СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Книга не содержит сложных формул и теоретических выкладок. Здесь также нет кочующих из издания в издание готовых решений. Тем более нет почерпнутых из оригинальной документации громоздких схем и огромных таблиц с машинными командами процессора: все это при необходимости интересующийся читатель всегда сможет найти сам. Эта книга – руководство «как делать» и «почему именно так», сборник, в который вошли самые разнообразные материалы, начиная с общих идей и рассуждений и заканчивая конкретными примерами их реализации.

Цель книги – создать у читателя, уже знакомого с микроконтроллерами, единое цельное представление о том, какое разнообразие аспектов необходимо учитывать и с каким широким спектром проблем приходится сталкиваться в практической работе. Несмотря на то что некоторые примеры ориентированы на микроконтроллеры Microchip® PIC18 и PIC24, рассмотренные подходы и приемы могут оказаться полезными при разработке программного обеспечения и для других устройств.

БЛАГОДАРНОСТИ

Благодарность моим родителям за помощь и поддержку в моем увлечении радиоэлектроникой и вычислительной техникой.

Благодарность коллективу Центра интеллектуальных медицинских систем «ИМЕДИС» и его генеральному директору Готовскому Михаилу Юрьевичу, без поддержки которых эта книга не увидела бы свет.

Благодарность сотрудникам испытательной лаборатории технических средств по требованиям электромагнитной совместимости (ИЛ ТС ЭМС) ФБУ «РОСТЕСТ–МОСКВА» и ее руководителю Чеботареву Станиславу Николаевичу за многолетнее плодотворное сотрудничество.

ВВЕДЕНИЕ

Микроконтроллер (microcontroller), как следует из названия, – это всего-навсего маленький (micro) инструмент для управления (control) какими-либо процессами реального мира. С этой целью в микроконтроллере предусмотрены периферийные аппаратные средства: порты ввода-вывода, АЦП, ЦАП, счетчики-таймеры, приемопередатчики. Обработка и формирование сигналов, связанных с этими процессами, осуществляются сердцем микроконтроллера – однокристальной микропроцессорной системой в соответствии с программным обеспечением, обычно хранящимся в энергонезависимой памяти. Маленьким микроконтроллер делает конструктивное объединение всех перечисленных устройств на одном кристалле.

Следует понимать, что процессами реального мира управляет не только программное обеспечение, которое всего лишь закодированная последовательность действий, не отдельно взятый микроконтроллер, для которого как минимум необходим источник питания, а готовое устройство – система, объединяющая аппаратные и программные средства. Применение программно-управляемых компонентов позволяет упростить разработку и поддержку таких систем, поскольку основные затраты переносятся из аппаратной, схемотехнической в программную, алгоритмическую область. По сравнению с аппаратными решениями, разработка программного обеспечения – «виртуальное конструирование» – имеет неоспоримые преимущества:

- гибкость алгоритмов цифровой обработки данных;
- возможность быстрой модификации системы;
- простота отладки: симуляторы, эмуляторы, внутрисхемные отладчики;
- повторное использование готовых программ и библиотек;
- использование стандартной аппаратуры для решения разных задач.

Альтернативным вариантом «виртуального конструирования», обладающим теми же преимуществами, является разработка с применением программно-конфигурируемых устройств – ПЛИС. Хорошая перспектива у гибридных систем, сочетающих программно-управляемые и программно-конфигурируемые компоненты.

В принципе, для программного управления можно задействовать мощные микропроцессорные системы, вычислительные возможности которых позволяют запускать универсальные операционные системы с широким спектром прикладного программного обеспечения. Однако их применение может оказаться технически невозможным в условиях жестких требований к компактности, надежности, пониженному энергопотреблению, низкой стоимости, наличию встроенных средств, облегчающих интеграцию с системой и объектом управления. Напротив, микроконтроллеры предоставляют возможность конструктивно объединить аппаратные и программные средства в одном узле – «интеллектуальном» датчике или исполнительном механизме, что, в частности, повышает его надежность, обеспечивая при этом требуемую функциональность.

Датчик угла поворота руля, необходимый в системе динамической стабилизации автомобиля ESP/DSC, содержит интегрированный микроконтроллер, передающий

данные блоку управления по двухпроводной цифровой шине CAN (рис. 1). Таким образом, длина аналоговых сигнальных цепей сведена к минимуму.

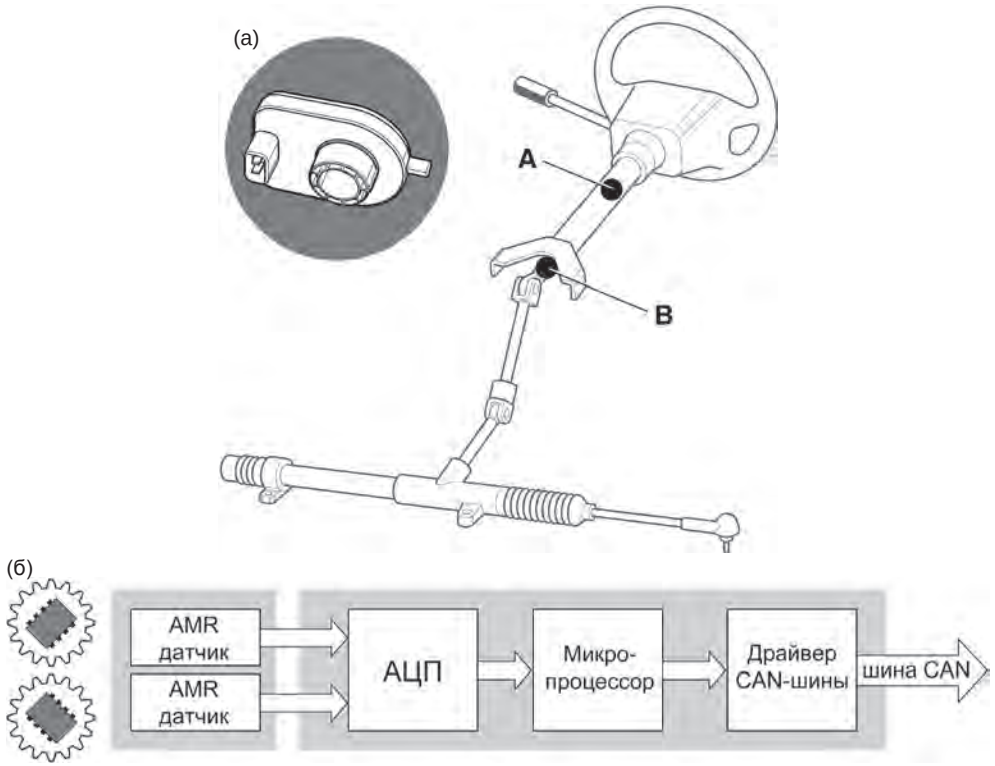

Рис. 1. Датчик угла поворота руля автомобиля (а) и его функциональная схема (б) [5]

Тем не менее микроконтроллерам присущ и ряд недостатков: малая емкость оперативной и постоянной памяти, низкое (по сравнению с полнофункциональными универсальными микропроцессорными системами) быстродействие, отсутствие, в подавляющем большинстве, аппаратно-реализованных механизмов управления памятью и разграничения доступа, необходимость модификации программного обеспечения под конкретную аппаратную реализацию. Из-за этого программирование для микроконтроллеров превращается в балансирование между большим количеством жестких функциональных требований и весьма ограниченными аппаратными возможностями.

Например, емкость встроенной оперативной памяти микроконтроллера недостаточна для реализации подходящего алгоритма. Замена микроконтроллера на модель с большей емкостью памяти нецелесообразна из-за унификации производства, требований по потреблению или наличию в используемом микроконтроллере специфических аппаратных средств. Это заставляет либо модифицировать алгоритм, выполняя большее число действий для экономии памяти, либо добавлять в схему внешнюю память, ограничив число доступных портов

ввода-вывода. Выбор первого решения в большей степени определяется профессионализмом программиста и влияет лишь на трудоемкость разработки, что при серийном производстве несущественно увеличивает стоимость конечного изделия. Второе решение, облегчающее программисту жизнь, приводит не только к удорожанию каждого изделия, но и снижает надежность за счет установки дополнительных компонентов, менее устойчивых к внешним помехам, например к электростатическим разрядам.

В целом программирование для микроконтроллеров имеет ряд специфических особенностей:

- состав и функции оборудования, а также перечень программно решаемых задач *определяются на этапе разработки*;
- разработка программного и аппаратного обеспечения проводится в тесной взаимосвязи;
- обязательно используется постоянная и/или стираемая энергонезависимая память;
- программное обеспечение не модифицируется во время работы: обновление проводится путем замены микроконтроллера, с помощью внутрисхемных программаторов (in-circuit programmers) или встроенных загрузчиков (bootloaders);
- применяются эффективные и не требовательные к ресурсам алгоритмы;
- программное обеспечение монолитно, разработчик полностью отвечает за его качество даже при использовании сторонних программных модулей или библиотек;
- программное обеспечение разрабатывается на другой, более мощной платформе, исполняемый код загружается в микроконтроллер для отладки или в процессе производства готового изделия;
- отсутствуют специфические для универсальных микропроцессорных систем проблемы: многопользовательский доступ, защита от компьютерных вирусов и т. п.

Перечисленные особенности разработки программного обеспечения характерны не только для микроконтроллеров, но и для более широкого класса так называемых «встраиваемых систем» (embedded systems). Программирование для них в настоящее время считается отдельным направлением [30].

Средства разработки программного обеспечения для встраиваемых систем и микроконтроллеров также достаточно разнообразны [65]. Они включают как полноценные инструменты, позволяющие получить исполняемый код:

- императивные процедурные языки (C, ассемблер),
- системы визуального программирования;

так и вспомогательные инструментальные средства:

- языки описания и моделирования (VHDL, verilog),
- средства статической и динамической проверки исходного кода,
- системы управления проектами (RCS, CVS).

К системам визуального программирования можно отнести Actum® Realizer®, который позволяет разработать схему устройства и осуществить ее программное моделирование (рис. 2).

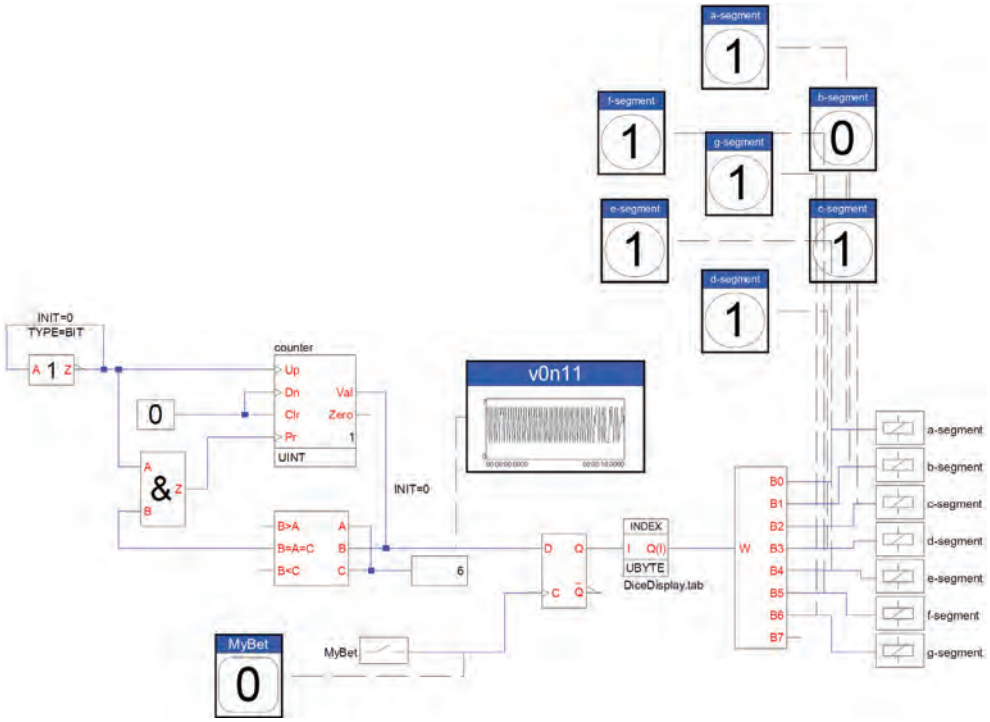

Рис. 2. Автомат, бросающий игральную кость, – Actum® Realizer [www.actum.com]

Для поддерживаемого микроконтроллера Realizer может сгенерировать программный код как на языке ассемблера, так и на языке C.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 * (c) Copyright Actum Solutions 1990.. 2000. All Rights Reserved.
 *
 * File: ros552.c
 * Language: 80552 C
 * Compiler: Tasking CC51 V4
 * Description: Realizer Operating System for a 80C552
 *
 * Revision log:
 * Ver. Date Id Description
 * 1.00 960201 RPJK First release
 * 1.01 000327 MBR Names of functions
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include "stdlib.h"
#include "string.h"
typedef unsigned char byte; /* 8 bit type for common use */
typedef unsigned int word; /* 16 bit type for common use */
#define TRUE 1
```

```
#define FALSE 0
#define DOGTIME 0x00
#define Lo10ms 0x00
#define Hi10ms 0xDC
rom byte copyright[] = "ROS552 [rev 1.01], (c) Copyright 1992,00 by Actum
Solutions";

/*
 * External functions generated by the Realizer
 */
extern void realInit(void);
extern void realMain(short);
extern unsigned char realBit[];
extern unsigned char realByte[];
extern void setByteValue(long out, short typeOut, long value);
extern long convertByteValue(long in, short typeIn);

/*
 * Variables used by ROS
 */

word adcValues[8]; /* array to store the Analog input values */
byte chnl; /* Current AD channel */
byte rlzTick; /* variable to store the last 10 msec ticks */
byte tick; /* variable to store the last 10 msec ticks */

/*************************************************************
 *
 * void digin(long out, short typeOut, char *ioPort, short typeIoPort)
 *
 * description: Operating system specific interface function
 * Interfaces with a DIGIN symbol by means of the NAME attribute
 *
 */

void digin(long out, short typeOut, char *ioPort, short typeIoPort)
{
 if (strncmp(ioPort,"P1_0",4) == 0)
 realBit[out] = P1_0;
 else if (strncmp(ioPort,"P1_1",4) == 0)
 realBit[out] = P1_1;
 else if (strncmp(ioPort,"P1_2",4) == 0)
 realBit[out] = P1_2;
 else if (strncmp(ioPort,"P1_3",4) == 0)
 realBit[out] = P1_3;
}
```

Средства разработки могут быть как «заточены» под конкретный микроконтроллер или их семейство, так и быть унифицированными, пригодными с небольшими дополнениями для широкого ряда устройств. Отметим успешный проект Arduino [80], среда разработки которого предназначена для некоей стандартной аппаратной платформы, предоставляющей единый типовой интерфейс. Сама же платформа, теоретически, может быть реализована с использованием любых компонентов (рис. 3).

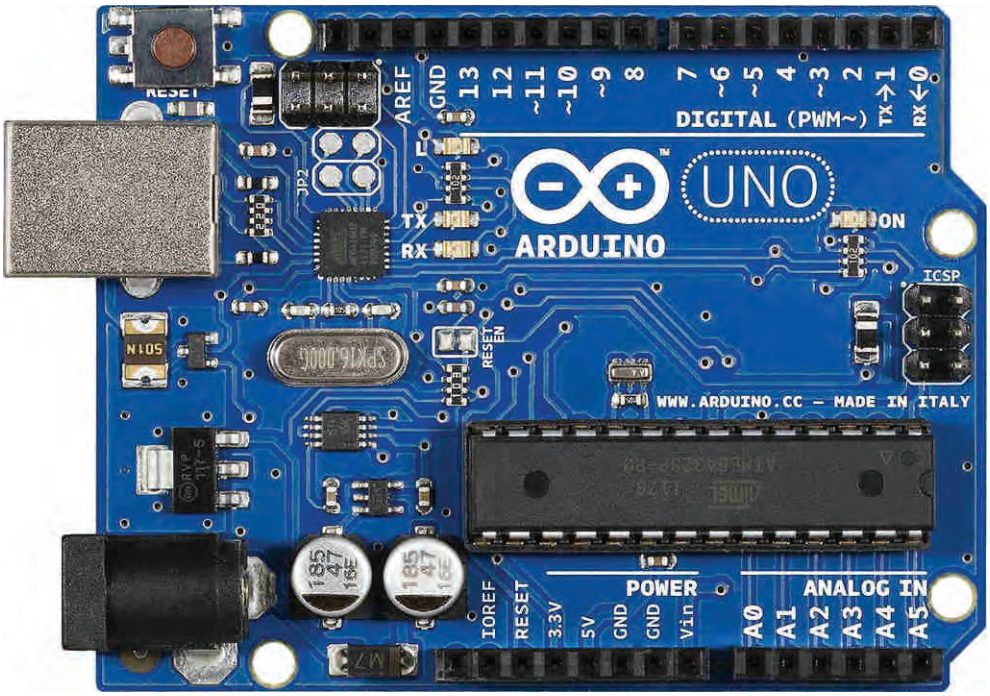

Рис. 3. Основная плата Arduino Uno [robotechshop.com]

Зачастую подобные проекты служат обучающим целям, поэтому наглядны и хорошо документированы. Так, Arduino фактически представляет собой модульный радиоконструктор с возможностью программирования основной платы. Одно из многочисленных руководств по Arduino – Arduino Cookbook – содержит не только подробно разобранные примеры простых устройств, но и разъясняет азы электроники со схемами и иллюстрациями, позволяющими даже ребенку собрать и запустить работающую схему (рис. 4).

Подобные идеи – интегрировать средства взаимодействия специализированной аппаратуры с универсальной высокоуровневой средой разработки непосредственно в аппаратное обеспечение – возникли достаточно давно. Уже в 1998 году существовал проект Parallax BASIC Stamp, который представлял собой небольшую плату-модуль, работающую как интерпретатор программ на языке высокого уровня, похожем на язык BASIC (рис. 5).

Тем не менее возможностей, предоставляемых подобными высокоуровневыми средствами разработки, может оказаться недостаточно. В сложных проектах реального времени необходим непосредственный доступ к периферийным устройствам микроконтроллера и аппаратным средствам системы, например к контроллеру прерываний. Наиболее универсальным и гибким инструментом разработки, обеспечивающим, что немаловажно, переносимость программного обеспечения между разными аппаратными платформами, пока остается язык C – де-факто стандарт системного программирования [2, 7]. Известные недостатки этого языка

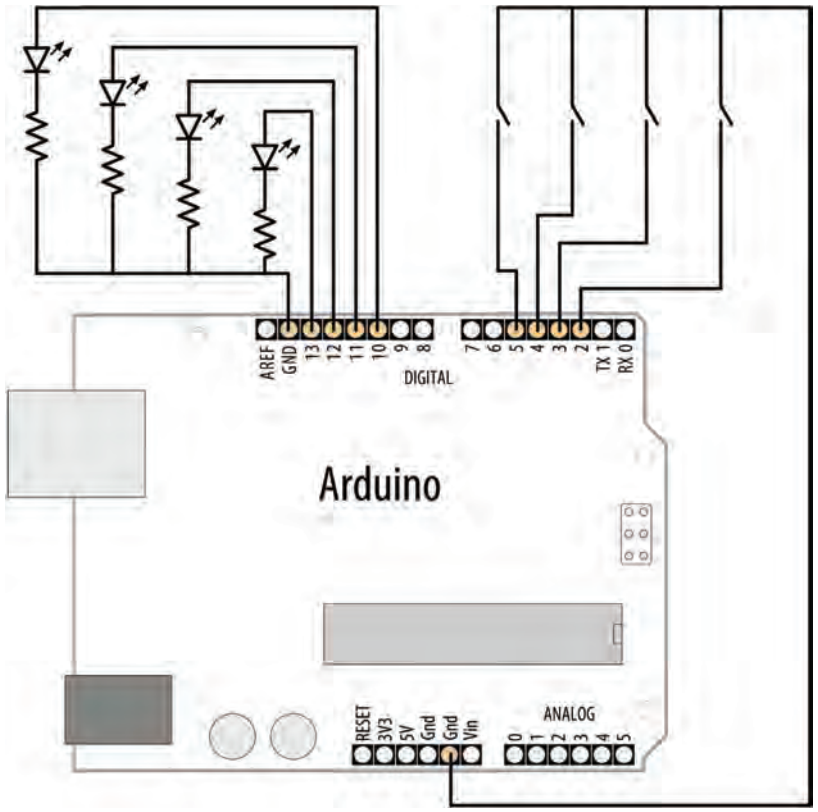

Рис. 4. Подключение светодиодов и кнопок к основной плате Arduino Uno [80]

компенсируются соблюдением требований по проектированию и кодированию, а также применением автоматических анализаторов кода.

В любом случае, вне зависимости от применяемых инструментов и целевой платформы, разработка программного обеспечения – процесс итеративный, включающий в той или иной форме определенные этапы или фазы, которые в рамках зарубежных или отечественных стандартов формализованы вплоть до мельчайших подробностей [5, 17]. К основным этапам разработки относятся:

- анализ требований – определение функций и ограничений системы;
- проектирование – декомпозиция на взаимодействующие подсистемы;
- кодирование – реализация подсистем на выбранном языке программирования;
- тестирование – проверка работоспособности подсистем и системы в целом.

Для каждого из этапов в настоящее время наработан обширнейший теоретический и практический материал, при этом многие идеи и рекомендации можно успешно применять и при программировании для микроконтроллеров. Собственно, главы этой книги как раз и построены таким образом, чтобы систематизировать сведения, которые могут оказаться полезными на каждом из этапов. В тексте используются общепринятые определения и терминология, ко-

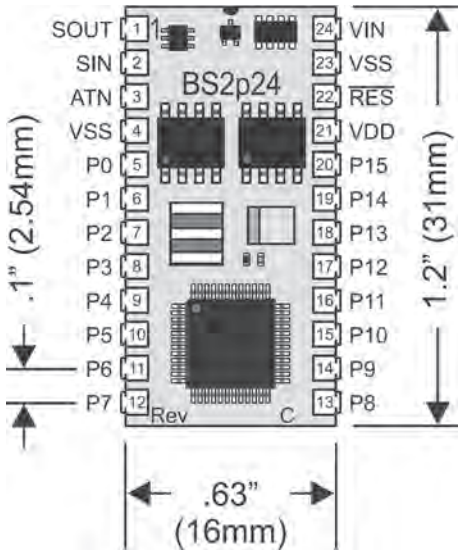

Рис. 5. Модуль BASIC Stamp 2p24 выполнен в 24-выводном типоразмере DIP [100]

торые могут показаться непривычными для разработчиков, ориентированных на какую-то единственную платформу или инструментальное средство. Например, критическая секция – это фрагмент кода, который должен выполняться строго монопольно в многозадачной среде, тогда как в системе Windows® – это конкретный объект API, реализующий один из способов такого выполнения.

Некоторые рассуждения в этой книге могут показаться достаточно абстрактными, но они позволяют увидеть и понять цельную картину. В конечном счете все средства разработки вне зависимости от уровня абстракции и предоставляемого интерфейса предназначены для генерации исполняемого кода – последовательности инструкций для выбранного микроконтроллера. Поэтому разработчик в любом случае должен быть знаком с общими принципами построения и логикой выполнения программного обеспечения.

Многие практические рекомендации проиллюстрированы универсальными примерами на языке C, компилятор которого доступен почти для любого микроконтроллера. Есть примеры и на языке ассемблера для популярных 8- и 16-разрядных микроконтроллеров семейств PIC18 и PIC24 фирмы Microchip®, для которых использовались среда Microchip® MPLAB IDE и компиляторы Microchip® MCC18/MCC30. Такие примеры наглядно показывают, с какими нюансами можно столкнуться на самом низком уровне, что может оказаться полезным при разработке программного обеспечения для любых других микроконтроллеров.

ГЛАВА СТРАНИЦА

1 АНАЛИЗ ТРЕБОВАНИЙ

1.1. Идентификация

Программное обеспечение микроконтроллера – неотъемлемая часть системы, без которой аппаратные средства нежизнеспособны, более того, конечный пользователь системы, которому обычно важен функционал, может даже не догадываться о наличии в ней программно-управляемых компонентов. Поэтому не будет большой ошибкой рассмотреть требования к системе как к единому целому и проследить, как они отражаются на программном обеспечении. Опыт показывает, что большая часть требований может быть выполнена только при комплексном подходе, затрагивающем как аппаратные, так и программные средства.

Анализ требований необходим для определения состава и характеристик оборудования, трудоемкости разработки. В свою очередь, это позволяет оценить сроки и стоимость проектирования и изготовления готового устройства, без которых его коммерческое применение представляется весьма туманным. Расплывчатые или неполные требования приводят не только к непредусмотренному росту затрат и усложнению изделия, но и существенно увеличивают количество ошибок, то есть ухудшают его качество. Это особенно сказывается на прикладном программном обеспечении. Мир не стоит на месте: развитие одного и того же приложения на протяжении десятков лет сопряжено с появлением новых и изменением старых требований. Программное обеспечение постоянно модифицируется, дорабатывается и в итоге становится громоздким, содержит множество программных «заплат» и по своей структуре начинает соответствовать определению «спагетти». Требования, которых нет, – это ошибка, исправить которую труднее всего [1].

Идентификация требований – сама по себе не простая задача. По аналогии с описанием сложного трехмерного объекта с помощью нескольких простых двумерных проекций, разрабатываемая система также должна рассматриваться с разных точек зрения (рис. 1.1). Исходя из этого, требования объединяют в следующие группы [68]:

- функциональные;
- качества;
- операционные;
- параметрические;
- проектные.

Функциональные требования определяют поведение системы, то есть то, что система должна выполнять и ради чего она разрабатывается. Например, для медицинского аппарата магнитной терапии – формировать переменное магнитное поле заданной конфигурации, напряженности и частоты.

Требования к качеству количественно уточняют, как система должна выполнять свои функции. Например, амплитуда импульсов на заданной нагрузке должна устанавливаться в диапазоне от 0 до 15 В с точностью 0,1 В. В реальных условиях этого требования недостаточно, поскольку немаловажным является также и время установления заданной амплитуды, например не более 0,3 с. Требования к временным (динамическим) характеристикам также называются требованиями реального времени. Вне зависимости от конкретной реализации, быстродействия, применения специализированных аппаратных или программ-

ных средств, систему в целом можно назвать системой реального времени только в том случае, если она обеспечивает:

- корректное выполнение всех функций;
- выполнение всех функций в течение заданного интервала времени.

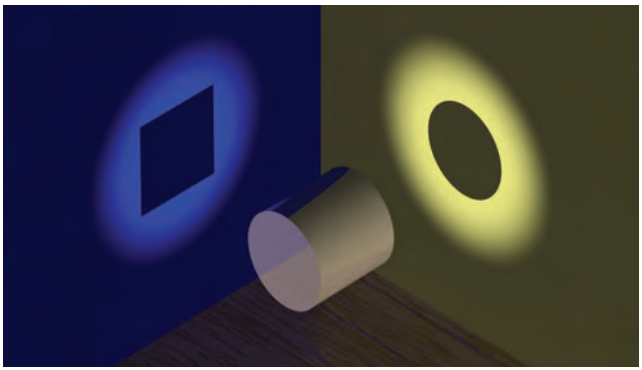

Рис. 1.1. Иногда то, что мы видим, зависит от угла зрения [www.jyoti.ru]

Операционные требования определяют, каким образом система должна взаимодействовать с внешним миром, то есть описывают ее внешний (семантический) интерфейс [33]. В самом общем случае система, управляющая реальным процессом, имеет входы, на которые поступают внешние команды и сигналы о состоянии этого процесса, и выходы, с которых снимается информация о состоянии системы и управляющие воздействия. Таким образом, система предоставляет два интерфейса: интерфейс с человеком (пользовательский) и интерфейс с объектом управления.

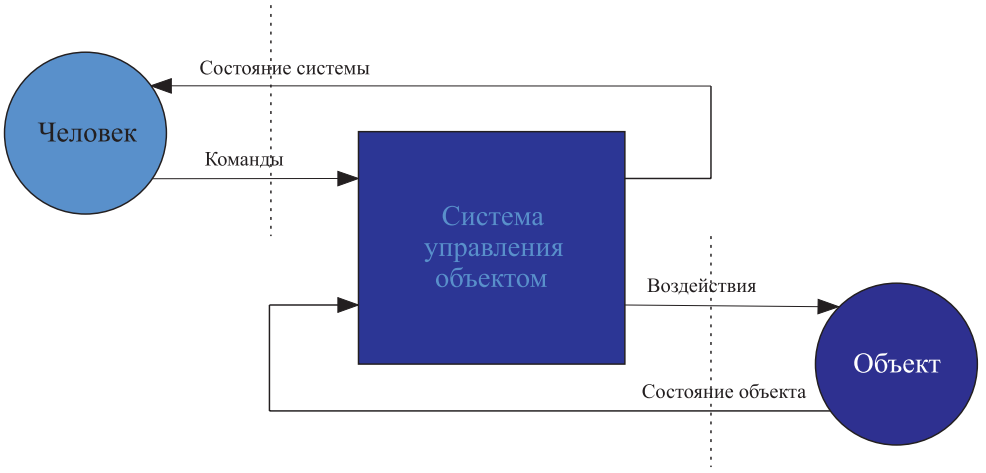

Рис. 1.2. Типичные интерфейсы системы управления

Характеристики этих интерфейсов существенно различаются, например темп формирования управляющих воздействий обычно превышает темп поступления команд, так как для выполнения одной команды может потребоваться целая последовательность управляющих воздействий. Кроме того, в роли источника команд не обязательно должен выступать человек, команды могут поступать и от другой системы управления более высокого уровня.

Для медицинского аппарата операционные требования, по сути, описывают интерфейсы с оператором (выбор терапевтических программ и установка параметров), пациентом (подключение электродов, устройств магнитной терапии) и другим оборудованием (питающая сеть, персональный компьютер). Каждый из этих интерфейсов предусматривает взаимодействие или с человеком, или с окружающей средой, поэтому крайне важно, чтобы устройство в процессе функционирования или само по себе не причиняло им ущерба. Подобные требования – требования безопасности – должны обязательно учитываться в проекте. Например, материал электродов должен быть биологически совместим, терапевтический ток не должен ни при каких условиях превышать допустимой величины и т. п.

Параметрические требования описывают характеристики системы как таковой, вне связи с функциями, которые она выполняет. Например, необходимость постоянного ношения, портативное или стационарное исполнение определяют допустимые габариты и вес устройства, необходимость продолжительной автономной работы от аккумуляторных батарей – потребляемую мощность.

Наконец, *проектные требования* имеют отношение к самому процессу проектирования и производства. Например: сроки разработки, стоимость прототипа и серийного изделия, возможность дальнейшей модификации или использования в других проектах.

Если конкретизировать все перечисленное для встраиваемых систем, то вне зависимости от назначения системы наиболее существенными окажутся требования:

- реального времени;
- безопасности;
- надежности;
- защищенности;
- энергопотребления;
- себестоимости.

В процессе разработки требования могут быть удовлетворены тремя путями:

- унаследовано – применением определенных моделей или средств разработки, например малопотребляющих электронных компонентов или программных стандартов MISRA-C;
- синтаксически, когда соответствие требованиям доказывается теоретически, например путем точного расчета времени выполнения задач в многозадачной среде;
- семантически, когда соответствие проверяется только в результате специальных испытаний или тестирования.

1.2. Требования реального времени

Требования реального времени определяются, исходя из динамических характеристик контролируемых процессов и связанных с ним событий. При этом под событием понимается такое изменение среды, которое подразумевает определенную реакцию системы. Например, изменение атмосферного давления является событием для барометра, но не является событием для секундомера.

Событие – абстрактное понятие: события могут генерироваться как внешней средой, так и внутри самой системы ее компонентами. Пример внешнего события – нажатие на кнопку. Задержка, с которой система должна обработать это нажатие, и станет исходным требованием реального времени. В свою очередь, механическое воздействие на кнопку приводит к возникновению вторичного внутреннего события – изменению напряжения на одном из входов микроконтроллера. Другой пример – генерация импульсов заданной частоты. В этом случае первичным является внутреннее событие – срабатывание программируемого таймера. При этом исходные требования реального времени указываются не для таймера, а для внешних событий – формируемых на выходе сигналов, поскольку измерение частоты осуществляется внешним наблюдателем (рис. 1.3).

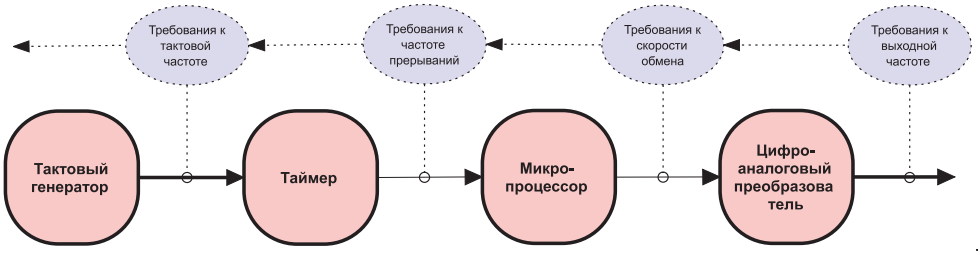

Рис. 1.3. Миграция требований и характеристик системы

Одни события могут порождать другие события более высокого уровня абстракции. Например, в результате обработки низкоуровневого события – изменения напряжения в цепи кнопки, блоком подавления дребезга генерируются события среднего уровня – нажатие и отпускание кнопки. В свою очередь, на их основе можно сгенерировать высокоуровневые события – двойное нажатие или нажатие с повтором, если кнопка удерживается продолжительное время (рис. 1.4).

Опрос клавиатуры с периодом, превышающим длительность дребезга, – самый простой способ подавления дребезга. Во многих случаях достаточно периода 50 мс.

В системах с микроконтроллерами обработка событий обычно осуществляется программно. Для такой обработки характерны два момента: во-первых, при обработке события система в течение определенного времени оказывается невосприимчивой к новым аналогичным событиям, а во-вторых, сама обработка представляет собой последовательность инструкций процессора, каждая из которых выполняется неделимо в темпе, задаваемом тактовой частотой. Таким образом,

программно-управляемая система всегда функционирует дискретно, причем минимально возможный интервал дискретности – одна инструкция процессора. Такие системы относятся к классу конечных динамических систем [4].

Пример программной обработки тактовой частоты – генерация меандра.

```
while (1) {                    00106      D7FE       BRA  0x104
      PORTAbits.RA0 ^= 1;      00104      7080       BTG  0xf80,  0,  ACCESS
}
```

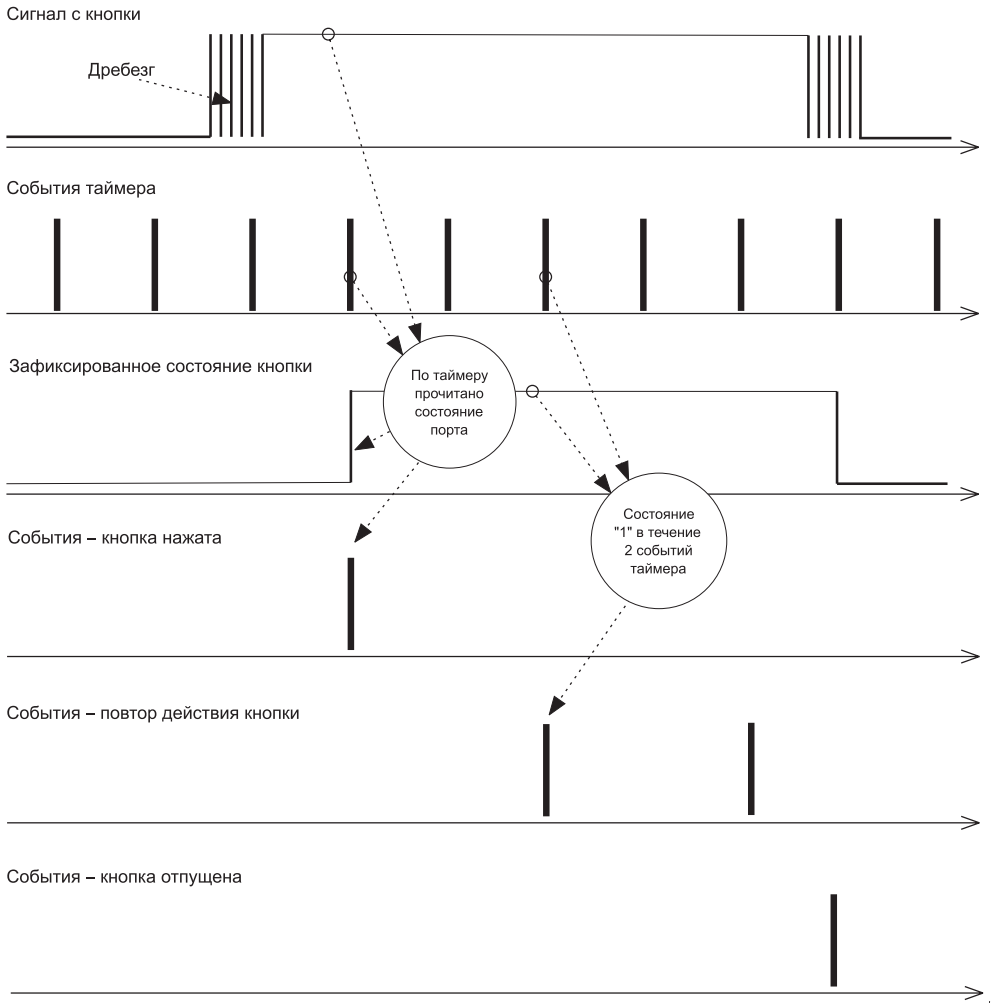

Рис. 1.4. Генерация первичных и вторичных событий

Напротив, контролируемые системой реальные процессы непрерывны, как и сигналы с большинства датчиков, измеряющих физические параметры. Такие

сигналы – непрерывные во времени и принимающие бесконечное множество значений – также называют аналоговыми (рис. 1.5). Характеристиками таких сигналов могут быть частотный спектр или занимаемая полоса, динамический диапазон, время установления сигнала с заданной точностью.

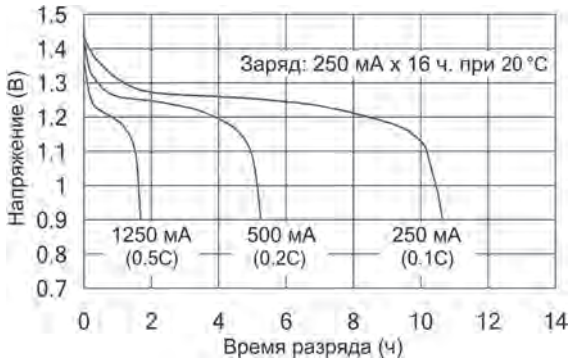

Рис. 1.5. Зависимость напряжения NiMH-аккумулятора GP2700 при его разряде может быть использована для определения возможной продолжительности работы [104]

Для программной обработки непрерывные сигналы необходимо преобразовать в цифровую форму:

- представить периодически следующими мгновенными значениями – выборками (sample), т. е. провести дискретизацию по времени;
- ограничить число возможных значений выборки – провести дискретизацию по уровню (рис. 1.6).

В этом случае исходным событием является синхроимпульс, а вторичным – значение получаемой по этому импульсу выборки. Если система получает и обрабатывает выборки с достаточно высокой частотой (малым периодом), по сравнению со скоростью изменения входного сигнала, внешнему наблюдателю может показаться, что обработка выполняется непрерывно.

Для обоснованного определения периода выборок и соответствующих временных требований следует применить теорему Котельникова или критерий Найквиста (критерий заворота), которые связывают спектральные характеристики сигналов с периодом дискретизации: непрерывный сигнал, не содержащий в своем спектре частот, превышающих F_B, может быть представлен своими выборками, следующими с периодом

$$T \le \frac{1}{2F_B}.$$

Например, период дискретизации низкочастотного сигнала с полосой от 0 до 1 МГц не должен превышать 0,5 мкс.

Наконец, остается выполнить дискретизацию (квантование) по уровню – представить выборку числом фиксированной разрядности. Квантование выпол-

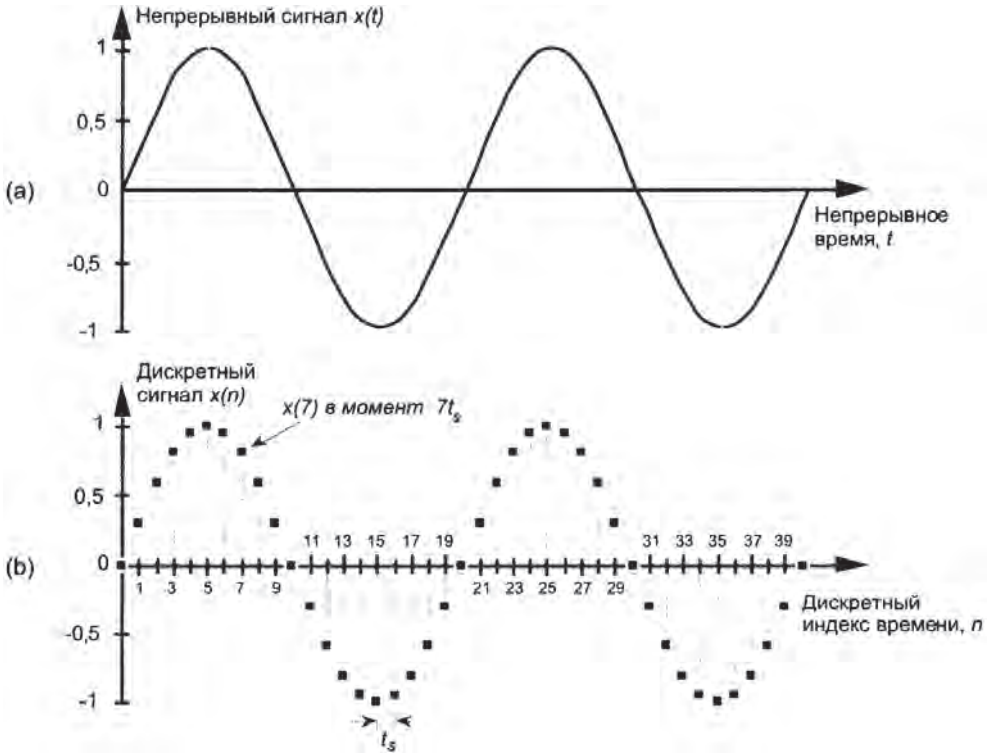

Рис. 1.6. Непрерывные (аналоговые) и дискретные (представленные последовательностью выборок) сигналы [14]

няется, исходя из требований к точности представления сигнала или динамическому диапазону. Динамическим диапазоном (DR, dynamic range) называется отношение максимального воспринимаемого уровня входного напряжения к минимальному, выраженное в децибелах:

$DR = 20\lg(2^N)$.

Часто процессы дискретизации по времени и уровню объединяют одним термином – аналого-цифровое преобразование (АЦП, ADC).

Для 12-разрядного АЦП (N = 12) DR = 72,24 дБ.

Перед проведением АЦП необходима предварительная фильтрация сигнала. Это связано с тем, что спектры всех реальных сигналов бесконечны, и в процессе дискретизации высокочастотная часть спектра может неоднократно попасть в область низких частот (рис. 1.7, 1.8). Иногда достаточно фильтра в виде простой RC-цепочки.

Вне зависимости от природы сигналов их программная обработка сводится к обработке последовательности событий в соответствии со следующей временной диаграммой (рис 1.9).

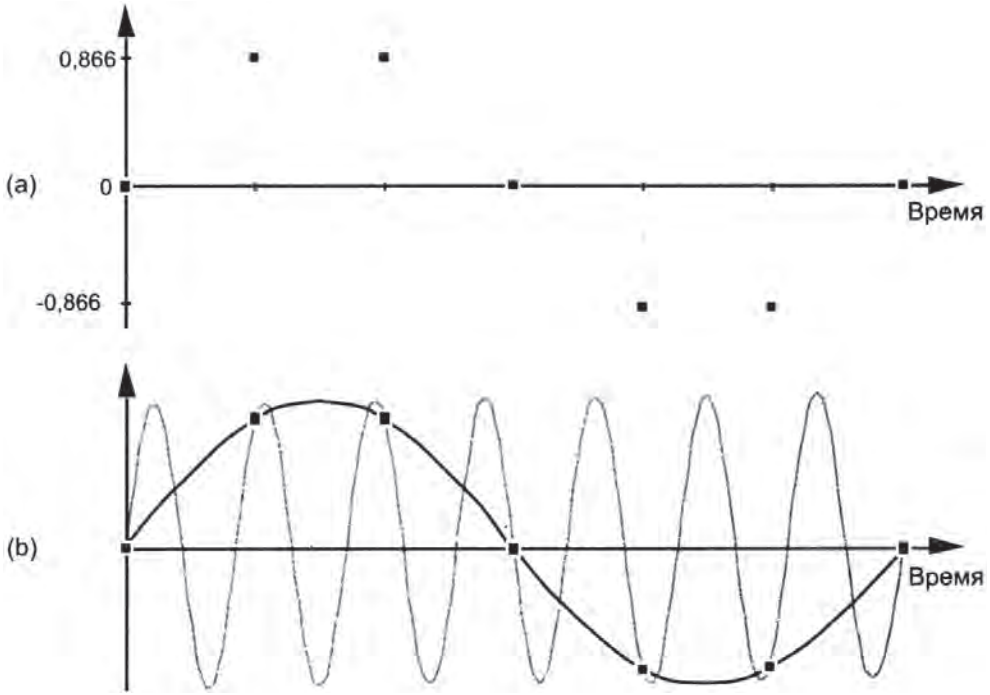

Рис. 1.7. Частотная неоднозначность или стробоскопический эффект при дискретизации сигналов, содержащих частоты, превышающие частоту дискретизации [14]

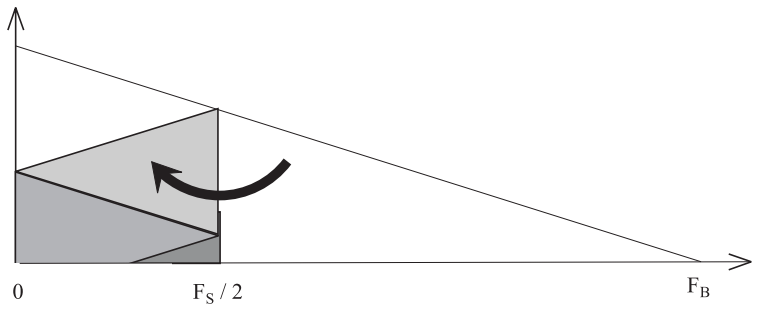

Рис. 1.8. Тот самый заворот спектра:
F_B – верхняя граничная частота, F_S – частота дискретизации (сэмплирования)

Таким образом, требования реального времени для каждого из событий сводятся к указанию интервала T_0–T_4 – допустимого (критического) времени реакции системы (deadline, DL). При этом никакого значения не имеет способ, которым система зафиксирует и обработает событие, главное условие, чтобы эффективное время реакции T_0–T_3, включающее время выполнения функции обработки (execution time, ET), не превысило допустимого. Такие условия называются условиями «жесткого» реального времени (hard realtime) [30].

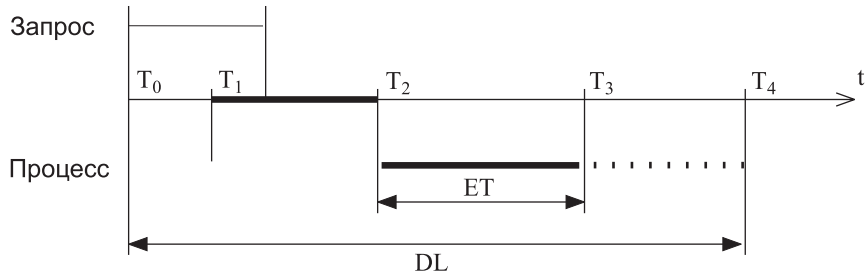

Рис. 1.9. Временная диаграмма программной обработки событий [6]:
T_0 – момент изменения входного сигнала (запрос), T_1 – момент фиксации события,
T_2 – момент запуска соответствующего процесса обработки,
T_3 – момент завершения обработки и формирования выходных сигналов,
T_4 – максимально допустимое время выполнения

Для ряда процессов требования могут быть указаны для средних величин, например допустимое среднее время реакции системы, требуемая пропускная способность или вычислительная производительность. Такие требования – требования «мягкого» реального времени (soft realtime), являясь строгими с аналитической точки зрения, допускают превышение среднего времени обработки для отдельно взятого конкретного события (рис 1.10).

Типичная система управления одновременно обрабатывает события в условиях «жесткого» и «мягкого» реального времени. События первой группы обычно связаны с контролируемым объектом, цифровой обработкой или синтезом сигналов и привязаны к синхросигналам временной сетки (clock driven). События второй группы связаны с пользовательским или межсистемным интерфейсом и имеют асинхронную, случайную природу (event-driven). Соответствующие функции обработки объединяют, соответственно, в синхронную и асинхронную подсистемы (рис. 1.11).

Время реакции системы на любое событие определяется задержкой фиксации и продолжительностью его обработки. Эти параметры, в свою очередь, зависят от выбранных алгоритмов и их реализации.

Фиксация события – это изменение хода вычислительного процесса и/или изменение параметров вычислений, вызванные этим событием. Выделяют следующие способы фиксации:

- опрос (polling) – предусмотренное в выполняемой программе чтение данных из порта ввода-вывода (общей ячейки памяти) и обнаружение изменений, вызванных внешним событием; в отсутствие события процедура опроса выполняется «вхолостую»;
- прерывание (interrupt) – инициируемое внешним событием прерывание текущих вычислений и незамедлительный запуск подпрограммы – обработчика прерывания (interrupt service routine, ISR);
- комбинированный, например прямой, доступ к памяти (direct memory access, DMA).

Классическое применение опроса – циклический ввод данных, обработка и вывод. Обычно период такого цикла задается встроенным таймером, завершение работы которого также фиксируется методом опроса.

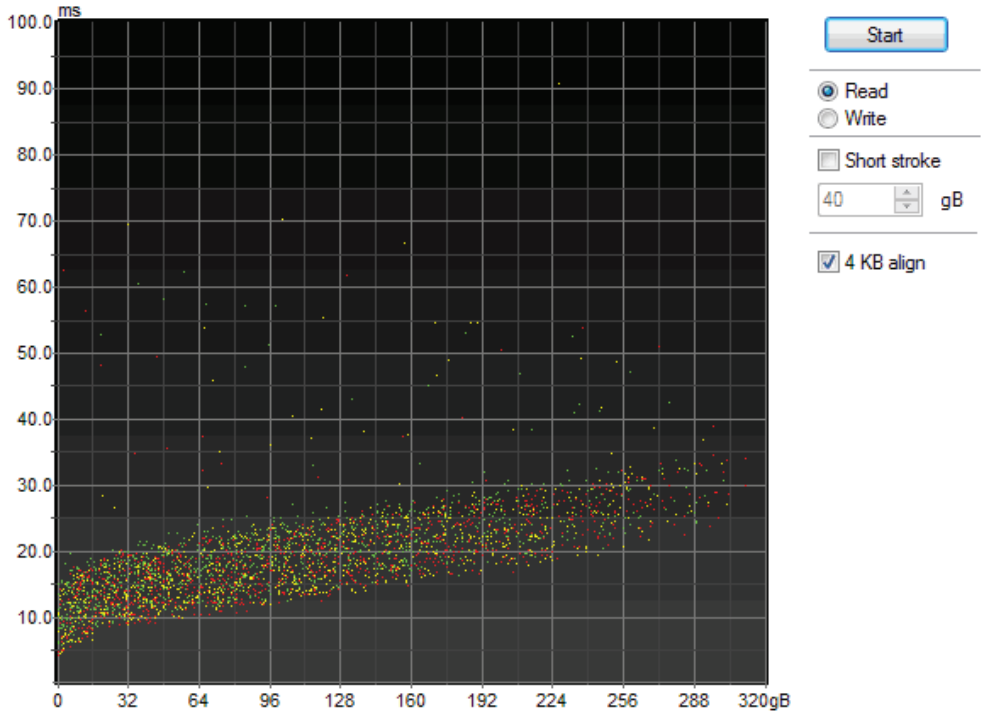

Рис. 1.10. Распределение времени доступа к произвольным блокам данных фиксированного размера жесткого диска ST9320325AS (320 Gb), полученное утилитой HD Tune Pro 5.50. Для блоков размером до 64 Кбайт среднее время чтения не превышает 20 мс, хотя в отдельных случаях время доступа может достигать 60 мс

```
void   main (void)
{
      StartTimer();              /* запустить таймер с заданным периодом */
      while (TRUE) {
      while (!TimerFinished()); /* опрос срабатывания таймера - исходное событие */
      StartTimer();             /* перезапустить таймер */
          InputData();          /* опрос других устройств и ввод данных -
                                   вторичное событие */
      ProcessData();           /* обработка данных */
      OutputData();            /* вывод данных */
      }
}
```

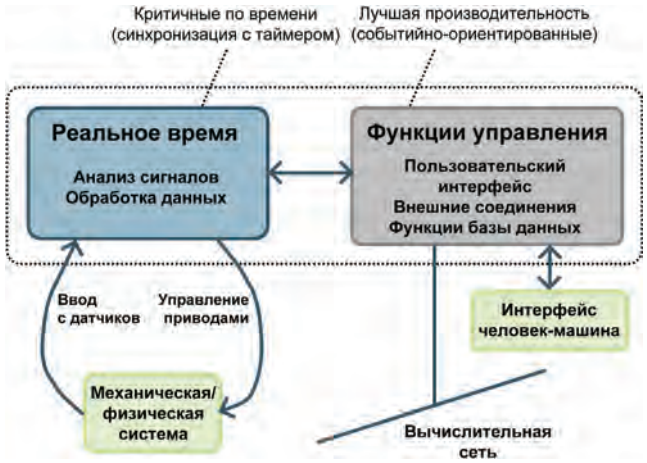

Рис. 1.11. Типичная промышленная система управления [83]

Такой подход – синхронное программирование [33] – достаточно прост и хорошо подходит для цифровой обработки сигналов. Применяемые алгоритмы могут быть как простыми: сглаживание, усреднение, так и достаточно сложными: быстрое преобразование Фурье, цифровая фильтрация [14]. В тех случаях, когда требуется высокое быстродействие, необходимые вычисления могут быть частично или полностью реализованы аппаратно, например с помощью встроенного специализированного сигнального процессора (DSP).

Другим достоинством синхронного программирования является простота проверки требований реального времени: достаточно оценить динамические параметры каждой функции:

- WCET (worst-case execution time) – наихудшее время выполнения;
- ACET (average-case execution time) – среднее время выполнения;
- BCET (best-case execution time) – наилучшее время выполнения (рис. 1.12).

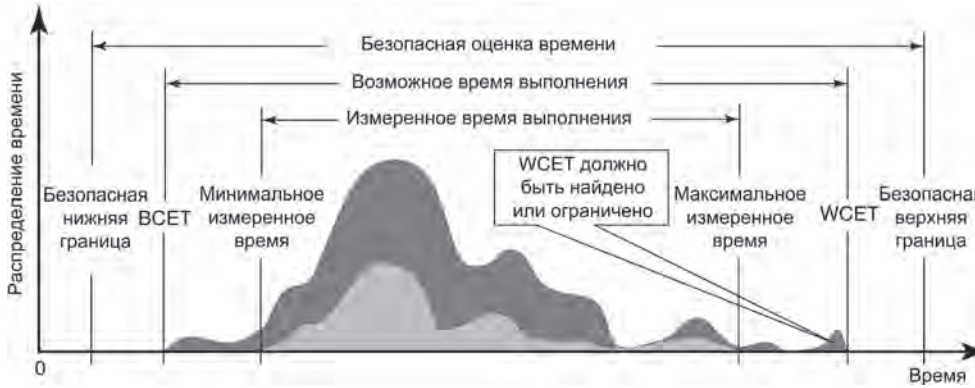

Рис. 1.12. Взаимосвязь между измеренным, возможным, наилучшим (BCET) и наихудшим (WCET) временем выполнения [33]

Ключевая характеристика – WCET – это максимально возможное время выполнения исполняемого кода функции, исключая задержки, связанные с обработкой прерываний или других задач в многозадачной среде [70]. Очевидно, что для синхронной программы в отсутствие прерываний достаточно того, чтобы WCET не превышало максимально допустимого времени выполнения (deadline).

Для точной оценки WCET (вручную или же с помощью автоматизированных средств) необходимо провести:

- анализ потока данных – последовательностей вызова процедур, ветвления и циклов (алгоритмические факторы);
- низкоуровневый анализ времени выполнения каждой процедуры, инструкции, операции ввода-вывода (аппаратные факторы).

Для автоматизированного анализа потоков данных предлагаются разнообразные универсальные алгоритмы: путей, деревьев, IPET. Низкоуровневый анализ тесно связан с архитектурой микропроцессора. Например, для RISC-микроконтроллеров Microchip® PIC18 время выполнения каждой инструкции составляет четыре такта (один цикл), кроме инструкций перехода, которые выполняются за восемь тактов (два цикла). Ситуация существенно усложняется для больших суперскалярных процессоров с предвыборкой команд и блоком предсказания ветвлений (рис. 1.13).

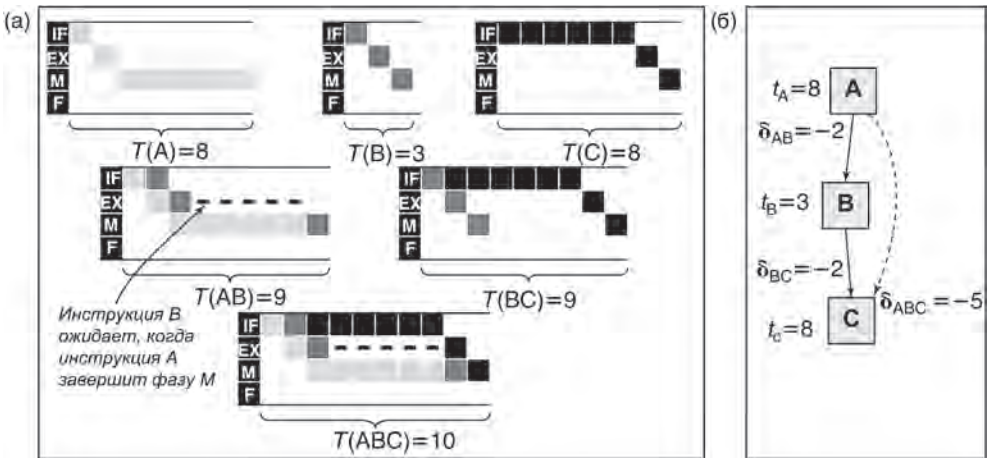

Рис. 1.13. Выполнение последовательности инструкций A, B, C: обычное, последовательное – T(A) + T(B) + T(C) = 19 циклов, с конвейеризацией – T(ABC) = 10 циклов [33]

Для оценки времени выполнения для таких процессоров существуют коммерческие программные комплексы, чаще всего гибридные (расчет–выполнение) с визуализацией результатов. Их применение может быть регламентировано индустриальными стандартами программного обеспечения, например DO-178B/C. Подобные средства требуют определенных подготовительных шагов и точной настройки [69] (рис. 1.14).

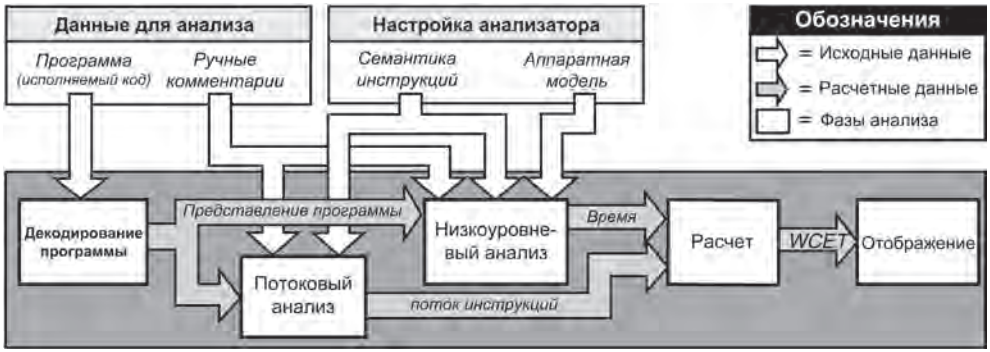

Рис. 1.14. Принцип работы систем автоматизированного анализа WCET [33]

В менее сложных случаях необходимые расчеты WCET можно выполнить вручную, воспользовавшись листингом дизассемблера и оценив время выполнения инструкции для самого длинного пути выполнения.

Расчет WCET для функции вычисления контрольной суммы CRC8 табличным методом, компилятор – Microchip® MCC18.

```
BYTE   ImswCRC (BYTE *pbuf, BYTE len)     /* префикс функции - 6 Tcy */

                        001D0      CFD9        MOVFF 0xfd9, 0xfe6        2
                        001D2      FFE6        NOP                       -
                        001D4      CFE1        MOVFF 0xfe1, 0xfd9        2
                        001D6      FFD9        NOP                       -
                        001D8      E984        SUBFSR 0x2, 0x4           1
                        001DA      E841        ADDFSR 0x1, 0x1           1
{
      BYTE   crc;

      crc = 0;          /* инициализация - 1 Tcy */

                        001DC      6A04        CLRF [x4]                 1

      while (len--) {   /* оператор цикла: */
                        /* выполнение (len > 0) - 6 Tcy, */
                        /* завершение (len == 0) - 5 Tcy */

                        001DE      5000        MOVF [0], W               1
                        001E0      0600        DECF [0], F               1
                        001E2      0900        IORLW 0                   1
                        001E4      E012        BZ 0x20a                  1/2
                        00208      D7EA        BRA 0x1de                 2/-

            crc = Crc8Tbl[crc ^ *pbuf++];     /* тело цикла - 18 Tcy */

                        001E6      EB01        MOVSF [0x1], 0xfe9        2
                        001E8      FFE9        NOP                       -
                        001EA      EB02        MOVSF [0x2], 0xfea        2
                        001EC      FFEA        NOP                       -
```

```
                    001EE     2A01      INCF [0x1], F          1
                    001F0     0E00      MOVLW 0                1
                    001F2     2202      ADDWFC [0x2], F        1
                    001F4     50EF      MOVF 0xfef, W, ACCESS  1
                    001F6     1804      XORWF [0x4], W         1
                    001F8     6AF7      CLRF 0xff7, ACCESS     1
                    001FA     0F2C      ADDLW 0x2c             1
                    001FC     6EF6      MOVWF 0xff6, ACCESS    1
                    001FE     0E00      MOVLW 0                1
                    00200     22F7      ADDWFC 0xff7, F, ACCESS 1
                    00202     0008      TBLRD*                 2
                    00204     50F5      MOVF 0xff5, W, ACCESS  1
                    00206     6E04      MOVWF [0x4]            1
    }
    return crc;     /* загрузка возвращаемого */
                    /* значения - 3 Tcy */
                    0020A     5004      MOVF [0x4], W          1
                    0020C     D000      BRA 0x20e              2
}                   /* суффикс функции - 5 Tcy */
                    0020E     E942      SUBFSR 0x1, 0x2        1
                    00210     CFE7      MOVFF 0xfe7, 0xfd9     2
                    00212     FFD9      NOP                    -
                    00214     0012      RETURN 0               2
```

$$WCET = 6 + 1 + (6 + 18)\cdot len + 5 + 3 + 5 = 20 + 24\cdot len.$$

Время выполнения можно найти и путем измерений: как при тестировании реальной системы, так и с применением симуляторов или автоматизированных тестовых процедур (рис. 1.15).

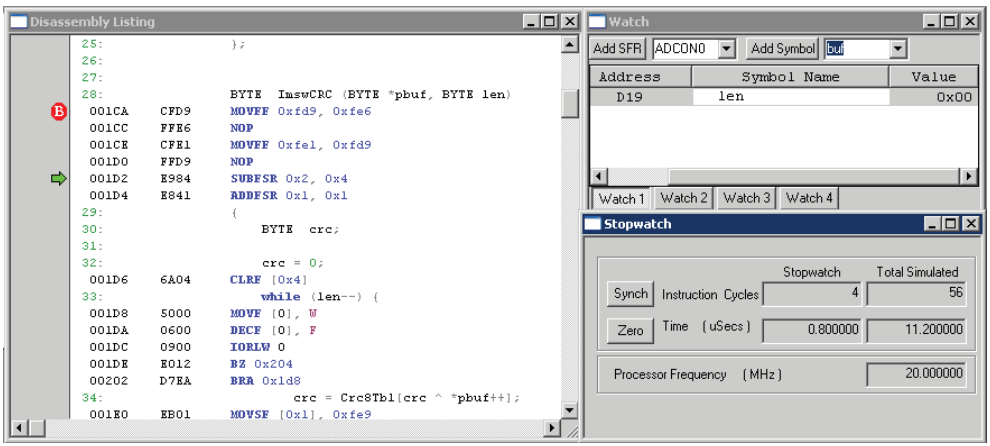

Рис. 1.15. Пошаговая отладка по листингу, счетчик циклов сброшен в точке останова – Microchip® MPLAB 8.80