MSoC self-paced learning project_2
FP_ACCUMULATOR

R09943017 林奕廷

1. Introduction

這項專案的目的是藉由Vivado HLS來實作一個浮點數累加器。Top function在fp_accum.cpp檔中定義。原本的code主要如下：

```
float hls_fp_accumulator(float window[NUM_ELEM])
{

    float result = 0.0;

    L1:for(unsigned char x=0; x<NUM_ELEM;x++)
    {
        result = result + window[x];
    }

    return result;
}
```

第一種方法將累加的過程再時間上展開，每個cycle把input的window加到result，最後輸出。

```
float hls_fp_accumulator(float window0[NUM_ELEM])
{

    float window1[NUM_ELEM/2] = {0.0};
    float window2[NUM_ELEM/4] = {0.0};
    float window3[NUM_ELEM/8] = {0.0};
    float window4[NUM_ELEM/16]= {0.0};
    float window5[NUM_ELEM/32]= {0.0};
    float window6[NUM_ELEM/64]= {0.0};

    float result = 0.0;

    L1: for(ap_uint<7> x=0; x.to_uint()<NUM_ELEM/2; x++)
    {
#pragma HLS PIPELINE
         window1[x] = window0[x] +  window0[NUM_ELEM/2+x];
    }
    L2: for(ap_uint<7> x=0; x.to_uint()<NUM_ELEM/4; x++)
    {
#pragma HLS PIPELINE

         window2[x] = window1[x] +  window1[NUM_ELEM/4+x];
    }
    L3: for(ap_uint<7> x=0; x.to_uint()<NUM_ELEM/8; x++)
    {
#pragma HLS PIPELINE

         window3[x] = window2[x] +  window2[NUM_ELEM/8+x];
    }
    L4: for(ap_uint<7> x=0; x.to_uint()<NUM_ELEM/16; x++)
    {
#pragma HLS PIPELINE

         window4[x] = window3[x] +  window3[NUM_ELEM/16+x];
    }
    L5: for(ap_uint<7> x=0; x.to_uint()<NUM_ELEM/32; x++)
    {
#pragma HLS PIPELINE

         window5[x] = window4[x] +  window4[NUM_ELEM/32+x];
    }

 #pragma HLS PIPELINE

          window6[x] = window5[x] +  window5[NUM_ELEM/64+x];
     }

     result = window6[0] + window6[1];

     return result;
 }

 #endif
```

第二種方法則是將累加的過程平行展開。此做法開了另外6個window，以adder tree的方式進行。

2. HLS C-simulation

透過C-simulation執行資料夾中提供的testbench

```
int main(void)
{
    int x, y;
    int ret_val = 0;

    float ref_window[NUM_ELEM];
    float hls_window[NUM_ELEM];
    float threshold = ((float)1.0)/1024;

    for (x=0; x < NUM_ELEM; x++)
    {
            ref_window[x] = (65536)*PseudoCasual();
            hls_window[x] = ref_window[x] ;
    }
    // REF
    float ref_res  = ref_fp_accumulator(ref_window);
    // DUT
    float hls_res  = hls_fp_accumulator(hls_window);
    // check results
    float total_error = 9.5367e-07f;

    float diff = ref_res - hls_res;
    if (diff < threshold) diff = 0-diff; // take absolute value
    if (diff > threshold)
    {
        total_error += (float) diff;
    }
    printf("\n%010.4f\t%010.4f\t%010.4f\n", ref_res, hls_res, total_error);
    if (total_error < 1.0)
    {
        ret_val=0;
        printf("TEST OK!\n");
    }
    else
    {
        ret_val=1;
        printf("TEST FAILED!\n");
    }
    return ret_val;

}
```

得到的結果如下所示：

```
Starting C simulation ...
/home/itlin/xilinx/Vivado/2019.2/bin/vivado_hls /home/itlin/Desktop/109-1/MSOC/FP_ACCUM/solution1/csim.tcl
INFO: [HLS 200-10] Running '/home/itlin/xilinx/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado_hls'
INFO: [HLS 200-10] For user 'itlin' on host 'itlin-ubuntu-18-04' (Linux_x86_64 version 5.4.0-54-generic) on Thu Dec 24 19:56:25 CST 2020
INFO: [HLS 200-10] On os Ubuntu 18.04.5 LTS
INFO: [HLS 200-10] In directory '/home/itlin/Desktop/109-1/MSOC'
Sourcing Tcl script '/home/itlin/Desktop/109-1/MSOC/FP_ACCUM/solution1/csim.tcl'
INFO: [HLS 200-10] Opening project '/home/itlin/Desktop/109-1/MSOC/FP_ACCUM'.
INFO: [HLS 200-10] Opening solution '/home/itlin/Desktop/109-1/MSOC/FP_ACCUM/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Setting target device to 'xc7z020-clg484-1'
INFO: [SIM 211-2] *************** CSIM start ***************
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
   Compiling ../../../../HLx_Examples-master/Math/fp_accum/fp_accum.cpp in debug mode
   Generating csim.exe


3670058.0000    3670058.5000    00000.5000
TEST OK!
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

3. HLS Synthesis

Adder tree版本的合成結果如下：

Synthesis Report for 'hls_fp_accumulator'

General Information

Date:	Thu Dec 24 19:58:13 2020
Version:	2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project:	FP_ACCUM
Solution:	solution1
Product family:	zynq
Target device:	xc7z020-clg484-1

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	9.140 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
301	301	3.010 us	3.010 us	301	301	none

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	816	-
FIFO	-	-	-	-	-
Instance	-	2	205	390	-
Memory	4	-	128	12	0
Multiplexer	-	-	-	609	-
Register	0	-	1778	352	-
Total	4	2	2111	2179	0
Available	280	220	106400	53200	0
Utilization (%)	1	~0	1	4	0

Detail

- Instance
- DSP48E
- Memory
- FIFO
- Expression
- Multiplexer
- Register

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_rst	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_start	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_done	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_idle	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_ready	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_return	out	32	ap_ctrl_hs	hls_fp_accumulator	return value
window0_address0	out	7	ap_memory	window0	array
window0_ce0	out	1	ap_memory	window0	array
window0_q0	in	32	ap_memory	window0	array
window0_address1	out	7	ap_memory	window0	array
window0_ce1	out	1	ap_memory	window0	array
window0_q1	in	32	ap_memory	window0	array

Export the report(.html) using the Export Wizard

Open Analysis Perspective Analysis Perspective

第一種方法(時間上累加)的結果如下：

Synthesis Report for 'hls_fp_accumulator'

General Information

Date: Thu Dec 24 20:03:20 2020
Version: 2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project: FP_ACCUM
Solution: solution2
Product family: zynq
Target device: xc7z020-clg484-1

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.256 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
897	897	8.970 us	8.970 us	897	897	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	28	-
FIFO	-	-	-	-	-
Instance	-	2	205	390	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	62	-
Register	-	-	88	-	-
Total	0	2	293	480	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	~0	~0	0

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_rst	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_start	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_done	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_idle	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_ready	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_return	out	32	ap_ctrl_hs	hls_fp_accumulator	return value
window_address0	out	7	ap_memory	window	array
window_ce0	out	1	ap_memory	window	array
window_q0	in	32	ap_memory	window	array

Export the report(.html) using the Export Wizard

Open Analysis Perspective Analysis Perspective

觀察上面兩種實作方式的合成結果可以看出兩種方法是resource和latency的tradeoff。Adder tree由於是空間上的展開因此鎖需要的運算單元相較於第一種方式還要來得多，也因此需要的執行時間比較少。

4. Cosimulation

執行Cosimulation得結果如下：

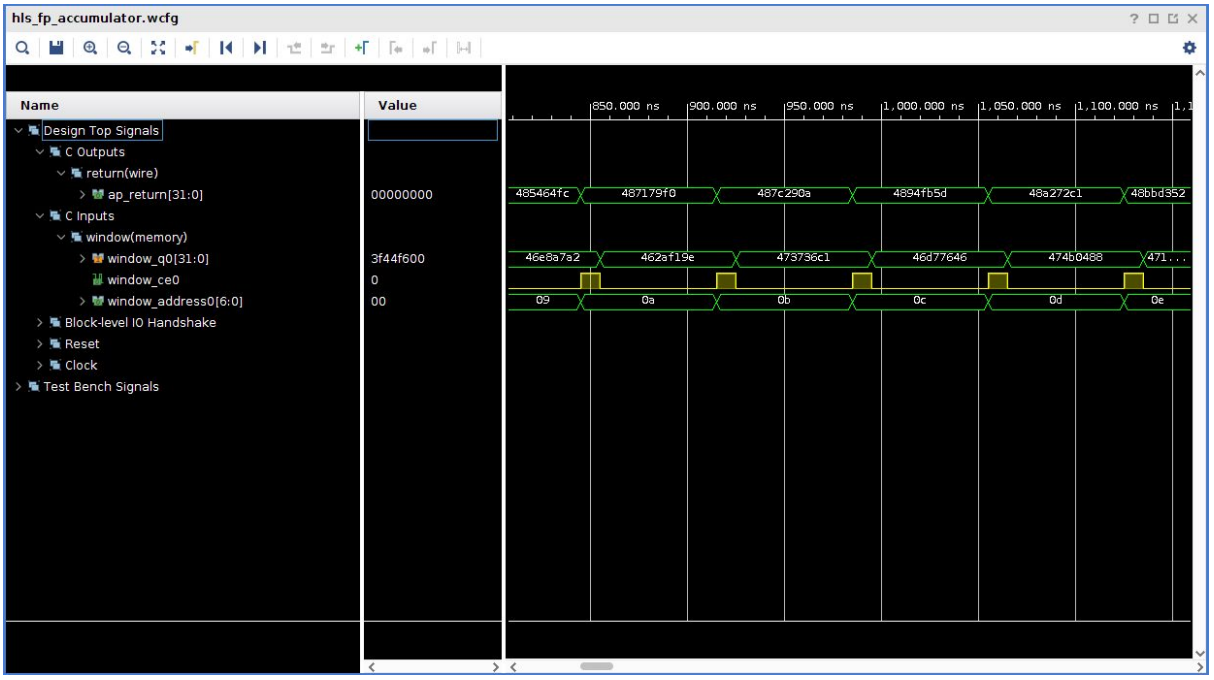
Cosimulation Report for 'hls_fp_accumulator'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	897	897	897	NA	NA	NA

Export the report(.html) using the Export Wizard

波形如下圖所示(一部分截圖)：



5. Improvement

```
float hls_fp_accumulator(float window[NUM_ELEM])
{

    float result = 0.0;
    float tmp_result_a = 0.0;
    float tmp_result_b = 0.0;
    float tmp_result_c = 0.0;
    float tmp_result_d = 0.0;
#pragma HLS array_partition variable=window block factor=4 dim=1
    L1:for(unsigned char x=0; x<NUM_ELEM/4;x++)
    {
#pragma HLS pipeline II=1
        tmp_result_a = tmp_result_a + window[4 * x];
        tmp_result_b = tmp_result_b + window[4 * x + 1];
        tmp_result_c = tmp_result_c + window[4 * x + 2];
        tmp_result_c = tmp_result_c + window[4 * x + 3];
    }
    result = tmp_result_a + tmp_result_b + tmp_result_c + tmp_result_d;
    return result;
}
```

這樣的作法會在空間上平行展開成四份硬體，最後在把這些partial sum加在一起。如此一來整體的運算時間就可以縮短成1/4。為了要達成這樣的平行度，

Input的array需要partition，才能夠在每次的運算提供四個數值，達成需求。如此優化的結果如下：

Synthesis Report for 'hls_fp_accumulator'

General Information

Date:	Thu Dec 24 20:27:19 2020
Version:	2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project:	FP_ACCUM
Solution:	solution3
Product family:	zynq
Target device:	xc7z020-clg484-1

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	14.512 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
268	268	3.889 us	3.889 us	268	268	none

Detail

- Instance
- Loop

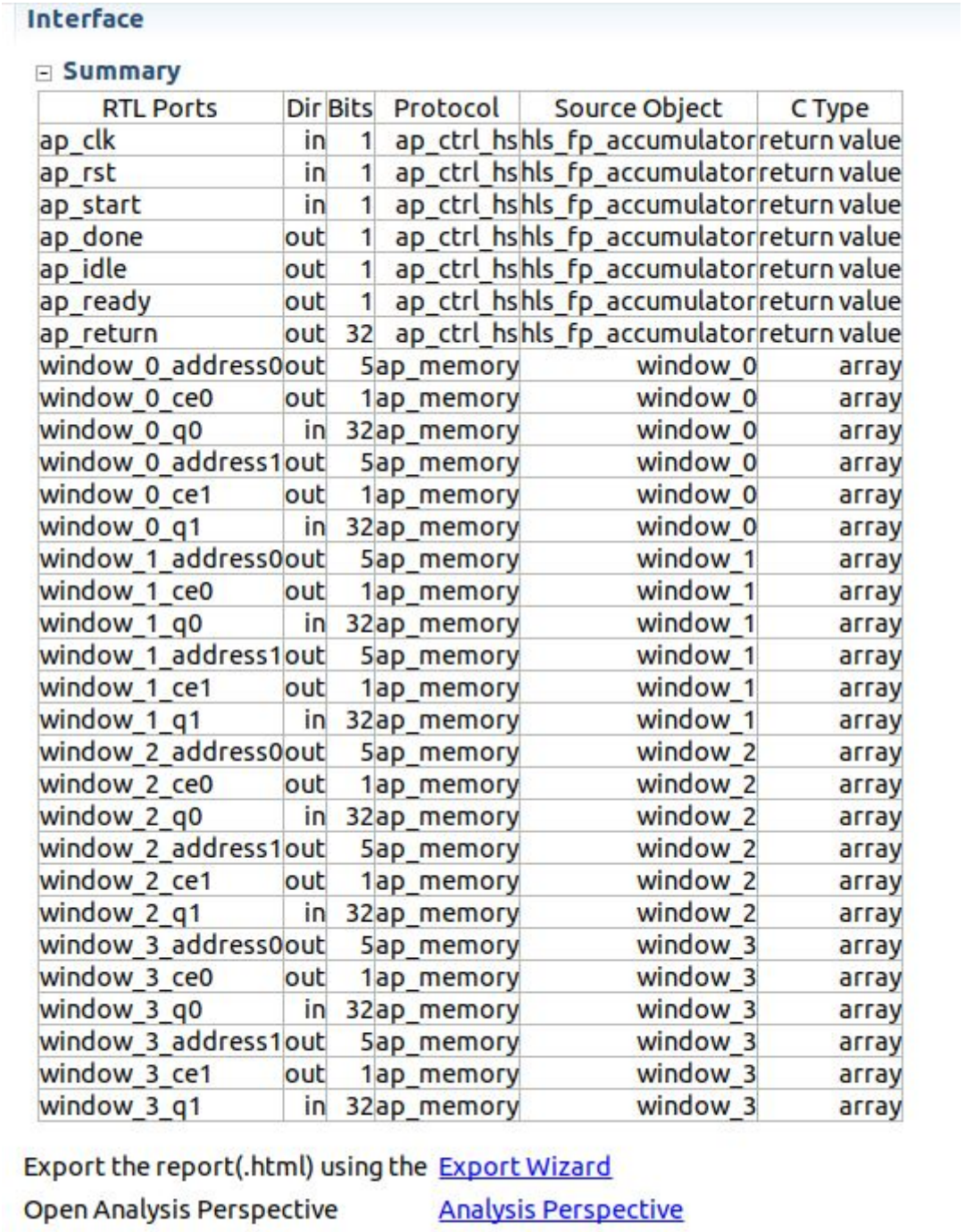
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	43	-
FIFO	-	-	-	-	-
Instance	-	4	410	822	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	386	-
Register	-	-	362	-	-
Total	0	4	772	1251	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	2	0

Detail

- Instance
- DSP48E
- Memory
- FIFO
- Expression
- Multiplexer
- Register



Interface

Summary

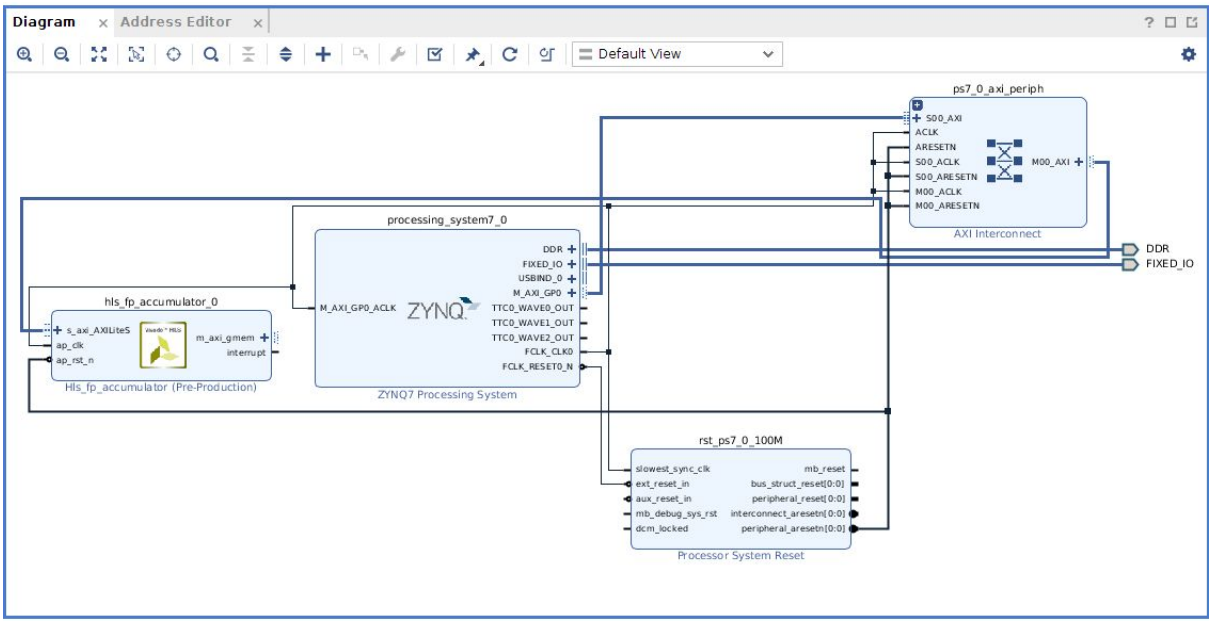
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_rst	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_start	in	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_done	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_idle	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_ready	out	1	ap_ctrl_hs	hls_fp_accumulator	return value
ap_return	out	32	ap_ctrl_hs	hls_fp_accumulator	return value
window_0_address0	out	5	ap_memory	window_0	array
window_0_ce0	out	1	ap_memory	window_0	array
window_0_q0	in	32	ap_memory	window_0	array
window_0_address1	out	5	ap_memory	window_0	array
window_0_ce1	out	1	ap_memory	window_0	array
window_0_q1	in	32	ap_memory	window_0	array
window_1_address0	out	5	ap_memory	window_1	array
window_1_ce0	out	1	ap_memory	window_1	array
window_1_q0	in	32	ap_memory	window_1	array
window_1_address1	out	5	ap_memory	window_1	array
window_1_ce1	out	1	ap_memory	window_1	array
window_1_q1	in	32	ap_memory	window_1	array
window_2_address0	out	5	ap_memory	window_2	array
window_2_ce0	out	1	ap_memory	window_2	array
window_2_q0	in	32	ap_memory	window_2	array
window_2_address1	out	5	ap_memory	window_2	array
window_2_ce1	out	1	ap_memory	window_2	array
window_2_q1	in	32	ap_memory	window_2	array
window_3_address0	out	5	ap_memory	window_3	array
window_3_ce0	out	1	ap_memory	window_3	array
window_3_q0	in	32	ap_memory	window_3	array
window_3_address1	out	5	ap_memory	window_3	array
window_3_ce1	out	1	ap_memory	window_3	array
window_3_q1	in	32	ap_memory	window_3	array

Export the report(.html) using the Export Wizard

Open Analysis Perspective Analysis Perspective

從latency中可以看出我的作法相較於第一種和第二種的實作方式能夠有效降低整體的執行時間。第二種方式原本的code在沒有加上pragma的情況下會合成出BRAM，造成一個cycle最多只能計算兩個值相加，增加整體latency。若把第二種方式再加上array partition的pragma就可以解決此問題。此外，我使用的硬體資源也比第二種方式還要來得少。

6. System block diagram



此圖為系統的架構圖。IP使用AXILite的方式設定register參數。另外PL和PS端使用AXIMaster的界面進行資料傳輸。

```
#pragma HLS INTERFACE s_axilite port=output
#pragma HLS INTERFACE m_axi depth=128 port=window offset=slave
#pragma HLS INTERFACE s_axilite port=return
```

Host program執行的結果如下：

```
Entry: /usr/lib/python3/dist-packages/ipykernel_launcher.py
System argument(s): 3
Start of "/usr/lib/python3/dist-packages/ipykernel_launcher.py"
============================
Kernel execution time: 0.0001461505889892578 s
software sum:  64.0406
hardware sum:  64.04058837890625
total error:  0.0
TEST OK!

============================
Exit process
```

7. Github submission

project中產生的.bit, .hwh和host program皆放在github中：
https://github.com/Lin0611/MSOC_1091_self_paced/blob/main/README.md