**Евтушенко Олег, группа P41193**

**Lab SCR1 pipeline**

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Команда | Arch #1 | Arch #2 |
| 1 | JAL | RVIM | RVIMC |

**1 Часть**

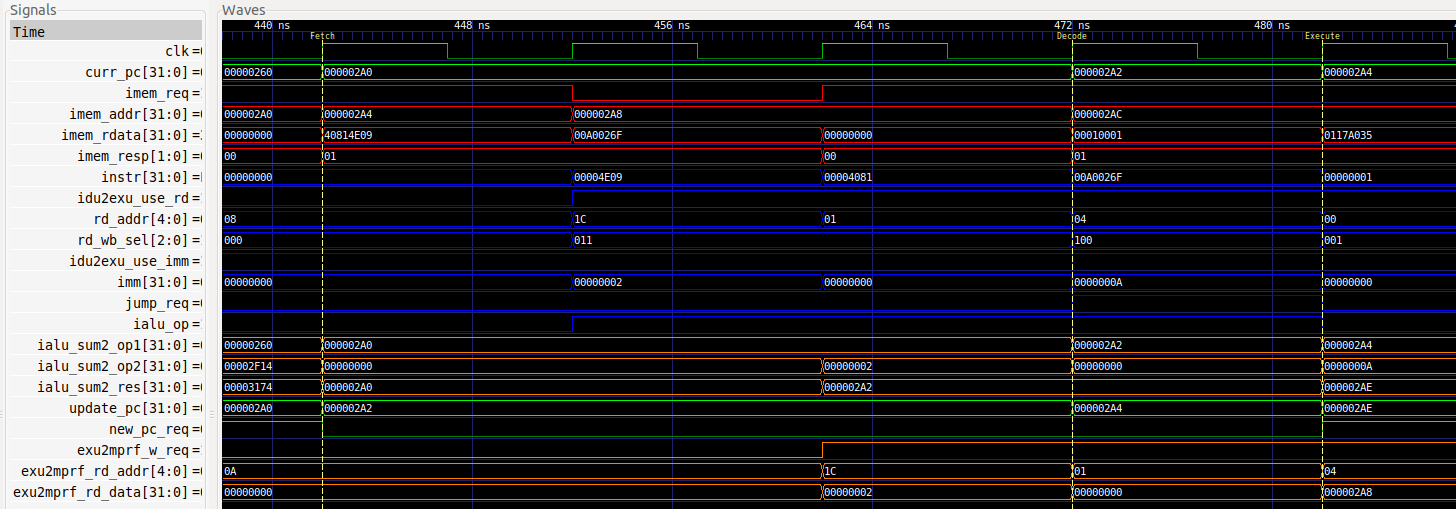
Команда JAL – это команда безусловного перехода, которая записывает в регистр rd = PC +4 и обновляет значение PC = PC + offset.

Для разборки был выбран файл с тестом «jal.S».

Отрывок из dump-файла, где используется команда JAL:

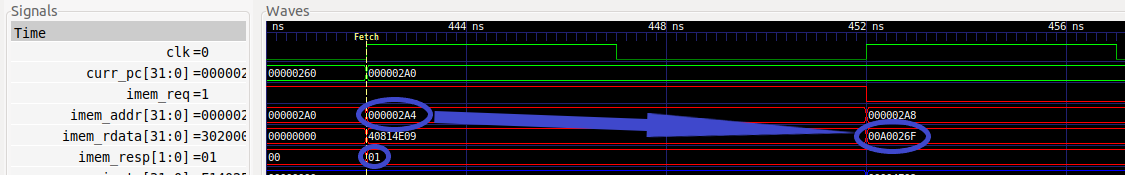
|  |
| --- |
| Jal.dump  …  Дизассемблирование раздела .text:  000002a0 <\_run\_test>:  2a0: 4e09 li t3,2  2a2: 4081 li ra,0  2a4: 00a0026f jal tp,2ae <target\_2>  000002a8 <linkaddr\_2>:  2a8: 0001 nop  2aa: 0001 nop  2ac: a035 j 2d8 <fail>  … |

Результирующая вейвформа:



Описание сигналов:

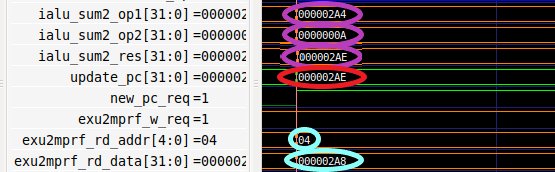
* clk - тактовый импульс
* curr\_pc - текущее значение счетчика команд, соответствует стадии Execution (=000002A4)
* набор сигналов для Instruction Fetch:
  + imem\_req - запрос от процессора в память инструкций (=1)
  + imem\_addr - адрес запроса памяти инструкций (=000002A4)
  + imem\_resp - ответ памяти инструкций (=01)
  + imem\_rdata - данные чтения памяти инструкций (=00A0026F)



* набор сигналов для Decode:
  + instr – закодированная инструкция (=00A0026F)
  + idu2exu\_use\_rd – сигнал о том, что в команде есть RD (=1)
  + rd\_addr – номер регистра (=04), x4 = tp
  + rd\_wb\_sel – код регистра (100)
  + idu2exu\_use\_imm – сигнал о том, что в команде есть immediate (=1)
  + imm – значение immediate (=0000000A)
  + jump\_req – сигнал о том, что это команда перехода (=1)
  + ialu\_op – сигнал о том, что будет использована АЛУ (=1)



* Набор сигналов для Execute:
  + ialu\_sum2\_op1 – первый операнд АЛУ (=000002A4)
  + ialu\_sum2\_op2 – второй операнд АЛУ (=0000000A)
  + ialu\_sum2\_res – результат АЛУ (=000002AE)
  + new\_pc\_req – сигнал о том, что PC будет перезаписан (=1)
  + update\_pc – перезапись PC (=000002AE)
  + exu2mprf\_w\_req – сигнал записи значения в регистровый файл
  + exu2mprf\_rd\_addr – адрес регистра для записи в регистровый файл (=04), x4 = tp
  + exu2mprf\_rd\_data – значение для записи в регистровый файл (=000002A8).



**2 Часть**

Были запущены два теста-бенчмарка Coremark и Dhrystone для двух архитектур по заданию: RVIM и RVMIC. Для этого был изменён файл «scr1\_arch\_description.svh»:

`define SCR1\_RVM\_EXT \\ Для добавления M архитектуры

`define SCR1\_RVC\_EXT \\ Для добавления С архитектуры

И при запуске указывалась IM / IMC архитектура

make run\_verilator\_wf BUS=AHB ARCH=IM/ IMC IPIC=0

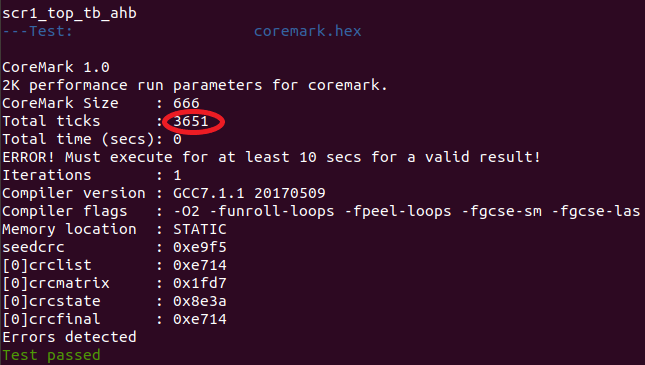
Были получены следующее результаты:

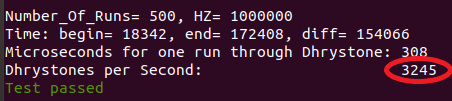
|  |  |  |
| --- | --- | --- |
| Тест | RVIM | RVIMC |
| Coremark (Total ticks) | 3533 Total ticks | 3651 Total ticks |
| Coremark (size memory) | 89,9 Кбайт | 75,5 Кбайт |
| Dhrystone (per Seconds) | 3288 per Seconds | 3245 per Seconds |
| Dhrystone (size memory) | 45,6 Кбайт | 41,6 Кбайт |

При выборе разных архитектур, для теста-бенчмарка Dhrystone, количества итераций в секунду (изменилось на 43) и размер занимаемой памяти (на 4 Кбайта) изменяется в небольших размерах. А для теста-бенчмарка Coremark количество итераций в секунду изменяется на 118, при это размер занимаемой памяти увеличивается на 14,4 Кбайт.

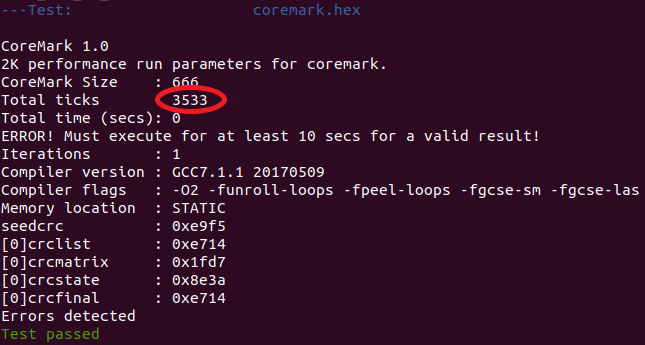
Из полученных результатов можно сделать вывод, что тест-бенчмарка Coremark занимает почти в два раза больше памяти, чем тест Dhrystone, но при этом количество итераций в секунду выдаёт выше, по сравнению с Dhrystone.

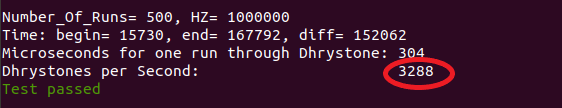
Скриншоты тест-бенчмарков Coremark и Dhrystone для ARCH = RVIMC





Скриншоты тест-бенчмарков Coremark и Dhrystone для ARCH = RVIM



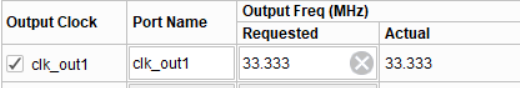


**3 часть**

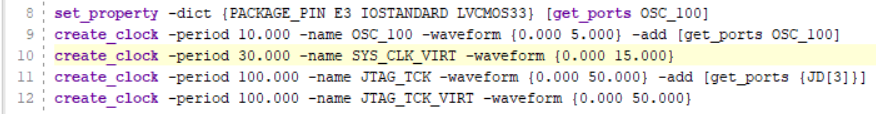
|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Arch #1 | Arch #2 | Конфигурация |
| 1 | RVIM | RVIMC | IMC\_MAX |

**IMC\_MAX**

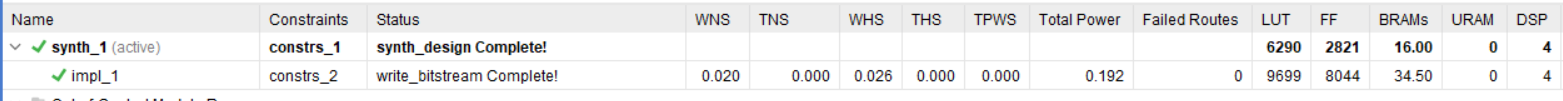
Для проверки максимальной частоты установил частоту 33.333Mhz в sys\_pll.bd в Properties:



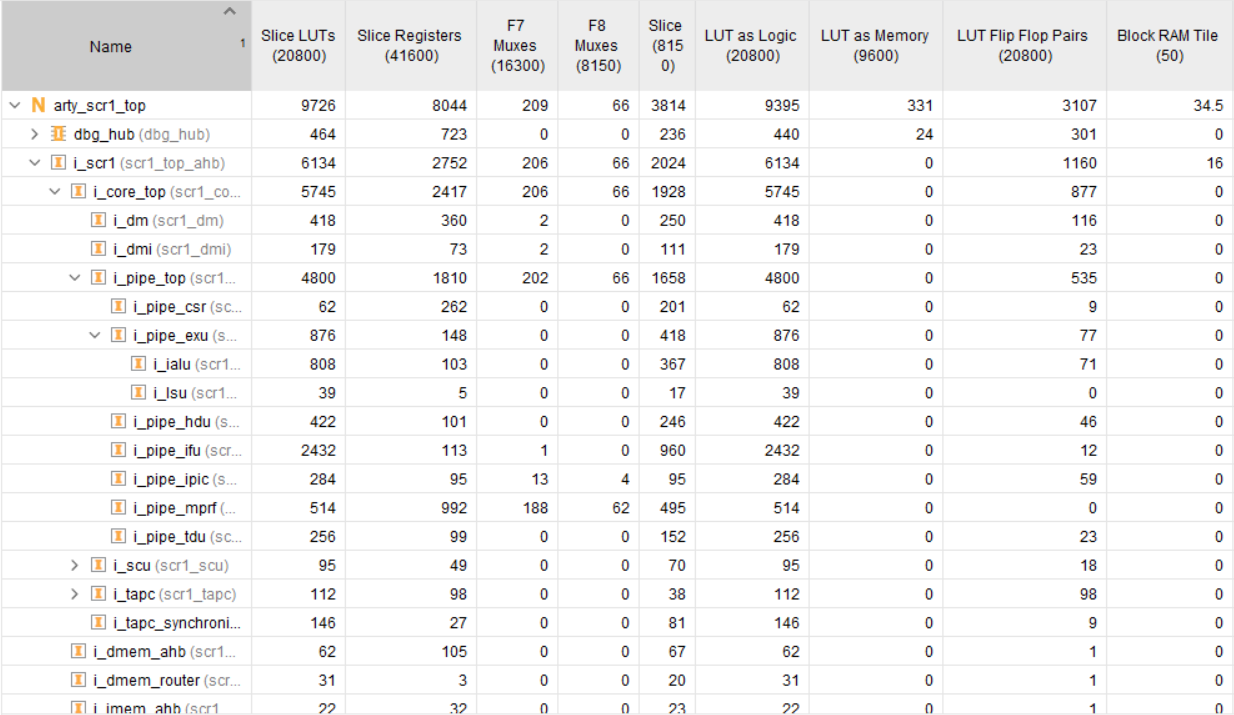
А также период 30ns для SYS\_CLK\_VIRT:



Получил WNS = 0.020ns, следовательно Fmax = 33.333Mhz

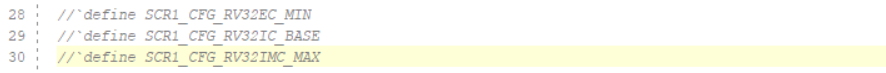


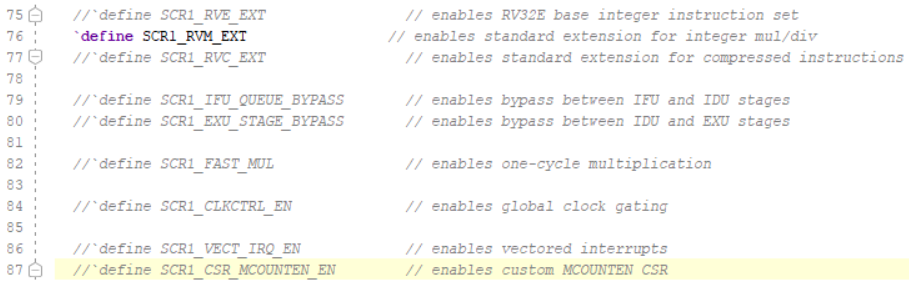
Модуль, который занимает больше всего места на кристалле – i\_pipe\_ifu = 2432 LUT



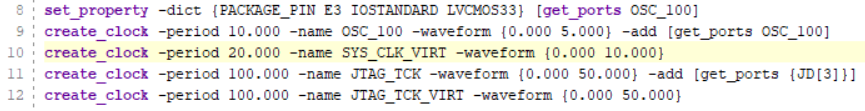
**RVIM**

Для выбора RVIM архитектуры изменил файл «scr1\_arch\_description.svh» (отключил RV32IMC\_MAX и выбрал RVIM):

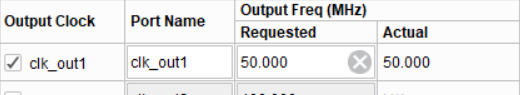




Предположил, что Fmax = 50Mhz и установил, соответственно, период SYS\_CLK\_VIRT = 20ns:



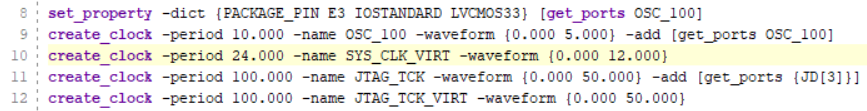
А также установил частоту 50Mhz в sys\_pll.bd в Properties.



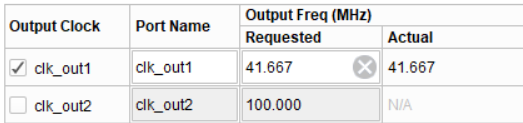
Получил WNS = -4.341 ns



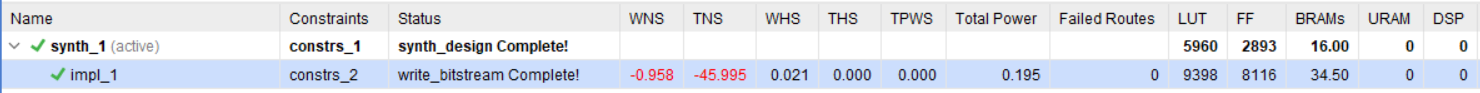
Так как WNS отрицательный, можно сделать вывод, что запаса по времени удержания и установки не хватает приблизительно на 4 ns, следовательно период нужно увеличить на 4 ns (SYS\_CLK\_VIRT = 24ns). Исходя из этого, F = \*1000 = 41.667 Mhz.



А также установил частоту 41.667 Mhz в sys\_pll.bd в Properties



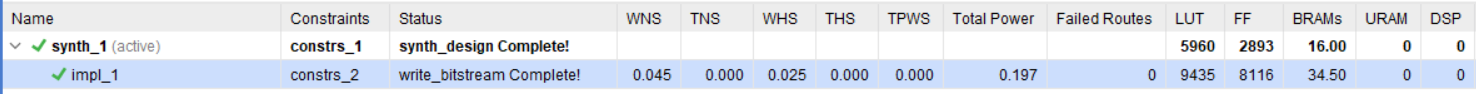
Получил WNS = -0.958 ns



Устранить небольшое отрицательное значение WNS можно с помощью встроенных средств Vivado, а именно включить strategy Perfomence\_ExtraTimingOpt + phys\_opt\_design.



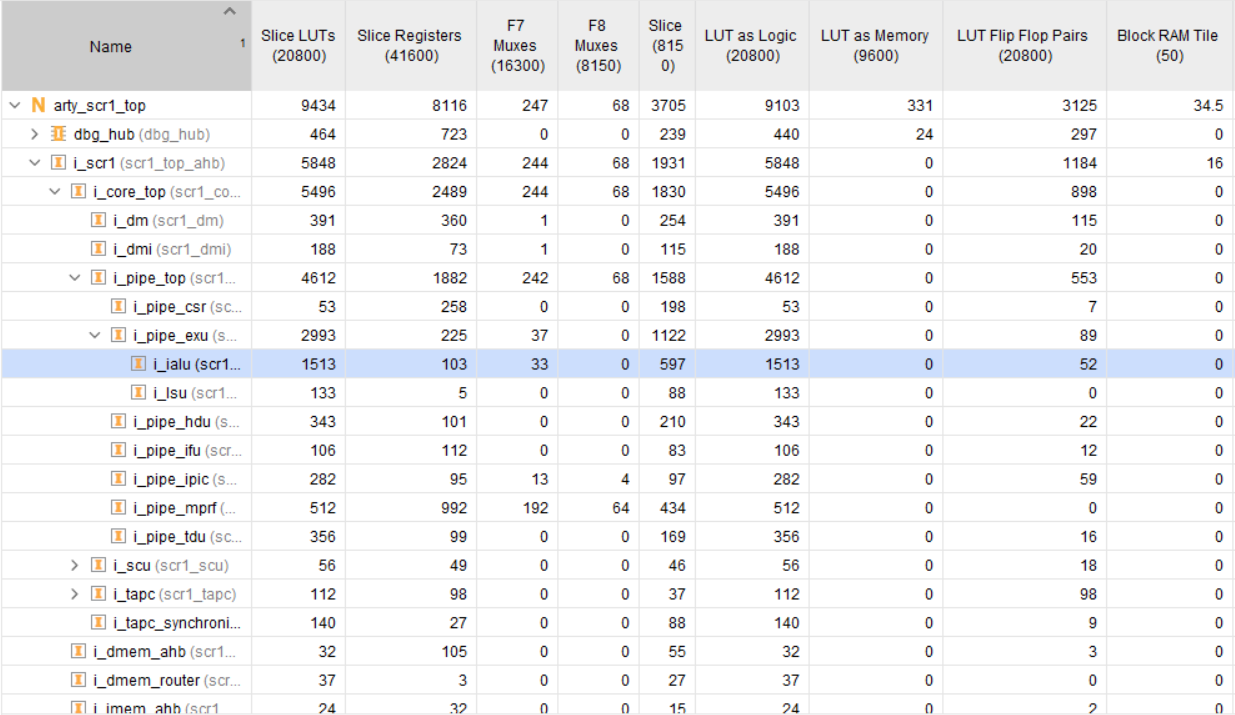
Результаты STA получились следующие:



Получил WNS = 0.045.

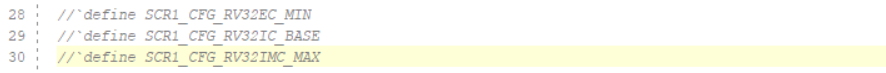
Максимальная частота (Fmax) для **RVIM**= **41.667** **Mhz**.

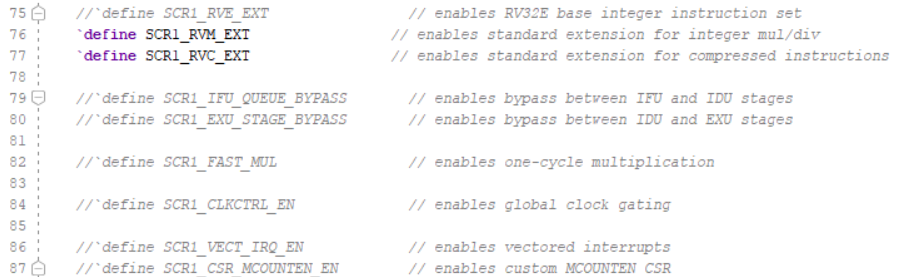
Модуль, который занимает больше всего места на кристалле – i\_pipe\_exu = 2993 LUT. При этом i\_alu = 1513 LUT.



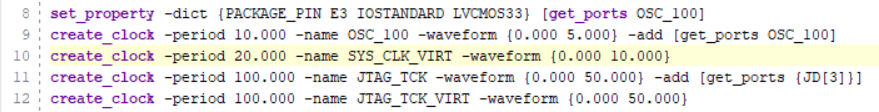
**RVIMC**

Для выбора RVIM архитектуры изменил файл «scr1\_arch\_description.svh» (отключил RV32IMC\_MAX и выбрал RVIM):

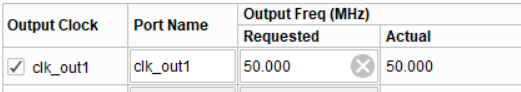




Предположил, что Fmax = 50Mhz и установил, соответственно, период SYS\_CLK\_VIRT = 20ns:



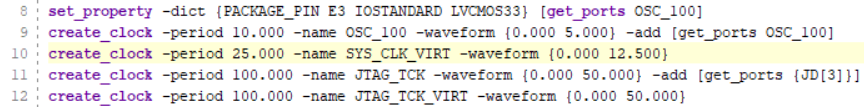
А также установил частоту 50Mhz в sys\_pll.bd в Properties.



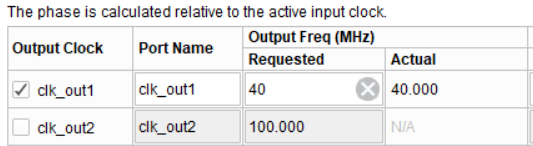
Получил WNS = -5.167 ns



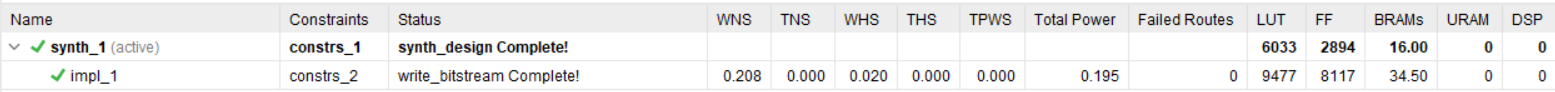
Так как WNS отрицательный, можно сделать вывод, что запаса по времени удержания и установки не хватает приблизительно на 5 ns, следовательно период нужно увеличить на 5 ns (SYS\_CLK\_VIRT = 25ns).

****

Исходя из этого, F = \*1000 = 40 Mhz.

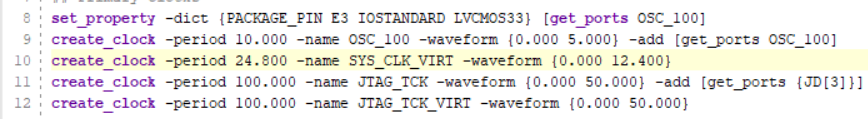
****

Получил WNS = 0.208 ns

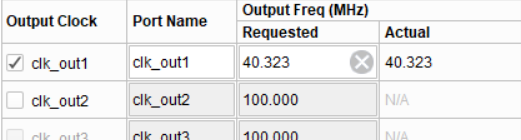


Так как WNS положительный, можно сделать вывод, что запаса по времени удержания и установки хватает.

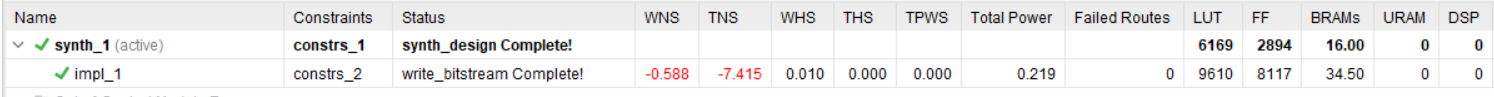
Следовательно, можно уменьшить период на размер запаса (0,2 ns).



Исходя из этого, F = \*1000 = 40.323 Mhz.



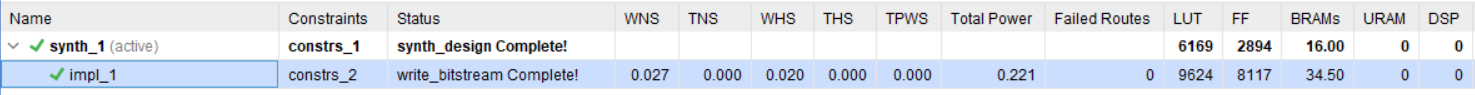
Получил WNS = -0.588 ns



Устранить небольшое отрицательное значение можно с помощью встроенных средств Vivado, а именно включить strategy Perfomence\_ExtraTimingOpt + phys\_opt\_design.



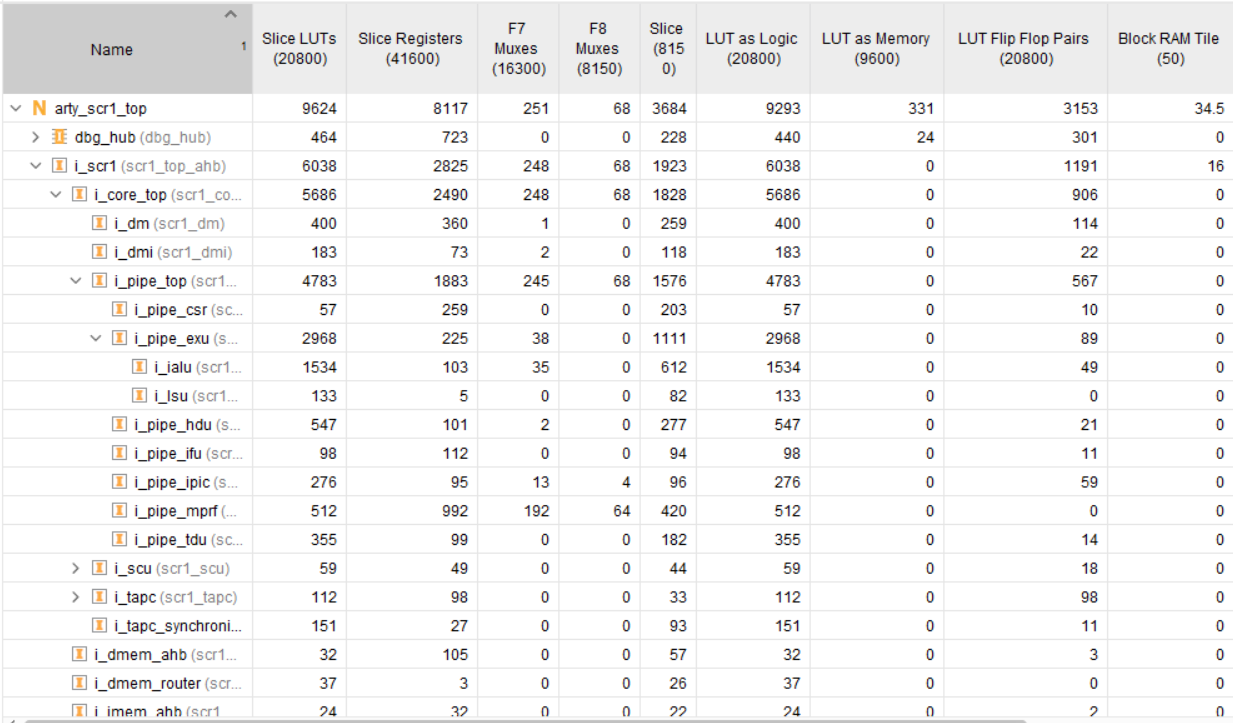
Результаты STA получились следующие:



Получил WNS = 0.027 ns.

Максимальная частота (Fmax) для **RVIMC** = **40.323 Mhz.**

Модуль, который занимает больше всего места на кристалле – i\_pipe\_exu = 2968 LUT. При этом i\_alu = 1534 LUT.



**Выводы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № |  | IMC\_MAX | RVIM | RVIMC |
| 1 | LUT | 9699 | 9435 | 9624 |
| Flip-Flop | 8044 | 8116 | 8117 |
| i\_pipe\_hdu | 422 LUT | 343 LUT | 547 LUT |
| 2 | Fmax, Mhz | 33.333 | 41.667 | 40.667 |

Предполагаю, что наибольшее различие занимаемой площади между RVIM и RVIMC в модуле **«i\_pipe\_hdu»** (Hart debug unit), потому что, при добавлении архитектуры C добавляется функции отладки для этой архитектуры и соответственно, увеличивается интерфейс модуля отладки, следовательно необходимо больше LUT (Look up table), что соответствует увеличению последовательностной части модуля C. Следовательно, путь из TCM будет длиннее для RVIMC.

Из этого следует, что максимальная частота для RVIMC должна быть меньше, чем для RVIM, так как у RVIM путь меньше (из-за того, что нет C).