

Vinculum II — с чего начать?

Работа с портами ввода/вывода

Сергей ДОЛГУШИН
dsa@efo.ru

В 2010 году компания FTDI начала производство нового хост-контроллера USB Vinculum II. Его технические характеристики и возможности были представлены в [1]. В этой статье мы рассмотрим принципы работы с данным контроллером на примере работы с символьным дисплеем Winstar.

Кратко напомним основные характеристики хост-контроллера Vinculum II. Он представляет собой систему на кристалле, включающую в себя: 16-разрядное процессорное ядро, выполненное по Гарвардской архитектуре; два блока USB, которые могут выполнять функции периферийного устройства или хост-контроллера; набор интерфейсных модулей (UART, 2×SPI slave, SPI master, параллельный 8-разрядный FIFO, ШИМ, отладочный интерфейс); мультиплексор, предназначенный для коммутации внутренних блоков контроллера и внешних выводов; 4 канала прямого доступа к памяти (DMA); флэш-память размером 256 кбайт и 16 кбайт ОЗУ. На рис. 1 изображены основные узлы хост-контроллера VNC2. Микросхемы серии Vinculum II доступны в корпусах LQFP и QFN с количеством выводов для каждого типа 32, 48 и 64.

Контроллеры в 48-выводных корпусах повыводно совместимы с VNC1, при этом не потребуются вносить кардинальные изменения в дизайн платы.

Так же, как и предшественница, новая микросхема VNC2 содержит два USB-интерфейса, которые могут быть сконфигурированы для выполнения функций хоста или периферийного устройства. Интерфейсы совместимы со спецификациями USB 1.1 и USB 2.0, поддерживают два режима передачи: низкоскоростной (1,5 Мбит/с) и высокоскоростной (12 Мбит/с). Стоит отметить, что USB-интерфейсы контроллера Vinculum II поддерживают все типы передачи, определенные спецификацией USB:

- Interrupt (обеспечивает периодический опрос устройств на предмет наличия данных для передачи, чаще всего используется в HID-устройствах).
- Bulk (передача больших блоков данных с проверкой целостности данных).
- Isochronous (передача данных, для которых требуется постоянная скорость обмена).
- Control (используется для передачи служебных данных).

Более детальное сравнение хост-контроллеров Vinculum и Vinculum II приведено в статье [1].

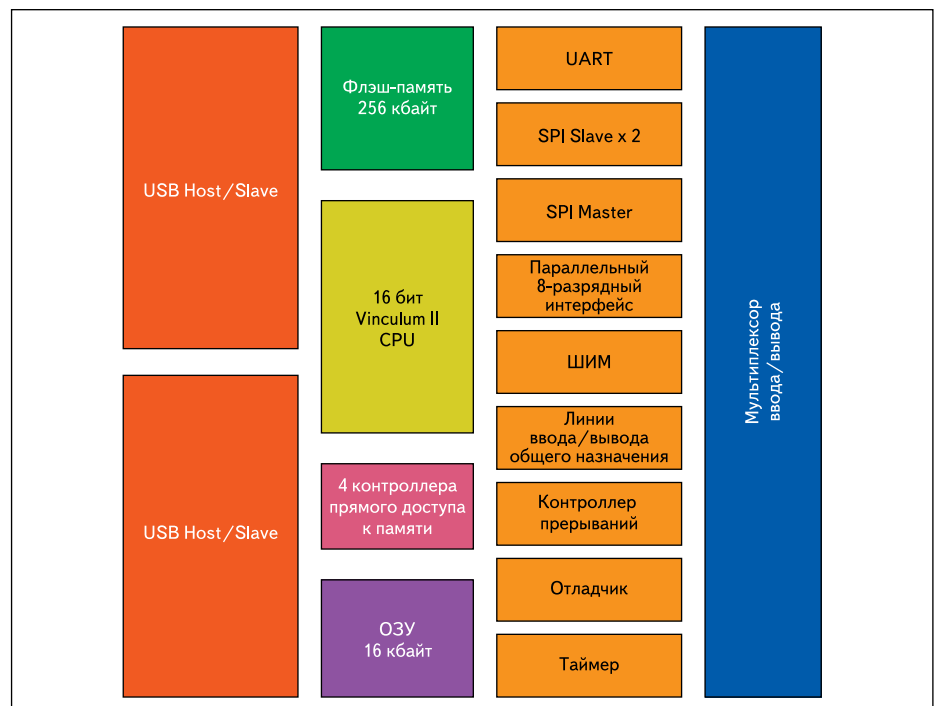


Рис. 1. Основные узлы контроллера Vinculum II

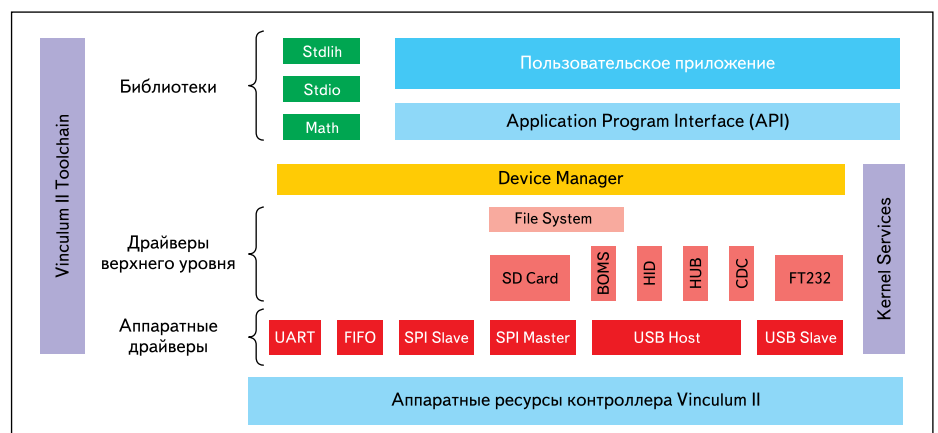


Рис. 2. Программное обеспечение Vinculum II

Основной спецификой Vinculum II является операционная система реального времени VOS (Vinculum OS): все процессы вы-

полняются под ее управлением. Помимо приоритетов поддерживаются и другие механизмы для регулирования совместного до-

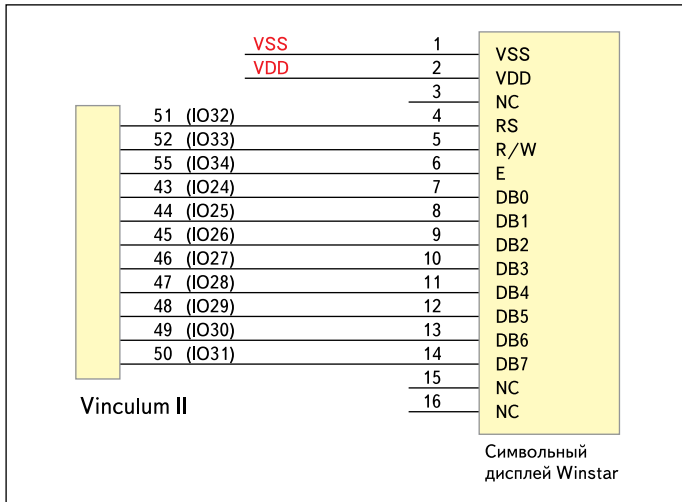


Рис. 3. 8-разрядный параллельный интерфейс

ступа к ресурсам: семафоры, мьютексы и критические секции. Эти методы позволяют избежать одновременного доступа к одному ресурсу нескольких задач, запрещают прерывать выполнение заранее определенных программистом задач и т. д.

Все аппаратные ресурсы новой микросхемы доступны через специализированные драйверы, с помощью стандартных API-функций верхнего уровня (рис. 2). В данной статье на примере работы с портами ввода/вывода мы рассмотрим работу со стандартными функциями управления драйверами. Базовые основы по работе с Vinculum II IDE и процесс создания приложения в этой среде разработки наглядно описаны в [2]. Здесь на этих моментах мы останавливаться не будем.

В качестве внешнего устройства, которым будет управлять хост-контроллер, взят стандартный ЖК-дисплей компании Winstar. Символьные ЖК-дисплеи широко применяются в различных устройствах и имеют простой интерфейс и набор команд. Для управления дисплеем будем использовать 8-разрядный параллельный интерфейс (рис. 3), для подключения которого потребуется два порта ввода/вывода хост-контроллера VNC2. Тем, кому будет интересен вариант работы с дисплеем в режиме 4-разрядного параллельного интерфейса, рекомендуем обратить внимание на пример, который предлагает производитель [3]. Он будет полезнее приводимого в данной статье — с точки зрения работы с дисплеем, но не показывает всех нюансов работы с несколькими портами ввода/вывода.

Разработка программы для Vinculum II напоминает разработку программы для ОС Windows или Linux. Физические устройства скрыты от программиста, управление ими осуществляется с помощью драйверов, доступ к которым обеспечивается API-функциями. Это позволяет, не вдаваясь в трудности освоения периферии контроллера, полностью сконцентрироваться на разработке своего приложения. Забегая вперед скажем, что в общем виде доступ к драйверам осуществляется с помощью двух функций: `vos_dev_write()` и `vos_dev_read()`. Указанные функции обращаются к драйверу требуемого периферийного блока контроллера, а драйвер производит обмен данными с указанным блоком. Такой подход упрощает освоение контроллера и модификацию приложения, например, при смене одного интерфейса на другой.

Само приложение для Vinculum II структурно представляет собой набор задач (task), каждая из которых выполняет конкретное действие. Задача, в свою очередь, включает в себя набор команд, необходимых переменных и структур, носящих название контекста (context) задачи. Поток (thread) является структурой, описывающей задачу и ее текущее состояние. Любое приложение должно содержать как минимум один поток, в котором выполняется пользовательская задача.

Любое приложение для Vinculum II может быть представлено в виде последовательности действий, приведенной на рис. 4. В соответствии с ней каждый проект начинается с процесса инициализации

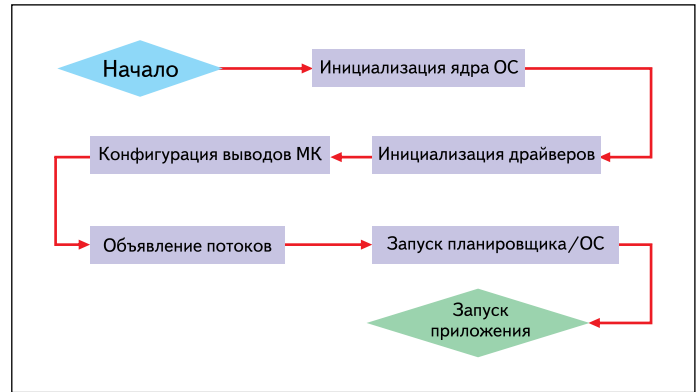


Рис. 4. Структура приложения и порядок его инициализации

ОС и драйверов, конфигурации выводов микроконтроллера, объявления потоков и запуска планировщика ОС. Все изображенные на этой схеме блоки входят в функцию `Main()`. Эта функция завершается вызовом планировщика ОС (scheduler), с помощью которого ОС управляет всеми процессами приложения. Инициализация ОС запускается вызовом функции:

```
vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);
```

Это стартовый этап любого приложения для Vinculum II. Функция инициализирует необходимые для работы ОС переменные. Она также передает три параметра, два из которых, `VOS_QUANTUM` и `VOS_TICK_INTERVAL`, определяют временные параметры периода выполнения задачи. Параметр `VOS_QUANTUM` определяет время, в течение которого выполняется процесс. Его уменьшение приводит к более частому переключению между задачами и наоборот. `VOS_TICK_INTERVAL` определяет частоту вызова планировщика. Производитель не рекомендует изменять параметр `VOS_TICK_INTERVAL`. В случае если время реакции на событие критично, производитель рекомендует использовать параметры `VOS_QUANTUM = 10` и `VOS_TICK_INTERVAL = 1` в сочетании с установкой высшего приоритета наиболее важному потоку. Переменная `NUMBER_OF_DEVICES` определяет количество периферийных узлов, которые будут использоваться в приложении. Для описываемого примера это значение равно 2, то есть каждый порт ввода/вывода считается за отдельное устройство, несмотря на то, что для работы с ними используется один драйвер:

```
vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
```

Устанавливаем частоту работы ядра контроллера; могут быть заданы значения 48, 24 и 12 МГц. Далее конфигурируется мультиплексор ввода/вывода ИОМух и осуществляется настройка параметров выводов микроконтроллера:

```
if (vos_get_package_type() == VINCULUM_II_64_PIN)
{
    vos_iomux_define_output(51, IOMUX_OUT_GPIO_PORT_A_0);
    vos_iomux_define_output(52, IOMUX_OUT_GPIO_PORT_A_1);
    vos_iomux_define_output(55, IOMUX_OUT_GPIO_PORT_A_2);

    // Шина данных, в данном примере инициализируем только на выход
    vos_iomux_define_output(43, IOMUX_OUT_GPIO_PORT_B_0);
    ...
    vos_iomux_define_output(50, IOMUX_OUT_GPIO_PORT_B_7);
}

vos_ioCELL_set_config(51, VOS_IOCELL_DRIVE_CURRENT_8MA, VOS_IOCELL_TRIGGER_NORMAL,
VOS_IOCELL_SLEW_RATE_FAST, VOS_IOCELL_PULL_UP_75K);
... (конфигурация для всех выводов одинакова)
vos_ioCELL_set_config(50, VOS_IOCELL_DRIVE_CURRENT_8MA, VOS_IOCELL_TRIGGER_NORMAL,
VOS_IOCELL_SLEW_RATE_FAST, VOS_IOCELL_PULL_DOWN_75K);
```

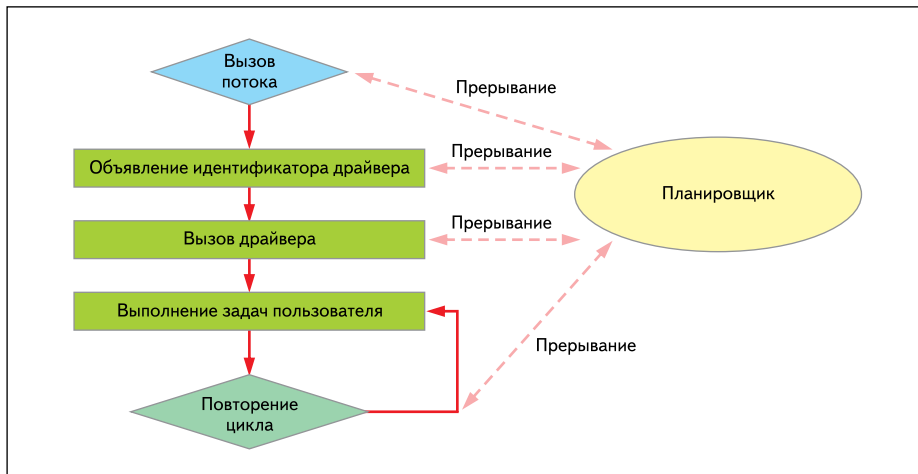


Рис. 5. Диаграмма вызова потока

Функции `vos_iomux_define_output()`, `vos_iomux_define_input()` обеспечивают подключение выводов микроконтроллера к той или иной периферии соответственно на выход или на вход. Назначение выводов микроконтроллера той или иной функции можно выполнять вручную или с помощью утилиты *I/O MUX Configuration Utility*, входящей в состав среды разработки. Функция `vos_iocell_set_config()` позволяет настроить параметры выводов микроконтроллера:

- Первый параметр определяет номер вывода и соответствующего ему блока.
- Второй — нагрузочная способность: 4 (задано по умолчанию), 8, 12 или 16 мА.
- Третий — возможность включения на вход триггера Шмитта (выключен по умолчанию).
- Четвертый — установка скорости нарастания фронта: fast (по умолчанию) и slow.
- Пятый — подключение подтягивающих к высокому или низкому уровням резисторов сопротивлением 75 Ом (по умолчанию отключены).

После завершения конфигурации выводов можно приступить к инициализации драйверов:

```
gpioCtx.port_identifier = GPIO_PORT_A;
gpio_init(VOS_DEV_GPIO_PA, &gpioCtx);
```

```
gpioCtx.port_identifier = GPIO_PORT_B;
gpio_init(VOS_DEV_GPIO_PB, &gpioCtx);
```

В данном листинге осуществляется инициализация всех используемых в приложении драйверов. В описываемом примере используются два порта ввода/вывода, для каждого из которых необходимо выполнить такую процедуру. Для каждого используемого в приложении устройства должны быть инициализированы драйверы, один или несколько. Инициализация связывает идентификатор устройства с экземпляром (*instance*) драйвера, требуемого для его работы.

Далее объявляются потоки, в которых будут выполняться пользовательские зада-

чи. В простейшем случае поток может быть один:

```
tcbFirmware = vos_create_thread(29, SIZEOF_THREAD_MEMORY,
firmware, 0);
```

При объявлении потока задаются его приоритет, размер выделяемой для его выполнения памяти, указатель на функцию и дополнительные аргументы. Приоритет потока может быть задан от 1 до 32, большее значение — больший приоритет. Размер выделяемой памяти в этой версии Vinculum II IDE определяется пользователем «на глаз», на основе следующих рекомендаций. Начальное значение объема выделяемой памяти для большинства потоков может быть выбрано в диапазоне от 256 байт до 1 кбайт. Далее, после тестирования работы потока, значение может быть уменьшено или увеличено. Признаком переполнения памяти является зависание потока в процессе выполнения. FTDI рекомендует производить корректировку размера небольшими порциями, по 64 или 128 байт. В версии 1.4.0 будет добавлена специальная утилита для контроля памяти, используемой потоками.

Запуск планировщика ОС завершает этап инициализации, и функция `Main()` передает ему управление приложением:

```
vos_start_scheduler();
main_loop;
goto main_loop;
```

После начала работы планировщика запускаются пользовательские задачи, причем процесс, имеющий наибольший приоритет, выполняется первым. Планировщик с частотой, установленной в функции `vos_init()`, отслеживает выполнение потоков и их приоритеты. При необходимости запуска задачи с большим приоритетом, чем у выполняемой, последняя закрывается с сохранением всех промежуточных результатов (рис. 5).

После блока инициализации остается описать саму задачу вывода информации

на дисплей, это делает функция `firmware()`, и функции управления дисплеем. Из функций управления рассмотрим одну — запись команд в дисплей. Для понимания принципа работы этого будет достаточно:

```
void Write_Com(VOS_HANDLE hLCDA, VOS_HANDLE hLCDB,
unsigned char byte) // запись команды
{
    unsigned char cmd;
    // определены в заголовочном файле проекта, здесь показаны
    для понимания текста программы
    #define lcd_e 0x04
    #define lcd_wr 0x02
    #define lcd_rs 0x01
    //
    cmd &= ~lcd_rs; // RS = 0;
    vos_dev_write(hLCDA, &cmd, 1, NULL);
    cmd &= ~lcd_wr; // W/R = 0;
    vos_dev_write(hLCDA, &cmd, 1, NULL);
    cmd |= lcd_e; // E = 1;
    vos_dev_write(hLCDA, &cmd, 1, NULL);
    vos_delay_msecs(1);
    vos_dev_write(hLCDB, &byte, 1, NULL);
    vos_delay_msecs(1);

    cmd &= ~lcd_e; // E = 0;
    vos_dev_write(hLCDA, &cmd, 1, NULL);
    vos_delay_msecs(1);
}
```

Функция `Write_Com()` обеспечивает формирование необходимой последовательности сигналов для записи команды во внутренние регистры дисплея в соответствии с диаграммой обмена для интерфейса 6800. Порт А хост-контроллера управляет служебными сигналами, порт В выполняет роль шины данных. Доступ к портам обеспечивается посредством драйвера GPIO, обращение к последнему осуществляется с помощью API-команд `vos_dev_read()` и `vos_dev_write()`. В соответствии с листингом, для записи в порт необходимо передать 4 параметра в функцию `vos_dev_write()`.

Первый — дескриптор (*handle*) драйвера GPIO, соответствующий порту ввода/вывода, к которому будет производиться обращение. Дескрипторы возвращаются функцией открытия драйвера `vos_dev_open()` для каждого порта отдельно, в примере открытие драйвера будет производиться в функции `firmware()`. По данному дескриптору функции чтения/записи понимают, к какому драйверу они должны обращаться.

Второй параметр — данные, которые передаются в порт.

Третий параметр функции `vos_dev_write()` — данные, которые необходимо записать (или считать для функции `vos_dev_read()`).

Четвертый — количество записываемых (считываемых) байт, он является опциональным параметром и показывает, сколько данных записано (считано). В нашем случае этот параметр не нужен и может быть пустым.

Функции `vos_dev_read()` и `vos_dev_write()` обеспечивают универсальный доступ ко всем драйверам. ОС по дескриптору понимает, к какому устройству (драйверу) будет происходить обращение. Такой подход упрощает изменение приложения, например, в случае смены одного интерфейса на другой. Изменится только часть, касающаяся настройки драйвера, но основная программа изменений не потребует.

Управление нужными выводами контроллера осуществляется в обычном порядке, с использованием логических функций И, ИЛИ, НЕ. Например, *cmd* != *lcd_e*, устанавливаем линию 2 порта А в состояние «логической 1» и далее передаем переменную *cmd* в функцию записи *vos_dev_write()* в порт.

Следующий листинг показывает основную функцию программы — *firmware()*. Именно она вызывается в объявленном в процессе инициализации потоке *tcbFirmware*, описанном выше.

```
void firmware(void)
{
    VOS_HANDLE hGpioA;
    VOS_HANDLE hGpioB;

    unsigned char i,mode,CHAR_ADD, CHAR_LAT, CHAR_RUS, font;

    CHAR_RUS=0xa0;
    CHAR_LAT=0x41;

    hGpioA = vos_dev_open(VOS_DEV_GPIO_PA);
    hGpioB = vos_dev_open(VOS_DEV_GPIO_PB);

    gpio_ioctl_cb_t gpio_iocb;
    // параметры конфигурации драйвера GPIO, которые определяют,
    // какие линии порта будут работать на выход, какие на вход.
    gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
    gpio_iocb.value = 0xFF; // все на выход

    vos_dev_ioctl(hGpioA, &gpio_iocb); // передача параметров в экземпляр драйвера GPIO, закрепленного за портом А
    vos_dev_ioctl(hGpioB, &gpio_iocb); // передача параметров в экземпляр драйвера GPIO, закрепленного за портом В

    OLED_INI(hGpioA, hGpioB);

    do
    {
        Write_Com(hGpioA, hGpioB, CLEAR); // очистка дисплея
        vos_delay_msecs(2);

        Write_Com(hGpioA, hGpioB, RHOME);
        vos_delay_msecs(2);
        Write_Com(hGpioA, hGpioB, 0x80); // вывод в первую строку дисплея
        lcd_str = " Vinculum II + ";
        // Send string to LCD
        write_str(hGpioA, hGpioB, lcd_str);
    } while (1);
}
```

```
Write_Com(hGpioA, hGpioB, RHOME);
vos_delay_msecs(2);
Write_Com(hGpioA, hGpioB, 0xC0); // вывод во вторую
строку дисплея
lcd_str = " LCD ";
// Send string to LCD
write_str(hGpioA, hGpioB, lcd_str);

vos_delay_msecs(2000);
} while (1);
```

После определения локальных переменных, как в обычном приложении, открываем драйверы GPIO для обоих задействованных портов контроллера. Открытие драйвера осуществляется функцией *vos_dev_open()* для каждого порта отдельно. Результатом открытия порта является получение собственного идентификатора для каждого используемого в нашем приложении порта. Эти идентификаторы используются для указания функциям чтения, записи и управления объекта, с которым будет происходить обмен. Аналогично осуществляется открытие других периферийных узлов хост-контроллера VNC2 и получение их идентификаторов.

Далее, для конфигурации параметров драйвера, необходимо выделить область памяти, в которой будут храниться необходимые для управления драйвером параметры. Эта область памяти в документации производителя носит название *control block*. Выделение происходит при создании структуры *gpio_ioctl_cb_t*. Параметры, которые могут быть переданы в эту структуру, описаны в файле справки Vinculum II IDE и соответствующих заголовочных файлах проекта. Для каждого периферийного узла эти параметры свои.

Приведенный пример показывает основные шаги по созданию приложения для хост-

контроллера USB Vinculum II. На первый взгляд, наличие ОС и отсутствие прямого доступа к периферии контроллера усложняют процесс разработки. Но это первые впечатления. При более близком знакомстве с Vinculum II наличие ОС и набора универсальных API-команд позволяет сократить время разработки и упростить модификацию приложения. В помощь разработчикам компания FTDI, кроме аппаратных отладочных средств [1], предлагает много полезных примеров приложений, которые могут быть использованы в качестве базы для собственных приложений.

Новые возможности хост-контроллера Vinculum II позволяют реализовать множество интересных идей, например видеорегистратор. Пример такого приложения производитель дает в исходных кодах [7]. Широкий выбор недорогих отладочных модулей позволяет быстро и при минимальных затратах протестировать возможности нового хост-контроллера. Надеемся, что наши статьи об этом процессоре будут интересны и помогут быстрее освоить его.

Литература

1. Долгушин С. Vinculum II — новый хост-контроллер USB от FTDI // Компоненты и технологии. 2010. № 9.
2. Vinculum II Tool Chain Getting Started Guide
3. AN_153 Vinculo LCD Interface Example
4. AN_151 Vinculum II User Guide
5. Technical Note TN_108 VINCULUM Chipset Feature Comparison
6. AN_157 Vinculum II Memory Management
7. AN_158 Vinculum-II Webcam Application Using OLED Display