SoC Lab4-1

組別: 第十六組　成員:周聖平、蔡以心、張煒侖

一、 Explanation of your firmware code

- How does it execute a multiplication in assembly code

Assembly Code

CPU 在開始執行 myprj 的時候，OS 會先去解析 spiflash 的.hex 檔上的檔案的 Header 和 section，以了解程式的入口點。

接下來，CPU 從程式的入口點開始執行。它會逐一執行執行程式的機器碼指令，這些指令會依序儲存在記憶體中。

在 Lab4-1 中，記憶體就會是 BRAM，而 user_proj_example.counter 則會負責去做 CPU 與 BRAM 的溝通橋樑。

等待 CPU 將指令寫入記憶體以後，就會去讀取 BRAM 的資料去執行 assembly code，也就是 counter_la_fir.c 編譯後的組語。

counter_la_fir.c Implementation

FIR 是輸入訊號跟 FIR 係數進行摺積，公式可以表示為

$y[n] = \sum_{k=0}^{10} x[n-k] * b_k$。

$y[0] = b_0 * x[0]$
$y[1] = b_0 * x[1] + b_1 * x[0]$
…

可以用兩層的 for 迴圈實現，外層的 for 迴圈是計算第 i 個輸出 y[i]，內層的 for 迴圈是將各項 b_j *x[i-j]累加起來。

```
for(i=0; i<N; i++){
   for(j=0; j<=i; j++){
        outputsignal[i] = outputsignal[i] + inputsignal[i-j] * taps[j];
   }
}
```

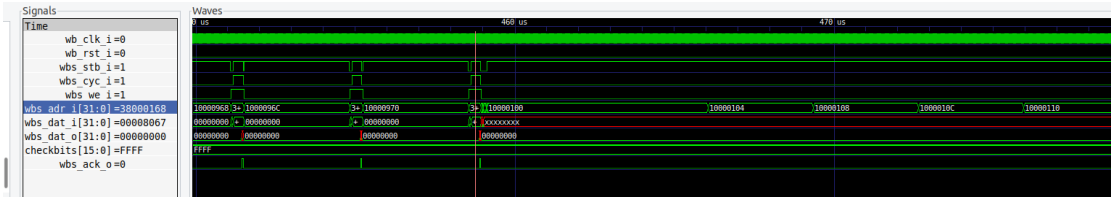
- What address allocate for user project and how many space is required to allocate to firmware code

MAX

```
11 MEMORY {
12         vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
13         dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
14         dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
15         flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
16         mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
17         mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
18         hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
19         csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
20 }
```

Bram 起始位置是 0x38000000，長度為 0x00400000 = 4194304 bytes = 16384(2^14)words，在 BRAM 中，最多可以用 2^14(16384)個 word 來去儲存 firmware code。

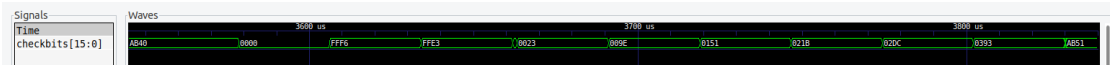
REALISTIC



從波形中可以看到 CPU 在寫入 BRAM 時，它的 wbs_adr_i 最多到 0x38000168，其中 0x168 = 360 bytes = 90 words，這就代表在 BRAM 中，最少需要 2^7(128)個 word 來去儲存 firmware code，因此 BRAM 的 N 需要 >= 7。

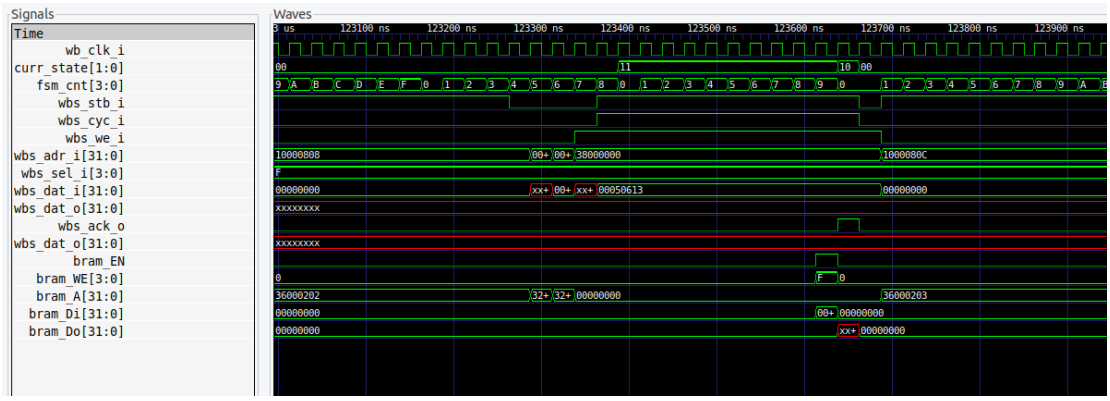
二、Interface between BRAM and wishbone

- Waveform from xsim

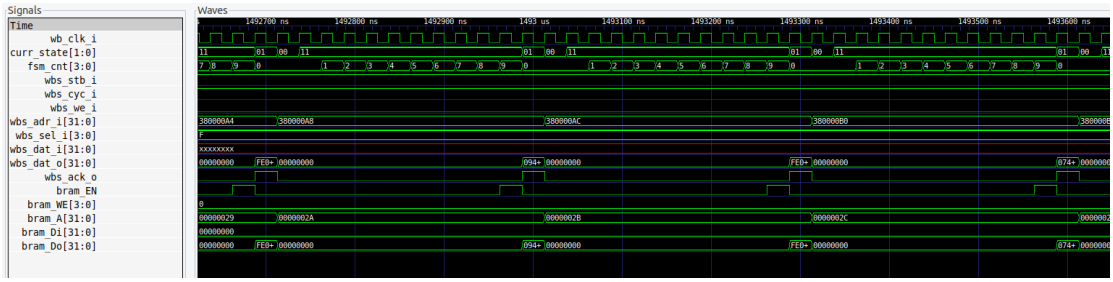


- FSM

WRITE



READ



FSM 共有四個 State，RECV(00)、RD(01)、WR(10)、DELAY(11)。

RECV： 當 RECV 接收到 wbs_stb_i && wbs_cyc_i 時，就會進入到 DELAY。

DELAY: DELAY 狀態是為了要去滿足 Parameter Delay，當 DELAY 到最後一個 clock 時，就會去 Access BRAM，也就是去對 BRAM 做讀寫的動作。而接下來，將會由 wbs_we_i 來決定是要跳去 RD 還是 WR。

RD: RD 狀態會將 bram_Do 輸出到 wbs_dat_o，並將 wbs_ack_o 拉為 H，接下來，就會跳回 RECV 等待下個 Request。

WR: 進到 WR 狀態時，會將 wbs_ack_o 拉為 H，由於在 DELAY 狀態的最後一個 clock 就會去完成寫入 BRAM 的動作，因此接下來，只要跳回 RECV 等待下個 Request 就好。

三、 Synthesis report

LUT: 50　Register: 6

```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   50 |     0 |          0 |     53200 |  0.09 |
|   LUT as Logic          |   50 |     0 |          0 |     53200 |  0.09 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |    6 |     0 |          0 |    106400 | <0.01 |
|   Register as Flip Flop |    6 |     0 |          0 |    106400 | <0.01 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    1 |     0 |          0 |     26600 | <0.01 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, i

```

BRAM:4

```

2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    4 |     0 |          0 |       140 |  2.86 |
|   RAMB36/FIFO*    |    4 |     0 |          0 |       140 |  2.86 |
|     RAMB36E1 only |    4 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |       280 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accom
 accommodate a RAMB18E1
```

四、 Other discoveries

CPU 存入記憶體空間

在設計 BRAM 大小時，剛開始以為 BRAM 的大小會是.hex 檔的大小，結果在看完波形後，突然發現對不上，原本 7.4KB(.hex)只寫入了 360B 到 BRAM 裡面，兩者的落差蠻大。

後面就開始上網查資料，才發現到.elf 檔案裏面包含了很多資訊，而 CPU 在執行的時候，只會將機器碼指令存到記憶體中，這可能也就解釋了為什麼 CPU 只存入了一些資料到 BRAM 裡面。

只是負責解析.elf 的應該是 OS(?)，但在 lab-exmem-fir 檔案中，找不太到相關的資料，因此不是很確定目前的結論是否正確。

Fir.c 的 IO Mapping

fir.c 在函式中使用指標傳遞長度為 11 的陣列，運算的結果會在 mprj_io 的第 31 到 16 的位元上，也是使用這 16 個位元傳遞 start mark、end mark。