```
void interrupt tmr0_int(void)
{
    if (TOIF) {          // Обработчик прерывания от таймера TMR0.
          TOIF = 0;      // Сбросить флаг прерывания от таймера TMR0.
          RTC++;         // Инкремент счетчика реального времени.
          // Здесь можно разместить дополнительный код для обработки
          // прерываний от таймера TMR0.
    } // Конец обработчика прерываний от таймера TMR0.
    // Здесь можно разместить дополнительные обработчики прерываний.
} // Конец обработчика прерываний.

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS) {       // Запись полубайта в ЖКИ.
unsigned int i;
    Data = 0;                           // Очистка сдвигового регистра.
    for (i = 0; i < 6; i++) {           // Повторить шесть раз.
          Clock = 1; Clock = 0;         // Строб Clock
                                        // для записи нулевых разрядов.
    }
    Data = 1;                           // Строб E.
    Clock = 1; Clock = 0;               // Строб Clock.
    Data = RS;                          // Бит R/S.
    Clock = 1; Clock = 0;
    for (i = 0; i < 4; i++) {           // Записать 4 бита.
          if ((Nybble & 0x008) != 0)
                Data = 1;
          else
                Data = 0;
          Clock = 1; Clock = 0;         // Строб Clock.
          Nybble = Nybble << 1;         // Сдвиг перед записью
                                        // нового полубайта.
    }
    Data = 1; Data = 0;                 // Строб E.
} // Конец подпрограммы LCDNybble.

LCDByte(char Byte, char RS) {           // Запись байта в ЖКИ.
int LBDlay;
    LCDNybble((Byte >> 4) & 0x00F, RS); // Старшая тетрада.
    LCDNybble(Byte & 0x00F, RS);        // Младшая тетрада.
    if ((Byte < 4) && (RS == 0))        // Выясняем тип команды.
          LBDlay = RTC + 6;             // Задержка на 5 мс.
    else
          LBDlay = RTC + 2;             // Задержка на 1 мс.
    while(LBDlay != RTC);         // Закончить выполнение задержки.
} // Конец подпрограммы LCDByte.

// Главная программа:
```

```
void main(void)
{
int i;

    OPTION = 0x0D1;        // Предделитель работает с таймером TMR0,
                           // коэффициент деления равен 4.

    TMR0 = 0;                    // Начальный сброс таймера.
    T0IE = 1;                    // Разрешить прерывания от таймера.
    GIE = 1;                     // Разрешить обработку прерываний.

    // Здесь можно выполнить инициализацию других периферийных устройств.

    Clock = 0;  Data = 0;  // На обеих линиях двухпроводного интерфейса
                           // установлены сигналы низкого уровня.

    ClockTRIS = 0;         // Оба вывода, к которым подключен
    DataTRIS = 0;          // интерфейс ЖКИ, переведены
                           // в режим выходных.

    Dlay = RTC + 20;             // Шаг 1 - ждать более 15 мс
    while (Dlay != RTC);         // после включения питания.

    LCDNybble(0x003, 0);         // Шаг 2 - полубайт для инициализации
    Dlay = RTC + 6;              // и задержка на 5 мс.
    while (Dlay != RTC);

    LCDNybble(0x003, 0);         // Шаг 3 - полубайт
    Dlay = RTC + 1;              // для инициализации и задержка
    while (Dlay != RTC);         // более 160 мкс (1 мс).

    LCDNybble(0x003, 0);         // Шаг 4 - в третий раз повторяем
    Dlay = RTC + 1;              // инициализирующий полубайт
    while (Dlay != RTC);         // и задержку.

    LCDNybble(0x002, 0);         // Шаг 5 - устанавливаем
    Dlay = RTC + 1;              // курсор в начальную позицию.
    while (Dlay != RTC);         // Задержка.

    LCDByte(0x028, 0);           // Шаг 6 - устанавливаем режим
                                 // (4-битные данные, одна строка,
                                 // размер символов 5x7).

    LCDByte(0x008, 0);           // Шаг 7 - выключение дисплея.

    LCDByte(0x001, 0);           // Шаг 8 - очистка экрана.

    LCDByte(0x006, 0);           // Шаг 9 - режим смещения
                                 // (сдвиг курсора включен,
                                 // смещение изображения выключено).

    LCDByte(0x00E, 0);           // Шаг 10 - включение дисплея.

    for (i = 0; i < 12; i++)     // Вывод сообщения "Hello World!".
          LCDByte(Message[i], 1);

    while (1 == 1) {             // Бесконечный цикл.

    // Здесь можно разместить код верхнего уровня.

    }

}  // Конец главной программы.
```

Приведенный текст программы не должен вызвать затруднений, так как он мало отличается от наших предыдущих приложений. Как и прежде, для реализаций функций электронного уровня мы используем прерывания от таймера TMRO, генерируемые каждую миллисекунду.

Но загружать эту программу в микроконтроллер было бы преждевременно. Здесь мы допустили грубую ошибку, разместив часть кода, работающего с дисплеем, в главной программе, то есть там, где должны располагаться операторы, реализующие функции биологического уровня. В результате программа будет работоспособна только в том случае, если жидкокристаллический дисплей является единственным периферийным устройством. Обеспечить возможность выполнения этим приложением каких-либо других функций будет затруднительно.

Решить проблему можно, используя модель *конечного автомата.* Каждый раз при вызове обработчика прерываний мы будем проверять текущее состояние конечного автомата и выполнять действия, необходимые на данном этапе.

Измененный текст программы вы найдете в файле Code\Lcd\lcd2.c:

```
#include <pic.h>

// 17 февраля 2002 - программа управления жидкокристаллическим
// дисплеем, измененная в соответствии с правилами "трех уровней".
//
// Для подключения дисплея к микроконтроллеру используется
// двухпроводной интерфейс на основе сдвигового регистра 74LS174.
//
// Используемые аппаратные средства:
// микроконтроллер PIC16F84/PIC16F627,
// тактовая частота 4 МГц,
// используется керамический резонатор,
// RB1 - линия Clock (тактирование),
// RB2 - линия Data (данные).
//

// Глобальные переменные и константы:
int RTC = 0;                        // Счетчик реального времени.
volatile char LCDDlay = 20;         // Длительность задержки.
volatile char LCDState = 1;         // Номер текущего состояния
                                    // конечного автомата.

static volatile bit Clock           @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS       @ (unsigned)&TRISB*8+1;
static volatile bit Data            @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS        @ (unsigned)&TRISB*8+2;

char * MessageOut; // Указатель на строку выводимого сообщения.
volatile char MessageOuti = 0;   // Смещение, указывающее позицию
                                 // выводимого символа.
char Message[13] = "Hello World!";      // Текст сообщения.
char Message2[11] = "\376\3002nd Line";

// Слово конфигурации:
#if defined (_16F84)
    #warning PIC16F84 selected
    __CONFIG(Ox03FF1);     // Для МК PIC16F84:
                           // кварцевый тактовый генератор,
```

```
                        // таймер PWRT включен,
                        // сторожевой таймер выключен,
                        // защита кода отключена.
#elif defined(_16F627)
    #warning PIC16F627 with internal oscillator selected
    _CONFIG(0x03F61);   // Для МК PIC16F627:
                        // внешний тактовый генератор XT,
                        // RA6/RA7 используются для ввода-вывода,
                        // внешний сигнал сброса,
                        // таймер PWRT включен,
                        // сторожевой таймер выключен,
                        // защита кода отключена,
                        // детектор BODEN включен.
#else
    #error Unsupported PICmicro MCU selected
#endif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS) {       // Запись полубайта в ЖКИ.

unsigned int i;

    Data = 0;                           // Очистка сдвигового регистра.
    for (i = 0; i < 6; i++) {           // Повторить шесть раз.
        Clock = 1; Clock = 0;           // Строб Clock
                                        // для записи нулевых разрядов.
    }

    Data = 1;                           // Строб E.
    Clock = 1; Clock = 0;               // Строб Clock.

    Data = RS;                          // Бит R/S.
    Clock = 1; Clock = 0;

    for (i = 0; i < 4; i++) {           // Записать 4 бита.
        if ((Nybble & 0x008) != 0)
            Data = 1;
        else
            Data = 0;
        Clock = 1; Clock = 0;           // Строб Clock.
        Nybble = Nybble << 1;           // Сдвиг перед записью
                                        // нового полубайта.
    }

    Data = 1; Data = 0;                 // Строб E.

}   // Конец подпрограммы LCDNybble.

LCDByte(char Byte, char RS) {           // Запись байта в ЖКИ.
int L8Dlay;

    LCDNybble((Byte >> 4) & 0x00F, RS); // Старшая тетрада.
    LCDNybble(Byte & 0x00F, RS);        // Младшая тетрада.

    // Обратите внимание, что текст подпрограммы изменился.

}   // Конец подпрограммы LCDByte.

LCDInit()               // Инициализация ЖКИ.
{
```

```
    Clock = 0; Data = 0;  // На обеих линиях двухпроводного интерфейса
                          // установлены сигналы низкого уровня.

    ClockTRIS = 0;        // Оба вывода, к которым подключен
    DataTRIS = 0;         // интерфейс ЖКИ, переведены в
                          // режим выходных.

}  // Конец подпрограммы инициализации.

LCDOut(char * const LCOString)   // Вывод строки на ЖКИ.
{

    while (LCDState);     // Ждать, когда можно будет начать запись.

    MessageOut = LCDString;      // Загрузка строки для вывода.
    LCDState = 100;              // Начинаем запись строки на ЖКИ.

}  // Конец подпрограммы LCDOut.

// Обработчик прерываний:
void interrupt tmr0_int(void)
{

char temp;

    if (TOIF) {           // Обработчик прерываний от таймера TMRO.

          TOIF = 0;       // Сбросить флаг прерываний от таймера TMRO.

          RTC++;          // Инкремент счетчика реального времени.

          // Здесь можно разместить дополнительный код
          // для обработки прерываний от таймера TMRO.

          // Конечный автомат:
          switch(LCDState) {    // В зависимости от текущего состояния:
                case 1:                 // Начать инициализацию ЖКИ.
                      if (--LCDDlay == 0)
                            LCDState++;
                      break;            // Ждать 20 мс.

                case 2:                 // Шаг 2.
                      LCDNybble(0x003,  0);
                      LCDDlay = 5;
                      LCDState++;
                case 3:                 // Ждать выполнения команды.
                      if (--LCDDlay == 0)
                            LCDState++;
                      break;
                case 4:
                      LCDNybble(0x003,  0);   // Шаг 3.
                      LCDState++;
                      break;
                case 5:
                      LCDNybble(0x003,  0);   // Шаг 4.
                      LCDState++;
                      break;
                case 6:
                      LCDNybble(0x002,  0);   // Шаг 5.
                      LCDState++;
                      break;
```

```
case 7:
        LCDByte(0x028, 0);      // Шаг 6.
        LCDState++;
        break;
case 8:
        LCDByte(0x008, 0);      // Шаг 7.
        LCDState++;
        break;
case 9:
        LCDByte(0x001, 0);      // Шаг 8.
        LCDState++;
        LCDDlay = 5;            // Ждать 5 мс.
        break;
case 10:                // Ждать выполнения команды.
        if (--LCDDlay == 0)
                LCDState++;
        break;
case 11:
        LCDByte(0x006, 0);      // Шаг 9.
        LCDState++;
        break;
case 12:
        LCDByte(0x00E, 0);      // Шаг 10.
        LCDState = 0;           // Все готово.
        break;

case 100:       // Вывод сообщения MessageOut.
        switch (temp = MessageOut[MessageOuti++]) {
                case '\0':      // Конец сообщения?
                        LCDState = 0;
                                // ЖКИ теперь свободен.
                        MessageOuti = 0;
                                // Возврат индекса
                                // на начало строки.
                        break;
                case '\f':      // Очистка экрана.
                        LCDByte(0x001, 0);
                        LCDState++;
                        LCDDlay = 5;
                        break;
                case 254:       // Перед записью команды.
                        if ((temp =
        MessageOut[MessageOuti++]) == 0)
                                LCDState = 0;
                        else {
                                if (temp < 4) {
                                        LCDState++;
                                        LCDDlay = 5;
                                } // endif
                                LCDByte(temp, 0);
                        }
                        break;
                default:        // Все другие символы.
                        LCD8yte(temp, 1);
        } // Конец внутреннего оператора switch.
        break;
case 101:                       // Задержка.
```

```
                        if (--LCDDlay == 0)
                              LCDState--;
                        break;
            }  // Конец внешнего оператора switch.

    } // Конец оператора if.

    // Здесь можно разместить другие обработчики прерываний.

}  // Конец обработчика прерываний.

// Главная программа:
void main(void)
{

    OPTION = 0xOD1;        // Предделитель работает с таймером TMRO,
                           // коэффициент деления равен 4.

    TMRO = 0;              // Начальный сброс таймера TMRO.
    TOIE = 1;              // Разрешение прерываний от таймера TMRO.
    GIE = 1;               // Разрешение обработки прерываний.

    LCDInit();     // Инициализация порта, к которому подключен ЖКИ.

    // Здесь можно проинициализировать другие периферийные устройства.

    LCDOut(Message);       // Передача строки для вывода на дисплей.

    LCDOut(Message2);      // Вторая строка сообщения.

    while (1 == 1) {       // Бесконечный цикл.

           // Здесь можно разместить код биологического уровня.

    }
}  // Конец главной программы.
```

Как видим, текст подпрограммы LCDNybble не изменился, а из подпрограммы LCDByte исчезли все вызовы функции задержки.

Появилась новая подпрограмма LCDOut, предназначенная для вывода строки на жидкокристаллический дисплей. В этой подпрограмме после того, как конечный автомат закончил запись предыдущей строки, выполняется копирование указателя на новую строку. Заметим, что при раннем вызове функции LCDOut существует опасность зависания главной программы, поэтому читатель, возможно, захочет добавить к приведенному тексту еще одну подпрограмму для опроса текущего состояния конечного автомата, управляющего процессом записи строк на жидкокристаллический дисплей:

```
Int LCDPoll ()            // Опрос состояния конечного автомата.
{
    if (LDState)
           return 0;      // ЖКИ занят или еще не проинициализирован.
    else
           return 1;      // Можно начинать запись следующей строки.

}
```

Эта функция возвращает 1, если дисплей пока не проинициализирован или еще не закончена запись предыдущей строки. Теперь вызов LCDOut можно поместить в условный оператор, который обращается к функции LCDPoll.

Обратите внимание еще на несколько отличий от предыдущей программы. Во-первых, здесь мы используем строки в формате ASCIIZ, то есть применяем нулевой байт в качестве терминатора строки. Во-вторых, строка сообщения теперь содержит управляющие коды. Символ \f (form feed) используется для перехода на новую строку, что для нашего однострочного дисплея эквивалентно очистке экрана. Перед записью таких команд необходимо послать в контроллер ЖКИ байт с кодом 254 (то есть OxOFE). Этот байт воспринимается большинством жидкокристаллическихдисплеев как указатель того, что за ним последует не ASCII-код символа, а код команды (как вы помните, первые 32 символа, соответствующие обычным управляющим кодам, не являются таковыми для жидкокристалличес-ких дисплеев, поэтому приходится использовать какой-то условный сигнал для подачи команд управления экраном).

В программе на языке С самый простой способ подачи таких команд - предварять каждый управляющий код символом \376 (как нетрудно проверить, это восьмеричная запись числа 254). Например, команда очистки экрана, эквивалентная команде 0x001, запишется в виде строки из двух символов: \376\001.

В табл. 4.10 приведены некоторые часто используемые команды.

Таблица 4.10. Строковый формат некоторых команд ЖКИ

Команда	*Строка*
Очистка экрана	*\376\001*
Перемещение курсора в начальную позицию экрана	*\376\002*
Выключение дисплея	*\376\010*
Включение дисплея, курсор спрятан	*\376\014*
Включение дисплея, видимый курсор	*\376\016*
Перемещение курсора на позицию, адрес которой — 0x040	*\376\300*

Допустим, мы хотим очистить экран, вывести в двух строках сообщение о том, что робот движется вперед:

```
Robot Moving
Forward
```

и спрятать курсор. Для этого в программе на языке С надо использовать строку, содержащую управляющие символы:

```
"\fRobot Moving\376\300Forward\376\014\0"
```

Возможно, вы полагаете, что метод проектирования, основанный на вынесении интерфейсных функций в процедуру обработки прерываний, эффективен только с точки зрения расширяемости программ, но требует большего объема памяти микроконтроллера. Пора развеять это заблуждение. Если вы сравните размер машинного кода, полученного в результате компиляции обоих вариантов нашей программы, то будете немало удивлены. Хотя исходный текст программы `lcd2` на языке С занимает больше места и содержит дополнительные функции (в частности, добавлены новые ветви оператора выбора для обработки управляющих символов), размер

машинного кода после всех изменений получается даже чуть меньше, чем в результате компиляции программы `lcd`.

Разумеется, вы захотите использовать описанные в этой главе методы работы с периферийными устройствами при проектировании реальных приложений, в которых должны будут уживаться друг с другом разные интерфейсы. Поэтому за основу следующих проектов будет взята описанная в этом разделе конструкция, чтобы можно было продемонстрировать, как работает предложенный метод в ситуации, когда новые функциональные возможности добавляются к уже работающему устройству. Для экономии места в следующих программах будут приводиться только заголовки тех функций, которые обеспечивают вывод информации на жидкокристаллический дисплей, а пропущенные фрагменты кода — обозначаться многоточием.

4.18. ДАТЧИКИ

При слове «датчик» я всегда вспоминаю телевизионный сериал «Star Trek», в котором эти таинственные устройства могли обнаруживать вещество, энергию и, конечно, различные формы жизни.

Датчики, используемые в робототехнике, могут служить источником самой различной информации об окружающей среде, но сам алгоритм обработки этой информации определяется не датчиком, а кодом верхнего (биологического) уровня.

В следующих нескольких разделах я продемонстрирую примеры подключения различных датчиков к микроконтроллерам и приведу текст соответствующих программ. Существует множество различных способов реализации описанных функций; в других книгах этой серии многие вопросы разобраны более подробно. Здесь же я ставил себе задачу познакомить читателя с датчиками различного типа и рассмотреть основные методы программирования. Многие входные устройства - например, устройства ввода навигационной информации от спутника (Global Positioning System - GPS) или от обычного компаса, приборы измерения наклона, видеокамеры и др. - при этом вынужденно остались за пределами нашего внимания.

Несмотря на огромное количество самых разнообразных устройств ввода информации, общий принцип работы с ними остается неизменным. В любом случае вы должны опираться на рассматриваемый в этой книге метод «трех уровней».

Для упрощения отладки устройств разумно разместить поблизости от каждого входного датчика какое-либо устройство индикации, которое бы подключалось к нему непосредственно или через специальный переключатель и обеспечивало бы обратную связь, предоставляя разработчику необходимую информацию о состоянии датчика. Вы сэкономите много часов, если поверите, что входные сигналы часто не соответствуют тому, что вы о них думаете.

Разумеется, при этом не следует забывать, что в результате работы индикаторов может проявиться другая - паразитная - обратная связь, в результате действия которой входные датчики будут воспринимать информацию не столько от

окружающей среды, сколько от выходных устройств, которые сигнализируют об их собственном состоянии. Кроме того, следует добиваться того, чтобы ваше присутствие как можно меньше влияло на поведение робота. Во многих случаях ваше устройство должно реагировать только на окружающую обстановку и игнорировать любые формы жизни вблизи себя.

4.19. МЕХАНИЧЕСКИЕ ДАТЧИКИ

Подвижные автоматические устройства должны иметь возможность определять присутствие объектов на своем пути и вокруг себя, например, для того, чтобы выбрать маршрут движения или избежать столкновения. Самый простой способ реализации этой возможности - использование механических контактов. Вообще говоря, я считаю, что робот с контактными усиками выглядит несколько неуклюже, да и в большинстве случаев трудно обеспечить достаточную надежность их работы, но иногда без них просто не обойтись.

Принцип работы контактных детекторов поясняется на рис. 4.48. Длинный отрезок тонкой гибкой проволоки при прикосновении к какому-либо предмету приводит в действие микропереключатель. Длина усика и, соответственно, расстояние, на котором срабатывает такой датчик, должна быть достаточной для того, чтобы робот успел изменить направление движения или остановиться. Кроме того, в данной конструкции проволока играет роль рычага, позволяющего даже малому усилию, приложенному со стороны длинного его конца, вызвать переключение механических контактов.

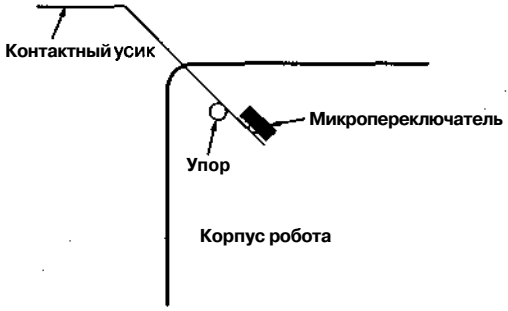

Рис. 4.48. ***Контактный*** детектор столкновений

Подобные детекторы обычно устанавливаются спереди, но в некоторых случаях они не помешают по бокам и сзади, чтобы предотвратить возможные столкновения с предметами, когда робот осуществляет маневр, резко изменяя направление движения или разворачиваясь.

Следует иметь в виду, что при срабатывании механического переключателя его контакты в течение нескольких первых миллисекунд входят в соприкосновение друг с другом сотни раз, генерируя шумовой сигнал, показанный на рис. 4.49. Это

явление называется *дребезгом контактов*. Поэтому приходится применять специальные методы подавления дребезга. В противном случае может возникнуть ситуация, когда робот, правильно прореагировав на первый импульс, будет вынужден повторить реакцию и на следующие паразитные импульсы, которые на самом деле не являются результатом повторного срабатывания датчика.

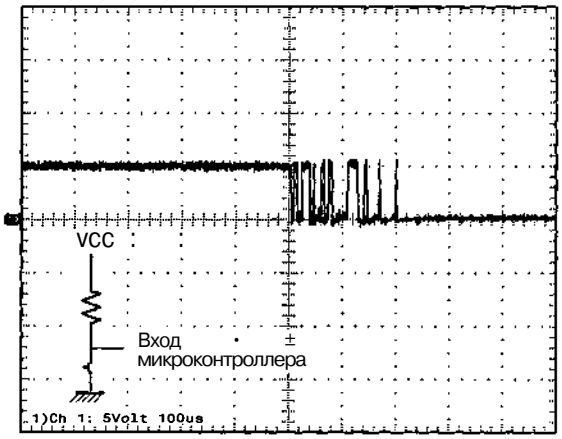

Рис. 4.49. *Так* выглядит дребезг механических ***контактов*** на *экране осциллографа*

Обычно при подавлении эффекта дребезга механических контактов считают, что их состояние не может измениться за время около 20 мс, то есть все изменения, происходящие за этот промежуток времени после первого срабатывания, не учитывают. Вносимая в результате работы этого алгоритма задержка обычно вполне допустима. Можно предложить для подавления дребезга следующий фрагмент кода:

```
while (1==1) {           // Бесконечный цикл

   ...     // Выполнение операций до срабатывания переключателя.

   if (Button == Press){ // Ждем, чтобы замкнувшиеся контакты остались
                         // в том же состоянии в течение 20 мс.
      Debounce = 0;
      while ( ( Debounce != 20msecs ) && ( Button == Press ) )
          for (Debounce=0; (Button==Press) && (I<20msecs); Debounce++);

      // Здесь надо разместить код
      // для реагирования на замыкание контактов.

      Debounce = 0;      // Ждем, чтобы разомкнувшиеся контакты остались
                         // в том же состоянии в течение 20 мс.
      while ( ( Debounce != 20msecs ) && ( Button == Release ) )
          for (Debounce=0; (Button==Press) && (I<20msecs); Debounce++);

   } // Конец оператора if.
```

```
    ...     // Выполнение операций после срабатывания переключателя.
}   // Конец оператора while.
```

Если вы сделаете контактный детектор не слишком удачно, то может оказаться, что он срабатывает при движении робота. При использовании микропереключателя от ложных срабатываний может защитить небольшая пружина, удерживающая контакты в разомкнутом состоянии. Если вместо микропереключателя ваша схема представляет собой пару механических контактов (один из которых подсоединен к общей шине питания, а второй - к «подтягивающему» резистору, подключенному к положительной шине питания), то использование описанной выше процедуры подавления дребезга приведет к тому, что ваш робот не будет реагировать на любые кратковременные удары. Кроме того, вам может не понравиться, что противодребезговая процедура после выполнения фрагмента кода, который производит необходимые действия после замыкания контактов, ждет, пока они снова разомкнутся.

Чтобы избежать всех этих проблем, желательно использовать логический флажок, устанавливая его на период времени около 20 мс, в течение которых мы игнорируем все изменения состояния контактов, а для отсчета времени использовать таймер микроконтроллера. Тогда противодребезговый фрагмент кода в главной программе будет выглядеть следующим образом:

```
while (1 == 1) {                        // Бесконечный цикл.
    ...     // Выполнение операций до срабатывания переключателя.
    if (ButtonPressFlag) {          // Ждать замыкания контактов.
          // Здесь надо разместить код
          // для реагирования на замыкание контактов.

    } // Конец оператора if.
    // Выполнение операций после срабатывания переключателя.
}   I/ Конец оператора while.
```

Соответствующая процедура обработки прерываний от таймера TMRO:

```
void interrupt tmr0_int (void)
{
   if (T0IF) {           // Если запрос был от таймера TMRO.
         T0IF =0;        // Сбросить флаг прерывания.
         RTC++;           // Инкремент счетчика реального времени.
         ...             // Здесь можно разместить код для реализации
                         // других действий, которые необходимо
                         // выполнять каждую миллисекунду.
         if (Button == Press){ // Если контакты замкнулись.
               if (ButtonReleaseCounter != 0) { // В первый раз?
                     ButtonPressCounter = 0;
                     ButtonReleaseFlag  =0;
               } else
                     if (++ButtonPressCounter >= 20) {
                        .    ButtonPressFlag =1;
                             ButtonPressCounter = 19;
```

```
                    > else ;
            } else {                // Если контакты разомкнулись.
                  if (ButtonPressCounter != 0) {   // В первый раз?
                        ButtonReleaseCounter =0;
                        ButtonPressFlag = 0;
                  } else
                        if (++ButtonReleaseCounter >= 20) {
                              ButtonReleaseFlag = 1;
                              ButtonReleaseCounter  =  19;
                        } else;
            }
      }
>           // Конец обработчика прерываний.
```

Здесь при замыканий контактов флаг ButtonReleaseFlag сбрасывается и начинается отсчет времени (каждую миллисекунду выполняется инкремент счетчика ButtonPressCounter), который продолжается, пока контакты не будут разомкнуты. При очередном вызове обработчика, если контакты все еще находятся в замкнутом состоянии, а флаг ButtonReleaseFlag сброшен, устанавливается флаг ButtonPressFlag; это значение используется в главной программе и сигнализирует о том, что в данный момент выполняется противодребезговая задержка.

После размыкания контактов флаг ButtonReleaseFlag сбрасывается в нулевое состояние и снова делается задержка на 20 мс, только теперь с каждой миллисекундой выполняется инкремент счетчика ButtonReleaseCounter.

Следует помнить, что контактные усики, которые вы используете в своих конструкциях, должны быть металлическими, и их обязательно следует заземлять. Дело в том, что при движении робота в результате трения различных материалов друг о друга образуется статический заряд, который в непроводящем материале (в случае использования, например, пластмассовых усиков) накапливается до тех пор, пока при очередном касании какого-либо объекта не вызовет электрический пробой, а это может привести к повреждению электронных схем.

Наряду с микропереключателями в качестве детекторов касания можно использовать и другие приспособления:

- заземленное металлическое кольцо, которое при столкновении робота с каким-либо объектом деформируется и переключает уровень сигнала на одном из расположенных по периметру робота металлических контактов, соединенных через «подтягивающий» резистор *с* положительной шиной питания. Такой способ хорош, когда требуется определить, в каком именно месте произошло касание робота о внешний объект. Но если кольцо имеет достаточную массу, то из-за инерции при резкой остановке или в начале движения оно может вызывать ложные срабатывания детекторов;
- резиновая трубка вокруг робота, которая позволяет при касании изменить внутреннее давление, действующее на контакты. В такой конструкции трудно обеспечить включение датчика, зато трубка работает в качестве амортизатора и заметно снижает чувствительность к различным вибрациям и толчкам, возникающим при движении робота;

- датчик останова двигателя также может быть использован в качестве детектора столкновений. Если робот при своем движении столкнулся с каким-либо объектом, то он останавливается, что приводит к торможению двигателя. Однако при этом через обмотку двигателя течет значительный ток, создавая риск повреждения робота (из-за сильного столкновения) или электронной схемы, которая управляет скоростью вращения двигателя. К тому же данный метод снижает срок службы батарей питания.

Следует заметить, что при использовании контактных датчиков разработчик должен потратить много времени на проведение кропотливых экспериментов, прежде чем ему удастся подобрать правильную длину и форму контактного усика. В моих многочисленных экспериментах даже после того, как конструкция датчика, наконец, приближалась к некоторому идеалу, все рушилось, когда робот на полной скорости сталкивался с каким-либо предметом или кто-нибудь из детей хватал робота за проволочный усик.

Когда есть такая возможность, я предпочитаю использовать неконтактные способы обнаружения объектов, например обсуждаемые ниже инфракрасный детектор или ультразвуковой измеритель расстояния. Тем не менее многие разработчики рассматривают контактный метод как самый надежный: ведь при использовании любого бесконтактного датчика всегда существует опасность не заметить какой-либо объект, который не отражает сигнал, излучаемый роботом.

4.20. ПОДАВЛЕНИЕ ДРЕБЕЗГА КОНТАКТОВ

Одним из первых соображений, которые побудили меня предложить использование прерываний от таймера для реализации функций электронного и механического уровней, была как раз мысль о том, что такой подход позволит без излишних хлопот и траты процессорного времени подавлять влияние дребезга механических переключателей, часто применяемых в различных автоматических устройствах в качестве источников входных сигналов. Если в нашем распоряжении имеется процедура, которая выполняется каждую миллисекунду, то реализация необходимой задержки, требуемой для подавления дребезга, существенно упрощается. В качестве примера приведем фрагмент обработчика прерываний от таймера:

```
if (!Pin)          // Если на выводе микроконтроллера
                   // сигнал низкого уровня,
                   // соответствующий замкнутым контактам,
    if (ButtonPress < 20)
          ButtonPress++;
    else;          // то подавляем дребезг при замыкании контактов.

else               // Если сигнал высокого уровня
                   // (то есть контакты разомкнуты),
    if (ButtonPress != 0)
          ButtonPress--;
    else;          // то подавляем дребезг при размыкании контактов.
```

Здесь используется условный оператор, который при замыкании контактов начинает отсчет времени, выполняя инкремент счетчика ButtonPress, пока его значение не достигнет 20 (это означает, что контакты оставались замкнутыми в течение 20 мс), после чего значение счетчика больше не изменяется. Но как только контакты разомкнутся (уровень сигнала на выводе микроконтроллера станет высоким), начинается обратный отсчет - каждую миллисекунду значение счетчика уменьшается на 1, пока не достигнет 0.

В этом случае код верхнего уровня будет выглядеть следующим образом:

```
if (ButtonPress == 0)
{
   // Выполняем действия, соответствующие разомкнутым контактам.
}
else
   if (ButtonPress == 20)
   {
         // Выполняем действия, соответствующие замкнутым контактам.
   }
   else
   {
         // Еще длится противодребезговая задержка.
   }
```

Оба приведенных фрагмента должны повториться в программе по одному разу для каждого имеющегося в конструкции механического переключателя.

Чтобы проиллюстрировать этот способ подавления дребезга в реальном приложении, я собрал схему, показанную на рис. 4.50.

За основу взят наш предыдущий проект с жидкокристаллическим дисплеем. Единственное, что добавлено к прошлой схеме, - резистор, светодиод и кнопка.

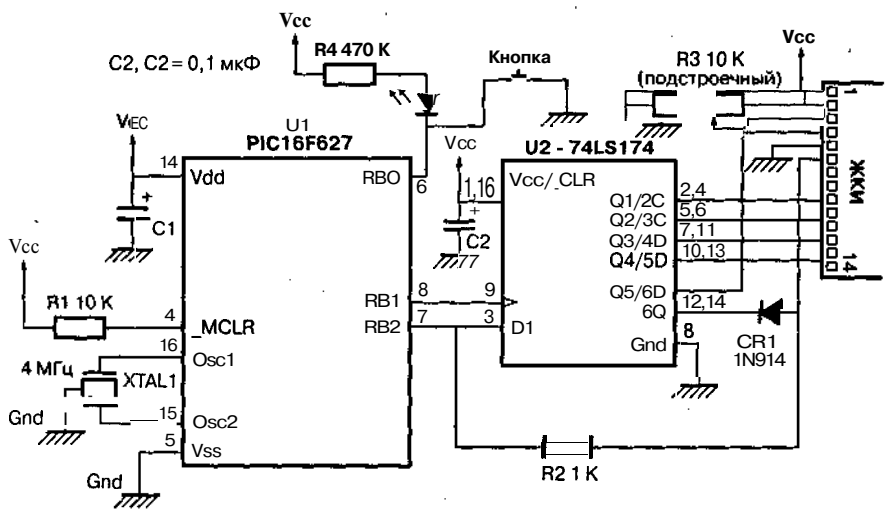

Рис. 4.50. *Схема с* **механическими** *переключателями*

Разумеется, ни дисплей, ни светодиод не нужны для работы противодребезговой процедуры, но я хотел показать, как различные интерфейсные модули работают в одном приложении.

Мы будем использовать дисплей для индикации состояния кнопки. Когда кнопка отжата, на выводе RBO микроконтроллера благодаря «подтягивающему» резистору действует сигнал высокого уровня. В этом случае дисплей будет выключен. При нажатии кнопки начинает работать противодребезговая процедура, и на дисплее высвечивается надпись `"DEBOUNCING"`. Спустя 20 мс, если кнопка все еще нажата, появляется надпись `"PRESSED"`. В главной программе опрашивается значение счетчика ButtonPressed, и в зависимости от этого выбирается одно из трех возможных действий.

Когда ваше устройство заработает, то вы увидите, что надпись `"DEBOUNCING"` появляется на краткий промежуток времени после нажатия и после отпускания кнопки.

Схема размещения элементов на макетной плате показана на рис. 4.51, который практически совпадает с рис. 4.47.

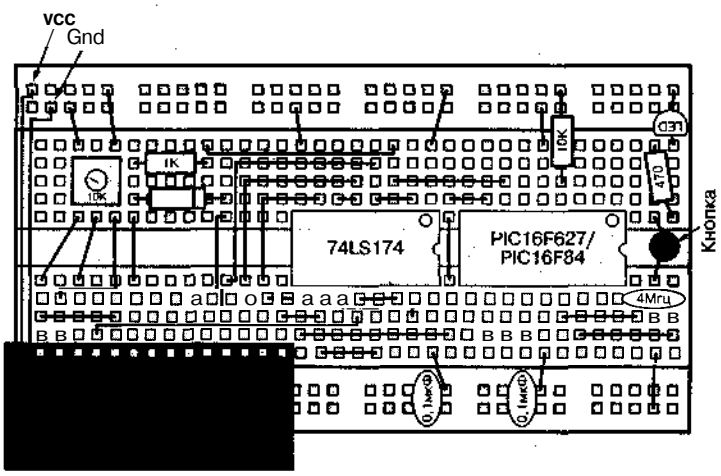

Рис. 4.51. Схема размещения элементов

Желательно взять самый дешевый микропереключатель, чтобы проверить работу противодребезговой процедуры в наиболее неблагоприятных условиях.

Список используемых элементов приведен в табл. 4.11.

Таблица 4.11. Список используемых элементов

Обозначение	Элемент	Примечание
U1	PIC16F627 или PIC16F84	Микроконтроллер
U2	74LS174 (K555TM9)*	Сдвиговый регистр
CR1	1N914 (КД521)*	Любой кремниевый диод

Таблица 4. 11. Список используемых элементов (окончание)

Обозначение	Элемент	Примечание
CR2	Светодиод	Любого типа
R1	10 кОм; 0,25 Вт	Для «подтягивания» напряжения на выводе _MCLR до напряжения положительной шины питания
R2	1 кОм; 0,25 Вт	
R3	10 кОм, подстроечный	Для регулирования контрастности
R4	470 Ом; 0,25 Вт	
C1, C2	0,1 мкФ	Для фильтрации напряжения питания микросхем
	Микропереключатель	С нормально разомкнутыми контактами
ЖКИ	Жидкокристаллический дисплей (14 выводов), совместимый с Hitachi 44780	
XTAL1	Керамический резонатор на 4 МГц, со встроенными конденсаторами	Для генератора тактовых импульсов микроконтроллера
	Материалы	Макетная плата, монтажные провода, источник питания +5 В

*В скобках указаны отечественные аналоги, добавленные при переводе. -Прим. перев.

Приведенный ниже текст программы вы найдете в файле Code\Whisker\whisker.c:

```
#include <pic.h>

// 19 апреля 2002 - подавление дребезга контактов.
//
// Для индикации состояния кнопки используются светодиод
// и жидкокристаллический дисплей, подключенный к микроконтроллеру
// через двухпроводной интерфейс на основе сдвигового регистра 74LS174.
//
// Используются прерывания от таймера TMR0.
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627/PIC16F84 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RB0 - кнопка (с внешним "подтягивающим" резистором
// (внутренние "подтягивающие" резисторы порта PORTB
// микроконтроллера не используются);
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея;
//

// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.
volatile char LCDDlay = 20;     // Длительность задержки.
volatile char LCDState = 1;     // Номер текущего состояния
                                // конечного автомата.
```

```
static volatile bit Clock        @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS    @ (unsigned)&TRISB*8+1;
static volatile bit Data         @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS     @ (unsigned)&TRISB*8+2;
char * MessageOut; // Указатель строки выводимого сообщения.
volatile char MessageOuti = 0;   // Смещение, указывающее позицию
                                 // выводимого символа.

char Message1[2] = "\f";         // Очистка дисплея.
char Message2[12] = "\fdebouncing";
char Message3[9] = "\fPRESSED";

volatile char ButtonPress = 0;   // Счетчик
                                 // для противодребезговой процедуры.

// Слово конфигурации:
#if defined (_16F84)
   #warning PIC16F84 selected
   _CONFIG(0x03FF1);    // Для МК PIC16F84:
                        // кварцевый тактовый генератор,
                        // таймер PWRT включен,
                        // сторожевой таймер выключен,
                        // защита кода отключена.

#elif defined(_16F627)
   #warning PIC16F627 with external oscillator selected
   __CONFIG(0x03F61);   // Для МК PIC16F627:
                        // внешний тактовый генератор XT,
                        // RA6/RA7 используются для ввода-вывода,
                        // внешний сигнал сброса,
                        // таймер PWRT включен,
                        // сторожевой таймер выключен,
                        // защита кода отключена,
                        // детектор BODEN включен.
#else
   #error Unsupported PICmicro MCU selected
#endif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS)         // Запись полубайта в ЖКИ.

   ...

LCDByte(char Byte, char RS)             // Запись байта в ЖКИ.

   ...

LCDInit()                               // Инициализация ЖКИ.

   ...

LCDOut(char * const LCDString)   // Вывод строки на ЖКИ.

   ...

// Обработчик прерываний:
void interrupt tmr0_int(void)
{

char temp;
```

```
if (TOIF) {          // Обработчик прерываний от таймера TMRO.
      TOIF = 0;      // Сбросить флаг прерываний от таймера TMRO.
      RTC++;         // Инкремент счетчика реального времени.
      // Здесь можно разместить дополнительный код
      // для обработки прерываний от таймера TMRO.
      // Начало противодребезговой процедуры:
      if (!RBO)            // Кнопка нажата.
            if (ButtonPress < 20)
                  ButtonPress++;
            else;
      else                 // Кнопка отжата.
            if (ButtonPress != 0)
                  ButtonPress--;
            else;
      // Конечный автомат для ЖКИ;
      switch(LCDState) { // В зависимости от текущего состояния:
            case 1:              // Начать инициализацию ЖКИ.
                  if (--LCDDlay == 0)
                        LCDState++;
                  break;         // Ждать .20 мс.
            case 2:                        // Шаг 2.
                  LCDNybble(0x003, 0);
                  LCDDlay = 5;
                  LCDState++;
            case 3:              // Ждать выполнения команды.
                  if (--LCDDlay == 0)
                        LCDState++;
                  break;
            case 4:
                  LCDNybble(0x003, 0);  // Шаг 3.
                  LCDState++;
                  break;
            case 5:
                  LCDNybble(0x003, 0);  // Шаг 4.
                  LCDState++;
                  break;
            case 6:
                  LCDNybble(0x002, 0);  // Шаг 5.
                  LCDState++;
                  break;
            case 7:
                  LCDByte(0x028, 0);    // Шаг 6.
                  LCDState++;
                  break;
            case 8:
                  LCDByte(0x008, a);    // Шаг 7.
                  LCDState++;
                  break;
            case 9:
                  LCDByte(0x001, 0);    // Шаг 8.
                  LCDState++;
                  LCDDlay = 5;          // Ждать 5 мс.
```

```
                        break;
                case 10:                // Ждать выполнения команды.
                        if (--LCDDlay == 0)
                                LCDState++;
                        break;
                case 11:
                        LCDByte(0x006, 0);      // Шаг 9.
                        LCDState++;
                        break;
                case 12:
                        LCDByte(0x00E, 0);      // Шаг 10.
                        LCDState = 0;           // Все готово.
                        break;

                case 100:       // Вывод сообщения.
                        switch (temp = MessageOut[MessageOuti++]) {
                                case "\0":      // Конец сообщения?
                                        LCDState = 0;
                                        MessageOuti = 0;
                                        break;
                                case "\f":      // Очистка дисплея.
                                        LCDByte(0x001, 0);
                                        LCDState++;
                                        LCDDlay = 5;
                                        break;
                                case 254:       // Команда?
if ((temp = MessageOut[MessageOuti++]) == 0)
                                        LCDState = 0;
                                else {
                                        if (temp < 4) {
                                                LCDState++;
                                                LCDDlay = 5;
                                        }
                                        LCDByte(temp, 0);
                                }
                                break;
                                default:        // Обычные символы.
                                        LCDByte(temp, 1);
                        } // Конец внутреннего оператора switch.
                        break;
                case 101:                               //  Message Delay
                        if (--LCDDlay == 0)
                                LCDState-;
                        break;

        } // Конец внешнего оператора switch.

    } // Конец обработчика прерываний от таймера.

    // Здесь можно разместить другие обработчики прерываний.

} // Конец обработчика прерываний.

// Главная программа:

void main(void)
Ч
```

```
    OPTION = 0x0D1;         // Предделитель работает с таймером TMR0,
                            // коэффициент деления равен 4.

    TMR0 = 0;               // Начальный сброс таймера TMR0.
    T0IE = 1;                 // Разрешение прерываний от таймера TMR0.
    GIE = 1;                // Разрешение обработки прерываний.

    LCDInit();      // Инициализация порта, к которому подключен ЖКИ.

    // Здесь можно проинициализировать другие периферийные устройства.

    while (1 == 1) {        // Бесконечный цикл.

            // Здесь можно разместить код биологического уровня.

            switch(ButtonPress) { // Каково состояние кнопки?

                    case 0:         // Кнопка отжата.
                            LCDOut(Message1);
                            break;
                    case 20:        // Кнопка нажата.
                            LCDOut(Message3);
                            break;
                    default:        // Длится противодребезговая задержка.
                            LCDOut(Message2);
                            break;
            }

    }
}   // Конец главной программы.
```

Чтобы убедиться в правильности работы этой программы, желательно выполнить ее моделирование с помощью симулятора MPLAB и только потом подключать микроконтроллер к программатору.

Для имитации нажатий кнопки можно использовать асинхронные стимулы (порядок работы с ними уже обсуждался в главе 3).

После успешной компиляции проекта выполните команду **Debug ⇒ Asynchronous Stimulus** (Отладка ⇒ Асинхронные входные воздействия) - рис. 4.52. В появившемся диалоговом окне (рис. 4.53) надо щелкнуть правой кнопкой мыши по изображению любой кнопки, а затем указать режим работы **Toggle** (Переключатель) и имя вывода RB0, как показано на рис. 4.54.

Теперь состояние вывода RBO будет переключаться на противоположное каждый раз, когда во время отладки в этом диалоговом окне вы будете щелкать правой кнопкой мыши по изображению выбранной кнопки.

Как уже говорилось в главе 3, асинхронные стимулы очень удобны для моделирования внешних воздействий, которые могут произойти в любой момент. Но во многих случаях, когда требуется подавать входные сигналы с заданными задержками во времени, лучше прибегнуть к другим способам моделирования, имеющимся в MPLAB. Мы обязательно используем их в следующих проектах.

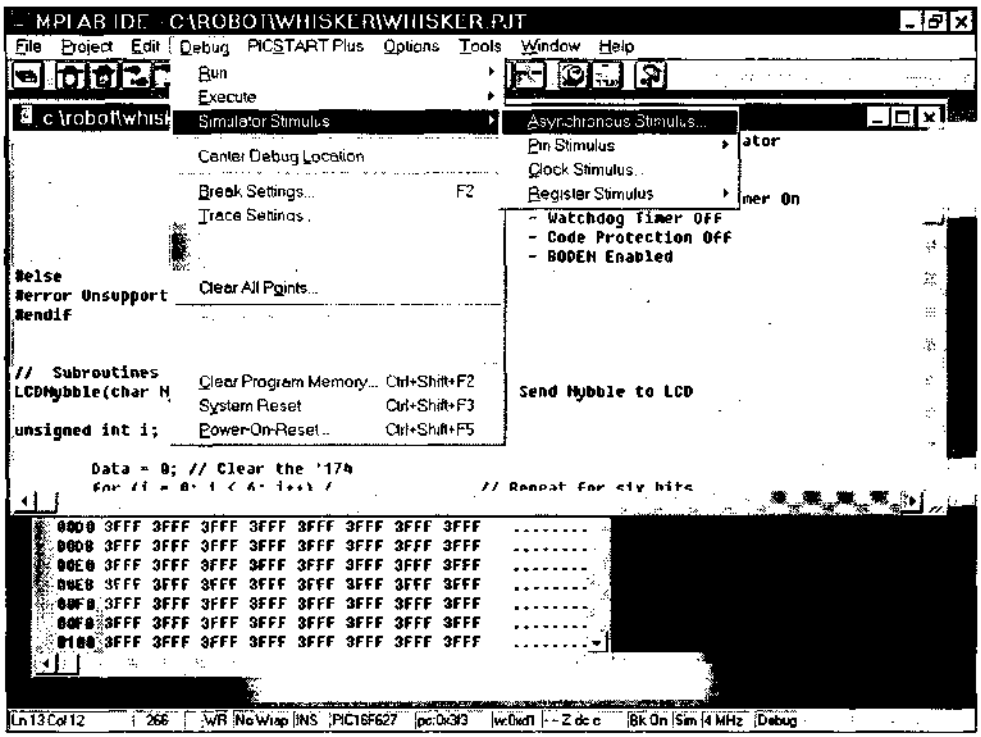

Рис. 4.52. *Выбор* асинхронных входных воздействий

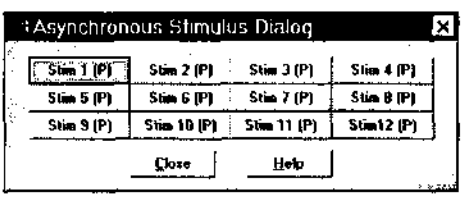

Рис. 4.53. Диалоговое окно выбора асинхронных входных воздействий

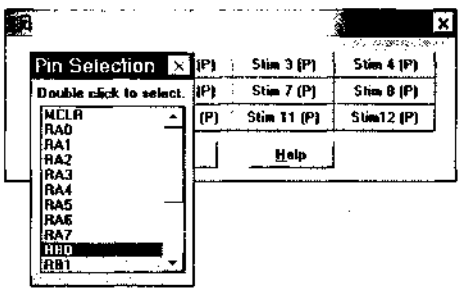

Рис. 4.54
Указание вывода микроконтроллера, на который будет **подаваться** входное воздействие

4.21. ИНФРАКРАСНЫЙ ДЕТЕКТОР СТОЛКНОВЕНИЙ

Разумеется, у вас дома найдется хотя бы один инфракрасный пульт дистанционного управления телевизором или каким-нибудь другим электронным прибором.

Когда я впервые приобрел такой пульт лет 20 назад, то испытал его работу в различных условиях, чтобы выяснить, с какого расстояния он начинает действовать, проходит ли его сигнал через стену и другие преграды, такие как картины, стекла, растения и т.д.

Кому-то такое поведение могло бы показаться странным, но в результате подобных испытаний родилась идея использовать инфракрасный излучатель и приемник отраженного сигнала в качестве детектора столкновений.

Принцип работы инфракрасного детектора поясняется на рис. 4.55. На корпусе робота устанавливаются инфракрасный излучатель и приемник. Если какой-либо объект, который отражает инфракрасные лучи, оказывается слишком близко от робота, то отраженный сигнал становится достаточно сильным. Он улавливается приемником и служит сигналом о скором столкновении. В качестве источника инфракрасных лучей обычно используется светодиод инфракрасногодиапазона.

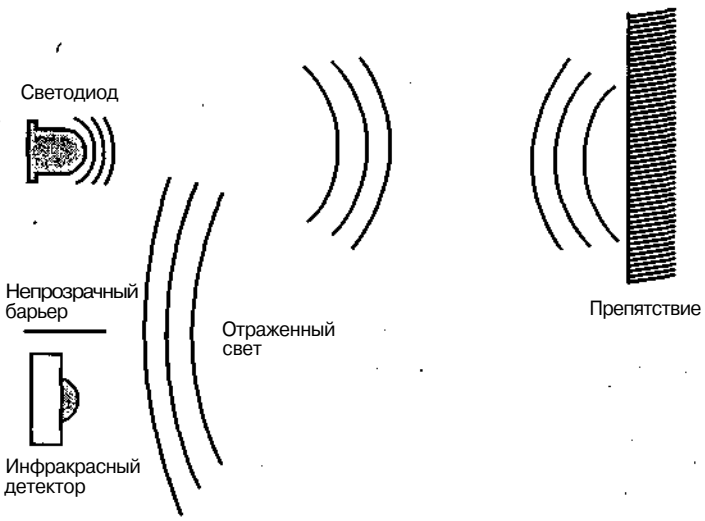

Рис. 4.55. *Принцип работы инфракрасного* **детектора**

Следует помнить, что существует огромное количество твердых и непрозрачных для обычного света материалов, которые могут оказаться прозрачными для инфракрасных лучей[1]. Прекрасный пример материала, являющегося преградой для инфракрасного излучения, - это металлическая пластина, которую необходимо установить между излучателем и приемником, чтобы последний мог реагировать только на отраженный сигнал. Также можно использовать черную изоляционную ленту или резиновую трубку.

[1] Читатель может, например, убедиться, что используемый для монтажа электронных схем гетинакс прекрасно пропускает инфракрасное излучение. - *Прим. перев.*

Упомянутый бесконтактный метод обнаружения объектов (как и описанная ниже ультразвуковая дальнометрия) имеет несомненные преимущества: ведь в этом случае исключено механическое воздействие на детектор, которое может нарушить его работу. Кроме того, нет необходимости подавлять дребезг механических контактов.

Разумеется, за эти преимущества приходится расплачиваться усложнением аппаратных средств и программного обеспечения. Дело не только в стоимости элементов. Инфракрасный светодиод и фотоприемник не так уж дороги. Но применение данного метода обнаружения объектов накладывает определенные ограничения как на используемый микроконтроллер, так и на управляющую его работой программу. Описанный здесь инфракрасный детектор может иметь широкую сферу применений, но в некоторых ситуациях он может стать источником проблем.

Как известно, интенсивность излучения обратно пропорциональна квадрату расстояния до источника. Например, на расстоянии одного сантиметра от излучателя свет в четыре раза ярче, чем на расстоянии двух сантиметров, и в 16 раз ярче, чем на расстоянии четырех сантиметров.

Чем меньше сила света, падающего на фотоприемник, тем меньший ток течет через него. Чтобы можно было произвольно задать расстояние, на котором должен сработать наш детектор, нужно иметь возможность регулировать яркость излучателя. Существует по крайней мере три способа реализовать такую регулировку:

- использовать подстроечный резистор, включенный последовательно со светоизлучающим диодом (чем больше его сопротивление, тем меньше яркость светодиода);
- изменять частоту излучаемого сигнала (чем дальше она выходит за рабочий диапазон фотоприемника, тем слабее тот реагирует на излучаемый сигнал);
- использовать широтно-импульсную модуляцию излучаемого сигнала (этот способ легко реализовать, если микроконтроллер имеет встроенные средства для формирования ШИМ-сигнала).

Наш детектор должен воспринимать только тот сигнал, что генерируется излучателем, и не должен реагировать на инфракрасное излучение от любого другого источника (например, на солнце, заглянувшее в окно, или нагретую электроплитку). Поэтому приходится использовать узкополосный фильтр, настроенный на ту частоту, которую имеет сигнал, генерируемый излучателем; в нашем случае это частота 38 кГц.

- Для регулирования яркости излучателя пригодится встроенный в микроконтроллер модуль ШИМ - вот почему в одном из предыдущих проектов мы использовали программный метод генерации ШИМ-сигнала, оставив аппаратный модуль для других целей.

Если инфракрасный излучатель ориентирован преимущественно в прямом направлении, то детектор будет срабатывать на достаточном расстоянии от

объектов, расположенных спереди, но дальность срабатывания для объектов, расположенных сбоку, может оказаться существенно меньше. Это можно использовать, разместив один излучатель спереди и два фотоприемника по бокам робота.

Как уже говорилось, напряжение на излучатель подается в виде импульсов определенной частоты (чтобы можно было отфильтровывать паразитное действие других источников инфракрасного диапазона) и длительности (чтобы можно было регулировать яркость излучения). Обычно для срабатывания детектора достаточно нескольких импульсов - от двух до семи (рис. 4.56). Напомним, что светодиод горит при низком уровне напряжения на внешнем выводе микроконтроллера.

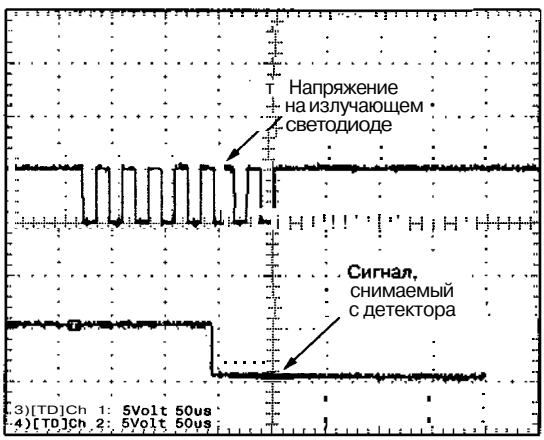

Рис. 4.56. *Диаграммы сигналов* ***излучателя*** *и приемника*

Кроме того, желательно совместить работу инфракрасного обнаружителя объектов и инфракрасного приемника сигналов пульта дистанционного управления. Для этого я обычно генерирую восемь световых импульсов с интервалом в миллисекунду. После восьмого импульса в течение некоторого времени фотоприемник используется для проверки того, не посылает ли пульт дистанционного управления какой-либо сигнал, а затем излучатель снова формирует восемь световых импульсов, необходимых для работы обнаружителя объектов.

4.22. ИНФРАКРАСНЫЙ ОБНАРУЖИТЕЛЬ ОБЪЕКТОВ

Хотя проект, рассматриваемый в этом разделе, даже проще, чем многие другие, представленные в книге, я должен признаться, что процесс написания программы оказался весьма нелегким. Только в результате продолжительных усилий удалось, наконец, добиться правильной работы приложения, и, как обычно, только после этого стала

очевидной его простота. К работающему проекту оказалось несложно подключить другие интерфейсы, такие как приемник сигналов дистанционного управления.

Как и в предыдущем случае, разработка нового устройства будет вестись на основе проекта с жидкокристаллическим индикатором. Это даст возможность в очередной раз продемонстрировать, как программные модули, работающие с различными периферийными устройствами, уживаются друг с другом в одном приложении. Как и прежде, наш обычный подход к разработке управляющих программ облегчит возможность совмещения различных функций.

Для формирования импульсов прямоугольной формы частотой 38 кГц, которые будут подаваться на светодиод инфракрасного диапазона, используется встроенный в микроконтроллер модуль ШИМ. Как уже говорилось, первоначально я разрабатывал свои проекты на основе микроконтроллера PIC16F84, а он не имеет аппаратных средств для генерации широтно-модулированного сигнала. Что касается других МК, то компилятор PICC Lite в то время не поддерживал возможность генерации машинного кода для какого бы то ни было микроконтроллера, имеющего встроенный модуль ШИМ. Поэтому в первом варианте этого проекта для генерации ШИМ-сигнала частотой 32 кГц мне пришлось прибегнуть к программированию на языке ассемблера.

После того как вышла новая версия компилятора PICC Lite, тексты программ были модифицированы для использования с микроконтроллером PIC16F627, который имеет встроенный модуль ШИМ. При тактовой частоте 4 МГц генерация ШИМ-сигнала частотой 38 кГц осуществляется довольно просто:

```
CCPR1L  = 13;                  // Длительность импульса 50%.
CCP1CON = 0b000001111;         // Включение модуля ШИМ.
PR2 = 25;                      // Частота 38 кГц соответствует
                               // периоду примерно 26 мкс.
TMR2 = 0;                      // Сброс таймера TMR2.
T2CON = 0b000000100;           // Включение таймера TMR2,
                               // предделитель 1:1.
TRISB3 = 0;                    // Вывод RB3 (выходной ШИМ-сигнал)
                               // переводится в режим выходного.
```

На рис. 4.57 приведена принципиальная схема, а на рис. 4.58 показана схема размещения элементов на макетной плате.

По сравнению с предыдущими проектами способ прокладки соединительных проводов остался прежним, хотя пришлось внести некоторые изменения, чтобы освободить место для инфракрасного излучателя и подстроечного резистора.

Светодиод инфракрасного диапазона подключен ко входу RB3 микроконтроллера через подстроечный резистор, чтобы можно было не только программно, но и аппаратно регулировать яркость его свечения.

В качестве приемника отраженного сигнала используется детектор с тремя выводами, взятый со стандартного приемника дистанционного управления телевизором или музыкальным центром (рассчитанный на частоту сигнала 38 кГц). Для фильтрации сигнала, снимаемого с детектора, применен простой RC-фильтр, образованный резистором R5 и конденсатором C3. На всякий случай установлен

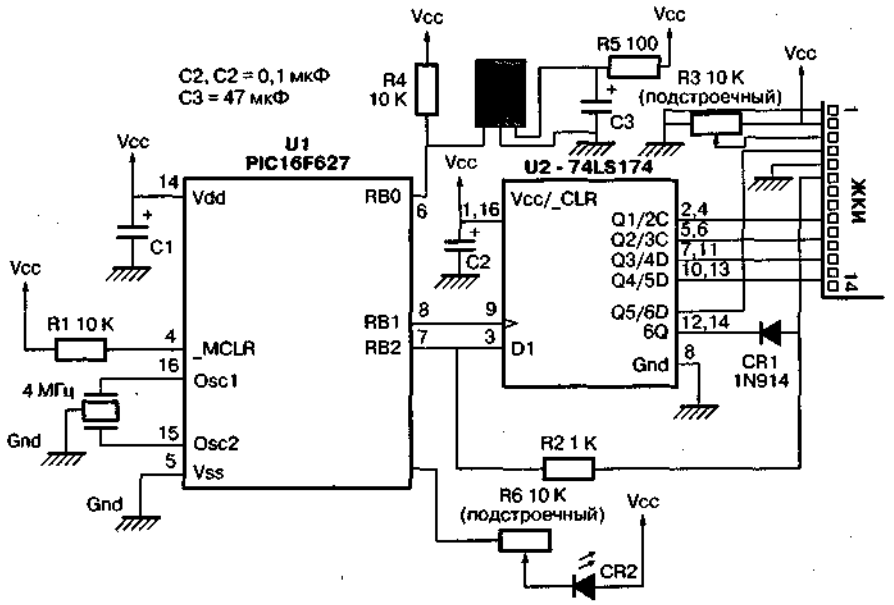

Рис. 4.57. Принципиальная схема *обнаружителя* объектов

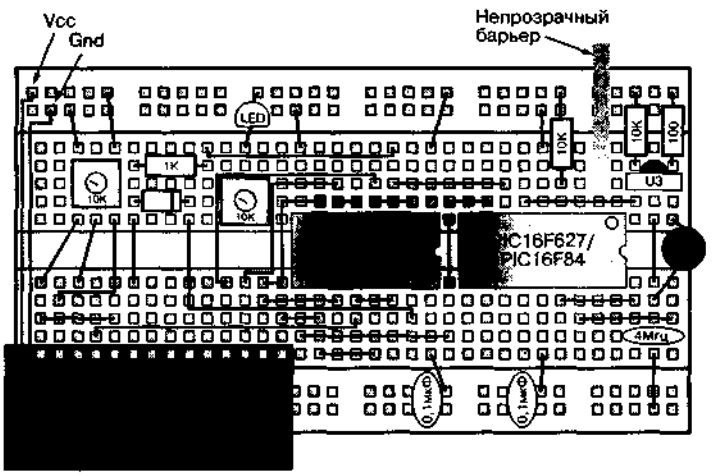

Рис. 4.5S. Схема размещения элементов ***обнаружителя объектов*** на макетной плате

«подтягивающий» резистор R4, так как многие инфракрасные детекторы имеют выход с открытым коллектором.

В табл. 4.12 приведен список использованных в схеме элементов.

Здесь указаны светодиод и детектор инфракрасного диапазона фирмы Waitrony только потому, что в Торонто они являются одними из самых доступных и дешевых, но вы можете взять практически любые детектор и светодиод.

Таблица 4.12. Список использованных элементов

Обозначение	Элемент	Примечание
U1	PIC16F627 или PIC16F84	Микроконтроллер
U2	74LS174(K555TM9)*	Сдвиговый регистр
U3	Детектор P/C-1018SCL фирмы Waitronyили любой другой того же типа	Инфракрасный детектор с тремя внешними выводами
CR1	1N914(КД521)*	Любой кремниевый диод
CR2	Светодиод IE-0530HP или любой другой того же типа	Можно использовать любой светодиод инфракрасного диапазона
RJ, R4	10 кОм; 0,25 Вт	«Подтягивающие» резисторы
R2	1 кОм; 0,25 Вт	
R3, R6	10 кОм, подстроечный	Для регулирования контрастности дисплея и яркости светодиода
R5	100 Ом; 0,25 Вт	Фильтрация сигнала
C1, C2	0,1 мкФ	Для фильтрации напряжения питания микросхем
C3	47 мкФ, 16 В	Фильтрация сигнала
ЖКИ	Жидкокристаллический дисплей (14 выводов), совместимый с Hitachi 44780	
XTAL1	Керамический резонатор на 4 МГц, со встроенными конденсаторами	Для генератора тактовых импульсов микроконтроллера
	Материалы	Макетная плата, монтажные провода, источник питания +5 В

** В скобках указаны отечественные аналоги, добавленные при переводе. — Прим. перев.*

Разумеется, детектор принимаемого сигнала должен быть ориентирован в ту же сторону, что и излучающий светодиод. Между детектором и излучателем необходимо установить непрозрачную для инфракрасных лучей перегородку.

Когда я собрал эту схему, она сначала не заработала (в отличие от всех предыдущих, которые начинали работать сразу же). Потребовалось несколько недель напряженных усилий, прежде чем удалось найти причину. Ошибку удалось обнаружить только после того, как я немного переделал схему, вернувшись к старой конструкции, которая уже доказала свою работоспособность во время создания одного из моих прошлых проектов около года назад.

Оказалось, что инфракрасный детектор рассчитан на работу с более-менее постоянным сигналом и имеет возможность подстраивать встроенный в него фильтр, адаптируя его к частоте принимаемого сигнала. Данная особенность не была указана в документации. В нашей конструкции, как уже говорилось, сигнал делится всего на восемь периодов, после чего на время прекращается (в одном из следующих проектов эта пауза будет использоваться для приема сигналов пульта дистанционного управления), поэтому фильтр внутри инфракрасного детектора не успевает подстроиться к сигналу. Проблема решилась с помощью небольшого фрагмента, добавленного к тексту обработчика прерываний от таймера:

```
TRISB3 = 0;          // Разрешить прохождение ШИМ-сигнала на выход
                     // (светодиод горит).
```

```
while (TMR0 < 64);// Ждать, пока детектор настроится на сигнал.

if (!RB0)                // Если на входе низкий уровень,
   Collision = 1; // то выдача сигнала об обнаружении объекта,
else                     // в противном случае никаких объектов нет.
   Collision = 0;

TRISB3 = 1;              // Запрещение выхода ШИМ-сигнала
                         // (светодиод потушен).
```

Здесь ШИМ-сигнал пропускается на выход микроконтроллера только в течение 64 циклов работы счетчика таймера TMR0. При частоте тактирования 4 МГц и коэффициенте предделителя 1:4 это соответствует промежутку времени длительностью 256 мкс. При частоте 38 кГц за это время формируются восемь или девять импульсов ШИМ-сигнала. В результате своих предыдущих экспериментов с инфракрасными детекторами я пришел к мысли, что, возможно, в детекторе выполняется синхронизация частоты настройки фильтра с частотой принимаемого сигнала.

Пытаясь найти ошибку, я сравнил фрагмент программы, управляющий работой модуля ШИМ, со своей прошлогодней разработкой, в которой использовалась та же самая электрическая схема. Используя приемник старого робота, удалось убедиться в том, что новая схема действительно генерирует сигнал. Что касается программы, то не было никаких сомнений в ее правильности. Не зная, что делать дальше, я попробовал вместо одиночных импульсов посылать пакеты из восьми-девяти импульсов, и тут новая схема неожиданно заработала!

Этот случай прекрасно доказывает, что во время отладки приложения надо проверить *все* возможности, так как любые наши предположения или наблюдения могут оказаться не слишком правильными - либо возникнет обстоятельство, обратить внимание на которое не приходило нам в голову.

Окончательную версию программы вы найдете в файле Code\Irdetect\ irdetect.c. Здесь генерируется ШИМ-сигнал частотой 38 кГц. При наличии отраженного сигнала (то есть когда поблизости имеется какой-либо объект), на жидкокристаллическом дисплее высвечивается надпись COLLISION. Если отраженного сигнала нет, то на дисплей ничего не выводится.

Подстроечный резистор R6 должен быть откалиброван, чтобы можно было устанавливать его движок в нужное положение, соответствующее заданной дальности срабатывания детектора. Прежде чем начать калибровку, следует убедиться, что между светодиодным излучателем и детектором установлен экран, не пропускающий инфракрасное излучение. На рис. 4.58 показано, как следует расположить этот экран. Должен предупредить вас, что для такого экрана не столь уж просто найти подходящий материал. Вполне непрозрачные для видимого света предметы, в частности плотная бумага или ткань, могут оказаться прозрачными для инфракрасных лучей. Рекомендую выбрать фольгу или черную изоляционную ленту.

Для градуировки резистора расположите белую плотную бумагу, например визитную карточку, в четырех-пяти дюймах (то есть на расстоянии 10-12 см) от излучателя и приемника и осторожно перемещайте движок резистора до тех пор,

пока не произойдет срабатывание детектора — иными словами, пока на дисплее не высветится соответствующее сообщение. Теперь, когда при движении робота какой-либо объект окажется на этом расстоянии от детектора, на дисплее будет появляться надпись COLLISION.

Приведем текст программы irdetect.c:

```
«include <pic.h>

// 17 апреля 2002 - инфракрасный обнаружитель объектов.
//
// Для индикации используется жидкокристаллический дисплей,
// подключенный к микроконтроллеру через двухпроводной интерфейс
// на основе сдвигового регистра 74LS174.
//
// Используются прерывания от таймера TMR0.
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RB0 - инфракрасный детектор,
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея;
// RB3 - выход широтно-модулированного сигнала.

// Глобальные переменные и константы:
int RTC = 0;                        // Счетчик реального времени.
volatile char LCDDlay = 20;         // Длительность задержки.
volatile char LCDState = 1;         // Номер текущего состояния
                                    // конечного автомата.

static volatile bit Clock       @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS   @ (unsigned)&TRISB*8+1;
static volatile bit Data        @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS    @ (unsigned)&TRISB*8+2;

char * MessageOut; Указатель на строку выводимого сообщения.
volatile char MessageOuti = 0;    // Смещение, указывающее позицию
                                  // выводимого символа.
char Message1[2] = "\f";          // Очистка дисплея.
char Message2[11] = "\fCOLLISION";

volatile char Collision = 0; // Сначала никаких объектов поблизости нет.
volatile char OldCollision = 0;

// Слово конфигурации:
#if defined(_16F627)
   #warning PIC16F627 with external oscillator selected
   __CONFIG(0x03F61);    // Для МК PIC16F627:
                         // внешний тактовый генератор XT,
                         // RA6/RA7 используются для ввода-вывода,
                         // внешний сигнал сброса,
                         // таймер PWRT включен,
                         // сторожевой таймер выключен,
                         // защита кода отключена,
                         // детектор BODEN включен.
```

```
«else
   Serror Unsupported PICmicro MCU selected
#endif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS)          // Запись полубайта в ЖКИ.

   ...

LCDByte(char Byte, char RS)              // Запись байта в ЖКИ.

   ...

LCDInit()                                // Инициализация ЖКИ.

   ...

LCDOut(char * const LCDString)   // Вывод строки на ЖКИ.

   ...

// Обработчик прерываний:
void interrupt tmr0_int(void)
{

char temp;

   if (T0IF) {           // Обработчик прерываний от таймера TMR0.

         T0IF = 0;       // Сбросить флаг прерываний от таймера TMR0.

         RTC++;          // Инкремент счетчика реального времени.

         // Здесь можно разместить дополнительный код
         // для обработки прерываний от таймера TMR0.

         // Вывести последовательность импульсов
         // для детектора столкновений:

         TRISB3 = 0;     // Разрешить вывод ШИМ-импульсов.

         while (TMR0 < 64);     // В течение 64 «тиков» таймера.

         if (!RB0)                    // Если уровень низкий,
               Collision = 1;         // значит обнаружен обьект;
         else                         // в противном случае
                                      // объектов нет.
               Collision = 0;

         TRISB3 = 1;           // Запретить вывод ШИМ-сигнала.

         // Конечный автомат для ЖКИ:

         switch(LCDState) { // В зависимости от текущего состояния:
               case 1:                // Начать инициализацию ЖКИ.
                     if (--LCDDlay == 0)
                           LCDState++;
                     break;           // Ждать 20 мс.

               case 2:                        // Шаг 2.
                     LCDNybble(0x003,  0);
                     LCDDlay = 5;
                     LCDState++;
               case 3:                // Ждать выполнения команды.
                     if (--LCDDlay == 0)
                           LCDState++;
```

```
                break;
        case 4:
                LCDNybble(0x003, 0);   // Шаг 3.
                LCDState++;
                break;
        case 5:
                LCDNybble(0x003, 0);   // Шаг 4.
                LCDState++;
                break;
        case 6:
                LCDNybble(0x002, 0);   // Шаг 5.
                LCDState++;
                break;
        case 7:
                LCDByte(0x028, 0);     // Шаг 6.
                LCDState++;
                break;
        case 8:
                LCDByte(0x008, 0);     // Шаг 7.
                LCDState++;
                break;
        case 9:
                LCDByte(0x001, 0);     // Шаг 8.
                LCDState++;
                LCDDlay = 5;           // Ждать 5 мс.
                break;
        case 10:              // Ждать выполнения команды.
                if (--LCDDlay == 0)
                        LCDState++;
                break;
        case 11:
                LCDByte(0x006, 0);     // Шаг 9.
                LCDState++;
                break;
        case 12:
                LCDByte(0x00E, 0);     // Шаг 10.
                LCDState = 0;          // Все готово.
                break;

        case 100:       // Вывод сообщения.
                switch (temp = MessageOut[MessageOuti++]) {
                        case "\0":     // Конец сообщения?
                                LCDState = 0;
                                MessageOuti = 0;
                                break;
                        case "\f":     // Очистка дисплея.
                                LCDByte(0x001, 0);
                                LCDState++;
                                LCDDlay = 5;
                                break;
                        case 254:      // Команда?
if ((temp = MessageOut[MessageOuti++]) == 0)
                                        LCDState = 0;
                                else {
                                        if (temp < 4) {
                                                LCDState++;
                                                LCDDlay = 5;
                                        }
```

```
                                        LCDByte(temp, 0);
                                    }
                                    break;
                                default:        // Обычные символы.
                                        LCDByte(temp, 1);
                            } // Конец внутреннего оператора switch.
                            break;
                    case 101:                               //  Message Delay
                            if (--LCDDlay == 0)
                                    LCDState--;
                            break;

            } // Конец внешнего оператора switch.

    } // Конец оператора if.

    // Здесь можно разместить другие обработчики прерываний.

> // Конец обработчика прерываний.

// Главная программа:

void main(void)
{

    OPTION = 0x0D1;         // Предделитель работает с таймером TMR0,
                            // коэффициент деления равен 4.
    TMR0 = 0;               // Начальный сброс таймера THR0.
    TOIE = 1;                // Разрешение прерываний от таймера TMR0.
    GIE = 1;                // Разрешение обработки прерываний.

    LCDInit();      // Инициализация порта, к которому подключен ЖКИ.
    // Здесь можно проинициализировать другие периферийные устройства.

    CCPR1L = 13;                    // Ширина импульсов - 50% периода.
    CCP1CON = 00000001111;                  // Включить модуль ШИМ.

    PR2 = 26;               // При частоте 38 кГц период равен 26 мкс.
    TMR2 = 0;                       // Сброс таймера TMR2.
    T2CON = 0b000000100;            // Включение таймера TMR2,
                                    // коэффициент предделителя 1:1.

    while (1 == 1) {            . // Бесконечный цикл.

            // Здесь можно разместить код биологического уровня.

            if (Collision != OldCollision) {
                    OldCollision = Collision;       // Сохранить текущее
                                                    // состояние детектора.
                    if (Collision)
                            LCDOut(Message2);       // Вывести сообщение.
                    else
                            LCDOut(Message1);
                    while (LCDState);       // Ждать готовности ЖКИ.

            }

    }
} I/ Конец главной программы.
```

Когда вы закончите эксперименты с этим устройством, не разбирайте его, так как оно вам понадобится при разработке следующих двух проектов.

4.23. ДИСТАНЦИОННОЕ УПРАВЛЕНИЕ РОБОТОМ

Всегда желательно иметь какую-либо возможность управлять действиями и/или передвижением робота. Разумеется, для этих целей лучше всего подходит беспроводной метод, например радиоуправление или использование привычного пульта дистанционного управления. Для работы удобен стандартный телевизионный пульт, цифровые кнопки от 1 до 9 которого, расположенные в виде матрицы 3x3, хорошо подходят для выбора направления движения, а кнопки увеличения/уменьшения громкости - для регулирования скорости движения. Оставшиеся незанятыми кнопки (0 и включение питания) можно приспособить для подачи роботу каких-либо команд.

Я использовал телевизионный пульт дистанционного управления фирмы Sony. Стандарт Sony предписывает передачу команд импульсами света инфракрасного диапазона; при этом применяется код Manchester. В манчестерской кодировке информация представляется последовательностью импульсов, длительность каждого из которых может иметь одно из трех возможных значений: T, 2T или 4T, где T = 0,60 мс - период тактовых импульсов кодера. Пример сигнала в манчестерской кодировке представлен на рис. 4.59, а на рис. 4.60 показан сигнал на входе приемника, полученный с помощью осциллографа.

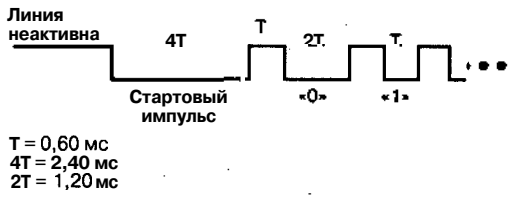

Рис. 4.59. *Манчестерский* код

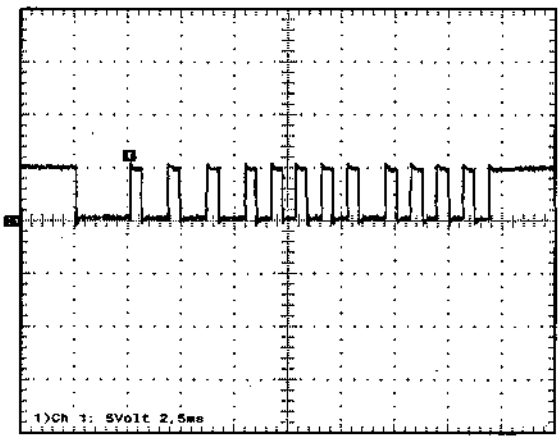

Рис. 4.60. Осциллограмма *сигнала* дистанционного управления

Информационный пакет состоит из 13 импульсов отрицательной полярности. Это значит, что сигнал высокого уровня представляет паузу между импульсами, а сами импульсы кодируются низким уровнем напряжения. Первый импульс низкого уровня, длительностью 4T, является стартовым; за ним следуют информационные. Логический 0 представляется импульсом низкого уровня длительностью 2T, а логическая 1 - длительностью T. Все информационные импульсы отделяются друг от друга синхроимпульсами высокого уровня длительностью T.

В табл. 4.13 показано, какие последовательности импульсов соответствуют различным кнопкам пульта дистанционного управления Sony.

Таблица 4.13. Команды инфракрасного пульта дистанционного управления фирмы Sony

Кнопка	*Код*
0	*0b0011011101111*
1	*ОЬООПОПОПП*
2	*ОЬООППППОПП*
3	*ОЬОПОПППОПП*
4	*ОЬОООПППОПП*
5	*ОЬОПОПППОПП*
6	*0b0010111101111*
7	*ОЬОПООПППОПП*
8	*ОЬОООПППОПП*
9	*ОЬОПППОПОПП*
Увеличить громкость	*ОЬОПОПОПОПП*
Уменьшить громкость	*ОЬОООПОПОПП*
Канал +1	*ОЬОПППОПОПП*
Канал –1	*ОЬООПППОПОПП*
Предыдущий канал	*ОЬОООПООПОПП*
Приглушить звук	*ОЬОПОПОПОПП*
Питание	*ОЬООПОПОПОПП*

Каждый информационный пакет повторяется примерно каждые 50 мс. Передатчик не имеет встроенной функции автоповтора, поэтому он должен обеспечиваться внутри приемника. При разработке дистанционно управляемых автоматических устройств это не является проблемой - ведь повторяющиеся команды удобны для того, чтобы поддерживать выполнение какой-либо функции в течение всего времени, пока нажата соответствующая клавиша. Во многих моих конструкциях двигатель робота, выполнив очередную команду дистанционного управления, остается включенным еще в течение 200 мс, ожидая следующего пакета. Такой защитный интервал предотвращает остановку робота в случае, если следующий пакет данных не будет получен вовремя или связь между передатчиком и приемником на короткое время нарушится.

Описанный здесь принцип работы пульта дистанционного управления довольно прост, но все же у разработчиков могут возникнуть проблемы с пониманием

того, как надлежащим образом декодировать поступающие команды. Наиболее очевидный способ заключается в том, чтобы принимать импульсы и на ходу определять их длительность с помощью следующей программы:

```
void main(void)    // Декодирование команд дистанционного управления.
{
unsigned int i;
unsigned int DataPacketStart; // Время получения начала импульса.
unsigned int DataPacketEnd;   // Время окончания импульса.
unsigned int DataPacket;      // Пакет данных.

while (1 == 1) {        // Бесконечный цикл.
   while (IRData == High); // Ждать прихода импульса низкого уровня.

   DataPacketStart = THR0;   // Время начала импульса низкого уровня.
   while (IRData == low);       // Ждать конца текущего импульса.
   DataPacketEnd = TMR0 - DataPacketStart; // Длительность импульса.

   if ( (DataPacketEnd > 2.2msecs) && (DataPacketEnd < 2.6msecs) ) <
                                    // Следующие 12 бит.
       DataPacket = i = 0;  // Инициализация.
       DataPacketEnd = 1.8msecs;

       while ( (i < 12) && (DataPacketEnd != 0) ) {
              while (IRData == High); // Ждать начала импульса.
              DataPacketStart = TMR0;      // Запомнить время
                                           // начала импульса.
              while (IRData == low);   // Ждать конца импульса.

              DataPacketEnd = TMR0 - DataPacketStart;
                                  // Длительность импульса.
              if ( (DataPacketEnd > 0.45msec) &&
                                   (DataPacketEnd < 0.75msec) )
                     DataPacket = (DataPacket << 1) + 1;
              else
                     if ((DataPacketEnd > 0.95msec) &&
                                   (DataPacketEnd < 1.35msec)
                            DataPacket = (DataPacket «1) + 1;
                     else          // Ошибка.
                            DataPacketEnd = 0; // Конец.
                            i++; // Указатель - на следующий бит.
       }      // Конец внутреннего цикла while.
       if (DataPacketEnd  != 0)
              printf ("Data Packet = %i\n", DataPacket);
   } I/  Конец оператора if.
}  // Конец внешнего оператора while.
> // Конец программы.
```

Этот метод немного громоздкий, но хорошо показывает принцип работы приемника сигналов дистанционного управления. Проблема, однако, в том, что приведенная программа не может быть использована в наших конструкциях: во-первых, нарушены все наши правила о разделении функций различных уровней, а во-вторых, опрос инфракрасного детектора производится постоянно, и мы не сможем совместить прием сигналов дистанционного управления с работой инфракрасного обнаружителя объектов.

Разумней будет, если мы разрешим формирование запроса на прерывание по приходу каждого импульса на вход микроконтроллера, который подключен к инфракрасному детектору, а процедуру измерения длительности этих импульсов I! декодирования принимаемых команд поручим обработчику прерываний. В следующем разделе будет показано, как воплотить эту идею в жизнь.

4.24. ПРИЕМНИК СИГНАЛОВ ДИСТАНЦИОННОГО УПРАВЛЕНИЯ

Как уже упоминалось, многие радиолюбители избегают использования стандартных инфракрасных пультов дистанционного управления в своих устройствах, боясь сложностей, с которыми рассчитывают столкнуться при проектировании приемника и декодера команд. В этом разделе мы рассмотрим пример такого приемника докажем, что эти опасения беспочвенны. Программная реализация нашего декора не окажется слишком сложной, и он будет прекрасно работать совместно с другими интерфейсами, которые уже реализованы в вышеописанных проектах.

В одной из моих предыдущих книг, «Programming and Customizing PICmicro Microcontrollers» (Программирование и использование микроконтроллеров PICmicro), было рассмотрено несколько способов реализации приемника сигналов дистанционного управления на основе различных микроконтроллеров PICmicro. Как всегда, если какая-либо задача уже решена одним способом, то реализация других методов ее решения не вызывает особых проблем.

В самом начале процесса проектирования необходимо четко определить, какие команды должен принимать декодер, в каком виде они будут подаваться и какие возможности микроконтроллера будут использоваться при декодировании.

Если вы читали эту книгу с самого начала, то для вас уже должна быть очевидной формулировка, описывающая работу декодера: *декодер должен принимать сигналы пульта дистанционного управления фирмы Sony и передавать биологическому коду соответствующие команды.*

Здесь необходимо сделать два замечания. Во-первых, желательно выполнять отбраковку ошибочно принятых данных. Если длительность какого-либо импульса слишком отличается от трех разрешенных значений, то разумно будет проигнорировать весь 12-битный пакет. Во-вторых, для простоты можно не принимать в расчет стартовый бит в начале каждого пакета.

Мы уже обсуждали в главе 3 возможные источники прерываний в микроконтроллерах PICmicro. Поэтому без лишних комментариев выберем для нашего проекта вывод 6 микроконтроллера PIC16F84/PIC16F627 (RBO/INT), который может быть использован для формирования запроса на прерывание при каждом изменении уровня входного сигнала.

Необходимо, чтобы обработчик прерывания вызывался каждый раз, когда уровень принятого инфракрасным детектором сигнала изменяется с низкого на высокий, как показано на рис. 4.61. Тогда, измеряя время, прошедшее с момента последнего вызова, можно оценить ширину импульса и таким образом отличить 0 от 1.

Для отсчета времени, как обычно, можно использовать таймер TMRO, сохраняя его текущее значение в специально отведенной для этого переменной.

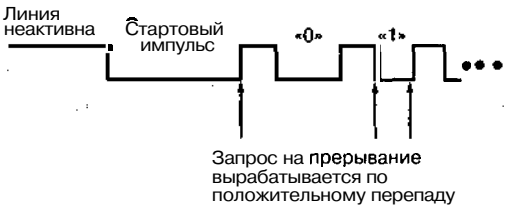

Рис. 4.61. Формирование запроса на прерывание по *положительному фронту входного сигнала при* приеме сигналов дистанционного **управления**

Приведенная в предыдущем разделе программа учитывала длительность стартового импульса, но мы упростим код, если вовсе будем его игнорировать: ведь для определения длительности очередного импульса он совершенно не нужен.

При каждом вызове обработчика прерываний, осуществляемом по приходе положительного фронта очередного импульса, мы будем проверять, как давно был выполнен предыдущий вызов. Если прошло более 9 мс, то надо прекратить процесс декодирования до прихода следующего пакета. Как уже говорилось, пакеты повторяются каждый 50 мс. Поэтому, пропустив текущий пакет, мы всегда имеем шанс без ошибок принять следующий.

Для реализации нашего нового проекта не понадобится переделывать предыдущую схему, но, разумеется, придется изменить текст программы. Для отображения принимаемых команд будет использоваться все тот же жидкокристаллический дисплей. Текст программы вы найдете в файле Code\Remote\remote.c:

```
#include <pic.h>

// 17 апреля 2002 - адаптировано к новой версии компилятора.
// 15 февраля 2002 - прием и отображение команд инфракрасного пульта
// дистанционного управления фирмы Sony.
//
// Для индикации используется жидкокристаллический дисплей,
// подключенный к микроконтроллеру через двухпроводной интерфейс
// на основе сдвигового регистра 74LS174.
//
// Используются прерывания от таймера TMR0.
//
//
// Формат пакета данных:
//
//       Стартовый бит     1                    0
// ---+                  +-----+      +-----+          +-------
//    I                  I     I      I     I          I
//    +------------------+     +------+     +----------+
//
//    |   2,4 мс        |540мкс|660мкс|540мкс|   1,2 мс   |
//
//              |  1,2 мс   |    1,76 мс     |
//
//               |   300    |     440       |
//
```

```
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F84/PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RB0 - детектор инфракрасного излучения
// (прерывания по положительному перепаду):
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея.

// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.
volatile char LCDDlay = 20;     // Длительность задержки.
volatile char LCDState = 1;     // Номер текущего состояния
                                // конечного автомата.

static volatile bit Clock       @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS   @ (unsigned)&TRISB*8+1;
static volatile bit Data        @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS    @ (unsigned)&TRISB*8+2;

char » MessageOut; // Указатель на строку выводимого сообщения.
volatile char MessageOuti = 0;  // Смещение текущего символа в строке.

unsigned int  DataIn;    // Входной сигнал инфракрасного приемника.
unsigned char DataInCount = 0;  // Число импульсов для приема.
unsigned char DataReady = 0;    // Число принятых импульсов.
int  SaveRTC;                   // Время последнего прерывания.
int  CurrentRTC;
char Message[20] = "\f-> 0b0xxxxxxxxxxxx";      // Шаблон сообщения.

//  Слово конфигурации:
«if defined (_16F84)
   #warning PIC16F84 selected
   __CONFIG(0x03FF1);    // Для МК PIC16F84:
                         // кварцевый тактовый генератор,
                         // таймер PWRT включен,
                         // сторожевой таймер выключен,
                         // защита кода отключена.
#elif defined(_16F627)
   #warning PIC16F627 with external oscillator selected
   __CONFIG(0x03F61);    // Для МК PIC16F627:
                         // внешний тактовый генератор XT,
                         // RA6/RA7 используются для ввода-вывода,
                         // внешний сигнал сброса,
                         // таймер PWRT включен,
                         // сторожевой таймер выключен,
                         // защита кода отключена,
                         // детектор BODEN включен.
#else
   #error Unsupported PICmicro MCU selected
#endif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS)         // Запись полубайта в ЖКИ.

   ...
```

```
LCDByte(char Byte, char RS)              // Запись байта в ЖКИ.
   ...
LCDInit()                                // Инициализация ЖКИ.
   ...
LCDOut(char * const LCDString)   //  Вывод строки на ЖКИ.
   ...
//  Обработчик прерываний:
void interrupt tmr0_int(void)
{
char temp;
   if (TOIF) {            // Обработчик прерываний от таймера TMR0.
         TOIF = 0;        // Сбросить флаг прерываний от таймера TMR0.
         RTC++;           // Инкремент счетчика реального времени.
         // Здесь можно разместить дополнительный код
         // для обработки прерываний от таймера TMR0.
         // Проверка времени последнего вызова обработчика
         // и пропуск текущего пакета при необходимости:
         CurrentRTC = (RTC & 0xOFF) - ((SaveRTC >> 8) & 0xOFF);
         if (CurrentRTC < 0)
               CurrentRTC = 0 - CurrentRTC;
         if ((DataInCount != 0) && (CurrentRTC > 9))
               DataInCount = 0;       // Пропустить текущий пакет.
         // Конечный автомат для ЖКИ:
         ...
   }  // Конец обработчика прерываний от таймера.
   // Здесь можно разместить другие обработчики прерываний.
   if (INTF) {            // Обработчик прерываний по положительному
                          // фронту сигнала на выводе RB0/INT.
         if (DataReady)         // Предыдущий пакет уже принят?
               ;        // Игнорировать импульсы до конца пакета.
         else if (DataInCount == 0) { // Начало нового пакета.
               DataInCount = 12;      // Надо принять 12 импульсов.
               SaveRTC = ((RTC & 0xOFF) << 8) + TMR0;
                                      // Запомнили текущее время.
               DataIn = 0;
         }  else {                    // Обработка очередного бита.
               CurrentRTC = ((RTC & 0xOFF) << 8) + TMR0;
                                      // Узнали текущее время.
               if ((SaveRTC = CurrentRTC - SaveRTC) < 0)
                     SaveRTC = 0 - SaveRTC;
               if ((SaveRTC > 250) && (SaveRTC < 350)) {
                     DataIn = (DataIn << 1) + 1;  // Получена 1.
                     if (--DataInCount == 0)
                           DataReady = 1;
               } else if ((SaveRTC > 390) && (SaveRTC < 490)) {
                     DataIn = DataIn << 1;        // Получен 0.
```

```
                        if (--DataInCount == 0)
                                DataReady = 1;
                    } else                  // Ошибка.
                            DataInCount = 0;
                    SaveRTC = CurrentRTC;  // Запомнили текущее время.
                }

                INTF = 0;                   // Сброс флага прерывания.

        }       // Конец обработки прерываний по положительному фронту
                // сигнала на входе RBO/INT.

} // Конец обработчика прерываний.

// Главная программа:

void main(void)
{
int i;

    OPTION = 0x0D1;         // Предделитель работает с таймером TMR0,
                            // коэффициент деления равен 4.

    TMR0 = 0;               // Начальный сброс таймера TMR0.
    T0IE = 1;               // Разрешение прерываний от таймера TMR0.
    GIE = 1;                // Разрешение обработки прерываний.

    // Здесь можно проинициализировать другие периферийные устройства.

    LCDInit();      // Инициализация порта, к которому подключен ЖКИ.

    INTEDG = 1;     // Прерывания по положительному фронту сигнала
                    // на входе RBO/INT.
    INTE = 1;       // Разрешить прерывания по изменению сигнала
                    // на выходе RBO/INT.

    while (1 == 1) {                    // Бесконечный цикл.

            // Здесь можно разместить код верхнего уровня.

            if (DataReady) {                    // Если данные готовы.
                    while (LCDState != 0);              // Ждать готовности ЖКИ.
                    for (i = 0; i < 12; i++) {
                            if (DataIn & (1 << 11))
                                    Message[i + 7] = '1';
                            else
                                    Message[i + 7] = '0';
                            DataIn = DataIn << 1;
                    } // Конец цикла вывода кода принятой команды.
                    LCDOut(Message);            // Вывести сообщение.
                    DataReady = 0;              // Сбросить флаг
                                                // готовности данных.
            } // Конец оператора if.
    } // Конец оператора while.

} // Конец главной программы.
```

Код этой программы может показаться достаточно сложным, но на самом деле в нем не так уж трудно разобраться. Когда на входе RBO/INT микроконтроллера, к которому подключен инфракрасный приемник, уровень сигнала изменяется с низкого на высокий, то устанавливается флаг INTF и генерируется запрос на

прерывание. В процедуре обработки прерываний проверяется состояние этого флага. Если он установлен, значит, источником прерывания послужил вход RBO/INT. В таком случае обработчик прерываний по значению счетчика `DataInCount` проверяет, первый ли это импульс. Если значение счетчика нулевое, оно свидетельствует о том, что пришел стартовый импульс нового пакета данных. Далее мы загружаем в счетчик число ожидаемых импульсов (то есть значение 12) и запоминаем текущее время, чтобы при следующем вызове определить длительность импульса. Так продолжается до тех пор, пока не будут декодированы все 12 импульсов пакета данных.

После приема двенадцатого импульса устанавливается флаг `DataReady` - таким образом главной программе сообщается, что очередная команда дистанционного управления декодирована. Обработчик будет игнорировать все остальные пакеты, пока главная программа, выполнив команду, не сбросит этот флаг.

Принятая последовательность нулей и единиц для контроля выводится на жидкокристаллический дисплей. Заметим, что перед загрузкой в выводимую строку текста нового сообщения мы проверяем, закончился ли вывод предыдущего, чтобы содержимое строки Message случайно не изменилось, пока еще продолжается ее вывод на дисплей. Если пренебречь этой проверкой, может оказаться, что на экране мы увидим начало предыдущей команды и конец следующей.

В данном случае вероятность такого события не слишком велика, поскольку вывод на дисплей занимает времени намного меньше, чем промежуток между пакетами данных, но мы всегда должны быть готовы к тому, что нам придется модифицировать свои приложения для более сложных задач.

4.25. СОВМЕЩЕНИЕ РАБОТЫ ДЕТЕКТОРА ОБЪЕКТОВ И ПРИЕМНИКА КОМАНД ДИСТАНЦИОННОГО УПРАВЛЕНИЯ

Итак, наш робот уже умеет обнаруживать объекты и принимать сигналы дистанционного управления. В обоих проектах был использован детектор инфракрасного излучения. Поэтому будет логично объединить эти две функции в одном устройстве.

Так как в одном устройстве будут объединены функции по обнаружению близких объектов и приему сигналов дистанционного управления, то придется совместить оба типа сообщений на одном жидкокристаллическом дисплее. Сообщения о состоянии детектора объектов и об очередной принятой команде должны будут выводиться в разных частях экрана, не затирая друг друга.

Сначала я полагал, что для реализации проекта достаточно лишь немного модифицировать код обработчика прерываний от таймера, срабатывающего каждую миллисекунду. При каждом его вызове можно было бы проверять состояние входа RBO/INT, и, если уровень сигнала на нем активный (низкий), начинать измерение длительности импульса пакета данных, пришедшего от пульта дистанционного управления. Если при входе в обработчик уровень сигнала высокий, можно было бы сгенерировать очередной пакет импульсов частотой 38 кГц для инфракрасного излучателя. Идея казалась достаточно простой, но после нескольких часов безуспешных попыток заставить все это работать я был вынужден еще раз пересмотреть свои требования к системе, чтобы, наконец, понять, что же я хочу получить.

При первоначальной реализации проекта оказались допущены две ошибки. Во-первых, я забыл о своем решении игнорировать стартовый импульс пакета данных и использовать положительные (задние) фронты остальных двенадцати информационных импульсов для измерения длительности каждого из них. Совершенно справедливо посчитав, что раз ширина стартового импульса больше двух миллисекунд, то очередное прерывание от таймера, срабатывающего каждую миллисекунду, обязательно зафиксирует начало пакета данных, я забыл, что все еще использую задний фронт стартового импульса для вычисления длительности первого бита в пакете.

Поэтому в обработчике прерываний, обнаружив низкий уровень сигнала на входе RBO/INT, я разрешал прерывания по положительному перепаду напряжения на этом входе, а затем, как и в предыдущей программе, инициализировал счетчик DataInCount значением 12, хотя на самом деле надо было ожидать 13 срабатываний обработчика. Устройство начало работать только после того, как я изменил указанное значение на 13 и обеспечил пропуск интервала между двумя первыми срабатываниями (с того момента, когда был замечен низкий уровень сигнала на входе RBO/INT и было разрешено формирование запросов на прерывание по этому входу, до момента прихода первого такого запроса, соответствующего концу стартового импульса).

Вторая ошибка была обнаружена при испытании правильности приема команд. Иногда в момент нажатия какой-либо кнопки пульта дистанционного управления на экране появлялось сообщение COLLISION, то есть вместо приемника команд срабатывал детектор столкновений. Виновником ошибки оказался стартовый импульс пакета данных, в момент прихода которого инфракрасный излучатель обнаружителя объектов еще не успевал выключиться. Детектор воспринимал начало команды дистанционного управления как сигнал, отраженный от объекта.

Пришлось сделать так, чтобы детектор реагировал на принятый сигнал только в том случае, если он действует более 3 мс. Так как длительность стартового импульса равна 2,4 мс, задержка срабатывания гарантирует, что стартовый импульс будет проигнорирован детектором. Разумеется, уже с самого начало было непростительной ошибкой рапортовать о столкновении только на основании первого же срабатывания детектора. Я никогда бы не допустил подобной ошибки, если бы имел дело с механическим детектором столкновений, но, испытывая свои многочленные конструкции, основанные на инфракрасном обнаружителе объектов, я уже привык к его высокой надежности, поскольку еще ни разу не видел, чтобы он допускал ложные срабатывания.

После всех описанных изменений устройство стало работать почти без ошибок. Однако когда вблизи детектора находился какой-либо объект и в то же время я нажимал любую кнопку пульта дистанционного управления, каждый десятый раз при приеме команды происходила ошибка (единичный бит пакета данных принимался декодером за нулевой).

Теоретически такая ошибка могла объясняться тем, что во время выполнения фрагмента обработчика прерываний, реализующего вывод сообщений на экран жидкокристаллического дисплея, устройство не успевало правильно измерить ширину очередного импульса. Эта идея подтверждалась тем, что ошибка пропадала

при отключении вывода сообщений на экран (или строки COLLISION, или кода принятой команды).

Проблему можно решить, запретив работу приемника команд дистанционного управления на то время, пока подаются импульсы на светодиод обнаружителя объектов. Это не оказало бы влияния на работу приемника команд, так как пакеты от передатчика команд дистанционного управления повторяются каждые 50 мс, и, пропустив один из них, мы всегда имеем возможность принять и декодировать следующий.

Окончательный вариант программы содержится в файле Code\Combine\combine.c:

```
#include <pic.h>

// 19 апреля 2002 - обьединение функций детектора столкновений
// и приемника команд дистанционного управления.
//
// 15 февраля 2002 - прием команд инфракрасного пульта
// дистанционного управления фирмы Sony и отображение их
// на жидкокристаллическом индикаторе.
//
// Для работы инфракрасного детектора столкновений используется
// ШИМ-сигнал частотой 38 кГц.
//
// Для индикации используется жидкокристаллический дисплей,
// подключенный к микроконтроллеру через двухпроводной интерфейс
// на основе сдвигового регистра 74LS174.
//
// Формат пакета данных:
//
//       Стартовый бит     1                    0
// ---+                 +----+      +----+              +--------
//    I                 I    I      I    I              I
//    +-----------------+    +------+    +--------------+
//
//    |   2,4 мс       |540мкс|660мкс|540мкс|   1,2 мс   |
//
//                      I 1,2 мс   |   1,76 мс      |
//
//                      |   300    |    440         |
//
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RB0 - детектор инфракрасного излучения
// (прерывания по положительному перепаду);
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея.

// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.
```

```
volatile char LCDDlay = 20;      // Длительность задержки.
volatile char LCDState = 1;      // Номер текущего состояния
                                 // конечного автомата.
static volatile bit Clock        @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS    @ (unsigned)&TRISB*8+1;
static volatile bit Data         @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS     @ (unsigned)&TRISB*8+2;

char * MessageOut; // Указатель на строку выводимого сообщения.
volatile char MessageOuti = 0;   // Смещение текущего символа в строке.

unsigned int  DataIn;    // Входной сигнал инфракрасного приемника.
unsigned char DataInCount = 0;  // Число импульсов для приема.
unsigned char DataReady = 0;    // Число принятых импульсов.
int  SaveRTC;                   // Время последнего прерывания.
int  CurrentRTC;

char Message[22] = "\376\0022-> 0b0xxxxxxxxxxxx!";
                        // Шаблон для вывода декодированной команды.
char Message1[12] = "\376\300        ";        // Очистка экрана.
char Message2[12] = "\376\300COLLISION";

volatile char Collision = 0;     // Сначала обнаруженных объектов нет.
volatile char OldCollision = 0;

// Слово конфигурации:
#if defined(_16F627)
   _CONFIG(0x03F61);     // Для МК PIC16F627:
                         // внешний тактовый генератор XT,
                         // RA6/RA7 используются для ввода-вывода,
                         // внешний сигнал сброса,
                         // таймер PWRT включен,
                         // сторожевой таймер выключен,
                         // защита кода отключена,
                         // детектор BODEN включен.
«else
   (terror Unsupported PICmicro MCU selected
#endif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS)          // Запись полубайта в ЖКИ.

   ...
LCDByte(char Byte, char RS)              // Запись байта в ЖКИ.

   ...
LCDInit()                                // Инициализация ЖКИ.

   ...
LCDOut(char * const LCDString)   //  Вывод строки на ЖКИ.

   ...
//  Обработчик прерываний:
void interrupt tmr0_int(void)
{

char temp;
```

```
if (TOIF) {           // Обработчик прерываний от таймера TMR0.
      TOIF = 0;       // Сбросить флаг прерываний от таймера TMR0.
      RTC++;          // Инкремент счетчика реального времени.
      // Здесь можно разместить дополнительный код
      // для обработки прерываний от таймера TMR0.
      // Проверка времени последнего вызова обработчика
      // и пропуск текущего пакета при необходимости:
      if (DataInCount != 0) <
            if ((CurrentRTC = (RTC & 0x0FF) -
                ((SaveRTC >> 8) & 0x0FF)) < 0)
                  CurrentRTC = 0 - CurrentRTC;
                                 // Сколько прошло времени?
            if (CurrentRTC > 9) { // Слишком много.
                  DatalnCount = 0; // Пропустить этот пакет.
                  INTE = 0;
                  // Запретить прерывания по приходе данных.
            >
      } else if ((!RB0) && (!DataReady)) {// Пришел пакет?
            DatalnCount = 13;
                  // Ждем конца стартового импульса
                  // и еще 12 информационных импульсов.
            SaveRTC = ((RTC & 0x0FF) << 8) + TMR0;
                  // Запомнили текущее время.
            INTF = 0;      // Сбросить флаг прерывания
            INTE = 1;      // и разрешить прерывания по приходе
                           // сигнала на вход RB0/INT.
      } else {             // Детектор столкновений.
            TRISB3 = 0;    // Разрешить вывод ШИМ-сигнала.
            while (TMR0 < 64);     // В течение 64 "тиков"
                                   // работы таймера TMR0.
            if (!RB0)      // Если уровень низкий, то сработал
                           // детектор столкновений.
                  Collision++; // Ждем трех срабатываний подряд.
            else
                  Collision = 0;
            TRISB3 = 1;    // Запретить вывод ШИМ-сигнала.
      }  // Конец кода для управления детектором столкновений.
      // Конечный автомат для ЖКИ:
      ...
}  // Конец обработчика прерываний от таймера.
// Здесь можно разместить другие обработчики прерываний.
if (INTF) {           // Обработчик прерываний по положительному
                      // фронту сигнала на выводе RB0/INT.
      CurrentRTC = ((RTC & 0x0FF) << 8) + TMR0;
                                 // Запомнили текущее время.
      if ((SaveRTC = CurrentRTC - SaveRTC) < 0)
```

```
            SaveRTC = 0 - SaveRTC;
        if ((SaveRTC > 250) && (SaveRTC < 350)) {
            DataIn = (DataIn << 1) + 1;  // Получена 1.
            if (--DataInCount == 0) {
                DataReady = 1;
                INTE = 0;      // Конец, запретить прерывания,
            }  //  endif
        } else if ((SaveRTC > 390) && (SaveRTC < 490)) {
            DataIn = DataIn << 1;        // Получен 0.
            if (--DataInCount == 0) {
                DataReady = 1;
                INTE = 0;      // Конец, запретить прерывания.
            }  //  endif
        } else if (--DataInCount != 12) {   // Ошибка.
            DataInCount = 0;
            INTE = 0;              // Конец, запретить прерывания.
        }  //  endif
        SaveRTC = CurrentRTC;        // Запомнить текущее время.
        INTF = 0;                    // Сбросить флаг прерывания.

    }       // Конец обработки прерываний по положительному фронту
            // сигнала на входе RB0/INT.

} // Конец обработчика прерываний.

// Главная программа:

void main(void)
{
int i;

    OPTION = 0x0D1;         // Предделитель работает с таймером TMR0,
                            // коэффициент деления равен 4.

    TMR0 = 0;               // Начальный сброс таймера TMR0.
    T0IE = 1;               // Разрешение прерываний от таймера TMR0.
    GIE = 1;                // Разрешение обработки прерываний.

    // Здесь можно проинициализировать другие периферийные устройства.

    LCDInit();     // Инициализация порта, к которому подключен ЖКИ.

    INTEDG = 1;    // Прерывания по положительному фронту сигнала
                   // на входе RB0/INT.

    CCPR1L = 13;   // Ширина импульсов ШИМ составляет 50% периода.
    CCP1CON = 0b000001111;          // Включить модуль ШИМ.

    PR2 = 26;                   // При частоте ШИМ-сигнала 38 кГц
                                // период должен быть равен 26 мкс.

    TMR2 = 0;                   // Сброс таймера TMR2.
    T2CON = 0b000000100;        // Включение таймера TMR2,
                                // коэффициент предделителя 1:1.

    while (1 == 1) {            // Бесконечный цикл.

        // Здесь можно разместить код верхнего уровня.

        if (DataReady) {                    // Если данные готовы,
            while (LCDState != 0);          // ждать готовности ЖКИ.
            for (i = 0; i < 12; i++) {
```

```
                    if. (DataIn & (1 << 11))
                          Message[i + 8] = '1';
                    else
                          Message[i + 8] = '0';
                    DataIn = DataIn << 1;
                } // Конец цикла вывода кода принятой команды.
                LCDOut(Message);       //  Pass String to Output
                DataReady = 0;         //  Reset the Data Flag

          }   // Конец оператора if.

          if ((Collision == 0) && (OldCollision == 3)) {
                OldCollision = Collision;     // Запомнить текущее состояние
                                              // детектора столкновений.
                LCDOut(Message1);
          }
          if ((Collision == 3) && (OldCollision == 0)) {
                OldCollision = Collision;     // Запомнить текущее состояние
                                              // детектора столкновений.
LCDOut(Message2);
          }

    }  // Конец оператора while.

}  // Конец главной программы.
```

4.26. УЛЬТРАЗВУКОВОЙ ДАЛЬНОМЕР

Мы уже разобрали два метода реализации детектора столкновений: механический контактный и на основе приемника отраженного инфракрасного излучения. В этом разделе будет рассмотрен третий метод обнаружения объектов; он также будет бесконтактным, только на этот раз мы будем использовать ультразвук. Примерно таким же образом летучие мыши ориентируются в пространстве: они излучают вперед направленный пучок звуковых колебаний и ловят отраженный сигнал. Звуковые волны распространяются в воздухе с определенной скоростью, поэтому по задержке прихода отраженного сигнала можно с достаточной степенью точности судить, на каком расстоянии находится тот предмет, который отразил звук.

Оба бесконтактных способа обнаружения объектов (инфракрасный и ультразвуковой) прекрасно дополняют друг друга: предметы, которые не отражают инфракрасные лучи, например пластмассовые детали черного цвета, могут обнаруживаться ультразвуковым детектором, поскольку отражают звуковые волны, и наоборот, материалы, прозрачные для ультразвуковых колебаний (вроде тонкой ткани), могут быть замечены с помощью инфракрасного детектора.

Ультразвуковые дальномеры (*гидролокаторы,* или *сонары),* используемые на морских кораблях и подводных лодках, работают несколько иначе, чем описанный здесь дальномер. Излучатель гидролокатора не имеет узкой диаграммы направленности - он излучает ультразвуковой сигнал во всех направлениях, а для определения направления до ближайшего объекта используется узконаправленный микрофон. При этом расстояние до объекта вычисляется не по задержке прихода отраженного сигнала, а с помощью триангуляции по нескольким подряд сделанным изменениям при различных координатах движущегося корабля.

Дело в том, что скорость звука в воде сильно зависит от ее температуры, плотности, количества находящейся в ней соли и многих других факторов, поэтому не может служить основой для точных измерений.

Для ультразвукового обнаружителя объектов удобно использовать какой-нибудь готовый промышленный прибор, например дальномер фотоаппарата Polaroid 6500. Он обеспечивает довольно небольшой обзор, игнорируя все, что находится вне угла ±5°. Этот недостаток можно компенсировать, разместив дальномер на управляемом приводе, который мог бы периодически изменять его ориентацию, сканируя пространство необходимой ширины.

Значительно более серьезная проблема состоит в том, что дальномер потребляет много энергии, в результате чего резко снижается срок службы батарей автономного робота. Например, дальномер Polaroid 6500 во время генерации ультразвукового сигнала имеет ток потребления около 1 А. Разумеется, сигнал можно генерировать с перерывами, запасая во время паузы необходимую энергию в конденсаторах, но это только частично решает проблему.

При своей работе ультразвуковой дальномер посылает узконаправленный звуковой импульс и измеряет, с какой задержкой придет отраженный сигнал. В воздухе скорость звука составляет около 1087,4 футов в секунду (то есть примерно 331,4 м/с); значит, дистанцию длиной в один дюйм (2,54 см) туда и обратно звук преодолеет за 153,3 с. Если, например, объект находится на расстоянии одного сантиметра от ультразвукового детектора, то отраженный сигнал будет обнаружен спустя 60,4 мкс после того, как сработает излучатель. Большинство промышленных дальномеров (в том числе и Polaroid 6500) игнорирует отраженный сигнал, пришедший с задержкой, меньшей 2,76 мс, то есть исходящий от предметов, расположенных на расстоянии менее 18 дюймов (46 см).

Ультразвуковой дальномер Polaroid 6500 выпускается с 1970-х годов. Сначала он был очень дорогим, но сейчас широко доступен и весьма надежен в работе. Поэтому его часто используют радиолюбители в своих проектах.

Дальномер Polaroid 6500 состоит из двух частей: из небольшой печатной платы и круглого черного блока, в котором расположен излучатель. Плата соединена с детектором с помощью провода длиной около 15 дюймов (38 см). На плате расположен 9-контакный разъем, с помощью которого она может сопрягаться с различными устройствами. Назначение контактов этого разъема объясняется в табл. 4.14.

Таблица 4.14. Назначение контактов разъема дальномера Polaroid 6500

Вывод	Имя	Вход/выход	Комментарий
1	Gnd	–	
2	BLNK	Вход	Когда BLNK = 1, любой отраженный сигнал игнорируется
3	–		
4	INIT	Вход	Когда INIT = 1, изучается инфракрасный импульс
5	–		
6	OSC	Выход	Импульсы частотой 49,4 кГц

Таблица 4.14. Назначение контактов разъема дальномера Polaroid 6500 (окончание)

Вывод	Имя	Вход/выход	Комментарий
7	ECHO	Выход	Принятый отраженный сигнал (выход с открытым коллектором; необходимо использовать «подтягивающий» резистор сопротивлением 4,7 кОм)
8	BINH	Вход	Когда BINH = 1 (blank inhibit), выключается игнорирование сигнала, отраженного от близких объектов
9	Vcc	–	

Обычно при конструировании своих устройств я выпаиваю этот разъем и использую многожильный провод для подключения платы дальномера к основной схеме.

Во время работы дальномера на линиях BLNK и BINH поддерживается сигнал низкого уровня. При этом на линии INIT вырабатываются прямоугольные импульсы. Во время высокого уровня напряжения сигнала INIT излучатель дальномера вырабатывает ультразвуковой сигнал. Отраженный сигнал, принятый приемником, формируется на линии ECHO. На рис. 4.62 показаны осциллограммы обоих сигналов. На рисунке этого не видно, но после приема отраженного сигнала на линии INIT устанавливается низкий уровень напряжения.

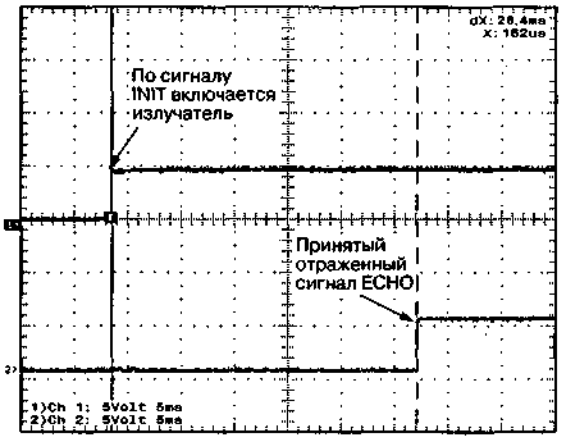

Рис. 4.62. Осциллограммы *излучаемого* и принимаемого сигналов *ультразвукового* дальномера

В документации рекомендовано включать между шинами питания конденсатор емкостью 1000 мкФ для снижения скачков напряжения во время работы излучателя.

Следует помнить, что для работы ультразвукового излучателя в схеме вырабатывается достаточно большое напряжение, и если держать модуль излучателя

в руках, то можно получить серьезный удар током. Ни в коем случае не допускайте того, чтобы сразу обе ваши руки контактировали с излучателем или соединительными проводами. По собственному опыту я могу утверждать, что даже касание одной рукой не проходит безболезненно.

Максимальное расстояние, на котором дальномер Polaroid 6500 может обнаруживать предметы, равно 35 футам (примерно 10,7м). Соответствующая задержка прихода отраженного сигнала составляет около 64,4 мс. Это значит, что если отраженный сигнал не обнаружен спустя 65 мс, то можно считать, что в зоне действия ультразвукового детектора нет посторонних объектов.

4.27. ПОДКЛЮЧЕНИЕ УЛЬТРАЗВУКОВОГО ДАЛЬНОМЕРА К МИКРОКОНТРОЛЛЕРУ

В одном из вышеописанных проектов для декодирования команд дистанционного управления мы осуществляли программное измерение длительности импульсов, подаваемых на вход RBO/INT микроконтроллера. Этот же принцип будет использоваться для измерения задержки прихода отраженного сигнала ультразвукового дальномера. Время, прошедшее с момента срабатывания ультразвукового излучателя до регистрации сигнала приемником, будет пересчитываться микроконтроллером в футы и дюймы и выводиться на экран жидкокристаллического дисплея.

Принципиальная схема рассматриваемого устройства показана на рис. 4.63, схема размещения элементов на макетной плате - на рис. 4.64, а список использованных элементов приведен в табл. 4.15.

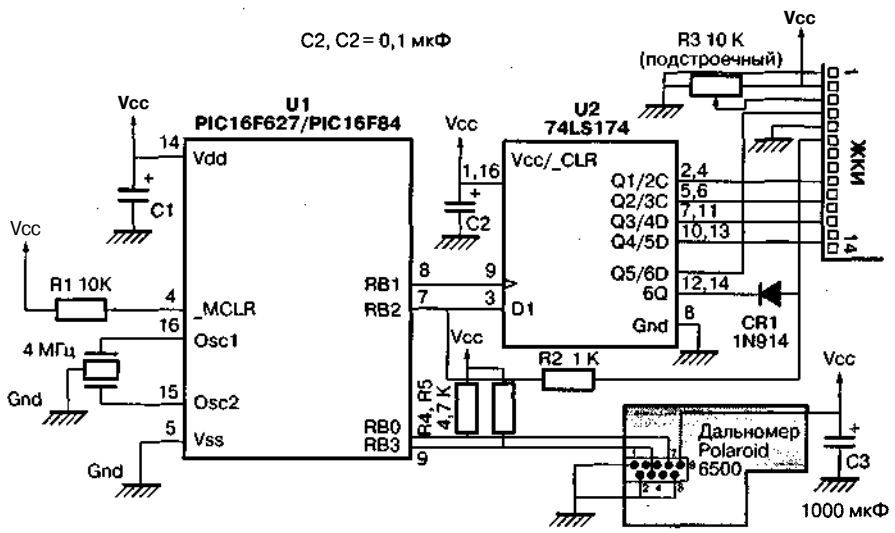

Рис. 4.63. *Принципиальная схема ультразвукового дальномера*

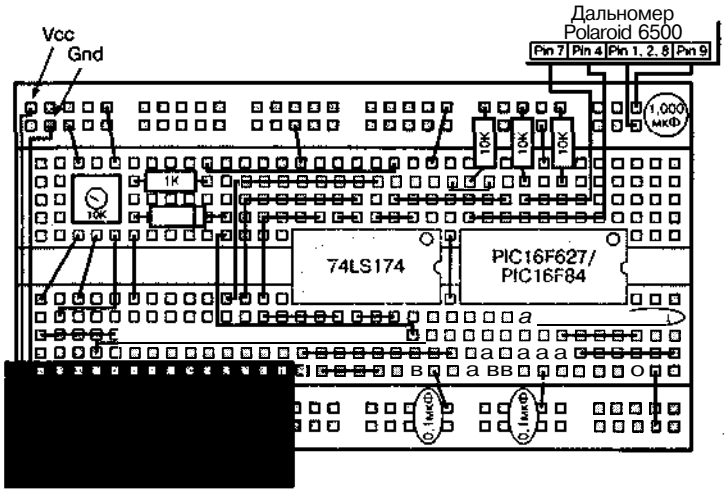

Рис. 4.64. Схема размещения элементов ***ультразвукового*** дальномера на макетной плате

Таблица 4.15. Список использованных элементов

Обозначение	*Элемент*	*Примечание*
U1	*PIC16F627 или PIC16F84*	*Микроконтроллер*
U2	*74LS174 (К555ТМ9)**	*Сдвиговый регистр*
CR1	*1N914 (КД521)**	*Любой кремниевый диод*
R1	*10 кОм; 0,25 Вт*	*Для «подтягивания» напряжения на выводе _MCLR до напряжения положительной шины питания*
R2	*1 кОм; 0,25 Вт*	
R3	*10 кОм, подстроечный*	*Для регулирования контрастности*
R4, R5	*4,7 Ом; 0,25 Вт*	
C1, C2	*0,1 мкФ*	*Для фильтрации напряжения питания микросхем*
C3	*1000 мкФ, 16 В*	*Оксидный (электролитический) конденсатор*
ЖКИ	*Жидкокристаллический дисплей /14 выводов), совместимый с Hitachi 44780*	
XTAL1	*Керамический резонатор на 4 МГц, со встроенными конденсаторами*	*Для генератора тактовых импульсов микроконтроллера*
Polaroid 6500	*Ультразвуковой дальномер*	
	Материалы	*Макетная плата, монтажные провода, источник питания +5 В*

** В скобках указаны отечественные аналоги, добавленные при переводе. — Прим. перев.*

Так как линии INIT и ECHO дальномера имеют выход с открытым коллектором, то приходится использовать «подтягивающие» резисторы R4 и R5 сопротивлением 4,7 кОм. Для фильтрации бросков напряжения на шинах питания, которые

возникают из-за большого тока потребления ультразвукового излучателя и могут негативно повлиять на работу всех остальных элементов схемы, используется конденсатор С3 емкостью 1000 мкФ.

В качестве источника питания для этого устройства лучше взять что-нибудь помощнее. Если для всех конструкций, рассмотренных выше, можно было использовать одну 9-вольтовую щелочную батарейку (или набор батареек АД соединенных последовательно), то наша новая схема с таким источником питания, скорее всего, не заработает.

Текст программы можно найти в файле Code\Ultra\ultrac:

```
#include <pic.h>

// 19 апреля 2002 - ультразвуковой дальномер Polaroid 6500.
// Индикация измеренного расстояния на жидкокристаллическом дисплее.
//
// Для индикации используется жидкокристаллический дисплей,
// подключенный к микроконтроллеру через двухпроводной интерфейс
// на основе сдвигового регистра 74LS174.
//
// Используются прерывания от таймера каждую миллисекунду.
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F84/PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RB0 - сигнал ECHO ультразвукового дальномера Polaroid 6500
//       (с "подтягивающим резистором");
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея;
// RB3 - сигнал INIT ультразвукового дальномера Polaroid 6500.

// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.
volatile char LCDDlay = 20;     // Длительность задержки.
volatile char LCDState = 1;     // Номер текущего состояния.

static volatile bit Clock       @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS   @ (unsigned)&TRISB*8+1;
static volatile bit Data        @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS    @ (unsigned)&TRISB*8+2;

char * MessageOut;              // Указатель на выводимое сообщение.
volatile char MessageOuti = 0;  // Смещение текущего символа в строке.
char Message[9] = "\fxx\' xx\""; // Формат вывода измеренной дальности.
char Message2[9] = "\fInvalid"; // Сообщение об ошибке.

unsigned int  CheckDlay = 50;   // Пауза 500 мс между моментами
                                // срабатывания ультразвукового излучателя.
unsigned int   PulseStartRTC;   // Когда начался излучаемый импульс.
unsigned int  PulseEndRTC;      // Когда закончился излучаемый импульс.
unsigned char PulseEndTMR0;     // Значение счетчика таймера TMR0
                                // в момент окончания излучаемого импульса.
```

```
unsigned char PulseState = 0;
                // 0 - ожидание перед новым измерением.
                // 1 - измерение задержки прихода отраженного сигнала.
                // 2 - расстояние измерено.
                // 3 - ошибка измерения.

unsigned long PulseTime;          // Задержка в миллисекундах.
unsigned long PulseInches;        // Количество дюймов.
unsigned long PulseFeet;          // Количество футов.

// Слово конфигурации:
#if defined (_16F84)
   #warning PIC16F84 selected
   __CONFIG(0x03FF1);     // Для МК PIC16F84:
                          // кварцевый тактовый генератор,
                          // таймер PWRT включен,
                          // сторожевой таймер выключен,
                          // защита кода отключена.
#elif defined(_16F627)
   #warning PIC16F627 with external XT oscillator selected
   _CONFIG(Ox03F61);      // Для МК PIC16F627:
                          // внешний тактовый генератор XT,
                          // RA6/RA7 используются для ввода-вывода,
                          // внешний сигнал сброса,
                          // таймер PWRT включен,
                          // сторожевой таймер выключен,
                          // защита кода отключена,
                          // детектор BODEN включен.
#else
   #error Unsupported PICmicro MCU selected
«endif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS)          // Запись полубайта в ЖКИ.
   ...
LCDByte(char Byte, char RS)              // Запись байта в ЖКИ.
   ...
LCDInit()                                // Инициализация ЖКИ.
   ...
LCDOut(char * const LCDString)   //  Вывод строки на ЖКИ.
   ...
//  Обработчик прерываний:
void interrupt tmr0_int(void)
{
char temp;
   if (T0IF) {            // Обработчик прерываний от таймера TMR0.
         T0IF = 0;        // Сбросить флаг прерываний от таймера TMR0.
         RTC++;           // Инкремент счетчика реального времени.
```

```
        // Здесь можно разместить дополнительный код
        // для обработки прерываний от таймера TMR0.

        // Работа с ультразвуковым дальномером:

        if (--CheckDlay == 0) {       // Пора выдать импульс
                                      // на ультразвуковой излучатель.
            PulseStartRTC = RTC;
            PulseState = 1;           // Измерение расстояния.
            RB3 = 1;                  // Подать сигнал ШТ.
            INTF = 0;                 // Сбросить флаг прерывания.
            INTE = 1;                 // Разрешить прерывания
                                      // по приходе сигнала на вход RB0/INT.
            CheckDlay = 500;          // Ждать 500 мс.
            if (RTC > 64900)          // Увеличить задержку
                CheckDlay += 200;     // на 200 мс.
        }

        if ((PulseState == 1) && ((PulseStartRTC + 60) < RTC)) {
            RB3 = 0;                  // Неправильная задержка,
            INTF = 0;                 // выключить дальномер.
            INTE = 0;
            PulseState = 3;
        }
        // Конец кода для работы с дальномером.


        // Конечный автомат для ЖКИ:

        ...

    } // Конец обработчика прерываний от таймера.

    // Здесь можно разместить другие обработчики прерываний.

    if (INTF) {           // Обработчик прерываний
                          // по фронту сигнала на выводе RB0/INT.
        PulseEndTMR0 = TMR0;  // Запомнили момент окончания импульса.
        PulseEndRTC = RTC;
        RB3 = 0;              // Сброс сигнала INIT дальномера.
        PulseState = 2;
        INTF = 0;     // Сбросили флаг прерывания.
        INTE = 0;     // Запретили прерывания по входу RB0/INT.

    }       // Конец обработки прерываний по фронту сигнала
        // на входе RB0/INT.

} // Конец обработчика прерываний.

// Главная программа:
void main(void)
{

unsigned int temp; // Служебная переменная.

    OPTION = 0x0D1;        // Предделитель работает с таймером TMR0,
                           // коэффициент деления равен 4.

    TMR0 = 0;              // Начальный сброс таймера TMR0.
    T0IE = 1;              // Разрешение прерываний от таймера TMR0.
    GIE = 1;               // Разрешение обработки прерываний.
```

```
LCDInit();      // Инициализация порта, к которому подключен ЖКИ.

// Здесь можно проинициализировать другие периферийные устройства.

RB3 = 0;              // Вывод RB3 перевели в режим выходного
TRISB3 = 0;           // и вывели сигнал низкого уровня.

while (1 == 1) {      // Бесконечный цикл.

       // Здесь можно разместить код верхнего уровня.

       switch(PulseState) {  // В зависимости от состояния.
              case 2:        // Вывод результата измерения.
                     PulseTime = (((long) PulseEndRTC * 256) +
                                  (long) PulseEndTMR0) * 40;
                     PulseTime -= ((long) PulseStartRTC *
                                                  (256 * 40));
                     PulseInches = PulseTime / 1533;
                     PulseFeet = PulseInches / 12;
                     PulseInches = PulseInches % 12;
                     if ((temp = PulseFeet / 10) == 0)
                            Message[1] = ' ';
                            // Расстояние меньше 10 футов,
                            // отображаем пробел
                            // вместо первой цифры.
                     else
                            Message[1] = temp + '0';
                     Message[2] = (PulseFeet % 10) + '0';
                     if ((temp = PulseInches / 10) == 0)
                            Message[5] = ' ';
                     else
                            Message[5] = temp + '0';
                     Message[6] = (PulseInches % 10) + '0';
                     LCDOut(Message);
                     PulseState = 0;  // Ждать следующего импульса.
                     break;
              case 3:                 // Ошибка.
                     LCDOut(Message2);
                     PulseState = 0;  // Ждать следующего импульса.
                     break;
       }  // Конец оператора switch.

   }  // Конец оператора while.

}  // Конец главной программы.
```

Здесь после включения питания и небольшой задержки (около 50 мс) начинается процесс измерения расстояния до ближайшего объекта в поле зрения детектора. Если объект не найден, то есть находится слишком близко (на расстоянии менее 46 см) или слишком далеко (дальше 11 м), то на жидкокристаллическом дисплее отображается сообщение INVALID (ошибка). Если расстояние до объекта лежит в пределах рабочего диапазона дальномера, то на дисплей выводится измеренная дальность в футах и дюймах.

Для перевода длительности задержки отраженного сигнала в единицы длины полученное количество микросекунд умножается на 10 и делится на 1533. Это эквивалентно делению на 153,3[1], но не требует использования вещественной

арифметики. После того как вычислено расстояние в дюймах, полученное значение делится на 12[2]; целая часть результата дает необходимое количество футов. Результат вычислений высвечивается на жидкокристаллическом дисплее.

Перед загрузкой этой программы в микроконтроллер желательно выполнить моделирование с помощью симулятора MPLAB. Для имитации входных воздействий я подготовил файл Code\Ultra\ultrasti:

```
! Моделирование сигнала, принимаемого ультразвуковым дальномером.
!
! Предполагаем, что излучатель посылает импульс
! через 51200 мкс после включения питания.
! Через 23148,3 мкс после этого приходит отраженный сигнал,
! что соответствует расстоянию 12' 7" (12 футов 7 дюймов).
!
! Момент прихода отраженного сигнала: 51200 + 23148 = 74348.
!
! Приемник дальномера подключен к выводу RB0 микроконтроллера.
!
! Майк Предко
! 19 апреля 2002 года
!
Step        RB0     RB4     RB5     RB6     RB7
1           0       0       0       0       0       ! Сначала все
                                                    ! уровни низкие.
15000       0       0       0       0       1
30000       0       0       0       1       1
45000       0       0       1       1       1
60000       0       1       1       1       1
74348       1       1       1       1       1       ! Пришел передний фронт
                                                    ! отраженного импульса.

75000       0       1       1       1       0       ! Задний фронт
                                                    ! отраженного импульса.
```

Здесь мы моделируем расстояние до объекта, равное 12 футам 7 дюймам (3,84 м). В файле стимулов указываются только входные сигналы; здесь не определяется значение выхода RB3, к которому подключена линия INIT дальномера. Состояние этой линии определяется программой, а не файлом стимулов. Мы предполагаем, что программа устанавливает на линии INIT сигнал высокого уровня через 51200 мкс после начала моделирования. Спустя 23148 мкс после этого приходит отраженный сигнал, то есть мы подаем единичный уровень на вход RB0 микроконтроллера.

Внимательный читатель заметит, что входы RB4, RB5, RB6 и RB7 в схеме не используются, поэтому их включение в файл стимулов может вызвать недоумение. Дело в том, что я (как, вероятно, и кто-нибудь из вас) часто терял много времени, выполняя пошаговый прогон программы, прежде чем обнаружить, что забыл в самом начале подключить файл стимулов. Тогда все приходится начинать заново.

[1] То есть на количество микросекунд, которое требуется звуку для преодоления расстояния длиной один дюйм (туда и обратно). - *Прим. перев.*

[2] В одном футе двенадцать дюймов. - *Прим. перев.*

Если указать в файле стимулов неиспользуемые входы и изменять их состояние с самых первых шагов, то уже в начале моделирования можно убедиться в том, что входные воздействия подаются правильно.

Разумеется, вы можете не использовать данный прием - в этом случае ваш файл стимулов для отладки описанной программы будет состоять всего из двух столбцов (номера шага и соответствующих значений на входе RBO).

Напомню, что для подключения файла стимулов к проекту надо выполнить команду главного меню **Debug ⇒ Simulator Stimulus ⇒ Pin Stimulus ⇒ Enable** (Отладка ⇒ Входные воздействия ⇒ Файл стимулов ⇒ Подключить). В папке Code\Ultra следует выбрать файл ultra.sti и нажать кнопку **OK.** После этого можно загрузить конфигурацию окна просмотра **Watch,** указав файл P16F627.wat.

Установите точку останова в первой строке ветви case 2 оператора выбора:

```
PulseTime = (((long) PulseEndRTC » 256) + (long) PulseEndTMR0) • 40;
```

Нажмите клавишу F6, чтобы подать сигнал сброса микроконтроллера, и запустите программу нажатием клавиши **F9.**

После того как сработает точка останова, выполните команду меню **Window ⇒ File Registers** (Окно ⇒ Регистровый файл), чтобы вывести на Рабочий стол MPLAB окно, в котором отображается содержимое регистров микроконтроллера. Чтобы легче было найти строку, где хранится текст выводимого на дисплей сообщения, надо переключить это окно в режим ASCII. Для этого щелкните по пиктограмме в левом верхнем углу окна File **Register Window** и выберите пункт **ASCII Display** (Текстовый формат).

Теперь выполните в пошаговом режиме (клавиша F7) еще несколько строк программы, пока выполнение не дойдет до строки

```
LCDOut(Message);
```

Постепенно вы сможете видеть, как будет заполняться строка сообщения. Обратите внимание, как медленно продвигается процесс моделирования. Каждая строка программы, содержащая оператор умножения или деления, выполняется несколько миллисекунд, не раз приостанавливаясь из-за процедуры обработки прерываний.

Вот почему вся обработка информации, достаточно сложная, должна проводиться в главной программе, а не в процедуре обработки прерываний. Если бы мы выполняли вычисления в обработчике прерываний, то многие запросы на прерывание были бы пропущены, так как во время работы обработчика другие прерывания запрещены.

Кроме того, при пошаговой отладке вы должны заметить, что расстояние вычисляется неправильно. Вместо ожидаемой нами строки

```
12' 7"
```

в окне **File Register Window** мы увидим

```
12' 1"
```

Причиной ошибки послужила небрежность, допущенная мною при составлении файла стимулов.

Выполните сброс (клавиша F6) и поставьте точку останова в строке

```
RB3 = 1;
```

обработчика прерываний. Если теперь запустить выполнение программы клавишей **F9,** то можно убедиться, что включение излучателя дальномера произойдет на 52037-м командном цикле, а не на 51200-м, как ожидалось. Лишние 837 циклов задержки соответствуют ошибке в 5,46 дюймов (13,87 см) при измерении дальности. При вычитании из 7 числа 5,46, округляя до целых, как раз получим значение 1, которое выдает наша программа.

Если мы теперь исправим файл стимулов так, чтобы отраженный сигнал появлялся не на 73248-м, а на 75185-м шаге моделирования, то увидим правильное значение 12' 7".

Советую вам вернуться к предыдущим проектам и создать похожий файл стимулов для моделирования приложений whisker.c и remote.c.

Возможно, кому-то процесс моделирования покажется скучным, но уверяю вас, что прогон программы в симуляторе сэкономит многие часы, которые вам пришлось бы провести, безуспешно пытаясь найти ошибку в неправильно работающем устройстве.

4.28. СВЕТОВЫЕ ДАТЧИКИ

Не так уж сложно наделить нашего робота элементарной способностью реагировать на свет. Разумеется, пока речь не идет о распознавании изображений, но мы без труда можем добавить к описанным выше конструкциям несколько световых датчиков и так изменить управляющую программу, чтобы робот двигался в сторону источника света, подобно тому как это делают насекомые.

Обычно для этого в передней части корпуса под углом 45° к продольной оси робота устанавливают два фотодиода, каждый из которых будет иметь свое поле обзора, как показано на рис. 4.65. Такое расположение световых датчиков позволит легко определить направление до источника света.

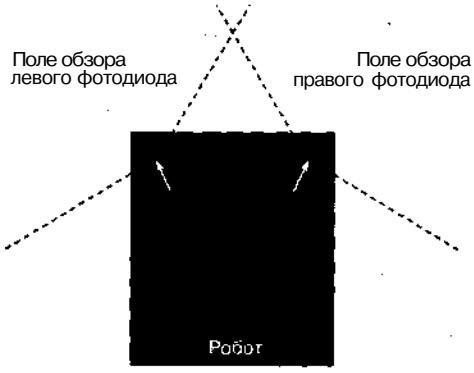

Рис. 4.65. *Границы поля обзора* левого и *правого* световых *датчиков*

Разумеется, можно не ограничиваться двумя фотодиодами спереди, а расположить их по всему периметру, чтобы робот мог реагировать и на те источники света, которые расположены сзади.

В качестве датчиков света подойдут не только фотодиоды, но также фототранзисторы или фоторезисторы из сульфида кадмия.

Принцип работы датчиков света основан на том, что сопротивление фоторезистора или фотодиода в обратном включении (либо коллектор-эмиттерное сопротивление фототранзистора) изменяется под действием света: оно уменьшается при увеличении освещенности. Что касается фотодиода, он может использоваться и в качестве фотоэлемента, то есть служить источником фототока, сила которого тем больше, чем ярче освещен фотодиод. Сила фототока очень мала - она должна быть увеличена с помощью специальной электронной схемы. При работе с фототранзисторами часто используют включение по схеме Дарлингтона, которое позволяет усилить сигнал.

Фоторезисторы обычно весьма инерционны (имеют время реакции порядка десятков и даже сотен миллисекунд), поэтому они не могут быть использованы для регистрации быстрых изменений освещенности, например для приема сигналов дистанционного управления. Даже, возможно, придется уменьшить скорость движения робота до минимума, чтобы он успевал сканировать все окружающее пространство в поисках источника света.

Тем не менее фоторезисторы из сульфида кадмия весьма популярны в радиолюбительских конструкциях; так, датчики на их основе обладают хорошей чувствительностью, то есть имеют максимальное отношение изменения тока через резистор к соответствующему изменению освещенности.

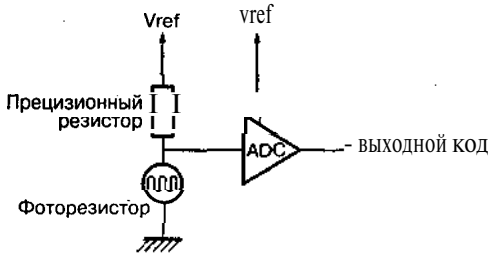

Рис. 4.66. *Измерение сопротивления фоторезистора с помощью АЦП*

Все три описанных типа световых датчиков формируют аналоговый сигнал. Но микроконтроллер работает только с сигналами, представленными в цифровой форме. Для того чтобы ввести в микроконтроллер сигнал, снимаемый с аналогового светового датчика, можно использовать *аналого-цифровой преобразователь* (АЦП). Схема подключения показана на рис. 4.66. Здесь фоторезистор является нижним плечом делителя напряжения источника питания, а в качестве верхнего плеча используется прецизионный резистор. Напряжение в средней точке делителя преобразуется с помощью АЦП в цифровой код и подается на вход микроконтроллера. Для работы АЦП обычно требуется высокостабильный источник опорного напряжения Vref. Сопротивление прецизионного резистора должно быть равно наибольшему возможному сопротивлению фоторезистора.

Если необходимо, чтобы выходной код АЦП увеличивался при возрастании освещенности фоторезистора, надо поменять местами плечи делителя, то есть переставить фоторезистор и прецизионный резистор.

Другой способ измерения уровня яркости не требует использования АЦП (рис. 4.67). Здесь вывод микроконтроллера подключен к средней точке делителя напряжения, нижнее плечо которого, как и в предыдущей схеме, образовано сопротивлением фоторезистора, а в верхнем плече установлен конденсатор. Сначала вывод микроконтроллера переводится в режим выходного и на него подается импульс напряжения; в результате происходит разряд конденсатора. После этого вывод микроконтроллера переключается на ввод сигнала, а конденсатор начинает снова заряжаться, напряжение на нем увеличивается. Как следствие, напряжение в средней точке делителя уменьшается. Когда оно достигнет порога переключения, на входе микроконтроллера будет зафиксирован сигнал низкого уровня. Время разряда конденсатора тем больше, чем меньше сопротивление фоторезистора, то есть чем меньше он освещен.

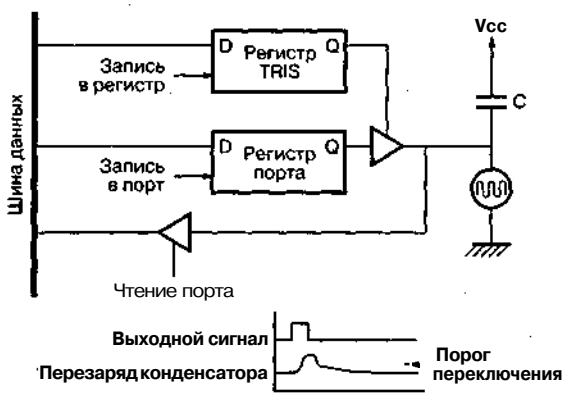

Рис. 4.67. *Измерение сопротивления фоторезистора с помощью* **RC-цепочки**

Для вычисления времени, в течение которого напряжение на входе микроконтроллера падает от напряжения питания делителя Vcc до напряжения переключения, можно использовать приближенную формулу

$$T = 2{,}2 \times R \times C.$$

Эта формула позволяет лишь оценить ожидаемое время разряда конденсатора по порядку величины - реальное время зависит не только от емкости конденсатора, но и от напряжения питания и характеристик входных цепей микроконтроллера, поэтому для более точных измерений приходится проводить индивидуальную градуировку.

Наконец, третий метод ввода аналогового сигнала основан на использовании встроенного в микроконтроллер аналогового компаратора (рис. 4.68). Здесь оба плеча делителя напряжения образованы сопротивлениями фоторезисторов, а сигнал с общей точки их соединения подается на вход микроконтроллера, где сравнивается компаратором с опорным значением Vref. Если напряжение, снимаемое с делителя, больше опорного (это говорит о том, что верхний фоторезистор освещен

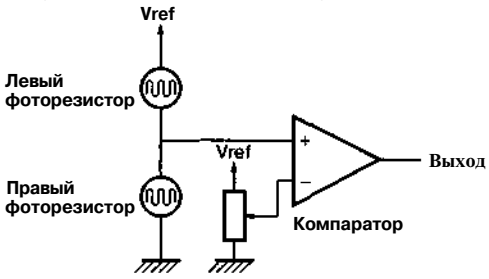

Рис. 4.68. Измерение *сопротивления фоторезисторов с* помощью компаратора

меньше, чем нижний), то на выходе аналогового компаратора будет сигнал высокого уровня, а в противном случае - сигнал низкого уровня.

Последний метод очень прост, но менее информативен, поскольку позволяет только определить, какой световой датчик освещен больше, а какой меньше, в то время как предыдущие два метода действительно обеспечивали измерение уровня освещенности.

Следует помнить, что все три метода ввода аналоговых сигналов имеют низкую помехоустойчивость. Может даже потребоваться выключать двигатели на то время, пока длится опрос световых датчиков.

Если вы хотите, чтобы ваш робот с высокой достоверностью определял направление до источника света в любой обстановке, то, вероятно, придется использовать небольшую телекамеру. Ее видеосигнал можно будет подвергнуть несложной обработке, чтобы найти наиболее освещенный участок комнаты.

4.29. ПОДКЛЮЧЕНИЕ СВЕТОВЫХ ДАТЧИКОВ К МИКРОКОНТРОЛЛЕРУ

В предыдущем разделе мы познакомились с тремя основными методами подключения световых датчиков к микроконтроллеру. Каждый из них имеет свои преимущества и недостатки; мы еще обсудим их, но сначала рассмотрим несколько конкретных примеров.

Обычно используется дифференциальная схема на двух фоторезисторах, включенных по схеме делителя напряжения (как на рис. 4.68). Когда оба фоторезистора освещены одинаково, то их сопротивления примерно равны, и в средней точке делителя напряжение равно половине напряжения питания. Если верхний фоторезистор освещен больше, чем нижний, то его сопротивление меньше; в этом случае напряжение, снимаемое с делителя, больше половины напряжения питания. Наоборот, если больше освещенность нижнего резистора, то напряжение в средней точке делителя меньше половины напряжения питания.

Чтобы узнать, какой из двух фоторезисторов освещен больше, достаточно использовать аналоговый компаратор, один вход которого должен быть подключен к выходу делителя, а на второй следует подать напряжение, равное половине

напряжения питания делителя. Как мы уже говорили в главе 3, микроконтроллер PIC16F627 имеет два встроенных аналоговых компаратора. Поэтому для нашего первого примера выберем именно этот способ подключения датчиков.

Принципиальная схема показана на рис. 4.69. Здесь мы будем формировать опорное напряжение для компаратора с помощью средств, встроенных в микроконтроллер. Мы должны запрограммировать внутренний источник, чтобы его напряжение равнялось половине напряжения питания микроконтроллера. Для проверки это напряжение подается на вывод RA2.

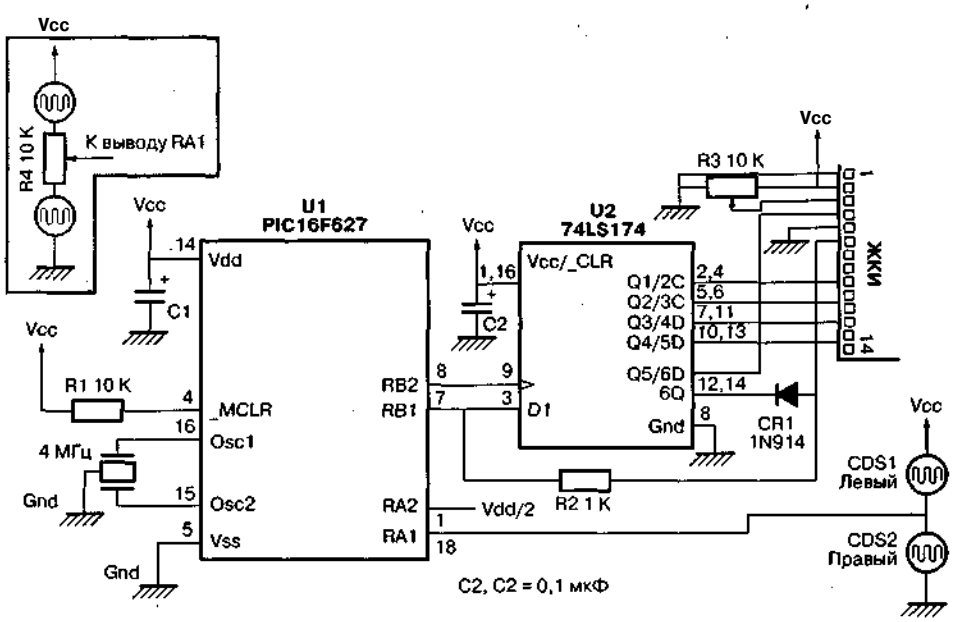

Рис. 4.69. Схема подключения фоторезисторов к микроконтроллеру

На рис. 4.70 показана схема размещения элементов на макетной плате. По сравнению с предыдущими конструкциями почти ничего не изменилось, только добавлены два фоторезистора. На рис. 4.70 не показаны длинные провода, используемые для подключения фоторезисторов. Возможно, для проверки работы нашего устройства вы разместите их прямо на макетной плате, но в работающем устройстве они должны быть вынесены на переднюю панель корпуса робота, как можно дальше друг от друга.

Чтобы в средней точке делителя, образованного сопротивлениями этих фоторезисторов, напряжение в точности равнялось половине напряжения питания, желательно использовать еще подстроечный резистор R4, с движка которого можно снимать выходное напряжение делителя (как показано на врезке к рис. 4.69, в левом верхнем углу).

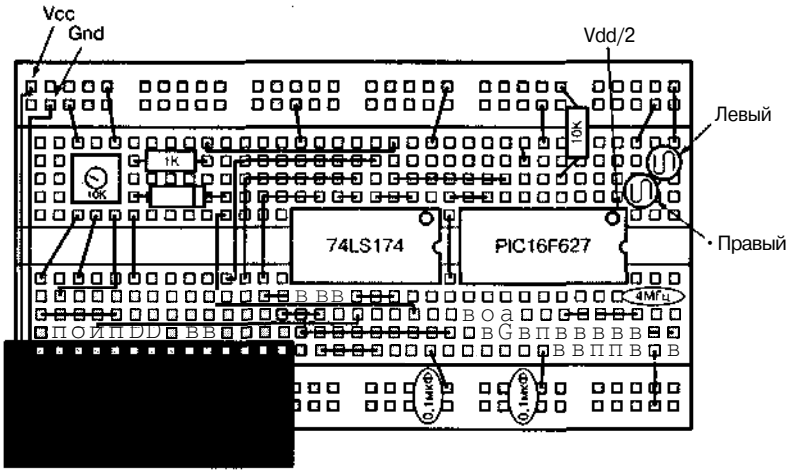

Рис. 4.70. Схема размещения элементов на макетной плате

Перечень использованных в схеме элементов приведен в табл. 4.16. Некоторые указанные там элементы потребуются нам только для второго примера.

Таблица 4.16. Перечень использованных элементов

Обозначение	Элемент	Примечание
U1	PIC16F627	Микроконтроллер
U2	74LS174 (К555ТМ9)*	Сдвиговый регистр
CR1	1N914 (КД521)*	Любой кремниевый диод
R1	10 кОм; 0,25 Вт	«Подтягивающий» резистор
R2	1 кОм; 0,25 Вт	
R3	10 кОм, подстроечный	Для регулирования контрастности дисплея и яркости светодиода
R4	10 кОм, подстроечный	Можно исключить
R5, R6	470 Ом; 0,25 Вт	Используются только во второй схеме
CDS1, CDS2	Фоторезистор, 10 кОм при нулевом освещении	
C1, C2	0,1 мкФ	Для фильтрации напряжения питания микросхем
C3, C4	0,01 мкФ	Используются только во второй схеме
ЖКИ	Жидкокристаллический дисплей (14 выводов), совместимый с Hitachi 44780	
XTAL1	Керамический резонатор на 4 МГц, со встроенными конденсаторами	Для генератора тактовых импульсов микроконтроллера
	Материалы	Макетная плата, монтажные провода, источник питания +5 В

* В скобках указаны отечественные аналоги, добавленные при переводе. - Прим. перев.

Текст программы можно найти в файле Code\Light\light1.c[1]:

```
#include <pic.h>

// 24 апреля 2002 - два фоторезистора, дифференциальная схема включения.
//
// Для индикации используется жидкокристаллический дисплей,
// подключенный к микроконтроллеру через двухпроводной интерфейс
// на основе сдвигового регистра 74LS174.
//
// Используются прерывания от таймера каждую миллисекунду.
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера
// (в электронной версии программы здесь перепутаны
// обозначения выводов. - Прим. перев.):
// RA1 - инверсный вход компаратора 2 - подключен к выходу делителя
//       напряжения, вернее плечо которого образовано левым
//       фоторезистором, а нижнее - правым;
// RA0 - инверсный вход компаратора 1 - подключен к общему проводу Vss;
// RA2 - выход, на котором можно контролировать правильность
//       формирования напряжения Vdd / 2;
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея.

// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.
volatile char LCDDlay = 20;     // Длительность задержки.
volatile char LCDState = 1;     // Номер текущего состояния.

static volatile bit Clock       @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS   @ (unsigned)&TRISB*8+1;
static volatile bit Data        @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS    @ (unsigned)&TRISB*8+2;

char * MessageOut;              // Указатель на выводимое сообщение.
volatile char MessageOuti = 0;  // Смещение текущего символа в строке.
char Message1[3] = "\f*";       // Звездочка выводится слева.
char Message2[18] = "\f                *"; // Звездочка выводится справа.
//                    1  2345678901234567

// Слово конфигурации:
#if defined(_16F627)
    #warning PIC16F627 with external XT oscillator selected
    __CONFIG(0x03F61);   // Для МК PIC16F627:
                         // внешний тактовый генератор XT,
                         // RA6/RA7 используются для ввода-вывода,
                         // внешний сигнал сброса,
                         // таймер PWRT включен,
```

[1] Электронное приложение к этой книге содержит тексты программ в том виде, в котором они представлены в оригинале книги. Обратите внимание на то, что электронная версия данной программы несколько отличается от печатного варианта. Обе версии работоспособны, хотя в оригинале имеется ошибка в комментариях. - *Прим. перев.*

```
                        // сторожевой таймер выключен,
                        // защита кода отключена,
                        // детектор BODEN включен.
#else
    #error Unsupported PICmicro MCU selected
ffendif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS)         // Запись полубайта в ЖКИ.

    ...

LCDByte(char Byte, char RS)             // Запись байта в ЖКИ.

    ...

LCDInit()                               // Инициализация ЖКИ.

    ...

LCDOut(char * const LCDString)   //  Вывод строки на ЖКИ.

    ...

int ADCPoll()      // Опрос состояния бита 7 регистра CMCON
                   // (выхода компаратора 2).
{

    return C2OUT;

}

//  Обработчик прерываний:
void interrupt tmr0_int(void)
{

char temp;         // Служебная переменная (используется в реализации
                   // конечного автомата для ЖКИ).

    if (TOIF) {            // Обработчик прерываний от таймера TMR0.

          TOIF = 0;        // Сбросить флаг прерываний от таймера TMR0.

          RTC++;           // Инкремент счетчика реального времени.

          // Здесь можно разместить дополнительный код
          // для обработки прерываний от таймера TMR0.

          // Конечный автомат для ЖКИ:

          // ...         (как и в предыдущих программах)

    }  // Конец обработчика прерываний от таймера.

    // Здесь можно разместить другие обработчики прерываний.

}  // Конец обработчика прерываний.

// Главная программа:
void main(void)
{

    OPTION = 0xOD1;        // Предделитель работает с таймером TMR0,
                           // коэффициент деления равен 4.

    TMR0 = 0;             // Начальный сброс таймера TMR0.
```

```
    TOIE = 1;              // Разрешение прерываний от таймера TMRO.
    GIE = 1;               // Разрешение обработки прерываний.

    LCDInit();     // Инициализация порта, к которому подключен ЖКИ.

    // Здесь можно проинициализировать другие периферийные устройства.

    CMCON = 0x002;         // Разрешение работы компараторов.
                           // Выходы не инвертируются.
                           // Бит переключения входов CIS = 0.
                           // Режим 2 (с внутренним источником
                           // опорного напряжения).

    TRISA = 0x003;         // Входными являются только выводы RAO и RA1.
           // Обращаем внимание читателя на еще одну ошибку в электронном
           // варианте программы; она приводит к тому, что невозможно
           // проконтролировать величину опорного напряжения
           // на выводе RA2. - Прим. перев.

    VRCON = 0xOEC;         // Разрешение работы внутреннего источника
                           // опорного напряжения Vref;
                           // выход Vref подключен к выводу RA2;
                           // верхний поддиапазон;
                           // биты задания величины напряжения; 12.

    while (1 == 1) {               // Бесконечный цикл.

           //  Здесь можно разместить код верхнего уровня.

           if (!ADCPoll())                 // Каково состояние бита 7
                                           // регистра CMCON?
                  LCDOut(Message1);        // Источник света
                                           // расположен слева.
           else
                  LCDOut(Message2);        // Источник света
                                           // расположен справа.

    }  // Конец оператора while.
}  // Конец главной программы.
```

Здесь после инициализации жидкокристаллического дисплея в бесконечном цикле производится опрос состояния компаратора. Если левый (верхний по схеме на рис. 4.69) фоторезистор освещен больше правого, то напряжение на выходе делителя больше половины напряжения питания. Так как выход делителя подключен к инвертирующему входу компаратора, а инвертирование выхода отключено, то на выходе C2OUT при этом будет напряжение низкого уровня (логический 0), а следовательно, на дисплей будет выводиться сообщение `Message1` (символ звездочки слева). Если источник света, наоборот, больше освещает правый фоторезистор, то будет выведено сообщение Message2 (символ звездочки справа).

Аналоговый компаратор работает непрерывно, для опроса его состояния нет надобности выполнять какую-либо последовательность действий, как часто бывает в цифровых схемах. Поэтому программа получилась достаточно простой.

Вывод курсора на экран не разрешается нигде. По большому счету, необходимость в этом возникает только в тех приложениях, которые требуют, чтобы

пользователь вводил какую-либо информацию. В случае, если дисплей нужен только для вывода данных, мигающий знак подчеркивания обычно только раздражает.

Недостаток схемы заключается в том, что робот получает только один бит информации: какой из двух фоторезисторов освещен больше. Мы не можем измерить, насколько больше, и выяснить, каков общий уровень освещенности. В ситуации, когда один фоторезистор залеплен непрозрачным материалом, а на другой направлен свет яркой лампы, робот будет реагировать так же, как если бы использовались два источника света, один из которых светит чуть ярче другого. Поведение робота не изменится, если в комнате будет полумрак и только в одном ее углу будет чуть мерцать свеча.

Другой недостаток, характерный для всех двухпозиционных автоматических устройств, заключается в том, что наш робот на самом деле не будет двигаться в сторону источника света по прямой траектории. Он будет вилять, то резко забирая вправо, то снова возвращаясь влево. Амплитуда колебаний может оказаться настолько велика[1], что робот так и не сможет приблизиться к источнику света из-за того, что после очередного резкого поворота этот источник окажется вне поля обзора обоих фоторезисторов.

Кроме того, может оказаться, что из-за разброса параметров фоторезисторов при равной их освещенности выходное напряжение делителя не равно половине напряжения питания. В результате колебательная траектория движения робота вместо того, чтобы стремиться точно к источнику света, будет направлена к некоторой точке в стороне от него. Но эта проблема, в отличие от предыдущих, легко решается путем добавления подстроечного резистора. Точной его регулировкой можно добиться, чтобы на дисплее мигали обе звездочки, если робот ориентирован в направлении источника света.

Если мы хотим, чтобы устройство действительно измеряло уровень освещенности, придется усложнить схему и управляющую программу. Микроконтроллер PIC16F627 не имеет встроенного АЦП, так что мы будем измерять сопротивление фоторезистора с помощью RC-цепочки, время разряда которой зависит от сопротивления R (см. рис. 4.67). Чем больше освещен фоторезистор, тем меньше его сопротивление и тем быстрее разряжается конденсатор.

Принципиальная схема второго устройства приведена на рис. 4.71. По сравнению с предыдущей здесь добавлены два резистора (R5 и R6) и конденсатора (C3 и C4). На рис. 4.72 показана схема размещения элементов на макетной плате. Перечень использованных элементов уже приводился в табл. 4.16.

Конечно, было бы интересно разобрать еще одну схему на основе АЦП, но на нашей макетной плате уже не осталось места для аналого-цифрового преобразователя. К тому же программа для работы с АЦП не слишком-то отличается от той, что мы неоднократно использовали для вывода информации на экран жидкокристаллического дисплея. Хотя я потратил немного времени на эксперименты

[1] Даже если читатель не изучал теорию автоматического управления и методы исследования систем на устойчивость, он может снизить скорость вращения двигателя с целью уменьшения амплитуды колебаний. - *Прим. перев.*

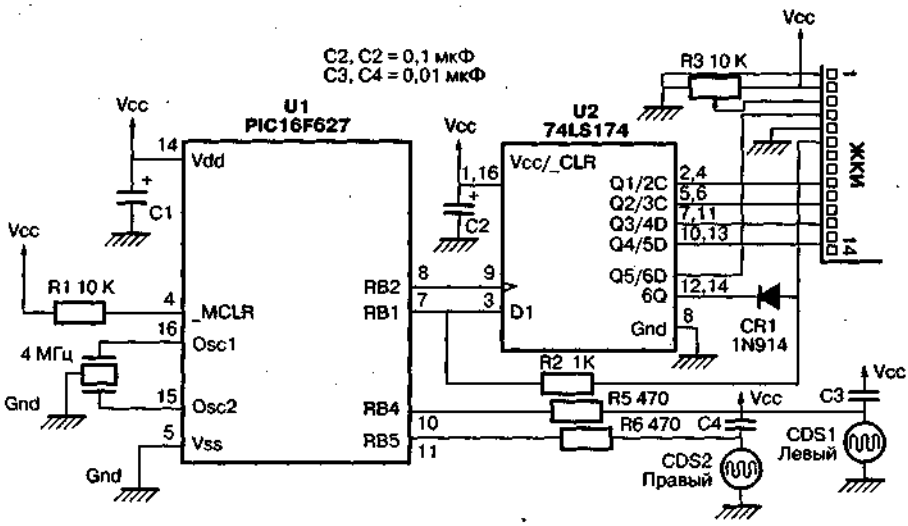

Рис. 4.71. Схема, позволяющая измерять *сопротивление фоторезисторов*

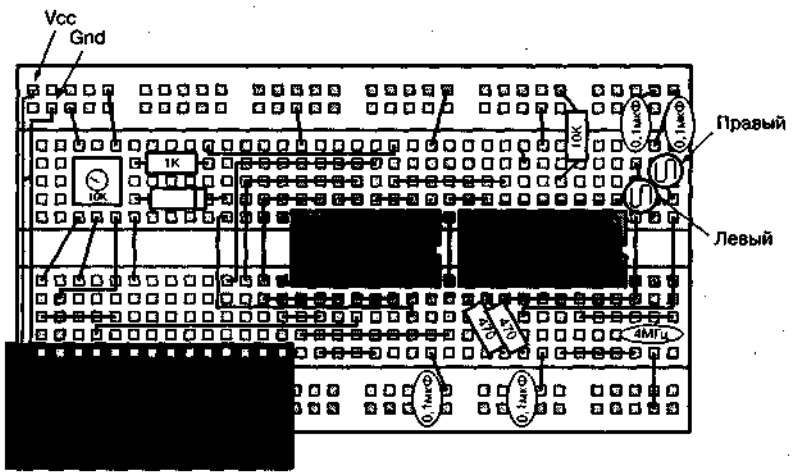

***Рис.* 4.72.** Схема размещения элементов на макетной *плате*

с методами подключения внешней микросхемы АЦП к микроконтроллеру, устройство получилось несколько сложным, и реализовать его не так-то легко.

Поэтому вернемся к нашему проекту. Исходный текст программы можно найти в файле Code\Light\light2.c:

```
#include <pic.h>

// 24 апреля 2002 - два фоторезистора, сопротивление которых
// измеряется с помощью RC-цепочек.
//
```

```
// Для индикации используется жидкокристаллический дисплей,
// подключенный к микроконтроллеру через двухпроводной интерфейс
// на основе сдвигового регистра 74LS174.
//
// Используются прерывания от таймера каждую миллисекунду.
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея;
// RB4 - левый фоторезистор;
// RB5 - правый фоторезистор.

// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.
volatile char LCDDlay = 20;     // Длительность задержки.
volatile char LCDState = 1;     // Номер текущего состояния.

static volatile bit Clock       @ (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS   @ (unsigned)&TRISB*8+1;
static volatile bit Data        @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS    @ (unsigned)&TRISB*8+2;

char * MessageOut; // Указатель на выводимое сообщение.
volatile char MessageOuti = 0;  // Смещение текущего символа в строке.
char MessageL[19] = "\376\200                *";    // Перемещение курсора
                                                    // в первую строку.
//                    1   2   34567890123456789
char MessageR[19] = "\376\300                *";    // Перемещение курсора
                                                    // во вторую строку.
volatile char ADCState = 0;     // Текущее состояние конечного автомата
                                // для работы с фоторезисторами.
volatile char ADCDlay = 1;      // Задержка 1 мс между операциями
                                // с фоторезисторами.
volatile char LeftADC = 0;      // Сопротивление левого фоторезистора.
volatile char RightADC = 0;     // Сопротивление правого фоторезистора.

// Слово конфигурации:
#if  defined(_16F627)
   #warning  PIC16F627 with external XT oscillator selected
   __CONFIG(0x03F61);     // Для МК PIC16F627:
                          // внешний тактовый генератор XT,
                          // RA6/RA7 используются для ввода-вывода,
                          // внешний сигнал сброса,
                          // таймер PWRT включен,
                          // сторожевой таймер выключен,
                          // защита кода отключена,
                          // детектор BODEN включен.
«else
   #error Unsupported PICmicro MCU selected
#endif

// Служебные подпрограммы:
LCDNybble(char Nybble, char RS)         // Запись полубайта в ЖКИ.
```

```
    ...
LCDByte(char Byte, char RS)              // Запись байта в ЖКИ.
    ...
LCDInit()                                // Инициализация ЖКИ.
    ...
LCDOut(char * const LCDString)   // Вывод строки на ЖКИ.
{
    while (LCDState);                    // Ждать готовности ЖКИ.
    MessageOut = LCDString;              // Загрузка выводимой строки.
    LCDState = 100;                      // Послать строку на ЖКИ.
}
//  Обработчик прерываний:
void interrupt tmr0_int(void)
{
char temp;
    if (TOIF) {              // Обработчик прерываний от таймера TMRO.
          TOIF = 0;          // Сбросить флаг прерываний от таймера TMRO.
          RTC++;             // Инкремент счетчика реального времени.
          // Здесь можно разместить дополнительный код
          // для обработки прерываний от таймера TMRO.
          // Конечный автомат для ЖКИ:
          ...
          // Конечный автомат для работы с фоторезисторами:
          if (!LCDState) {       // Если нет работы с ЖКИ.
                switch (ADCState) {
                      case 0:          // Начальная задержка.
                            if (--ADCDlay == 0)
                                  ADCState++;
                            break;
                      case 1:          // Начинаем подачу импульса
                                       // на обе RC-цепочки (1 мс).
                            TRISB4 = 0;
                            TRISB5 = 0;
                            RB4 = 1;
                            RB5 = 1;
                            ADCState++;     // Переход к следующему
                                            // шагу через 1 мс.
                            break;
                      case 2:
                            TRISB4 = 1;     // Закончить подачу
                                            // импульса на левую RC-цепочку.
                            temp = PORTB;   // Разрешить прерывания
                            // по изменении сигнала на входе.
                            RBIF = 0;
                            RBIE = 1;
```

```
                        ADCState++;
                        break;
                case 3:
                        LeftADC = 0xOFF;
                        temp = PORTB;
                        RBIF = 0;       // Запретить прерывания
                        RBIE = 0;       // по изменении входного
                                        // сигнала.
                        ADCState++;
                        break;
                case 4:
                        TRISB5 = 1;     // Закончить подачу
                                        // импульса на правую RC-цепочку.
                        temp = PORTB;
                        RBIF = 0;       // Разрешить прерывания по
                        R8IE = 1;       // изменении входного сигнала.
                        ADCState++;
                        break;
                case 5:
                        RightADC = 0xOFF;
                        temp = PORTB;
                        RBIF = 0;       // Запретить прерывания
                        RBIE = 0;       // по изменении входного
                                        //  сигнала.
                        ADCDlay = 10;
                        ADCState = 0;
                        break;
            }  // Конец оператора switch.
        }  // Конец оператора if.
    }  // Конец обработчика прерываний от таймера.

    // Здесь можно разместить другие обработчики прерываний.

    if (RBIF) {   // Обработка прерываний по изменении входного сигнала.
        switch(ADCState) {
            case 3:         // Работаем с левым фоторезистором.
                LeftADC = TMRO;       // Счетчик времени.
                ADCState++;
                break;
            case 5:         // Работаем с правым фоторезистором.
                RightADC = TMRO;      // Счетчик времени.
                ADCDlay = 10;
                ADCState = 0;
                break;
        }  // endswitch
        temp = PORTB;
        RBIF = 0;      // Очищаем флаг и запрещаем прерывания
        RBIE = 0;      // по изменении входного сигнала.
    }  // Конец обработки прерываний по изменении входного сигнала.
}  // Конец обработчика прерываний.

// Главная программа:
void main(void)
{
```

```
unsigned int i, j;
unsigned int tempLeft, tempRight;

    OPTION = 0x0D1;       // Предделитель работает с таймером TMR0,
                          // коэффициент деления равен 4.

    TH0 = 0;              // Начальный сброс таймера TMR0.
    T0IE = 1;             // Разрешение прерываний от таймера TMR0.
    GIE = 1;              // Разрешение обработки прерываний.

    LCDInit();    // Инициализация порта, к которому подключен ЖКИ.

    // Здесь можно проинициализировать другие периферийные устройства.

    while (1 == 1) {                    // Бесконечный цикл.

           //  Здесь можно разместить код верхнего уровня.

           while (ADCState != 2);               // Ожидаем начала работы
                                                // с фоторезисторами.
           while (ADCState);                    // Ожидаем окончания работы
                                                // с фоторезисторами.
           tempLeft = LeftADC;  tempRight = RightADC;
                        // Сохраняем измеренные значения
                        // сопротивлений фоторезисторов.

           j = (tempLeft / 8) + 1;      // Количество звездочек
                  // для индикации состояния левого фоторезистора.
           for (i = 2; i < 18; i++ )
                  if ((i - 2) <= j)     // Выводим звездочку на дисплей.
                         MessageL[i] = '*';
                  else                  // Выводим пробел.
                      MessageL[i] = ' ';

           j = (tempRight / 8) + 1;     // Количество звездочек
                  // для индикации состояния правого фоторезистора.
           for (i = 2; i < 18; i++ )
                  if ((i - 2) <= j)     // Выводим звездочку на дисплей.
                         MessageR[i] = '*';
                  else                  // Выводим пробел.
                         MessageR[i] = ' ';

           LCDOut(MessageL);      // Состояние левого датчика выводится
                                  // в первой строке дисплея.
           LCDOut(MessageR);      // Состояние правого датчика выводится
                                  // во второй строке дисплея.

    } // Конец цикла while.

}  // Конец главной программы.
```

Для определения времени заряда конденсатора здесь используются прерывания, формируемые при изменении уровня сигнала на входных линиях порта PORTB. Многие эксперты не рекомендуют использовать эту возможность микроконтроллеров PICmicro, но я придерживаюсь противоположного мнения. Многочисленные эксперименты с работающими устройствами показывают, что прерывания по

изменении входного сигнала от внешнего источника достаточно надежны и их использование не вызывает особых трудностей. Единственное, что при этом требуется учитывать, - некоторые особенности программирования:

- прерывания по изменению уровня сигнала на линиях порта PORTB формируются, только если эти линии работают в режиме входных;
- во время использования любой линии PORTB в качестве источника прерываний нельзя обращаться ни к одному разряду регистра этого порта (обращение к любому биту требует использования команды, работающей со всеми восемью разрядами регистра). При необходимости всегда можно использовать служебную переменную для хранения текущего значения данного регистра;
- для очистки флага прерывания надо использовать следующий код:

```
Variable = PORTB;       // Запомнили текущее состояние регистра порта
                        // в служебной переменной.
RBIF = 0;               // Очистили флаг прерывания.
RBIE = 0;               // Запретили (если необходимо) прерывания
                        // по изменении входного сигнала.
```

- следует убедиться, что изменение уровня входного сигнала не произойдет раньше, чем программа окажется готова обработать запрос на прерывание. Только из-за того, чтобы удовлетворить этому требованию, мне пришлось ввести в схему два ограничительных резистора R5 и R6, благодаря которым удалось снизить скорость перезаряда конденсаторов.

После включения питания микроконтроллера программа начинает цикл работы с фоторезисторами. На обе RC-цепочки подается прямоугольный импульс, затем засекается время и разрешаются прерывания по изменению уровня сигнала на входах RB4 и RB5. В момент возникновения прерывания определяется время перезаряда конденсатора и вычисляется значение сопротивления фоторезистора.

Конечный автомат, реализующий интерфейс с фоторезисторами, начинает свою работу, только если не активен конечный автомат, обслуживающий жидкокристаллический дисплей. Это сделано для того, чтобы одновременное выполнение обоих автоматов не привело к нарушению временных промежутков между отдельными операциями. Для тех же целей в главной программе перед началом вывода на дисплей ожидается завершение работы с фоторезисторами.

Хотя в нашей новой схеме микроконтроллер получает намного больше информации об уровне освещенности обоих фоторезисторов, на выполнение всех необходимых для этого операций требуется много времени, и робот может не успеть среагировать на то или иное изменение окружающей обстановки, особенно если скорость его движения достаточно велика.

В нашей программе один цикл опроса состояния двух фоторезисторов занимает 6 мс, что позволяет делать около 330 измерений в секунду. Но большинство современных АЦП работает намного быстрее.

4.30. ЗВУКОВЫЕ ДАТЧИКИ

Использование звукового датчика в электронной схеме не должно представлять особой сложности. У многих роботов часто имеется какое-либо устройство ввода звуковой информации, которое предоставляет пользователю возможность управлять ими, хлопая в ладоши или громко произнося команды. В некоторых автоматических устройствах звуковые сенсоры работают в качестве датчиков столкновений (прислушайтесь к звукам, которые раздаются, когда робот сталкивается с предметами). Подключить к микроконтроллеру микрофон, который бы мог распознавать громкие звуки, не так уж сложно, и мы не упустим эту возможность в наших экспериментах.

Многим сложнее всего понять, что аналоговые сигналы (в том числе и звуковые) могут быть представлены в виде суммы множества колебаний разной частоты. Обычно для такого разложения используются синусоидальный и косинусоидальный сигналы (рис. 4.73).

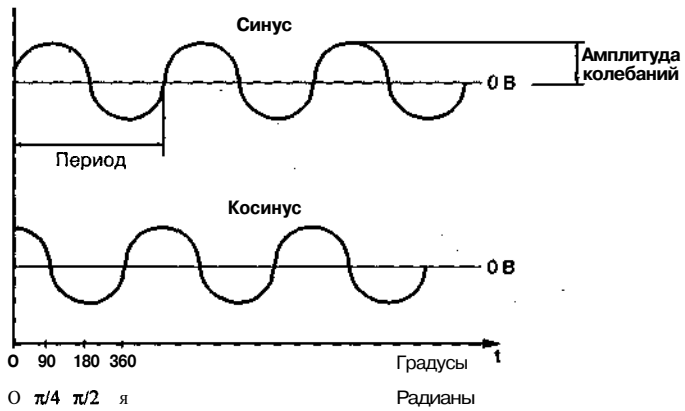

Рис. 4.73. ***Синусоидальное и косинусоидальное*** *колебание*

В виде суммы нескольких таких колебаний (их называют *гармониками),* имеющих разную частоту и амплитуду, можно представить любой аналоговый сигнал. Напомним, что синусоидальное колебание описывается формулой вида

$$x(t) = A \sin(2\pi t / T) = A \sin(2\pi ft),$$

где A - амплитуда, T - период, $f = 1/T$ - частота колебаний, t - время. Именно по этому закону изменяется координата точки, лежащей на ободе колеса радиусом A при его вращении. Когда колесо совершит полный оборот (360°), все будет повторяться с самого начала. Говорят, что начался новый период колебания.

Косинусоидальная волна сдвинута относительно синусоидальной по фазе на 90° (то есть на $\pi/2$):

$$x(t) = A \sin(2\pi ft) = A \cos(2\pi ft - \pi/2).$$

В электронике фазовый угол колебаний $j = 2\pi ft$ обычно измеряется в радианах. Напомним, что для перевода в радианы величины 9, выраженной в градусах, можно воспользоваться формулой

$$j = \pi\theta / 180,$$

где число $\pi = 3{,}14159265359$. Например, 45° - это $\pi/4 = 0{,}7854$ рад, а полный оборот (360°) - это $2\pi = 6{,}2832$ рад.

Тактовые импульсы, которые используются в микропроцессорной технике, обычно имеют прямоугольную форму. Последовательность прямоугольных импульсов называют *меандром*. Сигнал такого вида не слишком подходит для того, чтобы его представляли в виде суммы синусоид и косинусоид. На рис. 4.74 показано, что получится, если сложить три первые гармоники[1] в разложении прямоугольного сигнала, ширина импульса которого равна половине периода. Как можно видеть, суммарное колебание не слишком-то совпадает с прямоугольными импульсами. Для более точного представления следует учесть большее количество гармоник.

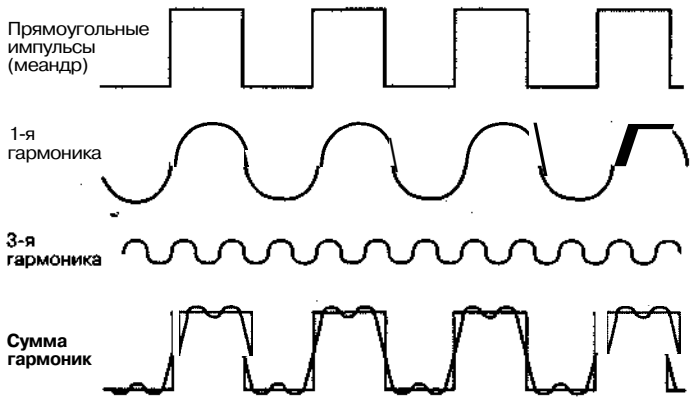

Рис. 4.74. Гармонический анализ прямоугольных импульсов

Таким образом, складывая несколько синусоидальных и косинусоидальных колебаний, можно с достаточной точностью воспроизвести любое аналоговое колебание. При этом частоты суммируемых гармоник всегда кратны частоте самого сигнала. Частота первой гармоники совпадает с частотой сигнала, частота второй

[1] Если сигнал нечетный, то есть его график антисимметричен относительно оси ординат (в данном примере этого можно добиться, расположив начало отсчета в начале какого-нибудь импульса), то все косинусоидальные гармоники будут иметь нулевую амплитуду. Поэтому на рис. 4.74 суммируются только синусоидальные гармоники: нулевая (постоянная составляющая, на рисунке не показана), первая, основная, гармоника (ее частота совпадает с частотой самого сигнала), вторая (также не показана на рисунке, так как в данном случае ее амплитуда равна 0) и третья гармоника, имеющая частоту, в три раза превышающую частоту сигнала. Следует отметить, что колебания, изображенные на рис. 4.73 и 4.74, которые буквально воспроизводятся здесь с английского оригинала, только отдаленно напоминают по форме синусоидальные. - *Прим. перев.*

гармоники в два раза больше частоты сигнала и т.д. Чтобы найти амплитуды гармоник, используют формулы

$$a_i = \frac{2}{T}\int_T x(t)\cos\frac{2\pi i t}{T}dt;\ b_i = \frac{2}{T}\int_T x(t)\sin\frac{2\pi i t}{T}dt,$$

где a_i - амплитуда косинусоид, а b_i - амплитуда синусоид в разложении сигнала; интегрирование ведется по любому отрезку, длина которого равна периоду сигнала. В этом случае само колебание может быть записано в виде ряда Фурье:

$$x(t) = \sum_{i=0}^{\infty}\left(a_i\cos\frac{2\pi i t}{T} + b_i\sin\frac{2\pi i t}{T}\right).$$

Например, рассмотренная выше последовательность прямоугольных импульсов, длительность каждого из которых равна половине периода, будет представлена в виде ряда

$$x(t) = A\sum_{i=0}^{\infty}\frac{1}{2i+1}\cdot\sin\frac{(2i+1)\pi t}{T}.$$

При синтезе сигнала мы должны рано или поздно оборвать бесконечный ряд. Чем позже мы это сделаем, тем больше гармоник учтем, и тем точнее будет полученная сумма представлять искомое колебание.

Но реальна и обратная ситуация: исходный прямоугольный сигнал можно пропустить через специальный *фильтр*, чтобы отсечь одни гармоники и учесть другие.

Фильтр, который пропускает только несколько первых гармоник, называется *фильтром нижних частот* (ФНЧ). Его *амплитудно-частотная характеристика* (АЧХ) представлена на рис. 4.75. Она показывает, во сколько раз ослабляется сигнал определенной частоты.

В электронике коэффициент усиления или ослабления часто измеряют в *децибелах.* Если речь идет о напряжении или токе сигнала, то ослабление в два раза соответствует 6 дБ. Полосу частот, которые пропускает фильтр, оценивают по уровню 3 дБ, что соответствует ослаблению в $\sqrt{2}$ = 1,414 раз. Другими словами, граница полосы пропускания ФНЧ (так называемая *частота среза)* - это частота, на которой коэффициент пропускания фильтра падает до 1/1,414 = 0,707 своей максимальной величины.

Часто для фильтрации сигналов используют так называемые *активные фильтры,* выполненные на основе операционных усилителей. Один такой пример - *фильтр Баттерворта* - показан на рис. 4.76.

Здесь использованы два резистора с одинаковым сопротивлением R и два конденсатора с одинаковой емкостью C. Другие два резистора R1 и R2 задают коэффициент усиления схемы. Для дополнительного усиления я часто использую второй каскад на операционном усилителе.

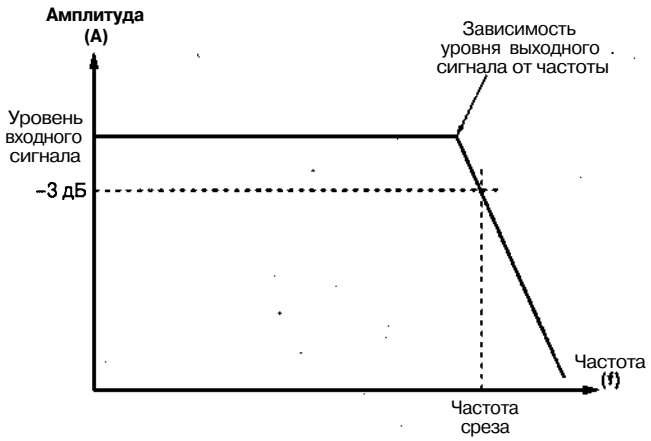

Рис. 4.75. *Амплитудно-частотная* характеристика фильтра *нижних* частот

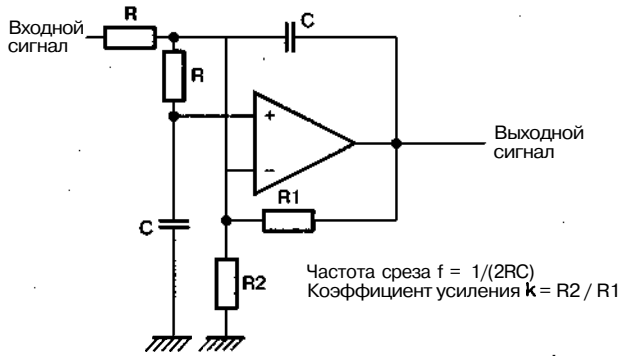

Рис. **4.76.** Фильтр ***Баттерворта***

Если коэффициент усиления достаточно велик, то сигнал на входе фильтра превращается в хороший меандр на выходе. Возможно, вы пришли в недоумение: мы начали с того, что раскладывали прямоугольные импульсы на гармоники, затем пропускали их через фильтр, чтобы отсечь высокочастотные составляющие, а закончили тем, что снова получили прямоугольные импульсы! Но в этом есть смысл. Ведь наша задача состоит не в том, чтобы получить меандр из него самого, а в том, чтобы преобразовать произвольное звуковое колебание в низкочастотную синусоиду. При этом, если на вход фильтра вместе со звуковой командой подается какое-либо более высокочастотное колебание (например, шум двигателей), то фильтр его ослабит, насколько возможно.

Существует большое количество разнообразных фильтров. Кроме ФНЧ, используются также ***фильтры*** *верхних частот* (ФВЧ), *полосовые* и ***режекторные*** фильтры. ФВЧ ослабляют сигналы низкой частоты, а пропускают только старшие гармоники (то есть по сравнению с ФНЧ ведут себя противоположным образом).

Полосовые фильтры пропускают колебания только в определенном диапазоне частот, а режекторные фильтры, наоборот, пропускают все частоты, кроме тех, что лежат в некотором диапазоне. Для фильтрации помех в робототехнике обычно используются полосовые фильтры и ФНЧ.

Итак, чтобы робот имел возможность реагировать на звуковые команды или мог обнаруживать по звуку момент столкновения с окружающими предметами, разработчик должен определить, в каком частотном диапазоне лежит звуковое колебание, на которое должен реагировать робот. Этот диапазон частот необходимо выделить с помощью соответствующего фильтра, чтобы отсеять все остальные звуки.

Проще сказать это, чем сделать, поэтому многие радиолюбители избегают использования звуковых датчиков в своих конструкциях. Часто бывает трудно найти звуковой диапазон, который отличает команду (или звук при столкновении) от посторонних шумов, которые действуют внутри и снаружи автоматического устройства.

Однако я считаю, что в большинстве практических случаев фильтр Баттерворта позволяет получить хорошие результаты.

Чтобы робот как можно меньше реагировал на собственные звуки, надо постараться изолировать датчики от его корпуса. Неплохим решением будет и использование картонного или пластикового конуса, который повысит направленность микрофона.

Современные операционные усилители слабочувствительны к электрическим помехам. Но работа с аналоговыми сигналами всегда требует осторожности. Выход активного фильтра лучше подключать не к стандартному порту ввода-вывода микроконтроллера, а к входу какого-нибудь счетчика (например, к счетчику таймера TMR1). В этом случае проще организовать защиту от дребезга и других паразитных колебаний.

4.31. РАСПОЗНАВАНИЕ ЗВУКОВЫХ КОМАНД

Вас не должно вводить в заблуждение громкое название этого раздела. На самом деле я немного преувеличил возможности описываемого здесь интерфейса. Наш робот сможет реагировать только на громкие звуки, такие как хлопки в ладоши или крики «Стоп!».

Принципиальная схема устройства показана на рис. 4.77. Здесь представлен двухкаскадный активный фильтр нижних частот, выход которого следует подключить ко входу таймера TMR1 (вывод RB6) микроконтроллера PIC16F627.

Здесь мы не приводим схему включения самого микроконтроллера и устройств вывода информации (которые, несомненно, необходимы хотя бы для того, чтобы вы имели возможность убедиться, что робот распознал вашу команду), - эти вопросы здесь не принципиальны. Вы можете использовать любую из схем, описанных выше.

В табл. 4.17 представлен перечень использованных элементов.

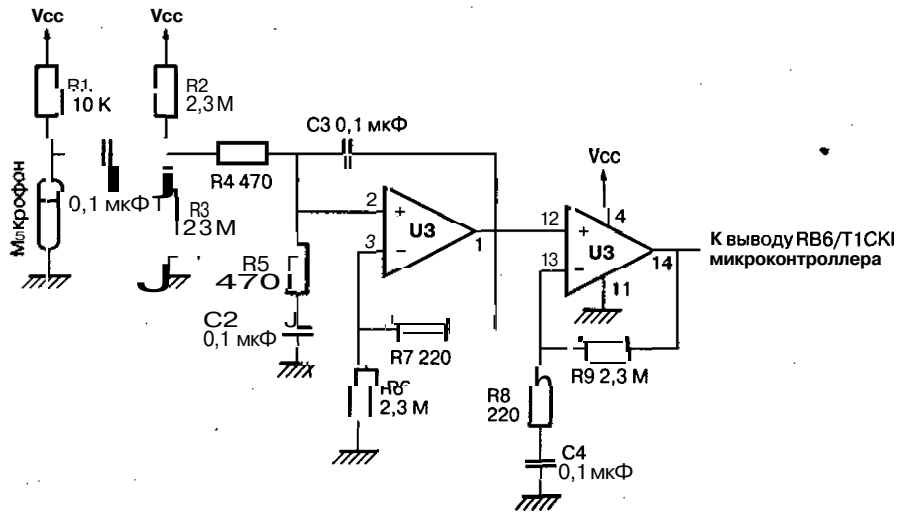

Рис. 4.77. Схема **ФНЧ** для ввода **аудиокоманд** в микроконтроллер

Таблица 4.17. Список элементов схемы, представленной на рис. 4.77

Обозначение	Элемент	Примечание
U3	*LM342 (К140УД2А или Б)**	*Операционный усилитель*
Микрофон		*Любой угольный или электретный*
R1	*10 кОм; 0,25 Вт*	
R2, R3, R6, R9	*2,3 МОм; 0,25 Вт*	
R4, R5	*470 Ом, 0,25 Вт*	
R7, R8	*220 Ом, 0,25 Вт*	
C1 -C4	*0,1 мкФ*	*Конденсаторы любого типа*
	Материалы	*Макетная плата с размещенным на ней микроконтроллером PIC16F627 и жидкокристаллическим дисплеем, монтажные провода, источник питания +5 В*

* *В скобках указаны отечественные аналоги, добавленные при переводе. — Прим. перев.*

Операционный усилитель LM324 в схеме ФНЧ применяется только потому, что он оказался у меня под рукой. Вы можете взять практически любой малосигнальный сдвоенный операционный усилитель (например, 17741). Я часто использую приведенную схему при конструировании самых различных автоматических устройств, так как она обладает хорошей устойчивостью к помехам.

На рис. 4.78 показаны сигналы на входе и выходе ФНЧ (я громко крикнул «D'oh!»). Схема отфильтровала высокочастотные составляющие, и на вход микроконтроллера прошло только несколько импульсов, которые можно проанализировать в программе.

Мы будем опрашивать текущее состояние счетчика таймера TMR1 каждые 20 мс, чтобы узнать, не был ли обнаружен какой-либо громкий звук. Если был, то

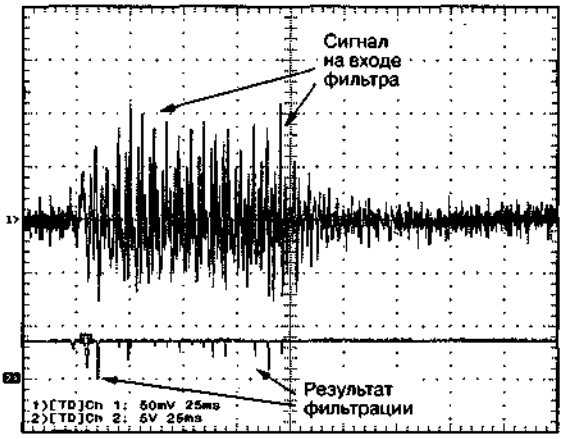

Рис. 4.78. *Действие фильтра нижних частот*

сообщение об этом отобразится вместе с текущим значением счетчика на жидкокристаллическом дисплее в течение одной секунды. В данном проекте можно использовать только микроконтроллер PIC16F627, так как у PIC16F84 нет второго таймера.

Текст программы вы найдете в файле Code\Sound\sound.c:

```
#include <pic.h>

// 27 января 2002 - сообщение о получении звуковой команды
// выводится на ЖКИ.
//
// Для фильтрации сигнала, снимаемого с микрофона, используется
// активный фильтр нижних частот на двух операционных усилителях.
// Выходной сигнал фильтра подается на вход счетчика таймера TMR1
// (вывод RB6/T1CKI микроконтроллера).
// Таймер опрашивается каждые 20 мс, и его текущее значение сравнивается
// с прошлым. Если оно увеличилось на 5 или более, то регистрируется
// наличие сигнала.
//
// Для индикации используется жидкокристаллический дисплей,
// подключенный к микроконтроллеру через двухпроводной интерфейс
// на основе сдвигового регистра 74LS174.
//
// Используются прерывания от таймера каждую миллисекунду.
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RB1 - линия Clock интерфейса жидкокристаллического дисплея;
// RB2 - линия Data интерфейса жидкокристаллического дисплея.

// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.
```

```
volatile char LCDDlay = 20;      // Длительность задержки.
volatile char LCDState = 1;      // Номер текущего состояния.

static volatile bit Clock        9 (unsigned)&PORTB*8+1;
static volatile bit ClockTRIS    9 (unsigned)&TRISB*8+1;
static volatile bit Data         @ (unsigned)&PORTB*8+2;
static volatile bit DataTRIS     9 (unsigned)&TRISB*8+2;

char * MessageOut; // Указатель на выводимое сообщение.
volatile char MessageOuti = 0;   // Смещение текущего символа в строке.

char Message1[18] = "\376\200sound - 0x0xxxx";
                        // Сообщение о полученной звуковой команде.
//                    1    2    3456789012345678
char Message2[18] = "\376\200                  ";
                        // Выводится, когда нет команды.

char soundCounter = 20;                 // Задержка 20 мс.

int OldTMR1 = 0;                        // Прошлое значение таймера.
int CurrentTMR1;                        // Текущее значение таймера.

volatile char soundState = 0;           // Есть звук?
                                        // 0 - нет;
                                        // 1 - есть, ждем дальше;
                                        // 2 .- звук окончен.
// Слово конфигурации:
#if defined(_16F627)
   «warning PIC16F627 with external XT oscillator selected
   __CONFIG(Ox03F61);    // Для МК PIC16F627:
                         // внешний тактовый генератор XT,
                         // RA6/RA7 используются для ввода-вывода,
                         // внешний сигнал сброса,
                         // таймер PWRT включен,
                         // сторожевой таймер выключен,
                         // защита кода отключена,
                         // детектор BODEN включен.
#else
   #error Unsupported PICmicro MCU selected
ffendif

// Служебные подпрограммы:

char GetHex(char Value)          // Перевод числа Value
{                                // в шестнадцатеричное представление.

char returnValue;

   if (Value > 9)
         returnValue = Value + 'A' - 10;
   else
         returnValue = Value + '0';

   returnreturnValue;

}  // Конец функции GetHex

Dlay(int msecs)          // Задержка на msecs миллисекунд.
{
```

```
volatile int variableDlay;

    variableDlay = RTC + msecs + 1;     // Конечное время.
    while (variableDlay != RTC);

}  // Конец функции Dlay

LCDNybble(char Nybble, char RS)         // Запись полубайта в ЖКИ.

    ...

LCDByte(char Byte, char RS)             // Запись байта в ЖКИ.

    ...

LCDInit()                               // Инициализация ЖКИ.

    ...

LCDOut(char « const LCDString)   // Вывод строки на ЖКИ.

...

//  Обработчик прерываний:
void interrupt tmr0_int(void)
{

char temp;
int  TMR1Value;

          T0IF = 0;      // Сбросить флаг прерываний от таймера TMR0.

          RTC++;         // Инкремент счетчика реального времени.

          // Здесь можно разместить дополнительный код
          // для обработки прерываний от таймера TMR0.

          // Работа с звуковым датчиком:

          if (soundState == 0)  // Проверка текущего состояния.
                 if (--soundCounter == 0) {   // Пора?
                        soundCounter = 20;    // Сброс.
                        TMRIValue = (TMR1H * 0x0100) + TMR1L;
                        if (TMRIValue >= (OldTMRI + 2))
                           soundState = 1;
                                 // Есть звук.
                        OldTMR1 = TMRIValue;
                        // Сохранили состояние счетчика таймера.
                 } else;
          else if (soundState == 1) // Звук продолжается?
                 if (--soundCounter == 0) {   // Пора?
                        soundCounter = 20;    // Сброс.
                        TMRIValue = (TMR1H • 0x0100) + TMR1L;
                        if (TMRIValue < (OldTMRI + 2))
                               soundState = 2;
                                      // Звук прекратился.
                        OldTMRI = TMRIValue;
                        // Сохранили состояние счетчика таймера.
                 } else;
```

```
            // Конечный автомат для ЖКИ:
            ...
        }  // Конец обработки прерываний от таймера.
        // Здесь можно разместить код для обработки других прерываний.
    }  // Конец обработчика прерываний.
    // Главная программа:
    void main(void)
    {
        OPTION = 0x0D1;        // Предделитель работает с таймером TMR0,
                               // коэффициент деления равен 4.
        TMR0 = 0;              // Начальный сброс таймера TMR0.
        T0IE = 1;              // Разрешение прерываний от таймера TMR0.
        GIE = 1;               // Разрешение обработки прерываний.
        LCDInit();     // Инициализация порта, к которому подключен ЖКИ.
        // Здесь можно проинициализировать другие периферийные устройства.
        TMR1H = TMR1L = 0;     // Инициализация таймера TMR1.
        T1CON = 0x003;         // Коэффициент деления 1:1;
                               // T1OSCEN (внутренний генератор) выключен;
                               // _T1SYNC активен (синхронизация
                               // с командными циклами включена);
                               // TMR1CS установлен - внешний источник;
                               // TMR1ON установлен - таймер TMR1 включен.
        while (1 == 1) {              // Бесконечный цикл.
            //  Здесь можно разместить код верхнего уровня.
            while (soundState != 2);      // Ждать появление звука.
            CurrentTMR1 = (TMR1H * 0x0100) + THR1L;
            Message1[13] = GetHex((CurrentTMR1 & 0xF000) >> 12);
            Message1[14] = GetHex((CurrentTMR1 & 0x0F00) >> 8);
            Message1[15]  = GetHex((CurrentTMR1 & 0x00F0) >> 4);
            Message1[16]  = GetHex(CurrentTMR1 & 0x0F);
            LCDOut(Message1);              // Вывести сообщение "Sound"
                                           //  (звук обнаружен).
            Dlay(1000);                    // Ждать 1 с.
            LCDOut(Message2);              // Стереть сообщение.
            soundState = 0;        // Сброс.  Ждать следующего звука.
        } // Конец цикла while.
    }  // Конец главной программы.
```

Испытание этого приложения не должно вызвать у вас проблем. Возможно, придется подобрать сопротивление резистора R1, чтобы амплитуда входного сигнала, снимаемого с микрофона, находилась в диапазоне 150-200 мВ, как показано на рис. 4.78.

4.32. УПРАВЛЕНИЕ ДВИГАТЕЛЕМ

Чтобы робот был достаточно мобильным, необходимо иметь какой-либо простой и надежный способ для включения и выключения его двигателей, быстрого изменения направления и, возможно, скорости движения. В больших устройствах для этого используется специальный привод (трансмиссия), но для маленьких механизмов более практичны электрические методы, например Н-образный мост, образованный четырьмя выключателями, в качестве которых обычно используются транзисторы.

Электродвигатели могут управляться с помощью тех же аппаратных и программных средств, которые описывались в предыдущих разделах. Но все сигналы, которые формировал микроконтроллер с помощью рассмотренных выше программ, были однополярными. Это значит, что даже если мы будем регулировать скорость вращения двигателя с помощью программного или аппаратного широтно-импульсного модулятора, двигатель сможет вращаться только в одном направлении. Чтобы изменить направление вращения (и тем самым дать роботу возможность двигаться не только вперед, но и назад), используем Н-образную мостовую схему, показанную на рис. 4.79.

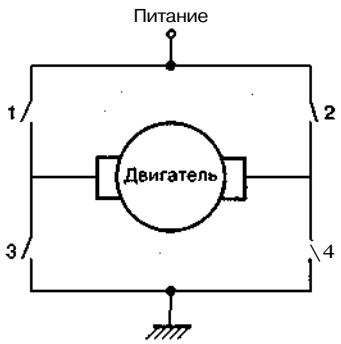

Рис. 4.79. Переключение направления вращения электродвигателя **постоянного тока**

В этой схеме если все выключатели разомкнуты, то обмотки электродвигателя обесточены. Когда замыкаются выключатели 1 и 4, то двигатель начинает вращаться в одном направлении. Если, разомкнув контакты 1 и 4, замкнуть выключатели 2 и 3, то через обмотки двигателя ток потечет в противоположном направлении; значит, направление вращения якоря тоже изменится на противоположное. Нельзя допустить замыкания обоих контактов, находящихся с одной стороны: при этом возникнет короткое замыкание и выйдет из строя источник питания и/или сами электронные ключи.

Если мы хотим изменять не только направление, но и скорость вращения, то управлять состоянием электронных ключей Н-образного моста должен широтно-импульсный модулятор (ШИМ). Чем большую часть периода длится импульс (см. рис. 4.33), тем больше средняя мощность, потребляемая двигателем, и тем быстрее он вращается.

Собрать схему, показанную на рис. 4.79, можно на обычных транзисторах. Но для этой цели выпускаются также специальные микросборки, состоящие из четырех электронных ключей. Обычно я применяю микросхему 293D (рис. 4.80). Она может быть использована для управления двумя двигателями, которые должны подключаться к выводам 3, 6, 11 и 14. Напряжение питания подается на выводы 2, 7, 10 и 15. К выводам 1 и 9 подводится напряжение, управляющее электронными ключами (это может быть ШИМ-сигнал).

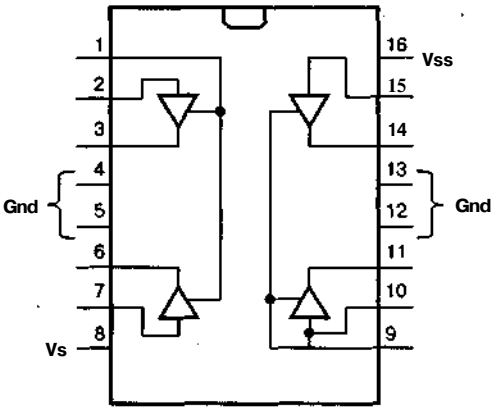

Рис. 4.80. *Микросхема 293D для* ***управления*** *направлением вращения* ***электродвигателя***

К выводу Vs подводится напряжение питания +5 В для самой микросхемы, а к выводу Vss - напряжение питания для электродвигателей; оно должно находиться в диапазоне от 4,5 до 36 В. Максимальный ток, который могут потреблять подключаемые к микросхеме двигатели, не должен превышать 500 мА. Внутри микросхемы 293D уже имеются специальные шунтирующие диоды, которые обычно подключаются параллельно обмоткам электродвигателей (а также к любым другим нагрузкам, имеющим преимущественно индуктивное сопротивление), поэтому при использовании 293D не требуются внешние шунтирующие диоды.

Схема подключения источника ШИМ-сигнала управления и одного электродвигателя к микросхеме 293D приведена на рис. 4.81.

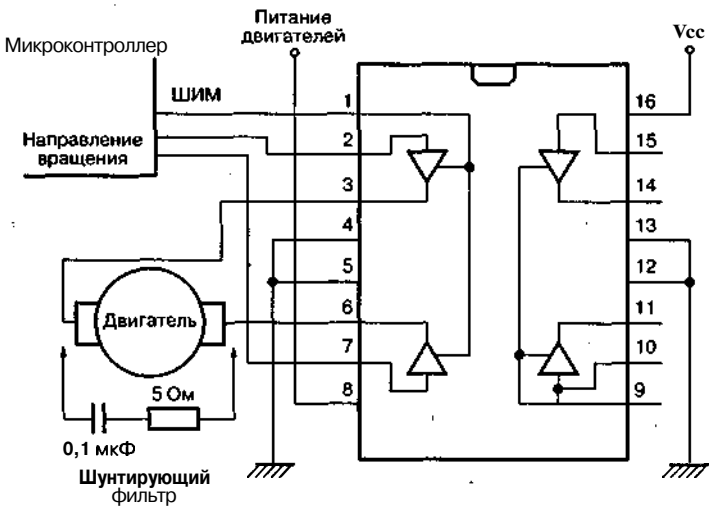

Рис. 4.81. *Управление* ***электродвигателем***

Параллельно щеткам электродвигателя здесь подключена цепочка из последовательно соединенных резистора и конденсатора - она предназначена для снижения импульсных помех, возникающих при его работе. Без такой защиты при включенном двигателе могут возникать перебои в работе микроконтроллера. Емкость конденсатора следует выбрать около 0,1 мкФ, а резистор должен иметь сопротивление примерно 5 Ом и быть рассчитан на мощность не менее 2 Вт.

Следует помнить, что в качестве электронных ключей в микросхеме 293D используются биполярные транзисторы, поэтому падение напряжения на каждом из них (в открытом состоянии) составляет около 0,7 В. Так как в Н-образной схеме последовательно соединены два ключа, то двигателю «достается» напряжение, которое на 1,4–1,5 В меньше, чем подаваемое на вход Vss. Умножив это напряжение на ток, потребляемый двигателем, можно оценить, какую тепловую мощность будет рассеивать микросхема. Хотя максимальный суммарный выходной ток микросхемы 293D может достигать 1 А, для нормальной работы желательно, чтобы его значение было меньше.

В литературе по робототехнике вы найдете более подробное изложение темы управления электродвигателями и встретите описание других электронных ключей, которые могут для этого использоваться, например КМОП транзисторы.

Вопрос об управлении электроприводом достаточно сложен — ему следует уделить большое внимание, так как его решение в немалой степени повлияет на эффективность и функциональность автоматического устройства. Но для управления небольшим роботом вполне подойдет вышеупомянутая микросхема 293D.

4.33. МОДЕЛИ ФИРМЫ TAMIYA В КАЧЕСТВЕ ПРОТОТИПА РАДИОЛЮБИТЕЛЬСКИХ КОНСТРУКЦИЙ

Не всегда просто найти для робота хороший прототип. Иногда радиолюбители используют электромеханические игрушки, однако зачастую у них достаточно узкое назначение, поэтому с ними тяжело работать. Можно, конечно, приобрести какой-нибудь из специальных наборов, но их стоимость достаточно высока. Прекрасный пример «золотой середины» между двумя названными крайностями - модели фирмы Tamiya[1].

В этом разделе мы рассмотрим, как простая модель (фактически игрушка) может быть использована в экспериментах по робототехнике. Описанная ниже конструкция недостаточно оптимальна и надежна, чтобы стать законченным промышленным устройством, но она послужит хорошей базой для дальнейших опытов.

Рис. 4.82. *Внешний вид робота, выполненного на основе модели мыши фирмы Tamiya*

На рис. 4.82 показано простое устройство на основе одной из разработок Tamiya, которую я несколько лет назад всего за несколько минут превратил из игрушки в самого настоящего робота. Здесь использована модель мыши (скорее, грызуна, чем одноименного компьютерного манипулятора); в каталоге она значится под именем *Wall-Hugging Mouse* или под номером 70068. В ее корпусе прекрасно удалось разместить дополнительную электронную схему для управления движением (на основе микроконтроллера, конечно).

[1] С продукцией этой японской фирмы вы можете познакомиться, заглянув на русскоязычный сайт www.tamiya.ru. - *Прим перев.*

Модель мыши имеет два колеса, каждым из которых управляет отдельный электродвигатель постоянного тока. Направление движения задается с помощью контактного детектора на основе микропереключателя. Один двигатель включается при замыкании микропереключателя, второй - при его размыкании. В результате мышь движется вдоль стенки по зигзагообразной траектории и может даже выбраться из несложного лабиринта[1].

Прежде чем начать экспериментировать с этой простой игрушкой, соберите набор и посмотрите, как он работает. При сборке лучше припаять провода, так как скрученные контакты быстро растреплются, пока мышка будет бегать по комнате. Проводить эксперименты с данной моделью крайне увлекательно, но они быстро надоедают, и скоро вы будете готовы использовать игрушку в качестве основы для каких-то более сложных конструкций.

Если вы уже готовы, то начнем:

1. Удалите батарейку и ее держатель.
2. Припаяйте двухконтактный разъем для 9-вольтовой батарейки.
3. Можно (хотя и необязательно) удалить микропереключатель, управляющий двигателями.
4. Просверлите отверстия для крепления печатной платы, на которой будет размещаться электронное устройство управления.

• В качестве детектора касания в нашей конструкции можно использовать все тот же микропереключатель, который уже имеется в комплекте с мышкой. Но для обнаружения объектов с обеих сторон нужны два контактных усика и два переключателя. Я взял второй из другого такого же набора.

Разумеется, еще лучше использовать инфракрасный обнаружитель объектов, вроде того, который уже описан в этой главе.

Принципиальная схема блока управления представлена на рис. 4.83. Внешний вид печатной платы показан на рис. 4.84. На то, чтобы собрать эту схему, лично я потратил два часа. В табл. 4.18 приведен список использованных элементов и материалов.

В устройстве используется 9-вольтовая щелочная батарея и переключатель (на днище мышки) для включения питания. Не забывайте, что переключатель нужно располагать *до* цепей питания (то есть диода CR3 и фильтрующих конденсаторов C5 и C7), иначе эти конденсаторы будут заряжены даже при выключенном питании - тогда их токи утечки будут разряжать батарейку, а кроме того, в момент включения питания могут возникнуть нежелательные импульсы.

Возможно, вас удивят большая емкость конденсатора C7 (1000 мкФ) и наличие блокирующего диода CR3 (см. рис. 4.83). Как будет показано ниже в этом разделе, применение одной 9-вольтовой батарейки для питания двух 1,5-вольтовых электродвигателей, микросхемы L293D и микроконтроллера - далеко не самое оптимальное, но вполне допустимое решение.

---

[1] Для этого не понадобится особый интеллект. Как известно, простой алгоритм, позволяющий найти путь из лабиринта (не обязательно самый короткий), заключается в том, чтобы все время держаться какой-либо стены - правой или левой. - *Прим перев.*

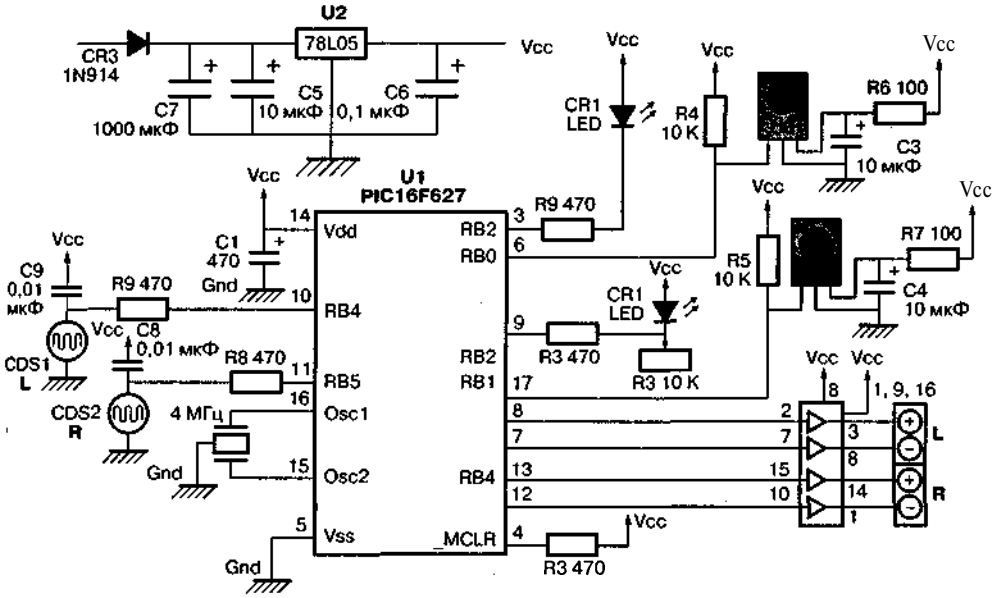

Рис. 4.S3. Схема для ***управления моделью*** *мыши*

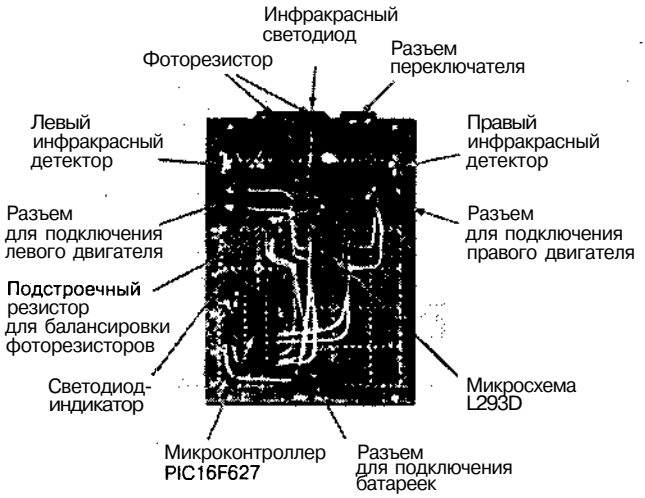

Рис. 4.84. Внешний вид печатной платы *блока управления*

Таблица 4.18. Список использованных элементов

Обозначение	*Элемент*	*Примечание*
U1	*PIC16F627*	*Микроконтроллер*
U2	*78L05 (КР142ЕН5А)**	*Микросхема стабилизатора напряжения*

Таблица 4.18. Список использованных элементов (окончание)

Обозначение	Элемент	Примечание
U3, U4	Инфракрасный детектор	P/C- 1018SCL или другой совместимый (три вывода)
U5	L293D	
CR1	Светодиод	Любой видимого диапазона
CR2	Светодиод	Любой инфракрасный
CR3	1N914	Любой кремниевый диод, рассчитанный на ток 200 мА
R1, R4, R5	10 кОм; 0,25 Вт	
R2, R3, R8, R9	470 Ом; 0,25 Вт	
R6, R7	100 Ом; 0,25 Вт	
CDS1, CDS2	Фоторезисторы, 10 кОм при нулевом освещении	При освещении должны уменьшать свое сопротивление
C1, C6	0,1 мкФ	Конденсаторы любого типа
C3 – C5	10 мкФ; 16 В	Оксидные конденсаторы
C7	1000 мкФ; 16 В	Оксидный конденсатор
C8, C9	0,01 мкФ	Конденсаторы любого типа
XTAL1	Керамический резонатор на 4 МГц, со встроенными конденсаторами	Для генератора тактовых импульсов микроконтроллера
	Материалы	Модель мыши фирмы Tamiya, макетная плата, монтажные провода, батарейка 9 В

**В скобках указаны отечественные аналоги, добавленные при переводе. -Прим. перев.*

Во время работы двигателей потребляемый ток увеличивается; как следствие, на внутреннем сопротивлении батарейки падает достаточно большое напряжение. Чтобы свести эти потери к минимуму, надо использовать достаточно качественную щелочную батарейку с низким внутренним сопротивлением. Чтобы обеспечить микроконтроллер стабильным питанием, используем фильтрующий конденсатор большой емкости. Схема источника питания в данном случае далека от идеальной, но практика показала, что со своей задачей она вполне справляется. И контактные датчики, и дистанционное управление работают без заметных перебоев.

Когда будете воспроизводить эту конструкцию, расположите панель для микроконтроллера в таком месте, откуда его можно было бы без труда извлечь, например в левой задней части робота. Поверьте, нет ничего более обидного, когда прекрасно работающую схему приходится разбирать только из-за того, что иначе невозможно вытащить микроконтроллер для перепрограммирования!

При монтаже клемм на двигатель необходимо следить за тем, чтобы клемма со знаком «плюс» располагалась со стороны передней части робота, а положительный вывод двигателя соединялся с ней красным проводом. Отрицательный провод должен быть черным: в противном случае легко перепутать провода при наладке схемы, а это может привести самое малое к тому, что робот поедет назад (если вообще сможет куда-либо ездить). Также имеет смысл делать провода как

можно более короткими, чтобы уменьшить риск их наматывания на колеса и другие движущиеся части.

Если вы читали главу без пропусков, у вас не должен вызвать затруднений монтаж фоторезисторов, а также инфракрасного датчика системы дистанционного управления и обнаружителя объектов. На схеме с рис. 4.83 легко узнать проекты, описанные выше в этой главе.

Текст управляющей программы вы найдете в файле Code\Wallhug\wallhug.c:

```
#include <pic.h>

// 28 апреля 2002 - модификация модели мыши фирмы Tamiya.
//
// Инфракрасный обнаружитель объектов (ШИМ-сигнал частотой 38 кГц).
//
// Дистанционное управление на основе инфракрасного детектора
// фирмы Sony.
//
// Для индикации используется светодиод.
//
// Формат пакета данных:
//
//       Стартовый бит     1                      0
// ---+                 +------+      +------+             +-------
//    |                 |      |      |      |             |
//    +-----------------+      +------+      +-------------+
//
//    |   2,4 мс       |540мкс|660мкс|540мкс|   1,2 мс    |
//
//                     | 1,2 мс   |    1,76 мс     |
//
//                     |     300     |         440         |
//
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RA0 - правый инфракрасный детектор для обнаружения объектов
// (опрашивается программно);
// RA4 - светодиод для индикации принятой команды дистанционного
// управления;
// RB0 - левый инфракрасный детектор для приема команд дистанционного
// управления фирмы Sony и для обнаружения объектов
// (прерывания по положительному перепаду);
// RB1 - левый двигатель (вывод "-");
// RB2 - левый двигатель (вывод "+");
// R83 - светодиод инфракрасного обнаружителя объектов;
// RB4 - левый фоторезистор;
// RB5 - правый фоторезистор;
// RB6 - правый двигатель (вывод "-");
// RB7 - правый двигатель (вывод "+");
//
```

```
// Глобальные переменные и константы:
int RTC = 0;                    // Счетчик реального времени.

#define remote0          0x06EF // Определения констант для команд
#define remote1          0x0FEF // дистанционного управления.
#define remote2          0x07EF
#define remote3          0x0BEF
«define remote4          0x03EF
#define remote5          0x0DEF
#define remote6          0x05EF
#define remote7          0x09EF
«define remote8          0x01EF
#define remote9          0x0EEF
#define remoteVolUp      0x0B6F
#define remoteVolDown    0x036F
#define remoteChanUp     0x0F6F
#define remoteChanDown   0x076F
#define remotePrevChan   0x022F
#define remoteMute       0x0D6F
#define remotePower      0x056F
unsigned int  DataIn;                   // Входные данные.
unsigned int  DataTime = 0;
unsigned char DataInCount = 0;          // Количество входных символов.
unsigned int  CurrentRTC;

volatile char LeftCollision  = 0;       // Есть объект слева?
volatile char RightCollision = 0;       // Есть объект справа?

volatile char ADCState = 0;     // Текущее состояние конечного
                                // автомата опроса фоторезисторов.
volatile char ADCDlay = 1;      // Задержка 50 мс между срабатываниями
                                // конечного автомата.
volatile unsigned char LeftADC = 0x0FF;
                                // Состояние левого фоторезистора.
volatile unsigned char RightADC = 0x0FF;
                                // Состояние правого фоторезистора.

#define goStop            0x030         // Стоп двигатели!
#define goForward         0x072         // Вперед!
#define goReverse         0x0B4         // Реверс!
#define turnLeft          0x074         // Левый двигатель, реверс!
                                        // (поворот налево).
«define turnRight         0x0B2         // Правый двигатель, реверс!
                                        // (поворот направо).
                                        // Замечание: на выводах
                                        // RB4/RB5 - высокий уровень.
#define LeftForward       1             // Левый двигатель, вперед!
«define LeftStop          0             // Левый двигатель, стоп!
#define LeftReverse       -1            // Левый двигатель, реверс!
#define RightForward      1             // Правый двигатель, вперед!
#define RightStop         0             // Правый двигатель, стоп!
#define RightReverse      -1            // Правый двигатель, реверс!

unsigned char motorState = goStop;      // Состояние двигателей.
unsigned char PWMDuty = 15;
unsigned char PWMCycle = 0;
unsigned char OpnCount = 1;

char ExecuteFlag = 0;           // Флаг для индикации состояния.
```

```
// Слово конфигурации:
#if defined(_16F627) .
    #warning PIC16F627 with external XT oscillator selected
    __CONFIG(0x03F21);     // Для МК PIC16F627:
                           // внешний тактовый генератор XT,
                           // RA6/RA7 используются для ввода-вывода,
                           // внешний сигнал сброса,
                           // таймер PWRT включен,
                           // сторожевой таймер выключен,
                           // защита кода отключена,
                           // детектор BODEN включен.
#else
    #error Unsupported PICmicro MCU selected
flendif

//  Обработчик прерываний:
void interrupt tmr0_int(void)
{

char temp;

    if (T0IF) {            // Обработчик прерываний от таймера TMR0.

          T0IF = 0;        // Сбросить флаг прерываний от таймера TMR0.

          RTC++;           // Инкремент счетчика реального времени.

          // Здесь можно разместить дополнительный код
          // для обработки прерываний от таймера TMR0.

          // Узнать состояние двигателей и выключить их:
          PORTB = temp = (PORTB & 0x039) | 0x030;

          // Проверка на потерянный пакет данных и останов до конца
          // текущего пакета:
          if ((!RB0) && (DataInCount == 0)) { // Есть входной пакет?

                DataInCount = 13;      // Ждем 12 информационных
                                       // импульсов и конец стартового.

                INTF = 0;              // Разрешение прерываний
                INTE = 1;              // по изменении сигнала.

          } else if (DatalnCount == 0) { // Обнаружение объектов.

                TRISB3 = 0;    // Разрешение выхода ШИМ-сигнала.

                while (TMR0 < 64);     // В течение 64 мс.

                if (!RB0)      // Если уровень низкий - обнаружен
                               // объект слева.
                      if (LeftCollision < 3)        // Ждать троекратного
                                                    // срабатывания.
                            LeftCollision++;
                      else;
                else
                      LeftCollision = 0;

                if (!RA0)      // Если уровень низкий - обнаружен
                               // объект справа.

                      if(RightCollision < 3)   // Ждать троекратного
                                               // срабатывания.
```

```
                    LeftCollision++;
                . else;
        else
                RightCollision = 0;

        TRISB3 = 1;

}

// Программная генерация ШИМ-сигнала для управления
// скоростью вращения двигателей.

if (PWMDuty == 0)           // Нулевая ширина импульсов?
        ;
else if (PWMDuty == 29) // Ширина импульса равна периоду?
        PORTB = temp | motorState;
else                            // Нечто среднее?
        if (PWMCycle <= PWMDuty)
                PORTB = temp | motorState;
        else;

if (++PWMCycle == 30)
        PWMCycle = 0;

// Индикация:
if ((!ExecuteFlag) && (--OpnCount == 0)) {
                                // Опрос фоторезисторов.
        OpnCount = 1;
        PORTB = (PORTB & 0x039) | (motorState = goStop);
        if (LeftADC < RightADC)
                RA4 = 0;        // Зажечь светодиод -
                                // источник света справа.
        else
                RA4 = 1;        // Потушить светодиод -
                                // источник света слева.

}

// Прием команд дистанционного управления:
if (OataTime) {
        if (DataInCount == 0) {      // Есть команда.
                DataInCount = 12;    // Ждать 12 импульсов.
                DataIn = 0;
        }  else {                    // Прием очередного бита.
                DataIn = DataIn << 1;
                if ((DataTime > 225) && (DataTime < 375)) {
                                                // Получена 1.
                        DataIn++;
                        if (--DataInCount == 0) {
                                OpnCount = 200;       // Конец.
                                if (DataIn == remote2) {
                                        motorState = goForward;
                                } else if (DataIn == remote4) {
                                        motorState = turnLeft;
                                } else if (DataIn == remote6) {
                                        motorState = turnRight;
                                } else if (DataIn == remote8) {
                                        motorState = goReverse;
```

```
                } else
                  if (DataIn == remoteVolUp) {
                       if (PWMDuty < 29)
                             PWMDuty++;
                       else;
                  } else
                  if (DataIn == remoteVolDown) {
                       if (PWMDuty != 0)
                             PWMDuty--;
                       else;
                  } else
                  if (DataIn == remotePower) {
                       ExecuteFlag = 1;
                       motorState = goStop;
                       OpnCount = 1; // Стоп.
                } else {
                       ExecuteFlag = 0;
                       motorState = goStop;
                       OpnCount = 1; // Стоп.
                }
          }
    } else if ((DataTime > 375) &&
          (DataTime < 500)) {  // Получен 0.
          if (--DataInCount == 0) {
                OpnCount = 200; // Конец.
                if (DataIn == remote2) {
                       motorState = goForward;
                } else if (DataIn == remote4) {
                       motorState = turnLeft;
                } else if (DataIn == remote6) {
                       motorState = turnRight;
                } else if (DataIn == remote8) {
                       motorState = goReverse;
                } else
                  if (DataIn == remoteVolUp) {
                       if (PWMDuty < 29)
                             PWMDuty++;
                       else;
                } else
                  if (DataIn == remoteVolDown) {
                       if (PWMDuty != 0)
                             PWMDuty-;
                       else;
                } else
                  if (DataIn == remotePower) {
                       ExecuteFlag = 1;
                       motorState = goStop;
                       OpnCount = 1; // Стоп.
                } else {
                       ExecuteFlag = 0;
                       motorState = goStop;
                       OpnCount = 1; // Стоп.
                }
          }
    } else if (--DataInCount != 12)      // Ошибка.
          DataInCount = 0;
```

```
            }
            DataTime = 0;          // Все сначала.
        }

        // Задержка:
        CurrentRTC += 0x0100;

        // Проверка на потерянный импульс:
        if ((DataInCount) && (CurrentRTC > 0x00900))
            DataInCount = 0;   // Сброс. Ждать следующей команды.

        // Конечный автомат для работы с фоторезисторами:
        if (!DataInCount) {   // Если не принимается команда.
            if (ADCState == 0) {  // Начальная задержка.
                if (--ADCDlay == 0)
                    ADCState++;
            } else if (ADCState == 1) {
                // Подать импульс на обе RC-цепочки (1 мс).
                PORTB = PORTB | 0x030;
                TRISB4 = 0;
                TRISB5 = 0;
                ADCState++;     // Переход к следующему
                                // состоянию через 1 мс.
            } else if (ADCState == 2) {
                TRISB4 = 1;    // Прекратить левый импульс.
                temp = PORTB;
                RBIF = 0;      // Разрешение прерываний
                RBIE = 1;      // по изменении сигнала.
                ADCState++;
            } else if (ADCState == 3) {
                LeftADC = 0xOFF;
                temp = PORTB;
                RBIF = 0;      // Запрещение прерываний
                RBIE = 0;      // по изменении сигнала.
                ADCState++;
            } else if (ADCState == 4) {
                TRISB5 = 1;    // Прекратить правый импульс.
                temp = PORTB;
                RBIF = 0;      // Разрешение прерываний
                RBIE = 1;      // по изменении сигнала.
                ADCState++;
            } else if (ADCState == 5) {
                RightADC = 0xOFF;
                temp = PORTB;
                RBIF = 0;      // Запрещение прерываний
                RBIE = 0;      // по изменении сигнала.
                ADCDlay = 50;
                ADCState = 0;
            }
        }
    }  // Конец обработчика прерываний от таймера.

    // Здесь можно разместить другие обработчики прерываний.

    if (INTF) {                // Обработка прерываний по изменении
                               // сигнала на входе RB0/INT.
        DataTime = CurrentRTC + TMR0; // Время для текущего бита.
```

```
            CurrentRTC = 0x0FFFF - TMR0; // Время для следующего бита.
            INTF = 0;                    // Сброс флага прерываний.
    }  // Конец обработчика прерываний по изменении сигнала на RB0/INT.

    if (RBIF) {           // Обработка прерываний по изменении
                          // сигнала на входных линиях порта PORTB.
            if (ADCState == 3) {
                    LeftADC = TMR0;       // Для левого фоторезистора.
                    ADCState++;
            } else {
                    RightADC = TMR0;      // Для правого фоторезистора.
                    ADCDlay = 50;         // Повторять каждые 50 мс.
                    ADCState = 0;
            }
            temp = PORTB;
            RBIF = 0;            // Запретить прерывания по изменении
            RBIE = 0;            // сигнала на входах PORTB.
    }       // Конец обработки прерываний по изменении
            // сигнала на входных линиях порта PORTB.

}  // Конец обработчика прерываний.

// Служебные подпрограммы:
void Dlay(int msecs)             // Задержка на msecs миллисекунд.
{

int valueDlay;

    valueDlay = RTC + msecs + 1; // Время окончания задержки.
    while (valueDlay != RTC);    // Ждать.

}  // Конец функции Dlay.

void LEDOutput(int state )       // Установка состояния светодиода.
{

    if (state)
            RA4 = 0;             // Включить светодиод.
    else
            RA4 = 1;             // Выключить светодиод.

}  // Конец функции LEDOutput.

int GetLeftLight()// Опрос состояния левого фоторезистора.
{

    return LeftADC;

}

int GetRightLight()       // Опрос состояния правого фоторезистора.
{

    return RightADC;

}

int GetLeftWhisker()      // Опрос состояния левого контактора.
{

    if (LeftCollision == 3)
            return 1;            // Есть касание.
    else
```

```
        return 0;              // Нет касания.
}
int GetRightWhisker()     // Опрос состояния правого контактора.
<
    if (RightCollision == 3)
        return 1;              // Есть касание.
    else
        return 0;              // Нет касания.
}
void LeftMotor(int Movement, int Speed) // Управление левым двигателем.
{
    switch (Movement) {
        case 1:                                  // Вперед.
            motorState = (motorState & 0x0F9) + 2;
            break;
        case 0:                                  // Стоп.
            motorState = motorState & 0x0F9;
            break;
        case -1:                                 // Назад.
            motorState = (motorState & 0x0F9) + 4;
            break;
    }
    PWMDuty = Speed;
}
void RightMotor(int Movement, int Speed)// Управление правым двигателем.
{
    switch (Movement) {
        case 1:                                  // Вперед.
            motorState = (motorState & 0x03F) + 0x040;
            break;
        case 0:                                  // Стоп.
            motorState = motorState & 0x03F;
            break;
        case -1:                                 // Назад.
            motorState = (motorState & 0x03F) + 0x080;
            break;
    >
    PWMDuty = Speed;
}
// Главная программа:
void main(void)
{
    OPTION = 0x0D1;        // Предделитель работает с таймером TMR0,
                           // коэффициент деления 1:4.
    TMR0 = 0;              // Начальный сброс таймера TMR0.
    T0IE = 1;              // Разрешение прерываний от таймера TMR0.
    GIE = 1;               // Разрешение обработки прерываний.
```

```
    // Здесь можно разместить код верхнего уровня.

    INTEDG = 1;           // Прерывания вырабатываются по положительному
                          // перепаду сигнала на выходе RBO/INT.

    CCPR1L = 13;          // Ширина ШИМ-импульсов равна 50% периода.
    CCP1CON = 0b000001111;          // Разрешить работу модуля ШИМ.

    PR2 = 26;                   // При частоте 38 кГц длительность
                                // периода равна 26 мкс.
    TMR2 = 0;                   // Сброс таймера TMR2.
    T2CON = 0b000000100;        // Включить таймер TMR2,
                                // коэффициент предделителя 1:1.

    CHCON = 0x007;              // Выключить компараторы порта PORTA.

    PORTA = 0x010;              // RA4 - выход
                                // (светодиод пока не горит).
    TRISA = OxOEF;

    PORTB = 0x000;
    TRISB = 0x039;              // RB7/RB6 и RB2/RB1 - выходы.

    while (1 == 1) {            // Бесконечный цикл.

        if (ExecuteFlag) {

            // Здесь можно разместить код для управления
            // действиями робота.

        } else {

            // Здесь можно разместить код для сброса
            // робота в начальное состояние.

        }

    }  // Конец цикла while.

} // Конец главной программы.
```

Используя эту программу в качестве основы, можно поэкспериментировать со следующим набором функций:

```
int GetLeftWhisker();           // Опрос состояния левого
                                // детектора столкновений.
int GetRightWhisker();          // Опрос состояния правого
                                // детектора столкновений.
// Детекторы могут быть контактными или инфракрасными.

int GetLeftLight();       // Опрос состояния левого фоторезистора.
int GetRightLight();      // Опрос состояния правого фоторезистора.
// Фоторезисторы могут быть включены по схеме делителя напряжения
// или в виде RC-цепочки.

void LeftMotor(int Movement [ , int Speed ] );        // Управление
                                                      // левым двигателем.
void RightMotor(int Movement [ , int Speed ] );       // Управление
                                                      // правым двигателем.
// Второй параметр (Speed) доступен при использовании инфракрасного
// пульта дистанционного управления, но недоступен для радиоуправления.

void LED(iut State);            // Включить/выключить светодиод.
```

```
void LCDOutput(char * Message);   // Вывод сообщения на ЖКИ.
void Dlaydnt msecs);              // Функция задержки.
```

При использовании инфракрасного пульта дистанционного управления я обычно назначаю для кнопки 2 команду «Вперед», 4 и 6 соответствуют поворотам влево и вправо, 8 - «Назад» (реверс обоих двигателей), 5 - «Стоп» (остановка обоих двигателей).

С помощью кнопки, которая в обычных применениях управляет громкостью звучания телевизора, можно изменять скорость движения робота. Ширина импульса ШИМ-сигнала, генерируемого в нашей программе для управления обоими двигателями, по умолчанию установлена равной половине периода (50%). Если довести это значение до 100%, то робот начнет довольно быстро носиться по комнате. К сожалению, в таком режиме батарейки хватит ненадолго.

В программе предполагается, что для начала выполнения кода биологического уровня надо нажать кнопку включения питания на пульте дистанционного управления. Для отключения кода следует подать сигнал «Стоп» (кнопка 5) или какую-либо другую прямую команду. Обеспечение или отключение в произвольное время «самостоятельности» робота может потребоваться для того, чтобы сначала с помощью дистанционного управления подвести его к какому-либо месту, затем запустить для проверки биологический код, а если робот попадет в трудную ситуацию, снова взять управление на себя.

Следует заметить, что функция, выполнение которой наиболее трудно обеспечить, - это именно дистанционное управление. Если вы достаточно придирчиво испытаете робота в действии, то обнаружите, что при нажатии на кнопки пульта он может начать двигаться рывками и вести себя так, будто вместо одной команды вы подали сразу несколько.

Дело в том, что при работе двигателей присутствует большое число помех, оказывающих влияние на функционирование всех остальных подсистем, в том числе и приемника команд дистанционного управления. В следующей главе мы обсудим некоторые возможные пути решения подобных проблем.

Для проверки работы механических и световых датчиков, а также инфракрасного обнаружителя объектов надо сначала остановить оба двигателя, после чего подождать как минимум 60 мс, чтобы дать возможность управляющей программе опросить состояние всех датчиков. Это время необходимо для наиболее медлительных входных устройств - RC-цепочек с фоторезисторами. Для проверки датчиков обнаружителя объектов достаточно и 10 мс.

А еще лучше выключать двигатели на 6 мс каждый раз, пока продолжается очередной цикл работы с фоторезисторами и датчиком обнаружителя объектов.

Электродвигатели, входящие в комплект рассматриваемой модели, рассчитаны на напряжение 1,5 В, поэтому для работы с нашей 9-вольтовой батарейкой они имеют слишком маленькое сопротивление и в результате потребляют слишком большой ток. Вы можете убедиться в том, насколько велики потери напряжения на электронных ключах микросхемы L293D и как сильно она греется при работе.

Если использовать электродвигатели, рассчитанные на более высокое напряжение, то батарейка прослужит намного дольше, а нагрев микросхемы L293D резкоуменьшится.

4.34. ОДОМЕТРИЯ

Одометр - это устройство, связанное с колесом робота, которое по количеству сделанных оборотов позволяет измерять пройденное им расстояние. Эти данные можно использовать в программе для вычисления текущих координат робота (навигация) или для измерения скорости его движения (спидометр). В этом разделе мы рассмотрим основные методы одометрии, которые можно применять в радиолюбительских конструкциях мобильных автоматических устройств.

Чтобы не разочаровывать читателя, должен сразу признаться, что здесь вы не найдете конкретных примеров реализации подобных функций. Во-первых, они были бы слишком громоздки на фоне остальных представленных в этой книге простых примеров, а во-вторых, я всерьез сомневаюсь в возможности реализации описанных в данном разделе алгоритмов с помощью микроконтроллера PIC16F627.

Сам по себе принцип измерения расстояния по количеству оборотов двигателя или колеса довольно прост. Не слишком сложна и реализация этого принципа. Именно так ваш компьютер узнает о перемещении манипулятора типа «мышь».

Если вы разберете компьютерную мышку (заметьте, я сказал «*если*»!), то обнаружите, что вращающийся внутри нее шарик соединен с двумя колесиками, имеющими по краям множество отверстий. По одну сторону от каждого колесика расположены два оптических излучателя, а по другую напротив каждого излучателя имеется оптический детектор (рис. 4.85). Пара «излучатель-детектор» вместе с прорезями на ободе колесика образуют так называемый *оптический прерыватель.* Во время вращения колесика принимаемый детектором сигнал периодически прерывается, так как между детектором и излучателем прозрачные и непрозрачные участки сменяют друг друга. Число принятых импульсов соответствует количеству отверстий; таким образом, обычный цифровой счетчик, подключенный к прерывателю, позволяет измерить угол, на который повернулось колесико. Два излучателя и детектора нужны для того, чтобы определять направление вращения.

Если мы хотим использовать тот же принцип в наших автоматических устройствах, то должны соединить оптический прерыватель непосредственно с валом

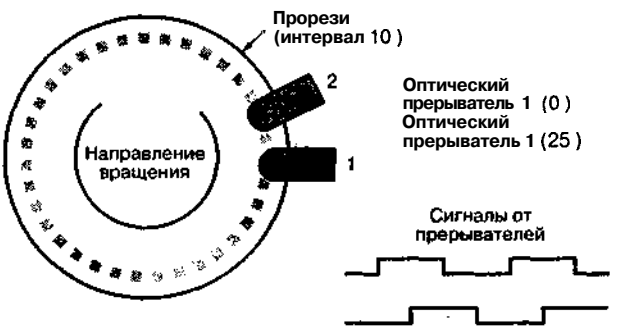

Рис. 4.85. *Оптический детектор* вращения колеса

двигателя. Прим этом в большинстве случаев достаточно одного излучателя и детектора, так как направление вращения известно управляющей программе.

Зная угол, на который повернулся вал в каждый момент времени, несложно вычислить скорость движения робота V: для этого можно воспользоваться формулой

$$V = 2\pi rN / (MT),$$

где r - радиус колеса робота; M - количество отверстий в колесике прерывателя; N - число импульсов, принятых счетчиком за время T.

Например, если колесо радиусом 3 см используется вместе с прерывателем, в котором имеется 30 отверстий, то 10 импульсов в секунду соответствуют скорости движения робота

$$V = 2 \times 3{,}14 \times 3 \text{ см} \times 10 / (30 \times 1 \text{ с}) = 6{,}28 \text{ см/с}.$$

Разумеется, это не слишком высокая скорость. Среднее значение обычно составляет около 15 см/с, а скорость, достигающая 1 м/с, по-настоящему впечатляет, особенно если робот при этом ни во что не врезается.

Используя один датчик и предполагая, что оба колеса вращаются с одинаковой скоростью, можно рассчитать длину пути, пройденного роботом за данный период времени.

Если ваше устройство основано на модели мыши фирмы Tamiya, вы вряд ли заставите его двигаться прямолинейно. Когда каждое колесо управляется отдельным двигателем и не принимаются специальные меры по точной подстройке скорости их вращения (а эти меры требуют использования двух прерывателей - по одному на каждое колесо), то разброс параметров двигателей и различные силы трения, действующие в левом и правом приводе, всегда приведут к тому, что робот будет стремиться повернуть в какую-либо сторону.

Впрочем, даже если скорости вращения левого и правого колеса немного отличаются друг от друга, разница эта настолько мала, что ею можно пренебречь при вычислении длины пройденного пути. Но, разумеется, этого же нельзя утверждать применительно к направлению движения: даже самый небольшой дисбаланс приводит к заметным искривлениям траектории.

Что касается методов стабилизации направления движения, основанных на регулировании скорости вращения левого и правого двигателей, то их рассмотрение и тем более программирование относятся к области высшего пилотажа в теории автоматического управления, а потому выходят за рамки данной книги (хотя это интересные темы и, быть может, я вернусь к ним в какой-нибудь следующей книге). Чтобы вы получили возможность оценить круг возникающих при этом проблем, приведем два основных момента, которые приходится учитывать.

Во-первых, надо знать инерцию двигателей (и самого робота) при изменении управляющих сигналов. Очевидно, что двигатели не могут сразу начать вращаться с той скоростью, которую мы для них хотим задать. Это очень важный вопрос, потому что два двигателя, которые по-разному ускоряют или замедляют свое вращение, даже хуже, чем два двигателя, вращающиеся с разной скоростью.

Во-вторых, надо рассчитать стабильный режим работы. А он зависит от того, насколько правильно двигатель «держит» заданную скорость.

Наиболее распространенный метод управления двигателями основан на использовании так называемого пропорционально-интегрально-дифференциального регулятора (ПИД). Входным сигналом регулятора обычно является отклонение измеряемой величины (в нашем случае это скорость вращения) от ее желаемого значения. Как следует из названия, ПИД-регулятор состоит из трех частей: пропорциональная часть вырабатывает управляющий сигнал, величина которого пропорциональна входному сигналу (то есть текущее отклонение скорости вращения двигателя от желаемого умножается на какой-то коэффициент); выходной сигнал интегральной части, как нетрудно догадаться, вычисляется с помощью интегрирования входной величины за некоторый уже прошедший промежуток времени (другими словами, на какой-то другой коэффициент умножается лишнее число оборотов, которые двигатель совершил за данное время); наконец, дифференциальная часть вырабатывает управляющий сигнал по величине скорости изменения входного сигнала (то есть на некий третий коэффициент умножается ошибка ускорения вращения двигателя). Все это проще выразить формулой

$$u = k_v v + k_n n + k_d a,$$

где v, n и a - это соответственно ошибка скорости, ошибка числа оборотов и ошибка ускорения вращения двигателя, а k_v, Ц и k_d - три коэффициента, значения которых необходимо подобрать так, чтобы достичь оптимальных характеристик проектируемой системы.

Существуют специальные методы (и программы) для расчета этих коэффициентов, но их можно подобрать и экспериментально, если предусмотреть какой-либо способ их оперативного изменения, например с помощью пульта дистанционного управления.

В первую очередь надо выбрать оптимальное значение коэффициента k_v пропорциональной части. Это лучше делать, испытывая работу двигателей и регулятора на неизменных режимах. Затем можно заняться подстройкой остальных двух коэффициентов, обращая внимание на моменты включения и выключения двигателей. Необходимо убедиться, что при смене режима работы не происходит слишком резких рывков или даже изменения направления вращения двигателей.

Вполне возможно реализовать с помощью микроконтроллера PIC16F627 все необходимые для ПИД-регулятора вычисления в промежутке между двумя вызовами обработчика прерываний от таймера TMRO, происходящих каждую миллисекунду. В принципе, эти вычисления лучше было оформить в виде биологического кода верхнего уровня, но на самом деле соответствующий уровень не совсем биологический. Он скорее механический, и его выполнение в главной программе может помешать работе действительно биологического кода. В некоторых случаях целесообразно рассмотреть вопрос о возможности использования отдельного микроконтроллера, который взял бы на себя функции ПИД-регулятора.

С помощью информации, получаемой от двух оптических прерывателей, можно с той или иной точностью вычислить текущие координаты робота, подобно тому как это делается в самолетах с помощью системы INS (Inertial Navigation System). В ней компьютеры с помощью нескольких гироскопов определяют, какое ускорение действует на воздушный лайнер, и тем самым вычисляют его текущие координаты.

Для обсуждения хотя бы самых основ этих методов нам потребуется высшая математика. Если вы не помните, что такое тригонометрия, вам стоит освежить свои знания, заглянув в учебник по математике для старших классов.

Как бы то ни было, сама по себе идея управления роботом на основе навигационной информации не так уж и сложна. Если известны начальная и конечная позиции робота, то можно вычислить направление движения и расстояние между ними.

Например, робот сейчас находится в точке плоскости (x, y) с координатами (x1, y1) = (0, 0), причем его ориентация совпадает с осью абсцисс, а мы хотим переместить его в точку с координатами (x2, y2) = (3, 4). Сначала надо рассчитать величину угла, на который должен повернуться робот. Тангенс этого угла определяется по формуле

$$\mathrm{tg}\alpha = \Delta y / \Delta x = (y2 - y1) / (x2 - x1).$$

Для нашего примера получаем

$$\mathrm{tg}a = Ay / Ax = 4 / 3 = 1{,}333...$$

Для вычисления самого угла приходится использовать функцию *арктангенс:*

$$a = \mathrm{arctg}\ Ay / \Delta x.$$

Для нашего случая arctg (4/3) = 53,13° = 0,927 рад.

Разумеется, вы помните теорему Пифагора, поэтому без труда вычислите, что после поворота на этот угол робот должен преодолеть

$$s = \sqrt{\Delta x^2 + \Delta y^2} = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

единиц расстояния, чтобы попасть в нужную точку.

Проблема состоит в том, что идеального робота не существует. Трудно добиться поворота на точно заданный угол, и нелегко правильно измерить пройденное расстояние, особенно если учесть, что робот перемещается не строго по прямой линии. На рис. 4.86 показана траектория, по которой движется робот, если разница в скорости вращения двух его колес остается строго постоянной. Эта траектория близка к окружности. Если робот будет двигаться в таком режиме достаточно долго, а угол поворота будет выдерживаться весьма точно, то, совершив полный оборот вокруг некоторой точки, робот вернется в то место, откуда начал свой путь.

Для определенности будем называть текущей позицией робота ту его точку, которая лежит на оси строго посередине между двумя его колесами. Рассмотрим для примера рис. 4.87. Робот медленно поворачивается направо; при этом его правое колесо проходит путь, длина которого чуть меньше той, что приходится

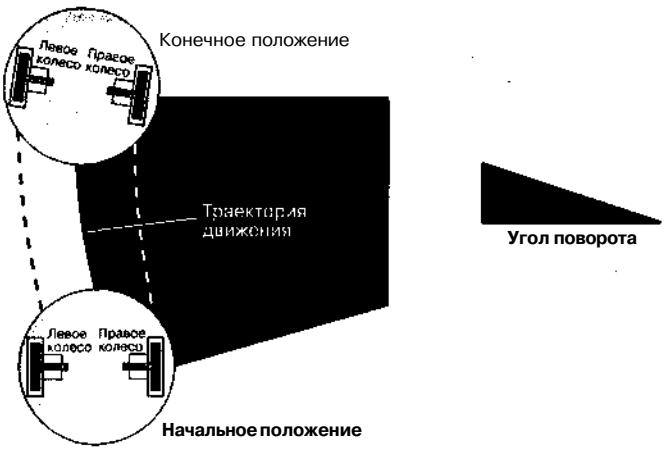

Рис. 4.86. Траектория *движения робота*

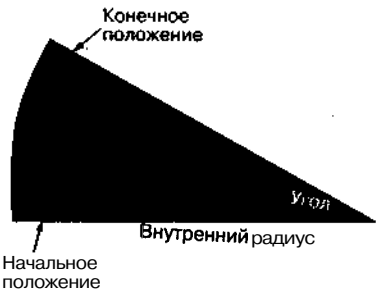

Рис. 4.87. Вычисление координат по данным ***одометрии***

преодолевать левому колесу. Обозначим буквой X длину пути, пройденную внутренним (при повороте направо - правым, при повороте налево - левым) колесом, а длину пути, пройденного за то же время вторым колесом, - буквой Y. Треугольник на рис. 4.87 при небольших углах поворота похож на прямоугольный. Тогда, согласно формулам тригонометрии, для внутреннего треугольника

$$X = R \sin\alpha; \quad X / R = \sin a.$$

Здесь R - радиус поворота (то есть радиус окружности, которую образует траектория движения робота); а - угол поворота.

Для наружного треугольника

$$Y = (R + Z) \sin\alpha; \quad Y / (R + Z) = \sin a,$$

где Z - расстояние между колесами.

Приравняв второе равенство, записанное для одного колеса, и второе равенство, записанное для другого колеса, получим

$$X / R = Y / (R + Z).$$

Отсюда можно найти радиус окружности, которую описывает внутреннее колесо:

$R = XZ / (Y - X)$.

При известных значениях X, Y и Z можно без труда определить R, что, в свою очередь, позволяет вычислить угол поворота:

$a = \arcsin(X / R)$.

Если, например, X = 2 см, Y = 2,1 см, а расстояние между колесами Z = 4 см, то

$R = XZ / (Y - X) = 2 \times 4 / (2{,}1 - 2) = 8 / 0{,}1 = 80$ см;

$a = \arcsin(X / R) = \arcsin(2 / 80) = \arcsin 0{,}025 = 0{,}025$ рад $= 1{,}43°$.

Заметим, что арксинус малого угла всегда примерно равен величине самого угла, выраженной в радианах.

Позицию робота мы договорились определять по координатам его средней точки, которая на Z/2 отстоит от внутреннего колеса. Поэтому радиус окружности, которую описывает эта точка, равен R + Z/2 = 82 см. Следовательно, конечные координаты робота

$y2 = (R + Z / 2) \sin\alpha = 82 \times \sin 0{,}025 = 2{,}05$ см;

$x2 = (R + Z / 2)(\cos\alpha - 1) \approx 0$.

Итак, конечные координаты робота в нашем примере (0; 2,05).

Глядя на эти формулы, вы можете решить, что теперь легко сумеете направить робота прямиком к любой точке комнаты. К сожалению, это не совсем так. Применяя формулы на практике, следует учитывать, что с увеличением расстояний, которые используются при вычислениях, растет и погрешность результата, поскольку перестают выполняться допущения, которые были сделаны при выводе формул. Кроме того, в программе придется использовать вычисления с плавающей точкой, причем все выкладки надо делать быстро, иначе вы не будете успевать остановить робота в нужной точке.

А если еще учесть неточность определения начальной позиции и весьма вероятные проскальзывания колес, особенно заметные, когда робот будет преодолевать неровные участки своего пути, наши выкладки окажутся еще менее надежными, а задача навигации на основе одометрии еще более усложнится.

4.35. РАДИОУПРАВЛЯЕМЫЙ СЕРВОПРИВОД

Для передачи команд дистанционного управления различными игрушками часто используются радиопередатчик и сервопривод с радиоприемником. Поэтому при конструировании робота мы можем использовать серводвигатели, установленные на радиоуправляемых моделях самолетов, автомобилей или лодок. Радиоуправляемый сервопривод (R/C servo) обычно имеет на выходе колесо, которое может

поворачиваться на угол в диапазоне от 0° до 90° (некоторые устройства - от 0° до 180°). Усилие (вращающий момент), развиваемое серводвигателем, обычно невелико, хотя существуют специальные устройства, в которых оно может достигать больших значений. Напряжение питания в большинстве случаев составляет +5 В. Никаких сложных подключений сервоприводы не требуют - чаще всего используются всего три внешних вывода: напряжение питания, общий вывод и входной сигнал управления.

Серводвигатель является аналоговым устройством, управляемым с помощью широтно-модулированного сигнала постоянной амплитуды. Обычно это импульсы длительностью от 1 до 2 мс, подаваемые каждые 20 мс (рис. 4.88).

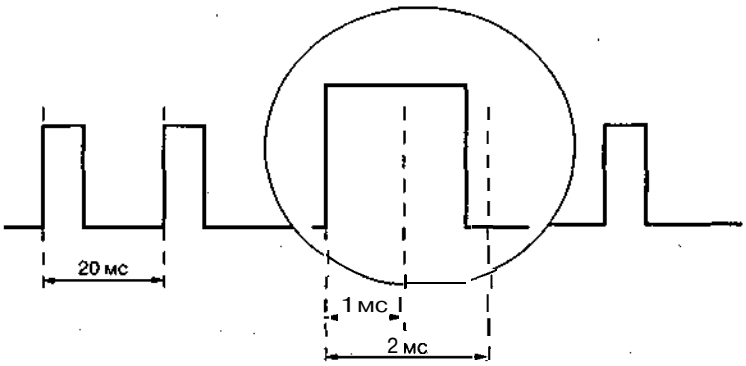

Рис. 4.88. *Временна́я диаграмма* ***ШИМ-сигнала*** *управления сервоприводом*

Бо́льшая длительность импульсов соответствует большему углу поворота серводвигателя. Для формирования управляющего ШИМ-сигнала можно использовать следующий код, который понадобится поместить в процедуру обработки прерываний от таймера:

```
Interrupt() {                    // Процедура обработки прерываний.
int i = 0;
 BitOutput( Servo, 1 )           // Выходной сигнал.
 for ( i = 0, i < (1 msec + ServoDlay); i++ );
 BitOutput( Servo, 2 );
 for ( ; i < 2 msec; i++ );      // Задержка на 2 мс.
}                                // Конец обработчика прерываний.
```

Эту программу несложно модифицировать для работы с несколькими серводвигателями (добавив новые выходные переменные Servo и счетчики для формирования задержек ServoDlay). Основное преимущество продемонстрированного подхода заключается в том, что значение переменной ServoDlay можно изменять, не оказывая никакого влияния на выполнение обработчика прерываний.

Приведенный код прекрасно работает с автономным радиоуправляемым устройством, но показывает не слишком хорошие результаты, если мы захотим

использовать его в качестве одного из многих интерфейсов, реализованных в виде процедуры обработки прерываний от таймера, срабатывающего каждую миллисекунду. В этом случае миллисекундная задержка оказывается слишком большой. Надо раза в четыре уменьшить период срабатываний таймера, доведя его значение хотя бы до 256 мкс. Обычно радиоуправляемые сервоприводы могут распознавать около пяти команд; возможные варианты и соответствующие длительности импульсов приведены в табл. 4.19.

Таблица 4.19. Команды радиоуправляемого сервопривода

Номер команды	*Задержка во времени, мс*	*Команда*
1	*1,00*	*Реверс на полной скорости*
2	*1,25*	*Реверс на половинной скорости*
3	*1,50*	*Стоп*
4	*1,75*	*Вперед на половинной скорости*
5	*2,00*	*Вперед на полной скорости*

Серводвигатель можно заставить вращаться постоянно, заменив в его схеме потенциометр, который входит в цепь обратной связи, управляющей положением привода, подстроечным резистором, а также убрав пластмассовый выступ, который ограничивает угол поворота привода, не давая ему повернуться на угол, больший 90°. Положение движка подстроечного резистора следует подобрать экспериментально. Мы еще вернемся к этому вопросу в следующем разделе.

4.36. ПРОСТОЕ РАДИОУПРАВЛЯЕМОЕ УСТРОЙСТВО

Здесь мы рассмотрим простую радиоуправляемую конструкцию на основе стандартного серводвигателя. Нам потребуется два таких двигателя, каждый из которых будет управлять соответствующим колесом робота. Радиоуправляемые серводвигатели не слишком дороги и достаточно неприхотливы с точки зрения как требуемых для их подключения аппаратных средств, так и управляющего программного обеспечения. Но их придется немного модифицировать, чтобы они могли крутиться постоянно.

На постройку описанной здесь конструкции я затратил не более получаса, включая то время, которое потребовалось на модификацию обоих серводвигателей.

Я использовал лист толстой фанеры размером 8x3,75 дюймов (203x95 мм). Перед размещением элементов поверхность фанеры желательно покрыть грунтовкой (для этого удобно использовать пульверизатор с краской), чтобы элементы приклеились к основанию. В крайнем случае можно обмотать уже собранное устройство изоляционной лентой.

Внешний вид приемника показан на рис. 4.89. Принципиальная схема приведена на рис. 4.90, а использованные элементы описаны в табл. 4.20.

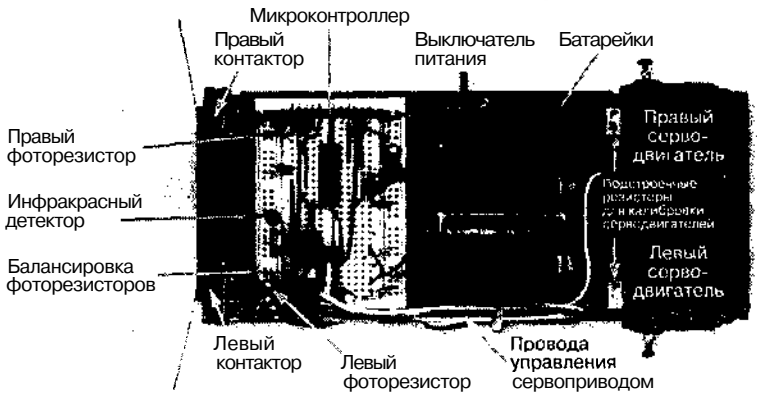

Рис. 4,89. *Приемник команд радиоуправления*

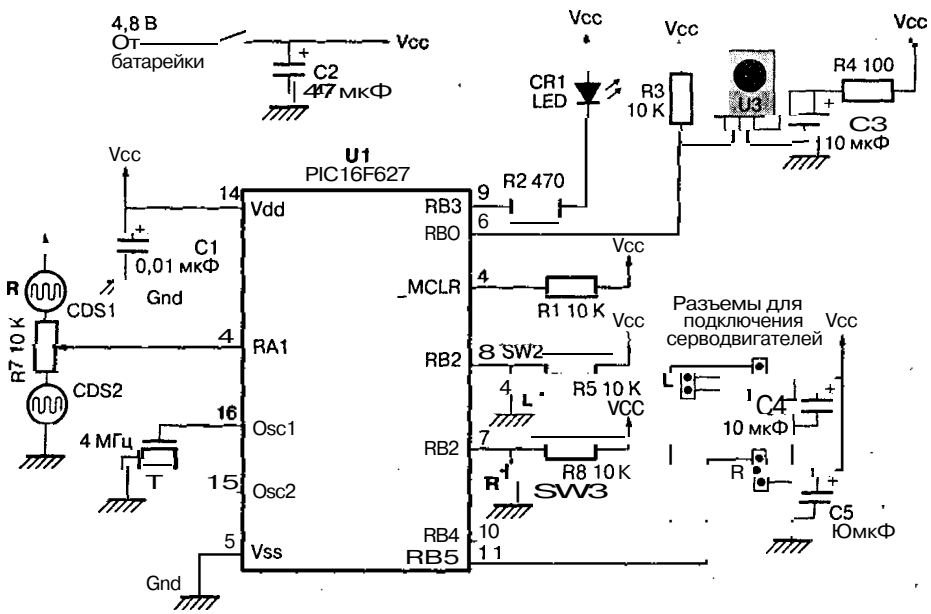

Рис. *4.90. Принципиальная схема приемника команд радиоуправления*

Таблица 4.20. Перечень использованных элементов

Обозначение	*Элемент*	*Примечание*
U1	*PIC16F627*	*Микроконтроллер*
U3	*Инфракрасный детектор*	*PIC-1018SCL фирмы Waitronyили другой совместимый /три вывода)*
CR1	*Светодиод*	*Любой видимого диапазона*
R1, K3, R5, R6, R8	*10 кОм; 0,25 Вт*	

Таблица 4.20. Перечень использованных элементов (окончание)

Обозначение	Элемент	Примечание
R2	470 Ом; 0,25 Вт	
R4	100 Ом, 0,25 Вт	
R7	10 кОм (подстроечный)	
CDS1, CDS2	Фоторезистор, 10 кОм при нулевом освещении	При освещении должен уменьшать свое сопротивление
C1	0,1 мкФ	Конденсатор любого типа
C2	47 мкФ; 16 В	Оксидный конденсатор
C3 – C5	10 мкФ; 16 В	Оксидный конденсатор
SW1	Выключатель любого типа	Выключатель питания
SW2, SW3	Микропереключатель	Для детектора столкновений
XTAL1	Керамический резонатор на 4 МГц, со встроенными конденсаторами	Для генератора тактовых импульсов микроконтроллера
	Материалы	Два модифицированных серводвигателя, два колесика от модели самолета, фанера, макетная плата, монтажные провода, четыре батарейки 1,2 В типа AA

Сборку устройства лучше производить в следующем порядке:

1. Модифицируйте оба сервопривода для постоянного вращения двигателей, как описано выше.
2. Подсоедините серводвигатели к колесам робота.
3. С помощью эпоксидной смолы закрепите на днище робота в передней его части светодиод для индикации срабатывания детектора столкновений (одновременно он может играть роль третьей точки опоры для всей конструкции).
4. Прикрепите к корпусу серводвигатели, батареи питания и контактные усики обоих детекторов столкновений.
5. Выполните необходимые соединения между компонентами конструкции.
6. Запрограммируйте микроконтроллер и проверьте, как работает дистанционное управление.

Все шесть этапов достаточно просты, но надо соблюдать внимательность, чтобы ничего не упустить.

Первый этап (модификация сервопривода) не должен вызвать затруднений, но прежде чем приобретать ту или иную конкретную модель, следует убедиться, что вы сможете найти для нее нужную документацию (возможно, в сети Internet), где будет подробно объяснено, как модифицировать привод для того, чтобы двигатели могли совершать полный оборот вокруг своей оси. В некоторых случаях такая операция оказывается довольно затруднительной. Лучше всего приобрести сервопривод попроще и подешевле.

Как уже говорилось, надо будет удалить встроенный потенциометр (или два резистора, которые его заменяют) и поставить вместо него подстроечный резистор,

который можно просто приклеить к корпусу двигателя. Резистор следует расположить так, чтобы вы могли изменить положение движка, не меняя положения робота, пока он стоит на месте. Разумеется, эту регулировку можно было бы реализовать программно, но управляющая программа от этого неоправданно усложнилась бы. Гораздо проще подстроить сервопривод с помощью отвертки.

Используемые в конструкции колеса должны подходить для серводвигателей. Самое простое — взять колесики от модели самолета (диаметром 2 или 3 дюйма, то есть 5-7,5 см) и просверлить в центре каждого из них небольшое отверстие, в которое мог бы пройти болт, - с его помощью колесики будут крепиться к серводвигателям. Затем надо наклеить сверху (на ступицу) декоративную крышку, продающуюся в комплекте с колесиком, используя, к примеру, эпоксидную смолу (рис. 4.91).

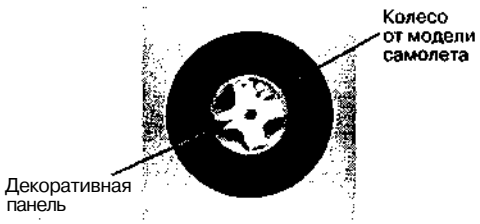

Рис. 4.91. Одно из **двух** колес для ***радиоуправляемого*** робота

После того как все части окажутся скреплены друг с другом, настанет время поместить их в корпус робота. Для крепления батареек, сервопривода и электронного блока управления можно обмотать их липкой лентой.

Микропереключатели детектора столкновений также можно приклеить к корпусу робота.

Работая с клеем или эпоксидной смолой, не забывайте следовать инструкциям завода-изготовителя и хорошо проветривайте помещение. Выделяемые при этом пары не слишком ядовиты, но могут вызывать неприятные ощущения.

На рис. 4.89 видно, что сбоку от батареек с одной стороны мы оставили место для выключателя питания, а с другой - для проводов, идущих к серводвигателям. Выключатель можно приклеить, а для крепления проводки следует просверлить отверстия в фанерном корпусе.

Если вы не новичок и умеете планировать свои действия на несколько шагов вперед, то можете приклеить оба контактных детектора, светодиод, переключатель, подстрочные резисторы серводвигателей и декоративные крышки колесиков за один раз, но если это первая ваша конструкция, лучше не пытаться все сделать единым махом.

В отличие от прошлых проектов, здесь не применяется стабилизатор напряжения. Как видно на рис. 4.89, в качестве источника питания используются четыре батарейки типа АА (они могут быть никель-кадмиевые или металлгидридные). Каждая из них дает напряжение 1,2 В, поэтому при последовательном соединении батареек получается источник питания напряжением 4,8 В, чего вполне достаточно для микроконтроллера и серводвигателей.

Подсоединение остальных элементов не должно вызвать у вас затруднений. Последовательно со светодиодом надо включить резистор 470 Ом. Светодиод будет сигнализировать о срабатывании левого или правого детектора столкновений.

Такая обратная связь очень пригодится при настройке контактных усиков. Кроме того, надо подключить «подтягивающие» резисторы сопротивлением 10 кОм к микропереключателем детекторов столкновений.

Хотя мы взяли сервопривод от радиоуправляемой модели, радиоприемник и передатчик нам не потребуются. В этой конструкции опять же использованы инфракрасный пульт дистанционного управления и инфракрасный детектор фирмы Sony. Все это работает точно так же, как и в предыдущем проекте, выполненном на основе модели мыши фирмы Tamiya.

А датчики света для простоты взяты из нашей первой конструкции такого типа. Если вы были внимательны, то уже заметили подстроечный резистор (элемент R7 на рис. 4.90), который предназначен для балансировки плеч делителя напряжения, образованного двумя фоторезисторами.

Когда все будет готово, можно приступить к программированию микроконтроллера PIC16F627. Исходный текст управляющей программы содержится в файле Code\Servo\servo2.c:

```
ttinclude<pic.h>

// 28 апреля 2002 - добавлено дистанционное управление.
// 23 февраля 2002 - разработано Майком Предко.
//
// Используются прерывания от таймера TMR0,
// вырабатываемые каждые 512 мкс.
//
// Замечания по аппаратным средствам:
// Микроконтроллер PIC16F627 работает на частоте 4 МГц.
// Используется внешний тактовый генератор.
//
// Подключение выводов микроконтроллера:
// RA1 - вход компаратора 1 (сигнал с делителя напряжения
//  на фоторезисторах);
// RA2 - контрольный выход внутреннего источника опорного напряжения
// для компараторов (половина напряжения питания);

// RB0 - инфракрасный детектор для приема команд дистанционного
// управления фирмы Sony (прерывания по положительному перепаду);
static volatile bit IRDetect            @ (unsigned)&PORTB*8+0;

// RB1 - правый контактор;
static volatile bit RightWhiskerPin     @ (unsigned)&PORTB*8+1;

// RB2 - левый контактор;
static volatile bit LeftWhiskerPin      @ (unsigned)&PORTB*8+2;

// RB3 - выход индикаторного светодиода;
static volatile bit LED                 @ (unsigned)&PORTB*8+3;
static volatile bit LEDTRIS             @ (unsigned)&TRISB*8+3;

// RB4 - левый серводвигатель;
static volatile bit leftServo           @ (unsigned)&PORTB*8+4;
static volatile bit leftServoTRIS       @ (unsigned)&TRISB*8+4;

// RB5 - правый серводвигатель.
static volatile bit rightServo          @ (unsigned)&PORTB*8+5;
```

```
static volatile bit rightServoTRIS  @ (unsigned)&TRISB*8+5;

// Глобальные переменные и константы:
unsigned int  RTC = 0;          // Счетчик реального времени.
unsigned char RTCOlay = 2;      // Задержка 1 мс (период срабатывания
                                // таймера равен 512 мс).

// Константы для серводвигателей:
#define LeftForward    38       // Для левого двигателя.
#define LeftStop       37
#define LeftReverse    36
«define RightForward   36       // Для правого двигателя.
«define RightStop      37
#define RightReverse   38
char ServoDlay = 1;             // Выход управления сервоприводом
                                // (каждые 20 мс).

char leftSpeed = LeftStop;      // Скорость вращения левого двигателя.
char rightSpeed = RightStop;    // Скорость вращения правого двигателя.

// Константы для декодера команд дистанционного управления:
#define remote0          0x06EF       // Цифры 0-9.
#define remote1          0xOFEF
#define remote2          0x07EF
#define remote3          0x0BEF
#define remote4          0x03EF
#define remote5          0xODEF
#define remote6          0x05EF
«define remote7          0x09EF
«define remote8          0x01EF
#define remote9          0xOEEF
#define remoteVolUp      0xOB6F       // Кнопка "Громче".
#define remoteVolDown    0x036F       // Кнопка "Тише".
«define remoteChanUp     0xOF6F       // Кнопка "Канал +1".
#define remoteChanDown   0x076F       // Кнопка "Канал -1".
#define remotePrevChan   0x022F       // Кнопка "Предыдущий канал".
«define remoteMute       0xOD6F       // Кнопка "Приглушить звук".
#define remotePower      0x056F       // Кнопка "Питание".

unsigned int  DataIn;           // Входные данные детектора команд управления.
unsigned char DataInCount = 0;  // Число символов для приема.
unsigned int  DataTime = 0;
unsigned int  CurrentRTC = 0;
int  OpnCount = 1;              // Если двигатели не работают,
                                //" можно проверить компаратор.

volatile char LeftWhisker = 0;  // Счетчик для левого контактора.
volatile char RightWhisker = 0; // Счетчик для правого контактора.

char ExecuteFlag = 0;    // Флаг для выполнения биологического кода.

// Слово конфигурации:
#if defined(_16F627)
   #warning PIC16F627 with external XT oscillator selected
   _CONFIG(Ox03F21);    // Для МК PIC16F627:
                        // внешний тактовый генератор XT,
                        // RA6/RA7 используются для ввода-вывода,
                        // внешний сигнал сброса,
                        // таймер PWRT включен,
```

```
                        // сторожевой таймер выключен,
                        // защита кода отключена,
                        // детектор BODEN включен.
#else
    (terror Unsupported PICmicro MCU selected
#endif

// Обработчик прерываний:
void interrupt tmr0_int(void)
{

    if (T0IF) {          // Обработчик прерываний от таймера TMR0.

        T0IF = 0;        // Сбросить флаг прерываний от таймера TMR0.

        if (--ServoDlay == 0) {

            ServoDlay = 40;        // Задержка 20 мс.

            leftServo = 1;         // Высокий уровень сигнала
            rightServo = 1;        // управления для обоих
                                   // серводвигателей.

        } else {

            if (leftSpeed >= ServoDlay)
                leftServo = 0; // Закончить подачу импульса
                               // для левого серводвигателя.
            if (rightSpeed >= ServoDlay)
                rightServo = 0;// Закончить подачу импульса
                               // для правого серводвигателя.
        }

        // Декодирование команд дистанционного управления
        // (если оно установлено в системе).

        if (DataTime) {
            if (DataInCount == 0) {      // Новый пакет.
                DataInCount = 12;        // Ждать 12 бит.
                DataIn = 0;
            } else {                     // Очередной бит.
                DataIn = DataIn << 1;
                if ((DataTime > 450) && (DataTime < 750)) {
                                         // Получена 1.
                    DataIn++;
                    if (--DataInCount == 0) {
                        OpnCount = 400; // Конец.
                        leftSpeed = LeftForward;
                        rightSpeed = RightForward;
                        if (DataIn == remote2)
                            ;
                        else if (DataIn == remote4)
                            leftSpeed = LeftReverse;
                        else if (DataIn == remote6)
                            rightSpeed = RightReverse;
                        else if (DataIn == remote8) {
                            leftSpeed = LeftReverse;
                            rightSpeed = RightReverse;
                        } else
```

```
                                    if (DataIn == remotePower) {
                                      ExecuteFlag = 1;
                                      leftSpeed = LeftStop;
                                      rightSpeed = RightStop;
                                      OpnCount = 1; // Конец.
                                    } else { // Стоп или ничего.
                                      ExecuteFlag = 0;
                                      leftSpeed = LeftStop;
                                      rightSpeed = RightStop;
                                      OpnCount = 1; // Конец.
                                    }
                                }
                        } else if ((DataTime > 750) &&
                                    (DataTime < 1000)) {
                                      // Получен 0.
                                if (-DataInCount == 0) {
                                    OpnCount = 400; // Конец.
                                    leftSpeed = LeftForward;
                                    rightSpeed = RightForward;
                                    if (Dataln == remote2)
                                      ;
                                    else if (Dataln == remote4)
                                      leftSpeed = LeftReverse;
                                    else if (Dataln == remote6)
                                      rightSpeed = RightReverse;
                                    else if (Dataln == remote8) {
                                      leftSpeed = LeftReverse;
                                      rightSpeed = RightReverse;
                                    } else
                                      if (Dataln == remotePower) {
                                        ExecuteFlag = 1;
                                        leftSpeed = LeftStop;
                                        rightSpeed = RightStop;
                                        OpnCount = 1; // Конец.
                                    } else { // Стоп или ничего.
                                        ExecuteFlag = 0;
                                        leftSpeed = LeftStop;
                                        rightSpeed = RightStop;
                                        OpnCount = 1; // Конец.
                                    }
                                }
                        } else                      // Ошибка.
                                DataInCount = 0;
                }
                DataTime = 0;
}

CurrentRTC += 0x0100; // Задержка для приема одного бита.

// Проверка на пропущенный импульс:
if ((DataInCount) && (CurrentRTC > 0x01200))
        DataInCount = 0;
                // Сброс и ожидание следующего пакета.

// Код для обработки каждого второго прерывания
// от таймера (работает через 1 мс):
if (--RTCDlay == 0) {
```

```
                RTCDlay = 2;   // Следующий раз - через один вызов.
                RTC++; // Инкремент счетчика реального времени.

                // Код для левого контактора:
                if (!LeftWhiskerPin) // Левый контактор сработал.
                        if (LeftWhisker < 20)          // Противодребезговая
                                LeftWhisker++;         // задержка.
                        else;                          // Нажат уже 20 мс.
                else                                   // Контактор разомкнут.
                        if (LeftWhisker != 0)          // Противодребезговая
                                LeftWhisker--;         // задержка.
                        else;                          // Разомкнут уже 20 мс.

                // Код для правого контактора:
                if (!RightWhiskerPin) // Правый контактор сработал.
                        if (RightWhisker < 20)         // Противодребезговая
                                RightWhisker++;        // задержка.
                        else;                          // Нажат уже 20 мс.
                else                                   // Контактор разомкнут.
                        if (RightWhisker != 0)         // Противодребезговая
                                RightWhisker--;                // задержка.
                        else;                          // Разомкнут уже 20 мс.

                // Работа с фоторезисторами и светодиодом:
                if ((!ExecuteFlag) && (--OpnCount == 0)) {
                        OpnCount = 1;
                        leftSpeed = LeftStop;
                        rightSpeed = RightStop;
                        if (C2OUT)
                                LED = 0;       // Светодиод включить.
                        else
                                LED = 1;       // Светодиод выключить.
                }

        }

    }  // Конец обработки прерываний от таймера.

    // Здесь можно разместить другие обработчики прерываний.

    if (INTF) {                 // Обработка прерываний по изменении
                                // сигнала на входе RB0/INT.
            DataTime = CurrentRTC + TMR0;// Время приема текущего бита.
            CurrentRTC = 0x0FFFF - THR0; // Время для следующего бита.
            INTF = 0;                   // Сброс флага прерывания.
    }  // Конец обработки прерываний по изменении входного сигнала.

}  // Конец обработчика прерываний.

// Служебные подпрограммы:
void  Dlay(int  msecs)              // Задержка на msecs миллисекунд.
{

int valueDlay;

    valueDlay = RTC + msecs + 1; // Время окончания задержки.
    while (valueDlay != RTC);     // Ждать.

}  // Конец функции Dlay.

void LEDOutput(int state )        // Управление состоянием светодиода.
```

```
{
    if (state)
           LED = 0;              // Светодиод включить.
    else
           LED = 1;              // Светодиод выключить.
} //  Конец функции LEDOutput.
int GetLeftLight()               // Опрос состояния фоторезисторов.
{
    if (C2OUT)
           return 0x080;         // Источник света слева.
    else
           return 0xOFF;
} //  Конец функции GetLeftLight.
int GetRightLight()
{
    if (!C2OUT)
           return 0x080;         // Источник света справа.
    else
           return 0xOFF;
} // Конец функции GetRightLight.
int GetLeftWhisker()             // Опрос состояния левого контактора.
{
    if (LeftWhisker == 20)
           return 1;             // Левый контактор сработал.
    else
           return 0;
} // Конец функции GetLeftWhisker.
int GetRightWhisker()            // Опрос состояния правого контактора.
{
    if (RightWhisker == 20)
           return 1;             // Правый контактор сработал.
    else
           return 0;
} // Конец функции GetRightWhisker.
void LeftMotor(int Movement)     // Управление левым двигателем.
{
    leftSpeed = Movement;
} //  Конец функции LeftMotor.
void RightMotor(int Movement)    // Управление правым двигателем.
{
    rightSpeed = Movement;
} // Конец функции rightMotor.
// Главная программа;
```

```
void main(void)
{
    OPTION = 0xOD0;        // Предделитель работает с таймером TMR0,
                           // коэффициент деления 1:2.
    TMR0 = 0;              // Начальный сброс таймера TMR0.
    TOIE = 1;             // Разрешение прерываний от таймера TMR0.
    GIE = 1;               // Разрешение обработки прерываний.
    // Здесь можно разместить код верхнего уровня.

    INTEDG = 1;           // Прерывания вырабатываются по положительному
                           // перепаду сигнала на выходе RB0/INT.
    INTF = 0;              // Сброс флага прерываний.
    INTE = 1;              // Разрешение прерываний по изменении
                           // сигнала на входе RB0/INT.

    CMCON = 0x002;         // Разрешение работы компараторов:
                           // выходы компараторов не инвертируются;
                           // бит CIS = 0;
                           // режим 2 (внутренний источник
                           // опорного напряжения).
    TRISA = 0x007;         // Входными являются только RA0 и RA1.
    VRCON = 0xOEC;         // Разрешение работы внутреннего источника
                           // опорного напряжения:
                           // для контроля его напряжение выводится
                           // на выход RA2,
                           // верхний поддиапазон,
                           // установочное значение равно 12.

    LED = 1;               // Разрешить работу выхода, к которому
    LEDTRIS = 0;           // подключен светодиод.

    leftServo = rightServo = 0;   // Начальные значения
                                  // для сигналов управления двигателями.
    leftServoTRIS = rightServoTRIS = 0;

    // Биологический код:
    while (1 == 1) {              // Бесконечный цикл.

        if (ExecuteFlag) {

        // Здесь надо разместить код, определяющий поведение робота.

        } else {

              // Код для инициализации состояния робота.

        }

    }  // Конец оператора while.

}  // Конец главной программы.
```

Теперь необходимо провести две операции по калибровке систем. Во-первых, следует проградуировать подстроечные резисторы серводвигателей. Это не должно вызвать у вас затруднений, особенно если вы уже испытали в работе предыдущую конструкцию. Надо просто включить питание робота и с помощью отвертки медленно изменять положение движка резистора, пока двигатель не остановится.

Во-вторых, необходимо настроить подстроечный резистор R7, который предназначен для балансировки плеч делителя напряжения на фоторезисторах. Ваша

задача - добиться такого положения его движка, при котором индикаторный светодиод мигает, если робот незначительно отклоняется от направления на источник света. В идеале все настройки надо проводить, создав те же условия (используя то же помещение, ту же освещенность), при которых планируется испытывать робота.

После всех проверок вы, возможно, захотите усовершенствовать конструкцию робота, добавив к ней более сложные интерфейсные модули, которые уже описывались в предыдущих проектах.

ГЛАВА 5
ВДОХНИТЕ В РОБОТА ЖИЗНЬ

В предыдущей главе мы рассмотрели достаточно много примеров реализации функций двух нижних (механического и электронного) уровней в виде обработчика прерываний от таймера, формируемых каждую миллисекунду. Что касается функций верхнего (биологического) уровня, то представленный код выполнял пока лишь только тестовые функции, позволяя проверить работу интерфейсов нижнего уровня. Одновременно мы обсудили методы совмещения нескольких интерфейсов в одной процедуре обработки прерываний.

В этой главе, наконец, будет показано, как реализуется выполнение функций верхнего уровня. Другими словами, мы обсудим *методы принятия решений,* которые и определяют функциональность робота. Для этого применяются самые различные подходы, и здесь будут рассмотрены только самые популярные из них. Большинство подходов базируется на методе *нелинейного программирования, с* помощью которого удается реализовывать достаточно сложные функции и который в большинстве случаев не требует больших объемов памяти или трудоемких вычислений. Именно эти достоинства становятся определяющими, когда речь идет о программировании микроконтроллеров, поскольку, в отличие от персональных компьютеров или, тем более, больших ЭВМ, здесь очень жестко задаются ограничения на время реакции программы и объем аппаратных ресурсов.

Еще одно преимущество описываемого здесь подхода состоит в том, что он позволяет свести достаточно сложный процесс проектирования программного обеспечения к более простой задаче, которая решается оптимальным подбором нескольких параметров. Это не означает, что мы больше не будем программировать (будем, насколько вы сможете выдержать), но после написания программы, которое теперь уже не осуществляется столь хаотично и неформально, понадобится выполнить настройку значений нескольких параметров, пока мы не добьемся нужного поведения робота. При этом настройка параметров в идеале не должна потребовать внесения изменений в текст программы.

При создании биологического кода необходимо помнить, что поведение робота кажется осмысленным только в том случае, если оно является *детерминированным.* Это значит, что любая реакция управляющей программы на то или иное входное воздействие не должна быть случайной. Только строго запланированные и предсказуемые (то есть повторяющиеся при одних и тех же условиях) реакции на изменения внешней обстановки можно считать проявлениями искусственного интеллекта.

5.1. ОПЕРАЦИОННЫЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Операционной системой реального времени (ОСРВ, Real Time Operating System - RTOS) называется программная среда, которая обеспечивает поддержку выполнения *прикладных программ* (tasks - задач), выступая, таким образом, в качестве своеобразного интерфейса между программным обеспечением и аппаратными средствами. ОСРВ предоставляет прикладным программам необходимый набор стандартных *сервисов* (интерфейсов, функций), позволяющий программам наиболее простым способом взаимодействовать друг с другом и с аппаратурой. Благодаря тому что часто используемые функции реализуются с помощью ОСРВ, программы становятся менее объемными и сложными, а сам процесс программирования - менее трудоемким. От обычных *операционных систем* (ОС) операционные системы реального времени отличаются тем, что обеспечивают поддержку выполнения задач, в которых большую роль играет соблюдение заданного времени реакции на то или иное внешнее событие.

Многие микроконтроллеры имеют встроенную поддержку какой-либо операционной системы. К сожалению, этого нельзя сказать о микроконтроллерах PICmicro среднего семейства. Поэтому в следующем разделе при обсуждении методов работы с ОСРВ мы будем вынуждены иметь дело больше с теоретическими концепциями, чем с конкретными примерами.

Лучшая формулировка, которую я могу предложить для определения ОСРВ, такова: *операционная система реального времени - это фоновая программа, которая управляет выполнением множества прикладных задач и облегчает их взаимодействие.*

Напомним, что операционная система, способная поддерживать выполнение сразу нескольких прикладных задач, называется *многозадачной.* При этом для выполнения каждой задачи обычно отводится малый промежуток (квант) процессорного времени, по истечении которого управление передается следующей задаче; данный процесс повторяется по кругу, создавая иллюзию одновременного выполнения многих задач.

Если вы знакомы с системным программным обеспечением персональных компьютеров, то, возможно, слышали что-то о процессах и потоках, а также различных методах планирования задач (в том числе и реального времени), используемых в современных операционных системах. Здесь мы не будем вдаваться во все эти тонкости, так как большинство из них оказывается совершенно не нужно, когда речь заходит о программировании микроконтроллеров. Даже различия между ОС и ОСРВ не будут нас волновать, так как ядро (kernel) тех и других устроено одинаково. Разумеется, при более пристальном рассмотрении могут быть выделены некоторые отличия, в частности касающиеся начальной загрузки системы. При запуске ОС персональных компьютеров вместе с ядром загружаются: 1) *базовая система ввода-вывода* (Basic Input/Output System - BIOS), осуществляющая поддержку ввода информации с клавиатуры и вывода ее на экран, а также работу с файлами на диске; 2) интерпретатор команд, которые вводит пользователь.

В ОСРВ вместе с ядром загружаются наиболее приоритетные задачи, выполнение которых жизненно необходимо для поддержания работоспособности всей системы.

Важная особенность ОСРВ, используемых в робототехнике, заключается в том, что они должны требовать минимально возможных аппаратных ресурсов, то есть экономно расходовать память программ и данных, а также процессорное время. Объем памяти микроконтроллеров сильно ограничен, и весьма накладно было бы хранить в ней, как это обычно бывает в ОС, код для поддержки всего набора возможных функций (в том числе и сетевых), даже если многие из них *в* данный момент не требуются.

Все задачи в ОСРВ (как и в ОС) имеют свой приоритет. В первую очередь процессорное время отводится самым приоритетным задачам. Планировщик использует так называемые *очереди,* в которых задачи ожидают своего выполнения. Обычно для задач, имеющих один и тот же уровень приоритета, используется отдельная очередь.

В каждый момент времени может выполняться только одна задача. Она будет приостановлена, когда истечет отведенный для нее квант времени, или произойдет какое-либо особое внешнее событие, или эта задача завершится сама по себе либо перейдет в режим ожидания некоторого события (например, запроса на прерывание от внешнего устройства). После этого из наиболее приоритетной очереди будет выбрана для выполнения следующая задача, и так будет повторяться снова и снова. Длительность кванта времени должна быть не слишком велика, чтобы все задачи могли получать процессорное время достаточно часто и тем самым имели возможность своевременно реагировать на все внешние события. В то же время длительность кванта не должна быть слишком маленькой, иначе большую часть времени микроконтроллер или микропроцессор будет занят переключением задач, а не их выполнением.

В каждом конкретном случае при разработке того или иного автоматического устройства в процессе его. испытания может потребоваться индивидуальная настройка длительности кванта времени. Все зависит от тех функций, которые должна выполнять система.

Уровни приоритета назначаются задачам в зависимости от того, как быстро они должны уметь реагировать на запрос, поступивший от внешнего устройства. Вас может удивить тот факт, что вроде бы наиболее важный по значимости код верхнего (биологического) уровня обычно имеет самый низкий уровень приоритета, в то время как менее важным механическим и электронным интерфейсам, обслуживающим различные периферийные устройства, зачастую присваивается наивысший уровень. Все дело в том, что робот должен быстро реагировать на запросы, поступающие от внешних устройств. Если какое-либо внешнее устройство формирует запрос на обслуживание, необходимо как можно быстрее выполнить его. Если никаких запросов нет, то может выполняться наименее приоритетный биологический код, определяющий поведение робота. Для успешного выполнения биологического кода должны вовремя выполняться все обеспечивающие его работу функции, а это в первую очередь функции нижних уровней.

При распределении приоритетов важно понимать, как протекают все процессы в системе и как они взаимодействуют. Очень просто попасть в ловушку, когда какая-либо важная задача ожидает результата выполнения менее приоритетной, которая, в свою очередь, не может завершиться из-за того, что ей мешает эта более приоритетная задача. Также возможны ситуации, когда какая-либо задача с низким приоритетом (но в данный момент необходимая) вообще не может начать свое выполнение, потому что ей все время мешает более приоритетная задача.

Очень часто в системах реального времени некоторые задачи не могут начать свое выполнение, потому что ждут определенного внешнего события (обычно запроса на прерывание от какого-то известного устройства). В ожидании внешнего события такие задачи размещаются в специальной очереди. ОСРВ должна уметь при наступлении события сообщить о нем всем ожидающим его задачам и переместить их в очередь задач, готовых к выполнению.

При этом ОСРВ должна распознать источник запроса на прерывание и передать управление соответствующему обработчику, который выполнит все необходимые действия по обработке этого запроса, в том числе не забудет сбросить контроллер прерываний и оповестит внешнее устройство, что его запрос уже обрабатывается.

Следует помнить, что в больших и сложных системах во многих случаях невозможно обеспечить немедленную обработку запроса на прерывание, и может пройти не одна сотня микросекунд, пока запрос действительно будет обработан (даже если процедура его обработки имеет достаточно высокий приоритет).

Чтобы минимизировать эту задержку, необходимо весьма тщательно проектировать код обработчика прерываний, стараясь выполнить все необходимые действия как можно быстрее. Иногда хороший результат может быть достигнут только ценой усложнения аппаратных средств, берущих на себя те функции, которые программное обеспечение не может выполнить достаточно быстро.

Хороший пример такого рода - широтно-импульсный модулятор, который мы рассматривали в прошлых главах. Он может быть реализован программно, но более эффективна его аппаратная реализация.

Разумеется, в любой ОСВР должен быть какой-то механизм, который позволяет через определенные промежутки времени выполнять те или иные действия, подобно тому как это было у нас при разработке интерфейсов, основанных на прерывании от таймера TMRO.

Часто требуется, чтобы задачи, выполняемые под управлением ОСРВ, взаимодействовали, обмениваясь необходимыми данными. Такое взаимодействие может быть организовано напрямую, с помощью глобальных (доступных всем задачам) переменных, но в большинстве случаев лучше использовать поддерживаемый операционной системой *механизм передачи сообщений.* Тогда не требуется принимать специальных мер по синхронизации доступа к глобальным переменным, следя за тем, чтобы значение любой из них, записанное одной задачей, не изменилось раньше времени другой.

Процесс передачи сообщений состоит из двух фаз: сначала одна задача посылает сообщение другой, а затем помещается в специальную очередь, где ожидает от второй задачи подтверждения того, что сообщение получено.

С другой стороны, вторая задача посылает первой запрос на пересылку очередного блока данных и ожидает его прихода. Это может быть сделано или до того, как первая задача пошлет свое сообщение, или после. После получения сообщения вторая задача высылает первой подтверждение, а сама переводится в очередь задач, готовых к выполнению, ожидая, когда планировщик задач выделит ей квант рабочего времени. После прихода подтверждения первая задача также переводится в эту очередь.

Для разграничения доступа задач к общим системным ресурсам используются так называемые *семафоры* - специальные флажки. Каждому *ресурсу* (переменной или порту ввода-вывода) соответствует свой флажок. Когда какая-либо задача хочет использовать общий ресурс, она проверяет состояние его флажка. Если он сброшен, значит, ресурс свободен. Задача занимает этот ресурс (то есть читает или изменяет значение переменной либо регистра порта ввода-вывода), но предварительно устанавливает флажок, таким образом сообщая остальным задачам, что данный ресурс занят. Все другие задачи, которые захотят работать с тем же ресурсом, запросив состояние флажка, будут помещены в очередь ожидания. По выполнении операции задача освобождает ресурс, снова сбрасывая его флажок и давая другим задачам возможность использовать его.

Например, если одна задача начинает вывод текстового сообщения на экран жидкокристаллического дисплея, то до окончания этого процесса никакая другая задача не может получить доступ к этому дисплею — в противном случае мы увидели бы на экране начало другой строки вместо продолжения предыдущей. Поэтому вывод информации на дисплей должен регулироваться с помощью семафора, и все задачи, прежде чем начать запись, должны проверять его состояние.

Если программист по окончании работы с общим ресурсом забудет освободить семафор, то заблокирует выполнение всех других задач, которым требуется тот же ресурс, и даже задача, вызвавшая ошибку при необходимости повторного обращения к этому ресурсу, не сможет получить к нему доступ.

В табл. 5.1 перечислены основные типы запросов к ОСРВ; там же указаны параметры этих запросов и формат ответа. В зависимости от используемых инструментальных средств, типа операционной системы и процессора для обращения к ОСРВ и посылки запросов могут использоваться макросы или подпрограммы.

Для идентификации задач, выполняемых в системе, используются номера или специальные метки — адресные ссылки. С их помощью операционная система узнает, кому надо послать то или иное сообщение. Номера занимают меньше места, чем адреса, поэтому последние реже используются в микроконтроллерах.

Заметим, что в табл. 5.1 не указаны обращения к процедурам обработки прерываний. Они весьма специфичны, и мнения разных разработчиков по поводу того, как должен быть реализован обработчик прерываний в ОСРВ, иногда диаметрально противоположны. Зато имеется функция задержки на заданное число миллисекунд, формируемой с помощью аппаратного таймера (в многозадачной среде реализовать задержку программно без использования таймера и системы прерываний не удастся, если только не заблокировать все остальные задачи).

Таблица 5. 1. Основные сервисы ОСРВ

Сервис	Формат вызова	Возвращаемые параметры	Комментарии
Запуск задачи	StartTask (Address, Priority [, TaskName])	Дескриптор (номер) задачи или ошибка	Запустить указанную задачу начиная с заданного адреса
Завершение задачи	End()	–	Удалить выполнявшуюся задачу из памяти
Переход к следующей задаче	Next()	~	Перейти к выполнению следующей задачи, не дожидаясь окончания или приостановления текущей
Посылка сообщения	Send(Task, Message)	"	Послать заданное сообщение указанной задаче и ждать подтверждения его получения
Проверка сообщений	Check()	Номер сообщения (или ошибка)	Проверить очередь задач, ожидающих подтверждения получения посланных сообщений
Ожидание сообщения	Wait()	Номер сообщения	Поместить задачу в очередь ожидания, где она будет находиться до тех пор, пока не получит сообщение от другой задачи
Чтение полученного сообщения	Read(Message#)	Текст сообщения (или ошибка)	Получить данные, посланные другой задачей
Подтверждение получения сообщения	Ack(Message#)	Нет (или ошибка)	Послать уведомление о получении сообщения с указанным номером
Захват ресурса	Get(Semaphore#)	–	Если ресурс свободен, то занять его; в противном случае ждать освобождения
Проверка ресурса	Poll(Semaphore#)	Нет (или ошибка, если ресурс уже занят)	Проверить состояние ресурса по его семафору
Освобождение ресурса	Free(Semaphore#)	Нет (или ошибка, если этот семафор не был занят данной задачей)	Освободить ресурс, сбросив его семафор
Задержка	Dlay(msecs)	–	Выполнить задержку на указанное количество миллисекунд

Работа операционной системы реального времени осуществляется следующим образом. После включения питания и загрузки ОСРВ обычно запускается специальная задача - назовем ее AllTask. Она выполняет двойную функцию. Во-первых, с ее помощью осуществляется запуск других программ (задач). Для выполнения этой функции используется системный вызов StartTask (см. табл. 5.1):

```
void AllTask()                  // Постоянно выполняемая задача.
{
   StartTask(Task1,  1);        // Запускается задача Task1.
   While(1 == 1);               // Выполняется бесконечный цикл.
}
```

В приведенном фрагменте кода происходит запуск задачи Task1, причем ей назначается уровень приоритета 1.

Заметим, что приоритет фоновой задачи AllTask равен 0, поэтому тело бесконечного цикла выполняется, только если нет какой-либо другой задачи, которая в любом случае будет более приоритетной.

При разработке программного обеспечения для управления роботами надо учитывать следующие правила:

- существует три типа задач: *управляемые оператором* (вручную), *биологические* (код верхнего уровня по нашей классификации) и *периферийные* (обслуживающие внешние устройства). Задачи первого типа используют различные средства ввода информации — кнопки на боковой панели робота, интерфейс с компьютером, приемник команд дистанционного управления, - которые позволяют человеку вмешиваться в процесс выполнения программы. Задачи второго типа (биологический код) обычно заняты опросом датчиков, обработкой поступающих с них данных и выработкой команд для исполнительных устройств. Как правило, в ОСРВ должна выполняться только одна задача данного уровня. Каждая задача третьего типа предназначена для работы с каким-то одним (и только одним) внешним устройством. Эти задачи не способны инициировать обмен данными с внешними устройствами - они только могут обслужить запросы, сформированные биологическим кодом;
- если задача биологического уровня в настоящий момент не активна, то производится вызов функции задержки на некоторое количество миллисекунд. Пока длится задержка, могут выполняться другие задачи;
- в некоторых ОСРВ не требуется в явном виде использовать оператор цикла: дойдя до конца, задача сама продолжает свое выполнение с самого начала. В других системах все может оказаться совершенно наоборот: завершенная задача удаляется из памяти. Поэтому лучше использовать явный цикл для продолжения выполнения и явный системный вызов End() для завершения задачи.

5.2. ПРИМЕР ПРИЛОЖЕНИЯ, РАБОТАЮЩЕГО ПОД УПРАВЛЕНИЕМ ОСРВ

В предыдущем разделе при обсуждении операционных систем реального времени мы так и не показали, как они позволяют упростить разработку приложений. Здесь мы рассмотрим этот вопрос на простом примере, чтобы вы воочию убедились, как облегчается при использовании ОСРВ процесс добавления новых функций к работающему приложению.

Хотя примеры, приведенные выше, разрабатывались именно таким образом, чтобы облегчить модернизацию программ и обеспечить безболезненное добавление новых интерфейсов, наиболее простой и эффективный способ модернизации, не требующий изменения уже работающих программ, основан на использовании многозадачной ОСРВ.

Назовем наше приложение «Таракан», потому что функция робота будет заключаться в том, чтобы убежать подальше от источника света, скрывшись в самом

темном углу комнаты. Для этого потребуются четыре задачи: первая будет выполнять биологический код, вторая - осуществлять опрос датчиков столкновений (контактных или инфракрасных), третья - опрашивать состояние световых датчиков, а четвертая - управлять работой двух двигателей, каждый из которых связан со своим (левым и правым) колесом. Биологический код должен будет по состоянию датчиков принимать решение о направлении движения, изменяя его, чтобы робот огибал возникшее на пути препятствие, но в любом случае пытался оказаться подальше от источника света.

Чтобы представить связи между задачами, вы можете нарисовать их на листе бумаги, изобразив сами задачи в виде вершин графа скругленной формы, а семафоры, используемые для управления доступом к ресурсам, - в виде прямоугольных блоков.

Обратившись к табл. 5.1, вы не сможете найти в ней какой-либо функции, позволяющей одной задаче узнать состояние другой. Для этого используется стандартный механизм пересылки сообщений. Одна задача всегда может послать другой сообщение, запросив нужные данные, а та, в свою очередь, пошлет первой задаче сообщение, где будут содержаться значения необходимых переменных.

Биологический код для нашего примера выглядит следующим образом:

```
void biologic()   // Биологический код для приложения "Таракан" (Roach)
{
int left, right;                // Состояние световых датчиков.
int msgnum;                     // Номер сообщения.
msg message;                    // Текст сообщения.
while (1 == 1) <                // Бесконечный цикл.
    if (Poll(control_sem) == free) {    // Если нет активной задачи.
          message = biologic_task;
          Send("bump", message); // bump - имя задачи,
                                 // обслуживающей детекторы столкновений.
          msgnum = Wait();
          message = Read (msgnum);
          Ack(msgnum);           // Узнали состояние детекторов.
          if (message != no_collision) {      // Нет столкновений.
                message = motor_stop;         // Остановить двигатели.
                Send("motor", message);
          } else <               // Двигаться от источника света.
                message = biologic_task;
                Send("light", message);              // Узнать состояние
                message = get_left_light_sensor;     // левого
                Send("light", message);              // светового датчика.
                msgnum = Wait();
                left = Read(msgnum);
                Ack(msgnum);

                message = biologic_task;
                SendC'light",  message);             // Узнать состояние
                message = get_right_light_sensor;    // правого
                SendC'light",  message);             // светового датчика.
                msgnum = WaitO;
                right = Read(msgnum);
                Ack(msgnum);
```

```
                // "Отвернуться" от света:
                if (right >= left)
                    . message = motor_left;
                else
                    message = motor_right;
                Send("motor", message);
            >
    }
    Dlay(100);              // Задержка на 100 мс
                            // перед новым опросом датчиков.
}           // Конец цикла while.
}           // Конец биологического кода.
```

Здесь мы предполагали, что значение, получаемое от светового датчика, тем больше, чем сильнее он освещен. Чтобы удалиться от источника света, наш «таракан» включает двигатель, противоположный тому датчику, который освещен ярче. Например, если источник света расположен слева (выполняется ветвь «иначе» условного оператора `if`), то будет включен правый двигатель.

Для работы с командами дистанционного управления перед входом в программу мы опрашиваем состояние семафора control_sem - этот вопрос будет обсуждаться чуть позже.

Для опроса датчиков столкновений предназначена следующая программа:

```
void bump()                 // Опрос датчиков столкновений.
{
int left, right;            // Счетчики для левого и правого датчиков.
int msgnum;                  // Номер сообщения.
msg message;                 // Текст сообщения.
msg Sender;                  // Задача - отправитель сообщения.

left = 0; right = 0;        // Инициализация счетчиков.
while (1 == 1) {            // Бесконечный цикл.

    if (leftsensor == collision) // Сработал левый контактор.
        if (left < 5)
            left++; // Противодребезговая задержка.
        else ;
    else                        // Нет объектов слева.
        if (left > 0)
            left--;             // Противодребезговая задержка.

    if (rightsensor == collision) // Сработал правый контактор.
        if (right < 5)
            right++;            // Противодребезговая задержка.
        else ;
    else                        // Нет объектов справа.
        if (right > 0)  .
            right--;            // Противодребезговая задержка.
    msgnum = Check();                   // Было сообщение?

    if (msgnum != no_message) (         // Да, было.
        Sender = Read(msgnum);          // Прочитать его
        Ack(msgnum);                    // и подтвердить получение.
        if (Left == 5)      .
            if (Right == 5)
```

```
                message = "Both";      // Оба контактора.
            else
                message = "Left";      // Левый контактор.
        else if (Right == 5)
            message = "Right"          // Правый контактор.
        else if ((Right != 0) | (Left != 0))
            message = "Indeterminate"  // Неопределенность.
        else          // Оба счетчика имеют нулевое значение.
            message = "no_collision";  // Нет контакта.
        Send(Sender, message);         // Послать сообщение.
    }
    Dlay(10);      // Теперь пусть выполняются и другие задачи.
}          // Конец цикла while.
}
```

Здесь в бесконечном цикле происходит опрос состояния обоих датчиков (в данном случае неважно, какого типа эти датчики - они могут быть и дистанционными, и механическими; в комментариях к программе используется термин «контактор»). С помощью счетчиков реализуется необходимая задержка, устраняющая возможный дребезг контактов, если датчики являются механическими. Получив сообщение от другой задачи, программа посылает в ответ строку, описывающую состояние датчиков; при этом используется системный вызов Send. Обратите внимание на небольшую задержку в конце программы: она нужна, чтобы планировщик мог передать управление какой-либо другой задаче.

Если контакторы к текущему времени не находились в определенном состоянии хотя бы 50 мс, то их состояние считается неопределенным (посылается строка `"Indeterminate"`).

Предварительный запрос от другой задачи нужен для того, чтобы программа знала, кому посылать сообщение о состоянии датчиков. При описании основных системных вызовов мы не конкретизировали, каким способом задача-получатель сообщения может узнать идентификатор задачи-отправителя. В большинстве ОСРВ сообщение на самом деле - не простая строка, а структура, состоящая из нескольких полей, в одном из которых хранится номер или имя отправителя.

Следующая программа предназначена для управления двигателями. Она включает нужный двигатель на 100 мс и ожидает прихода сообщения, которое будет содержать указания для дальнейших действий. Перед включением двигателя проверяется состояние семафора `motor_running` и, если он свободен (сброшен), устанавливаем его, чтобы другая задача, ответственная за опрос световых датчиков, могла узнать, что в данный момент ведется работа с двигателями (напомним, что измерение сопротивления фоторезисторов в прошлой главе мы договорились производить только при выключенных двигателях, когда уровень помех минимальный). Вот текст программы:

```
void motor()                    // Работа с двигателями.
{
int 'i;
int msgnum;                     // Номер сообщения.
msg message;                    // Текст сообщения.

i = 0;                          // Сброс счетчика.
```

```
while (1 == 1) {

    if (--i == 0) { // Если прошло 100 мс, остановить двигатель.
          motors = stop;
          i = 1;
          Free(motor_sem) ;
    }

    msgnum = Check();                    // Есть сообщение?
    if (msgnum != no_message) {          // Да.
          message = Read(msgnum) ;       // Прочитать его
          Ack(msgnum);                   // и послать подтверждение.
          motors = message;
          if (message == stop) {
                i = 1; // Было сообщение об остановке двигателей.
                Free(motor_sem);         // Освободить семафор.

          } else {                       // Двигатель работает.

                i = 10;
                Get(motor_sem);          // Занять семафор.
          }
    }

    Dlay(10);       // Дать возможность выполняться другим задачам.

}                   // Конец цикла while.
}
```

Четвертая задача предназначена для работы со световыми датчиками. Она ожидает запроса от другой задачи, а получив его, отсылает в ответ сообщение о состоянии датчиков. Текст программы:

```
void  light()               // Опрос световых датчиков.

{
int msgnum;                 // Номер сообщения.
msg message;                // Текст сообщения.
msg Sender;                 // Идентификатор задачи - отправителя сообщения.
while (1 == 1) {                    // Бесконечный цикл.
    msgnum = Wait()                 // Ожидаем сообщение.

    Sender = Read(msgnum);          // Запомнили идентификатор отправителя.
    Ack(msgnum);                    // Подтвердили получение сообщения.
    msgnum = Wait()                 // Ждем запрос.

    message = Read(msgnum);          // Прочитали текст запроса.
    Ack(msgnum);                    // Подтвердили получение запроса.
    if (Poll(motor_sem) != Free) {

          // Выключить двигатели на время измерения:
          message = motor_stop;
          Send("motor", message);

    }
    if (message == get_left_light_sensor)

          message = LeftLightLevel;
    else
          message = RightLightLevel;
    Send(Sender, message) ;
```

```
}          // Конец цикла while.
}
```

Здесь мы не вызываем в явном виде функцию задержки; ее роль играет функция ожидания прихода сообщения от другой задачи.

Для того чтобы обеспечить одновременное выполнение всех четырех описанных задач, создадим еще одну задачу под названием `Task1`:

```
void Task1()                    // Загрузка задач.
{

    Start(biologic, 2, "biologic");      // Запуск биологического кода.
    Start(motor, 3, "motor");            // Задача для работы с двигателями.
    Start(bump, 3, "bump");              // Задача для работы с контакторами.
    Start(light, 3, "light");            // Задача для работы
                                         // со световыми датчиками.
    End();          // Завершение задач и освобождение ресурсов.
}
```

Однако всего перечисленного еще недостаточно. Наше приложение работает правильно, но пока только один раз. «Убежав» от источника света, робот остается в темном месте комнаты до тех пор, пока не изменяется положение источника или пока вы сами не переносите «таракана» на новое, более освещенное место. Будучи достаточно ленивым, я решил добавить к приложению еще одну задачу для приема и декодирования команд, подаваемых с помощью инфракрасного пульта дистанционного управления.

Мы будем использовать шесть кнопок: две - для старта и останова, еще две - для ускорения и замедления движения, и последние две - для поворота налево и направо. Ниже приведен текст программы:

```
void control()          // Дистанционное управление.
{
Get(control_sem);        // Занять семафор (ждать прихода команды).

msg message;             // Текст сообщения.
while (1 == 1) {         // Бесконечный цикл.
    if (RemoteControl = ButtonPress) {   // Есть команда.
          switch(Button) {               // Какая кнопка нажата?
                case Forward:            // Команда "Вперед"
                                         // (ускорить движение).
                      if (Poll(control_sem) == Free)
                            Get(control_sem);
                      Send(motor, Forward);
                      break;
                case Reverse:            // Команда "Назад"
                                         // (замедлить движение).
                      if (Poll(control_sem) == Free)
                            Get(control sem);
                      Send(motor, Reverse);
                      break;
                case TurnLeft:           // Команда "Налево".
                      it (Poll(control_sem) == Free)
                            Get(control_sem);
                      Send (motor, TurnLeft);
```

```
                    break;
                case TurnRight:      // Команда "Направо".
                    if (Poll(control_sem) == Free)
                        Get(control_sem);
                    Send(motor, TurnRight);
                    break;
                Case Stop:           // Команда "Стоп".
                    if (Poll(control_sem) == Free)
                        Get(control_sem);
                    Break;
                Case Start:          // Команда "Старт".
                    Free(control_sem);
                    break;
            } // Конец оператора witch.
        } // Конец оператора if.

        Dlay(10);            // Дать возможность выполняться другим задачам.
    }                        // Конец цикла while.
}
```

Ниже приведена программа, которая обеспечивает запуск всех пяти задач:

```
void Task1()             // Загрузчик.
{
    Start(control, 1, "control");
    Start(biologic, 2, "biologic");
    Start(motor, 3, "motor");
    Start(bump, 3, "bump");
    Start(light, 3, "light");
    End();
}
```

Как вы видите, при добавлении новой задачи не пришлось изменять текст уже написанных программ: только добавилась новая строка в программу-загрузчик. Именно в этом и заключается важное преимущество операционных систем реального времени.

Еще одно преимущество ОСРВ состоит в том, что при их использовании заметно уменьшается размер памяти, требуемый для хранения данных.

На разработку всех пяти вышеописанных программ я затратил несколько часов. Если бы программа разрабатывалась как монолитное приложение теми методами, которые были рассмотрены в предыдущей главе, то есть без поддержки ОСРВ, мне пришлось бы потратить несколько недель. Даже если не учитывать всех трудностей, которые при этом пришлось бы преодолеть, выигрыш при использовании ОСРВ очевиден.

5.3. КОНЕЧНЫЕ АВТОМАТЫ

Для проектирования кода верхнего уровня также можно использовать модель конечного автомата (КА), которую мы уже упоминали в предыдущей главе. Этот подход позволяет отказаться от вложенных условных операторов и запутанных

проверок состояния различных флагов, которые в противном случае потребовалось бы использовать в программе для принятия тех или иных решений.

Первоначально конечные автоматы возникли как специальные цифровые устройства (рис. 5.1), которые могли выполнять заданную последовательность действий в определенном порядке.

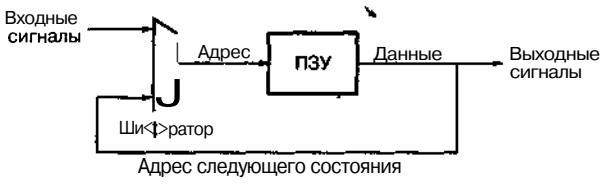

Рис. 5.1. *Аппаратная реализация конечного автомата*

По каждому фронту тактового генератора (на рис. 5.1 тактовый вход не показан) конечный автомат переходит в новое состояние - оно вычисляется исходя из текущего состояния автомата и значения входных сигналов. Состояние КА задается значением адреса `Addr` на входе постоянного запоминающего устройства (ПЗУ). Хранящаяся по этому адресу информация определяет, какими должны быть выходные сигналы К А и какой адрес будет на входе ПЗУ в следующем такте. Заметим, что заранее задана только часть адреса (некоторое число его разрядов); остальные биты адреса зависят от входных сигналов КА.

Для программной реализации КА хорошо подходит оператор выбора `switch`, который мы уже использовали при разработке конечных автоматов для работы с периферийными устройствами.

Приведем пример программной реализации КА для управления роботом. Допустим, мы хотим, чтобы наш робот двигался случайным образом:

```
main()                          // Случайные блуждания.
{
int State = 1;                   // Начальное состояние.
while(1 == 1)                    // Бесконечный цикл.
    switch (State) {             // Конечный автомат: .
          case 1:                // Двигаться вперед в течение 2 с.
                Move(Forward);
                Dlay(2000);      // Задержка на 2 с (используются
                                 // прерывания от таймера).
                if (Collision == "No") // Нет столкновения.
                      State = 5;     // Номер следующего состояния.
                else // Есть столкновение, надо объехать препятствие.
                      State = 10;    // Задний ход.
                break;
          case 5:                // Поворот налево (0,75 с).
                Move(TurnLeft);
                Dlay(750);
                if (Collision == "No") // Нет столкновения.
                      State = 1;     // Номер следующего состояния.
                else // Есть столкновение, надо объехать препятствие.
```

```
                    State = 10;    // Номер следующего состояния.
                break;
        case 10:       // Было столкновение, дать задний ход (1 с).
                Move(Reverse)  ;
                Dlay(1000) ;
                State = 15;    // Номер следующего состояния.
                break;
        case 15:       // Поворот направо (0,5 с).
                Move(TurnRight);
                Dlay(500);
                State = 1;     // Номер следующего состояния.
                break;

    }  // Конец оператора switch.

}   // Конец цикла while.

} // Конец примера.
```

Здесь перед переходом в очередное состояние выполняется задержка на определенное время, то есть функция `Dlay` выполняет роль тактовых импульсов, вызывающих срабатывание аппаратного конечного автомата.

При этом возникает новая проблема: во время действия задержки любые изменения входных сигналов будут проигнорированы, так как новое состояние КА уже определено (переменная `State` уже получила нужное значение).

Конечно, желательно, чтобы световые датчики и детекторы столкновений опрашивались постоянно. В принципе это не так уж сложно осуществить, но тогда надо добиться, чтобы подпрограмма задержки работала вне цикла конечного автомата, причем длительность задержки должна изменяться в зависимости от текущего состояния.

В следующем примере каждое состояние конечного автомата разбито на два. Первое состояние определяет характер движения робота и число циклов длительностью 50 мс каждый, в течение которых будет длиться задержка, а второе состояние соответствует периоду ожидания (в течение которого с каждым срабатыванием таймера уменьшается значение счетчика задержки), но в это же время происходит непрерывный опрос датчиков столкновений. Если какой-то датчик сработал, то выполняются необходимые действия.

```
main()                          // Случайные блуждания, вариант 2.
{
int  State = 1;                 //  Начальное  состояние.
int  DlayCount = 0;             // Счетчик задержки.
while (1 == 1) {          // Бесконечный цикл.
   switch(State) {
        case 1:         // Движение вперед (2 с)
                Move(Forward);
                DlayCount = 40;       // 40 раз выполнить
                                      // задержку на 50 мс.
                State = 2;      // Выполнение задержки
                                // (и опрос датчиков).
                break;
        case 2:         // Движение вперед (2 с).
                if (Collision == "No")          '    // Нет столкновений.
```

```
                        if (--DlayCount)
                                State = 2;  // Если не 0, то оставаться
                                            // в текущем состоянии.
                        else
                                State = 5;
                  else          // Было столкновение.
                        State = 10;
                  break;
            case 5:               // Поворот налево (0,75 с).
                  Move(TurnLeft);
                  DlayCount = 15;  // Соответствует задержке 750 мс.
                  State = 6;     // Выполнение задержки.
                  break;
            case 6:               // Поворот налево (0,75 с).
                  if (Collision == "No")
                        if (--DiayCount)
                                State = 6;
                        else
                                State = 1;
                  else
                        State = 10;
                  break;
            case 10;  // Было столкновение (ехать назад в течение 1 с).
                  Move(Reverse);
                  Dlay(1000) ;
                  State = 15;
                  break;
            case 15:       // Поворот направо (0,5 с).
                  Move(TurnRight);
                  Dlay(500);
                  State = 1;
                  break;
      }  // Конец оператора switch.

      Dlay(50);      // Общая задержка 50 мс.

   }  // Конец цикла while.

   }  // Конец примера.
```

Здесь состояния с номерами 10 и 15 не имеют задержек, потому что определяют действия робота при срабатывании датчиков столкновений. Предполагается, что, отъезжая чуть назад, робот не столкнется с каким-либо другим предметом. Вообще-то это предположение не совсем корректно для нашего беспорядочно двигающегося робота.

Лучше было бы разместить дополнительные датчики в задней части корпуса робота и опрашивать их состояние, пока он движется назад. При необходимости вы сможете самостоятельно внести соответствующие изменения в программу.

Достоинства конечно-автоматного подхода к процессу разработки управляющей программы - его простота и легкость, с которой могут быть добавлены новые состояния. Так как программный код, соответствующий каждому отдельному состоянию, достаточно прост, он не отнимает много процессорного времени, давая возможность выполняться функциям нижних уровней.

Однако при усложнении логики работы программы код КА также становится достаточно сложным для понимания и отладки. Тем не менее конечно-автоматный подход в большинстве случаев достаточно перспективен при проектировании программного обеспечения для управления роботами.

5.4. ДИСТАНЦИОННОЕ УПРАВЛЕНИЕ РОБОТОМ, СОВЕРШАЮЩИМ СЛУЧАЙНЫЕ БЛУЖДАНИЯ

В этом разделе мы рассмотрим несложный пример использования программного конечного автомата, который позволяет роботу, в свободном состоянии совершающему случайные блуждания, принимать и выполнять команды дистанционного управления.

Допустим, робот должен двигаться вперед в течение двух секунд, на следующей секунде совершать поворот направо (затратив на это 0,75 с), после чего все повторяется заново. Натолкнувшись на какой-либо предмет, робот сначала отъезжает назад (затратив на это полсекунды), а затем в течение следующих 0,5 с поворачивает в сторону, противоположную той, с которой зафиксировано срабатывание контактора. После объезда препятствия робот снова возвращается к своему обычному циклу «вперед-направо».

При использовании традиционного метода программирования (с помощью условных операторов) биологический код будет выглядеть следующим образом:

```
while (1 == 1) {

    CollisionFlag = 0;            // Сначала столкновений нет.

    LeftMotor(LeftForward);        // Левый двигатель включен.
    RightMotor(RightForward);     // Правый двигатель включен.
    Dlay = RTC + 2000;   .        // В течение 2 с.

    while (Dlay != RTC)
          if (GetLeftWhisker())         // Сработал левый контактор?
                CollisionFlag = 1;
          else if (GetRightWhisker())  // Сработал правый контактор?
                CollisionFlag = 2;

    if (!CollisionFlag) {

          // Код, выполняемый, когда столкновений нет.
          LeftMotor(LeftForward);  // Поворот направо.
          RightMotor(RightReverse); // Реверс правого двигателя.
          Dlay = RTC + 750;     // Задержка 0,75 с.
          while (Dlay != RTC)
                if (GetLeftWhiskerO)
                                  // Сработал левый контактор?
                      CollisionFlag = 1;
                else if (GetRightWhiskerO)
                                  // Сработал правый контактор?
                      CollisionFlag =2;
    }  // Конец условного оператора if.

    if (CollisionFlag •== 1) { // Столкновение зафиксировано слева.
```

```
            LeftMotor(LeftReverse);        // Обьезд.
            RightMotor(RightReverse);      // Реверс правого двигателя.
            Dlay = RTC + 500;              // В течение 0,5 с.
            while (Dlay != RTC);           // Задержка.
            LeftMotor(LeftForward);        // Поворачиваем направо.
            RightMotor(RightReverse);      // Реверс правого двигателя.
            Dlay = RTC + 500;              // Задержка 0,5 с.
            while (Olay != RTC);
    }  // Конец условного оператора if.

    if (CollisionFlag == 2){ // Столкновение зафиксировано справа.
            LeftMotor(LeftReverse);        // Обьезд.
            RightMotor(RightReverse);      // Реверс правого двигателя.
            Dlay = RTC + 500;      //  0,5 с.
            while (Dlay != RTC);
            LeftMotor(LeftReverse);        // Реверс левого двигателя.
            RightMotor(RightForward);      // Поворачиваем налево.
            Dlay = RTC + 500;              // 0,5 с.
            while (Dlay != RTC);

    }  // Конец условного оператора if.

} // Конец цикла while.
```

Эта программа вполне работоспособна, но выглядит довольно неуклюже. Вряд ли вы смогли бы быстро разобраться, что делает робот, если бы не дополнительные пояснения. Кроме того, весьма проблематично добавить к данному коду новые функции. Поэтому целесообразно выбрать какой-либо другой способ программирования.

Все намного проще, если мы используем конечно-автоматный подход:

```
while (1 == 1) {                // Бескойечный цикл.

    switch (State) {
            case 1:                 // Двигаться вперед в течение 2 с.
                    State++;        // Начать с состояния 2.
                    StateDlay = RTC + 2000;      // Задержка 2 с.
                    LeftMotor(LeftForward); // Левый двигатель работает.
                    RightMotor(RightForward); // Правый двигатель работает.
                    break;

            case 2:                     // Ждать 2 с.
                    if    (GetLeftWhisker()) // Сработал левый контактор.
                            State = 10;
                    else
                        if (GetRightWhisker()) // Сработал правый контактор.
                            State = 20;
                    else if (StateDlay == RTC)
                            State++;        // Нет столкновений.
                    break;

            case 3:                          // Поворот направо (0,75 с).
                    State++;
                    StateDlay = RTC + 750;
                    LeftMotor(LeftForward); // Левый двигатель работает.
                    RightMotor(RightReverse); // Реверс правого двигателя.
                    break;
```

```
    case 4:                 // Ждать 0,75 с.
        if (GetLeftWhisker())
            State = 10;
        else if (GetRightWhisker())
            State = 20;
        else if (StateDlay == RTC)
            State = 1;
        break;

    case 10:                // Препятствие слева.
        State++;
        StateDlay = RTC + 500;
        LeftMotor(LeftReverse);
        RightMotor(RightReverse);
        break;

    case 11:                // Ждать 0,5 с.
        if (StateDlay == RTC)
            State++;
        break;

    case 12:                // Объехать препятствие (0,5 с).
        State++;
        StateDlay = RTC + 500;
        LeftMotor(LeftForward);
        RightMotor(RightReverse);
        break;

    case 13:                // Ждать 0,5 с.
        if (StateDlay == RTC)
            State = 1;
        break;

    case 20:                // Препятствие справа.
        State++;
        StateDlay = RTC + 500;
        LeftMotor(LeftReverse);
        RightMotor(RightReverse);
        break;

    case 21:                // Ждать 0,5 с.
        if (StateDlay == RTC)
            State++;
        break;

    case 22:                // Объехать препятствие (0,5 с).
        State++;
        StateDlay = RTC + 500;
        LeftMotor(LeftReverse);
        RightMotor(RightForward);
        break;

    case 23:                // Ждать 0,5 с.
        if (StateDlay == RTC)
            State = 1;
        break;

  } // Конец оператора switch.

} // Конец цикла while.
```

Не так уж сложно добавить сюда какие-либо дополнительные функции. При этом просто увеличится число состояний. Заметим, что размер программы растет по линейному закону, в отличие от обычного метода программирования, когда сложность и длина программы при наращивании числа функций возрастают в геометрической прогрессии.

Здесь приведен только пример биологического кода. В электронном приложении к этой книге в файле Code\Statemc\wallhug.c приводится полный исходный текст программы для дистанционно управляемой модели мыши фирмы Tamiya, а в файле Code\Statemc\servo.c - аналогичный текст программы для управления роботом с двумя серводвигателями и световыми датчиками.

Если вы сравните эти файлы, то с удивлением обнаружите, что биологический код в обоих случаях совершенно одинаков. Единственное отличие касается функций `LeftMotor` и `RightMotor`. Скорость движения модели мыши может изменяться с помощью команд дистанционного управления (для этого приходится использовать широтно-модулированный сигнал), а у второго робота серводвигатели вращаются с постоянной скоростью.

Такой результат достигнут не только благодаря достоинствам конечно-автоматного подхода. Следует не забывать и о нашей концепции «трех уровней», согласно которой необходимо отделять реализацию механических и электронных интерфейсов от высокоуровневого биологического кода. Как вы видите, сочетание этих двух подходов дает прекрасные результаты.

5.5. ПОВЕДЕНЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Если задуматься, можно прийти к выводу, что поведение робота - это просто набор его реакций на те или иные внешние воздействия. Например, робот, имитирующий поведение маленького ребенка, может действовать следующим образом:

```
GetToyBehavior()          // Вижу игрушку, хочу играть...
{

printf("Мама, папа! Можно мне вон ту игрушку?!\n");
Response = Input.Data;    // Узнать реакцию родителей.

while. ((Response != "Да") && (Response != "Хорошо, мы купим ее")) {

    // Реагировать на отказ родителей купить игрушку.
    Position(LieDown);     // Лечь на пол, стучать ногами и руками
    Crying(Start);         // и кричать.
    printf("Ну, почему-у-у\n");

    if (Input.Pending) {

          // Если нет ответа, то ждать:
          Response = Input.Data;              // Узнать реакцию родителей.

        if (Response ! = "Перестань немедленно, а не то выключу!"){

              // Перестать капризничать.
              printf("Я больше не буду!")
              Response = "Да";
```

```
            } // Конец оператора if.
        ) // Конец оператора if.
    } // Конец цикла while.
    Position(Stand);          // Подняться с пола.
    Crying(Stop);             // Перестать кричать.
    } // Конец.
```

Этот шуточный пример довольно точно иллюстрирует поведение робота, который опрашивает состояние своих входных датчиков, чтобы узнать об окружающей его ситуации, а затем согласно заложенному в него алгоритму выбирает, как реагировать на то или иное изменение обстановки. Даже на этом простом примере нам удалось проиллюстрировать множественность целевых установок, характерную для реальных программ: цель ребенка — покупка понравившейся ему игрушки, но если дело заходит слишком далеко (к сожалению или к счастью, настоящего ребенка нельзя «выключить», но можно придумать более педагогичный выход из создавшегося положения), то перед угрозой наказания ребенок перестает капризничать.

Отличие большинства реальных случаев от нашего примера заключается в том, что робот в каждый момент времени не обязательно выполняет только одну задачу. Возможно, вы подумали, что очень сложно реализовать выполнение сразу многих функций в одной программе, но на самом деле это вполне возможно и часто не требует какого-то особо искусного программирования.

Единственное, что действительно представляет трудность, - выбор подходящего для данной ситуации механизма арбитража, который комбинировал бы решения, принимаемые по нескольким алгоритмам, чтобы выработать то единственное, что должно подаваться на двигатели робота.

Рассмотрим характерный для многих практических случаев пример, когда помимо основного кода верхнего уровня, принимающего решения и контролирующего поведение робота, имеется некоторая фоновая задача, которая при выполнении определенных условий включается в работу и перехватывает управление роботом.

Допустим, мы хотим, чтобы, завидев человека, робот тут же развернулся в его сторону и начал движение либо к нему, либо от него - в зависимости от того расстояния, на котором находится человек.

Для нашего проекта, который на этот раз так и останется только на бумаге, выберем робота с двумя сервоприводами (по одному для каждого колеса) и ультразвуковым дальномером. Допустим, для управления положением колес робота в выходной порт записываются целые 8-разрядные числа: −128 соответствует команде «полный реверс», 0 - «стоп», +127 - «полный вперед». Пусть имеются две функции: `distance()` - для определения расстояния до человека и `vector()` - для определения угла его расположения относительно робота. Обе функции возвращают целые числа со знаком (типа int). Данные о режиме работы обоих двигателей будем хранить в структуре

```
struct motor {
    int left;            // Левый двигатель.
    int right;           // Правый двигатель.
}
```

Для простоты будем полагать, что робот находится на ровной площадке бесконечных размеров, так что не требуется отвлекаться на опрос датчиков столкновений. Единственное, чем будет занят робот, - движение по своему маршруту (согласно заложенному в него алгоритму) и измерение расстояния до человека. На практике, конечно, придется побеспокоиться о том, чтобы различать нескольких людей, да и вообще как-то отличать человека от обычных предметов. Хотя здесь мы не обсуждаем все эти задачи, но при используемом подходе добавление новых функций к биологическому коду не составляет никакой проблемы.

Сначала давайте выясним, как сделать, чтобы робот поворачивался в сторону человека:

```
motor directionBehavior(motor Data)
{          // Управление двигателями (поворот в сторону человека).

int dir;   // Направление.

    Data.left = Data.right = 0;
    dir = direction();                  // Узнать направление.
    if (dir > 0)                        // Надо повернуться налево.
          if (dir < tuneAngle) {              // Медленный поворот.
                Data.left = -slowTurn;
                Data.right = slowTurn;
          } else if (dir < turnAngle) {
                Data.left = -mediumTurn;
                Data.right = mediumTurn;
          } else {                      // Быстрый поворот.
                Data.left = -fastTurn;
                Data.right = fastTurn;
          }

    else if (dir < 0)                   // Надо повернуться направо.
          if (dir > -tuneAngle) {       // Медленный поворот.
                Data.left =  slowTurn;
                Data.right = -slowTurn;
          } else if (dir > -turnAngle) {
                Data.left = mediumTurn;
                Data.right = -mediumTurn;
          } else {                      // Быстрый поворот.
                Data.left = fastTurn;
                Data.right = -fastTurn;
          }

    return Data;          // Вернуть два числа - команду двигателям.

}   // Конец.
```

Здесь предполагается, что все углы и скорости объявлены как глобальные переменные, то есть в главной программе. Если угол поворота, который необходимо

выполнить роботу, слишком велик, то поворот происходит быстро. После того как угол достигнет заданного предела, скорость замедляется. Когда робот развернулся в сторону человека, оба двигателя выключаются.

Если расстояние до человека больше заданного порога, то робот должен двигаться в его сторону:

```
motor closeBehavior(motor Data)
{          // Управление двигателями (движение в сторону человека).

int dist;                  // Расстояние до человека.

    dist = distance();     // Узнать расстояние до человека.
    Data.left = Data.right = 0;
    if (dist > closeClosest)
           if (dist > closeFurthest)
                  Data.left = Data.right = forwardFull;
           else
                  Data.left = Data.right =
                         ( forwardFull « (dist - closeClosest) ) /
                         (closeFurthest - closeClosest);

    return Data;

}          // Конец.
```

Чем дальше человек, тем быстрее робот стремится к нему подъехать. Когда расстояние сокращается до некоторого предела, команды перестают вырабатываться.

Обратите внимание, что для вычисления скорости мы сначала выполняем операцию умножения, а только потом - деления. Дело в том, что при делении целочисленных величин, если делитель больше делимого, результат будет нулевым. Поэтому сначала полная скорость умножается на разность текущего и минимального расстояний, и только затем результат делится на весь диапазон расстояний (от максимального до минимального). В результате скорость сближения убывает по линейному закону. Напомним, что использование вещественной арифметики существенно увеличило бы размер программы, так как все операции над числами в формате с плавающей точкой в микроконтроллерах PICmicro реализованы только программно.

Наконец, если расстояние до человека слишком мало, то робот должен двигаться назад:

```
motor awayBehavior(motor Data)
{                          // Управление двигателями (пятимся назад).

int dist;

    dist = distance0;      // Узнать расстояние до человека.
    Data.left = Data.right = 0;
    if (dist < awayFurthest)
          if (dist < awayClosest)
                 Data.left = Data.right = reverseFull; // Полный назад.
          else
                 Data.left = Data.right =
                        (reverseFull * (awayFurthest - dist)) /
                        (awayFurthest  -  awayClosest);
    return Data;
} // Конец.
```

Для выполнения всех трех задач составим главную программу:

```
void main(void)                    // Фоновое приложение.
{

motor directionData;               // Команда двигателям.
motor closeData;
motor awayData;
motor CombinedData;                // Комбинация всех трех команд.

   while (1 == 1) {                // Бесконечный цикл.
          pause(RunDlay);          // Сначала робот подчиняется
                                   // заложенному в него алгоритму.

          directionData = directionBehavior(directionData);
          closeData = closeBehavior(closeData);
          awayData = awayBehavior(awayData);

          // Сформировали три команды, теперь комбинируем из них одну:
          CombinedData.left = directionData.left +
                              closeData.left + awayData.left;

          CombinedData.right = directionData.right +
                               closeData.right + awayData.right;

          WheelCommand(CombinedData); // Послать команду для выполнения.

          pause(LoopDelay);        // Дать роботу время на выполнение
                                   // команды и новый цикл измерений.

   }      // Конец оператора while.
}         // Конец программы.
```

В табл. 5.2 перечислены все используемые здесь параметры и приведены их значения по умолчанию.

Таблица 5.2. Параметры фоновой задачи управления роботом

Параметр	Значение по умолчанию	Описание
`tuneAngle`	5°	Максимальный угол, при котором еще происходит медленный поворот
`turnAngle`	20°	Угол для средней скорости поворота
`slowTurn`	5	Медленная скорость поворота
`mediumTurn`	15	Стандартная скорость поворота
`fastTurn`	40	Скорость быстрого поворота
`closeClosest`	36 дюймов	Минимальное расстояние до человека
`closeFurthest`	48 дюймов	Предельное расстояние, на котором скорость приближения к человеку должна быть максимальной
`forwardFull`	50	Максимальная скорость приближения к человеку
`awayClosest`	36 дюймов	Минимальное расстояние до человека, на котором робот должен начать удаляться от него с максимальной скоростью
`awayFurthest`	48 дюймов	Максимальное расстояние до человека, на котором робот должен начать удаляться от него с минимальной скоростью
`reverseFull`	-50	Максимальная скорость при движении от человека

Значения этих параметров можно несколько изменить в ту или иную сторону, чтобы добиться наиболее естественного поведения робота.

Разберем простейшую ситуацию. Допустим, робот находится от человека на расстоянии 47 дюймов (119 см) и ориентирован так, что человек оказывается на 15° левее продольной оси робота. Наши три программы сформируют в этом случае три команды, представленные в табл. 5.3.

Таблица 5.3. Команды, сформированные программами

	directionBehavior	closeBehavior	awayBehavior	Combined
Left	-15	45	-4	26
Right	15	45	-4	56

Так как левый двигатель (соответствующий параметр - Left) будет вращаться медленнее правого (соответствующий параметр - Right), то робот будет поворачиваться налево, в сторону человека. При этом он будет приближаться к нему, пока не окажется слишком близко, после чего либо остановится, либо начнет отъезжать назад. Поведение робота было бы аналогичным, если бы он управлялся с помощью ПИД-регулятора или fuzzy-контроллера[1]. Но приведенная здесь программа заметно проще той, что пришлось написать для реализации ПИД-регулятора.

Еще раз обращаю ваше внимание на тот факт, что все параметры в программе целочисленные. Везде, где это возможно, я старался избегать использования арифметики с плавающей точкой.

5.6. НЕЙРОННЫЕ СЕТИ И ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Рассказывая о роботах, было бы невозможно обойти молчанием такое понятие, как *искусственный интеллект.* Многие полагают, что роботы (и компьютеры), наделенные искусственным интеллектом, должны быть столь же разумны, как и человек. В предыдущей главе этой книги мы рассматривали множество различных интерфейсов, позволяющих роботу воспринимать те или иные условия и проявлять Свою реакцию на их изменение, а в этой главе обсудили, как можно программно реализовать алгоритмы, заставляющие робота демонстрировать окружающим нечто похожее на разумное поведение. Но это были лишь попытки имитировать настоящий интеллект; наши конструкции не только не являлись разумными формами жизни, но даже не могли претендовать на «звание» искусственного интеллекта.

Очень приблизительное определение искусственного интеллекта выглядит следующим образом: *это умение машины выполнять поставленные задачи, пытаясь преодолеть противодействие окружающей обстановки и обучаясь,* то есть

[1] Fuzzy-контроллеры (англ. fuzzy - нечеткий) отличаются от обычных тем, что оперируют близкими человеку качественными понятиями «большой», «средний», «малый» и т.п. - *Прим. перев.*

извлекая из собственного опыта знания, которые затем можно будет использовать для того, чтобы выполнять свои задачи еще более эффективно.

В этом определении не упоминаются обработка визуальной информации, умение вести разговор или разрушить укрепления противника и другие аналогичные способности, которые, может быть, ассоциируются у многих с термином «искусственный интеллект». Легко впасть в заблуждение, если трактовать это понятие слишком широко.

Но даже реализация интеллекта в представленном здесь узком смысле - не такая уж простая задача. Дело осложняется тем, что мы не можем точно определить, что же такое «знание» или «обучение».

Здесь уместно вспомнить известный эксперимент с червяком в лабиринте. Червяка помещали в лабиринт, имеющий форму буквы «Т». Двигаясь от основания лабиринта вперед, червяк должен был выбрать, в какую сторону повернуть: налево или направо. В одном коридоре его ждал удар электрическим током, а в противоположном - корм. После нескольких неудачных попыток червяк в конце концов запомнил, куда следует идти, и всегда выбирал коридор с едой.

Этот эксперимент получил широкую известность благодаря не столько тому, что была доказана способность червей к получению простейших навыков, сколько другому замечательному факту. После того как достаточное количество особей было обучено поворачивать в определенном (одном и том же для всех) направлении, они были скормлены своим сородичам, которые еще не прошли такого курса обучения. Спустя некоторое время с насытившимися «новичками» провели аналогичный эксперимент. Оказалось, что при первом же прохождении лабиринта вместо ожидаемых 50% «попаданий» они выбирали правильное направление в 56% случаев[1].

Оставляя в стороне вопрос об обучении за обеденным столом, обсудим, каким способом можно заставить программу вести себя так же, как и не слишком интеллектуальный червяк. Иными словами, после нескольких тренировочных «забегов» в лабиринте робот, управляемый нашей программой, наконец должен научиться выбирать правильное направление. Если оперировать теми же понятиями, что и в опыте с червями, мы должны как-то запрограммировать боль от удара электрическим током и удовольствие, получаемое при приеме пищи (а возможно, и чувство радости от того, что электроды оказались в другом коридоре). Ведь даже заставить программу сделать случайный выбор в самом первом опыте - и то задача не из тривиальных.

[1] Эксперименты над плоскими червями (планариями) в лабиринте впервые были проведены американским исследователем Йерксом в 1912 году. Они показали, что черви способны обучаться и переобучаться, но полученный навык держался недолго (10-15 ч). Затем подобные эксперименты в различных модификациях (в том числе и опыты с кормлением необученных особей обученными) были повторены многими учеными. Справедливости ради следует заметить, что многие исследователи оспаривают существование эффекта передачи опыта при кормлении, так как при повторении экспериментов не получили аналогичного результата. Те же, кому удалось зафиксировать передачу навыков столь безжалостным образом, расходятся друг с другом в объяснении данного феномена. Диапазон предположений довольно широк - от передачи информации на уровне РНК до телепатии. - *Прим. перев.*

Когда речь идет о тренировке животных (или человека), все эти вопросы интуитивно понятны и близки нам, но как перевести их на язык программирования? Как заставить робота или компьютер чувствовать голод, жажду, боль, удовольствие?

При рождении мы получаем некоторый стандартный набор рефлексов, заложенных на генетическом уровне, и с их помощью постепенно обучаемся жить в мире, руководствуясь соответствующими концепциями и обобщая свой опыт каким-то неосознаваемым способом.

В детстве я однажды довольно сильно обжег руку утюгом. До сих пор, когда в моем сознании совмещаются образы боли, одежды, утюга и руки, я всегда вспоминаю о том случае, и мое тело автоматически делает шаг назад, а рука отдергивается, дабы уберечься от ожога. После этого я много раз обжигался в разных местах, но ни один из этих инцидентов (хотя некоторые из них тоже были весьма болезненными) не оставил в моей памяти такого заметного следа, и я не пытаюсь инстинктивно защитить другую руку или ногу. Уверен, вы тоже сможете припомнить какой-нибудь случай из своей жизни, когда некоторая комбинация внешних воздействий глубоко врезалась в вашу память и с тех пор при возникновении похожих ситуаций вы инстинктивно действуете так, чтобы оградить себя от повторения неприятных событий[1].

По большому счету, обучение людей или животных не должно отличаться от обучения компьютеров. Вспомните приведенное выше определение искусственного интеллекта - ведь оно похоже на описание действий любого разумного существа, хотя, к сожалению, не дает нам возможности понять, как именно мы обобщаем наблюдаемые факты, чтобы выбрать наиболее эффективный способ действий.

Я верю в то, что в ближайшем будущем появятся компьютерные программы, которые смогут и сдать экзамен, и поддержать разговор, но их, скорее всего, нельзя будет сравнить с настоящим (пусть даже и искусственным) интеллектом, если они не будут обладать возможностью обучаться, подобно тому как это делают животные и человек. Вероятно, пройдет немало лет, прежде чем ученые действительно смогут понять, как можно научить программы воспринимать и обобщать информацию, приходящую из окружающего мира.

Современное состояние науки об искусственном интеллекте позволяет предположить, что первые шаги к пониманию принципов работы нашего сознания мы можем преодолеть, изучая работу основных клеток, из которых состоит мозг животных и человека, - *нейронов*.

Эксперименты с искусственными моделями нервных клеток ученые начали проводить в 40-х годах XX века, пытаясь имитировать работу мозга. Большое

[1] Если читатель увлекается не только программированием, но и психологией, то, вероятно, слышал о нейролингвистическом программировании (НЛП), в котором среди прочих используется техника так называемых «якорей», когда за той или иной комбинацией внешних воздействий психотерапевт закрепляет в подсознании человека нужную реакцию. - *Прим. перев.*

количество нейронов соединялось, образуя так называемые искусственные *нейронные сети*.

Если вы знакомы с цифровой электроникой, то можете представить нейрон в виде некоторого многовходового логического элемента (AND, OR, XOR и т.п.). Но на самом деле это аналоговый элемент, который воспринимает несколько входных сигналов, причем имеет различную чувствительность к тем сигналам, которые поступают на разные его входы. Суммарное воздействие всех входных сигналов определяет *активность* нейрона - степень его возбуждения. Эта величина подвергается обработке и в виде результирующего сигнала подается на единственный выход нейрона. Выходной его сигнал, в свою очередь, является входным для многих других нервных клеток, которые соединены с рассматриваемым нейроном.

Можно придумать несколько хороших определений нейронной сети, но наилучшее из них выглядит следующим образом: нейронная сеть - это один или множество связанных друг с другом простых процессорных элементов, работающих параллельно, функционирование которых зависит от структуры сети, степени связи между элементами и алгоритма их работы[2].

Проще говоря, нейронная сеть - это набор связанных элементов, каждый из которых воспринимает свои входные сигналы и вырабатывает по некоторому правилу выходной сигнал. На рис. 5.2 показан пример простой нейронной сети, которая по состоянию двух световых датчиков и двух детекторов столкновений формирует команды для двух двигателей робота.

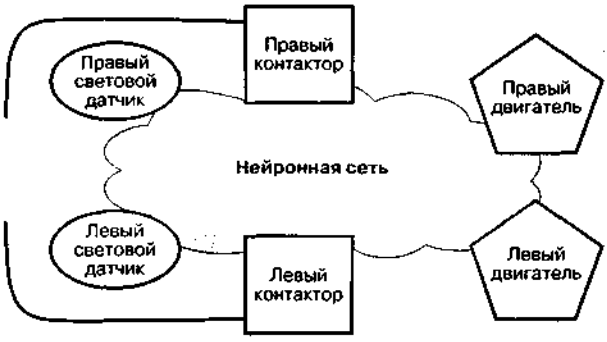

Рис. 5.2. *Нейронная* сеть **управляет** *двигателями робота*

На языке математики все это можно описать простой формулой:

Выходы = f (Входы),

где f - некоторое правило, согласно которому набор значений входных сигналов преобразуется в соответствующий набор значений выходных сигналов.

[2] DARPA Neural Network Study, 1988. - AFCEA International Press. - P. 60.

Алгоритм, определяющий это правило, может быть реализован с помощью логических функций, конечно-автоматных моделей или просто в виде некоторой последовательности операторов любого языка программирования, но важно понимать, что данный алгоритм не предопределяет реакцию сети на то или иное входное воздействие, а только описывает способ, который используется нейронной сетью для ее обучения.

Такое обучение строится на примерах, что позволяет провести аналогию с тренировкой животных. На вход сети подается тот или иной набор внешних воздействий, и выходные сигналы сравниваются с теми, которые мы желали бы иметь в данных условиях. Если ошибка сети (разница между желаемым и действительным значениями выходного сигнала) велика, то чувствительность входов каждого нейрона по определенным правилам изменяется таким образом, чтобы снизить эту ошибку. Так повторяется много раз для различных наборов входных воздействий и соответствующих наборов желаемых реакций, пока сеть не научится правильно реагировать на предлагаемые воздействия.

Очень важно правильно выбрать структуру нейронной сети, то есть количество составляющих ее нейронных элементов и связи между ними. Если нейронов будет слишком мало или связи между ними окажутся неэффективными, то сеть не сможет научиться тем вещам, которым мы хотим ее обучить. Если нейронов, наоборот, будет слишком много, то процесс обучения грозит затянуться на неопределенное время.

После начальной тренировки сеть может быть использована для управления роботом, но при этом ее обучение продолжится, если она сама будет способна каким-то образом оценивать качество своего поведения. Например, показанная на рис. 5.2 нейронная сеть может следить за временем, которое потребовалось ей на то, чтобы в определенной ситуации найти источник света, минимальное количество раз столкнувшись с окружающими предметами.

Кроме имитации поведения бабочки, летящей на свет, такой механизм обучения может быть использован и для более полезных применений, например для управления захватом робота-манипулятора, который должен найти и подобрать определенный предмет. Человек делает это совершенно не задумываясь, но программирование такой задачи традиционными методами довольно затруднительно.

Способ обучения нейронной сети, основанный на примерах, может быть реализован различными способами. По существу, такое обучение сродни тренировке собак: дрессировщик дает команду, после чего поощряет правильное ее выполнение и наказывает за неправильное. Это многим показажется удивительным, но существуют также методы обучения без учителя, когда на входы нейронной сети подаются различные наборы воздействий, но никто не говорит ей, какими в каждом из этих случаев должны быть ее выходные сигналы.

Для выбора правильного набора параметров нейронной сети можно использовать так называемые *генетические алгоритмы.* Они, как и сами нейронные сети, заимствованы исследователями у живой природы, поскольку основаны на имита-

ции происходящих в ней процессов размножения, изменчивости и естественного отбора. Генетический алгоритм позволяет набору (популяции) нескольких нейронных сетей, предназначенных для решения некоторой задачи, эволюционировать таким образом, чтобы в результате сформировать сеть, которая смогла бы решить эту задачу самым эффективным образом.

Обладает ли нейронная сеть искусственным интеллектом? Многие исследователи склонны ответить на этот вопрос положительно, ведь способ формирования реакции нейронной сети на входные воздействия очень напоминает процесс принятия решений живыми организмами. Но, на мой взгляд, это было бы правильнее назвать «искусственным рефлексом». Пока что нейронная сеть скорее моделирует рефлекторное движение руки человека, стукнувшего молотком по своему пальцу, чем проявление настоящего интеллекта.

ГЛАВА 6
ПРОЕКТИРОВАНИЕ АВТОМАТИЧЕСКИХ УСТРОЙСТВ

Тот факт, что эта глава находится в конце книги, может вызвать некоторое удивление. Но нам нужно было обсудить много важных вещей, прежде чем приступать к проектированию настоящих роботов.

В предыдущих главах мы рассмотрели, как реализуются различные интерфейсы и как программируется процесс принятия решений на верхнем уровне. Теперь пришла пора выяснить, как можно заставить все эти отдельные части работать совместно.

Но перед тем, как начать разработку автоматического устройства, надо спросить себя, для чего конкретно оно будет предназначено. Мало сказать: «Я хочу сконструировать робота, чтобы разобраться, как он работает». Надо сформулировать свои намерения более точно, например: <<Я хочу сконструировать двухколесного робота, каждое колесо которого управляется отдельным двигателем и который двигается по прямой линии, пока не встретит на своем пути препятствие, после чего объезжает его и снова продолжает движение в прежнем направлении».

После того как вы четко определите, что же хотите получить, можно приступать к проектированию. По мере своего продвижения вперед вы, вероятно, захотите кое-что изменить в исходной формулировке, но главное, чтобы у вас была отправная точка. Иначе вам даже не удастся толком описать, что необходимо добавить к первоначальному замыслу.

Важно, чтобы ваш первоначальный замысел был не слишком фантастичен. Многие радиолюбители поначалу хотят достичь результатов, в погоне за которыми даже целые коллективы профессиональных разработчиков безрезультатно потратили много лет и миллионы долларов. Классическим примером такого безумного проекта может служить робот, который по приказу своего хозяина умел бы приносить пиво из холодильника. Здесь сначала надо придумать, как робот откроет дверцу холодильника, как отыщет там бутылку с пивом, возьмет ее, закроет дверцу и, наконец, как найдет своего хозяина, чтобы отдать ему пиво. А если хозяин и холодильник находятся на разных этажах? И это только некоторые из самых очевидных вопросов. Уверен, немного поразмыслив, вы придумаете еще кучу других, не менее заковыристых. Но на главные вопросы надо найти ответ прежде, чем начинать работу над проектом. Иначе лучше и не начинать.

Я нисколько не преувеличиваю, говоря, что ученые потратили кучу времени и денег на подобные разработки. В самом деле, если робот должен уметь

при необходимости подняться вверх по лестнице, значит, это умение не должно зависеть от ширины и высоты ступенек, а также материала, из которого они сделаны.

В главе 4 мы рассмотрели множество различных интерфейсов; они были разработаны таким образом, чтобы в максимальной степени облегчить их интеграцию друг с другом в пределах одного большого проекта. Тем не менее очень часто при проектировании сложных автоматических устройств на основе микроконтроллеров бывает весьма нелегко достичь устойчивой работы всех подсистем. То не хватает объема памяти для размещения всех подпрограмм, то слишком долго выполняется биологический код, вследствие чего робот не успевает вовремя среагировать на все изменения окружающей обстановки, то один интерфейс мешает работе других... Эти проблемы по возможности лучше решать с самого начала, так как намного сложнее вносить изменения в почти готовое приложение.

Здесь стоит упомянуть такое явление, как «журнальная зависть». Часто, прочитав интересную статью в радиолюбительском журнале, понимаешь, что именно эту идею надо использовать в своей новой разработке, но проблема в том, что многим советам, которые дает автор статьи, нельзя последовать в точности. Некоторые комплектующие могут оказаться слишком дорогими или вовсе недоступными, поэтому придется искать им замену, и не всегда равноценную. Описание каких-либо особенностей конструкции в статье может быть опущено. В результате, даже повторяя чужую разработку, вы получаете нечто отличное от нее — и не обязательно с тем же результатом. Разумеется, в любом случае вы только выиграете, если постараетесь перенять опыт других радиолюбителей и использовать понравившиеся вам находки в своих проектах.

В конце концов, сконструировать робота довольно просто. Сложнее заставить его делать в точности то, что вы задумали. Поэтому в данной главе мы рассмотрим несколько полезных приемов отладки приложений, которые помогут вам понять, что на самом деле происходит с роботом в тех или иных случаях. Я подчеркиваю, что любые решения разработчик должен принимать, опираясь на данные экспериментов, а не на свои предположения по поводу того, что обусловливает неправильное поведение робота.

6.1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Хочу дать вам хороший совет: записывайте все! Заведите привычку носить с собой блокнот, и как только у вас появится стоящая идея, запишите ее. На самом деле человеческая память не так надежна, как многие склонны полагать. Должен признаться, не раз я часами - и безуспешно! - пытался вспомнить гениальную идею, которая осеняла меня во время завтрака.

Всегда старайтесь начинать с малого. Если вы задумали сконструировать робота-слугу, который всюду следует за вами и с готовностью выполняет любые ваши поручения, то разумнее начать с создания небольшой тележки (но с большим запасом места для добавления новых электронных блоков), которая только и умеет, что передвигаться по горизонтальной поверхности.

Планируя свою работу, всегда прогнозируйте самый худший случай (поверьте, вы с ним довольно скоро столкнетесь). Никогда не предполагайте, что первый же пришедший вам в голову способ реализации той или иной функции окажется удачным. Как правило, для достижения небольшого успеха требуется перепробовать несколько возможностей.

Возьмем для примера одну из программ, представленных в этой книге, - управление светодиодным индикатором. Пытаясь добиться того, чтобы микроконтроллер включал и выключал светодиод с заданной периодичностью, я был вынужден рассмотреть следующие возможности:

- программная реализация задержек;
- программный опрос таймера TMR0;
- часы реального времени на основе прерываний от таймера TMRO (именно этот метод и был использован);
- широтно-импульсная модуляция сигнала с применением таймера TMR2.

Какой из этих методов является наилучшим? Какие дополнительные возможности обеспечивает тот или иной способ реализации заданной функции? Какое решение наиболее простое и не затрудняет дальнейшую модернизацию устройства? Помните о том, что метод, прекрасно показавший себя на лабораторном столе, может оказаться совершенно ненадежным, работая в реальных условиях, поэтому вы всегда должны иметь пространство для маневра.

К сожалению, многие светлые мысли приходят слишком поздно, когда работа над проектом уже близится к концу. Поэтому иногда разумнее зафиксировать новую идею на бумаге, чтобы опробовать ее в будущем, вместо того чтобы без конца переделывать уже почти готовое устройство. Лучше руководствоваться правилом: «Если не сломано - не следует исправлять».

И последнее: не бойтесь заимствовать чужие идеи. Разумеется, я не подстрекаю вас к плагиату или похищению чужих чертежей, но свежие мысли и решения, почерпнутые вами у других разработчиков, иногда оказываются единственным выходом из тупика, в который вы зашли. К тому же чужие идеи всегда можно применить совершенно новым способом.

В сети Internet вы найдете огромное количество информации по микроконтроллерам и методам конструирования автоматических устройств, готовые схемы, идеи, документацию по различным комплектующим и многие другие сведения, которые будут для вас чрезвычайно полезными или, напротив, никогда не пригодятся. Используйте эту информацию. Но должен предостеречь: не все, что вы найдете в Сети, имеет практическую ценность. Многие данные могут быть не вполне корректными, а некоторые и вовсе неверными. Разумеется, это касается любой области, не только робототехники. И дело здесь не только в том, что разместить страничку в сети Internet может любой желающий. Нередко я встречал на разных сайтах так называемые «бесплатные» программы и проекты, которые были доступны для свободного скачивания, но на поверку оказывались неработоспособными, а лишь играли роль приманки, вынуждая вас приобрести более «продвинутый» проект или обратиться к его автору за консультацией - разумеется, тоже не бесплатной.

Итак, вы хотите приступить к проектированию нового автоматического устройства. Начать следует с *технического задания.* Звучит несколько официально, но имеется в виду всего лишь список требований, которым должно удовлетворять разрабатываемое устройство. Начните с простых утверждений, поясняющих, что и как должен делать робот.

Далее пункт «как» надо конкретизировать. Приведем примерный список основных вопросов:

Какую аппаратную платформу следует использовать?

Каковы должны быть размеры и вес устройства?

Какая необходима скорость передвижения?

Какие двигатели будут использоваться?

Что послужит источником питания?

Какие меры безопасности надо предусмотреть?

Какие контроллеры будут использованы?

На каком языке программирования будет разрабатываться управляющее программное обеспечение?

Какие датчики будут снабжать робота информацией об окружающей обстановке?

Какие будут применены исполнительные (выходные) устройства?

Какие методы ручного управления необходимы?

Каким должен быть интерфейс пользователя?

Какова будет стоимость всего проекта?

На такие вопросы нельзя формально ответить «нет» или «без ограничений». Вы можете считать робота совершенно безопасным (потому что он маленький), но если в нем используются свинцовые, кислотные или никель-кадмиевые батарейки, то они вредны для окружающей среды. А если робот весит несколько десятков килограммов, то трудно представить, что он может натворить, если выйдет из-под контроля из-за неправильной работы управляющего устройства.

Кроме списка аппаратных требований надо подготовить список требований к программному обеспечению. Следует определить:

- язык программирования;
- средства отладки приложения (эмулятор, симулятор);
- доступность средств загрузки программы в микроконтроллер (наличие специального программатора);
- используемые в программе векторы прерываний;
- программные интерфейсы, встроенные в систему;
- используемые в программе структуры данных.

Следует отметить, что оба списка требований взаимосвязаны. Выбор того или иного микроконтроллера ограничивает ваши возможности по выбору средств программирования и отладки.

Заметьте, что здесь пока не учтены такие важные требования, как, например, предположительный размер программы и данных. Знание этих параметров может существенно повлиять на выбор контроллера и методов программирования, но на начальных этапах проектирования очень трудно оценить, сколько памяти потребуется для

всего приложения. Я уже много лет занимаюсь такими вещами, но до сих пор часто выходит, что реальный объём готового приложения оказывается раза в два больше того, который был запланирован исходно.

Ничто не оказывает такого впечатления на зрителей, как быстрота передвижения робота. Как ни странно, ваша конструкция произведет впечатление более сложной и «умной», если будет двигаться быстро. Однако за это приходится расплачиваться, используя более мощные двигатели и, соответственно, источники питания. Кроме того, двигаясь с большой скоростью, робот должен уметь быстро реагировать на окружающую обстановку. Поэтому в погоне за высокими скоростями разработчик рискует приобрести себе кучу неприятностей в будущем, когда захочет модернизировать свой проект, так как зачастую ни малейшего запаса прочности уже не остается.

Чуть ниже мы обсудим, как составлять список требований к будущему проекту. Если вы радиолюбитель, а не профессиональный разработчик, то, возможно, все это покажется вам пустой тратой времени, но уверяю вас, что дело обстоит иначе. Хорошо составленное техническое задание существенно облегчит дальнейшую работу по проектированию автоматических устройств.

По мере того как вы будете продвигаться вперед, может возникнуть необходимость внесения тех или иных изменений в уже составленное техническое задание (например, если окажется, что выбранный вами микроконтроллер не поддерживает какую-либо возможность, которую вы предусмотрели в самом начале, или вам так и не удастся достать нужный элемент, а возможные варианты его замены не позволяют обойтись без некоторых переделок). Ничего страшного - смело вносите изменения в первоначальный список требований!

6.2. ВЫБОР ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

После того как будут определены и перечислены все функции, которые должен выполнять робот, можно приступить к выбору периферийных устройств. Если выбор окажется не оптимальным, вы рискуете потратить кучу времени на безуспешные попытки найти ошибку в программе, в то время как истинным виновником неправильной работы является какое-нибудь устройство ввода-вывода.

Например, может оказаться, что некоторое входное устройство слишком медлительно или требует сложного программирования; в результате процедура обработки прерываний будет отнимать слишком много командных циклов работы процессора и не оставит достаточно времени для выполнения биологического кода. Или некоторые интерфейсы в борьбе за общий аппаратный ресурс проявят плохую совместимость друг с другом, и приложение будет пропускать некоторые внешние события (скажем, команды дистанционного управления или моменты столкновений с окружающими предметами).

Продемонстрированный в данной книге подход, основанный на разделении функций механического, электронного и биологического уровней и реализации процедур первых двух уровней в виде обработчика прерываний от таймера, позволяет свести к минимуму вероятность подобных проблем, однако следует понимать, что

такое решение не является панацеей - вполне возможны случаи, когда разработчику придется проявить изобретательность, пытаясь увязать друг с другом различные интерфейсы. В любом случае при проектировании следует стремиться к тому, чтобы отдельные функции механического и электронного уровней как можно меньше взаимодействовали друг с другом.

Кроме того, не следует забывать, что для правильной работы биологического кода необходимо оставить достаточно процессорного времени на его выполнение. Поэтому надо стремиться к наиболее простой реализации механических и электронных интерфейсов, избегая использования в обработчике прерываний сложных вычислений, чисел в формате с плавающей точкой, операций в цикле и т.п. Все трудоемкие вычисления следует производить в главной программе (то есть с помощью биологического кода). Только тогда будет гарантия, что во время их выполнения не нарушится работа какого-либо периферийного устройства.

Таким образом, можно сформулировать следующие основные рекомендации по разработке функций механического и электронного уровней:

- используйте конечные автоматы для реализации механических и электронных интерфейсов. Это может несколько замедлить их работу, но зато не помешает работе других интерфейсов;
- не используйте переменные с плавающей точкой и массивы;
- старайтесь избегать сложных математических операций (умножение, деление, вычисление логарифмов и тригонометрических функций);
- используйте микроконтроллер с наиболее широким набором встроенных функций.

6.3. ВЫБОР ЭЛЕКТРОННЫХ КОМПОНЕНТОВ И МЕТОДОВ ПРОГРАММИРОВАНИЯ

При разработке технического задания пригодится следующий список вопросов, ответы на которые определят выбор электронных компонентов, требуемых при создании робота.

Какой микроконтроллер желательно использовать?

Какого типа будет тактовый генератор?

Каким должен быть источник питания?

Какие потребуются датчики и другие устройства ввода информации?

Какие потребуются двигатели и другие устройства вывода?

Нужно ли использовать запоминающие устройства (кроме памяти, встроенной в микроконтроллер)?

Насколько доступны необходимые электронные компоненты?

Последний вопрос особенно важен, если вы хотите опубликовать схему своего устройства и надеетесь, что другие радиолюбители смогут ее собрать. Например, в самом начале работы над программатором El Cheapo я использовал МОП транзистор с каналом р-типа, считая, что он широко распространен и вряд ли у кого-то возникнут трудности с его приобретением или заменой достаточно близким

аналогом. К сожалению, мое предположение не подтвердилось: многие радиолюбители стали жаловаться на то, что не могут достать полевой транзистор, так что его пришлось заменить распространенным биполярным p-n-p транзистором 2N3906.

Следует помнить, что в радиолюбительских конструкциях в первую очередь важна доступность элементов — зачастую она более значима, чем стоимость. Поэтому перед тем, как опубликовать свою схему в сети Internet или в радиолюбительском журнале, стоит просмотреть каталоги ведущих производителей радиоэлементов.

Рано или поздно те элементы, которые вы на протяжении многих лет использовали в своих конструкциях, устаревают. Это большая проблема, но часто только психологическая: во многих случаях среди новых элементов можно подыскать достаточно близкий аналог. Ну, а если нет, то приходится вносить изменения в устройство и в управляющую программу.

Чтобы обезопасить себя от возможных проблем, желательно уже в самом начале работы над проектом стараться использовать только самые распространенные и стандартные электронные компоненты. Для примера рекомендую:

- диоды 1N4001, 1N4192 (1N915);
- транзисторы 2N3904/2N3906, BC547/BC557;
- полевые транзисторы IRF510/IRF9510;
- керамические резонаторы ECS/Panasonic;
- инфракрасные светоизлучающие диоды и детекторы фирмы Watrony;
- логические элементы ТТЛ серии 74LS;
- интегральные стабилизаторы напряжения 78(L)xx;
- таймер (7)555.

Каких-либо проблем с резисторами и конденсаторами, как правило, не возникает. Иногда трудно найти танталовые конденсаторы, используемые для фильтрации напряжения питания логических микросхем, но их можно заменить керамическими.

Ключевой вопрос - выбор микроконтроллера. Обычно разработчики основывают свое решение не столько на объективном сравнении требований технического задания и характеристик того или иного микроконтроллера, сколько на своих пристрастиях и опыте предыдущих проектов. Не последними доводами служат доступность микроконтроллера и разнообразие программного обеспечения для него. Разумеется, не следует останавливать свой выбор на микроконтроллере, для работы которого требуются еще какие-то дополнительные элементы кроме источника питания 5 В, генератора тактовых импульсов и «подтягивающего» резистора для входа сброса.

Что касается быстродействия микроконтроллера (его максимальной тактовой частоты), этот параметр обычно не является существенным. Наоборот, для уменьшения мощности, потребляемой от источника питания, а также для снижения уровня помех я рекомендую вам выбирать для своих конструкций не слишком высокую частоту работы тактового генератора. Гораздо более важная характеристика контроллера - его устойчивость к помехам и работоспособность при помехах от периферийных устройств, входящих в конструкцию робота.

Наиболее оптимальным решением при конструировании электронных управляющих устройств мобильных роботов является использование печатного монтажа.

Разрабатывая печатную плату электронного блока, следует иметь в виду, что при работе автоматических устройств всегда присутствуют следующие мешающие факторы:

- электрические помехи;
- электромагнитные наводки;
- механические вибрации;
- акустические шумы;
- колебания температуры и влажности окружающей среды.

Кроме того, не следует забывать, что у мобильных автоматических устройств есть подвижные части, на которые могут наматываться провода, поэтому все соединения надо производить очень аккуратно.

Я советую использовать монтажные провода минимальной длины и паяные соединения; всегда заботьтесь о качестве заземления и надежности крепления механических узлов.

При размещении элементов схемы на печатной плате учтите, что в дальнейшем может возникнуть необходимость модернизации схемы или добавления новых блоков. Не старайтесь компоновать все элементы слишком плотно - оставляйте место для дополнительных элементов.

Следует подчеркнуть, что макетные платы, которые используются в проектах, описанных в главе 4, являются наихудшим решением с точки зрения надежности конструкции и возможности модернизации схемы. Они облегчают проверку несложных схем, но вы можете на собственном опыте убедиться, как нелегко оказывается разобраться в переплетении проводов, чтобы найти неисправность или подключить новый блок.

Для размещения микроконтроллера всегда необходимо использовать специальный разъем, который облегчит возможность его перепрограммирования или замены.

6.4. ИСПЫТАНИЯ РОБОТА

Разработчику следует определить список испытаний, которые должен пройти робот, прежде чем можно будет сказать, что работа закончена и готова к тому, чтобы вынести ее на суд зрителей. Типичный список испытаний выглядит следующим образом:

- проверка стабильности работы во всех режимах (в том числе при пуске-останове двигателей) при крайних значениях напряжения питания (4,75 и 5,25 В);
- проверка начальной инициализации при включении питания и готовности выполнения команд пользователя;
- проверка правильности выполнения команд пользователя;
- проверка правильности работы датчиков.

Кроме того, возможно, придется позаботиться о разработке необходимой документации, расчете надежности работы всех узлов и даже о тестировании разработанной конструкции на соответствие санитарным и прочим требованиям и нормам.

Хотя большинство микроконтроллеров сохраняет свою работоспособность при изменении напряжения питания в более широких пределах, чем указано выше, не стоит забывать и о других электронных компонентах, многие из которых выдерживают только небольшие колебания напряжения питания (±5%).

Помните о соблюдении мер безопасности. Разумеется, маленький робот не сможет причинить вреда окружающим, но он может разбиться сам, упав со стола, если вы вдруг потеряете над ним контроль.

По этой причине я советую всегда предусматривать какой-либо метод блокировки колес, который предохранит робота от поломки, если вы вдруг забудете выключить напряжение питания, так как даже полностью отлаженное устройство может неожиданно убежать от вас, случайно получив сигнал от светового датчика.

Во многих случаях при тестировании схемы недостаточно одного логического пробника. Идеальный вариант - использование осциллографа и логического анализатора, позволяющее проверить правильность всех выходных сигналов при всех возможных значениях входных.

По роду занятий я почти всю жизнь проверял правильность работы тех или иных устройств, поэтому вопрос о тестировании мне наиболее близок. К сожалению, даже самое добросовестное тестирование не дает полной гарантии работоспособности устройства во всех обстоятельствах: всегда существует огромное количество вариантов возможных неисправностей.

При тестировании в первую очередь надо убедиться, что на предусмотренные значения входных сигналов ваше устройство реагирует правильно. Следует контролировать значение этих входных сигналов, так как всегда может оказаться, что странности в поведении робота обусловлены не ошибкой в программе, а случайным отблеском, попавшим на световой датчик.

Если значения входных сигналов или служебных переменных управляющей программы вышли за допустимые пределы, то робот должен остановиться и каким-либо образом сигнализировать о возникшей проблеме. В наших проектах мы не использовали сторожевой таймер, но в реальных конструкциях он бывает весьма полезным, позволяя сформировать сигнал сброса микроконтроллера, если управляющая программа зависла и команда сброса сторожевого таймера не выполнилась вовремя.

Ваш проект должен быть подробно документирован. В противном случае вряд ли кто-нибудь захочет его реализовать. Разумеется, вся документация должна быть доступна в электронном виде. Самым простым и переносимым для электронных документов является обычный текстовый формат ASCII. Но если необходимо включение в текст рисунков, графиков, схем и/или формул, то лучшим решением будут доступные на всех компьютерных платформах гипертекстовый формат HTML и формат PDF фирмы Adobe.

6.5. ПОИСК ОШИБОК

Вместо термина *отладка* (debug), который традиционно ассоциируется с несколько небрежным и не слишком вдумчивым процессом устранения ошибок в программах, я предпочитаю использовать более солидное выражение *анализ ошибок* (failure analysis). Интересно, как простая замена технического термина сразу смещает акценты: не поиск «жучков», а структурированный анализ проблемы, позволяющий локализовать и устранить источник неправильной работы устройства!

Вместо того чтобы вслепую заменять все элементы подряд, надеясь, что после очередной замены устройство вдруг начнет работать правильно, примите к сведению следующий алгоритм:

- четко сформулируйте, в чем заключается проблема;
- рассмотрите возможные варианты причин неправильной работы;
- разработайте план выявления истинной причины и ее устранения.

Разработчик должен уметь провести подобный анализ в разнообразных ситуациях, в том числе и в самых крайних случаях, когда кажется, что устройство совсем не работает или работает «практически правильно».

Когда еще нет никаких идей по поводу того, что именно вызывает проблему, следует проверить логические уровни на всех элементах схемы. На данном этапе я обычно использую простой логический пробник. Если такое исследование не прояснило ситуацию, то, возможно, придется взять в руки авометр и проверить уровни напряжений на аналоговых и цифровых входах и выходах. Следует записывать все результаты измерений.

Проверяя работу встроенного тактового генератора, надо помнить, что дополнительная емкость, вносимая логическим пробником, может нарушить процесс генерации и вызовет зависание программы. Чтобы убедиться в том, что программа все еще выполняется, следует инициализировать выполнение какой-нибудь простой процедуры с четко видимым внешним эффектом (например, миганием светодиода).

От вас требуется предельное внимание. Мне стыдно признаться, сколько раз я готов был воскликнуть «Эврика!» и тут же обнаруживал, что проверяю не ту ножку микросхемы. После того как все замеры сделаны, отложите устройство в сторону и проанализируйте, значения каких сигналов оказались вне ожидаемого диапазона.

Любое несоответствие объясняется одой из двух причин: либо внешнее устройство подает на данный вывод неправильный сигнал, либо управляющая программа формирует на нем неправильное значение. Чтобы выяснить, с каким именно вариантом вы имеете дело, можно использовать омметр, проверяя с его помощью сопротивление между данным выводом и массой до и после его отключения от остальной схемы. К сожалению, для отключения вывода может потребоваться паяльник.

Следует соблюдать максимальную объективность, стараясь не делать слишком поспешных предположений относительно того, что привело к возникновению

ошибки. Неверно определив причину, люди начинают искать только подтверждения своей догадки, пропуская все остальные возможные варианты.

После проверки схемы можно приступать к тестированию программного обеспечения. Прежде всего я рекомендую вынуть микроконтроллер из схемы и проверить прошивку РПЗУ с помощью программатора. Обратите внимание на правильность слова конфигурации, проверив, корректно ли установлен тип тактового генератора, не разрешена ли случайно работа сторожевого таймера и т.п.

Если все в порядке, переходите к самой программе. При этом не следует забывать о необходимости подробно записывать все действия - как ни удивительно, это сэкономит много времени.

Например, если проблемы возникли с последовательным интерфейсом передачи данных, выясните, связана ли проблема с временными задержками, значениями отдельных битов или их количеством. После того как ситуация стала более определенной, можно выполнить моделирование работы программы с помощью симулятора, чтобы в еще большей степени локализовать источник неправильного поведения робота.

Теперь требуется выяснить, что стало причиной ошибки. Для этого может потребоваться мысленный эксперимент, в ходе которого вы должны представить себе во всех подробностях, как и что происходит в управляющей программе и самой схеме в тот или иной момент.

Следует понимать, что причин неисправности может быть сразу несколько. Работая над своими конструкциями, я не раз сталкивался со случаями, когда насчитывалось до пяти различных источников проблем, которые мешали правильной работе устройства.

Поэтому не следует быть слишком ленивым. Поверьте, во многих случаях вы только сэкономите время, если сделаете шаг немного в сторону и начнете с отладки более простого приложения, где должна будет проявиться одна из имеющихся проблем. Тогда можно будет без помех изучить ситуацию и опробовать различные варианты решения, после чего опять вернуться к сложному проекту.

6.6. МОДЕРНИЗАЦИЯ УСТРОЙСТВ

После того как ваша схема, наконец, заработает, вы захотите добавить к ней новые периферийные устройства и разнообразные функции. При этом у вас, скорее всего, не возникнет особых проблем с электроникой или механическими частями. Самым сложным, как правило, оказывается перераспределение аппаратных ресурсов, встроенных в микроконтроллер. Речь идет о памяти программ и данных, таймерах, обработчиках прерываний, блоках формирования ШИМ-сигнала.

Другая проблема возникает, когда вы хотите заимствовать для своего проекта программный код из какого-либо источника (например, Internet). В таком случае оказывается, что различные участки кода написаны с использованием

разных стилей и методик, поэтому может потребоваться кардинально переработать программу, чтобы заставить все это работать вместе.

Если приложение спроектировано правильно, то внесение необходимых изменений - не слишком сложная задача, но в противном случае проще написать все заново, чем пытаться как-то исправить отдельные фрагменты.

Уже в самом начале работы над проектом желательно закладывать в него возможность модификации. Своего рода образцом правильной разработки приложений могут служить примеры, приведенные в главе 4, где для реализации механических и электронных интерфейсов мы использовали прерывания от таймера, срабатывающего каждую миллисекунду.

При проектировании биологического кода надо помнить, что для его правильной работы в промежутке между запросами на прерывание следует оставлять по крайней мере 25% всех командных циклов, таким образом, отводя для выполнения обработчика прерываний не более 75% времени. Сложные вычисления нельзя реализовывать в процедуре обработки прерываний - их надо выносить в основную программу.

В идеале различные интерфейсные функции должны использовать разные аппаратные средства. Мы нарушили это правило лишь однажды, когда рассматривали приемник команд дистанционного управления, который использовал тот же инфракрасный детектор, что и обнаружитель объектов.

Применяя чужие разработки, не следует пытаться использовать в своем проекте программный код, для которого не имеется исходного текста. Дело в том, что в новом окружении даже самая правильная программа может потребовать некоторых переделок, а это невозможно, если имеется только объектный код.

Все описанные в данной книге устройства мы реализовали с использованием только одного микроконтроллера, но при модернизации уже имеющихся схем наилучшее решение иногда заключается в добавлении еще одного контроллера. Поэтому никогда не помешает оставить свободными какие-нибудь два вывода, которые затем пригодятся для обеспечения их взаимосвязи. Две линии нужны для реализации синхронного обмена, так как в этом случае не требуется загружать таймеры обоих микроконтроллеров.

Даже если в схеме использовано несколько микроконтроллеров, желательно, чтобы только один из них выполнял биологический код. В противном случае программное обеспечение заметно усложняется, поскольку требуется обеспечить взаимодействие отдельных частей биологического кода, выполняемого на разных процессорах. Кроме того, отладить такое распределенное приложение будет не очень-то просто.

И последнее замечание: не забывайте, что следует отключать двигатели на то время, пока микроконтроллер работает с периферийными устройствами. В течение опроса инфракрасных, световых и звуковых датчиков надо стремиться к тому, чтобы уровень помех был минимальным.

ПРИЛОЖЕНИЕ 1

Глоссарий

Здесь приведен список всех использованных в книге терминов и расшифровка аббревиатур.

Адрес — порядковый номер регистра, ячейки памяти команд или данных в пределах пространства памяти процессора.

Адресное пространство - диапазон возможных адресов ячеек памяти или портов ввода-вывода, с которыми может работать процессор.

Активные элементы - интегральные микросхемы, транзисторы и другие элементы электронных схем, для функционирования которых требуется внешний источник энергии.

Ампер - величина силы тока: 1 А = 1 Кл / 1 с. *См.* Кулон.

Аналоговый сигнал - сигнал, который может принимать любые значения в некотором диапазоне.

Анод - положительный электрод. *См.* Катод.

Асинхронная последовательная передача - метод обмена данными, в котором передача сообщений происходит побитно (по одной линии) и не используется дополнительная линия для передачи импульсов синхронизации.

Ассемблер - инструментальное программное средство, с помощью которого исходный текст программы, написанной на языке низкого уровня, преобразуется в объектный файл.

АЦП, аналого-цифровой преобразователь (ADC, **Analog-to-Digital** Converter) - микросхема, предназначенная для преобразования входного аналогового напряжения в цифровое представление. *См.* ЦАП.

Биполярные логические схемы - логические схемы, выполненные на базе биполярных транзисторов (единичные устройства или интегрированные сборки). См. n-p-n транзистор, p-n-p транзистор.

Битовая маска - набор битов, используемый для установки/сброса отдельных разрядов некоторого двоичного числа.

Внутрисхемное программирование **(ICP, In-Circuit** Programming) - загрузка программы в микроконтроллер без его извлечения из целевой схемы.

Восьмеричное число - число, представленное в позиционной системе счисления по основанию 8.

Гарвардская архитектура - архитектура процессора, имеющего раздельные шины для подключения памяти команд и памяти данных.

Герц - единица измерения частоты: 1 Гц = 1 $с^{-1}$.

Гистерезис - зависимость передаточной характеристики от направления изменения входного сигнала. Обычно используется для фильтрации шумов в цифровых схемах.

Графический интерфейс пользователя (GUI, Graphic User Interface) - простой, ориентированный на непрофессионального пользователя интерфейс ввода-вывода информации, который задействует экран, клавиатуру и мышь.

Двоичные числа - числа, представленные в позиционной системе счисления по основанию 2. Например, десятичное число $37 = 32 + 4 + 1 = 2^4 + 2^2 + 2^0 = 10101$.

Демультиплексор - логическая микросхема, используемая для передачи входного сигнала на одну из нескольких выходных линий. *См.* Мультиплексор.

Десятичное число - число, представленное в позиционной системе счисления по основанию 10.

Дребезг механических контактов - эффект, заключающийся в появлении многочисленных ложных всплесков сигнала при коммутации механических переключателей.

ЖКИ (LCD, Liquid Crystal Display) - жидкокристаллический индикатор, используемый для отображения информации (обычно символьной).

Защита от дребезга - устранение ложных всплесков сигнала, возникающих при коммутации механических переключателей.

Интерпретатор - программа, предназначенная для непосредственного выполнения исходного текста приложения (без его предварительной компиляции).

Инфракрасный свет (IR, infrared) - свет, имеющий длину волны 760 нм и более, невидимый для человеческого глаза.

Катод — отрицательный электрод. *См.* Анод.

Катушка индуктивности - проволочная обмотка, предназначенная для хранения энергии магнитного поля. Часто используется для генерации или фильтрации электрических сигналов.

Кварцевый резонатор - устройство, используемое для генерации тактовых импульсов с точно заданной частотой.

Керамический резонатор - устройство, используемое для генерации тактовых импульсов определенной частоты (обычно с меньшей точностью, чем у кварцевого резонатора).

КМОП логика (CMOS) - тип логических микросхем, основанных на использовании n- и p-канальных униполярных транзисторов.

Командный цикл - период времени (всегда кратный длительности тактового цикла), в течение которого выполняется одна машинная команда процессора.

Компаратор - электронное устройство с двумя входами, которое используется для сравнения напряжений.

Компилятор - инструментальное программное средство, предназначенное для преобразования исходного текста программы в объектный код.

Компоновщик (linker) - инструментальное программное средство, предназначенное для объединения нескольких объектных модулей и/или библиотек и формирования исполняемого файла.

Конденсатор - устройство для сохранения электрического заряда. Часто используется в электронных схемах для фильтрации сигналов. Существуют керамические, танталовые, электролитические и др. конденсаторы; танталовые и электролитические являются поляризованными, то есть при их подключении необходимо соблюдать полярность. Короткий вывод поляризованного конденсатора — это катод (отрицательный электрод).

Конечный автомат — аппаратное устройство или логическая модель системы с конечным набором возможных состояний. Переходы между состояниями осуществляются при выполнении заданных условий и отображаются с помощью графа переходов (состояний).

Конкатенация - объединение двух битовых или строковых последовательностей.

Константа - числовое значение, которое не может быть изменено в программе.

Кросс-ассемблер - инструментальное программное средство, позволяющее производить ассемблирование исходного текста программы, написанной для другой аппаратной платформы. *См.* Ассемблер.

Кросс-компилятор - инструментальное программное средство, позволяющее производить компиляцию исходного текста программы, написанной для другой аппаратной платформы. *См.* Компилятор.

Кулон - единица электрического заряда: 1 Кл = 1 А × 1 с. *См.* Ампер.

Логический анализатор - инструмент для проверки работы логических схем.

Логический вентиль (логический элемент) - электронная схема, используемая для выполнения логических операций.

Логический пробник - инструмент, позволяющий узнать уровень напряжения в заданной точке схемы.

Масса (земля, ground) - электрическая линия, относительно которой измеряются все напряжения в схеме.

Массив - упорядоченный набор однотипных переменных, доступ к которым производится с помощью номера (индекса).

Машинная команда — операция, входящая в систему команд микропроцессора (микроконтроллера).

Метка (label) - символьный идентификатор, используемый для адресации участка памяти или машинной команды.

Мультиплексор - логическая микросхема, используемая для передачи одного из входных сигналов на единственный выход. *См.* Демультиплексор.

Нечеткая логика (fuzzy logic) - логика, оперирующая «размытыми», нечетко определенными понятиями, например *высокий, низкий, большой, малый.*

Объектный файл - результат работы компилятора (в том числе ассемблера); входной файл для компоновщика.

Открытый коллектор (сток) - выход, который необходимо соединять с положительной шиной питания при помощи внешнего резистора. Выходы нескольких элементов с открытым коллектором (стоком) могут объединяться.

ОСРВ, операционная система реального времени - комплекс программных средств, предназначенных для поддержки выполнения программ в системах, где время реагирования на внешние воздействия строго ограничено.

Отладчик (debugger) - инструментальное программное средство, предназначенное для поиска ошибок в программах.

Отрицательная логика - соглашение, использующее 0 для обозначения высокого уровня напряжения (или активного состояния сигнала) и 1 — для низкого (или неактивного состояния).

Пассивные элементы - резисторы, конденсаторы, катушки индуктивности, диоды и другие элементы электронных схем, для функционирования которых не требуется внешний источник энергии.

Период сигнала - длительность одного полного колебания; величина, обратная частоте сигнала. Измеряется в секундах.

ПИД-регулятор - автоматический регулятор, выходной сигнал которого является суммой трех составляющих: пропорциональной, интегральной и дифференциальной.

Позиционная система счисления - метод представления числовых величин, в котором значение каждой цифры зависит от ее позиции в последовательности цифр, используемых для записи числа.

Положительная логика - соглашение, использующее 1 для обозначения высокого уровня напряжения (или активного состояния сигнала) и 0 - для низкого (или неактивного состояния).

Преобразователь уровней - устройство, используемое для сопряжения логических сигналов микросхем различных семейств (например, из ТТЛ в ЭСЛ или наоборот).

Прерывание - временная приостановка процесса выполнения основной программы и переход на специальную процедуру обработки прерывания.

Программирование микроконтроллера - загрузка программы в микроконтроллер с помощью специального программатора.

Программный опрос **(поллинг)** - периодический опрос состояния внешнего устройства в программном цикле.

Программный счетчик (PC, Program Counter) - специальный регистр микропроцессора, в котором хранится адрес выполняемой команды.

Редактор — приложение, используемое для ввода и изменения (редактирования) исходного текста программ. Обычно редакторы являются частью интегрированной среды разработки программ.

Рекурсия - вызов подпрограммы из этой же подпрограммы.

Светодиод (LED, Light-Emitted Diode) - диод, который при прохождении тока излучает электромагнитные волны (необязательно видимого диапазона).

Сила электрического тока - измеряется числом заряженных частиц, проходящих через поперечное сечение проводника в единицу времени. Единица измерения силы тока - ампер *(см.)*.

Симулятор - программное средство, предназначенное для моделирования выполнения программ без использования настоящего микроконтроллера.

СППЗУ, стираемое программируемое постоянное запоминающее устройство (EPROM, Erased Programmable Read Only Memory) - энергонезависимая микросхема памяти, содержимое которой может быть стерто и перезаписано.

Стек - *см.* LIFO.

Сторожевой таймер - используется для защиты программного обеспечения от зависания. Если вовремя не выполнится команда сброса сторожевого таймера, это означает, что программа зависла; в результате будет сформирован сигнал сброса микроконтроллера.

Тактовый генератор - используется для генерации импульсов, которые управляют работой микропроцессора (микроконтроллера).

Триггер – логическое устройство, используемое для хранения одного бита информации.

Указатель (pointer) - переменная, ссылающаяся на другую переменную.

Функция - подпрограмма, возвращающая результат своего выполнения в вызвавшую ее программу.

ЦАП, цифро-аналоговый преобразователь (DAC, Digital-to-Analog Converter) - устройство, предназначенное для преобразования цифрового кода на входе в аналоговое напряжение на выходе. *См.* АЦП.

Цифровой сигнал - сигнал, который может принимать значения из заданного набора допустимых уровней. Часто имеется всего два разрешенных уровня, условно обозначаемые 0 и 1.

Частота сигнала — количество полных колебаний в единицу времени. Измеряется в герцах: 1 Гц = 1 $с^{-1}$.

Шестнадцатеричное число - число, представленное в позиционной системе счисления по основанию 16. Для отображения цифр в шестнадцатеричной системе счисления используются десятичные цифры от 0 до 9 и латинские буквы от A до F.

ШИМ, широтно-импульсная модуляция - представление информации в виде импульсов одинаковой амплитуды и периода, но различной длительности.

Шина - набор линий для передачи информации.

Эмулятор - устройство, имитирующее поведение микроконтроллера в целевой схеме. Используется для отладки приложений.

Язык ассемблера - низкоуровневый язык программирования, ориентированный на систему команд конкретного процессора.

AND (И) - логическая операция, результатом которой является 1, только если все операнды равны 1.

ANSI, American National Standard Institute - Американский национальный институт стандартов.

ASCII, American Standard Character Interchange Interface - стандартный код для представления символьной информации, в котором каждый символ кодируется одним байтом.

ASCIIZ - строка символов в коде ASCII, заканчивающаяся нулевым байтом.

BCD, Binary Code Decimal (двоично-десятичный код) - код для представления числовых значений, в котором каждые четыре бита двоичного числа кодируются десятичными цифрами.

BOR, Brown-Out Reset - детектор понижения напряжения питания; схема, которая формирует сигнал сброса микроконтроллера, если его питание понижается до определенного значения.

CCP, Capture/Compare/Pulse Width Modulation (PWM) - модуль захвата, сравнения и широтно-импульсной модуляции (ШИМ).

CMOS, Complementary Metal-Oxide Semiconductor - комплементарная металл-окисел-полупроводниковая (КМОП) структура.

DIP, Dual In-line Package — корпус микросхемы с двухрядным расположением выводов.

EPROM, Erasable Programmable Read Only Memory - стираемое перепрограммируемое запоминающее устройство, СППЗУ *(см.)*.

FIFO, First In - First Out (первым вошел - первым вышел) - набор данных, организованный в виде очереди, чтение из которого производится в том же порядке, что и запись.

Flash - тип СППЗУ *(см.)* с поблочным электрическим стиранием.

I2C - протокол последовательного обмена информацией.

ISP, In-System Programming - *см.* Внутрисхемное программирование.

LCD - *см.* ЖКИ.

LED - *см.* Светодиод.

LIFO, Last In - First Out (последним вошел - первым вышел) - набор данных, организованный в виде стека, чтение из которого производится в порядке, обратном тому, в каком производилась запись.

MCU - микроконтроллер.

NOT (НЕ, отрицание) - одноместная логическая операция (или логический элемент с одним входом), инвертирующая логический сигнал.

OR (ИЛИ) - логическая операция, результатом которой является 0, только если все операнды равны 0.

OTP, One-Time Programmable - микросхема, поддерживающая возможность только однократного программирования.

POP - извлечение значения с вершины стека.

PUSH - запись значения в вершину стека.

PWM, Pulse Width Modulation - *см.* ШИМ.

RTOS, Real Time Operating System - *см.* ОСРВ.

WDT, WatchDog Timer - *см.* Сторожевой таймер.

XOR (Исключающее ИЛИ) - двуместная логическая операция (элемент с двумя входами), результатом которой является 0, только если оба операнда равны друг другу.

ZIF, Zero Insertion Force - разъем с «нулевым» усилием; предохраняет выводы микросхемы от повреждений при монтаже/демонтаже.

ПРИЛОЖЕНИЕ 2

СПРАВОЧНЫЕ ДАННЫЕ

Физические константы

В табл. П2.1 приведены значения часто используемых физических констант.

Таблица П2.1. Физические константы

Обозначение	Значение константы	Описание
а.е.	$1{,}4959787\times10^{11}$ м	Астрономическая единица (расстояние от Солнца до Земли)
c	$2{,}99792458\times10^{8}$ м/с	Скорость света в вакууме
e	2,7182818285	Основание натуральных логарифмов
ж	3,1415926535898	Число «пи» (отношение длины окружности к ее диаметру)
ε_0	$8{,}854\ 18782\times10^{-12}$ Ф/м	Электрическая постоянная
μ_0	$1{,}256637\times10^{-6}$ Гн/м	Магнитная постоянная
g	9,80665 м/с^2	Ускорение свободного падения
h	$6{,}626\,176\times10^{-34}$ Дж/Гц	Постоянная Планка
k	$1{,}380662\times10^{-23}$ Дж/К	Постоянная Больцмана
m_e	$0{,}910953\times10^{-30}$ кг	Масса покоя электрона
m_n	$1{,}674954\times10^{-27}$ кг	Масса покоя нейтрона
m_p	$1{,}672648\times10^{-27}$ кг	Масса покоя протона
K	8,314 Дж/(моль×К)	Универсальная газовая постоянная
v_s	331,45 м/с	Скорость звука в сухом воздухе на уровне моря при 20 °С
	1480 м/с	Скорость звука в чистой воде при 20 °С

Музыкальный звукоряд

В табл. П2.2 приведены частоты звуковых колебаний, соответствующих музыкальным нотам. Заметьте, что частота нот одной октавы отличается от следующей ровно в два раза.

Таблица П2.2. Музыкальный звукоряд

Нота	Обозначение	Частота, Гц
Соль	G	392
Соль-диез	G#	415,3

Таблица П2.2. Музыкальный звукоряд (окончание)

Нота	Обозначение	Частота, Гц
Ля	A	440
Си	B	493,9
До	C	523,3
До-диез	C#	554,4
Ре	D	587,3
Ре-диез	D#	622,3
Ми	E	659,3
Фа	F	698,5
Фа-диез	F#	740,0
Соль	G	784,0
Соль-диез	G#	830,6
Ля	A	880,0
Ля-диез	A#	932,3
Си	B	987,8

Частоты тонального набора телефонных аппаратов

В табл. П2.3 указаны частоты, используемые в тоновых номеронабирателях.

Таблица П2.3. Частоты тонального набора телефонных аппаратов

Частота, Гц	720?	1336	1477
697	1	2	3
770	4	5	6
852	7	8	9
941	*	0	#

Формулы электротехники

В данной книге используются обозначения:

V - напряжение;
I - сила тока;
R - сопротивление;
C - емкость;
L - индуктивность.

Закон Ома

$$V = IR.$$

Мощность

$P = VI.$

Последовательное соединение сопротивлений

$R_{общ} = R1 + R2 + ...$

Параллельное соединение сопротивлений

$R_{общ} = 1 / ((1 / R1) + (1 / R2) + ...).$

Два резистора, включенные параллельно

$R_{общ} = (R1 \times R2) / (R1 + R2).$

Последовательное соединение емкостей

$C_{общ} = 1 / ((1 / C1) + (1 / C2) + ...).$

Параллельное соединение емкостей

$C_{общ} = C1 + C2 + ...$

Мост Уитстона

При выполнении условия

$Ru = Rl \times R3 / R2$

(условие баланса моста) ток через измерительный прибор не течет.

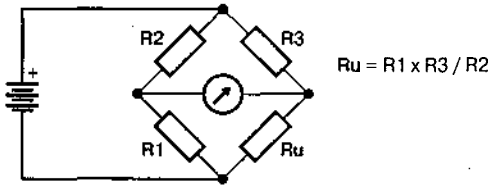

Рис. П2. Т. Мост Уитстона

Мощность

= VI.

Последовательное соединение сопротивлений

- R1 + R2 + ...

Параллельное соединение сопротивлений

= 1 / ((1 / R1) + (1 / R2) + ...).

Два резистора, включенные параллельно

(R1 x *R2) / (Rl + R2).*

Последовательное соединение емкостей

l / ((1 / C1) + (1 / C2) + ...).

Параллельное соединение емкостей

1 + C2 + ...

Мост Уитстона

ении условия

: R3/R2

анса моста) ток через измерительный прибор не течет.

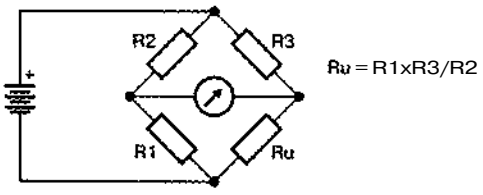

Рис. П2.1. *Мост Уитстона*

t - RC.

I

t - RC.

П

t - L/R.

Зар

$V(t) = V_0(1 - e^{-}$

$I(t) = I_0(1 - e^{-t/}$

Разря

$V(t) = V_0 e^{-t/}$

$I(t) = I_0 e^{-t/}$

Транс

$k = N_1 / N_2$,

где N_1 и N_2 – число вит

коэффициент трансфор

$U_2 = U_1 / k$,

а ток вторичной обмотки

$I_2 = I_1 k$.

Частота

Частота = Скорость/*Д*

Уравнение

PV - nRT.

Резонанс в LC-контуре

$t = RC$.

Постоянная времени RC-цепочки

$t = RC$.

Постоянная времени RL-цепочки

$t = L / R$.

Заряд конденсатора через резистор

$V(t) = V_0 (1 - e^{-t/\tau})$;

$I(t) = I_0 (1 - e^{-t/\tau})$.

Разряд конденсатора через резистор

$V(t) = V_0 e^{-t/\tau}$

$I(t) = I_0 e^{-t/\tau}$

Трансформатор тока/напряжения

$k = N_1 / N_2$,

где N_1 и N_2 - число витков первичной и вторичной обмоток соответственно; k – коэффициент трансформации. Напряжение на вторичной обмотке

$U_2 = U_1 / k$,

а ток вторичной обмотки

$I_2 = I_1 k$.

Частота и длина волны колебаний

Частота = Скорость / Длина волны.

Уравнение состояния идеального газа

$PV = nRT$.

Булева алгебра

A AND 1 = A
A AND 0 = 0;
A = A;
A AND NOT(A) = 0;
A AND A = A

A OR 0 = A
A OR 1 = 1;
NOT (NOT (A)) = A
A OR NOT (A) = 1;
A OR A = A

Коммуникативный закон

A AND B = B AND A A OR B = B OR A

Ассоциативный закон

(A AND B) AND C = A AND (B AND C) - A AND B AND C;
(A OR B) OR C = A OR (B OR C) - A OR B OR C.

Дистрибутивный закон

A AND (B OR C) - (A AND B) OR (A AND C);
A OR (B AND C) = (A OR B) AND (A OR C).

Теорема Моргана (соотношения взаимности)

NOT (A OR B) = NOT (A) AND NOT (B);
NOT (A AND B) - NOT (A) OR NOT (B).

Замечание. Логическая операция И часто обозначается знаком умножения, а логическое ИЛИ - знаком «плюс».

Соотношения единиц измерений физических величин

1 дюйм = 2,54 см
1 миля = 1,609 км
1 унция = 29,57 г
1 американский галлон = 3,78 л
1 атм. = 29,9213 мм рт. ст. = 14,696 фунта на кв. дюйм = 101,325 кПа

10 000 000 000 ангстрем = 1 м
1 ч = 3600 с
1 год = 8760 ч

Кратные и дольные приставки

Тера = 1000 гига
Гига = 1000 мега
Мега = 1000 кило
кило = 1000
1 = 100 санти
1 = 1000 милли

Кодовая таблица ASCII

В табл. П2.4 приведены ASCII-коды символов.

Таблица П2.4. Кодовая таблица ASCII

Биты 6–4	Управляющие символы		Печатные символы					
Биты 3-0	000	001	010	011	100	101	110	111
0000	NUL	DLE	Пробел	0	@	P	`	p
0001	SOH	DC/	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
ООП	ETX	DC3	#	3	C	S	c	s
0100	EVT	DC4	$	4	D	T	d	t
0101	ENO	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	CS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

ПРИЛОЖЕНИЕ 3

Сравнительная характеристика микроконтроллеров

В табл. П3.1 приведены основные сведения о микроконтроллерах, чаще всего используемых в автоматических устройствах.

Таблица ПЗ. 1. Основные параметры микроконтроллеров

Микро-контроллер	Кол-во выводов	Объем памяти программ, инструкций	Размер памяти данных, байт	Среда разработки	Программатор	Комментарии
Microchip PIC12C5xx	6	512байт – 1Кб	25-41	Microchip MPLAB	Последовательный ICSP (может быть встроенным)	Используется для периферийных контроллеров. Имеет встроенный генератор 4 МГц
PIC16C505	12	512 байт	72	Microchip MPLAB	ICSP	Для периферийных контроллеров или небольших автоматических устройств. Имеет встроенный генератор 4 МГц
PIC16HV540	12	512 байт	25	Microchip MPLAB	Параллельный	Имеет встроенный регулятор напряжения
PIC16C62x	13	512 байт – 2 Кб	80-96	Microchip MPLAB	ICSP	Имеет встроенный компаратор напряжения
PICI6F628	16	2 Кб	224	Microchip MPLAB	ICSP	Имеет встроенный генератор 4 МГц
PIC16F84(AI	13	1 Кб	68	Microchip MPLAB	ICSP	Имеет Flash-память
PIC16C71x	13	512 байт – 2 Кб	36-128	Microchip MPLAB	ICSP	Содержит четыре АЦП
PICJ6F87x	22, 33	8 Кб	368	Microchip MPLAB	ICSP	Имеет встроенный АЦП, Flash-память, отладчик (ICD)
PIC17C4x	33	8 Кб	232-454	Microchip MPLAB	Параллельный	Может работать с внешней 8/16-разрядной шиной
PIC18Fxx2	23-68	16 Кб	768-3840	Microchip MPLAB	ICSP	Встроенный АЦП, Flash-память, и внутренний аппаратный отладчик (ИС)
Parallax BASIC Stamp 1	8	256 байт	14	Parallax stampw	Параллельный порт	Flash-память программ, PBASIC
BASIC Stamp 2	16 + 2 I/O	2 Кб	26	Parallax stampw	RS-232	Flash-память программ, PBASIC
BASIC Stamp 2e	16 + 2 I/O	16 Кб	26 + 63	Parallax stampw	RS-232	Расширенный PBASIC
BASIC Stamp 2sx	16 + 2 I/O	16 Кб	26 + 63	Parallax stampw	RS-232	Расширенный PBASIC/высокоскоростной микроконтроллер

Таблица П3. 1. Основные параметры микроконтроллеров (окончание)

Микро-контроллер	Кол-во выводов	Объем памяти программ, инструкций	Размер памяти данных, байт	Среда разработки	Программатор	Комментарии
BASIC Stamp 2p	16/30 + I/O	16 Кб	26 + 63	Parallax stampw	RS-232	Расширенный PBASIC
Motorola 68HC08	13–51	0-32Кб	128 – 1 Кб			Множество дополнительных функций. Необязательная внешняя программная память
Motorola 68HC11	13–50	0-32Кб	64 – 2Кб			Много дополнительных функций. Необязательная внешняя программная память
Handy (на основе 68HC11)	20	32 Кб	32 Кб		RS-232	Специально для разработки роботов
OOPIC2	40	95 объектов	172	OOPic	I2C	Имеет различные интерфейсы
Atmel AT90S1200 (AVR)	15	1 Кб	64	AVR Studio	Последовательный или параллельный	Для подсистем ввода-вывода
AT90S8515 (AVR)	32	8 Кб	12	AVR Studio	Параллельный	Имеет компараторы и Flash-память
AT90S8535 (AVR)	32	8 Кб	512	AYR Studio	Параллельный	Имеет АЦП и Flash-память
Intel 87C51 и др.	32	4 Кб	128			Микроконтроллер i8051 с СППЗУ, последовательный порт, возможность доступа к внешней 8-разрядной памяти
87C52	32	8 Кб	256			Микроконтроллер i8052 с СППЗУ, последовательный порт, возможность доступа к внешней 8-разрядной памяти
Atmel AT89Cx051 (8051)	15	1-4 Кб	128			Микроконтроллер 8051 с встроенным компаратором напряжения и USART
Kg Systems BASIC-Tiger	38	128 Кб	128 Кб	В ASIC-Tiger	Последовательный	Аналоговый ввод-вывод, I2C, последовательный интерфейс, ЖКИ. Поддержка многозадачности
PC/104					Последовательный, Flash, СППЗУ	Шина ввода-вывода, совместимая с ПЭВМ

ПРИЛОЖЕНИЕ 4

Справочное руководство по языку PICC Lite

Представление констант

В табл. П4.1 приведены обозначения констант, представленных в различных системах счисления.

Таблица П4. /. Представление констант в различных системах счисления

Основание системы счисления	*Формат PICC Lite*	*Формат MPASM*
2	0bчисло *или* OBчисло	B'число'
8	Очисло *или* \число	O'число'
10	число	число *или* D'число' *или* .'число'
16	Охчисло *или* OXчисло	H'число' *или* Охчисло

Комментарии

Два способа выделения комментариев:

1. Двойной слэш:

```
// Все символы до конца строки считаются комментарием.
```

2. Слэш со звездочкой:

```
/* Все символы здесь считаются комментарием. */
```

Объявления переменных и констант

Объявление переменной или константы представлено в следующем формате:

```
[квалификатор] тип [* [квалификатор]] метка [= значение];
```

В квадратных скобках, как обычно, приведены необязательные элементы. В табл. П4.2 перечислены допустимые типы данных.

Таблица П4.2. Типы данных в PICC Lite

Тип	Разрядность, бит	Комментарии
bit	1	Логическое значение
char	8	Целое со знаком
unsigned char	8	Целое без знака
short	16	Целое со знаком
unsigned short	16	Целое без знака
int	16	Целое со знаком
unsigned int	16	Целое без знака
long	32	Целое со знаком
unsigned long	32	Целое без знака
float	24	Вещественное с плавающей точкой
double	24 (по умолчанию)	Вещественное с плавающей точкой (разрядность можно увеличить до 32 с помощью опции компилятора -D32J

Для явного приведения типа используется конструкция вида

(unsigned long) `StringAddr = 0x012345678;`

Описание массива переменных:

тип ИмяМассива [*КоличествоЭлементов*] [= {*значение1, значение2, ...*}]

Строковые переменные объявляются как массивы символов и хранятся в формате ASCIIZ:

```
char ИмяСтроки [МаксимальноеЧислоСимволов] = "Это строка";
```

Последний символ NUL с кодом 0x00 при объявлении строки вставляется автоматически, но программисту приходится помнить о нем, если создание и обработка строк происходят в самой программе.

Для обращения к отдельному символу строковой переменной предусмотрена та же нотация, что и для элементов массива:

```
b = String[n];          // Выбрали n-й символ строки.
```

Для объявления указателя используется символ * (звездочка):

```
char * String = "Это строка";
```

В микроконтроллере PIC16F84 переменные могут располагаться в первом банке памяти по адресам с OxOC до Ox4F, что соответствует абсолютным адресам с 0x08C до OxOCF; они могут быть отображены в нулевой банк памяти. В микроконтроллере PIC16F627 переменные хранятся как в нулевом, так и в первом банке памяти; отображение адресов между банками невозможно.

В языке PICC Lite, в отличие от стандартного С, нельзя использовать символ & для того, чтобы присвоить указателю адрес другой переменной.

Обращение к конкретным адресам регистров в языке PICC Lite представлено в следующем формате:

```
static unsigned char ИмяРегистра @ 0xАдресРегистра;
```

Для обращения к отдельным битам регистра служит запись вида

```
static bit ИмяБита @ (unsigned) &ИмяРегистра * 8 + НомерБита;
```

Если подключить к программе заголовочный файл pic.h, то можно использовать стандартные имена регистров и битов.

Чтобы сообщить компилятору, что другая программа (скажем, процедура обработки прерываний) может в любой момент изменить значение некоторой переменной, при ее описании надо указать квалификатор `volatile`, например:

```
volatile unsigned ИмяРегистра @ 0xАдресРегистра;
```

Для описания констант используется квалификатор const, например:

```
const int Label = Значение;
```

В результате переменная `Label1` будет размещена компилятором в ПЗУ, и ее значение нельзя будет изменить.

Объявление вида

```
const char * ptr;
```

создает указатель ptr на символ. Значение указателя может быть изменено, но сам символ - нет.

Наоборот, если записать

```
char * const ptr;
```

то можно будет изменить значение символьной переменной, но не указателя.

При использовании микроконтроллера PIC16F627 все объявленные в программе переменные размещаются в нулевом банке памяти, но можно использовать квалификатор `bank1`, если требуется хранить переменную в первом банке. Эта возможность недоступна для микроконтроллера PIC16F84.

Структуры и объединения

Формат описания структуры или объединения:

```
struct | union {
    type ИмяПоля1;
    type ИмяПоля2;
    ...
} ИмяОбъединения [ИмяПеременной];
```

Для доступа к полям структуры используется оператор ->. Для примера рассмотрим следующее определение:

```
struct <
    intdividend;          // Частное.
    intremainder;         // Остаток.
} div_result result;      // Результат деления.
```

Тогда можно записать:

```
result -> dividend  = x / y;
result -> remainder = x % y;
```

Приведем пример описания битовой структуры:

```
struct {
    unsigned : 3;          // Старшие три бита не будут использоваться.
    PageSel  : 2;          // Выбор номера страницы.
    OffSel   : 3;          // Смещение внутри страницы.
} PCLATH @ 0x00A;
```

Функции

Главная программа оформляется следующим образом:

```
void main(void) {
    ...                    // Текст программы.
>                          // Конец программы.
```

Формат описания функции:

```
ТипРезультата ИмяФункции ( [ Тип1 Параметр1 [, Тип2 Параметр2,...] ] ) {
    ...                    // Текст функции.
    return value;          // Возвращаемое значение.
}                          // Конец функции.
```

Для описания обработчика прерываний в заголовке функции надо указать модификатор interrupt:

```
void interrupt ИмяФункции (void) {
    ...
}
```

Операторы

В общем виде формат оператора языка PICC Lite выглядит следующим образом:

```
[(..] Переменная \ Константа [ Операция [(..] Переменная \ Константа ] [)..] ]
```

Оператор присваивания значения переменной:

```
Переменная - Выражение;
```

К управляющим конструкциям относятся следующие: if, while, do, for, switch.

Формат оператора if:

```
if ( Выражение )
    [ Оператор ]; | { Оператор; ... }
[ else [ Оператор ]; | { Оператор; ... } ]
```

Формат цикла while:

```
while ( Выражение ) [ Оператор ]; | { Оператор; ... }
```

Цикл for описывается так:

```
for ( [ Инициализация;                  // Инициализация параметра цикла.
    [ Условие [, Условие ]];            // Условие продолжения цикла.
    [ Выражение [, Выражение,...]])     // Изменение параметра цикла.
    Оператор \ { Оператор; ... }        // Тело цикла.
```

Для немедленного выхода из цикла можно использовать оператор break.

Команда continue пропускает ту часть тела цикла, которая расположена после нее, и переходит к следующему шагу. Эту команду можно указывать в циклах for, while и do/while.

Формат цикла с постусловием (он выполняется, пока условие истинно):

```
do Оператор; [ { Оператор; ... } while ( Условие );
```

Оператор выбора:

```
switch( Выражение ) {
    case Значение1: [ Оператор; ... ] [ break; ]
    case Значение2: [ Оператор; ... ] [ break; ]
    ...
    default: [ Оператор; ... ]
}
```

Инструкция

```
goto Метка
```

используется для перехода на указанный оператор (метка не может быть расположена вне тела функции). Формат описания метки:

```
Метка: Оператор;
```

Для возвращения результата выполнения функции используется оператор return:

```
return Выражение;
```

Операции

В табл. П4.3 приведены одноместные (унарные), в табл. П4.4 - двуместные (бинарные), а в табл. П4.5 - составные операции.

Таблица П4.3. Унарные операции

Обозначение	Операция
~	Поразрядная инверсия
++	Инкремент (увеличение на 1)
-	Изменение знака числа
--	Декремент (уменьшение на 1)

Таблица П4.4. Бинарные операции

Обозначение	Операция
!	Логическое отрицание
&&	Логическое И (AND)
&	Поразрядное И (AND) или ссылка
||	Логическое ИЛИ (OR)
|	Поразрядное ИЛИ (OR)
^	Поразрядное Исключающее ИЛИ (XOR)
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления
==	Сравнение на равенство
!=	Сравнение на неравенство
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
<<	Сдвиг влево
>>	Сдвиг вправо

Таблица П4.5. Составные операции

Символ	Код ASCII	Описание
\r	0x00D	Возврат каретки (CR)
\n	0x0 OA	Перевод строки (LF)
\f	0x00C	Перевод страницы (FF)
\b	0x008	Забой (BS)
\t	0x009	Горизонтальная табуляция (HT)
\v	0x00B	Вертикальная табуляция (VT)
\a	0x007	Звонок (BEL)
\'	0x027	Апостроф (')
\"	0x022	Двойная кавычка (")
\\	0x05C	Обратный слэш (\)
\ddd	–	Восьмеричный код
\xddd	0xOdd	Шестнадцатеричный код

Зарезервированные слова

Ниже приведен список зарезервированных слов, которые не могут быть использованы в качестве имен переменных, констант или меток:

```
auto                    if
bank1                   int
break                   interrupt
case                    persistent
char                    return
const                   signed
continue                static
default                 struct
do                      switch
else                    union
extern                  unsigned
fastcall                void
for                     volatile
goto                    while
```

Управляющие символы

В табл. П4.6 приведены специальные символы для управления экраном и положением курсора.

Таблица П4.6. Управляющие символы

Символ	*Код ASCII*	*Описание*
\r	0x00D	*Возврат каретки (CR)*
\n	0x00A	*Перевод строки (LF)*
\f	0x00C	*Перевод страницы (FF)*
\b	0x008	*Забой (BS)*
\t	0x009	*Горизонтальная табуляция (HT)*
\v	0x00B	*Вертикальная табуляция (VT)*
\a	0x007	*Звонок (BEL)*
\'	0x027	*Апостроф (')*
\"	0x022	*Двойная кавычка (")*
\\	0x05C	*Обратный слэш (\)*
\ddd	–	*Восьмеричный код*
\xddd	0x0dd	*Шестнадцатеричный код*

Директивы компилятора

В табл. П4.7 приведены директивы управления компиляцией. Все директивы начинаются с символа # и выполняются при компиляции программы.

Таблица П4.7. Директивы компилятора

Директива	Функция
#	Пустая директива
#asm . . . #endasm	Ассемблерная вставка
#assert Условие	Генерация ошибки, если условие ложно
#define Имя [(Параметры)] Текст	Определение идентификатора (имени), каждое вхождение которого в текст программы компилятор заменит указанным текстом
#include "Файл" | <Файл>	Включение в текст программы содержимого указанного файла. Если имя файла приведено в кавычках, то он ищется в текущем каталоге и во всех каталогах, которые указаны в переменной окружения PATH, о если в угловых скобках, то файл ищется в каталогах, указанных в переменной окружения INCLUDE
#error Сообщение	Генерация ошибки времени компиляции с выводом на экран указанного сообщения
#if Условие #elif,#else,или #endif.	Если условие истинно, компилируется код, следующий до директивы Если условие ложно, код до #elif, #else или #endif игнорируется
#ifdef Имя #endif,	Если указанное имя определено в программе с помощью директивы #define, то код, следующий до директивы #elif, #else или компилируется
#ifndef Имя #else	Если указанное имя не определено в программе с помощью директивы #define, то код, следующий до директивы #elif, или #endif, компилируется
#elif Условие	Эта директива работает как #else #if и позволяет избежать вложенных директив #if
#else	Используется совместно с директивами #if или #elif
#endif #ifdef	Используется для завершения директив #if, #elif, #else, или #ifndef
#line Номер ИмяФайла	Определяет номер строки и имя файла листинга
#number	Определяет номер строки файла листинга
«pragma interrupt_level 1	Следующая функция обработки прерываний будет вызывать другую функцию. Позволяет подавить сообщение об ошибке, которое в этом случае генерируется по умолчанию
#pragma jis	Разрешает строки JIS (двухбайтовый национальный набор символов)
#pragma nojis	Запрещает строки JIS
#pragma printf_check тип	Определяет, что следующая функция должна принимать строковые данные в формате printf
#pragma psect ИмяСегмента = Имя	Определяет, что данные будут помещены в определенный сегмент памяти
#pragma regused	
Регистр ...	Определяет регистры, которые будут сохранены в процедуре обработки прерывания
#undef Имя	Отменяет определение указанного имени
#warning Сообщение	Выводит предупреждающее сообщение во время компиляции

Параметры командной строки

В табл. П4.8 приведены параметры, которые можно указывать в командной строке при вызове компилятора PICC Lite.

Таблица П4.8. Параметры командной строки компилятора

Параметр	Значение
-processor	Определяет тип процессора (можно указывать только 16C84, 16F84, 16F84A или J6F627)
-A-опция	Передает параметр -опция ассемблеру
-AAHEX	Генерирует HEX-файл в символьном формате American Automation
-ASMLIST	Генерирует файл .LST
-BIN	Генерируется двоичный выходной файл
-C	Компиляция будет закончена созданием объектных файлов
-CRфайл	Генерирует файл перекрестных ссылок
-D24	Использует усеченный, 24-разрядный формат вещественных чисел
-D32	Использует полный (стандарт IEEE 754), 32-разрядный формат вещественных чисел
-Dмакрос	Определяет макрокоманду препроцессора (как директива #define)
-E	Определяет расширенный (символьный) формат для ошибок компилятора
-Eфайл	Перенаправляет сообщения об ошибках компилятора в указанный файл
-E+файл	Сообщения об ошибках дописываются в конец существующего файла
-FAKELOCAL	Задает формат MPLAB для отладочной информации
-FDOUBLE	Разрешает использовать функции специальной библиотеки для работы с 32-разрядными вещественными числами
-Gфайл	Генерирует расширенную таблицу имен
-HELP	Выводит подсказку по параметрам командной строки
-ICD	Генерирует код для внутрисхемного отладчика
-Iпуть	Определяет путь к каталогу с подключаемыми файлами
-INTEL	Генерирует выходной файл в шестнадцатеричном формате Intel (по умолчанию)
-Lбиблиотека	Определяет библиотеку, которая будет использоваться компоновщиком
-L-опция	Определяет параметр -опция, который будет использоваться компоновщиком
-Мфайл	Запрашивает создание MAP файла
-MOT	Генерирует выходной файл в шестнадцатеричном формате Motorola S1/S9
-Nчисло	Определяет размер идентификаторов (по умолчанию — 31 символ)
-NORT	Запрещает компоновщику подключать к приложению стандартные модули времени выполнения
-O	Разрешает оптимизацию кода
-Офайл	Определяет имя выходного файла
-P	Запускает препроцессор для обработки исходного текста программы
-PRE	Создает файлы с результатом работы препроцессора
-PROTO	Генерирует информацию о прототипах функций
-PSECTMAP	После компоновки показывает карту памяти
-Q	Определяет режим без вывода сообщений компиляции (данный параметр должен указываться первым)

Таблица П4.8. Параметры командной строки компилятора (окончание)

Параметр	Значение
-RESRAMдиапазон	Резервирует диапазон указанных адресов ОЗУ
-RESROMдиапазон	Резервирует диапазон указанных адресов ПЗУ
-ROMдиапазон	Определяет адресный диапазон для внешнего ПЗУ
-S	Останавливает компиляцию после получения исходных ассемблерных файлов
-SIGNED_CHAR	Определяет для символьных переменных формат «целое со знаком» (если при описании переменной явно не указан другой квалификатор)
-STRICT	Разрешает компилятору использовать строгие соглашения ANSI. В этом случае в программе нельзя использовать битовые переменные
-TEK	Генерирует выходной файл в шестнадцатеричном формате Tektronix
-Uимя	Отменяет определение указанного идентификатора
-UBROF	Генерирует выходной файл в формате UBROF
-V	Показывает в листинге параметры командной строки, заданные при вызове компилятора
-Wчисло	Устанавливает уровень предупреждающих сообщений компилятора
-X	Исключает локальные символы из таблицы имен
-Zgчисло	Определяет уровень оптимизации (максимум 3)

Стандартные функции PICC Lite

В табл. П4.9 приведен список встроенных функций PICC Lite.

Таблица П4.9. Стандартные функции PICC Lite

Функция	Заголовочный файл	Комментарии
int abs(int)	stdlib.h	Вычисляет абсолютное значение числа
double acos(double)	math.h	Вычисляет арккосинус (ответ в радианах)
double asin(double)	math.h	Вычисляет арксинус (ответ в радианах)
double atari (double)	math.h	Вычисляет арктангенс (ответ в радианах)
double atan2 (double, double)	math.h	Вычисляет арккотангенс (ответ в радианах)
double atof (const char *)	math.h	Преобразует строку ASCII в вещественное число
int atoi(const char *)	stdlib.h	Преобразует строку ASCII в целое число
int atol(const char *)	stdlib.h	Преобразует строку ASCII в длинное целое число
double ceil(double)	math.h	Округляет число с плавающей запятой до ближайшего целого (в большую сторону)
char *cgets(char *)	conio.h	Получает строку из стандартного потока ввода
double cos(double)	math.h	Вычисляет косинус угла (в радианах)
double cosh(double)	math.h	Вычисляет гиперболический косинус
void cputs(char *)	conio.h	Выводит строку символов на экран
void eeprom_write(unsigned char, unsigned char)	pic1684.h	Записывает байт в СППЗУ
unsigned char eeprom_read (unsigned char)	pic1684.h	Считывает байт из СППЗУ

Таблица П4.9. Стандартные функции PICC Lite (продолжение)

Функция	Заголовочный файл	Комментарии
double exp(double)	math.h	Вычисляет экспоненту
double fabs(double)	math.h	Вычисляет абсолютное значение числа
double floor(double)	math.h	Округляет число с плавающей запятой до ближайшего целого (в меньшую сторону)
double frexp(double, int *)	math.h	Вычисляет целую степень вещественного числа
char getch(void)	conio.h	Получает один символ из стандарта потока ввода. Если входной поток пуст, ожидает появления символа
char getche(void)	conio.h	Получает двухбайтовый символ из стандартного потока ввода (для получения старшего байта требуется повторный вызов этой функции)
void init_uart(void)	conio.h	Инициализирует UART
Bit isalnum(char)	ctype.h	Возвращает TRUE, если символ является буквенно-цифровым
Bit isalpha(char)	ctype.h	Возвращает TRUE, если символ является буквой
bit iscntrl(char)	ctype.h	Возвращает TRUE, если символ управляющий (0x000 - 0x01F или 0x07F)
bit isdigit(char)	ctype.h	Возвращает TRUE, если символ является цифрой
bit isgraph(char)	ctype.h	Возвращает TRUE, если символ является печатным, но не пробелом (то есть его код лежит в диапазоне 0x021 ~ 0x07E)
bit islower(char)	ctype.h	Возвращает TRUE, если символ — буква нижнего регистра
bit ispunct(char)	ctype.h	Возвращает TRUE, если символ является знаком пунктуации
bit isprint(char)	ctype.h	Возвращает TRUE, если символ является печатным (0x020 – 0x07E)
bit isspace(char)	ctype.h	Возвращает TRUE, если код символа лежит в диапазоне 0x09 – 0x0D или является пробелом
bit i supper(char)	ctype.h	Возвращает TRUE, если символ — буква верхнего регистра
bit isxdigit(char)	ctype.h	Возвращает TRUE, если символ - шестнадцатеричная цифра
char kbhit(void)	conio.h	Возвращает код нажатой клавиши
double ldexp(double, int)	math.h	Вычисляет целую степень целого числа
double log(double)	math.h	Вычисляет натуральный логарифм
double log10(double)	math.h	Вычисляет десятичный логарифм
const void* memchr(const void *, int, unsigned int)	string.h	Ищет первое вхождение символа
int memcmp(const void*, const void *, unsigned int)	string.h	Сравнивает две строки
void *memcpy(void *, const void *, unsigned int)	string.h	Копирует заданное число байтов
void *memmove(void *, const void *, unsigned int)	string.h	Копирует заданное число байтов (диапазон адресов источника и адресата могут перекрываться)
void *memset(void *, int, unsigned int)	string.h	Заполняет заданное число ячеек памяти одним и тем же символом

Таблица П4.9. Стандартные функции PICC Lite (продолжение)

Функция	Заголовочный файл	Комментарии
double modf (double, double *)	math.h	Разбивает число с плавающей точкой на целую и дробную части
double pow (double, double)	math.h	Возводит число в степень
int printf (const char *,...)2	stdio.h	Выводит текстовые строки на экран. Поддерживает стандартные символы форматирования: %d, %i (десятичное число); %o (восьмеричное число); %x, %X (шестнадцатеричное число в нижнем или верхнем регистре); %u (целое число без знака); %c (текстовый символ в коде ASCII); %s (строка ASCIIZ); %f (число с плавающей точкой); %#e, %#E (число с плавающей точкой, вместо символа # указывается точность); %g, %G (число с плавающей точкой, формат выбирается SB зависимости от значения); %p (указатель); %% (символ %)
void putch(char)	conio.h	Выводит один символ в стандартный поток вывода
int rand(void)	stdlib.h	Возвращает случайное число - см. srand
double sin(double)	math.h	Вычисляет синус угла (в радианах)
double sinh(double)	math.h	Вычисляет гиперболический синус
double sqrt(double)	math.h	Вычисляет квадратный корень числа
void srand(void)	stdlib.h	Инициализирует генератор случайных чисел - см. rand
char* strcat (char *, const char *)	string.h	Конкатенация строк ASCIIZ
const char* strchr (const char *, int)	string.h	Находит заданный символ в строке ASCIIZ
int strcmp(const char *, const char *)	string.h	Сравнивает две строки ASCIIZ
char *strcpy (char *,const char *)	string.h	Копирует строку ASCIIZ
unsigned int strcspn (const char *, const char *)	string.h	Возвращает число символов, которые входят во вторую строку, но отсутствуют в первой
char *strdup(const char *)	string.h	Дублирует строку
int stricmp(const char *, const char *)	string.h	Сравнивает строки (без учета регистра символов)
const char *stristr (const char *, const char*)	string.h	Возвращает указатель на вхождение второй строки в первую (без учета регистра символов)
char *strncat (char *, const char *, unsigned int)	string.h	Добавляет в конец первой строки ASCIIZ заданное количество символов второй
int strncmp(const char *, const char *, unsigned int)	string.h	Сравнивает заданное количество символов двух строк ASCIIZ
int strnicmp(const char *, const char *, unsigned int)	string.h	Сравнивает заданное количество символов двух строк ASCIIZ (без учета регистра символов)
unsigned int strlen (const char *)	string.h	Возвращает длину строки
char *strncpy(char *, const char *, unsigned int)	string.h	Копирует заданное число символов из второй строки в первую
const char* strpbrk (const char *, const char *)	string.h	Возвращает первое вхождение в первую строку любого символа второй строки

Таблица П4.9. Стандартные функции PICC Lite (окончание)

Функция	*Заголовочный файл*	*Комментарии*
const char* strrchr (const char *, int)	*string.h*	*Возвращает позицию последнего символа в строке ASCIIZ*
unsigned int strspn (const char *,const char *)	*string.h*	*Сравнивает две строки и возвращает номер позиции первого несовпадающего символа*
const char *strstr const char *)(const char *, const char *)	*string.h*	*Возвращает указатель на вхождение второй строки в первую*
double tan(double)	*math.h*	*Вычисляет тангенс угла (в радианах)*
double tanh(double)	*math.h*	*Вычисляет гиперболический тангенс*
char tolower(char)	*ctype.h*	*Преобразует символ к нижнему регистру*
char toupper(char)	*ctype.h*	*Преобразует символ к верхнему регистру*
unsigned int xtoi (const char *)	*stdlib.h*	*Преобразует строку ASCII в шестнадцатеричное число*

ПРИЛОЖЕНИЕ 5

Информация об авторе

Свои пожелания отправляйте по электронному адресу myke@passport.ca Вы также можете посетить сайт автора в сети Internet (www.myke.com).

Microchip

Microchip Technology, Inc.
www.microchip.com
На сайте находится полная документация в формате PDF и последние версии MPLAB.

Hi-Tech Software

Hi-Tech Software
www.htsoft.com
На сайте представлена информация о последних версиях компиляторов для различных микроконтроллеров и микропроцессоров (в том числе и о компиляторе PICC Lite).

Компиляторы для микроконтроллеров PICmicro

Custom Computer Services, Inc.
www.ccsinfo.com
ccs@ccsinfo.com

PicBasic и PicBasic Pro

MicroEngineering Labs, Inc.
www.melabs.com
info@melabs.com

Компилятор CC5X

B Knudsen Data
www.bknd.com
sales@bknd.com

ЭЛЕКТРОННЫЕ РЕСУРСЫ ПО РОБОТОТЕХНИКЕ

Seattle Robotics Society

www.seattlerobotics.org
Один из наиболее известных сайтов в Internet.

Silicon Valley HomeBrew Robotics Club

www.wildrice.com/HBRobotics/HBRCBuildersBook.html
Превосходное введение в робототехнику. Представлено большое количество инструкций по проектированию роботов и ссылок на соответствующие ресурсы.

robots.net

http://robots.net
Хороший сайт. Имеются ссылки на литературу, список разработок.

The Robot Menu

www.robotics.com/robomenu/index.html
Сайт посвящен человекоподобным роботам.

Robotics FAQ List

www.frc.ri.cmu.edu/robotics-faq
Охватывает практически все, чем занимается робототехника.

Innovatus

www.innovatus.com
Полезное программное обеспечение для разработчика автоматических устройств.

RoboClub

www.roboclub.ru

Практическая робототехника (на русском языке). Роботы домашние, военные, боевые, мобильные; автономные устройства, конструкции, компоненты, схемы, датчики, сенсоры, детекторы, исполнительные устройства, проекты, поведение, обучение, искусственный интеллект, алгоритмы, программирование, руководства, документация, покупка и продажа комплектующих, вопросы, советы, обзоры, соревнования. Каталог ссылок на ресурсы Internet по робототехнике можно найти на страничке http://links.roboclub.ru.

Internet-робототехника

www.applmat.ru/pages/i-robotics/rus/rhome.html

Исследования по Internet-робототехнике были начаты в секторе робототехники ИПМ им. М. В. Келдыша РАН и в Группе компьютерной графики университета Де Монтфорт (Милтон Кейнс, Англия) как продолжение совместного проекта INTAS по космической робототехнике. Практическую цель представляла разработка системы управления через Internet роботом-манипулятором PM-01 (PUMA 560).

Периодические издания

Circuit Cellar Ink: www.circellar.com
Microcontroller Journal: www.mcjournal.com
Nuts and Volts: www.nutsvolts.com
Everyday Practical Electronics: www.epemag.wimborne.CQuk

Поставщики комплектующих

ООО «МИКРО-ЧИП»
sales@microchip.ru
www.microchip.ru. www.imicro.ru
Тел./факс: (095) 963-96-01
На сайте www.microchip.ru имеется документация по микроконтроллерам PICmicro в формате PDF на русском языке.

Консультационно-технический центр по микроконтроллерам (КТЦ-МК)
www.cec-mc.ru. www.gaw.ru
Тел.: (095) 973-18-55, 942-34-16
Факс: (095) 973-18-64
Магазин «ЧИП и ДИП»

sales@chip-dip.ru
www.chip-dip.ru
Москва, ул. Гиляровского, д. 39
Тел.: (095) 284-56-78, 281-99-17
Факс: (095) 971-31-45

НПО «СИММЕТРОН»
npo@symmetron.ru
www.symmetron.ru
Москва, ул. 8 Марта, д. 8
Тел.: (095) 214-25-55, 212-33-08
Санкт-Петербург, ул. Таллинская, д. 7
Тел.: (812) 278-84-84, 444-02-68

Сведения о других поставщиках вы можете найти в Internet по адресам www.catalog.gaw.ru. www.einfo.ru. www.efind.ru. www.chipinfo.ru.

СПИСОК ЛИТЕРАТУРЫ

1. Предко М. Руководство по микроконтроллерам. В 2-х тт. - М.: Постмаркет, 2001.
2. Предко М. Справочник по PIC-микроконтроллерам. — М.: ДМК Пресс, 2002.
3. Таверньe К. PIC-микроконтроллеры. Практика применения. - М.: ДМК Пресс, 2002.
4. Миль Г. Модели с дистанционным управлением. - Л.: Судостроение, 1984.
5. Мацкевич В. В. Занимательная анатомия роботов. Серия «Научно-популярная библиотека школьника». - М.: Радио и связь, 1988.
6. Хейзерман Д. Как самому сделать робота. - М.: Мир, 1979.
7. Янг Дж. Ф. Робототехника. - Л.: Машиностроение, 1979.
8. Красковский Е. Я., Дружинин Ю. А., Филатов Е. М. Расчет и конструирование механизмов приборов и вычислительных систем. Учебное пособие для приборостроительных специальностей вузов. — М.: Высшая школа, 1991.
9. Механика промышленных роботов. Учебное пособие для вузов: в 3 кн. / Под ред. К. В. Фролова, Е. И. Воробьева. - М.: Высшая школа, 1988.
10. Смольников Б. А. Проблемы механики и оптимизации роботов. - М.: Наука, 1991.
11. Попов Е. П., Ющенко А. С. Роботы и человек. - М.: Наука, 1984.
12. Юревич Е. И. Основы робототехники. — Л.: Машиностроение, 1985.
13. Новиков Ю. В., Калашников О. А., Гуляев С. Э. Разработка устройств сопряжения для персонального компьютера типа IBM PC. Практ. пособие. - М.: ЭКОМ, 1997.
14. Белоусов И. Р. Управление роботами через сеть Интернет / Новое в управлении и автоматике. - М.: Наука, 2002.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

А

Автомат конечный 336
Алгоритм генетический 352
Анализ ошибок 363
Арбитраж целевых установок 344
Архитектура
  гарвардская 57
  принстонская 57
Ассемблер 40
АЦП 20, 46, 264
АЧХ 281

Б

Банк памяти 90
Бит четности 114
Булева алгебра 376

В

Ввод аналогового сигнала 264
Внутреннее сопротивление
источника 22
Внутрисхемное
программирование 57, 120
Выбор периферийных устройств 358
Выражение 47
Выход
  с открытым коллектором (OK) 56
  с открытым стоком 91

Г

Гармоника 279
Гармонический анализ сигналов 280
Гидролокатор 252
Гистерезис 92
Глобальная переменная 136

Д

Дальномер ультразвуковой 252, 255
Датчик
  звуковой 279
  световой 263
Декодирование команд
дистанционного управления 245
Декремент 41
Делитель напряжения 265
  балансировка 322
Детерминированное
поведение 324
Децибел 281
Дисплей
жидкокристаллический 34, 195
Дифференциальное включение
фоторезисторов 266

З

Задание входных воздействий
(стимулов) 66
Закон
  ассоциативный 376
  дистрибутивный 376
  коммутативный 376
  Ома 373

И

Излучатель
пьезоэлектрический 191
Измерение
  расстояния 305
  сопротивления 272
Индикатор
  жидкокристаллический 195
  звуковой 190

Инкремент 41
Интерпретатор 37, 43
Искусственный интеллект 348
Испытания роботов 361

К

Квитирование 145
КМОП 26, 56
Код
  Manchester 238
  NRZ 56
Компаратор 117
  аналоговый 265
Компилятор 37, 46
Компиляция 40
Компоновщик 38, 50
Конечный автомат 132
Контакт
нормально замкнутый 19
Контроллер 18
  жидкокристаллического дисплея 195
  нечеткий 348
Контроллер прерываний 30
Конфигурационный регистр 87

М

Меандр 280
Методы принятия решений 324
Микроконтроллер 18
МК 18
Модели Tamiya 291
Модернизация устройств 364
Модификатор 136
Модуляция
широтно-импульсная 178
Монтаж
  накруткой 13
  печатный 13
Мост
  Н-образный 289
  Уитстона 374
Музыкальный звукоряд 372
Мысленный эксперимент 364

Н

Нейрон 350
НЗК 19
Низковольтное программирование 88
Номеронабиратель тоновый 373

О

Обратная польская нотация 48
Обучение нейронной сети 352
Объектный файл 38
Одометр 305
Октава 372
Операционная система
  многозадачная 325
  реального времени 325
ОС 91
ОСРВ 325
Отладка 363
Отладчик внутрисхемный 62

П

Память 26
  динамическая 28
  статическая 28
ПЗУ 26, 337
ПИД-регулятор 307
ПО 36
Поведенческое программирование 343
Порт ввода-вывода 32
Постскалер 33
Прерывание 29
Прерывания по изменении уровня входного сигнала 278
Прерыватель оптический 305
Прескалер 33
Приемник команд дистанционного управления 248
Приоритет задачи 326
Программатор 9
  El Cheapo 38

Программирование нелинейное 324
Программная модель 51
Протокол синхронный 162

P

Радиоуправление 310
Разрешение коллизий на шине 166
Распознавание звуковых команд 283
Регенерация памяти 28
Регистр
  CCP1CON 108
  EECON 103
  INTCON 90, 94
  OPTION 101
  STATUS 97
  общего назначения 87
  флагов 90
Регистровые стимулы 69
Регистровый файл 52
Резонанс 375
Резонатор
  кварцевый 25
  керамический 25, 83
Рекурсия 62
Релаксационный генератор 24
Рефлексы 350
Робот
  мигалка 171
  мышь 295, 343
  пищалка 191
  радиоуправляемый 310, 316
  реагирующий
    на звук 279, 285
    на присутствие человека 344
    на свет 263, 269, 273
  ребенок 343
  с дальномером 252, 257
  с дистанционным управлением 238, 242, 248
  с жидкокристаллическим индикатором 204
  с инфракрасным детектором столкновений 234
  с механическими источниками входных сигналов 221
  с одометром 305
  с управляемыми двигателями 289, 295
  со световой индикацией 182
  таракан 330
РОН 87
Ряд Фурье 281

C

Сброс микроконтроллера 81
Светодиодный индикатор
  с общим анодом 168
  с общим катодом 168
Семафор 328
Серводвигатель 311
  градуировка 322
Сервопривод 310
Сеть нейронная 351
Симулятор 51
Система навигации INS 308
Системный ресурс 328
Скорость звука 253
Слово
  идентификации 122
  конфигурации 87, 122
Совмещение функций периферийных устройств 246
Сокет 150
Сонар 252
Стабилизация траектории 306
Стартовый бит 114
Стек 47
Стимул 65
Стоповый бит 114

T

Таймер 33
  PWRT 82
  сторожевой 97
Тактирование 82

Теорема Моргана 376
Терминатор строки 212
Тестирование 361
Техническое задание 357
Токен 46
Трансмиссия 289
Трансформатор 375
Триггер Шмитта 92

У

УВВ 32, 52
Управление
дистанционное 238, 340
Уровень
биологический 10
механический 10
электронный 10
Уровни программирования 10
Усилитель
операционный 281, 288

Ф

Файл стимулов 67, 261
ФАПЧ 56, 60, 84
ФВЧ 282
Физические константы 372
Фильтр
активный 281, 283
Баттерворта 281
верхних частот 282
нижних частот 281
полосовой 282
режекторный 282
Фильтрация сигнала 281
ФНЧ 281
Формат
INXM8 38
PDF 362
S9 38
Функция повторно входимая 62

Ц

ЦАП 119
Цепочка
RC 272, 375
RL 375

Ч

Частота среза фильтра 281
Червяк в лабиринте 349

Ш

ШИМ 44, 107-109, 168, 178, 289

Э

Эмулятор 52
внутрисхемный 142
терминала 148

A

ANSI 148
ANSI C 46
API 26
ASCII 37, 196, 377
ASCIIZ 159, 212

B

BOD 81
BRG 112
Brown-out detection 81

C

CAN 34
CCP 107
Clock Stimulus 68
CMOS 56
CP 88
CPP 101, 108
CTS 145

D

DCD 145
DCE 143
DIP 138
DRAM 28
DSR 145
DTE 143
DTR 145

E

EEPROM 27
EPROM 26

F

Flash 27

G

GPR 87
GPS 213
GUI 162

H

Handshaking 145
HI-TECH 10

I

I2C 33, 34, 162
ICE 64, 142
ICSP 57, 120
IDE 10, 28, 53
Interrupt handler 29
IRF 31

L

LCD 34
Linker 38
LVP 88

M

Microwire 34
MPLAB 10
MPLAB-ICD Debugger 53
Multimastering 163

N

NRZ 34, 114, 143
NTSC 34

O

OTP 27

P

PBasic 46
PC 76
PDF 362
PIC 58
PICC Lite 10
PICmicro 58
PIE1 95
PIR1 95
PISTART Plus 63
PROM 26
PROMATE 64
PWM 107, 109, 178

R

RAM 26, 28
Register Stimulus 69
RI 145
ROM 26
RPN 48
RS-232 34, 143
RS-485 34
RTOS 10, 130, 325
RTS 145

S

SCL 163
SDA 163
Simulator 51

SPI 34
SRAM 28
Stimulus 65
Stimulus file 66

T

TOIE 95
TMR0 95, 98
TRIS 77, 91
TTY 148

U

USART 65, 112
USB 34

W

Warnings 51
WDT 87

Z

ZIF 128, 139