VLSI System Design (Graduate Level)

Fall

HOMEWORK II

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 any waveform files in deliverables

Student name: \_\_\_陳柏廷\_\_\_ \_\_鄭惟\_\_

Student ID: \_\_P76091284\_\_ \_\_P76091226\_\_

Summary

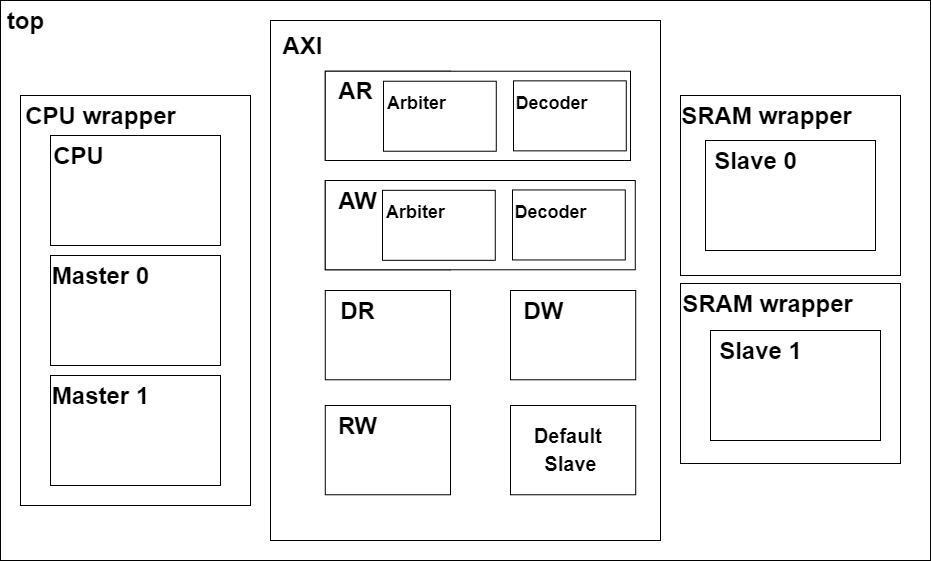
* 完成實作HW2之必要module
  + AXI(AR, DR, AW, DW, RW, Arbiter, Decoder)
  + Sram\_wrapper
  + CPU\_wrapper(Master)
* 將HW1的CPU增加stall線路使得在連接上AXI後能夠將CPU stall，讓CPU運作正確
* 重新撰寫驗證的Prog2和Prog3之main.c(由於compiler無法接受%及\*運算子)

貢獻度

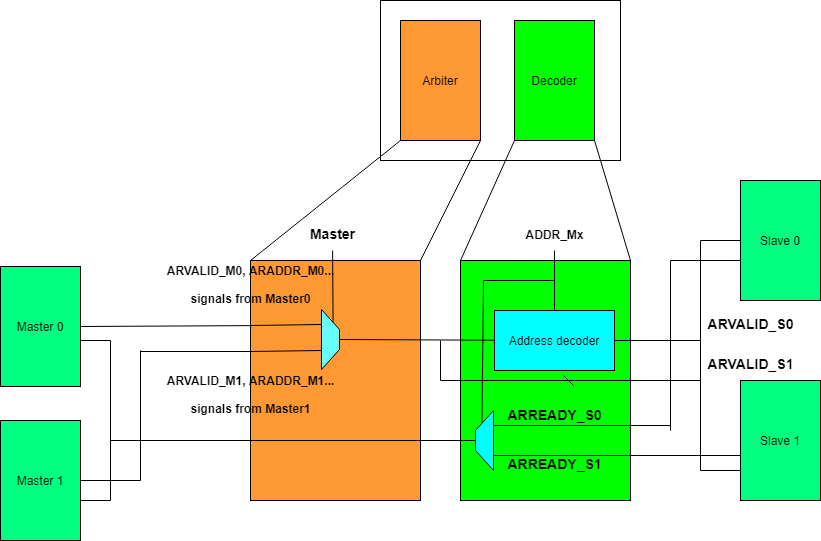
|  |  |
| --- | --- |
| 陳柏廷 | 鄭惟 |
| P76091284 | P76091284 |
| 50% | 50% |

Overview

下圖為整個Pipeline CPU with AXI module 示意圖

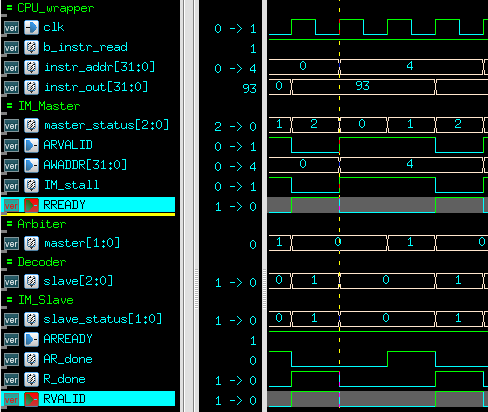
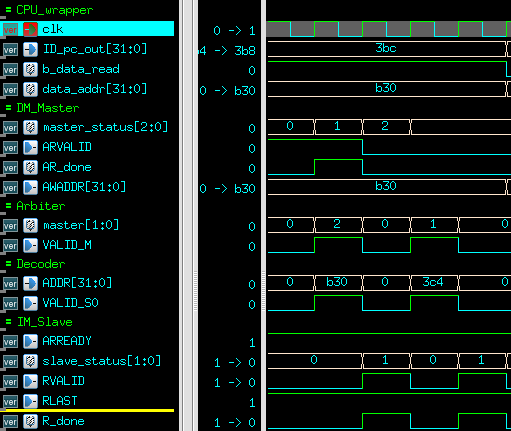


AW及AR中之Arbiter及Decoder示意圖

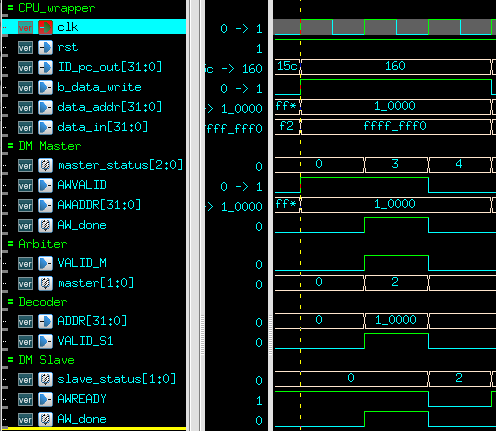


Waveform verification

由於此次重點在於AXI bus的新增，因此R-type、B-type及J-type就不再驗證，將會著重於LW及SW以及instruction如何從IM取得

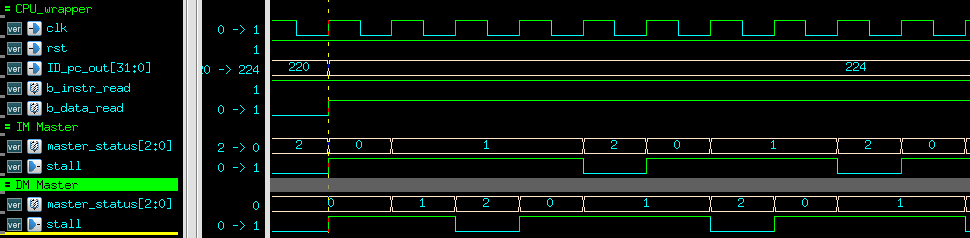
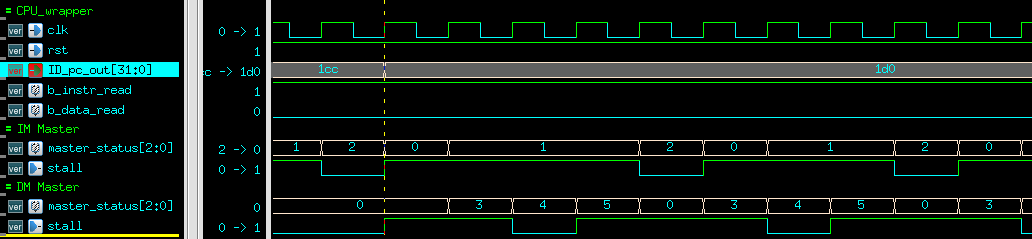
1. Get instructions from IM via AXI bus
   1. .
   2. 上圖紅框處為CPU將read(表示CPU取得指令)送給Master0(IM Master)，此時Master0會將IM\_stall拉高讓CPU停止運作，並將ARVALID及相關訊號線(ADDR、ID…等)送入AXI bus。
   3. 上圖白框處表示IM master在上一個cycle的status為3’b0(init)且送出ARVALID，因此在下一個cycle切換為3’b1(read\_addr)等待ARREADY
   4. 上圖黃框處為Arbiter接受的master及Decoder欲送入的slave，master為1表示接收來自Master0的請求，slave為0表示於取得IM 之SRAM的data。
   5. 上圖藍框處表示IM master在上一個cycle為3’b1(read\_addr)且因為AR channel已經handshake，所以在這個cycle切換至3’b10(read\_data)。
   6. 上圖綠框處IM slave之相關訊號線，slave與master狀態機略有不同，因上一個slave IM之status為2’b0(read\_addr)且收到ARVALID，因此status切換至2’b1(read\_data)  
      並在這個cycle完成R channel的handshake(nWave反白處)並送出從SRAM取得的data(instruction)。
2. LW
   1. 
   2. 上圖紅框處為CPU將b\_data\_read (表示DM取得data)送給Master1(DM Master)，此時Master1會將DM\_stall拉高讓CPU停止運作，並將ARVALID及相關訊號線(ADDR、ID…等)送入AXI bus。
   3. 上圖白框處表示DM master在上一個cycle的status為3’b0(init)且送出ARVALID，因此在下一個cycle切換為3’b10(read\_addr)等待ARREADY
   4. 上圖黃框處為Arbiter接受的master及Decoder欲送入的slave，master為1表示接收來自Master0的請求，Decoder根據ADDR為b30(slave 0)將VALID及其他相關訊號送至IM SRAM。
   5. 上圖藍框處表示DM master在上一個cycle為3’b1(read\_addr)且因為AR channel已經handshake，所以在這個cycle切換至3’b10(read\_data)。
   6. 上圖綠框處IM slave之相關訊號線，slave與master狀態機略有不同，因上一個slave IM之status為2’b0(read\_addr)且收到ARVALID，因此status切換至2’b1(read\_data)  
      並在這個cycle完成R channel的handshake(nWave反白處)並送出從SRAM取得的data(instruction)。
   7. 可以發現到此指令在DM master取得data後，CPU仍被stall，是因為IM master及DM master同時需要AXI bus取得data，但DM master會先於IM master獲得Arbiter鎖定，因此在DM master完成transfer後由IM接著使用AXI並stall CPU，並且可以發現到因為ARREADY一直為拉高的情形，因此IM slave在read\_data的同時可以完成與IM Master的AR handshake。
3. SW

因為write的動作較read步驟為多，因此這邊將每個channel的步驟拆開來觀察。

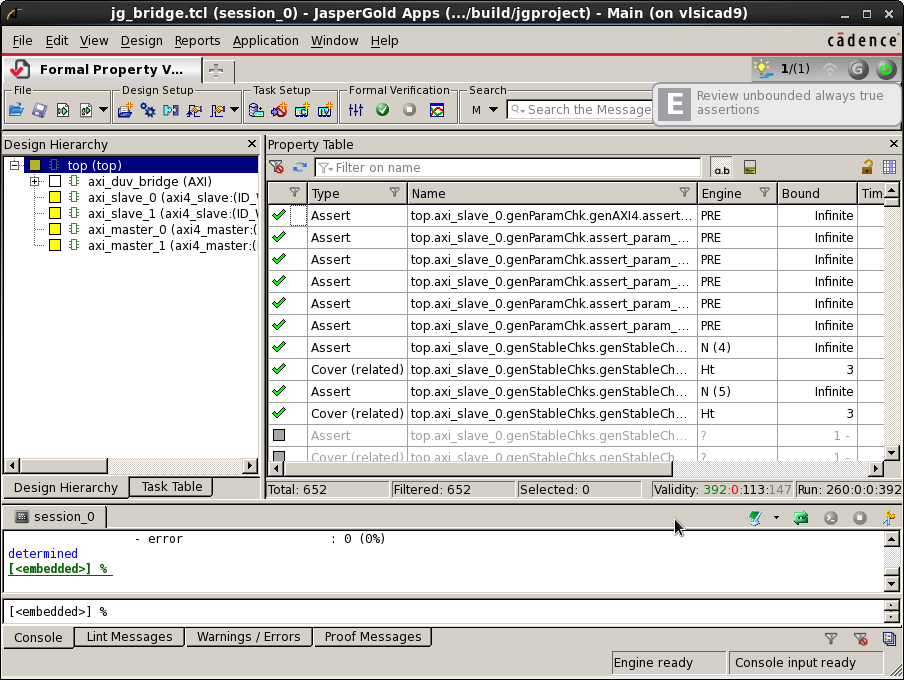
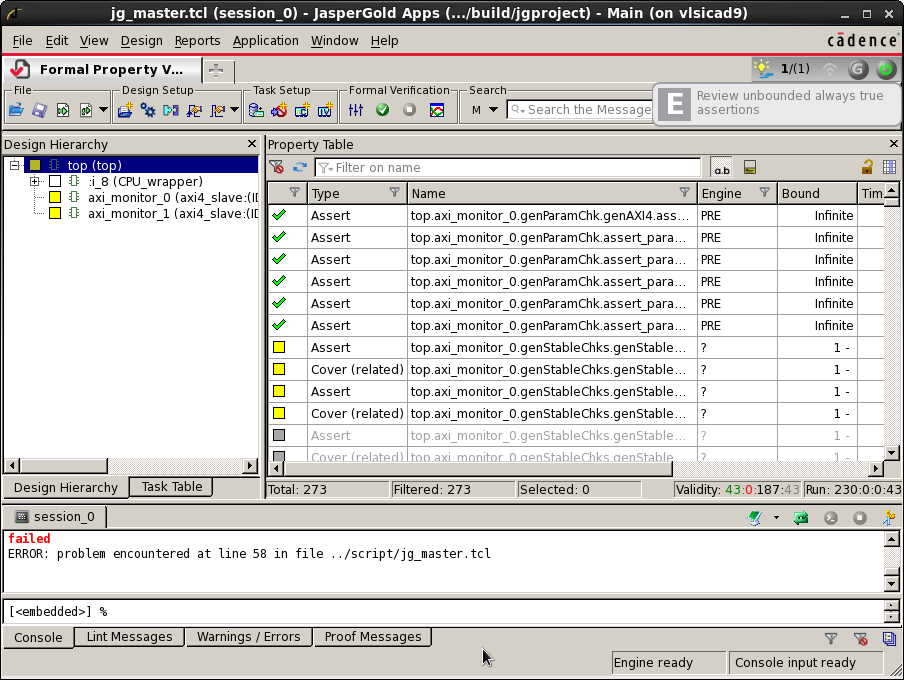
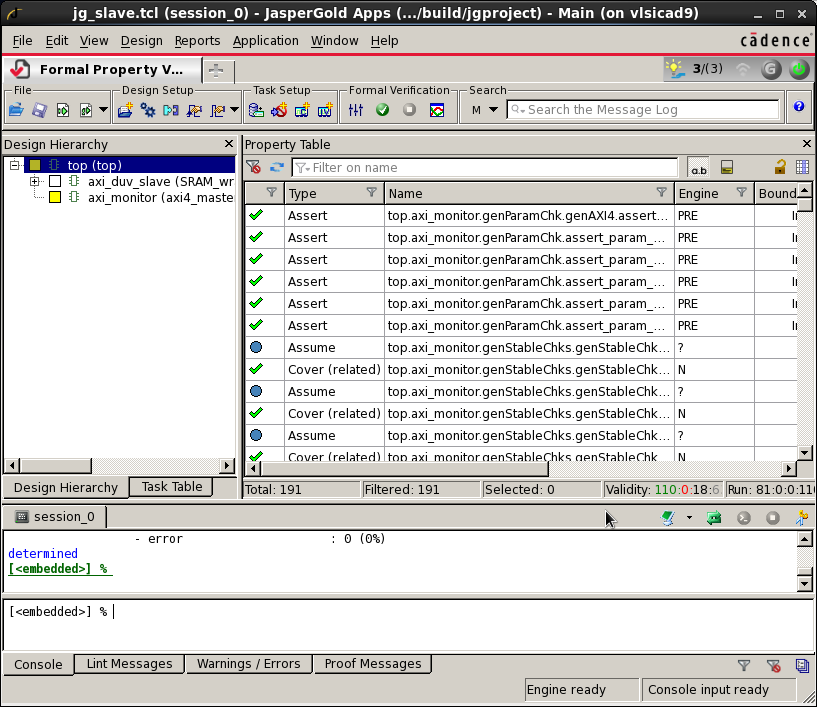
1. Write address channel
   1. 
   2. 上圖紅框處為CPU將b\_data\_write (表示DM寫入data)送給Master1(DM Master)，此時Master1會將DM\_stall拉高讓CPU停止運作，並將AWVALID及相關訊號線(ADDR、ID…等)送入AXI bus。
   3. 上圖白框處表示DM master在上一個cycle的status為3’b0(init)且送出AWVALID，因此在下一個cycle切換為3’b11(write\_addr)等待AWREADY
   4. 上圖黃框處為Arbiter接受的master及Decoder欲送入的slave，master為2表示接收來自Master1的請求，Decoder根據ADDR為1\_0000(slave 1)將VALID及其他相關訊號送至DM SRAM。
   5. 上圖藍框處表示DM master在上一個cycle為3’b11(write\_addr)且因為AW channel已經handshake，所以在這個cycle切換至3’b10(write\_data)。
2. Write data channel & Write response channel
   1. 
   2. 承前一個state，上圖藍框處表示DM master在上一個cycle為3’b11(write\_addr)且因為AW channel已經handshake，所以在這個cycle切換至3’b10(write\_data)。
   3. 上圖綠框處DM slave之相關訊號線，因上一個slave DM之status為2’b0(read\_addr)且收到AWVALID，因此status切換至2’b10(write\_data)  
      並在這個cycle接收到master欲寫入之data，並送入DM SRAM，完成W channel的handshake
   4. 上圖黃框處，可以發現到完成W channel handshake後即繼續下一個指令，而master及slave皆未回到init及read\_address的狀態，而是由於上一個cycle為write\_data而進入到write\_response。

Lesson learned

這次的作業，不僅是撰寫bridge以及用jaspergo驗證的複雜及難度，在寫完AXI的protocol後，我們還需要考量許多master及slave在transfer的過程中產生的問題，比如說2個master存取同一個slave，或是因stall的問題而導致CPU狀態卡死。

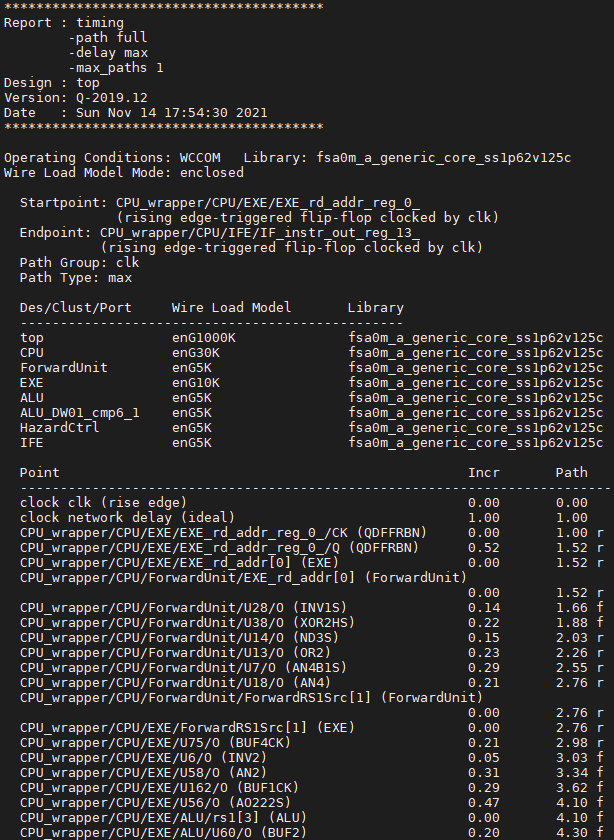
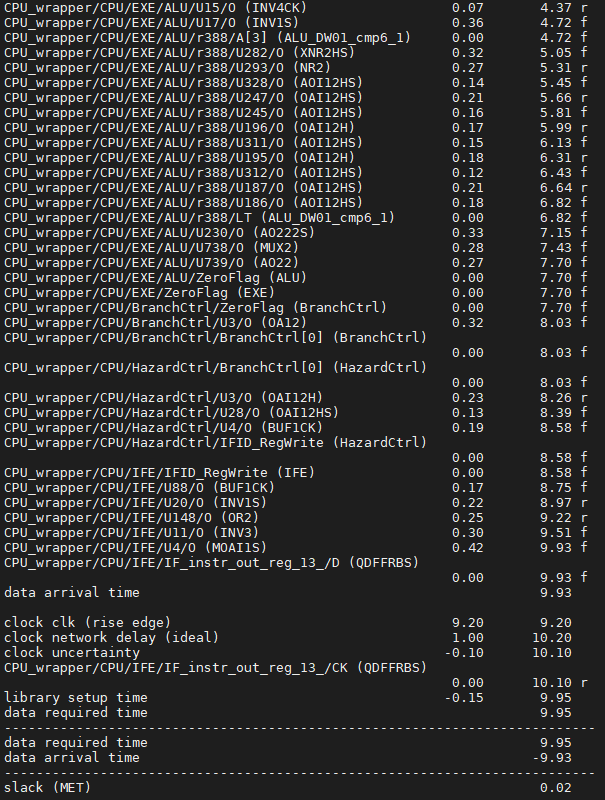
1. Data Master在取得data後仍繼續要求導致CPU dead lock(I): Master0(IM Master) read與Master1(DM Master) read
   1. 
   2. 如上圖紅框處，可以發現到DM master的status為2的時候已經完成了R channel handshake，並且將stall訊號拉低。
   3. 而上圖黃框處，則是IM Master等待DM master完成transfer後接續使用AXI來完成read instruction
   4. 那可以看到因為兩個master輪流的stall 導致CPU無法在繼續往下一個指令執行
   5. Resolution: 因此在DM Master 在取得data之後必須等到IM master也完成並拉低stall訊號之後才能在發起下一次的read or write request。
2. Data Master在取得data後仍繼續要求導致CPU dead lock(II): Master0(IM Master) read與Master1(DM Master) write
   1. 
   2. 與前一個狀況類似，但Master1此次為write data。

Verify AXI architecture

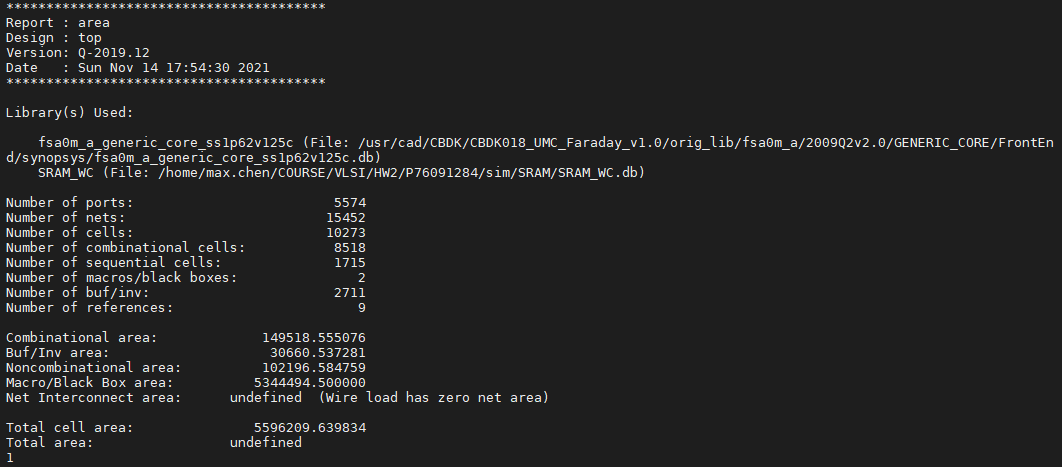
1. Bridge  
   
2. Master  
   
3. Slave  
   

Report

* Timing report

* Area report



Simulation results

|  |  |  |
| --- | --- | --- |
|  | rtl | syn |
| Prog0 |  |  |
| Prog1 |  |  |
| Prog2 |  |  |
| Prog3 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Prog0 | Prog1 | Prog2 | Prog3 |
| RTL | Pass | Pass | Pass | Pass |
| SYN | Pass | Pass | Pass | Pass |

Performance

|  |  |
| --- | --- |
| Cycle time | 9.2ns |
| Total Cell Area | 5596209.639834 |