# Графический интерфейс пользователя с применением микроконтроллеров Microchip

**Графический интерфейс пользователя (Graphical User Interface, GUI) — это система средств для взаимодействия пользователя с устройством, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, кнопок, полос прокрутки и т. п.). При работе с GUI пользователь имеет произвольный доступ (с помощью клавиатуры или устройств координатного ввода, например, touch-screen) ко всем видимым экранным объектам. Впервые графический интерфейс пользователя был реализован в операционных системах персональных компьютеров, но сейчас элементы GUI стали неотъемлемой частью даже простых бытовых и медицинских приборов, сотовых телефонов, устройств промышленной автоматики и многих других. Поскольку графический интерфейс становится все более и более востребованным, то становится очевидным желание разработчиков интегрировать элементы GUI в свои устройства. Естественно, что разработчики заинтересованы в снижении стоимости готового устройства, но для многих практическая реализация сложного графического интерфейса пользователя становится затруднительной задачей, так как требует много усилий и времени для создания собственной библиотеки или покупки готовых программных продуктов сторонних фирм.**

**Илья АФАНАСЬЕВ**
ilya@gamma.spb.ru

Компания Microchip, ведущий производитель микроконтроллеров, известна своими решениями, позволяющими снизить затраты как на разработку, так и общую стоимость изделия, благодаря комплексному подходу к реализации проекта. Бесплатная графическая библиотека Microchip позволяет легко реализовать графический интерфейс пользователя с использованием 16-разрядных микроконтроллеров PIC24 и цветных QVGA-дисплеев.

## Программный интерфейс приложения (API) графической библиотеки Microchip

Бесплатная графическая библиотека Microchip может обслуживать как монохромные индикаторы, так и многоцветные CSTN/TFT (16, 256 и 65 тыс. цветов) дисплеи, имеющие параллельный или последовательный ($I^2C$ или SPI) интерфейс связи с микроконтроллером. Применение индикатора с контроллером и графической памятью позволяет минимизировать требования по памяти, быстродействию и числу выводов управляющего микроконтроллера, поэтому графическим цветным TFT-модулем может управлять даже дешевый, например, 28-выводный контроллер PIC24FJ32GA002 с 32 кбайт Flash-памятью программ и 4 кбайт ОЗУ.

Библиотека разделена на три уровня: графические объекты (Graphics Objects Layer — GOL), графические примитивы (Graphics Primitive Layer) и драйверы устройств (Device Driver Layer). Уровень графических объектов содержит создание и управление сложными графическими объектами, которые, в свою очередь, создаются с помощью платформонезависимых графических примитивов, таких как линия, прямоугольник, окружность и т. п. Драйверы устройств специфичны для конкретного дисплея и предоставляют основные функции для более высоких уровней библиотеки.

Архитектура графической библиотеки Microchip приведена на рис. 1:

1. ***Application Layer*** — пользовательская программа, которая использует графическую библиотеку.
2. ***User Message Interface*** — этот уровень должен быть создан пользователем для предоставления сообщений библиотеке. Графическая библиотека поддерживает передачу сообщений с помощью клавиатуры или touch-панели индикатора.
3. ***Graphics Object Layer*** — этот уровень отрисовывает такие элементы управления, как кнопки, слайдеры, окна и т. д.
4. ***Graphics Primitives Layer*** — этот уровень реализует простейшие графические объекты (линии, прямоугольники, окружности и т. п.).
5. ***Device Display Driver*** — этот уровень управляет индикатором и зависит от типа применяемого дисплея.
6. ***Graphics Display Module*** — графический дисплей.

Реализация графической библиотеки предоставляет две конфигурации — блокирующую и не блокирующую (Blocking и Non-Blocking). Для блокирующей конфигурации функции вывода графических объектов задерживают выполнение программы, пока графический объект не будет выведен полностью. Для не блокирующей конфигурации функции вывода графических объектов не ждут выполнения отрисовки и передают управление пользовательской программе. Различные конфигурации позволяют включать графическую библиотеку в программы

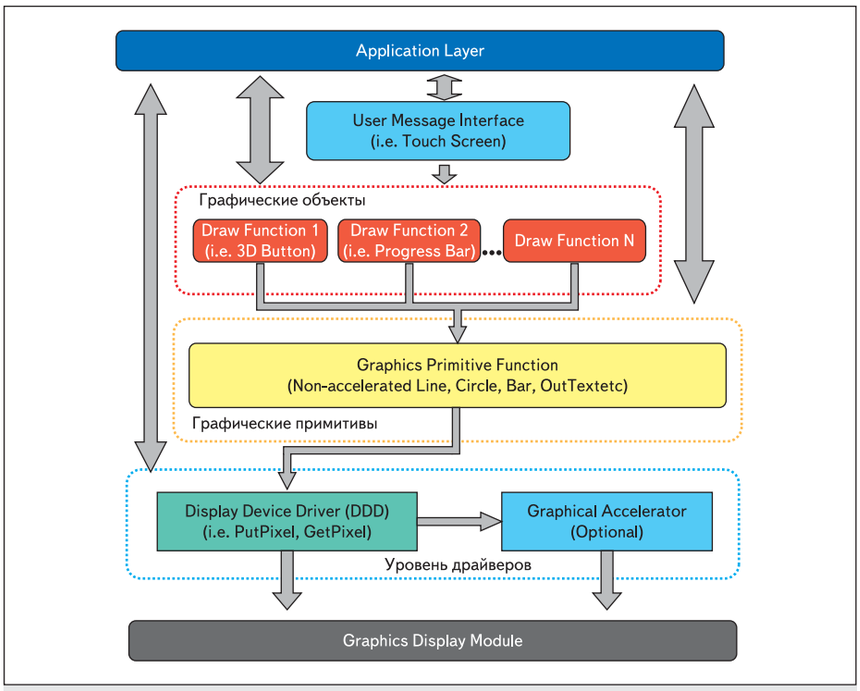


**Рис. 1.** Архитектура графической библиотеки

на основе операционных систем реального времени (RTOS) и более эффективно использовать ресурсы микроконтроллера.

## Уровень драйверов устройства

Каждый тип дисплея имеет свои характеристики (интерфейс связи, графический контроллер). Для связи библиотеки с дисплеем определен набор функций — драйвер дисплея. Текущая версия графической библиотеки поддерживает несколько типов графических контроллеров Samsung S6D0129/S6D0139, Renesas R61505U, Solomon Systech SSD1339, LG LGDP4531 и Densitron HIT1270. Однако возможна поддержка и других контроллеров, для этого из всей библиотеки нужно модифицировать лишь функции драйверов устройств. Доступность библиотеки в исходных кодах позволяет осуществить поддержку дисплеев с различными графическими контроллерами.

**Таблица 1.** Функции драйвера устройств

| Имя функции | Описание |
|---|---|
| ResetDevice | Инициализация дисплея |
| GetMaxX | Возвращает размер экрана по оси x |
| GetMaxY | Возвращает размер экрана по оси y |
| SetColor | Устанавливает текущий цвет отображения |
| GetColor | Возвращает текущий цвет отображения |
| SetActivePage | Устанавливает текущую графическую страницу |
| SetVisualPage | Устанавливает текущую отображаемую графическую страницу |
| PutPixel | Модификация пикселя экрана |
| GetPixel | Возвращает цвет пикселя |
| PutImage | Прорисовка изображения на экране |
| SetClipRgn | Задает текущие границы региона |
| GetClipLeft, GetClipTop, GetClipRight, GetClipBottom | Возвращает левую, верхнюю, правую и нижнюю границу региона |
| SetClip | Разрешает или запрещает границы региона |
| IsDeviceBusy | Проверяет, если контроллер дисплея занят выполнением предыдущей операции |
| SetPalette | Установка регистров цветовой гаммы контроллера |

Программный интерфейс приложения уровня драйверов дисплея используется для сброса дисплея, определения размеров экрана, выделения области, установки текущего цвета и реализации аппаратно-зависимых функций для работы с дисплеем (табл. 1).

## Графические примитивы

Уровень графических примитивов содержит базовые графические функции и «общается» с индикатором с помощью драйверов устройств, поэтому функции графических примитивов становятся независимыми от типа применяемого дисплея. Функции создания графических примитивов могут быть реализованы в уровне драйверов устройства, если применяемый дисплей имеет графический ускоритель (табл. 2).

**Таблица 2.** Функции уровня графических примитивов

| Имя функции | Описание |
|---|---|
| InitGraph | Инициализация контроллера дисплея, установка типа линии сплошная, цвет экрана черный, цвет объекта белый, установка курсора в верхний левый угол |
| ClearDevice | Очистка экрана, установка курсора в позицию 0,0 |
| GetX | Возвращает положение курсора по координате x |
| GetY | Возвращает положение курсора по координате y |
| MoveTo | Устанавливает курсор в новое положение x, y |
| MoveRel | Устанавливает курсор в новое положение относительно текущих координат. dX и dY могут быть положительными или отрицательными |
| OutChar | Выводит символ по текущим координатам |
| OutText | Выводит строку символов по текущим координатам. Строка должна заканчиваться нулем |
| OutTextXY | Выводит строку символов по координатам x и y. Строка должна заканчиваться нулем |
| GetTextHeight | Возвращает высоту символа. Для выбранного шрифта изображение всех символов имеет одинаковую высоту |
| GetTextWidth | Возвращает длину строки символов для выбранного шрифта |
| SetFont | Устанавливает текущий шрифт для функций OutChar(), OutText() и OutTextXY() |
| SetLineType | Устанавливает тип линии |
| Line | Рисует линию установленного типа от точки x1, y1 в x2, y2 |
| LineRel | Рисует линию установленного типа от текущей точки в заданную по смещению |
| LineTo | Рисует линию установленного типа от текущей точки в заданную |
| Circle | Рисует окружность с заданным центром и радиусом |
| FillCircle | Рисует закрашенную окружность с заданным центром и радиусом |
| DrawPoly | Рисует полигон установленным типом линии, используя заданное число узловых точек |
| Rectangle | Рисует прямоугольник по координатам: верхний левый угол, нижний правый угол |
| Bar | Рисует закрашенный прямоугольник по координатам: верхний левый угол, нижний правый угол |
| GetImageWidth | Возвращает ширину картинки |
| GetImageHeight | Возвращает высоту картинки |

## Графические объекты (Graphics Objects Layer — GOL)

С точки зрения реализации графического интереса наибольший интерес представляют графические объекты. Графическая библиотека Microchip реализует 3D графические объекты, такие как кнопки, слайдеры и др. (табл. 3).

Для графических объектов, предоставляемых библиотекой, возможна установка различных цветовых стилей. Для таких объектов, как кнопки, возможно задание разных текстовых меток и картинок для исходного и «нажатого» состояния.

Графическая библиотека Microchip поддерживает работу с пользовательскими шрифтами в 8-разрядной кодировке (ASCII), то есть поддерживает любые языки, в которых меньше 129 символов. Шрифты могут храниться как массивы в памяти программ в секции const, что ограничивает размер секции в 32 кбайт, либо во внешней памяти. Графическая библиотека использует представление символов шрифта как отдельные изображения. Высота каждого символа в одном шрифте одинакова, варьируется только его ширина. Таким образом, размер символа «1» меньше, чем символа «Щ». Задание первого и последнего символа используемого шрифта позволяет уменьшить размер памяти, требуемый для хранения знакогенератора. Например, если будут использоваться только заглавные английские буквы, то оригинальная ASCII таблица может быть уменьшена (табл. 4).

Как результат, для такого знакогенератора потребуется выделение меньшего размера памяти. Если в используемом шрифте часть символов не будет использоваться, то размер кода также может быть уменьшен, если ширину неиспользуемых символов задать равной нулю.

**Таблица 3.** Графические объекты, реализованные в библиотеке Microchip версии v1.0

| Имя функции | Вид объекта |
|---|---|
| Button | Button, Pressed, Disabled, Focused, LO, ON, OFF |
| Slider | |
| Window | Window |
| Group Box | Group Box |
| Static Text | Microchip Graphics Library Static Text. Различные кодировки и шрифты. Различные цветовые схемы. Выравнивание по центру, левому и правому краю |
| Radio Button | Radio buttons |
| Check Box | Checkboxes |
| Picture | Touch screen to continue |
| Progress Bar | Progress bar |
| Scroll Bar | Set time and date |
| List Box | List box |
| Edit Box | Edit box |
| Meter | Meter |
| Dial | Dial |

**Таблица 4.** Пример уменьшенной таблицы шрифтов

| Код ASCII | 65 | 66 | 67 | • | • | • | 87 | 88 | 89 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|
| Символ | A | B | C | • | • | • | W | X | Y | Z |

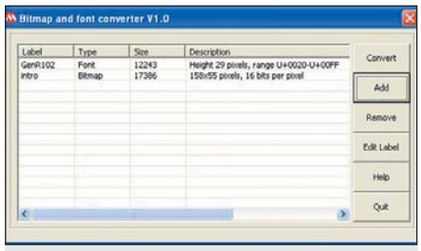


**Рис. 2.** Утилита для преобразования шрифтов и графических файлов в С-массивы данных

Графическая библиотека может применять все шрифты Windows. Вместе с библиотекой Microchip предлагается программа, которая позволяет конвертировать растровые шрифты (файлы с расширением *.fnt) и true-type шрифты (файлы с расширением *.ttf или *.oft) в массивы данных для использования совместно с библиотекой (рис. 2).

Как правило, шрифты защищены авторскими правами, поэтому необходимо убедиться, что вы имеете право использовать тот или иной шрифт. Существуют бесплатные, свободно распространяемые шрифты, часть из которых можно найти по ссылке http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=OFL_fonts. Для получения лучшего результата отображения шрифтов на выбранном дисплее, возможно, понадобится редактирование изображения шрифтов или создание собственного шрифта. Для этого можно использовать, например, бесплатный редактор растровых шрифтов Fony (рис. 3), доступный по ссылке http://hukka.furtopia.org.

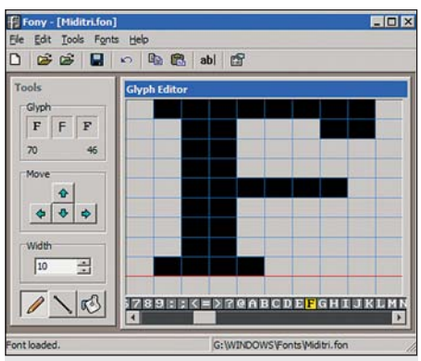


**Рис. 3.** Редактор шрифтов Fony

Графическая библиотека также поддерживает вывод пользовательских изображений. Они хранятся в памяти с расположением точек слева направо и сверху вниз. Цветовая палитра задается таким образом, что цвету с кодом 0 соответствует первая строка в таблице палитры и цвету с кодом 255 (если такой существует) соответствует 256-я строка палитры. Для изображений с более чем 256 цветами палитра цветов не требуется (табл. 5).

Для преобразования изображений в массивы данных используется та же самая утилита, что и для конвертации шрифтов (рис. 3).

Библиотека поддерживает масштабирование изображений, и пользователь может изменить размер выводимого на экран изображения (рис. 4).

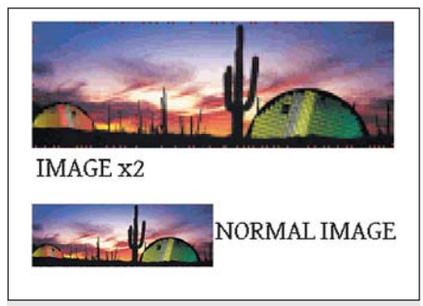


**Рис. 4.** Масштабирование изображений

## Взаимодействие объектов графической библиотеки

### *Состояния объектов*

Графические объекты могут иметь два типа статуса: статус состояния и статус отображения. Статус состояния определяет действия над объектом и его вид. Статус отображения показывает, что объект требует частичной или полной перерисовки или должен быть спрятан. Некоторые общие статусы объектов приведены в таблице 6 и показаны на рис. 5. Разные типы графических объектов имеют свои

**Таблица 5.** Поддерживаемые форматы изображений

| Кодировка | Тип изображения | Описание |
|---|---|---|
| 1bpp | черно-белое | Каждый байт содержит 8 пикселей (точек), старшему биту соответствует левый пиксель. Цветовая палитра содержит 2 значения |
| 4bpp | 16-цветные изображения | Каждый байт хранит 2 пикселя. Правой тетраде соответствует левый пиксель. Цветовая палитра содержит 16 значений |
| 8bpp | 256-цветные изображения | Каждый байт хранит 1 пиксель. Цветовая палитра содержит 256 значений |
| 16bpp | Hi-color Images | Каждые 2 байта (16 бит) хранит один пиксель. Цветовая палитра не требуется. Цвета кодируются в формате 5-6-5: Bits[15:11] = красный (RED) Bits[10:05] = зеленый (GREEN) Bits[04:00] = синий (BLUE) |

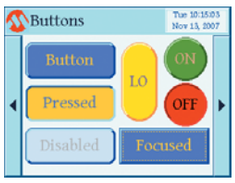


**Рис. 5.** Демонстрация состояний объектов. Кнопка Disabled имеет статус OBJ_DISABLED, кнопка Focused — OBJ_FOCUSED

**Таблица 6.** Состояния графических объектов

| Объект | Тип | Описание |
|---|---|---|
| OBJ_FOCUSED | Статус состояния | Объект в выделенном состоянии. Обычно используется для отображения выбранного в данный момент объекта. Может использоваться, например, для работы с клавиатурой |
| OBJ_DISABLED | Статус состояния | Объект в неактивном состоянии и игнорирует все сообщения |
| OBJ_DRAW_FOCUS | Статус отображения | Выделенное состояние объекта должно быть перерисовано |
| OBJ_DRAW | Статус отображения | Объект должен быть перерисован полностью |
| OBJ_HIDE | Статус отображения | Объект должен быть спрятан путем закраски объекта цветом фона. Наивысший приоритет. Если объект имеет состояние OBJ_HIDE, то другие статусы объекта игнорируются |

**Таблица 7.** Параметры цветовой схемы

| Параметр | Описание |
|---|---|
| EmbossDkColor | Темный цвет боковой грани 3D объекта |
| EmbossLtColor | Светлый цвет боковой грани 3D объекта |
| TextColor0<br>TextColor1 | Цвет текста, используемый для подписи объекта |
| TextColorDisabled | Цвет текста, используемый для подписи объекта в состоянии Disable |
| Color0<br>Color1 | Цвет объекта |
| ColorDisabled | Цвет объекта в состоянии Disable |
| CommonBkColor | Цвет фона. Обычно используется для скрытия объекта на экране |
| pFont | Указатель на шрифт, используемый в объекте |

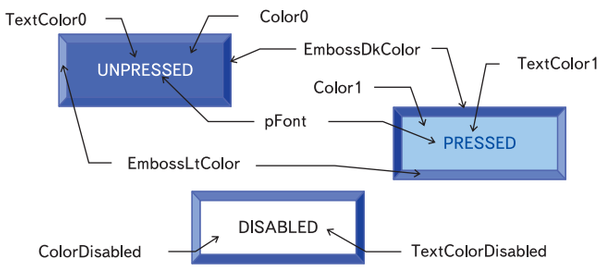


**Рис. 6.** Цветовая схема графического объекта button

статусы, о которых можно узнать в документации на API графической библиотеки.

Поясним на примере, как изменяются статусы объектов при работе библиотеки совместно с индикатором с сенсорной панелью. Если (рис. 5) происходит касание сенсорного экрана на одной из кнопок, то эта кнопка получает статус состояния BTN_FOCUSED, BTN_PRESSED (признак того, что кнопка имеет нажатое состояние) и статус отображения BTN_DRAW_FOCUS. Полученные статусы сообщают библиотеке о том, что фокус перемещается на выбранную кнопку, эта кнопка нажата и должна быть перерисована в новом виде. После того как кнопка будет перерисована, ее статус отображения сбросится, и изображение кнопки останется в неизменном состоянии до тех пор, пока не будут произведены другие действия и не будет изменен ее статус отображения. Если используется клавиатурный ввод, то, как правило, одной из механических кнопок перемещается фокус между графическими объектами на экране дисплея (та или иная графическая кнопка получает статус BTN_FOCUSED и BTN_DRAW_FOCUS), а другой «фиксируется» нажатие выбранной графической кнопки, которая дополнительно получает статус BTN_PRESSED. Если какая-либо из кнопок имеет состояние BTN_DISABLED, то эта кнопка игнорирует все сообщения до тех пор, пока пользовательская программа не отменит этот статус.

### *Цветовые схемы*

Все графические объекты используют цветовые схемы, которые определяют шрифт и цвета объекта. При создании объекта ему задается конкретная пользовательская цветовая схема. Если цветовая схема не задана, то используется цветовая схема по умолчанию. В таблице 7 собраны все компоненты цветовой схемы.

На рис. 6 изображена цветовая схема графического объекта «кнопка» (button).

### *Список активных объектов*

При создании объектов графическая библиотека динамически группирует их в список, отображает их на текущем экране и получает сообщения для этих объектов. Сообщения графических объектов и функции их вывода работают с одним и тем же списком *активных объектов*. Допускается создание множества списков объектов, но только один из них может быть активным в данный момент времени. Для множественных списков объектов пользовательская программа должна управлять переключением между списками и тем самым отображать на экране те или иные графические объекты. Реализация описанной схемы приводит к легкому управлению отображения объектов, так как позволяет пользовательской программе рассматривать каждый список как выводимую на экран страницу со своими графическими объектами: на экран выводятся только те из них, которые присутствуют в списке активных объектов.

### *Отображение графических объектов*

Для прорисовки графических объектов должна вызываться функция управления выводом на экран GOLDraw (). Эта функция анализирует состояние объектов в списке активных объектов и прорисовывает те из них, у которых установлен тот или иной статус отображения. После завершения работы функции GOLDraw () объекты с активным статусом отображения будут перерисованы, а соответствующий статус будет автоматически сброшен. Первый созданный объект будет прорисован первым.

### *Механизм передачи сообщений*

Переносимость и мобильность является одной из ключевых особенностей графической библиотеки Microchip. Библиотека поддерживает ввод данных от различных устройств ввода, таких как сенсорный экран (touch-screen), клавиатура, мышь и т. п. Любое устройство ввода может послать сообщение графическому объекту. Сообщение имеет следующую структуру:

```
typedef struct {
    BYTE type;
    BYTE event;
    int param1;
    int param2;
} GOL_MSG;
```

Поле type определяет идентификатор устройства ввода. Поле event определяет тип действия. Поля type и event будут определять, как параметры param1 и param2 будут интерпретироваться графической библиотекой. Для некоторых случаев используется только параметр param1. Например, при использовании индикатора с сенсорным экраном поля структуры GOL_MSG будут иметь значения, определенные в таблице 8.

После определения нажатия на сенсорный экран пользовательская программа должна заполнить структуру сообщения и передать ее в библиотеку с помощью функции GOLMsg (GOL_MSG* pMsg). Графический объект, который содержит координаты x и y, изменит свой статус, основываясь на своем текущем состоянии и переданном событии.

**Таблица 8.** Определение сообщений сенсорного экрана

| Поле | Определение | Описание |
|---|---|---|
| Type | TYPE_TOUCHSCREEN | В качестве устройства ввода используется сенсорный экран |
| Event | Может принимать несколько значений | |
| | EVENT_INVALID | Ошибка |
| | EVENT_MOVE | Перемещение |
| | EVENT_PRESS | Произошло нажатие |
| | EVENT_RELEASE | Произошло отпускание |
| param1 | x-координата точки касания | |
| param2 | y-координата точки касания | |

## Практическая реализация графического интерфейса пользователя

Графическая библиотека Microchip написана для работы с 16-разрядными микроконтроллерами PIC24FJxxx и 32-разрядными PIC32 с интегрированным параллельным мастер-портом (Parallel Master Port — PMP). Наличие PMP позволяет осуществить быстрый обмен данными между микроконтроллером и дисплеем. Большой объем памяти микроконтроллеров (до 512 кбайт) и широкий набор периферии (по 2 USART, SPI, I²C, часы реального времени, модуль вычисления CRC) делают контроллеры PIC24 и PIC32 идеальными для широкого класса приложений.

В состав библиотеки входят подробные примеры использования. При использовании всех функций библиотеки требуется примерно 24 кбайт программной памяти. Каждый объект динамически выделяет от 2 до 24 байт ОЗУ, таким образом, графическим цветным TFT-индикатором может управлять даже дешевый 28-выводный контроллер PIC24FJ32GA002 с 32 кбайт Flash-памятью программ и 4 кбайт ОЗУ. При необходимости пользовательские шрифты и картинки могут храниться во внешней энергонезависимой памяти.

Библиотека разработана для легкой интеграции графического интерфейса в разрабатываемое устройство. Использование готовых графических объектов требует от программиста минимального количества строк кода. Библиотека содержит хорошо документированный API, применяя который, можно создавать и управлять работой графических объектов. Обычно поведение графического объекта управляется библиотекой. Управление поведением объекта облегчено реализацией механизма передачи сообщений, описанного ранее. Полученные сообщения обрабатываются, и состояние объекта изменяется на основании содержания сообщения. Библиотека автоматически меняет вид объекта и перерисовывает его на экране дисплея.

Рассмотрим пример применения графической библиотеки Microchip. На рис. 7 представлен простой алгоритм программы.

Для начала использования библиотеки программисту требуется создать небольшой код. Во-первых, в программе должны быть подключены модули библиотеки и драйвер дисплея. Далее нужно выполнить инициализацию дисплея, вызвав функцию *InitGraph ()*, в которой производится сброс дисплея, установка курсора в начало координат (положение 0, 0). Затем вызывается функция *GOLCreateScheme ()*, с помощью которой задается текущая цветовая схема для используемых графических объектов. Если не предусматривается изменений в используемой цветовой схеме, то можно вместо вызовов функций *InitGraph ()* и *GOLCreateScheme ()* применить вызов только функции *GOL_Init ()*. Если создается новая цветовая схема, то нужно вставить примерно такой код:

```
GOL_SCHEME* altScheme;
                // объявляем альтернативную цветовую схему
altScheme = GOLCreateScheme ();
                // создаем альтернативную цветовую схему
altScheme-> TextColor0 = BLACK;
                // устанавливаем цвет для color 0
altScheme-> TextColor1 = BRIGHTBLUE;
                // устанавливаем цвет для color
```

Следующий шаг — это создание графических объектов. Функции *ObjCreate (,,)* предоставляют возможность создания различных графических объектов. Это может быть одиночный вызов функции *BtnCreate (,,)*, который создаст объект button, или вызов нескольких функций для создания нескольких объектов. Для примера создадим три графических объекта (две кнопки и один слайдер, рис. 8):

```
BtnCreate (ID_BTN1,          // Идентификатор кнопки 1
      20, 160, 150, 210,     // Размер и положение кнопки
      0,                     // радиус скругления углов
      BTN_DRAW,              // установить статус объекта:
                             // отрисовать кнопку
      NULL,                  // не использовать картинку
                               в качестве изображения
      «LEFT»,                // написать на кнопке этот текст
      NULL);                 // использовать цветовую схему
                               по умолчанию
BtnCreate (ID_BTN2,          // Идентификатор кнопки 2
      170, 160, 300, 210,
      0,                     //
      BTN_DRAW,
      NULL,
      «RIGHT»,
      NULL);
SldCreate (ID_SLD1,          // Идентификатор слайдера
      20, 105, 300, 150,     // Размер и положение слайдера
      SLD_DRAW,              // установить статус объекта:
                             // отрисовать слайдер
      100,                   // диапазон значений
      5,                     // приращение значения
      30,                    // установить ползунок
                               в это значение
      NULL);                 // использовать цветовую схему
                               по умолчанию
```

Все эти вызовы функций отображены на рис. 7 как *ObjCreate ()*, где *Obj* заменяется на *Btn* для кнопок и на *Sld* для слайдера. Каждый объект библиотеки имеет свою функцию *ObjCreate ()*, которые возвращают указатель на новый созданный объект. Если объект успешно создан, то он автоматически добавляется в активный список объектов.

После того как графические объекты созданы, их нужно прорисовать на экране, для этого служит функция *GOLDraw ()*. Эта функция анализирует состояние объектов. Если объект требует отрисовки, то этот объект будет перерисован. В приведенном примере задано состояние кнопок *BTN_DRAW*, а слайдера — *SLD_DRAW*, то есть данные объекты требуют прорисовки. После вызова функции *GOLDraw ()* объекты будут отображены на дисплее, а запрос на прорисовку объектов будет сброшен. Состояние объекта может быть изменено программно или с помощью запросов от устройств ввода, таких как клавиатура, сенсорный экран и т. п. Для данного примера используем индикатор с сенсорным экраном.

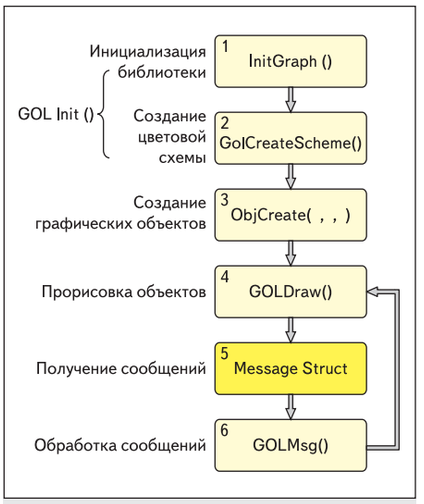


**Рис. 7.** Алгоритм программы

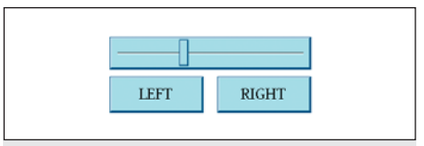


**Рис. 8.** Графические объекты, созданные в примере программы

Сенсорный экран заполняет структуру сообщений, если произошло касание экрана (рис. 7, шаг 5). Это сообщение будет обработано библиотекой при вызове функции *GOLMsg ()*, в которой происходит анализ того, какой из объектов создал сообщение. Состояние объекта будет изменено в соответствии с сообщением, и при следующем вызове функции *GOLDraw ()* этот объект будет перерисован на экране. Кнопка будет отображена нажатой при касании экрана в ее области, а движок слайдера будет перемещен при «перетаскивании» его по экрану (рис. 9).

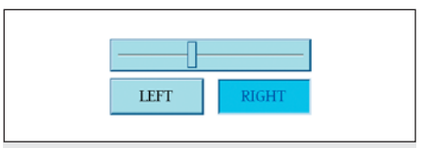


**Рис. 9.** Изменение состояния кнопки при касании сенсорного экрана

В приведенном примере выполнены стандартные функции, присущие созданным графическим объектам: с помощью сенсорного экрана можно нажать на кнопки и передвинуть движок слайдера. Попробуем изменить стандартное поведение. Пусть нажатие на кнопку LEFT или RIGHT будет сдвигать положение движка слайдера соответственно влево или вправо.

Для добавления пользовательских действий над объектами используется функция *GOLMsgCallback ()*. Она вызывается в функции *GOLMsg ()* каждый раз, когда графический объект получает новое сообщение.

Для реализации управления положением движка слайдера с помощью кнопок добавим код в функцию *GOLMsgCallback ()*:

```
WORD GOLMsgCallback (WORD objMsg, OBJ_HEADER* pObj,
GOL_MSG* pMsg){
    WORD objectID;
    SLIDER *pSldObj;
// получение идентификатора объекта,
который создал сообщение
    objectID = GetObjID (pObj);
    if (objectID == ID_BTN1) {
                // проверяем, что сообщение от 1-й кнопки
                // и кнопка нажата
        if (objMsg == BTN_MSG_PRESSED) {
            // устанавливаем указатель на слайдер
              с именем ID_SLD1
        pSldObj = (SLIDER*) GOLFindObject (ID_SLD1);
            // уменьшение значения положения
              движка слайдера
        SldDecPos (pSldObj);
            // установить перерисовку движка слайдера
        SetState (pSldObj, SLD_DRAW_THUMB);
    }
    }
    if (objectID == ID_BTN2) {
                // проверяем, что сообщение от 2-й кнопки
                // и кнопка нажата
        if (objMsg == BTN_MSG_PRESSED) {
            // устанавливаем указатель
              на слайдер с именем ID_SLD1
            pSldObj = (SLIDER*) GOLFindObject (ID_SLD1);
            // увеличение значения положения
              движка слайдера
            SldIncPos (pSldObj);
            // установить перерисовку движка слайдера
            SetState (pSldObj, SLD_DRAW_THUMB);
        }
    }
    // мы должны возвратить 1 для обновления кнопок
      (эффекты нажатия и отпускания)
    return 1;
}
```

Приведенный код изменяет положение движка слайдера при касании сенсорного экрана в области одной из графических кнопок. Кнопка LEFT сдвигает движок влево, кнопка RIGHT — вправо. Функция *GOLMsgCallback ()* должна возвращать 1 для разрешения действий графических объектов, присущих им по умолчанию, то есть кнопки будут прорисовываться в нажатом или нормальном состоянии, а движок слайдера будет иметь возможность передвигаться с помощью сенсорного экрана.

Как видим, нам потребовалось написать минимум кода для создания трех графических объектов и осуществления взаимодействия между ними с помощью сенсорного экрана дисплея.

Графическая библиотека Microchip предоставляет широкое поле деятельности для разработчика и позволяет осуществлять различные действия с целыми графическими объектами или с дисплеем на уровне драйверов. Так,

**Рис. 10.** Дочерняя плата Graphics PICtail™ Plus

например, иллюстрации отображаемых графических объектов в этой статье созданы путем получения цвета отдельных пикселей, формирования bmp-файла и передачи копии экрана дисплея через COM-порт в компьютер.

Для начала работы с графической библиотекой компания Microchip предлагает дочернюю плату Graphics PICtail™ Plus (номер для заказа — AC164127). Graphics PICtail™ Plus — это демонстрационная плата для изучения графической библиотеки и освоения работы с цветными ЖКИ-дисплеями. Плата содержит графический (65 К цветов) QVGA модуль с разрешением 320×240 точек и с резистивной touch-панелью. Модуль поддерживает портретную и ландшафтную ориентацию, содержит встроенную 4-Мбит Flash-память для возможности хранения графических элементов, звуковой излучатель и разъем для подключения к плате Explorer 16. Дочерняя плата подключается к демонстрационной плате Explorer 16.

Демонстрационная плата Explorer 16 (номер для заказа — DM240001) — это дешевое средство отладки для ознакомления и начала работы с высокопроизводительными семействами 16-разрядных микроконтроллеров Microchip PIC24 и контроллерами цифровой обработки сигналов dsPIC33F. Плата имеет возможность работы с внутрисхемным отладчиком ICD-2 и внутрисхемным эмулятором REAL-ICE для быстрой отладки приложений. Комплект содержит 2 дочерние платы с контроллерами PIC24FJ128GA010 и dsPIC33FJ256GP710, возможно подключение процессорных модулей с 32-разрядным контроллером PIC32. Предусмотрено подключение к плате дополнительных интерфейсных модулей расширения, таких как модули IrDA, Ethernet интерфейсов, SD и MMC карт памяти, плат для работы со звуком, и, конечно же, платы с графическим индикатором.

## Литература


1. www.microchip.com/GRAPHICS
2. Microchip Graphics Library API
3. AN1136. How to Use Widgets in Microchip Graphics Library
4. http://ru.wikipedia.org/wiki/BMP