Современная среда разработки mikroC для программирования микроконтроллеров на языке высокого уровня Си

(часть 3)

Олег Вальпа (Челябинская обл.)

Приводится описание современной, мощной и удобной среды разработки mikroC, которая включает большую библиотеку готовых функций для работы с разнообразными интерфейсами и устройствами и позволяет быстро создавать эффективные программы на языке высокого уровня Си для микроконтроллеров семейств PIC, AVR, MCS-51 и др.

Функции для работы с MMC- и SD-картами памяти

Multi Media Card (MMC) – это стандарт карт флэш-памяти. MMC-карты в настоящее время имеют объём до нескольких десятков гигабайт и используются в сотовых телефонах, mp3-плеерах, цифровых камерах и PDA.

Secure Digital (SD) – это стандарт карт флэш-памяти, основанный на старом формате Multi Media Card (MMC). SD-карты в настоящее время имеют объём до нескольких десятков гигабайт. Их модификации с уменьшенными размерами Mini SD и Micro SD используются в сотовых телефонах, mp3-плеерах, цифровых камерах и PDA.

Среда MikroC предоставляет библиотеку для доступа к данным в MMC через SPI. Эта библиотека также поддерживает карты памяти SD. Данная библиотека имеет следующие ограничения:

- библиотека работает только с семейством PIC18;
- библиотечные функции создают и читают файлы только из корневого каталога;
- библиотечные функции при записи файла заносят данные в обе таблицы – FAT1 и FAT2, но при чтении используют данные только из таблицы FAT1. То есть восстановление данных при разрушении FAT1 не производится;
- начиная с версии 5.0.0.3, библиотека может обмениваться данными с носителем, который содержит Master Boot Record (MBR) в секторе 0. Она читает необходимую информацию из MBR и переходит к первому доступному primary логическому разделу. За детальной информацией о MBR, физических и логических устройствах, primary/secondary-разделах и таблицах разделов следует обратиться к другим ресурсам, например, Wikipedia и т.п.

Библиотека включает в свой состав следующие функции работы с картами: Mmc_Init, Mmc_Read_Sector, Mmc_Write_Sector, Mmc_Read_Cid и Mmc_Read_Csd. Кроме того, имеются следующие функции для работы с файлами:

- Mmc_Fat_Init;
- Mmc_Fat_Assign;
- Mmc_Fat_Reset;
- Mmc_Fat_Read;
- Mmc_Fat_Rewrite;
- Mmc_Fat_Append;
- Mmc_Fat_Delete;
- Mmc_Fat_Write;
- Mmc_Fat_Set_File_Date;
- Mmc_Fat_Get_File_Date;
- Mmc_Fat_Get_File_Size;
- Mmc_Fat_Get_Swap_File.

Описание этих функций приведено в таблицах 22 – 38 соответственно.

Таблица 22. Описание функции Mmc_Init

Прототип	unsigned short Mmc_Init(unsigned short *port, unsigned short pin)
Возвращаемое значение	Возвращает 0, если MMC присутствует и успешно инициализирована, в противном случае возвращает 1
Описание	Инициализирует MMC через аппаратный SPI-интерфейс, где состояние вывода chip select задаётся аргументами port и pin; а порт и выводы интерфейса определяются установками аппаратуры SPI для соответствующего контроллера. Mmc_Init должна вызываться перед использованием других функций этой библиотеки
Требования	Функция Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH); должна быть вызвана перед использованием Mmc_Init

Пример	Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE,
	CLK_IDLE_LOW, LOW_2_HIGH);
	// Цикл, пока MMC не будет инициализирована
	while (Mmc_Init(&PORTC, 2));

Таблица 23. Описание функции Mmc_Read_Sector

Прототип	unsigned short Mmc_Read_Sector(unsigned long sector, unsigned short *data)
Возвращаемое значение	Возвращает 0 при успешном чтении или 1, если имеет место ошибка
Описание	Функция читает один сектор (512 байт) из MMC, адрес сектора задаётся аргументом sector. Читаемые данные сохраняются в массиве data
Требования	MMC должна быть проинициализирована с помощью функции Mmc_Init.
Пример	error = Mmc_Read_Sector(sector, data)

Таблица 24. Описание функции Mmc_Write_Sector

Прототип	unsigned short Mmc_Write_Sector(unsigned long sector, unsigned short *data)
Возвращаемое значение	Возвращает 0 при успешной записи; 1, если была ошибка в передаче команды записи; 2, если была ошибка записи
Описание	Функция записывает 512 байт из data в сектор с адресом sector MMC
Требования	MMC должна быть проинициализирована с помощью функции Mmc_Init.
Пример	error = Mmc_Write_Sector(sector, data)

Таблица 25. Описание функции Mmc_Read_Cid

Прототип	unsigned short Mmc_Read_Cid(unsigned short * data_for_registers)
Возвращаемое значение	Возвращает 0, если чтение успешно, и 1 в случае ошибки
Описание	Функция читает CID-регистр и возвращает 16 байт содержания в data_for_registers
Требования	MMC должна быть проинициализирована с помощью функции Mmc_Init
Пример	error = Mmc_Read_Cid(data)

Таблица 26. Описание функции Mmc_Read_Csd

Прототип	unsigned short Mmc_Read_Csd(unsigned short *data_for_registers)
Возвращаемое значение	Возвращает 0, если чтение успешно, и 1 в случае ошибки
Описание	Функция читает CSD-регистр и возвращает 16 байт содержания в data_for_registers
Требования	MMC должна быть проинициализирована с помощью функции Mmc_Init
Пример	error = Mmc_Read_Csd(data)

Таблица 27. Описание функции Mmc_Fat_Init

Прототип	unsigned short Mmc_Fat_Init(char *port, unsigned short pin)
Возвращаемое значение	Функция возвращает 0, если инициализация успешна, и 1, если загрузочный сектор не найден, или 255, если карта не обнаружена
Описание	Инициализирует подпрограммы работы с системой FAT MMC/SD-карт; линия CS для связи задаётся аргументами port и pin parameters. Эта функция должна быть вызвана перед использованием остальных библиотек FAT MMC
Требования	Функция Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH); должна быть вызвана перед использованием Mmc_Init
Пример	Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH); Mmc_Fat_Init(&PORTC, 2);

Таблица 28. Описание функции Mmc_Fat_Assign

Прототип	unsigned short Mmc_Fat_Assign(char *filename, char create_file)
Возвращаемое значение	Возвращает 1, если файл присутствует (или файла нет, но новый файл создаётся), или 0, если файл отсутствует и новый файл не создаётся
Описание	Назначает файл для FAT-операций. Если файл отсутствует, функция создаёт новый файл с заданным названием. Аргумент filename – это название файла (должно быть в формате 8.3 UPPERCASE). Аргумент create_file – это признак создания новых файлов. Если create_file отличается от 0 – создаётся новый файл (если нет файла с указанным названием)
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init
Пример	Mmc_Fat_Assign('MIKROELE.TXT',1)

Таблица 29. Описание функции Mmc_fat_Reset

Прототип	void Mmc_fat_Reset(unsigned long *size)
Возвращаемое значение	Нет
Описание	Открывает существующий файл для чтения, size – возвращает размер файла в байтах
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init Файл должен быть назначен. См. Mmc_Fat_Assign
Пример	Mmc_Fat_Reset(size)

Таблица 30. Описание функции Mmc_Fat_Read

Прототип	void Mmc_Fat_Read(unsigned short *bdata)
Возвращаемое значение	Нет
Описание	Читает байт данных из файла в bdata
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init Файл должен быть назначен. См. Mmc_Fat_Assign Файл должен быть открыт для чтения. См. Mmc_Fat_Reset
Пример	Mmc_Fat_Read(character)

Функция Mmc_Set_Reg_Adr предназначена только для внутренних потребностей компилятора. Схема подключения карты памяти к микроконтроллеру показана на рисунке 27. В нижеследующем примере программы 1024 байта записываются в сектора 55 и 56, а затем сектора читаются и выводятся в порт USART для визуального контроля:

```
unsigned i;
unsigned short tmp;
unsigned short data[512];
void main() {
Usart_Init(9600);
// Инициализация интерфейса MMC
Spi_Init_Advanced(MASTER_OSC_DIV1
6, DATA_SAMPLE_MIDDLE,
CLK_IDLE_LOW, LOW_2_HIGH);
while (Mmc_Init(&PORTC, 2)) ;
// Заполнение буфера символами
"M"
for (i = 0; i <= 511; i++)
data[i] = "M";
// Запись его в сектор 55 MMC
tmp = Mmc_Write_Sector(55, data);
// Заполнение буфера символами
"E"
for (i = 0; i <= 511; i++)
data[i] = "E";
```

Таблица 31. Описание функции Mmc_fat_Rewrite

Прототип	void Mmc_fat_Rewrite()
Возвращаемое значение	Нет
Описание	Открывает файл для записи. Если существует файл с заданным названием, файл будет перезаписан
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init Файл должен быть назначен. См. Mmc_Fat_Assign
Пример	Mmc_Fat_Rewrite

Таблица 32. Описание функции Mmc_fat_Append

Прототип	void Mmc_fat_Append()
Возвращаемое значение	Нет
Описание	Открывает файл для записи. Если существует файл с заданным названием, данные будут дописаны в его конец
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init. Файл должен быть назначен. См. Mmc_Fat_Assign
Пример	Mmc_Fat_Append

Таблица 33. Описание функции Mmc_Fat_Delete

Прототип	void Mmc_Fat_Delete()
Возвращаемое значение	Нет
Описание	Удаляет файл с MMC
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init. Файл должен быть назначен. См. Mmc_Fat_Assign
Пример	Mmc_Fat_Delete

Таблица 34. Описание функции Mmc_fat_Write

Прототип	void Mmc_fat_Write(char *fdata, unsigned data_len);
Возвращаемое значение	Нет
Описание	Записывает данные в файл на MMC. Аргумент fdata – записываемые данные. Аргумент data_len – количество байтов информации для записи
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init. Файл должен быть назначен. См. Mmc_Fat_Assign. Файл должен быть открыт для записи. См. Mmc_Fat_Rewrite или Mmc_Fat_Append
Пример	Mmc_Fat_Write(file_contents, 42); // записать данные в назначенный файл

Таблица 35. Описание функции Mmc_fat_Set_File_Date

Прототип	void Mmc_fat_Set_File_Date(unsigned int year, unsigned short month, unsigned short day, unsigned short hours, unsigned short mins, unsigned short seconds)
Возвращаемое значение	Нет
Описание	Устанавливает атрибуты времени для файла. Можно устанавливать year, month, day. hours, mins, seconds
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init. Файл должен быть назначен. См. Mmc_Fat_Assign. Файл должен быть открыт для записи. См. Mmc_Fat_Rewrite или Mmc_Fat_Append
Пример	Mmc_Fat_Set_File_Date(2005, 9, 30, 17, 41, 0)

Таблица 36. Описание функции Mmc_fat_Get_File_Date

Прототип	void Mmc_fat_Get_File_Date(unsigned int *year, unsigned short *month, unsigned short *day, unsigned short *hours, unsigned short *mins)
Возвращаемое значение	Нет
Описание	Читает атрибуты времени файла. Можно читать year, month, day, hours, mins
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init. Файл должен быть назначен. См. Mmc_Fat_Assign
Пример	Mmc_Fat_Get_File_Date(year, month, day, hours, mins);

Таблица 37. Описание функции Mmc_fat_Get_File_Size

Прототип	unsigned long Mmc_fat_Get_File_Size()
Возвращаемое значение	Размер файла в байтах
Описание	Функция позволяет получить размер файла в байтах
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init. Файл должен быть назначен. См. Mmc_Fat_Assign
Пример	Mmc_Fat_Get_File_Size

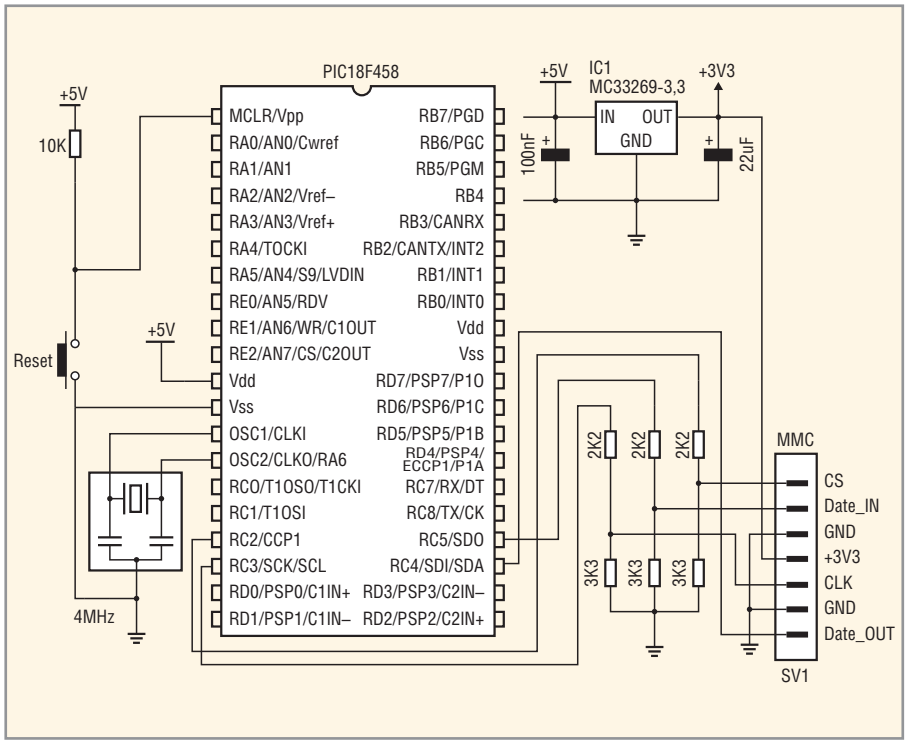

Рис. 27. Схема подключения карты памяти к микроконтроллеру

```
// Запись его в сектор 56 MMC
tmp = Mmc_Write_Sector(56, data);
/** Теперь чтение секторов 55 и
56 **/
// Чтение сектора 55
tmp = Mmc_Read_Sector(55, data);
// Отправка 512 байтов из буфера
в USART
if (tmp == 0)
for (i = 0; i < 512; i++)
Usart_Write(data[i]);
// Чтение сектора 56
tmp = Mmc_Read_Sector(56, data);
// Отправка 512 байтов из буфера
в USART
if (tmp == 0)
for (i = 0; i < 512; i++)
Usart_Write(data[i]);
} // end
```

Функции для звука

Среда MikroC предоставляет звуковую библиотеку, которая позволяет организовать звуковую сигнализацию в разрабатываемом устройстве. Для этого потребуется только пьезоэлектрический излучатель и назначенный порт.

Библиотека включает в свой состав функцию инициализации Sound_Init и функцию формирования звука Sound_Play. Описание этих функций приведено в таблицах 39 и 40 соответственно.

Следующий пример демонстрирует использование функций звуковой библиотеки для генерации тонового сигнала с помощью пьезоэлектрического излучателя. Данный программный код может быть использован на любом микроконтроллере, который имеет PORTB и АЦП на порту PORTA. Частота звука задаётся входным напряжением АЦП (младший байт кода этого напряжения используется как период сигнала $T = 1/f$).

```
unsigned adc_value;
void main() {
PORTB = 0; // Очистка PORTB
TRISB = 0; // PORTB - выход
INTCON = 0; // Запрет всех прерываний
ADCON1 = 0x82; // Конфигурирова-
ние VDD как опорного напряжения,
// и выбор канала АЦП
TRISA = 0xFF; // PORTA - вход
Sound_Init(&PORTB, 2); // Инициа-
лизация выдачи звука на RB2
do { // Цикл проигрывания звука:
adc_value = Adc_Read(2); // Полу-
чить младший байт АЦП
Sound_Play(adc_value, 200); //
Проиграть его
} while (1);
}// end
```

Функции для работы с однопроводным интерфейсом One Wire

Библиотека для работы с интерфейсом One Wire (1-Wire) представляет собой набор функций для связи с внешним устройством по одно-

Таблица 38. Описание функции Mmc_Fat_Get_Swap_File

Прототип	unsigned long Mmc_Fat_Get_Swap_File(unsigned long sectors_cnt)
Возвращаемое значение	Номер начального сектора swap-файла, если он был создан, в противном случае возвращает 0
Описание	Эта функция используется для создания swap-файла на MMC/SD-носителе. Она принимает аргумент sectors_cnt в качестве количества последовательных секторов, которые предполагается использовать в качестве swap-файла. В процессе исполнения функция ищет доступные последовательно расположенные сектора, количество которых задано аргументом sectors_cnt. Если на носителе существует такая область, в ней создается файл MIKROSWP.SYS и область соответствующим образом обозначается в таблицах FAT. Файлу присваиваются атрибуты: system, archive и hidden, чтобы отличать его от других файлов. Если файл с именем MIKROSWP.SYS уже существует на этом носителе, данная функция удаляет его до создания нового такого же. Назначение swap-файла состоит в том, чтобы сделать процесс чтения и записи на MMC/SD-носитель настолько быстрым, насколько это возможно, благодаря использованию функций Mmc_Read_Sector() и Mmc_Write_Sector() непосредственно, без потенциального разрушения FAT-системы. Swap-файл может рассматриваться как «окно» на носителе, куда пользователь может свободно писать/читать данные каким ему угодно способом. Его главное назначение в библиотеке mikroC состоит в обеспечении процесса быстрого сбора данных, когда критичный ко времени процесс сбора данных заканчивается, данные можно переписать в «нормальный» файл в требуемом формате
Требования	Порты должны быть проинициализированы для FAT-операций с MMC. См. Mmc_Fat_Init

Пример	// Попытка создания swap-файла размером не менее 1000 секторов
	// Если удаётся, то номер стартового сектора посылается в USART
	void M_Create_Swap_File() {
	size = Mmc_Fat_Get_Swap_File(1000);
	if (size) {
	Usart_Write(0xAA);
	Usart_Write(Lo(size));
	Usart_Write(Hi(size));
	Usart_Write(Higher(size));
	Usart_Write(Highest(size));
	Usart_Write(0xAA);
	}
	}//end

Таблица 39. Описание функции Sound_Init

Прототип	void Sound_Init(unsigned short *port, unsigned short pin)
Возвращаемое значение	Нет
Описание	Настраивает для микроконтроллера вывод pin порта port на выход
Требования	Номер вывода должен быть в диапазоне 0 – 7
Пример	Sound_Init(&PORTB, 2); // Инициализирует генерацию звука на RB2

Таблица 40. Описание функции Sound_Play

Прототип	void Sound_Play(unsigned short period_div_10, unsigned num_of_periods)
Возвращаемое значение	Нет
Описание	Воспроизводит звук на заданном порту и выводе, заданном с помощью функции Sound_Init. Аргумент period_div_10 – период звуковой частоты в циклах микроконтроллера, поделенный на 10. Генерация звука продолжается в течение заданного аргументом (num_of_periods) числа периодов
Требования	Чтобы звук воспроизводился, необходимо подключение пьезоэлектрического излучателя (или другого звукового излучателя) на заданном выводе порта. Кроме того, перед выводом следует вызвать функцию Sound_Init, чтобы подготовить аппаратуру к генерации звука
Пример	Требуется воспроизвести звук частотой 1 кГц, период которого составляет $T = 1/f = 1$ мс или 1000 циклов для частоты 4 МГц. Отсюда можно определить первый аргумент: period_div_10=1000/10 = 100. Тогда проиграть num_of_periods=150 периодов можно вызовом функции: Sound_Play(100, 150)

Таблица 41. Описание функции Ow_Reset

Прототип	unsigned short Ow_Reset(unsigned short *port, unsigned short pin)
Возвращаемое значение	0, если DS1820 присутствует, и 1 – если нет
Описание	Выдаёт сигнал сброса для One Wire DS1820. Аргументы port и pin определяют подключение DS1820 к микроконтроллеру
Требования	Работает только с цифровым термометром DS1820

Пример	//Сброс DS1820, который подключен к выводу RA5:
	Ow_Reset(&PORTA, 5)

Таблица 42. Описание функции Ow_Read

Прототип	unsigned short Ow_Read(unsigned short *port, unsigned short pin)
Возвращаемое значение	Данные, прочитанные из внешнего устройства по шине OneWire
Описание	Читает один байт данных по шине OneWire
Требования	Нет
Пример	unsigned short tmp; ... tmp = Ow_Read(&PORTA, 5);

проводной шине, например, с цифровым термометром DS1820. Протокол этого интерфейса, требующего для связи всего один провод, поддерживает ведущий (master) и ведомый (slave) микроконтроллер. Благодаря используемой для этого интерфейса конфигурации аппаратуры (подтяжка к питанию и драйверы с открытым коллектором), он позволяет устройству slave получать питание по проводнику, используемому для связи.

Протокол интерфейса One Wire имеет следующие основные характеристики:

- только одно ведущее устройство в системе;
- скорость обмена до 16 Кбод;
- расстояние между устройствами до 300 м;
- небольшие пакеты данных;
- низкая стоимость.

Каждое устройство One Wire шины имеет уникальный 64-битовый регистрационный номер (8 бит – тип устройства, 48 бит – серийный номер и 8 бит – CRC), поэтому большое количество ведомых устройств могут сосуществовать на одной шине.

Следует заметить, что для работы с цифровыми термометрами фирмы Dallas частота тактового генератора микроконтроллер должна быть не менее 4 МГц.

Для работы с USART используются функции Ow_Reset, Ow_Read и Ow_Write. Описание этих функций представлено в таблицах 41 – 43 соответственно.

Следующий пример (см. дополнительные материалы на сайте журнала) программы демонстрирует чтение микроконтроллером значений температуры от цифрового датчика температуры DS1820, подключенного к выводу RA5, и вывод их на ЖКИ.

Схема подключения цифрового термометра DS1820 по шине One Wire к выводу RA5 микроконтроллера показана на рисунке 28.

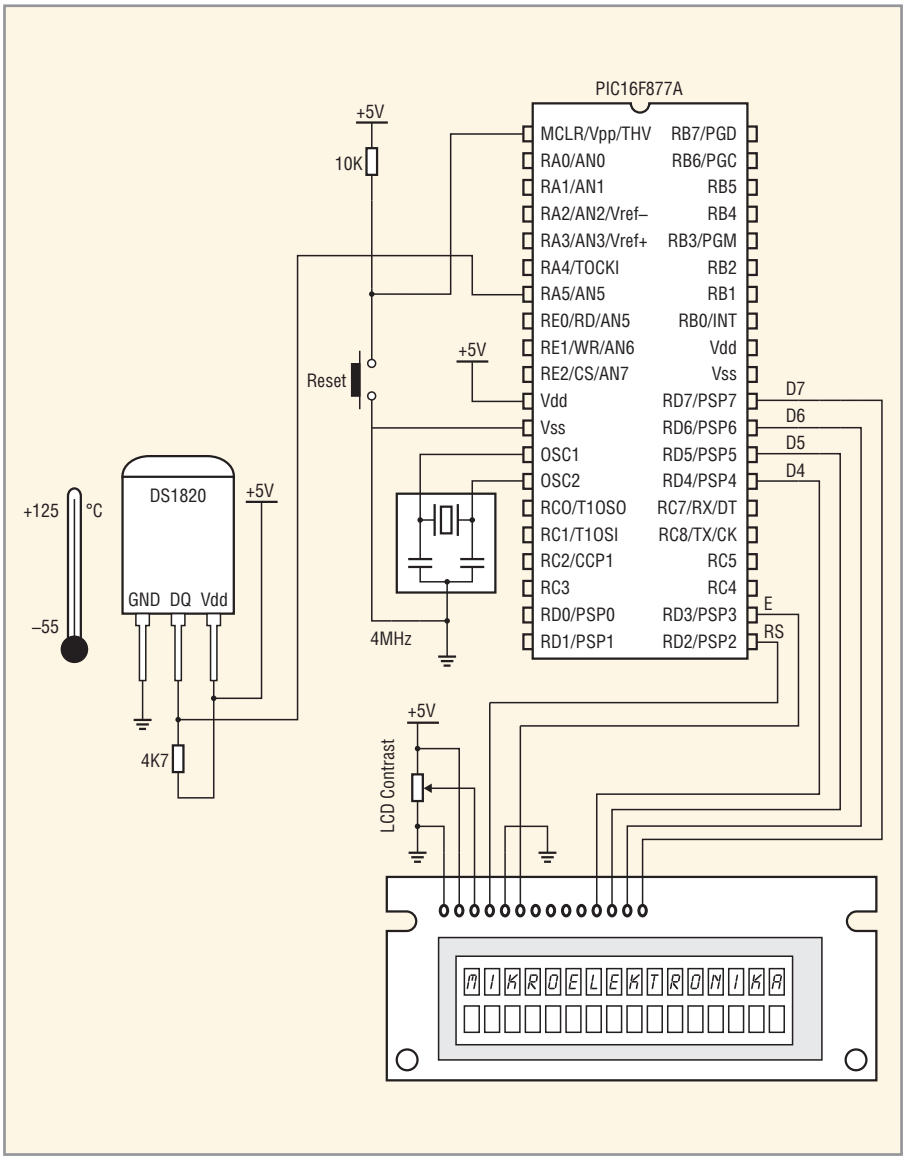

Рис. 28. Схема подключения цифрового термометра DS1820 к микроконтроллеру по шине One Wire

Функции для работы с аппаратной реализацией SPI-интерфейса

Аппаратный модуль SPI имеется в ряде МК семейства PIC. Среда разработки mikroC предоставляет библиотеку для инициализации модуля SPI и удобства работы с ним.

С помощью SPI-интерфейса можно легко связать микроконтроллер с внеш-

Таблица 43. Описание функции Ow_Write

Прототип	void Ow_Write(unsigned short *port, unsigned short pin, unsigned short par)
Возвращаемое значение	Нет
Описание	Передаёт один байт данных (аргумент par) по шине OneWire
Требования	Нет
Пример	Ow_Write(&PORTA, 5, 0xCC)

Таблица 44. Описание функции Spi_Init

Прототип	void Spi_Init(void)
Возвращаемое значение	Нет
Описание	Конфигурирование и инициализация SPI установками по умолчанию. Spi_Init_Advanced или Spi_Init должны вызываться перед использованием любых других функций библиотеки SPI. Установки по умолчанию: режим ведущего, частота синхронизации Fosc/4, низкий уровень при отсутствии синхросигнала на соответствующем выводе, передача по фронту, стробирование входных данных в середине интервала синхронизации. Для установки собственных настроек необходимо использовать функцию Spi_Init_Advanced
Требования	Требуется PIC-микроконтроллер с аппаратным модулем SPI
Пример	Spi_Init()

Таблица 45. Описание функции Spi_Init_Advanced

Прототип	void Spi_Init_Advanced(unsigned short master, unsigned short data_sample, unsigned short clock_idle, unsigned short transmit_edge)
Возвращаемое значение	Нет

Описание	Конфигурирование и инициализация SPI. Spi_Init_Advanced или Spi_Init должны вызываться перед использованием любых других функций библиотеки SPI. Аргумент mast_slav определяет режим работы SPI и может принимать следующие значения:
	MASTER_OSC_DIV4 // Ведущий, частота синхр.=Fosc/4
	MASTER_OSC_DIV16 // Ведущий, частота синхр.=Fosc/16
	MASTER_OSC_DIV64 // Ведущий, частота синхр.=Fosc/64
	MASTER_TMR2 // Ведущий, частоту синхр. задает TMR2
	SLAVE_SS_ENABLE // Ведомый, разрешен Slave select
	SLAVE_SS_DIS // Ведомый, запрещён Slave select
	Аргумент data_sample определяет, когда стробируются входные данные, и может принимать следующие значения:
	DATA_SAMPLE_MIDDLE // Входные данные стробируются
	// в середине интервала синхронизации
	DATA_SAMPLE_END // Входные данные стробируются
	// в конце интервала синхронизации
	Аргумент clock_idle определяет состояние вывода синхронизации при отсутствии обмена и может принимать следующие значения:
	CLK_IDLE_HIGH // Высокий уровень на выводе
	// синхронизации при отсутствии обмена
	CLK_IDLE_LOW // Низкий уровень на выводе
	// синхронизации при отсутствии обмена
	Аргумент transmit_edge может принимать следующие значения:
	LOW_2_HIGH // Данные передаются по фронту
	HIGH_2_LOW // Данные передаются по спаду

Требования	Требуется PIC-микроконтроллер с аппаратным модулем SPI
Пример	SPI устанавливается в режим ведущего с частотой синхронизации = Fosc/4, входные данные стробируются в середине интервала синхронизации, вывод синхронизации в состоянии низкого уровня при отсутствии обмена и данные передаются по фронту: Spi_Init_Advanced(MASTER_OSC_DIV4, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH)

Таблица 46. Описание функции Spi_Read

Прототип	unsigned short Spi_Read(unsigned short buffer)
Возвращаемое значение	Возвращает принятый байт
Описание	Запускает обмен посылкой байта buffer, принимает данные и возвращает принятый байт по окончанию приёма
Требования	Требуется PIC-микроконтроллер с аппаратным модулем SPI. SPI должен быть предварительно проинициализирован функциями Spi_Init_Advanced или Spi_Init
Пример	short take, buffer; ... take = Spi_Read(buffer);

Таблица 47. Описание функции Spi_Write

Прототип	void Spi_Write(unsigned short data)
Возвращаемое значение	Нет
Описание	Записывает передаваемый байт data в регистр данных SSPBUF, что немедленно вызывает передачу
Требования	Требуется PIC-микроконтроллер с аппаратным модулем SPI. SPI должен быть предварительно проинициализирован функциями Spi_Init_Advanced или Spi_Init
Пример	Spi_Write(1)

ними микросхемами АЦП, ЦАП, памяти и т.д. Для этого потребуется только МК с аппаратно реализованным модулем SPI (например, PIC16F877). Некоторые микроконтроллеры, имеющие два модуля SPI, например P18F8722, требуют предварительно определить модуль, который будет использован. Для этого достаточно добавить номер 1 или 2 к имени функции Spi. Например, Spi2_Write().

Также, с целью обратной совместимости с предыдущими версиями компилятора и облегчения управления кодами, микроконтроллеры с несколькими модулями SPI имеют Spi-библиотеку, которая идентична Spi1 (т.е. можно использовать Spi_Init() вместо Spi1_Init() для операций с SPI).

Библиотека SPI включает в свой состав следующие функции: Spi_Init, Spi_Init_Advanced, Spi_Read и Spi_Write. Описание этих функций приведено в таблицах 44 – 47 соответственно.

Следующий пример демонстрирует, как использовать функции библиотеки SPI. Предлагаемая аппаратная конфигурация: вывод выборки ведомого max7219 (драйвер 8-сегментного светодиодного дисплея на 8 позиций) соединён с RC1 PIC, выводы SDO и SCK PIC – с соответствующими выводами max7219, вывод SDI PIC не используется.

```
//------------------- Объявления
функций
void max7219_init1();
//------------------------------
- конец
unsigned short i;

void main() {
 Spi_Init(); // Стандартное кон-
фигурирование SPI
 //Вместо Spi_init можно исполь-
зовать Spi_Init_Advanced,
 // как показано в следующей
строке:

//Spi_Init_Advanced(MASTER_OSC_DI
V64, DATA_SAMPLE_MIDDLE,
 // CLK_IDLE_HIGH, HIGH_2_LOW);
 TRISC &= 0xFD;
 max7219_init1(); // Инициализа-
ция max7219
 for (i = 1; i <= 8u; i++) {
 PORTC &= 0xFD; // Выбор max7219
 Spi_Write(i); // Посылка i в
max7219 (номер позиции)
 Spi_Write(8 - i); // Посылка 8-i
в max7219 (цифра в этой позиции)
 PORTC |= 2; // Отмена выбора
max7219
 }
 TRISB = 0;
 PORTB = i;
}//end
```

Заключение

Теперь разработчики программ для микроконтроллерных устройств могут воспользоваться средой mikroC компании MikroElektronika, имеющей в своём составе множество готовых функций, позволяющих использовать всю внутреннюю архитектуру МК с многочисленными типами интерфейсов и различные стандартизованные внешние устройства.

Литература

1. www. mikroe.com.
2. www.microchip.com.