Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Микропроцессорные средства и системы

КОНТРОЛЬНАЯ РАБОТА

по теме

Микроконтроллер MSP430F5529. Использование флеш-памяти и статической памяти (SRAM).

Выполнил
студент гр. 250541 Бобрик В.Ю.

Проверил
доцент, к.т.н. каф. ЭВМ Селезнев И.Л.

Минск 2025

СОДЕРЖАНИЕ

1 Общая характеристика микроконтроллера MSP430F5529

Микроконтроллер MSP430F5529 относится к серии процессоров для обработки смешанных сигналов со сверхнизким энергопотреблением. Он предназначен для построения встраиваемых систем, где требуется сочетание низкого энергопотребления, высокой интеграции периферийных устройств и возможности работы как с цифровыми, так и с аналоговыми сигналами.

В основе архитектуры микроконтроллера MSP430F5529 лежит 16-разрядная ортогональная RISC-архитектура, обеспечивающая простоту командного набора и высокую эффективность выполнения инструкций. Организация памяти построена по фон-Неймановскому принципу: единая адресная шина используется как для команд, так и для данных, что упрощает взаимодействие между различными подсистемами. Набор инструкций включает 27 базовых команд, расширяемых до 51, а также 37 дополнительных инструкций с поддержкой 20-битной адресации и 11 инструкций для работы с 20-битными операндами. Поддерживается семь согласованных способов адресации, что позволяет гибко работать с данными в различных режимах.

Разработчик имеет полный программный доступ ко всем ключевым регистрам, включая счетчик команд (PC), регистр состояния (SR) и указатель стека (SP). Однотактные регистровые операции и большой регистровый файл снижают количество обращений к памяти и повышают быстродействие. Архитектура поддерживает 20-битную адресную и 16-битную шину данных, а встроенный генератор констант облегчает выполнение часто используемых операций. Важной особенностью является возможность прямых пересылок «память-память» без промежуточного использования регистров. Гибкая система тактирования и наличие нескольких режимов пониженного энергопотребления позволяют адаптировать микроконтроллер под различные сценарии работы, а переход из спящего режима в активный занимает всего около шести микросекунд. Основные особенности архитектуры MSP430F5529 приведены в таблице 1.1.

Технические параметры MSP430F5529 позволяют использовать его в системах с ограниченным энергопотреблением. MSP430F5529 обеспечивает производительность до 25 миллионов инструкций в секунду. Он рассчитан на работу при напряжении питания от 1,8 до 3,6 В и отличается крайне низким энергопотреблением: в режиме хранения данных оно составляет порядка 0,1 мкА, а в режиме работы часов реального времени – около 2,5 мкА. Ток утечки выводов не превышает 50 нА, что делает устройство особенно привлекательным для автономных систем с питанием от батарей. Технические характеристики микроконтроллера приведены в таблице 1.2.

Таблица 1.1 – Основные особенности архитектуры MSP430F5529

Характеристика	Описание
Архитектура	16-разрядная ортогональная RISC
Организация памяти	Фон-Неймановская адресная шина общей памяти и шина данных
Набор инструкций	27 (51) команд + 37 расширенных инструкций (20-бит адрес) + 11 адресных инструкций
Адресация	7 согласованных способов
Доступ к регистрам	Полный программный доступ (PC, SR, SP)
Операции	Однотактные регистровые
Регистровый файл	Большой размер, снижает количество обращений к памяти
Шины	20-битная адресная шина, 16-битная шина данных
Генератор констант	6 встроенных значений
Пересылки	Память-память без промежуточного регистра
Тактирование	Гибкая система
Энергосбережение	Несколько режимов, моментальный выход в активный режим порядка 6 мкс

Таблица 1.2 – Технические характеристики MSP430F5529

Параметр	Значение
Производительность	до 25 MIPS
Напряжение питания	1,8–3,6 В
Ток утечки вывода	50 нА
Потребление в режиме хранения данных	0,1 мкА
Потребление в режиме RTC	2,5 мкА

В состав микроконтроллера входит 128 килобайт флеш-памяти и 8 килобайт статической оперативной памяти. Он имеет 80 выводов, из которых 63 могут использоваться как линии ввода-вывода общего назначения. Для организации временных процессов предусмотрены четыре асинхронных 16-разрядных таймера/счетчика с различным числом регистров захвата, сторожевой таймер и таймер реального времени. Система управления питанием PMM включает блоки защиты от падений напряжения (BOR) и контроля уровня питания (SVS).

Для обмена данными микроконтроллер оснащен универсальным последовательным коммуникационным интерфейсом USCI, поддерживающим два канала UART/LIN/IrDA/SPI и два канала I2C/SPI. Дополнительно реализованы три канала прямого доступа к памяти (DMA), аппаратный умножитель-накопитель 32×32 бита, компаратор и 12-разрядный аналого-цифровой преобразователь с шестнадцатью каналами. Среди периферийных возможностей имеется полноскоростной интерфейс USB 2.0 со встроенным стабилизатором напряжения 3,3 В, интерфейс SIF для измерения

линейных и угловых перемещений, контроллер ЖК-индикатора на 128 сегментов и внутренний генератор частоты с цифровым управлением.

2 Подсистема памяти микроконтроллера MSP430F5529

Подсистема памяти микроконтроллера MSP430F5529 построена на основе фон-Неймановской архитектуры, что означает наличие единого адресного пространства для команд и данных. В отличие от гарвардской модели, где код и данные разделены и обрабатываются раздельными шинами, фон-Неймановский подход обеспечивает большую гибкость: программист и компилятор могут свободно распределять память между программным кодом и переменными в зависимости от конкретных задач. Это решение упрощает разработку и позволяет использовать единый набор инструкций для работы как с программами, так и с данными.

Адресное пространство MSP430F5529 имеет ширину 20 бит, что позволяет адресовать до 1 мегабайта (1 МБ) памяти. Внутри этого пространства размещаются разные типы памяти и периферийные модули. В верхней части адресного диапазона находятся векторы прерываний и системные регистры, ниже располагается флеш-память объемом 128 КБ, используемая для хранения программного кода. Область SRAM объемом 8 КБ занимает отдельный сегмент адресного пространства и доступна для размещения переменных и стека. Еще одна часть адресов отведена под регистры периферийных устройств, что позволяет обращаться к ним как к обычным ячейкам памяти.

Логическое распределение адресного пространства играет ключевую роль в эффективном программировании микроконтроллера. Память периферийных устройств отображена в общее адресное пространство, что позволяет использовать для управления периферией те же команды, что и для работы с оперативной памятью. Это значительно упрощает код и снижает количество специализированных инструкций. Такое объединение накладывает ограничение на организацию защиты памяти: в отличие от систем с менеджером памяти (MMU), в MSP430F5529 отсутствуют аппаратные механизмы разграничения доступа к различным областям памяти. За корректность работы с памятью (например, за предотвращение записи в сегмент кода) полностью отвечает программное обеспечение и компилятор, что требует от разработчика повышенного внимания при работе с указателями и адресами.

Флеш-память микроконтроллера MSP430F5529 хранит основной программный код и постоянные данные, необходимые для функционирования устройства. Ее объем составляет 128 килобайт, что позволяет реализовывать достаточно сложные алгоритмы и прикладные задачи. Информация в ней сохраняется даже при полном отключении питания, что делает ее аналогом постоянного запоминающего устройства, но с возможностью многократной перезаписи.

Особенностью флеш-памяти является то, что операции записи и стирания выполняются не так же просто, как операции чтения. Для изменения содержимого необходимо предварительно стереть целый блок памяти, после чего в него можно записывать новые данные. Кроме того, процесс записи требует большего времени и энергозатрат по сравнению с чтением.

Флеш-память MSP430F5529 разделена на сегменты фиксированного размера, каждый из которых может быть стерт только целиком. Это означает, что даже для изменения одного байта необходимо очистить весь сегмент и затем заново записать его содержимое. Количество циклов записи и стирания ограничено, обычно оно составляет порядка 100 тысяч операций, поэтому при проектировании приложений рекомендуется минимизировать частоту перезаписи и использовать специальные приемы, такие как циклическая запись или распределение данных по разным сегментам. Texas Instruments в своих руководствах подчеркивает необходимость учитывать эти ограничения, чтобы продлить срок службы микроконтроллера.

Для работы с флеш-памятью в составе программных библиотек предусмотрены специальные функции, которые позволяют безопасно выполнять операции записи и стирания. Эти функции учитывают аппаратные особенности микроконтроллера, такие как необходимость временного отключения прерываний или использование специальных управляющих регистров. Доступ к флеш осуществляется через высокоуровневые API, которые скрывают от разработчика низкоуровневые детали, но при этом обеспечивают корректность и надежность работы.

Флеш-память MSP430F5529 имеет ограниченное число циклов записи и стирания, поэтому при проектировании приложений важно учитывать приемы, позволяющие продлить срок ее службы. Для компенсации ограниченного ресурса перезаписи применяются специальные программные техники. Одной из ключевых является выравнивание износа (wear leveling). Если необходимо хранить счетчики или часто обновляемые параметры, их можно записывать не в одно и то же место, а циклически перемещать по сегменту или нескольким сегментам. Это позволяет равномерно распределить износ ячеек. Еще один подход заключается в использовании буферизации: данные сначала накапливаются в SRAM, а затем записываются во флеш-память блоками, что уменьшает количество операций стирания. В некоторых случаях применяют дублирование сегментов: один сегмент хранит актуальные данные, второй – резервную копию, и при обновлении запись выполняется в новый сегмент, после чего старый помечается как неактивный и стирается при следующем цикле. Это не только продлевает жизнь памяти, но и повышает отказоустойчивость системы при сбое питания во время операции записи. Для безопасной работы с флеш-памятью используются специальные библиотечные функции. Эти функции учитывают аппаратные особенности, такие как необходимость временного отключения прерываний или использование специальных управляющих регистров. Высокоуровневые

API скрывают от разработчика низкоуровневые детали, обеспечивая корректность и надежность операций стирания и программирования.

Статическая оперативная память SRAM в MSP430F5529 имеет объем 8 килобайт и используется как основная рабочая область для выполнения программ. В ней размещаются локальные и глобальные переменные, массивы, буферы обмена и стек вызовов функций. SRAM теряет свое содержимое при отключении питания, но обеспечивает значительно более высокую скорость доступа, что делает ее незаменимой для операций, требующих частого обновления данных.

Архитектура микроконтроллера позволяет обращаться к SRAM напрямую через 20-битную адресную шину, а 16-битная шина данных обеспечивает эффективное чтение и запись слов. Высокая скорость работы этой памяти особенно важна при обработке сигналов в реальном времени, когда задержки недопустимы. Большой регистровый файл ядра снижает количество обращений к памяти, но SRAM остается местом, где хранятся промежуточные результаты вычислений и данные, поступающие от периферийных устройств.

При компиляции программы память SRAM делится на несколько областей. В секции .data размещаются инициализированные глобальные переменные, в секции .bss – неинициализированные, а стек используется для хранения локальных переменных и адресов возврата при вызове функций. Стек в MSP430F5529 растет в сторону уменьшения адресов, и его размер определяется настройками компилятора и линкера. Такое распределение позволяет эффективно использовать ограниченный объем оперативной памяти и обеспечивает предсказуемое поведение программы.

Особое значение имеет взаимодействие SRAM с контроллером прямого доступа к памяти. DMA способен записывать данные от АЦП, интерфейсов SPI или UART непосредственно в оперативную память, минуя центральное ядро. Это позволяет разгрузить процессор и сократить энергопотребление, что особенно важно для автономных систем. В прикладных задачах SRAM часто используется как буфер при обмене с внешними носителями, например, SD-картой: данные сначала загружаются в оперативную память, где могут быть проверены или преобразованы, и только затем передаются дальше.

В MSP430F5529 реализовано три канала DMA, каждый из которых может быть настроен на работу с различными источниками и приемниками данных. DMA поддерживает передачу информации между периферийными модулями и памятью без участия центрального процессора, что особенно полезно при работе с потоками данных. Например, результаты преобразования АЦП могут автоматически записываться в SRAM, а затем блоками передаваться на SD-карту. Аналогично, данные из UART или SPI могут напрямую поступать в оперативную память, освобождая ядро от рутинных операций копирования.

Оперативная память в MSP430F5529 ограничена объемом 8 килобайт, поэтому ее рациональное использование имеет большое значение. Одним из

приемов является использование переменных минимального возможного размера: для счетчиков и флагов достаточно 8-битных типов, что позволяет экономить память по сравнению с 16- или 32-битными. Компилятор MSP430 поддерживает оптимизацию размещения переменных, но разработчик может дополнительно управлять этим процессом, например, выносить редко используемые массивы во флеш-память и обращаться к ним только при необходимости. Еще один способ экономии – использование стека для временных данных вместо глобальных переменных: локальные переменные освобождают память после завершения функции, что снижает общий расход SRAM. Для буферов обмена с периферией можно применять динамическое распределение: один и тот же участок памяти используется поочередно для разных задач, если они не выполняются одновременно. В совокупности эти приемы позволяют эффективно использовать ограниченный объем оперативной памяти и реализовывать более сложные алгоритмы без увеличения аппаратных ресурсов.

Еще одним фактором, влияющим на производительность, является отсутствие кэширования инструкций и данных. В отличие от более мощных микроконтроллеров, MSP430F5529 выполняет каждое обращение к памяти напрямую, что упрощает архитектуру, но снижает эффективность при выполнении программ с частыми обращениями к одним и тем же данным.

Подсистема памяти MSP430F5529 имеет свои ограничения. Объем оперативной памяти составляет всего 8 килобайт, и этого может быть недостаточно для задач, связанных с обработкой больших массивов данных или хранением нескольких буферов одновременно. Разработчику приходится тщательно оптимизировать использование памяти и выбирать наиболее экономичные структуры данных. Ресурс флеш-памяти ограничен примерно ста тысячами циклов записи и стирания, поэтому при частом обновлении данных существует риск преждевременного выхода из строя отдельных сегментов. Дополнительные сложности создает сегментная организация: для изменения даже одного байта требуется стереть и перепрограммировать целый блок, что увеличивает время выполнения операций и энергопотребление. При работе с большими потоками информации ограниченной оказывается и пропускная способность шин: даже при использовании DMA скорость обмена ограничена архитектурой микроконтроллера.

Наиболее распространенным примером работы с внешними носителями является использование SD-карты, подключаемой через интерфейс SPI. Такая организация позволяет расширить объем доступной памяти и хранить большие массивы данных, которые невозможно разместить во встроенной флеш или SRAM. Подключение карты требует учета особенностей шины SPI: на одном канале может находиться несколько устройств, и при программировании необходимо следить за тем, чтобы в каждый момент времени активным оставалось только одно из них.

Инициализация SD-карты выполняется последовательностью команд, переводящих ее в режим SPI. После подачи питания карта находится в

состоянии, совместимом с протоколом MMC, и для перехода в нужный режим требуется отправка команд сброса и инициализации. Только после этого становится возможным чтение и запись блоков данных. Каждая операция сопровождается передачей служебных байтов и подтверждений, что обеспечивает корректность обмена. Размер блока по умолчанию составляет 512 байт. В таком размере данные передаются между картой и микроконтроллером.

Для упрощения работы с SD-картой в составе программного обеспечения MSP430F5529 используются высокоуровневые библиотеки. Одной из них является FatFs, реализующая файловую систему FAT и предоставляющая привычные функции для чтения и записи файлов. На уровне аппаратной абстракции применяются модули HAL, которые отвечают за настройку интерфейса SPI, управление сигналами выбора устройства и передачу блоков данных. В совокупности эти средства позволяют разработчику работать с SD-картой как с обычным диском, не вникая в детали протокола обмена.

Использование SD-карты тесно связано с организацией внутренней памяти микроконтроллера. SRAM в этом случае выполняет роль буфера: данные сначала загружаются в оперативную память, где могут быть проверены или преобразованы, и только затем записываются во флеш или на карту. При этом контроллер DMA обеспечивает передачу информации напрямую между периферией и SRAM, разгружая центральное ядро и снижая энергопотребление. Такая схема делает возможным эффективный обмен большими объемами данных даже в условиях ограниченных ресурсов микроконтроллера.

3 Практическая часть

Перед началом работы с микроконтроллером MSP430F5529 необходимо выполнить его базовую инициализацию. Любая программа начинается с отключения сторожевого таймера, настройки тактирования и конфигурации портов ввода-вывода. Такой минимальный проект можно рассматривать как аналог «Hello, world!»: вместо вывода текста на экран он заставляет мигать светодиод, подключенный к выводу P1.0.

```
#include <msp430.h>

// Главная функция
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Остановка сторожевого таймера

        // Настройка порта: светодиод на P1.0
    P1DIR |= BIT0;              // P1.0 как выход
    P1OUT &= ~BIT0;             // Сбросить P1.0 (светодиод выключен)

    while (1) {
```

```
        P1OUT ^= BIT0;          // Инверсия состояния светодиода
        __delay_cycles(1200000); // Задержка ~0.1 сек при 12 МГц
    }
}
```

Программа начинается с подключения заголовочного файла `msp430.h`, который содержит определения регистров и констант для работы с микроконтроллером. Это позволяет обращаться к аппаратным ресурсам по понятным именам, а не по числовым адресам.

В функции `main` первым делом останавливается сторожевой таймер (`WDT`). По умолчанию он включен и вызывает регулярные сбросы системы.

После настройки тактирования конфигурируется порт ввода-вывода. Линия `P1.0` переводится в режим выхода, а ее начальное состояние устанавливается в «0», то есть светодиод, подключенный к этому выводу, выключен.

Основной цикл программы представляет собой бесконечный цикл `while(1)`. Внутри него выполняется инверсия состояния вывода `P1.0`, что приводит к миганию светодиода. Между переключениями вставлена задержка с помощью функции `__delay_cycles`, которая простаивает заданное количество тактов процессора. При частоте 12 МГц задержка в 1,2 миллиона тактов соответствует примерно 0,1 секунды.

Чтение из флеш-памяти не требует специальных регистров: после инициализации достаточно обратиться к нужному адресу как к обычной памяти. Это позволяет без накладных расходов проверять, что было сохранено ранее.

```
unsigned int flash_read(void) {
    unsigned int *Flash_ptr = (unsigned int *) FLASH_SEGMENT;
    return *Flash_ptr;          // вернуть сохраненное слово из флеш
}
```

Запись данных в флеш-память происходит по следующему алгоритму: разблокировка контроллера, стирание сегмента, программирование слова и повторная блокировка. Перед записью необходимо стереть сегмент памяти, так как флеш-ячейки можно программировать только из состояния "1" в "0". Стирание выполняется только целиком для сегмента, поэтому даже обновление одного слова требует предварительного `erase`. В реальных проектах к этой последовательности добавляют контроль завершения операций и проверку записанных данных по чтению.

```
#define FLASH_SEGMENT   0x1800  // адрес целевого сегмента флеш

void flash_write(unsigned int value) {
    unsigned int *Flash_ptr = (unsigned int *) FLASH_SEGMENT;

    FCTL3 = FWKEY;              // снять блокировку контроллера флеш
```

```
    FCTL1 = FWKEY + ERASE;      // разрешить стирание выбранного
сегмента
    *Flash_ptr = 0;        // запись «пустого» слова инициирует
стирание

    FCTL1 = FWKEY + WRT;        // перевести флеш в режим записи
    *Flash_ptr = value;         // записать новое значение

    FCTL1 = FWKEY;              // выйти из режима записи
    FCTL3 = FWKEY + LOCK;       // вернуть аппаратную блокировку флеш
    if(FCTL3 & FAIL) {
         // Обработка ошибки записи
    }
}
```

SRAM обеспечивает прямую побайтовую адресацию с высокой скоростью доступа. В отличие от флеш-памяти, не требует специальных процедур для записи. Переменные автоматически размещаются компилятором в различных секциях (.data, .bss), но можно явно управлять размещением.

```
// Глобальные переменные в разных секциях SRAM
uint8_t global_initialized = 100; // Секция .data (инициализированные)
uint16_t global_zero[32];         // Секция .bss (неинициализированные)

void sram_operations(void) {
    // Локальные переменные в стеке
    uint8_t stack_variable = 50;
    uint32_t large_buffer[128];

    // Прямая запись в SRAM через указатели
    uint8_t* sram_ptr = (uint8_t*)0x2400; // Указатель на конкретный
адрес
    *sram_ptr = 0xAA;                     // Запись байта

    // Чтение из SRAM
    uint8_t read_value = *sram_ptr;

    // Работа с массивами в SRAM
    for(int i = 0; i < 128; i++) {
        large_buffer[i] = i * 2;          // Запись в массив
    }

    // Копирование блоков памяти
    uint8_t src_data[64] = {1,2,3,4,5};
    uint8_t dest_data[64];
    memcpy(dest_data, src_data, 64);      // Копирование SRAM->SRAM
}
```

При ограниченном объеме SRAM (8КБ) критически важно эффективное управление памятью. Используются переменные минимального размера, константы размещаются во флеш-памяти, с предпочтением локальных переменных глобальным.

```
// Константные данные во флеш-памяти
const char error_messages[][32] = {
    "Success", "Timeout", "CRC Error"  // Хранится во флеш
};

void function1(void) {
    // 8-битные переменные вместо 16-битных
    uint8_t temp_value, humidity_value;  // Экономит 50% памяти

    // Локальный буфер (освобождается после функции)
    uint8_t temp_buffer[256];

    // Использование битовых полей для флагов
    struct {
        uint8_t data_ready : 1;
        uint8_t error_flag : 1;
        uint8_t reserved : 6;
    } status_flags;  // Все флаги в 1 байте вместо нескольких

    // Временные вычисления в стеке
    int16_t intermediate_result = process_temp(temp_buffer);
}
```

DMA позволяет передавать данные между периферией и памятью без участия CPU, что экономит энергию и разгружает процессор. Настраивается через регистры управления каналами DMA.

```
uint16_t adc_results[256];
void init_dma_for_adc(void) {
    // Настройка источника (АЦП) и приемника (SRAM)
    DMA0SA = (void*)&ADC12MEM0;          // Адрес регистра АЦП
    DMA0DA = (void*)adc_results;         // Адрес буфера в SRAM
    DMA0SZ = 256;                        // Количество передач

    // Конфигурация канала DMA
    DMA0CTL = DMADT_5 |                  // Повторяющийся режим
              DMADSTINCR_3 |        // Авто-инкремент адреса приемника
              DMAEN |                    // Включение DMA
              DMAIE;                     // Разрешение прерываний

    // Запуск по готовности АЦП
    ADC12CTL1 |= ADC12SHS_0;             // Запуск от таймера
}
```

SD-карта подключается через SPI и требует специальной последовательности инициализации. Данные передаются блоками по 512 байт, используется буферизация в SRAM.

```
uint8_t sd_buffer[512];  // Буфер в SRAM для работы с SD-картой

// Чтение блока с SD-карты в SRAM
```

```
uint8_t sd_read_block(uint32_t sector, uint8_t* buffer) {
    sd_command(CMD17, sector);           // Команда чтения блока
    // Ожидание стартового байта
    while(spi_transfer(0xFF) != 0xFE);
    // Чтение 512 байт в SRAM-буфер
    for(int i = 0; i < 512; i++) {
        buffer[i] = spi_transfer(0xFF);
    }

    // Чтение CRC
    spi_transfer(0xFF);
    spi_transfer(0xFF);

    return 1; // Успех
}

// Запись блока из SRAM на SD-карту
uint8_t sd_write_block(uint32_t sector, uint8_t* buffer) {
    sd_command(CMD24, sector);           // Команда записи блока
    spi_transfer(0xFE);                  // Стартовый байт
    // Запись 512 байт из SRAM-буфера
    for(int i = 0; i < 512; i++) {
        spi_transfer(buffer[i]);
    }
    // Передача CRC
    spi_transfer(0xFF);
    spi_transfer(0xFF);
    return 1; // Успех
}
```

Для отладки работы подсистемы памяти могут быть полезны следующие функции:

```
// Чтение счетчика циклов записи флеш
uint32_t read_flash_cycles(void) {
    uint32_t* cycle_counter = (uint32_t*)0x1880;
    return *cycle_counter;
}

// Расчет использованной SRAM
uint16_t get_used_sram(void) {
    extern uint16_t __data_start, __data_end, __bss_start, __bss_end;
    uint16_t data_size = &__data_end - &__data_start;
    uint16_t bss_size = &__bss_end - &__bss_start;
    return data_size + bss_size + get_stack_usage();
}

// Мониторинг стека
uint16_t get_stack_usage(void) {
    extern uint16_t __stack_start;
    uint16_t current_sp;
    __asm(" mov r1, %0" : "=r" (current_sp));
    return __stack_start - current_sp;
}
```

```
// Свободная куча
uint16_t get_free_heap(void) {
    extern uint16_t __heap_start, __heap_end;
    return &__heap_end - &__heap_start;
}

// Проверка целостности флеш-памяти
uint8_t verify_flash_integrity(void) {
    uint16_t test_pattern = 0x55AA;
    uint16_t* test_addr = (uint16_t*)0x1820;
    flash_write(test_pattern);
    return (*test_addr == test_pattern) ? 1 : 0;
}

// Статистика использования DMA
void debug_dma_status(void) {
    printf("DMA0 transfers: %u\n", DMA0SZ);
    printf("DMA1 transfers: %u\n", DMA1SZ);
    printf("DMA2 transfers: %u\n", DMA2SZ);
}
```

Заключение

В работе рассмотрены архитектура и ключевые характеристики микроконтроллера MSP430F5529, а также особенности его подсистемы памяти: флеш-памяти и статической оперативной памяти (SRAM). Показаны практические приемы работы с памятью: чтение и запись флеш-памяти, подготовка и валидация данных в SRAM, буферизация, использование DMA и организация обмена с внешними носителями, такими как SD-карта, через SPI.

MSP430F5529 представляет собой подходящую платформу для встраиваемых приложений благодаря низкому энергопотреблению, гибкой системе тактирования и набору периферии (USB, ADC, DMA, UART/SPI/I2C). 20-битная адресация и 16-битная шина данных обеспечивают удобство работы с единым адресным пространством и упрощают программную модель устройства.

При этом существуют важные ограничения: флеш имеет сегментную организацию и ограниченное число циклов стирания (порядка 10^5), а SRAM ограничена 8 КБ объема. Для надежной работы рекомендуется применять буферизацию и записывать данные во флеш блоками, реализовывать выравнивание износа и схемы двойной записи для устойчивости при прерывающих операциях. Важны проверки целостности данных (CRC, верификация чтением) и отказоустойчивая обработка ошибок контроллера флеш, а в критических секциях – отключение прерываний и таймауты при ожидании завершения операций.

Оптимизация использования SRAM достигается через выбор минимально достаточных типов данных, разумное распределение между глобальными и локальными объектами, выравнивание буферов для DMA и избегание крупных аллокаций на стеке. Использование DMA для передачи данных между периферией и SRAM позволяет разгрузить CPU и снизить энергопотребление при обработке потоков данных.

При соблюдении перечисленных практик и учете аппаратных ограничений MSP430F5529 обеспечивает достаточно возможностей для реализации надежных и экономичных встраиваемых систем. Правильная организация работы с флеш-памятью и SRAM, комбинация буферизации, выравнивания износа и проверки целостности позволят продлить срок службы устройства и повысить устойчивость приложений к ошибкам и сбоям питания.