Лабораторная работа №2. «Системные шины»

Системная шина

Системная шина (system bus) – совокупность сигнальных линий, служащих для обмена информацией между элементами системы на кристалле или на печатной плате. Сигналы системной шины в зависимости от назначения можно разделить на три группы. Линии системной шины, отвечающие за передачу данных, называется шиной данных. Линии, передающие адрес, называются шиной адреса, а прочие управляющие сигналы – шиной управления.
Как было замечено ранее, основной функцией системной шины является обмен информацией. В простейшем случае происходит обмен информацией между одним процессорным ядром и множеством контроллеров. Обобщённая структурная схема подключения процессора и контроллеров к системной шине показана на рисунке 3.1.

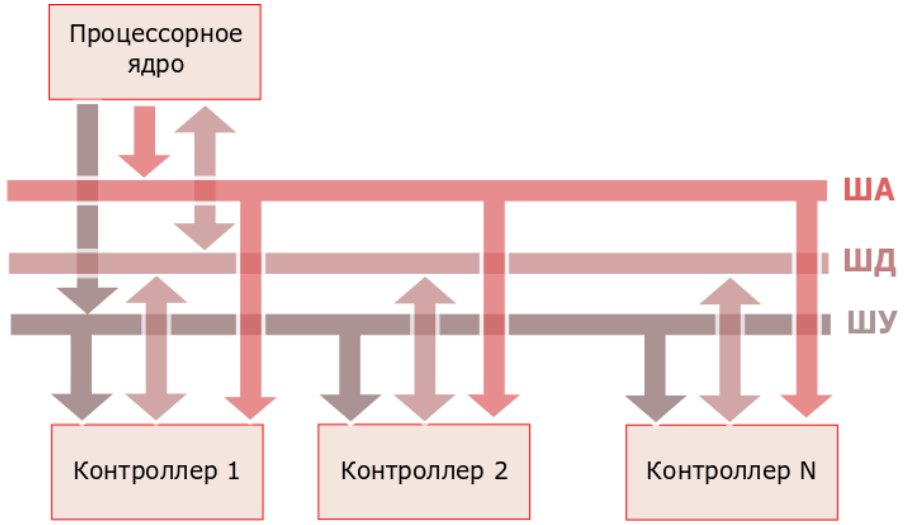

Рисунок 3.1 – Обобщённая структурная схема подключения процессора и контроллеров к системной шине.

На рисунке 3.2 приведена диаграмма обобщенной конструкции системной шины.

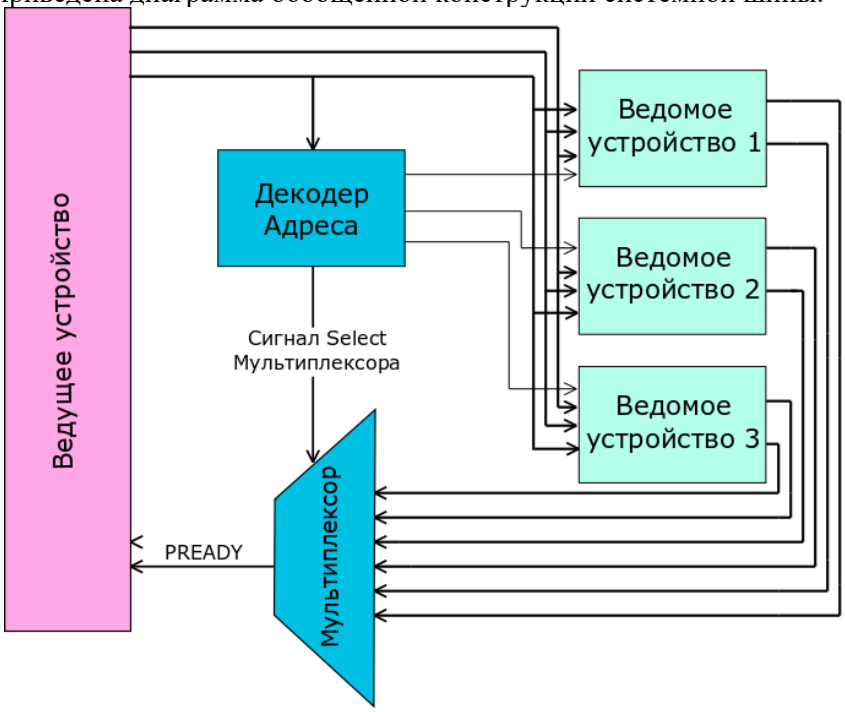

Рисунок 3.2 – Обобщённая конструкция системной шины.

Существует ряд терминов, которые обычно используются в спецификациях шин для систем на кристалле.

Ведущее устройство (Master) – устройство, которое инициирует передачу данных для чтения или записи по системной шине (например, процессор или ЦОС-процессор).
Ведомое устройство (Slave) – устройство, которое не инициирует обмен данными и отвечает только на входящие запросы на передачу.
Декодер относится к логическому блоку, который дешифрует адрес назначения передачи данных, инициированной ведущим устройством, и выбирает соответствующее ведомое устройство для приема данных.
Мультиплексор используется для мультиплексирования шины считанных данных и ответных сигналов от подчиненных устройств к ведущему. Декодер обеспечивает управление мультиплексором.

Теперь вы знаете, что такое системная шина, ее обобщенную организацию и основные термины, относящееся к понятию системная шина. Реализация СнК не возможна без системной шины. Однако действительно ли использование стандартизированных системной шины оправдано? Да, использование стандартизированных системных шин имеет ряд преимуществ, среди которых:

- Существует огромное количество IP-блоков, которые мы можем подключить в проект, который использует стандартизированную системную шину, без изменения исходных файлов IP-блока;
- Масштабируемость проекта – возможность с легкостью подключать новые IP-блоки или даже несколько одинаковых;
- Простота ПО для систем со стандартизированными системными шинами, так как организация адресного пространства таких шин достаточно простая.

Операции чтения и записи на системной шине называются транзакциями. Существует три вида транзакций, каждая из которых используется в определенных стандартах системных шин, в зависимости от назначения:

1. Single

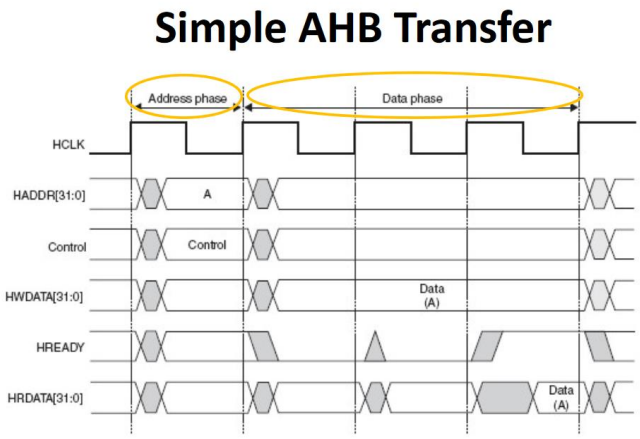

Рисунок 3.3 – Single транзакция.

2. Pipeline

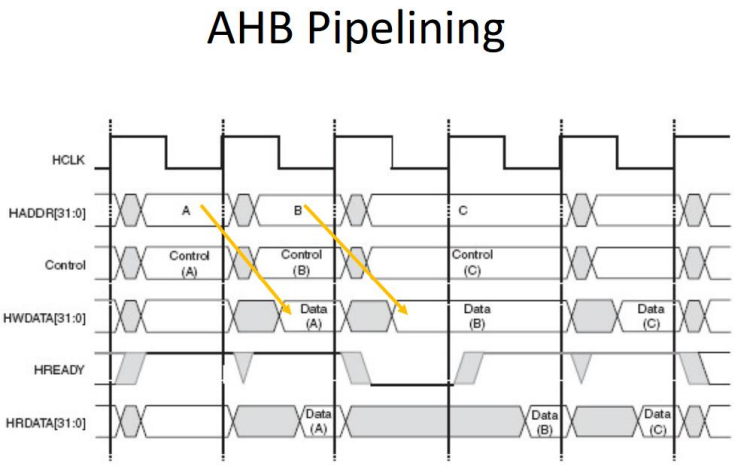

Рисунок 3.4 – Pipeline транзакция.

3. Burst

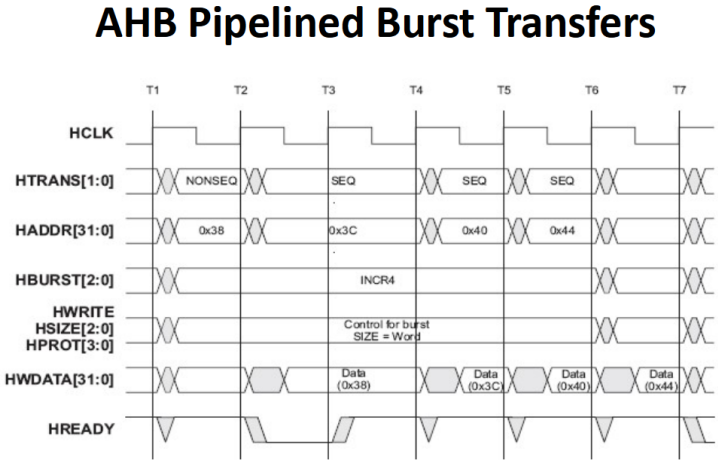

Рисунок 3.5 – Burst транзакция.

Транзакции отличаются скоростью передачи данных, сложностью арбитража и реализацией. Pipeline является промежуточным вариантом между двумя другим и используется для повышения производительности. Burst транзакция используется для обмена большим количеством данных, например для передачи данных во внешнюю память.
Рассмотрим типичную транзакцию по системной шине:

1. Ведущее устройство выбирает одно периферийное устройство и назначает адрес системной шине. В тоже время он устанавливает на шину управляющие сигналы (например, сигнал чтения);
2. Ведущее устройство ожидает ответа ведомого устройства (например, периферийного устройства).
3. Как только ведомое устройство готово, оно отправляет запрошенные данные ведущему устройству. Одновременно ведомое устройство устанавливает сигнал готовности на шине управления.
4. Наконец, мастер считывает переданные данные и может инициировать следующую транзакцию.

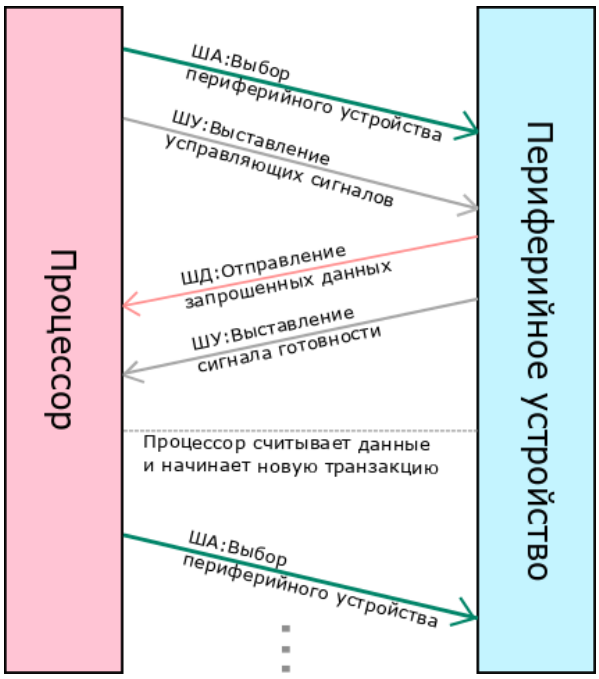

Рисунок 3.6 – Типичная транзакция по системной шине.

Классификация системных шин и существующие стандарты.

Используемые в настоящее время шины отличаются по:

- **Разрядности** (8, 16, 32, 64 бит). Чем больше разрядность шины, тем больше информации может быть передано за один цикл чтения или записи по каналу. Разрядность шины адреса можно определять независимо от разрядности шины данных. Разрядность шины адреса показывает, сколько ячеек памяти можно адресовать при передаче данных.
- **Способу передачи** сигнала (последовательные или параллельные)
- **Пропускной способности.** Ширина полосы пропускания называется также пропускной способностью и показывает общий объем данных, который можно передать по шине за данную единицу времени.
- Шины могут быть **синхронными** (осуществляющими передачу данных только по тактовым импульсам) и **асинхронными** (осуществляющими передачу данных в произвольные моменты времени).

Далее рассмотрим существующие стандарты системных шин.

Advanced Microcontroller Bus Architecture (AMBA) — это открытый стандарт требований к внутрикристалльным межсоединениям для соединения и управления функциональными блоками в системах на кристалле. Она облегчает развитие многопроцессорных разработок с большим числом контроллеров и периферии. Несмотря на название, с самого своего начала, AMBA имела виды, уходящие далеко за границы микроконтроллерных устройств. Сегодня AMBA широко применяется в ряде частей ASIC и SoC, включая прикладные процессоры, применяемые в современных небольших переносных устройствах вроде смартфонов.

Существует несколько разновидностей стандарта AMBA. На рисунке 3.7 представлены первые 4 стандарта AMBA. В таблице 3.1 представлены наиболее популярные шины стандартов AMBA.

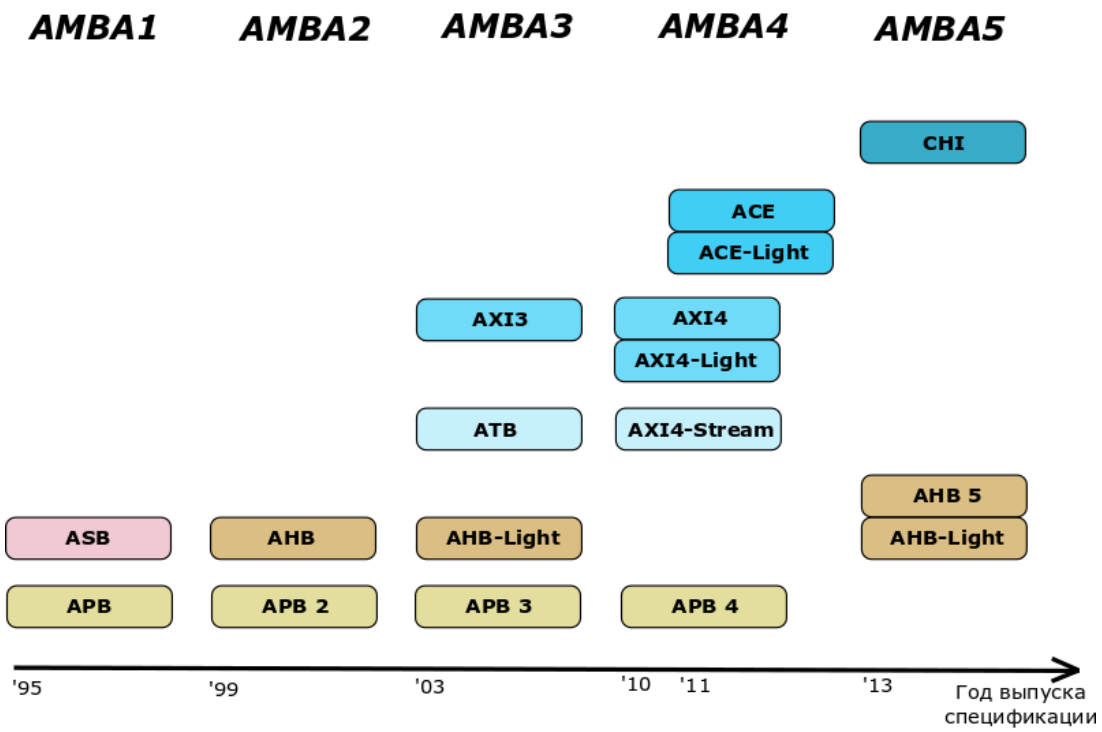

Рисунок 3.7 – Стандарты AMBA.

Таблица 3.1. Шины AMBA.

Стандарт	Шина	Описание
AMBA1	Advanced System Bus (ASB)	ASB — это первое поколение системной шины AMBA. ASB реализует функции, необходимые для высокопроизводительных систем, включая: пакетные передачи, конвейерные передачи, режим мульти-мастера (наличие более 1 ведущего устройства). В качестве ведущих устройств могут быть, например, процессорное ядро, ЦОС-процессор. В качестве ведомых устройств мост APB, устройства памяти, а также любые другие. Однако периферийные устройства с низкой пропускной способностью обычно находятся на шине APB.
AMBA1, AMBA2, AMBA3, AMBA4	Advanced Peripheral Bus (APB, APB3, APB4)	APB является частью иерархии шин AMBA и оптимизирована для минимального энергопотребления и снижения сложности интерфейса. APB выглядит как локальная вторичная шина, которая инкапсулирована как одно ведомое устройство AHB или ASB. Мост APB выглядит как подчиненный модуль, который обрабатывает передачу управляющих сигналов от имени локальной периферийной шины. APB следует использовать для взаимодействия с любыми периферийными устройствами, которые имеют низкую пропускную способность и не требуют высокой производительности интерфейса конвейерной шины.
AMBA2, AMBA5	Advanced High-performance Bus (AHB, AHB5)	AHB — это шина второго поколения шин AMBA, предназначенная для удовлетворения требований высокопроизводительных синтезируемых конструкций. Это высокопроизводительная системная шина, которая поддерживает несколько ведущих устройств шины и обеспечивает работу с высокой пропускной способностью. AHB реализует функции, необходимые для высокопроизводительных систем с высокой тактовой частотой, включая: пакетные передачи, разделенные транзакции, однотактная процедура арбитража мастеров, реализация передачи без третьего состояния и более широкие конфигурации шины данных (64/128 бит). Мостовое соединение между этим более высоким уровнем шины и текущим ASB / APB может быть эффективно выполнено, чтобы обеспечить простую

		интеграцию любых существующих конструкций. Широко применяется в разработках, основанных на ARM7, ARM9 и ARM Cortex-M.
AMBA3, AMBA4	Advanced Extensible Interface (AXI3, AXI4)	Протокол AXI поддерживает высокопроизводительные высокочастотные системы. Протокол AXI подходит для широкополосных сетей и систем с низкой задержкой, обеспечивает высокочастотную работу без использования сложных мостов, отвечает требованиям интерфейса широкого диапазона компонентов, подходит для контроллеров памяти с высокой начальной задержкой доступа, обеспечивает гибкость в реализации архитектур межсоединений, имеет обратную совместимость с существующими интерфейсами AHB и APB. Основные характеристики протокола AXI: отдельные фазы адреса и данных, поддержка передачи данных без выравнивания с использованием байтовых стробов, использует пакетные транзакции только с выданным начальным адресом, отдельные каналы чтения и записи данных, поддержка выдачи нескольких ожидающих адресов, поддержка завершения транзакции вне порядка. Широко применяется в процессорах ARM Cortex-A, включая Cortex-A9.
AMBA3, AMBA5	Advanced High-performance Bus Lite (AHB-Lite)	AHB-Lite является упрощенным вариантом шины AHB. В отличие от AHB, эта системная шина поддерживает одно ведущее устройство, отсутствует арбитраж. AHB-Lite обеспечивает работу в широкой полосе пропускания и реализует функции, необходимые для высокопроизводительных систем с высокой тактовой частотой, в том числе: пакетные передачи, реализация без синхронизации, широкие конфигурации шины данных, 64, 128, 256, 512 и 1024 бит. Наиболее распространенными ведомыми устройствами AHB-Lite являются устройства внутренней памяти, интерфейсы внешней памяти и периферийные устройства с высокой пропускной способностью. Хотя периферийные устройства с низкой пропускной способностью могут быть включены в качестве ведомых устройств AHB-Lite, по соображениям производительности системы они обычно находятся на APB. Мостовое соединение между этим более высоким уровнем шины и APB выполняется с использованием ведомого устройства AHB-Lite, известного как мост APB.
AMBA4	AXI Coherency Extensions (ACE)	Протокол ACE расширяет протокол AXI4 дополнительным средством оповещения передач широкой когерентности и обеспечивает поддержку аппаратно-согласованных кэшей. Свойство когерентности заключается в том, что измененная одним ведущим устройством область памяти принимает новое значение и для остальных ведущих устройств. Протокол ACE обеспечивает: Барьерные транзакции, которые гарантируют упорядочение транзакций в системе, функциональность распределенной виртуальной памяти для управления виртуальной памятью.
AMBA4	AXI Coherency Extensions Lite (ACE-Lite)	Интерфейс ACE-Lite является подмножеством полного интерфейса ACE. ACE-Lite используется ведущими компонентами, которые не имеют аппаратных когерентных кэшей. Lite состоит из интерфейса AXI4 с дополнительными сигналами на канале чтения адреса и канале записи адреса. ACE-Lite не включает в себя: адресный канал snoop, канал ответа snoop, сигнал подтверждения чтения, сигнал подтверждения записи, любые дополнительные биты ответа чтения.
AMBA4	Advanced	Протокол AXI включает в себя спецификацию AXI4-Lite,

	Extensible Interface 4 Lite (AXI4-Lite)	подмножество AXI4 для связи с более простыми интерфейсами управления внутри компонентов, которые не требуют полной функциональности AXI4. Основные функциональные возможности AXI4-Lite: все транзакции имеют длину пакета 1, все обращения к данным используют полную ширину шины данных - AXI4-Lite поддерживает ширина шины данных 32-битная или 64-битная, все обращения не изменяются, не буферизируются, исключительные обращения не поддерживаются.
AMBA4	Advanced Extensible Interface 4 Stream (AXI4-Stream v1.0)	Протокол AXI4-Stream используется в качестве стандартного интерфейса для подключения компонентов, которые хотят обмениваться данными. Интерфейс может использоваться для подключения одного ведущего устройства, которое генерирует данные, к одному подчиненному устройству, которое получает данные. Протокол также можно использовать при подключении большего количества основных и подчиненных компонентов. Протокол поддерживает несколько потоков данных, используя один и тот же набор общих проводов, что позволяет создать общее межсоединение, которое может выполнять операции увеличения, уменьшения размера и маршрутизации. Интерфейс AXI4-Stream также поддерживает широкий спектр различных типов потоков. Потоковый протокол определяет связь между передачами и пакетами.

Сегодня эти протоколы являются де-факто стандартными для встраиваемых процессоров, поскольку они хорошо описаны и могут применяться без отчислений.

Еще одним из популярных на текущий день стандартов является **Wishbone**. Шина Wishbone — параллельная шина для объединения модулей в системе на кристалле. Шина описана в открытой спецификации, и широко используется в проектах цифровых систем с открытым исходным кодом на сайте OpenCores.org. Стандарт допускает присутствие нескольких ведущих устройств в системе, а также различные топологии соединения модулей.

Общие характеристики:

- ширина шин адреса и данных: 8, 16, 32, 64 бит;
- тип шины: параллельная;
- внутренняя шина, используется только для соединения модулей на кристалле.

Адресное пространство системы на кристалле.

При подключении различных периферийных устройств к системной шине возникает вопрос, каким образом различать эти контроллеры между собой? Решением данной проблемы является присвоение индивидуального диапазона адресов каждому из контроллеров. Таким образом, взаимодействие процессора с конкретным контроллером происходит посредством обращения по адресам заданного диапазона.
Весь доступный диапазон адресов называется адресным пространством (Memory Map) системы на кристалле, а диапазон адресов, выделенный для отдельно взятого контроллера, - адресным пространством контроллера. Базовым адресом контроллера называется адрес начала выделенного диапазона. Структурная схема адресного пространства системы на кристалле приведена на рисунке 3.8.

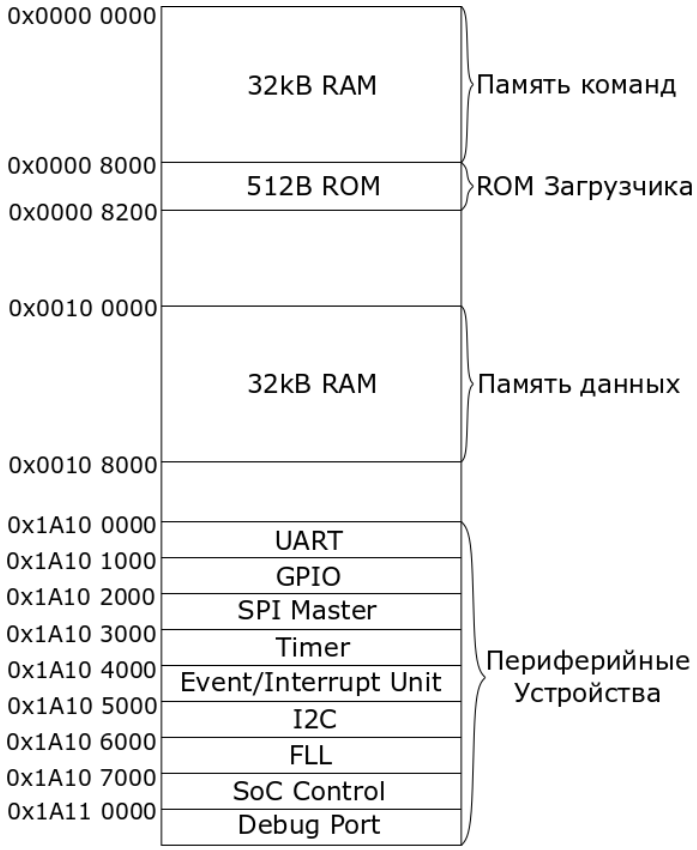

Рисунок 3.8 – Структурная схема адресного пространства.

Адресное пространство, выделенное под каждое периферийное устройство, определяется его стартовым (базовым) адресом и размером адресного пространства. Для обмена данными в периферийном устройстве (контроллере или вычислительном блоке) существует набор регистров, каждый из которых имеет уникальный адрес. Регистры адресуются относительно стартового адреса. Абсолютный адрес регистра в адресном пространстве системы на кристалле определяется как сумма базового адреса периферийного устройства и смещения (относительного адреса).
Так как архитектура RISC-V является load-store архитектурой, то для обращения к ячейке адресного пространства используются инструкции load и store для загрузки данных в регистры процессора и записи данных из регистра в ячейку памяти соответственно.
Следует отметить, что в RISC-V используется побайтовая адресация. Так как шина данных является 32-разрядной, то при обращении по адресу ноль происходит считывание или запись первых четырех байт (0,1,2,3). Этот факт необходимо учитывать при проектировании вычислительных блоков и при назначении регистрам адресов.

Системная шина APB.

В нашей системе на кристалле мы используем системную шину **APB (Advanced Peripheral Bus)**.

Спецификация: https://web.eecs.umich.edu/~prabal/teaching/eecs373-f12/readings/ARM_AMBA3_APB.pdf

Шина APB — это часть семейства шин AMBA 3 фирмы ARM. Она представляет собой универсальный интерфейс для подключения периферийных устройств.
На рисунке 3.9 представлена диаграмма системной шины APB.Декодер управляется ведущим устройством, которое выставляет на системную шину адрес одного из ведомых устройств. По адресу декодер определяет, какое из устройств выбрано для обмена данными и формирует сигнал *PSEL*.Декодер формирует сигнал *Select* для мультиплексора, который указывает, данные какого из ведомых устройств должны быть выставлены на системную шину.

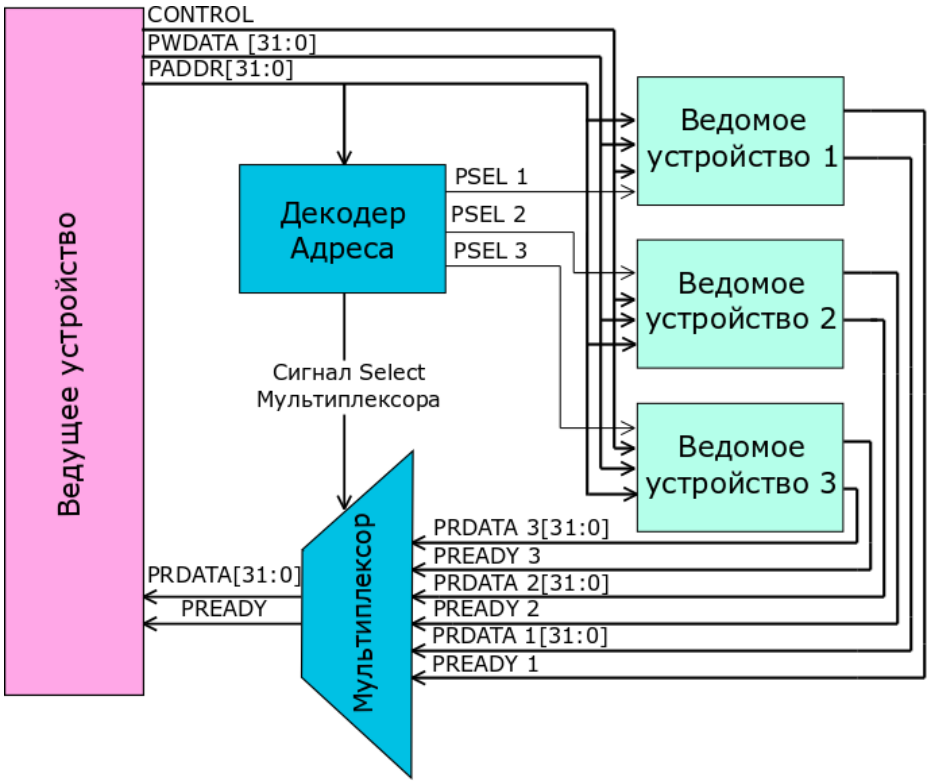

Рисунок 3.9 – Диаграмма системной шины APB.

Передача данных по шине APB состоит из двух фаз: фазы адреса и фазы данных. Фаза адреса всегда занимает один такт шины, а фаза данных может содержать состояния ожидания и длиться несколько тактов. Простейший цикл записи без состояний ожидания происходит следующим образом (рис 3.10). В фазе адреса по нарастающему фронту тактового сигнала PCLK ведущее устройство устанавливает следующие сигналы:

- адрес ведомого устройства PADDR;
- сигнал записи PWRITE (активный уровень — высокий);
- сигнал выбора устройства PSEL (активный уровень — высокий);
- данные для записи PWDATA.

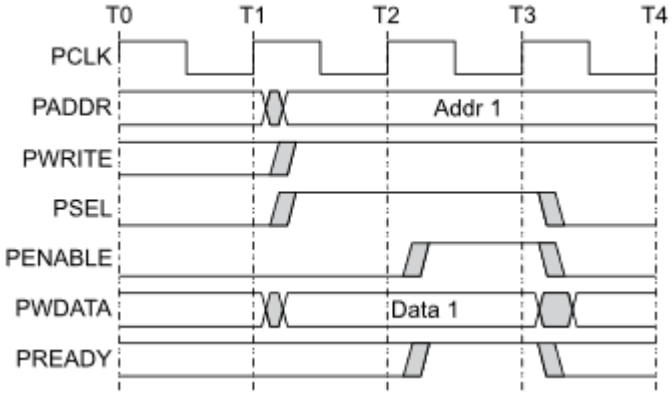

Рисунок 3.10 – Цикл записи по шине APB без состояния ожидания.

Состояния этих сигналов сохраняются и в фазе данных. По второму фронту тактового сигнала устанавливается сигнал PENABLE (активный уровень — высокий). Это означает начало фазы записи данных. До следующего такта ведомое устройство должно установить сигнал PREADY(активный уровень — высокий) и принять передаваемые данные. Получив сигнал PREADY, ведущее устройство по третьему такту снимает сигнал PENABLE. Сигнал выбора PSEL при этом также снимается, даже если следующее обращение будет происходить к тому же самому устройству. На этом цикл записи заканчивается.

Периферийное устройство может задержать окончание цикла записи (рис 3.11). Для этого оно должно при активном сигнале PENABLE не устанавливать сигнал PREADY до тех пор, пока не

закончит прием данных. В таком случае цикл записи закончится по первому фронту тактового сигнала, на котором будет обнаружен активный уровень PREADY.

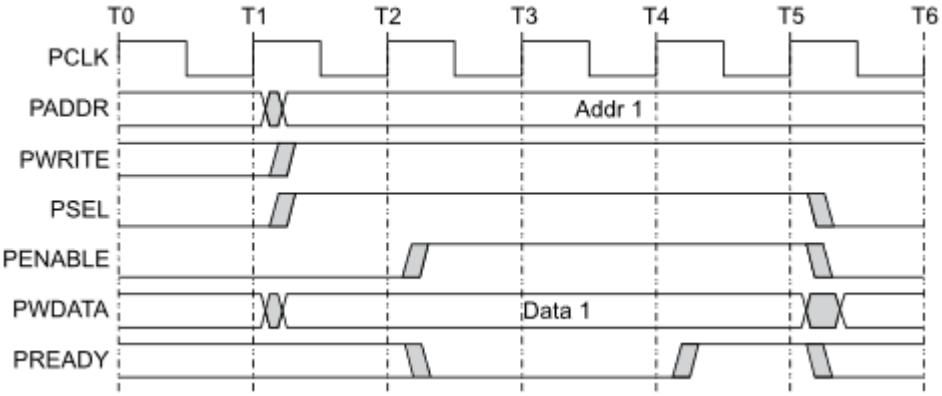

Рисунок 3.11 – Цикл записи по шине APB с состоянием ожидания.

Временные диаграммы циклов чтения без состояний ожидания (рис 3.12) и с ними (рис 3.13) выглядят похоже. Первое отличие состоит в том, что в фазе адреса сигнал PWRITE имеет низкий уровень. В этом случае при активном уровне сигнала PENABLE ведомое устройство должно выставить данные на линиях чтения PRDATA, сопровождая их активным уровнем сигнала PREADY. Если периферийное устройство хочет задержать цикл чтения, оно должно снять сигнал PREADY при высоком уровне сигнала PENABLE. Тогда ведущее устройство перейдет в состояние ожидания до тех пор, пока не получит активного уровня сигнала PREADY.

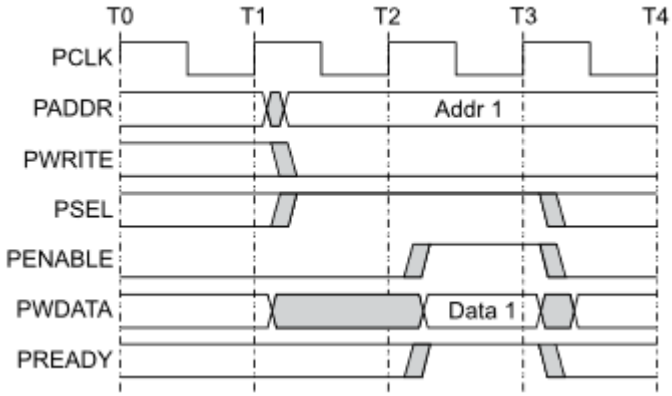

Рисунок 3.12 – Цикл чтения по шине APB без состояния ожидания.

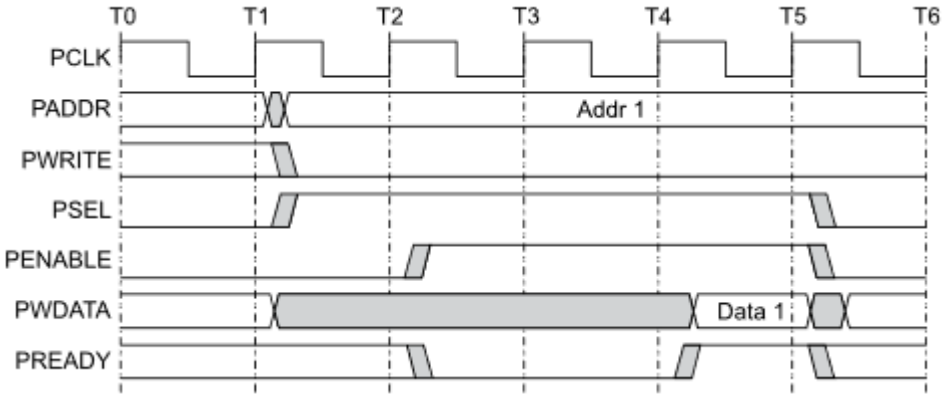

Рисунок 3.13 – Цикл чтения по шине APB с состоянием ожидания.

В таблице 3.2 представлены сигналы системной шины APB:

Таблица 3.2 – Сигналы шины APB.

Имя сигнала	**Описание**
PCLK	Тактовая частота. Все действия по шине APB происходят по переднему фронту сигнала PCLK.

PRESETn	Сброс. Активный уровень сигнала – низкий.
PADDR	Шина адреса. Может иметь разрядность до 32 бит. Поступает от ведущего устройства.
PSEL	Выбор. Ведущее устройство вырабатывает этот сигнал отдельно для каждого периферийного обустройства. Этот сигнал показывает, что ведомое устройство выбрано для обмена данными.
PENABLE	Разрешение. Этот сигнал указывает на фазу данных (второй и следующие такты передачи по APB).
PWRITE	Направление. Высокий уровень сигнала указывает на цикл записи данных в периферийное устройство, а низкий – цикл чтения из периферийного устройства.
PWDATA	Шина данных для записи. Может иметь разрядность до 32 бит.
PREADY	Сигнал готовности периферийного устройства.
PRDATA	Шина данных для чтения. Может иметь разрядность до 32 бит.
PSLVERR	Ошибка передачи. Периферийные устройства не обязаны его поддерживать. Если сигнал не используется, то не соответствующий вход ведущего устройства подается низкий уровень.

Проектирование аппаратного вычислителя.

Постановка задачи: спроектировать аппаратный вычислитель с подключением по системной шине APB. Для примера рассмотрим проектирование вычислителя контрольной суммы CRC8. Для каких целей может потребоваться такой блок? Например, если устройство, в состав которого входит вычислитель, должно в режиме реального времени принимать массивы данных и проверять их целостность. Тогда полученный массив данных отправляется на блок CRC8, в котором происходит вычисление и далее через механизм прямого доступа к памяти записывается в память. Таким образом, можно проверять целостность данных, не затрачивая при этом процессорное время, как при использовании программных реализаций алгоритма.
Вычислитель должен принимать на вход данные по системной шине, от которых он вычисляет контрольную сумму и по сигналу чтения выдавать на выходную шину данных значение CRC8.
Обобщенная структурная схема вычислителя CRC8, подключенного по шине APB приведена на рис. 3.14.

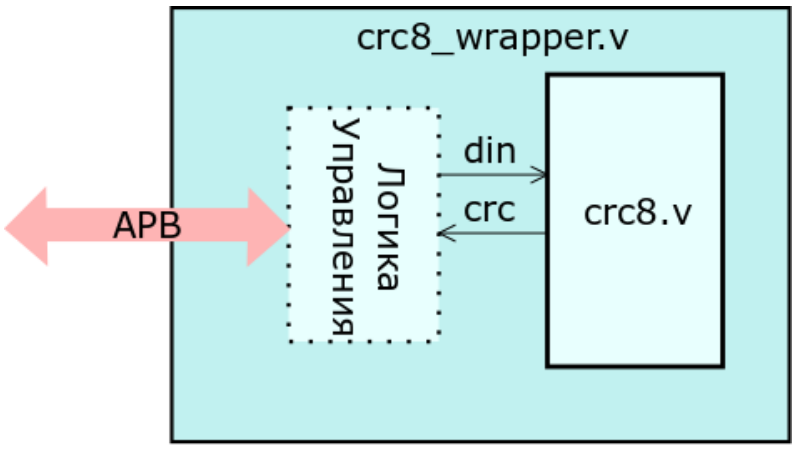

Рисунок 3.14– Структурная схема вычислителя CRC8.

Вычислитель состоит из двух частей: модуля, реализующего непосредственно сам алгоритм CRC8 и модуля-оболочки, в который подключается модуль-вычислитель контрольной суммы. Также модуль-оболочка содержит в себе логику управления системной шиной.
Рассмотрим подробнее механизм управления обмена информацией по системной шине, который реализует модуль-оболочка. Как было сказано ранее, каждому из периферийных устройств выделено свое адресное пространство с базовым адресом. При подключении вычислителя в проект его будет нужно указать, а пока необходимо разобраться с адресацией внутри адресного пространства вычислителя. Ранее было рассказано про смещения, относительно базового адреса, которые определяют абсолютные адреса регистров.

$$Addr_{reg} = Addr_{base} + Offset_{reg},$$

где $Addr_{base}$ – базовый адрес контролера, а $Offset_{reg}$ – смещение относительно базового адреса. Регистры вычислительного блока хранят в себе данные, которые потребуются для считывания или записанные данные. Какие регистры могут потребоваться для данного вычислителя? В самом базовом случае это будет регистр для записи данных для вычисления и регистр для считывания значения CRC. Таким образом получается два регистра, а значит два адреса, каждый из которых будет иметь свое уникальное смещение. В системе используется 32-разрядная системная шина и побайтовая адресация, поэтому необходимо использовать адреса кратные 4, чтобы покрывать пословно адресное пространство.

Реализация вычислителя CRC8.

Для примера рассмотрим проектирование вычислителя контрольной суммы CRC8. В данном вычислительном блоке будет модуль, реализующий алгоритм CRC8, а также модуль-оболочка. Начнем с проектирования модуля, реализующего алгоритм CRC8 (Заданный порождающий полином: $g(x) = x^8+x^5+x^4+1$).
Во-первых, на вход модуля поступают:

- сигнал тактирования *clk_i;*
- сигнал сброса *rst_i*;
- сигнал валидности данных data_valid_i;
- шина данных *din_i;*
- сигнал запроса на чтения *crc_rd* вычисленного значения CRC8 - *crc_o.*
  Разрядность данных была выбрана 8 бит.

Работа модуля построена по принципу работы конечного автомата с 3 состояниями (рис.3.15):

Таблица 3.3 – Состояния автомата модуля-вычислителя.

Состояние	Описание
IDLE	Состояние бездействия. Ожидание сигнала валидности данных или запроса на считывание CRC8.
BUSY	Состояние вычисления CRC8.
READ	Состояние считывания значения CRC8.

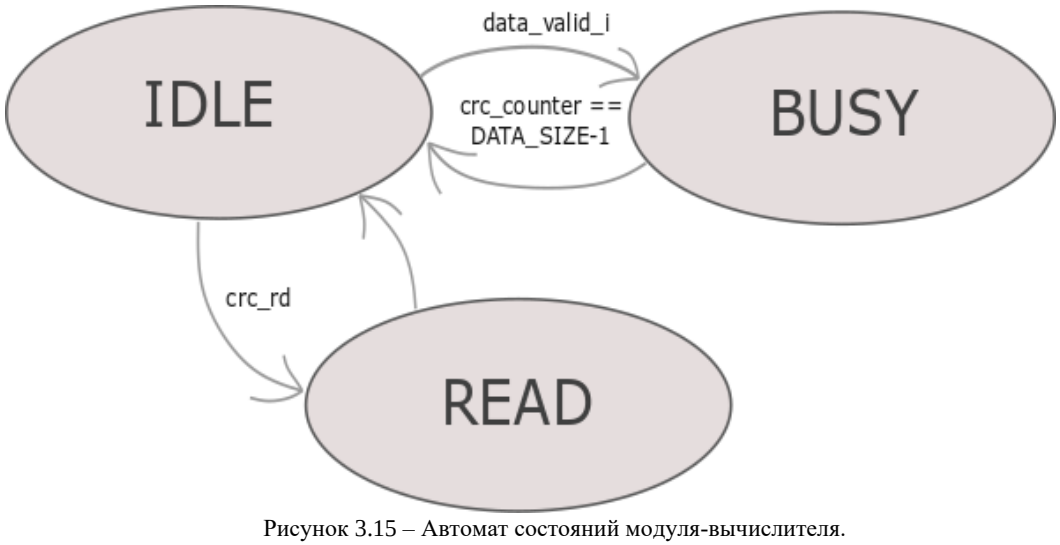

Рисунок 3.15 – Автомат состояний модуля-вычислителя.

*crc_counter - счетчик для расчёта количества бит входного байта данных, обработанных алгоритмом вычисления контрольной суммы CRC8..

Код модуля-вычислителя с комментариями приведен в Листинге 3.1.
Листинг 3.1 – Модуль-вычислитель контрольной суммы CRC8

```
module crc8 (
            input           clk_i,
            input           rst_i,
            input      [7:0] din_i,
            input           data_valid_i,
            input           crc_rd,
            output reg [7:0] crc_o);


// Параметры для состояний автомата
localparam IDLE = 2'b00;
localparam BUSY = 2'b01;
localparam READ = 2'b10;

reg  [2:0] state;        // Регистр состояний
reg  [7:0] data_current; // Текущие данные (сдвиговый регистр)
reg  [3:0] crc_counter;   // Регистр счетчик обработанных бит входного байта данных для состояния вычисления

always @(posedge clk_i)
begin
    if (rst_i) // Сигнал сброса – обнуляем все регистры
    begin
        state        <= IDLE;
        data_current <= 8'b0;
        crc_o        <= 8'b0;
        crc_counter   <= 4'b0;
    end
    else
    begin
        case (state)
        IDLE:
            begin
                crc_counter <= 4'b0;
                if (data_valid_i) // Если пришли новые данные – переходим
                                  // в состояние вычисления
                begin
                    state        <= BUSY;
                    data_current <= din_i;
                end
                else if (crc_rd)state <= READ; // Если пришел запрос на чтение
                                               // переходим в состояние чтения
            end
        BUSY:
            begin
                crc_o[7] <= crc_o[0]^data_current[0];
                crc_o[6] <= crc_o[7];
                crc_o[5] <= crc_o[6];
                crc_o[4] <= crc_o[5];
                crc_o[3] <= (crc_o[0] ^ data_current[0])^ crc_o[4];
                crc_o[2] <= (crc_o[0] ^ data_current[0])^ crc_o[3];
                crc_o[1] <= crc_o[2];

                crc_o[0] <= crc_o[1];
                data_current <= {1'b0,data_current[7:1]};
                crc_counter   <= crc_counter+ 1'b1;
                if(crc_counter == 4'b111) state <= IDLE;
            end
        READ:
            begin
                crc_o <= 8'b0;
                state <= IDLE;
```

```
            End
          endcase
        end
      end
endmodule
```

Следующим шагом является написания модуля-оболочки.
Так как реализуется контроллер, совершающий обмен данными по интерфейсу APB с 32-разрядной шиной в заголовке необходимо указать все сигналы интерфейса. Фрагмент кода, содержащий заголовок модуля оболочки приведен в Листинге 3.2.

Листинг 3.2 – Заголовок модуля-оболочки

```
module wrapper_crc8(
                    input             p_clk_i,
                    input             p_rst_i,
                    input      [31:0] p_dat_i,
                    output reg [31:0] p_dat_o,
                    input             p_sel_i,
                    input             p_enable_i,
                    input             p_we_i,
                    input      [31:0] p_adr_i,
                    output reg        p_ready
                    );
```

Для подключения модуля-вычислителя CRC необходимо объявить wire, которые будут подключаться к выводам модуля. Фрагмент кода приведен в Листинге 3.3.

Листинг 3.3 – Подключение модуля-вычислителя в модуль-оболочку

```
wire [7:0] din_i; // Объявляем провода, которые будут подключаться к
                  // сигналам модуля
wire [7:0] crc_o;
wire       crc_rd;
wire       data_valid_i;

crc8 crc8(.clk_i(p_clk_i),  // При подключении модуля указываем имя
                            // и название модуля name module_name (...
          .rst_i(!p_rst_i), // Подключаем к каждому сигналу модуля
                            // провод .signal_name(wire_name), ...
          .din_i(din_i),
          .data_valid_i(data_valid_i),
          .crc_rd(crc_rd),
          .crc_o(crc_o));
```

Ранее было определено, что в базовой реализации будут использоваться два адреса, для обмена данными по системной шине – запись данных и чтение с адресами смещений 0 и 4 соответственно. Также необходимо генерировать строб cs, который будет определять фазу данных транзакции для того, чтобы считать данные с шины либо выставить данные на нее. Фрагмент кода приведен в Листинге 3.4.

Листинг 3.4 – Генерация строба cs

```
reg cs_1;
reg cs_2;
// Формирование строба cs цикла чтения или записи по системной шине
always @ (posedge p_clk_i)
begin
    cs_1 <= p_enable_i & p_sel_i;
    cs_2 <= cs_1;
end
wire cs = cs_1 & (~cs_2);
```

Листинг 3.5 – Формирование выходных данных системной шины и данных на модуль-вычислитель

```
// Формирование выходных данных системной шины
always @ (*)
begin //Для чтения crc используем адрес 1
  if (cs & (~p_we_i) & p_adr_i[3:0] == 4'd4 )p_dat_o <= {24'b0, crc_o};
end

// Формирование сигналов на модуль-вычислитель

//Для записи данных для расчета crc используем адрес 0
assign data_valid_i = (cs &  p_we_i & p_adr_i[3:0] == 4'd0);

//Для записи данных для расчета crc используем адрес 0
assign din_i = (cs & p_we_i & p_adr_i[3:0] == 4'd0);

//Для чтения crc используем адрес 4
assign crc_rd = (cs & ~p_we_i & p_adr_i[3:0] == 4'd4);
```

Листинг 3.6 – Формирование сигнала p_ready

```
reg cs_ack1;
reg cs_ack2;
// Формирование сигнала готовности системной шины p_ready
always @ (posedge p_clk_i)
  begin
   cs_ack1 <= cs_2;
   cs_ack2 <= cs_ack1;
 end

always @ (posedge p_clk_i)
  begin
    p_ready <= (cs_ack1 & (~cs_ack2));
  end
```

Как можно видеть в Листинге 3.5, на выходную шину данных значение crc выставляется только тогда, когда пришел запрос на обмен данными (строб cs), сигнал операции чтения *(~p_we_i)* и совпадает адрес (*p_adr_i == 32'd4*). Аналогично выполняется операция записи, с той лишь разницей, что сигнал *p_we_i* должен быть равен единице (рис.3.16 и рис.3.17).

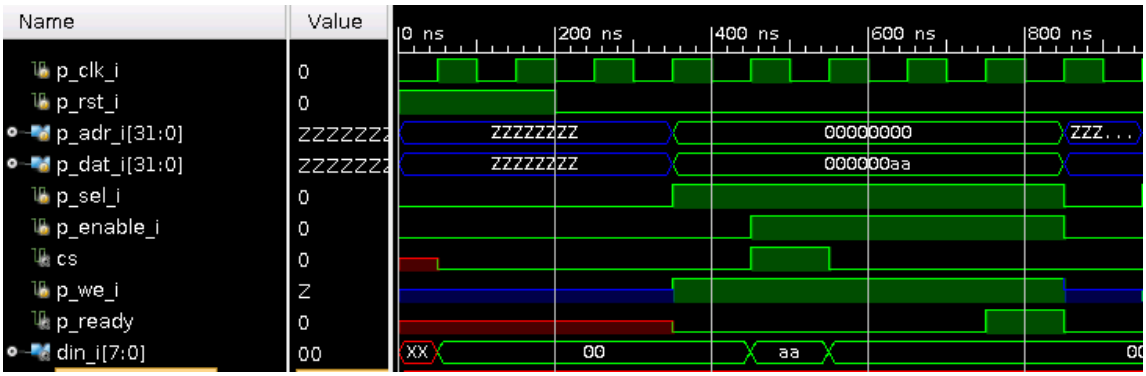

Рисунок 3.16 – Запись по адресу 32’d0.

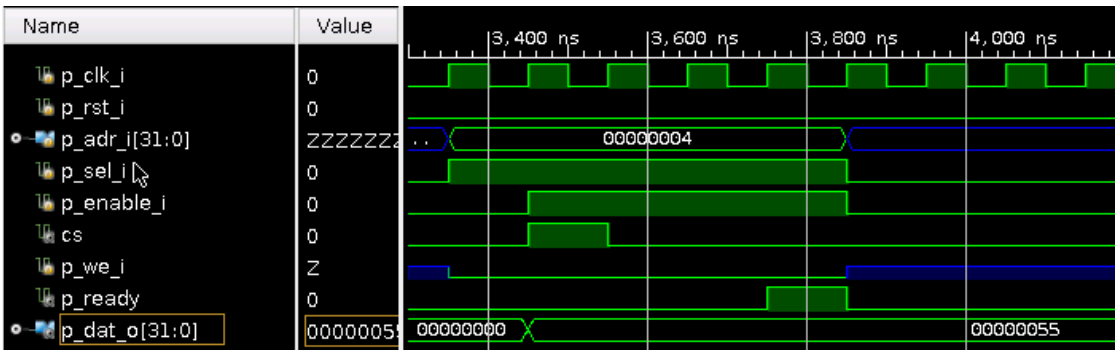

Рисунок 3.17 – Чтение по адресу 32'd4.

Моделирование работы блока CRC8.

Следующая задача состоит в том, чтобы проверить написанный нами вычислитель. Одним из инструментов верификации является моделирование. Для моделирования работы вычислителя или любого отдельного модуля и даже всей системы существуют специальные модули – testbench. Testbench имитирует входные воздействия и подает их на проверяемый модуль.
Для проверки блока сымитируем две транзакции записи данных по системной шине и одну транзакцию чтения значения CRC8.
Для проверки блока необходимо подключить его в testbench (аналогичным образом, как и модуль-вычислитель в оболочку). Фрагмент кода приведен в Листинге 3.7.

Листинг 3.7 – Подключение вычислителя в testbench

```
reg         p_clk_i;
reg         p_rst_i;
reg  [31:0] p_dat_i;
wire [31:0] p_dat_o;
reg         p_enable_i;
reg         p_sel_i;
reg         p_we_i;
reg  [31:0] p_adr_i;
wire        p_ready;

wrapper_crc8 wrapper_crc8
      ( .p_clk_i    (p_clk_i),
        .p_rst_i    (p_rst_i),
        .p_dat_i    (p_dat_i),
        .p_dat_o    (p_dat_o),
        .p_enable_i (p_enable_i),
        .p_sel_i    (p_sel_i),
        .p_we_i     (p_we_i),
        .p_adr_i    (p_adr_i),
        .p_ready    (p_ready)
        );
```

Следует обратить внимание, что в testbench при подключении проверяемого модуля используются регистры для входных сигналов и провода для выходных. Таким образом можно генерировать входные воздействия.
Одной из самых простых конструкций testbench является «initial»-блок, который определяет, какие действия должны быть сделаны при старте программы. Этот блок не является синтезируемым. С помощью такого блока зададим начальные значения сигналов. Фрагмент кода приведен в Листинге 3.8.

Листинг 3.8 – initial-блок генерации начальных значений сигналов

```
initial
begin
    p_dat_i    = 'hz;
    p_enable_i = 0;
    p_sel_i    = 0;
    p_we_i     = 'hz;
    p_adr_i    = 'hz;
    p_rst_i    = 1;
    #200
    p_rst_i    = 0; // Запись #200 обозначает что смена значения
                    // сигнала сброса произойдет через 200нс.
end
```

Еще одна конструкция, которую часто используют при моделировании – «forever»-блок, который обеспечивает циклическое исполнение куска кода. Такой блок очень полезен для генерации сигнала тактовой частоты. Фрагмент кода приведен в Листинге 3.9.

Листинг 3.9 – Генерация сигнала тактовой частоты

```
initial
begin
    p_clk_i=0;
    forever #50 p_clk_i = ~p_clk_i; // Сигнал инвертируется каждые 50нс
end
```

Следующей задачей при написании testbench контроллера является имитация обмена данными по системной шине – циклов чтения и записи. Такие конструкции могут быть использованы при тестировании очень много раз. Например, для нашего случая это может быть посылка 20 байт для вычисления контрольной суммы с последующим считыванием результата. Использовать «initial»-блок в таких случаях просто нерационально, так как он будет содержать в себе большое число однотипных действий. Удобно использовать «task»-блоки, которые содержат в себе эти последовательности действий, например последовательность действий при цикле записи по системной шине. А далее вызывать эти «task»-блоки в «initial»-блоке с необходимыми нам параметрами.
Также удобно использовать вывод информации на консоль при помощи display.
Пример «task»-блока для записи по APB приведен в Листинге 3.10.

Листинг 3.10 – task для записи по системной шине APB

```
task write_register; // Название task
input [31:0] reg_addr; // Параметры передаваемые в task,
                       // в нашем случае адрес и данные
input [31:0] reg_data;
begin
  @ (posedge p_clk_i); // Ожидаем один такт
  // Формируем посылку согласно документации на APB
  p_adr_i    = reg_addr; // Выставляем значения на шины адреса и данных
  p_dat_i    = reg_data;
  p_enable_i = 0;
  p_sel_i    = 1;
  p_we_i     = 1;

  @ (posedge p_clk_i); // Ожидаем один такт

  p_enable_i = 1;

  wait (p_ready); // Ожидаем появление сигнала p_ready

// Вывод информации о совершенной операции
  $display("(%0t) Writing register [%0d] = 0x%0x", $time, p_adr_i, reg_data);
  @ (posedge p_clk_i);

  // Возвращаем сигналы в исходное состояние
  p_adr_i    = 'hz;
  p_dat_i    = 'hz;
  p_enable_i = 0;
  p_sel_i    = 0;
  p_we_i     = 'hz;
end
endtask
```

Аналогичным образом написан task для чтения по APB. Фрагмент кода приведен в Листинге 3.11.

Листинг 3.11 – task для чтения по системной шине APB

```
task read_register;
input [31:0] reg_addr;
begin
    @ (posedge p_clk_i);

    p_adr_i    = reg_addr;
    p_enable_i = 0;
    p_sel_i    = 1;
    p_we_i     = 0;

    @ (posedge p_clk_i);

    p_enable_i = 1;

    wait (p_ready);

    $display("(%0t) Reading register [%0d] = 0x%0x", $time, p_adr_i, p_dat_o);

    @ (posedge p_clk_i);

    p_adr_i    = 'hz;
    p_enable_i = 0;
    p_sel_i    = 0;
    p_we_i     = 'hz;
end
endtask
```

В блоке initial вызван два раза task для записи по системной шине и один раз для чтения. Фрагмент кода приведен в Листинге 3.12.

Листинг 3.12 – Вызов блоков task в блоке initial

```
initial
begin
    write_register(32'd0, 32'hAA);
    #1200 write_register(32'd0, 32'h33);
    #1200 read_register(32'd4);
end
```

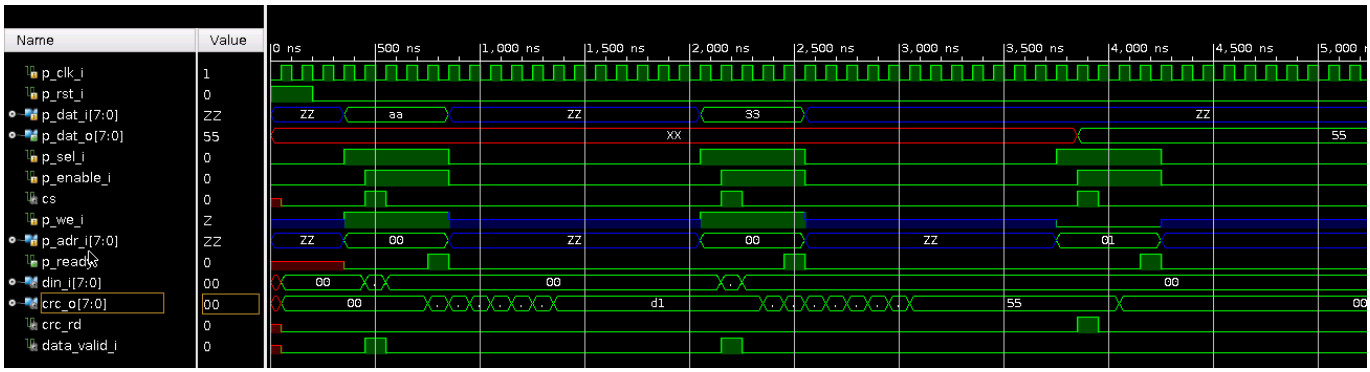

Рисунок 3.18 – Результат моделирования.

На временной диаграмме, изображенной на Рисунке 3.18, видно 2 цикла записи (данные 32’hAA и 32’h33), и один цикл чтения результата вычисления CRC.