**Лабораторная работа 6**

**Копирование память-память в режиме DMA**

Мощевикин А.П., ПетрГУ

***Структура памяти микроконтроллера Комдив-МК***

Разрядность виртуального адресного пространства микроконтроллера Комдив-МК – 64 бита. Разрядность физического адреса процессора – 36 бит.

При работе с платой БАГЕТ-ПЛК-01 можно использовать либо оперативную память 512 кбайт, встроенную в микроконтроллер, либо внешнюю 512 Мбайт, подключенную по интерфейсу DDR3L (выбор определяется перемычкой SA4). В любом случае размер оперативной памяти намного меньше максимально возможного предела для 36-битного пространства; он даже не превышает 4 Гбайт – пространства, описываемого 32-мя битами.

В соответствии с архитектурой MIPS обращение как к оперативной памяти, так и внешним устройствам происходит по адресам в общем линейном виртуальном пространстве. Все операции проходят через блок управления памятью внутри процессорного ядра.

Все адресное пространство делится на сегменты. В зависимости от значения адреса (используемого сегмента памяти), блок управления памятью может использовать или не использовать:

- трансляцию адресов и

- кэширование.

Одну и ту же ячейку физической памяти можно отобразить на разные сегменты.

В [1, раздел 6] указано, что адреса в диапазоне 0x80000000..0x9FFFFFFF (пространство размером 512 МБайт) относятся к сегменту CKSEG0, в котором процессор кэширует память. А адреса 0xA0000000..0xBFFFFFFF (также 512 МБайт) относятся к сегменту KSEG1, в котором кэширование запрещено. В обоих сегментах трансляция адресов не используется.

Как уже было сказано, за каждым периферийным устройством закреплен свой уникальный адрес в виртуальном пространстве памяти. При обращении к устройству кэширование адресного пространства должны быть запрещено. Чтение и запись обязательно проводить каждый раз заново. Нельзя пользоваться устаревшими кэшированными значениями. Поэтому адреса всех периферийных устройств внутри микроконтроллера Комдив-МК находятся в некэшируемом сегменте KSEG1 в диапазоне 0xA0000000..0xBFFFFFFF.

Макроопределения, связанные с позиционированием ячеек памяти в разные сегменты, приведены в файле ./psp\_mc/arch/kmk/addrspace.h.

#define KSEG0 0x80000000

#define KSEG1 0xa0000000

#define CPHYSADDR(a) ((\_ACAST32\_(a)) & 0x1fffffff)

#define CKSEG1ADDR(a) (((\_ACAST32\_(a)) & 0x1fffffff) | KSEG1)

Макрос CPHYSADDR(a) формирует физический адрес из виртуального (вырезает младшие 29 разрядов).

Макрос CKSEG1ADDR(a) переносит адрес (виртуальный или физический) в некэшируемый сегмент CKSEG1: сначала вырезает младшие 29 разрядов, а затем выставляет старшие три бита соответствующим образом (0b101).

В ходе компиляции и работы программы разные переменные размещаются в разных местах памяти. Распечатать адрес переменной можно с помощью вспомогательной функции intToHex(unsigned long num, char \*hexStr), текст которой приведен в приложении А. Она преобразует целое беззнаковое число в шестнадцатеричную строку.

Например, следующий код печатает адрес третьего элемента массива arr (операция & возвращает адрес переменной; в квадратных скобках написано число 2, поскольку нумерация начинается с нуля).

char s[20];

unsigned long arr[16];

intToHex(&arr[2], s);

\_\_print\_console("0x");

\_\_print\_console(s);

\_\_print\_console("\n\r");

***Использование квалификатора volatile***

Компилятор формирует исполняемый бинарный код из текста программы и библиотек. В зависимости от примененных ключей компиляции (опций), выходной код может быть оптимизирован по объему, по быстродействию или учитывать что-то иное. При оптимизации кода разные компиляторы используют разные подходы.

Компилятор имеет право убирать неиспользуемый код, уменьшать количество действий, переставлять и передвигать куски кода по своему усмотрению. Все это приводит к тому, что бинарный файл может выполнять совсем не те действия и не в той последовательности, как того хотел программист.

Важным способом уменьшения степени свободы компилятора является явное указание ему не оптимизировать взаимодействие с некоторой переменной. Это делается с использованием ключевого слова (квалификатора) volatile при объявлении переменной. Например,

volatile int i;

volatile uint8\_t \*pReg;

Оно сообщает компилятору, что такая переменная может быть изменена в любой момент. Даже если компилятору кажется, что с ней не должны происходить операции чтения / записи. Поэтому каждое обращение к такой переменной необходимо выполнять так, как написано в исходном коде.

Чаще всего квалификатор volatile применяется:

1. при обращении к регистру процессора или периферийного устройства, отображенного на адресное пространство как ячейка памяти (memory mapped register, MMR);
2. для данных, доступ к которым осуществляется с помощью DMA;
3. для данных (например, глобальных переменных), которые изменяются в одном из обработчиков прерываний аппаратуры;
4. для данных (например, глобальных переменных), которые являются общими для нескольких потоков / задач.

Для полного осознания всех аспектов, связанных с volatile, рекомендуется изучить статьи [2-4].

***Прямой доступ к памяти***

При создании пользовательских программ часто возникает необходимость скопировать один массив данных в другой. Это можно сделать, написав код, который в цикле сначала считывает порцию данных в одном месте в оперативной памяти, а затем записывает ее в другое место. Обычно этот код исполняется центральным процессором.

Второй способ, который активно применяется в микроконтроллерной среде, связан с использованием контроллера прямого доступа к памяти ПДП (DMA – Direct Memory Access).

В этом случае, в коде программы контроллеру ПДП формулируется задача скопировать данные (откуда, куда и сколько), и запускается процесс. Важно еще раз отметить, что задачу копирования выполняет контроллер ПДП, а не процессорное ядро. В момент копирования оно может выполнять другой код. Таким образом, повышается быстродействие всей системы. Об окончании копирования контроллер ПДП обычно уведомляет процессор, используя прерывания.

Необходимо отметить, что в современных микроконтроллерах прямой доступ возможен не только на операциях память-память, но и при взаимодействии с периферийными устройствами. Например, при непосредственной пересылке данных SPI-память, UART-память, АЦП-память, пр.

В рамках этой лабораторной работе изучаются настройки только для передачи память–память.

При работе с памятью с помощью DMA необходимо отключать кэширование этих блоков памяти. Если не отключить, то получится, что два устройства – DMA-контроллер и блок управления памятью в процессорном ядре – могут одновременно изменять значения в одних и тех же ячейках ОЗУ. Возможна ситуация, при которой сначала DMA-контроллер сначала скопирует массив данных на новое место, а затем кэш-контроллер на новом месте перетрет данные старыми значениями. Т.е. произойдет «порча» данных.

Отключить кэширование переменной можно, используя обращение к ней через адресное пространство 0xA0000000..0xBFFFFFFF в сегменте KSEG1.

Например, после сборки проекта массив volatile int a[20000] размером 80000 байтов разместился по базовому адресу 0x80002848 в сегменте CKSEG0 (узнать это можно, изучив файл \*.dis).

При выполнении в программе простой операции присваивания

a[200] = 3;

в 200-тую ячейку массива a[] занесется значение 3, а часть массива a[] автоматически попадет в кэш процессора.

А при выполнении той же операции, но через некэшируемое адресное пространство KSEG1,

\*(volatile int \*)(CKSEG1ADDR(&a[200])) = 3;

кэш процессора не будет затронут. А, следовательно, данные в массиве a[] не смогут быть в последствии «испорчены» кэш-контроллером.

В приведенной выше строке сначала берется адрес ячейки с номером 200, затем он переводится в пространство адресов 0xA0000000..0xBFFFFFFF в сегменте KSEG1 (в адресе взводится биты 30-32 устанавливаются в значение 0b101), далее приводится к указателю на тип int с квалификатором volatile, и, наконец, берется значение по этому указателю.

***Организация контроллера ПДП в Комдив-МК***

Контроллер ПДП (IDMA) в Комдив-МК поддерживает 12 каналов передачи данных: память–память, память–ЦАП, АЦП–память и память–UART в обе стороны. DMA-передача в интерфейсе SPI реализована другими встроенным средствами.

Базовый адрес контроллера ПДП определен в файле ./psp\_mc/arch/kmk/uke.h.

#define IDMACTRL\_BASE 0xbb506000

Контроллер ПДП содержит 12 одинаковых наборов 64-разрядных регистров [5]. Ниже приведен их перечень для нулевого канала. Адреса других каналов получаются добавлением смещения + 0x40\*n.

Src 0 – регистр источника,

Dst 0 – регистр назначения,

Size-control 0 – регистр размера и управления,

Count-control 0 – регистр счета и управления,

Descr 0 – регистр адреса дескриптора,

ZYX 0 – регистр размера трехмерного ПДП,

Step 0 – регистр приращения значения адреса при изменении координат Y и Z.

Макросы ниже задают смещения этих регистров.

#define IDMA\_SRC\_REG\_OFFS 0x00

#define IDMA\_DST\_REG\_OFFS 0x08

#define IDMA\_SIZE\_REG\_OFFS 0x10

#define IDMA\_COUNT\_REG\_OFFS 0x18

#define IDMA\_DESCR\_REG\_OFFS 0x20

#define IDMA\_ZYX\_REG\_OFFS 0x28

#define IDMA\_STEP\_REG\_OFFS 0x30

Регистры Src 0 и Dst 0 содержат 64-битные адреса источника и приемника блока данных. Особенностью контроллера ПДП является то, что он работает с реальными физическими адресами, а не с виртуальными. Поэтому адреса, записываемые в эти регистры, необходимо модифицировать с помощью макроопределения CPHYSADDR(a).

Битовую структуру регистра размера и управления Size-control, регистра счета и управления Count-control, а также регистра адреса дескриптора Descr можно представить следующими определениями.

#define IDMA\_SIZE\_S0 0 // тип источника, 0 – для памяти

#define IDMA\_SIZE\_PS 1 // номер устройства источника (2 бита), 00 – память

#define IDMA\_SIZE\_S1 3 // тип источника, 0 – память

#define IDMA\_SIZE\_D0 4 // тип назначения, 0 – память

#define IDMA\_SIZE\_PD 5 // номер устройства назначения (2 бита), 00 – память

#define IDMA\_SIZE\_D1 7 // тип назначения, 0 – память

#define IDMA\_SIZE\_IE 8 // разрешение прерывания, берется из дескриптора, 1 – разрешено

#define IDMA\_SIZE\_IRQ 9 // состояние прерывания, запись 1 сбрасывает запрос

#define IDMA\_SIZE\_BS 10 // перестановка байтов (два бита), 00 – для LITTLE ENDIAN

#define IDMA\_SIZE\_SIZE 32 // длина передачи в байтах (32 бита), берется из дескриптора

#define IDMA\_COUNT\_ST 0 // запуск ПДП по 1, завершение пред. операции и останов по 0,

// сбрасывается автоматически по завершению

#define IDMA\_COUNT\_RE 1 // ошибка чтения

#define IDMA\_COUNT\_MODE 2 // режим работы, 1 – работа с дескрипторами

#define IDMA\_COUNT\_3D 3 // режим работы, 0 - линейный

#define IDMA\_COUNT\_WE 4 // ошибка записи

#define IDMA\_COUNT\_P 7 // пауза

#define IDMA\_COUNT\_SZ 8 // размер передачи по шине AXI (2 бита), 2 – 512 байт

#define IDMA\_COUNT\_SС8\_11 24 // состояние прерываний в каналах 8..11 (4 бита)

#define IDMA\_COUNT\_M8\_11 28 // маска событий для запуска ПДП в каналах 8..11 (4 бита)

#define IDMA\_COUNT\_COUNT 32 // кол-во дескрипторов в таблице (16 битов), можно читать

#define IDMA\_COUNT\_M1\_7 48 // маска событий для запуска ПДП в каналах 0..7 (8 битов)

#define IDMA\_COUNT\_CS1\_7 56 // состояние прерываний в каналах 0..7 (8 битов)

#define IDMA\_DESCR\_ADDR 3 // указатель на таблицу дескрипторов (32 бита)

При передаче память–память задача контроллеру ПДП может быть поставлена двумя способами.

1. Через регистры Src, Dst и Size-control.
2. В виде перечня дескрипторов, каждый из которых содержит настройки для очередной транзакции.

Таблица дескрипторов может содержать 65355 записей. Каждый дескриптор занимает 32 байта в памяти и включает 4 восьми-байтовых поля, которые контроллер ПДП автоматически копирует один-в-один в регистры Src, Dst и Size-control (за исключением 9-го бита I в регистре Size-control). Последнее поле длиной 8 байтов – зарезервировано. Таким образом, каждый дескриптор содержит одну задачу DMA-транзакции.

***Действия программиста при реализации однократной транзакции память-память с использованием ПДП (без использования прерываний)***

Для использования режима ПДП при копировании областей ОЗУ программист должен выполнить следующее в своей программе.

1. Организовать / выделить оперативную память под области «откуда» и «куда», например, создать два глобальных массива с квалификатором volatile.
2. Избегая кэширования заполнить массивы начальными значениями (при необходимости).
3. Выбрать канал DMA. Корректно остановить предыдущую DMA-транзакцию (если она была). Для этого обнулить все 64 бита в регистре Count-control. Вычитывать значение бита ST в регистре Count-control до тех пор, пока он не станет равным 0.
4. Привести значения адресов памяти «откуда» и «куда» к физическим адресам, написать их в регистры Src и Dst. Пример:

\*((volatile unsigned long \*)(IDMACTRL\_01\_BASE + IDMA\_SRC\_REG\_OFFS + 4)) = 0UL;

\*((volatile unsigned long \*)(IDMACTRL\_01\_BASE + IDMA\_SRC\_REG\_OFFS)) = \

CPHYSADDR(&a[0]);

1. Уменьшить расчетное значение длины копируемого сегмента в байтах на единицу и записать результат в поле SIZE регистра Size-control. Пример:

\*((volatile unsigned long \*)(IDMACTRL\_01\_BASE + IDMA\_SIZE\_REG\_OFFS + 4)) = \

4\*ARR\_SIZE -1;

\*((volatile unsigned long \*)(IDMACTRL\_01\_BASE + IDMA\_SIZE\_REG\_OFFS)) = \

(1 << IDMA\_SIZE\_IRQ);

1. Стартовать DMA-транзакцию, записав «1» в бит ST в регистре Count-control.
2. Ожидать окончания DMA-транзакции, вычитывая бит ST в регистре Count-control до тех пор, пока он не станет равным 0. Пример:

while (\*((volatile unsigned long \*)(IDMACTRL\_01\_BASE + IDMA\_COUNT\_REG\_OFFS)) & \

(1 << IDMA\_COUNT\_ST)) {}

Примечание. Для уменьшения вероятности «порчи» данных при копировании вследствие теневого кэширования можно дополнительно сбросить (обнулить) кэш процессора непосредственно перед стартом операции копирования.

***Задания для самостоятельной работы***

Разработчики платы БАГЕТ-ПЛК1-01 [6] рекомендуют использовать среду разработки и структуру проекта, описанные в [7] и [8, раздел 3].

Задания к лабораторной работе подразумевают, что работа выполняется в ОС Windows с предустановленной виртуальной машиной WSL2 с Debian [7] и ППП МК [8-9]. Проекты на языке Cи следует создавать в папке ./psp\_mc/apps/ .

1. Используя исходный код лабораторной работы 1 для проекта на языке Си, создать программу, в которой создано и задействовано (!) несколько переменных (глобальных и локальных) с разными модификаторами static и/или volatile. Для каждой из этих переменных распечатать ее адрес в линейном пространстве, а также адреса, модифицированные макроопределениями CPHYSADDR(a) и CKSEG1ADDR(a). Проанализировать результат. Есть ли упоминания об этих переменных в дизассемблированном образе скомпилированного проекта (в файле \*.dis)? Распечатать адрес функции main(). Сравнить ее адрес с адресом из файла \*.dis.
2. Доработать программу из п.1. Организовать копирование блока памяти с помощью контроллера DMA. Вывести хотя бы несколько значений исходного и скопированного массивов и сравнить между собой. Оценить скорость копирования в двух случаях: при использовании внутренней статической и внешней DDR памяти (определяется перемычкой SA4, ее переключать можно только в обесточенном состоянии). Для замера времени работы можно использовать \_\_micros().

Для вывода результата в консоль можно воспользоваться функцией \_\_print\_console(). Для вывода целочисленных значений можно воспользоваться функциями itoa() и intToHex().

***Список литературы***

[1] Микросхема интегральная К5500ВК018. Указания по применению. Часть 2. Ядро микропроцессора. 64-разрядный микропроцессор. // ЮКСУ.431295.019Д4.1

[2] Как использовать ключевое слово volatile на языке C // https://microsin.net/programming/avr/how-to-use-volatile-in-c.html

[3] AVR GCC: оптимизация и проблема перетасовки кода // https://microsin.net/programming/avr/avr-gcc-code-ordering-with-optimisation.html

[4] 9 способов испортить код с помощью volatile // https://microsin.net/programming/avr/nine-ways-to-break-your-systems-code-using-volatile.html

[5] Микросхема интегральная К5500ВК018. Указания по применению. Часть 8. Контроллер DMA // ЮКСУ.431295.019Д4.7

[6] ПЛК «БАГЕТ-ПЛК1-01» Руководство по эксплуатации. ЮКСУ.421457.002-01РЭ. https://www.niisi.ru/БАГЕТ-ПЛК1-01\_РЭ\_v3.3.pdf

[7] Мощевикин А.П., Голяков М.А. Установка среды разработки в Windows 10 для БАГЕТ-ПЛК1-01 // BAGET-PLK1-01\_getting\_started\_v\_2\_X.doc

[8] Пакет поддержки программирования микроконтроллера (ППП МК). Описание применения. ЮКСУ.91264-01 31 01 // ППП\_МК\_31\_Описание\_применения v12.docx

[9] Пакет поддержки программирования микроконтроллера (ППП МК). Руководство программиста. ЮКСУ.91264-01 33 01 // ППП\_МК\_33\_Руководство\_программиста\_v5.docx

Приложение А

void intToHex(unsigned long num, char \*hexStr) {

const char \*hexDigits = "0123456789ABCDEF";

int index = 0;

int i;

if (num == 0) {

hexStr[index++] = '0';

hexStr[index] = '\0';

return;

}

while (num > 0) {

int remainder = num % 16;

hexStr[index++] = hexDigits[remainder];

num /= 16;

}

hexStr[index] = '\0';

for (i = 0; i < index / 2; i++) {

char temp = hexStr[i];

hexStr[i] = hexStr[index - i - 1];

hexStr[index - i - 1] = temp;

}

}