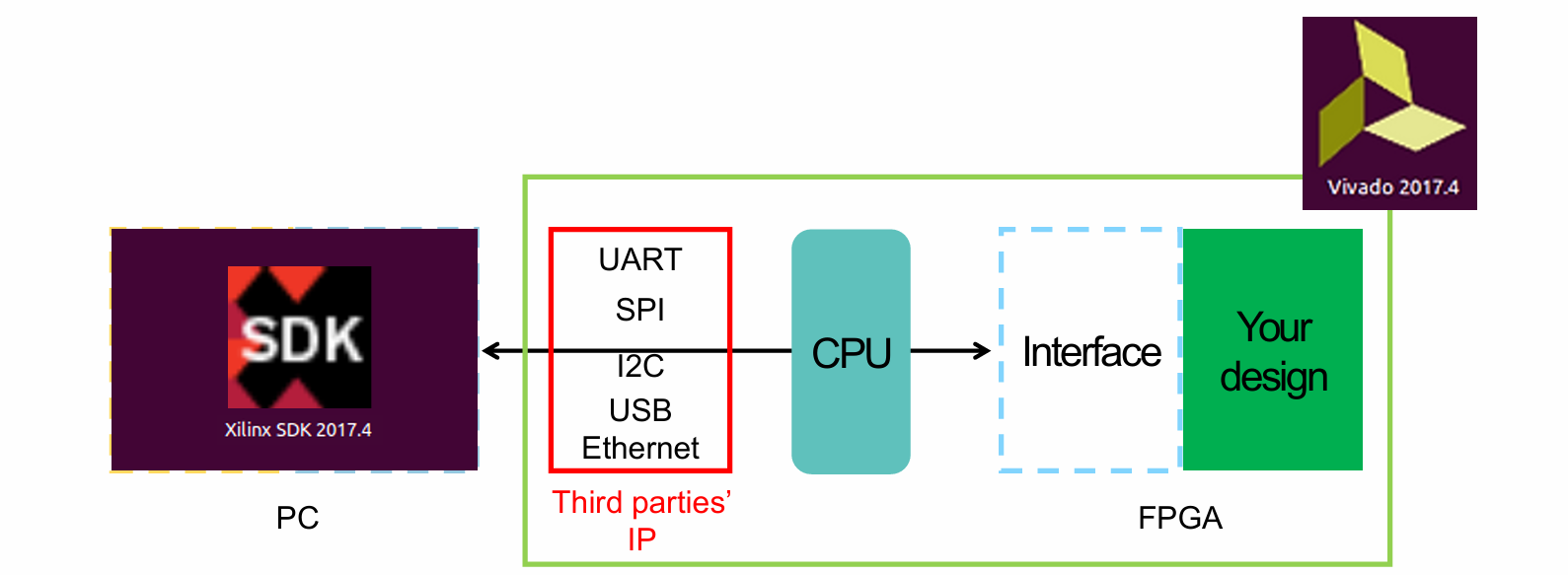
**Design of Chip Security - Spring 2024**

Lab03 Report

Name: 王品然

Student ID: 113063572

1. Overview



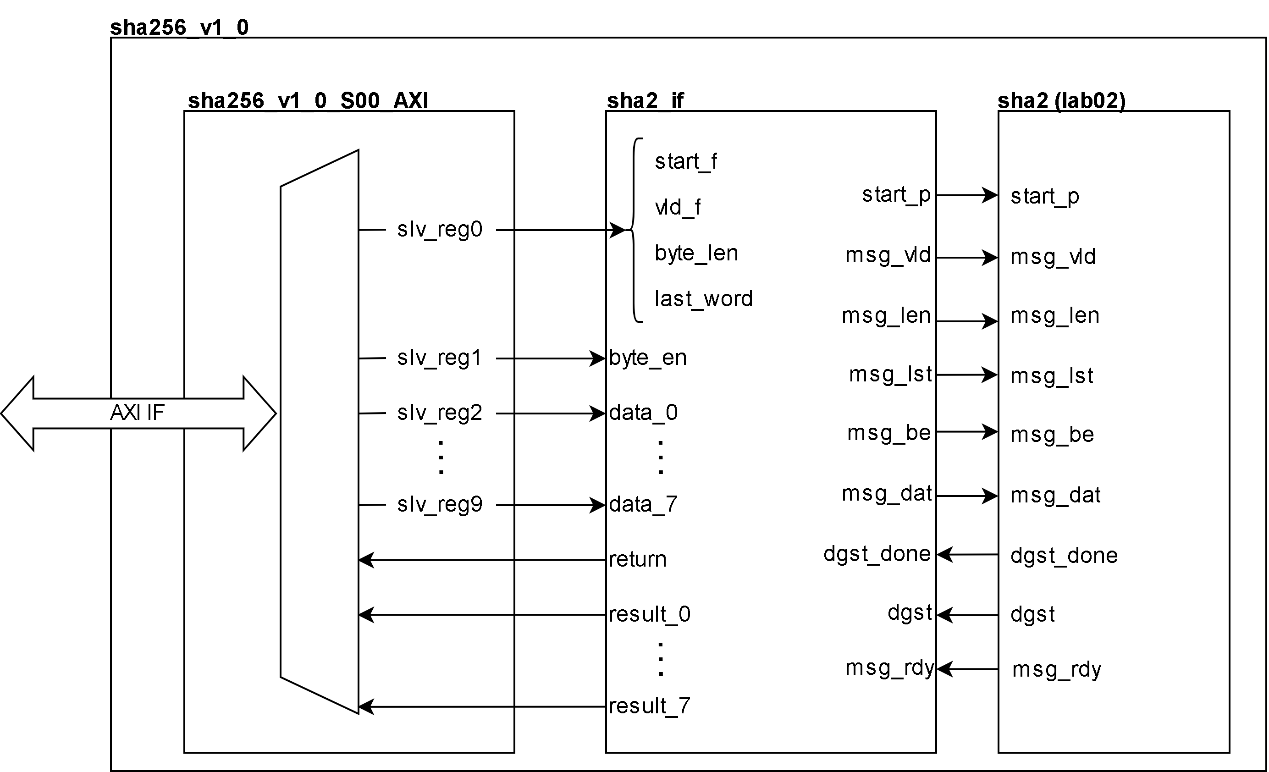
**Figure 1.** System-level block diagram for demonstration (adapted from Lecture 7 slides)

根據上圖以及lecture 7的介紹，為了完成FPGA prototyping，需要將先前設計完成的sha2.v模組透過AXI接口封裝為IP，與以MicroBlaze為核心的 CPU 系統整合，加入至助教提供的Block Design中，接著進行synthesis與implementation並產生bitstream燒入FPGA板中，接著在SDK編寫C語言，最終實現主機透過SDK軟體輸入訊息，由FPGA計算其SHA2-256雜湊值並回傳後，將結果顯示於terminal上。

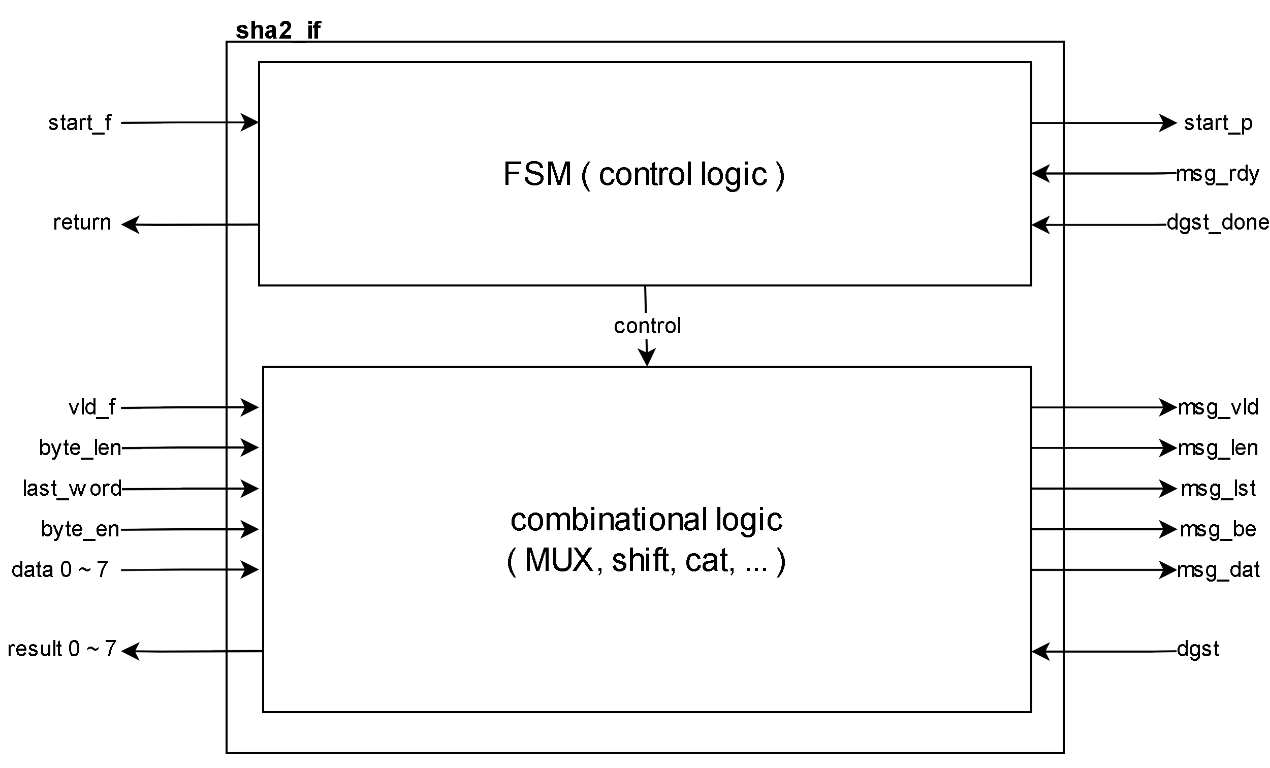
1. Pack sha2 into an IP

在AXI registers planning的階段，我初步的分配想法是說，將所有要輸入給sha2的data都在SDK中算好並存入slv\_reg，分配一個slv\_reg存放控制訊號(如start和valid)；兩個slv\_reg存放64-bit的message長度；一個slv\_reg存放標示是否是last word的訊號；一個slv\_reg存放標示byte是否enable的訊號；八個slv\_reg存放最多8個word的message，而從sha2 輸出到AXI的訊號就不多開register存放了。這樣的分配優點是可以明確分辨每個slv\_reg的功能，但會浪費許多沒用到的register，因此將原本需要64-bit的message長度，縮減到6-bit的寬度，僅用來處存最多32個byte的輸入長度(單位：byte)，並將start, valid, last word, message length合起來放到同一個slv\_reg中。但仍然有一個問題是，C program無法產出start\_p這樣的pulse訊號，也很難產生出msg\_vld這種需要即時與sha2互動並遵守protocol的訊號，所以需要新增一個module將C program 傳過來的start和valid的level訊號轉換成為可以傳給sha2的訊號。

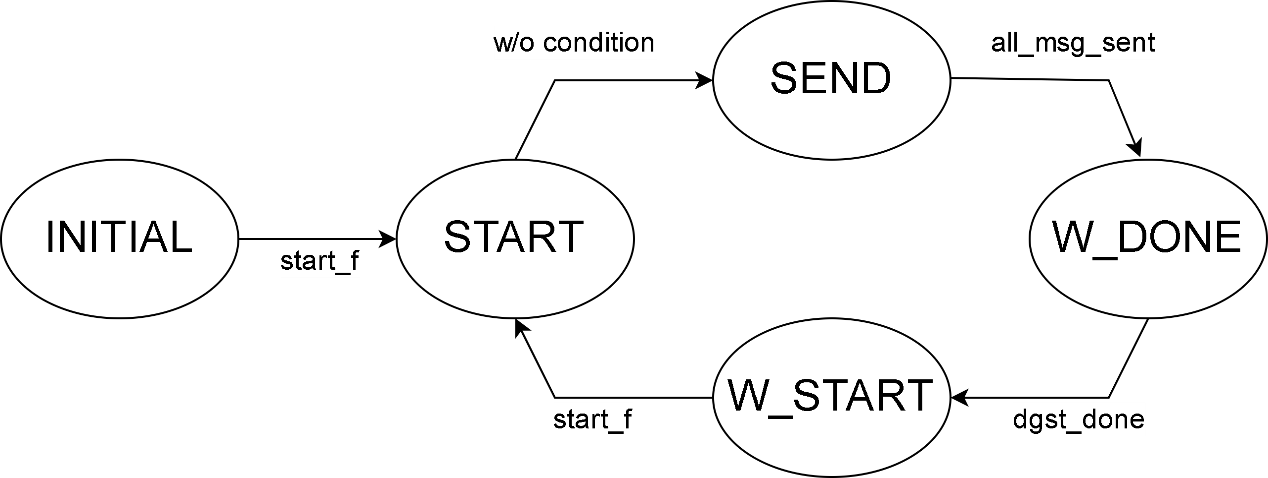
因此我在sha256\_v1\_0\_S00\_AXI (以下簡稱S00\_AXI)與sha2之間新增一個interface module，使其負責接收並處理SDK寫入slv\_reg中的資料，再將其傳給sha2；負責接收並處理sha2回傳的資料，再將其傳給S00\_AXI；最後還要滿足sha2 module的I/O protocol。根據上述這些AXI registers planning的想法以及interface所需具備的功能，以下分別是sha256\_v1\_0的module structure和我所設計的sha2\_if (即為上述的interface)的module structure及其FSM：



**Figure 2.** sha256\_v1\_0 module structure



**Figure 3.** sha2\_if module structure



**Figure 4.** sha2\_if module FSM

**訊號名稱說明：**

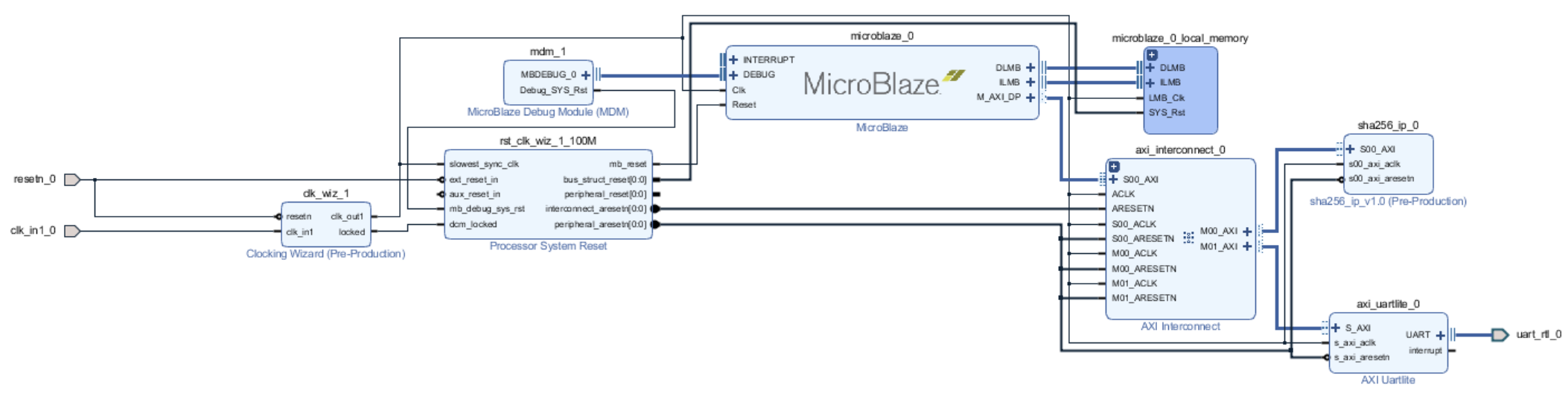
1. start\_f: 由程式寫入的1-bit訊號，代表這輪雜湊值運算要開始了。
2. vld\_f: 由程式寫入的1-bit訊號，代表要傳給sha2的資料都已寫入slv\_reg。
3. byte\_len: 由程式寫入6-bit寬度的資料，表示這次的message有幾個byte。
4. last\_word: 由程式寫入8-bit寬度的資料，表示每個word是否為last word，與start\_f, vld\_f, byte\_len，一起組成slv\_reg0的前16 bit訊號 (參考Figure 2.)。
5. byte\_en: 由程式寫入32-bit寬度的資料，分別表示每個byte是否為有效。
6. data 0 ~ 7: 由程式寫入的8個32-bit寬度的資料，分別就是message的8個word。
7. return: 由sha2\_if給到AXI的32-bit訊號，其中return[30]代表sha2\_if已完成傳送資料給sha2；return[31]代表sha2\_if已準備好digest。
8. result 0 ~ 7: 由sha2\_if給到AXI的8個32-bit訊號，分別放入digest的256 bit。

**sha2\_if運作說明：**

sha2\_if在reset後會在INITIAL狀態等待S00\_AXI傳來start\_f，作為進入START狀態的訊號，在START狀態下，sha2\_if會拉高start\_p以及將byte\_len組成msg\_len並傳給sha2，START會無條件進入SEND狀態，換言之，start\_p只會被拉高1 cycle以符合protocol。在SEND狀態下，若vld\_f = 1，則將msg\_vld拉起，代表已經準備好要傳送的msg\_dat, msg\_lst, msg\_be，之後每當收到msg\_rdy = 1，就代表已完成傳送一組msg\_dat, msg\_lst, msg\_be，使用counter計算以傳送幾組，當傳完該次message的所有word後 (word數量由byte\_len經過組合邏輯後得到)，代表Figure 4.的all\_msg\_sent，進入W\_DONE狀態。在W\_DONE狀態下，sha2\_if會將return的中一個bit (return[30]) 拉起，告訴C program：「sha2\_if已完成傳送資料，可以將start\_f和vld\_f拉low。」並且會在這個state持續等待直到dgst\_done = 1，進入W\_START狀態。在W\_START狀態下會將return的中一個bit (return[31]) 拉起，告訴C program：「sha2\_if已準備好digest。」並且會在這個state持續等待直到start\_f = 1，進入START狀態，開啟下一個循環。值得注意的是，在我的設計中，無論狀態為何，dgst[256:0]都assign給result 0 ~ 7。

1. Replace default IP with my sha256 IP

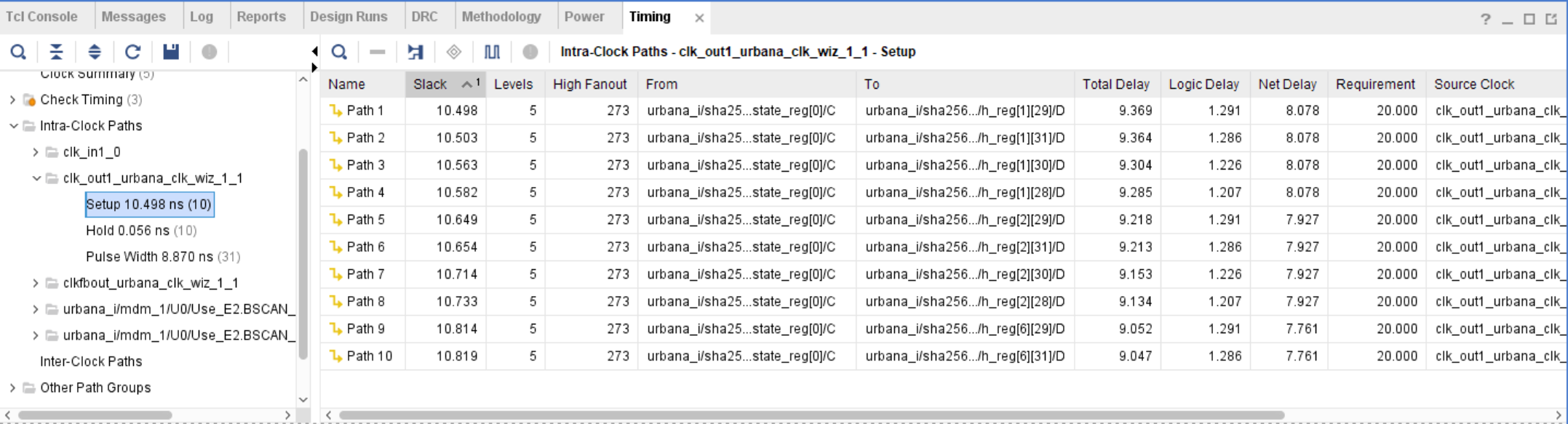
在上一步完成sha256\_v1\_0的設計後，將sha2與AXI接口封裝為IP，接下來將此IP放入助教提供的CPU system取代原本在裡面的舊IP，如下圖：



**Figure 5.** CPU system block diagram with my IP

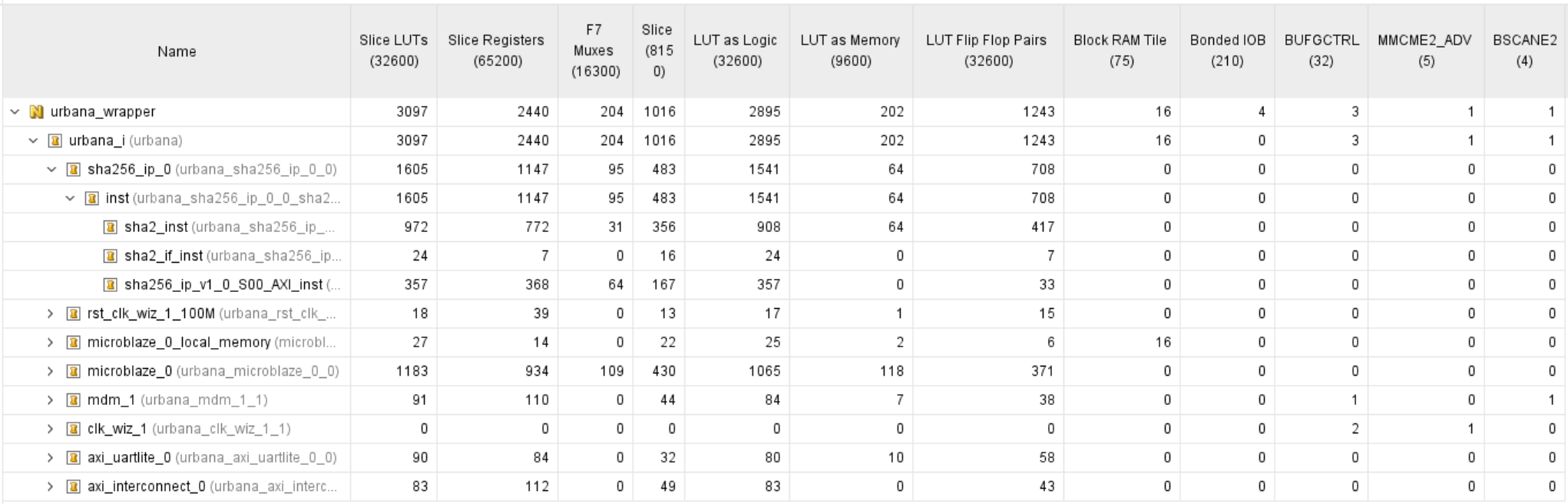
1. Run synthesis and implementation

接下來跑synthesis與implementation，並檢查implement後的timing summary中的setup time是否仍有slack，如下圖：

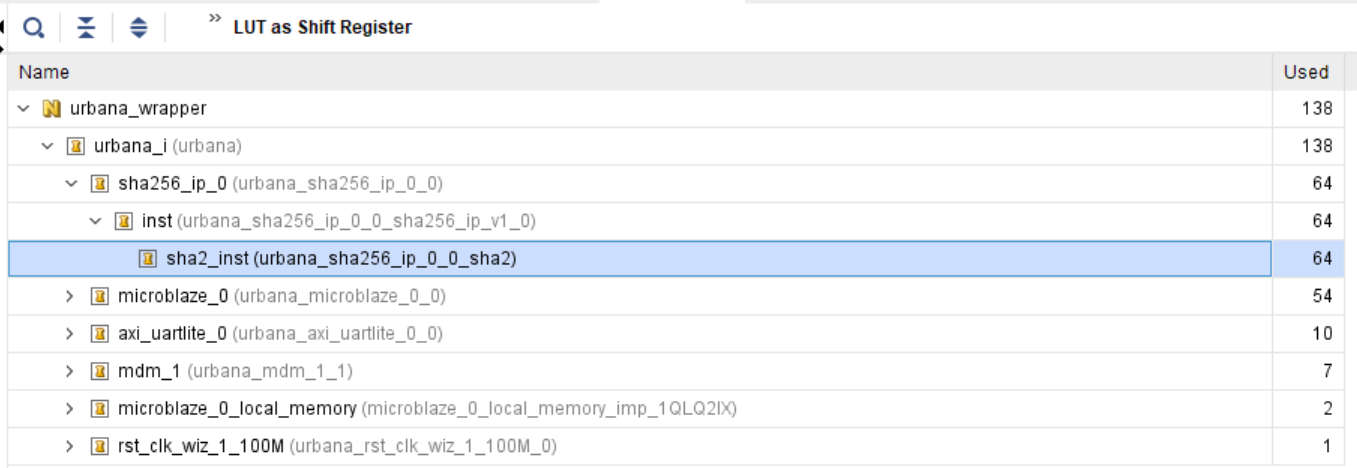


**Figure 6.** setup time report

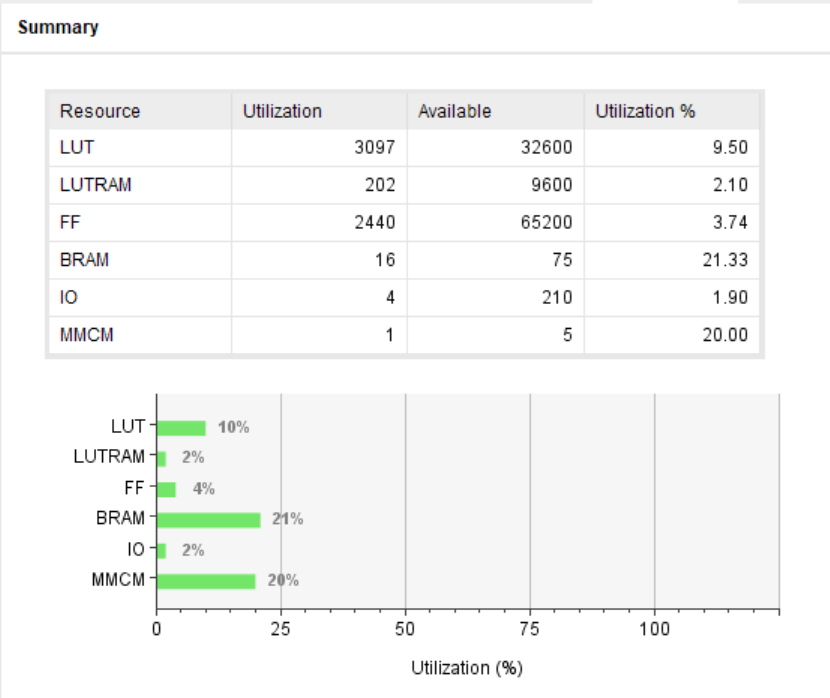
接著檢查utilization report，如下：



**Figure 7.** utilization hierarchy table



**Figure 8.** utilization “LUT as Shift Register” table

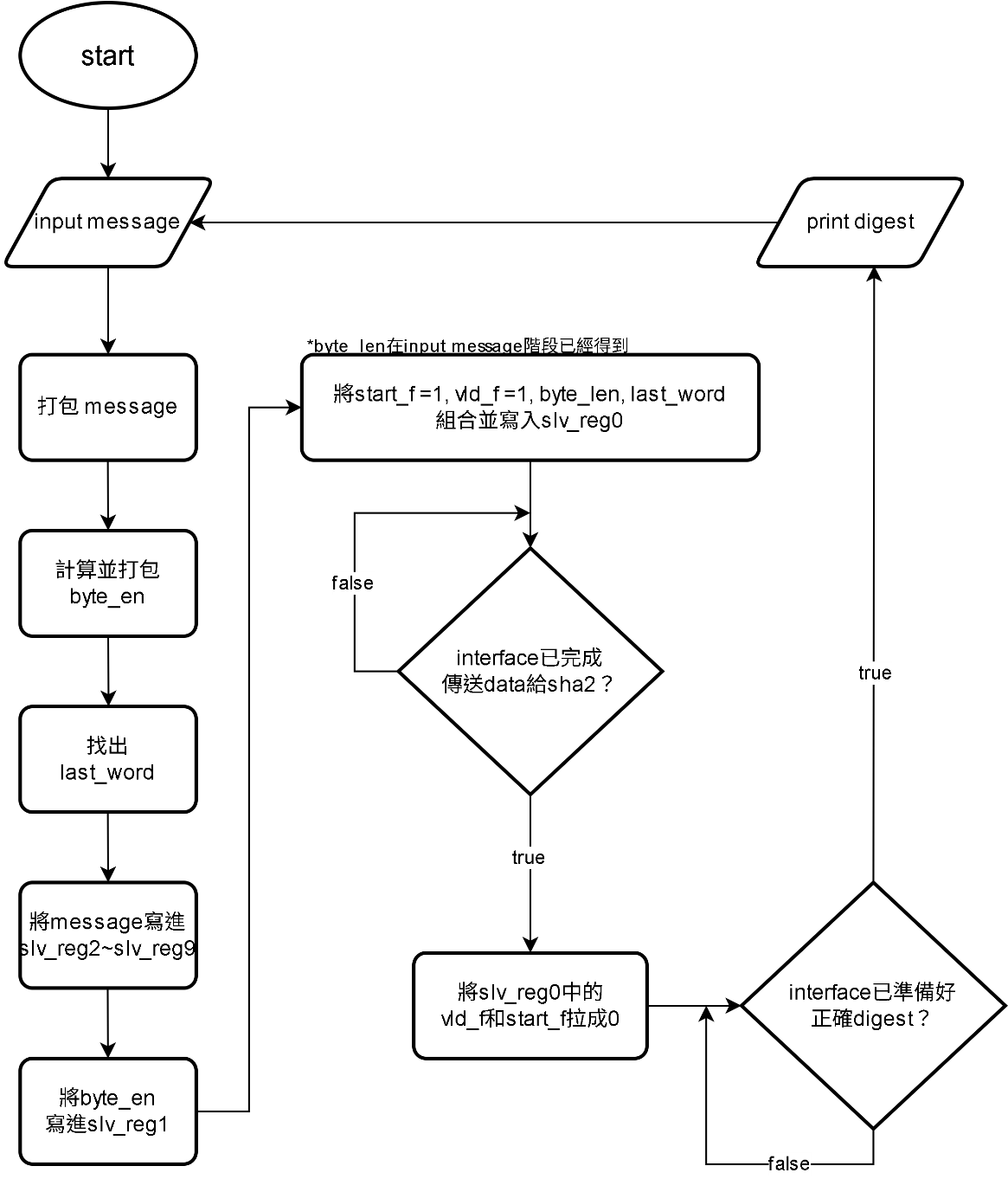


**Figure 9.** utilization summary table

從Figure 7. 中可以看到sha2\_if\_inst使用了7顆DFF符合預期，分別是3-bit reg負責存放五個state，以及4-bit reg負責存放已經傳了幾個word。也可以看到sha2\_inst使用了772顆DFF，大幅低於預期的1098，可能是某些register被視作shift register並用LUT實現，如Figure 8. 所示，並未用到DFF。最後看到sha256\_ip\_v1\_0\_S00\_AXI\_inst使用了368顆DFF，大致符合預期，因為我使用了10個slv\_reg，因此占用320顆DFF，剩下的48顆可能用在存放address或其他AXI的data。從Figure 9. 中可以看到FPGA還有很多resource沒有被用到，顯示這次的design對這張板子來說還有餘裕。

1. C program in SDK

根據前面設計的sha2\_if，C program的主要任務在於製作使用者友善的操作介面、處理並提供sha2\_if資料以及接收sha2\_if的回傳資料。因此設計C code有如下的操作流程，以flow chart呈現，並以文字加以說明：



**Figure 10.** Flow chart of my C code

**說明：**

1. input message：

印出“message: ”，提示使用者輸入訊息，逐byte接收使用者輸入的message，並時時印出接收到的字母讓使用者看到，邊接收邊計算共有幾個byte，並將此數量記下當作byte\_len，接收message直到使用者按下enter或是已輸入32 byte。完成input message後terminal要換行。

1. 打包message：

將接收到的message分裝成8個word，不滿8個word的部分填入0。

1. 計算並打包byte\_en：

計算每個word分別對應的msg\_be應該要是多少，並將其彙整為32-bit的byte\_en訊號。

1. 找出last\_word：

計算每個word分別對應的msg\_lst，並將此8-bit訊號稱為last\_word。

1. 將message寫進slv\_reg2 ~ slv\_reg9

使用Xil\_Out32將打包好的8個message word寫入AXI的slv\_reg2 ~ slv\_reg9。

1. 將byte\_en寫進slv\_reg1

使用Xil\_Out32將打包好的byte\_en寫入AXI的slv\_reg1。

1. 將start\_f =1, vld\_f =1, byte\_len, last\_word組合並寫入slv\_reg0

使用Xil\_Out32將start\_f =1, vld\_f =1, byte\_len, last\_word組合並寫入slv\_reg0。

1. interface已完成傳送data給sha2？

使用Xil\_In32讀取return[30]的訊號 (代表sha2\_if已進入W\_DONE狀態)，作為判斷條件，為1時，繼續進行程式，否則繼續等待。

1. 將slv\_reg0中的vld\_f, start\_f拉成0

使用Xil\_Out32將slv\_reg0中代表vld\_f和start\_f的bit改成0。

1. interface已準備好正確digest？

使用Xil\_In32讀取return[31]的訊號 (代表sha2\_if已進入W\_START狀態)，作為判斷條件，為1時，繼續進行程式，否則繼續等待。

1. print digest

印出“Digest: ”並換行，分段讀取並接續印出result 0 ~ 7的值，並在印到第四個result時換行，讓使用者方便閱讀。最後再換一次行並回到input message。

1. 心得

這次的lab對我來說是三次裡面最難的一個，因為初次接觸FPGA加上沒有寫過C code，因此有很多陌生的領域。可能是因為有很多沒聽過的名詞出現，所以在教學FPGA的課堂上也沒有學得很清楚，但還好在回去複習lecture 7的講義以及跟著步驟實作一次後，有大致清楚整個流程，再加上助教時間來問助教我不清楚的地方，非常有幫助。後來做作業的過程中遇到最大的麻煩其實還是對於整個流程沒有到很熟悉，導致第一次跑的時候，digest出不來，用C code print出每個步驟後，找到是卡在C program一直沒有收到sha2\_if回傳的訊號，因此一度懷疑是sha2\_if的設計有問題，因此還大費周章在寫了一個testbench去驗sha2\_if + sha2的功能是否正確，結果發現功能是正確的，後來又重新下載nthu\_sys\_lcm\_only再重新generate bitstream並燒錄FPGA再跑一次，digest就印出來了，但是也發現endian有錯，在改過endian後，就成功了。