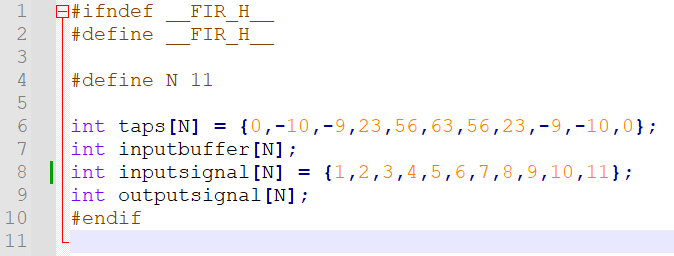
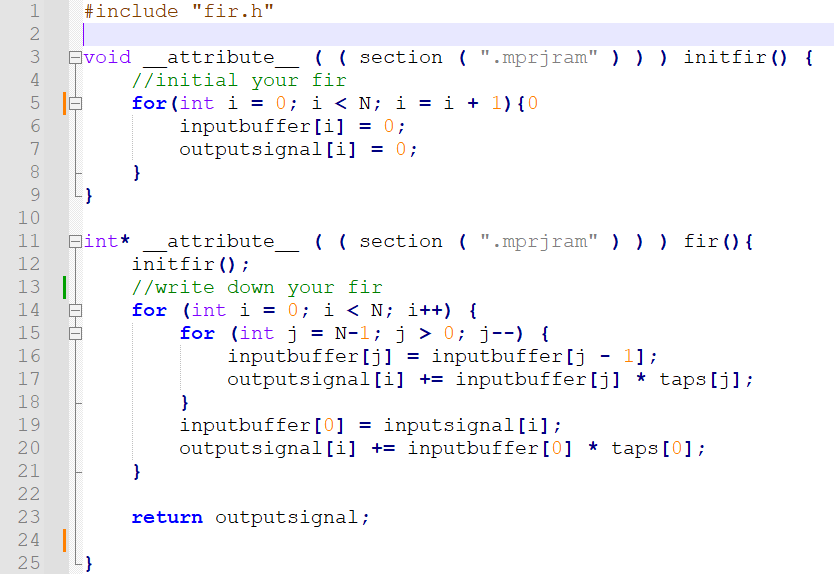
SOC Design Lab4-1

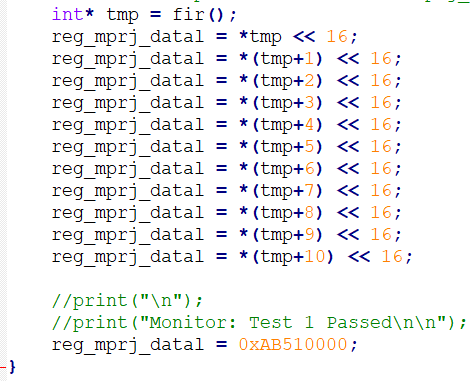
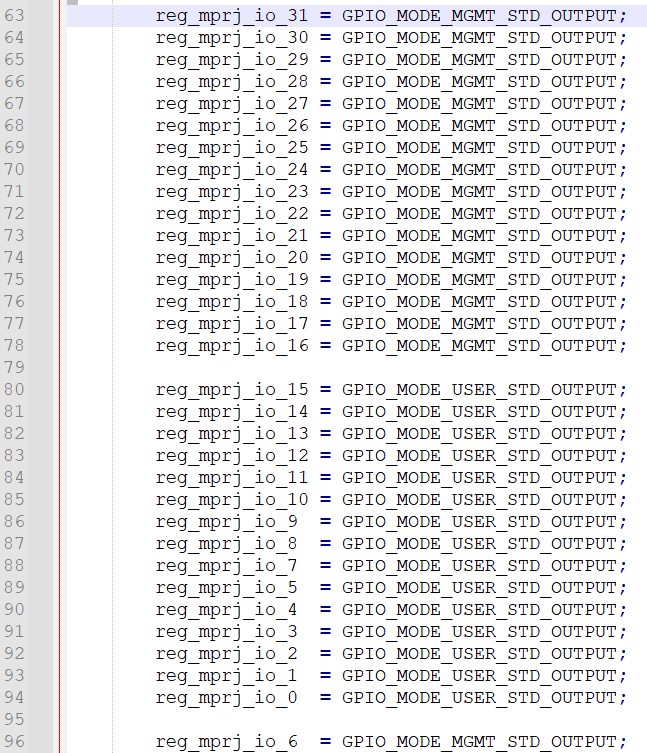
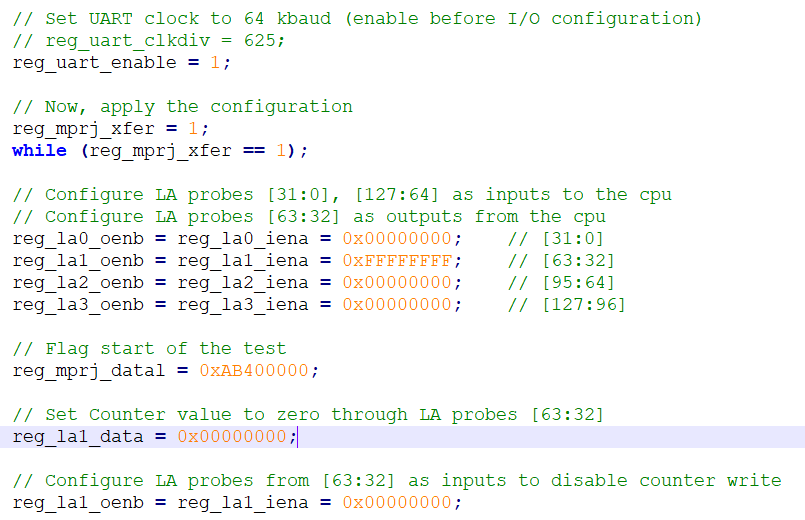
* Explanation of your firmware code：

Firmware code裡再fir.h中定義了fir的計算中所使用到的參數，以及變數空間，在之後的assembly code中找到再fir head中看到這部分的宣告，接下來fir.c中首先會先對輸入暫存以及輸出暫存空間進行初始話的動作，那這部分也會再assembly code中的initfir找到，而fir的計算則是放在fir() function中，我們利用了兩層的for迴圈，其中內層利用shift register的概念去將inputbuffer的值去向後推移，同時將推移完的結果與coefficient相乘，那外層部分則是算出每一個時間點的outputsignal，將新來的X[i]送進inputbuffer，同時也計算x[i]tap[0]至outputsignal完成fir運算，最後再將輸出return回去。



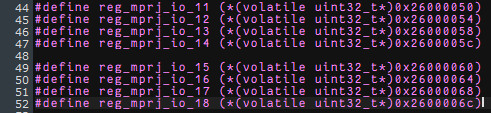
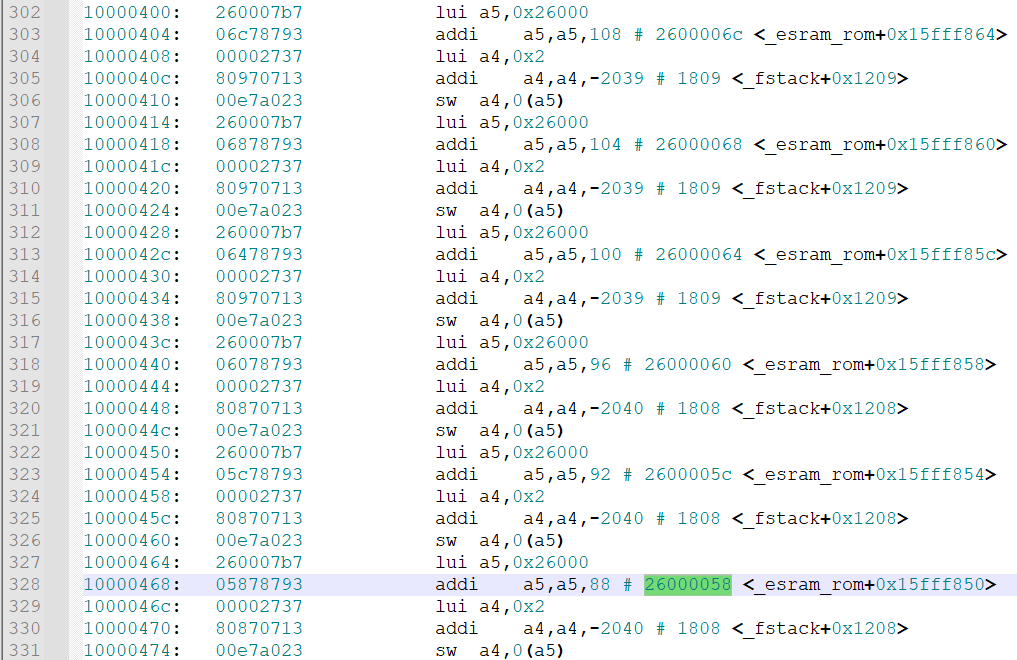


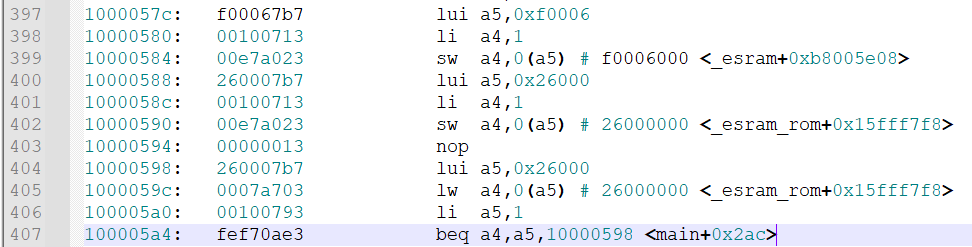
在counter\_la\_fir.c中則是定義了完整的firmware code的流程，也就是說，上面的fir.c只是其中的一小部分功能而已，其中先定義了mprj io，將上半部分給了management core、下半部分給了User project，接著在傳xfer flag去確認CPU有執行剛才的設定接著在對LA bus做些initial，以及mprj\_datal廣播0xAB40，讓  
3我們在testbench中能抓到這個開始訊號。最後在呼叫fir function將結果傳回到tmp中，並藉由mprj上半部的IO port傳回management core



* + How does it execute a multiplication in assembly code：

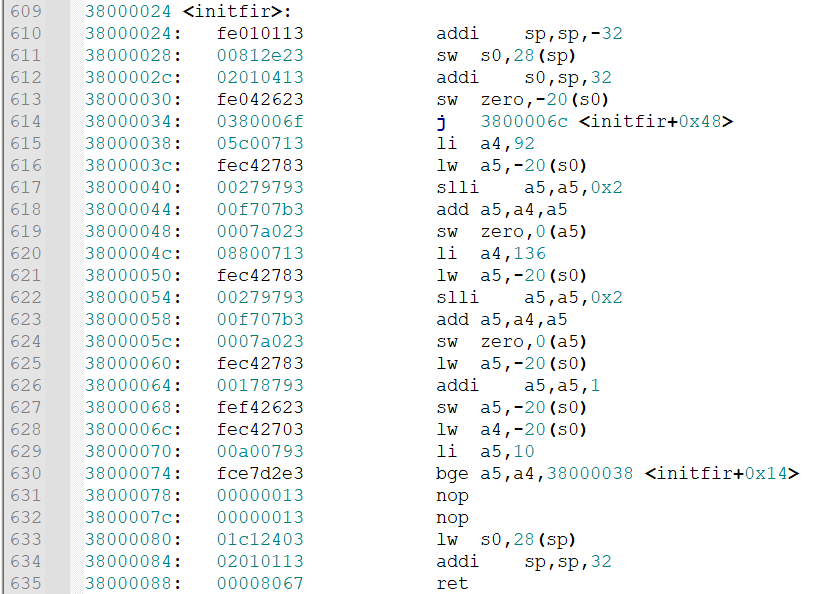
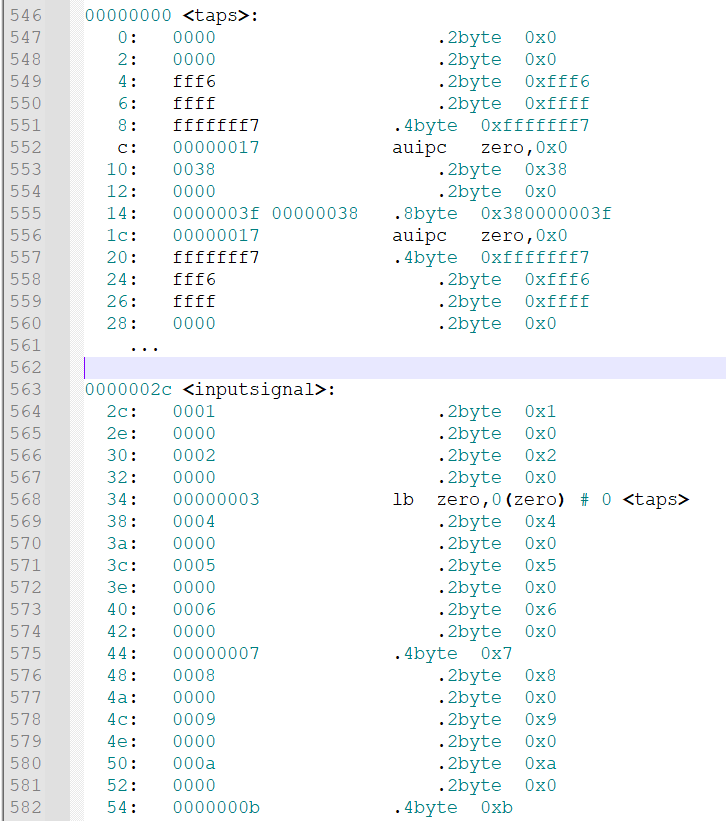
在main的assembly code中首先是定義MPRJ IO port，在Caravel.h中定義了mprj的實際address map，並依照counter\_la\_fir.c將前半部分給management core後半部給user project，分別對應到0x1809、0x1808指令，因此在main可以看到他將這些address給予各自的GPIO指令，assembly code都是用a5去計算offset、address以及存入GPIO指令。



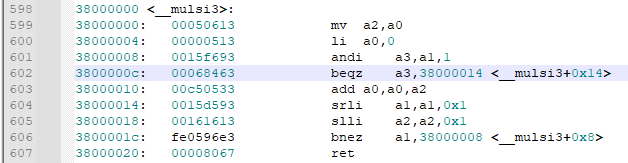
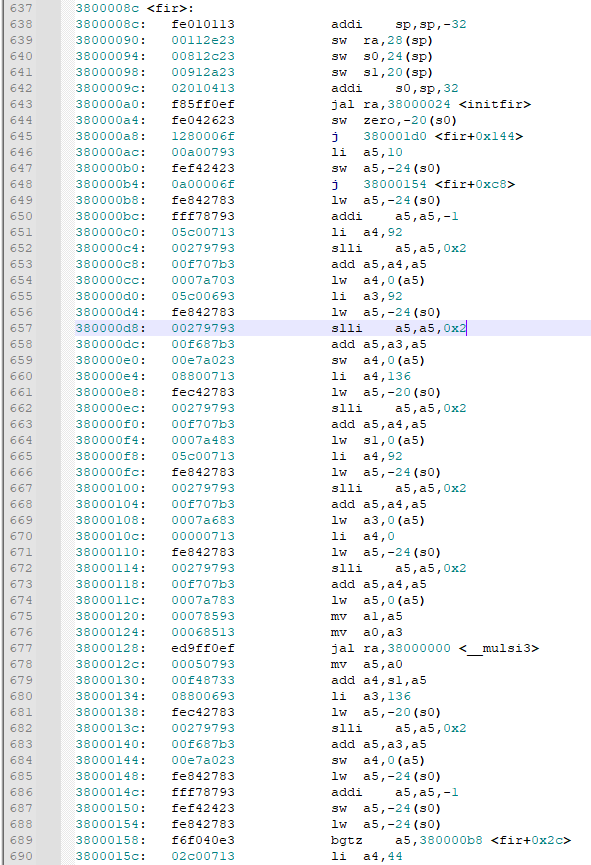
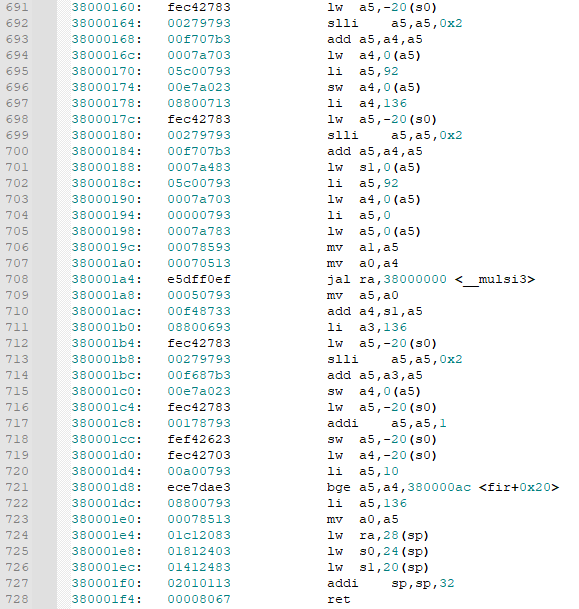


接著將xfer利用a4設成1，而404~407行是用while，load xfer signal至a5確認他有沒有回傳1

接著是fir的assembly code，首先在main中會先跳到fir，而在fir中會先跳去initfir，去做initial的動作這些動作都是由0x38開頭的，而fir.h中的那些變數宣告則是在address 0x0000~0x0054，也就分配在diff的memory中，和存在mprjram的0x38000000不同位置。



而乘法由\_mulsi3來實現，首先將a0和a1將參數給進來，在把被乘數(a0)的值送進a2中，並將a0歸零來當作累加的位置，a1會一直確認LSB的值=1，有就累加被乘數，=0就不累加，並shift a1與a2的值，直到a1的值=0，他是用shift right a1與 a1 and 0x1來確定是否要乘積，shift left a2來進行乘法進位，最後再累加起來的拆解方法完成乘法。



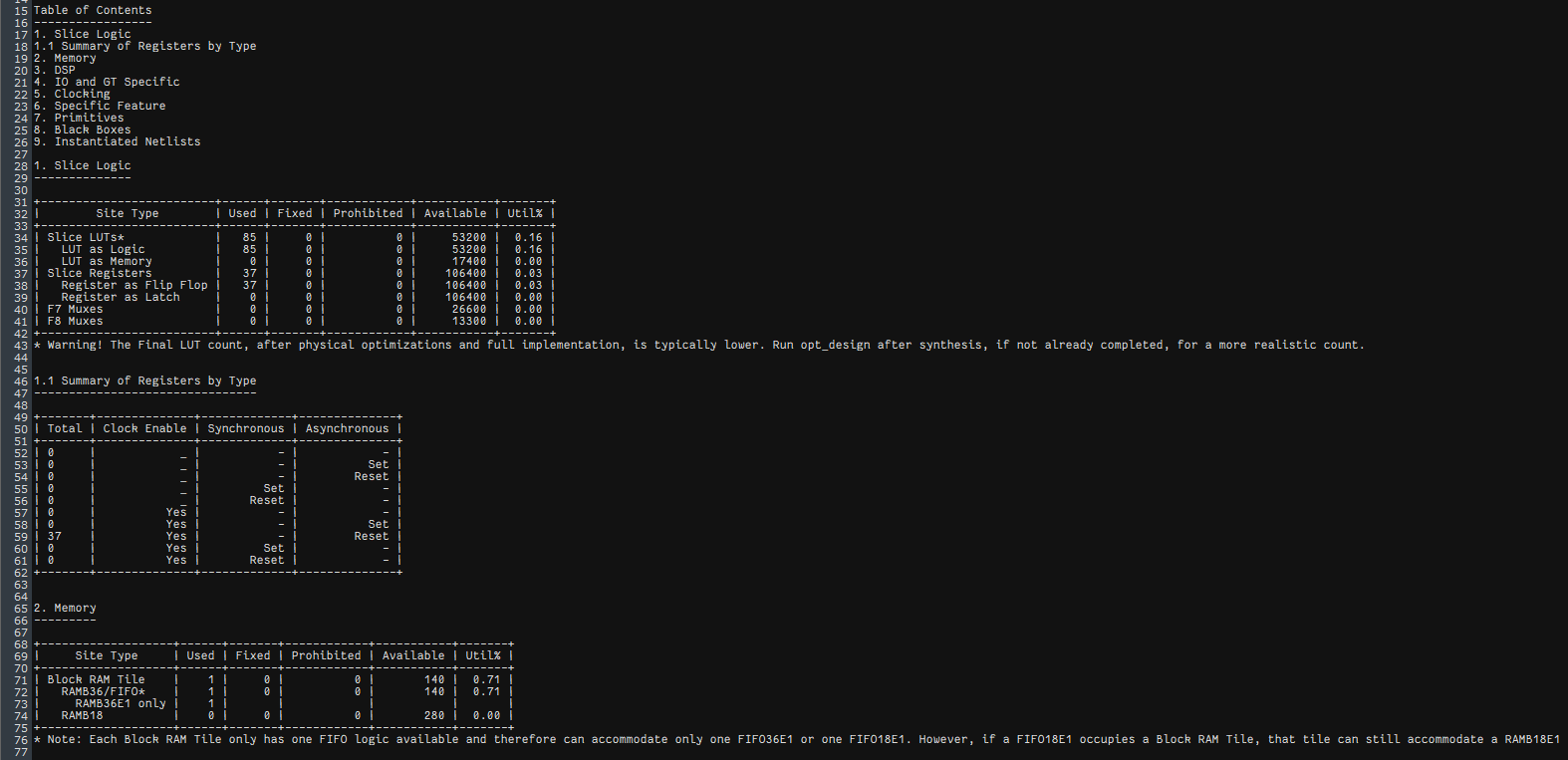
* + What address allocate for user project and how many space is required to allocate to firmware code：

Fir部分(包含\_mulsi3、initialfir、fir)這三個共佔了0x38000000~0x380001f7 = 504 bytes

Main部分：從0x100002ec~100007a8 = 1216 bytes

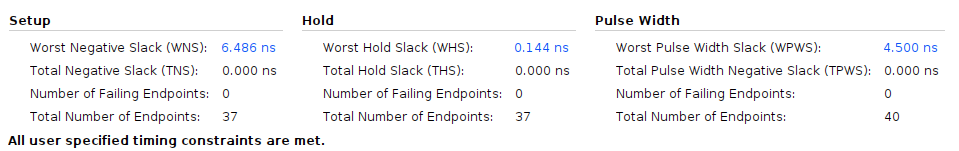
Total space for firmware code is 1216 + 504 = 1720 bytes

* Resource usage: including FF, LUT, BRAM

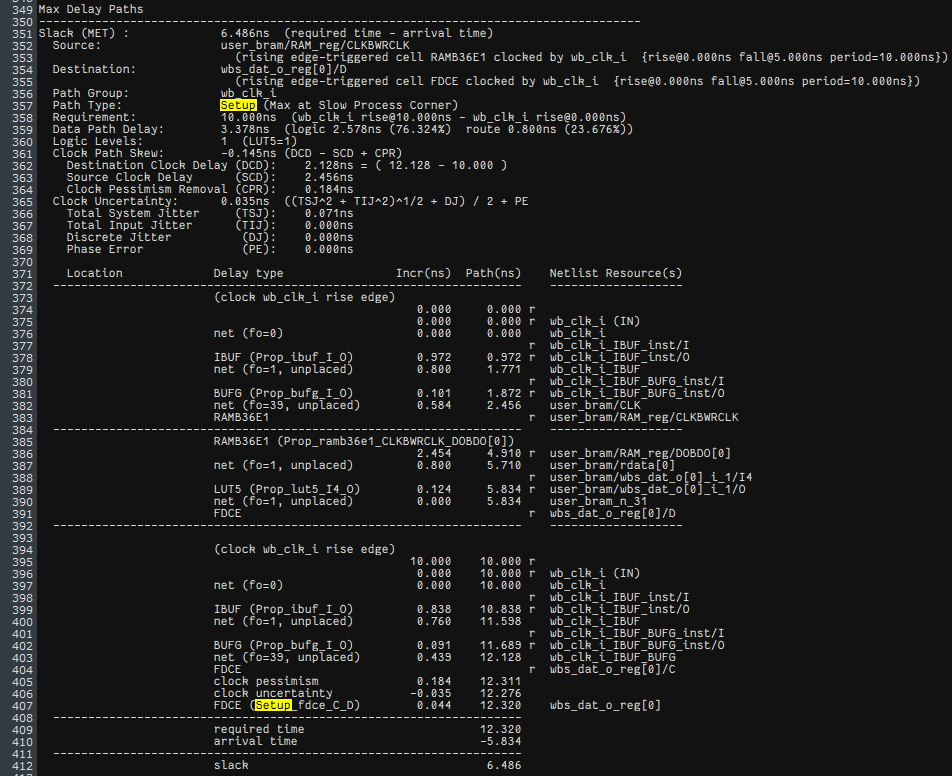
本次合成只包含bram.v & user\_proj\_example.counter.v 兩個檔案

* Timing Report：

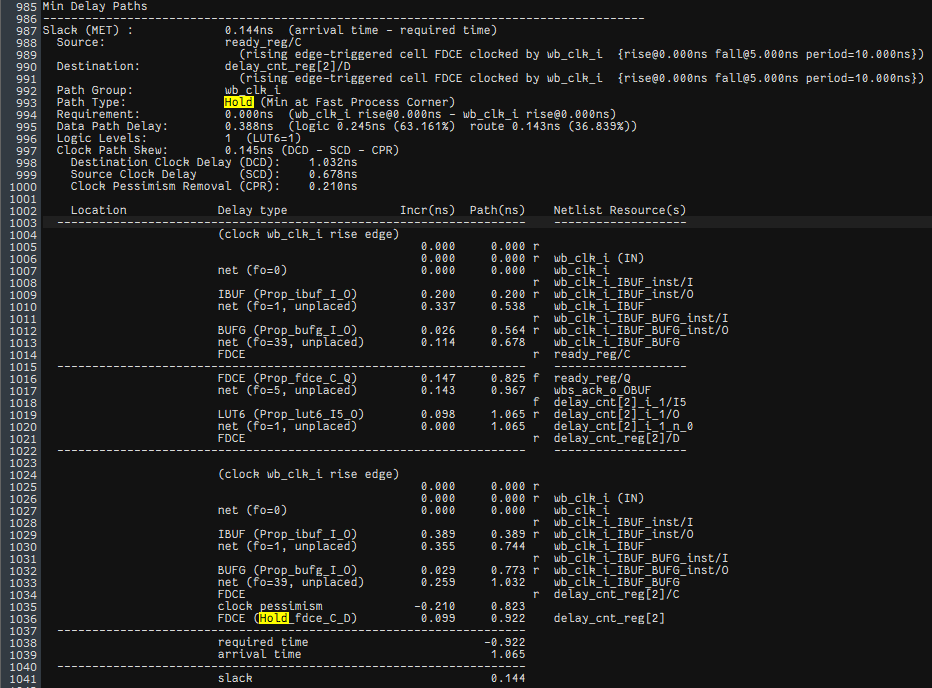
本次實驗合成clock time為10ns



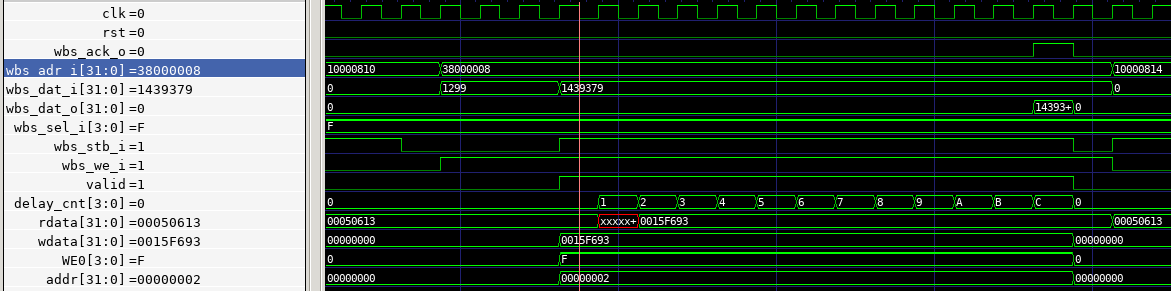
* + Setup longest timing path –6.486ns (require – arrival time) ：



* + Hold min timing path：



* Simulation Waveform：
  + Interface between BRAM and wishbone：

當stb\_i與sel\_i都為high時，才會開啟Bram與WB的通道，並由we\_i=1去決定是要寫入值到Bram，或是(=0)讀取Bram，那他會藉由WE0 signal傳至Bram，而Bram的addr則是將offset 0x38000000去掉，並右移2位，以word line形式儲存，並延遲DELAYS+2的cycle後再輸出wb\_ack\_0以模擬memory的低速下的workload。

* + FSM：

當stb\_i與sel\_i都為high，並且wb\_adr\_i是0x38開頭的Address，valid才會為一，並啟動delay\_cnt計數，直到計數至DELAYS+2後才將wb\_ack\_0拉為high已表示完成整個BRAM and wishbone的資料交換。