# Проектирование процессорных ядер. Часть 4. Интеграция процессорного ядра в цифровой проект

**Илья ТАРАСОВ,**
**д. т. н.**
ilya_e_tarasov@mail.ru

**В данной части цикла статей рассматриваются практические вопросы интеграции софт-процессоров в цифровые проекты на базе ПЛИС. Многообразие задач и известных технических решений заставляет фокусироваться на наиболее значимых и эффективных подходах, которые к тому же имеют приемлемую трудоемкость. Рассмотрены вопросы оптимизации проекта на базе ПЛИС, построение системной шины и использование внешней памяти с целью получения полнофункционального процессорного устройства.**

## Введение

Нужно признать, что небольшие коллективы или индивидуальные разработчики, скорее всего, не смогут создать процессор, который мог бы стать массовым продуктом на рынке и вытеснить хотя бы часть конкурентов. Разнообразие технических решений и стоящих перед процессорами задач сводит на нет поиски архитектуры, идеальной при любых условиях использования, а необходимость оптимизации схемы и ее топологической реализации приведет к тому, что тактовая частота получаемого прототипа собственного процессора окажется ниже по сравнению с готовыми промышленными решениями. Поэтому перед началом трудоемкого и длительного процесса низкоуровневой оптимизации необходимо разработать и сбалансировать системную архитектуру, учесть требования типовых решаемых задач и оценить перспективы достижения технических характеристик. Часто подчеркивается, что решения, принимаемые на уровне системной архитектуры и RTL, оказывают гораздо большее влияние на интегральные характеристики проекта, чем низкоуровневая оптимизация отдельных схемотехнических узлов. Это в полной мере относится и к процессорам, поскольку попытка скопировать известные серийные продукты приведет, вероятнее всего, к получению менее производительного устройства, не обладающего видимыми преимуществами.

## Оптимизация характеристик процессора средствами САПР

Схематично соотношение между тактовой частотой и трудоемкостью работ при создании процессора можно проиллюстрировать рис. 1.

Обозначения на рис. 1 являются крайне приблизительными и отражают общие соотношения, а не конкретные показатели, которые заметно варьируются от одного проекта к другому. Общая идея рис. 1 заключается в том, что относительно первого варианта проекта, показанного в виде области RTL, обычно возможно провести ряд необходимых мероприятий, улучшающих характеристики проекта.

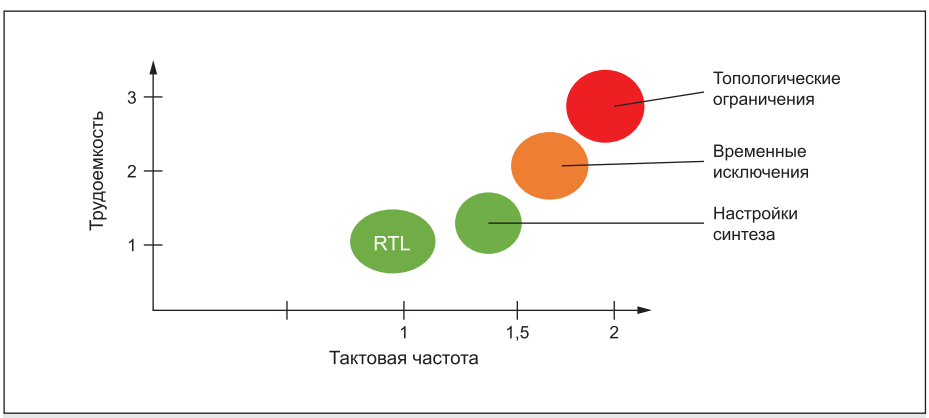


**Рис. 1.** Соотношение между тактовой частотой и трудоемкостью работ при создании процессора

Первым шагом обычно становится работа с настройками САПР, как в части синтеза, так и в части размещения и трассировки. На этих этапах настройки имеют разное влияние на результаты. В целом можно указать, что поиск оптимальных настроек САПР способен усовершенствовать характеристики практически любого цифрового проекта, но точные показатели улучшения прогнозировать крайне сложно. Можно ожидать до 50% повышения тактовой частоты, однако это не является каким-либо правилом или общей тенденцией.

Методика выбора настроек САПР Xilinx Vivado для получения оптимальных результатов изложена в [1] в разделе Timing Closure. Настройки сгруппированы в так называемые стратегии, предназначенные для оптимизации по какому-либо параметру. Целями оптимизации может быть как повышение тактовой частоты, так и уменьшение размера в логических ячейках или уменьшение суммарной потребляемой мощности. Оптимизация по одному из параметров, скорее всего, произойдет за счет ухудшения других показателей проекта, к тому же рассматривать тактовую частоту как единственный показатель качества проекта не вполне корректно. В итоге же этап подбора стратегии для САПР является ожидаемым шагом.

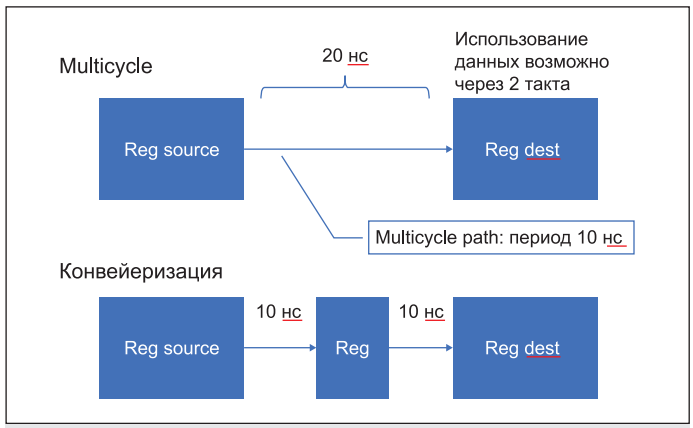


**Рис. 2.** Сравнение эффектов от проектного ограничения multicycle и конвейеризации

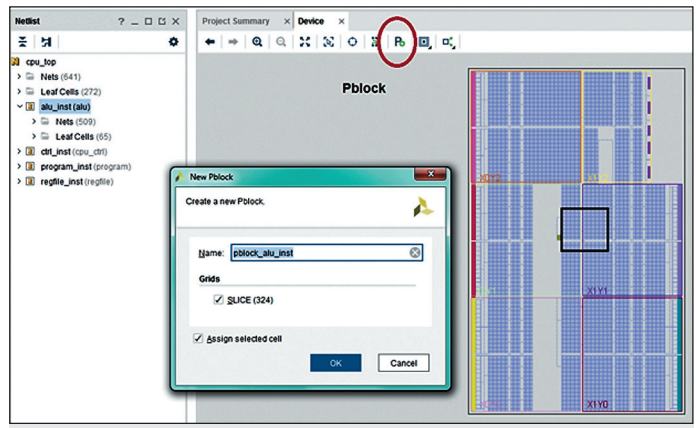


**Рис. 3.** Интерфейс САПР Vivado в режиме работы с топологическими ограничениями

Выбор стратегии САПР не предполагает чрезмерно сложных действий со стороны разработчика и на практике сводится к созданию списка планируемых к опробованию стратегий и их пакетному запуску. В САПР Vivado уже встроен интерфейс, позволяющий запускать один и тот же проект с разными настройками и сравнивать результаты.

Следующий шаг, уже требующий определенной квалификации разработчика,— введение проектных исключений. Это директивы для САПР, применяемые на стадии оценки характеристик, которые сводятся к заданию специальных ограничений в формате xdc. Важно понимать, что проектные исключения не являются сами по себе способом повышения тактовой частоты, а всего лишь ослабляют строгость контроля, устраняя отдельные элементы проверок. Разумеется, для этого необходимы объективные основания для введения исключений, поскольку для рассмотренных ранее модулей процессора оснований для введения проектных исключений в подавляющем большинстве случаев не имеется.

К важнейшим исключениям относятся:

1. Multicycle path. Этот вид исключений разрешает САПР увеличивать допустимую задержку в цепи в multicycle factor. Исключение применимо, если в проекте имеется цепь, в которой передаваемые данные гарантированно не будут использоваться на следующем такте. Такие цепи являются редкими внутри процессорного ядра, и задачу повышения тактовой частоты вместо этого следует решать путем конвейеризации.
   На рис. 2 показано сравнение эффектов от проектного ограничения multicycle и конвейеризации. Если некоторая цепь имеет задержку в 20 нс, есть возможность установить multicyclefactor, равный 2. В таком случае трассировка проекта будет считаться успешной для периода тактового сигнала 10 нс (для указанной цепи период будет умножен на multicyclefactor), однако это не означает, что сигнал каким-то образом будет передан регистру назначения за 10 нс. Использование данных будет допустимо не на следующем такте, а через два такта, что следует учесть в программном коде.
   Альтернативным решением является введение конвейеризующего регистра. При его оптимальном размещении длинная цепочка будет поделена приблизительно пополам, так что окажется возможным добиться работы с периодом 10 нс. В такой системе автоматически образуется дополнительная латентность передачи данных, так что доступность значения в регистре-источнике через два такта определяется схемотехникой.
2. Установка значения задержки (max_delay). Данный вид исключения схож по эффекту с multicycle path, поскольку перекрывает проверку по умолчанию. В этом случае речь опять же идет не о каком-либо принудительном повышении тактовой частоты за счет неких «скрытых резервов», а об освобождении САПР от необходимости обеспечить задержку, определяемую периодом тактового сигнала, для всех цепей.
3. Игнорирование задержки (falsepath). Для такой цепи любая получившаяся задержка распространения сигнала будет считаться приемлемой. На практике установку falsepath допустимо делать, например, для сигналов управления внешними светодиодами. Очевидно, что пользователь не успеет заметить, что светодиод загорелся не через 10, а через 20 или даже 100 нс. Однако не все выходные сигналы допускают установку их выходных цепей в falsepath. Например, если выходной сигнал взаимодействует с внешним устройством, отвечающим на него отправкой данных, увеличение задержки для выходного сигнала приведет к задержке ответа внешней микросхемы. Таким образом, в итоге выигрыш в скорости работы получить не удастся.

САПР Vivado содержит достаточное количество справочных сведений и предлагает удобные графические интерфейсы для работы с исключениями. Однако основная трудность их применения состоит не в сложности описания, а в необходимости глубокого понимания итогового эффекта и поиске цепей, которые действительно повлияют на характеристики проекта.

Следующим, существенно более трудоемким шагом становится работа с топологическими ограничениями. На рис. 3 показан интерфейс САПР Vivado в режиме работы с топологическими ограничениями.

Топологические ограничения означают то, что при размещении компонентов на кристалле САПР начинает руководствоваться явно задаваемыми директивами пользователя. Самая простая разновидность топологических ограничений — привязка сигнала проекта к внешнему выводу ПЛИС.

Аналогичным образом можно осуществить привязку компонентов процессора к определенным ячейкам ПЛИС. На рис. 3 видно, как в окне Device указан прямоугольник, в котором должны быть размещены компоненты модуля АЛУ. Для этого в списке Netlist (в окне слева) выбран модуль АЛУ, и после нажатия на отмеченную кнопку New Pblock можно «нарисовать» на кристалле желаемое расположение выбранного блока. Диалоговое окно служит для подтверждения выбора, где можно указать имя и подтвердить тип ресурсов ПЛИС, подлежащий ограничению. В итоге в файле проектных ограничений появятся строки, аналогичные показанным в листинге 1.

```
create_pblockpblock_alu_inst
add_cells_to_pblock [get_pblockspblock_alu_inst] [get_cells -quiet [list alu_inst]]
resize_pblock [get_pblockspblock_alu_inst] -add {SLICE_X32Y59:SLICE_X51Y79}
```

**Листинг 1.** Пример топологических ограничений в формате xdc

Выделяемая область не становится принадлежащей блоку АЛУ эксклюзивно. При необходимости в этих координатах могут быть размещены компоненты других модулей, однако все логические ячейки, реализующие функции АЛУ, не должны выходить за пределы отмеченного прямоугольника.

Введение топологических ограничений является трудоемким и неоднозначным процессом. Само по себе введение ограничений не способствует улучшению характеристик проекта, и в конечном итоге некорректные ограничения могут привести к снижению достигаемой тактовой частоты. На практике бессистемное задание ограничений, скорее всего, приведет к серии результатов трассировки, где вряд ли будет наблюдаться последовательный рост тактовой частоты. Такая работа может занять длительное время и создать иллюзию продвижения вперед.

Практическое использование топологических ограничений можно разбить на два этапа. На первом этапе следует указать очевидные правила установки отдельных модулей системы на кристалл ПЛИС. При этом глубина детализации может быть не слишком большой: в частности, если задать размещение АЛУ относительно регистрового файла, а тем более компонентов АЛУ относительно друг друга, это будет выглядеть излишеством. Рекомендации Xilinx сводятся к размещению блоков относительно внешних выводов, а также вокруг аппаратных компонентов. На первом этапе топологические ограничения могут быть нестрогими и захватывать избыточное количество ячеек. Допускается перекрытие топологических регионов, особенно для сильно связанных модулей.

На этапе финальной оптимизации проекта может потребоваться тонкая настройка критических цепей. Подобный инструмент вряд ли способен кардинально улучшить тактовую частоту (явно не следует ожидать частоты 200 вместо 150 МГц только за счет перестановки нескольких компонентов), однако общей рекомендацией к применению индивидуальной топологической настройки является наличие небольшого количества цепей, проваливающих временные проверки менее чем на 1 нс. Разумеется, это нестрогий критерий.

Возвращаясь к рис. 1, можно увидеть, что топологические ограничения характеризуются несущественным влиянием на тактовую частоту при значительном росте трудоемкости. Управление топологией проекта может сильно затянуть сроки его реализации, однако любое изменение проекта на RTL-уровне приведет к изменению схемы, и проведенная ранее тонкая топологическая оптимизация окажется неактуальной. Поэтому в исследовательских проектах достаточно ограничиться подбором настроек САПР, имея в виду, что впоследствии будет выполнена топологическая оптимизация проекта для конкретной аппаратной платформы.

Важный фактор, влияющий на итоговую тактовую частоту, — взаимодействие процессорного ядра с другими компонентами системы. Может оказаться так, что оптимально разработанное процессорное ядро после установки на ПЛИС большого логического объема будет вынуждено работать с периферийными устройствами, которые размещены по всему кристаллу, и тактовая частота неминуемо ухудшится. Для анализа характеристик системы в целом следует рассмотреть такой важный компонент процессорного проекта, как системная шина.

## Проектирование системной шины

Системная шина представляется неотъемлемой частью проекта на базе процессора и служит для соединения процессорного ядра и периферийных устройств. После работ по созданию ядра необходимо обеспечить его подключение к периферийным устройствам, без чего нельзя наглядно продемонстрировать практический эффект от процессора. Может оказаться так, что неэффективное исполнение системной шины сведет на нет усилия по оптимизации процессорного ядра.

Назначение системной шины — объединение отдельных компонентов процессорной системы и обеспечение взаимодействия между ними. Минимальный состав системной шины проиллюстрирован на рис. 4.

В целом системная шина подразделяется на шину адреса (ША), шину данных (ШД) и шину управления (ШУ). Конкретный состав сигналов и правила работы определяются многими факторами, с учетом широкого спектра системных шин и особенностей их назначения. Например, применявшаяся в ранних поколениях PC системная шина ISA использовала сигналы стробирования вместо тактового сигнала, 8-разрядную шину данных (с вариантом в 16 разрядов) и была предназначена для подключения внешних устройств с помощью плат расширения. Соответственно, скорость обмена по этой шине была невысокой.

Следует заранее разделить шины на внешние и накристальные. К внешним шинам предъявляются несколько иные требования, выполнение которых в пределах одного кристалла может оказаться нецелесообразным. К примеру, важным вопросом является реализация шины данных. На рис. 4 показано применение двух отдельных шин данных, отличающихся направлением передачи. Вместе с тем для внешних шин данные обычно мультиплексированы, то есть предусмотрена одна двунаправленная шина, переключающая направление передачи в соответствии с сигналами управления (чтение/запись).

Поскольку современные ПЛИС уже не имеют внутри матрицы ячеек двунаправленных буферов, физическая реализация шины данных все равно будет демультиплексированной (с двумя отдельными магистралями данных для записи и чтения). Поэтому не имеет смысла реализовывать на кристалле мультиплексированную шину данных и, следовательно, описывать сигнал данных как двунаправленный.

Кроме того, исходя из современной тенденции к построению синхронных схем, все операции по системной шине необходимо производить строго по фронту тактового сигнала. А значит, периферийные устройства для такой шины могут быть реализованы в соответствии с примером, приведенным в листинге 2.

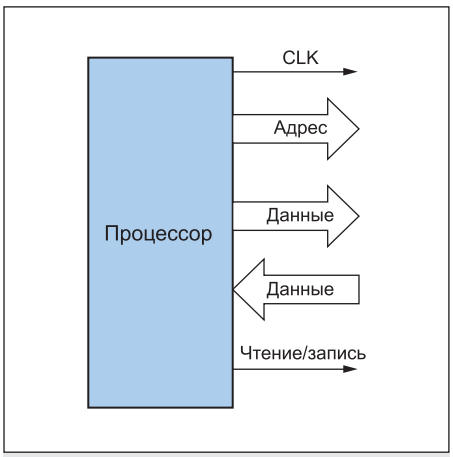


**Рис. 4.** Простая системная шина

```
process(clk)
begin
   if rising_edge(clk) then
   if addr = 1000 and write_enable = '1' then reg <= data; end if;
   end if;
end process;

data_out <= reg when addr = 1000else (others => '0');
```

**Листинг 2.** Пример периферийного устройства, реализующего запись в регистр

В примере показано использование основных сигналов системной шины. По фронту тактового сигнала проверяется совпадение адреса, выставленного на шине, с адресом регистра (в данном случае это явно заданная константа 1000) и наличие сигнала «разрешение записи». Если указанные условия выполняются, по фронту тактового сигнала во внутренний регистр периферийного модуля записывается текущее состояние шины данных.

При чтении на шину данных помещается содержимое внутреннего регистра. Контроллер системной шины должен быть реализован таким образом, чтобы правильно выбрать один из сигналов, поступающих из внешних периферийных устройств. Если при записи возможно подключение выходных сигналов процессора к нескольким периферийным модулям, что не вызывает электрического конфликта, то при чтении периферийные устройства не могут самостоятельно отключиться от шины, если адрес им не принадлежит. Языки описания аппаратуры допускают использование высокоимпедансного состояния 'Z' для описания сигналов, однако внутри современных ПЛИС нет буферов с высокоимпедансным состоянием, поэтому физическая реализация компонентов, объединяющих данные для чтения, все равно будет выполнена с помощью мультиплексоров.

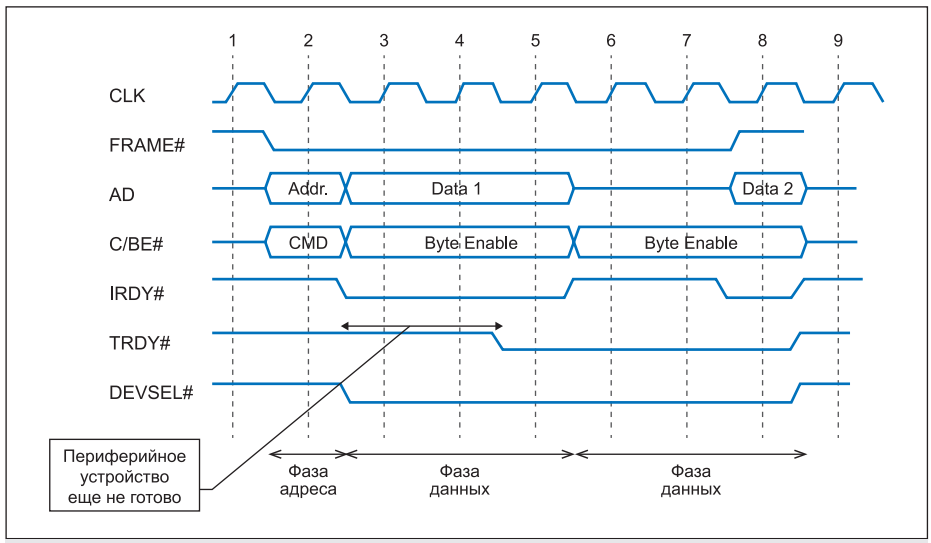


**Рис. 5.** Диаграмма обмена данными по шине PCI

Дальнейший анализ приведенного простого примера предоставляет множество путей улучшения и оптимизации системной шины. Очевидным способом улучшения приведенного описания является замена явной константы 1000 на параметризуемый адрес. Например, введение в описание компонента generic BASEADDR : integer := 1000 позволит использовать символическую константу BASEADDR, а при установке периферийного модуля в систему оперативно изменять его базовый адрес во избежание конфликтов на системной шине. Однако это не главная проблема, с которой можно столкнуться в крупном проекте.

Приведенное описание системного регистра позволяет выявить простой факт — данный регистр использует тактовый сигнал процессорного ядра, поэтому работает на частоте ядра в том же тактовом домене. С одной стороны, это эффективное решение, поскольку делает поведение периферийного регистра полностью прозрачным, а обмен с ним — максимально быстрым (в пределах одного периода тактового сигнала). С другой — применение подобной системной шины делает частоту проекта зависящей от количества периферийных устройств, подключаемых к шине процессора. Это не создает проблем для небольшой ПЛИС или проекта с исходно умеренной тактовой частотой. Однако для софт-процессоров, предлагаемых для широких кругов разработчиков, было бы желательно избавить пользователей проекта от необходимости самостоятельно поддерживать высокий уровень тактовой частоты и не терять его при подключении множества простых периферийных модулей. Кроме того, если падение частоты произойдет из-за подключения периферийных компонентов с заведомо более низкой интенсивностью обмена данными, такое решение будет выглядеть довольно странным.

Для системных шин существует несколько способов решения подобной проблемы. В каждом случае конкретные детали решения являются специфичными и имеют множество практических реализаций.

В рамках синхронного подхода можно использовать сигналы готовности к обмену. На рис. 5 показаны диаграммы обмена данными по системной шине PCI.

Шина PCI [2] предназначена для применения в качестве внешней шины процессоров x86. С точки зрения информации, рассмотренной выше, она синхронная (так как работа происходит по фронту тактового сигнала), шина данных является не только мультиплексированной, но и объединенной с шиной адреса. На диаграмме это сигнал AD, который на первой фазе («фаза адреса») передает адрес, после чего переключается на передачу данных. Четырехразрядный сигнал управления C/BE# передает тип операции (команду) в фазе адреса и индивидуальные сигналы разрешения доступа к отдельным байтам в фазах данных. Такой подход позволяет сократить количество линий в разъеме PCI.

Можно увидеть, что используемые на практике тактовые частоты для PCI составляют 33 или 66 МГц. Это невысокие значения, которые не показывали пропорционального роста по мере увеличения доступных для процессоров тактовых частот. Более того, спецификация PCI вводит специальный сигнал, с помощью которого карта расширения сообщает о возможности поддержки обмена с частотой 66 МГц. Если хотя бы одно устройство на шине не поддерживает такую частоту, системный контроллер будет работать с частотой 33 МГц. Все это является следствием того, что к шине PCI необходимо подключать множество устройств на внешних картах расширения и длина печатных проводников уже не может игнорироваться. Таким образом, внешние системные шины имеют очевидное ограничение тактовой частоты, определяемое длиной проводников.

Помимо чисто экстенсивного подхода, связанного с повышением быстродействия всех периферийных устройств, можно применить сигналы подтверждения готовности, которые на рис. 5 представлены в виде пары IRDY/TRDY. Значок # показывает, что активный уровень у этих сигналов низкий, а обозначения расшифровываются как Initiator Ready и Target Ready. На диаграмме видно, что обмен данными произойдет на такте 5, на котором оба сигнала готовности имеют активный уровень. Таким образом, при наличии этих сигналов процессор может задержать операцию на шине, удерживая сигнал IRDY в неактивном состоянии. После формирования адреса медленное периферийное устройство может задержать свой сигнал TRDY, например конвейеризуя проверку (что потребует несколько тактов). Возможность использования конвейеризации способствует повышению тактовой частоты периферийных устройств, что позволяет им работать на относительно высокой тактовой частоте процессора.

Аналогичное решение применено в распространенной шине Wishbone. Эта шина достаточно распространена в накристальных проектах процессоров и имеет большое количество разработанных для нее периферийных устройств. Интерфейс шины Wishbone показан на рис. 6.

На приведенном рисунке можно видеть элементы уже рассмотренных схемотехнических узлов. Wishbone имеет демультиплексированные шины данных (с отдельными магистралями для записи и чтения данных), операция записи сопровождается сигналом we (write enable) и дополнительно сигналом sel (select) для сопровождения каждого из байтов. Реализован также механизм подтверждения готовности — ведущее устройство сигнализирует о готовности обмена сигналом stb (strobe), а ведомое устройство должно ответить сигналом ack (acknowledge).

Другой вариант подключения медленных устройств — использование так называемых мостов (bridge). Мост представляет собой специальный контроллер, выступающий как периферийное устройство для скоростной шины, однако в свою очередь поддерживающий собственную шину с медленным темпом обмена. Таким образом, сохранение высокой тактовой частоты для всех устройств становится необязательным, достаточно обеспечить работу на этой частоте с мостом, который уже самостоятельно инициирует обмен данными с низкоскоростной периферией. Пример взаимодействия шин показан на рис. 7.

На рисунке видно, что высокоскоростные устройства, такие как накристальная блочная память, контроллер памяти DDR и Ethernet, подключены к высокоскоростной шине Processor Local Bus. В проектах на ПЛИС она

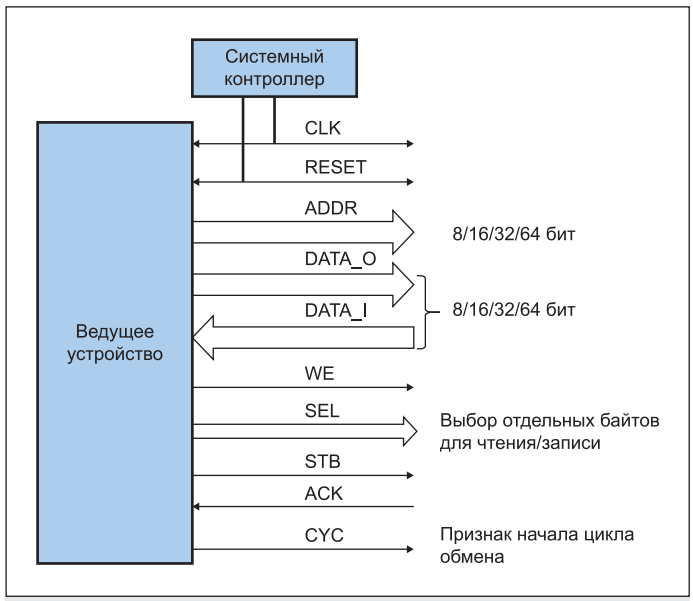


**Рис. 6.** Интерфейс системной шины Wishbone

использовала тактовые частоты 150–200 МГц. Для подключения низкоскоростных устройств, таких как UART, I²C и простые линии ввода/вывода, предназначена шина On-chip Peripheral Bus с типичной тактовой частотой 33–50 МГц. Подобный подход сокращает размер кристалла ПЛИС, для которого требуется трассировка скоростных цепей.

Для системных шин часто рассматриваются некоторые подходы, повышающие их практическую эффективность в процессорных системах. Несмотря на возможно высокие значения тактовых частот, всегда остается вопрос, насколько быстрым может быть обмен пользовательскими данными на практике. Анализ рис. 5 для шины PCI позволяет очертить круг возникающих проблем. Они связаны, прежде всего, с наличием для этой шины отдельных фаз (адреса/команды и данных), а также с тем, что и процессор, и периферийное устройство могут задержать передачу данных. Поэтому последовательная передача данных по такой шине показывает существенно меньшую усредненную скорость по сравнению с 33 млн передач данных в секунду (для частоты 33 МГц).

Одно из известных решений — пакетная передача данных по системной шине. Такой способ часто применяют при реализации прямого доступа к памяти (ПДП, также DMA — Direct Memory Access). Если инициировать этот режим, то после передачи адреса можно передавать последовательные значения данных, которые будут записываться в очередные ячейки памяти. При этом контроллер шины должен самостоятельно обеспечить последовательное увеличение адреса. Пакетный режим действительно удобен именно для доступа к памяти, поскольку имеет практическую значимость — с его помощью легко заполнять память из внешнего источника данных, не зависящего от процессора.

Сценарий, в котором данные передаются не от процессора к периферийному устройству, а между двумя устройствами, требует специального подхода к организации системной шины. Если в рассмотренном выше простом примере сигналы адреса и управления являлись выходами процессора, то при необходимости их формирования каким-то другим устройством на системной шине неминуемо образуется электрический конфликт. И если нужно управлять работой системной шины от нескольких устройств, понадобится специальное устройство управления, которое и станет определять, какое именно устройство будет подключать свои сигналы к системной шине. Для таких систем предусмотрена следующая терминология:

- ведущее устройство (master) — устройство, которое может инициировать обмен данными по системной шине, формируя сигналы адреса и управления и принимая или передавая данные;

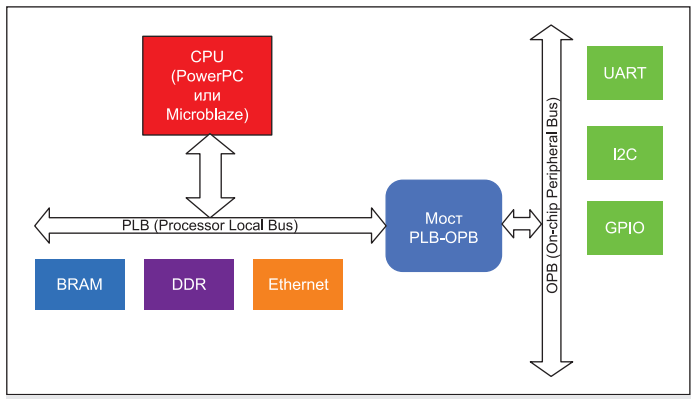


**Рис. 7.** Использование моста в технологии CoreConnect для процессорных ядер PowerPC и MicroBlaze в ПЛИС Xilinx

- ведомое устройство (slave) — устройство, которое наблюдает за сигналами адреса и управления, передавая или принимая данные по запросу;
- арбитр (arbiter) — устройство, которое определяет, какое именно ведущее устройство является в данный момент активным и может управлять системной шиной.

Пример подключения арбитра и нескольких ведущих устройств показан на рис. 8. Предполагается, что контроллер Ethernet действует в высокоскоростном режиме и способен сформировать поток данных с высокой интенсивностью. Поэтому, чтобы не занимать ресурсы процессора, этот контроллер самостоятельно подключается к системной шине и передает данные непосредственно в память, подключенную к ней.

Проектирование арбитра системной шины также имеет свои особенности. Если ведущие устройства используют пакетную передачу данных, необходимо исключить ситуацию, когда цикл обмена будет неожиданно прерван, а управление шиной передано другому ведущему устройству. Такая же ситуация может возникнуть и в других случаях, когда цикл обмена занимает более одного такта.

Самым распространенным способом управления шиной представляется использование трехступенчатой схемы. При этом реализуется такая последовательность:

1. Ведущее устройство, претендующее на работу с шиной, устанавливает сигнал запроса (обычно он обозначается REQ, от слова request).
2. Арбитр, анализируя состояние шины, выдает устройству сигнал подтверждения (GNT, от grant).
3. Ведущее устройство подтверждает захват шины и устанавливает сигнал LOCK (locked).

Пока ведущее устройство удерживает сигнал LOCK, арбитр не предоставляет доступ к шине другим ведущим устройствам. По завершении работы с шиной ведущее устройство снимает сигнал LOCK и арбитр может предоставить доступ другому ведущему устройству.

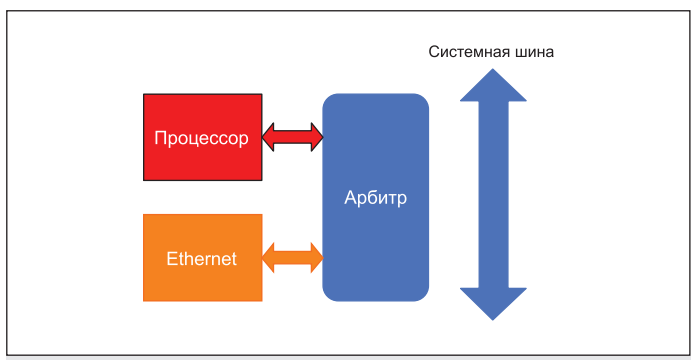


**Рис. 8.** Использование арбитра для доступа к системной шине

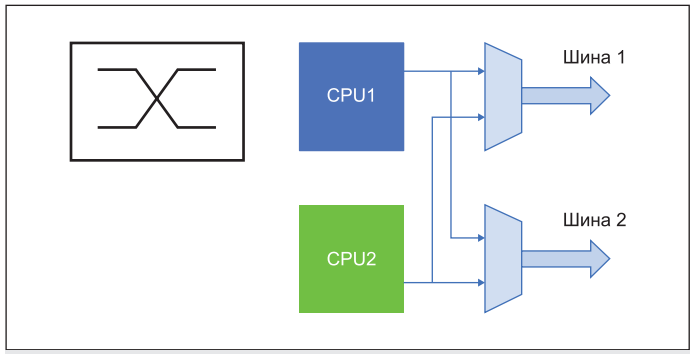


**Рис. 9.** Коммутатор (crossbar): обозначение и принцип реализации

Порядок выбора ведущих устройств для доступа к шине — это предмет отдельного рассмотрения. Если на шине находятся несколько ведущих устройств, простейшими способами выбора активного устройства являются доступ с фиксированным приоритетом и поочередный доступ.

При доступе с фиксированным приоритетом каждое из ведущих устройств подключается к соответствующему входу арбитра. Арбитр предоставляет доступ «верхнему» из всех устройств, дающих запросы на доступ. Если пронумеровать входы REQ, начиная с 0, то устройство номер 0 будет получать доступ с наивысшим приоритетом, то есть захватывать шину, если она свободна, а устройство номер 1 — только если устройство номер 0 не претендует в этот момент на захват шины. Такой порядок работы чреват ситуациями, когда ведущие устройства с низким приоритетом будут получать доступ к системной шине слишком редко, что приведет к их простоям и снижению общей производительности. С другой стороны, правильный выбор приоритетов доступа будет способствовать своевременной обработке запросов от ведущих устройств, выполняющих наиболее критичные задачи.

Поочередный доступ к шине предусматривает, что ведущее устройство, получившее доступ, при освобождении шины перемещается в конец списка, и при очередном запросе наивысший приоритет будет иметь следующее за ним устройство. Такой подход способствует более равномерному распределению времени между претендентами на системную шину.

Возможны и промежуточные варианты — например, в зависимости от критичности выполняемых задач и интенсивности обмена некоторые устройства могут перемещаться не в конец списка, а в его середину или на фиксированное место (получая таким образом 50% времени доступа). Вопросы проектирования арбитров системной шины нельзя считать до конца изученными, и в этой области остается пространство для дальнейших практических работ.

Для ПЛИС наличие большого числа внутренних трассировочных ресурсов делает удобной реализацию многослойных (layered) системных шин. В наиболее простом случае каждый процессор может иметь собственную системную шину с подключенными к ней периферийными устройствами. При этом, однако, необходимо решить вопрос архитектурного планирования и распределить периферийные устройства по процессорным ядрам. В таком случае придется также реализовать какой-то механизм обмена данными между процессорами, если все же требуется, чтобы разные процессоры могли управлять одним и тем же устройством.

Другой вариант организации нескольких шин — использование коммутатора (crossbar). Он представляет собой мультиплексор, обеспечивающий подключение к каждой из системных шин одного из процессоров. Условное обозначение такого узла на схеме и принцип реализации показаны на рис. 9.

Мультиплексоры, представленные на рис. 9, очевидно требуют управления. Для каждой из системных шин необходимо подключение только одного процессора (хотя и необязательно, чтобы в каждый момент какой-то процессор осуществлял обмен данными по шине). Эта задача также решается арбитром, который в данном варианте несколько усложняется. Увеличение количества процессорных ядер и системных шин ведет к дальнейшему усложнению коммутаторов и схем предоставления доступа, поэтому такой подход к реализации многоядерных систем имеет свои естественные ограничения.

В настоящее время многослойные системные шины используются в софт-процессорах и аппаратных процессорных подсистемах Xilinx. Подробно с характеристиками ПЛИС с процессорными ядрами на кристалле можно ознакомиться в [4].

По поводу хорошо известной для ПЛИС Xilinx аббревиатуры AXI (Advanced eXtensible Interface) необходимо сделать определенное уточнение. Компания Xilinx отмечает, что AXI в общем виде не является системной шиной или спецификацией интерфейса, а задает общие правила обмена, которые в зависимости от особенностей реализации превращаются в конкретную системную шину. В настоящее время Xilinx предлагает три варианта таких шин:

- AXI4 — полнофункциональная реализация шины, поддерживающая как одиночный, так и пакетный обмен данными;
- AXI4 Lite — упрощенная реализация, не поддерживающая пакетный обмен данными;
- AXI4 Stream — «потоковая» реализация, поддерживающая только пакетный обмен данными, при этом фаза адреса как таковая отсутствует, данный вариант системной шины оптимален для высокоскоростной передачи данных в системе «точка-точка».

Разновидности шины AXI представляют практический интерес ввиду того, что САПР Vivado предоставляет готовые конфигураторы подобных интерфейсов. После настройки IP-ядра будет сгенерирован HDL-код на выбранном пользователем языке, который может быть применен в проекте пользователя, без процессора Microblaze или ARM.

При реализации контроллера системной шины со стороны софт-процессора самым простым вариантом становится использование операндов АЛУ в качестве сигналов адреса и данных. Сигнал разрешения записи может быть сформирован путем простой проверки, как показано в примере:

```
we <= '1' when cmd = <код команды записи> else '0';
```

Это упрощенный пример, поскольку он не учитывает состояние процессора, которое может выражаться как одной переменной состояния (работа/ожидание), так и индивидуальными флагами готовности работы каждой из ступеней конвейера. В конкретной реализации сигнал разрешения записи должен устанавливаться только в том случае, если команда находится на активной стадии конвейера.

Нужно отметить, что подобный подход делает работу с системной шиной практически неотличимой от работы с памятью. Единственным различием будет используемый сигнал «разрешение записи», а временные диаграммы обмена окажутся одинаковыми. В этом случае можно и не реализовывать отдельные команды доступа к периферийным устройствам, а вместо этого «подставлять» регистры вместо соответствующих ячеек памяти данных. Данный подход носит название memory mapped I/O («устройства ввода/вывода, отображенные на память»). Отдельные команды доступа к системной шине подразумевают, что используются дополнительные возможности, такие как фазированный обмен по шине (отдельная передача адреса и данных), обеспечение проверки готовности и т. п.

В целом для софт-процессоров, изначально предназначаемых для работы в составе ПЛИС, предпочтительно начинать проектирование с простых вариантов системной шины, пользуясь возможностями ПЛИС по трассировке большого количества проводников. Поскольку при построении полностью автономной, работающей на одном кристалле процессорной системы нет необходимости выводить большое количество сигналов за пределы микросхемы, можно рассматривать варианты системных шин с несколькими слоями, в том числе полностью независимые (чтобы избежать применения сложных коммутаторов, способных оказать негативное влияние на тактовую частоту).

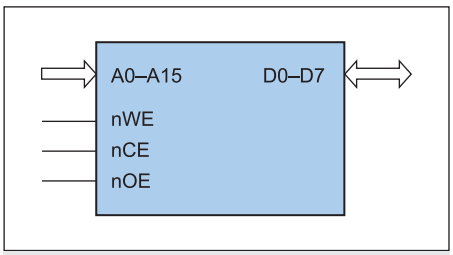


**Рис. 10.** Флэш-память с параллельным интерфейсом

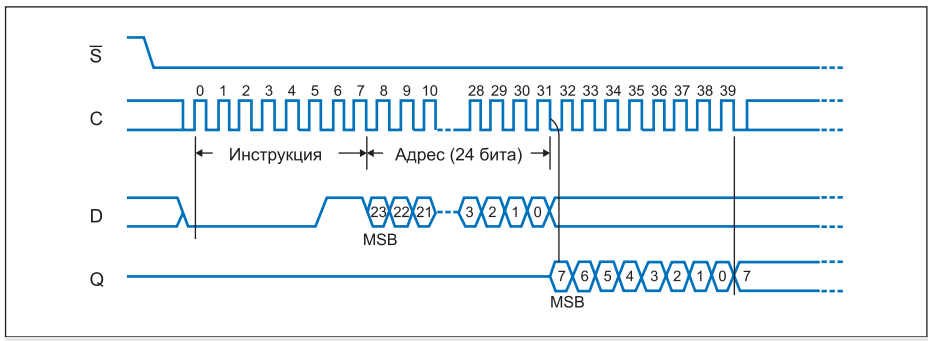


**Рис. 11.** Временные диаграммы обмена данными с флэш-памятью по SPI-интерфейсу

## Память и внешние интерфейсы

В рассмотренных ранее примерах архитектур использовалась блочная память FPGA. Такие модули представляются практически идеальным решением для построения процессорных систем на кристалле, поскольку блочная память не только обеспечивает доступ к данным за один такт (то есть не предполагает дополнительной латентности), но и находится непосредственно на кристалле в окружении логических ячеек, что упрощает трассировку. Кроме того, блочная память является двупортовой (true dual-port), что допускает не только увеличение пропускной способности такой памяти, но и построение надежно работающих схем разделения доступа к данным несколькими процессорами или процессором и устройством загрузки/отладки (что будет рассмотрено далее).

Определенным недостатком блочной памяти считается ее относительно небольшой объем. Несмотря на то, что он для разных микросхем ПЛИС находится в диапазоне от сотен килобит до десятков мегабит, вряд ли целесообразно превращать дорогостоящую ПЛИС в большую микросхему памяти, хранящую данные для процессора, занимающего очень маленькую часть программируемых ресурсов такой ПЛИС. Поэтому в целом ряде проектов необходимо добавлять внешние микросхемы памяти как для обеспечения дополнительного объема, так и для обеспечения новых функций — прежде всего энергонезависимости.

Наиболее распространенным типом энергонезависимой памяти является flash-ПЗУ («постоянное запоминающее устройство»). Английский аналог данной аббревиатуры — ROM (Read-Only Memory), то есть «память только для чтения». Название отражает тот факт, что ранние разновидности ROM были однократно программируемыми (на основе пережигаемых перемычек) или допускали стирание с помощью ультрафиолетового излучения. Это требовало специального оборудования, а потому на практике при работе в составе электронного изделия такие микросхемы не подразумевали запись данных. Современные разновидности энергонезависимой памяти обычно допускают электрическое стирание и перезапись данных в процессе работы, и понятие «только для чтения» стало в некотором роде условным.

Наиболее простым для подключения является параллельный интерфейс микросхем памяти. Пример флэш-памяти с параллельным интерфейсом проиллюстрирован на рис. 10.

Показанные сигналы имеют следующее назначение:

- A — адрес ячейки памяти;
- D — данные (двунаправленная шина);
- nWE (write enable) — разрешение записи (обычно имеет активный низкий уровень сигнала);
- nCE (chip enable) — разрешение работы микросхемы (обычно имеет активный низкий уровень сигнала);
- nOE (output enable) — разрешение подключения выхода к шине данных.

Таким образом, если микросхема памяти выбрана (nCE = ‘0’), запись неактивна (new = ‘1’) и выходы подключены к шине данных (nOE = ‘0’), то после изменения адреса на шине A микросхема через некоторое время выдаст на шину D содержимое соответствующей ячейки. Данный вариант интерфейса не использует тактовый сигнал, специфицируя вместо этого время между изменением адреса и появлением данных.

Для современных микросхем флэш-памяти типичное время задержки составляет 55, 70, 90 и 120 нс. С учетом того, что память является внешней по отношению к ПЛИС микросхемой, необходимо добавить время распространения сигналов по проводникам печатной платы, в итоге цикл чтения из такой микросхемы составит 100–150 нс, что соответствует тактовой частоте 7–10 МГц. Это очень мало по сравнению с типичными тактовыми частотами, получаемыми при работе софт-процессоров с накристальной блочной памятью (50–200 и даже более МГц). Поэтому использовать внешнюю флэш-память в качестве памяти программ софт-процессора в общем случае нецелесообразно. Ее очевидным преимуществом над блочной памятью ПЛИС является большой объем, составляющий обычно единицы и десятки мегабит.

Кроме параллельного интерфейса, многие микросхемы внешней памяти имеют интерфейс SPI. Временные диаграммы обмена данными с флэш-памятью по интерфейсу SPI представлены на рис. 11.

К достоинствам такого интерфейса следует отнести малое количество линий (4), используемое для обмена. Однако тактовая частота SPI, допускаемая разработчиками подобных микросхем, составляет 20–105 МГц. На рис. 11 видно, что при работе по SPI протокол обмена чуть более сложен, чем «отправка адреса — прием данных». Если речь идет о произвольном (не пакетном, из последовательно расположенных ячеек) чтении, то отправка адреса должна предваряться командой READ. В итоге передача в ПЛИС 8 бит данных занимает около 40 тактов по интерфейсу SPI, если каждый раз цикл чтения производить с нуля. Для тактовой частоты 100 МГц удастся таким образом передать всего 2,5 Мбайт/с, что при 1-байт команде процессора ограничит его рабочую частоту величиной 2,5 МГц.

Исходя из этого, микросхемы флэш-памяти предпочтительно использовать как внешнее устройство хранения данных большого объема, при необходимости копируя часть данных в накристальную память ПЛИС. Причем память команд и память данных процессора, вовлеченные в основной цикл работы конвейера, представляют собой блоки BRAM.

Интерфейс SPI используют также микросхемы EEPROM («память с электрическим стиранием»), которые отличаются от флэш-памяти меньшим объемом, но большим количеством циклов перезаписи. Вообще, EEPROM-память допускает индивидуальную работу с каждой ячейкой, тогда как флэш-память требует специального цикла стирания содержимого всей памяти или одного блока (обычно 256 или 512 байт). Микросхемы EEPROM удобны для хранения часто изменяющихся данных, таких как результаты регулярных измерений. Кроме EEPROM, SPI используют и некоторые перспективные типы памяти, такие как FRAM и MRAM. Часто разные производители применяют одинаковый электрический интерфейс и расположение выводов, что облегчает установку памяти нового типа в уже разработанные изделия.

Статическая память, или память с произвольным доступом (RAM, Random-Access Memory), является энергозависимой. Это

означает, что сохранение данных требует наличия питания, однако процесс записи в эту память выполняется без предварительного стирания, и нет ограничения на количество циклов перезаписи.

Статическая память (SRAM, Static RAM) может иметь асинхронный или синхронный интерфейс. Асинхронный интерфейс полностью аналогичен асинхронному интерфейсу флэш-ПЗУ с тем отличием, что типичное время доступа составляет 10 или 12 нс. Существуют и более медленные варианты (они могут быть устаревшими или обеспечивать пониженное энергопотребление), однако величина задержки доступа к данным в 10 нс может использоваться для оценки быстродействия такой памяти при работе совместно с софт-процессором. Величина 10 нс выглядит существенно лучше, чем 55 нс для флэеш-ПЗУ. Тем не менее 10 нс в данном случае не означают тактовую частоту 100 МГц, поскольку речь идет о задержке внутри самой микросхемы памяти, к которой также следует добавить задержку распространения сигнала по печатной плате и задержку в блоках ввода/вывода ПЛИС.

Статическая память с синхронным интерфейсом использует тактовый вход. Как и для любого синхронного устройства, все операции с такой памятью происходят по фронту тактового сигнала. Типичные тактовые частоты существующих на рынке микросхем памяти составляют 100–250 МГц, однако некоторые микросхемы могут подразумевать латентность доступа (то есть запрошенные выходные данные появляются после нескольких циклов тактового сигнала). Кроме того, ряд микросхем памяти требовал, чтобы при переключении между операциями чтения и записи вставлялись пустые такты. Микросхемы без подобного требования разными производителями обозначаются как ZBT SRAM (ZBT — Zero Bus Turnaround, нулевое время переключения шины) или NoBL (No Bus Latency). Подключение статической памяти с синхронным интерфейсом предусматривает предварительное изучение временных диаграмм ее работы с целью уточнить необходимое количество тактов на основные операции и имеющуюся латентность.

Более высокая скорость работы статической памяти в принципе допускает ее использование в «основном цикле» работы конвейера. При этом необходимо ожидать заметного снижения тактовой частоты, и основанием для применения внешней памяти может быть в основном ее большой объем. В связи с чем для софт-процессоров становится актуальным повышение компактности программного кода, чтобы основные программные компоненты размещались целиком на кристалле ПЛИС, а обращение к внешней памяти происходило редко и под контролем софт-процессора, явным образом перемещающего требуемые ему блоки данных.

Из сочетания быстрой накристальной и медленной внешней памяти органично следует подход, активно применяемый в современных высокопроизводительных процессорных системах, — кэш-память (cache). Поскольку затруднительно (и чрезмерно дорого) сделать всю память работающей на частоте процессорного ядра, можно ограничиться помещением в процессор небольшого объема высокоскоростной памяти. Остальную память выбирают таким образом, чтобы обеспечивать нужный большой объем, но скорость ее работы может быть ниже. Однако неминуемо возникает вопрос: какие области памяти необходимо сделать быстрыми? Например, если 32 кбит памяти будут работать на частоте процессорного ядра, а 4 Гбит — на частоте, меньшей в 3–5 раз, вероятность того, что часто используемый программный код окажется именно в блоке 32 кбайт, весьма низка. Выпуская процессор для массового применения, невозможно сказать заранее, к каким адресам памяти чаще всего будут происходить обращения. Вот почему быстрая память используется в качестве временного буфера данных, не имеющего жестко фиксированного адреса. Вместо этого в такую память загружаются данные или программный код, обращение к которому производится в настоящий момент времени. Термин cache соответствует понятию «кошелек», или «карманные деньги», проводя аналогию с деньгами, которые человек берет с собой для немедленного использования, хотя основная часть его средств находится дома или на банковском счете.

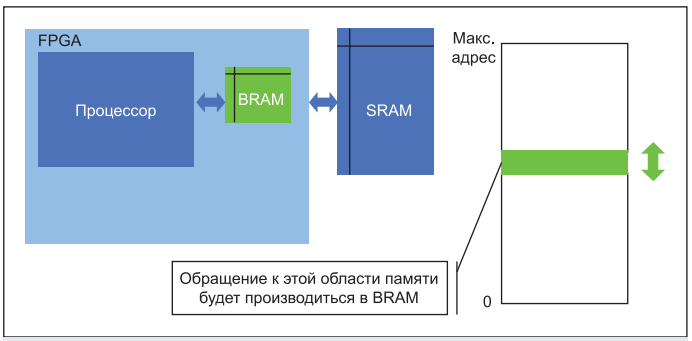


**Рис. 12.** Принцип использования кэш-памяти в процессоре

Реализация кэш-памяти и порядок ее взаимодействия с основной памятью — сложная тема, предназначенная для обсуждения в большой серии публикаций. Представленная на рис. 12 схема крайне упрощена и служит в основном для формулирования вопросов, подлежащих решению.

Например, показано, что некоторая область основной памяти «перекрыта» кэш-памятью. Поскольку процессор может работать с внутренней памятью быстрее, запись с кэш может произойти за один такт, поэтому запись в основную память не производится. Однако возникает вопрос: что делать, когда придется изменить диапазон кэшируемых адресов? Внешняя память SRAM не получала копию данных, записанных в кэш, а значит, работа программы рискует оказаться потерянной. Чтобы этого не произошло, при изменении диапазона кэшируемых адресов все сделанные изменения должны быть записаны в основную память.

Реализация механизма записи данных из кэш-памяти в основную память может быть выполнена одним из двух основных способов. «Сквозная запись» (write through) означает, что при каждой записи в кэш-память копия данных записывается и в основную память. «Отложенная запись» (write back) подразумевает, что изменения, сделанные в кэш-памяти, накапливаются, а при необходимости изменить диапазон кэшируемых адресов все изменения записываются в основную память. Первый способ обеспечивает постоянное соответствие кэш-памяти и основной памяти (с учетом того, что контроллер внешней памяти может работать независимо), однако при интенсивной записи внешняя память способна замедлить работу процессора. При реализации алгоритма записи write back характерны длительные циклы обновления основной памяти, инициируемые контроллером при необходимости изменения кэшируемого диапазона.

Можно сформулировать еще несколько вопросов, подлежащих решению. Для программиста несложно представить ситуацию, когда используемые данные расположены в двух отстоящих друг от друга диапазонах адресов (например, копирование одного массива в другой). Если размер кэш-памяти недостаточен, чтобы вместить оба массива, контроллер будет постоянно переключаться между двумя массивами, что вызовет замедление работы. Простым решением является применение нескольких областей кэш-памяти. В таком случае и массив-источник, и массив-приемник окажутся в кэш-памяти, процессор произведет обработку данных только с помощью BRAM, а затем изменения будут однократно записаны в основную память.

Наличие нескольких областей (страниц) кэш-памяти приводит разработчиков на достаточно сложный путь. Прежде всего, теперь при попытке доступа к памяти необходимо проверять все имеющиеся страницы кэша, поскольку запрашиваемый адрес может находиться в любой из них. Такая проверка должна быть произведена с небольшой задержкой, чтобы не снизить тактовую частоту (иначе польза от кэш-памяти резко снижается). Можно представить, что наличие 256 независимых страниц кэш-памяти серьезно усложнит схему проверки адреса, и реализация такой схемы в ПЛИС окажется неэффективной.

Важная алгоритмическая проблема — смена кэшируемых адресов. Например, если обращение к памяти производится таким образом,

что используется не кэш, а внешняя память, необходимо выполнить вытеснение страницы, то есть записать ее содержимое в основную память и переключиться на кэширование другого диапазона. Здесь сразу возникают вопросы, не имеющие однозначного решения. Сколько обращений к внешней памяти должно произойти, чтобы было принято решение о вытеснении? Какую из страниц следует освободить?

В плане выбора страницы для вытеснения известны несколько базовых подходов. Подход LRU (Least Recently Used) вытесняет страницу, к которой не обращались дольше всех. Подход MRU (Most Recently Used), напротив, вытесняет страницу, использованную последней. В первом случае имеется в виду, что давно не применявшаяся страница, скорее всего, уже не нужна. Во втором — что часто используемая страница, вероятно, использована полностью и в будущем обращения к ней маловероятны. Однозначный выбор между этими подходами сделать невозможно.

На практике реализация собственной подсистемы управления кэш-памятью является сложным и неоднозначным процессом, существенно зависящим от использованной элементной базы. Если речь не идет о прототипировании процессора для массового применения, работа над собственным контроллером кэш-памяти способна существенно затормозить проект, тем более что итоговые результаты в немалой степени зависят от множества факторов — архитектуры и организации кэш-памяти (write through/write back, количество страниц, алгоритм вытеснения), ее топологической оптимизации, выбранной микросхемы внешней памяти и т. д.

Кроме того, хотя построение многоядерных систем пока не было рассмотрено, следует упомянуть о такой важной проблеме, как когерентность кэш-памяти (cache coherency). На первый взгляд многоядерный процессор с многоуровневой кэш-памятью представляется высокопроизводительной системой, а большой и сложно организованный кэш создает впечатление высокой производительности. Однако при работе нескольких ядер с одной областью памяти возникает серьезная проблема.

Допустим, два процессора выполнили кэширование одной и той же переменной. Рассматриваем ситуацию, когда ядро 1 записало в эту переменную значение 5, а ядро 2 — значение 10, причем с точки зрения алгоритмов и логики программы ядро 2 приступило к работе строго после ядра 1. В этом случае программист ожидает, что переменная примет значение 10.

Однако при совместном кэшировании (и игнорировании требования когерентности) ситуация может оказаться неопределенной. Так, если ядро 2 вытеснит страницу кэша раньше, чем ядро 1, то сначала в основную память попадет правильное значение 10, а затем ядро 1 запишет в эту же память собственную версию кэшированной страницы, в которой содержится значение 5. Это и соответствует ситуации потери когерентности кэш-памяти.

Очевидным и самым простым решением является отслеживание каждым ядром состояния кэш-памяти других ядер. Если те данные, которые записываются одним ядром, кэшируются другими ядрами, они также должны получить копию записываемых данных. Это дополнительно существенно усложняет проектирование, поскольку вместо повышения производительности можно в итоге получить ее снижение из-за усложнения схем контроллера памяти и снижения тактовой частоты. Немаловажно и отслеживание логических ошибок при проектировании.

Отдельный важный вопрос — применение динамической памяти. Этот тип памяти отличается высокой тактовой частотой, используемой при обмене данными, большим объемом (значительно больше, чем у статической памяти), но большей сложностью работы. Можно посоветовать обратиться к рекомендациям Xilinx по реализации интерфейсов памяти [5] и перечислить основные проблемы работы с ней.

Динамическая память построена на основе массива конденсаторных ячеек, в отличие от триггеров, используемых в статической памяти. Конденсатор имеет гораздо меньший размер, однако быстро теряет электрический заряд. Поэтому требуется выполнять постоянное обращение к памяти, что вызывает обновление (регенерацию) заряда. Регенерация должна выполняться регулярно, и современные микросхемы памяти обычно реализуют ее самостоятельно. Однако необходимость выполнения регенерации задает нижний предел тактовой частоты, подаваемой на такую микросхему, поэтому для ее нормальной работы нужны определенные усилия как со стороны ПЛИС, так и со стороны разработчика печатной платы.

Динамическая память не предполагает прямого доступа к ячейкам по принципу «адрес на входе — данные на выходе». Вместо этого применяется доступ к встроенному контроллеру с посылкой команды, адреса и, возможно, пакета данных. Используется синхронный интерфейс, причем для повышения производительности работа выполняется не только по фронту, но и по спаду тактового сигнала. Это отражено в названии современных микросхем динамической памяти — DDR (Double Data Rate). В настоящее время происходит переход от интерфейса DDR3 к DDR4, хотя этим перечень используемых микросхем не ограничивается.

Для простых проектов не рекомендуется выполнять проектирование собственных контроллеров динамической памяти. Обеспечение простого функционирования динамической памяти требует достаточно сложной схемы, как по функциональному наполнению, так и по описанию проектных ограничений. Даже низкоскоростные варианты контроллеров предполагают реализацию целого перечня требований, в том числе и к печатной плате [6].

При необходимости реализации динамической памяти (она имеет очевидную привлекательность из-за большого объема) рекомендуется использовать бесплатно предоставляемое IP-ядро Xilinx Memory Interface Generator. Оно обеспечивает, прежде всего, задание проектных ограничений, поддерживающих работоспособность контроллера памяти в условиях жестких требований к задержкам распространения сигналов.

## Заключение

Данная часть статьи скорее описывает предостережения и компромиссы, чем предлагает эффективные решения. В современных процессорных системах предусмотрено множество технологий, которые кажутся привычными и простыми в силу широкого распространения. Однако такие приемы, как кэширование (тем более в многоядерной системе), многослойные системные шины, динамическая память DDR3/4, высокоскоростные интерфейсы, предполагают высокую квалификацию разработчика, требуют большой трудоемкости для реализации, отладки и воспроизведения показателей при серийном производстве (для печатных плат), поэтому оставляют мало пространства для исследовательской деятельности. Более привлекательной для получения практических результатов выглядит разработка специализированных софт-процессоров, действующих преимущественно с накристальной памятью и предназначенных для решения частных задач в составе комплексных вычислительных систем.

Далее в цикле статей будет рассмотрена разработка инструментальных средств компиляции, отладки и загрузки программ, а также вопросы совместной оптимизации архитектуры процессора и системы команд с учетом требований программного обеспечения.

## Литература


1. UltraFast Design Methodology Guide forthe Vivado Design Suite. www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug949-vivado-design-methodology.pdf
2. www.pcisig.com/
3. AXI ReferenceGuide. www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf
4. www.xilinx.com/products/silicon-devices/soc.html
5. www.xilinx.com/products/intellectual-property/mig.html
6. www.xilinx.com/support/documentation/white_papers/wp484-a7-s7-ddr2-3-pcb.pdf