Lab_B

Travelling Salesperson Problem　Hardware Accelertor

電子所碩一 林修賢

Problem Description

- Travelling salesman problem(TSP):

  Given a list of cities and the distances between each pair of cities, finding shortest possible route that visits each city exactly once and returns to the origin city.

  NP-hard problem, its time complexity is O(N!)

下面是原版問題用 software 去解的時間

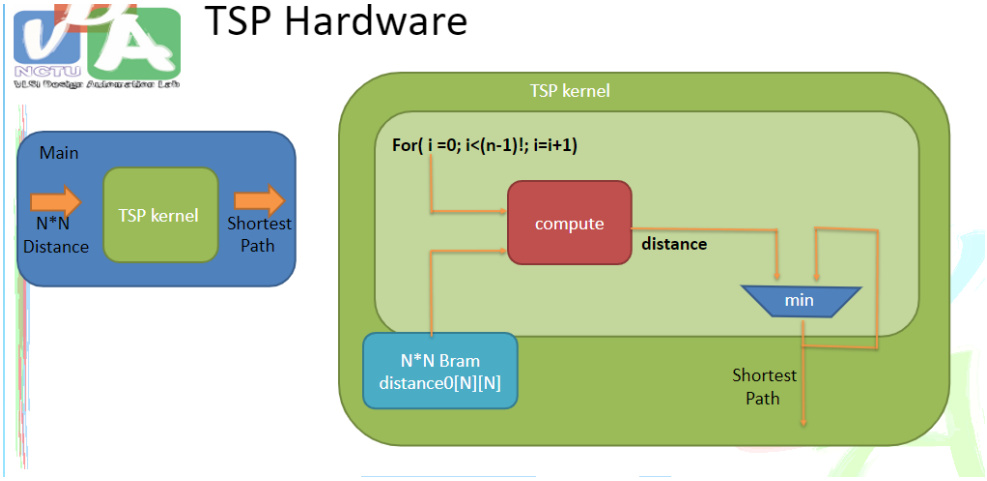
(naïve TSP by software with 13 city)

這在此 lab，我們的城市數目訂為 13，cycle 都用 10ns。

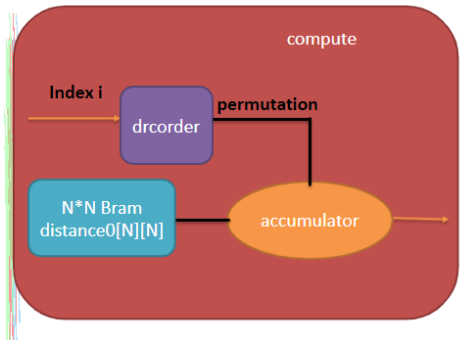
下面是我把 official 的 design 和其 opt 版本可視化

Official Design (reference)

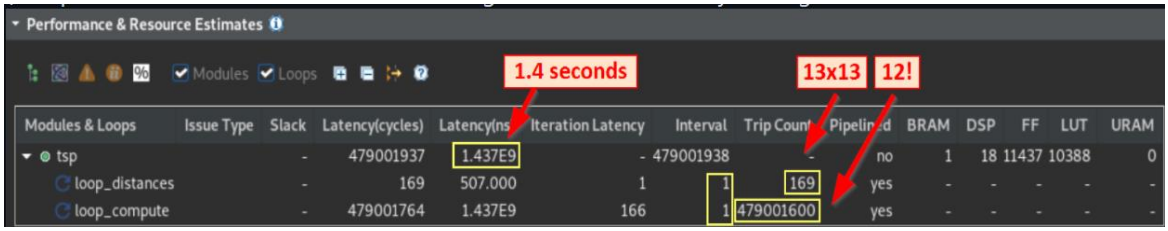
TSP Hardware

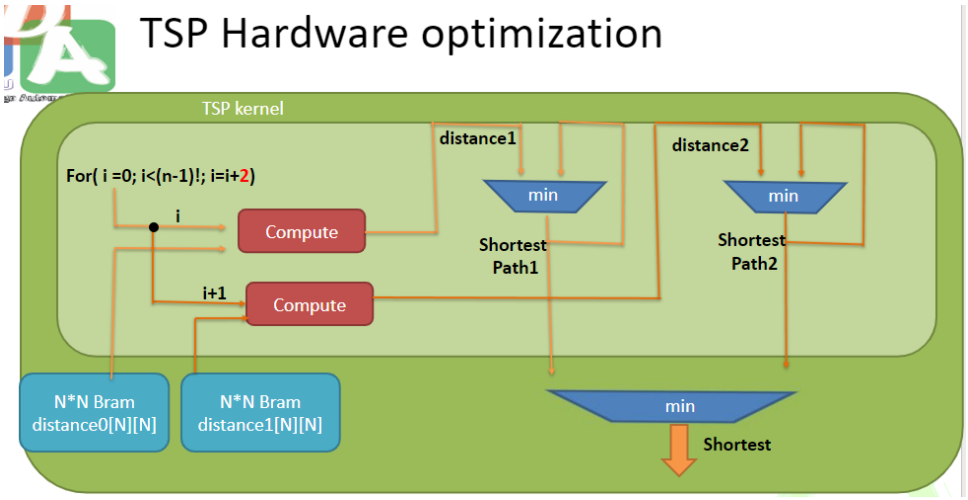


TSP Hardware



- Ex: N=13 , index= 0
  - Permutation is 1 2 3 4 5 6 7 8 9 10 11 12 13
- Ex: N=13 , index= 479001599
  - Permutation is 13 12 11 10 9 8 7 6 5 4 3 2 1

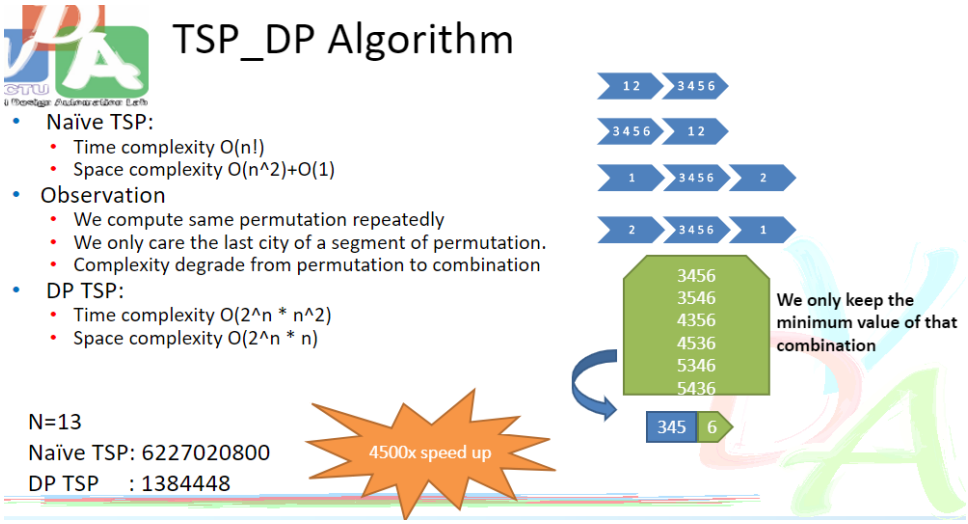


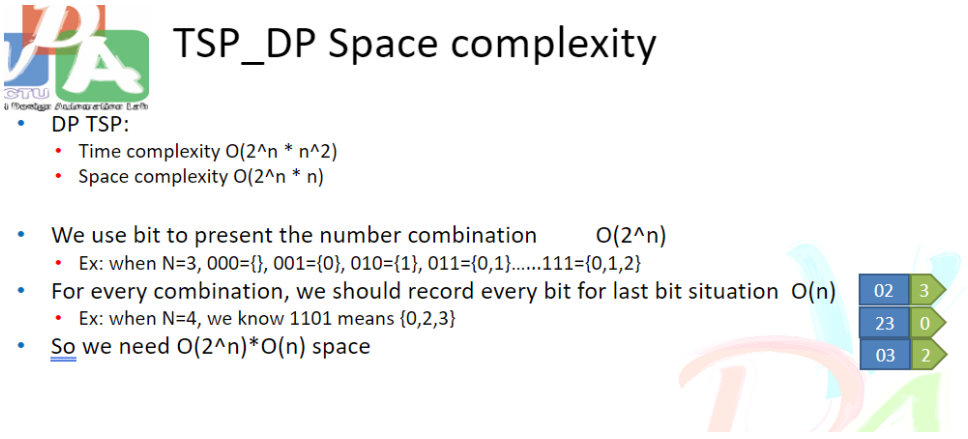


It optimization speed up at most N times speed. (N is 13 in this case)

但上面的 naive 算法太爛了，硬體比軟體只有加速 30 倍(1.4 秒)，就算用他的優化平行也頂多 390 倍。

於是我用 DP 去加速

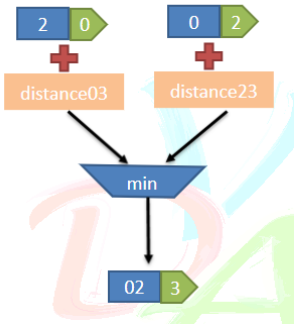




TSP_DP Time complexity

- DP TSP:
  - Time complexity O(2^n * n^2)
  - Space complexity O(2^n * n)
- How do we compute a block?
- For every block, we cost O(N) to compute

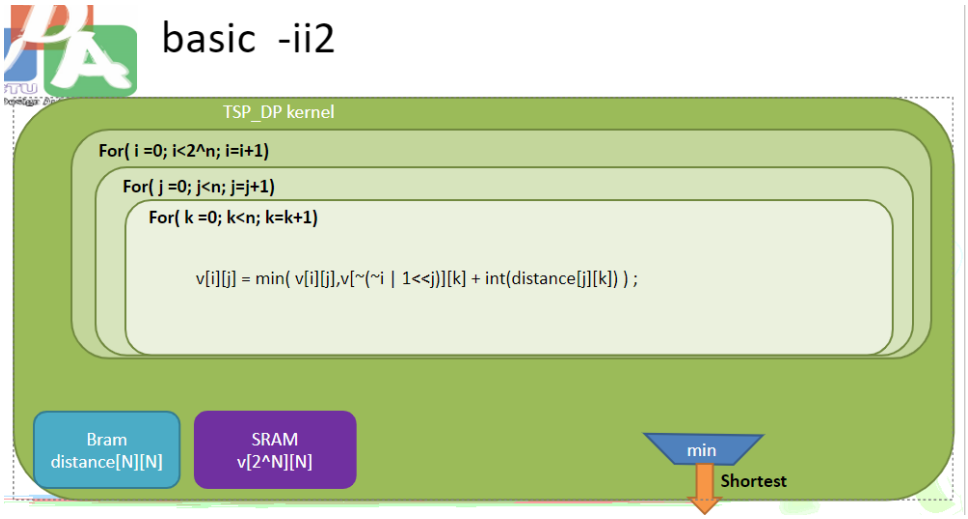
Example : N=4,{0,2,3},last bit is 3



My Design

優化 1 (Algorithm optimization)

<硬體原始版>，直接把 DP 演算法用軟體思維寫上去



每次都對同塊記憶體做一次讀一次寫，所以必定要 stall 一次(ii=2)。

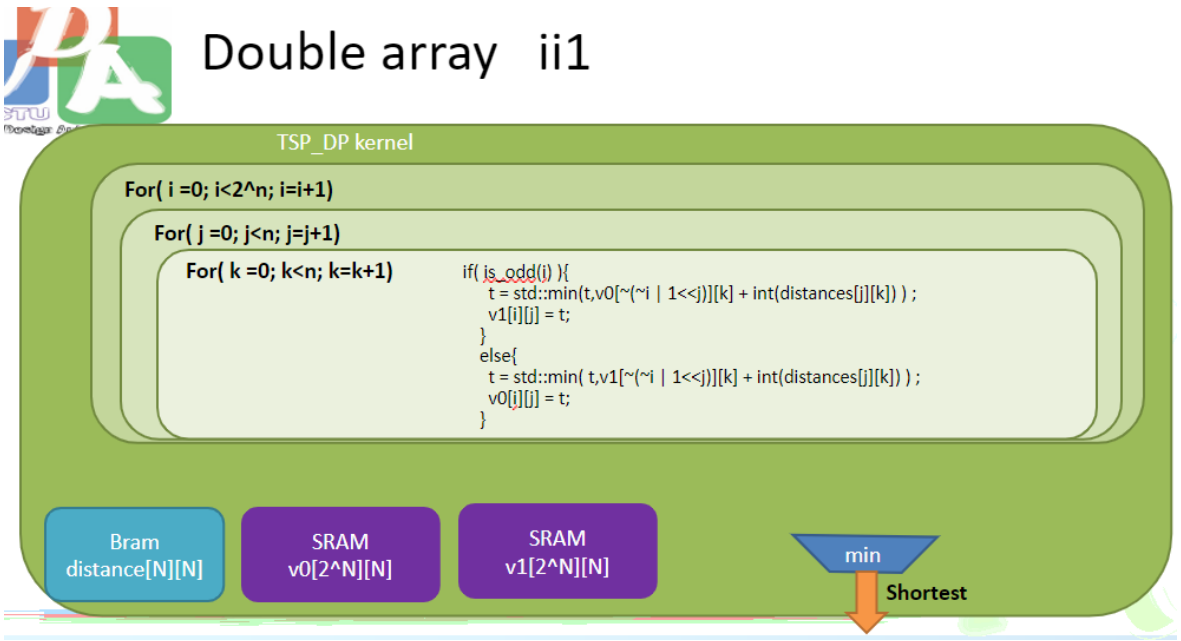
Synthesis | Cosimulation

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
tsp				-	2874749	2.875E7	-	2874750	-	no	257	4	494	1270	0
tsp_Pipeline_loop_distances				-	171	1.710E3	-	171	-	no	0	0	10	66	0
loop_distances				-	169	1.690E3	1	1	169	yes	-	-	-	-	-
tsp_Pipeline_loop_inialization				-	106500	1.065E6	-	106500	-	no	0	1	51	150	0
loop_inialization_VITIS_LOOP				-	106498	1.065E6	4	1	106496	yes	-	-	-	-	-
tsp_Pipeline_loop_fill_dsp_blc	II Violation			-	2768226	2.768E7	-	2768226	-	no	0	3	346	661	0
loop_fill_dsp_block1_loop_fill	II Violation	Memory Dependency	2	-	2768224	2.768E7	7	2	1384110	yes	-	-	-	-	-
tsp_Pipeline_loop_select_canc				-	15	150.000	-	15	-	no	0	0	74	181	0
loop_select_candidate				-	13	130.000	3	1	12	yes	-	-	-	-	-

13^2*2^13 大概是 140 萬，結果因為最內層 ii=2，硬生生的變成 280 萬 cycle....

優化 2 (data dependency optimization)

我發現每次讀跟寫 memory 的 address 都差 1bit，所以我直接複製用一塊記憶體，就不會每次讀寫都產生 stall 了



Synthesis | Cosimulation

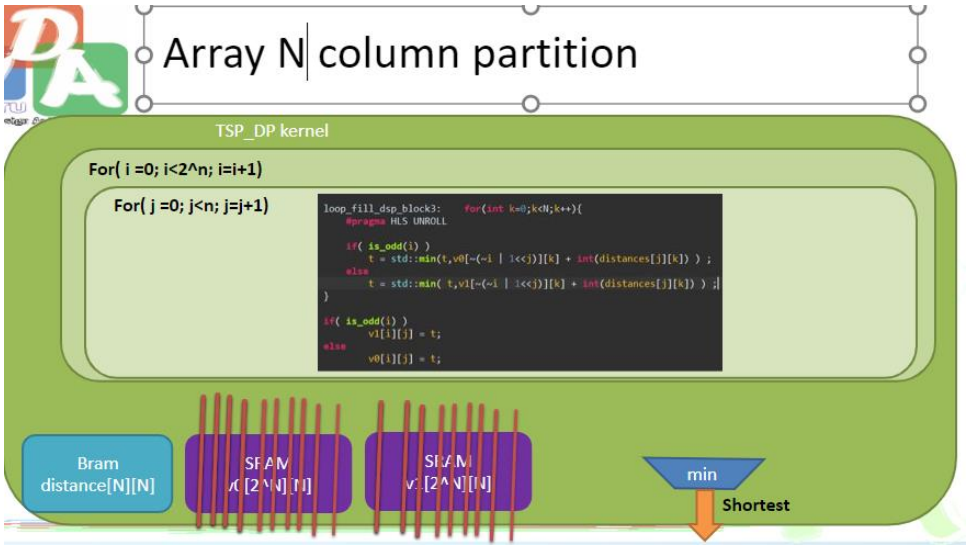
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
tsp				-	1695381	1.695E7	-	1695382	-	no	513	4	548	1462	0
tsp_Pipeline_loop_distances				-	171	1.710E3	-	171	-	no	0	0	10	66	0
loop_distances				-	169	1.690E3	1	1	169	yes	-	-	-	-	-
tsp_Pipeline_loop_inialization				-	106500	1.065E6	-	106500	-	no	0	1	51	150	0
loop_inialization_VITIS_LOOP				-	106498	1.065E6	4	1	106496	yes	-	-	-	-	-
tsp_Pipeline_loop_select_canc				-	15	150.000	-	15	-	no	0	0	108	308	0
loop_select_candidate				-	13	130.000	3	1	12	yes	-	-	-	-	-
loop_fill_dsp_block1				-	1588860	1.589E7	194	-	8190	no	-	-	-	-	-
tsp_Pipeline_VITIS_LOOP_28_				-	15	150.000	-	15	-	no	0	0	41	82	0
VITIS_LOOP_28_1				-	13	130.000	1	1	13	yes	-	-	-	-	-
tsp_Pipeline_loop_fill_dsp_blc				-	175	1.750E3	-	175	-	no	0	2	264	517	0
loop_fill_dsp_block2_loop_fill				-	173	1.730E3	6	1	169	yes	-	-	-	-	-

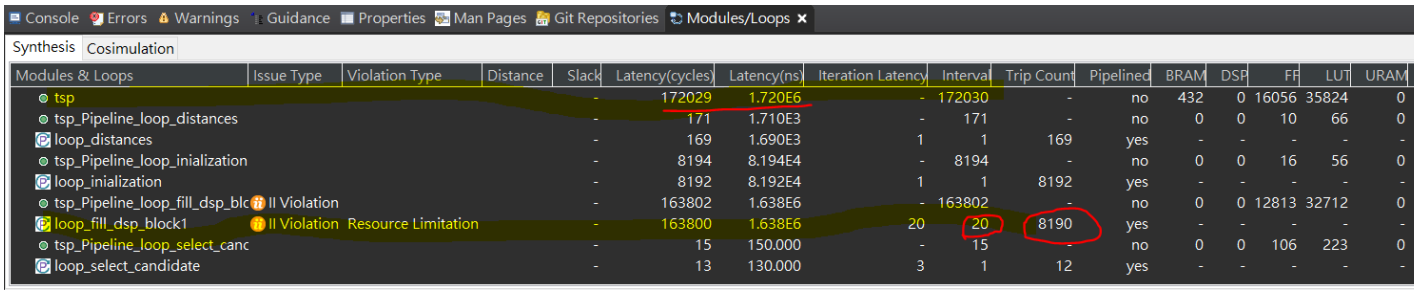
ii = 1

*雖然有一些 overhead，但實際值 1695382 跟理論上最佳化([2^N]*N*N,N=13) =1384448，算是同一個量級的

優化 3 (parallel optimization)

我需要讀 N 個值做運算，算出最小的值，並寫到另個 memory 中，這 13 個值每次都在不同的 array column，所以我可以把兩塊 array 個別切成 13 個 column，同時對內部做 unroll





Console Errors Warnings Guidance Properties Man Pages Git Repositories Modules/Loops

Synthesis Cosimulation

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
tsp				-	172029	1.720E6	-	172030	-	no	432	0	16056	35824	0
tsp_Pipeline_loop_distances				-	171	1.710E3	-	171	-	no	0	0	10	66	0
loop_distances				-	169	1.690E3	1	1	169	yes	-	-	-	-	-
tsp_Pipeline_loop_inialization				-	8194	8.194E4	-	8194	-	no	0	0	16	56	0
loop_inialization				-	8192	8.192E4	1	1	8192	yes	-	-	-	-	-
tsp_Pipeline_loop_fill_dsp_blc	II Violation			-	163802	1.638E6	-	163802	-	no	0	0	12813	32712	0
loop_fill_dsp_block1	II Violation	Resource Limitation		-	163800	1.638E6	20	20	8190	yes	-	-	-	-	-
tsp_Pipeline_loop_select_canc				-	15	150.000	-	15	-	no	0	0	106	223	0
loop_select_candidate				-	13	130.000	3	1	12	yes	-	-	-	-	-

我把 loop3 unroll 後，他也自動幫我把 loop2 unroll 了，原本每次 loop 2 要花 169cycle，

現在每次只要花 20cycle，理論上我切成 13 塊，如果可以完全平行就可以降成 13，但在 loop 轉換時會有一些 data dependency 的 over heap，那個是沒辦法消除了。

理想值([2^N]*N*N/N,N=13)=106,496 跟我們實際 172030 其實是一樣量級，不過因為沒有完全平行(轉換寫入 memory 時有 overhead)，且當計算量越小時，memory 初始化所花的時間看起來就會比較顯著。

優化 4 (area optimization)

後來才發現原來我用的 BRAM 數量超過 100%，他可能幫我合成比較遠的 memory 了

應該降低 array 的大小。

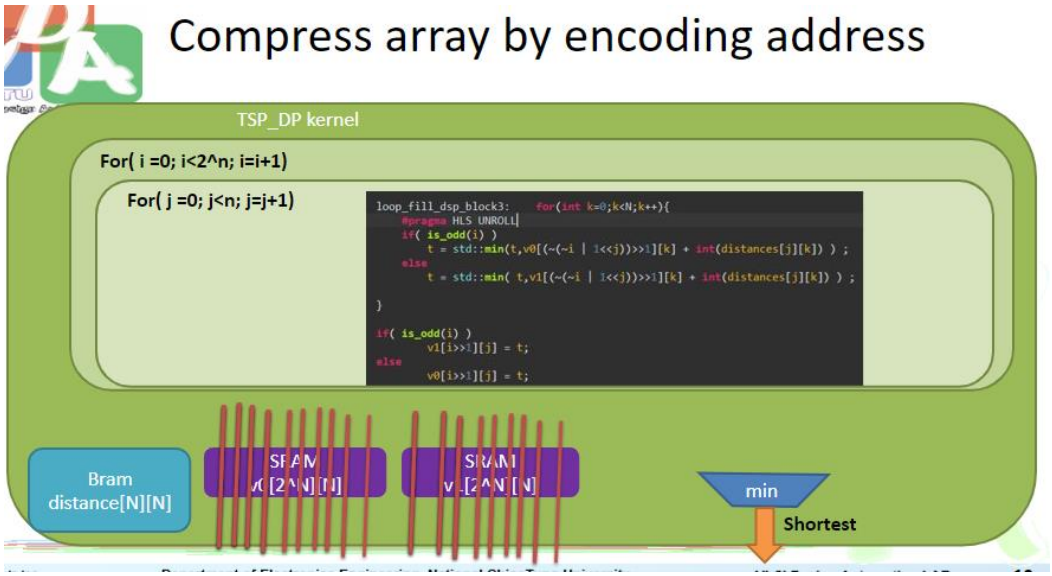
Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	-	-	12945	33057	-
Memory	432	-	0	0	0
Multiplexer	-	-	-	2765	-
Register	-	-	3111	-	-
Total	432	0	16056	35824	0
Available	280	220	106400	53200	0
Utilization (%)	154	0	15	67	0

(優化 3 時的使用資源)

剛剛多用一倍的記憶體，但每塊實際上只用到一半的量

我發現可以用一些編碼，做到空間的壓縮，那兩塊 memory 的大小可以減半



Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	-	-	11523	33904	-
Memory	224	-	0	0	0
Multiplexer	-	-	-	2999	-
Register	-	-	3111	-	-
Total	224	0	14634	36905	0
Available	280	220	106400	53200	0
Utilization (%)	80	0	13	69	0

(優化 4 時的使用資源)

原本只是想要優化面積，

沒想到因為 BRAM 放的下去的關係，他自己幫我全部轉 BRAM 了，

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM
tsp				-	77847	7.780E5	-	77848	-	no	224
tsp_Pipeline_loop_distances				-	171	1.710E3	-	171	-	no	0
loop_distances				-	169	1.690E3	1	1	169	yes	-
tsp_Pipeline_loop_inialization				-	4098	4.098E4	-	4098	-	no	0
loop_inialization				-	4096	4.096E4	1	1	4096	yes	-
tsp_Pipeline_loop_fill_dsp_blc	II Violation			-	73716	7.370E5	-	73716	-	no	0
loop_fill_dsp_block1	II Violation	Resource Limitation		-	73714	7.370E5	14	9	8190	yes	-
tsp_Pipeline_loop_select_canc				-	15	150.000	-	15	-	no	0

速度起飛。

他幫我第二層 loop 自動做 unloop，20 cycle -> 9 cycle，這是我不曾想到的

總時間也從剛剛的 1.716E6 ns 變成 7.78E5 ns。

總結

我的硬體最佳化 --> 7.78E5 ns

* 可比 SW TSP_naive 版 (43 秒)的快 55270 倍。

*可比 SW TSP_dp 版本(0.01 秒)快 13 倍。

*可以比他的參考版的快 1833 倍，比他平行優化快 141 倍。