Embedded Linux 系統實作 – HLS

Zynq UltraScale+ *MPSoC[v2019.2 – v2020.1]*

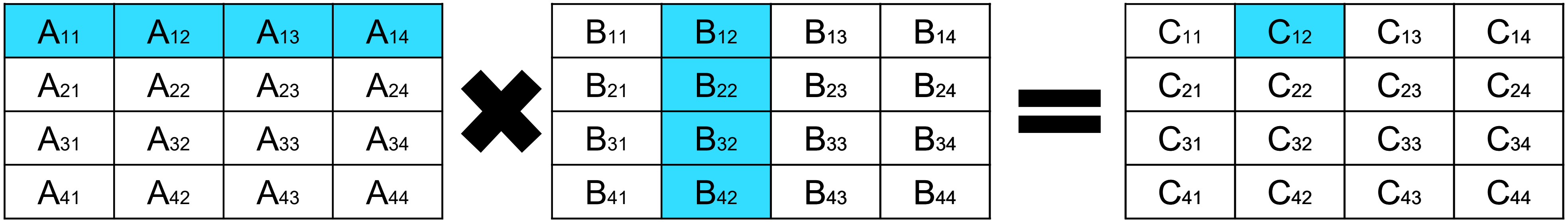
Agenda

- 矩陣運算原理
- HLS IP 設計流程
  - CPP 與 Test Bench
  - 建立 HLS 專案
  - HLS C 模擬與合成
  - HLS 優化
  - HLS 解決方案比較
- 系統架構－加法器
- HLS IP 設計流程
  - CPP 與 Test Bench
  - HLS 介面定義
  - HLS 優化
  - HLS 合成與模擬
  - HLS IP 導出
- Vivado 專案
  - HLS IP 設定
  - IPI 設定
- Linux 設計
  - 修改 Device tree 內容
  - 修改 Driver 內容
- 測試 API

矩陣運算原理

矩陣運算原理

- C[4][4] = A[4][4] * B[4][4]



A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}

✖

B_{11}	B_{12}	B_{13}	B_{14}
B_{21}	B_{22}	B_{23}	B_{24}
B_{31}	B_{32}	B_{33}	B_{34}
B_{41}	B_{42}	B_{43}	B_{44}

=

C_{11}	C_{12}	C_{13}	C_{14}
C_{21}	C_{22}	C_{23}	C_{24}
C_{31}	C_{32}	C_{33}	C_{34}
C_{41}	C_{42}	C_{43}	C_{44}

- $C_{11} = A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31} + A_{41} * B_{41}$
- $C_{12} = A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32} + A_{41} * B_{42}$
- $C_{13} = A_{11} * B_{13} + A_{12} * B_{23} + A_{13} * B_{33} + A_{41} * B_{43}$
- $C_{14} = A_{11} * B_{14} + A_{12} * B_{24} + A_{13} * B_{34} + A_{41} * B_{44}$

⋮

- 轉換成 C 程式碼：

```
for (i=0; i<4; i++)
    for (j=0; j<4; j++)
        for (k=0; k<4; k++)
             C[i][j] = C[i][j] + A[i][k] * B[k][j]
```

HLS IP 設計流程

HLS IP 設計流程 – CPP 與 Test Bench

- matrix_mul.cpp

```
#include "matrix_mul.h"
void matrix_mul(ap_int<8> A[4][4],ap_int<8> B[4][4],ap_int<16> C[4][4])
{
        for(int i=0;i<4;i++)
        {
                for(int j=0;j<4;j++)
                {
                        C[i][j]=0;
                        for(int k=0;k<4;k++)
                        {
                                C[i][j]=C[i][j]+A[i][k]*B[k][j];
                        }
                }
        }
}
```

- matrix_mul.h

```
#ifndef __MATRIX_MUL__
#define __MATRIX_MUL__

#include "ap_fixed.h"
void matrix_mul(ap_int<8> A[4][4],ap_int<8> B[4][4],ap_int<16> C[4][4]);

#endif
```

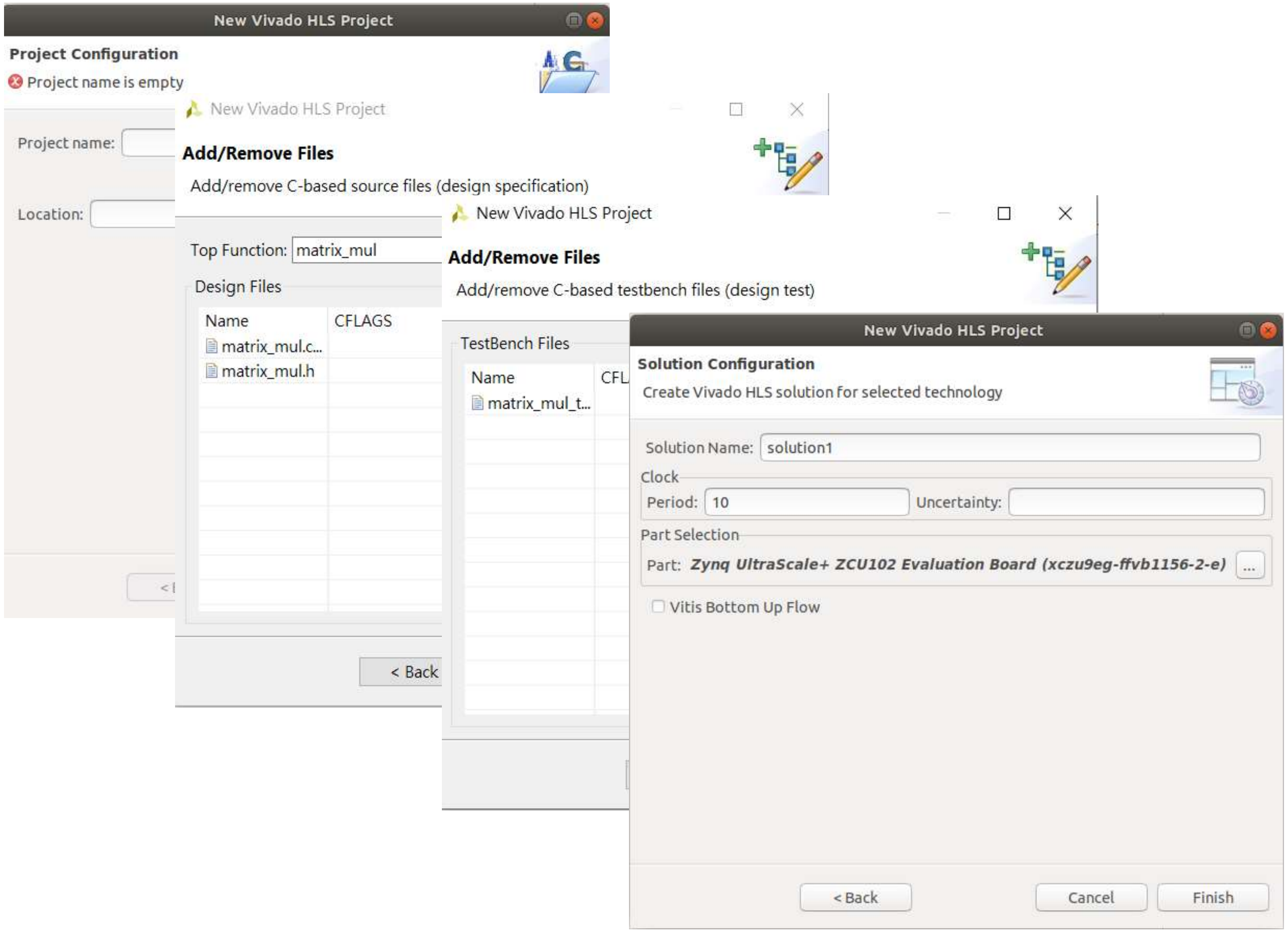
HLS IP 設計流程 – CPP 與 Test Bench

- matrix_mul_tb.cpp

```
#include "matrix_mul.h"
#include <iostream>

int main()
{
        ap_int<8> A[4][4];
        ap_int<8> B[4][4];
        ap_int<16> C[4][4];

        for(int i=0;i<4;i++)
                for(int j=0;j<4;j++)
                {
                        A[i][j]=i*4+j;
                        B[i][j]=A[i][j];
                }

        matrix_mul(A,B,C);

        for(int i=0;i<4;i++)
                for(int j=0;j<4;j++)
                        std::cout<<"C["<<i<<","<<j<<"]="<<C[i][j]<<std::endl;

        return 0;
}
```

HLS IP 設計流程 –建立 HLS 專案

- 流程：
  1. Create New Project
  2. Add File...
  3. Top Function
  4. Add TestBench File...
  5. 選擇 ZCU102
  6. Finish



New Vivado HLS Project

Project Configuration

Project name is empty

Project name:

Location:

New Vivado HLS Project

Add/Remove Files

Add/remove C-based source files (design specification)

Top Function: matrix_mul

Design Files

Name	CFLAGS
matrix_mul.c...	
matrix_mul.h	

New Vivado HLS Project

Add/Remove Files

Add/remove C-based testbench files (design test)

TestBench Files

Name	CFL
matrix_mul_t...	

New Vivado HLS Project

Solution Configuration

Create Vivado HLS solution for selected technology

Solution Name: solution1

Clock

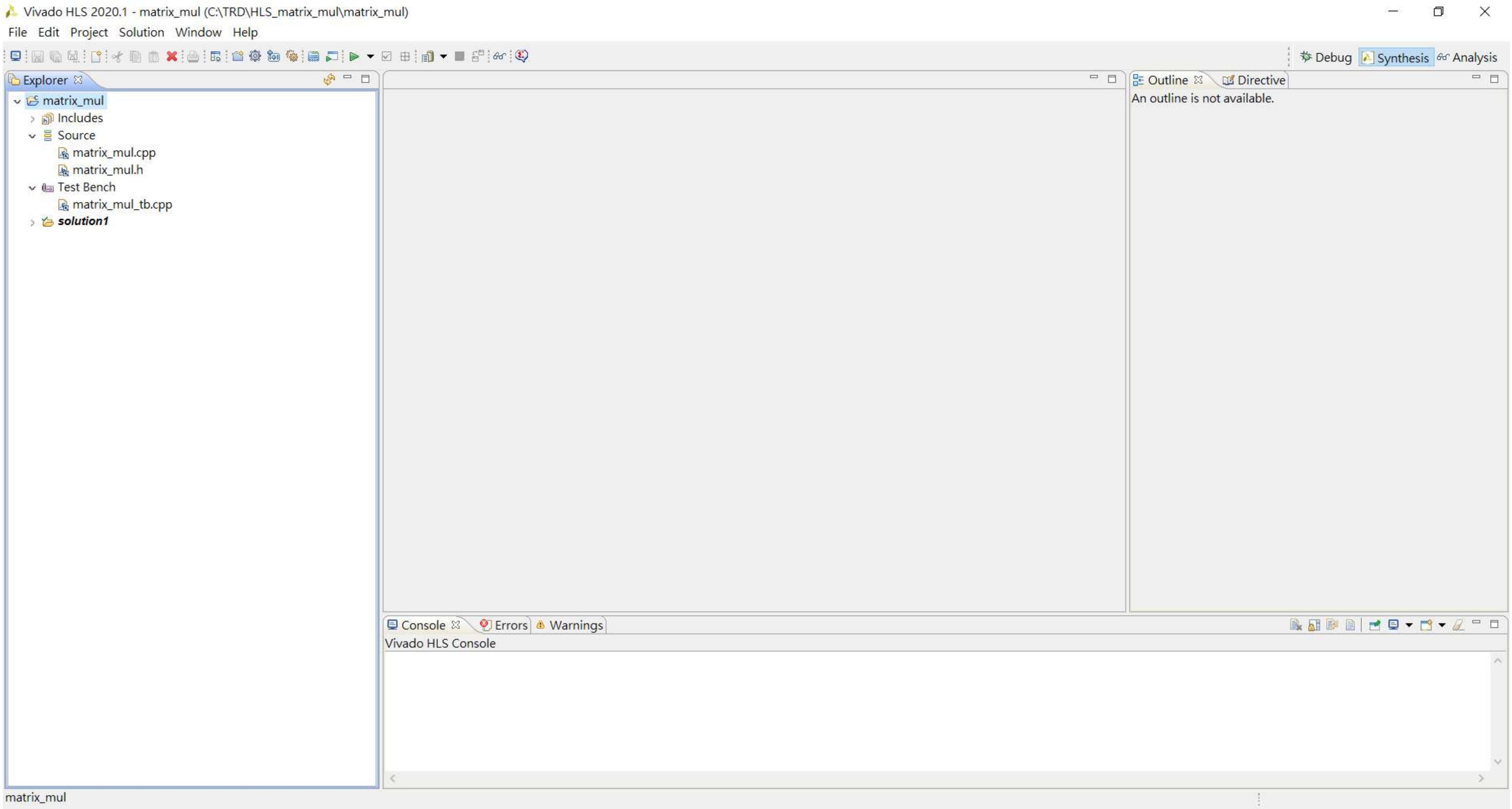
Period: 10 Uncertainty:

Part Selection

Part: *Zynq UltraScale+ ZCU102 Evaluation Board (xczu9eg-ffvb1156-2-e)*

□ Vitis Bottom Up Flow

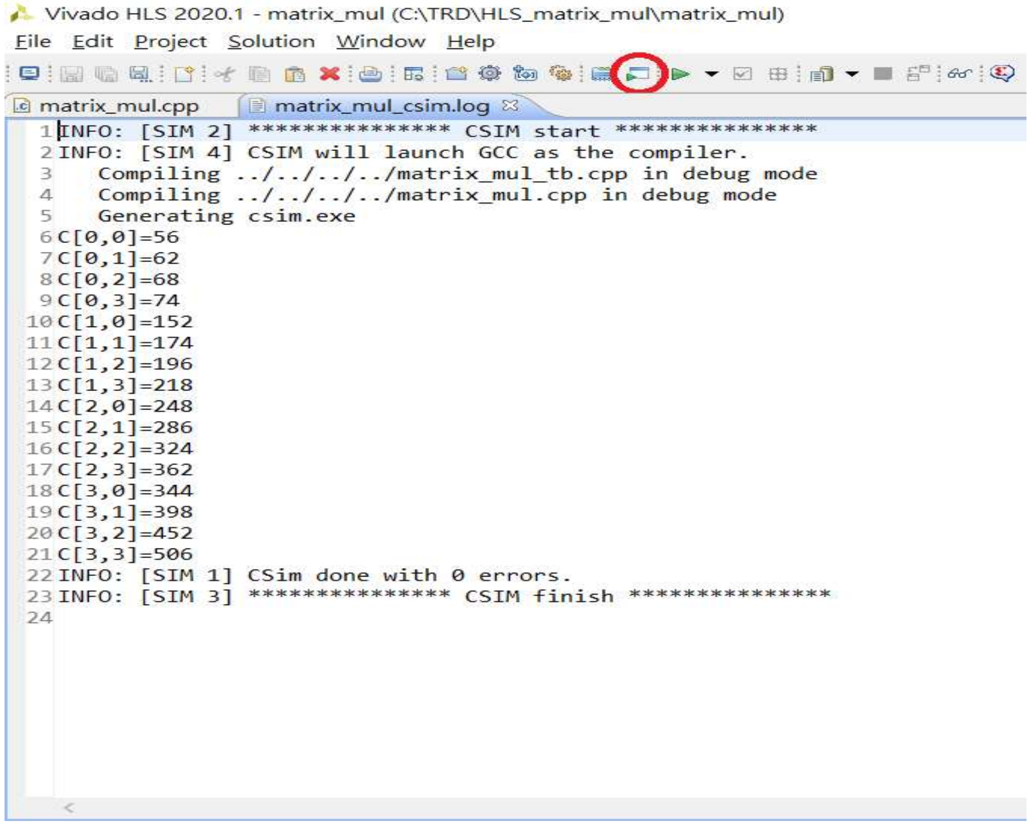
HLS IP 設計流程 –建立 HLS 專案



HLS IP 設計流程 – HLS C 模擬與合成

HLS 模擬：

– Run C Simulation

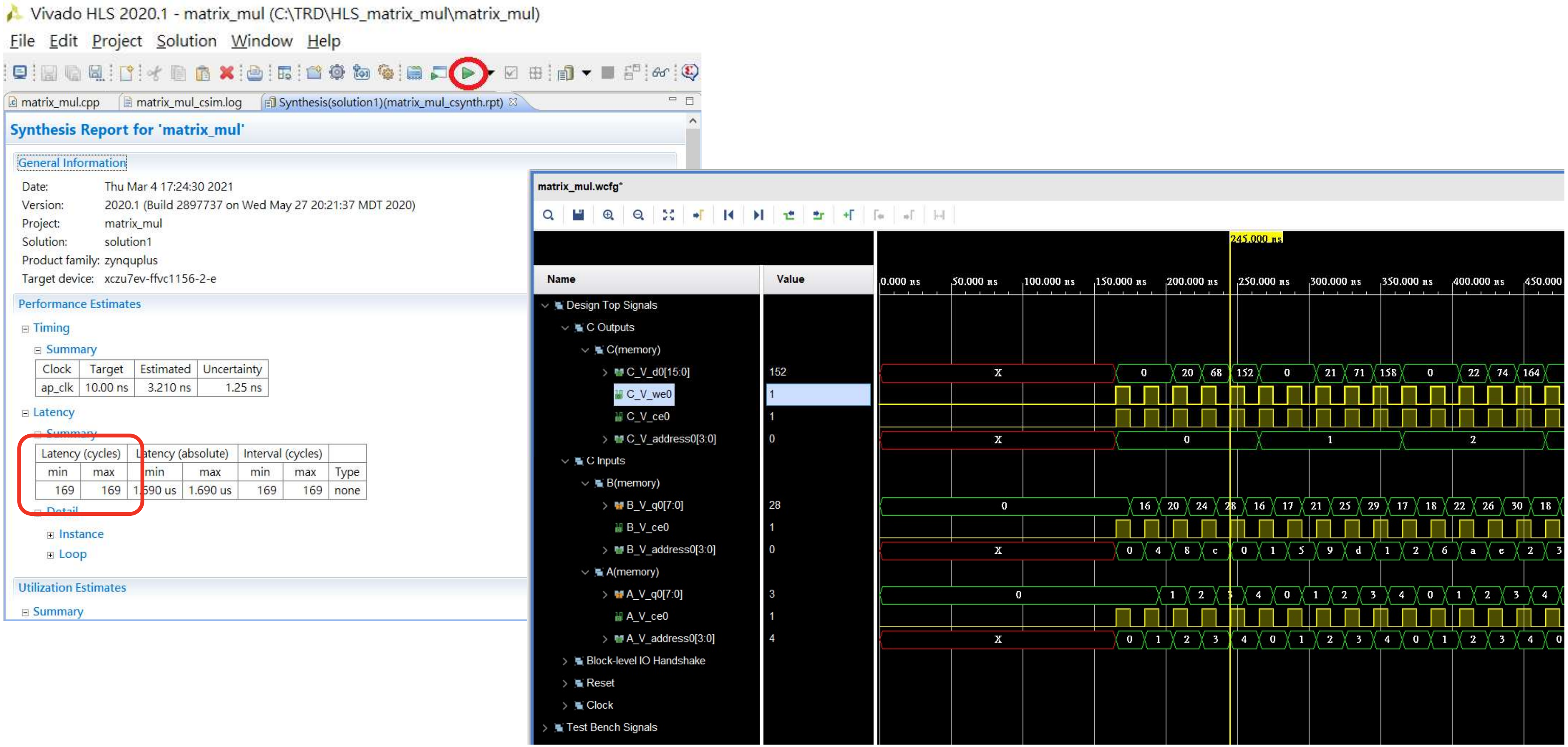


```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../matrix_mul_tb.cpp in debug mode
   Compiling ../../../../matrix_mul.cpp in debug mode
   Generating csim.exe
C[0,0]=56
C[0,1]=62
C[0,2]=68
C[0,3]=74
C[1,0]=152
C[1,1]=174
C[1,2]=196
C[1,3]=218
C[2,0]=248
C[2,1]=286
C[2,2]=324
C[2,3]=362
C[3,0]=344
C[3,1]=398
C[3,2]=452
C[3,3]=506
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

HLS IP 設計流程 – HLS C 模擬與合成

- HLS 合成：

– Run C Synthesis



Vivado HLS 2020.1 - matrix_mul (C:\TRD\HLS_matrix_mul\matrix_mul)

File Edit Project Solution Window Help

matrix_mul.cpp | matrix_mul_csim.log | Synthesis(solution1)(matrix_mul_csynth.rpt)

Synthesis Report for 'matrix_mul'

General Information

Date:	Thu Mar 4 17:24:30 2021
Version:	2020.1 (Build 2897737 on Wed May 27 20:21:37 MDT 2020)
Project:	matrix_mul
Solution:	solution1
Product family:	zynquplus
Target device:	xczu7ev-ffvc1156-2-e

Performance Estimates

⊟ Timing

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	3.210 ns	1.25 ns

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
169	169	1.690 us	1.690 us	169	169	none

⊟ Detail

⊞ Instance

⊞ Loop

Utilization Estimates

⊟ Summary

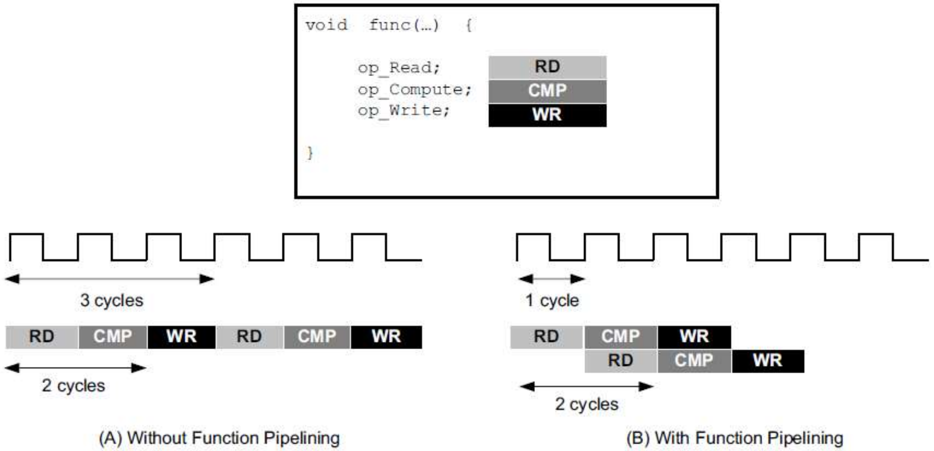
matrix_mul.wcfg*

Name	Value
Design Top Signals	
C Outputs	
C(memory)	
C_V_d0[15:0]	152
C_V_we0	1
C_V_ce0	1
C_V_address0[3:0]	0
C Inputs	
B(memory)	
B_V_q0[7:0]	28
B_V_ce0	1
B_V_address0[3:0]	0
A(memory)	
A_V_q0[7:0]	3
A_V_ce0	1
A_V_address0[3:0]	4
Block-level IO Handshake	
Reset	
Clock	
Test Bench Signals	

HLS IP 設計流程 – HLS 優化

- **Pipeline 定義：**
  - 下一個操作不需要等上一個操作完成所有步驟，就可以進行運作，用於函數與循環。

- **範例：**
  - ➢1-50：如果沒有 pipeline 優化指令，則函數每 3 個時鐘週期讀取一個數值，每 2 個時鐘週期輸出一個數值。函數的 Initiation Interval（II）為 3，latency 為2。加了 pipeline 之後，II = 1 。
  - ➢1-51：使用 pipeline 優化指令，可以將一個 8 時鐘週期的循環（II = 3）改為 4時鐘週期。



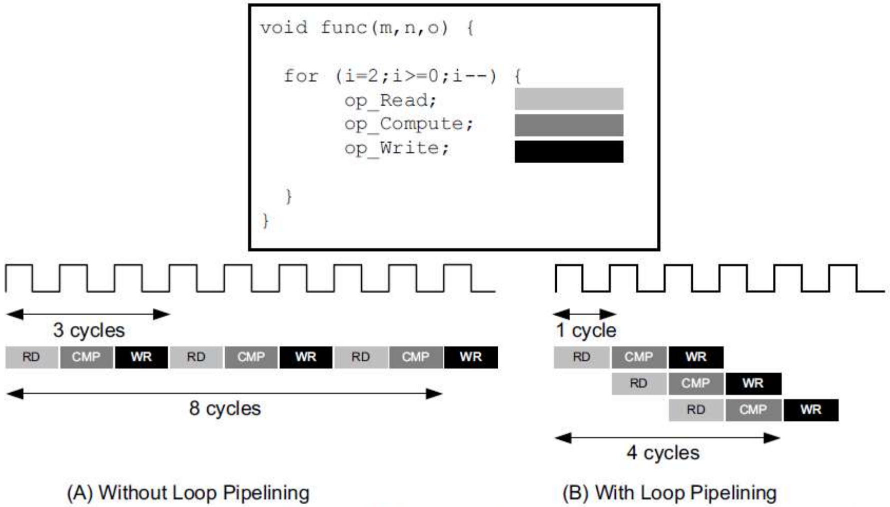
```
void  func(…)  {

    op_Read;
    op_Compute;
    op_Write;

}
```

(A) Without Function Pipelining

(B) With Function Pipelining

Figure 1-50 : Function Pipelining Behavior

X14269



```
void func(m,n,o) {

  for (i=2;i>=0;i--) {
      op_Read;
      op_Compute;
      op_Write;

  }
}
```

(A) Without Loop Pipelining

(B) With Loop Pipelining

Figure 1-51 : Loop Pipelining

X14277

HLS IP 設計流程 – HLS 優化

- **Pipeline 定義：**
  - ✓**Rewinding pipelined loops**：可以使循環成為函數中的頂級循環或在使用 **DATAFLOW** 優化的區域中使用的循環。

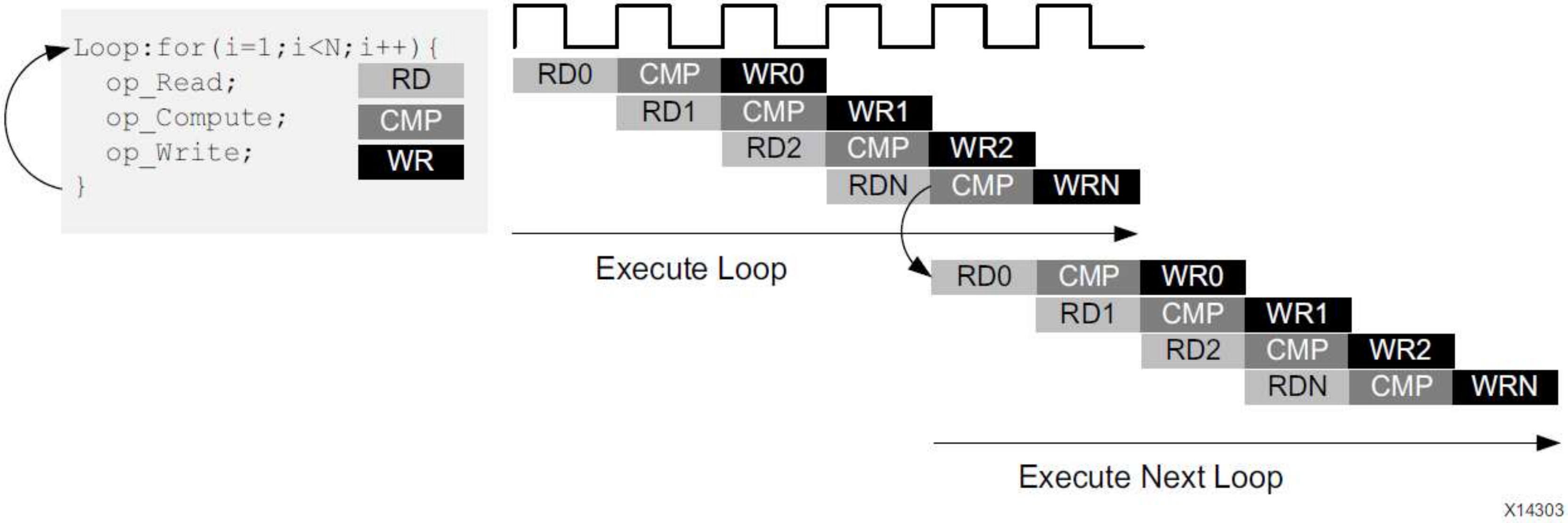
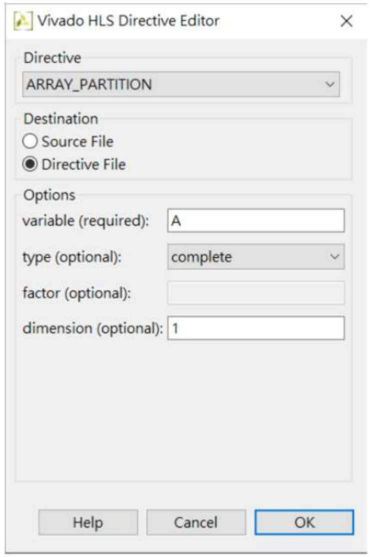


Figure 1-63: **Loop Pipelining with Rewind Option**

- **範例：**
  - ➢**1-63**：顯示對循環進行流水處理時使用 **Rewind** 的操作。在循環迭代計數結束時，循環立即開始重新執行。

HLS IP 設計流程 – HLS 優化

- ARRAY_PARTITION 定義：
  - type：
    - ✓Block：將數值以 Block 型態分割，並且資料是按順序放置。
    - ✓Cyclic：將數值以 Cyclic 型態分割，並且資料是按交叉放置。
    - ✓Complete：將數值全部打散，可同时獲取所有資料，是以暫存器方式實現。
  - dimension：允許對不同維度進行分割，dim = 1 代表是一維，dim = 2 代表是二維。



Vivado HLS Directive Editor

Directive

ARRAY_PARTITION

Destination

○ Source File

● Directive File

Options

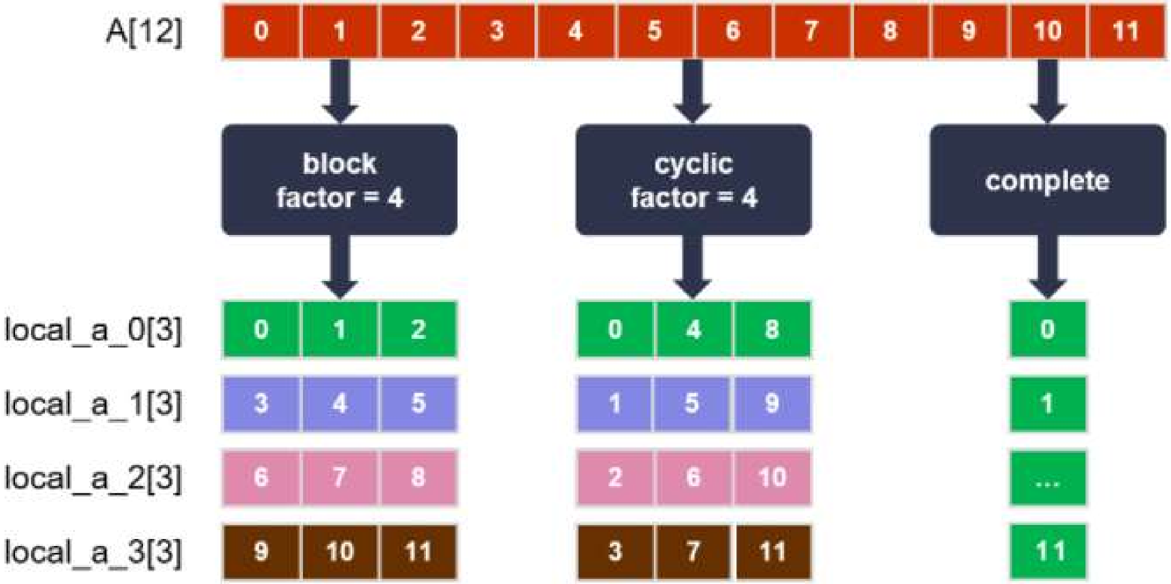
variable (required): A

type (optional): complete

factor (optional):

dimension (optional): 1

- 範例：

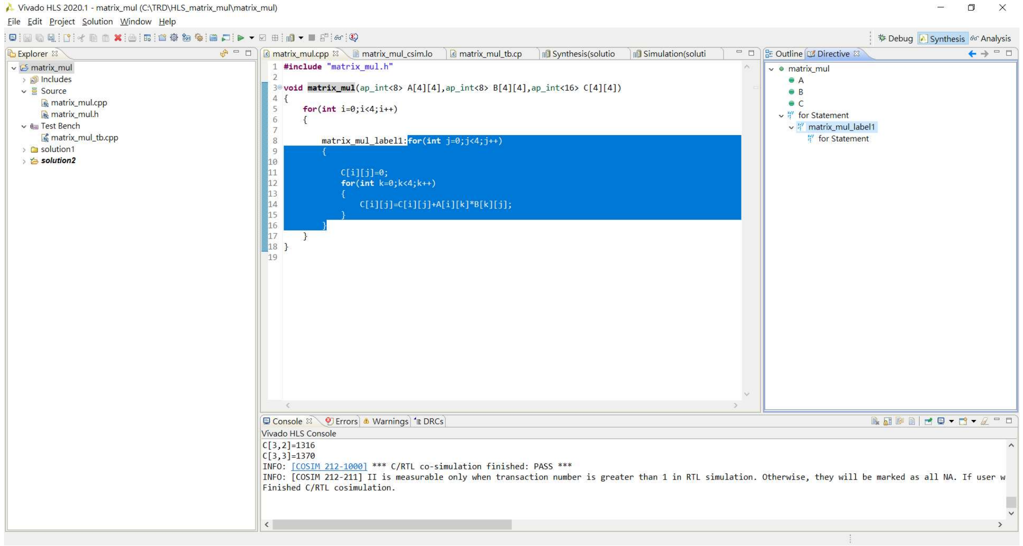


A[12]	0	1	2	3	4	5	6	7	8	9	10	11

	block factor = 4			cyclic factor = 4			complete
local_a_0[3]	0	1	2	0	4	8	0
local_a_1[3]	3	4	5	1	5	9	1
local_a_2[3]	6	7	8	2	6	10	...
local_a_3[3]	9	10	11	3	7	11	11

HLS IP 設計流程 – HLS 優化

- Pipeline 設定：
  - matrix_mul_label1上右鍵 -> Insert Directive…
  - Directive -> PIPELINE
  - 選取 enable loop rewinding



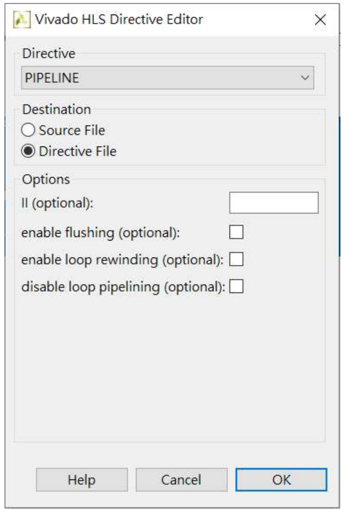
Vivado HLS 2020.1 - matrix_mul (C:\TRD\HLS_matrix_mul\matrix_mul)

```
#include "matrix_mul.h"

void matrix_mul(ap_int<8> A[4][4],ap_int<8> B[4][4],ap_int<16> C[4][4])
{
    for(int i=0;i<4;i++)
    {

        matrix_mul_label1:for(int j=0;j<4;j++)
        {

            C[i][j]=0;
            for(int k=0;k<4;k++)
            {
                C[i][j]=C[i][j]+A[i][k]*B[k][j];
            }
        }
    }
}
```

Vivado HLS Console

```
C[3,2]=1316
C[3,3]=1370
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulation. Otherwise, they will be marked as all NA. If user w
Finished C/RTL cosimulation.
```

- ARRAY_PARTITION 設定：
  - A/B 上右鍵 -> Insert Directive…
  - Directive -> ARRAY_PARTITION
  - dim -> 2/1

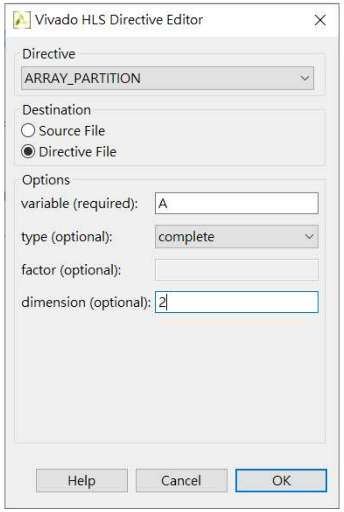


Vivado HLS Directive Editor

Directive: PIPELINE

Destination
○ Source File
● Directive File

Options
II (optional):
enable flushing (optional): □
enable loop rewinding (optional): □
disable loop pipelining (optional): □

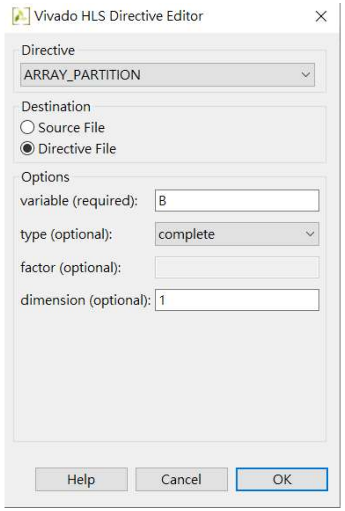


Vivado HLS Directive Editor

Directive: ARRAY_PARTITION

Destination
○ Source File
● Directive File

Options
variable (required): A
type (optional): complete
factor (optional):
dimension (optional): 2



Vivado HLS Directive Editor

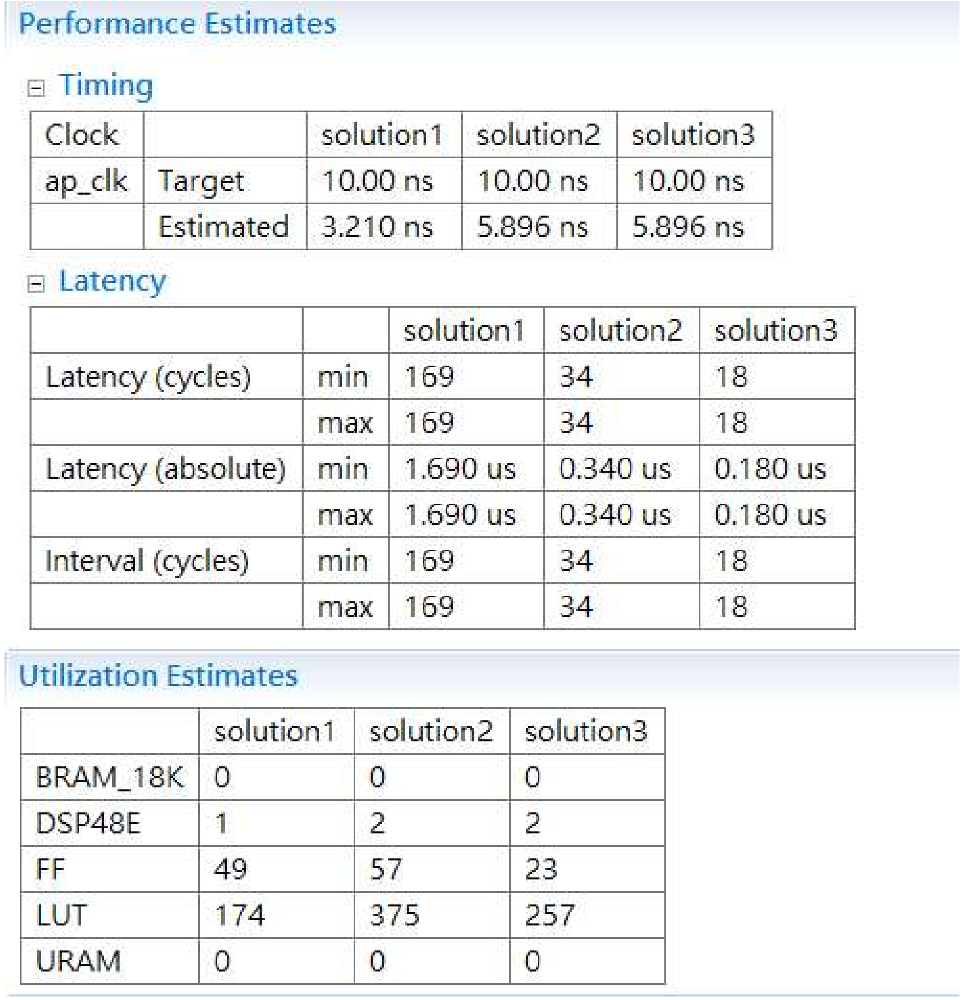
Directive: ARRAY_PARTITION

Destination
○ Source File
● Directive File

Options
variable (required): B
type (optional): complete
factor (optional):
dimension (optional): 1

HLS IP 設計流程 – HLS 解決方案比較

- solution1 設定：
  - 沒有任何優化
- solution2 設定：
  - 優化 Pipeline
- solution3 設定：
  - 優化 Pipeline
  - 優化 ARRAY_PARTITION



Performance Estimates

⊟ Timing

Clock		solution1	solution2	solution3
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	3.210 ns	5.896 ns	5.896 ns

⊟ Latency

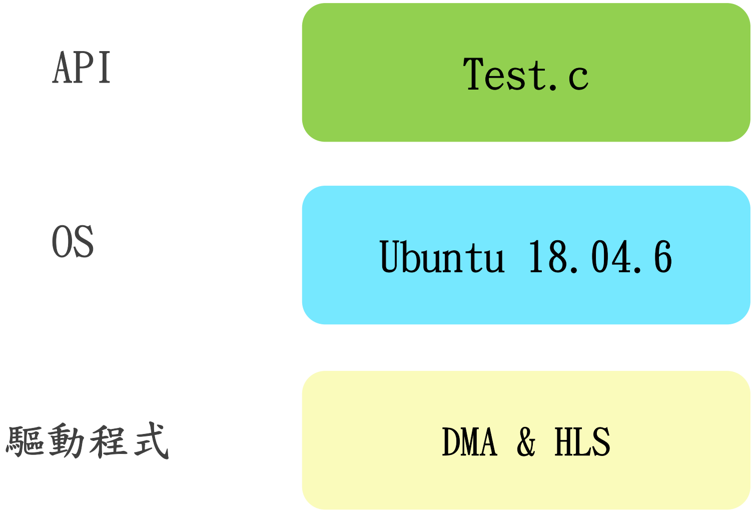
		solution1	solution2	solution3
Latency (cycles)	min	169	34	18
	max	169	34	18
Latency (absolute)	min	1.690 us	0.340 us	0.180 us
	max	1.690 us	0.340 us	0.180 us
Interval (cycles)	min	169	34	18
	max	169	34	18

Utilization Estimates

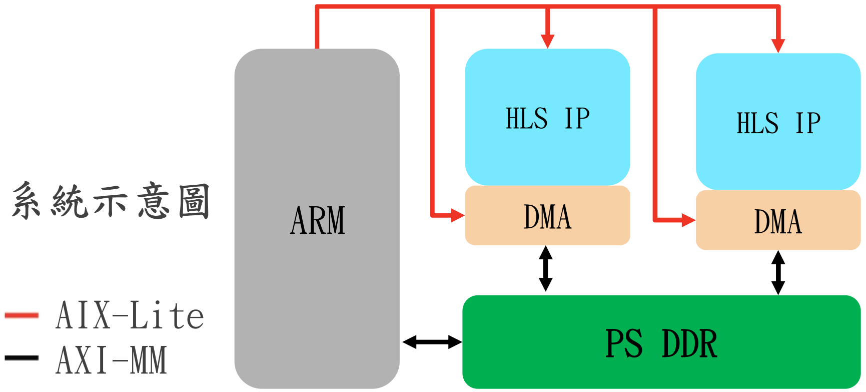
	solution1	solution2	solution3
BRAM_18K	0	0	0
DSP48E	1	2	2
FF	49	57	23
LUT	174	375	257
URAM	0	0	0

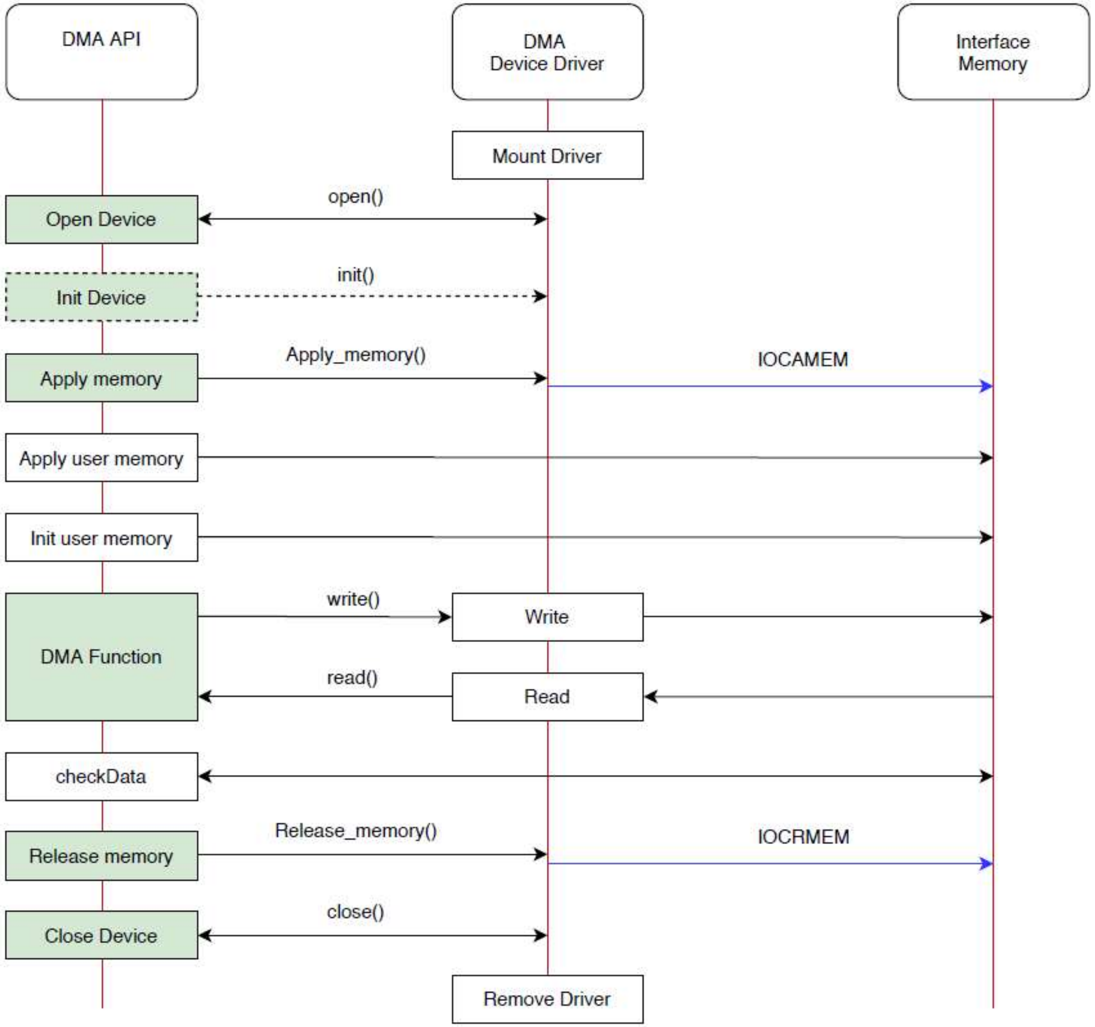
系統架構 – 加法器

系統架構 – 加法器


API
Test.c
OS
Ubuntu 18.04.6
驅動程式
DMA & HLS

系統示意圖


ARM
HLS IP
DMA
HLS IP
DMA
PS DDR


DMA API
DMA Device Driver
Interface Memory
Mount Driver
Open Device
open()
Init Device
init()
Apply memory
Apply_memory()
IOCAMEM
Apply user memory
Init user memory
DMA Function
write()
Write
read()
Read
checkData
Release memory
Release_memory()
IOCRMEM
Close Device
close()
Remove Driver

HLS IP 設計流程

HLS IP 設計流程 – CPP 與 Test Bench

- Example.cpp

```
#include "ap_axi_sdata.h"

void example(ap_axis<32,1,1,1> A[50], ap_axis<32,1,1,1> B[50]){

        int i;

        add_loop:
        for(i = 0; i < 50; i++){
                B[i].data = A[i].data.to_int() + 5;
                B[i].keep = A[i].keep;
                B[i].strb = A[i].strb;
                B[i].user = A[i].user;
                B[i].last = A[i].last;
                B[i].id = A[i].id;
                B[i].dest = A[i].dest;
        }
}
```

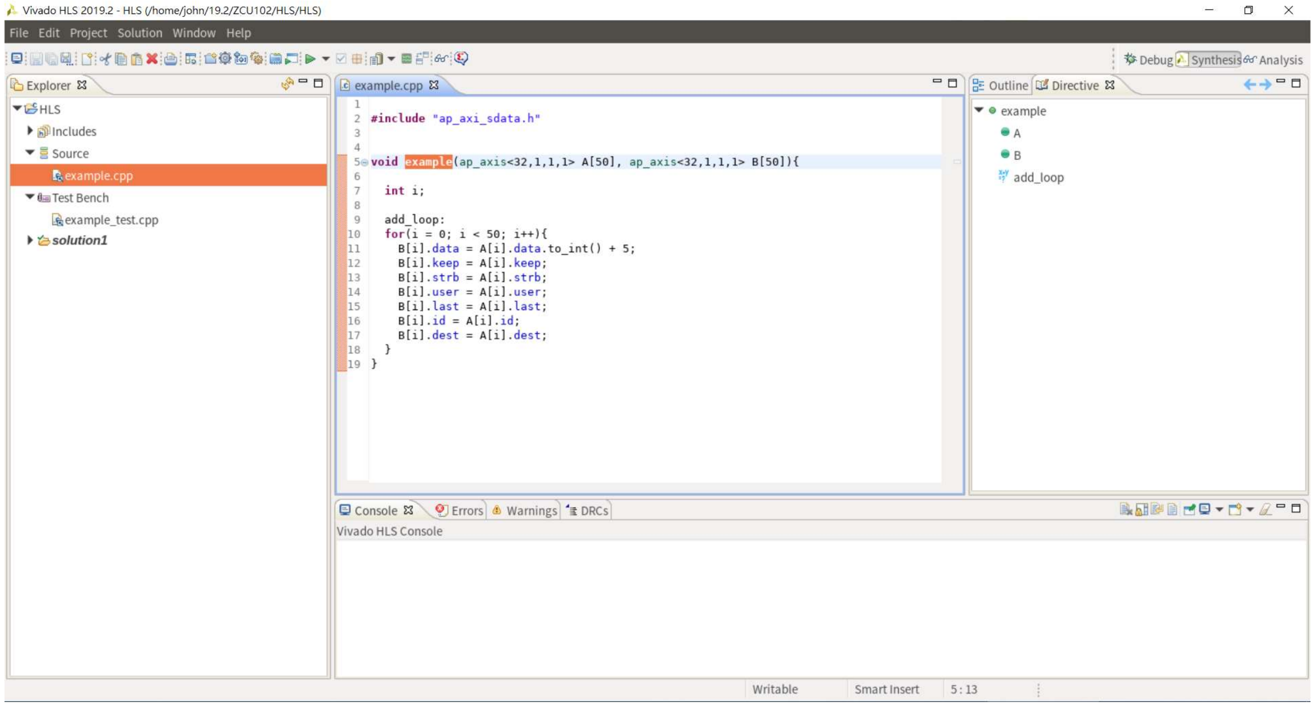
HLS IP 設計流程 – CPP 與 Test Bench

- Example_test.cpp

```
#include <stdio.h>
#include "ap_axi_sdata.h"
void example(ap_axis<32,1,1,1> A[50], ap_axis<32,1,1,1> B[50]);

int main(){
        ap_axis<32,1,1,1> A[50];
        ap_axis<32,1,1,1> B[50];

        for(int i=0; i < 50; i++){
                A[i].data = i;
                A[i].keep = 1;
                A[i].strb = 1;
                A[i].user = 1;
                A[i].last = 0;
                A[i].id = 0;
                A[i].dest = 1;
        }

        example(A,B);
        for(int i=0; i < 50; i++){
                if(B[i].data.to_int() != (i+5)){
                        printf("ERROR: HW and SW results mismatch\n");
                        return 1;
                }
        }

        printf("Success: HW and SW results match\n");
        return 0;
}
```

HLS IP 設計流程 – HLS 介面定義

- 介面分類：
  - AXI4-Stream：PL 端資料輸入與輸出介面
  - AXI4-Lite：ARM 透過 AXI4-Lite 介面進行暫存器進行操作
- 設定：
  – Source -> example.cpp -> example function -> Directive

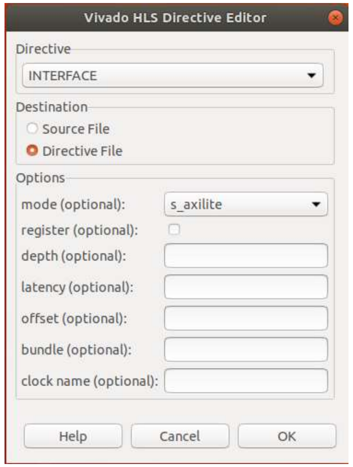


```cpp

#include "ap_axi_sdata.h"


void example(ap_axis<32,1,1,1> A[50], ap_axis<32,1,1,1> B[50]){

  int i;

  add_loop:
  for(i = 0; i < 50; i++){
    B[i].data = A[i].data.to_int() + 5;
    B[i].keep = A[i].keep;
    B[i].strb = A[i].strb;
    B[i].user = A[i].user;
    B[i].last = A[i].last;
    B[i].id = A[i].id;
    B[i].dest = A[i].dest;
  }
}
```

HLS IP 設計流程 – HLS 介面定義

- **AXI4-Lite 設定：**
  - – example 上右鍵 -> Insert Directive…
  - – Directive -> INTERFACE
  - – mode -> s_axilite

- **AXI4-Stream 設定：**
  - – A/B 上右鍵 -> Insert Directive…
  - – Directive -> INTERFACE
  - – mode -> axis



Directive

INTERFACE

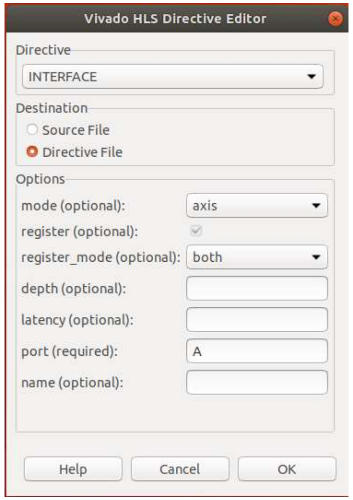
Destination

○ Source File

● Directive File

Options

mode (optional):	s_axilite
register (optional):	□
depth (optional):	
latency (optional):	
offset (optional):	
bundle (optional):	
clock name (optional):	



Directive

INTERFACE

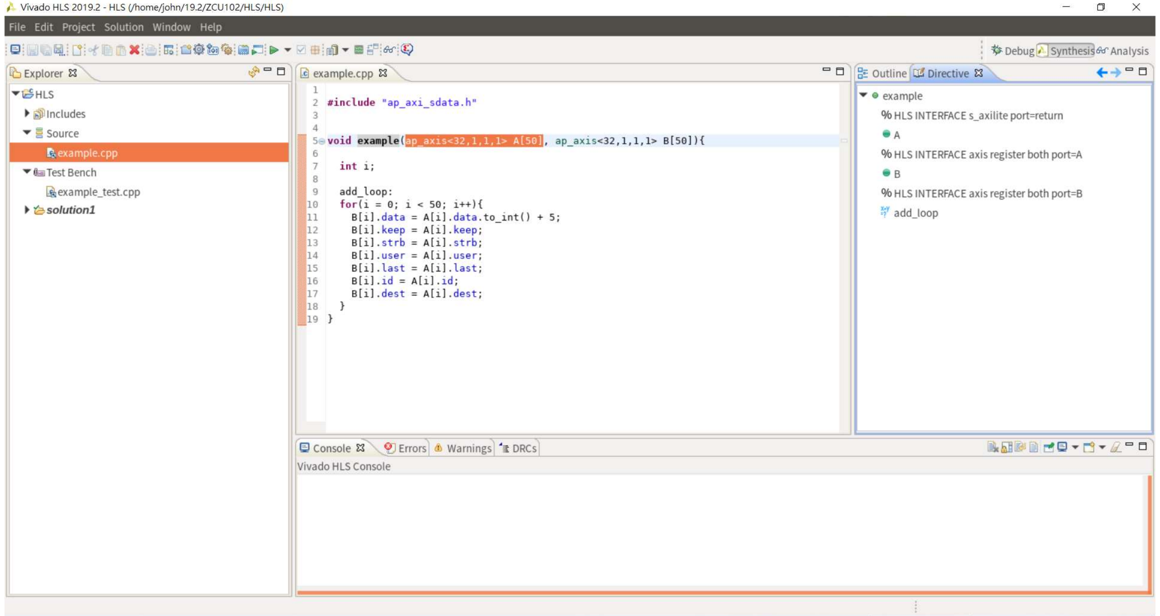
Destination

○ Source File

● Directive File

Options

mode (optional):	axis
register (optional):	☑
register_mode (optional):	both
depth (optional):	
latency (optional):	
port (required):	A
name (optional):	



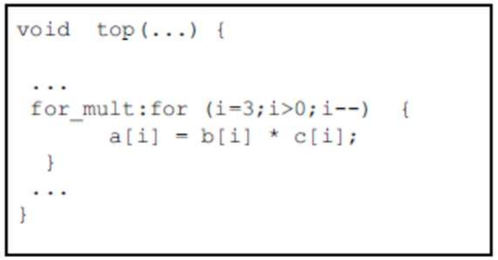
```
#include "ap_axi_sdata.h"


void example(ap_axis<32,1,1,1> A[50], ap_axis<32,1,1,1> B[50]){

  int i;

  add_loop:
  for(i = 0; i < 50; i++){
    B[i].data = A[i].data.to_int() + 5;
    B[i].keep = A[i].keep;
    B[i].strb = A[i].strb;
    B[i].user = A[i].user;
    B[i].last = A[i].last;
    B[i].id = A[i].id;
    B[i].dest = A[i].dest;
  }
}
```

- example
  - % HLS INTERFACE s_axilite port=return
  - A
  - % HLS INTERFACE axis register both port=A
  - B
  - % HLS INTERFACE axis register both port=B
  - add_loop

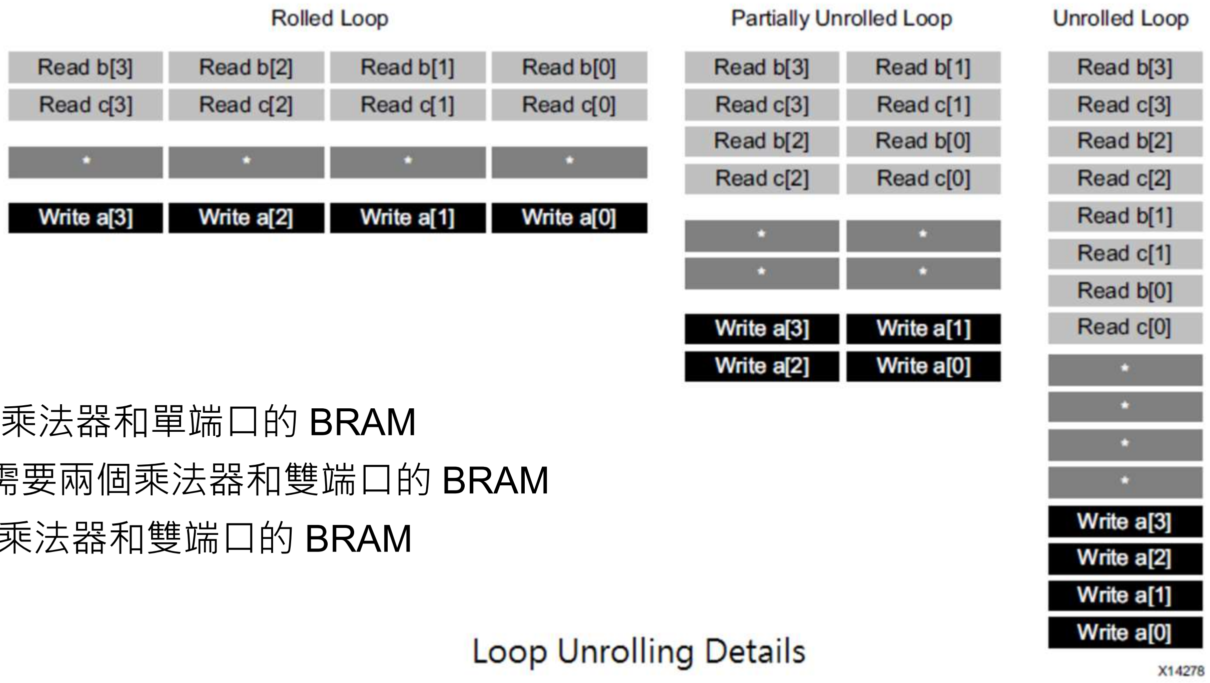
HLS IP 設計流程 – HLS 優化

UNROLL 定義：

- ✓Rolled loop：每個函式都是單獨的時間周期。
- ✓Partially unrolled loop：Loop 將被 2 的因子展開。
- ✓Unrolled loop：完全 unrolled，因此需要將資料進行劃分。



```
void  top(...) {

 ...
 for_mult:for (i=3;i>0;i--)  {
       a[i] = b[i] * c[i];
  }
 ...
}
```



Rolled Loop

Read b[3] | Read b[2] | Read b[1] | Read b[0]
Read c[3] | Read c[2] | Read c[1] | Read c[0]
* | * | * | *
Write a[3] | Write a[2] | Write a[1] | Write a[0]

Partially Unrolled Loop

Read b[3] | Read b[1]
Read c[3] | Read c[1]
Read b[2] | Read b[0]
Read c[2] | Read c[0]
* | *
* | *
Write a[3] | Write a[1]
Write a[2] | Write a[0]

Unrolled Loop

Read b[3]
Read c[3]
Read b[2]
Read c[2]
Read b[1]
Read c[1]
Read b[0]
Read c[0]
*
*
*
*
Write a[3]
Write a[2]
Write a[1]
Write a[0]

Loop Unrolling Details

X14278

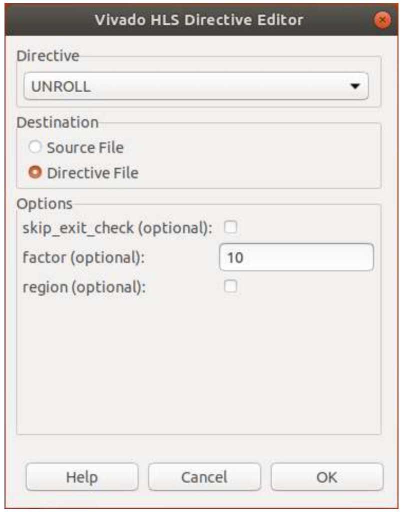
範例：

- ➢Rolled loop：需要四個時間周期，只需要一個乘法器和單端口的 BRAM
- ➢Partially unrolled loop：需要兩個時間周期，需要兩個乘法器和雙端口的 BRAM
- ➢Unrolled loop：需要四次資料寫讀，需要四個乘法器和雙端口的 BRAM

HLS IP 設計流程 – HLS 優化

- **UNROLL 設定：**
  - **add_loop 上右鍵 -> Insert Directive…**
  - **Directive -> UNROLL**
  - **factor -> 10**

- **Pipeline 設定：**
  - **add_loop 上右鍵 -> Insert Directive…**
  - **Directive -> PIPELINE**
  - **選取 enable loop rewinding**

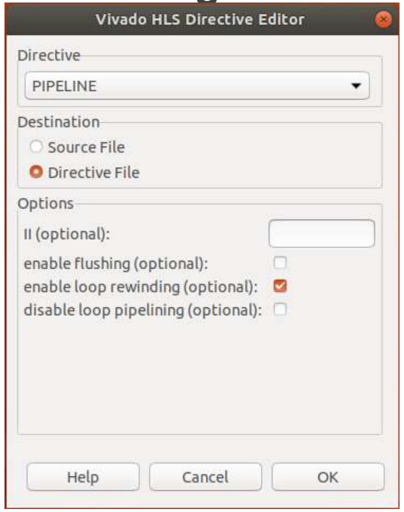


Vivado HLS Directive Editor

Directive: UNROLL

Destination
○ Source File
● Directive File

Options
skip_exit_check (optional): □
factor (optional): 10
region (optional): □

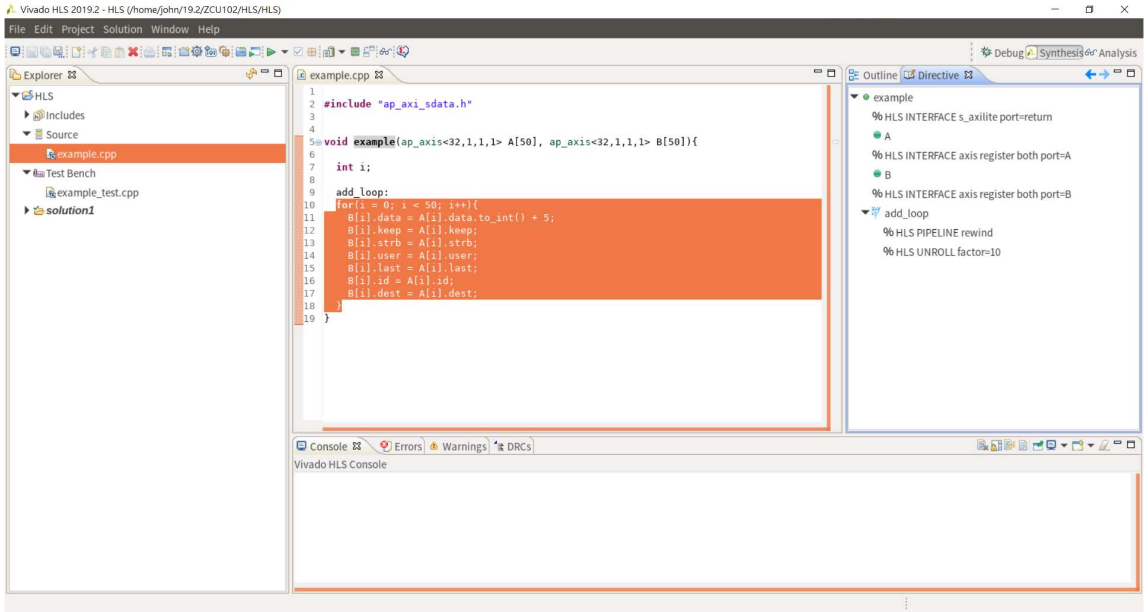


Vivado HLS Directive Editor

Directive: PIPELINE

Destination
○ Source File
● Directive File

Options
II (optional):
enable flushing (optional): □
enable loop rewinding (optional): ☑
disable loop pipelining (optional): □



```
#include "ap_axi_sdata.h"


void example(ap_axis<32,1,1,1> A[50], ap_axis<32,1,1,1> B[50]){

  int i;

  add_loop:
  for(i = 0; i < 50; i++){
    B[i].data = A[i].data.to_int() + 5;
    B[i].keep = A[i].keep;
    B[i].strb = A[i].strb;
    B[i].user = A[i].user;
    B[i].last = A[i].last;
    B[i].id = A[i].id;
    B[i].dest = A[i].dest;
  }
}
```

- example
  - % HLS INTERFACE s_axilite port=return
  - A
  - % HLS INTERFACE axis register both port=A
  - B
  - % HLS INTERFACE axis register both port=B
  - add_loop
    - % HLS PIPELINE rewind
    - % HLS UNROLL factor=10

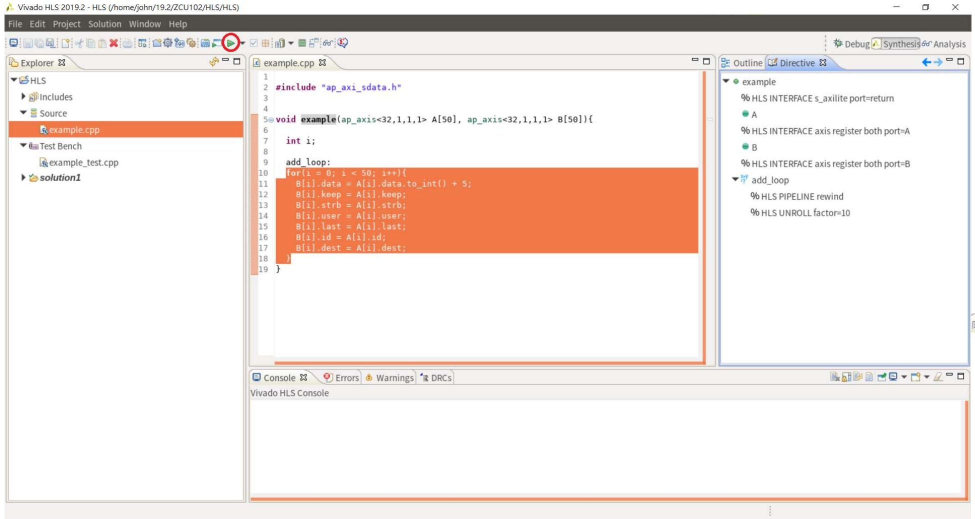
HLS IP 設計流程 – HLS 合成與模擬

HLS 合成：

- Run C Synthesis

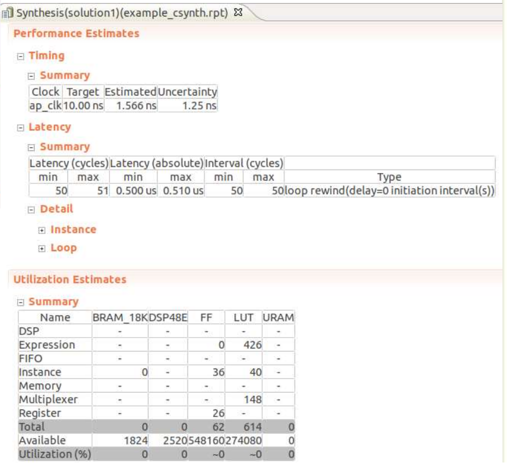
✓確認資源使用率、延遲與間隔時間

✓需要確認 AXI4-Lite 與 AXI4-Stream 介面是否正確產生



```
#include "ap_axi_sdata.h"


void example(ap_axis<32,1,1,1> A[50], ap_axis<32,1,1,1> B[50]){

  int i;

  add_loop:
  for(i = 0; i < 50; i++){
    B[i].data = A[i].data.to_int() + 5;
    B[i].keep = A[i].keep;
    B[i].strb = A[i].strb;
    B[i].user = A[i].user;
    B[i].last = A[i].last;
    B[i].id = A[i].id;
    B[i].dest = A[i].dest;
  }
}
```

- example
  - % HLS INTERFACE s_axilite port=return
  - A
  - % HLS INTERFACE axis register both port=A
  - B
  - % HLS INTERFACE axis register both port=B
  - add_loop
    - % HLS PIPELINE rewind
    - % HLS UNROLL factor=10



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	1.566 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
50	51	0.500 us	0.510 us	50	50	loop rewind(delay=0 initiation interval(s))

Detail

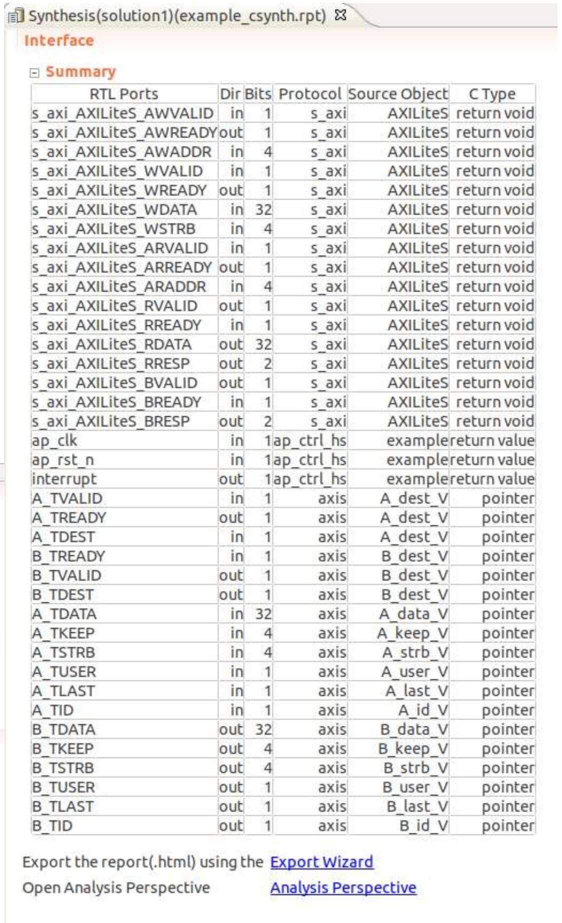
Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	426	-
FIFO	-	-	-	-	-
Instance	0	-	36	40	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	148	-
Register	-	-	26	-	-
Total	0	0	62	614	0
Available	1824	2520	548160	274080	0
Utilization (%)	0	0	~0	~0	0



Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_AXILiteS_AWVALID	in	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_AWREADY	out	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_AWADDR	in	4	s_axi	AXILiteS	return void
s_axi_AXILiteS_WVALID	in	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_WREADY	out	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_WDATA	in	32	s_axi	AXILiteS	return void
s_axi_AXILiteS_WSTRB	in	4	s_axi	AXILiteS	return void
s_axi_AXILiteS_ARVALID	in	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_ARREADY	out	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_ARADDR	in	4	s_axi	AXILiteS	return void
s_axi_AXILiteS_RVALID	out	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_RREADY	in	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_RDATA	out	32	s_axi	AXILiteS	return void
s_axi_AXILiteS_RRESP	out	2	s_axi	AXILiteS	return void
s_axi_AXILiteS_BVALID	out	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_BREADY	in	1	s_axi	AXILiteS	return void
s_axi_AXILiteS_BRESP	out	2	s_axi	AXILiteS	return void
ap_clk	in	1	ap_ctrl_hs	example	return value
ap_rst_n	in	1	ap_ctrl_hs	example	return value
interrupt	out	1	ap_ctrl_hs	example	return value
A_TVALID	in	1	axis	A_dest_V	pointer
A_TREADY	out	1	axis	A_dest_V	pointer
A_TDEST	in	1	axis	A_dest_V	pointer
B_TREADY	in	1	axis	B_dest_V	pointer
B_TVALID	out	1	axis	B_dest_V	pointer
B_TDEST	out	1	axis	B_dest_V	pointer
A_TDATA	in	32	axis	A_data_V	pointer
A_TKEEP	in	4	axis	A_keep_V	pointer
A_TSTRB	in	4	axis	A_strb_V	pointer
A_TUSER	in	1	axis	A_user_V	pointer
A_TLAST	in	1	axis	A_last_V	pointer
A_TID	in	1	axis	A_id_V	pointer
B_TDATA	out	32	axis	B_data_V	pointer
B_TKEEP	out	4	axis	B_keep_V	pointer
B_TSTRB	out	4	axis	B_strb_V	pointer
B_TUSER	out	1	axis	B_user_V	pointer
B_TLAST	out	1	axis	B_last_V	pointer
B_TID	out	1	axis	B_id_V	pointer

Export the report(.html) using the Export Wizard

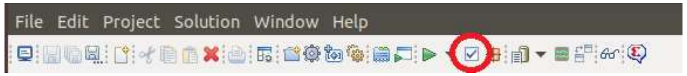
Open Analysis Perspective Analysis Perspective

HLS IP 設計流程 – HLS 合成與模擬

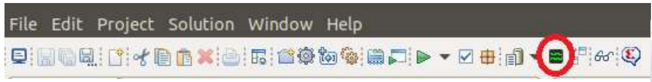
HLS 模擬：

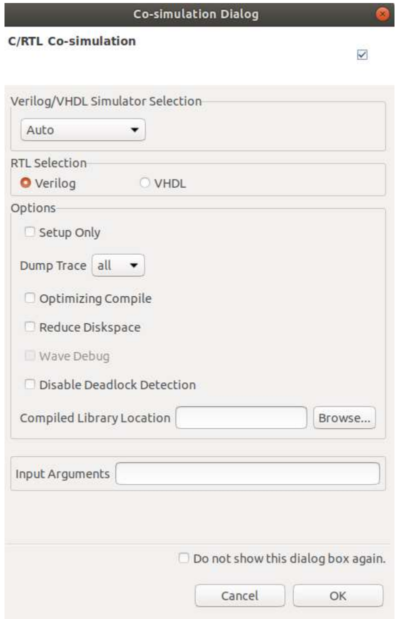
– Run C/RTL Cosimulation

– Dump Trace -> all [查看全部端口與信號波形]



– Open Wave Viewer…





Co-simulation Dialog

C/RTL Co-simulation

Verilog/VHDL Simulator Selection: Auto

RTL Selection: ● Verilog ○ VHDL

Options

□ Setup Only

Dump Trace: all

□ Optimizing Compile

□ Reduce Diskspace

□ Wave Debug

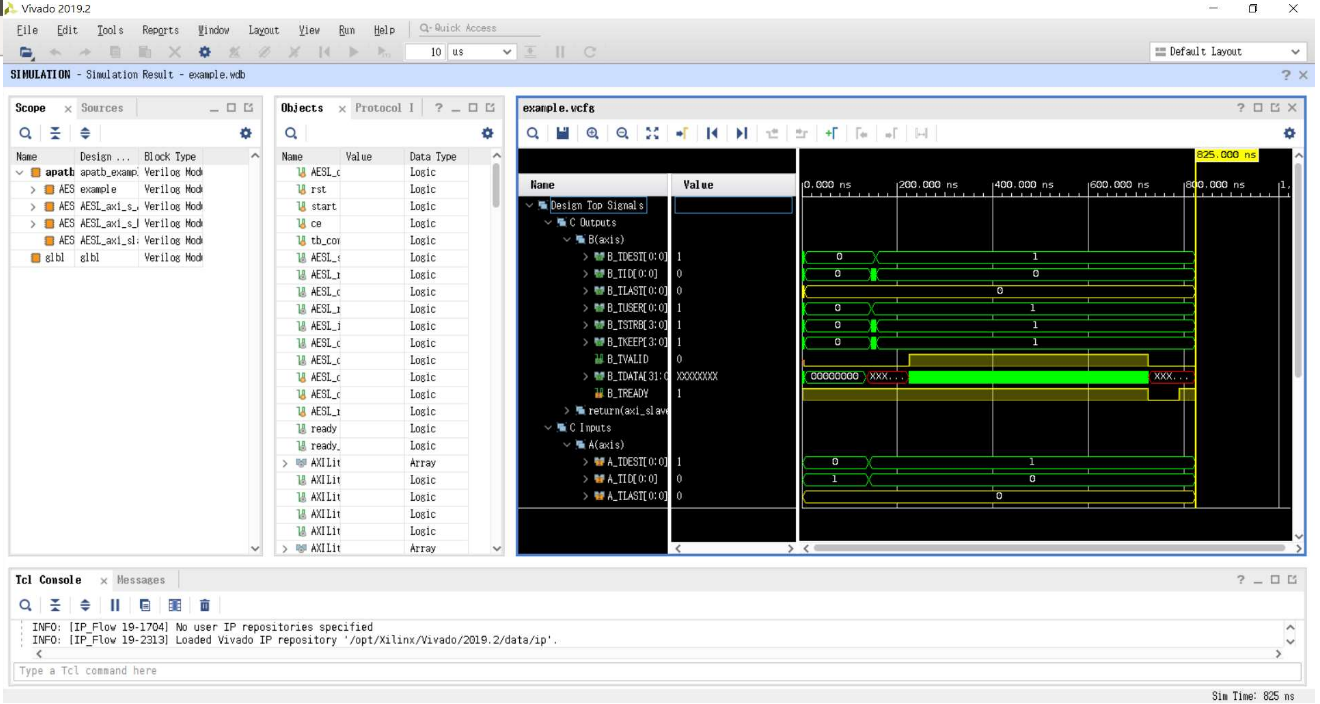
□ Disable Deadlock Detection

Compiled Library Location: Browse...

Input Arguments:

□ Do not show this dialog box again.

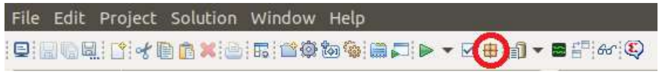
Cancel OK



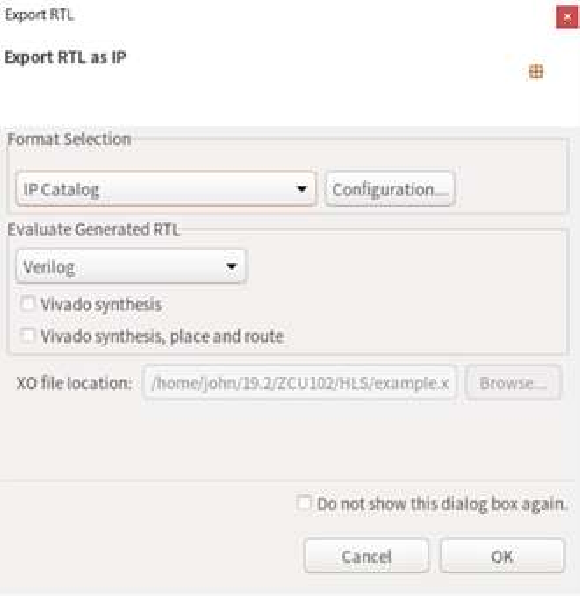
```
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/opt/Xilinx/Vivado/2019.2/data/ip'.
```

HLS IP 設計流程 – HLS IP 導出

- HLS IP 導出：
  - Export RTL



  - <HLS 專案>/solution1/impl/ip 找到 HLS IP.zip
  - <HLS 專案>/solution1/impl/ip/drivers 找到相關驅動程式


Export RTL

Export RTL as IP

Format Selection

IP Catalog ▾ Configuration...

Evaluate Generated RTL

Verilog ▾

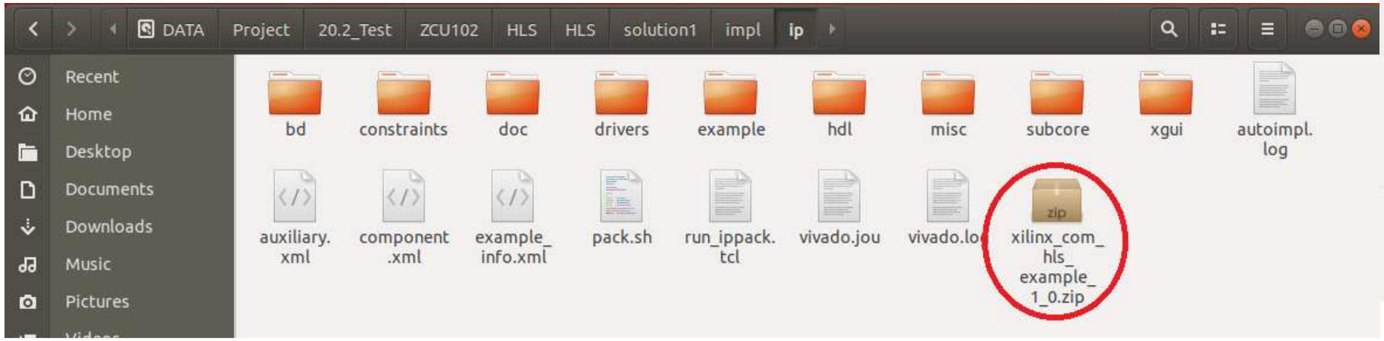
□ Vivado synthesis

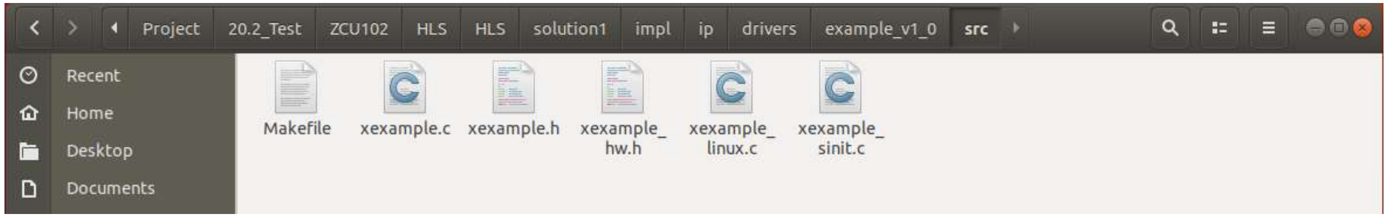
□ Vivado synthesis, place and route

XO file location: /home/john/19.2/ZCU102/HLS/example.x Browse...

□ Do not show this dialog box again.

Cancel OK

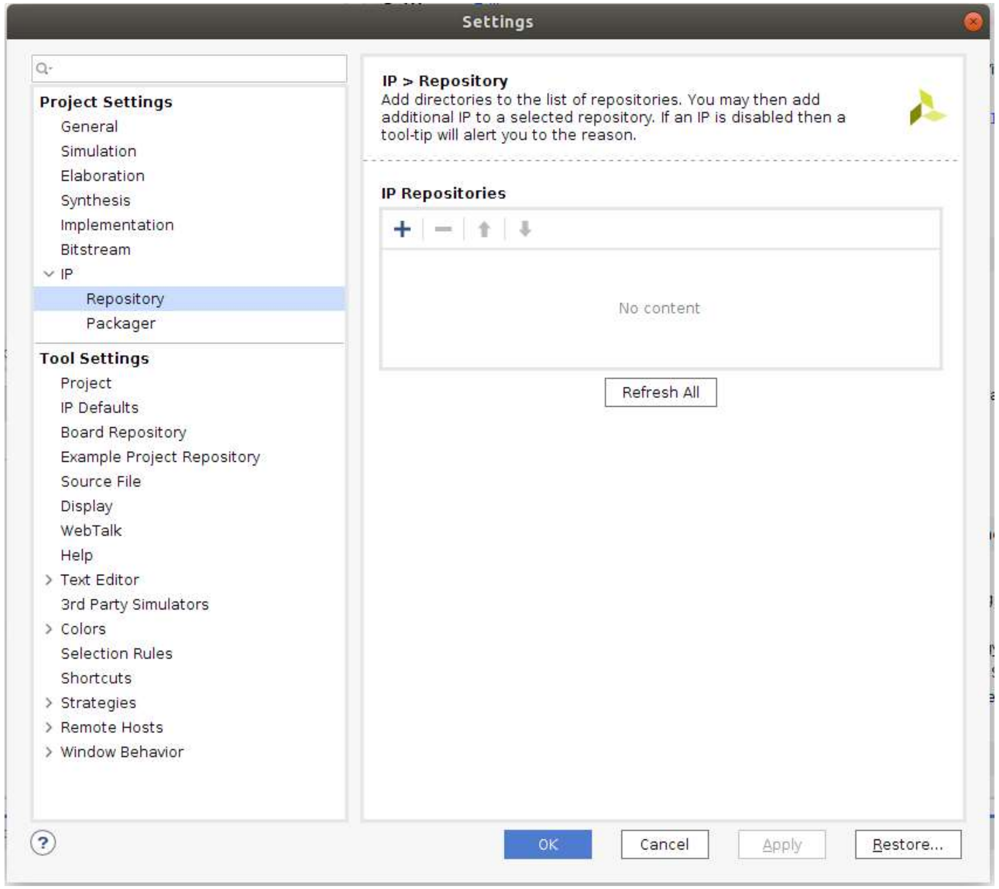




Vivado 專案

Vivado 專案 – HLS IP 設定

- HLS IP 設定：
  - Project Manager -> Settings
  - Project Settings -> IP Repository
  - IP Repositories -> 增加 IP 路徑



Settings

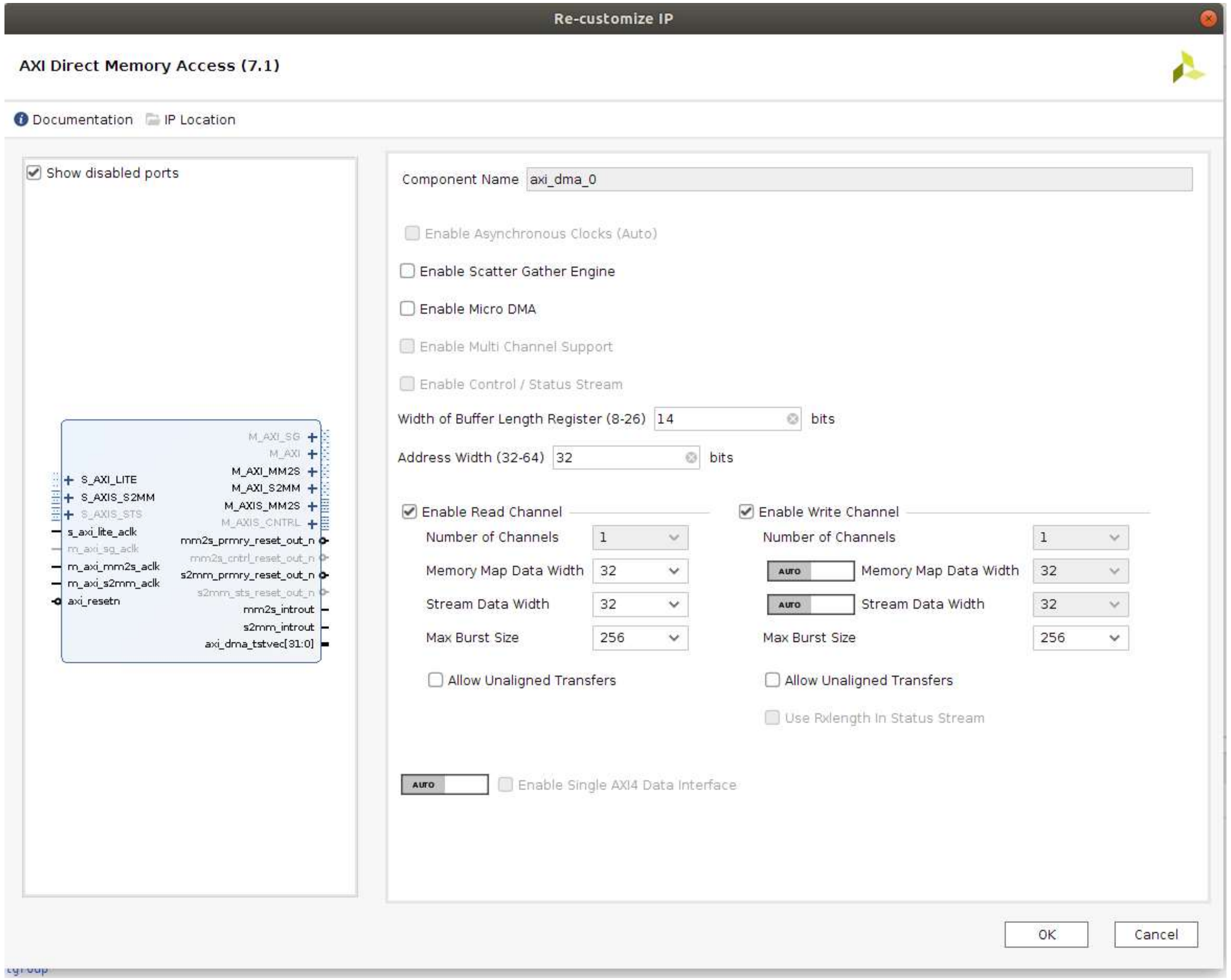
IP > Repository
Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

No content

Vivado 專案 – IPI 設定

- DMA 設定：
  - 關閉 Scatter 功能
  - Read/Write Max Burst Size 設定 256



Re-customize IP

AXI Direct Memory Access (7.1)

Documentation IP Location

☑ Show disabled ports

Component Name: axi_dma_0

□ Enable Asynchronous Clocks (Auto)

□ Enable Scatter Gather Engine

□ Enable Micro DMA

□ Enable Multi Channel Support

□ Enable Control / Status Stream

Width of Buffer Length Register (8-26): 14 bits

Address Width (32-64): 32 bits

☑ Enable Read Channel

- Number of Channels: 1
- Memory Map Data Width: 32
- Stream Data Width: 32
- Max Burst Size: 256
- □ Allow Unaligned Transfers

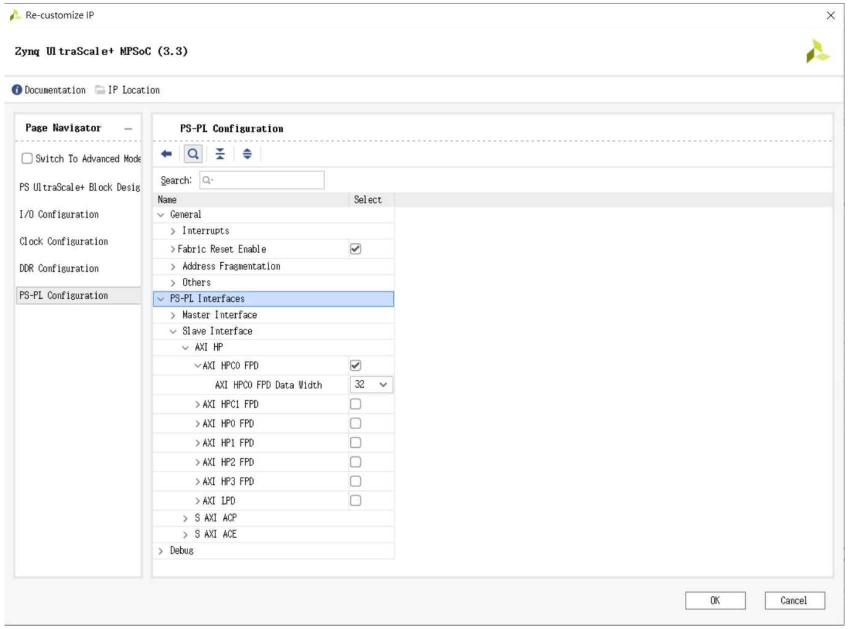
☑ Enable Write Channel

- Number of Channels: 1
- AUTO Memory Map Data Width: 32
- AUTO Stream Data Width: 32
- Max Burst Size: 256
- □ Allow Unaligned Transfers
- □ Use Rxlength In Status Stream

AUTO □ Enable Single AXI4 Data Interface

Vivado 專案 – IPI 設定

- MPSoC 設定：
  - 開啟 AXI-HPCO FPD
  - Bus 寬度為 32
  - 開啟 GPIO EMIO [Reset HLS IP]
- Slice設定：
  - 選定 EMIO 10 為 HLS IP 重置腳位



Re-customize IP

Zynq UltraScale+ MPSoC (3.3)

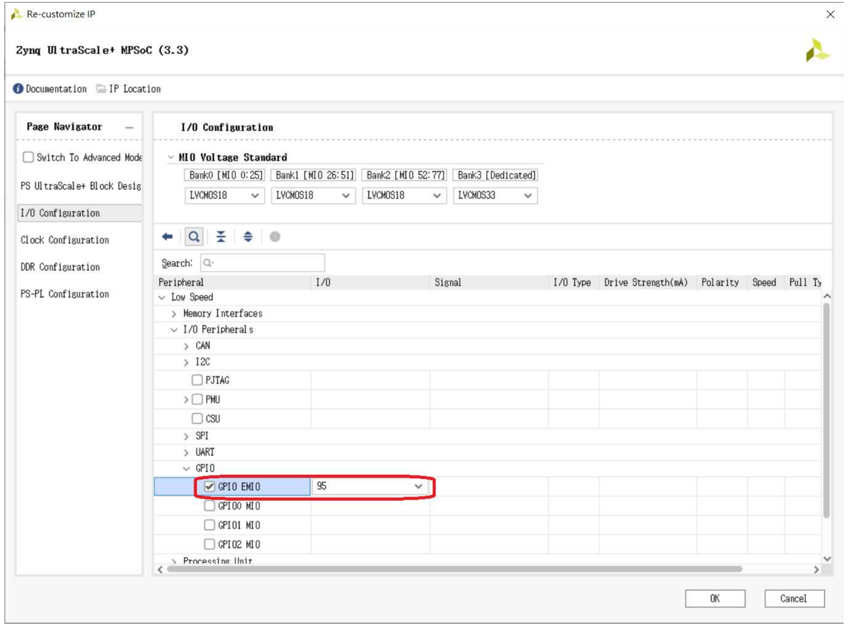
Documentation IP Location

Page Navigator: ☐ Switch To Advanced Mode / PS UltraScale+ Block Design / I/O Configuration / Clock Configuration / DDR Configuration / PS-PL Configuration

PS-PL Configuration

Search:

Name	Select
∨ General	
› Interrupts	
› Fabric Reset Enable	☑
› Address Fragmentation	
› Others	
∨ PS-PL Interfaces	
› Master Interface	
∨ Slave Interface	
∨ AXI HP	
∨ AXI HPC0 FPD	☑
AXI HPC0 FPD Data Width	32
› AXI HPC1 FPD	☐
› AXI HP0 FPD	☐
› AXI HP1 FPD	☐
› AXI HP2 FPD	☐
› AXI HP3 FPD	☐
› AXI LPD	☐
› S AXI ACP	
› S AXI ACE	
› Debug	



Re-customize IP

Zynq UltraScale+ MPSoC (3.3)

Documentation IP Location

Page Navigator: ☐ Switch To Advanced Mode / PS UltraScale+ Block Design / I/O Configuration / Clock Configuration / DDR Configuration / PS-PL Configuration

I/O Configuration

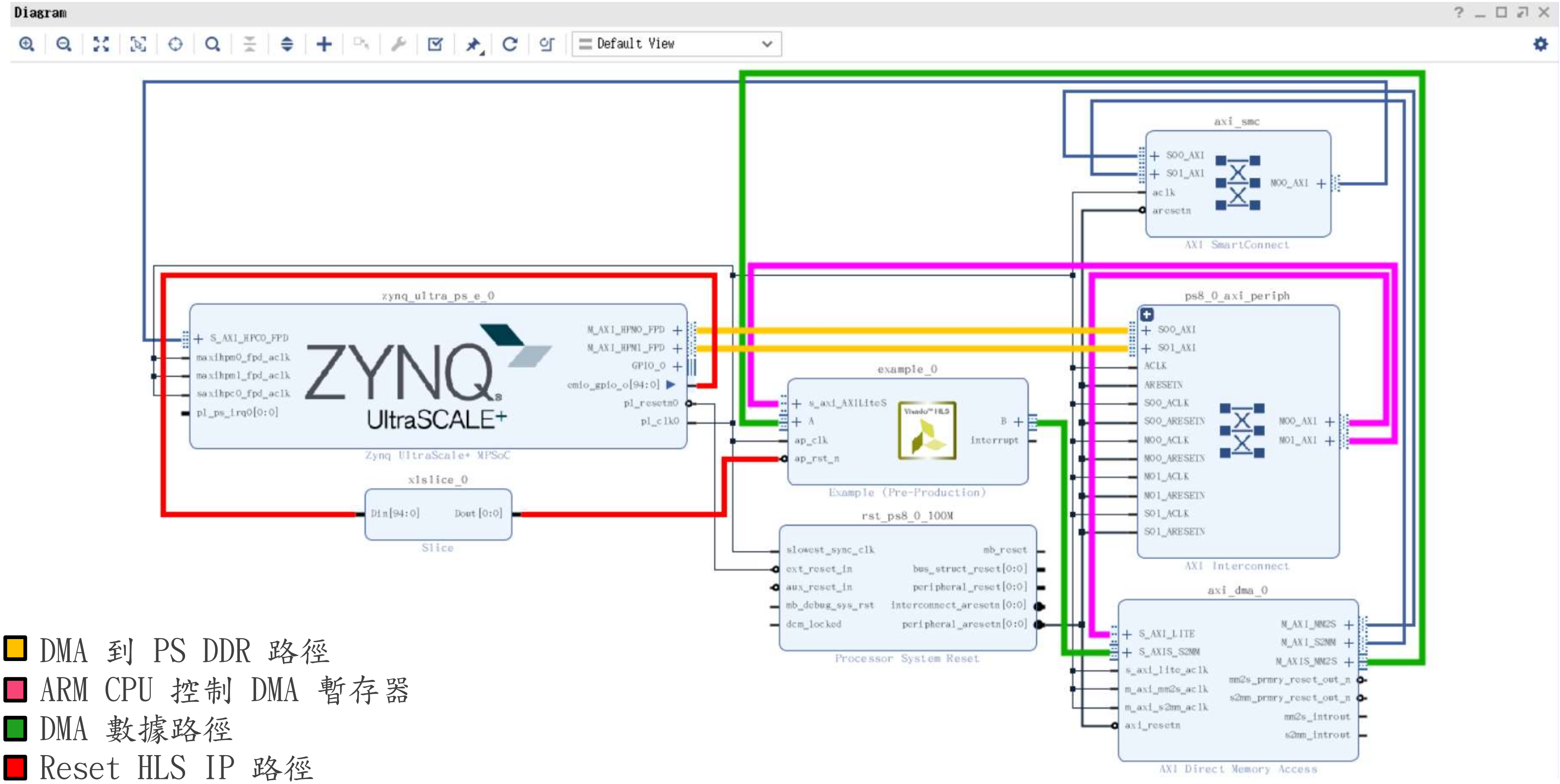
MIO Voltage Standard

Bank0 [MIO 0:25]	Bank1 [MIO 26:51]	Bank2 [MIO 52:77]	Bank3 [Dedicated]
LVCMOS18	LVCMOS18	LVCMOS18	LVCMOS33

Search:

Peripheral	I/O	Signal	I/O Type	Drive Strength(mA)	Polarity	Speed	Pull Ty
∨ Low Speed							
› Memory Interfaces							
∨ I/O Peripherals							
› CAN							
› I2C							
☐ PJTAG							
› ☐ PMU							
☐ CSU							
› SPI							
› UART							
∨ GPIO							
☑ GPIO EMIO	95						
☐ GPIO0 MIO							
☐ GPIO1 MIO							
☐ GPIO2 MIO							
› Processing Unit							

Vivado 專案 – IPI 設定



🟨 DMA 到 PS DDR 路徑

🟪 ARM CPU 控制 DMA 暫存器

🟩 DMA 數據路徑

🟥 Reset HLS IP 路徑

Linux 設計

Linux 設計 – 修改 Device Tree 內容

- **HLS IP 設定：**
  - ✓ **MPSoC EMIO 起始編號從 78 [對應 EMIO 0]**

```
example_0: example@a0010000 {
/* This is a place holder node for a custom IP, user may need to update the entries */
        clock-names = "ap_clk";
        clocks = <&zynqmp_clk 71>;
        compatible = "xxx,add-1.0";
        reg = <0x0 0xa0010000 0x0 0x10000>;
        reset-gpios = <&gpio 88 0>;
        xlnx,s-axi-axilites-addr-width = <0x4>;
        xlnx,s-axi-axilites-data-width = <0x20>;
};
```

Linux 設計 – 修改 Driver 內容

ARM GPIO 架構：

- ✓ 在 kernel 3.13 後，Linux 引入新的 GPIOD API，所有函數皆以 gpiod_ 為前綴。
- ✓ 新的 GPIO 方式，不需要手動釋放資源。
- ✓ 使用 GPIO 函數前需要引用頭文件 #include <linux/gpio/consumer.h>
- ✓ 取得 GPIO 資源並回傳一個 struct gpio_desc 的結構指標：
  - ➢ struct gpio_desc *devm_gpiod_get(struct device *dev, const char *con_id, enum gpiod_flags flags)
    - flags：用於指定方向和初始值
      1. GPIOD_ASIS 或 0：表示不初始化 GPIO
      2. GPIOD_IN：初始化 GPIO 作為輸入
      3. GPIOD_OUT_LOW / GPIOD_OUT_HIGH：將 GPIO 初始化作為輸出，值為 0 / 1
      4. GPIOD_OUT_LOW_OPEN_DRAIN / GPIOD_OUT_HIGH_OPEN_DRAIN：與上列相同，但強制以 Open Drain 的方式使用，用於 I2C 狀態上
    - con_id：
      - ◆ devm_gpiod_get -> _gpio_get_index [gpiolib.c] -> of_find_gpio()，該方法在查詢 Device Tree GPIO 資源已經自動加了 “-gpio”、”-gpios” 字尾，所以在使用時，要對匹配字串進行相應的修改。
例：

        Device Tree：reset-gpios = <&gpio 88 0>;

        Driver：rst_gpio = devm_gpiod_get(&pdev->dev, "reset", GPIOD_OUT_LOW);

Linux 設計 – 修改 Driver 內容

- ARM GPIO 架構：[續]
  - ✓ 讀取 GPIO 數值：
    - ➢ int gpiod_get_value_cansleep(const struct gpio_desc *desc)
  - ✓ 設定 GPIO 數值：
    - ➢ void gpiod_set_value_cansleep(struct gpio_desc *desc, int value)
  - ✓ 設定 GPIO 方向：
    - ➢ int gpiod_direction_input(struct gpio_desc *desc)
    - ➢ int gpiod_direction_output(struct gpio_desc *desc, int value)
  - ✓ 檢查 GPIO 方向：
    - ➢ int gpiod_get_direction(const struct gpio_desc *desc)
    - ➢ 返回值為 GPIOF_DIR_IN 或者 GPIOF_DIR_OUT

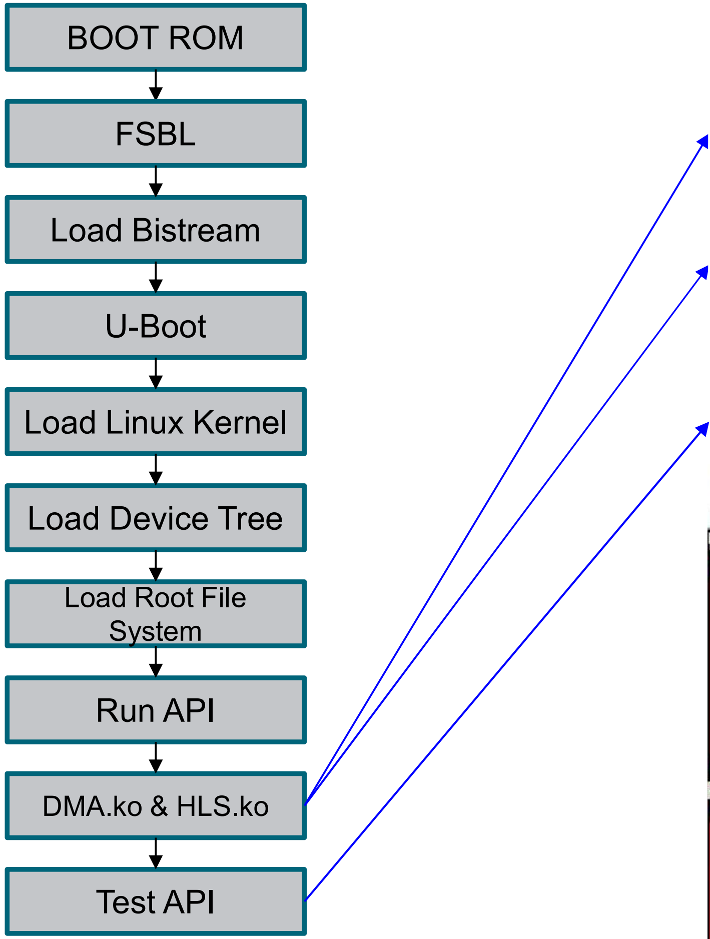
Linux 設計 – 修改 Driver 內容

- HLS IP 驅動：

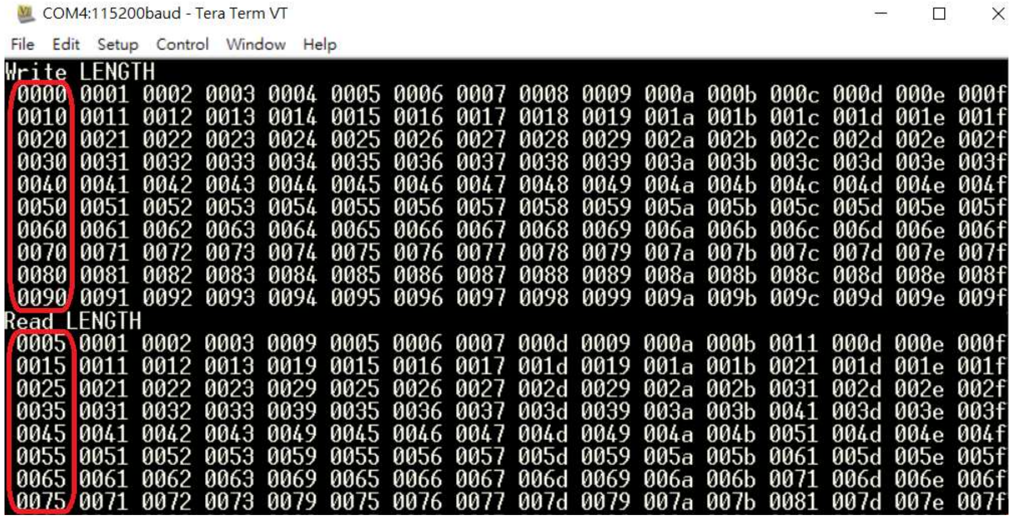
```
                         ⁞
#include <linux/gpio/consumer.h>
                         ⁞
struct gpio_desc *rst_gpio;                  /* Init/Reset HLS IP*/

static int add_probe(struct platform_device *pdev)
{
                         ⁞
        rst_gpio = devm_gpiod_get(&pdev->dev, "reset", GPIOD_OUT_LOW);
        if (IS_ERR(rst_gpio)) {
                if (PTR_ERR(rst_gpio) != -EPROBE_DEFER)
                        dev_err(&pdev->dev, "Reset GPIO not setup in DT");
                return -ENOENT;
        }

        /* Release reset for HLS IP */
        gpiod_set_value_cansleep(rst_gpio, 1);
                         ⁞
}
```

測試 API

測試 API



功能	檔案名稱	編譯指令
DMA 驅動程式	dma.c / dma.h	
	Makefile	make
HLS 驅動程式	hls_add.c / hls_add.h	
	Makefile	make
應用程式	test.c / test.h	aarch64-linux-gnu-gcc test.c –o test


COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Write LENGTH
 0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000a 000b 000c 000d 000e 000f
 0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 001a 001b 001c 001d 001e 001f
 0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002a 002b 002c 002d 002e 002f
 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039 003a 003b 003c 003d 003e 003f
 0040 0041 0042 0043 0044 0045 0046 0047 0048 0049 004a 004b 004c 004d 004e 004f
 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005a 005b 005c 005d 005e 005f
 0060 0061 0062 0063 0064 0065 0066 0067 0068 0069 006a 006b 006c 006d 006e 006f
 0070 0071 0072 0073 0074 0075 0076 0077 0078 0079 007a 007b 007c 007d 007e 007f
 0080 0081 0082 0083 0084 0085 0086 0087 0088 0089 008a 008b 008c 008d 008e 008f
 0090 0091 0092 0093 0094 0095 0096 0097 0098 0099 009a 009b 009c 009d 009e 009f
Read LENGTH
 0005 0001 0002 0003 0009 0005 0006 0007 000d 0009 000a 000b 0011 000d 000e 000f
 0015 0011 0012 0013 0019 0015 0016 0017 001d 0019 001a 001b 0021 001d 001e 001f
 0025 0021 0022 0023 0029 0025 0026 0027 002d 0029 002a 002b 0031 002d 002e 002f
 0035 0031 0032 0033 0039 0035 0036 0037 003d 0039 003a 003b 0041 003d 003e 003f
 0045 0041 0042 0043 0049 0045 0046 0047 004d 0049 004a 004b 0051 004d 004e 004f
 0055 0051 0052 0053 0059 0055 0056 0057 005d 0059 005a 005b 0061 005d 005e 005f
 0065 0061 0062 0063 0069 0065 0066 0067 006d 0069 006a 006b 0071 006d 006e 006f
 0075 0071 0072 0073 0079 0075 0076 0077 007d 0079 007a 007b 0081 007d 007e 007f
```