Self-Paced Learning 4

Floating Point Matrix Multiplication

Github Link: matrix_multiply_hls R08943129 羅宇呈

1. Brief introduction of the system

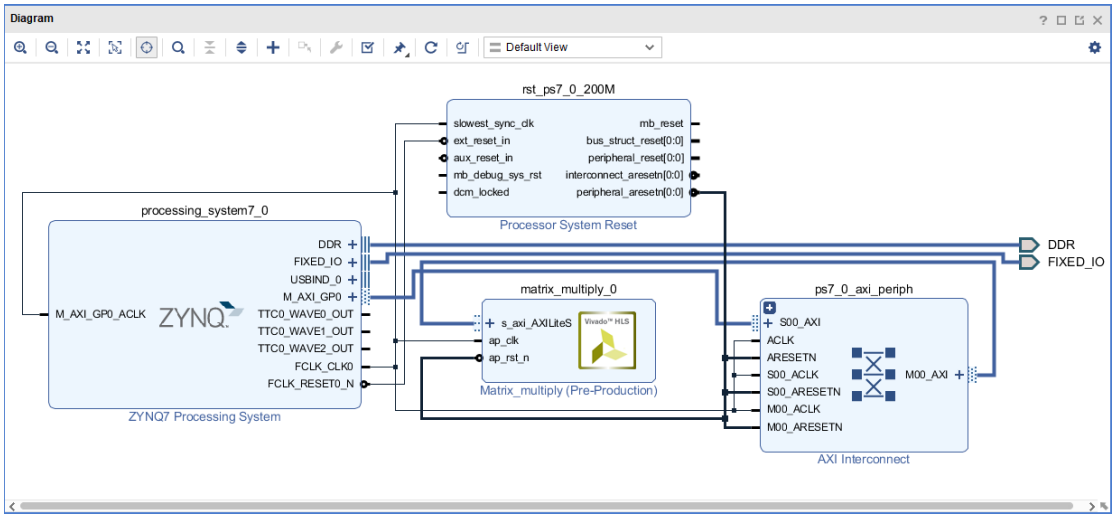


Fig.1 System Block Diagram

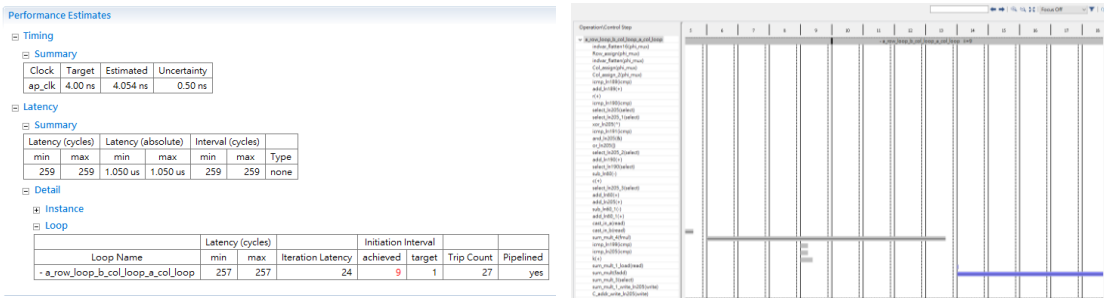
這個 Lab 我實作的是 3x3 的浮點數矩陣乘法運算，浮點數的矩陣乘法經常在影像處理或者線性代數運算器中出現，因此在大量需要矩陣運算的演算法中，這個 IP 非常重要，在介面的選擇上，Kernel Function 所使用的 Interface 都是 AXI-SLAVE，並且讓 ZYNQ 的 Processing System 作為 MASTER 讀寫 IP。

```
void matrix_multiply_default(const InputType A [RowsA][ColsA],
                             const InputType B [RowsB][ColsB],
                             OutputType C [RowsC][ColsC])
{
  // Check defined array dimensions are compatible
  // - The ROWS and COLS value of the traits considers the transpose operation request for A & B
#ifndef __SYNTHESIS__
  assert(MULTIPLIER_TRAITS::ColsATrans == MULTIPLIER_TRAITS::RowsBTrans);
  assert(RowsC == MULTIPLIER_TRAITS::RowsATrans);
  assert(ColsC == MULTIPLIER_TRAITS::ColsBTrans);
#endif
  // Use the traits struct to specify the correct type for the intermediate variables
  typename MULTIPLIER_TRAITS::INPUT_T cast_in_a, cast_in_b;
  typename MULTIPLIER_TRAITS::MULT_T  mult;
  typename MULTIPLIER_TRAITS::ACCUM_T recast_mult, sum_mult;

  a_row_loop: for (int r = 0; r < MULTIPLIER_TRAITS::RowsATrans; r++) {
    b_col_loop: for (int c = 0; c < MULTIPLIER_TRAITS::ColsBTrans; c++) {
      a_col_loop: for (int k = 0; k < MULTIPLIER_TRAITS::ColsATrans; k++) {
        #pragma HLS PIPELINE II = MULTIPLIER_TRAITS::INNER_II
        cast_in_a = GetMatrixElement<TransposeFormA,RowsA,ColsA,InputType>(A, r, k);
        cast_in_b = GetMatrixElement<TransposeFormB,RowsB,ColsB,InputType>(B, k, c);
        mult      = cast_in_a * cast_in_b;

        // Cast mult to the correct output size before adding.
        recast_mult = mult;
        if (k==0)
          sum_mult = recast_mult;
        else
          sum_mult += recast_mult;

        // Store result
        if (k==MULTIPLIER_TRAITS::ColsATrans-1) C[r][c] = sum_mult;
      }
    }
  }
}
```

Fig. 2 Source Code

2. Optimization Flow (Observation)

我先對 Iteration 的 for loop 加上 pipeline II=1 直接 Synthesis，卻發現 II 的 Constraint 發生 Violation，後來打開 Analysis 發現原因是因為浮點數的乘法和加法並沒有連貫的串在一起



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	4.00 ns	4.054 ns	0.50 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
259	259	1.050 us	1.050 us	259	259	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- a_row_loop_b_col_loop_a_col_loop	257	257	24	9	1	27	yes

Fig. 3 II Violation of Baseline Design

```
a_row_loop: for (int r = 0; r < MULTIPLIER_TRAITS::RowsATrans; r++) {
  b_col_loop: for (int c = 0; c < MULTIPLIER_TRAITS::ColsBTrans; c++) {
    a_col_loop: for (int k = 0; k < MULTIPLIER_TRAITS::ColsATrans; k++) {
      #pragma HLS PIPELINE II = 1
      cast_in_a = GetMatrixElement<TransposeFormA,RowsA,ColsA,InputType>(A, r, k);
      cast_in_b = GetMatrixElement<TransposeFormB,RowsB,ColsB,InputType>(B, k, c);
      mult [k] = cast_in_a * cast_in_b;
    }
    sum_mult = mult [0];
    acc_loop : for (int k = 0; k < MULTIPLIER_TRAITS::ColsATrans; k++) {
        #pragma HLS PIPELINE II = MULTIPLIER_TRAITS::INNER_II
        #pragma HLS dependence variable=sum_mult inter false
        sum_mult = mult [k] + sum_mult;
    }
    C [r][c] = sum_mult;
  }
 }
}
```

Fig . 4 Modified Code

因此我將 Code 改寫，試圖把乘法和加法**分開成兩個 loop**，先將乘法的部分暫存起來再累加，期望可以讓兩個 loop 各自優化成 II=1，然而發現雖然 Timing 有改善，但是 Timing Violation 仍然發生。

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	4.00 ns	3.356 ns	0.50 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
433	433	1.732 us	1.732 us	433	433	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- a_row_loop_b_col_loop	432	432	48	-	-	9	no
+ a_col_loop	11	11	10	1	1	3	yes
+ acc_loop	30	30	11	10	1	3	yes

Fig. 5 Timing Violation of Modified Design

最後我再修改 Code，將累加的每個值都暫存起來，**讓整個電路的 Topology 變成 Linear**，並加上 Dependency 的 Pragma 將 Interloop Dependency 設為 False，另外我也將這些暫存值的 Array Partition 展開成 DFF，最後將每個 Loop 都設上 pipeline II=1

```
a_row_loop: for (int r = 0; r < MULTIPLIER_TRAITS::RowsATrans; r++) {
  #pragma HLS PIPELINE II = 1
  b_col_loop: for (int c = 0; c < MULTIPLIER_TRAITS::ColsBTrans; c++) {
    #pragma HLS PIPELINE II = 1
    a_col_loop: for (int k = 0; k < MULTIPLIER_TRAITS::ColsATrans; k++) {
      #pragma HLS PIPELINE II = 1
      cast_in_a = GetMatrixElement<TransposeFormA,RowsA,ColsA,InputType>(A, r, k);
      cast_in_b = GetMatrixElement<TransposeFormB,RowsB,ColsB,InputType>(B, k, c);
      mult [k] = cast_in_a * cast_in_b;
    }
    sum_mult [0] = mult [0];
    acc_loop : for (int k = 0; k < MULTIPLIER_TRAITS::ColsATrans; k++) {
        #pragma HLS PIPELINE II = MULTIPLIER_TRAITS::INNER_II
        #pragma HLS dependence variable=sum_mult inter false
        sum_mult [k] = mult [k] + sum_mult [k-1];
    }
    C [r][c] = sum_mult [MULTIPLIER_TRAITS::ColsATrans-1];
  }
 }
}
```

Fig. 6 Final Design

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	4.00 ns	3.356 ns	0.50 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
44	44	0.176 us	0.176 us	44	44	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- a_row_loop	42	42	41	1	1	3	yes

Fig. 7 All Loop Meet Constraints

最後終於成功將所有的 loop 都展開成 II=1，而且相較 Baseline Design，Latency 大大降低，原先為 259 個 cycle，最終 Design 為 42 個 cycle 即可做完。

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	397	-
FIFO	-	-	-	-	-
Instance	-	45	5076	7155	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	111	-
Register	0	-	1187	160	-
Total	0	45	6263	7823	0
Available	280	220	106400	53200	0
Utilization (%)	0	20	5	14	0

Fig. 8 Hardware Utilization

3. C/RTL Cosimulation

Cosimulation Report for 'matrix_multiply_top'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	79	79	79	NA	NA	NA

Export the report(.html) using the Export Wizard

Fig. 8 Cosimulation Pass

4. IP Python Verification

```
        regIP.write(0x40 + 4*(i*3+j), int(float_to_hex(A[i][j]), 0))
        regIP.write(0x80 + 4*(i*3+j), int(float_to_hex(B[i][j]), 0))

time.sleep(1)

C_ans = np.zeros((3,3))

for i in range(3):
    for j in range(3):
        c = regIP.read(0xc0 + 4*(i*3+j))
        C_ans [i][j] = hex_to_float(c)

print ('-------Start Simulation-------')
print ('Input A : ')
print (A)
print ('Input B : ')
print (B)
print ('Golden C : ')
print (C_gold)
print ('Your C : ')
print (C_ans)
print ('--------End Simulation--------')
```

```
-------Start Simulation-------
Input A :
[[ 0.8246028   0.44052646  0.04849746]
 [ 0.67588117  0.7594554   0.60442036]
 [ 0.00811407  0.07848773  0.19777778]]
Input B :
[[ 0.96779452  0.13899302  0.1688476 ]
 [ 0.90893875  0.38340596  0.53569744]
 [ 0.53657465  0.58399185  0.11783364]]
Golden C :
[[ 1.22448015  0.31183663  0.38093573]
 [ 1.66872917  0.73809905  0.59218028]
 [ 0.18531583  0.14672107  0.06672059]]
Your C :
[[ 1.22448015  0.31183663  0.3809357 ]
 [ 1.66872919  0.73809904  0.59218031]
 [ 0.18531585  0.14672107  0.0667206 ]]
--------End Simulation--------
```

Fig. 9 Functionality Pass