MSoC self-paced learning project_3 FIR_example

R09943017 林奕廷

1. Introduction

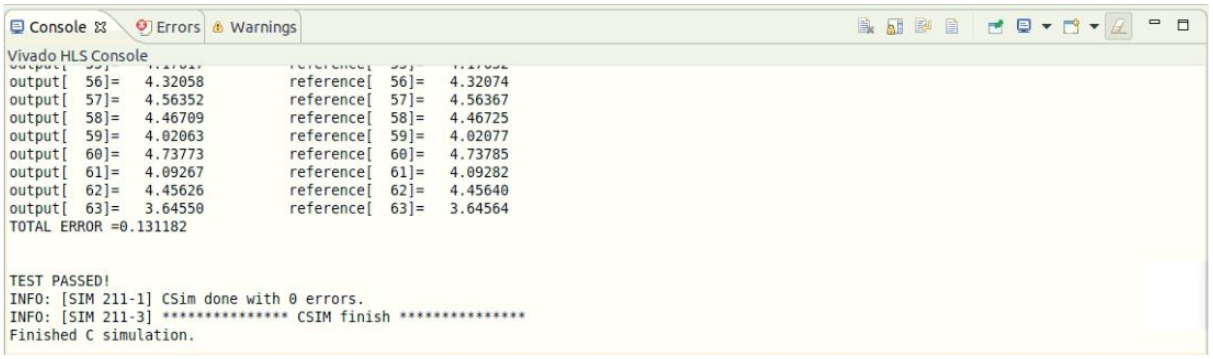
這項專案的目的是藉由Vivado HLS來實作一個16-taps的FIR filter。Top function在fir.cpp檔中定義。原本的code：

```
out_data_t fir_filter (inp_data_t x,   coef_t c[N])
{


  static inp_data_t shift_reg[N];

  acc_t acc = 0;
  acc_t mult;
  out_data_t y;

  Shift_Accum_Loop: for (int i=N-1;i>=0;i--)
  {
#pragma HLS LOOP_TRIPCOUNT min=1 max=16 avg=8

     if (i==0)
     {
       //acc+=x*c[0];
       shift_reg[0]=x;
     }
     else
     {
            shift_reg[i]=shift_reg[i-1];
            //acc+=shift_reg[i]*c[i];
     }
     mult = shift_reg[i]*c[i];
     acc  = acc + mult;

  }

  y = (out_data_t) acc;

  return y;
```

fir_filter會將輸入訊號存進內部的shift register。在之後的每個迴圈中tap coefficient會和當下的shift register output相乘後存回acc中。如此一來，進行fir_filter多次後便可以對完整的輸入訊號完成操作。

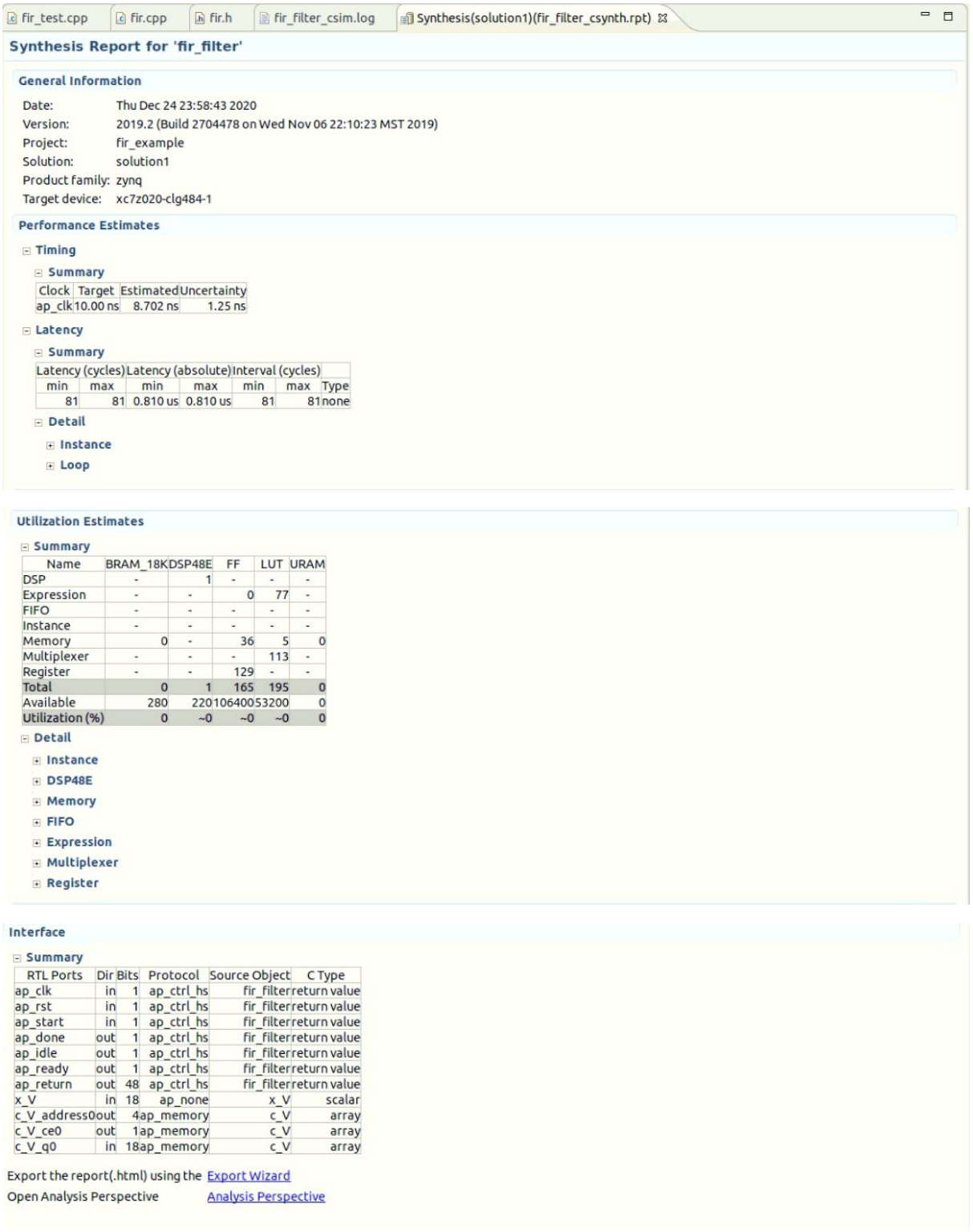
2. HLS C-simulation

透過C-simulation執行資料夾中提供的testbench會得到以下模擬結果：



3. HLS Synthesis

fir_filter的合成結果如下圖所示：



fir_test.cpp | fir.cpp | fir.h | fir_filter_csim.log | Synthesis(solution1)(fir_filter_csynth.rpt)

Synthesis Report for 'fir_filter'

General Information

Date: Thu Dec 24 23:58:43 2020
Version: 2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project: fir_example
Solution: solution1
Product family: zynq
Target device: xc7z020-clg484-1

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.702 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
81	81	0.810 us	0.810 us	81	81	none

Detail

Instance
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	77	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	0	-	36	5	0
Multiplexer	-	-	-	113	-
Register	-	-	129	-	-
Total	0	1	165	195	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	~0	~0	0

Detail

Instance
DSP48E
Memory
FIFO
Expression
Multiplexer
Register

Interface

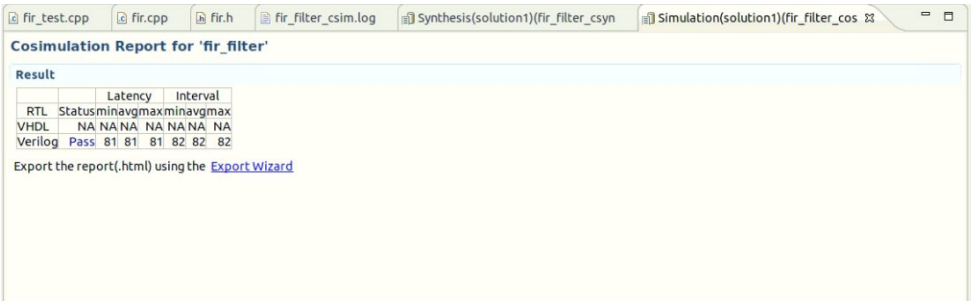
Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir_filter	return value
ap_rst	in	1	ap_ctrl_hs	fir_filter	return value
ap_start	in	1	ap_ctrl_hs	fir_filter	return value
ap_done	out	1	ap_ctrl_hs	fir_filter	return value
ap_idle	out	1	ap_ctrl_hs	fir_filter	return value
ap_ready	out	1	ap_ctrl_hs	fir_filter	return value
ap_return	out	48	ap_ctrl_hs	fir_filter	return value
x_V	in	18	ap_none	x_V	scalar
c_V_address0	out	4	ap_memory	c_V	array
c_V_ce0	out	1	ap_memory	c_V	array
c_V_q0	in	18	ap_memory	c_V	array

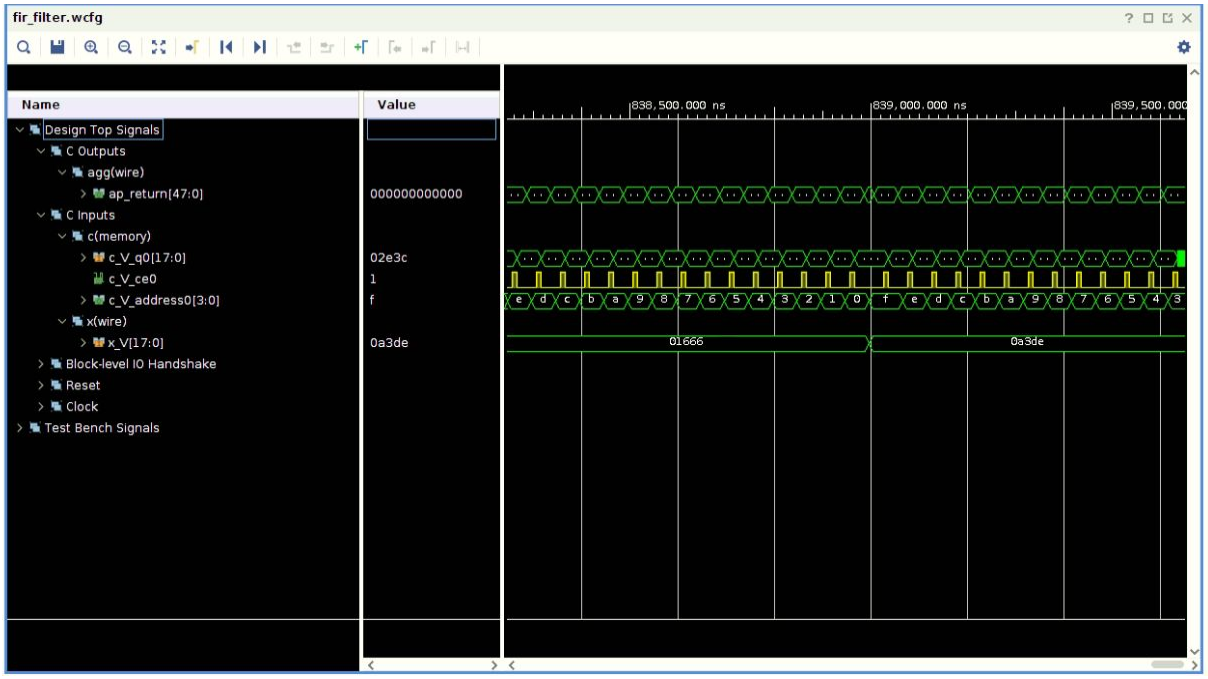
Export the report(.html) using the Export Wizard
Open Analysis Perspective Analysis Perspective

4. Cosimulation

執行Cosimulation得結果如下：



波形如下圖所示(一部分截圖)：



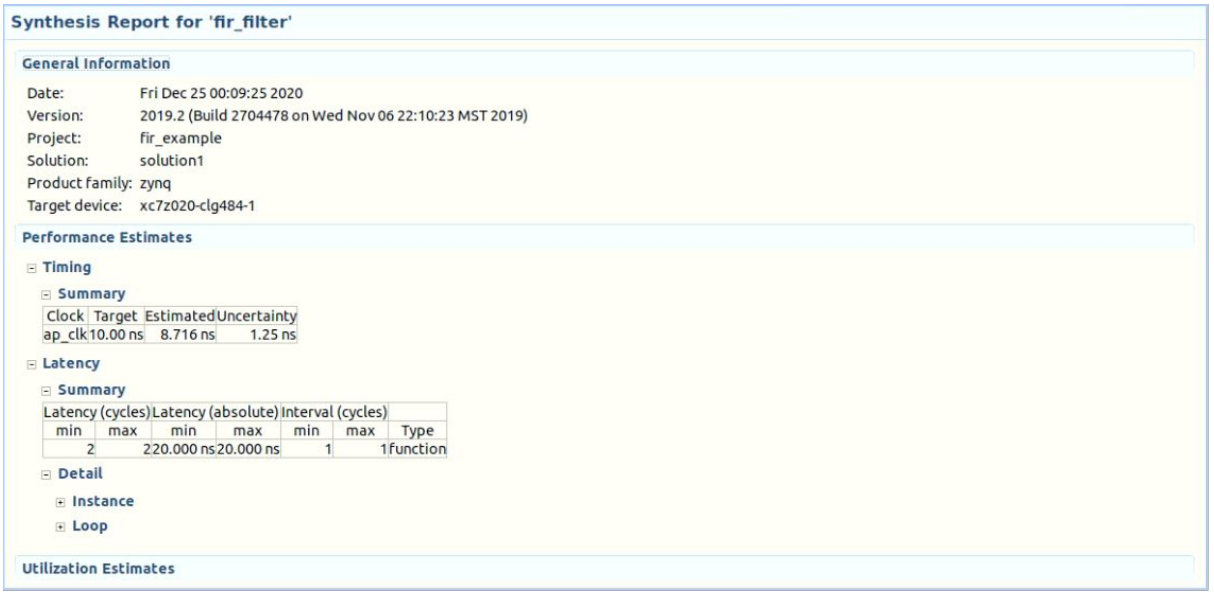
5. Improvement

為了增進整體fir filter運算時的throughtput，我對於Shift_Accum_loop進行了unrolling + pipeling，使得一次fir_filter的運算可以平行做完。為了達成此throughtput，fir filter所使用的coefficients array c需要完全的partition，使得運算時可一次拿到全部的coefficients。修改後的code如下：

fir_test.cpp | *fir.cpp | fir.h | fir_filter_csim.log | Simulation(solution1)(fir_filter_cosim | Synthesis(solution1)(fir_filter_csynth.

```
1  #include "fir.h"
2
3  out_data_t fir_filter (inp_data_t x,   coef_t c[N])
4  {
5    static inp_data_t shift_reg[N];
6  #pragma HLS array_partition variable=c complete
7    acc_t acc = 0;
8    acc_t mult;
9    out_data_t y;
10 #pragma HLS pipeline II=1
11   Shift_Accum_Loop: for (int i=N-1;i>=0;i--)
12   {
13 #pragma HLS LOOP_TRIPCOUNT min=1 max=16 avg=8
14     if (i==0){
15       //acc+=x*c[0];
16       shift_reg[0]=x;
17     }
18     else{
19           shift_reg[i]=shift_reg[i-1];
20           //acc+=shift_reg[i]*c[i];
21     }
22     mult = shift_reg[i]*c[i];
23     acc  = acc + mult;
24   }
25   y = (out_data_t) acc;
26   return y;
27 }
28
```

合成的結果：



Synthesis Report for 'fir_filter'

General Information

Date:	Fri Dec 25 00:09:25 2020
Version:	2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project:	fir_example
Solution:	solution1
Product family:	zynq
Target device:	xc7z020-clg484-1

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.716 ns	1.25 ns

Latency

Summary

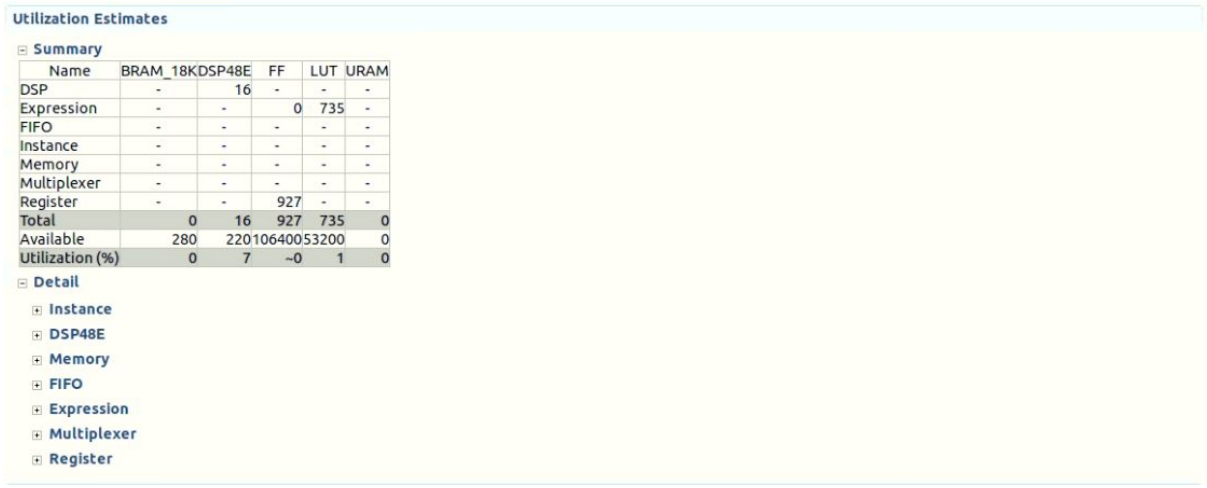
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2		220.000 ns	20.000 ns	1	1	function

Detail

Instance

Loop

Utilization Estimates



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	16	-	-	-
Expression	-	-	0	735	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	927	-	-
Total	0	16	927	735	0
Available	280	220	106400	53200	0
Utilization (%)	0	7	~0	1	0

Detail

Instance

DSP48E

Memory

FIFO

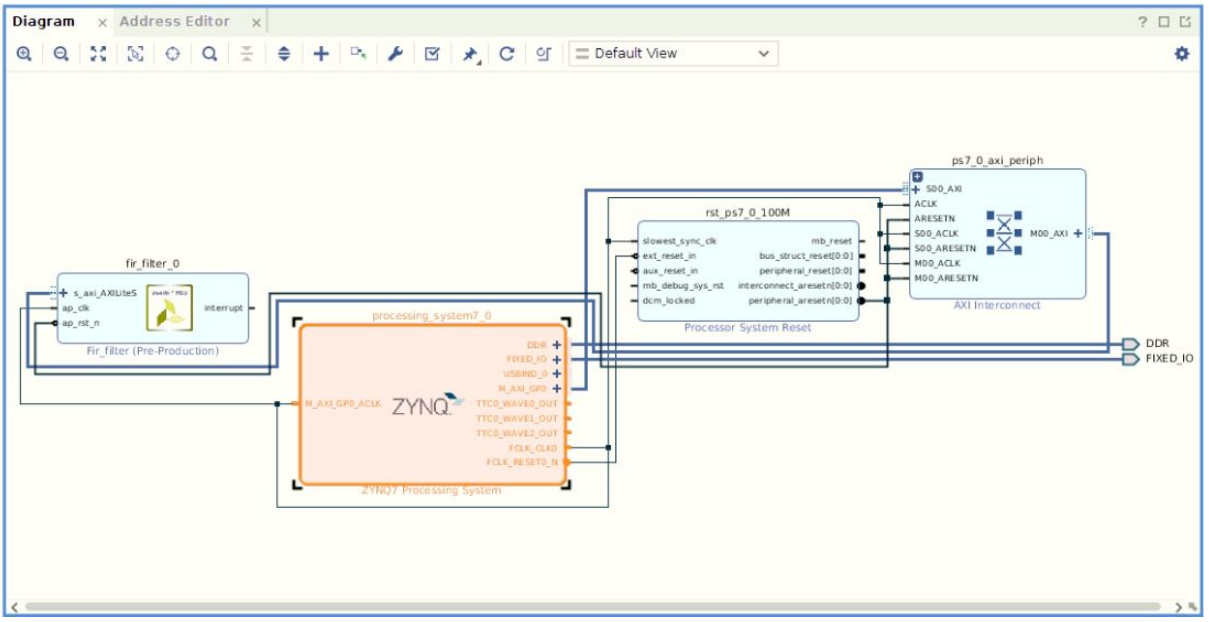
Expression

Multiplexer

Register

如此優化以後，整體運算的latency大幅減少，相對的會使用到比較多的硬體資源。如果應用中latency是關鍵時可以使用此種tradeoff。

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端也是使用AXILite的界面進行資料傳輸。

```
#pragma HLS INTERFACE s_axilite port=x
#pragma HLS INTERFACE s_axilite port=c
#pragma HLS INTERFACE s_axilite port=return
```

Host program執行的結果如下：

```
print("Exit process")
output = 4.82527  ref = 4.82540
output = 4.04163  ref = 4.04177
output = 4.57077  ref = 4.57088
output = 3.99791  ref = 3.99804
output = 3.55887  ref = 3.55898
output = 4.25025  ref = 4.25037
output = 3.92238  ref = 3.92250
output = 4.34971  ref = 4.34984
output = 3.59672  ref = 3.59683
output = 4.12284  ref = 4.12297
output = 3.12869  ref = 3.12881
output = 4.25798  ref = 4.25811
output = 3.46025  ref = 3.46037
output = 4.13056  ref = 4.13068
output = 3.81068  ref = 3.81079
output = 4.46850  ref = 4.46862
TEST PASSED
============================
Exit process
In [ ]:
```

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