НАВИГАЦИЯ

Глава II

Порты ввода-вывода общего назначения

Цель работы:

– получение навыков работы с цифровыми сигналами;
– изучение портов ввода-вывода общего назначения;
– приобретение опыта работы с жидкокристаллическим индикатором.

Оборудование:

– отладочный комплект для микроконтроллера 1986ВЕ92У;
– программатор-отладчик J-LINK (или аналог);
– персональный компьютер.

Программное обеспечение:

– операционная система Windows 7 / 8 / 10;
– среда программирования Keil µVision MDK-ARM 5.24;
– драйвер программатора J-LINK;
– примеры кода программ.

2.1. Подготовка к работе

Начиная с этой главы будет широко использоваться **метод обратной разработки**, идея которого заключается в исследовании некоторого готового программного проекта с целью понять принцип его работы.

Перед непосредственной работой с данными проектами рекомендуется создать на локальном диске папку с именем *Backup* и скопировать в нее папку *Samples* со всем ее содержимым. Такой процесс называется **резервным копированием** и позволяет восстановить исходные файлы в случае необходимости.

2.2. Порты ввода-вывода общего назначения

Порты ввода-вывода общего назначения (англ. General-Purpose Input/Output, **GPIO**) – это основной интерфейс для связи между микроконтроллером и различными внешними устройствами.

Порты состоят из **параллельных линий**, каждая из которых может использоваться для ввода или вывода информации. Порты могут быть ориентированы как на цифровые сигналы, так и на аналоговые. Наиболее часто используются цифровые режимы портов; аналоговые задействуют лишь при работе с аналого-цифровыми и цифро-аналоговыми преобразователями, компараторами и внешними кварцевыми резонаторами.

Линии, сконфигурированные как **цифровой ввод**, используют для подключения к системе механических кнопок, электромагнитных реле или приема цифрового сигнала. Иными словами, **для ввода сигналов с внешних устройств в микроконтроллер**.

Линии, сконфигурированные как **цифровой вывод**, используют для формирования управляющих воздействий. Это позволяет, например, зажечь светодиод или лампу накаливания, запустить электродвигатель, передать логическую последовательность.

Иными словами, такие линии предназначены **для вывода сигналов из микроконтроллера на внешние устройства.**

Почти все линии микроконтроллеров серии 1986BE9x могут функционировать в режиме цифрового ввода или вывода (но не ограничены этим). Функция каждой линии задается программно. Для удобства использования линии ввода-вывода на аппаратном уровне объединены в порты. Порты именованы латинскими литерами: *PORTA* (порт A), *PORTB* (порт B), *PORTC* (порт C) и т.д. Линии при этом нередко называют так, что, например, нулевая линия порта C – это *PC0,* а пятая линия порта F – *PF5*, и т.д.

Для примера, микроконтроллер 1986BE91T имеет 96 линий ввода-вывода, разделенные на 6 портов: *A*...*F*; по 16 линий в каждом. При этом с целевым микроконтроллером 1986BE92У ситуация несколько иная: он имеет 43 линии, распределенные по портам согласно таблице 2.1.

Таблица 2.1 – Распределение линий ввода-вывода по портам в микроконтроллере 1986BE92У

Наименование порта	**Количество линий**	**Наименование линий**
PORTA	8	PA0…PA7
PORTB	11	PB0…PB10
PORTC	3	PC0…PC2
PORTD	8	PD0…PD7
PORTE	6	PE0…PE3, PE6…PE7
PORTF	7	PF0…PF6

Схема расположения выводов микроконтроллера 1986BE92У (корпус H18.64-1B) показана на рисунке 2.1. Следует иметь в виду, что для микроконтроллера в пластмассовом корпусе (LQFP64) расположение выводов будет несколько отличаться.

Функциональная схема отдельной линии ввода-вывода изображена на рисунке 2.2. Все элементы схемы, за исключением контактного, расположены **внутри** микроконтроллера.

Два быстродействующих диода (один из них подключается к цепи питания, другой – к земле) предназначены для ограничения входного напряжения. Резистор *ESD R* (англ. Electric Static Discharge) предназначен для ограничения входного тока. Чрезмерные значения напряжения и тока могут возникнуть на линиях вследствие разных причин: ошибок при разработке схемы, выхода части схемы из строя, статического электричества и т.д. Такое несложное схемотехническое решение позволяет значительно повысить долговечность микросхемы.

Цифровые выводы микроконтроллеров серии 1986BE9x способны выдавать ток до ~6 мА каждый (до ~100 мА суммарно). Это позволяет, например, подключить к выводам яркие светодиоды. Однако для управления элементами с большей нагрузкой (лампы, приводы, двигатели) необходимо использовать дополнительные электрические схемы.

Назначение остальных элементов схемы будет рассмотрено в процессе программной конфигурации линий ввода-вывода.

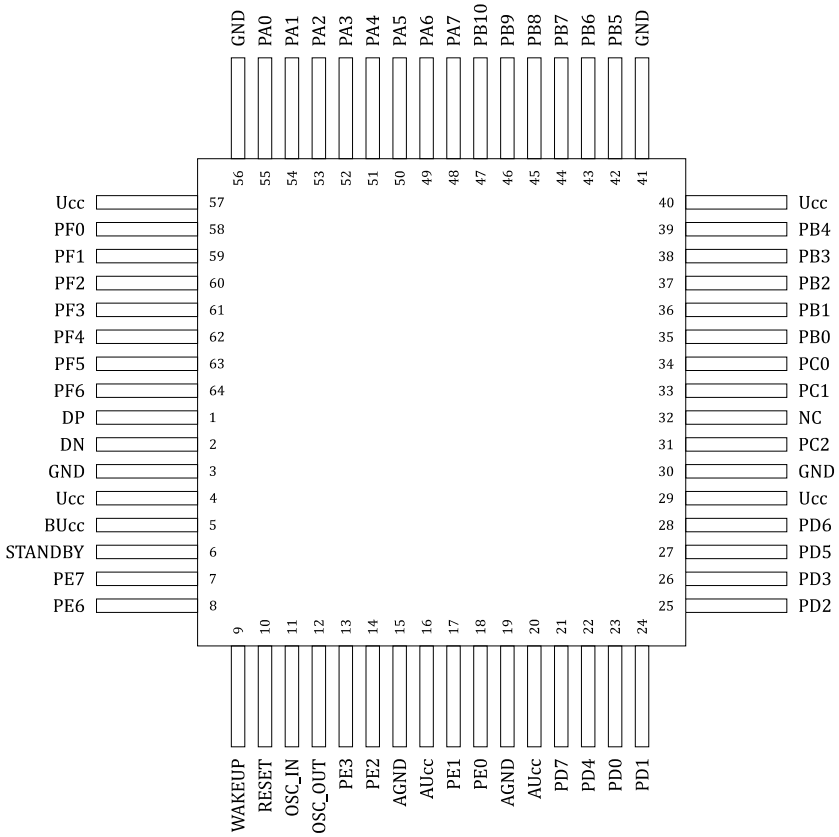


Рисунок 2.1 – Схема расположения выводов микроконтроллера 1986ВЕ92У

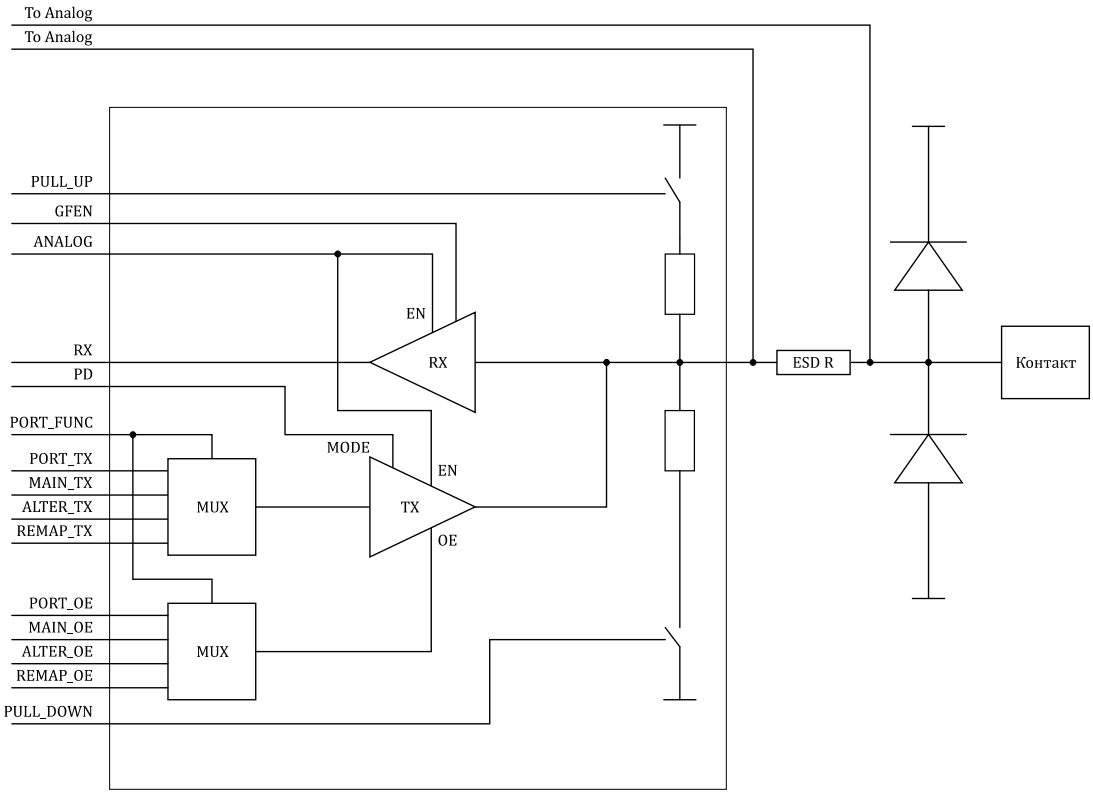


Рисунок 2.2 – Функциональная схема отдельной линии ввода-вывода

2.3. Конфигурация портов ввода-вывода

Порты ввода-вывода в микроконтроллерах серии 1986BE9x имеют широкий функционал, поэтому перед использованием должны быть соответствующим образом сконфигурированы.

Для каждого периферийного блока в адресном пространстве микроконтроллера существует область, предназначенная для работы с ним. Адресное пространство микроконтроллера подробно описывается официальной документацией; в частности, адреса портов приведены в таблице 2.2.

Таблица 2.2 – Базовые адреса портов ввода-вывода

Базовый адрес	Условное наименование	Описание
0x400A8000	MDR_PORTA	Порт A
0x400B0000	MDR_PORTB	Порт B
0x400B8000	MDR_PORTC	Порт C
0x400C0000	MDR_PORTD	Порт D
0x400C8000	MDR_PORTE	Порт E
0x400E8000	MDR_PORTF	Порт F

Конфигурация блока заключается в записи требуемых значений в определенные ячейки памяти (**регистры**). Для каждого порта ввода-вывода таких регистров выделено 8 (таблица 2.3). Адреса регистров задаются в виде байтового смещения относительно базового адреса. Поскольку регистры 32-разрядные, типовое значение смещения равно 4 байтам.

Таблица 2.3 – Регистры управления портами ввода-вывода

Адресное смещение	Условное наименование	Описание
0x00	RXTX	Состояние
0x04	OE	Направление передачи
0x08	FUNC	Функция
0x0C	ANALOG	Режим работы
0x10	PULL	Подтяжка
0x14	PD	Управление выводом
0x18	PWR	Крутизна фронтов
0x1C	GFEN	Цифровой фильтр

Таким образом, чтобы, например, включить цифровой фильтр на пятой линии порта B (назначение цифрового фильтра будет рассмотрено далее), следует записать единицу в пятый бит регистра, расположенного по адресу 0x400B001C (базовый адрес порта B + смещение на 28 (0x1C) байт). Программно это можно реализовать так:

```
// Включение цифрового фильтра на линии PB5
*(uint32_t *)(0x400B001C) |= (1 << 5);
```

Такая запись, однако, неочевидна и сложна для восприятия. По этой причине был разработан глобальный заголовочный файл ***1986VE9x.h***. Этот файл содержит адреса всех периферийных блоков микроконтроллера в виде констант. Регистры в нем реализованы в виде полей структуры, к примеру:

```
// Структура портов ввода-вывода
typedef struct
{
  __IO uint32_t RXTX;
  __IO uint32_t OE;
  __IO uint32_t FUNC;
  __IO uint32_t ANALOG;
  __IO uint32_t PULL;
  __IO uint32_t PD;
  __IO uint32_t PWR;
  __IO uint32_t GFEN;
} MDR_PORT_TypeDef;
```

Используя возможности файла *1986VE9x.h*, описанную выше настройку можно выполнить таким образом:

```
// Включение цифрового фильтра на линии PB5
MDR_PORTB->GFEN |= (1 << 5);
```

Кроме того, в файле также реализованы **позиционные константы битовых полей**. Эти константы содержат информацию о позиции тех или иных функциональных битов в регистре. С их помощью битовый сдвиг можно делать не на какое-то число, а на позиционную константу:

```
// Включение цифрового фильтра на линии PB5
MDR_PORTB->GFEN |= (1 << PORT_GFEN5_Pos);
```

Информативность такого вида записи гораздо выше, чем в первом варианте; при наличии небольшого опыта работы с микроконтроллерами программный код воспринимается даже без комментариев.

В рамках данной главы в учебных целях будет использоваться цифровая запись сдвигов; позиционные константы будут широко использованы в дальнейшем.

Далее рассмотрим структуру и назначение регистров, приведенных в таблице 2.3. Однако **важно помнить**, что перед тем, как приступать к конфигурации какого-либо периферийного блока, в том числе порта, нужно **включить его тактирование**:

```
// Включение тактирования порта A
MDR_RST_CLK->PER_CLOCK |= (1 << RST_CLK_PCLK_PORTA_Pos);
```

Сейчас эту информацию следует принять как данность; подробное рассмотрение системы тактирования будет приведено в следующей главе.

Регистр RXTX

Таблица 2.4 – Описание регистра *MDR_PORTx->RXTX*

№ битов	Функциональное имя битов	Описание
31...16	–	–
15...0	RXTX[15...0]	Состояние линий порта: 0 – низкий логический уровень (0 В); 1 – высокий логический уровень (3.3 В).

Регистр *RXTX* отображает состояние всех линий порта. Обращение к этому регистру следует выполнять только после того, как целевые линии будут полностью сконфигурированы, т.е. должным образом настроены остальные регистры порта.

Если линия выполняет функцию цифрового вывода, то запись значения в соответствующий ей бит регистра *RXTX* позволяет управлять ее состоянием. К примеру, для создания напряжения на линии PA2 можно написать так:

```
// Создание высокого логического уровня на линии PA2
MDR_PORTA->RXTX |= (1 << 2);
```

Если линия работает как цифровой ввод, то чтение значения соответствующего ей бита регистра *RXTX* позволяет определить логический уровень внешнего сигнала. Например, сигнал с линии PA3 можно сохранить в переменную таким образом:

```
// Определение логического уровня на линии PA3
uint8_t data = (MDR_PORTA->RXTX >> 3) & 0x01;
// (Сдвиг вправо на 3 бита позволяет перенести целевой бит в младший разряд,
// а операция «&» со значением 0x01 маскирует старшие биты)
```

Регистр OE

Таблица 2.5 – Описание регистра *MDR_PORTx->OE*

№ битов	Функциональное имя битов	Описание
31...16	–	–
15...0	OE[15...0]	Направление линий: 0 – ввод данных; 1 – вывод данных.

Регистр *OE* определяет направление передачи данных по линии. Чтобы, к примеру, использовать линию PA2 для вывода данных, а линию PA3 – для ввода, следует выполнить такую настройку:

```
MDR_PORTA->OE |= (1 << 2);    // Настройка линии PA2 на вывод данных
MDR_PORTA->OE &= ~(1 << 3);   // Настройка линии PA3 на ввод данных
```

Регистр FUNC

Таблица 2.6 – Описание регистра *MDR_PORTx->FUNC*

№ битов	Функциональное имя битов	Описание
31…2	FUNCx[31…2]	Аналогично FUNC0 для остальных линий.
1…0	FUNC0[1…0]	Функция линии 0: 00 – ввод/вывод общего назначения; 01 – основная; 10 – альтернативная; 11 – переопределенная.

Регистр *FUNC* задает функцию линии ввода/вывода. Дело в том, что каждая линия микроконтроллеров серии 1986BE9x может быть не только цифровым вводом или выводом, но и выполнять функции различных периферийных блоков. Например, линия PD0 может быть использована в качестве инверсного канала таймера 1, прямого канала таймера 3, линии приема контроллера UART2, а также нулевого канала АЦП в случае использования аналогового режима (об этом далее). Данные функции условно именуют **основной**, **альтернативной**, **переопределенной**. Распределение функций по линиям приведено в приложении в **таблице П.1** (к этой таблице в процессе работы потребуется обращаться достаточно часто).

Структура регистра *FUNC* такова, что для выбора функции каждой линии в нем отведено 2 бита. В частности, для линии PA0 – это биты 0 и 1, для линии PA1 – биты 2 и 3, и т.д. Таким образом, чтобы сконфигурировать линию PA3 как ввод/вывод общего назначения (00_2 согласно таблице 2.6) нужно указать так:

```
// Настройка линии PA3 для работы в качестве ввода/вывода общего назначения
MDR_PORTA->FUNC &= ~(3 << 6);
// (Для каждой линии предназначено два бита, а целевой является линия 3,
// поэтому выполняется сдвиг влево на 2 * 3 = 6 битов;
// значение 3₁₀ = 11₂ используется для того, чтобы сбросить два бита)
```

При этом есть один нюанс: если требуется задать основную или альтернативную функцию (значения 01_2 и 10_2 соответственно), то это следует делать в две операции: сначала сбросить оба бита в ноль, а затем установить требуемый бит в единицу:

```
// Задание альтернативной функции для линии PA3
MDR_PORTA->FUNC &= ~(3 << 6);  // Сброс двух битов
MDR_PORTA->FUNC |=  (2 << 6);  // Установка требуемого бита
```

Если ограничиться только установкой битов, то во многих случаях нельзя гарантировать верное значения этого параметра.

Регистр ANALOG

Таблица 2.7 – Описание регистра *MDR_PORTx->ANALOG*

№ битов	Функциональное имя битов	Описание
31...16	–	–
15...0	ANALOG[15...0]	Режим работы линии: 0 – аналоговый; 1 – цифровой.

Регистр *ANALOG* определяет режим работы линии: аналоговый или цифровой. В большинстве случаев используется цифровой режим; аналоговый режим выбирают лишь при работе с аналого-цифровым и цифро-аналоговым преобразователями, а также компаратором.

Для примера, сделать линию PA3 цифровой можно так:

```
// Выбор цифрового режима для линии PA3
MDR_PORTA->ANALOG |= (1 << 3);
```

Регистр PULL

Таблица 2.8 – Описание регистра *MDR_PORTx->PULL*

№ битов	Функциональное имя битов	Описание
31...16	PULL_UP[31...16]	Подтяжка линии к потенциалу питания (3.3 В): 0 – подтяжка отключена; 1 – подтяжка включена.
15...0	PULL_DOWN[15...0]	Подтяжка линии к потенциалу земли (0 В): 0 – подтяжка отключена; 1 – подтяжка включена.

Регистр *PULL* управляет подключением линий к земле либо цепи питания через специальные **подтягивающие резисторы** номиналом ~50 КОм.

В целом, подтяжка повсеместно используется в электрических схемах для гарантии определенного устойчивого состояния на линии в случаях, когда управляющий элемент отключен или его выводы находятся в высокоимпедансном состоянии (т.н. «открытый сток»).

Идейно, наличие таких резисторов непосредственно внутри микроконтроллера позволяет снизить размер печатной платы разрабатываемого устройства, количество требуемых электронных компонентов, число паек и стоимость изделия в целом.

Однако с учетом относительно небольшого вырабатываемого тока цифровых выводов и высокого значения сопротивления этих резисторов, им трудно найти практическое применение, и в большинстве случаев их отключают, а для подтяжки задействуют внешние элементы.

Структура регистра *PULL* разбита на два блока: младшие 16 битов отвечают за подтяжку линий к потенциалу земли, старшие 16 битов – за подтяжку к потенциалу питания. В этом случае для отключения подтяжек линии PA3 можно записать так:

```
// Отключение подтяжек линии PA3
MDR_PORTA->PULL &= ~(1 << 3);    // Подтяжка к земле
MDR_PORTA->PULL &= ~(1 << 19);   // Подтяжка к питанию
```

Конечно, поскольку операция производится с одним регистром, ее можно выполнить в одно действие. Такая запись используются для наглядности.

Регистр PD

Таблица 2.9 – Описание регистра *MDR_PORTx->PD*

№ битов	Функциональное имя битов	Описание
31...16	SHM[31...16]	Триггер Шмитта: 0 – отключен; 1 – включен.
15...0	PD[15...0]	Управление линией: 0 – драйвер; 1 – открытый сток.

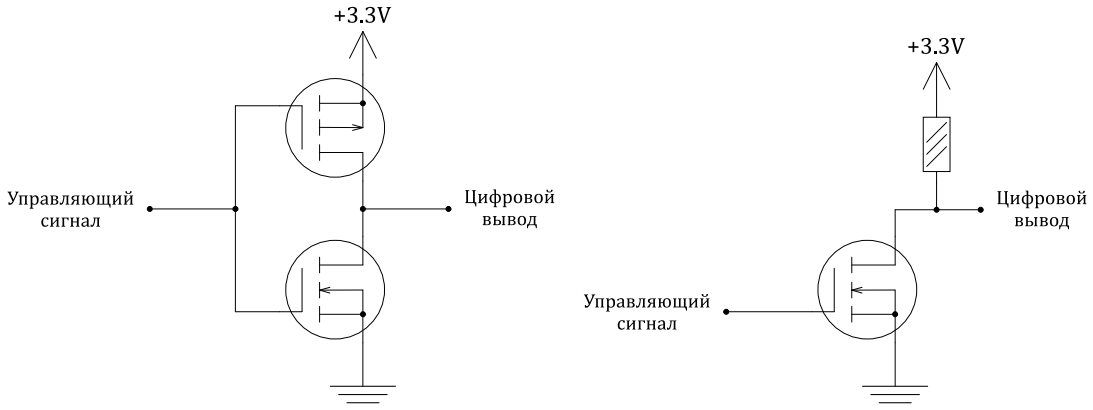
Регистр *PD* разделен на два функциональных блока.

Одноименный блок *PD* занимает 16 младших битов и определяет режим работы **цифрового вывода**. Реализовано два режима: «драйвер» и «открытый сток». Следует отметить, что этот параметр не используется для линий, настроенных на ввод данных.

Режим «драйвер» является стандартным режимом работы цифрового вывода. В этом режиме линия подключается через два транзистора, один из которых соединен с землей, другой – с питанием (рисунок 2.3а). В каждый момент времени активен только один транзистор, что позволяет создать два устойчивых состояния – низкое и высокое.

В режиме «открытый сток» линия подключается к затвору одного транзистора, исток которого соединен с землей, а сток, соответственно, открыт для соединения с внешней схемой. Таким образом, вывод может находиться в двух состояниях: низком, если транзистор открыт; и **высокоимпедансном** (англ. Hi-Z), если транзистор закрыт. В высокоимпедансном состоянии уровень сигнала на линии не соответствует ни логическому нулю, ни логической единице, поэтому данное состояние еще называют **третьим**. По этой причине данный режим обычно используют для работы в схемах с подтяжкой к питанию (рисунок 2.3б).

Блок *SHM* занимает 16 старших битов и позволяет подключить к **цифровому вводу** триггер Шмитта. Данный триггер идейно создает на линии **петлю гистерезиса**. Но практическая значимость этого параметра в микроконтроллерах серии 1986BE9x довольно мала (подробности в разделе 2.4), поэтому триггер можно отключать без сомнений.



а) б)

Рисунок 2.3 – Схемы подключения цифрового вывода:
а) в режиме «драйвер»; б) в режиме «открытый сток» с подтяжкой к питанию

В итоге, типичная конфигурация линии PA2 в качестве цифрового вывода, управляемого драйвером, и линии PA3 в качестве цифрового ввода может выглядеть так:

```
// Управление линией PA2 (цифровой вывод)
MDR_PORTA->PD &= ~(1 << 2);   // Управление (драйвер)
MDR_PORTA->PD &= ~(1 << 18);  // Триггер Шмитта (не используется)

// Управление линией PA3 (цифровой ввод)
MDR_PORTA->PD &= ~(1 << 3);   // Управление (не используется)
MDR_PORTA->PD &= ~(1 << 19);  // Триггер Шмитта (отключен)
```

Регистр PWR

Таблица 2.10 – Описание регистра *MDR_PORTx->PWR*

№ битов	Функциональное имя битов	Описание
31...2	PWRx[31...2]	Аналогично PWR0 для остальных линий.
1...0	PWR0[1...0]	Крутизна фронтов выходных импульсов: 00 – передача отключена; 01 – низкая крутизна (длительность фронта ~100 нс); 10 – средняя крутизна (длительность фронта ~20 нс); 11 – высокая крутизна (длительность фронта ~10 нс).

Регистр *PWR* определяет **крутизну фронтов** выходных импульсов **цифрового вывода**. Дело в том, что прямоугольные импульсы, широко используемые в цифровой электронике, в действительности всегда являются трапецеидальными. То, насколько эти импульсы близки к прямоугольной форме, определяется их длительностью и временем переходного процесса: чем меньше время переходного процесса, тем более крутыми становятся фронты импульсов.

На рисунке 2.4 приведена сравнительная осциллограмма прямоугольных импульсов на частоте 2.25 МГц с низкой и высокой крутизной фронтов. Как видно из рисунка, высокая крутизна фронтов обеспечивает большую длительность сигнала (на инженерном

жаргоне этот период называют «полка»). При этом, однако, возникает колебательность переходного процесса, создающая широкий спектр помех. В целом, высокая крутизна импульсов необходима при работе на высоких частотах (более ~5 МГц); например, при передаче данных по высокоскоростному интерфейсу. В остальных же случаях рациональнее будет использовать низкую крутизну.

Рисунок 2.4 – Осциллограмма прямоугольных импульсов:
сверху – низкая крутизна фронтов, снизу – высокая крутизна фронтов

Для определения мощности передачи для каждой линии в структуре регистра *PWR* отведено 2 бита. По аналогии с работой с регистром *FUNC*, для задания, к примеру, низкой крутизны фронтов выходных импульсов на линии PA2 следует указать:

```
// Установка низкой крутизны фронтов выходных импульсов на линии PA2
MDR_PORTA->PWR &= ~(3 << 4);  // Сброс двух битов
MDR_PORTA->PWR |= (1 << 4);   // Установка требуемого бита
```

Регистр GFEN

Таблица 2.11 – Описание регистра *MDR_PORTx->GFEN*

№ битов	Функциональное имя битов	Описание
31...16	–	–
15...0	GFEN[15...0]	Цифровой фильтр: 0 – отключен; 1 – включен.

Регистр *GFEN* позволяет подключить к **цифровому вводу** фильтр, который будет отсеивать входные импульсы длительностью **менее 10 нс**. Т.е. если на ввод с таким фильтром попадет указанный импульс, то он будет интерпретирован, как помеха, и не изменит логическое состояние линии.

Цифровой фильтр значительно увеличивает помехозащищенность приемного тракта, однако его не следует использовать при работе с высокочастотными сигналами (более ~25 МГц), т.к. это может привести к утрате информационной составляющей.

Чтобы включить фильтрацию на линии PA3 достаточно написать:

```
// Включение цифрового фильтра на линии PA3
MDR_PORTA->GFEN |= (1 << 3);
```

*
**

Рассмотрение регистров портов ввода-вывода на этом завершено. Конфигурацию линий обычно проводят единым программным блоком по функциональному назначению; в заключение приведем два типовых блока: один для конфигурации линии PB4 в качестве цифрового вывода, второй – линии PC2 в качестве цифрового ввода:

```
// Включение тактирования порта B
MDR_RST_CLK->PER_CLOCK |= (1 << RST_CLK_PCLK_PORTB_Pos);

// Конфигурация линии PB4 в качестве цифрового вывода
MDR_PORTB->OE     |=  (1 << 4);   // Направление данных (вывод)
MDR_PORTB->PULL   &= ~(1 << 4);   // Подтяжка к земле (отключена)
MDR_PORTB->PULL   &= ~(1 << 20);  // Подтяжка к питанию (отключена)
MDR_PORTB->ANALOG |=  (1 << 4);   // Режим работы линии (цифровой)
MDR_PORTB->FUNC   &= ~(3 << 8);   // Функция линии (ввод-вывод)
MDR_PORTB->PD     &= ~(1 << 4);   // Управление линией (драйвер)
MDR_PORTB->PD     &= ~(1 << 20);  // Триггер Шмитта (не используется)
MDR_PORTB->PWR    &= ~(3 << 8);   // Сброс битов регистра PWR
MDR_PORTB->PWR    |=  (3 << 8);   // Крутизна фронтов (низкая)
MDR_PORTB->GFEN   &= ~(1 << 4);   // Цифровой фильтр (не используется)

// Включение тактирования порта C
MDR_RST_CLK->PER_CLOCK |= (1 << RST_CLK_PCLK_PORTC_Pos);

// Конфигурация линии PC2 в качестве цифрового ввода
MDR_PORTC->OE     &= ~(1 << 2);   // Направление данных (ввод)
MDR_PORTC->PULL   &= ~(1 << 2);   // Подтяжка к земле (отключена)
MDR_PORTC->PULL   &= ~(1 << 18);  // Подтяжка к питанию (отключена)
MDR_PORTC->ANALOG |=  (1 << 2);   // Режим работы линии (цифровой)
MDR_PORTC->FUNC   &= ~(3 << 4);   // Функция линии (ввод-вывод)
MDR_PORTC->PD     &= ~(1 << 2);   // Управление линией (не используется)
MDR_PORTC->PD     &= ~(1 << 18);  // Триггер Шмитта (отключен)
MDR_PORTC->PWR    &= ~(3 << 4);   // Крутизна фронтов (не используется)
MDR_PORTC->GFEN   |=  (1 << 2);   // Цифровой фильтр (включен)
```

2.4. Гистерезис

Гистерезис – это свойство системы, представляющее собой зависимость ее отклика не только от входного сигнала, но и от ее текущего состояния. В электротехнике гистерезис используют для стабилизации цифровых значений и снижения влияния электрических флуктуаций.

На рисунке 2.5 изображено три графика: первый представляет собой входной аналоговый сигнал, второй – двоичную интерпретацию сигнала без использования гистерезиса, третий – с использованием гистерезиса.

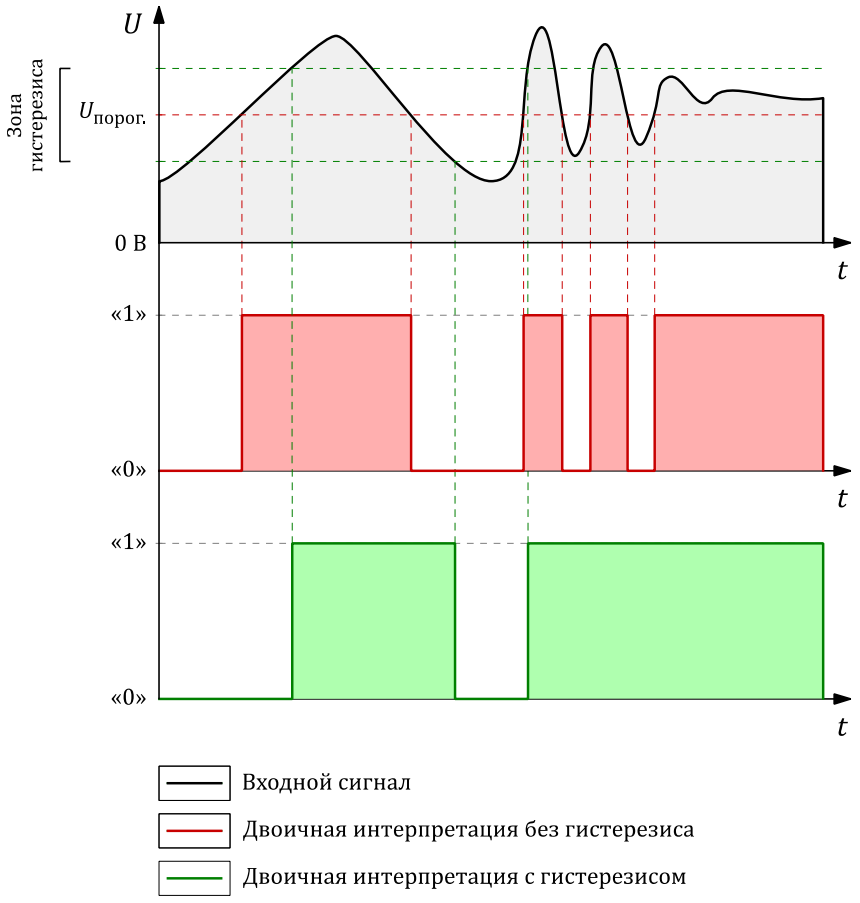


Рисунок 2.5 – Зависимость состояния системы от входного сигнала при отсутствии и наличии гистерезиса

Как видно из рисунка 2.5, при отсутствии гистерезиса изменение состояния системы происходит при пересечении сигналом значения $U_{порог.}$ В таком случае, колебания входного сигнала в окрестностях этого значения повлекут за собой и колебания логического уровня.

Гистерезис создает в системе два порога изменения состояния:

– ($U_{порог.} + \Delta U$) при «0» → «1»;

– ($U_{порог.} - \Delta U$) при «1» → «0».

Удвоенное значение ΔU называют **шириной зоны гистерезиса**. Чем шире эта зона, тем выше стабильность системы, но ниже чувствительность к изменению входного сигнала.

Согласно документации, в микроконтроллерах серии 1986ВЕ9х, основанных на КМОП-логике, используются следующие диапазоны напряжений:

– 0.0 В...0.8 В – логический ноль;

– 2.0 В...3.6 В – логическая единица.

В действительности, однако, переключение осуществляется при пересечении значения ~1.6 В. Зона гистерезиса практически отсутствует. Подключение к линии интегрированного триггера Шмитта не дает значимого результата. Это, в целом, не большая проблема, но при работе с внешним сигналом, подверженным сильным помехам, рекомендуется устанавливать внешний триггер Шмитта, либо реализовывать гистерезис программно.

2.5. Светодиоды

Светодиоды (англ. Light-Emitting Diode, **LED**) широко используются в современных электронных устройствах в целях индикации. Они гораздо долговечнее и экономичнее ламп накаливания: светодиоды ярко горят при сравнительно небольшом токе (1...20 мА) и напряжении (1.8...3.2 В). Существую светодиоды разных цветов, яркости, размера и формы.

На рисунке 2.6 приведен фрагмент электрической схемы отладочной платы, на котором изображено подключение двух светодиодов; работа с ними осуществлялась в рамках главы I.

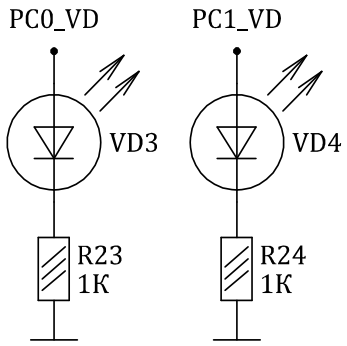


Рисунок 2.6 – Электрическая схема подключения светодиодов

Согласно схеме, светодиоды подключены непосредственно к линиям PC0 и PC1 микроконтроллера с использованием токоограничивающих резисторов; такие резисторы в схеме необходимы, иначе ток через светодиоды будет слишком большим: это практически не увеличит яркость, но значительно снизит срок службы светодиодов и перегрузит порт микроконтроллера.

Если на линии PC0 или PC1 (сконфигурированной, как **цифровой вывод**) установить логическую единицу, то на аноде соответствующего светодиода появится электрический потенциал, в то время как катод будет иметь потенциал земли; в результате протекающий через светодиод ток создаст световое излучение. Если же на линии установить логический ноль, то разность потенциалов между анодом и катодом светодиода будет равна 0 В, и ток через него не потечет.

Необходимо отметить, что на отладочной плате для микроконтроллера 1986ВЕ92У **линии PC0 и PC1 также используются в качестве шины данных для работы с жидкокристаллическим дисплеем**. Это объясняется ограниченным количеством выводов – нет ни одного свободного вывода, все заняты какими-либо функциями отладочной платы. По причине такого схемотехнического решения **одновременно работать с интегрированными в плату светодиодами и дисплеем невозможно**.

Жидкокристаллический дисплей является очень удобным инструментом для отображения информации. Поэтому для световой индикации рекомендуется использовать внешние по отношению к отладочной плате светодиоды. С этой целью на **модуле расширения** реализован набор из пяти разноцветных светодиодов (раздел 2.9).

Поскольку все линии микроконтроллера, как уже было сказано, заняты различными функциями отладочной платы и самого микроконтроллера, то для подключения внешних устройств потребуется отказаться от каких-то неиспользуемых или малозначительных функций. Определять такие линии следует по таблице П.1. Для подключения светодиодов, например, можно задействовать линии PB0...PB4, отведенные под работу интерфейса JTAG-A (при условии, конечно, что программирование микроконтроллера производится по интерфейсу JTAG-B).

2.6. Механические кнопки

Широкое распространение в микроконтроллерных системах имеют механические кнопки, переключатели и иные механические коммутаторы. С их помощью пользователь может иметь возможность управления системой.

Кнопки обычно подключают к цифровым вводам микроконтроллера. На рисунке 2.7 изображен фрагмент электрической схемы отладочной платы, демонстрирующий подключение кнопок *UP*, *DOWN*, *LEFT*, *RIGHT* и *SELECT*.

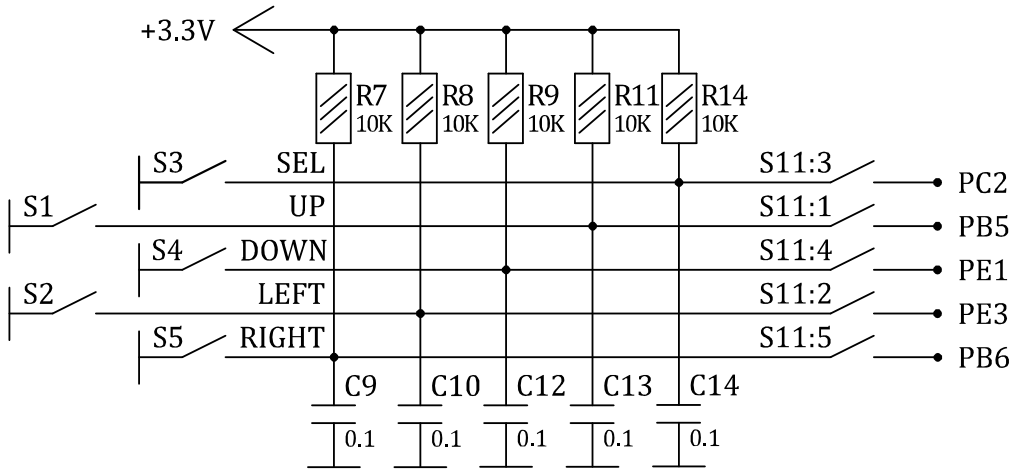


Рисунок 2.7 – Электрическая схема подключения механических кнопок

Кнопки на схеме обозначены ключами *S1*...*S5*. При этом к линиям ввода-вывода микроконтроллера кнопки подключены через переключатель *S11:1*...*S11:5*; это позволяет отключить от линий резисторы и конденсаторы при использовании линий по иному назначению. Данный переключатель расположен **с обратной стороны отладочной платы** (рисунок 2.8). Перед использованием механической кнопки следует убедиться, что соответствующий ей ключ замкнут, т.е. находится в положении *ON*. Для изменения состояния переключателя может потребоваться острый предмет: щуп мультиметра или осциллографа, скрепка или т.п.

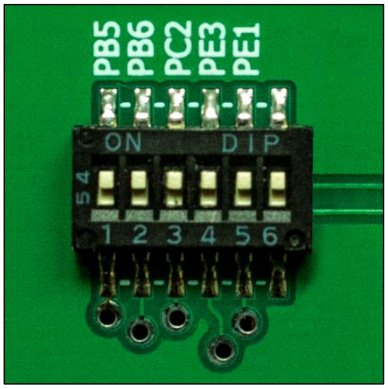


Рисунок 2.8 – Переключатель, коммутирующий механические кнопки с линиями ввода-вывода микроконтроллера

2.6.1. Принцип работы

Рассмотрим принцип работы механических кнопок на примере кнопки *SELECT*.

В соответствии с рисунком 2.7, в исходном состоянии (когда кнопка НЕ нажата) ключ *S3* разомкнут, а линия *PC2* подтянута к цепи питания через резистор *R14*. В это время на линии устанавливается **высокий логический уровень**.

При нажатии кнопки *SELECT* ключ *S3* замыкается, подтягивая линию к земле. В этот момент на линии устанавливается **низкий логический уровень**.

Таким образом, применительно к микроконтроллеру работа с кнопкой сводится к периодической проверке состояния линии; если на линии обнаруживается логический ноль, то это следует интерпретировать, как нажатие кнопки; логическая единица – отсутствие нажатия:

```
while (1)
{
  // Проверка нажатия кнопки SEL (PC2)
  if (MDR_PORTC->RXTX & (1 << 2) == 0)
  {
    // Выполнение соответствующих операций
    ...
  }

  // Задержка перед началом следующей проверки (~100 мс)
  DELAY(100);
}
```

Обратите внимание, что проверка нажатия в большинстве случаев должна сопровождаться некоторой задержкой, иначе она будет осуществляться с частотой процессорного ядра, что приведет к многократной неконтролируемой обработке. Значение задержки следует выбирать таким образом, чтобы создать оптимальную частоту обработки нажатий. Чем больше время задержки, тем дольше потребуется держать кнопку, чтобы система обработала нажатие. С другой стороны, чем меньше время задержки, тем больше процедур обработки система может выполнить в период однократного нажатия. На практике обычно используются частоту обработки 10...20 Гц и задержку 50...100 мс соответственно.

Альтернативой задержке может быть программное ожидание отпускания кнопки:

```
while (1)
{
  // Проверка нажатия кнопки SEL (PC2)
  if (MDR_PORTC->RXTX & (1 << 2) == 0)
  {
    // Выполнение соответствующих операций
    ...

    // Ожидание отпускания кнопки SEL (PC2)
    while (MDR_PORTC->RXTX & (1 << 2) == 0);
  }
}
```

Такой алгоритм стабильно обрабатывает однократные нажатия (при условии отсутствия дребезга контактов), но не позволяет осуществлять «прокрутку зажатием кнопки», что может быть неудобно при работе с большими списками.

2.6.2. Дребезг контактов

Замыкание контактов кнопки при ее нажатии происходит не мгновенно: при ближайшем рассмотрении сигнала на линии, к которой подключена кнопка, можно наблюдать переходной процесс, подобный изображенному на рисунке 2.9.

Рисунок 2.9 – Осциллограмма переходного процесса при нажатии кнопки (дребезг контактов)

Данный переходной процесс носит название **дребезг контактов;** он представляет собой многократные неконтролируемые замыкания и размыкания контактов вследствие упругости материалов и деталей контактной системы: некоторое время контакты отскакивают друг от друга при соударениях, размыкая и замыкая электрическую цепь.

В зависимости от размеров, массы, материала и конструкции контактной системы время переходного процесса (период дребезга) может составлять от 1 до 100 мс.

Дребезг контактов применительно к микроконтроллерной технике приводит к **ложным обработкам**: из-за высокой рабочей частоты система может интерпретировать колебательность переходного процесса, как несколько нажатий кнопки.

Существует аппаратные и программные методы подавления дребезга контактов.

К аппаратным методам можно отнести, например, обработку контактных пар, жидкой ртутью. В таком случае при размыкании механических контактов между ними образуется перемычка из ртути, сохраняющая электрическую связь. Данный метод применим лишь в слаботочных электромеханических ключах.

Широкое распространение получило **сглаживание переходного процесса с использованием конденсатора.** В частности, на отладочных платах для микроконтроллеров серии 1986ВЕ9х конденсаторы установлены параллельно каждой кнопке (рисунок 2.7). Пока кнопка не нажата, конденсатор заряжается от положительного потенциала линии. При нажатии кнопки конденсатор начинает разряжаться; при этом изменение напряжения на линии будет соответствовать изменению заряда на конденсаторе, т.е. будет иметь **монотонный характер**. Номинал конденсатора следует выбирать так, чтобы время его разряда было больше периода дребезга.

Что касается программных методов подавления дребезга, то их также существует несколько. Оптимальным методом с точки зрения простоты и надежности, по мнению автора, является **двукратная проверка с временной задержкой**. Суть метода сводится к организации двух проверок замыкания контактов – первичной и контрольной – с интервалом заведомо большим, чем период дребезга. Если обе проверки оказались положительными, то контакты считаются устойчиво замкнутыми; если вторая проверка оказалась отрицательной, то состояние контактов интерпретируется как разомкнутое или неустойчивое:

```
while (1)
{
  // Первичная проверка нажатия кнопки SEL (PC2)
  if (MDR_PORTC->RXTX & (1 << 2) == 0)
  {
    // Задержка для подавления дребезга контактов (~10 мс)
    DELAY(10);

    // Контрольная проверка нажатия кнопки SEL (PC2)
    if (MDR_PORTC->RXTX & (1 << 2) == 0)
    {
      // Выполнение соответствующих операций
      ...
    }
  }

  // Задержка перед началом следующей проверки (~100 мс)
  DELAY(100);
}
```

Следует еще раз отметить, что задержка в виде пустого цикла счета, организованного макросом DELAY (раздел 1.5.3), **не является верным программным подходом**, и использована здесь лишь для упрощения. Рекомендуемая альтернатива данному приему будет рассмотрена при знакомстве с сервисами операционных систем реального времени.

2.7. Блокировка интерфейсов JTAG

Для программирования микроконтроллера используется интерфейс JTAG, сигнальные линии *TDI*, *TCK, TMS* и *TRST* которого должны представлять собой цифровой ввод. В микроконтроллерах серии 1986BE9x нет специальных линий для этого интерфейса – он реализован на линиях ввода/вывода общего назначения. Если в процессе программной конфигурации эти линии были перенастроены на цифровой вывод, то программирование и отладку по данному интерфейсу далее произвести не удастся. Иными словами, **интерфейс JTAG будет заблокирован**.

Для смягчения этого обстоятельства реализовано два идентичных интерфейса – JTAG-A и JTAG-B, – подключенных к разным линиям (таблица 2.11); если заблокирован один интерфейс, можно использовать другой. Однако **следует избегать одновременного использования линий обоих интерфейсов JTAG, т.к. это может привести к полной блокировке отладочного интерфейса.**

Таблица 2.11 – Подключение интерфейсов JTAG к микроконтроллеру

Наименование интерфейса	Наименование сигнальной линии	Используемая линия микроконтроллера
JTAG-A	TDO	PB0
	TMS	PB1
	TCK	PB2
	TDI	PB3
	TRST	PB4
JTAG-B	TDO	PD0
	TMS	PD1
	TCK	PD2
	TDI	PD3
	TRST	PD4

Если блокировка отладочных интерфейсов все-таки произошла, то следует выполнить такие действия:

1. Подключить программатор к интерфейсу JTAG-B.

2. Перевести микроконтроллер в режим загрузки с внешней памяти через JTAG-B путем установки переключателей (рисунок 1.3, позиция 4) в положение «0–1–0».

3. Выполнить стирание флеш-памяти микроконтроллера командой *Flash* → *Erase*.

После этого можно перевести переключатели обратно в положение «0–0–0» и работать с микроконтроллером через интерфейс JTAG-B как прежде.

2.8. Жидкокристаллический дисплей

Для отображения буквенно-цифровой информации на отладочных платах для микроконтроллеров серии 1986ВЕ9х установлены **жидкокристаллические дисплеи MT-12864J** (рисунок 1.3, позиция 12). Разрешение дисплеев – **128 на 64 пикселя**.

Дисплей подключается к плате через 20-контактный разъем, и управляется непосредственно линиями ввода-вывода микроконтроллера (таблица 2.12).

Таблица 2.12 – Подключение жидкокристаллического индикатора к микроконтроллеру

№	Наименование вывода дисплея	Используемая линия микроконтроллера	Функция линии
1	U_{CC}	–	Питание цифровой части
2	GND	–	Общий вывод (земля)
3	U_O	–	Питание ЖК-панели
4	DB0	PA0	Линия 0 шины данных
5	DB1	PA1	Линия 1 шины данных
6	DB2	PA2	Линия 2 шины данных
7	DB3	PA3	Линия 3 шины данных
8	DB4	PA4	Линия 4 шины данных
9	DB5	PA5	Линия 5 шины данных
10	DB6	PF2	Линия 6 шины данных
11	DB7	PF3	Линия 7 шины данных
12	E1	PB7	Выбор первого кристалла
13	E2	PB8	Выбор второго кристалла
14	RES	PB9	Сброс
15	R/W	PB10	Выбор: чтение/запись
16	A0	PC0	Выбор: команды/данные
17	E	PC1	Стробирование данных
18	U_{EE}	–	Вывод преобразователя
19	A	–	Анод подсветки
20	K	–	Катод подсветки

Управление пространством дисплея осуществляется через **два драйвера,** представляющих собой специальные микросхемы: первый драйвер отвечает за левую половину дисплея (область 64 на 64 пикселя), второй драйвер – за правую. Область оперативной памяти каждого драйвера разбита на **восемь страниц** размером **64 на 8 бит**. В свою очередь, каждая страница состоит из **64 столбцов** с адресами 0x00...0x3F (рисунок 2.10).

	Драйвер 1					Драйвер 2				
	0	1	2	...	63	0	1	2	...	63
0										
1										
2										
3		Страница 0					Страница 0			
4										
5										
6										
7										
0										
1										
2										
3		Страница 1					Страница 1			
4										
5										
6										
7										
...										
0										
1										
2										
3		Страница 7					Страница 7			
4										
5										
6										
7										

Рисунок 2.10 – Структура пространства жидкокристаллического индикатора

Итак, для отображения информации на дисплей необходимо выбрать кристалл (линии E1, E2), страницу (линии DB0...DB2) и столбец (линии DB0...DB5). Далее следует сформировать на шине данных требуемое значение. **Шина данных** представляет собой 8 параллельных линий (DB0...DB7). Цифровые значения этих линий будут определять выбранный столбец пикселей: если на линии логический ноль, то соответствующий ей пиксель будет светлым; если логическая единица – темным. Далее, чтобы сформированное

на шине значение записалось в память дисплея, требуется сгенерировать стробирующий импульс на линии E. Так производится отображение любой информации в пространство дисплея.

Для работы с дисплеем MT-12864J разработан специальный библиотечный модуль *lcd.c*. В рамках данного цикла работ предлагается использовать функции указанного модуля, не углубляясь в их реализацию, т.к. это достаточно большой объем специфичного материала.

Для использования функций модуля *lcd.c* достаточно подключить его к проекту и связать его заголовок с другими модулями директивой `#include`:

```
// Подключение заголовочного файла для работы с жидкокристаллическим дисплеем
#include “lcd.h”
```

Наибольший интерес в процессе работы будут представлять следующие функции:

```
// Инициализация жидкокристаллического дисплея
void LCD_Init(void);

// Отображение символа
void LCD_PutSymbol(uint8_t symbol,         // ASCII-код символа
                   uint8_t x,              // Координата по оси абсцисс (0...20)
                   uint8_t y);             // Координата по оси ординат (0...7)

// Отображение строки
void LCD_PutString(const char *string,     // Указатель на массив со строкой
                   uint8_t y);             // Номер строки (0...7)

// Очистка строки
void LCD_ClearString(uint8_t y);           // Номер строки (0...7)

// Прокрутка строки
void LCD_ScrollString(const char *string,  // Указатель на массив со строкой
                      uint8_t y,           // Номер строки (0...7)
                      int8_t *shift,       // Указатель на сдвиговый счетчик
                      uint8_t direction);  // Направление сдвига
                                           // (LCD_SCROLL_LEFT / LCD_SCROLL_RIGHT)
```

Следует иметь в виду, что процедуру инициализации `LCD_Init()` необходимо вызывать до использования других функций дисплея, т.к. она выполняет требуемую конфигурацию линий ввода/вывода и подготавливает к работе драйверы.

Для отображения буквенно-цифровой информации реализован специальный файл *font.h*, содержащий кодировку всех символов в соответствии с ASCII-таблицей. Размер символов равен **6 на 8 пикселей**; такая кодировка обеспечивает хорошую удобочитаемость и высокую информационную вместимость дисплея (8 строк по 21 символу). Каждый символ в файле кодируется шестью однобайтовыми числами, как показано на рисунке 2.11.

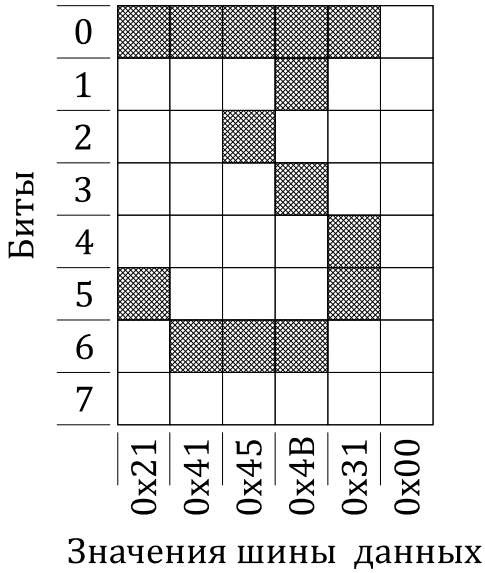


Рисунок 2.11 – Формирование символа «3» на жидкокристаллическом дисплее

Для того чтобы отобразить, например, цифру «3» (ASCII-код – 0x33) в правом нижнем углу дисплея, следует написать:

```
// Отображение цифры «3» в правом нижнем углу дисплея
LCD_PutSymbol(0x33, 20, 7);
```

Если кодировка среды программирования Keil µVision также настроена на ASCII (*Edit* → *Configuration* → *Encoding*: *Russian Windows 1251*), то для выполнения указанной операции гораздо удобнее использовать такую запись:

```
// Отображение цифры «3» в правом нижнем углу дисплея
LCD_PutSymbol('3', 20, 7);
```

Для отображения строки можно либо предварительно сформировать ее содержание, либо определить содержание при вызове функции:

```
// Первый вариант
const char *message = "Привет, мир!";
LCD_PutString(message, 4);

// Второй вариант
LCD_PutString("Привет, мир!", 4);
```

Очень часто возникает задача отображения на дисплей какого-то сформированного числа. Для этого число прежде необходимо преобразовать в символьный вид. Проще всего эта операция выполняется через функцию стандартных библиотек языка Си:

```
// Запись форматированной строки в строку с ограничением по размеру
int snprintf(char *s,              // Указатель на массив для строки
             size_t n,             // Допустимый размер строки
             const char *format,   // Форматированная строка
             ...);                 // Перечисление значений
```

Данная функция имеет широкие возможности, однако перечислять их все не имеет смысла. Для ее понимания рассмотрим пример:

```
// Пусть имеется два числа;
// требуется отобразить на дисплей операцию деления этих чисел и их частное
int32_t a = 112;
int32_t b = 14;

char message[22];  // Массив для строки

// Форматирование строки
snprintf(message, 22, "e.g.: %d / %d = %d", a, b, a/b);

// Отображение строки на дисплей
LCD_PutString(message, 4);
```

Указанные операции приведут к выводу на дисплей такой строки:

```
e.g.: 112 / 14 = 8
```

Обратите внимание на размер массива – 22; такой размер выбран не случайно. Ширина дисплея в символах равна 21; при этом любая строка в языке Си должна заканчиваться **терминирующим нулем**: это 22-ой элемент. То же самое значение определяет допустимый размер строки. Равенство этих значений является важным моментом, т.к. если сформированная строка окажется больше ширины дисплея, то лишние символы будут просто отброшены, а не займут память за пределами массива. Модуль *lcd.c* содержит специальную константу `LCD_STR_LEN`, определяемую как *размер дисплея плюс единица*; эту константу рекомендуется использовать вместо цифрового значения:

```
char message[LCD_STR_LEN];
...
```

Далее нетрудно заметить, что `%d` при форматировании заменяется на значения `a`, `b` и `a/b` в указанном порядке. Количество таких значений может быть различно; в нашей практике, однако, в большинстве случаев будет достаточного одного. Формат `%d` предполагает вывод десятичного знакового числа (англ. Decimal), однако это не единственный вариант (таблица 2.13).

Таблица 2.13 – Типы форматирования функции `snprintf`

Формат	Дескрипция	Пример
%d	Десятичное знаковое число	192
%u	Десятичное беззнаковое число	192
%o	Восьмеричное беззнаковое число	300
%X	Шестнадцатеричное беззнаковое число	C0
%f	Число с плавающей точкой	192.87
%e	Число с плавающей точкой в экспоненциальной форме	1.9287e+2
%c	Символ с кодом, соответствующим аргументу	A
%%	Знак процента	%

Форматирование чисел с плавающей точкой по умолчанию имеет точность 6 знаков; в большинстве случаев такая точность избыточна, и знаки лишь займут место на дисплее. Для отображения иного количества знаков можно использовать запись формата со спецификатором: `%.2f`; в этом случае точность составит 2 знака.

Более подробно ознакомиться с техническим описанием жидкокристаллического дисплея MT-12864J можно по официальной документации. [x]

2.9. Модуль расширения

Модуль расширения для работы с микроконтроллерной техникой (рисунок 2.12) разработан автором данной книги с целью увеличения функционала отладочных комплектов.

Рисунок 2.12 – Внешний вид модуля расширения

Модуль состоит из следующих функциональных узлов:

– 5 разноцветных светодиодов (красный, желтый, зеленый, синий, белый);

– линейный потенциометр (20 КОм);

– термистор (10 КОм);

– двухосевой джойстик (10 КОм);

– лампа накаливания (3 В, 0.4 А);

– вентилятор (3.3 / 5 В, 0.2 А);

– 4-разрядный 7-сегментный индикатор, подключенный через сдвиговый регистр;

– датчик температуры и атмосферного давления BMP280;

– часы реального времени DS1337;

– датчик освещения MAX44009EDT.

Подключение модуля к отладочной плате производится через штыревые разъемы с использованием кабельных перемычек. **Важно помнить, что подключение должно осуществляться при обесточенной системе.**

Все узлы модуля будут постепенно рассмотрены в последующих главах. Требуемые способы подключения в каждом случае будут приводиться в разделах «*Описание программных проектов*».

Описание программных проектов

Для корректного исполнения программных проектов требуется подключить модуль расширения к отладочной плате согласно таблице 2.14.

Таблица 2.14 – Подключение модуля расширения к отладочной плате

№ п/п	Модуль расширения		Отладочная плата	
	Имя контакта	Имя разъема	Имя контакта	Имя разъема
1	GND	XP3	1	X26
2	LEDR		13	
3	LEDY		14	
4	LEDG		15	
5	LEDB		16	

В проекте **Sample 2.1** реализовано последовательное циклическое переключение четырех светодиодов модуля расширения (красный, желтый, зеленый и синий) по нажатию кнопки *SEL*.

Проект **Sample 2.2** представляет собой простейший пример работы с ОСРВ. Программа состоит из трех потоков: первые два выполняют мигание парами светодиодов (красный и желтый; зеленый и синий), а третий отображает бегущую строку на жидкокристаллический дисплей.

В проекте **Sample 2.3** разработано текстовое меню, позволяющее управлять каждым из четырех светодиодов. Переключение между пунктами меню осуществляется кнопками *UP* и *DOWN*; кнопка *SEL* инвертирует состояние выбранного светодиода.

Задачи для самостоятельной работы

[*проект Sample 2.1*]

1. Добавьте в цикл работы управление белым светодиодом.

2. Реализуйте переключение между светодиодами в обоих направлениях через кнопки *LEFT* и *RIGHT*.

[*проект Sample 2.2*]

3. Ознакомьтесь с главой IIS в целях выполнения следующих задач.

4. Скорректируйте программный код таким образом, чтобы красный и желтый светодиоды мигали с частотой 2 Гц, а зеленый и синий – 5 Гц.

5. Измените содержание бегущей строки на нечто оригинальное, при этом:

– поместите ее на предпоследнюю строку дисплея;

– задайте противоположное направление перемещения;

– увеличьте скорость ее перемещения в 4 раза относительно исходной.

[*проект Sample 2.3*]

6. Организуйте отображение на дисплей количества горящих в каждый момент времени светодиодов в отдельном потоке.

Контрольные вопросы

1. Что такое регистр памяти?
2. Что такое порт ввода-вывода общего назначения?
3. Сколько портов ввода-вывода имеют микроконтроллеры серии 1986BE9x?
4. Какой максимальный ток способны вырабатывать порты микроконтроллера?
5. Какие настройки предусмотрены для портов ввода-вывода?
6. Как программно изменить значение цифрового вывода?
7. Как программно определить значение цифрового ввод?
8. Что такое дребезг контактов?
9. Что следует делать при блокировке интерфейсов JTAG?
10. Как использовать жидкокристаллический индикатор MT-12864J?