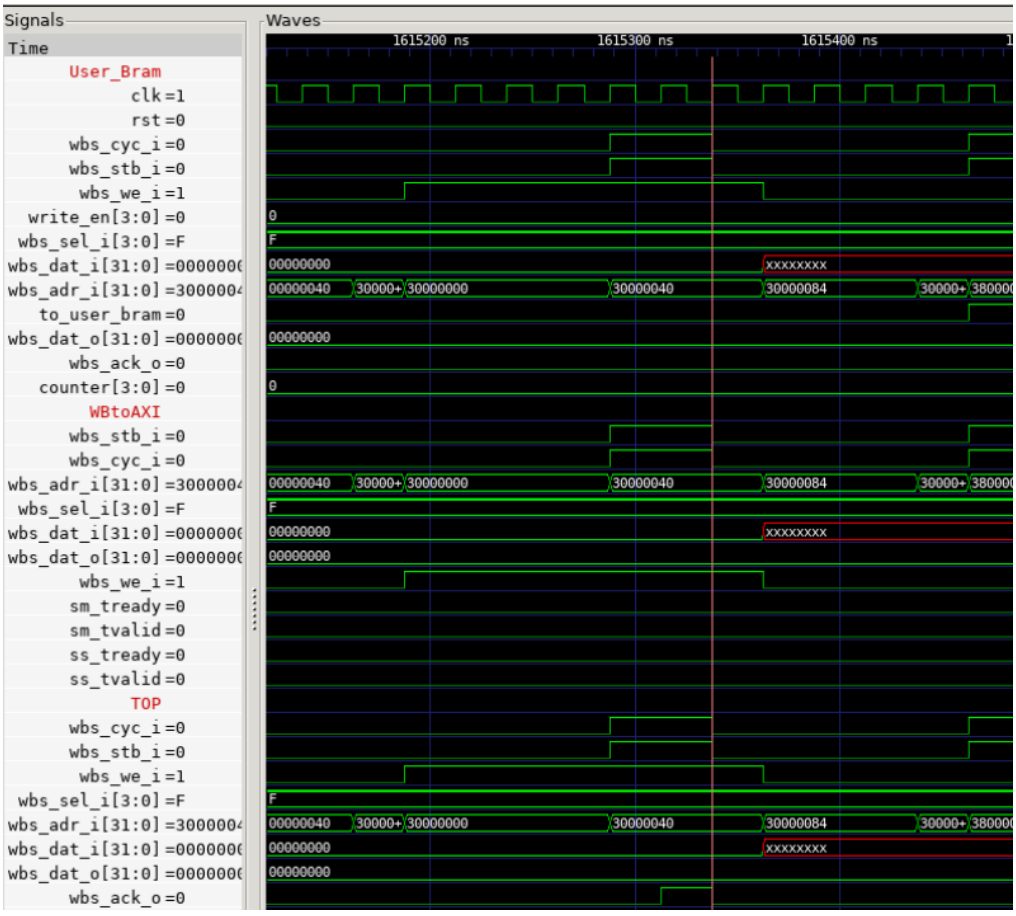
SOC Design Laboratory Lab4-2

311651055_林柏宇
交大電物碩二

波型與數據分析：

1. 當軟體將數值指定給預設的地址，CPU 將啟動一個 Wishbone 寫入 Wishbone Cycle，將數值發送到該地址空間。以下圖為例，軟體想要初始化位於 0x30000040 的 tap_1 時，CPU 將生成一個 Wishboe 交易，將數值寫入 0x30000040。當我們想要為其他預設的地址分配數值時，相同的事情也會發生。

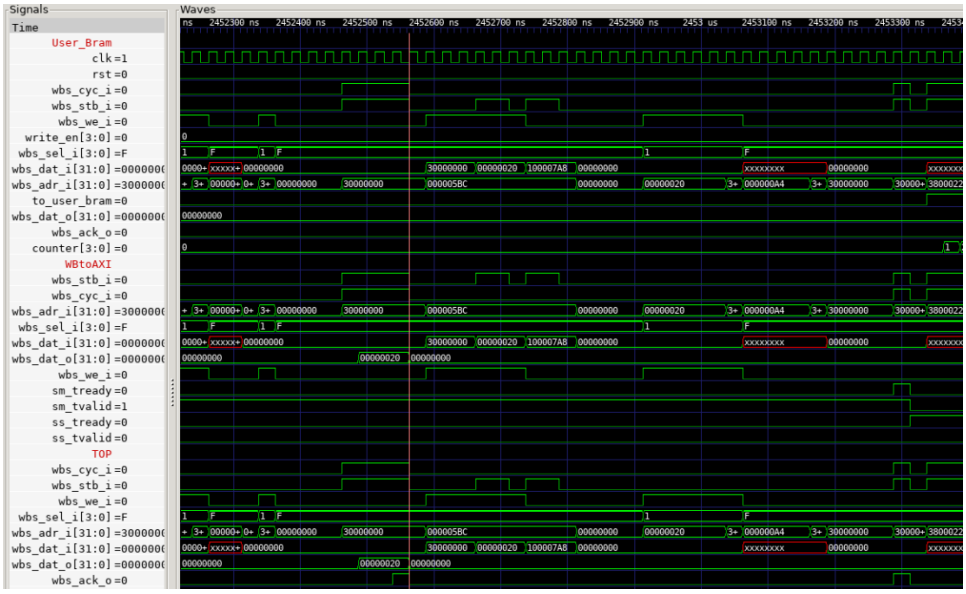



```
void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
    //initial your fir
    // {0,-10,-9,23,56,63,56,23,-9,-10,0};
    tap_1 = 0;
    tap_2 = -10;
    tap_3 = -9;
    tap_4 = 23;
    tap_5 = 56;
    tap_6 = 63;
    tap_7 = 56;
    tap_8 = 23;
    tap_9 = -9;
    tap_10= -10;
    tap_11= 0;

    datalength = 64;

    reg_mprj_datal = 0x00A50000;
    status = 0x00000001;

}
```

```
// fir input X[n]
#define inputsignal        (*(volatile uint32_t*)0x30000080)
// fir output Y[n]
#define outputsignal       (*(volatile uint32_t*)0x30000084)
// taps[n]
#define tap_1      (*(volatile uint32_t*)0x30000040)
#define tap_2      (*(volatile uint32_t*)0x30000044)
#define tap_3      (*(volatile uint32_t*)0x30000048)
#define tap_4      (*(volatile uint32_t*)0x3000004c)
#define tap_5      (*(volatile uint32_t*)0x30000050)
#define tap_6      (*(volatile uint32_t*)0x30000054)
#define tap_7      (*(volatile uint32_t*)0x30000058)
#define tap_8      (*(volatile uint32_t*)0x3000005c)
#define tap_9      (*(volatile uint32_t*)0x30000060)
#define tap_10     (*(volatile uint32_t*)0x30000064)
#define tap_11     (*(volatile uint32_t*)0x30000068)
// data length
#define datalength (*(volatile uint32_t*)0x30000010)
// status
#define status     (*(volatile uint32_t*)0x30000000)
```

2. 當軟體從預設地址提取數值時，CPU 將啟動一個 Read Cycle 到該地址，有對該地址空間有控制權的裝置會通過 Wishbone 將相應數據返回給 CPU。

```
int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(int method){
    initfir();
    //write down your fir

    int x[64];
    int s;
    for (int i = 0; i < 64; i++) {
        if(method == 1)
            x[i] = i;
        else if(method == 2)
            x[i] = 64 - i;
        else if(method == 3)
            x[i] = 1;
    }

    s = status;
    for (int i = 0; i < 64; i++) {
        while (!((s >> 4) & 1) && i != 0)
            s = status;
        inputsignal = x[i];

        while (!((s >> 5) & 1))
            s = status;
        ans[i] = outputsignal;
    }

    s = status;
    reg_mprj_datal = ((0x000000FF & ans[63]) << 24) | 0x005A0000;

    return ans;
}
```



3. FIR 實際情況與理論上模擬運算的比較：

計算 FIR 時，一旦我們從 AXI-Stream 獲取輸入數據，就可以直接開始乘法和累加過程。同時，輸入數據被保持並存儲在數據 BRAM 中，直到答案完全計算出來。在這一刻，我們可以進行握手並發送輸出數據。因此，理論上只需要 11 + 1 個周期來計算一個結果點。

理論上單位 cycle 的運算量: 1/12 (1/cycle)

Rate: 12 (cycle)

實際上單位 cycle 的運算量：

(2449987500 - 2449137500) ps / (25000) ps = 31 個周期

會造成此情況是因為 BRAM 使用效率太低。

4. 延遲：(2470312500 - 2449162500) ps / (25000) ps = 847 個周期

5. Improve：

(1)提高單位 cycle 的運算量：

(a)利用分離的讀寫端口 BRAM，隱藏寫入過程於當前讀取過程中。

(b)讓 Wishbond 到 AXI 接口傳遞信號而不鎖存數據，減少額外延遲。

(2) 其他提升：

(a)增加乘法器和加法器(平行化)。

(b)多運用 pointer 減少數據 BRAM 中實際位移與訪問次數。

6.

Github：https://github.com/leolin0501/soclab.github.io/tree/main/Lab4

HackMD：https://hackmd.io/gzN6anM4SDKoXhKOAAJzDA