (сообщение на экране).
Переход по событию e2 (завершился отсчет таймера) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 4 (режим записи).
Переход по событию e2 (завершился отсчет таймера) в состояние 6 (сообщение на экране).
Переход по событию e2 (завершился отсчет таймера) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 3 (выбор раскладки).
Переход по событию e1 (нажата клавиша) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 3 (выбор раскладки).
Переход по событию e2 (завершился отсчет таймера) в состояние 6 (сообщение на экране).
Переход по событию e2 (завершился отсчет таймера) в состояние 1 (рабочий режим).
Переход по событию e2 (завершился отсчет таймера) в состояние 2 (режим энергосбережения).
Переход по событию e1 (нажата клавиша) в состояние 1 (рабочий режим).
Переход по событию e1 (нажата клавиша) в состояние 1 (рабочий режим).
Переход по событию e4 (получен сигнал окончания) в состояние 1 (рабочий режим).
```

Приложение 2. Исходные тексты для компьютера

AREMOTE.C

```
#include <stdlib.h>
#include <stdio.h>
#include "sys.h"
#include "util.h"

#define  TIMEOUT_SLEEP_MODE     150
#define  TIMEOUT_CHOOSE_PAD      10
#define  TIMEOUT_PRESS_KEY       10
#define  TIMEOUT_RECEIVE_SIGNAL  60
#define  TIMEOUT_SHOW_MESSAGE     3

int buf[MAX_SIGNALS];                           // Буфер сигналов
int len;                                        // Размер занятой части буфера
int pad;                                        // Номер текущей раскладки
int key_for_bind;

int key = KEY_NOKEY;                            // Номер нажатой клавиши
int _signal;                                    // Принятый сигнал

FILE* f;                                        // Файл протокола

int full()
{
    return len == MAX_SIGNALS;
}

int x0()
{
    return key == KEY_RESET;
}

int x1()
{
    return key == KEY_RECORD;
}

int x2()
{
    return key == KEY_SET_PAD;
}

int x3()
{
    return full();
}

void z00()
{
    hash_load();
    pad = 0;
    say_ready();
}

void z01()
{
    hash_send(key);
}

void z02()
{
    message(MSG_WORK);
    set_timer(TIMEOUT_SLEEP_MODE);
}

void z03()
{
    message(MSG_CHOOSE_PAD);
    set_timer(TIMEOUT_CHOOSE_PAD);
}

void z04()
```

```
{
    pad = key;
}

void z05()
{
    message(MSG_PAD_NOT_CHOSEN);
}

void z06()
{
    message(MSG_PRESS_KEY);
    set_timer(TIMEOUT_PRESS_KEY);
}

void z07()
{
    message(MSG_RECORD_TIME_IS_UP);
}

void z08()
{
    key_for_bind = key;
    message(MSG_LISTEN);
    len = 0;
}

void z09()
{
    set_timer(TIMEOUT_RECEIVE_SIGNAL);
    if (full()) message(MSG_BUFFER_IS_FULL);
}

void z10()
{
    buf[len++] = _signal;
}

void z11()
{
    hash_bind();
    message(MSG_KEY_BINDED);
}

void z12()
{
    message(MSG_LISTEN_TIME_IS_UP);
}

void z13()
{
    message(MSG_BUFFER_OVERFLOW);
}

void z14()
{
    set_timer(TIMEOUT_SHOW_MESSAGE);
}

int Y;

void ARemote(int event)
{
    switch (Y)
    {
        case 0:
            if (event == 0)
            {
                z00();
                Y = 1;
            }
            break;
        case 1:
            if (event == 1)
                if      (x0())     z00();
                else if (x1())     Y = 4;
                else if (x2())     Y = 3;
                else               z01();
            else if (event == 2)   Y = 2;
            break;
```

```
        case 2:
            if (event == 1)        Y = 1;
            break;
        case 3:
            if (event == 1 && !x0() && !x1() && !x2())
            {
                z04();
                Y = 1;
            }
            else if (event == 2)
            {
                z05();
                Y = 6;
            }
            break;
        case 4:
            if (event == 1 && !x0() && !x1() && !x2())
            {
                z08();
                Y = 5;
            }
            else if (event == 2)
            {
                z07();
                Y = 6;
            }
            break;
        case 5:
            if (event == 2)
            {
                z12();
                Y = 6;
            }
            else if (event == 3)
                if (x3())
                {
                    z13();
                    Y = 6;
                }
                else z10();
            else if (event == 4 || event == 1 && x1())
            {
                z11();
                Y = 6;
            }
            break;
        case 6:
            if (event == 2)  Y = 1;
    }

    storeLog(event, Y);

    switch (Y)
    {
        case 1: z02(); break;
        case 3: z03(); break;
        case 4: z06(); break;
        case 5: z09(); break;
        case 6: z14();
    }
}

void start()
{
    Y = 0;        //  Y0:  Init  - Начальное состояние
    ARemote(0);   //  e0:  Reset - Начало работы
}

void handleEvents()
{
    if (keypressed(&key))          ARemote(1);
    if (time_is_up())              ARemote(2);
    if (receive_signal(&_signal))  ARemote((_signal == STOP_SIGNAL) ? 4 : 3);
}

void done()
{
    fclose(f);
    printf("\n\nРабота завершена.\n");
    exit(0);
```

```
}

void main()
{
    init_all();
    start();
    for (;;)
    {
        handleEvents();
        if (key == KEY_ESC) done();
    }
}
```

SYS.C

```
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include "sys.h"

#define  FILE_NAME  "AREMOTE.LOG"
 extern  FILE* f;

int kbdBuf[KBD_BUF_LEN];
int kbdI, kbdJ = 0;

void init_all()
{
    f = fopen(FILE_NAME, "w");
    printf("\nЭмулятор универсального ИК пульта для бытовой техники\n"
           "    Выполнили: С.Э.Вельдер, Ю.Д.Бедный, гр. 3538\n"
           "                        2005 г.\n\n"
           "                     Управление:\n\n"
           "ESC            - выход из программы\n"
           "0..9           - цифровые клавиши на пульте\n"
           "Space (Пробел) - клавиша <Reset> на пульте\n"
           "Tab            - клавиша смены раскладки на пульте\n"
           "+              - клавиша записи/сохранения сигнала на пульте\n"
           "A..Z, a..z     - сигналы внешнего устройства\n"
           "Enter          - сигнал окончания от внешнего устройства\n\n");
}

int next(int i)
{
    return (++i == KBD_BUF_LEN) ? 0 : i;
}

int prev(int i)
{
    return ((i == 0) ? KBD_BUF_LEN : i) - 1;
}

void wait_for_key()
{
    while (!kbhit());
    while (kbhit()) getch();
}

void message(const char *s)
{
    int i;
    printf("\r");
    for (i = 0; i < SPACES; i++) putch(' ');
    printf("\r%s ", s);
}

void say_ready()
{
    message(MSG_READY);
    wait_for_key();
}

void send(const int *signals)
{
    char s[MAX_SIGNALS];
    int i, x;
    for (i = 0; x = signals[i]; i++) s[i] = x;
```

```
    s[i] = 0;
    printf("\r");
    for (i = 0; i < SPACES; i++) printf(" ");
    printf("\rУстройству был послан сигнал: \"%s\". Нажмите любую клавишу... ", s);
    wait_for_key();
}

void getTime(TTime *_time)
{
    struct time t;
    gettime(&t);
    _time->h = t.ti_hour;
    _time->m = t.ti_min;
    _time->s = t.ti_sec;
}

int keypressed(int *key)
{
    int ch;
    checkKbdBuf();
    if (kbdI == kbdJ) return 0;
    ch = kbdBuf[kbdI];
    kbdI = next(kbdI);
    switch(ch)
    {
        case  27:                   // ESC
            *key = KEY_ESC;
            return 0;
        case  32:                   // Space
            *key = KEY_RESET;
            return 1;
        case   9:                   // Tab
            *key = KEY_SET_PAD;
            return 1;
        case '+':
            *key = KEY_RECORD;
            return 1;
    }
    if (ch >= '0' && ch <= '9')
    {
        *key = ch - '0';
        return 1;
    }
    kbdI = prev(kbdI);
    return 0;
}

int receive_signal(int *signal)
{
    int ch;
    checkKbdBuf();
    if (kbdI == kbdJ) return 0;
    ch = kbdBuf[kbdI];
    kbdI = next(kbdI);
    if (ch == 13)
    {
        *signal = STOP_SIGNAL;
        return 1;
    }
    if (ch >= 'A' && ch <= 'Z' || ch >= 'a' && ch <= 'z')
    {
        *signal = ch;
        return 1;
    }
    kbdI = prev(kbdI);
    return 0;
}

void checkKbdBuf()
{
    if (kbhit())
    {
        if (next(kbdJ) != kbdI)
        {
            int ch = getch();
            kbdBuf[kbdJ] = ch;
            kbdJ = next(kbdJ);
        }
        while (kbhit()) getch();
    }
```

```
}

void storeLog(int event, int y)
{
    char *events[5] = {
                        "начало работы",
                        "нажата клавиша",
                        "завершился отсчет таймера",
                        "получен сигнал данных",
                        "получен сигнал окончания"
                      };
    char *states[7] = {
                        "начальное состояние",
                        "рабочий режим",
                        "режим энергосбережения",
                        "выбор раскладки",
                        "режим записи",
                        "ожидание сигнала",
                        "сообщение на экране"
                      };
    fprintf(f, "Переход по событию e%d (%s) в состояние %d (%s).\n",
               event, events[event], y, states[y]);
}
```

UTIL.C

```
#include "sys.h"
#include "util.h"

extern int buf[MAX_SIGNALS];                    // Буфер сигналов
extern int len;                                 // Размер занятой части буфера
extern int pad;                                 // Номер текущей раскладки
extern int key_for_bind;

int arr[NUM_PADS][NUM_KEYS][MAX_SIGNALS + 1];   // Содержимое памяти пульта

int timer_sec;                                  // На сколько установлен таймер
int timer_off = 1;                              // Выключен ли таймер
TTime start_time;

void set_timer(int sec)
{
    timer_off = 0;
    timer_sec = sec;
    getTime(&start_time);
}

int time_is_up()
{
    TTime now;
    long d;
    if (timer_off) return 0;
    getTime(&now);
    d = (((long)now.h - start_time.h) * 60 + now.m - start_time.m) * 60 + now.s - start_time.s;
    if (d < 0) d += 86400L;
    timer_off = d >= timer_sec;
    return timer_off;
}

void hash_load()
{
    int i, j;
    for (i = 0; i < NUM_PADS; i++)
        for (j = 0; j < NUM_KEYS; j++)
            arr[i][j][0] = 0;
}

void hash_bind()
{
    int i;
    for (i = 0; i < len; i++)  arr[pad][key_for_bind][i] = buf[i];
    arr[pad][key_for_bind][i] = 0;
}

void hash_send(int key)
{
    send(arr[pad][key]);
```

```
}
```

SYS.H

```
#ifndef __SYS
#define __SYS

#define  SPACES       79
#define  STOP_SIGNAL  -1
#define  KBD_BUF_LEN  16

#define  NUM_KEYS     16                      // Число цифорвых клавиш
#define  NUM_PADS     NUM_KEYS                // Число раскладок
#define  MAX_SIGNALS  10                      // Максимум сигналов на клавишу

#define  KEY_NOKEY    -2
#define  KEY_ESC      -1
#define  KEY_RESET    NUM_KEYS
#define  KEY_SET_PAD  (NUM_KEYS + 1)
#define  KEY_RECORD   (NUM_KEYS + 2)

#define  MSG_READY              "Инициализация завершена. Пульт готов к работе. Нажмите любую клавишу..."
#define  MSG_WORK               "Ожидание команды..."
#define  MSG_CHOOSE_PAD         "Выберите раскладку..."
#define  MSG_PAD_NOT_CHOSEN     "Время ожидания выбора номера раскладки истекло."
#define  MSG_PRESS_KEY          "Ожидается клавиша для записи..."
#define  MSG_RECORD_TIME_IS_UP  "Время ожидания нажатия истекло."
#define  MSG_LISTEN_TIME_IS_UP  "Время ожидания сигнала истекло."
#define  MSG_LISTEN             "Ожидается сигнал..."
#define  MSG_KEY_BINDED         "Сигнал ассоциирован с введенной клавишей."
#define  MSG_BUFFER_OVERFLOW    "Буфер сигналов переполнен. Данные утеряны."
#define  MSG_BUFFER_IS_FULL     "Буфер сигналов заполнен. Чтобы сохранить его, остановите запись."


typedef struct { int h, m, s; } TTime;

void init_all();
void message(const char *s);
void send(const int *signals);
void getTime(TTime *time);
int keypressed(int *key);
int receive_signal(int *signal);
void checkKbdBuf();
void storeLog(int event, int y);

#endif
```

UTIL.H

```
#ifndef __UTIL
#define __UTIL

void set_timer(int sec);
int time_is_up();
void hash_load();
void hash_bind();
void hash_send(int key);

#endif
```

Приложение 3. Исходные тексты для микроконтроллера

AREMOTE.C

```
#include "sys.h"
#include "util.h"

#define  TIMEOUT_SLEEP_MODE      10
#define  TIMEOUT_CHOOSE_PAD      10
#define  TIMEOUT_PRESS_KEY       10
#define  TIMEOUT_RECEIVE_SIGNAL  10
#define  TIMEOUT_SHOW_MESSAGE     3

int buf[MAX_SIGNALS];                           // Буфер сигналов
int len;                                        // Размер занятой части буфера
int pad;                                        // Номер текущей раскладки
int key_for_bind;

int key;                                        // Номер нажатой клавиши
int _signal;                                    // Принятый сигнал

int full()
{
    return len == MAX_SIGNALS;
}

int x0()
{
    return key == KEY_RESET;
}

int x1()
{
    return key == KEY_RECORD;
}

int x2()
{
    return key == KEY_SET_PAD;
}

int x3()
{
    return full();
}

void z00()
{
    hash_load();
    pad = 0;
    say_ready();
}

void z01()
{
    hash_send(key);
}

void z02()
{
    message(MSG_WORK);
    set_timer(TIMEOUT_SLEEP_MODE);
}

void z03()
{
    message(MSG_CHOOSE_PAD);
    set_timer(TIMEOUT_CHOOSE_PAD);
}

void z04()
{
    pad = key;
}
```

```
void z05()
{
    message(MSG_PAD_NOT_CHOSEN);
}

void z06()
{
    message(MSG_PRESS_KEY);
    set_timer(TIMEOUT_PRESS_KEY);
}

void z07()
{
    message(MSG_RECORD_TIME_IS_UP);
}

void z08()
{
    key_for_bind = key;
    message(MSG_LISTEN);
    len = 0;
}

void z09()
{
    set_timer(TIMEOUT_RECEIVE_SIGNAL);
    if (full()) message(MSG_BUFFER_IS_FULL);
}

void z10()
{
    buf[len++] = _signal;
}

void z11()
{
    hash_bind();
    message(MSG_KEY_BINDED);
}

void z12()
{
    message(MSG_LISTEN_TIME_IS_UP);
}

void z13()
{
    message(MSG_BUFFER_OVERFLOW);
}

void z14()
{
    set_timer(TIMEOUT_SHOW_MESSAGE);
}

int Y;

void ARemote(int event)
{
    switch (Y)
    {
        case 0:
            if (event == 0)
            {
                z00();
                Y = 1;
            }
            break;
        case 1:
            if (event == 1)
                if      (x0())     z00();
                else if (x1())     Y = 4;
                else if (x2())     Y = 3;
                else               z01();
            else if (event == 2)   Y = 2;
            break;
        case 2:
            if (event == 1)        Y = 1;
            break;
        case 3:
```

```
            if (event == 1 && !x0() && !x1() && !x2())
            {
                z04();
                Y = 1;
            }
            else if (event == 2)
            {
                z05();
                Y = 6;
            }
            break;
        case 4:
            if (event == 1 && !x0() && !x1() && !x2())
            {
                z08();
                Y = 5;
            }
            else if (event == 2)
            {
                z07();
                Y = 6;
            }
            break;
        case 5:
            if (event == 2)
            {
                z12();
                Y = 6;
            }
            else if (event == 3)
                if (x3())
                {
                    z13();
                    Y = 6;
                }
                else z10();
            else if (event == 4 || event == 1 && x1())
            {
                z11();
                Y = 6;
            }
            break;
        case 6:
            if (event == 2)  Y = 1;
    }

    storeLog(event, Y);

    switch (Y)
    {
        case 1: z02(); break;
        case 3: z03(); break;
        case 4: z06(); break;
        case 5: z09(); break;
        case 6: z14();
    }
}

void start()
{
    Y = 0;        //  Y0:  Init  - Начальное состояние
    ARemote(0);   //  e0:  Reset - Начало работы
}

void handleEvents()
{
    if (keypressed(&key))          ARemote(1);
    if (time_is_up())              ARemote(2);
    if (receive_signal(&_signal))  ARemote((_signal == STOP_SIGNAL) ? 4 : 3);
}

void main()
{
    init_all();
    start();
    for (;;) handleEvents();
}
```

SYS.C

```
#include "..\src_sdk\ADuC812.h"
#include "..\src_sdk\lcd.h"
#include "..\src_sdk\max.h"

#include "sys.h"

//Константы скорости
#define S9600   0xFD
#define S4800   0xFA
#define S2400   0xF4
#define S1200   0xE8
#define REPEAT_DELAY 300

void InitSIO(char speed, bit sdouble)
{
    TH1      = speed;
    TMOD    |= 0x20; //Таймер 1 будет работать в режиме autoreload

    if(sdouble)
        PCON|=0x80;//Если sdouble==1, то скорость удваивается
    else
        PCON&=~0x80;

    TCON    |= 0x40; //Запуск таймера 1
    SCON     = 0x50; //Настройки последовательного канала
    ES       = 0;    //Запрещение прерываний от приемопередатчика
}

void init_all()
{
    InitLCD();
    InitSIO(S9600, 0);
}

void wait_for_key()
{
    int key;
    while (keypressed(&key));
    while (!keypressed(&key));
    while (keypressed(&key));
}

void message(const char *s)
{
       LCD_Clear();
       LCD_Type(s);
}

void say_ready()
{
    message(MSG_READY);
    wait_for_key();
}

void send(const int *signals)
{
    char c;
    int i;
    for (i = 0; c = (char)signals[i]; i++)
    {
        TI = 0;
        SBUF = c;
        while (!TI);
    }
}

int is_pressed(unsigned char row, unsigned char col)
{
    unsigned char addr = 0x080000, tmp;
    WriteMax(addr, ~(1 << col));
    tmp = ReadMax(addr) >> 4;
    return !(tmp & (1 << row));
}

int keypressed(int *key)
{
```

```
    int i, j;
    for (i = 0; i < NUM_KEYS; i++)
    {
        if (is_pressed(i >> 2, i & 3))
        {
            *key = i;
            for (i = 0; i < REPEAT_DELAY; i++)
                for (j = 0; j < REPEAT_DELAY; j++);
            return 1;
        }
    }
    return 0;
}

int receive_signal(int *signal)
{
    if (RI) {
        RI = 0;
        *signal = SBUF;
        return 1;
    }
    return 0;
}

void storeLog(int event, int y)
{
}
```

UTIL.C

```
#include "..\src_sdk\ADuC812.h"
#include "..\src_sdk\lcd.h"
#include "..\src_sdk\rtc.h"
#include "..\src_sdk\max.h"

#include "sys.h"
#include "util.h"

extern int buf[MAX_SIGNALS];                    // Буфер сигналов
extern int len;                                 // Размер занятой части буфера
extern int pad;                                 // Номер текущей раскладки
extern int key_for_bind;

int arr[NUM_PADS][NUM_KEYS][MAX_SIGNALS + 1];   // Содержимое памяти пульта

int timer_sec;                                  // На сколько установлен таймер
int timer_off = 1;                              // Выключен ли таймер
TIME start_time;

void set_timer(int sec)
{
    timer_off = 0;
    timer_sec = sec;
    GetTime(&start_time);
}

int time_is_up()
{
    TIME now;
    long d;
    if (timer_off) return 0;
    GetTime(&now);
    d = (((long)now.hour - start_time.hour) * 60 + now.min - start_time.min)
        * 60 + now.sec - start_time.sec;
    if (d < 0) d += 86400L;
    timer_off = d >= timer_sec;
    return timer_off;
}

void hash_load()
{
    int i, j;
    for (i = 0; i < NUM_PADS; i++)
        for (j = 0; j < NUM_KEYS; j++)
            arr[i][j][0] = 0;
}
```

```
void hash_bind()
{
    int i;
    for (i = 0; i < len; i++)  arr[pad][key_for_bind][i] = buf[i];
    arr[pad][key_for_bind][i] = 0;
}

void hash_send(int key)
{
    send(arr[pad][key]);
}
```

SYS.H

```
#ifndef __SYS
#define __SYS

#define  STOP_SIGNAL  '\n'

#define  NUM_KEYS     16                        // Число цифорвых клавиш
#define  NUM_PADS     NUM_KEYS                  // Число раскладок
#define  MAX_SIGNALS  10                        // Максимум сигналов на клавишу

#define  KEY_RESET    (NUM_KEYS - 3)
#define  KEY_SET_PAD  (NUM_KEYS - 2)
#define  KEY_RECORD   (NUM_KEYS - 1)

#define  MSG_READY              "Ready for work. Press any key."
#define  MSG_WORK               "Waiting for command..."
#define  MSG_CHOOSE_PAD         "Choose layout..."
#define  MSG_PAD_NOT_CHOSEN     "Layout is not chosen in time."
#define  MSG_PRESS_KEY          "Press key for bind..."
#define  MSG_RECORD_TIME_IS_UP  "Nothing was pressed."
#define  MSG_LISTEN_TIME_IS_UP  "Can't catch signal."
#define  MSG_LISTEN             "Listening for signal..."
#define  MSG_KEY_BINDED         "Key has been binded."
#define  MSG_BUFFER_OVERFLOW    "Buffer overflow. Data is lost."
#define  MSG_BUFFER_IS_FULL     "Buffer's full. Pls, finish rec."

void init_all();
void message(const char *s);
void send(const int *signals);
int keypressed(int *key);
int receive_signal(int *signal);
void storeLog(int event, int y);

#endif
```

UTIL.H

```
#ifndef __UTIL
#define __UTIL

void set_timer(int sec);
int time_is_up();
void hash_load();
void hash_bind();
void hash_send(int key);

#endif
```