**Draft**

16

SELEN

Reference manual

Процессорная система на кристалле Selen: руководство и описание

**https://github.com/Zhikharev/selen**

*Список изменений*

|  |  |
| --- | --- |
| **29.12.15** | **Добавлена таблица интерфейса взаимодействия ядра с кэш-памятью** |
| **04.01.16** | **Изменена разрядность сигнала core\_req\_size с 4 до 3 бит** |
| **05.01.16** | **Добавлено описание характеристик кэш-памяти 1 уровня** |
| **07.01.16** | **Добавлено более подробное описание интерфейсов ядра** |
| **11.01.16** | **Добавлен шаблон оглавления** |
| **16.01.16** | **Из интерфейса ядра удалён сигнал core\_req\_be, он будет формироваться в L1 исходя из core\_req\_size и core\_req\_addr** |
| **17.01.16** | **Добавлены временные диаграммы интерфейса ядра с кэшем инструкций** |
| **21.01.16** | **Добавлено описание взаимодействия ядра с кэшем данных** |
| **24.01.16** | **Добавлена блок-схема конвейера ядра** |
| **29.01.16** | **Добавлено начальное описание конвейера. Добавлены диаграммы взаимодействия с подсистемой памяти** |
| **30.01.16** | **Добавлено описание ядра** |
| **04.02.16** | **Улучшено форматирование, добавлены новые рисунки. Описание ядра дополнено, исправлена блок схема проекта** |

Оглавление

[Общее описание системы 3](#_Toc442351301)

[Процессорный кластер 4](#_Toc442351302)

[Описание ядра 4](#_Toc442351303)

[Описание конвейера 4](#_Toc442351304)

[Взаимодействие с подсистемой памяти 8](#_Toc442351305)

[Взаимодействие с кэш-памятью инструкций 8](#_Toc442351306)

[Взаимодействие с кэш-памятью данных 9](#_Toc442351307)

[Кэш-память первого уровня 10](#_Toc442351308)

[Периферийный кластер 11](#_Toc442351309)

[SPI 11](#_Toc442351310)

[Программная модель 11](#_Toc442351311)

[Прототип 11](#_Toc442351312)

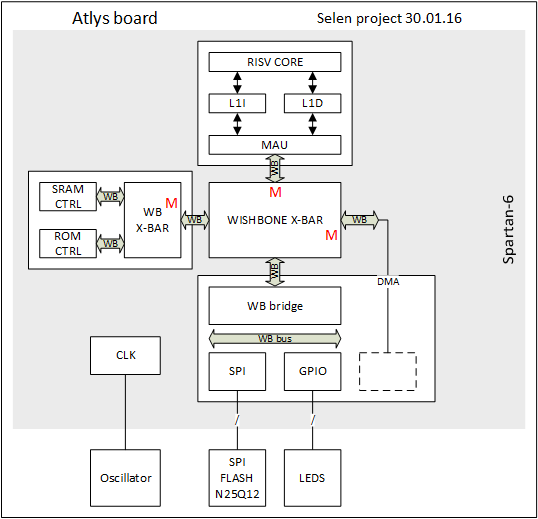
# 

# ****Общее описание системы****

**Selen – одноядерная микропроцессорная система на кристалле. В состав СнК входят:**

* **32-ух разрядное процессорное ядро с архитектурой системы команд RISC-V;**
* **Кэш-память первого уровня (инструкций и данных);**
* **Wishbone коммуникационная среда;**
* **ROM память;**
* **SRAM память;**
* **SPI контролер;**
* **GPIO контролер.**

**Система ориентирована для синтеза на базе ПЛИС фирмы Xilinx Spartan-6. В качестве прототипа используется отладочная плата фирмы Digilent Atlys. Блок схема проекта представлена на рисунке 1.**



**Рисунок 1. Блок схема проекта Selen**

# ****Процессорный кластер****

**Процессорный кластер состоит из 32-ух битного ядра, поддерживающего систему команд RISCV (см. приложение 1), кэш-памяти инструкций и кэш-памяти данных и блока доступа в память (Memory Access Unit), поддерживающего интерфейсов Wishbone.**

## ****Описание ядра****

В проекте представлено ядро собственной разработки, поддерживающее систему команд RISCV. Принципиальная блок схема ядра представлена на рисунке 2.

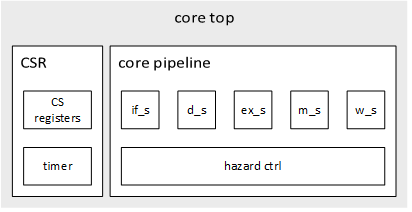


Рисунок 2. Принципиальная блок схема ядра

Основным элементом ядра является вычислительный конвейер core pipeline. Конвейер состоит из пяти стадий (if\_s, d\_s, ex\_s, m\_s, w\_s) и блока обработки конфликтов hazard\_ctrl.

Ядро содержит набор управляющих и статусных регистров CS registers и счётчик системных тактов timer (CSR – control status registers). В управляющих регистрах хранится информация об окне некэшируемых запросов.

## Описание конвейера

Принципиальная блок схема конвейера представлена на рисунке 3.

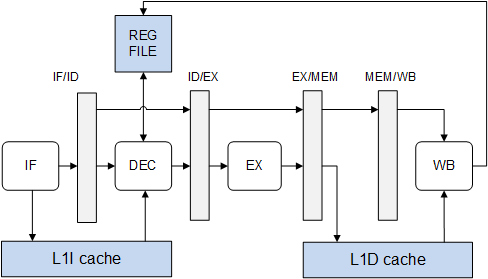


Рисунок 3. Блок схема конвейера

Конвейер состоит из пяти стадий:

1. IF (Instruction Fetch) – стадия подкачки инструкций. На данной стадии происходит выставление запроса в кэш-инструкций и формирование нового значения программного счётчика (NPC). На следующею стадию передается инструкция, текущее значение программного счетчика(current pc) и значение программного счетчика +4(pc next = current pc + 4) Возможные остановки: промах в кэш-памяти инструкций или полная остановка конвейера;
2. DEC (Decode) – стадия декодирования. На данной стадии происходит декодирование инструкции и формирование сигналов управления. Кроме того осуществляется доступ к регистрового файлу. На данной стадии может сформироваться сигнал остановки стадий IF и DEC при промахе в кэш-память инструкций (с формирование nop для следующих стадий). На следующею стадию передаются данные описанные в таблице 1;
3. EX (Execution) – стадия вычислений. На данной стадии происходит выполнение арифметико-логических операций и операции формирования адресов (адрес доступа в память, адрес перехода). Возможные остановки: полная остановка конвейера. На следующую стадию передаются данные описанные в таблице 2;
4. MEM (Memory) – стадия обращения в память. На данной стадии происходит обращение в кэш данных для инструкций доступа в память. Возможные остановки: полная остановка конвейера. Данные передаваемые на следующую стадию описаны в таблице 3;
5. B (Write Back) - стадия записи данных в регистровый файл. На данной стадии происходит формирование записи в регистровый файл. Кроме того может сформироваться сигнал полной остановки конвейере при отсутствии данных из кэш-памяти данных.

Таблица 1. Выходных данных со стадии DEC

|  |  |  |
| --- | --- | --- |
| **Имя** | **Разрядность** | **Описание** |
| Шина l1d\_bus | | |
| l1d\_ld\_or\_st\_dec | [0:0] | Распределения типа обращения в память(load or store) |
| l1d\_val\_dec | [0:0] | Значимость данных для кэш-памяти данных |
| l1d\_size\_dec | [2:0] | Размер запроса в кэш-память данных |

|  |  |  |
| --- | --- | --- |
| **Имя** | **Разрядность** | **Описание** |
| Шина mux\_bus | | |
| is\_imm\_dec | [0:0] | Управляющий сигнал для мультиплексора, выбирающего между imm или src1(src\_a) |
| is\_pc\_dec | [0:0] | Управляющий сигнал для мультиплексора, выбирающего между pc или scr2(srcb) |
| is\_src1\_dec | [0:0] | Управляющий сигнал для мультиплексора, выбирающего между pc + 4 или src1(srca) |
| is\_otw\_pc\_dec | [0:0] | Управляющий сигнал для мультиплексора, выбирающего между pc и выбором мультиплексора is\_src1(Mux 4 на схеме) |
| is\_mem\_dec | [0:0] | Управляющий сигнал для мультиплексора, выбирающего между результатом с памяти и с ALU |

|  |  |  |
| --- | --- | --- |
| **Имя** | **Разрядность** | **Описание** |
| Шина передачи данных hazard\_ctrl (hazard\_bus\_dec) | | |
| rs1\_dec | [4:0] | Адрес регистра источника 1 |
| rs2\_dec | [4:0] | Адрес регистра источника 2 |
| rd\_dec | [4:0] | Адрес регистра назначения |

|  |  |  |
| --- | --- | --- |
| **Имя** | **Разрядность** | **Описание** |
| sx\_imm\_dec | [31:0] | Операнд принятый из кода команды со знаковым расширением |
| srca\_dec | [31:0] | Операнд из регистрового файла |
| srcb\_dec | [31:0] | Операнд из регистрового файла |
| pc\_dec | [31:0] | Текущее значение программного счетчика |
| pc \_4\_dec | [31:0] | Текущее значение программного счетчика + 4 |
| brnch\_cnd\_dec | [1:0] | Условия ветвления |
| we\_reg\_file\_dec | [0:0] | Разрешение записи в регистровый файл |
| alu\_op\_dec | [3:0] | Код операции для ALU |

Таблица 2. Выходные данных со стадии EX

|  |  |  |
| --- | --- | --- |
| **Имя** | **Разрядность** | **Описание** |
| brnch\_taken\_ex | [0:0] | Результат вычислений условия ветвления |
| alu\_result\_ex | [31:0] | Результат вычисления ALU |
| sx\_imm\_ex | [31:0] | Операнд с кода команды |
| w\_data\_ex(src2) | [31:0] | Данные для записи в память |
| addr\_ex | [31:0] | Адрес перехода |
| pc \_4 | [31:0] | Текущее значение программного счетчика + 4 |
| we\_reg\_file\_dec | [0:0] | Разрешение записи в регистровый файл |
| hz\_bus | [16:0] | Включает в себя значения регистров источников(source) и регистра назначения(destination) для регистрового файла |
| rs1\_ex | [4:0] | Адрес регистра источника 1 |
| rs2\_ex | [4:0] | Адрес регистра источника 2 |
| rd\_ex | [4:0] | Адрес регистра назначения |
| mux\_bus[5] | [0:0] | Значение сигнала управления для 6 мультиплексора(распределяет поток с памяти и с alu для записи в reg\_file) |

Таблица 3. Выходные данные со стадии MEM

|  |  |  |
| --- | --- | --- |
| **Имя** | **Разрядность** | **Описание** |
| alu\_resul\_mem(or address for dl1) | [31:0] | Результат вычисления ALU |
| sx\_imm\_mem | [31:0] | Операнд с кода команды |
| hz\_bus\_mem | [16:0] | Шина передачи данных hazard\_ctrl(описание в таблице выше) |
| pc\_4 | [31:0] | Текущее значение программного счетчика + 4 |
| we\_reg\_file\_dec | [0:0] | Разрешение записи в регистровый файл |

## Описание конфликтов конвейера и модуля hazard control

Модуль hazard control содержит 7 выходов и 14 входов, основная задача решение конфликтов конвейера. Ниже приведена таблица входов и выходов hazard control

|  |  |  |
| --- | --- | --- |
| Имя | Направление и назрядность | Назначение |
| Rst\_n | Input[0:0] | Системный сброс используется как сигнал сброса всех регистров. |
| Haz\_enb\_bus\_out | Output[3:0] | Шина сигнала разрешения работы ко всем регистрам |
| Haz\_kill\_bus\_out | Output[3:0] | Шина очистки регистров |
| Haz\_pc\_stop\_out | Output [0:0] | Сигнал остановки счета pc |
| Haz\_nop\_gen\_out | Output[0:0] | Генератор nop команды стадии decode |
| Haz\_mux\_trn\_out | Output[0:0] | Сигнал управления мультиплексором transfer(addr or next\_pc) |
| Haz\_bp\_mux\_exe\_out | Output[3:0] | Шина select сигнала bp мультиплексоров exe стадии |
| Haz\_bp\_mux\_mem\_out | Output[0:0] | Шина select сигнала bp мультиплексоров mem стадии |
| Haz\_bus\_exe\_s\_in | Input[14:0] | Шина передачи адреса rs1 rs2 rd со стадии exe в haz\_ctrl |
| Haz\_bus\_mem\_s\_in | Input[14:0] | Шина передачи адреса rs1 rs2 rd со стадии mem в haz\_ctrl |
| Haz\_rd\_wb\_s\_in | Input[4:0] | Шина передачи адреса rd со стадии wb в haz\_ctrl |
| Haz\_we\_reg\_file\_exe\_s\_in | Input[0:0] | Сигнал разрешения записи в reg\_file со стадии exe |
| Haz\_we\_reg\_file\_mem\_s\_in | Input[0:0] | Сигнал разрешения записи в reg\_file со стадии mem |
| Haz\_we\_reg\_file\_wb\_s\_in | Input[0:0] | Сигнал разрешения записи в reg\_file со стадии wb |
| Haz\_brnch\_tknn\_in | Input[0:0] | Сигнал с модуля alu о взятие ветвления |
| Haz\_stall\_dec\_in | Input[0:0] | Сигнал stall cо стадии dec(случай отсутствий ack c instruction cash) |
| Haz\_stall\_wb\_in | Input[0:0] | Сигнал stall cо стадии dec(случай отсутствий ack c data cash) |
| Haz\_cmd\_dec\_s\_in | Input[1:0] | Сигнал типа команды со стадии dec |
| Haz\_cmd\_exe\_s\_in | Input[1:0] | Сигнал типа команды со стадии exe |
| Haz\_cmd\_mem\_s\_in | Input[1:0] | Сигнал типа команды со стадии mem |
| Haz\_cmd\_wb\_s\_in | Input[1:0] | Сигнал типа команды со стадии mem |

**Возможные конфликты**

**1)**read after write

Не готовность операнда в силу стадийности операции. Решение – передача(forwarding) данных со следующих стадий. Осуществляется проверка равности операндов адреса источника и адреса назначения для каждого из входов АЛУ. Приоритет отдается данным со стадии mem так они более актуальны.

if((rd\_mem\_loc == rs1\_exe\_loc)&&(rs1\_exe\_loc !=5'b0)&&(haz\_we\_reg\_file\_mem\_s\_in)) hazard\_exe\_bp\_loc = `M2E\_SRC1\_BP;

if((rd\_mem\_loc == rs2\_exe\_loc)&&(rs2\_mem\_loc !=5'b0)&&(haz\_we\_reg\_file\_wb\_s\_in))hazard\_exe\_bp\_loc = `M2E\_SRC2\_BP;

if((haz\_rd\_wb\_s\_in == rs1\_exe\_loc)&&(rs1\_exe\_loc !=5'b0)&&(haz\_we\_reg\_file\_wb\_s\_in)) hazard\_exe\_bp\_loc = `W2E\_SRC1\_BP;

if((haz\_rd\_wb\_s\_in == rs2\_exe\_loc)&&(rs2\_exe\_loc !=5'b0)&&(haz\_we\_reg\_file\_wb\_s\_in)) hazard\_exe\_bp\_loc = `W2E\_SRC2\_BP;

if((haz\_rd\_wb\_s\_in == rs2\_mem\_loc)&&(rs2\_mem\_loc !=5'b0)&&(haz\_we\_reg\_file\_wb\_s\_in)) hazard\_mem\_bp\_loc = `W2M\_BP\_ON;

end

**2) не определенные по времени ответы (ожидание ответа с cash)**

Реакция на stall со стадии decod, который является результатом отсутствия ответа с instruction cash – остановка (enb\_off) регистра if/dec и, как следствие, остановка pc

if(haz\_stall\_dec\_in)begin

haz\_enb\_bus\_loc[`REG\_IF\_DEC] = `REG\_ENB\_OFF;

haz\_nop\_gen\_loc = `NOP\_GEN\_ON;

end

assign haz\_pc\_stop\_out = (haz\_enb\_bus\_loc == `ENB\_FULL\_ON)?1'b0:1'b1;

Реакция на stall со стадии wb, который является результатом отсутствия ответа с data cash – полная остановка конвейера и pc

if(haz\_stall\_wb\_in) begin

haz\_enb\_bus\_loc = `ENB\_FULL\_OFF;

end

**3)transfer control instruction(branch or jump)**

Реакция на branch – если ветвление не произошло то работа конвейера на останавливается. Если произошло ветвление то происходит очистка регистров if/dec dec/exe.

if(haz\_cmd\_exe\_s\_in == `HZRD\_BRNCH)begin

if(haz\_brnch\_tknn\_in == 1'b1) begin

mux\_1\_loc = 1'b1;

haz\_kill\_bus\_loc = `KILL\_BRNCH;

end

else begin

mux\_1\_loc =1'b0;

haz\_kill\_bus\_loc = `KILL\_FULL\_OFF;

end

end

реакция на команду jump – со стадии decode приходит сигнал о команде типа jm, hazard control посылает сигнал об остановке if/dec регистра и его сброса(они посылаются одновременно). Далее на стадии exe формируется сигнал mux\_trn и передается в регистр exe/mem. В это же время посылается сингал nop\_gen и выставляется enb\_on на регистр if/dec. Когда команда jump находится на стадии mem генерируется nop\_gen, все регистры работают.

if(haz\_cmd\_dec\_s\_in == `HZRD\_JMP)begin

haz\_kill\_bus\_loc[`REG\_IF\_DEC] = `REG\_KILL\_ON;

haz\_enb\_bus\_loc[`REG\_IF\_DEC] = `REG\_ENB\_OFF;

end

if(haz\_cmd\_exe\_s\_in == `HZRD\_JMP)begin

haz\_nop\_gen\_loc = `NOP\_GEN\_ON;

mux\_1\_loc = 1'b1;

end

if(haz\_cmd\_mem\_s\_in == `HZRD\_JMP)begin

haz\_enb\_bus\_loc[`REG\_IF\_DEC] = `REG\_ENB\_ON;

haz\_nop\_gen\_loc = `NOP\_GEN\_ON;

end

## Взаимодействие с подсистемой памяти

Взаимодействие с подсистемой памяти реализовано согласно протоколу, представленному на рисунке 4.

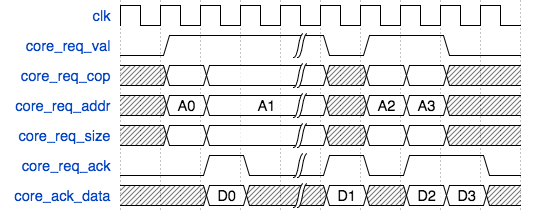


Рисунок 4. Протокол взаимодействия ядра с подсистемой памяти

Ядро выставляет core\_req\_val для подтверждения значимости запроса на шине и вместе с этим код операции core\_req\_cop, адрес обращения core\_req\_addr и размер запроса core\_req\_size. Со стороны подсистемы памяти выставляется core\_req\_ack для подтверждения значимости данных и вместе с этим данные core\_ack\_data.

## ****Взаимодействие с кэш-памятью инструкций****

## 

Рисунок 5.

**Таблица 4. Интерфейс взаимодействия ядра с кэш-памятью инструкций**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Разрядность** | **Направление** | **Описание** |
| **i\_req\_val** | **[0:0]** | **Output** | **Значимость запроса** |
| **i\_req\_addr** | **[31:0]** | **Output** | **Адрес запроса** |
| **i\_req\_ack** | **[0:0]** | **Input** | **Значимость ответа** |
| **i\_ack\_rdata** | **[31:0]** | **Input** | **Данные ответа** |

## ****Взаимодействие с кэш-памятью данных****

На стадии М формируется запрос в память. Адрес запроса берётся с регистра конвейера EX\_M\_R. Инструкция переходит на следующую стадию(WB). На стадии WB из памяти должен вернуться ответ, если он есть то используются полученные данные и конвейер работает в штатном режиме. Если нет, то формируется полная остановка конвейера.

Следует заметить, что когда первая инструкция (I1) переходит со стадии M на стадию WB, на стадию M приходит следующая инструкция (I2). Если это инструкция работы с памятью, то так же выставляется запрос в память. При формирования остановки конвейера, запрос в память от инструкции продолжает висеть.

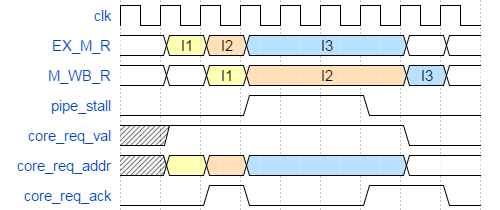


Рисунок 6.

При обращение в память ядро сообщает подсистеме памяти какой тип запроса: кэшируемое обращение или некэшируемой. В управляющих регистрах ядра хранится информация об окне некэшируемых адресов

**Таблица 5. Интерфейс взаимодействия ядра с кэш-памятью данных**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Разрядность** | **Направление** | **Описание** |
| **d\_req\_val** | **[0:0]** | **Output** | **Значимость запроса** |
| **d\_req\_addr** | **[31:0]** | **Output** | **Адрес запроса** |
| **d\_req\_cop** | **[2:0]** | **Output** | **Типа запроса:**  **3’b000 – RD**  **3’b001 – WR**  **3’b010 – RDNC**  **3’b011 - WRNC**  **Остальные – зарезервировано** |
| **d\_req\_wdata** | **[31:0]** | **Output** | **Данные для записи** |
| **d\_req\_size** | **[2:0]** | **Output** | **Размер запроса (1, 2 или 4 байта)** |
| **d\_req\_ack** | **[0:0]** | **Input** | **Значимость ответа** |
| **d\_ack\_rdata** | **[31:0]** | **Input** | **Данные ответа** |

## Кэш-память первого уровня

Характеристики кэш-памяти инструкций:

* Ассоциативность – 4 канала;
* Объём – 32 KB;
* Длина строки (размер блока данных) – 32 B;
* Общее количество строк – 32 KB / 32B = 1K;
* Алгоритм вытеснения: LRUm;
* Write-through, No-Write-Allocate.

Общая формула расчёт размера выглядит следующим образом: , где, *N* – объём кэш-памяти, *l* – длина кэш-строки, *m* – ассоциативность, *s* – количество сетов, *b* – количество банков.

При *N* = 32 KB, *m* = 4, *l* = 32 B и *b* = 1, s получается равным 256.

При такой структуре памяти, физический адрес состоит из 3 элементов:

* tag – тэг адреса, хранится вместе с кэш-строкой;
* idx – номер сета;
* offset – смещение внутри кэш-строки.

Для предлагаемых параметров и длине физического адреса 32 бита получаем:

* tag = 19 бит;
* idx = 8 бит;
* offset = 5 бита.

Таблица 1. Разбиение физического адреса в L1

|  |  |  |
| --- | --- | --- |
| Тэг | Индекс | Номер байта |
| [31:13] | [12:5] | [4:0] |

Таблица 2. Внутренние проверки

|  |  |
| --- | --- |
| Свойство | Описание |
| offset\_allign\_p |  |
| tag\_cmp\_with\_val\_vect\_p | one hot |
| lru\_way\_vect\_p | one hot |

# ****Периферийный кластер****

## ****SPI****

## Программная модель

# Прототип