Advanced VLSI System Design

Fall 2024

HOMEWORK IV

REPORT

Must do self-checking before submission:

Compress all files described in the problem into one tar

All SystemVerilog files can be compiled under SoC Lab environment

All port declarations comply with I/O port specifications

Organize files according to File Hierarchy Requirement

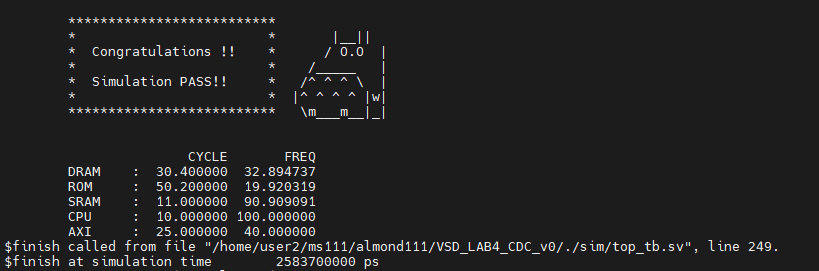
No waveform files in deliverables

Student name: \_\_\_\_王劭懷 \_\_\_\_

Student ID: \_\_\_\_M16121077\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Performance & Area | | | |
|  | rtl | syn | pr |
| Prog0 time | 23489130 | 234891300 | 234944500 |
| Prog1 time | 18041640 | 180416400 | 180471500 |
| Prog2 time | 22031450 | 220314500 | 220411400 |
| Prog3 time | 142649150 | 1426442100 | 1426596000 |
| Area(um^2)  In APR | 3298938.395904 | | |
| CPU cycle | 1.9 | | |

Refer to the figure to fill in the simulation time !!



// Initiate your report from this point.

# Summary

# 在本次的作業我獨自一人成功設計了所有的要求，其中包含兩個分別為Instruction和Data的Cache，並且為2-way set Cache，透過FSM來規劃Hit和Miss的不同處理方式，並且分析Cache 的Hit rate。

# 跨不同時域的設計則使用Asynchronous FIFO來處理不同時間域的資料傳遞，並詳述不同AXI通道中的FIFO運作原理，並以參數化設計實現彈性化資料長度配置。

# 在AXI外部包裝了一層AXI Wrapper，並插入AFIFO，以支援不同頻率的模組之間的互動，保留舊有的AXI設計，對外提供更強大的時鐘處理能力。

# 工具檢查，透過SPYGLASS檢查CDC沒有致命錯誤，僅有少量的警告，並有做出相對應的處理，Superlint的檢查有列出最常見的警告，並解釋這些問題對整體的影響有限。

# 模擬結果在RTL、SYN、APR都有成功跑過所有測資，並顯示了Cache在不同prog下的Hit rate，其中指令快取的Hit Rate普遍較高。

# Design

## Caches (Instruction Cache, Data Cache)

一張含有 圖表, 工程製圖, 方案, 行 的圖片

自動產生的描述

L1C\_inst FSM :

一張含有 圖表, 行, 樣式, 折紙 的圖片

自動產生的描述

在狀態為s\_addr時等待core\_req訊號為1時才進入下個state s\_data，同時在core\_req時將cpu\_iaddr的address向cache發出位址的請求。

在s\_data的狀態時，cache已讀出該位址中的data，並且會比對tag array中的資料與cpu傳來的地址中的cache部分是否相同。而因為我們使用的是2-way set cache，故分成hit0與hit1分別代表是哪一個tag array hit，將其兩個訊號做or則表示為hit訊號，若為hit則透過core\_wait\_CD\_i判斷L1C\_data的cache是否還在執行，如果為0則回到s\_addr的狀態並將instruction回傳至CPU，若core\_wait\_CD\_i為1則在s\_data的狀態中等待。

而若hit為0時，表示cache miss，此時進入s\_miss的狀態，發出req的訊號給CPU wrapper將dram中的值搬移至Cache中，而在每次rvalid\_m0\_i的訊號為1時，則將從CPU\_wrapper讀取進來的值寫入da\_reg的暫存器最低位址中，並將原先的da\_reg左移32個bit。當I\_wait為0時，表示CPU\_wrapper已讀取完最後一個值，但因為其訊號是和rvalid\_m0\_i 與CPU\_wrapper中的RLAST同時，所以要搭配I\_wait 為0且rvalid\_m0\_i為1時，才代表正在回傳最後一個讀取的值，此時進入s\_missdata的狀態，並同時將讀取回來存放在da\_reg的值存入cache的data\_array中。

進到s\_missdata的狀態的用意是因為寫入仍需兩個cycle完成，故多進來這個狀態做等待，如果直接回到s\_addr，則因為instruction通常是連續執行，會立即發出request的訊號，並導致還未成功寫入完成便讀出錯誤的值。而在每一次的miss情況中，都會重新回到s\_addr的狀態，如此一來，再執行一次便會得到hit的結果，而每次hit時才更新CPU的內容，雖然每次miss用這種做法會增加兩個cycle，但是因為instruction cache的hit rate很高，data cache的request不高，所以多出來的消耗算是可接受範圍，且有助於降低電路的複雜性。

L1C\_data FSM :

一張含有 圖表, 寫生, 折紙, 樣式 的圖片

自動產生的描述

Data Cache的狀態機相較於Instruction的複雜一些，因為多了寫入的操作，且因為不可能同時讀同時寫，所以我合併了讀寫的狀態機。

當狀態為s\_addr時與L1C\_inst一樣等待core\_req，且在s\_data的狀態時判斷hit或miss。

與L1C\_inst不同的是，因為data cache多了寫入的操作，所以在s\_data的時候會判斷core\_write是否為1，且CPU wrapper無論Cache有沒有hit都會做寫入，故在s\_write的狀態中只要等待D\_wait和core\_wait\_CI\_i都為0即可回到s\_addr的狀態。而若非core\_write則是做讀取的操作，讀取的狀態與Instruction cache相同，若hit則等待L1C\_inst執行完成回到s\_addr，若miss則進入miss狀態對CPU wrapper要求進行讀取，並待CPU wrapper回傳資料後進入missdata等待1cycle回到s\_addr的狀態。

## FIFO

一張含有 文字, 圖表, 方案, 圖解 的圖片

自動產生的描述

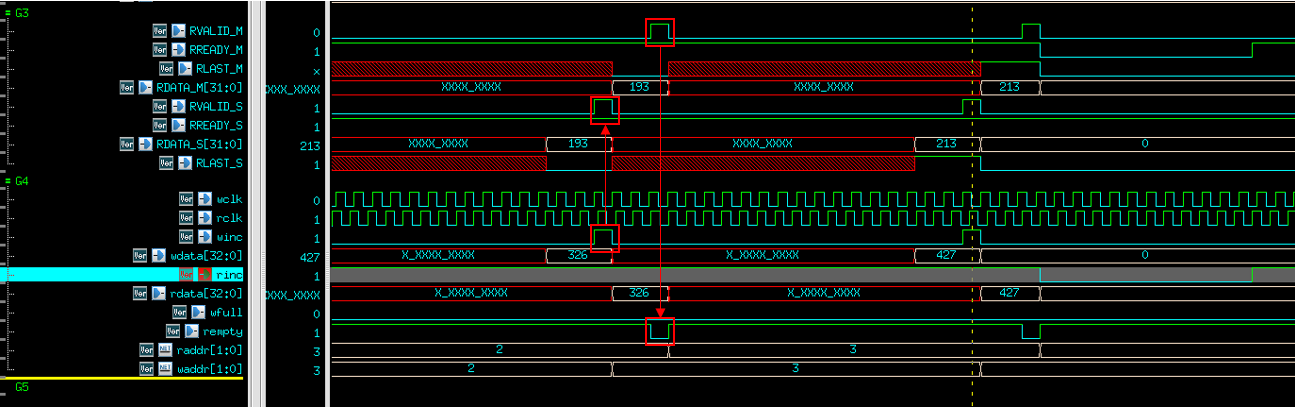
Asynchronous FIFO的電路我是參照Sunburst的論文Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog將電路刻出來，並且在FIFO的資料大小根據不同通道要傳送的資料量不同而透過參數化的方式在呼叫時更改。

以AR 通道為例，將ARVALID 與winc連為一條線，故當Master的ARVALID為1時則將address的資料透過wdata寫入FIFO中，而wfull的反向當作ARREADY，如此一來便可當作是handshake，因為若為full的話ARREADY為0，ARVALID要等待ARREADY訊號將其資料hold住。反之若Full為0，則ARREADY為1，此時可將資料先寫入FIFO，無需等待Slave端的READY訊號。

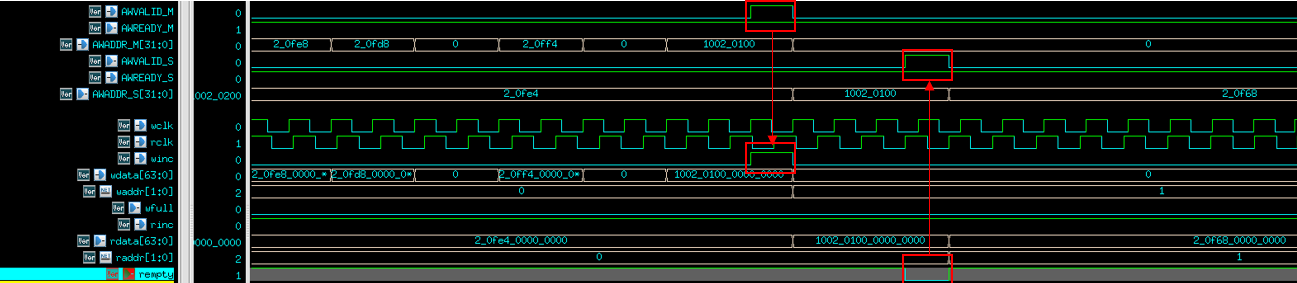
而在Slave端的AR通道，則是將ARREADY與rinc連起，ARVALID與rempty的反向連起，如此一來ARREADY即使一直都為1，表示持續向FIFO發起讀取的request，但是因為FIFO中為empty的所以ARVALID仍為0，故須等待，直到write端寫入完成後，且gray code的pointer成功跨clock domain寫入後，才會將rempty歸0，此時ARVALID才會為1，亦即那個當下才有辦法將FIFO中的資料讀取出來。AR channel 的實際運作方式如下:



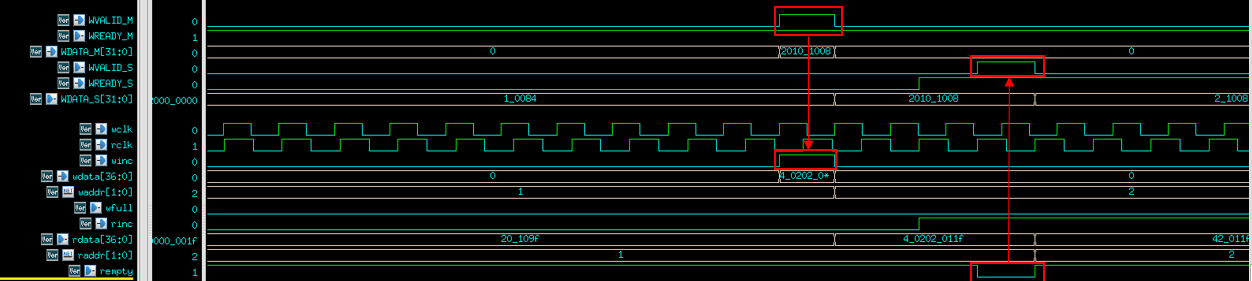
R channel 則與AR channel 相反，因為資料的寫入是從slave端傳回Master端，所以winc連接到的是RVALID\_S，rempty的反向則是連到RVALID\_M，RREADY\_M和RREADY\_S則分別連到rinc與wfull的反向，實際運作情形如下 :



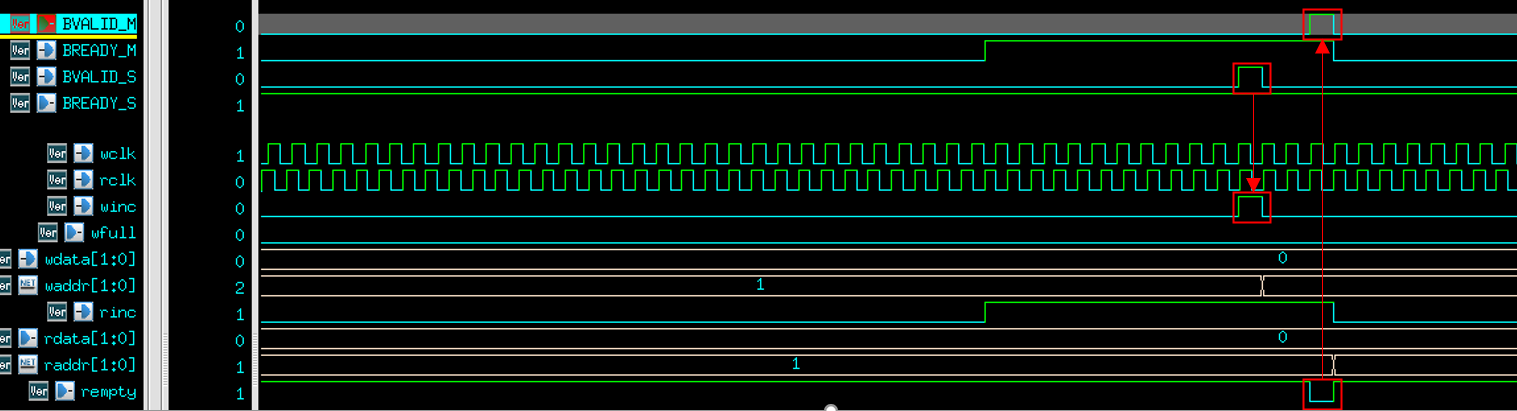
AW channel 與 AR channel 類似，運作方式如下，差別在於必要傳送的資料不同，需寫入FIFO的資料長度便不同，若資料都固定可直接hardwire為該訊號，縮小FIFO的長度大小，藉以減少面積。



W channel與R channel不同，比較像是AW channel，因為寫入也是從master端傳送給slave端，而寫入的資料包含wdata和wlast等等的訊號，因此寫入長度會比單純的wdata還要長，運作方式如下:

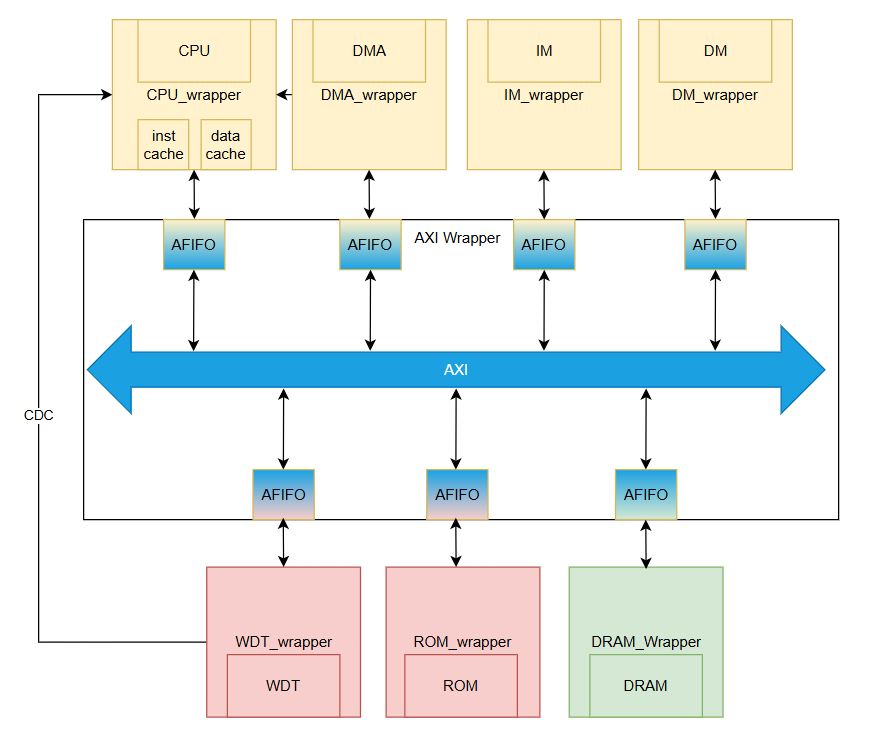


B channel因為只需傳送OKAY的訊號，重點在於valid和ready，所以只要2bit，但因為是自己的設計每次回傳都會是一樣的，原先想說其實可以把資料hardwire，要CDC的訊號只要2 bit flip flop就可，但為了與其他通道保持一致，所以還是做了與其他相同的AFIFO運作方式 如下:



## AXI

為了簡化設計，我對舊有的AXI沒有做任何的改動，而是在最外層包了一層AXI wrapper，在對外的接口都插入了AFIFO來對不同頻率域的Module做處理，僅有WDT的interrupt是透過2D flipflop傳入CPU，AFIFO的工作原理如上一項所述，Block design如下:



# Simulation Results

# AREA

# 

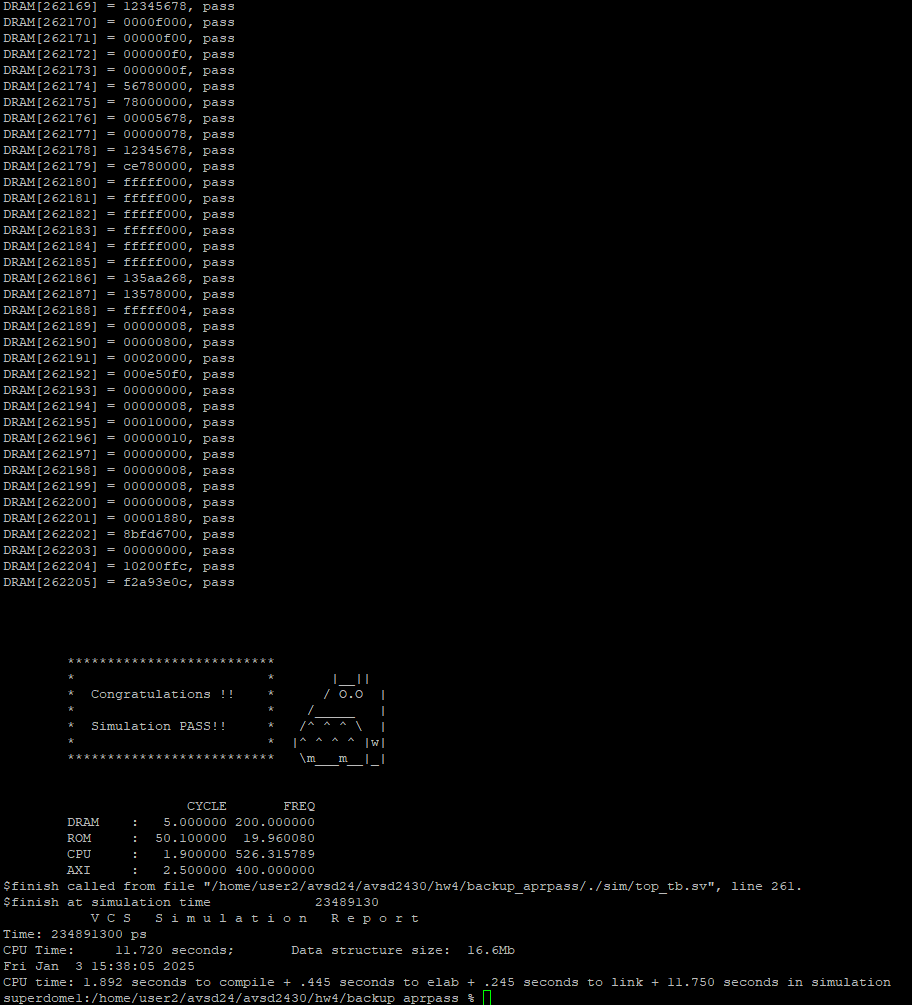
# COST

# 電路面積落在 2.16 mm² ≤ 面積 < 4 mm² 的範圍內

# 該範圍的固定價格為 4,068,000 元

## Prog0 pass

Rtl



Syn



Pr

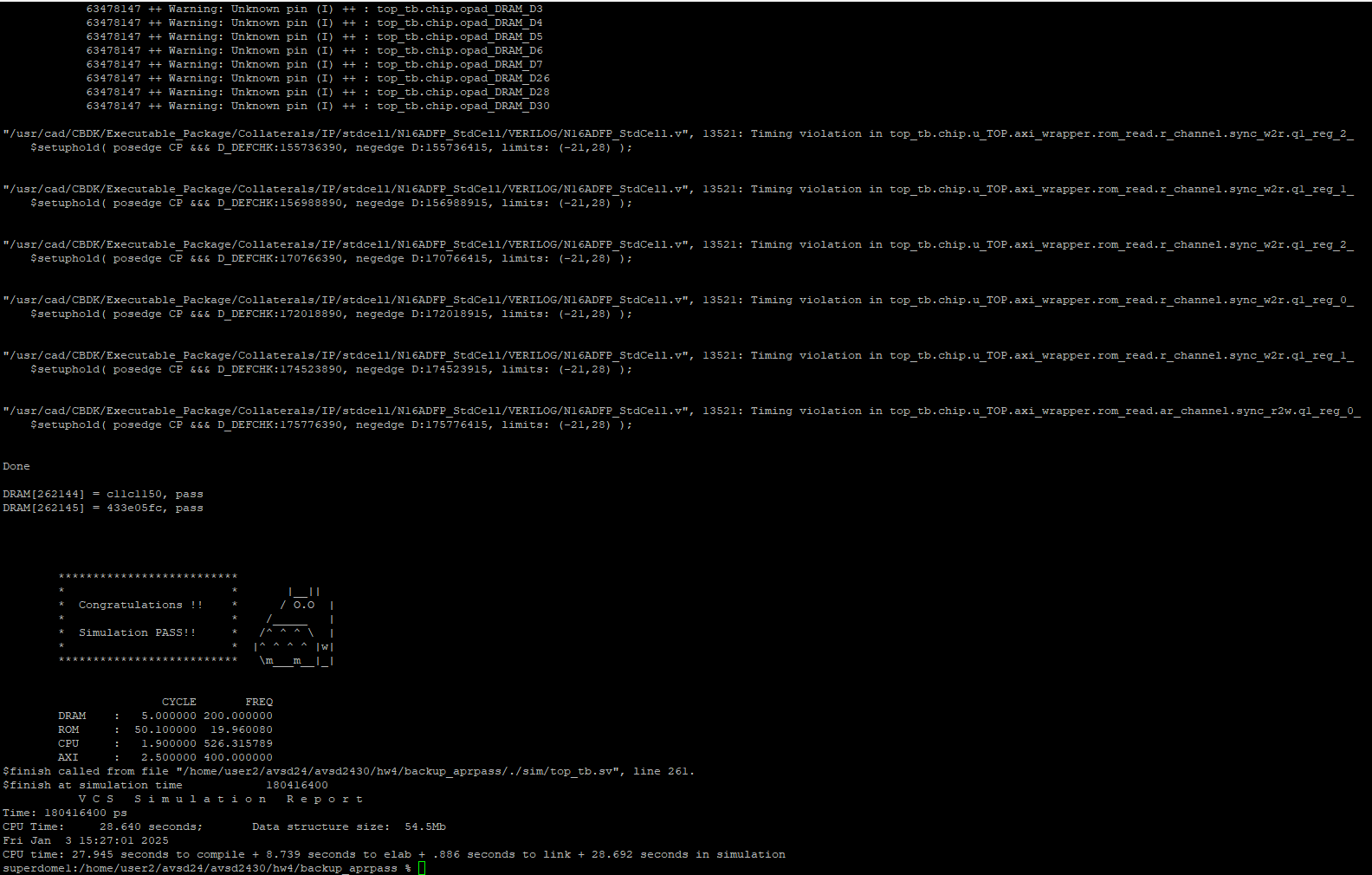


## Prog1 pass

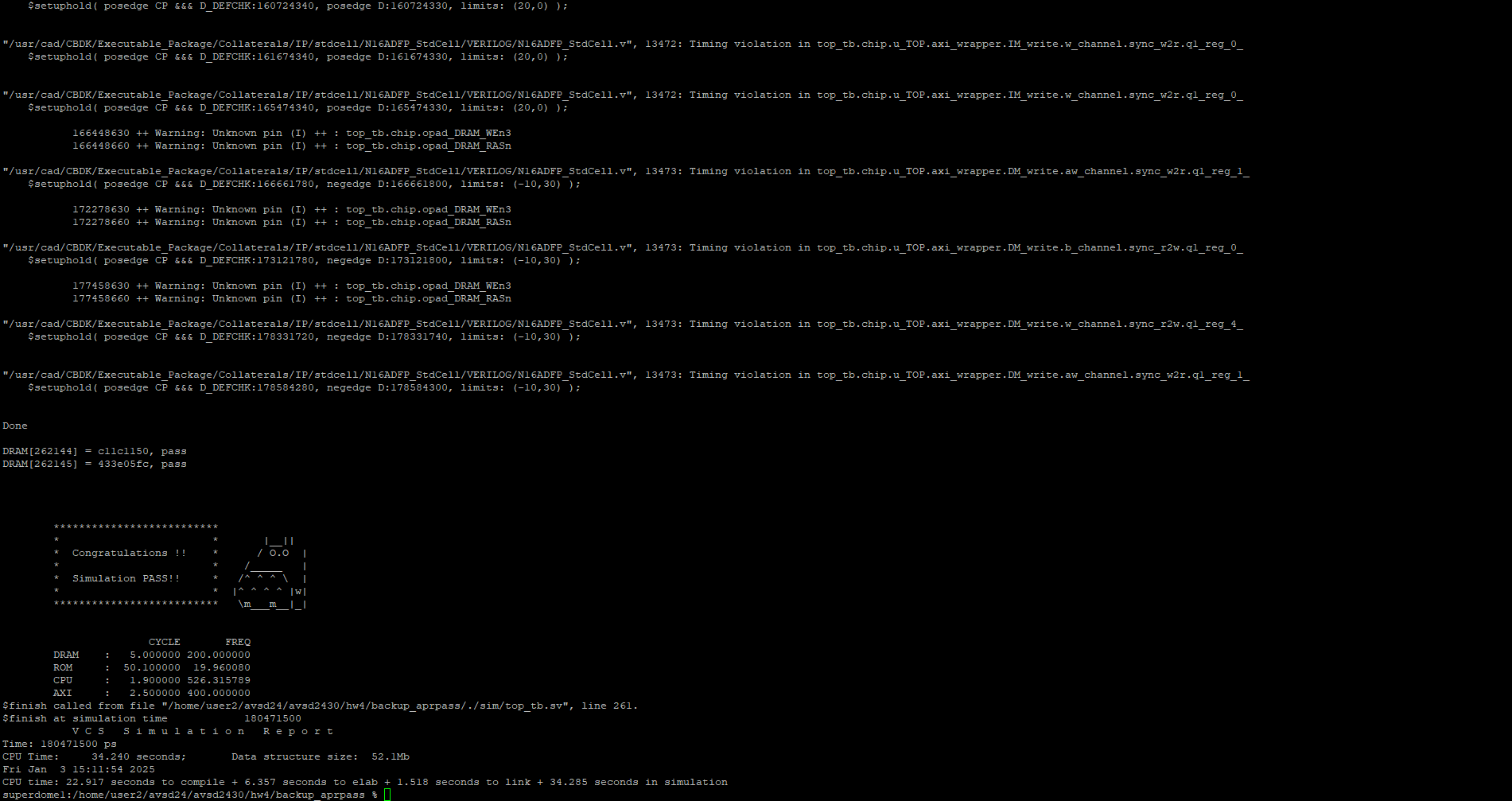
Rtl



Syn

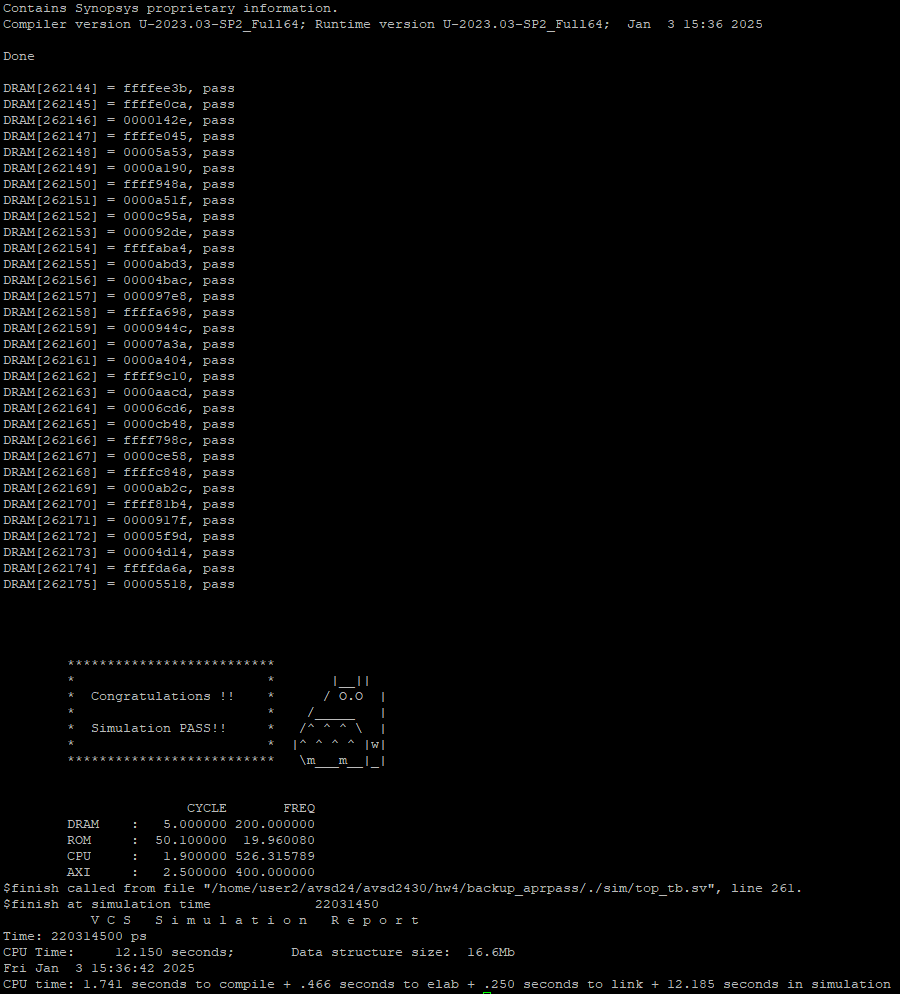


Pr

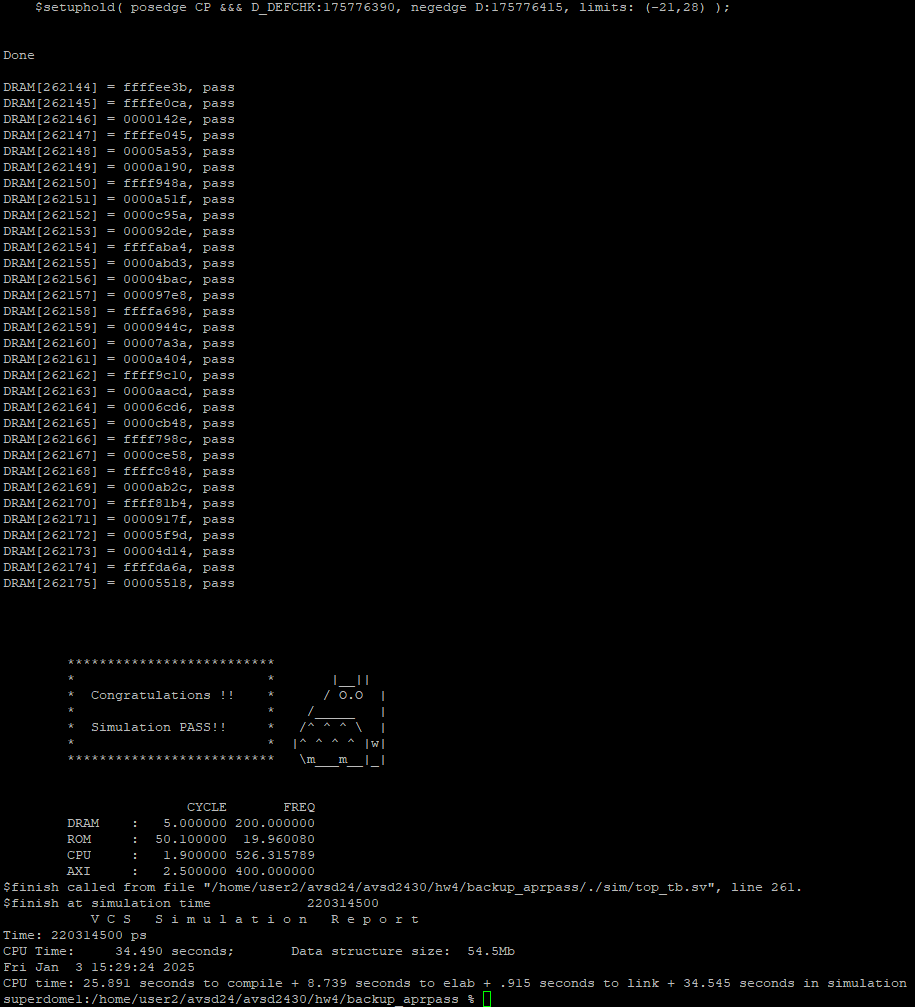


## Prog2 pass

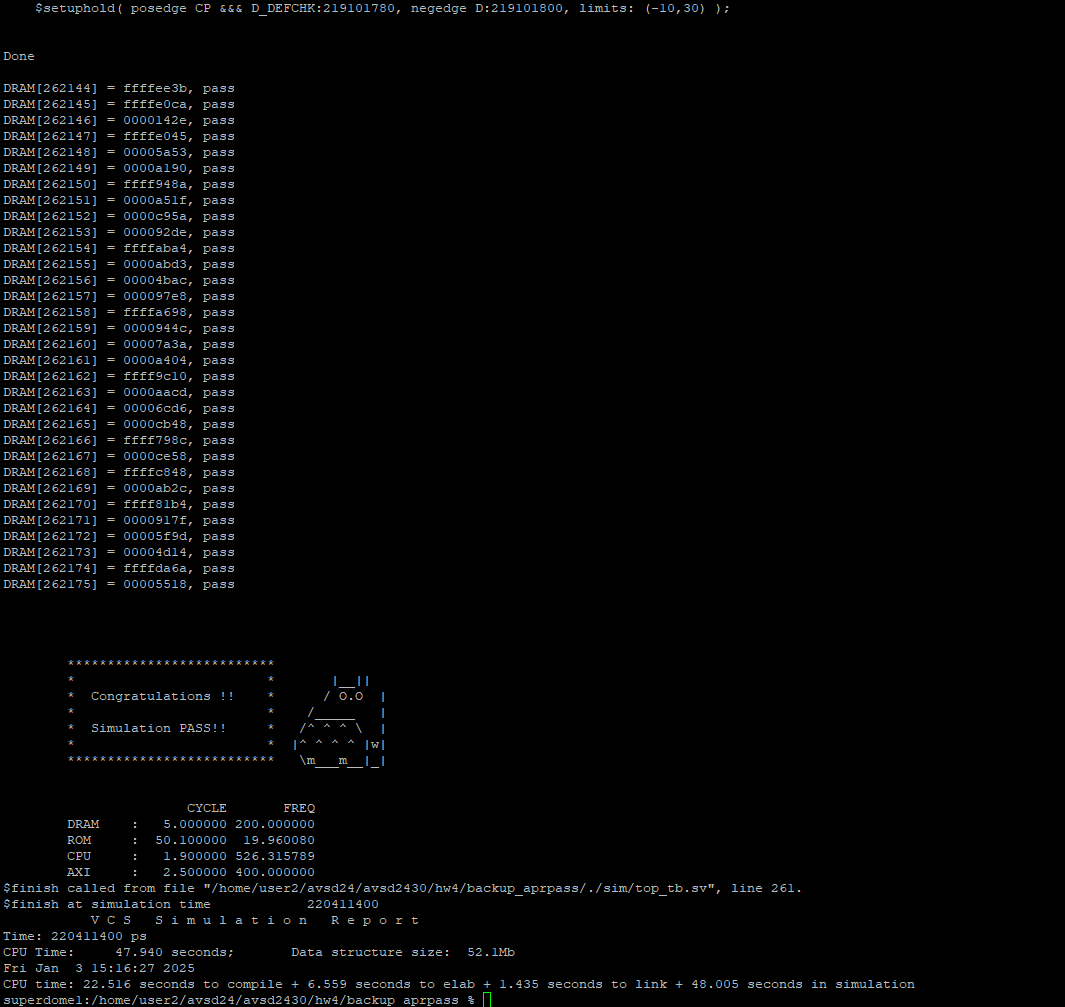
Rtl



Syn



Pr

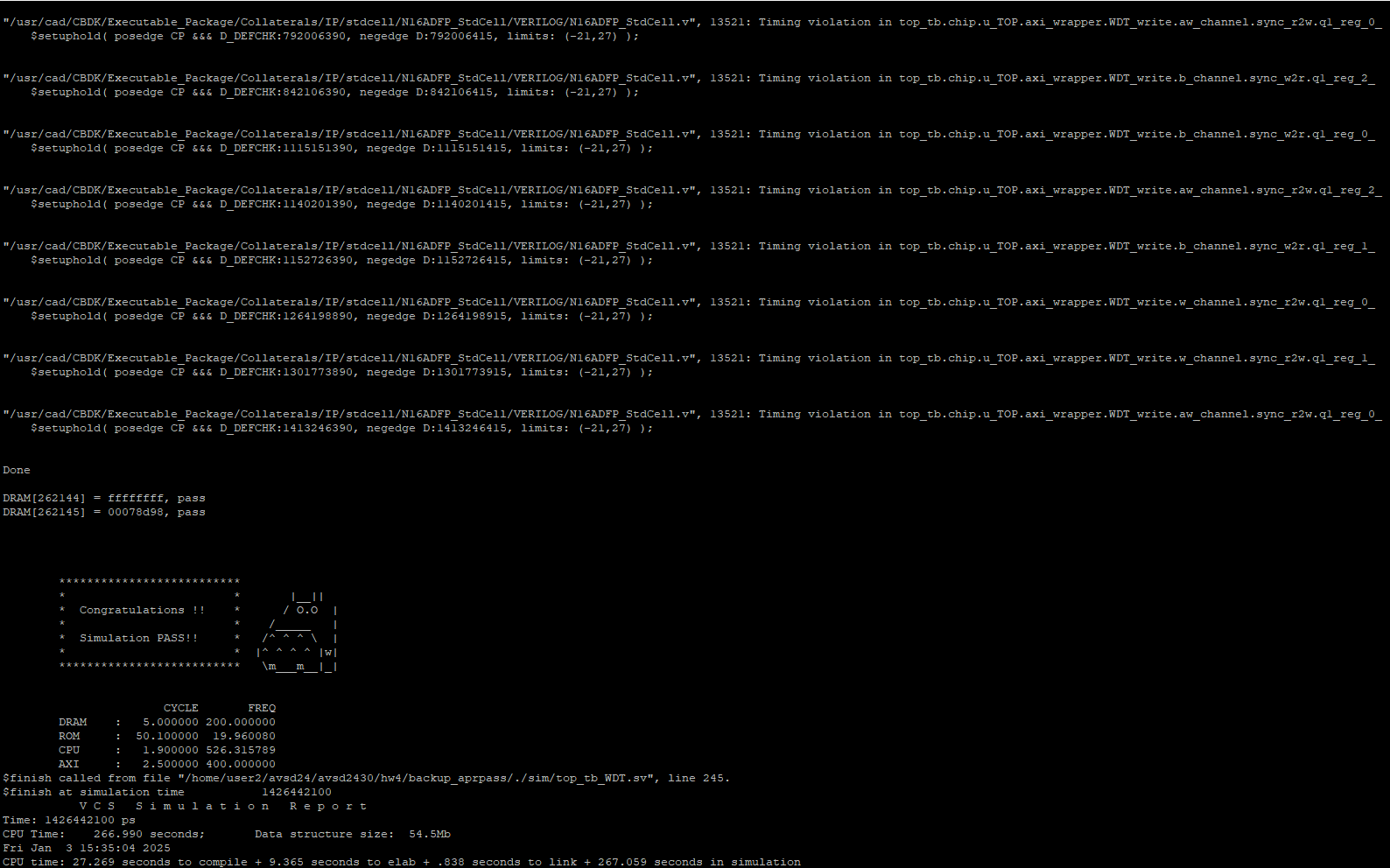


## Prog3 pass

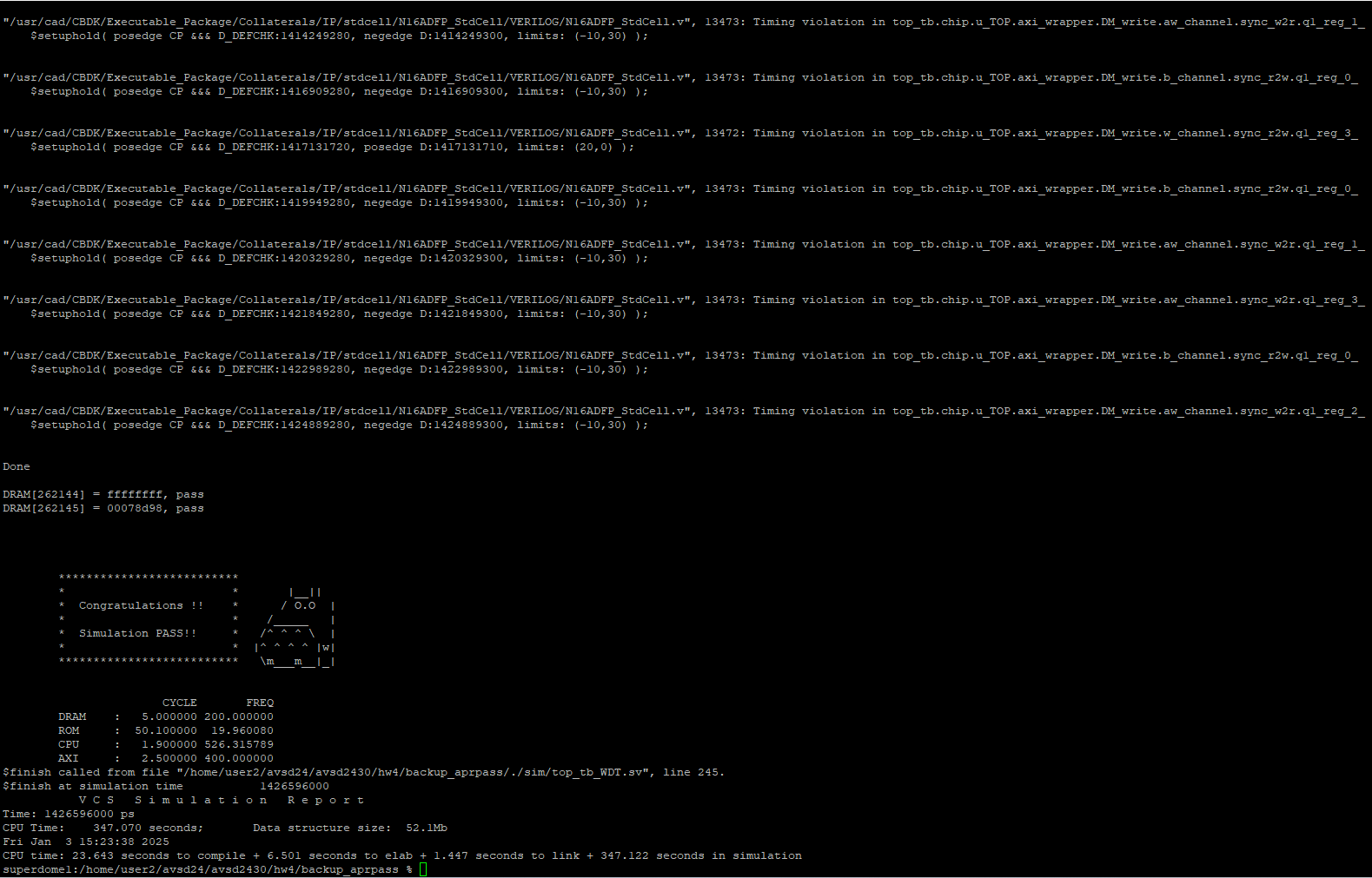
Rtl



Syn



Pr



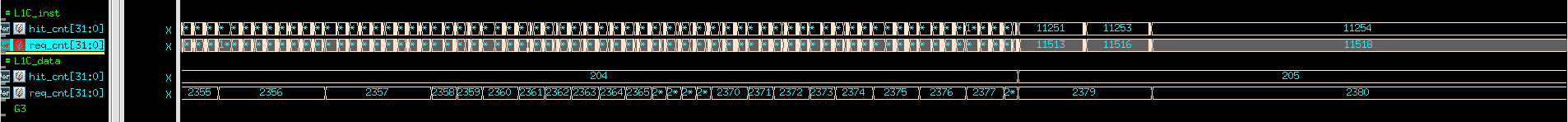
在所有的模擬中，RTL和SYN可以跑過CYCLE 1.0，但是APR的繞線相對沒這麼理想，所以需調大到1.9才可跑過所有測資。而在實驗過程中也發現IO的擺放會影響到效能，一開始我將IOPOWER與COREPOWER分別擺放在1/3與2/3處，僅能跑過CYCLE 大小為2.4。後來將其改為IOPOWER與COREPOWER都放在最中間，便可將CYCLE縮小到1.9。

## Cache Hit Rate

Prog0 :

Instruction Cache : 11254 / 11518 = 97.7%

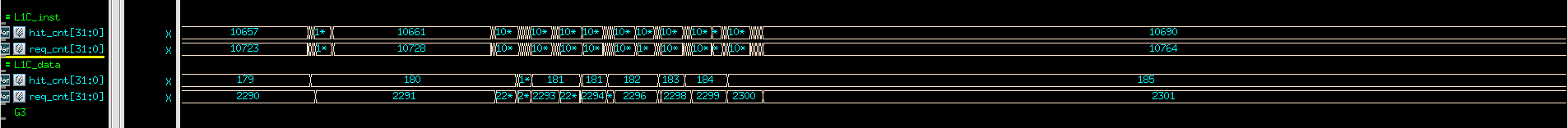
Data Cache : 205 / 2380 = 8.6 %



Prog1 :

Instruction Cache : 10690 / 10764 = 99.3 %

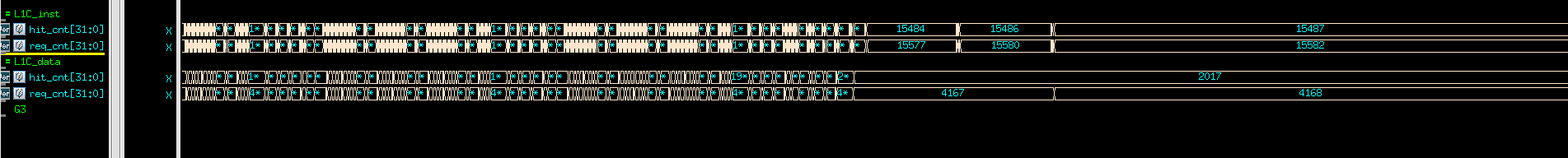
Data Cache : 185 / 2301 = 8.0 %



Prog2 :

Instruction Cache : 15487 / 15582 = 99.4 %

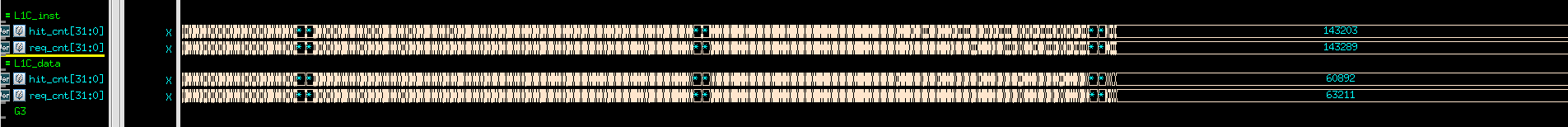
Data Cache : 2017 / 4168 = 48.4 %



Prog3

Instruction Cache : 143203 / 143289 = 99.9 %

Data Cache : 60892 / 63211 = 96.3 %



在結果中顯示Instruction Cache普遍hit rate較高，因為指令都是連續讀取，每四個就至少有3個會hit，所以保底就有75%，加上這次的CACHE大小滿大的又是做2 way，所以hit rate都是九十幾%。Data Cache的Hit Rate則相較沒那麼高，因為前面幾個測資都沒有做甚麼重複讀取相同位址的地方。而在模擬的過程中我發現Write Miss的結果都不會重複讀到，因此在寫入時將判斷是否有hit拿掉，都直接對index做寫入對結果是不會有任何影響的。

# Spyglass CDC check

# Total Fatal : 0

# Total Error : 0

# Total Warning : 7

# Total Information : 1088

# 

# 原本在跑Spyglass的verify時，凡是使用ROM clock domain的AFIFO會出現Ac\_conv02的error，並顯示gray code : FAILED，經研究發現引起的原因是兩個clock domain若相差太大，read 的時鐘若頻率太低，write\_pointer做完CDC之後得到的將是離散的取樣值，並不會像是gray code一樣一次變動一個bit。

# 但是，這個問題並無法解決，因為較快的clock domain往慢的domain傳送資料，要想一個不漏唯一的辦法就是hold住，等到資料被取走了再傳下一個，但是write\_pointer只要FIFO沒滿就可以持續增加寫入的資料，並不會hold住，故必然會有這個問題。

# 然而，這個問題也不需要解決，因為只要FIFO夠大，即使讀到write\_pointer是離散的gray code，也不會影像到FIFO判斷空或滿，只要FIFO的判斷不出錯，非同步FIFO的行為就不會有問題，只是效率可能會稍微降低。舉例來說write\_pointer傳到read端的時候，實際的write\_pointer不可能比read接收到的還要小，而若讀取這端判斷到空，也只會是一個假空，暫時讀取資料，但並不會導致出錯，因為這樣做只是降低了讀的效率(提早判斷空)，但經延遲後還是會讀取到正確的值，不會導致錯誤的結果。

# 但是SPYGLASS仍然會報錯誤訊息，因此我去看Reference的Datasheet發現針對下列訊息的說明:

# 

# 若可以很確定該處的處理上不會有問題，便可使用其提供的方法第四點最後行，使用constraint使其不要report出錯誤訊息，於此便解決了上述的問題。

# 

# 

# 最後verify只出現了如下的warning，為時間無法延長到無限長故只能

# 顯示出部分證明的結果，但是我把時間設定到很長很長還是會有這樣

# 的訊息，故判斷為可忽略的警告。

# 

# Verify\_struct的部分也有出現許多warning，透過與verify相同的方式使用constraint 設定cdc\_filter\_coherency -unrelated，便可屏蔽掉，但最後仍會顯示兩個警告訊息如下:

# 

# 經研究reference datasheet後判斷其非嚴重的影響，有可能只是重複檢查等等的情況，故判斷可忽略此警告。

# Superlint check

## Coverage

一張含有 文字, 螢幕擷取畫面, 軟體, 陳列 的圖片

自動產生的描述

## 3 most frequent warning/errors

1.FLP\_NO\_ASRT : 此錯誤訊息是因為FlipFlop沒有用asynchronous reset而導致的，但非很嚴重的問題僅是作為提醒。

2. FLP\_NR\_ASRT : 暫存器同時有asynchronous和synchronous的訊號，可能導致rtl和gatelevel產生不同模擬結果，但我最後還是跑過了應該影響不大。

3. REG\_NR\_UASR : 暫存器沒有被assign但是有被使用，原因是我有些是直接從module接出來的，但是systemverilog支援用logic同時代表reg和wire，所以才出現此訊息，忽略了也沒關係。

# ICC2 APR Layout flow

# 設計設置 (Design Setup)：

# 導入Gate-Level Netlist和 MCMM的時序約束。

# 設置Floorplan、Scan Chain(這次沒有) 和Power Intent。

# 設計規劃 (Design Planning)：

# 規劃Core區域、I/O PAD及Standard Cell的擺放。

# 建立電源網絡，包括Power Ring、Power Rail及Power Stripe。

# 元件佈局 (Placement)：

# 決定邏輯元件（例如gate、flipflop等）的物理位置，同時考慮功耗、延遲等因素。

# 時鐘樹合成 (CTS)：

# 合成時鐘樹網絡，分配時鐘信號，並進行後續的時序與信號完整性驗證。

# 布線 (Routing)：

# 在元件之間建立物理連接，包括Global Route、Detail Route及Route\_opt階段。

# 處理天線效應，layer jumping或二極體插入解決violations問題。

# 晶片收尾 (Chip Finishing)：

# 進行製造檢查 (DFM)，插入redundant via、metal filler以提升可靠性。

# 輸出 GDSII 檔案作為layout數據。

# 設計檢查 (DRC 與 LVS)：

# 進行設計規則檢查 (DRC) 和layout與schematic一致性檢查 (LVS)。

# 面積與成本計算：

# 計算晶片的面積並評估其製造成本。

# DFM\_DRC

# 

# DRC

# 

# LVS

# 

# DRC的結果非常多，估計是因為我APR的CHIP.func是用1.0去做APR，

# Lessons Learned

 **設計挑戰與解決方法**  
在設計快取系統時，為了處理快取未命中（Cache Miss）的情況，我採用了額外增加兩個時鐘週期的解決方案。雖然這略微降低了性能，但大幅簡化了狀態機設計並提高了穩定性。這教會我如何在性能與設計複雜度之間進行權衡。

 **工具應用與技術提升**  
在使用 Spyglass 進行時鐘域跨越 (CDC) 檢查時，我學到了如何設置有效的constraint來過濾非致命警告，例如gray code錯誤的處理方式。這不僅提升了我的工具使用能力，也讓我更清楚設計中可能對性能無影響的問題應如何處理。

 **效能與成本的取捨**  
在晶片面積的優化過程中，我理解到透過合理的Floorplan與元件佈局策略，可以有效降低製造成本，並在不影響主要性能的前提下達成設計目標。

 **未來改進建議**  
若再次進行類似設計，我會在早期階段優化地板規劃並加強對工具的熟悉度，尤其是 ICC2 的佈局與布線流程。此外，我會預留更多時間進行驗證，避免後期修改帶來的額外成本。

 **個人成長**  
本次專案中，我加強了對 SystemVerilog 語言的掌握，並學習到如何以符合規範的方式組織程式碼，這對未來的設計與合作將大有幫助。

# Contribution

|  |  |  |
| --- | --- | --- |
| Name | 王劭懷 |  |
| Contribution percentage | 100 % |  |
| Contribution parts | ALL |  |