Matrix_mul Implements on the Vivado and SDK

一、IP 設計原理與相應規格

SIGNAL TABLE:

I/O			
signal	Bits width	port	description
clk	1	input	Clock
rstn	1	input	Reset negative
start	1	input	Start calculation
data_sizes	4	input	Input data_size. Like n*n, data_sizes is n.
rdata	16	input	Read data
ren	1	output	Read enable
raddr	4	output	Read address
wen	1	output	Write enable
wdata	16	output	Write data
finish	1	output	Work finish
state	5	output	Finite state machine states
input_data_num	9	output	Count input data number
out_data_num	9	output	Count output data number

CTR /Memory			
signal	Bits width	type	description
max_size		parameter	Multiplier number and can calculate max matrix size
end_input_A	1	reg	Ending state input_A
end_input_B	1	reg	Ending state input_B
end_calculate	1	reg	Ending state calculate
ready	1	reg	Data is valid and can write out
ready_temp0	1	reg	Temp ready. Connect to ready_temp1

ready_temp1	1	reg	Temp ready. Connect to ready_temp2
ready_temp2	1	reg	Temp ready. Connect to ready
mul_ending	1	reg	Mul state ending. All datas have multiplied
add_ending1	1	reg	Add state ending temp,connect to add_ending2
add_ending2	1	reg	Add state ending temp,connect to end_calculate
mul_numA_index		integer	Mul data row index offset
mul_numB_index		integer	Mul data col index offset
data_inA	16*16	reg	MatrixA data in
data_inB	16*16	reg	MatrixB data in
data_tmp	32*5	reg	Mul result
add_temp1	32*3	reg	Add result temp
add_temp2	32*2	reg	Add result temp
add_out	32	reg	Adder tree result

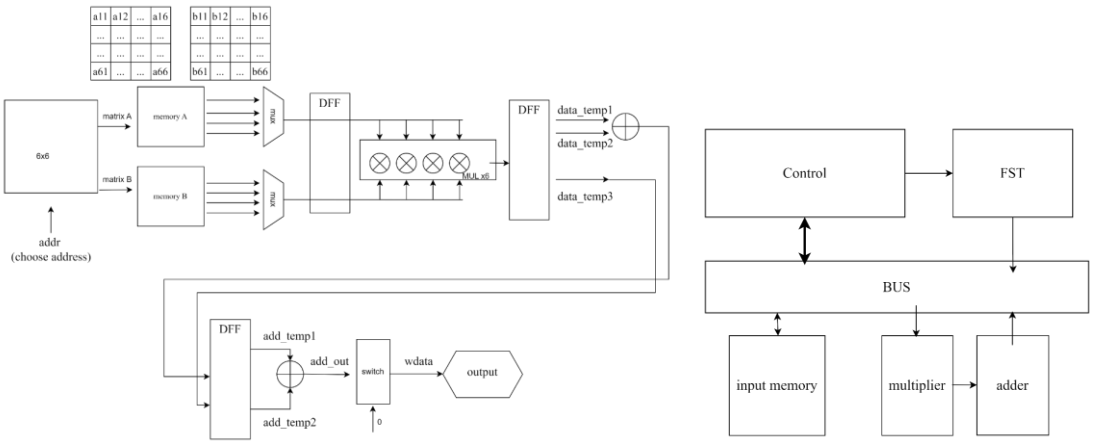
FST		
signal	type	description
IDLE	paremeter	Idle
INPUT_A	paremeter	Input matrix data A
INPUT_B	paremeter	Input matrix data B
CALCULATE	paremeter	Calculation and Write out
FINISH	paremeter	One calculation finish

規格:

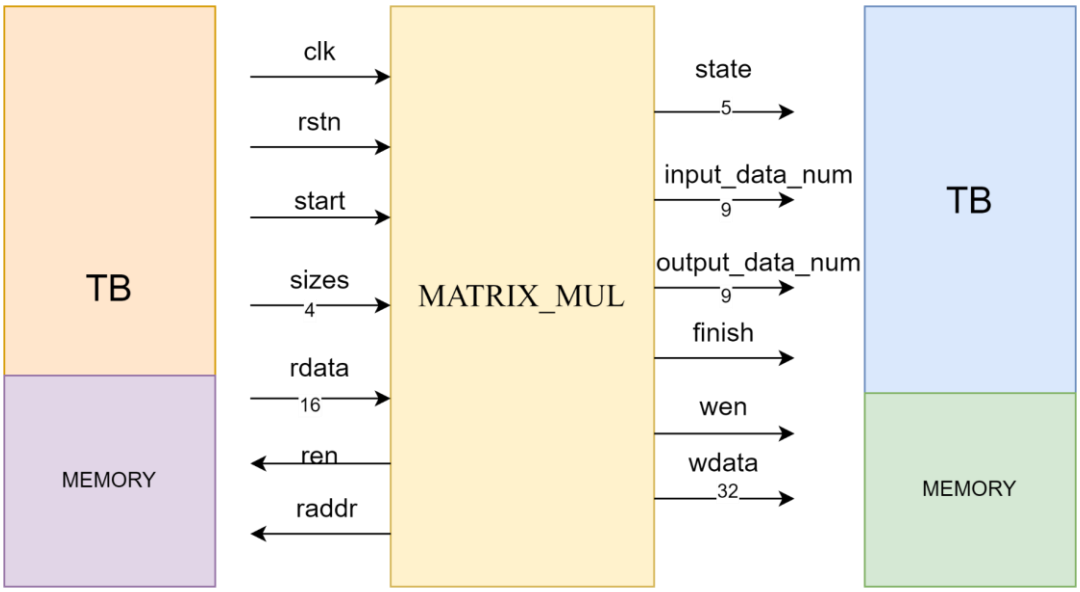
支援輸入資料	支援輸出資料	支援階層	Memory
16 bits	32 bits	1x1 - 6x6	6*6*36 bits x2
Mul	Adder	支援 Data type	BUS
6	4	有號數	AXI

IP 設計:

電路架構: 6x6 matrix_mul



I/O:

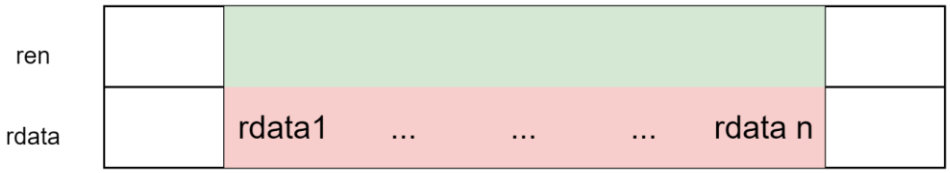


運作說明:

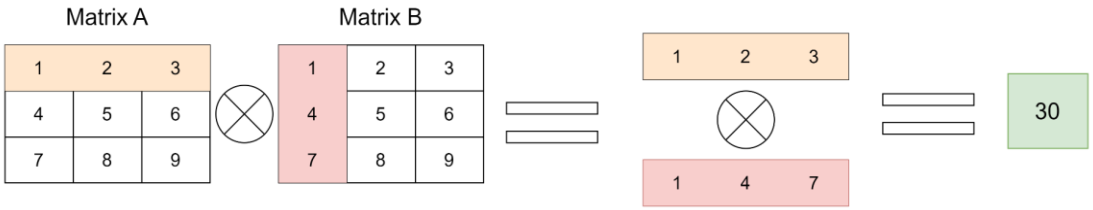
1.當 start 拉起 state 從 IDLE 進入 INPUT_A

2.ren 拉起，讀取 matrix A 的資料，由開頭依序每列輸入，當 input_data_num = data_size^2-1 ，state 轉換成 INPUT_B (輸入端為 data_inA[0]，shift register to data_inA[data_size-1])

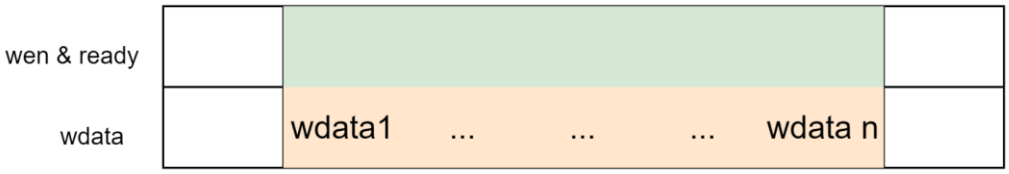
3.ren 保持拉起，讀取 matrix B 的資料，由開頭依序每列輸入，當 input_data_num = data_size^2-1 ，state 轉換成 CALCULATE (輸入端為 data_inB[0]，shift register to data_inB[data_size-1])



4.開始將資料從 dat_inA 和 data_inB 送入乘法器，data_inA 把每列的資料送入，data_inB 把每行的資料送入，index 由 out_indexA & out_indexB 進行控制



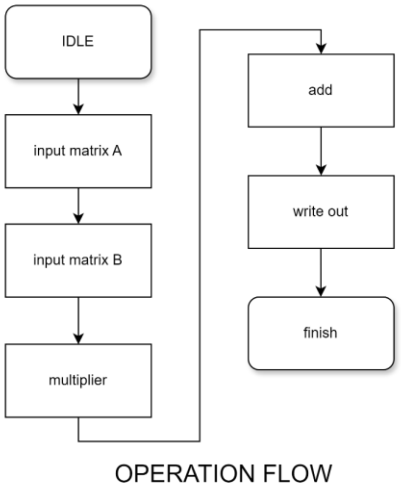
5.當第一筆資料送入乘法器時，ready temp 訊號拉起，並依循 pipeline 前進，最後在第一筆資料計算完成時同時拉起 ready，表示第一筆資料計算完成，可以進行輸出，使時 wen 拉起並開始輸出結果



6.當最後一筆資料送入乘法器時，ending 訊號拉起，並且隨著 pipeline 移動，在最後結果計算完成時拉起 end_calculate 訊號且 wen 放下，state 進入 finish 表示計算結束

7.state finish 在下個週期自動進入 IDLE 待機，如果 start 訊號沒有放下，則繼續計算下筆資料

流程圖:



流程: 讀取 matrix - > calculate -> output

設計原理

採傳統矩陣行列運算的規則，透過資料分配的方式實現對應元素的乘法，經過 pipeline adder tree 後輸出計算結果。在 state 為 calculate 時且 wen 訊號拉起，wdata 就會同步輸出。

Adder 採 pipeline 設計 adder tree，並在第一筆資料拉起 ready 以及最後一筆資料拉起 ending，分別用於 wen 始拉起的訊號以及輸出與計算階段結束訊號。

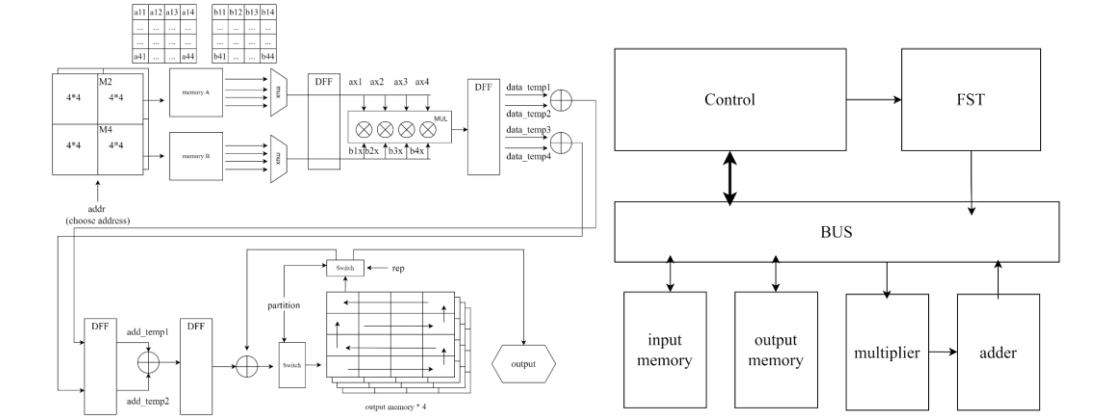
矩陣各元素相乘採利用 index 搬運對應的 matrix 資料給乘法器進行計算。

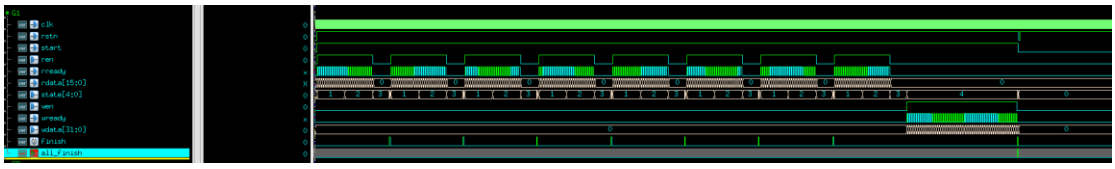
補充: 要增加支援的階層數，只要增加 input memory 大小、adder 數量以及乘法器數量，隨著階層數的增長，面積成長的幅度較大，但在低階層數的設計，可以在 cycle 以及 area 取得良好的平衡，面積不會太大。

改進方向:

可採用分割矩陣的方式，減少對 input memory 的需求量，但會需要 output memory，但總體面積增長是大幅下降。此外，此設計也需要增加重複計算的判斷，目前已成功實踐，但在階層自由度上會有所限制，如以四階矩陣當作分割矩陣的單位，則支援階層為 1x1 – 4x4、8x8 以及 4 的倍數(根據設計決定)。

架構: 8x8 partition matrix_mul





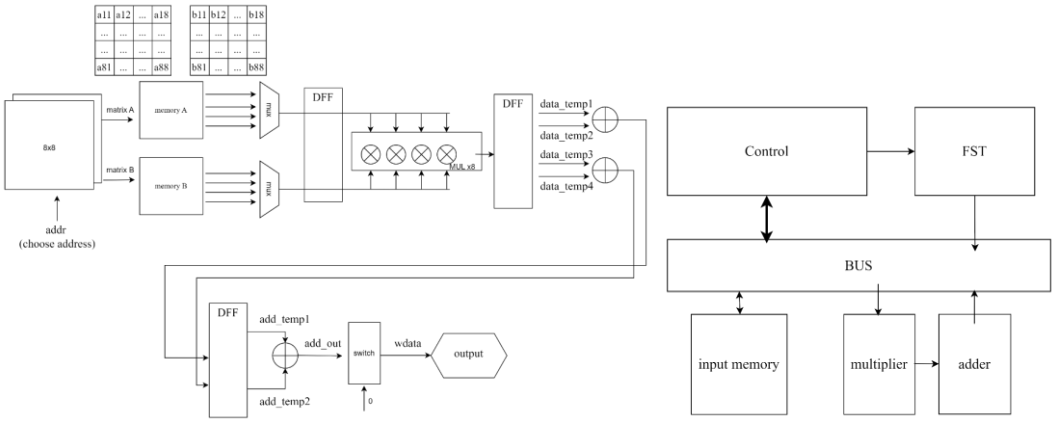
設計原理: 利用四階矩陣去實現 8x8 的分割矩陣運算，把 8x8 切成 4x4 進行運算，並透過疊加的方式實現矩陣運算。如上圖分成 8 個階段相乘與 1 個輸出。

功能:支援 1x1 – 4x4 & 8x8 矩陣的直接運算，5x5-7-7 可透過 8x8 矩陣 padding 後運算。

在此版本中有使用 handshake 機制，通過每次輸入拉起 rready 確認每個資料寫入有效性(data_in 要先準備好)，以及每次 wready 拉起確認每個輸出資料的有效性。同時這個設計是為了可以逐步寫入或讀出資料，不受讀寫時間限制，如果通過使用者操作 enter 輸入資料。

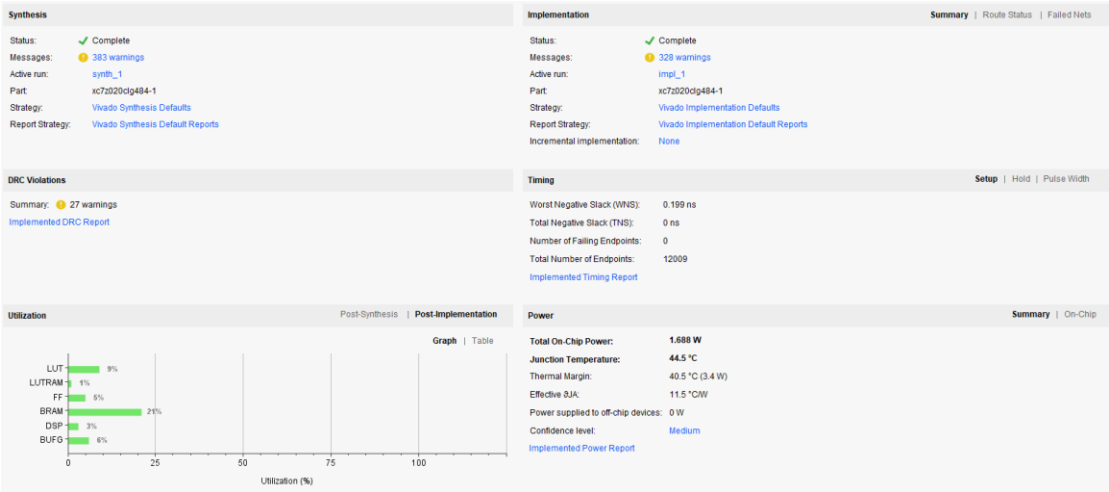
電路升級 8x8 矩陣:

架構



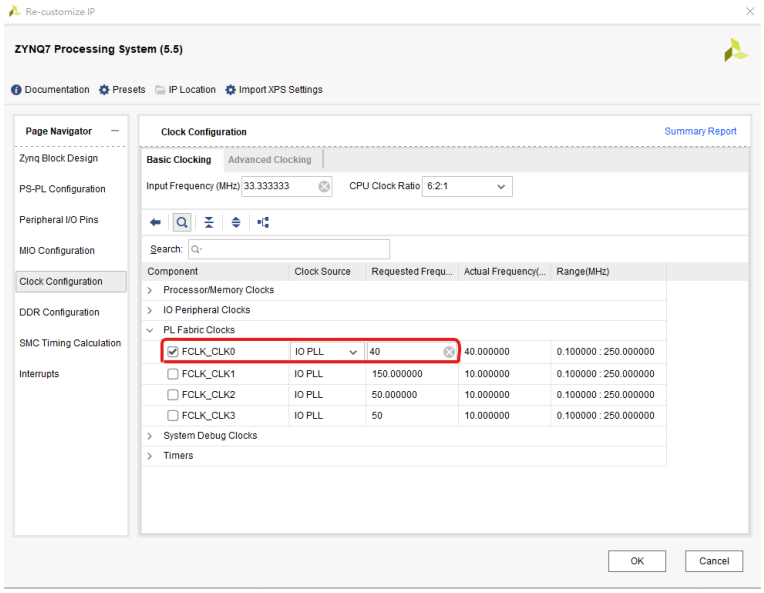
功能: 能夠支援 1x1 – 8x8 矩陣相乘

Overview:

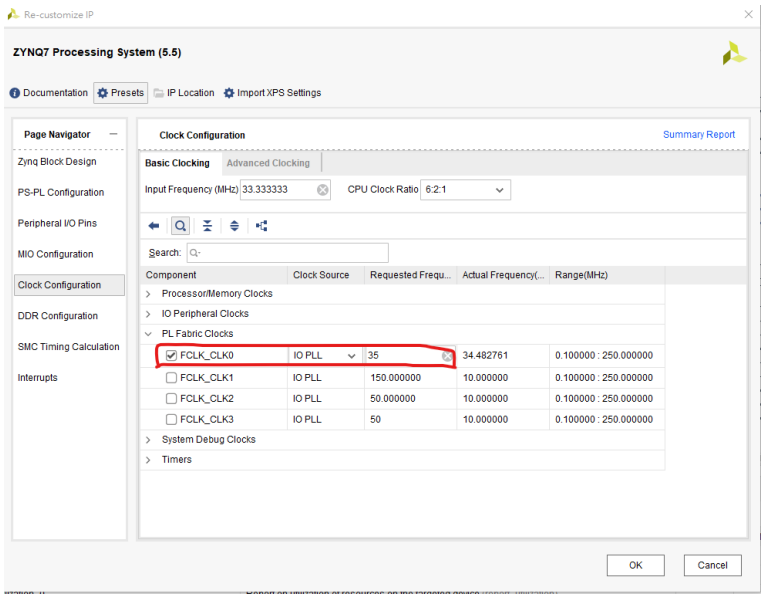


調整操作頻率以解決 Timing Violation

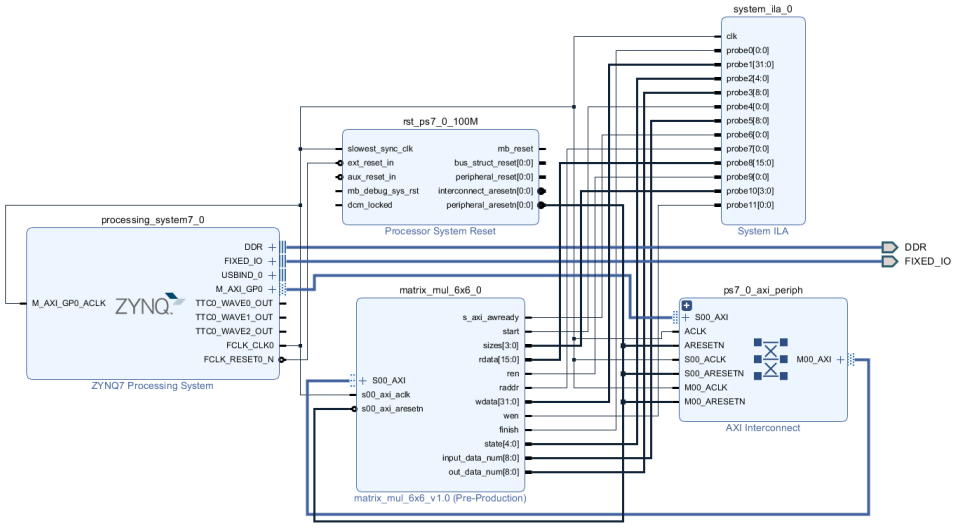
如果 wdata width = 16 bits，可在 40MHz 工作



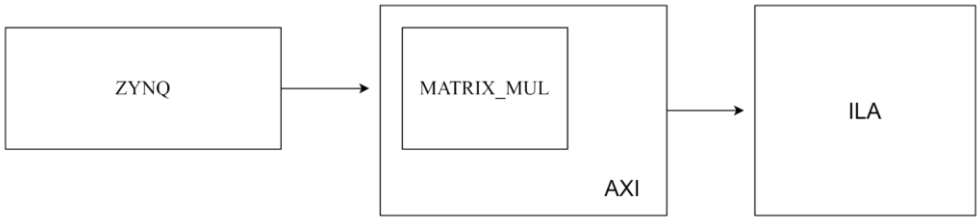
如果 wdata width = 32 bits，可在 35MHz 工作



Wrapper:



Vivado Block Design



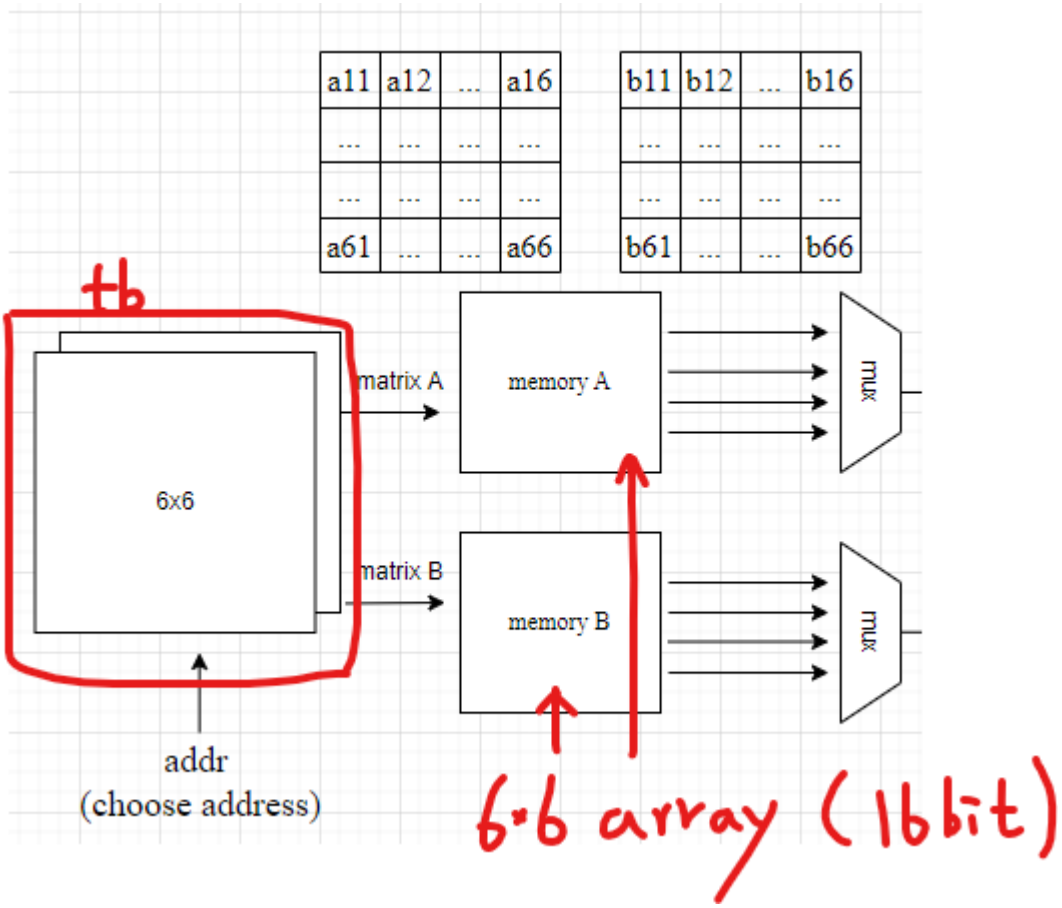
Wrapper Block Diagram

IP Package:

1.Matrix_mul connects to AXI Wrapped IP

2.Wrapped IP connects to CPU and ILA for waveform view

Memory Mapping:



AXI BUS DESIGN:

matrix_mul_6x6_v1_0

```
    // ILA
    output wire s_axi_awready,
    output wire start,
    output wire [3:0] sizes,
    output wire[15:0] rdata,
    output wire ren,
    output wire raddr,
    output wire [31:0] wdata,
    output wire wen,
    output wire finish,
    output wire [4:0] state,
    output wire [8:0] input_data_num,
    output wire [8:0] out_data_num
);
```

ILA port

```
    //ILA signal
    .s_axi_awready(s_axi_awready),
    .start(start),
    .sizes(sizes),
    .rdata(rdata),
    .ren  (ren),
    .raddr(raddr),
    .wdata(wdata),
    .wen  (wen),
    .finish(finish),
    .state(state),
    .input_data_num(input_data_num),
    .out_data_num(out_data_num)
);
```

接 matrix_mul_6x6_v1_0_S00_AXI port

matrix_mul_6x6_v1_0_S00_AXI

```
    // ILA
    output wire s_axi_awready,
    output wire start,
    output wire [3:0] sizes,
    output wire[15:0] rdata,
    output wire ren,
    output wire raddr,
    output wire [31:0] wdata,
    output wire wen,
    output wire finish,
    output wire [4:0] state,
    output wire [8:0] input_data_num,
    output wire [8:0] out_data_num

);
```

ILA port

```
// Address decoding for reading registers
case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
  4'h0   : reg_data_out <= slv_reg0;
  4'h1   : reg_data_out <= slv_reg1;
  4'h2   : reg_data_out <= slv_reg2;
  4'h3   : reg_data_out <= ren;
  4'h4   : reg_data_out <= raddr;
  4'h5   : reg_data_out <= wen;
  4'h6   : reg_data_out <= wdata;
  4'h7   : reg_data_out <= finish;
  4'h8   : reg_data_out <= state;
  4'h9   : reg_data_out <= input_data_num;
  4'hA   : reg_data_out <= out_data_num;
  4'hB   : reg_data_out <= slv_reg11;
  4'hC   : reg_data_out <= slv_reg12;
  4'hD   : reg_data_out <= slv_reg13;
  4'hE   : reg_data_out <= slv_reg14;
  4'hF   : reg_data_out <= slv_reg15;
  default : reg_data_out <= 0;
endcase
```

AXI register setting

```
// Add user logic here
assign s_axi_awready = (S_AXI_AWREADY == 1) ? 1:0;
assign start = slv_reg0;
assign sizes = slv_reg1;
assign rdata = slv_reg2;

matrix_mul u_matrix_mul(
        .clk   (S_AXI_ACLK  ),
        .rstn  (S_AXI_ARESETN ),
        .start(start),
        .sizes(sizes),
        .rdata(rdata),
        .ren   (ren),
        .raddr(raddr),
        .wdata(wdata),
        .wen   (wen),
        .finish(finish),
        .state(state),
        .input_data_num(input_data_num),
        .out_data_num(out_data_num)
       );
// User Logic ends
```

接 matrix_mul port

二、運算結果與驗證

SDK 設計:

指定起始位置

matrix_mul_6x6_0	0x43c00000	0x43c0ffff	S00_AXI

```
volatile int *pointer = (int*)0x43c00000;
```

相關訊號控制

```
pointer[0] = 1; // start
pointer[1] = 6; // data sizes
                // modify this signal can change simulated matrix hierarchy
pointer[2] = 0; // rdata
```

start

階層

rdata initializes

基本只操作 pointer[1]，其為 data size，可以決定測試的階層數。

Start = 1 電路開始運作，rdata = 0 為初始化輸入數據。

資料輸入

```
while(1){
    // only do one time
    if(pointer[8] == 1 || pointer[8] == 2){
        for(i=0;i< pointer[1] * pointer[1] ;i++){

            pointer[2] = i+1;
            usleep(0.01);  // delay
        }
        print("END1 \n\r");
        break;
    }
}
```

1.While 創造資料輸入的時序可能

2.通過 for loop 鎖定連續輸入，解決因為 AXI 會出現 handshake 的傳輸延遲。

3.用 usleep 創造延遲讓輸入訊號可以分開，中間的數據都會被省略，不會被輸入。Delay 大小需要測試，我盡量挑每個訊號錯開的那組數據。

4.此作法 delay 有其極限，再小電路也不會有相應的延遲效果。此外，因為電路讀取資料的 clock 很短，所以很多時候 input 階段的 rdata 都是同一值(因為 delay 仍太大)。但仍有些地方 rdata 訊號會交錯，可以做一些簡單的驗證。

5. break 用於輸入一組數據完就結束(但因延遲問題，這個功能沒有被完全實現)

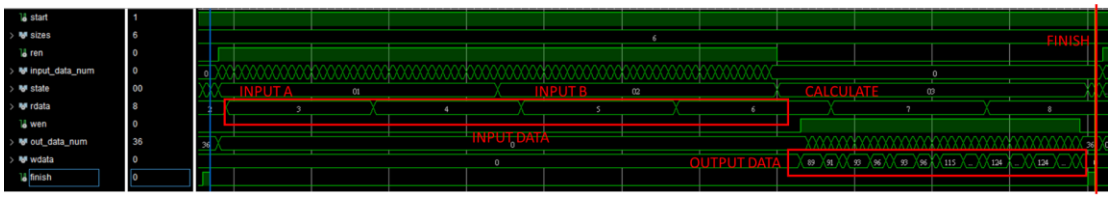
運作流程: 資料輸入 -> 結束，觀看波型找到相應的輸入部分，確認輸入與輸出資料的正確性。

驗證方式

由 vivado sdk input data, 觀察 ILA 波型，看看 input A 、input B 的訊號輸入有哪些(我會調整 delay 大小，盡量讓訊號間有重疊，創造不一樣的輸入)，接著將其輸入 matlab 計算結果與 wen 拉起的 wdata 去做比對。

運算結果與驗證

ARRAY: 6x6



INPUT: 由左而右，由上而下

Matrix A:

2	3	3	3	3	3
3	3	3	3	3	3
3	3	3	3	3	3
3	3	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4

Matrix B:

4	4	4	5	5	5
5	5	5	5	5	5
5	5	5	5	5	5
5	5	5	5	5	6
6	6	6	6	6	6
6	6	6	6	6	6

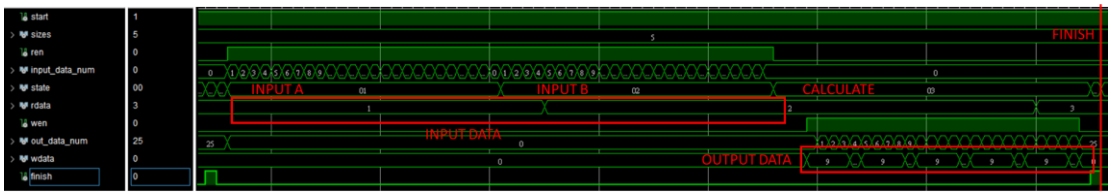
OUTPUT: 由左而右，由上而下

89	89	89	91	91	94
93	93	93	96	96	99
93	93	93	96	96	99
115	115	115	118	118	122
124	124	124	128	128	132

124	124	124	128	128	132

計算正確!

ARRAY: 5x5



INPUT:

Matrix A:

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

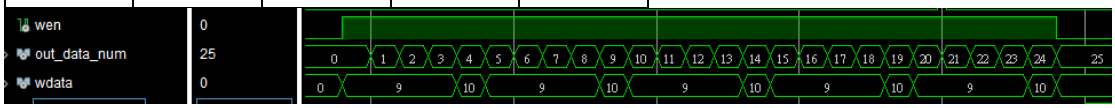
Matrix B:

1	1	1	2	2
2	2	2	2	2
2	2	2	2	2
2	2	2	2	2
2	2	2	2	2

OUTPUT:

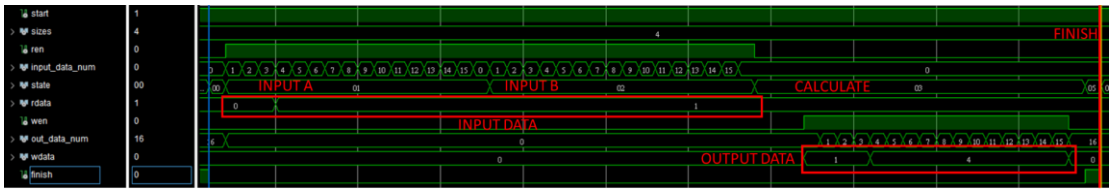
9	9	9	9	10

9	9	9	9	10
9	9	9	9	10
9	9	9	9	10
9	9	9	9	10



計算正確!

ARRAY: 4x4



INPUT:

Matrix A:

0	0	0	1
1	1	1	1
1	1	1	1
1	1	1	1

Matrix B:

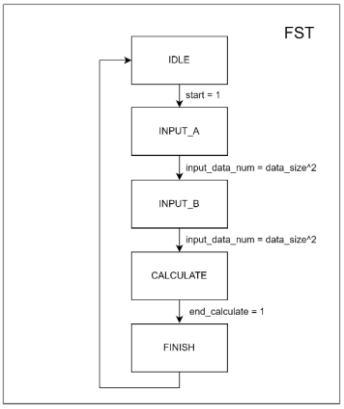
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

OUTPUT:

1	1	1	1	
4	4	4	4	
4	4	4	4	
4	4	4	4	

計算正確!

FST



IDLE	待機，start 拉起時始運作	
	轉換條件	Start = 1
INPUT_A	輸入 Matrix A data。One-by-one input data	
	轉換條件	Input_data_num = data_size^2
INPUT_B	輸入 Matrix B data。One-by-one input data	
	轉換條件	Input_data_num = data_size^2
CALCULATE	每週期計算，當 wen 拉起時輸出 wdata	
	轉換條件	end_calculate 拉起(由最後輸入的資料決定)
FINISH	運算結束，返回 IDLE	
	轉換條件	無條件轉到 IDLE

三、問題與討論

1. AXI 只能用 output wire or input wire，其餘會發生找不到 D pin 的錯誤
2. 確認 port 有沒有空接，不然可能在合成會出現抱錯(概率出現)
3. Power 過高會導致合成失敗
4. 在 design 可使用的操作頻率，在 vivado 很多時候做不到，如這次的作業，design 可以跑到 142MHz，但 vivado 跑在 35-40MHz。