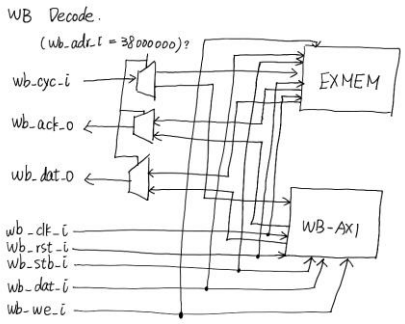
SoC Lab4-2

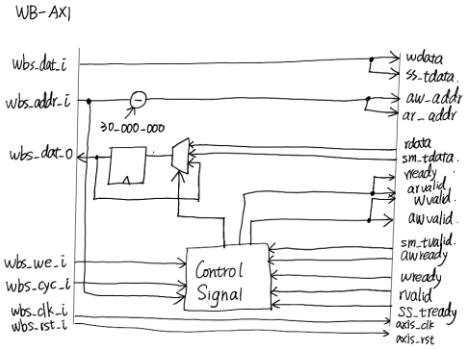
一、Design block diagram – datapath, control-path

Datapath

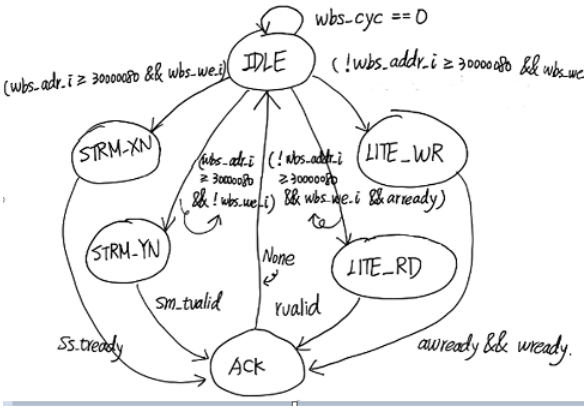
WB Decoder



WB-AXI



Control path(WB-AXI)



在 WB-AXI 中，一共定義了六個狀態。

IDLE: 在這個狀態中，會去看 WB 的 ADDR、WE 來判斷是下個 cycle 要進入哪個狀態。

LITE_WR: 由於 FIR 的設計是 AW、W 是各自獨立的通道且深度只有 1，因此如果兩個 ready 皆為 1 的話，就代表 AW、W 都有成功寫入，返回可接收的狀態。

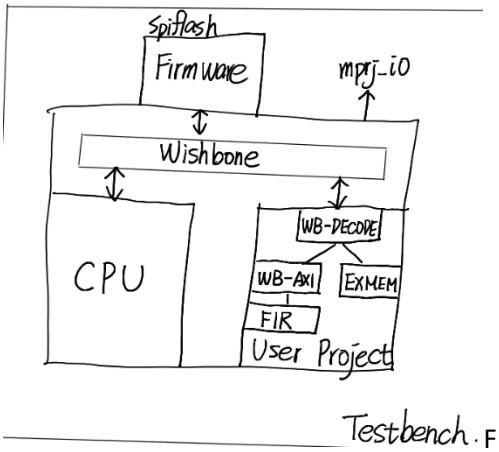
LITE_RD: 只要判定 rvalid 是否為 1，就能知道是否能將 rdata 傳送回 WB。

STRM_XN:當 ss_tready=1 時，此時就能將 wbs_dat_i 傳送到 AXI 中。

STRM_YN: 只要判定 sm_tvalid 是否為 1，就能知道是否能將 Y[n]傳送回 WB。

ACK:當上述的狀態在完成動作時，就會統一到 ACK 狀態，返回 ACK 到 WB。

二、The interface protocol between firmware, user project and testbench



Firmware 會透過 WB 跟 CPU 做溝通，而當 CPU 獲取到指令後，就會將它寫入 EXMEM。

此時會先透過 WB BUS 傳送到 User Project 中，之後 WB Decoder 會將它轉傳到 EXMEM。

而當指令中需要使用到 FIR 硬體時，就會先透過 WB BUS 送到 User Project，之後 WB-AXI 會先將它轉成 AXI，再送到 FIR。

而在 Firmware 中，Caravel 會將運算結果傳送到 mprj_io，此時 Testbench 就能從 mprj_io 接收運算結果。

三、Waveform and analysis of the hardware/software behavior.

Software:

```
void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
// initial your fir
    int i;
    uint32_t Mask, Status;

// send data length
    reg_fir_datalen = fir_test_len;

// send taps
    for(i=0; i<N; i++){
        *(reg_fir_coeff+i) = taps[i];
    }

// check ap_idle = 1
    // fir_control[2] = ap_idle -> Mask
    Mask = 0;
    Mask |= (1 << 2);

    Status = reg_fir_control & Mask;
    while(Status != 4){
        Status = reg_fir_control & Mask;
    }

// send ap_start
    // set fir_control[0] = 1 -> ap_start = 1
    reg_fir_control = 1;

}
```

在 Init 中，首先會先去傳送這次要測驗的長度(Data length)。

之後就會開始傳送 Taps，而因為 Taps 數量有點多，因此是採用 pointer 的方式去賦值，相對方便許多。

當參數都傳送完畢後，就會去檢查 ap_idle 是否為 1，因為怕其他位元會去干擾到判斷，因此在判斷之前會去使用 Mask，將該位元單獨抓出來做判斷。

當 ap_idle 為 1 時，就會跳出 while，傳送 ap_start(reg_fir_control = 1)。

```c
int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
    initfir();
    //write down your fir
    int i;
    uint32_t Mask, Status;

    for(i=0; i<fir_test_len; i++){
        // check X[n] = 1 is ready to accept input.
        Mask = 0;
        Mask |= (1 << 4);

        Status = reg_fir_control & Mask;
        while(Status != 16){
            Status = reg_fir_control & Mask;
        }
        // send X[n]
        reg_fir_x = i+1;

        // check when Y[n] is ready
        Mask = 0;
        Mask |= (1 << 5);

        Status = reg_fir_control & Mask;
        while(Status != 32){
            Status = reg_fir_control & Mask;
        }

        // receive Y[n]
        outputsignal[i] = reg_fir_y;
    }

    return outputsignal;
}
```

在開始執行 fir 之前，會先去呼叫 Init()，初始化 FIR 硬體。

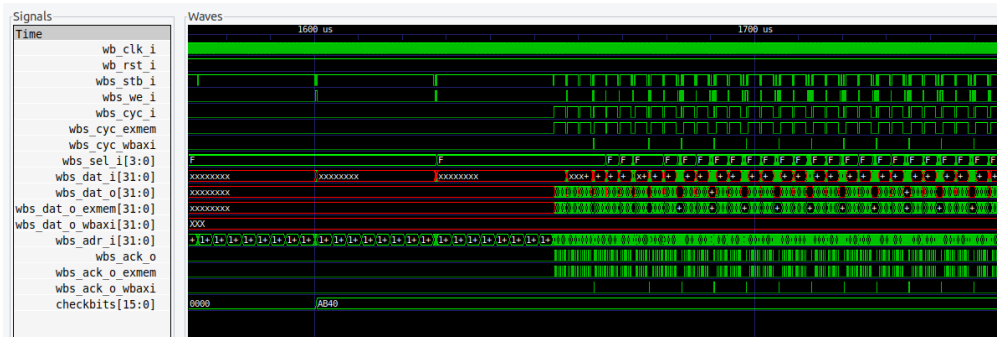
接下來，就可以開始傳送測資。

首先，會先去檢查 X[n] 是否處於 ready 的狀態，當檢查條件成立後，才會去執行傳送 X[n]。

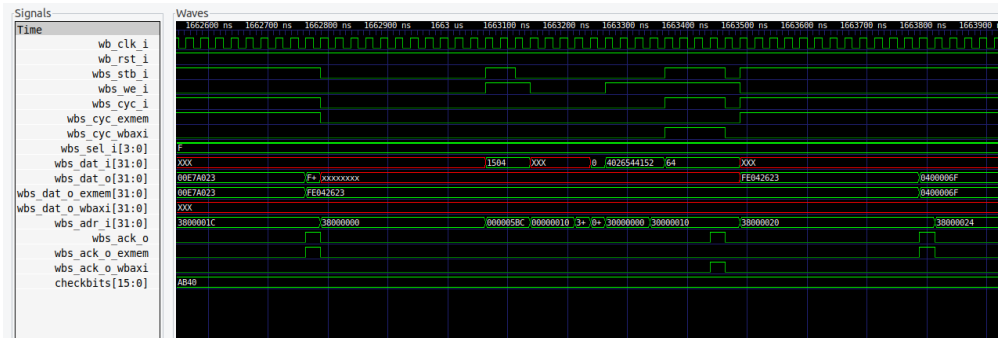
之後，就會則是去檢查 Y[n] 是否已經運算完畢，當檢查條件成立後，才會去執行接收 Y[n]。

最後，執行完所有測資後，就會返回一個指標，讓 counter_la_example.c 能夠將結果丟到 mprj_io 上，之後再由 Testbench 去檢查結果是否正確。

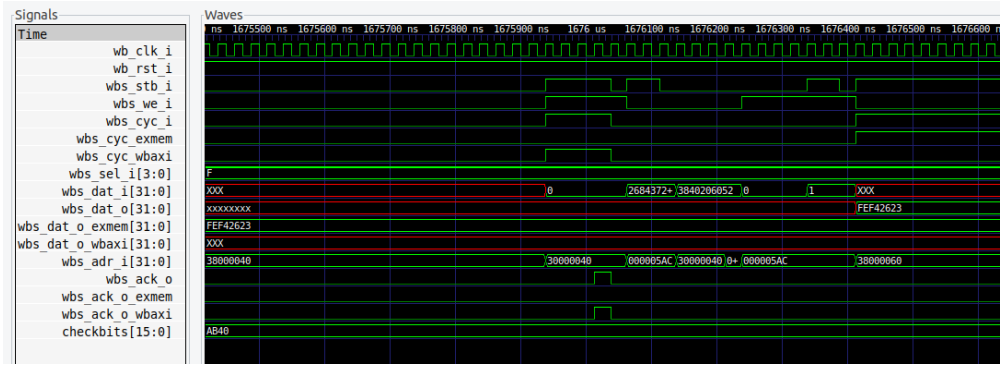
Hardware:



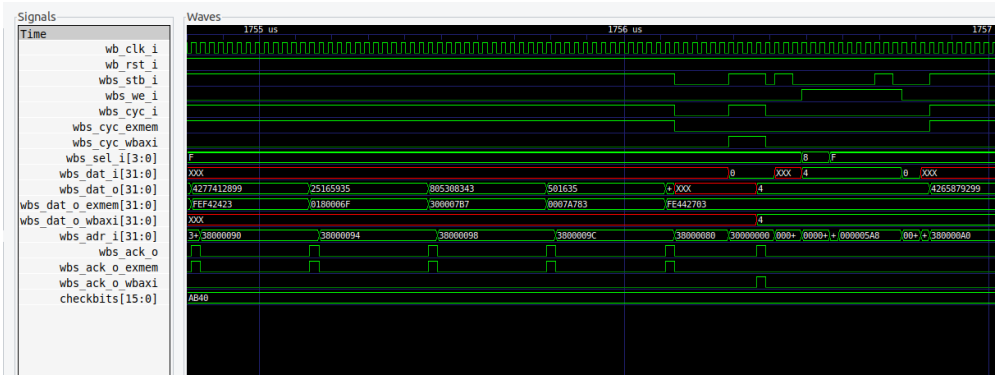
當 Checkbits 接收到 AB40 時，就代表 FIR 運算開始。



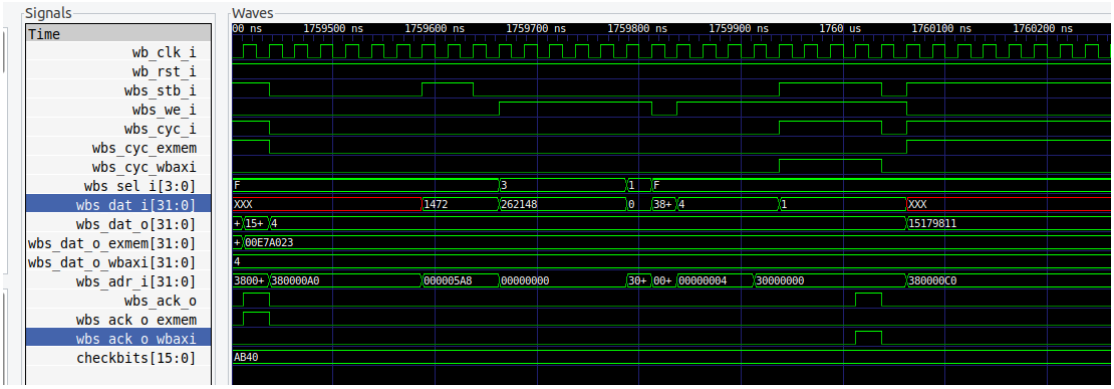
接下來，會去送測試長度(64)到 0x30000010，表示這次的測試共有 64 筆測資。



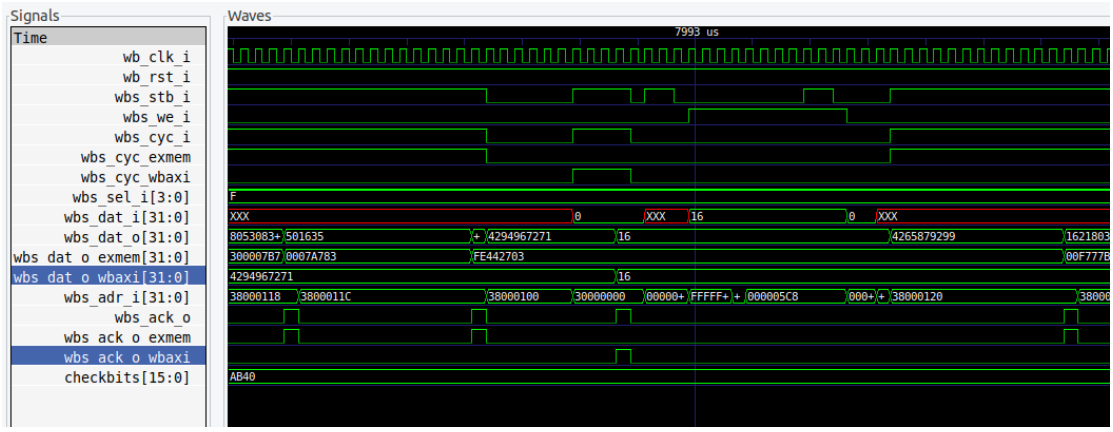
送完測試長度後，就要開始傳送 Taps(0x30000040)。



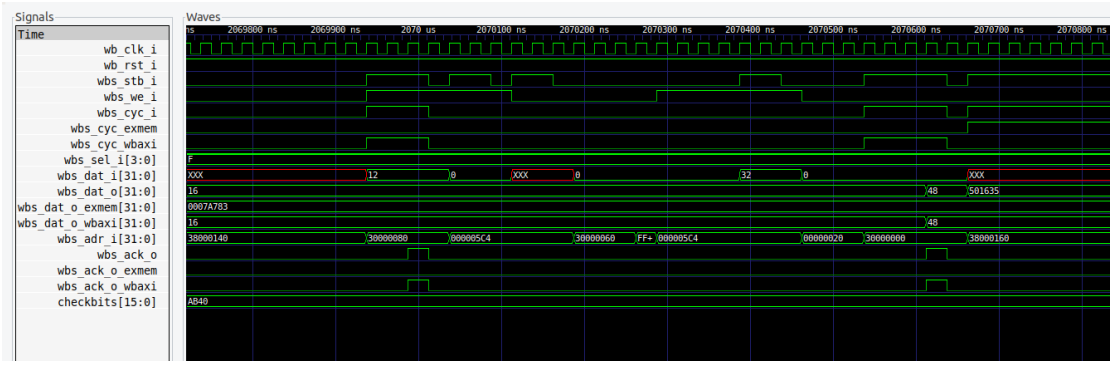
之後就要去檢查 ap_idle(0x3000000 [2])是否為 1，如果為 1 就能開始 FIR 運算。



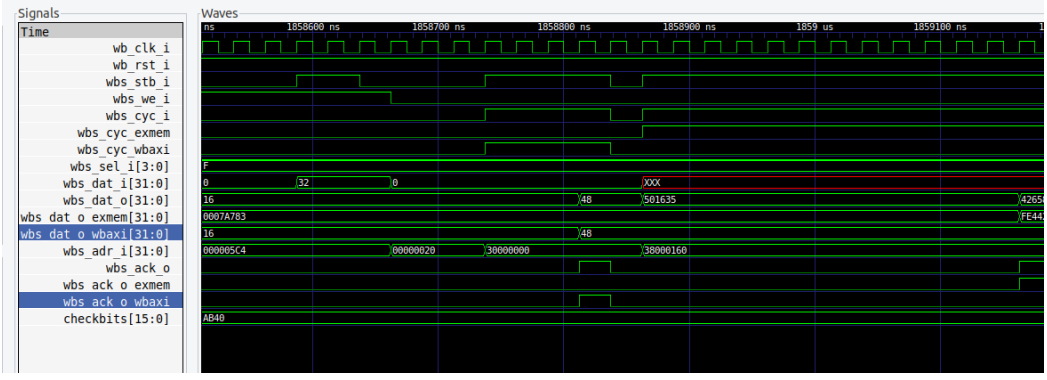
檢查完 ap_idle 為 1 後，就能送 ap_start(0x3000000 [0])訊號給 FIR。



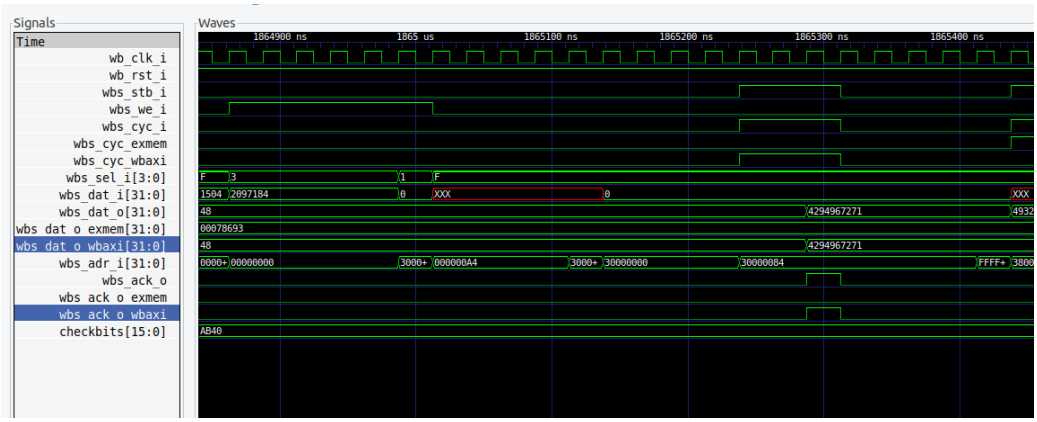
在開始傳送 X[n]前，會先去檢查 X[n] ready(0x3000000 [4])是否為 1。



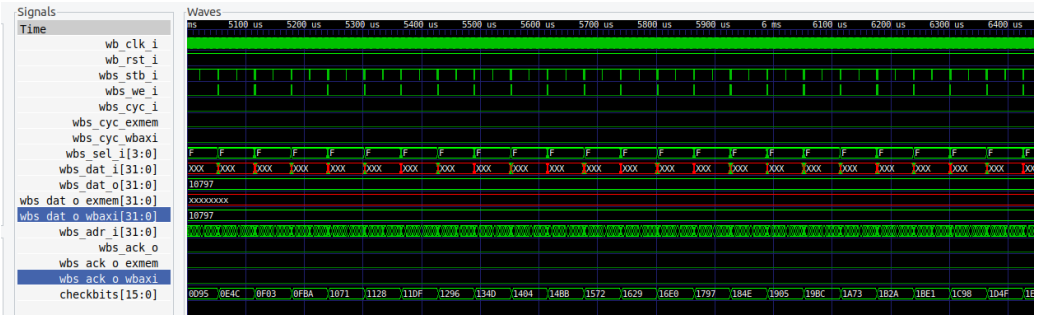
檢查完 X[n] ready(0x3000000 [4])為 1 後，就能傳送 X[n] (0x3000080)到 FIR。



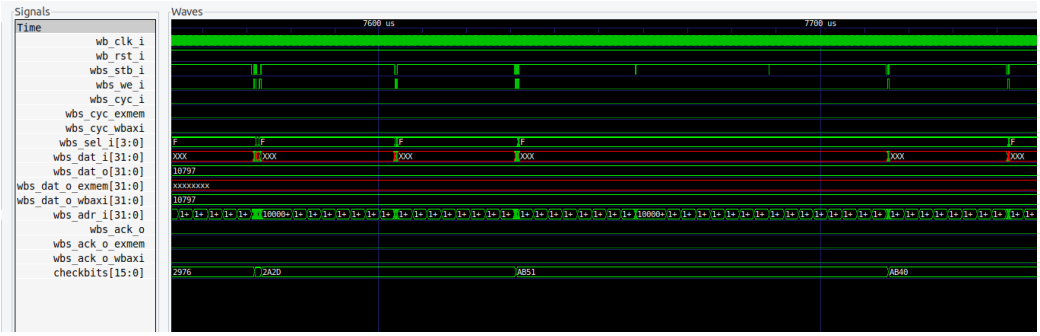
在開始傳送 Y[n]前，會先去檢查 Y[n] valid(0x3000000 [5])是否為 1。



檢查完畢後，就能傳送 Y[n] (0x3000084)到 FIR。



當測資都傳送完畢後，接下來就要開始丟到 Checkbits 去與 Testbench 做溝通，並檢查運算的結果是否正確。

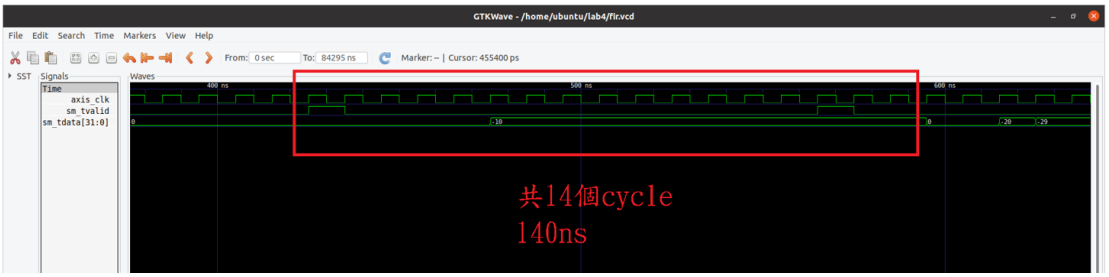


當結果都檢查完畢後，最後就會傳送 AB51，代表結束這一輪的測試。

五、What is the FIR engine theoretical throughput, i.e. data rate? Actually measured throughput?

Theoretical throughput: 1 ouput per 140ns

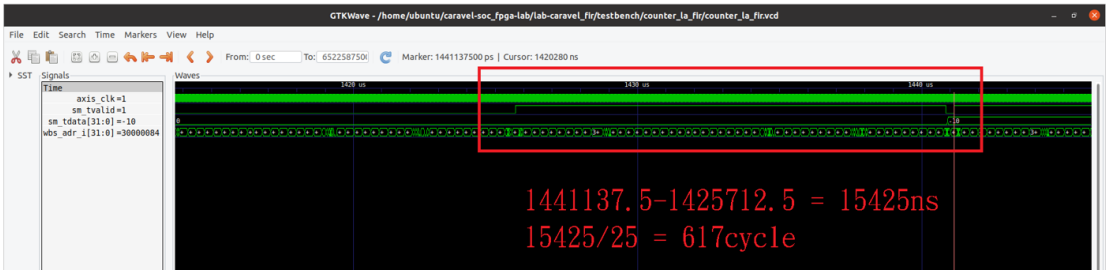
Theoretical data rate: 7.14M per output



FIR 運算的時間 從輸入 X[n]開始算起，大約需要 14 個 cycle 就能完成 Y[n]的計算。

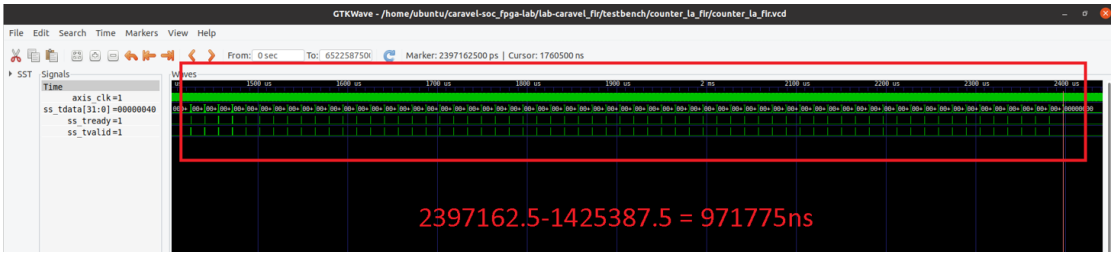
Actually measured throughput: 1 ouput per 15425ns

Actually measured data rate: 64.8K per output



實際上測量到的 Throughput 是大約 617 cycles / output，這部分與 Theoretical 有著很大的差異，而之所以造成 Theoretical 跟 Actually 會有這麼大的差異，從觀察到的結果來看，很大的原因是因為 CPU 在丟入 X[n]之後便會去執行其他指令，而 FIR 其實已經早就算好 Y[n] 等待著 CPU 接收，結果 CPU 過了很長的時間才去執行接收 Y[n]的指令，最後就導致它的 Data rate 變得非常低。

六、What is latency for firmware to feed data?

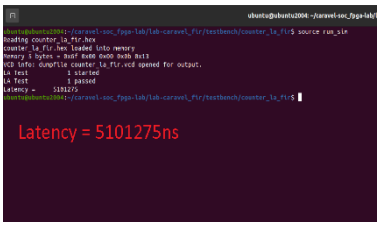


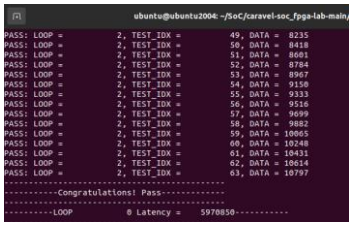
CPU 從第一筆 X[n]送到最後一筆 X[n]的 Latency 是 971775ns(38817cycles)。而每筆 X[n]的 Latency 則是 15425ns(617cycles)。

七、What techniques used to improve the throughput?

由於 Throughput 是被 CPU 所主導的，因此 Firmware Code 是怎麼寫的就很重要。

無 Check : 5101275ns　　VS　　有 Check:5970850ns





在傳送 X[n]、Y[n]前，如果有先去檢查 X[n] ready(0x3000000 [4])、Y[n] valid(0x3000000 [5])的話，CPU 勢必就需要多花一些指令去做檢查。

然而我們從波型上來看的話，其實在 FIR 運算的過程中，CPU 是會跑去執行其他程式的指令，而那個間隔是非常久的，久到超過 FIR 運算時間，因此在 CPU 傳送 X[N]、接收 Y[N]之前，有沒有去檢查 ready、valid 都不會影響運算的結果。

於是我們就拿有無 Check 來做比較，左邊是都不去檢查 直接傳送 X[N] 、接收 Y[N]，而右邊則是在每次傳送前，都會去檢查 FIR 是否處於可傳送狀態。

結果顯示，無 Check 的總體 Latency 有明顯較有 Check 的 Latency 低很多，因此在這個 case 中，如果想要 Throughput 增加，可以選擇不要去檢查 X[n] ready(0x3000000 [4])、Y[n] valid(0x3000000 [5])，而關於正確性則是沒有影響。

八、 Github Link

https://github.com/PatriChou/lab-caravel-fir.git