

Creating Vivado HLS Project

Innova Lee(이상훈)
gcccompil3r@gmail.com

Introduction

이 Lab 에서 Vivado HLS Tool 을 사용하여
High-Level Synthesis 하는 기본 정보를 제공한다.
GUI Mode 에서 Vivado HLS 를 사용하여 Project 를 만든다.
제공된 Design 을 Simulate 하고 Synthesize(합성) 하고 구현한다.

Objectives

이 Lab 을 완료한 후 아래의 것을 할 수 있을 것이다:

- Vivado HLS GUI 를 통해 새로운 Project 생성
- Design 시뮬레이션 하기
- Design 합성하기
- Design 구현하기
- Vivado HLS 의 분석 기능을 사용하여 Design 분석 수행
- Vivado 및 XSim 시뮬레이터를 사용하여 시뮬레이터 출력 분석

Procedure

이 Lab 은 세부적인 지침에 대한 정보를
일반적인 소개부터 단계적으로 제공하도록 구성되어있다.
Lab 을 진행하려면 세부 지침을 따르라.

이 Lab 은 8 가지 기본 단계로 구성된다:
Vivado HLS 에서 새로운 Project 를 만들고,
시뮬레이션을 실행하고 Debug 를 실행하고 Design 을 합성하고,
분석 perspective 를 열고, RTL Co-Simulation 을 실행하고,
Vivado 및 XSim 을 사용하여 Simulation 결과를 보고,
Vivado HLS 에서 Design 을 내보내고 구현한다.

General Flow for this Lab

- 1 단계: 새로운 프로젝트 생성하기
- 2 단계: C 시뮬레이션 구동하기
- 3 단계: Debugger 구동하기
- 4 단계: Design 합성하기
- 5 단계: Analysis Perspective 를 사용하여 분석하기
- 6 단계: C/RTL Co-Simulation 구동하기
- 7 단계: Vivado 내에 Simulation 결과 살펴보기
- 8 단계: RTL 을 내보내고 구현하기

Create a New Project

1-1. Zynq xc7z020clg484-1(ZedBoard) 혹은 xc7z010clg400-1(Zybo) 를 Targeting 하여 Vivado HLS 에서 새로운 Project 생성하기

1-1-1. Vivado HLS 를 띄운다:

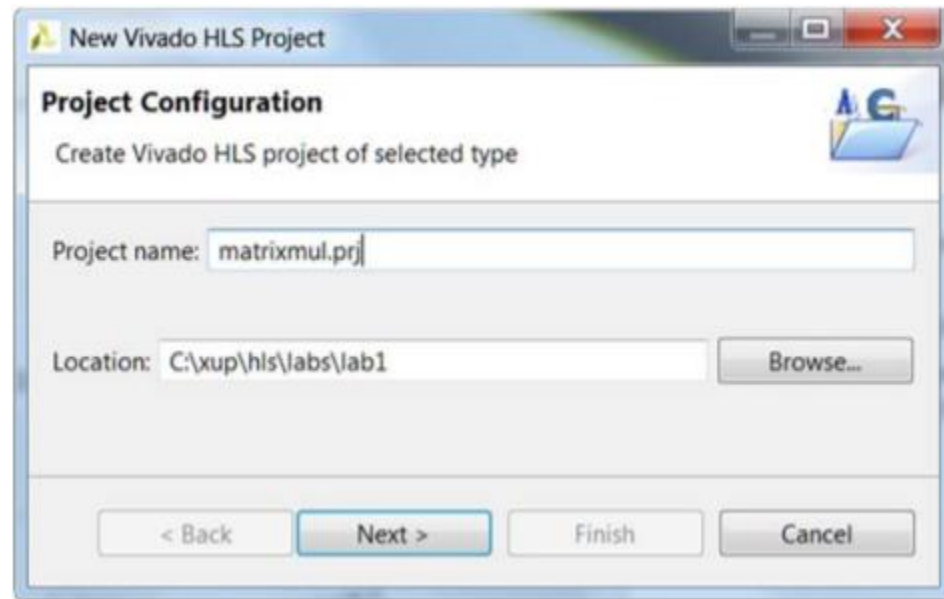
/opt/Xilinx/Vivado_HLS/2017.1/bin/vivado_hls GUI 창이 나타나는 것을 볼 수 있다.



1-1-2. GUI 창에서 Create New Project 를 누른다.
New Vivado HLS Project 창이 열린다.

1-1-3. Location 필드에 있는 Browse... 버튼을 누르고
lab1 코드를 찾아서 OK 를 누른다.

1-1-4. Project 이름을, matrixmul.prj 로 입력한다.



1-1-5. Next 를 누른다.

1-1-6. Add/Remove Files 창에서 Top Function 이름으로 matrixmul 을 입력
(matrixmul 이라는 합성될 제공된 소스 코드가 포함하는 함수)

1-1-7. Add Files... 버튼을 누르고
lab1 폴더에서 matrixmul.cpp 파일을 선택하고 Open 을 누른다.

1-1-8. Next 를 누른다.

1-1-9. testbench 를 위해 Add/Remove Files 에서 Add Files... 버튼을 누르고
lab1 폴더에서 matrixmul_test.cpp 파일을 선택하고 Open 을 누른다.

1-1-10. Files List 창에서 matrixmul_test.cpp 를 선택하고
Edit CFLAG... 버튼을 누르고 -DHW_COSIM 을 입력하고 OK 를 누른다.
(나중에 사용될 custom flag 를 정의함)

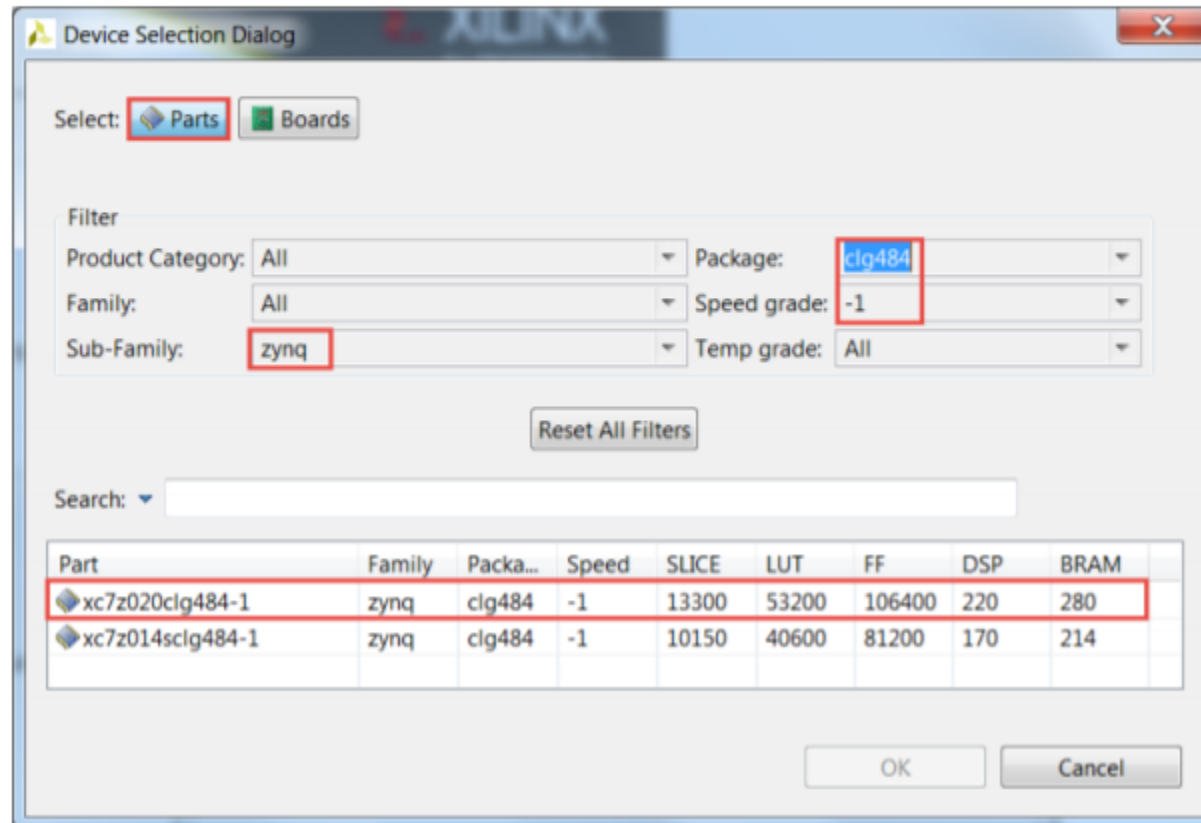
1-1-11. Next 를 누른다.

1-1-12. Solution Configuration 페이지에서
Solution Name 필드를 solution1 로 하고
clock period 를 10(ZedBoard 를 위해) 혹은 8(Zybo 를 위해) 으로 설정함.
불확실한 필드를 비워두면 ZedBoard 의 기본값으로 1.25,
Zybo 의 기본값으로 1.25 가 지정될 것이다.

Part Selection 절에서 ... 버튼을 클릭한다.

1-1-13. Device Selection Dialog 페이지에서, Parts Specify 필드를 선택하고, Filter 부분에 Package 에 clg400, Speed grade 에 -1, 그리고 Sub-Family 에 zynq 를 넣은 후에 xc7z020clg484-1(ZedBoard) 혹은 xc7z010clg400-1(Zybo) 파트를 선택하고 OK 를 누른다: (clg400(Zybo), clg484(ZedBoard))

- Family: **Zynq**
- Sub-Family: **Zynq**
- Package: **clg484** (for ZedBoard) or **clg400** (for Zybo)
- Speed Grade: **-1**



여기서 xc7z010clg400-1 을 선택하도록 한다.
SLICE 4400, LUT 17600, FF 35200, DSP 80, BRAM 120 에 해당한다.

Device Selection Dialog

Select:

Parts

Boards

Filter

Product Category: All

Package: clg400

Family: All

Speed grade: -1

Sub-Family: zynq

Temp grade: All

Reset All Filters

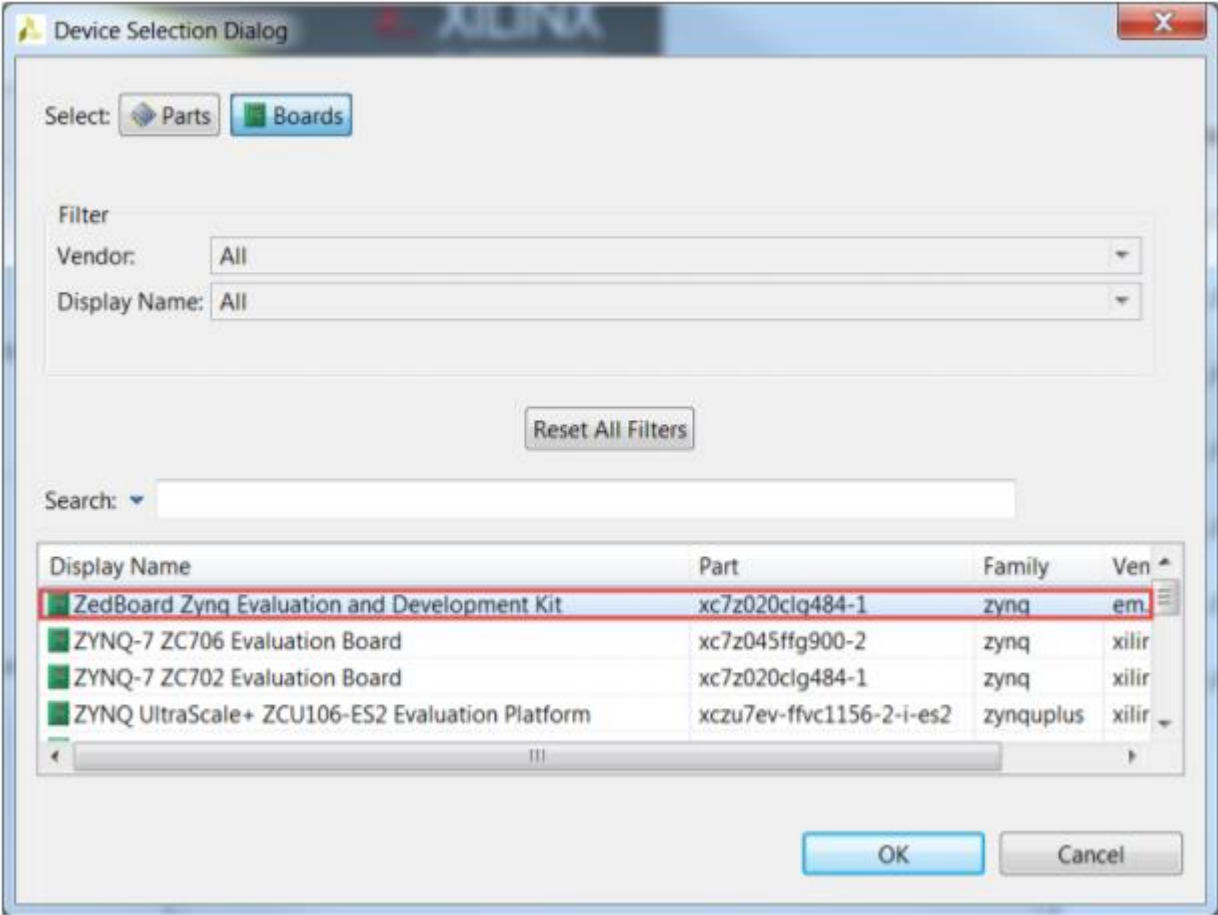
Search:

Part	Family	Packa...	Speed	SLICE	LUT	FF	DSP	BRAM
xc7z020clg400-1	zynq	clg400	-1	13300	53200	106400	220	280
xc7z014sclg400-1	zynq	clg400	-1	10150	40600	81200	170	214
xc7z010clg400-1	zynq	clg400	-1	4400	17600	35200	80	120
xc7z007sclg400-1	zynq	clg400	-1	3600	14400	28800	66	100

OK

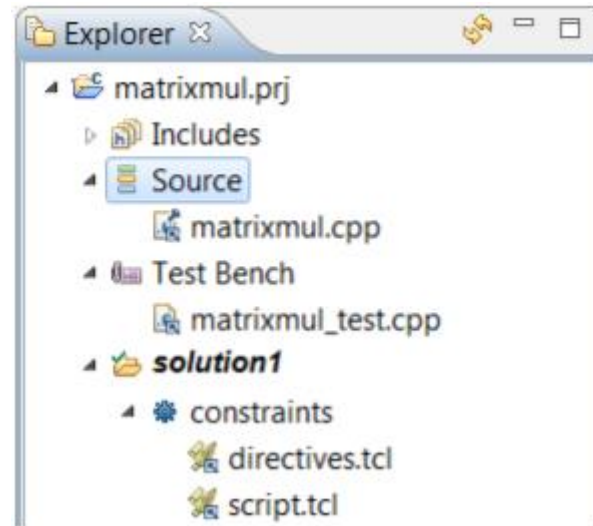
Cancel

Board 지정 옵션(ZedBoard 에만 해당)을 선택하고 원하는 Target Board 가 나열된 경우 나열된 Board 중 하나를 선택할 수도 있다.



1-1-14. Finish 를 클릭한다.

Explorer View 에서 생성된 프로젝트를 볼 수 있다.
다양한 하위 폴더를 확장하여 각 하위 폴더 아래의 항목을 확인한다.



1-1-15. 정보 창에 소스 폴더의 내용을 열기 위해 matrixmul.cpp 를 더블클릭한다.

```
67 #include "matrixmul.h"
68
69 void matrixmul(
70     mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
71     mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
72     result_t res[MAT_A_ROWS][MAT_B_COLS])
73 {
74     // Iterate over the rows of the A matrix
75     Row: for(int i = 0; i < MAT_A_ROWS; i++) {
76         // Iterate over the columns of the B matrix
77         Col: for(int j = 0; j < MAT_B_COLS; j++) {
78             // Do the inner product of a row of A and col of B
79             res[i][j] = 0;
80             Product: for(int k = 0; k < MAT_B_ROWS; k++) {
81                 res[i][j] += a[i][k] * b[k][j];
82             }
83         }
84     }
85 }
```

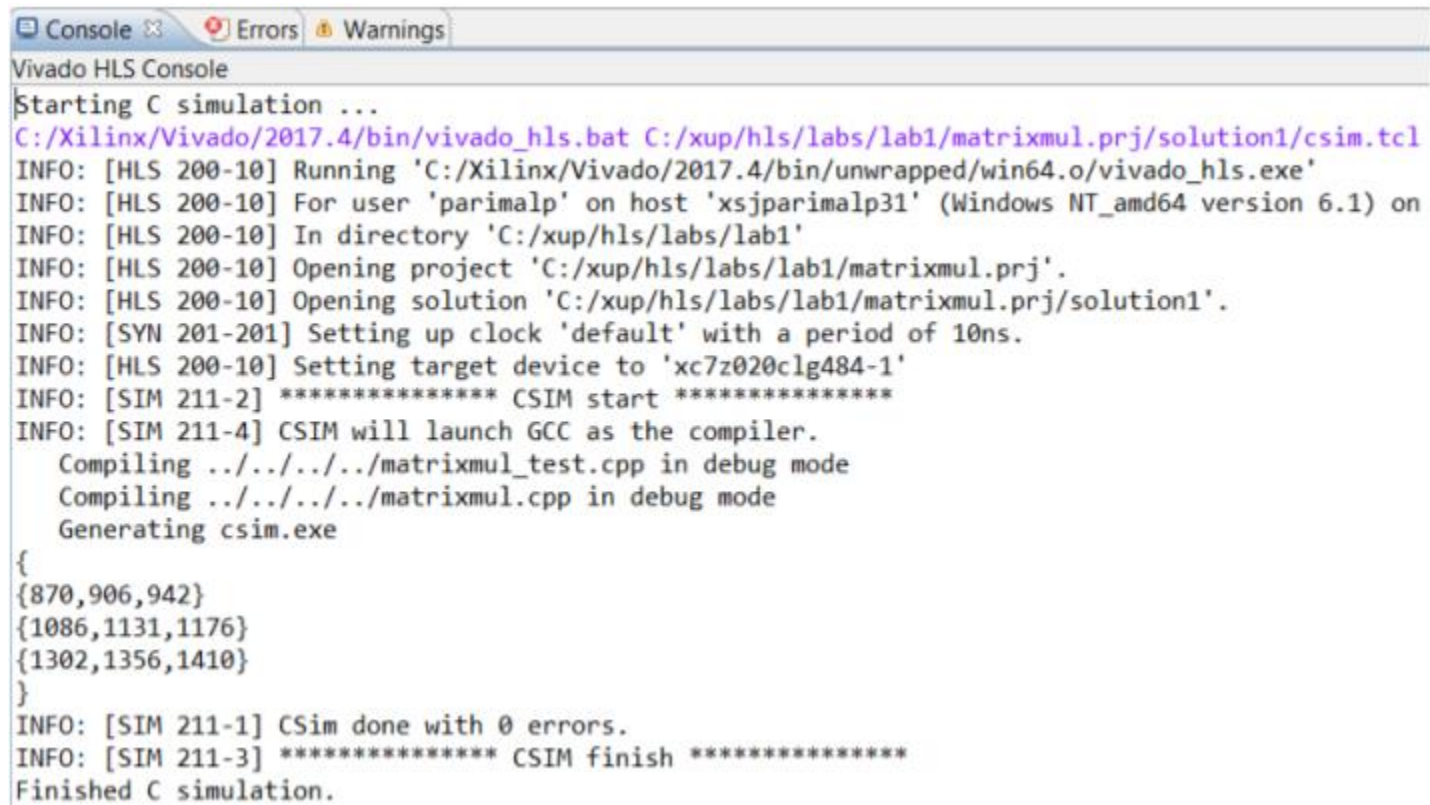
이 설계는 3 개의 중첩된 Loop 로 구성된 행렬 곱셈 구현임을 알 수 있다.
Product 루프는 실제 Matrix 요소 곱과 합계를 수행하는 가장 내부 Loop 이다.
Col Loop 는 전달된 행 요소 데이터가 있는
다음 열 요소 데이터를 Product Loop 에 공급하는 외부 Loop 다.
마지막 Row 는 가장 바깥 쪽 Loop 다.
res[i][j] = 0(79 번 라인)은 새로운 행 요소가 전달되고
새로운 열 요소가 사용될 때마다 결과를 다시 설정한다.

Run C Simulation

2-1. Run C simulation to view the expected output

2-1-1. Project -> Run C Simulation 혹은 tool bar 버튼에 있는 플레이 버튼을 누르고 C Simulation Dialog 창에서 OK 를 누른다.

2-1-2. 파일이 컴파일 될 것이고 콘솔 창에서 출력을 볼 수 있을 것이다.



```
Console x Errors Warnings
Vivado HLS Console
Starting C simulation ...
C:/Xilinx/Vivado/2017.4/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmul.prj/solution1/csim.tcl
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2017.4/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'parimalp' on host 'xsjparimalp31' (Windows NT_amd64 version 6.1) on
INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab1'
INFO: [HLS 200-10] Opening project 'C:/xup/hls/labs/lab1/matrixmul.prj'.
INFO: [HLS 200-10] Opening solution 'C:/xup/hls/labs/lab1/matrixmul.prj/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Setting target device to 'xc7z020clg484-1'
INFO: [SIM 211-2] ***** CSIM start *****
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
  Compiling ../../../../matrixmul_test.cpp in debug mode
  Compiling ../../../../matrixmul.cpp in debug mode
  Generating csim.exe
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
Finished C simulation.
```

2-1-3. 내용을 보기 위해 Explorer 에서 testbench 폴더 하위의 matrixmul_test.cpp 를 더블 클릭한다.

일부 값으로 초기화된 2 개의 입력 행렬과
알고리즘을 실행하는 코드가 표시되어야 한다.
HW_COSIM 이 정의된 경우(프로젝트 설정 중에 수행됨)
matrixmul 함수가 호출되고 계산된 결과의 출력을 호출된 함수에서
반환된 출력과 비교하여 결과가 일치하면 Test 를 출력한다.

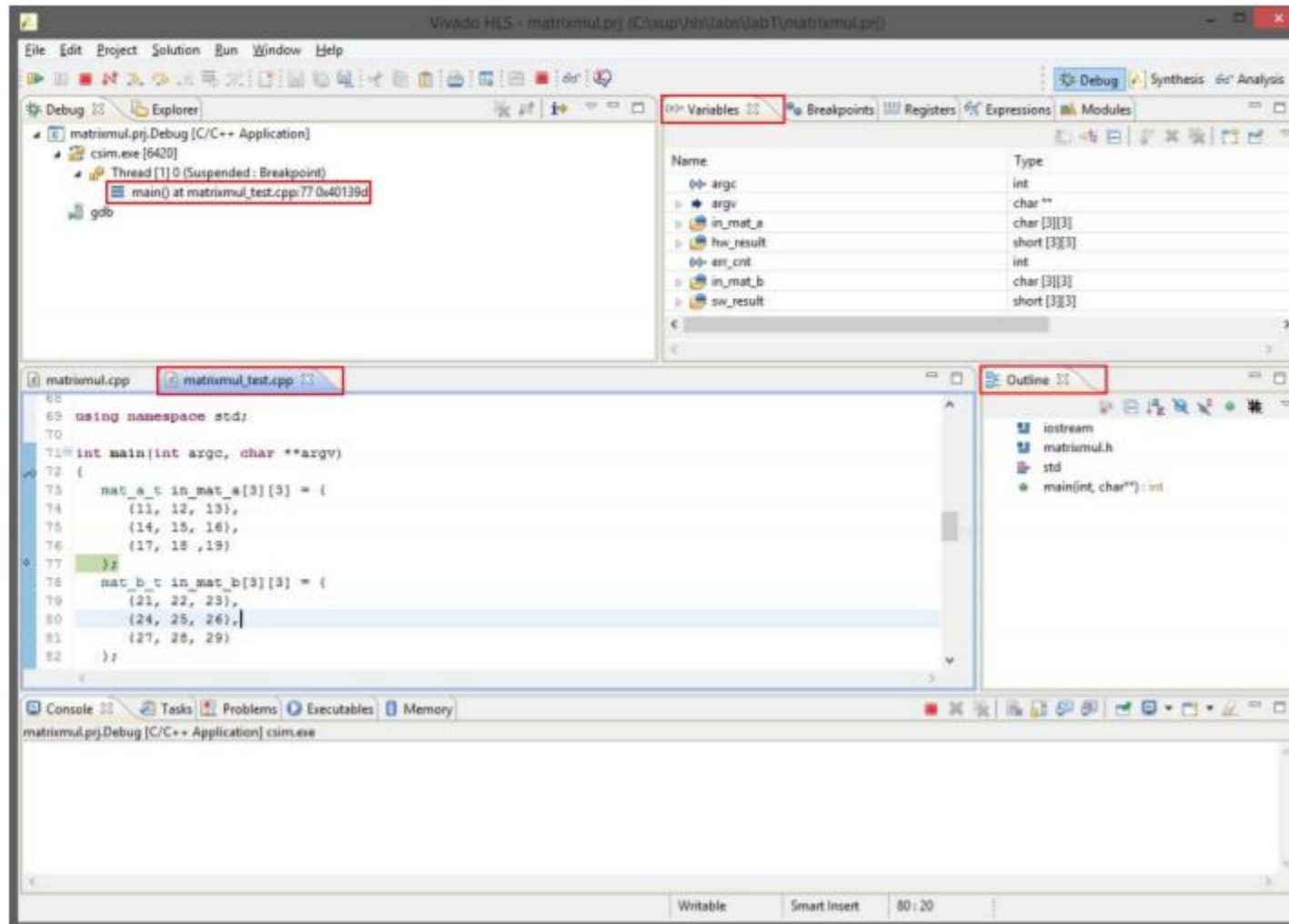
HW_COSIM 이 정의되지 않았으면
단순히 계산된 결과를 출력하고 matrixmul 함수를 호출하지 않을 것이다.

Run Debugger

3-1. Debugger Mode 에서 Application 을 구동하고 Program 의 동작 이해하기

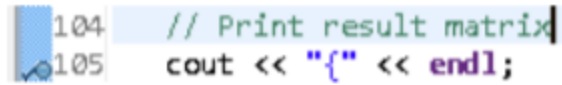
3-1-1. Project -> Run C Simulation 을 선택하거나
tools bar 버튼에서 플레이 버튼을 누른다.
Launch Debugger 옵션을 선택하고 OK 를 누른다.

3-1-2. Debug Perspective 는 Source View 에서 matrixmul_test.cpp 를 보여줄 것이고, Variables view 에서는 argc 와 argv 변수들이 보여질 것이며, Outline View 는 현재 Scope 에 있는 Objects 를 보여주고, Thread 가 생성되었으며 Program 이 main() 함수 entry point 에서 일시 정지되었음을 볼 수 있을 것이다.



3-1-3. Source View 에서 Scroll Down 하여 Output Console 창에서 output "{" 이 있는 line 105 에서 파란 여백을 더블클릭한다.
이것은 line 105 에서 break-point 를 걸 것이다.

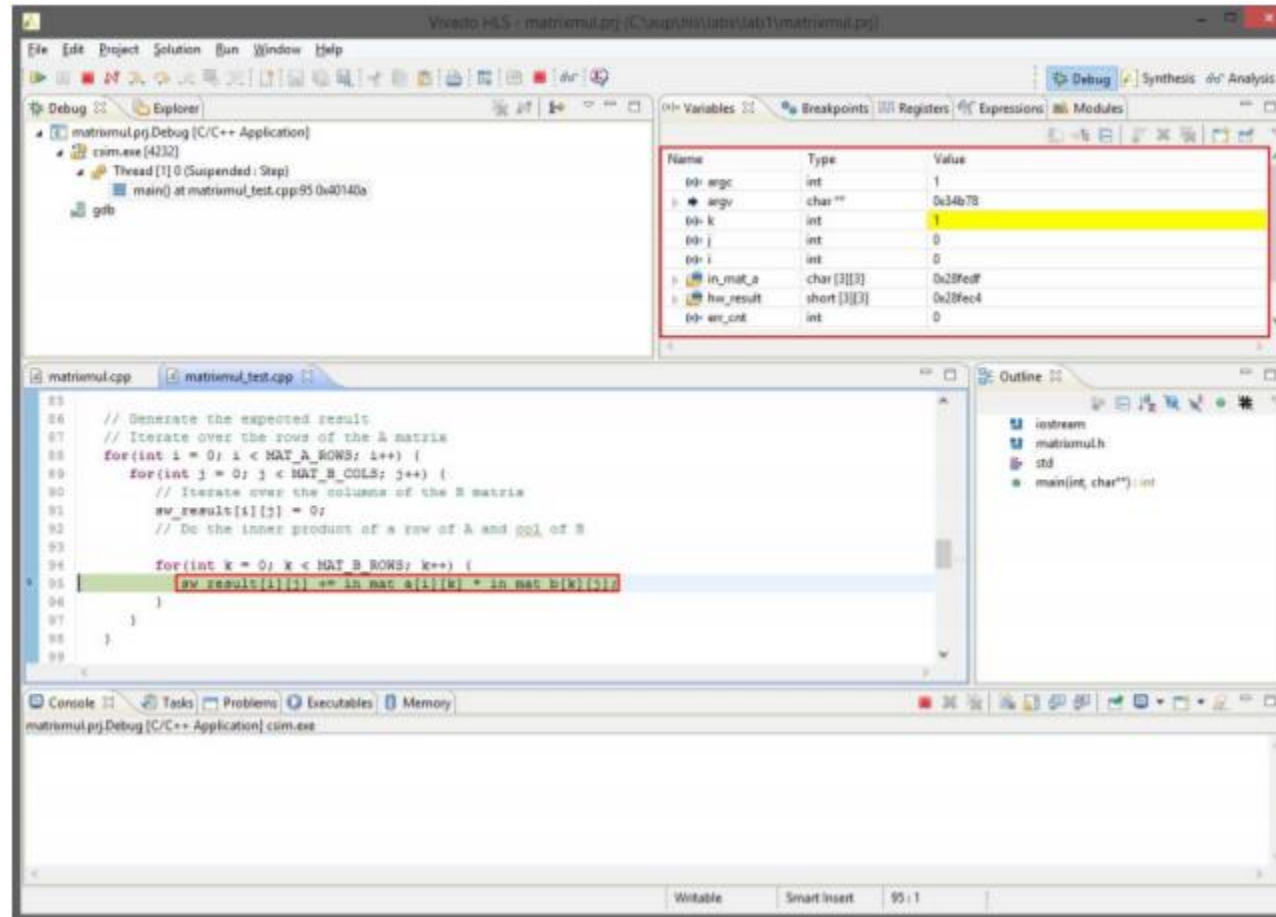
breakpoint 는 파란색 원과 tick 으로 마킹된다.

A screenshot of a code editor with a light blue background. Line 104 is highlighted in light blue and contains the comment `// Print result matrix`. Line 105 contains the code `cout << "{" << endl;`. A blue vertical bar is positioned to the left of line 105, and a small blue circle with a white tick inside is placed on this bar, indicating a breakpoint. The line numbers 104 and 105 are visible in the left margin.

```
104 // Print result matrix
105 cout << "{" << endl;
```

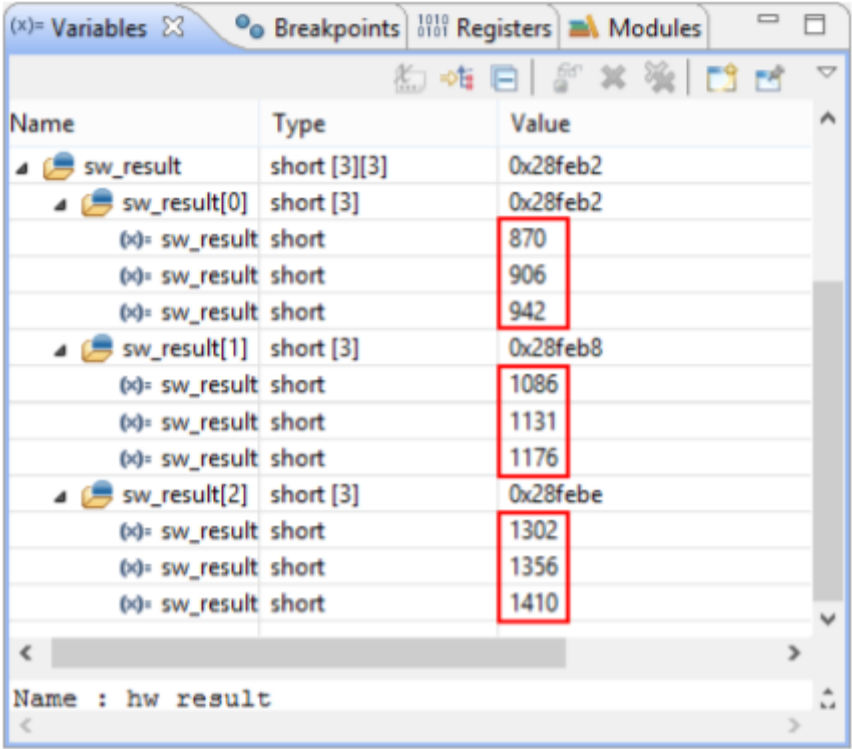
3-1-4. 유사하게 matrixmul() 함수상의 line 101 에 breakpoint 를 설정한다.

3-1-5. 간간히 Step Over(F6) 버튼을 사용하여 실행 절차를 관찰하고 변수의 갱신을 관찰하여 SW 결과가 계산되는 것을 살펴볼 수 있다.



3-1-6. Resume(플레이) 버튼을 누르거나 F8 을 눌러서 모든 계산을 완료하고 line 101 에서 정지한다.

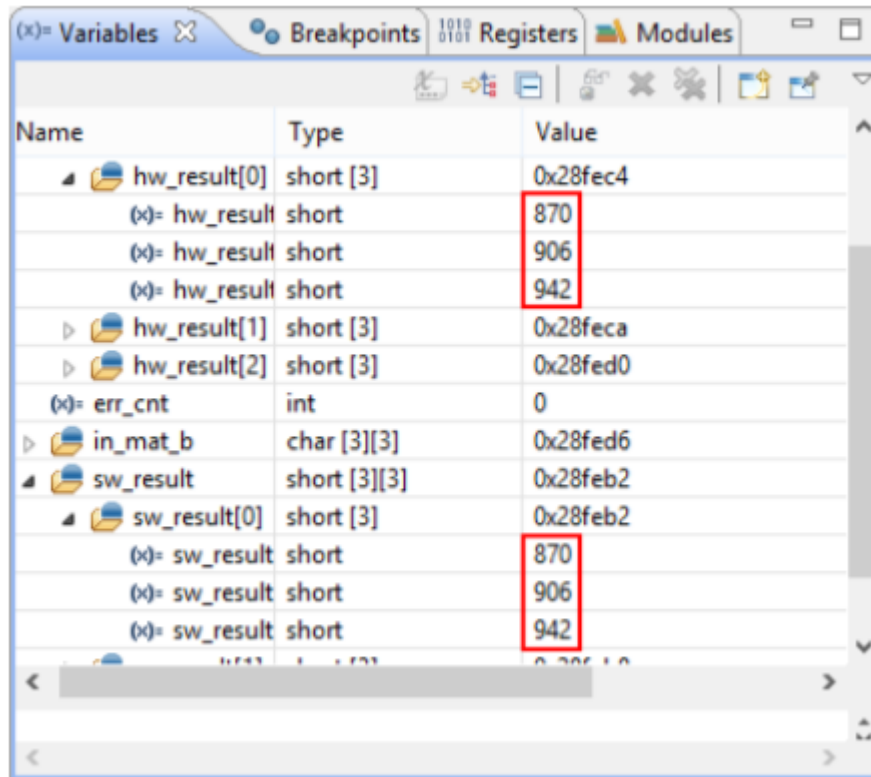
3-1-7. Variables View 에서 계산된 SW 결과를 관측해보자



Name	Type	Value
sw_result	short [3][3]	0x28feb2
sw_result[0]	short [3]	0x28feb2
(x)= sw_result	short	870
(x)= sw_result	short	906
(x)= sw_result	short	942
sw_result[1]	short [3]	0x28feb8
(x)= sw_result	short	1086
(x)= sw_result	short	1131
(x)= sw_result	short	1176
sw_result[2]	short [3]	0x28febe
(x)= sw_result	short	1302
(x)= sw_result	short	1356
(x)= sw_result	short	1410

Name : hw result

- 3-1-8. Step Into(F5) 버튼을 클릭하여 합성할 matrixmul Module 을 탐색하고 Module 의 line 75 에서 실행이 일시 중지되었는지 확인한다.
- 3-1-9. Step Over(F6) 를 사용하여 계산된 결과를 관측한다.
일단 만족되면 Step Return(F7) 버튼을 사용하여 함수에서 돌아올 수 있다.
- 3-1-10. breakpoint 를 설정했으므로 Program 실행은 line 105 에서 일시 중단된다.
Variables View 에서 계산된 SW 및 HW(기능) 결과를 관측하라.



Name	Type	Value
hw_result[0]	short [3]	0x28fec4
(x)= hw_result	short	870
(x)= hw_result	short	906
(x)= hw_result	short	942
hw_result[1]	short [3]	0x28feca
hw_result[2]	short [3]	0x28fed0
err_cnt	int	0
in_mat_b	char [3][3]	0x28fed6
sw_result	short [3][3]	0x28feb2
sw_result[0]	short [3]	0x28feb2
(x)= sw_result	short	870
(x)= sw_result	short	906
(x)= sw_result	short	942

- 3-1-11. line 134(return err_cnt;) 에 breakpoint 를 설정하고 Resume 을 누른다.
breakpoint 를 만날 때까지 실행을 계속한다.
Console 창은 앞서 살펴본 결과를 보여준다.
- 3-1-12. Resume 버튼을 누르거나 Terminate 버튼을 눌러 Debugging Session 을 마무리한다.

Synthesize the Design

4-1. Synthesis View 로 전환하고 기본값을 사용하여 Design 을 합성하라.
합성 결과를 보고 이 단계의 세부 질문에 나열된 질문에 답해보자 ~

4-1-1. tools bar 에 있는 Synthesis 를 눌러 Synthesis View 로 전환한다.

4-1-2. Solution -> Run C Synthesis -> Active Solution 을 선택하거나 플레이 버튼을 눌러 synthesis 절차를 시작한다.

4-1-3. synthesis 가 완료되면 합성 결과가 Outline Pane 에 출력될 것이다.
Outline Pane 을 사용하여 간단하게 보고서의 모든 부분을 탐색 할 수 있다.

Synthesis Report for 'matrixmul'

General Information

Date: Tue Feb 13 07:51:22 2018
Version: 2017.4 (Build 2086221 on Fri Dec 15 21:13:33 MST 2017)
Project: matrixmul.prj
Solution: solution1
Product family: zynq
Target device: xc7z020clg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.70	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
79	79	79	79	none

Synthesis Report for 'matrixmul'

General Information

Date: Sat Feb 24 07:33:18 2018
Version: 2017.4 (Build 2086221 on Fri Dec 15 21:13:33 MST 2017)
Project: matrixmul.prj
Solution: solution1
Product family: zynq
Target device: xc7z010clg400-1

Performance Estimates

Timing (ns)

Summary

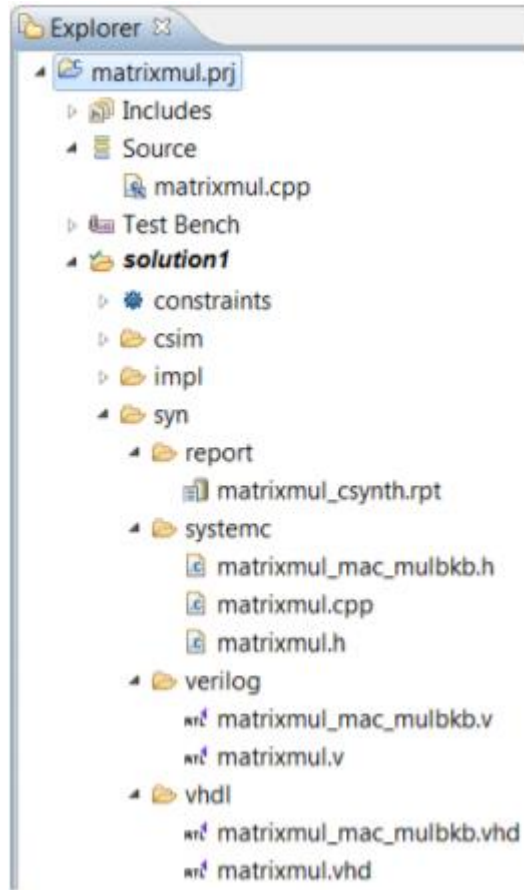
Clock	Target	Estimated	Uncertainty
ap_clk	8.00	6.38	1.00

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
106	106	106	106	none

4-1-4. Explorer 에서 solution1 을 확장하면 보고서 파일을 포함하여 생성된 여러 파일에 접근할 수 있게 된다.



Solution1 폴더의 syn 폴더가 Explorer View 에서 펼쳐지면 report 파일, 생성된 소스(VHDL, Verilog, 헤더 및 cpp) 파일 아래에 report, SystemC, Verilog 및 VHDL 하위 폴더가 표시된다. 이들 항목 중 하나를 더블 클릭하면 Information 창에 파일이 열린다.

또한 Target Design 에 계층 구조 함수가 있는 경우 하위 수준 함수에 해당하는 보고서도 만들어진다.

4-1-5. 합성 보고서는 설계의 예상 대기 시간뿐만 아니라 성능 및 자원 추정도 표시한다.

4-1-6. 오른쪽에 Scroll 막대를 사용하여 아래로 Scroll 하여 아래 질문에 답하라.

Estimated Clock Period:

Worst case Latency:

Number of DSP48E Used:

Number of FFs Used:

Number of LUTs Used:

4-1-7. 보고서는 또한 Tool 에 의해 생성된 최상위 인터페이스 신호를 보여준다.

Interface					
Summary					
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	matrixmul	return value
ap_rst	in	1	ap_ctrl_hs	matrixmul	return value
ap_start	in	1	ap_ctrl_hs	matrixmul	return value
ap_done	out	1	ap_ctrl_hs	matrixmul	return value
ap_idle	out	1	ap_ctrl_hs	matrixmul	return value
ap_ready	out	1	ap_ctrl_hs	matrixmul	return value
a_address0	out	4	ap_memory	a	array
a_ce0	out	1	ap_memory	a	array
a_q0	in	8	ap_memory	a	array
b_address0	out	4	ap_memory	b	array
b_ce0	out	1	ap_memory	b	array
b_q0	in	8	ap_memory	b	array
res_address0	out	4	ap_memory	res	array
res_ce0	out	1	ap_memory	res	array
res_we0	out	1	ap_memory	res	array
res_d0	out	16	ap_memory	res	array

ap_clk, ap_rst 및 ap_idle 과 ap_ready 제어 신호가 기본적으로 Design 에 자동으로 추가된다.
이 신호는 Handshaking 신호로 사용되어 Design 이 다음 계산 명령(ap_ready) 를 시작할 준비가 되었는지, 다음 계산이 시작되는지(ap_start), 계산이 완료될 때(ap_done) 표시된다.
다른 신호는 설계 및 기본 또는 지정된 Interface 의 입력 및 출력 신호를 기반으로 생성된다.

Analyze using Analysis Perspective

5-1. Analysis Perspective 로 전환하고 Design 동작을 이해하자

5-1-1. Solution -> Open Analysis Perspective 를 클릭하거나 Debug 창 옆의 Synthesis 옆의 안경 모양 Analysis 를 눌러서 Analysis Viewer 를 열도록 한다.

Analysis Perspective 는 아래와 같이 5 개의 창으로 구성된다.
Module 및 Loops 계층 구조는 기본적으로 확장되지 않은 상태로 표시된다.

Module Hierarchy Pane 은 전체 Design 에 대한 성능 및 Area 정보를 보여주며 계층 구조를 탐색하는데 사용할 수 있다.
Performance Profile Pane 이 표시되며 이 계층 구조에 대한 성능 정보를 보여준다.
이 2 개의 Pane 에 있는 정보는 Synthesis 보고서의 앞부분에서 검토한 정보와 유사하다.

Performance View 는 오른쪽 창에 표시된다.
이 View 는 특정 Block 에서의 연산이 Clock Cycle 에 어떻게 스케줄링 되는지를 보여준다.

* 왼쪽 열에는 자원 목록이 있다.

* 맨 윗줄에는 Design 의 제어 상태(c0 ~ c5) 가 나열된다.
제어 상태는 연산을 Clock Cycles 로 스케줄하기 위해 High-Level Synthesis 에서 사용되는 내부 상태다.
제어 상태와 RTL Finite State Machine(FSM) 의 최종 상태 간에 밀접한 상관 관계가 있지만 일대일 맵핑은 없다.

Vivado HLS 2017.4 - matrixmul.prj (C:\xup\hls\labs\lab1\matrixmul.prj)

File Edit Project Solution Window Help

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
matrixmul	0	1	44	184	79	80	none

Performance Profile Resource Profile

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
matrixmul	-	79	80	-	-
Row	no	78	-	26	3

matrixmul_test Synthesis(solut Performance(sol

Current Module : matrixmul

Operation\Control Step	C0	C1	C2	C3	C4
1-...Row					

Performance Resource

Properties Warnings

Property	Value
----------	-------

5-1-2. Loop 행의 '+' 를 클릭하여 확장한 다음
하위 루프 Col 및 Product 를 클릭하여 Loop 계층을 완전히 확장하라.

Current Module : matrixmul

	Operation\Control Step	C0	C1	C2	C3	C4
1	⊟Row					
2	i(phi mux)					
3	exitcond2(icmp)					
4	i 1(+)					
5	tmp s(-)					
6	⊟Col					
7	j(phi mux)					
8	exitcond1(icmp)					
9	j 1(+)					
10	tmp 2(+)					
11	⊟Product					
12	res load(phi mux)					
13	k(phi mux)					
14	node 40(write)					
15	exitcond(icmp)					
16	k 1(+)					
17	tmp 4(+)					
18	tmp 11(-)					
19	tmp 12(+)					
20	a load(read)					
21	b load(read)					
22	tmp 7(*)					
23	tmp 8(+)					

이를 통해 Row 의 첫 번째 상태(C1)에서 Loop 종료 조건이 확인되고 수행된 추가 작업이 있음을 알 수 있다.
이 추가 기능은 반복 회수를 세는 카운터일 가능성이 높으며 이를 확인할 수 있다.

Loop 에서 생성된 연산은 노란색으로 표시되고 표준 연산은 자주색으로 표시되며 하위 블록은 초록색으로 표시된다(이 경우 하위 수준 함수는 없다)

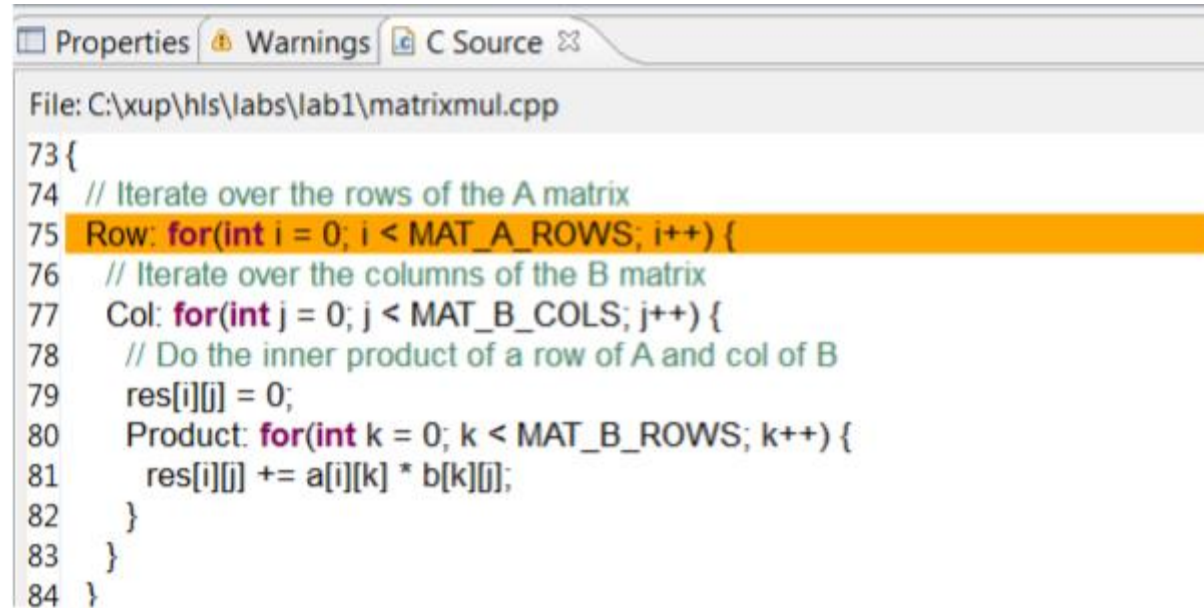
5-1-3. 상태 C1 의 가산기에 자주색 블록을 선택하고 마우스 오른쪽 버튼으로 클릭한 다음 Goto Source 를 선택한다.

Source Code Pane 이 열릴 것이며 Row Loop 인덱스가 테스트되고 증가되는 line 75 가 강조 표시된다.
다음 상태(C2)에서 Col Loop 를 실행하기 시작한다.

Current Module : matrixmul

	Operation\Control Step	C0	C1	C2	C3	C4
1	⊟Row					
2	i(phi mux)					
3	exitcond2(icmp)					
4	i 1(+)					
5	tmp s(-)					
6	⊟Col					
7	j(phi mux)					
8	exitcond1(icmp)					
9	j 1(+)					
10	tmp 2(+)					
11	⊟Product					
12	res load(phi mux)					
13	k(phi mux)					
14	node 40(write)					
15	exitcond(icmp)					
16	k 1(+)					
17	tmp 4(+)					
18	tmp 11(-)					
19	tmp 12(+)					
20	a load(read)					
21	b load(read)					
22	tmp 7(*)					
23	tmp 8(+)					

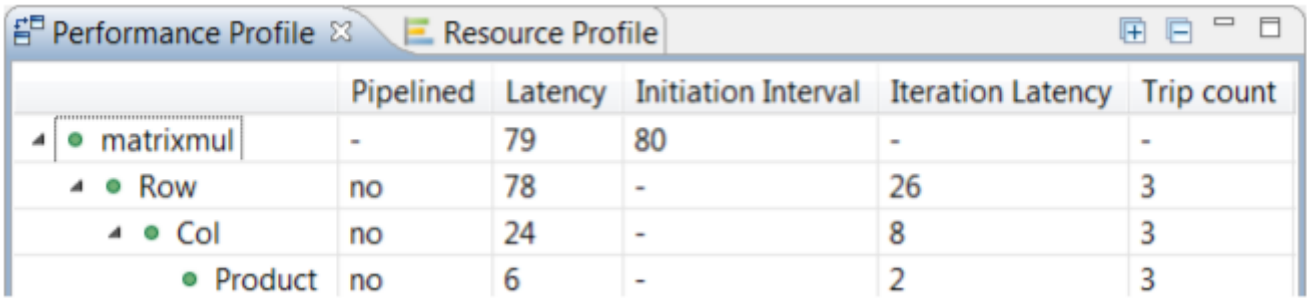
Performance Resource



```
Properties Warnings C Source
File: C:\xup\hls\labs\lab1\matrixmul.cpp
73 {
74 // Iterate over the rows of the A matrix
75 Row: for(int i = 0; i < MAT_A_ROWS; i++) {
76 // Iterate over the columns of the B matrix
77 Col: for(int j = 0; j < MAT_B_COLS; j++) {
78 // Do the inner product of a row of A and col of B
79 res[i][j] = 0;
80 Product: for(int k = 0; k < MAT_B_ROWS; k++) {
81 res[i][j] += a[i][k] * b[k][j];
82 }
83 }
84 }
```

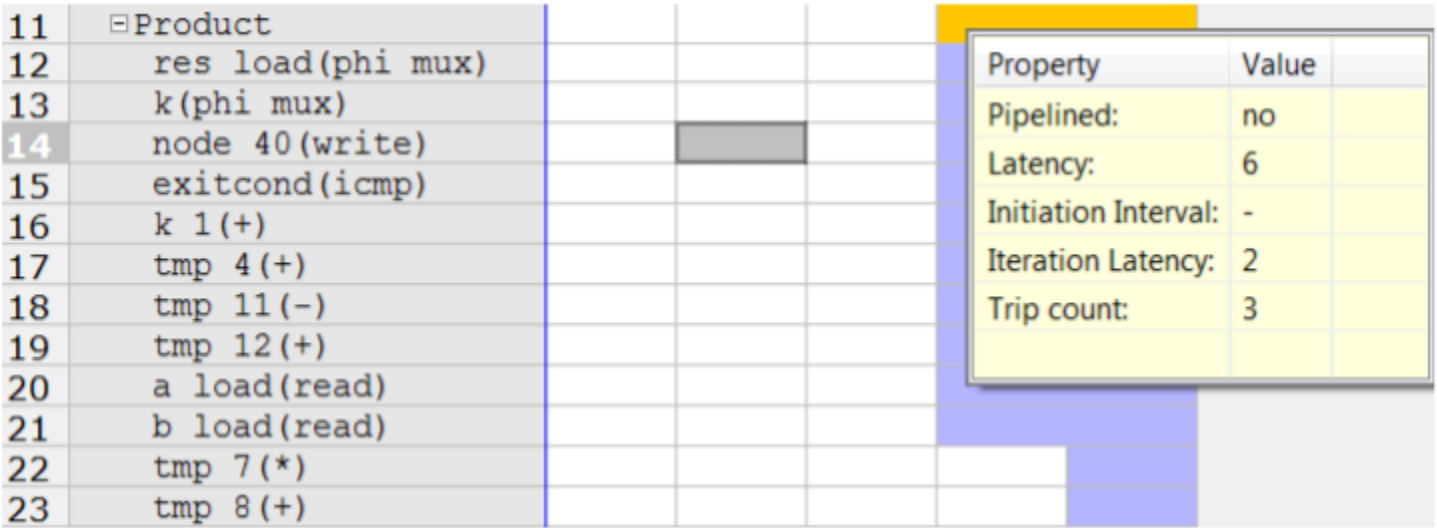
5-1-4. Col Loop 에서 C2-8 자주색 셀을 클릭하면 소스 코드 강조 표시(line 79) 업데이트를 볼 수 있다.

5-1-5. Performance Profile Hierarchy 를 확장하고 중첩 Loop 각각에 대한 반복 대기 시간, 반복 횟수 및 전반적인 대기 시간을 기록한다.



	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
matrixmul	-	79	80	-	-
Row	no	78	-	26	3
Col	no	24	-	8	3
Product	no	6	-	2	3

반복 횟수는 Performance View 위로 마우스를 가져가면 나타난다(Dialog Box 에 Loop 통계가 표시됨)

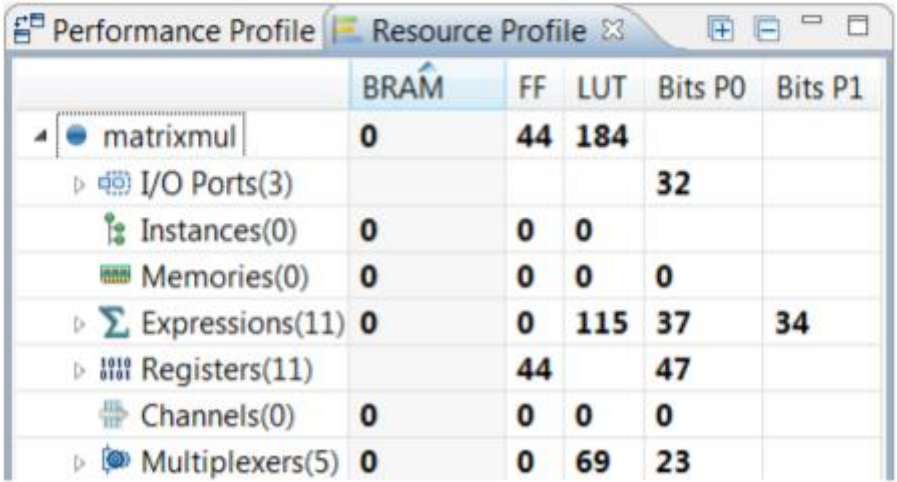


11	Product			
12	res load(phi mux)			
13	k(phi mux)			
14	node 40(write)			
15	exitcond icmp)			
16	k 1(+)			
17	tmp 4(+)			
18	tmp 11(-)			
19	tmp 12(+)			
20	a load(read)			
21	b load(read)			
22	tmp 7(*)			
23	tmp 8(+)			

Property	Value
Pipelined:	no
Latency:	6
Initiation Interval:	-
Iteration Latency:	2
Trip count:	3

이 루프는 파이프라인 되지 않으므로 초기 interval 에는 숫자가 없다.

- 5-1-6. Module Hierarchy 의 matrixmul 항목에 대해 next 를 누르고
Design 에 하위 수준 함수가 정의되어 있지 않으므로 항목이 확장되지 않았음을 확인하라.
- 5-1-7. Resource Profile 탭을 선택하고 다양한 자원과 사용된 위치를 관찰하라.
Expressions and Registers 섹션을 확장하여 자원이 어떤 조작에 사용되는지 확인할 수 있다.



	BRAM	FF	LUT	Bits P0	Bits P1
matrixmul	0	44	184		
I/O Ports(3)				32	
Instances(0)	0	0	0		
Memories(0)	0	0	0	0	
Expressions(11)	0	0	115	37	34
Registers(11)		44		47	
Channels(0)	0	0	0	0	
Multiplexers(5)	0	0	69	23	

- 5-1-8. Performance Matrix Tab 에서 Resource Tab(페이지 하단에 있음)을 선택하고
I/O Port 및 Memory Port 항목을 확장하여 작업 유형, 사용된 리소스 및 사용중인 상태를 확인한다.

Current Module : matrixmul

	Resource\Control Step	C0	C1	C2	C3	C4
1	I/O Ports					
2	a(p0)				read	
3	b(p0)				read	
4	res(p0)				write	
5	Memory Ports					
6	res(p0)				write	
7	a(p0)				read	
8	b(p0)				read	
9	Expressions					
10	i phi fu 79		phi_mux			
11	i 1 fu 127		+			
12	tmp s fu 149		-			
13	exitcond2 fu 121		icmp			
14	j phi fu 90			phi_mux		
15	tmp 2 fu 171			+		
16	j 1 fu 161			+		
17	exitcond1 fu 155			icmp		
18	res load phi fu...				phi_mux	
19	k phi fu 114				phi_mux	
20	k 1 fu 187				+	
21	tmp 12 fu 225				+	
22	tmp 4 fu 197				+	
23	tmp 11 fu 219				-	
24	exitcond fu 181				icmp	
25	grp fu 243					+

5-1-9. Synthesis Tool Bar 버튼을 클릭하여 Synthesis View 로 다시 전환한다.

Run C/RTL Co-Simulation


6-1. VHDL 의 기본 설정으로 C/RTL Co-Simulation 을 실행하라.
시뮬레이션이 통과하는지 확인하라.

6-1-1. Solution -> Run C/RTL Cosimulation 을 선택하거나
Synthesis View 인 경우 Toolbar 버튼을 클릭하여
Dialog Box 를 열면 원하는 Simulation 을 선택하고 실행할 수 있다.

C/RTL Co-Simulation Dialog Box 가 열릴 것이다.

6-1-2. VHDL 옵션이 선택된것을 확인하라.

이를 통해 VHDL 을 사용하여 Simulation 을 수행할 수 있다.
Verilog 를 사용하여 검증을 수행하려면
Verilog 를 선택하고 Drop-Down 메뉴에서 시뮬레이터를 선택하거나
PATH 변수에 나타나는 첫 번째 시뮬레이터를 Tool 에서 사용할 수 있게 한다.

 Co-simulation Dialog

C/RTL Co-simulation☒

Verilog/VHDL Simulator Selection

Auto

RTL Selection

☐ Verilog☒ VHDL

Options

☐ Setup Only

Dump Trace

none

☐ Optimizing Compile

☐ Reduce Diskspace

Compiled Library Location

Browse...

Input Arguments

☐ Do not show this dialog box again.

OK

Cancel

6-1-3. OK 를 눌러 VHDL Simulation 을 구동한다.

C/RTL Co-Simulation 이 실행되어 여러 파일을 생성 및 컴파일 한 다음 Design 을 시뮬레이션 한다.
이것은 3 단계를 거친다.

- * 먼저 VHDL Test Bench 를 실행하여 RTL Design 을 위한 입력을 생성한다.
- * 둘째, 새로 생성된 입력을 포함한 RTL Test Bench 가 생성되고
RTL Simulation 이 수행된다.
- * 마지막으로 RTL 의 출력이
VHDL Test Bench 에 다시 적용되어 결과를 확인한다.

Console 창에서 진행 상황과 Test 가 통과되었다는 Message 를 볼 수 있다.
따라서 합성된 Design 을 위한 별도의 Test Bench 작성이 필요 없다.

```

Starting C/RTL cosimulation ...
C:/Xilinx/Vivado/2017.4/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmul.prj/solution1/cosim.tcl
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2017.4/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'parimalp' on host 'xsjparimalp31' (Windows NT_amd64 version 6.1) on '
INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab1'
INFO: [HLS 200-10] Opening project 'C:/xup/hls/labs/lab1/matrixmul.prj'.
INFO: [HLS 200-10] Opening solution 'C:/xup/hls/labs/lab1/matrixmul.prj/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Setting target device to 'xc7z020clg484-1'
INFO: [COSIM 212-47] Using XSIM for RTL simulation.
INFO: [COSIM 212-14] Instrumenting C test bench ...
    Build using "C:/Xilinx/Vivado/2017.4/msys/bin/g++.exe"
    Compiling apatb_matrixmul.cpp
    Compiling matrixmul.cpp_pre.cpp.tb.cpp
    Compiling matrixmul_test.cpp_pre.cpp.tb.cpp
    Generating cosim.tv.exe
INFO: [COSIM 212-302] Starting C TB testing ...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.

***** xsim v2017.4 (64-bit)
**** SW Build 2086221 on Fri Dec 15 20:55:39 MST 2017
**** IP Build 2085800 on Fri Dec 15 22:25:07 MST 2017
** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

source xsim.dir/matrixmul/xsim_script.tcl
# xsim {matrixmul} -autoloadwcfg -tclbatch {matrixmul.tcl}
Vivado Simulator 2017.4
Time resolution is 1 ps
source matrixmul.tcl
## run all
Note: simulation done!
Time: 975 ns Iteration: 1 Process: /apatb_matrixmul_top/generate_sim_done_proc
Failure: NORMAL EXIT (note: failure is to force the simulator to stop)
Time: 975 ns Iteration: 1 Process: /apatb_matrixmul_top/generate_sim_done_proc
$finish called at time : 975 ns
## quit
INFO: [Common 17-206] Exiting xsim at Tue Feb 13 09:29:51 2018...
INFO: [COSIM 212-316] Starting C post checking ...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater th
Finished C/RTL cosimulation.

```


6-1-4. Simulation 검증이 완료되면 Simulation 보고서 탭이 열리고 결과가 표시된다.
 보고서는 Simulation 통과 또는 실패 여부를 나타낸다.
 또한 보고서에는 측정된 대기 시간 및 간격이 표시된다.

VHDL 만 선택 했으므로 Delay 및 간격(Interval)이
 다음 Clock Cycle 이후에 다음 입력을 제공할 수 있음을 나타낸다.
 Design 이 Pipeline 화 되어 있지 않으므로 Latency + 1 Clock Cycle 이 된다.

Cosimulation Report for 'matrixmul'								Cosimulation Report for 'matrixmul'							
Result								Result							
		Latency			Interval					Latency			Interval		
RTL	Status	min	avg	max	min	avg	max	RTL	Status	min	avg	max	min	avg	max
VHDL	Pass	79	79	79	NA	NA	NA	VHDL	Pass	106	106	106	NA	NA	NA
Verilog	NA	NA	NA	NA	NA	NA	NA	Verilog	NA	NA	NA	NA	NA	NA	NA

(a) ZedBoard

(b) ZYBO

Viewing Simulation Results in Vivado

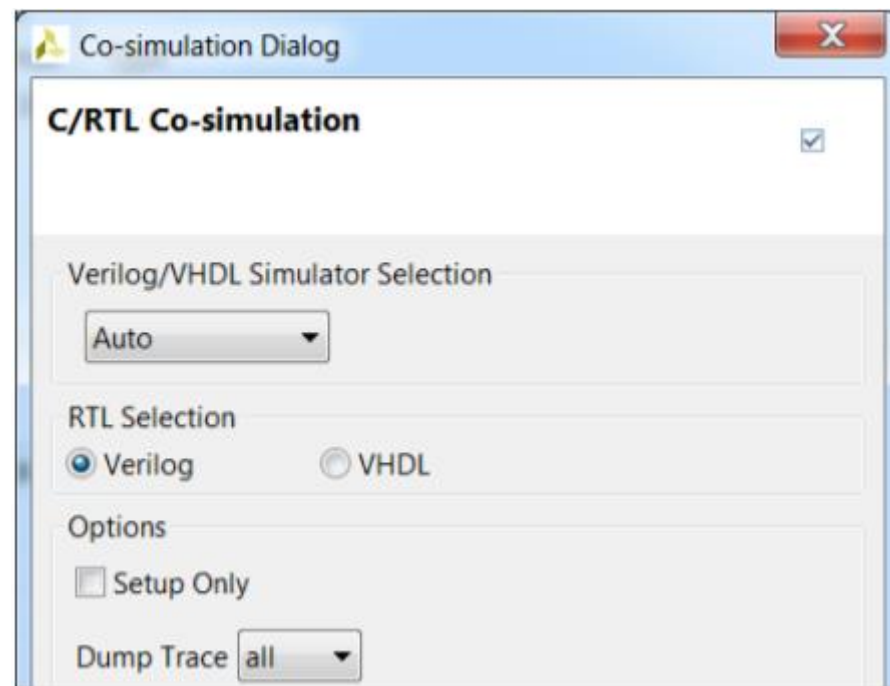
7-1. Dump 추적 옵션을 선택하여 Verilog Simulation 을 실행하라.

7-1-1. Solution -> Run C/RTL Co-Simulation 을 선택하거나 Synthesis View 에서 버튼을 클릭하여 Dialog Box 를 열면 원하는 Simulation 을 실행할 수 있다.

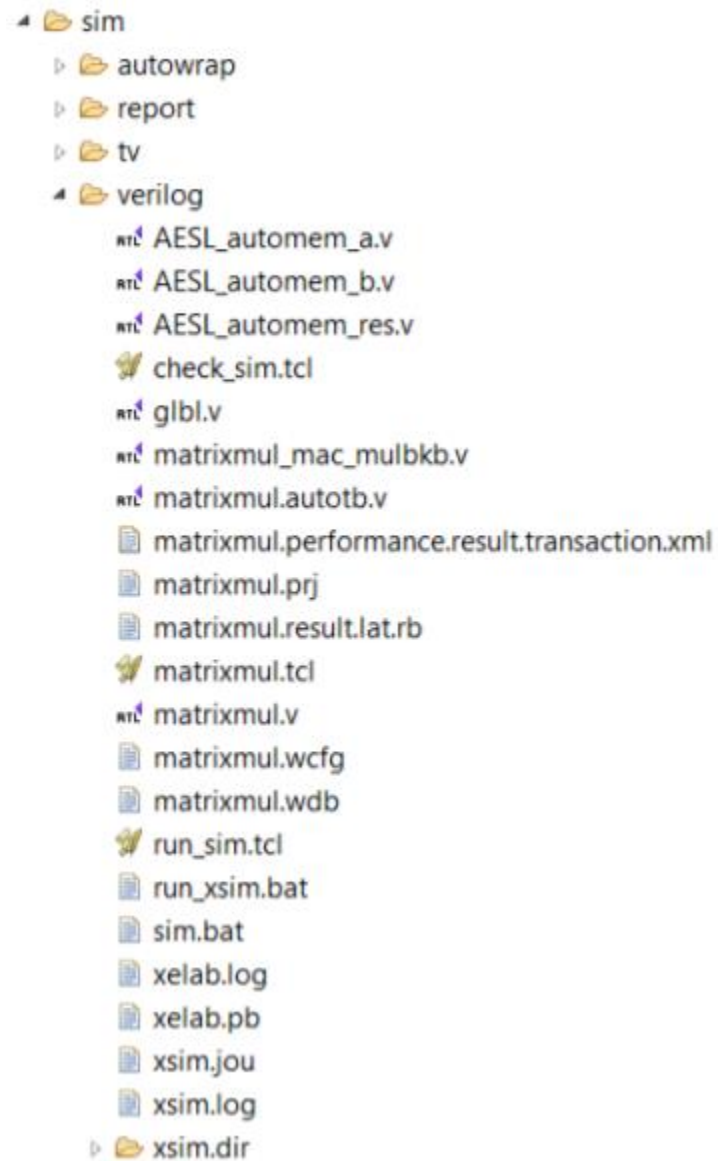
7-1-2. Verilog/VHDL Simulator Section 옵션을 Auto 로 유지하면서 Verilog RTL Selection 옵션을 클릭한다.

필요에 따라 Drop-Down 버튼을 클릭하고 XSim, ISim, ModelSim 및 Riviera 의 사용 가능한 목록에서 원하는 Simulator 를 선택할 수 있다.

7-1-3. Dump Trace 옵션을 모두 선택하고 확인을 클릭하라.



RTL 검증이 완료되면 Verilog Simulation 을 통과한(그리고 측정된 대기 시간 및 간격) Co-Simulation 보고서가 자동으로 열린다.
또한 Dump Trace 옵션이 사용되었고 Verilog 가 선택되었기 때문에 Verilog Simulation 디렉토리에서 2 개의 추적 파일 항목을 볼 수 있다.



Co-Simulation 보고서는 Test 가 지연 및 Latency 와 함께 Verilog 에 전달되었음을 보여준다.

Cosimulation Report for 'matrixmul'

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

(a) ZedBoard

Cosimulation Report for 'matrixmul'

Result							
RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	106	106	106	NA	NA	NA
Verilog	Pass	106	106	106	NA	NA	NA

(b) ZYBO

7-2. Dumped Traces 분석하기

7-2-1. Wave Viewer 를 열려면 스코프 파형 아이콘을 누른다.
Vivado 가 시작되고 Wave Viewer 가 열린다.

7-2-2. Waveform 창에서 필요에 따라 상단 신호 Design 을 확장한다.

7-2-3. 한 번의 반복으로 전체 시뮬레이션을 보려면
전체 확대 / 축소 도구에 해당하는 돋보기 아이콘을 누른다.

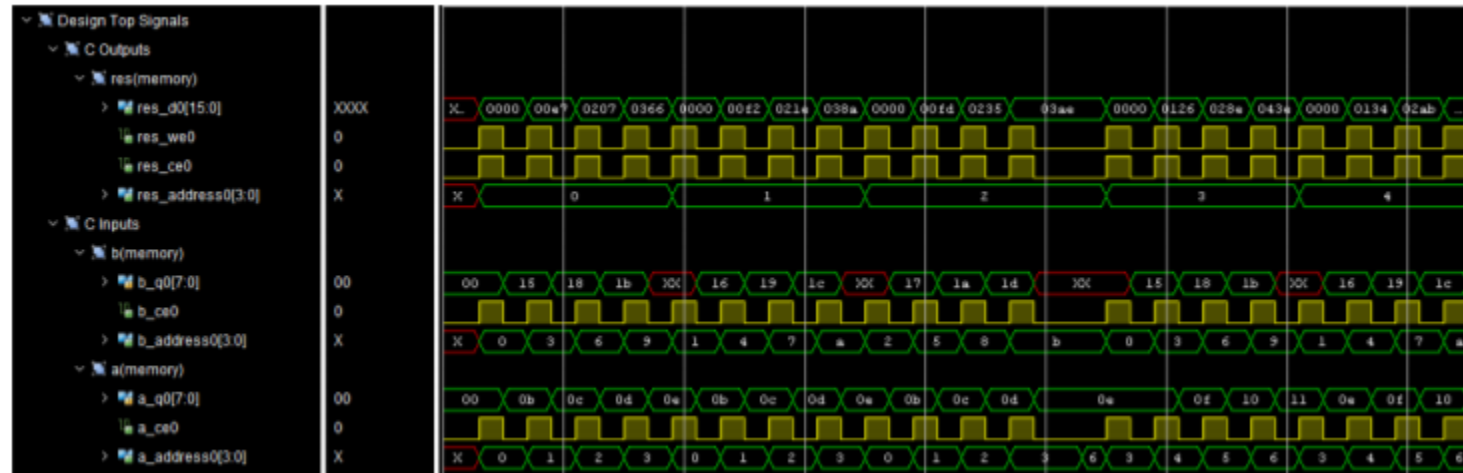
7-2-4. Waveform 창에서 a_address0 을 선택하고
마우스 오른쪽 버튼을 클릭한 다음 Radix -> Unsigned Decimal 을 선택하라.
마찬가지로 b_address0 및 res_address0 신호도 동일하게 처리하라.

7-2-5. 마찬가지로 a_q0, b_q0, 그리고 res_d0 radix 를 부호있는 0 진수로 설정한다.
그러면 파형을 관측할 수 있다.



ap_start 가 assert 되자마자
ap_idle 은 Design 이 계산 모드에 있음을 나타내기 위해 de-assert 된다.
ap_idle 신호는 진행 절차 완료를 나타내는
ap_done 이 선언될 때까지 de-assert 상태를 유지한다.
이는 106 clock cycles Latency 를 나타낸다.

7-2-6. 확대 버튼을 사용하여 ~ 120 ns 및 ~ 500 ns 의 영역을 표시한다.



Design 은 a_address0, a_ce0, b_address0, b_ce0 신호를 제공하여
요소 데이터를 예상하고 res_d0, res_we0
및 res_ce0 을 사용하여 결과를 출력한다.

7-2-7. 시뮬레이션의 다양한 부분을 보고 Design 작동 방식을 이해하라.

7-2-8. 작업이 완료되면 File -> Exit 를 선택하여 Vivado 를 닫는다.
Message 가 표시되면 확인을 클릭하고
저장하지 않고 프로그램을 닫으려면 취소를 클릭한다.

Export RTL and Implement

8-1. Vivado HLS 에서 VHDL 을 Language 로 선택하여 Design 을 Export 하고 Evaluate 옵션을 선택하여 Implementation 을 실행한다.

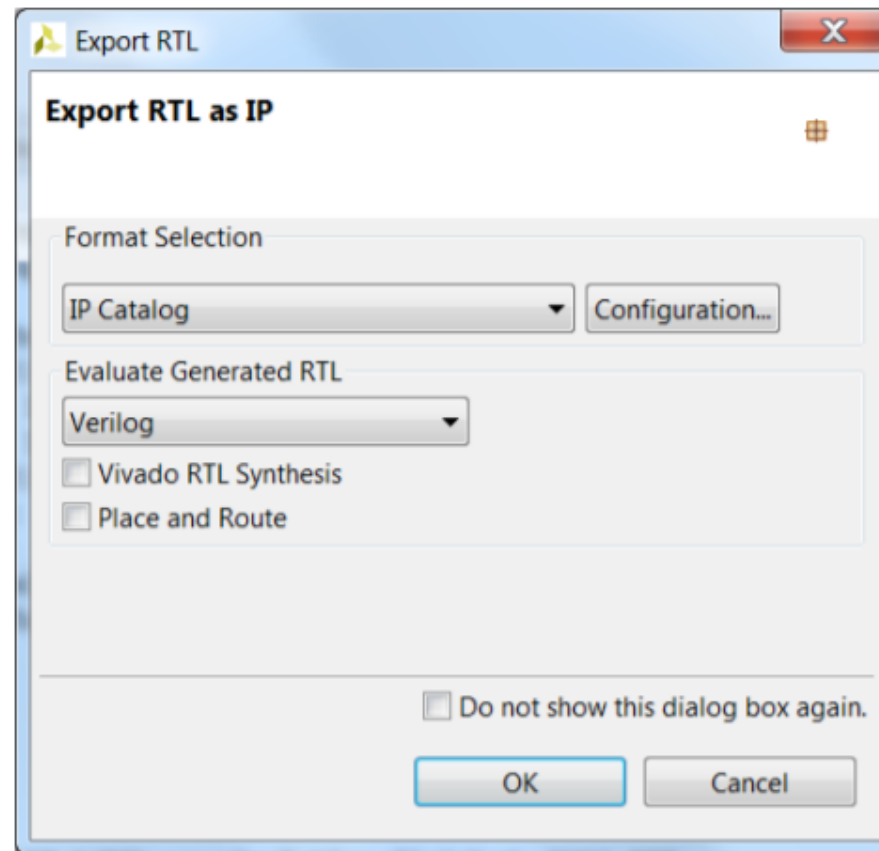
8-1-1. Vivado-HLS 에서 Solution -> Export RTL 을 선택하거나 노란 상자 박스에 + 표시가 있는 아이콘을 클릭하여 Dialog Box 를 열면 원하는 Implementation 을 실행할 수 있다.

Export RTL Dialog Box 가 열릴 것이다.

Format Selection: IP Catalog
Evaluate Generated RTL: Verilog

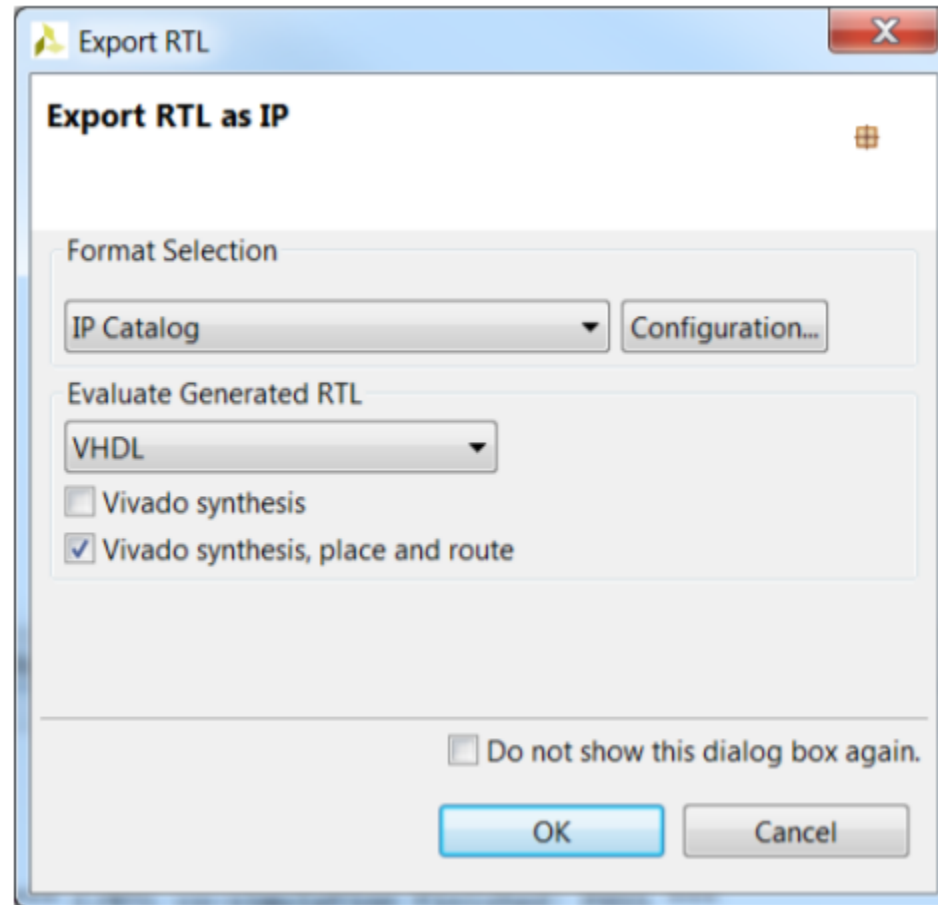
위와 같이 기본 설정을 사용하면 IP Packaging Process 가 실행되어 Vivado IP Catalog Package 를 생성한다.

Format Selection Drop-Down 메뉴에서 사용할 수 있는 또 다른 옵션은 DSP 용 System Generator 를 만드는 것이다.



8-1.2. Evaluate Generated RTL 필드의 Drop-Down 메뉴를 클릭하고 VHDL 을 선택하라.

8-1-3. Vivado RTL Synthesis 및 Place and Route 확인란을 클릭하여 Implementation Tool 을 실행한다.



8-1-4. OK 를 클릭하여 Implementation 이 시작된다.
Vivado HLS Console 창에서 진행 상황을 볼 수 있다.
이것은 여러 단계를 거친다.

- * RTL 을 IP-XACT 형식으로 IP 로 Export 하기
- * Evaluate 옵션을 선택하고 RTL Evaluation
 - * Synthesis 을 수행함
 - * 부품 배치 및 배선을 수행함

```
Starting export RTL ...
C:/Xilinx/Vivado/2017.4/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmul.prj/solution1/export.tcl
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2017.4/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'parimalp' on host 'xsjparimalp31' (Windows NT_amd64 version 6.1) on T
INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab1'
INFO: [HLS 200-10] Opening project 'C:/xup/hls/labs/lab1/matrixmul.prj'.
INFO: [HLS 200-10] Opening solution 'C:/xup/hls/labs/lab1/matrixmul.prj/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Setting target device to 'xc7z020clg484-1'
INFO: [IMPL 213-8] Exporting RTL as a Vivado IP.

***** Vivado v2017.4 (64-bit)
**** SW Build 2086221 on Fri Dec 15 20:55:39 MST 2017
**** IP Build 2085800 on Fri Dec 15 22:25:07 MST 2017
** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

source matrixmul.tcl -notrace
Command: synth_design -top matrixmul -part xc7z020clg484-1 -no_iobuf -mode out_of_context
Starting synth_design
Attempting to get a license for feature 'Synthesis' and/or device 'xc7z020'
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7z020'
INFO: Launching helper process for spawning children vivado processes
INFO: Helper process launched with PID 5748

-----
Starting RTL Elaboration : Time (s): cpu = 00:00:06 ; elapsed = 00:00:06 . Memory (MB): peak = 383.078 ; gain = 111.633
-----
INFO: [Synth 8-638] synthesizing module 'matrixmul' [C:/xup/hls/labs/lab1/matrixmul.prj/solution1/impl/vhdl/matrixmul.vhd:33]
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "none" *) [C:/xup/hls/labs/lab1/matrixmul.prj/solution1/impl/vhdl/m
INFO: [Synth 8-5534] Detected attribute (* fsm_encoding = "none" *) [C:/xup/hls/labs/lab1/matrixmul.prj/solution1/impl/vhdl/m
```

```
Implementation tool: Xilinx Vivado v.2017.4
Project:             matrixmul.prj
Solution:            solution1
Device target:       xc7z020clg484-1
Report date:         Tue Feb 13 10:29:28 -0800 2018

#=== Post-Implementation Resource usage ===
SLICE:               10
LUT:                 34
FF:                  27
DSP:                  1
BRAM:                 0
SRL:                  0
#=== Final timing ===
CP required:         10.000
CP achieved post-synthesis: 3.019
CP achieved post-implementation: 3.728
Timing met
INFO: [Common 17-206] Exiting Vivado at Tue Feb 13 10:29:28 2018...
Finished export RTL.
```

실행이 완료되면 Implementation 보고서가 Information Pane 에 표시된다.

```
Implementation tool: Xilinx Vivado v.2017.4
Project:             matrixmul.prj
Solution:            solution1
Device target:       xc7z020clg484-1
Report date:         Tue Feb 13 10:29:28 -0800 2018

#=== Post-Implementation Resource usage ===
SLICE:               10
LUT:                 34
FF:                  27
DSP:                 1
BRAM:                0
SRL:                 0
#=== Final timing ===
CP required:         10.000
CP achieved post-synthesis: 3.019
CP achieved post-implementation: 3.728
Timing met
INFO: [Common 17-206] Exiting Vivado at Tue Feb 13 10:29:28 2018...
Finished export RTL.
```

실행이 완료되면 Implementation 보고서가 Information Pane 에 표시된다.

Export Report for 'matrixmul'

General Information

Report date: Tue Feb 13 10:29:28 -0800 2018
Project: matrixmul.prj
Solution: solution1
Device target: xc7z020clg484-1
Implementation tool: Xilinx Vivado v.2017.4

Resource Usage

	VHDL
SLICE	10
LUT	34
FF	27
DSP	1
BRAM	0
SRL	0

Final Timing

	VHDL
CP required	10.000
CP achieved post-synthesis	3.019
CP achieved post-implementation	3.728

Timing met

(a) ZedBoard

Export Report for 'matrixmul'

General Information

Report date: Sat Feb 24 07:57:50 -0800 2018
Project: matrixmul.prj
Solution: solution1
Device target: xc7z010clg400-1
Implementation tool: Xilinx Vivado v.2017.4

Resource Usage

	VHDL
SLICE	10
LUT	33
FF	28
DSP	1
BRAM	0
SRL	0

Final Timing

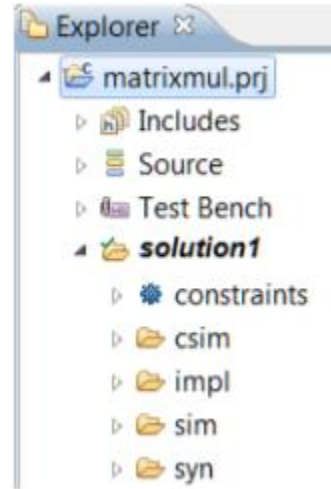
	VHDL
CP required	8.000
CP achieved post-synthesis	3.019
CP achieved post-implementation	3.884

Timing met

(b) ZYBO

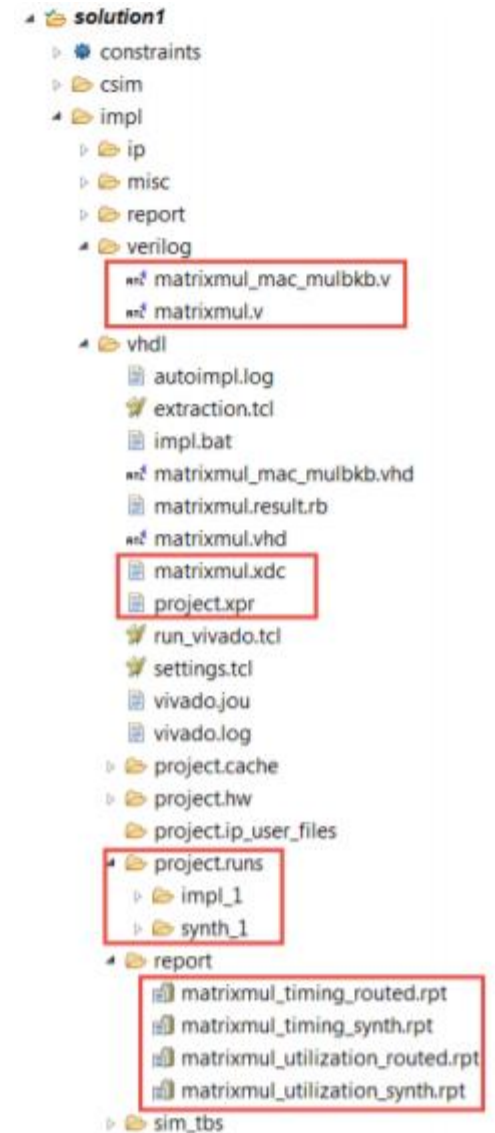
Timing Constraint(타이밍 제약 조건), 달성된 기간(3.703 [ZedBoard], 3.704[Zybo] ns), 사용된 Resource 의 유형 및 양을 관찰한다.

8-1-5. Explorer 를 축소하고 ip, report, Verilog 및 VHDL 하위 폴더가 만들어지는 위치에 impl 폴더가 만들어졌는지 확인한다.



8-1-6. Verilog 및 VHDL 하위 폴더를 확장하고 Verilog 하위 폴더에만 rtl 파일이 있는 반면 VHDL 하위 폴더에는 합성 및 구현 작업이 여러 파일과 하위 폴더로 구성되어 있으므로 관찰해보도록 한다.

여기는 project.xpr 파일(Vivado Project File), matrixmul.xdc 파일(타이밍 제한 파일), project.runs 폴더(합성 및 구현 실행에 의해 생성된 synth_1 및 impl_1 하위 폴더 포함) 폴더가 포함된다.



8-1-7. ip 폴더를 확장하고 zip 파일(xilinx_com_hls_matrixmul_1_0.zip)로 Packaging 된 IP 를 관찰하여 Vivado IP Catalog 에 추가할 준비가 되었다.

8-1-8. File -> Exit 를 선택하여 Vivado HLS 를 닫는다.

Conclusion

이 Lab 에서 Vivado HLS 를 사용하여 High-Level Synthesis 설계 흐름의 주요 단계를 완료했다. Project 를 만들고 Source 파일을 추가하고 Design 을 합성하고 Design 을 Simulation 한 다음 Design 을 구현했다. 또한 Analysis 기능을 사용하여 Scheduling 및 Binding 하는 방법을 배웠다.

8-1-7. ip 폴더를 확장하고 zip 파일(xilinx_com_hls_matrixmul_1_0.zip)로 Packaging 된 IP 를 관찰하여 Vivado IP Catalog 에 추가할 준비가 되었다.

8-1-8. File -> Exit 를 선택하여 Vivado HLS 를 닫는다.

Conclusion

이 Lab 에서 Vivado HLS 를 사용하여 High-Level Synthesis 설계 흐름의 주요 단계를 완료했다. Project 를 만들고 Source 파일을 추가하고 Design 을 합성하고 Design 을 Simulation 한 다음 Design 을 구현했다. 또한 Analysis 기능을 사용하여 Scheduling 및 Binding 하는 방법을 배웠다.

References

1. <https://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-high-level-synthesis-flow-zynq.html>