|  |  |  |
| --- | --- | --- |
| Лабораторная работа №2 | M3136 | 2022 |
| Моделирование схем в Verilog | Панюхин Никита Константинович | |
|

**Цель работы:** построение кэша и моделирование системы “процессор-кэш-память” на языке описания Verilog.

**Инструментарий и требования к работе:**используемый язык: SystemVerilog, компиляция и симуляция – Icarus Verilog 12

**Описание**

<Пишем условие работы TODO Формулировка задачи из условия.>

**Вариант**: 1

# Вычисление недостающих параметров системы

Параметры из условия для варианта №1:

* Ассоциативность: **CACHE\_WAY = 2**
* Размер тэга адреса: **CACHE\_TAG\_SIZE = 10 бит**
* Размер кэш-линии (полезных данных): **CACHE\_LINE\_SIZE = 16 байт**
* Кол-во кэш-линий: **CACHE\_LINE\_COUNT = 64**
* Размер памяти: **MEM\_SIZE = 512 Кбайт**

Вычислим остальные параметры:

* Размер кэша:

**CACHE\_SIZE = CACHE\_LINE\_SIZE × CACHE\_LINE\_COUNT = 64 × 16 байт = 1024 байт**

* Кол-во наборов кэш-линий:

**CACHE\_SETS\_COUNT = CACHE\_LINE\_COUNT / CACHE\_WAY = 64 / 2 = 32**

* Размер индекса в наборе кэш-линий:

**CACHE\_SET\_SIZE = = = 5 бит**

* Размер смещения**:**

**CACHE\_OFFSET\_SIZE = = = 4 бит**

* Размер адреса**:**

**CACHE\_ADDR\_SIZE = = = 19 бит**

Размер частей адреса кэша:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **tag** | **set** | **offset** |
| Размер | **CACHE\_TAG\_SIZE** | **CACHE\_SET\_SIZE** | **CACHE\_OFFSET\_SIZE** |
| **10 бит** | **5 бит** | **4 бита** |
| **19 бит** | | |

Таблица №1 – Интерпретация адреса кэшем

Отсюда получим размеры шин A1 и A2:

* **ADDR1\_BUS\_SIZE = =  
  = = 15 бит**
* **ADDR2\_BUS\_SIZE = CACHE\_TAG\_SIZE + CACHE\_SET\_SIZE = 15 бит**

Размеры шин C1 и C2, очевидно, равны логарифму от максимального значения сигнала на них, то есть и соответственно.

Все параметры системы указаны в файле [src/parameters.sv](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/src/parameters.sv)

# Аналитическое решение задачи

Аналитическое решение было выполнено с помощью кода на языке Python (находится в файле [“Analytical solution/solution.py”](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/Analytical%20solution/solution.py)).

Поскольку в задаче требуется лишь посчитать количество кэш-попаданий и суммарное число тактов работы программы, можно смоделировать лишь часть процессов, достаточную для получения ответа. Так, например, можно опустить все операции с памятью и значениями переменных, поскольку, записанные в память (Mem) не влияют на оба требуемых ответа. Также, поскольку в задаче нет вызовов операции read32, упрощается операция read\_bytes – байты данных передаются одновременно в такт с командой C1\_RESPONSE, и считать дополнительный такт не надо[[➜]](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/Analytical%20solution/solution.py#L94).

В остальном, программа на питоне полностью повторяет программу на Verilog, смысл большинства задержек подписан в коде. В некоторых местах требуется подождать изменения значения синхронизации (например, команда отправляется по спаду CLK и принимается по фронту CLK, см. пункт Соглашения), для этого используется функция Cache.wait\_clk

В результате получаем следующие ответы:

|  |
| --- |
| Total time: 4613397.0 tacts  Cache hits: 228080/249600 = 91.38% |

# Моделирование заданной системы на Verilog

Поскольку система в чём-то вариативна, нужно описать некоторые соглашения. Опишу пунктами

При моделировании были использованы следующие соглашения:

Далее соглашения будут помечены значком « »

**Общение по шине:**

* Данные на шине меняются при CLK = 0 (по спаду, negedge), а принимаются на другой стороне при CLK = 1 (по фронту, posedge)

// Примеры:

// байт(bbytes\_start + bbyte) -> D2[pos\_left:pos\_right]

// 0 + 0 -> [7:0]

// 0 + 1 -> [15:8]

// 1 + 0 -> [7:0]

// 1 + 1 -> [15:8]

// 2 + 0 -> [7:0]

// 2 + 1 -> [15:8]

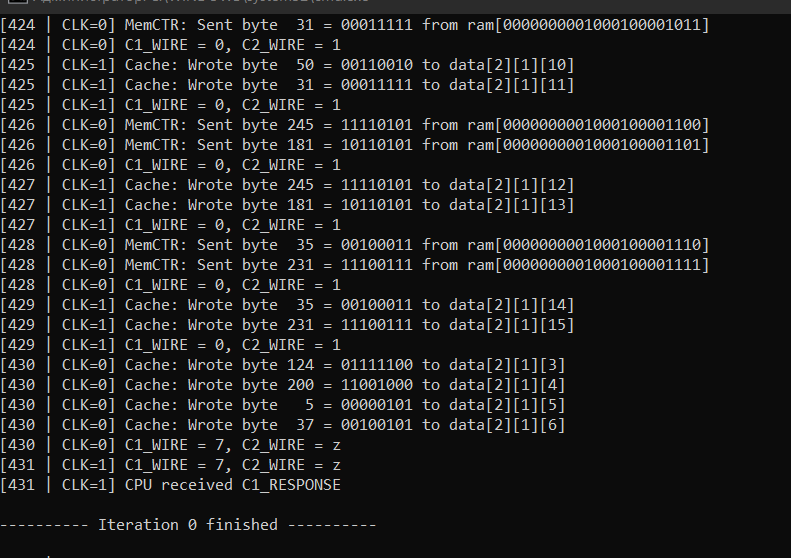
// Тогда: байт [bbytes\_start + bbyte] нужно записать в D2[bbyte \* 8 + 7:bbyte \* 8]

// Пример: отправялем два байта reg byte1 = 236 (11101100); reg byte2 = 122 (01111010)

// Тогда отправится reg D2 = 01111010|11101100 - конкатенация двух байтов

NOP ставиться по спаду CLK, когда он равен 0

CPU встроен в testbench

После read\_line\_from\_Mem мы ждём лишний такт. Можно без него.

Переход на следующую итерацию цикла +#2 лишних оставим

# Воспроизведение задачи на Verilog

# Сравнение полученных результатов

# Листинг кода