[SoC Lab] Lab4-1

tags: `SoC Lab`, `SOC Design`

Team 13

Student ID	Name
311551095	林聖博
312551174	張祐誠

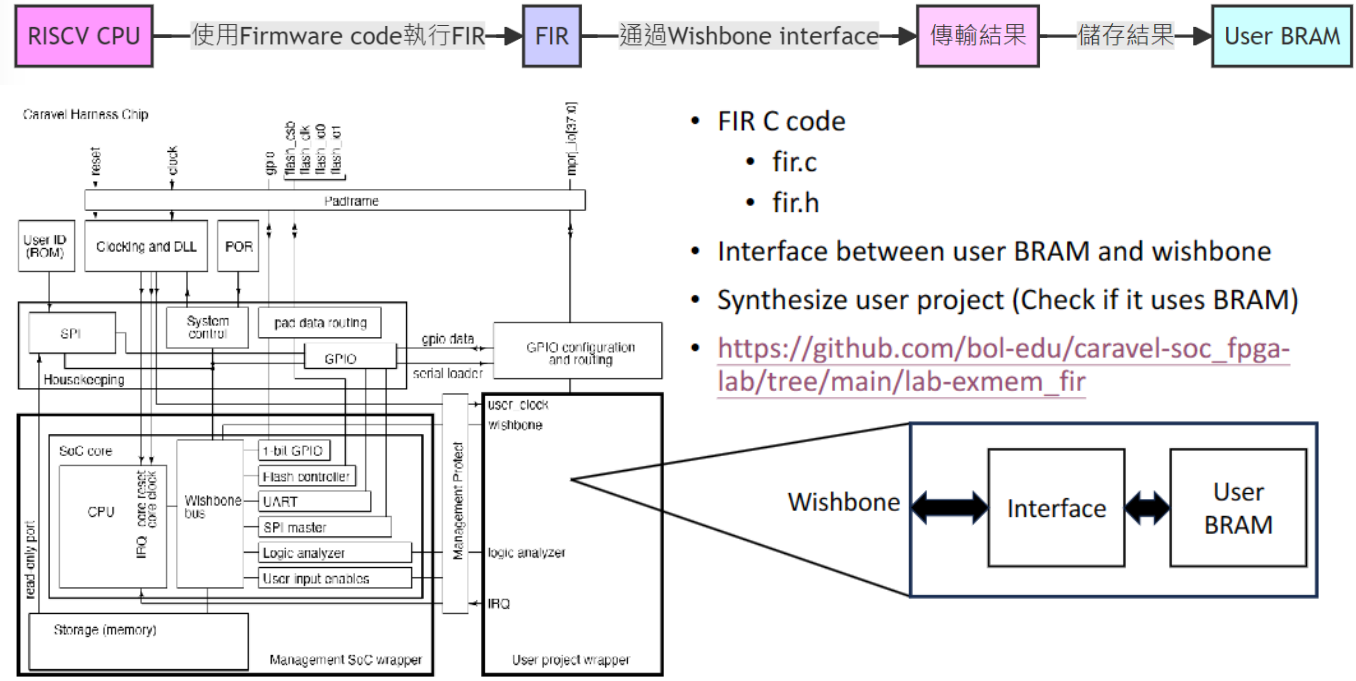
附上此篇Hackmd Linkhttps://hackmd.io/@Sheng08/rJCgSEmET

Lab 4 Spec

- Lab4-1 (lab-exmem_fir)
  - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab-exmem_fir
- Lab4-2 (lab-caravel_fir)
  - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab-caravel_fir

Overview

Lab4-1 需要撰寫firmware code（包括`fir.c`和`fir.h`文件）來實現FIR Engine。並且需要為Wishbone 和 User BRAM（Block RAM）建立 interfaces。



Explanation of your firmware code

Lab4-1/testbench/counter_la_fir/fir.c

```c
#include "fir.h"

void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
        //initial your fir
        for (int i=0; i<N; i++) inputbuffer[i] = 0;
        for (int i=0; i<N; i++) outputsignal[i] = 0;
}

int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
        initfir();
        //write down your fir
        int sum;
        int getData;
        for (int i=0; i<N; i++) {
            sum = 0;
                getData = inputsignal[i];
                inputbuffer[i] = getData;
                for (int j=0; j<i+1; j++) {
                        sum += inputbuffer[i-j] * taps[j];
                }
                outputsignal[i] = sum;
        }

        return outputsignal;
}

```

Lab4-1/testbench/counter_la_fir/fir.h

```c
#ifndef __FIR_H__
#define __FIR_H__

#define N 11

int taps[N] = {0,-10,-9,23,56,63,56,23,-9,-10,0};
int inputbuffer[N];
int inputsignal[N] = {1,2,3,4,5,6,7,8,9,10,11};
int outputsignal[N];
#endif
```

1. `initfir()`：
   - 用於初始化FIR濾波器
   - `__attribute__ ((section (".mprjram")))`：
     - 此屬性指定`initfir()`放置於特定的記憶體區段，並命名為`.mprjram`
   - 初始化兩個`inputbuffer[]`與`outputsignal[]`將所有元素設置為`0`

2. `fir()`：
    - 主要實現FIR濾波器(執行FIR運算)
    - 循環`N=11`次，其中`N`為輸入訊號的樣本數（`fir.h`）
    - 在Loop中
        - 將`sum`設置為0，保存當前樣本的濾波輸出
        - 讀取當前輸入樣本到`getData`中
        - 將當前輸入樣本存儲在`inputbuffer`中
        - 透過對`inputbuffer`與Fir係數（`taps`）進行卷積來計算濾波輸出（`sum`）直到當前樣本index完成為止
            - `inputbuffer`執行類似於shift register的功能，用於存儲和更新輸入資料。每次迭代時，新的輸入資料會被加入到`inputbuffer`，並對齊tap coefficients。
        - 將計算出的輸出存儲在`outputsignal[]`中
            - 通過累加FIR計算中的每次部分和來計算單點結果（即當前輸入對應的輸出值）
    - 最後，將return指向`outputsignal`，該陣列包含了濾波後的訊號
    - Note:

```
getData = inputsignal[i];
inputbuffer[i] = getData;
```

此作用類似模擬 Lab3 Fir行為，將資料先讀到一個Buffer進行存放，要使用時則去該 Buffer 存取。

How does it execute a multiplication in assembly code

Lab4-1/testbench/counter_la_fir/counter_la_fir.elf-fir.s

```
178     .L9:
179             .loc 1 19 24 discriminator 3
180             lw      a4,-24(s0)
181             lw      a5,-28(s0)
182             sub     a5,a4,a5
183             .loc 1 19 22 discriminator 3
184             lui     a4,%hi(inputbuffer)
185             addi    a4,a4,%lo(inputbuffer)
186             slli    a5,a5,2
187             add     a5,a4,a5
188             lw      a3,0(a5)
189             .loc 1 19 34 discriminator 3
190             lui     a5,%hi(taps)
191             addi    a4,a5,%lo(taps)
192             lw      a5,-28(s0)
193             slli    a5,a5,2
194             add     a5,a4,a5
195             lw      a5,0(a5)
196             .loc 1 19 28 discriminator 3
197             mv      a1,a5
198             mv      a0,a3
199             call    __mulsi3
200             mv      a5,a0
201             mv      a4,a5
202             .loc 1 19 8 discriminator 3
203             lw      a5,-20(s0)
204             add     a5,a5,a4
205             sw      a5,-20(s0)
206             .loc 1 18 25 discriminator 3
207             lw      a5,-28(s0)
208             addi    a5,a5,1
209             sw      a5,-28(s0)
```

```
1   #include "fir.h"
2
3   void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
4           //initial your fir
5           for (int i=0; i<N; i++) inputbuffer[i] = 0;
6           for (int i=0; i<N; i++) outputsignal[i] = 0;
7   }
8
9   int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
10          initfir();
11          //write down your fir
12          int sum;
13          int getData;
14          for (int i=0; i<N; i++) {
15              sum = 0;
16                  getData = inputsignal[i];
17                  inputbuffer[i] = getData;
18                  for (int j=0; j<i+1; j++) {
19                          sum += inputbuffer[i-j] * taps[j];
20                  }
21                  outputsignal[i] = sum;
22          }
23
24          return outputsignal;
25  }
26
```

> Lab4-1/testbench/counter_la_fir/counter_la_fir.out

```
38000000 <__mulsi3>:
38000000:       00050613                mv      a2,a0
38000004:       00000513                li      a0,0
38000008:       0015f693                andi    a3,a1,1
3800000c:       00068463                beqz    a3,38000014 <__mulsi3+0x14>
38000010:       00c50533                add     a0,a0,a2
38000014:       0015d593                srli    a1,a1,0x1
38000018:       00161613                slli    a2,a2,0x1
3800001c:       fe0596e3                bnez    a1,38000008 <__mulsi3+0x8>
38000020:       00008067                ret
```

```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
        mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
        hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
        csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

在`counter_la_fir.out` 可以發現`__mulsi3`，進行 multiplication ，並且其位置符合`3800_0000` => User Project Memory Starting

`__mulsi3`: 使用 Bit operations 和 Shift 來計算兩個整數（a0 和 a1）的乘積，並將結果存儲在 a0 中。

1. **初始化**：
   - `mv a2,a0`：將 a0 的值（一個乘數）移到 a2, a2 用作累加器
   - `li a0,0`：將 a0 初始化為 0，用於存儲最終的乘積結果
2. **乘法的逐位元計算**：
   - 此過程檢查 a1 的每一位，並根據每一位是 1 還是 0 來決定是否將 a2 的值（當前累加的乘積）加到 a0 上
   - `andi a3,a1,1`：透過AND檢查 a1 的最低位（LSB）是否為 1
   - `beqz a3,38000014 <__mulsi3+0x14>`：如果 a3 為 0（即 a1 的 LSB 為 0），則跳過加法
   - `add a0,a0,a2`：如果 a1 的 LSB 為 1，將 a2 的值加到 a0 上
3. **Shift操作**：
   - `srli a1,a1,0x1`：將 a1 **向右**移位一位，準備檢查下一位
   - `slli a2,a2,0x1`：將 a2 **向左**移位一位，相當於乘以 2，為下一次可能的加法做準備
4. **循環和結束條件**：
   - `bnez a1,38000008 <__mulsi3+0x8>`：如果 a1 非零，則繼續循環，再次檢查下一位
   - 當 a1 為零時，結束循環
5. **返回結果**：
   - `ret`：return function，此時 a0 包含最終的乘積結果

總結: `__mulsi3`透過**Bit operations 和 Shift **，以一種高效且低層次的技巧方式，來實現整數的乘法運算。此方法可以不直接使用乘法指令，而透過位移和加法來達到相同的效果，可提升處理器的效率。

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

Lab4-1/firmware/sections.lds

```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
        mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
        hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
        csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

Lab4-1/testbench/counter_la_fir/counter_la_fir.out

```
Disassembly of section .mprjram:

38000000 <__mulsi3>:
38000000:       00050613                mv      a2,a0
38000004:       00000513                li      a0,0
38000008:       0015f693                andi    a3,a1,1
3800000c:       00068463                beqz    a3,38000014 <__mulsi3+0x14>
38000010:       00c50533                add     a0,a0,a2
38000014:       0015d593                srli    a1,a1,0x1
38000018:       00161613                slli    a2,a2,0x1
3800001c:       fe0596e3                bnez    a1,38000008 <__mulsi3+0x8>
38000020:       00008067                ret

38000024 <initfir>:
38000024:       fe010113                addi    sp,sp,-32
38000028:       00812e23                sw      s0,28(sp)
3800002c:       02010413                addi    s0,sp,32
38000030:       fe042623                sw      zero,-20(s0)
38000034:       0240006f                j       38000058 <initfir+0x34>
38000038:       05c00713                li      a4,92
3800003c:       fec42783                lw      a5,-20(s0)
38000040:       00279793                slli    a5,a5,0x2
38000044:       00f707b3                add     a5,a4,a5
38000048:       0007a023                sw      zero,0(a5)
3800004c:       fec42783                lw      a5,-20(s0)
38000050:       00178793                addi    a5,a5,1
38000054:       fef42623                sw      a5,-20(s0)
38000058:       fec42703                lw      a4,-20(s0)
3800005c:       00a00793                li      a5,10
38000060:       fce7dce3                bge     a5,a4,38000038 <initfir+0x14>
38000064:       fe042423                sw      zero,-24(s0)
38000068:       0240006f                j       3800008c <initfir+0x68>
3800006c:       08800713                li      a4,136
38000070:       fe842783                lw      a5,-24(s0)
38000074:       00279793                slli    a5,a5,0x2
38000078:       00f707b3                add     a5,a4,a5
3800007c:       0007a023                sw      zero,0(a5)
38000080:       fe842783                lw      a5,-24(s0)
38000084:       00178793                addi    a5,a5,1
38000088:       fef42423                sw      a5,-24(s0)
3800008c:       fe842703                lw      a4,-24(s0)
38000090:       00a00793                li      a5,10
38000094:       fce7dce3                bge     a5,a4,3800006c <initfir+0x48>
38000098:       00000013                nop
3800009c:       00000013                nop
380000a0:       01c12403                lw      s0,28(sp)
380000a4:       02010113                addi    sp,sp,32
380000a8:       00008067                ret
```

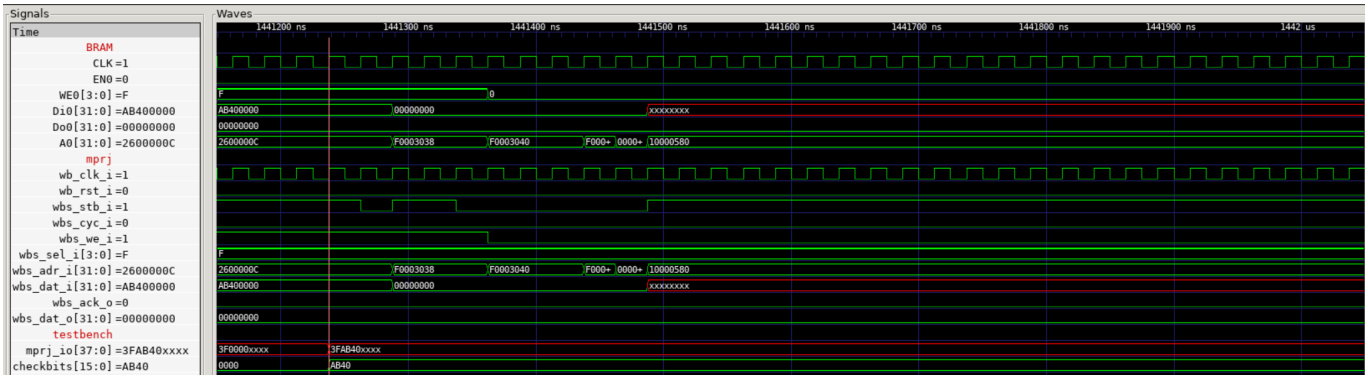
```
380000ac <fir>:
380000ac:       fe010113                addi    sp,sp,-32
380000b0:       00112e23                sw      ra,28(sp)
380000b4:       00812c23                sw      s0,24(sp)
380000b8:       02010413                addi    s0,sp,32
380000bc:       f69ff0ef                jal     ra,38000024 <initfir>
380000c0:       fe042423                sw      zero,-24(s0)
380000c4:       0cc0006f                j       38000190 <fir+0xe4>
380000c8:       fe042623                sw      zero,-20(s0)
380000cc:       02c00713                li      a4,44
380000d0:       fe842783                lw      a5,-24(s0)
380000d4:       00279793                slli    a5,a5,0x2
380000d8:       00f707b3                add     a5,a4,a5
380000dc:       0007a783                lw      a5,0(a5)
380000e0:       fef42023                sw      a5,-32(s0)
380000e4:       05c00713                li      a4,92
380000e8:       fe842783                lw      a5,-24(s0)
380000ec:       00279793                slli    a5,a5,0x2
380000f0:       00f707b3                add     a5,a4,a5
380000f4:       fe042703                lw      a4,-32(s0)
380000f8:       00e7a023                sw      a4,0(a5)
380000fc:       fe042223                sw      zero,-28(s0)
38000100:       0600006f                j       38000160 <fir+0xb4>
38000104:       fe842703                lw      a4,-24(s0)
38000108:       fe442783                lw      a5,-28(s0)
3800010c:       40f707b3                sub     a5,a4,a5
38000110:       05c00713                li      a4,92
38000114:       00279793                slli    a5,a5,0x2
38000118:       00f707b3                add     a5,a4,a5
3800011c:       0007a683                lw      a3,0(a5)
38000120:       00000713                li      a4,0
38000124:       fe442783                lw      a5,-28(s0)
38000128:       00279793                slli    a5,a5,0x2
3800012c:       00f707b3                add     a5,a4,a5
38000130:       0007a783                lw      a5,0(a5)
38000134:       00078593                mv      a1,a5
38000138:       00068513                mv      a0,a3
3800013c:       ec5ff0ef                jal     ra,38000000 <__mulsi3>
38000140:       00050793                mv      a5,a0
38000144:       00078713                mv      a4,a5
38000148:       fec42783                lw      a5,-20(s0)
3800014c:       00e787b3                add     a5,a5,a4
38000150:       fef42623                sw      a5,-20(s0)
38000154:       fe442783                lw      a5,-28(s0)
38000158:       00178793                addi    a5,a5,1
3800015c:       fef42223                sw      a5,-28(s0)
38000160:       fe842703                lw      a4,-24(s0)
38000164:       fe442783                lw      a5,-28(s0)
38000168:       f8f75ee3                bge     a4,a5,38000104 <fir+0x58>
3800016c:       08800713                li      a4,136
38000170:       fe842783                lw      a5,-24(s0)
38000174:       00279793                slli    a5,a5,0x2
38000178:       00f707b3                add     a5,a4,a5
3800017c:       fec42703                lw      a4,-20(s0)
38000180:       00e7a023                sw      a4,0(a5)
38000184:       fe842783                lw      a5,-24(s0)
38000188:       00178793                addi    a5,a5,1
3800018c:       fef42423                sw      a5,-24(s0)
38000190:       fe842703                lw      a4,-24(s0)
38000194:       00a00793                li      a5,10
38000198:       f2e7d8e3                bge     a5,a4,380000c8 <fir+0x1c>
3800019c:       08800793                li      a5,136
380001a0:       00078513                mv      a0,a5
380001a4:       01c12083                lw      ra,28(sp)
380001a8:       01812403                lw      s0,24(sp)
380001ac:       02010113                addi    sp,sp,32
380001b0:       00008067                ret
```

在`section.lds`中，可以找到分配給 user project 的地址為`0x38000000`。在透過觀察`counter_la_fir.out`，確定整個 `mprjram` 為 `0x1b0` 需要 432 Byte 的記憶體空間，大小在十六進制中表示為`0x1b0`。

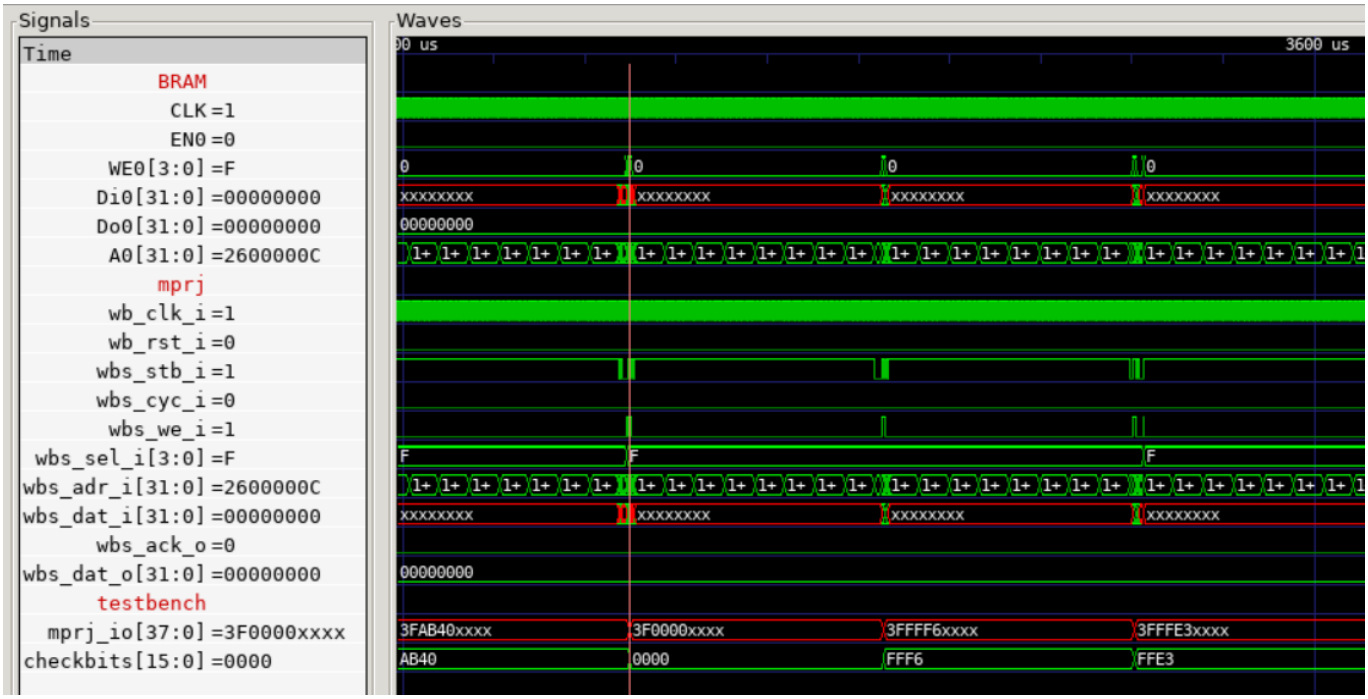
Interface between BRAM and wishbone

Waveform from xsim

當 `mprj_io[37:0] = 16'hAB40`, 開始執行 FIR

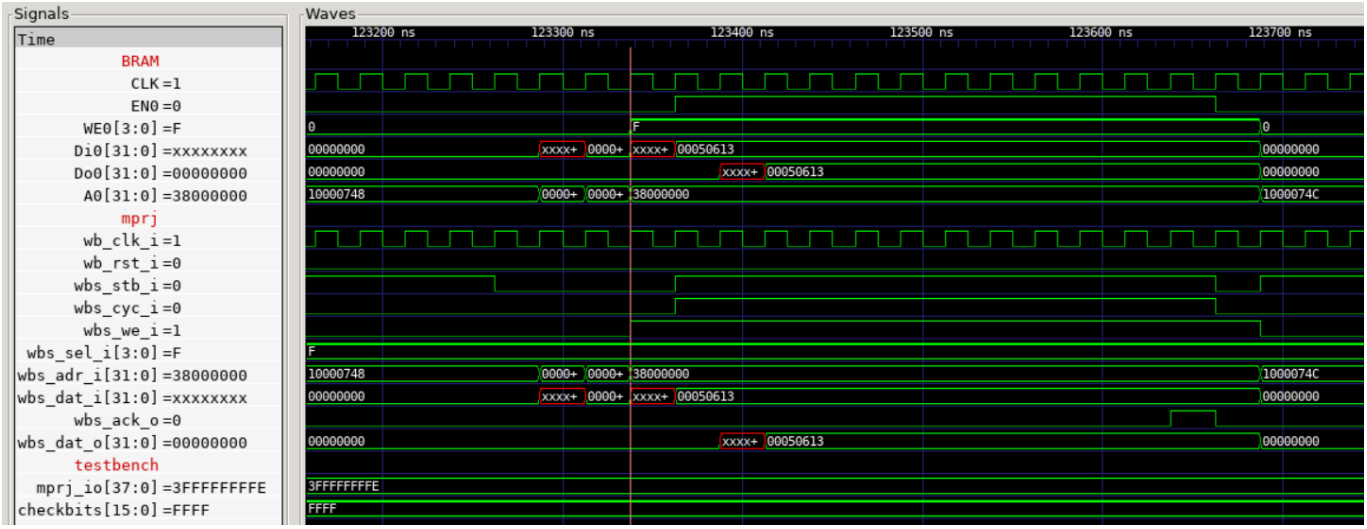


Wishbone 將計算結果傳送到 interface



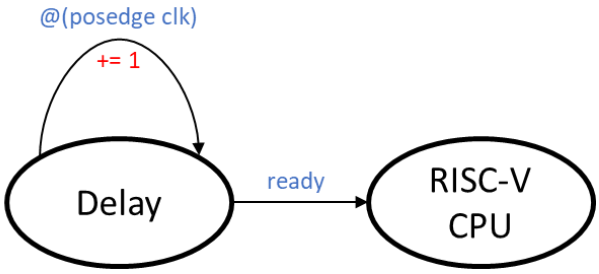
`check_bits = 'hAB51` 結束 FIR

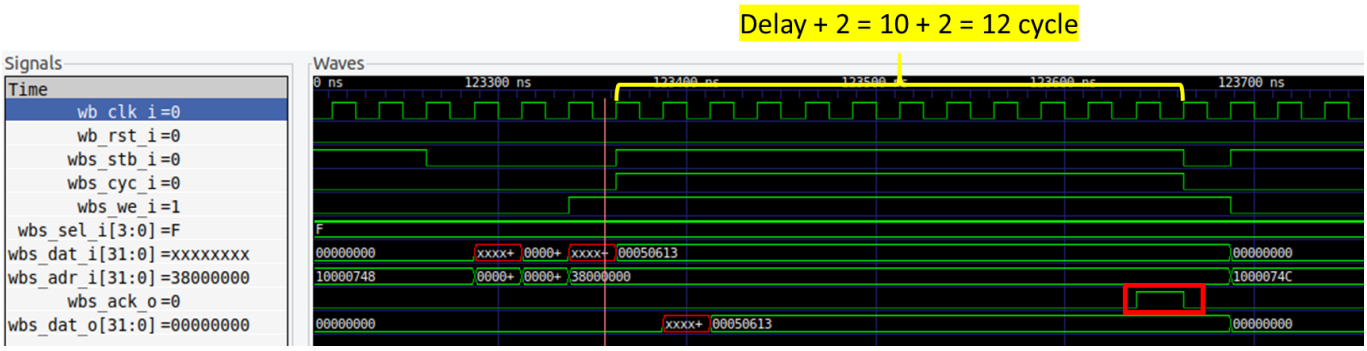
將結果寫到 BRAM



FSM

```
// write a sequential logic that increments the count register until the value is eqaul to DELAYS
always @(posedge clk) begin
    if (rst) begin
        delay_count <= 0;
        ready <= 0;
    end else begin
        ready <= 0;
        if (valid && !ready) begin
            if (delay_count == DELAYS) begin
                ready <= 1;
                delay_count <= 0;
            end else begin
                delay_count <= delay_count + 1;
            end
        end
    end
end
```





1. 當 Counter 保持在0時，狀態為 idle，沒有進行資料傳輸
2. 當 Counter 開始計數時，狀態變為 processing
3. 當 Counter 達到 DELAY 時，狀態變為 response，此時將回傳 ack 和資料給CPU，並重新設定 Counter為 0，狀態為 idle

Synthesis report

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	52	0	0	53200	0.10
LUT as Logic	52	0	0	53200	0.10
LUT as Memory	0	0	0	17400	0.00
Slice Registers	33	0	0	106400	0.03
Register as Flip Flop	33	0	0	106400	0.03
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	2	0	0	140	1.43
RAMB36/FIFO*	2	0	0	140	1.43
RAMB36E1 only	2				
RAMB18	0	0	0	280	0.00

Github link

- https://github.com/Sheng08/SoC-Lab-FIR_Lab4

🚀 補充

> 無