Lab A

R11943128 陳維隆

這次的Lab是要利用不同優化的方式實現DCT function，並且分析電路行為及觀察在不同優化下的Analysis report。

* **Introduce DCT fubction**

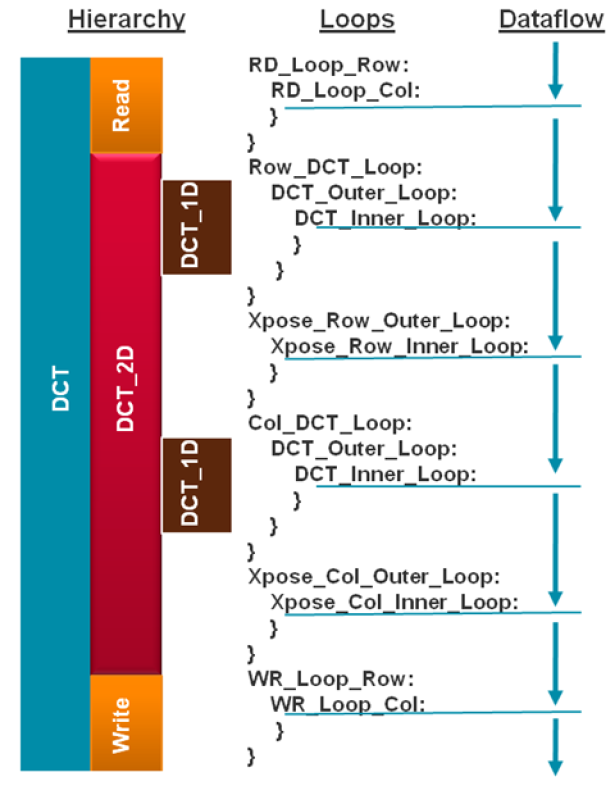
The left side of Figure shows the code hierarchy.

* Top-level function dct has three sub-functions: read\_data, dct\_2d and write\_data.
* Function dct\_2d has a single sub-function dct\_1d.

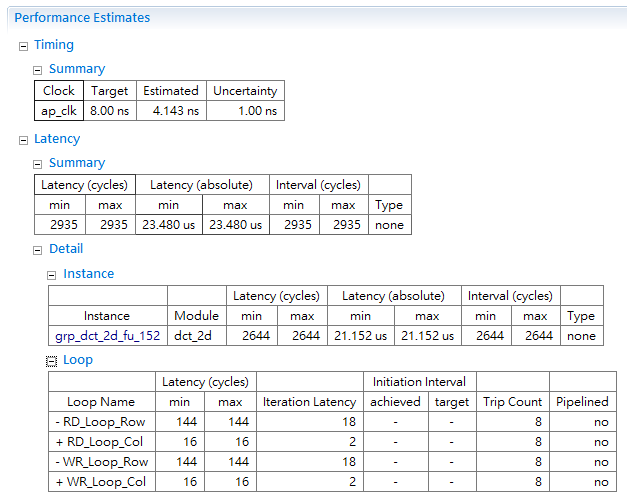
The center of Figure shows loops inside each of the functions.

The right side of shows the how the data is processed through the functions and loops.

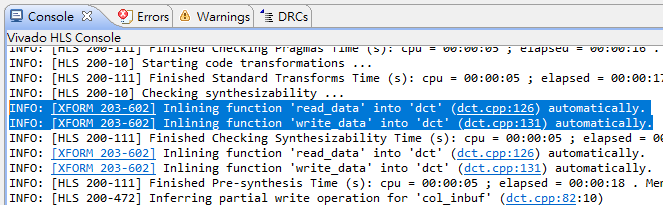
* The read\_data function executes, and the data is processed through loop RD\_Loop\_Row, which has a sub-loop RD\_Loop\_Col.
* In function dct\_2d, Row\_DCT\_Loop processes the data. Row\_DCT\_Loop has two nested loops inside it: DCT\_output\_loop and DCT\_inner\_loop.
* DCT\_inner\_loop calls function dct\_1d.
* The write\_data function is same as the read\_data.



* **Baseline**

以下是未進行任何優化(baseline)跑C synthesis的結果

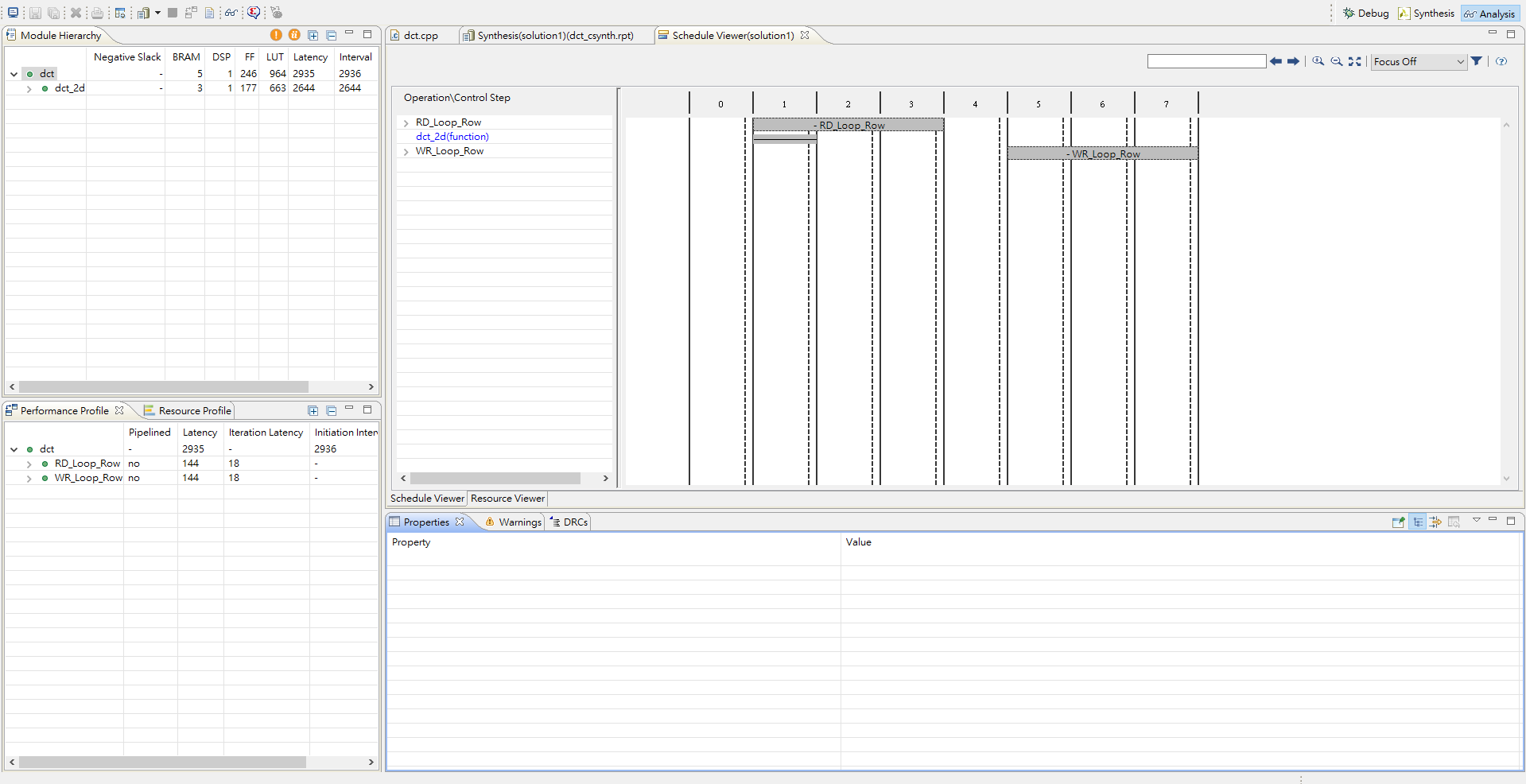
* 從input到output共花了2935個cycles，且Type為 none表示未進行pipelined
* 如下圖，在執行synthesis的時候，系統自動將”read\_data”和”write\_data”兩個functions inlined，所以在上圖的表格中top level只有dct\_2d一個instance(紅圈處)。



* 以RD\_Loop\_Row為例，解釋為何loop latency是144 cycles:

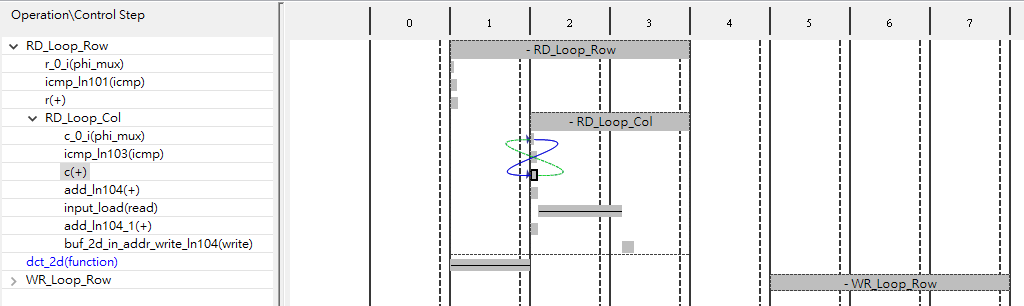
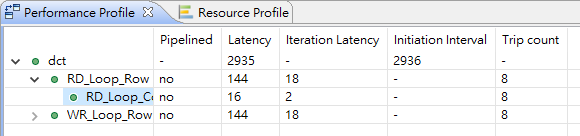
在sub-loop “RD\_Loop\_Col”中iteration latency = 2，而這個loop執行了8次(Trip Count)，所以總共2 \* 8 = 16 cycles。外層loop “RD\_Loop\_Row”進出sub-loop需要各花一個cycle，因此iteration latency = (1+16+1) = 18，所以

total loop latency = 8(Trip Count) x 18 = 144 clock。

* 全部的Latency = 144(RD\_Loop\_Row) + 2644(dct\_2d) + 144(WR\_Loop\_Row) + 3(enter each of those three blocks) = 2935 cycles
* **Analysis report**

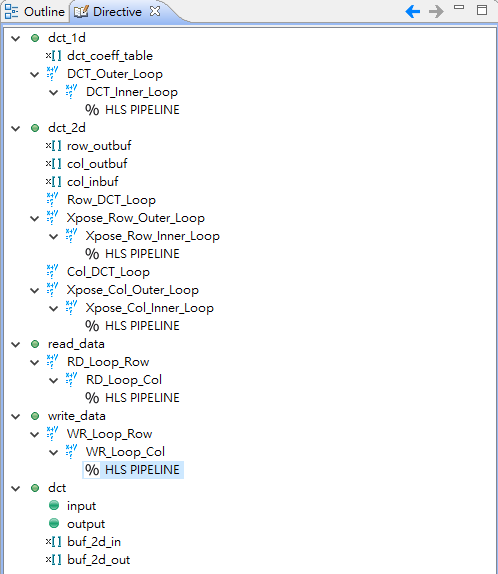
在Analysis的畫面中可以看到以下幾個panes

* Module Hierarchy: Show both the performance and area information for the entire design.
* Performance Profile:Show the performance details for this level of hierarchy.
* Schedule Viewer:Show how the operations in this particular block are scheduled into clock cycles.

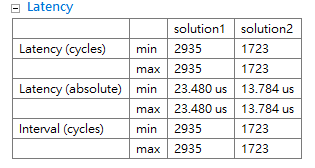
 

在Schedule Viewer中可以清楚看到每個function/loop(藍圈處)在不同cycles中如何運作，而上圖中綠藍的loop代表此function迭代(trip count)的次數，可以在Performance Profile的視窗中看到(紅圈處)

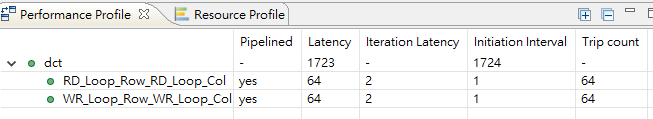
* **Apply Loop Pipelining**

以下是將其中幾個inner-loop插入pipeline的Directive

執行C Synthesis後將report拿來與baseline(solution 1)做比較，可以看出latency少了將近50%



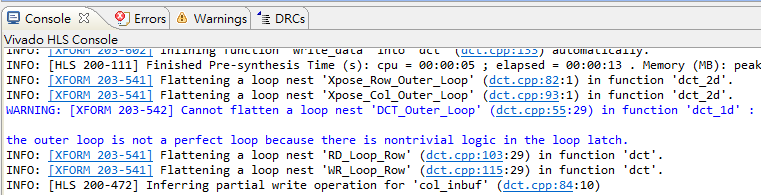
Improved almost 50%



從Performance Profile可以看到這兩個Loop的Latency從144降到64 clock cycles，計算Latency的方式也因為Pipeline的關係變為:

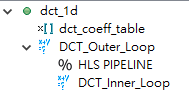
Latency = iteration latency + (tripcount \* interval) = 64

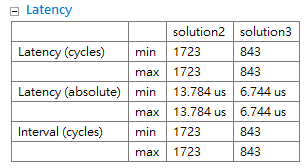
除了做Pipeline之外，系統自動將以下四個loop(下圖紅框處)做flattening，而中間的WARNING是因為DCT\_Outer\_loop跟DCT\_Inner\_loop之間還有dst[k] = DESCALE(tmp, CONST\_BITS)這個statement，所以無法做Flattening。

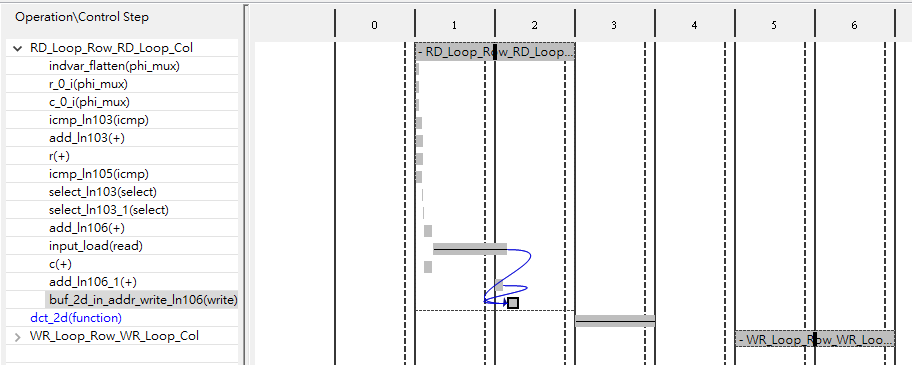


* **Apply Loop Optimization**

從上面的分析可看出dct\_1d是Latency最大的function，於是Pipelining DCT\_Outer\_Loop可以將Latency降至843 cycles

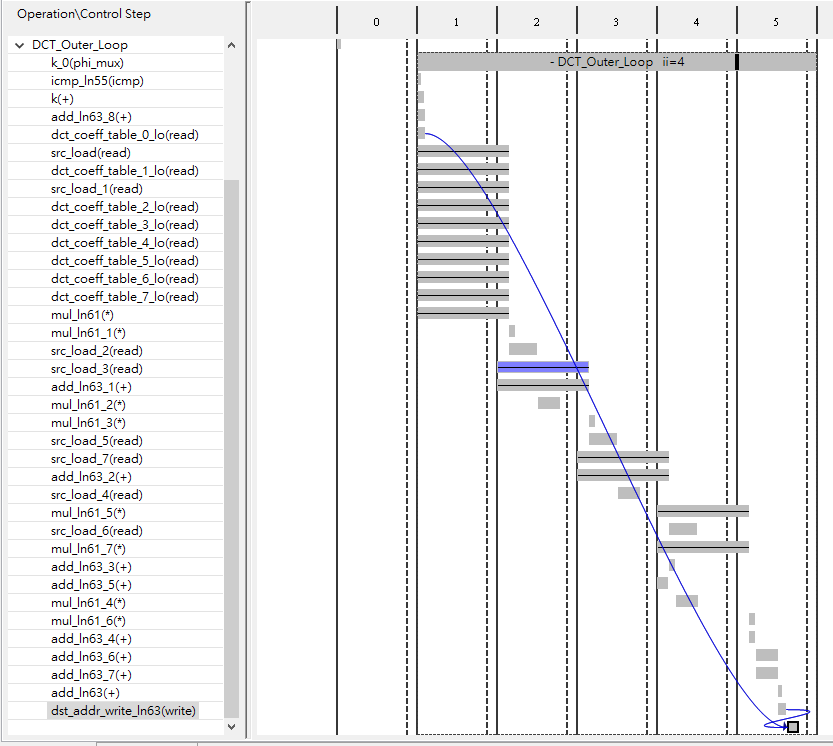


****

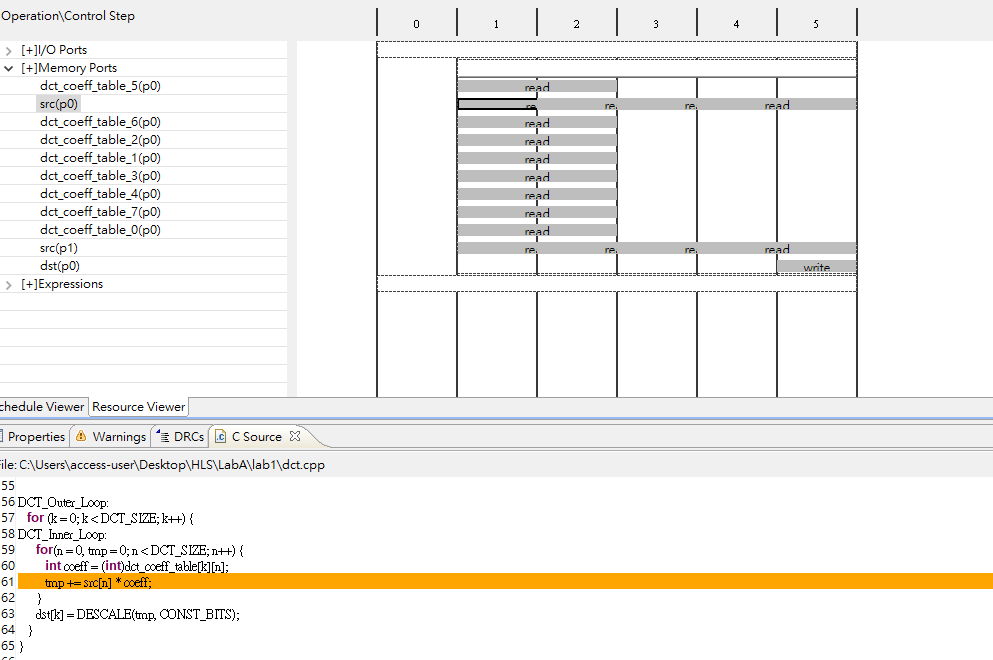
****

從Schedule Viewer中可以看到RD\_Loop從開始到結束(紅色箭頭)只花了近2個clock cycles，在Latency方面有顯著的改善，WR\_Loop同理

下圖為dct\_1d的流程圖

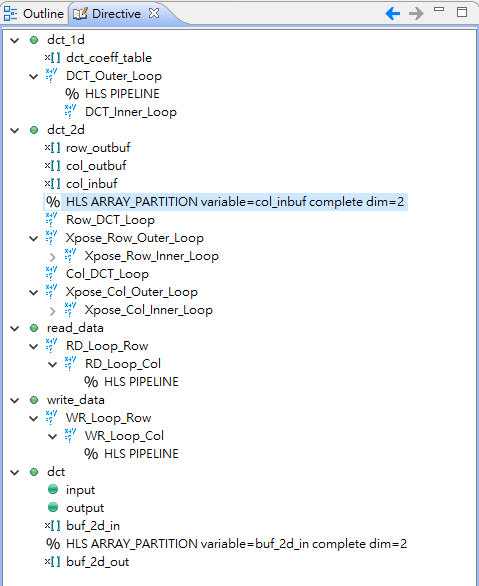


在Resource Viewer可以觀察在不同resources下是如何讀取memory data，以及對應的source code，read operation在第一個cycle generate the address for the block RAM，第二個cycle read the data，後面會介紹如何利用partition來做優化

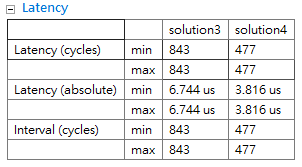


* **Partition Block RAMs**

下圖兩個input array(紅框)都是2-dimensional(8x8)的矩陣，使用預設的方式就會有一個single block RAM有64 elements，但我們可以partition second dimension，分別產生八個block RAMs，每個RAM只有一個row，這樣可以讓data平行做處理。

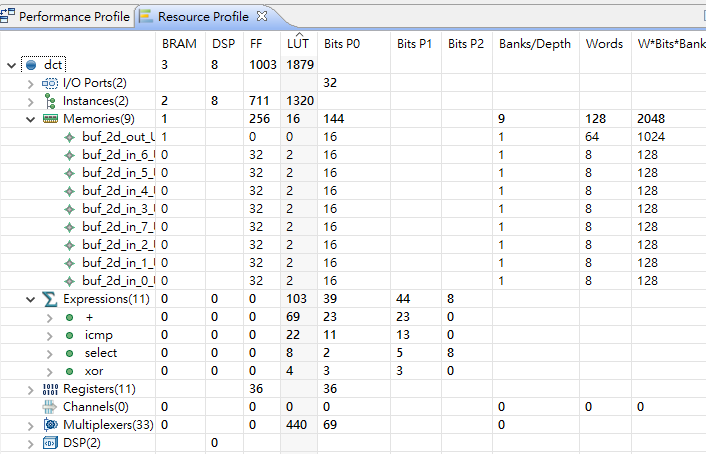


以下是solution3(pipeline)跟solution4(partition)的比較，因為改善了dct\_1d block中src讀取data的速度，latency降到477 cycles



下圖Resource Profile顯示出在不同架構下所使用到的不同資源，以dct為例:

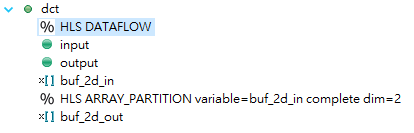
* 兩個I/O ports
* 兩個instances(sub-block)
* 九個memory，其中八個是partition切出來的
* 其中使用最多的logic是adders，共69個



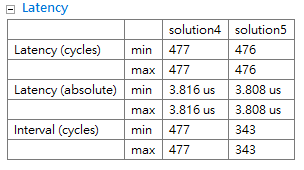
先在這裡做個小總結，從solution 1(baseline)的近2600 clock cycles經過flattening, pipeline, partition之後優化至477 clocks，接下來要專注在單一loop/function進行dataflow optimization，讓loops跟functions可以平行運算，再提升整體的performance。

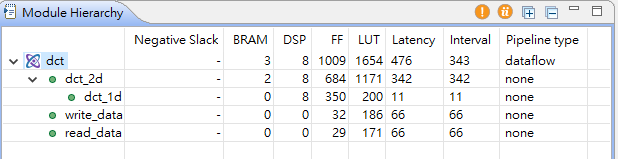
* **Apply Dataflow Optimization**

在dct底下加入dataflow的directive



比起solution4(partition)，dataflow optimization改善了Interval，雖然還是需要花476 clocks cycles才能產生output，但可以在343 clocks之後接收新的input





從上圖可以觀察出三件事:

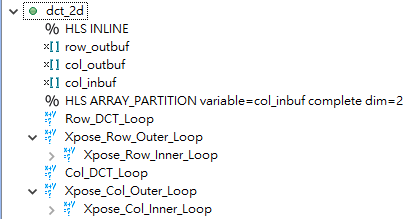
1. Dct整體的interval小於每個sub blocks latency的加總，代表blocks之間是有在平行運算的
2. Dct的interval幾乎被dct\_2d dominate，因此這個sub-block是關鍵的因素

將dct\_2d展開觀察會發現interval等於latency，表示其中並沒有達成parallel，以下列出三種優化的方式:

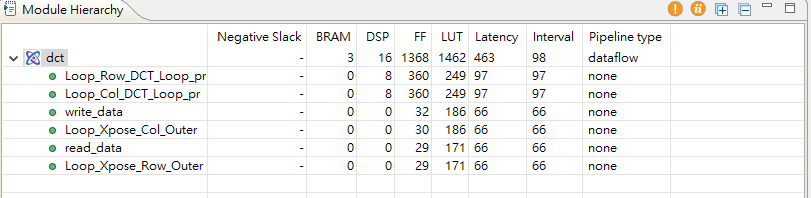
1. 將dct\_2d作pipeline，但這會讓所以loop unroll，有可能會讓performance下降
2. 將dct\_2d做dataflow optimization
3. Inline dct\_2d 函數，把dct\_2d裡面的loop拉到最高的hierarchy，讓其包含至top-level的dataflow optimization

* **Optimize the Hierarchy for Dataflow**

在這裡我們選擇使用上述的第三種優化方式: 將dct\_2d做inline



Synthesis後可看到系統inline了所有loop，同時進行top-level的dataflow optimization，將interval降至98 clock cycles



以下是co-sim的波型圖

