Sparse Matrix Vector Multiplication

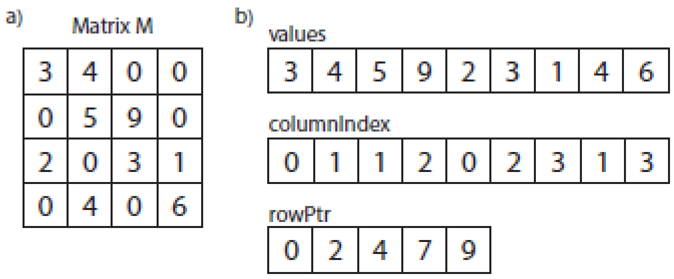
1. HLS C-sim/Synthesis/Cosim (Screenshot + brief intro) (1%)

Sparse Matrix Vector 是一種 Matrix 的表示式，它可以減少資料的儲存量，而 Sparse Matrix Vector Multiplication (SPMV)就是拿 SMV 來做矩陣乘法運算。圖 1(a)為一個 Sparse Matrix，圖 1(b)為這個 Matrix 的 Compressed Row Storage (CRS) form，它有三個 Array，分別是 values、columnIndex、rowPtr。以下介紹其轉換方式:

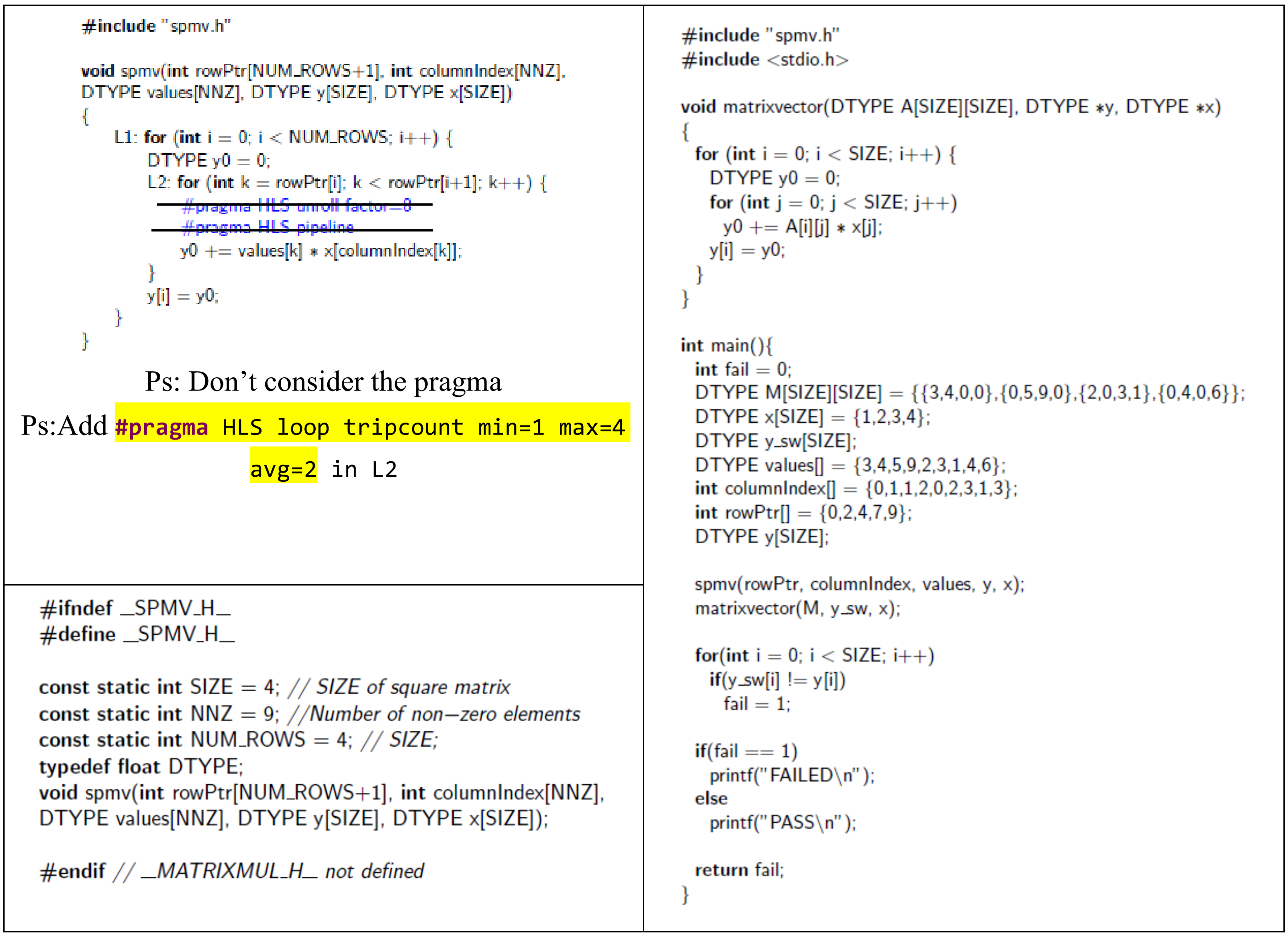
Values: 將非零的數值填入(從 row 開始)

columnIndex: 非零的數值在哪一個 column

rowPtr: 每一列的第一個值為非零數值的第幾個(第一個則填入 0，而最後一格為非零數值的總數量)



◎圖 1



```
#include "spmv.h"

void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
    L1: for (int i = 0; i < NUM_ROWS; i++) {
        DTYPE y0 = 0;
        L2: for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
            ~~#pragma HLS unroll factor=8~~
            ~~#pragma HLS pipeline~~
            y0 += values[k] * x[columnIndex[k]];
        }
        y[i] = y0;
    }
}
```

Ps: Don't consider the pragma

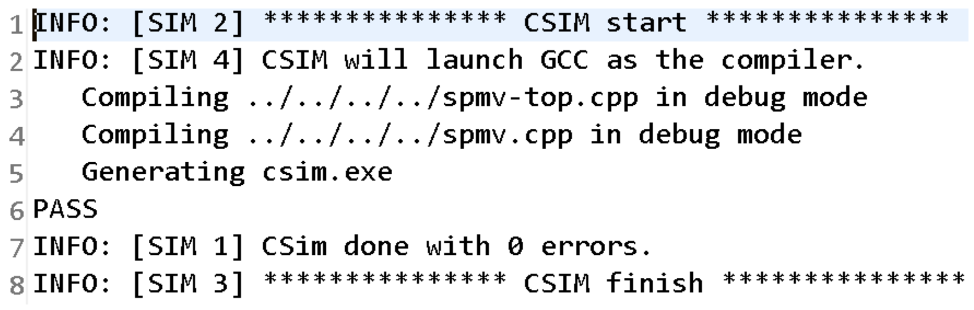
Ps:Add `#pragma HLS loop tripcount min=1 max=4 avg=2` in L2

```
#ifndef __SPMV_H__
#define __SPMV_H__

const static int SIZE = 4; // SIZE of square matrix
const static int NNZ = 9; //Number of non-zero elements
const static int NUM_ROWS = 4; // SIZE;
typedef float DTYPE;
void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE]);

#endif // __MATRIXMUL_H__ not defined
```

```
#include "spmv.h"
#include <stdio.h>

void matrixvector(DTYPE A[SIZE][SIZE], DTYPE *y, DTYPE *x)
{
  for (int i = 0; i < SIZE; i++) {
    DTYPE y0 = 0;
    for (int j = 0; j < SIZE; j++)
      y0 += A[i][j] * x[j];
    y[i] = y0;
  }
}

int main(){
  int fail = 0;
  DTYPE M[SIZE][SIZE] = {{3,4,0,0},{0,5,9,0},{2,0,3,1},{0,4,0,6}};
  DTYPE x[SIZE] = {1,2,3,4};
  DTYPE y_sw[SIZE];
  DTYPE values[] = {3,4,5,9,2,3,1,4,6};
  int columnIndex[] = {0,1,1,2,0,2,3,1,3};
  int rowPtr[] = {0,2,4,7,9};
  DTYPE y[SIZE];

  spmv(rowPtr, columnIndex, values, y, x);
  matrixvector(M, y_sw, x);

  for(int i = 0; i < SIZE; i++)
    if(y_sw[i] != y[i])
      fail = 1;

  if(fail == 1)
    printf("FAILED\n");
  else
    printf("PASS\n");

  return fail;
}
```

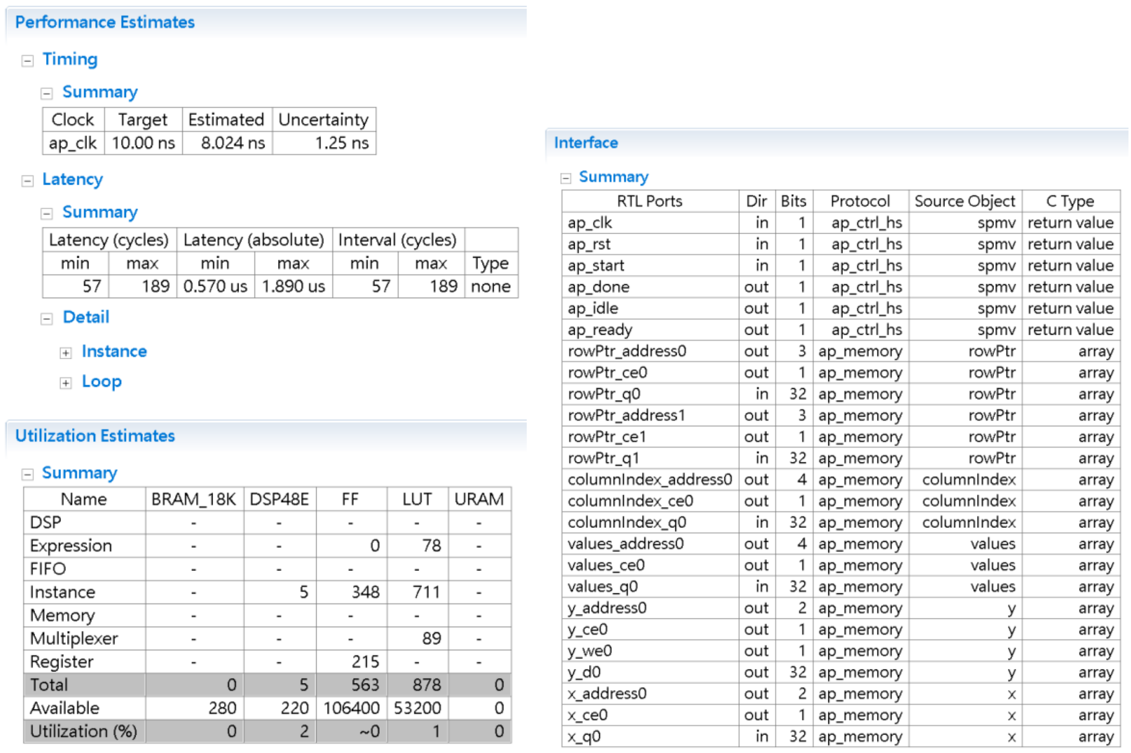
◎圖 2 Baseline Code (spmv.cpp、spmv.h、spmv-top.cpp)

The Screenshot is from the Baseline Code



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../spmv-top.cpp in debug mode
   Compiling ../../../../spmv.cpp in debug mode
   Generating csim.exe
PASS
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

◎圖 3 C-sim



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.024 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
57	189	0.570 us	1.890 us	57	189	none

Detail

Instance

Loop

Utilization Estimates

Summary

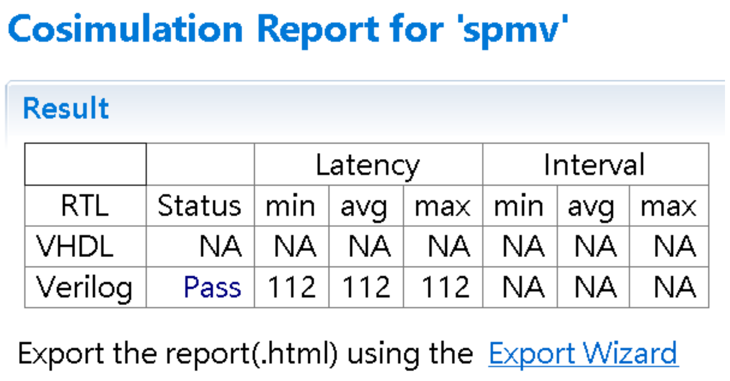
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	78	-
FIFO	-	-	-	-	-
Instance	-	5	348	711	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	89	-
Register	-	-	215	-	-
Total	0	5	563	878	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	~0	1	0

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	spmv	return value
ap_rst	in	1	ap_ctrl_hs	spmv	return value
ap_start	in	1	ap_ctrl_hs	spmv	return value
ap_done	out	1	ap_ctrl_hs	spmv	return value
ap_idle	out	1	ap_ctrl_hs	spmv	return value
ap_ready	out	1	ap_ctrl_hs	spmv	return value
rowPtr_address0	out	3	ap_memory	rowPtr	array
rowPtr_ce0	out	1	ap_memory	rowPtr	array
rowPtr_q0	in	32	ap_memory	rowPtr	array
rowPtr_address1	out	3	ap_memory	rowPtr	array
rowPtr_ce1	out	1	ap_memory	rowPtr	array
rowPtr_q1	in	32	ap_memory	rowPtr	array
columnIndex_address0	out	4	ap_memory	columnIndex	array
columnIndex_ce0	out	1	ap_memory	columnIndex	array
columnIndex_q0	in	32	ap_memory	columnIndex	array
values_address0	out	4	ap_memory	values	array
values_ce0	out	1	ap_memory	values	array
values_q0	in	32	ap_memory	values	array
y_address0	out	2	ap_memory	y	array
y_ce0	out	1	ap_memory	y	array
y_we0	out	1	ap_memory	y	array
y_d0	out	32	ap_memory	y	array
x_address0	out	2	ap_memory	x	array
x_ce0	out	1	ap_memory	x	array
x_q0	in	32	ap_memory	x	array

◎圖 4 C-Synthesis



Cosimulation Report for 'spmv'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	112	112	112	NA	NA	NA

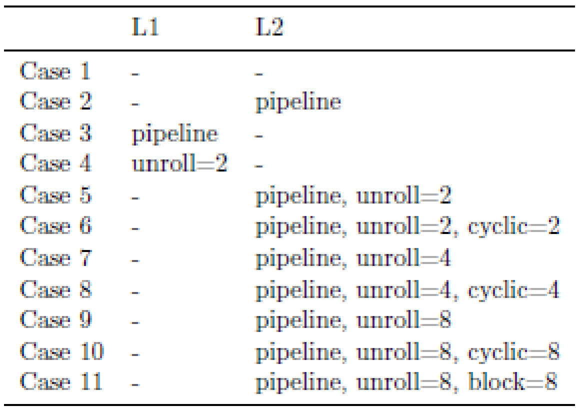
Export the report(.html) using the Export Wizard

◎圖 5 Cosim

2. Improvement – throughput, area (1%)

我們可以對於 L1, L2 這兩個 loop 做優化

Solution1: 測試圖 6 中不同的組合

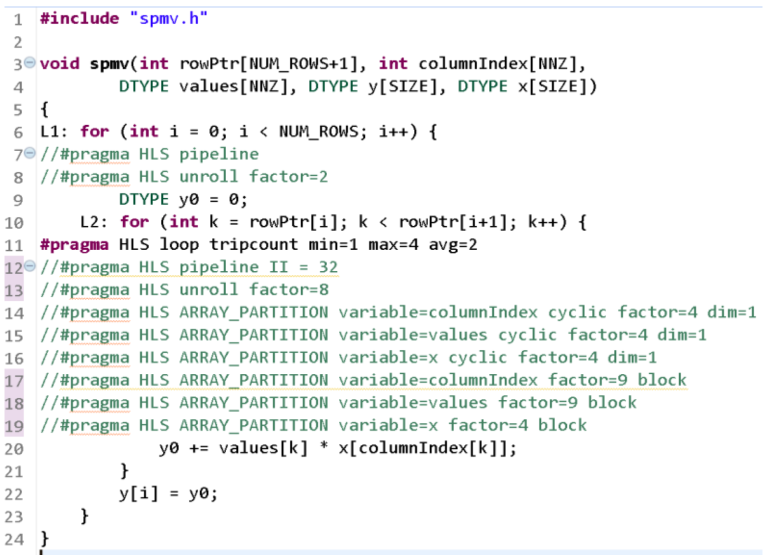


	L1	L2
Case 1	-	-
Case 2	-	pipeline
Case 3	pipeline	-
Case 4	unroll=2	-
Case 5	-	pipeline, unroll=2
Case 6	-	pipeline, unroll=2, cyclic=2
Case 7	-	pipeline, unroll=4
Case 8	-	pipeline, unroll=4, cyclic=4
Case 9	-	pipeline, unroll=8
Case 10	-	pipeline, unroll=8, cyclic=8
Case 11	-	pipeline, unroll=8, block=8

◎圖 6 Potential optimizations for sparse matrix-vector multiplication

An partially unrolled Version

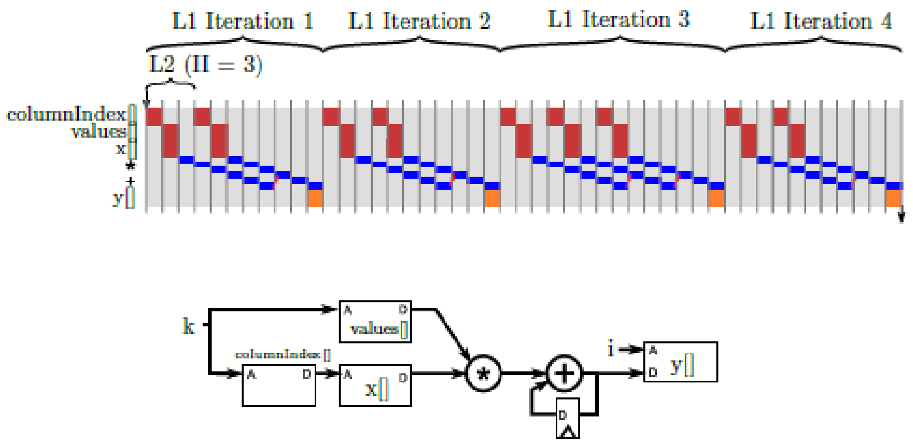
	L1	L2	All loop constraints were satisfied?	If not satisfied, how to solve?
Case1			Yes	
Case2		Pipeline	No	II = 5
Case3	Pipeline		Yes	
Case4	Unroll=2		Yes	
Case5		Pipeline, Unroll=2	No	II = 8
Case6		Pipeline, Unroll=2, cyclic = 2	No	II = 8
Case7		Pipeline, Unroll=4	No	II = 16
Case8		Pipeline, Unroll=4, cyclic = 4	No	II = 16
Case9		Pipeline, Unroll=8	No	II = 32
Case10		Pipeline, Unroll=8, cyclic = 8	No	II = 32
Case11		Pipeline, Unroll=8, block = 8	No	II= 32 block factor(9,9,4)



```
#include "spmv.h"

void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
        DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
L1: for (int i = 0; i < NUM_ROWS; i++) {
//#pragma HLS pipeline
//#pragma HLS unroll factor=2
        DTYPE y0 = 0;
    L2: for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
#pragma HLS loop tripcount min=1 max=4 avg=2
//#pragma HLS pipeline II = 32
//#pragma HLS unroll factor=8
//#pragma HLS ARRAY_PARTITION variable=columnIndex cyclic factor=4 dim=1
//#pragma HLS ARRAY_PARTITION variable=values cyclic factor=4 dim=1
//#pragma HLS ARRAY_PARTITION variable=x cyclic factor=4 dim=1
//#pragma HLS ARRAY_PARTITION variable=columnIndex factor=9 block
//#pragma HLS ARRAY_PARTITION variable=values factor=9 block
//#pragma HLS ARRAY_PARTITION variable=x factor=4 block
            y0 += values[k] * x[columnIndex[k]];
        }
        y[i] = y0;
    }
}
```

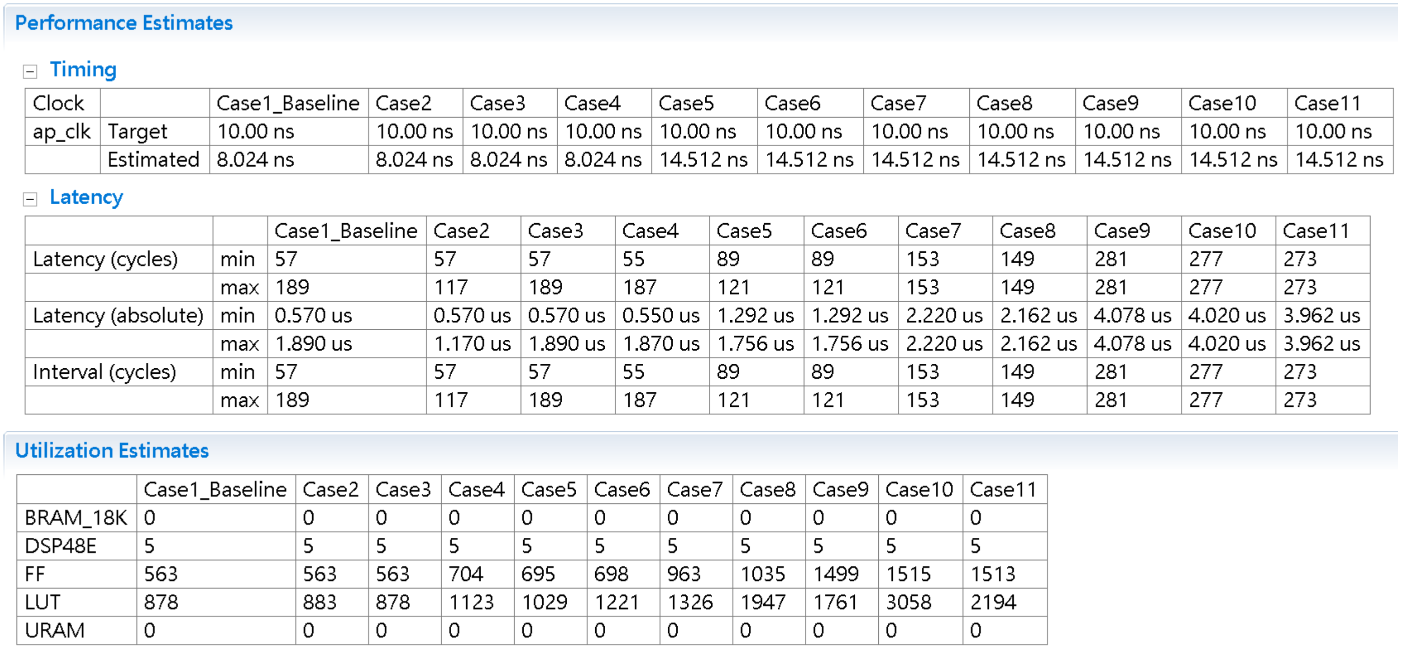
◎圖 7 Code with pragma (Please edit according to different case)

圖 8 為講義將 L2 這個加入 pipeline 設計(Case2)，而其 II=3，原因是在加法器的部分預計會有 3 clock 的 latency。(但我實際跑出來需要 II = 5)



◎圖 8 Architecture and behavior of the spmv code with a pipelined inner loop.

圖 9 所示經過幾個測試後，發現 pipeline 無法實現 II=1，且 unroll 後將使 latency 更大，Case2 似乎為目前最好的解。


Performance Estimates

Timing

Clock		Case1_Baseline	Case2	Case3	Case4	Case5	Case6	Case7	Case8	Case9	Case10	Case11
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns	10.00 ns	10.00 ns	10.00 ns	10.00 ns	10.00 ns	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.024 ns	8.024 ns	8.024 ns	8.024 ns	14.512 ns	14.512 ns	14.512 ns	14.512 ns	14.512 ns	14.512 ns	14.512 ns

Latency

		Case1_Baseline	Case2	Case3	Case4	Case5	Case6	Case7	Case8	Case9	Case10	Case11
Latency (cycles)	min	57	57	57	55	89	89	153	149	281	277	273
	max	189	117	189	187	121	121	153	149	281	277	273
Latency (absolute)	min	0.570 us	0.570 us	0.570 us	0.550 us	1.292 us	1.292 us	2.220 us	2.162 us	4.078 us	4.020 us	3.962 us
	max	1.890 us	1.170 us	1.890 us	1.870 us	1.756 us	1.756 us	2.220 us	2.162 us	4.078 us	4.020 us	3.962 us
Interval (cycles)	min	57	57	57	55	89	89	153	149	281	277	273
	max	189	117	189	187	121	121	153	149	281	277	273

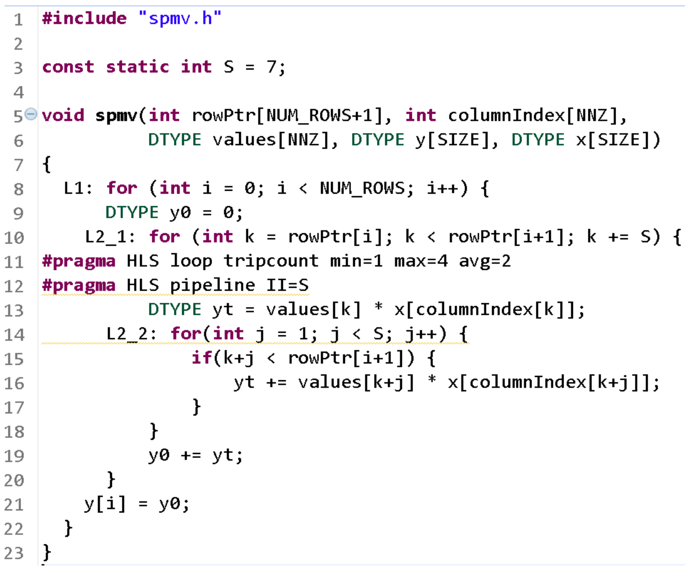
Utilization Estimates

	Case1_Baseline	Case2	Case3	Case4	Case5	Case6	Case7	Case8	Case9	Case10	Case11
BRAM_18K	0	0	0	0	0	0	0	0	0	0	0
DSP48E	5	5	5	5	5	5	5	5	5	5	5
FF	563	563	563	704	695	698	963	1035	1499	1515	1513
LUT	878	883	878	1123	1029	1221	1326	1947	1761	3058	2194
URAM	0	0	0	0	0	0	0	0	0	0	0

◎圖 9 Compare Report

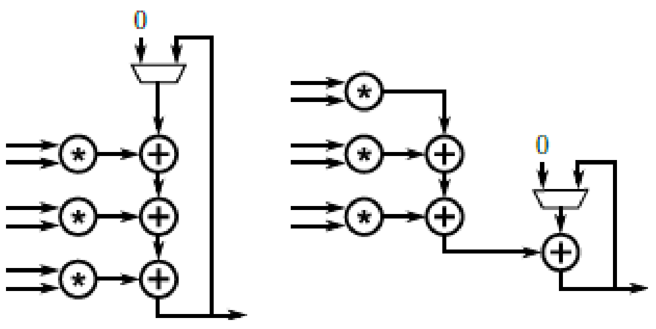
Solution2: Partially Unroll

Case5 在 L2 中進行 pipeline(II=8)和 unrolled=2，我們要優化這個 Case，可讓架構在 loop 有更多的並行性。再圖 10 可以看出其 Latency(absolute)可減少 19.7%。

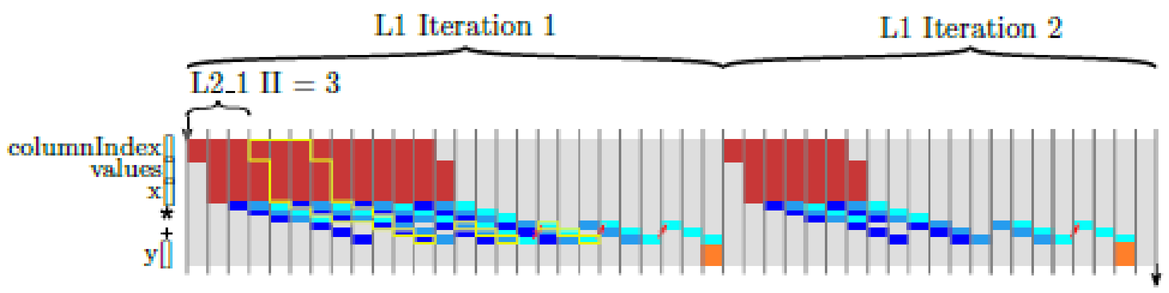


```
#include "spmv.h"

const static int S = 7;

void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
          DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
  L1: for (int i = 0; i < NUM_ROWS; i++) {
      DTYPE y0 = 0;
    L2_1: for (int k = rowPtr[i]; k < rowPtr[i+1]; k += S) {
#pragma HLS loop tripcount min=1 max=4 avg=2
#pragma HLS pipeline II=S
          DTYPE yt = values[k] * x[columnIndex[k]];
      L2_2: for(int j = 1; j < S; j++) {
              if(k+j < rowPtr[i+1]) {
                  yt += values[k+j] * x[columnIndex[k+j]];
              }
          }
          y0 += yt;
      }
    y[i] = y0;
  }
}
```

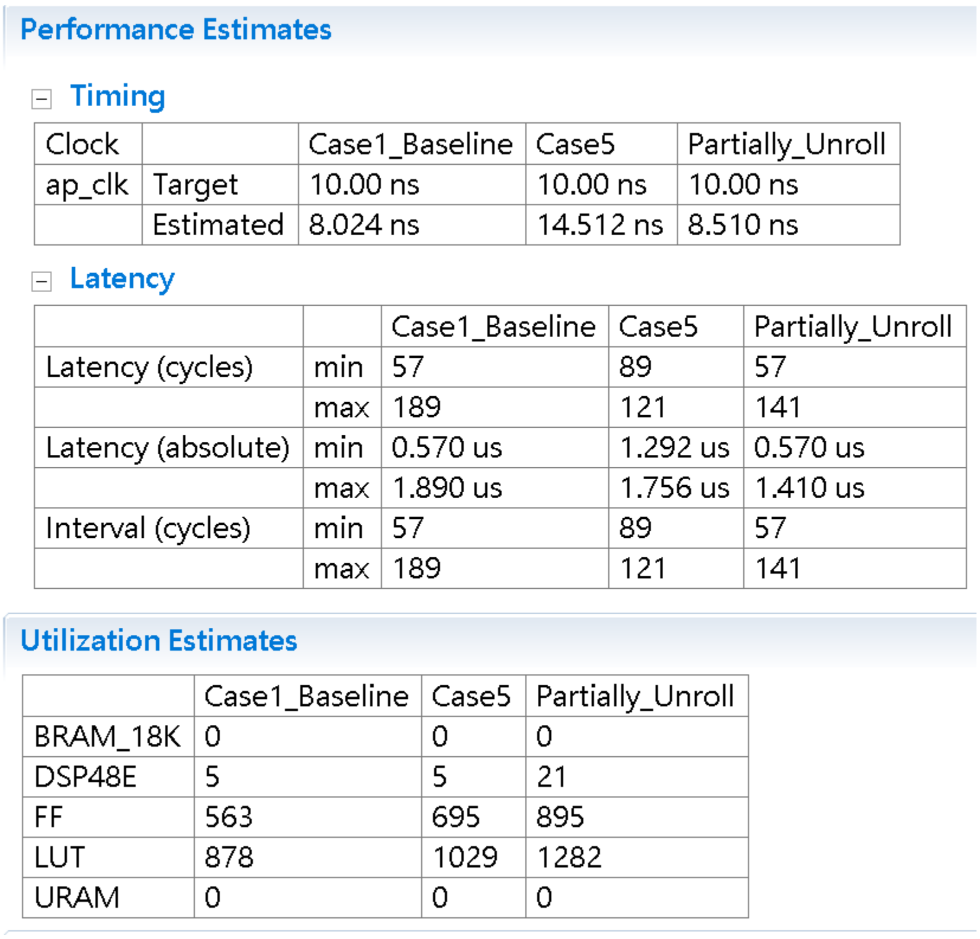
◎圖 10 A partially unrolled version of the spmv code



◎圖 11 Two diiferent partially unrolled versions of an accumulation. The version on the left has a recurrence with three additions, whereas the version on right only has one addition in the recurrence(左:是我們想達成的，右是實際的)



◎圖 12 Architecture and behavior of the spmv code based on the partially unrolled and pipelined inner loop



Performance Estimates

Timing

Clock		Case1_Baseline	Case5	Partially_Unroll
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.024 ns	14.512 ns	8.510 ns

Latency

		Case1_Baseline	Case5	Partially_Unroll
Latency (cycles)	min	57	89	57
	max	189	121	141
Latency (absolute)	min	0.570 us	1.292 us	0.570 us
	max	1.890 us	1.756 us	1.410 us
Interval (cycles)	min	57	89	57
	max	189	121	141

Utilization Estimates

	Case1_Baseline	Case5	Partially_Unroll
BRAM_18K	0	0	0
DSP48E	5	5	21
FF	563	695	895
LUT	878	1029	1282
URAM	0	0	0

◎圖 13 Compare Report

3. Github submit (1%)

https://github.com/r08943099/MSOCFall2020

4. Complexity (1% - Instructor/ TA decide)