Xilinx HLS Example – Squared_Difference_Accumulator

Self-paced Report
R09943020 電子所碩一 杜承諺

1. HLS C-sim/Synthesis/Cosim (Screenshot + brief intro)

系統介紹:

Input
- 為兩個長度為 N 的 int16 array A, B

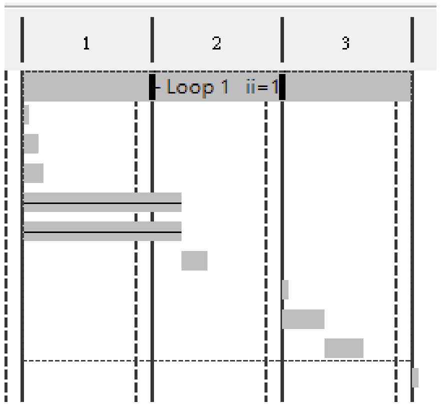
Output
- 為 A, B 中每個值相減平方後累加的結果

```
typedef ap_int<16> din_t;
typedef ap_int<48> dout_t;

#define N 10

void diff_sq_acc(din_t a[N], din_t b[N], dout_t *dout);
```

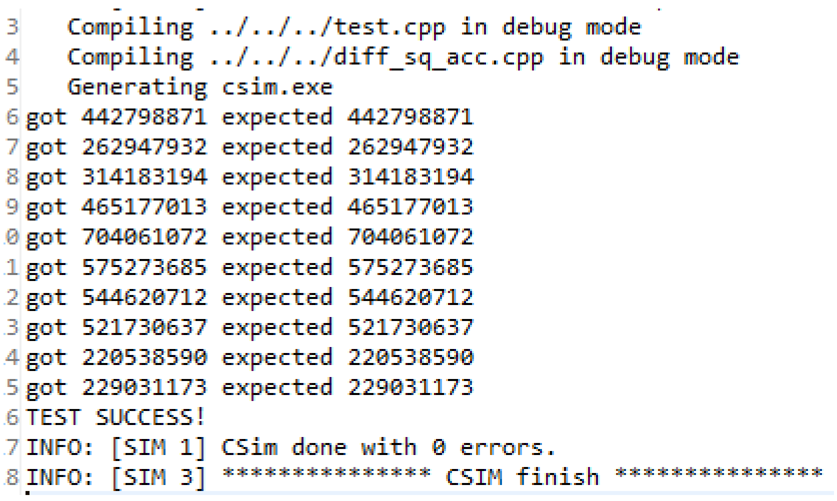
Solution1 (source code version)
- 沒有加入 axi 的介面，純粹的 HLS 實作
- 用 pipeline 減少 latency

```
for(i=0; i<N; i++)
{
    #pragma HLS PIPELINE II=1

    a_reg  = a[i];
    b_reg  = b[i];
    sub = a_reg - b_reg;
    sub2 = sub*sub;
    acc += sub2;
}
```

- 從 Timeline analysis 中也可以看到 ii = 1

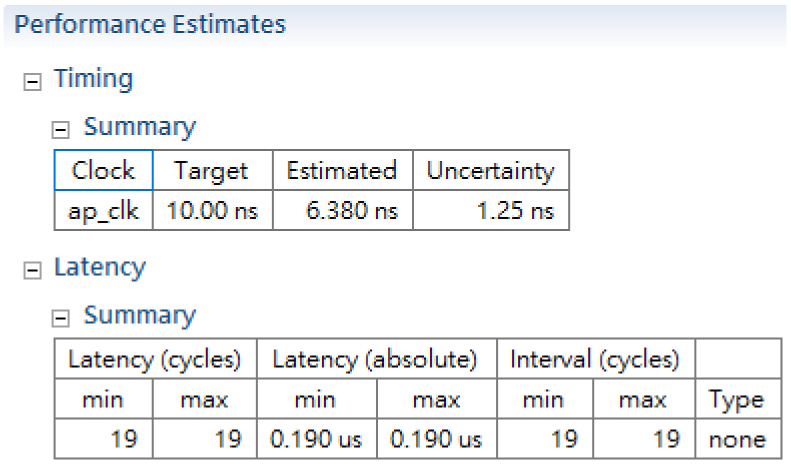


C-sim



```
   Compiling ../../../test.cpp in debug mode
   Compiling ../../../diff_sq_acc.cpp in debug mode
   Generating csim.exe
got 442798871 expected 442798871
got 262947932 expected 262947932
got 314183194 expected 314183194
got 465177013 expected 465177013
got 704061072 expected 704061072
got 575273685 expected 575273685
got 544620712 expected 544620712
got 521730637 expected 521730637
got 220538590 expected 220538590
got 229031173 expected 229031173
TEST SUCCESS!
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Synthesis



Performance Estimates

Timing

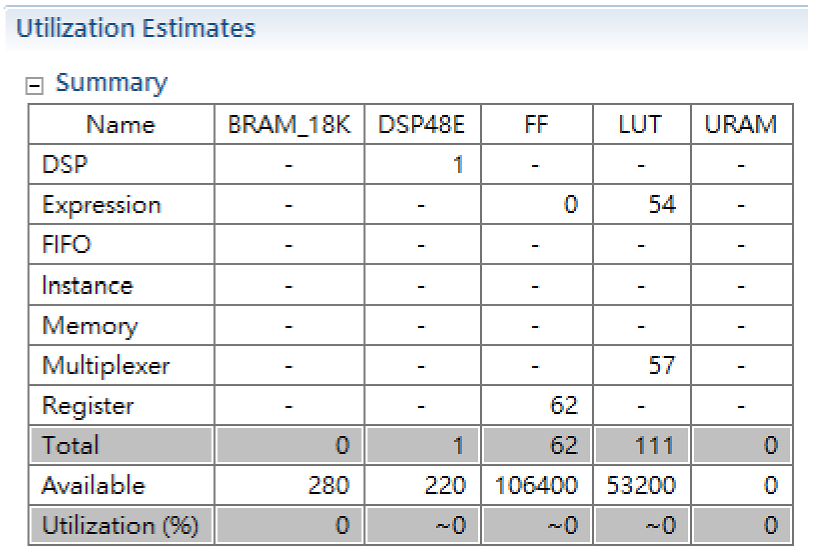
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.380 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
19	19	0.190 us	0.190 us	19	19	none

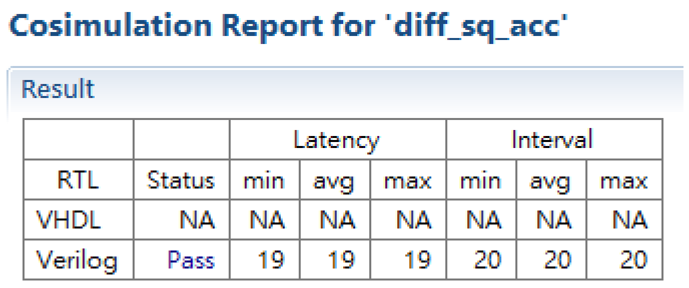


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	54	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	57	-
Register	-	-	62	-	-
Total	0	1	62	111	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	~0	~0	0

Co-sim



Cosimulation Report for 'diff_sq_acc'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	19	19	19	20	20	20

2. Improvement throughput/area

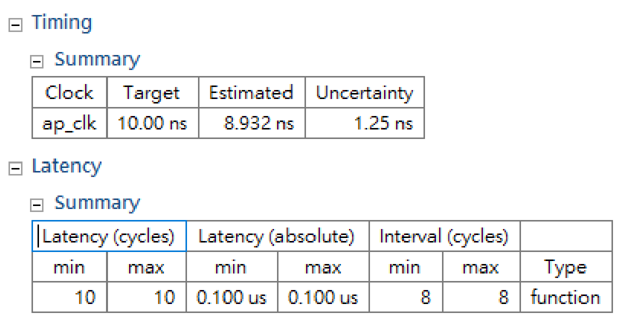
我嘗試了三種不同的做法

Solution2 (使用 UNROLL 來平行計算)

- 將 pipeline pragma 移到迴圈外面，等效於 Unroll 整個 loop

```
#pragma HLS PIPELINE II=1
    for(i=0; i<N; i++)
    {
        a_reg  = a[i];
        b_reg  = b[i];
        sub = a_reg - b_reg;
        sub2 = sub*sub;
        acc += sub2;
    }
```

- 我們預期 unroll 後只需要 2 cycle 來進行計算，不過 HLS 合成結果卻顯示需要 10 cycle



Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.932 ns	1.25 ns

Latency

Summary

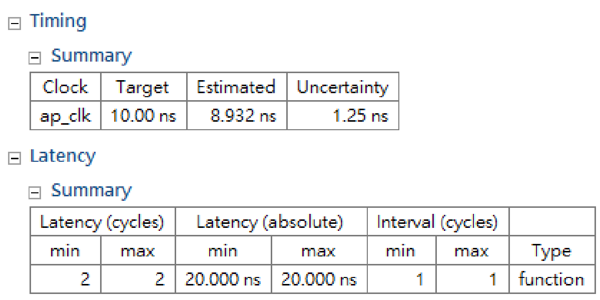
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
10	10	0.100 us	0.100 us	8	8	function

Solution3 (使用 UNROLL + ARRAY PARTITION)

- Sol 2 無法達到平行，其中原因來自 A, B array 的 bandwidth 不夠，於是我對 A, B 做 Array partition

```
#pragma HLS ARRAY_PARTITION variable=b complete dim=1
#pragma HLS ARRAY_PARTITION variable=a complete dim=1
```

- 經過這樣的處理後，即可將 latency 縮到最短



Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.932 ns	1.25 ns

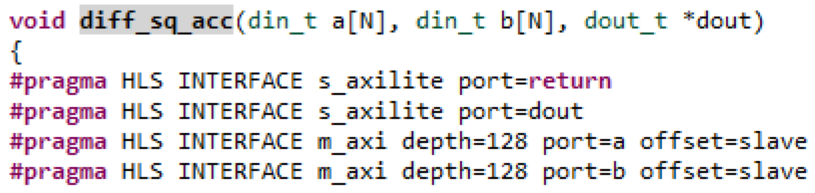
Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	20.000 ns	20.000 ns	1	1	function

Solution4 (solution1 + AXI interface)

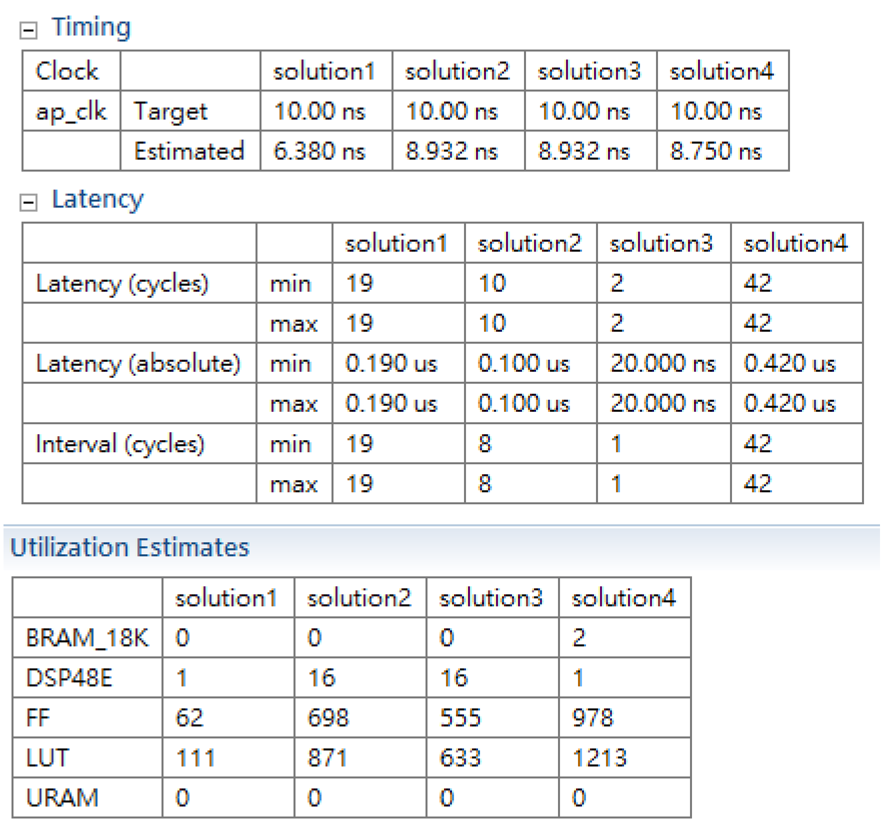
- Source code 沒有提供介面，為了要能成功放上 zedboard 執行，我們必須加上相應的介面
- 使用 AXI lite 介面來傳輸控制信號以及 dout
- 使用 m_axi 介面來傳輸輸入矩陣



```
void diff_sq_acc(din_t a[N], din_t b[N], dout_t *dout)
{
#pragma HLS INTERFACE s_axilite port=return
#pragma HLS INTERFACE s_axilite port=dout
#pragma HLS INTERFACE m_axi depth=128 port=a offset=slave
#pragma HLS INTERFACE m_axi depth=128 port=b offset=slave
```

總結與比較

- Sol1 所用資源最小
- Sol3 因為平行處理，所用資源較多但速度最快
- Sol4 加入 interface 後 latency 會被介面的傳輸給限制住，所用的資源也最多



⊟ Timing

Clock		solution1	solution2	solution3	solution4
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns	10.00 ns
	Estimated	6.380 ns	8.932 ns	8.932 ns	8.750 ns

⊟ Latency

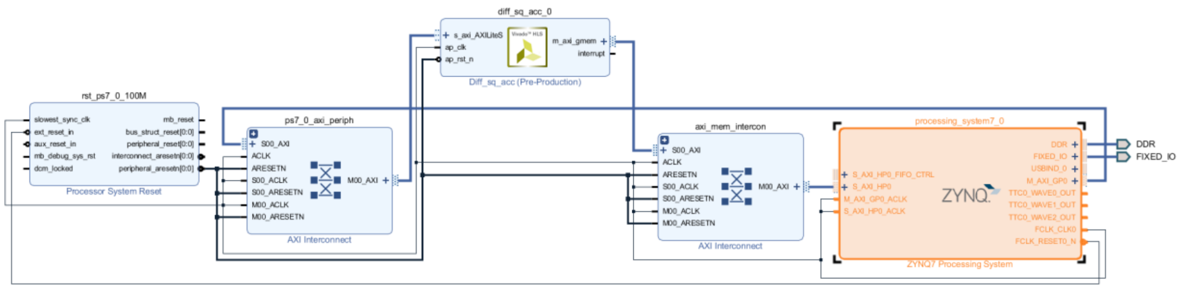
		solution1	solution2	solution3	solution4
Latency (cycles)	min	19	10	2	42
	max	19	10	2	42
Latency (absolute)	min	0.190 us	0.100 us	20.000 ns	0.420 us
	max	0.190 us	0.100 us	20.000 ns	0.420 us
Interval (cycles)	min	19	8	1	42
	max	19	8	1	42

Utilization Estimates

	solution1	solution2	solution3	solution4
BRAM_18K	0	0	0	2
DSP48E	1	16	16	1
FF	62	698	555	978
LUT	111	871	633	1213
URAM	0	0	0	0

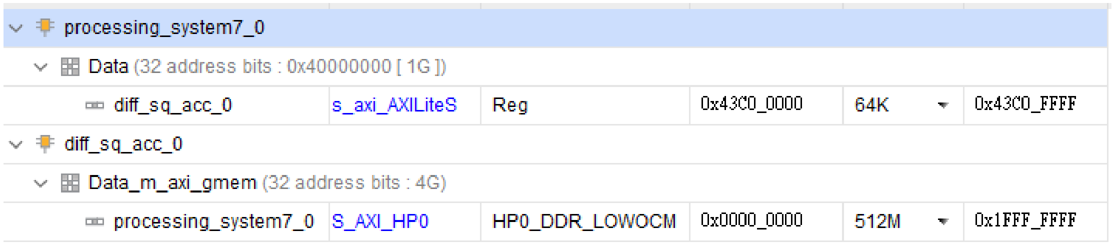
3. System level bring up (Pynq)

System Block Diagram



Address Map

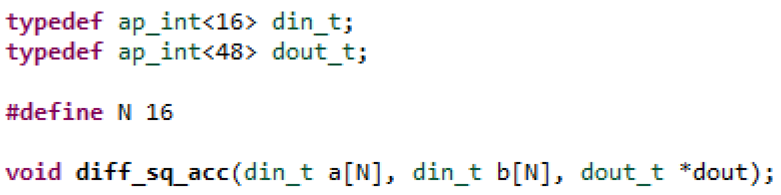
```
AXILiteS
0x00 : Control signals
       bit 0  - ap_start (Read/Write/COH)
       bit 1  - ap_done (Read/COR)
       bit 2  - ap_idle (Read)
       bit 3  - ap_ready (Read)
       bit 7  - auto_restart (Read/Write)
       others - reserved
0x04 : Global Interrupt Enable Register
       bit 0  - Global Interrupt Enable (Read/Write)
       others - reserved
0x08 : IP Interrupt Enable Register (Read/Write)
       bit 0  - Channel 0 (ap_done)
       bit 1  - Channel 1 (ap_ready)
       others - reserved
0x0c : IP Interrupt Status Register (Read/TOW)
       bit 0  - Channel 0 (ap_done)
       bit 1  - Channel 1 (ap_ready)
       others - reserved
0x10 : Data signal of a_V
       bit 31~0 - a_V[31:0] (Read/Write)
0x14 : reserved
0x18 : Data signal of b_V
       bit 31~0 - b_V[31:0] (Read/Write)
0x1c : reserved
0x20 : Data signal of dout_V
       bit 31~0 - dout_V[31:0] (Read)
0x24 : Data signal of dout_V
       bit 15~0 - dout_V[47:32] (Read)
       others   - reserved
0x28 : Control signal of dout_V
       bit 0  - dout_V_ap_vld (Read/COR)
       others - reserved
(SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)
```



processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
diff_sq_acc_0	s_axi_AXILiteS	Reg	0x43C0_0000	64K	0x43C0_FFFF
diff_sq_acc_0					
Data_m_axi_gmem (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF

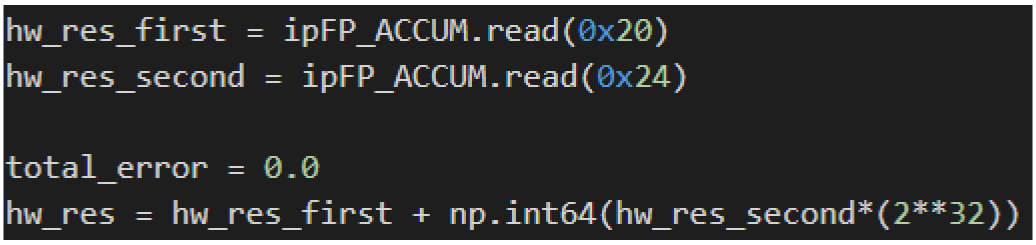
Host program

- 特別注意我們 dout 的 data width 是 48bit



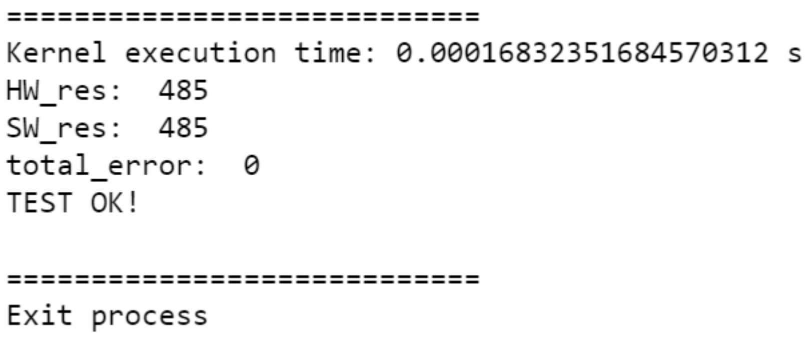
```
typedef ap_int<16> din_t;
typedef ap_int<48> dout_t;

#define N 16

void diff_sq_acc(din_t a[N], din_t b[N], dout_t *dout);
```

- 所以 dout 需要分成兩次來傳，因為 M_AXI data width=32
- 所以在 host program 必須把 dout 還原回來



```
hw_res_first = ipFP_ACCUM.read(0x20)
hw_res_second = ipFP_ACCUM.read(0x24)

total_error = 0.0
hw_res = hw_res_first + np.int64(hw_res_second*(2**32))
```

執行結果



```
============================
Kernel execution time: 0.00016832351684570312 s
HW_res:  485
SW_res:  485
total_error:  0
TEST OK!

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

4. Github submit

https://github.com/MRdudu156/MSoC-2020-Fall-self-paced/tree/main/SQUARED_DIFF_ACCUM