

Zynq를 활용한 SoC / FPGA 설계

HDL 및 HLS를 이용한 설계 이해 및 실습





CONTENTS

- 01. SoC 설계 이론
- 02. ZYNQ를 이용한 SoC 설계 이해
- 03. ZYNQ를 이용한 설계 실습(HDL)
- 04. HLS를 이용한 FPGA 설계

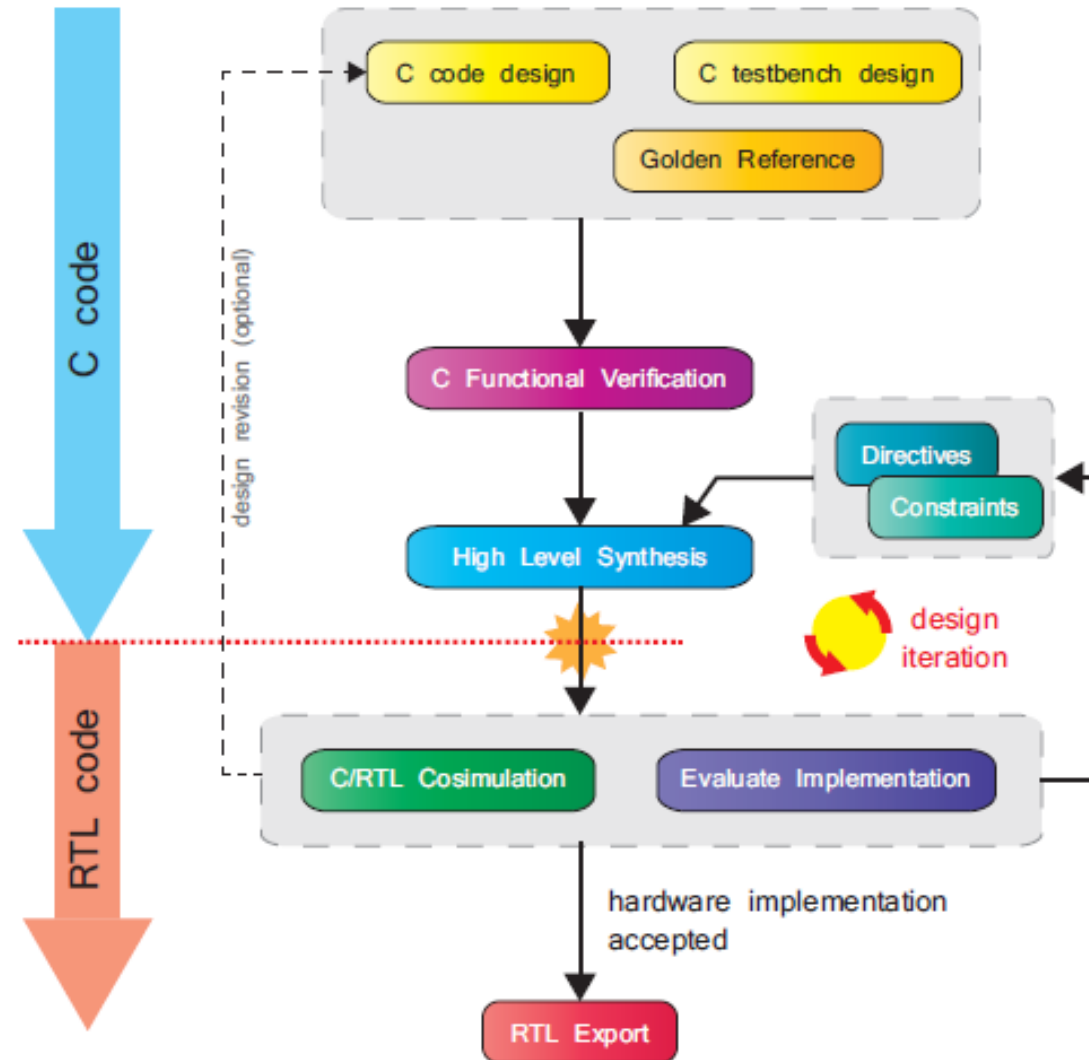
HLS를 이용한 FPGA 설계

CONTENTS

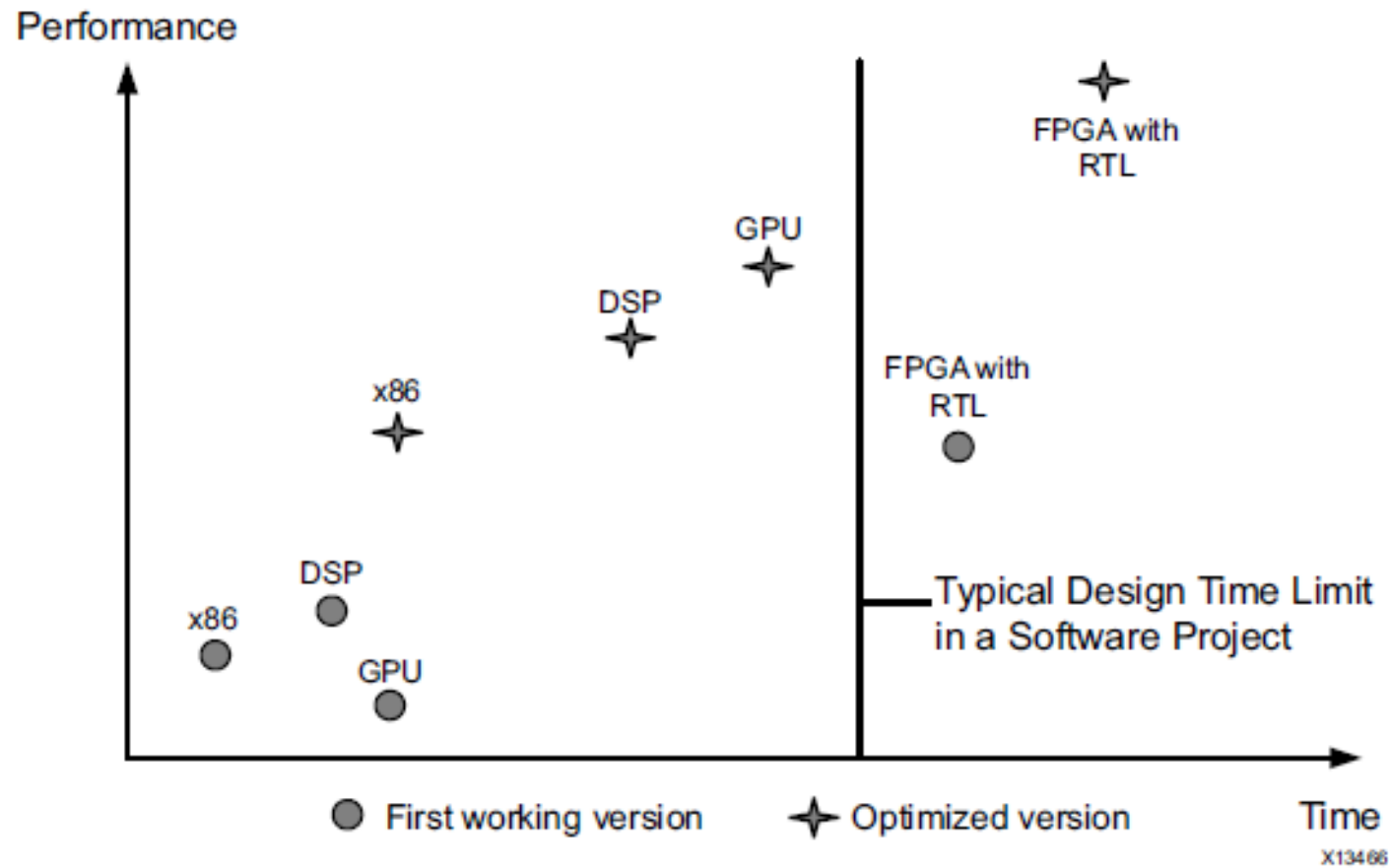
- 01. HLS란?
- 02. HLS를 사용한 시스템 설계
- 03. HLS를 이용한 FPGA 설계 실습

01. HLS(High-Level Synthesis)

- Xilinx에서 개발한 Tool
- C-level Design / Verify
 - 고급 언어(C/C++/System C)로 작성된 소스코드를 RTL 레벨로 변환
 - C-Level에서의 디자인 검증
- HDL 기반 설계 대비 생산성 향상
- 시스템 기준 → HW 가속으로 성능 향상

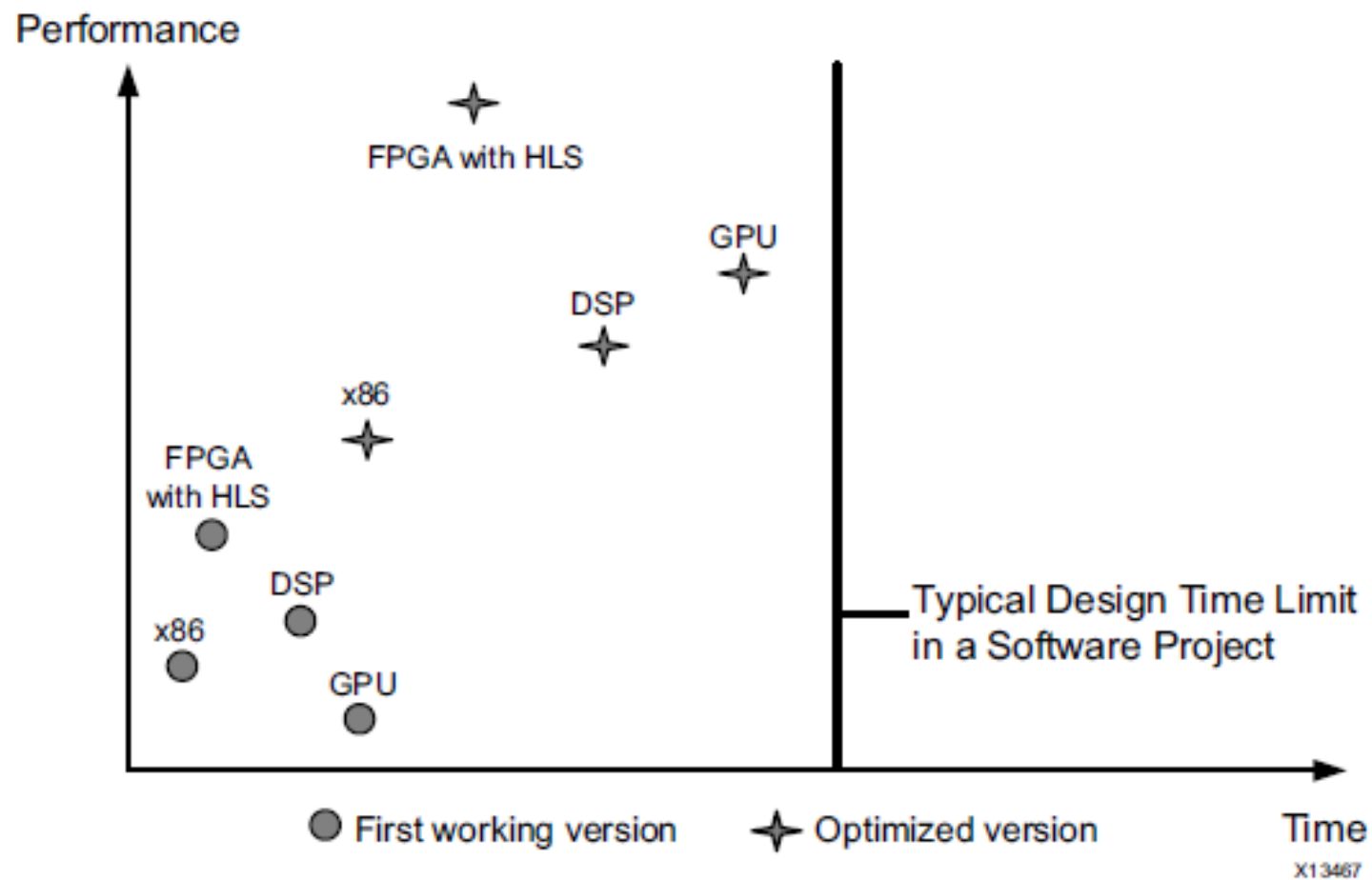


Why HLS?



시간 vs. 성능

Why HLS?

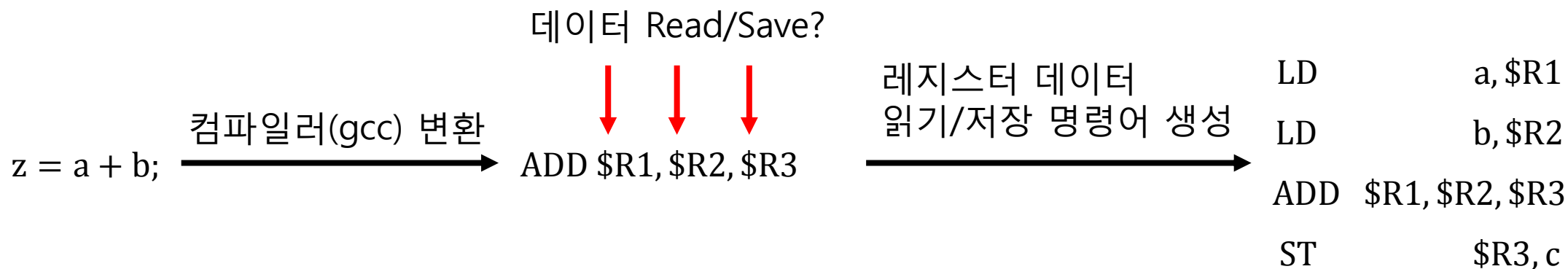


시간 vs. 성능

프로세서 vs. FPGA

프로세서에서의 프로그램 실행

→ 여러 개의 명령어의 집합

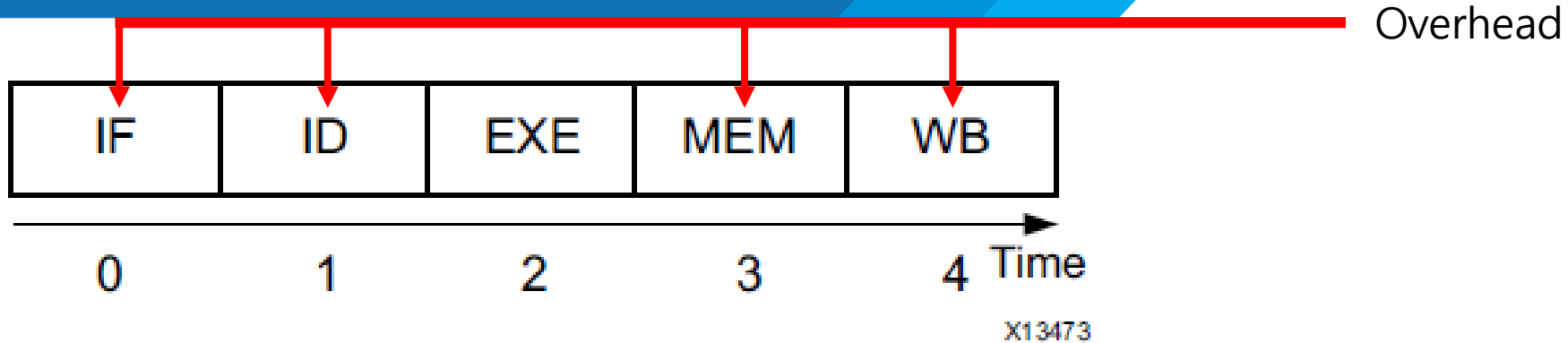


a, b, c Load에 필요한 클럭 수

캐시 내부 : ~100

DRAM : 수 백, 수 천

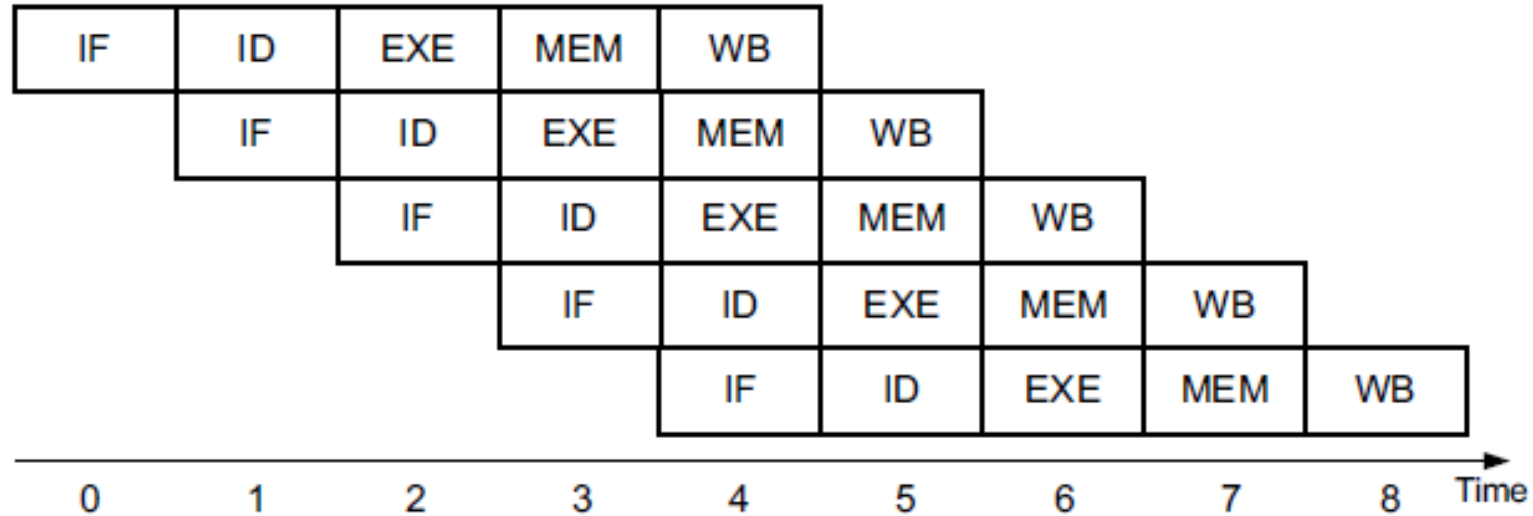
프로세서 vs. FPGA



프로세서에서의 명령어 실행 주기

1. Instruction fetch (IF)
2. Instruction decode (ID)
3. Execute (EXE)
4. Memory operations (MEM)
5. Write back (WB)

프로세서 vs. FPGA



다중 명령어 실행 유닛을 가진 프로세서

X13474

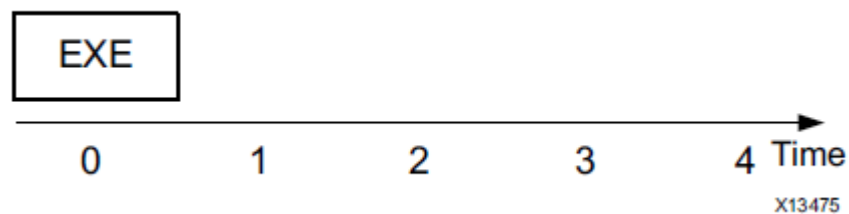
Instruction Pipelining

- 최적의 경우 한 클럭 사이클마다 하나의 EXE 실행

프로세서 vs. FPGA

FPGA(HLS)에서의 프로그램 실행

→ 단일 명령어 실행



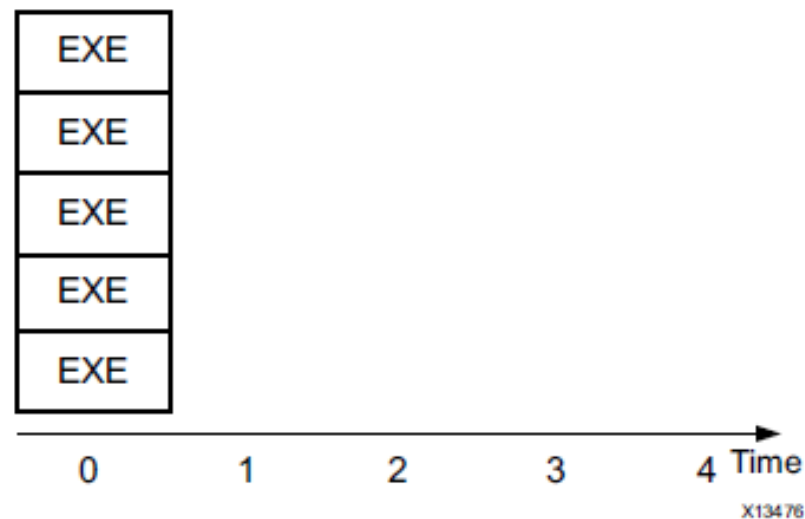
FPGA에서의 명령어 실행

동일한 계산 구조(ALU)가 아닌, target specific한 계산 회로 구현

프로세서 vs. FPGA

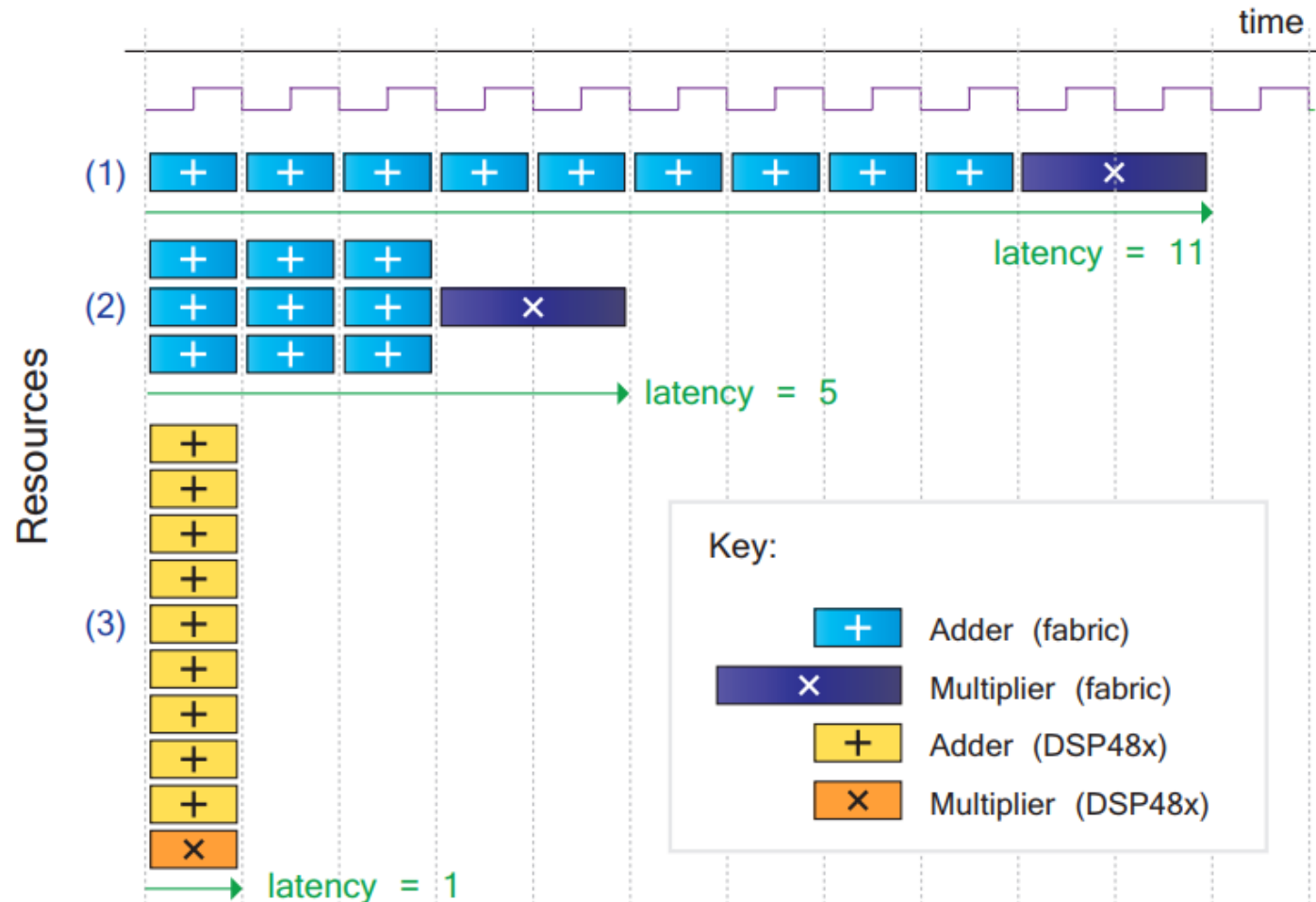
FPGA(HLS)에서의 다중 명령어 실행

→ 오버헤드 없이 동시에 여러 명령어를 실행할 수 있음



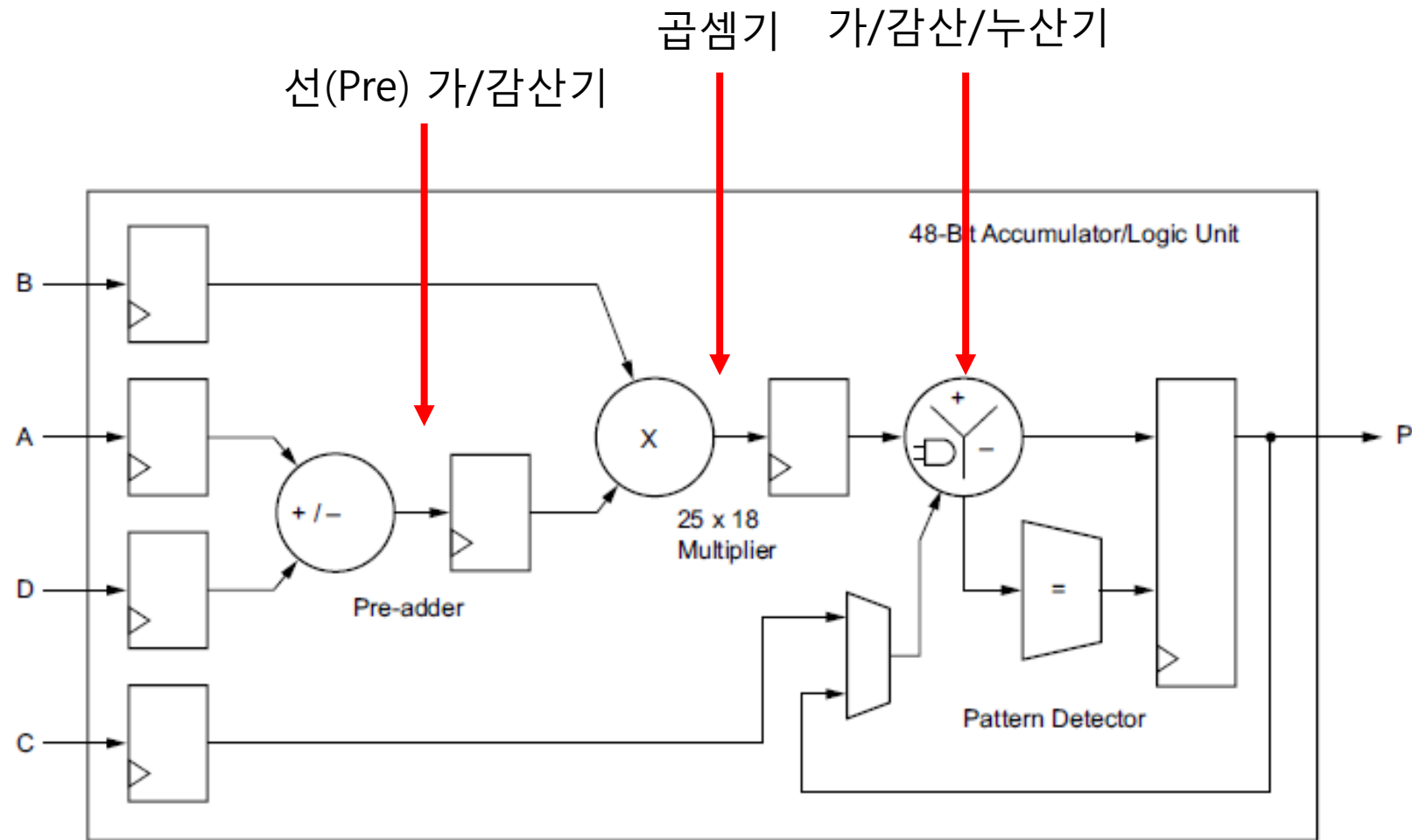
FPGA에서의 다중 명령어 실행

Parallelization

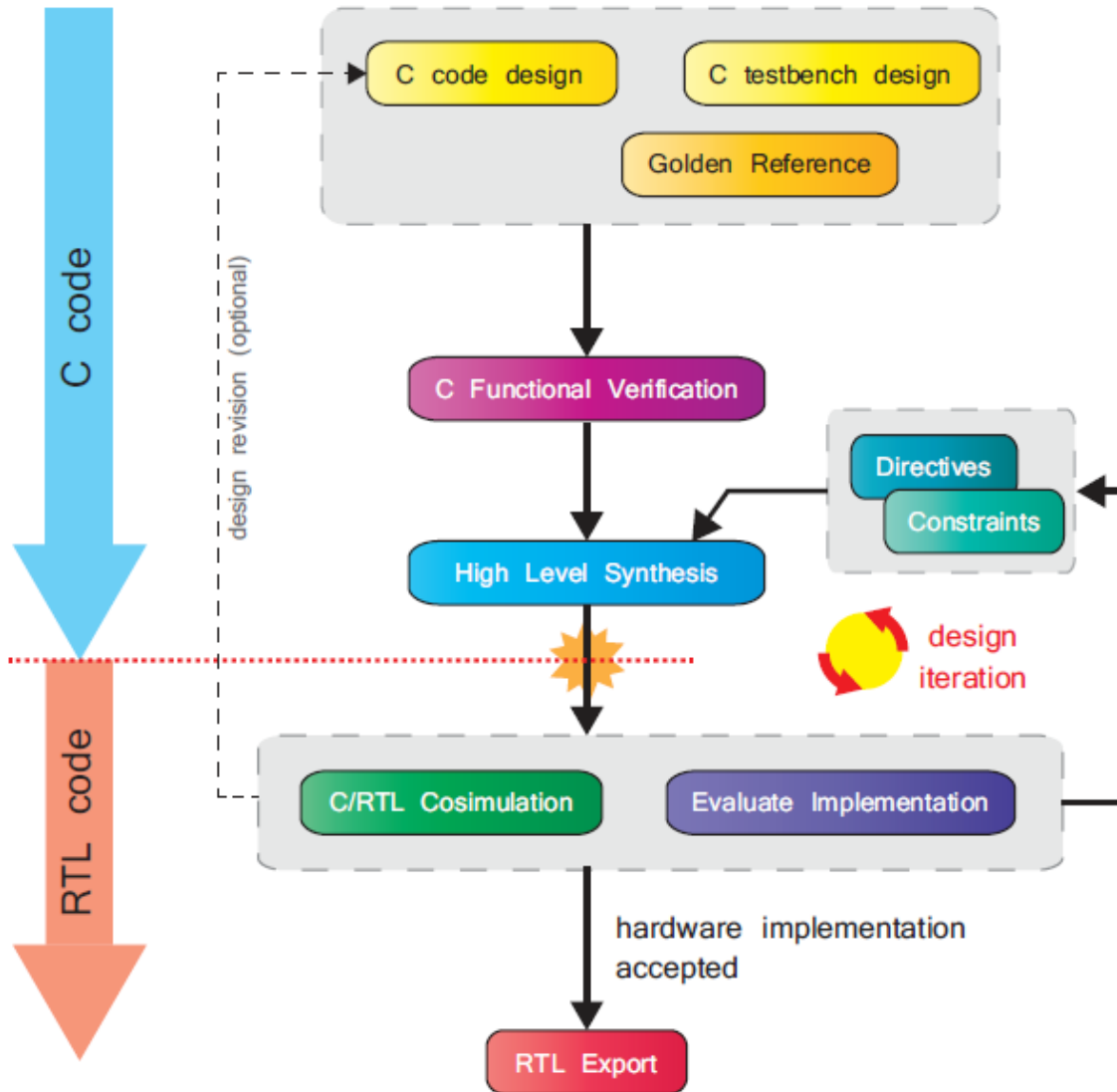


BRAM, DSP

- Block RAM
 - RAM
 - ROM
 - FIFO buffer
- DSP48E1(ALU)
 - 3가지 블록의 조합
 - $p = a \times (b + d) + c$
 - $p += a \times (b + d)$



02. HLS를 사용한 시스템 설계



• Inputs

- C code Design
- C Testbench Design
- Golden Reference(출력 데이터)
- Directives
- Constraints

• High-Level Synthesis

- Analysis, Processing C-based Code
- Directives, Constraints 만족 확인

HLS Synthesis Basic Rules

- Scheduling
 - Operation의 순서, 병렬화 결정
 - Dependency, Clock cycle, FPGA 속도, User directives에 따라 결정됨
- Binding
 - 사용할 HW 리소스 결정(연산의 경우 내부 DSP48 블록 사용)
 - FPGA의 종류에 따라 결정됨
- Control logic extraction
 - 제어(loop 문, if-else 문, case 문)등을 FSM(Finite State Machine)으로 구현하기 위해 제어 로직을 추출

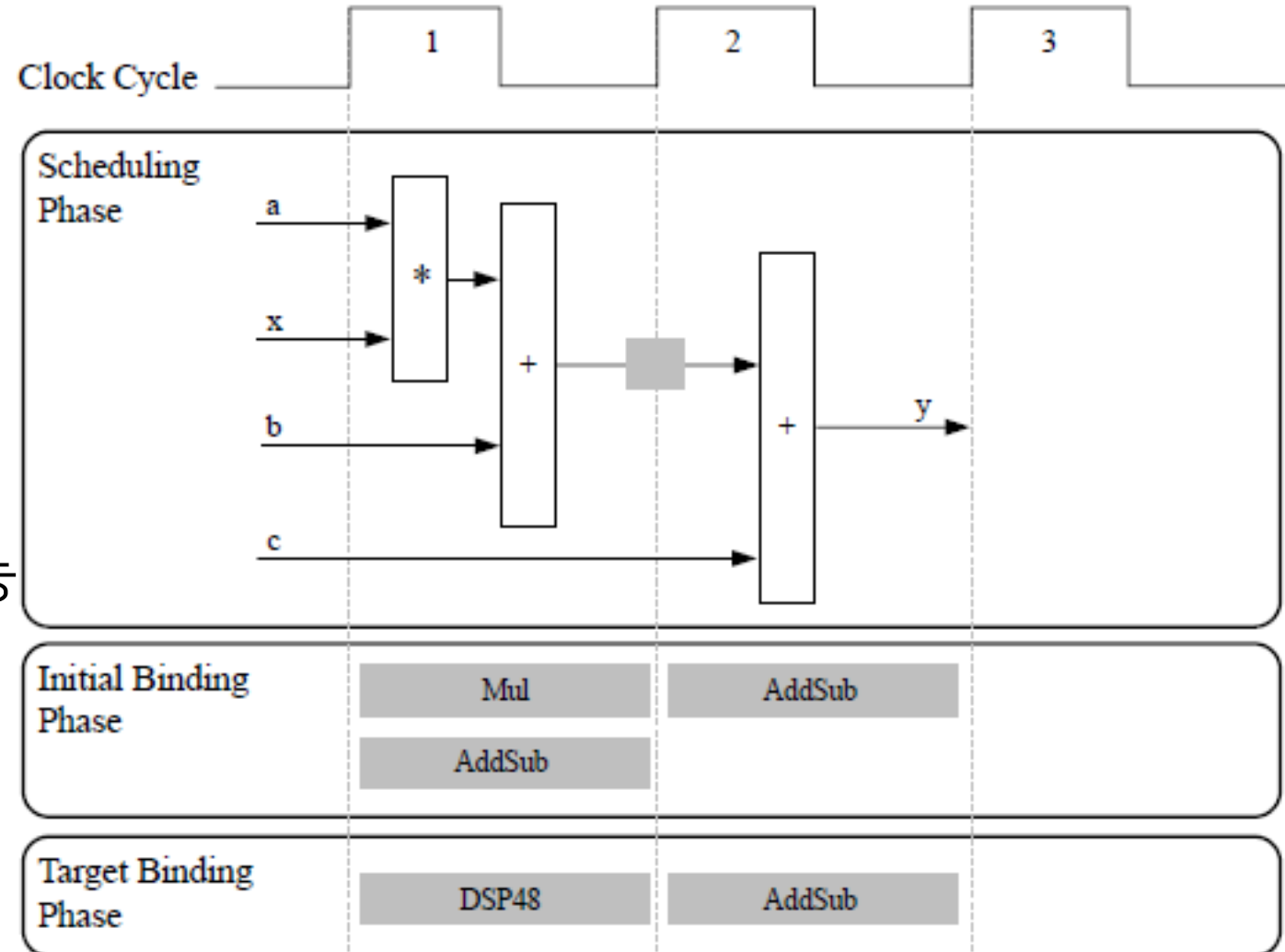
HLS Synthesis Basic Rules

- Function Arguments → I/O Port
- Function → RTL Block
 - 하나의 함수는 하나의 RTL 블록으로 구현됨
- 반복문(Loop)
 - RTL 블록 단위 반복
 - 반복 없이 병렬화
 - FSM 등을 이용한 파이프라인 중첩
- 어레이의 경우 Block RAM 내부에 저장

Scheduling, Binding EX.

```
int foo(char x, char a, char b, char c){  
    char y;  
    y = x * a + b + c;  
    return y;  
}
```

- Scheduling Phase
 - 연산 선후관계에 따른 배치
 - $(x * a + b)$ 한 클럭 안에 계산 가능
- Binding Phase
 - *, + 로 연산 구분
 - FPGA 내부 DSP48 사용



X14220-061518

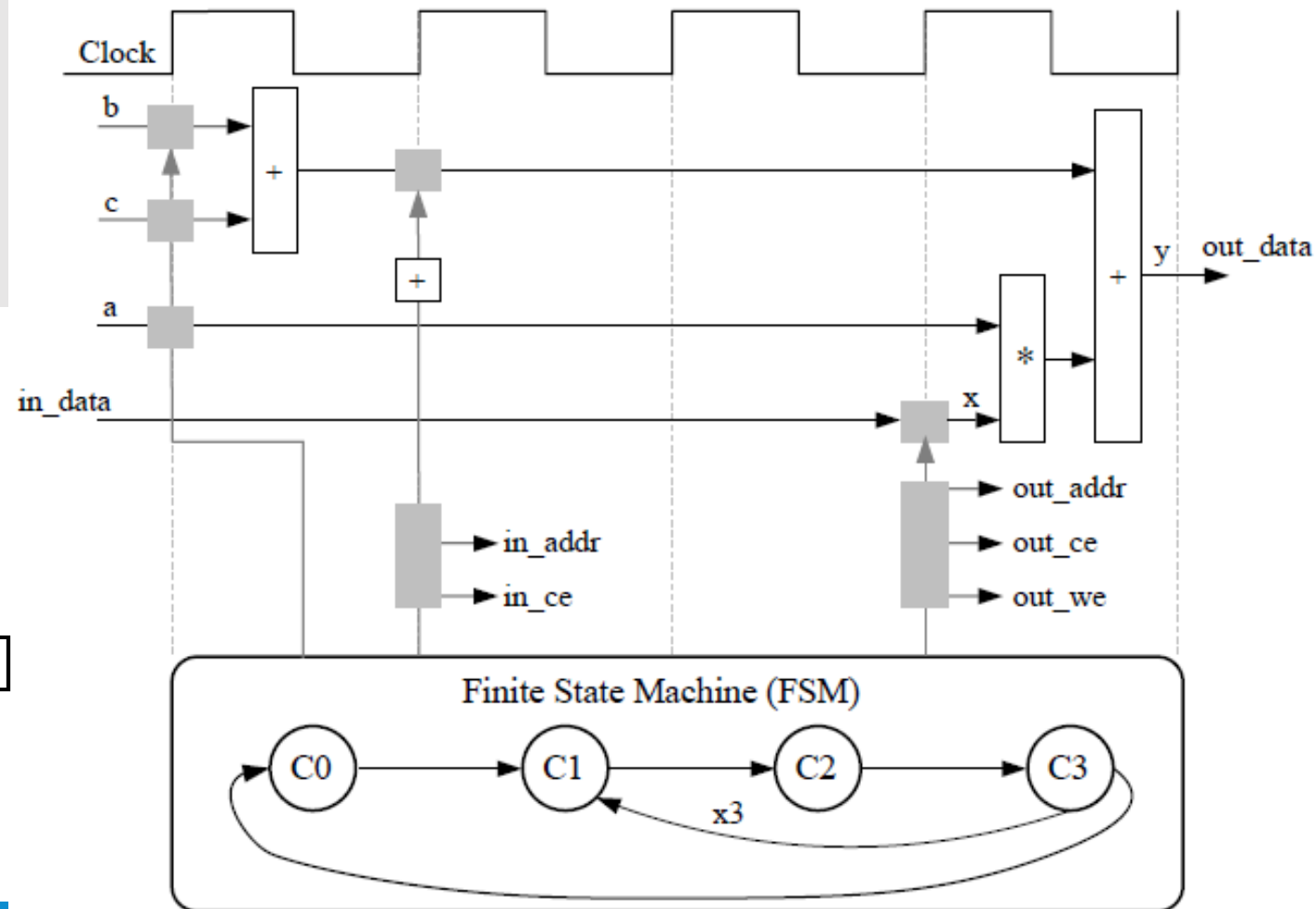
Control Logic

```
void foo(int in[3], char a, char b, char c, int out[3]) {  
    int x,y;  
    for(int i = 0; i < 3; i++) {  
        x = in[i];  
        y = a * x + b + c;  
        out[i] = y;  
    }  
}
```

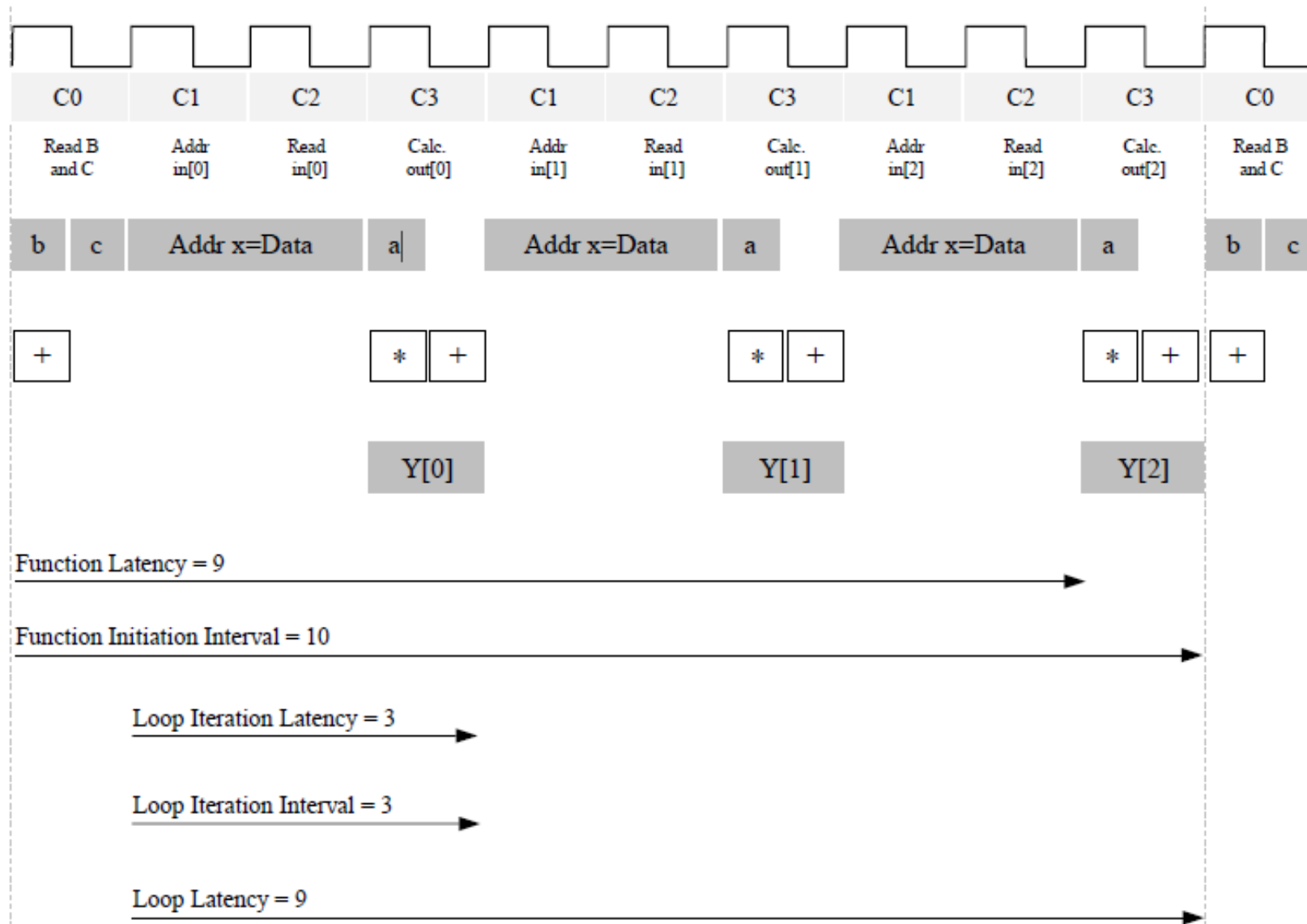
- 반복문

- C0 - (C1,C2,C3) X3 - C0
- C0: $b + c$
- C1: in[n]의 Addr, Chip enable
- C2: read from in[n], store at x[n]
- C3: $y = a * x + b + c$

- Array: 모듈 외부에 BRAM 구현



Control Logic



Data Type

Type	Description	Number of Bits ^a	Range ^b
char	Representation of the basic character set.	8	-128 to 127
signed char		8	-128 to 127
unsigned char		8	0 to 255
short int	A reduced precision version of int, requiring less storage.	16	-32,768 to 32,767
unsigned short int		16	0 to 65,535
int	The basic integer data type.	32	-2,147,483,648 to 2,147,483,647
unsigned int		32	0 to 4,294,967,295
long int	In many cases the long int type will be the same length as int, i.e. 32 bits.	32	-2,147,483,648 to 2,147,483,647
unsigned long int		32	0 to 4,294,967,295
long long int	An extended precision integer type.	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long int		64	0 to 18,446,744,073,709,551,615
float	Single precision floating point (IEEE 754)	32	-3.403e ⁺³⁸ to 3.403e ⁺³⁸
double	Double precision floating point (IEEE 754)	64	-1.798e ⁺³⁰⁸ to 1.798e ⁺³⁰⁸

```
int foo(char x, char a, char b, char c)
```

- C언어의 Data Type은 고정
- Data bit, Sign, floating point
- $S = A * B;$
- A, B: 18-bit, S: 36-bit (RTL)
- A, B: 32-bit, S: 64-bit (C-Data type)
- DSP48E: 1 → 4 Slices

Data Type

Language	Integer Data Type	Description	Required Header
C	int N (e.g. int7)	signed integer of N bits precision	#include "ap_cint.h"
	uint N (e.g. uint7)	unsigned integer of N bits precision	
C++	ap_int< N > (e.g. ap_int<7>)	signed integer of N bits precision	#include "ap_int.h"
	ap_uint< N > (e.g. ap_uint<7>)	unsigned integer of N bits precision	

Arbitrary Data Type

Verilog: input wire data[17:0];

HLS-C: uint18 data;

Data Type

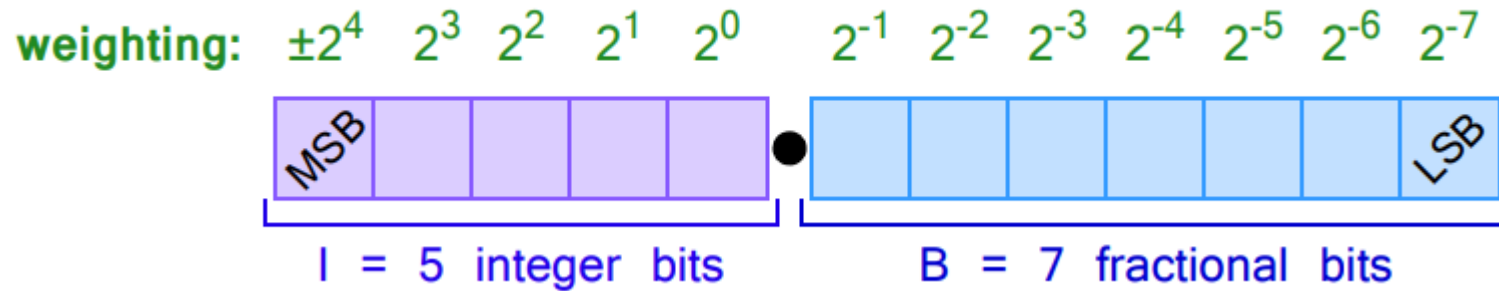
```
// C code example
#include "ap_cint.h"

void top_level_function (..)
{
    // declarations
    int6 small_signed;
    uint10 big_unsigned;
    int22 vbig_signed;
    ...
}
```

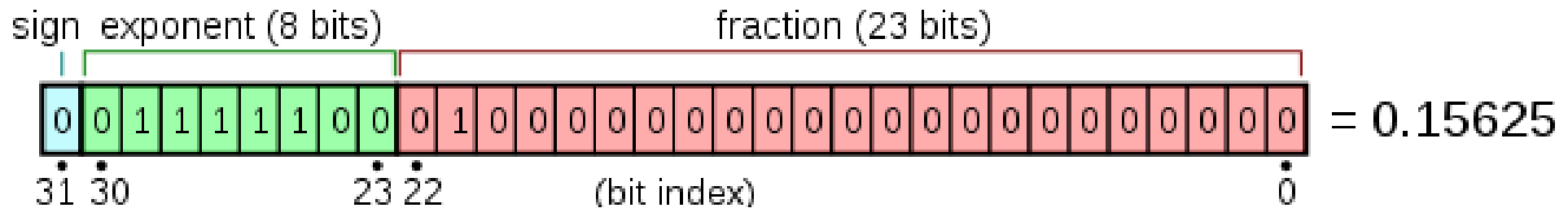
```
// C++ code example
#include "ap_int.h"

void top_level_function (..)
{
    // declarations
    ap_int<6> small_signed;
    ap_uint<10> big_unsigned;
    ap_int<22> vbig_signed;
    ...
}
```

Data Type(floating point)



$$I = 5, B = 7, W = I + B = 12$$



Arbitrary Float Point

Language	Fixed Point Data Type	Description	Required Header
C++	<code>ap_fixed<W,I,Q,O,N></code>	Signed fixed point number of <i>I</i> integer bits and <i>W-I</i> fractional bits.	#include "ap_fixed.h"
	<code>ap_ufixed<W,I,Q,O,N></code>	Unsigned fixed point number of <i>I</i> integer bits and <i>W-I</i> fractional bits.	

```
// C++ code example
#include "ap_fixed.h"

void top_level_function (...)
{
    // declarations
    ap_ufixed<8,3> small_unsigned; // 3 int, 5 fract, defaults
    ap_fixed<10,4,AP_RND> big_signed; // round to + inf.
    ap_ufixed<10,4,AP_RND_ZERO> big_unsigned; // round to zero
    ap_fixed<21,10,AP_TRN,AP_SAT> vbig_signed; // trunc., satur.
    ap_ufixed<21,10,AP_RND_CONV> vbig_unsigned; // conv. round.
    ...
}
```

Arbitrary Data Type
Verilog: input wire data[1
HLS-C: uint18 data;

Pointers

- Array Access
- Pointer to External Memory
- Dynamic Memory Allocation
→ Not supported

```
int A[10];  
int *pA;
```

```
pA = A;
```

```
void foo(int *data_in,...)  
{  
    int item1, item2, item3;  
  
    item1 = *data_in;  
    item2 = *(data_in + 1);  
    item3 = *(data_in + 2);  
    ...  
}
```

```
int *A = malloc(10*sizeof(int));
```

I/O Port

```
void find_average_of_best_X (int *average, int samples[8], int X)
{
    // body of function (statements, sub-function calls, etc.)
}
```

- Port Name
- Direction
- Data Type, Dimension



Port Direction

C/C++ Function Argument	RTL Port Type
An argument which is read from and never written to	in
An argument which is written to and never read from	out
A value output by the function return statement	out
An argument which is both written to and read from	inout (bidirectional)

Port Interface Protocol Types

- ap: AXI Protocol
- ack: Acknowledgement
- vld: valid / ovld: out valid
- hs: Hand shaking(ack, vld)
- memory: array

Table 15.7: Protocol synthesis: supported types and defaults (S = supported; D = default) [18]

Argument Type	Variable			Pointer Variable			Array			Reference Variable		
	pass-by-value			pass-by-reference			pass-by-reference			pass-by-reference		
Interface Type ^a	I	IO	O	I	IO	O	I	IO	O	I	IO	O
ap_none	D	-	-	D	S	S	-	-	-	D	S	S
ap_stable	S	-	-	S	S	-	-	-	-	S	S	-
ap_ack	S	-	-	S	S	S	-	-	-	S	S	S
ap_vld	S	-	-	S	S	D	-	-	-	S	S	D
ap_ovld	-	-	-	-	D	S	-	-	-	-	D	S
ap_hs	S	-	-	S	S	S	S	-	S	S	S	S
ap_memory	-	-	-	-	-	-	D	D	D	-	-	-
bram	-	-	-	-	-	-	S	S	S	-	-	-
ap_fifo	-	-	-	S	-	S	S	-	S	S	-	S
ap_bus	-	-	-	S	S	S	S	S	S	S	S	S
axis	S	-	-	S	-	S	S	-	S	S	-	S
s_axilite	S	-	S	S	S	S	-	-	-	S	S	S
m_axi	-	-	-	S	S	S	S	S	S	S	S	S

a. Reading along the row: I = input port; IO = inout (bidirectional) port; O = output port.

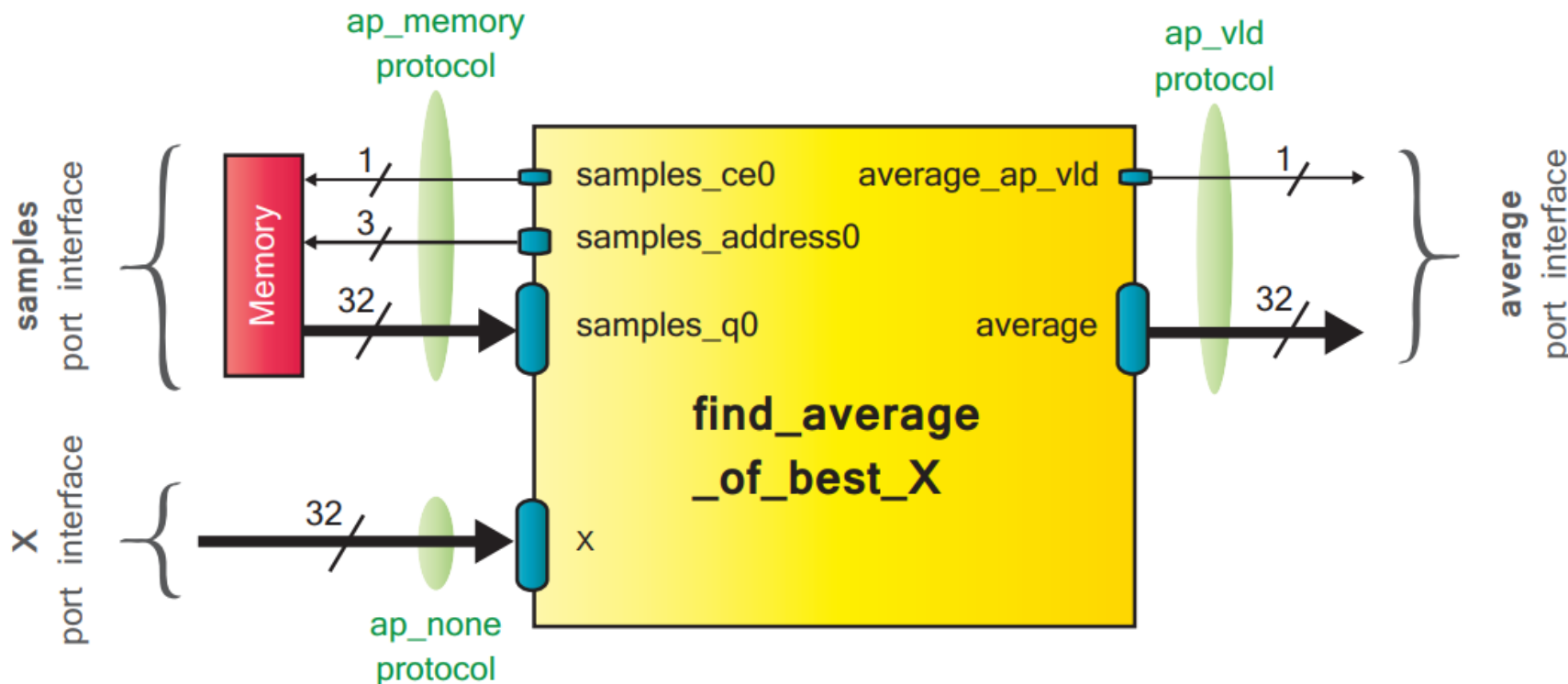
Port Interface

```
void find_average_of_best_X (int *average, int samples[8], int X)
{
    // body of function (statements, sub-function calls, etc.)
}
```

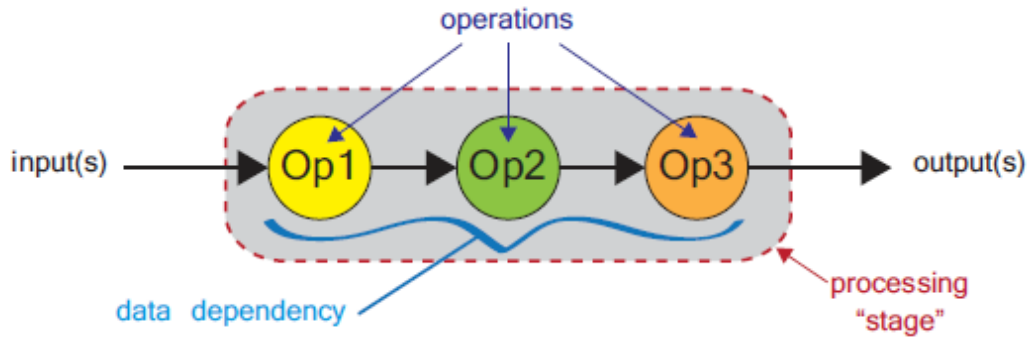
x : pass-by value → ap_none

samples[8] : array input → ap_memory

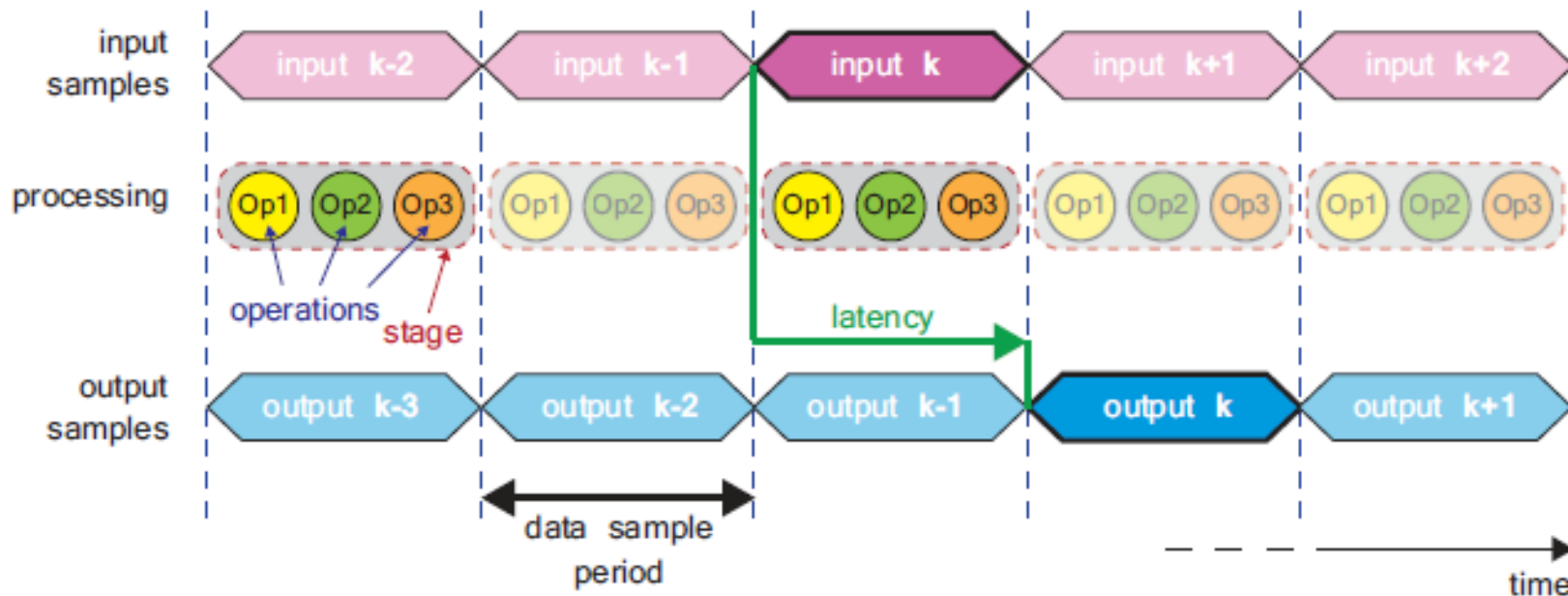
average : output pointer → ap_vld



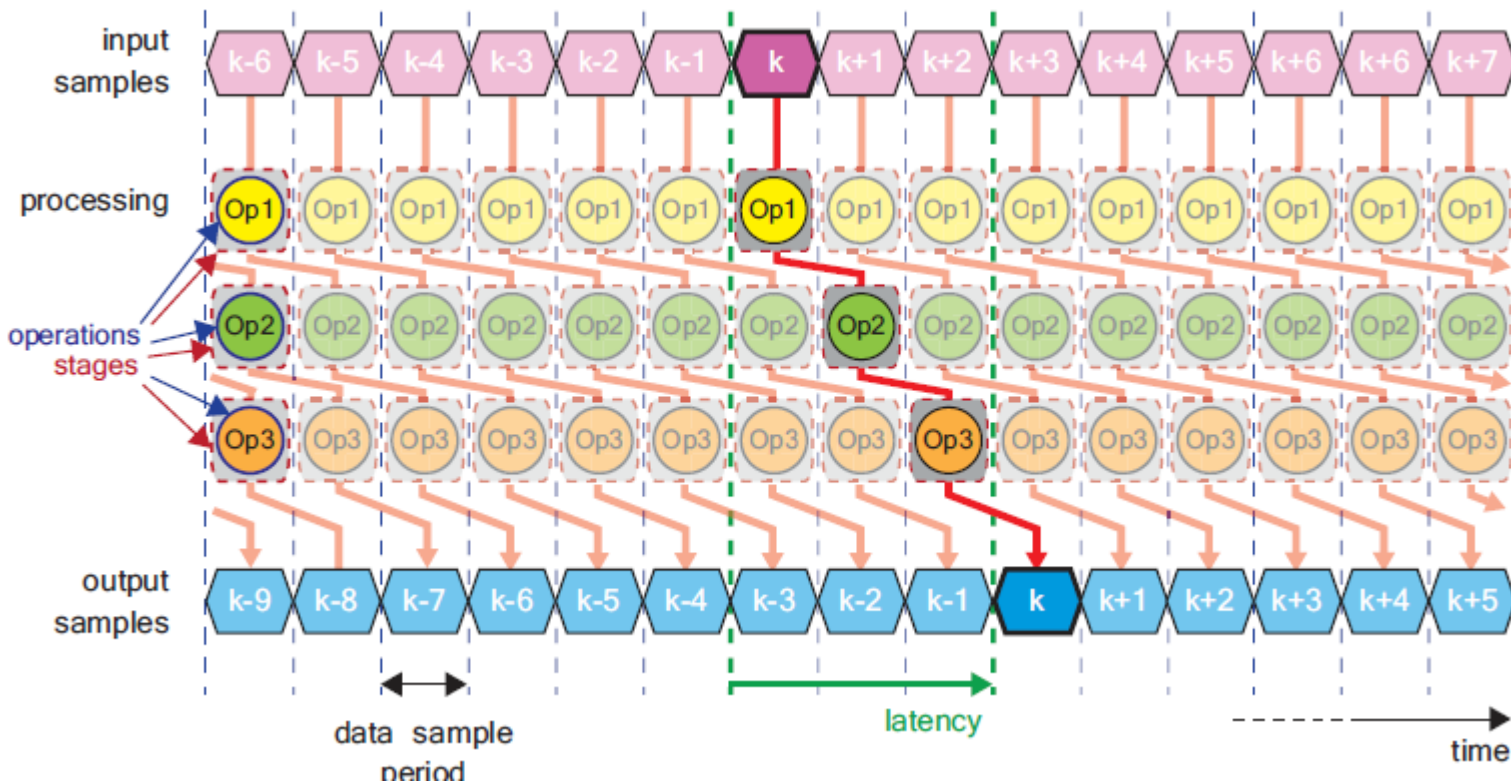
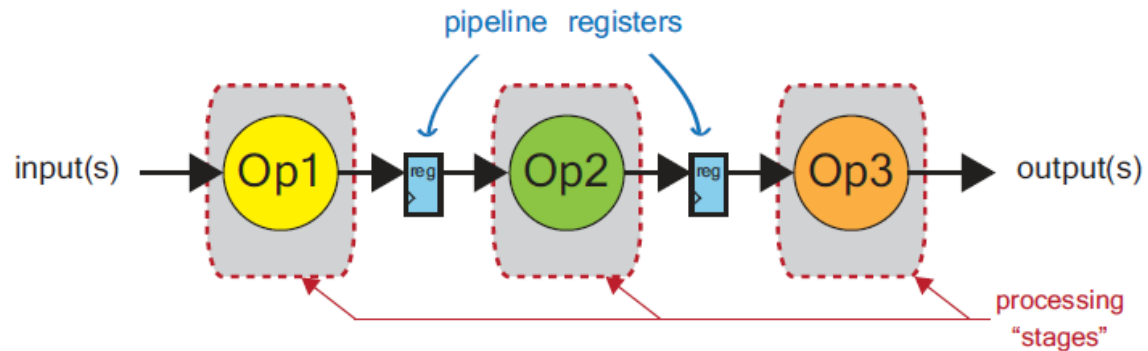
Pipelining



- If data dependency
 - $Op1 \rightarrow Op2 \rightarrow Op3$
 - Latency = $T(Op1 + Op2 + Op3)$
- Low Throughput



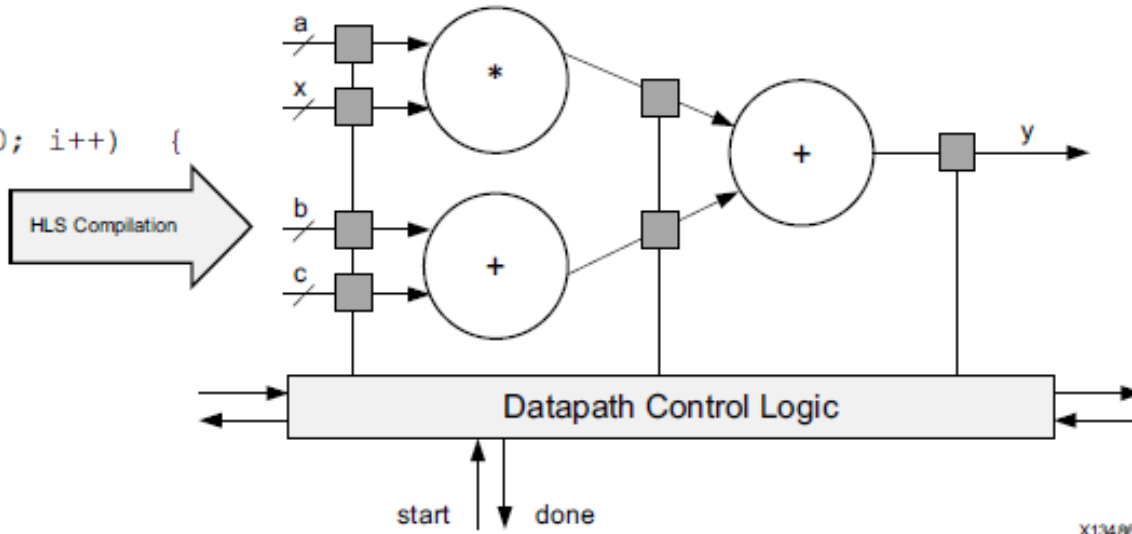
Pipelining



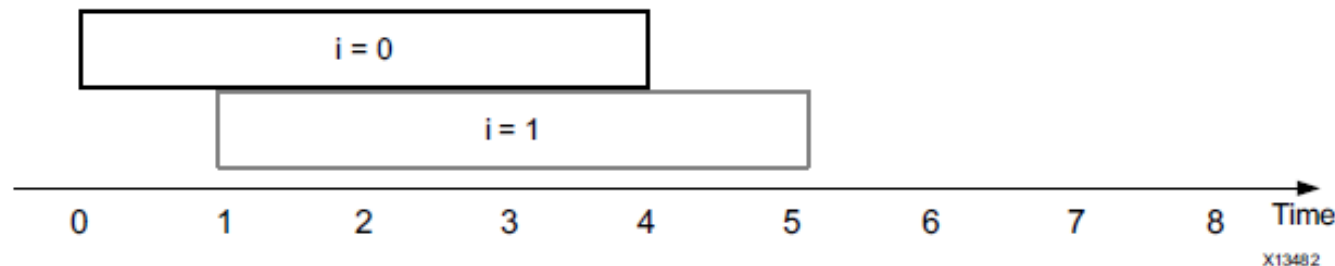
- 레지스터 추가
- 각각의 Operation 은 개별 Processing Stage
- Throughput 3 배

Control Algorithms - Loop

```
int a,b,c,x,y;  
for(int i = 0; i < 20; i++) {  
    x = get();  
    y = a*x + b + c;  
    send(y);  
}
```



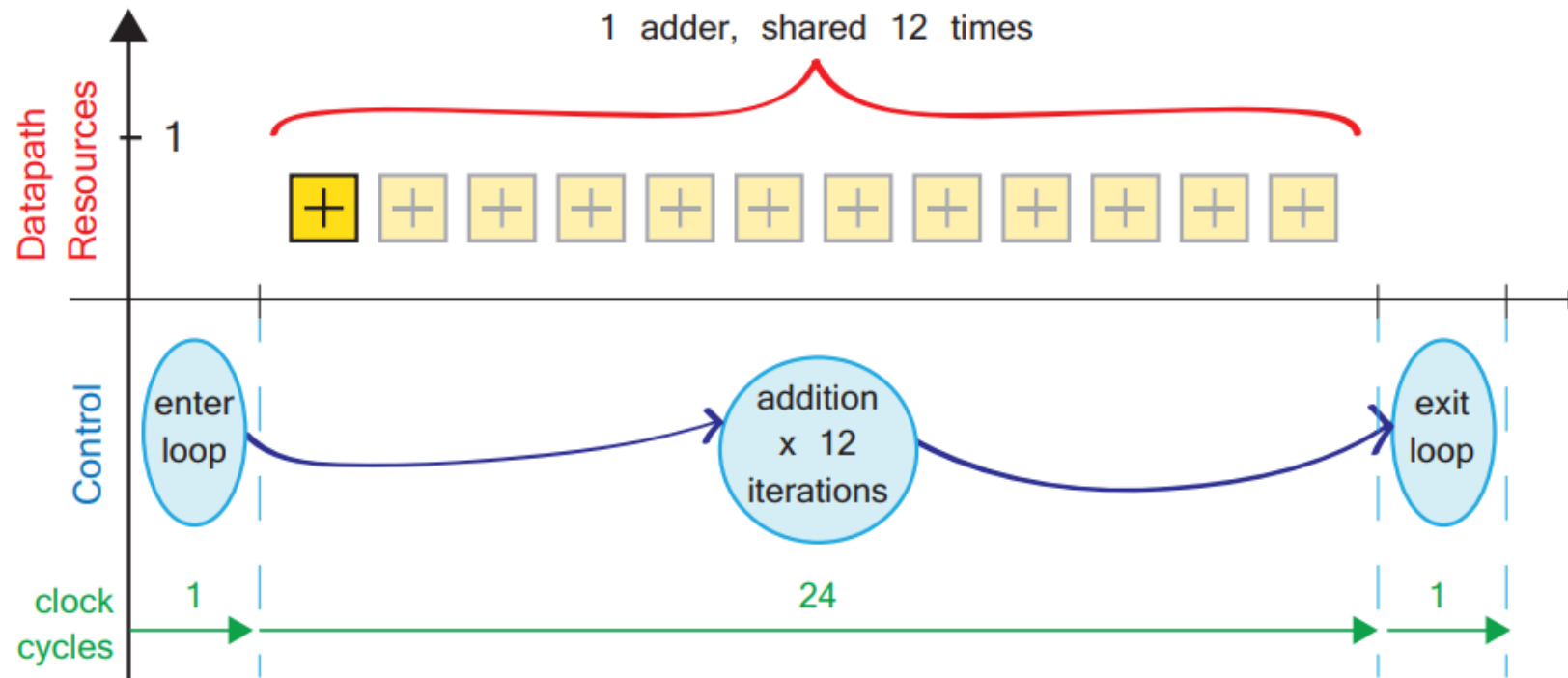
- FSM으로 구현
- Control Logic / 연산 구분
- 설정에 따라(Directives)
 - 루프 삭제(병렬 계산 처리)
 - 루프 Pipelining



Loop

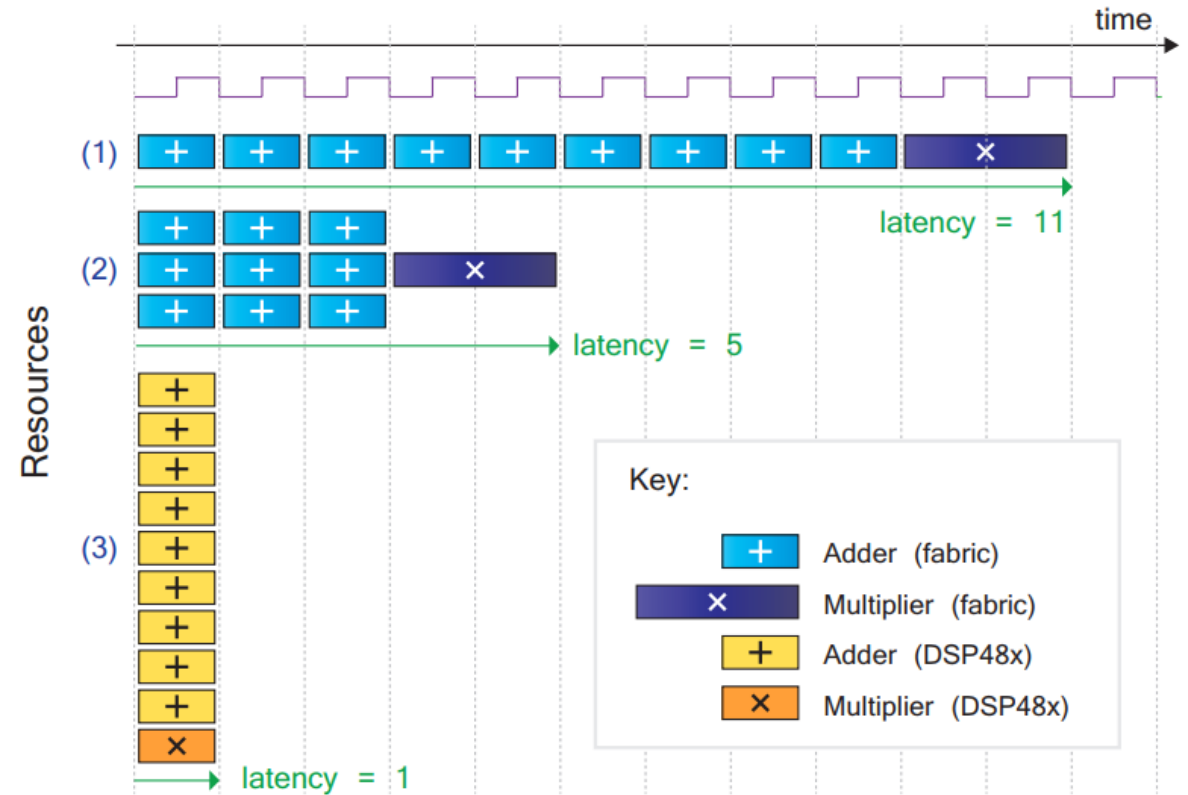
```
void add_array (short c[12], short a[12], short b[12])
{
    short j;                                // loop variable

    add_loop: for (j=0;j<12;j++) {           // loop through elements (x12)
        c[j] = a[j] + b[j];                 // addition operation
    }
}
```



Loop Optimization

- Unrolling: (1) \rightarrow (2)
- Merging
- Pipelining(Dependency related)



Loop Optimization

- Unrolling
- Merging
- Pipelining(Dependency related)

```
void add_mult (short c[12], short m[12], short a[12], short b[12])
{
    short j;

    add_loop: for (j=0;j<12;j++) {
        c[j] = a[j] + b[j];
    }

    mult_loop: for (j=0;j<12;j++) {
        m[j] = a[j] * b[j];
    }
}
```

clock cycles

FSM behaviour

```
graph TD
    enter((enter)) --> adds((adds))
    adds --> exit_enter((exit/enter))
    exit_enter --> mults((mults))
    mults --> exit((exit))
```

State	Clock Cycles
enter	1
adds	24
exit/enter	1
mults	48
exit	1

```
void add_mult (short c[12], short m[12], short a[12], short b[12])
{
    short j;

    add_mult_loop: for (j=0;j<12;j++) {
        c[j] = a[j] + b[j];
        m[j] = a[j] * b[j];
    }
}
```

clock cycles

FSM behaviour

```
graph TD
    enter((enter)) --> adds_mult((adds & mult))
    adds_mult --> exit((exit))
```

State	Clock Cycles
enter	1
adds & mult	48
exit	1

Loop Optimization

- Unrolling
- Merging
- Pipelining(Dependency related)

default

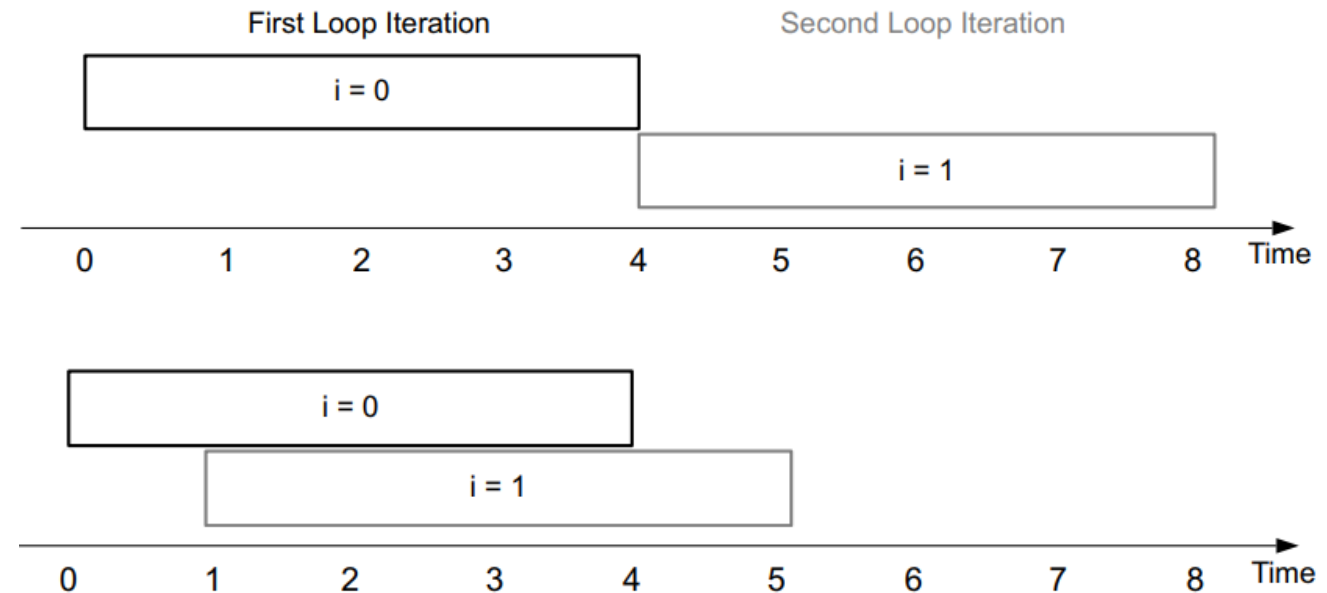
- Resource Optimized

Pipelined

- loop initialization interval (II)

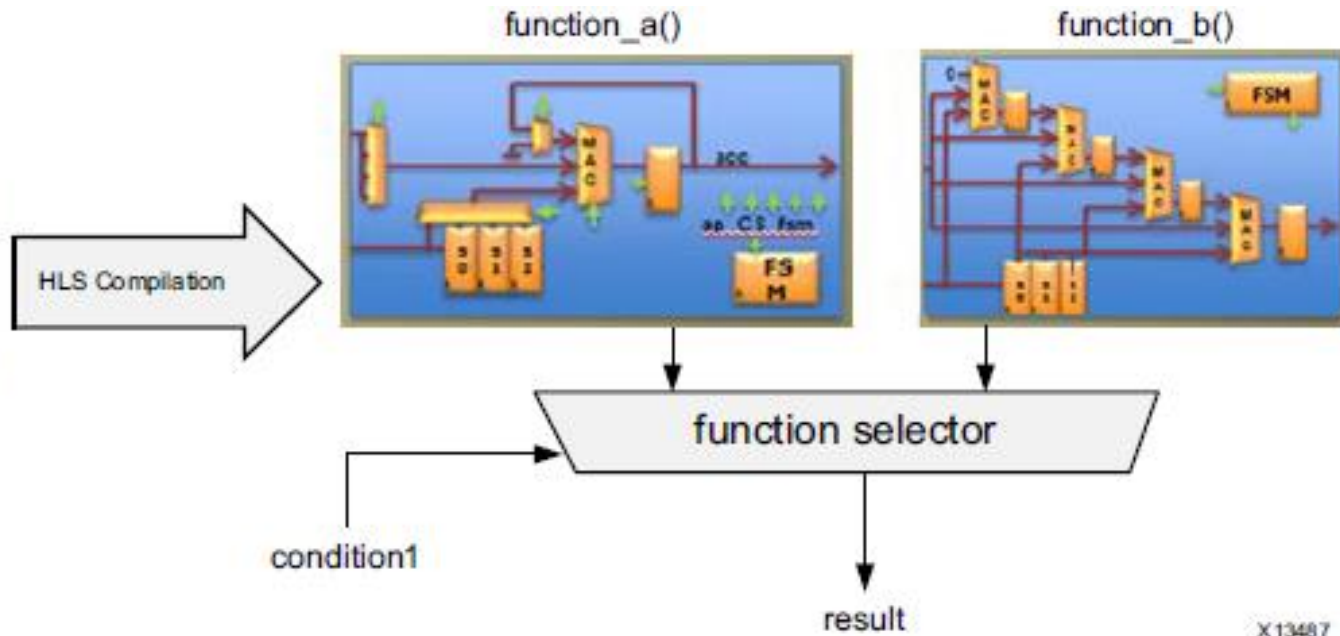
```
#pragma HLS PIPELINE II=1
```

```
for(i=0; i < 10; i++)  
{  
    A = A + (B[i] * C[i]);  
}
```



Control Algorithms - 조건문

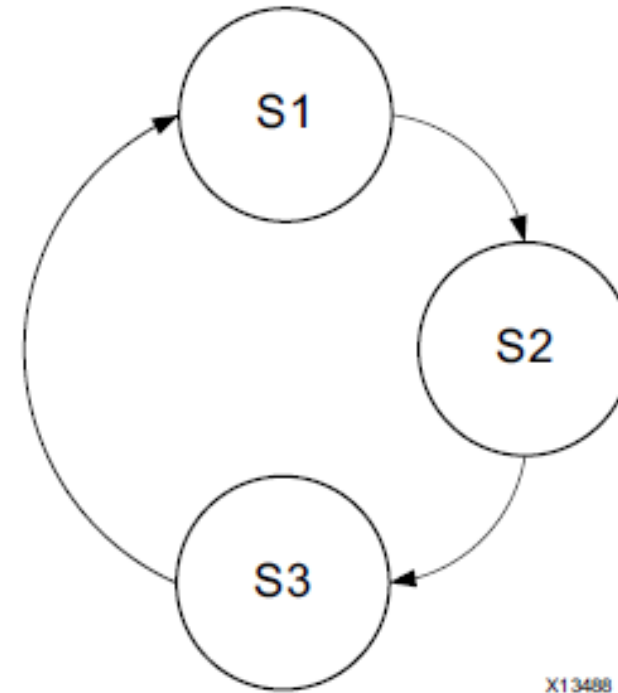
```
if (condition1) {  
    result = function_a();  
}  
else {  
    result = function_b();  
}
```



- 각 조건에 따른 함수(회로) 모두 구성
- 조건에 따라 함수 선택

Control Algorithms – Switch Case

```
switch (X) {  
  case S1: ... X = S2; break;  
  case S2: ... X = S3; break;  
  case S3: ... X = S1; break;  
}
```



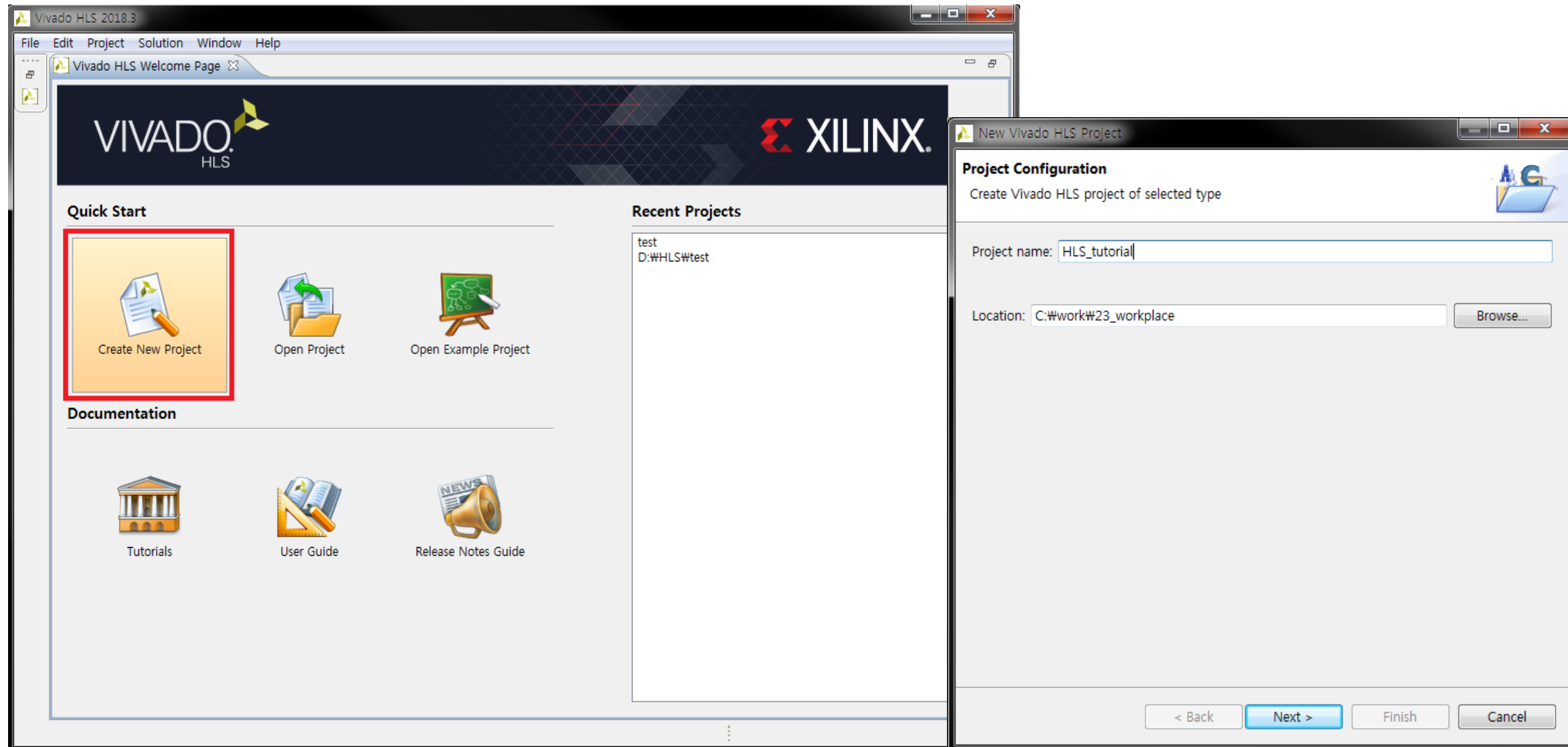
- FSM으로 구현

03. HLS를 이용한 FPGA 설계 실습

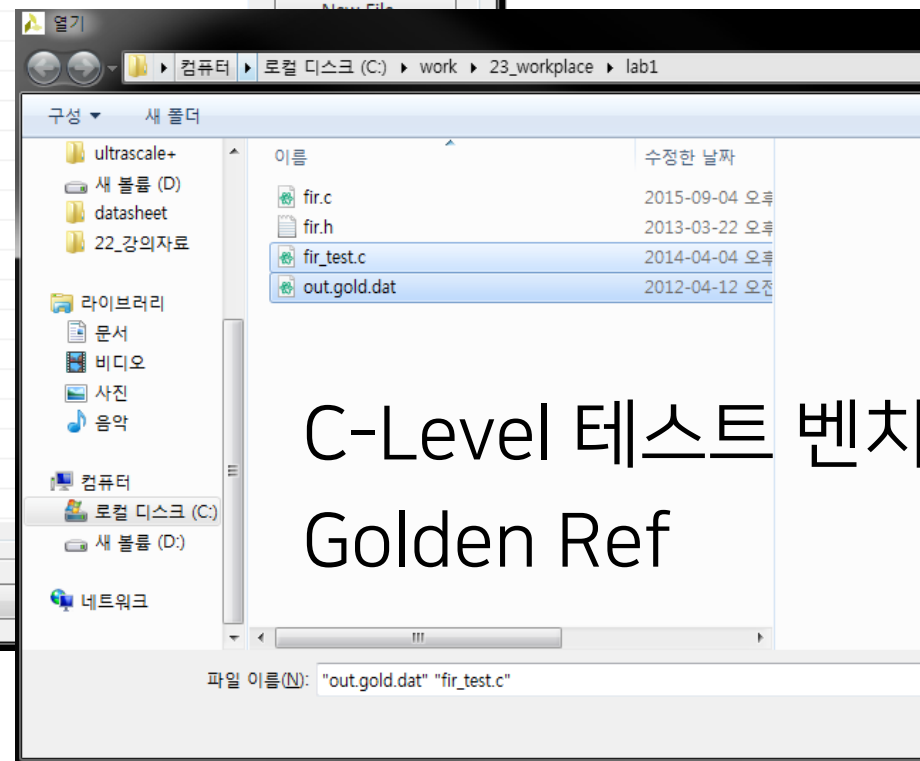
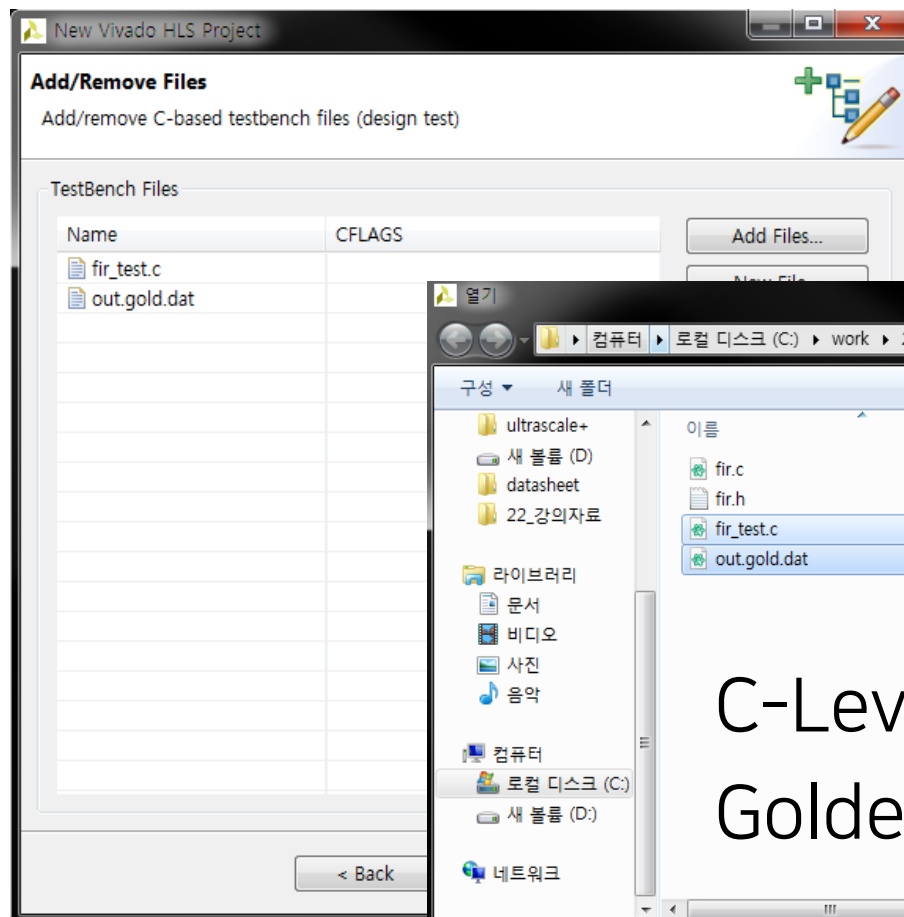
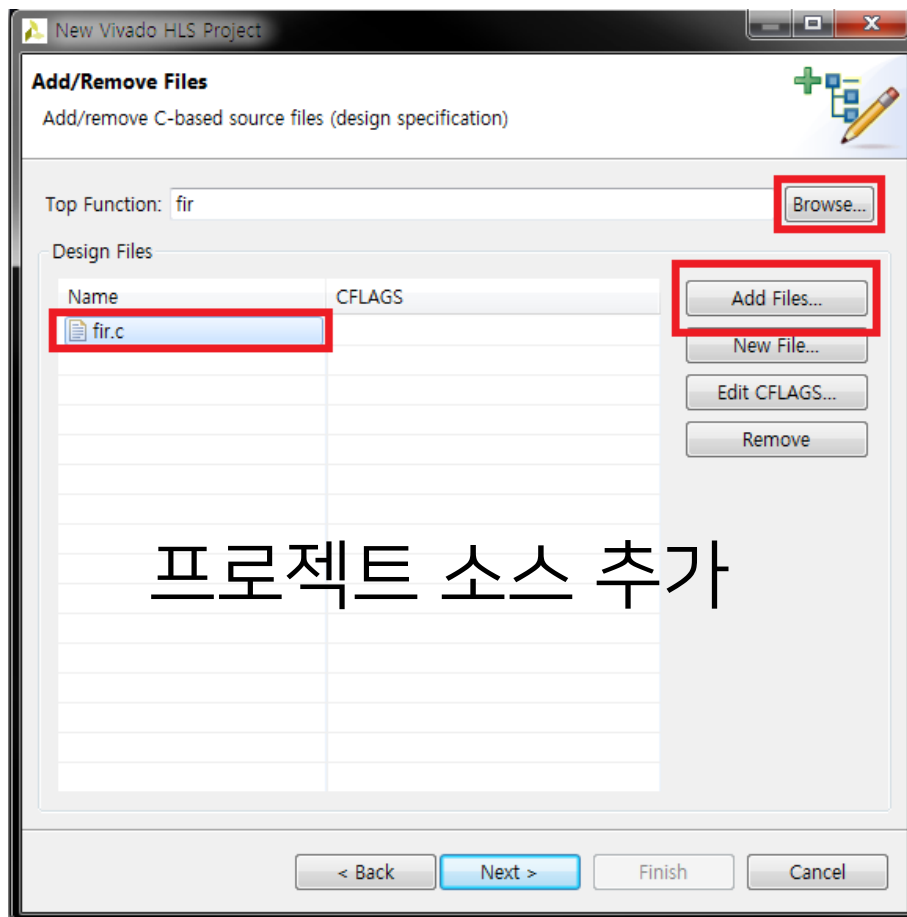
- Xilinx ug-871 Vivado High Level Synthesis Tutorial
- Source Code
- C-Level Test bench, Golden Ref
- HLS IP



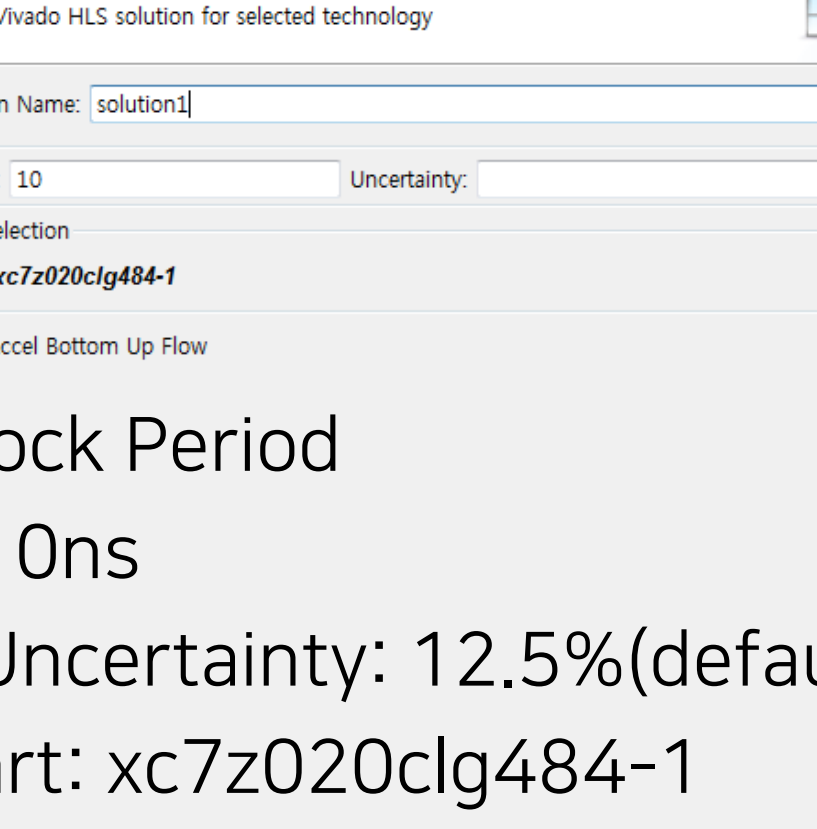
Vivado HLS Tutorial



Vivado HLS Tutorial



Vivado HLS Tutorial



New Vivado HLS Project

Solution Configuration

Create Vivado HLS solution for selected technology

Solution Name:

Clock

Period: Uncertainty:

Part Selection

Part: **xc7z020clg484-1**

☐ SDAccel Bottom Up Flow

< Back Finish Cancel

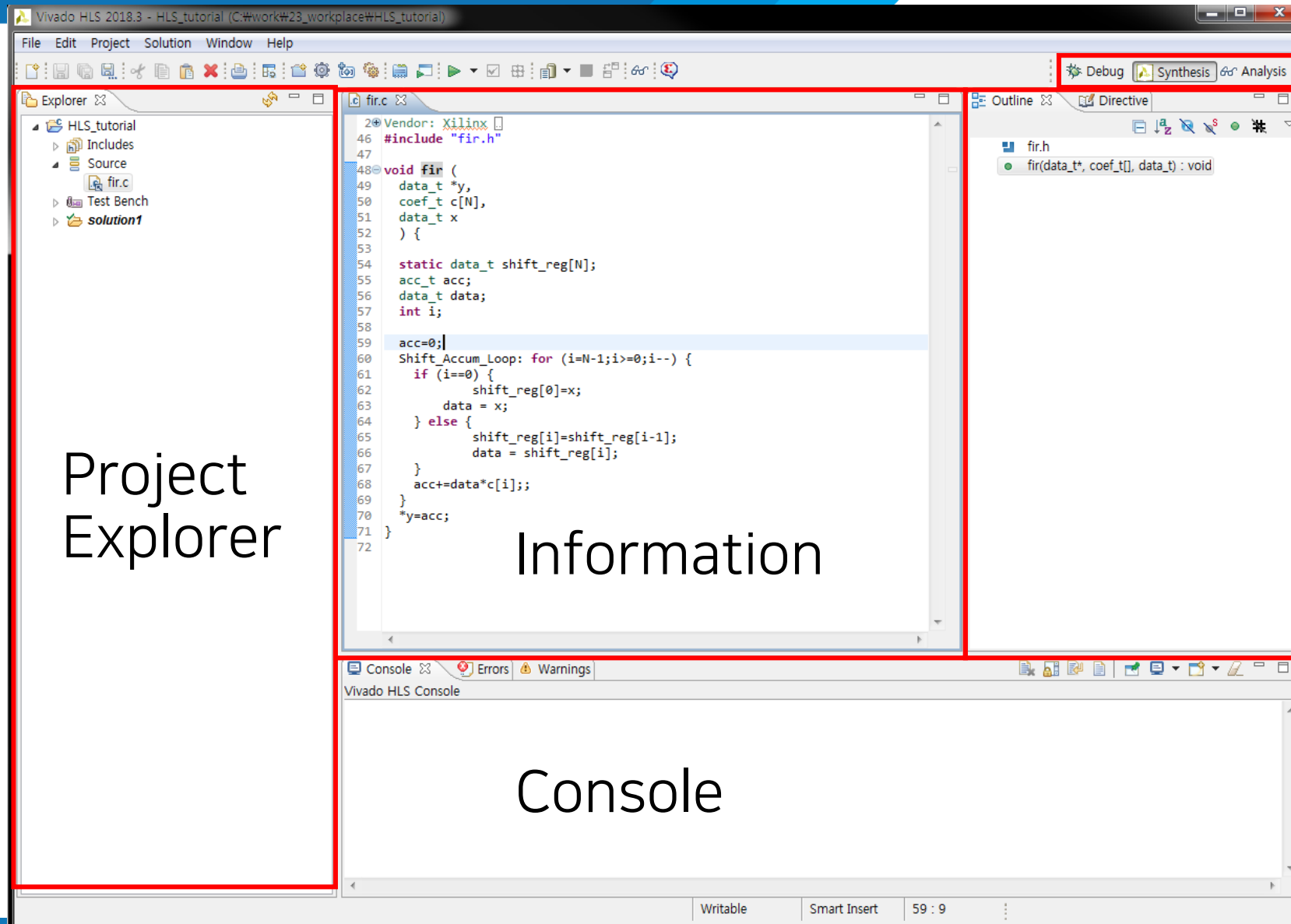
Clock Period

- 10ns
- Uncertainty: 12.5%(default)

Part: xc7z020clg484-1

[illegible]

Vivado HLS GUI



Perspectives

Project
Explorer

Information

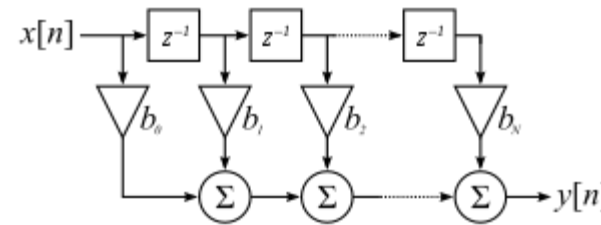
Console

fir.c(FIR 필터 함수)

```

fir.c x fir_test.c
2*Vendor: Xilinx
46 #include "fir.h"
47
48 void fir (
49     data_t *y,
50     coef_t c[N],
51     data_t x
52 ) {
53
54     static data_t shift_reg[N];
55     acc_t acc;
56     data_t data;
57     int i;
58
59     acc=0;
60     Shift_Accum_Loop: for (i=N-1;i>=0;i--) {
61         if (i==0) {
62             shift_reg[0]=x;
63             data = x;
64         } else {
65             shift_reg[i]=shift_reg[i-1];
66             data = shift_reg[i];
67         }
68         acc+=data*c[i];
69     }
70     *y=acc;
71 }
72
```

입력 신호 열 중에서 유한 샘플(N개)에 일정한 계수를 곱하여 더한 합을 말함(정보통신기술용어해설)



$$y[n] = b_0 x[n] + b_1 x[n-1] + \dots + b_N x[n-N]$$
$$= \sum_{i=0}^N b_i \cdot x[n-i],$$

data = shift_reg[10,9,8,7,..., 1], x
acc += data * c[i]

Multiply & Add for each iteration

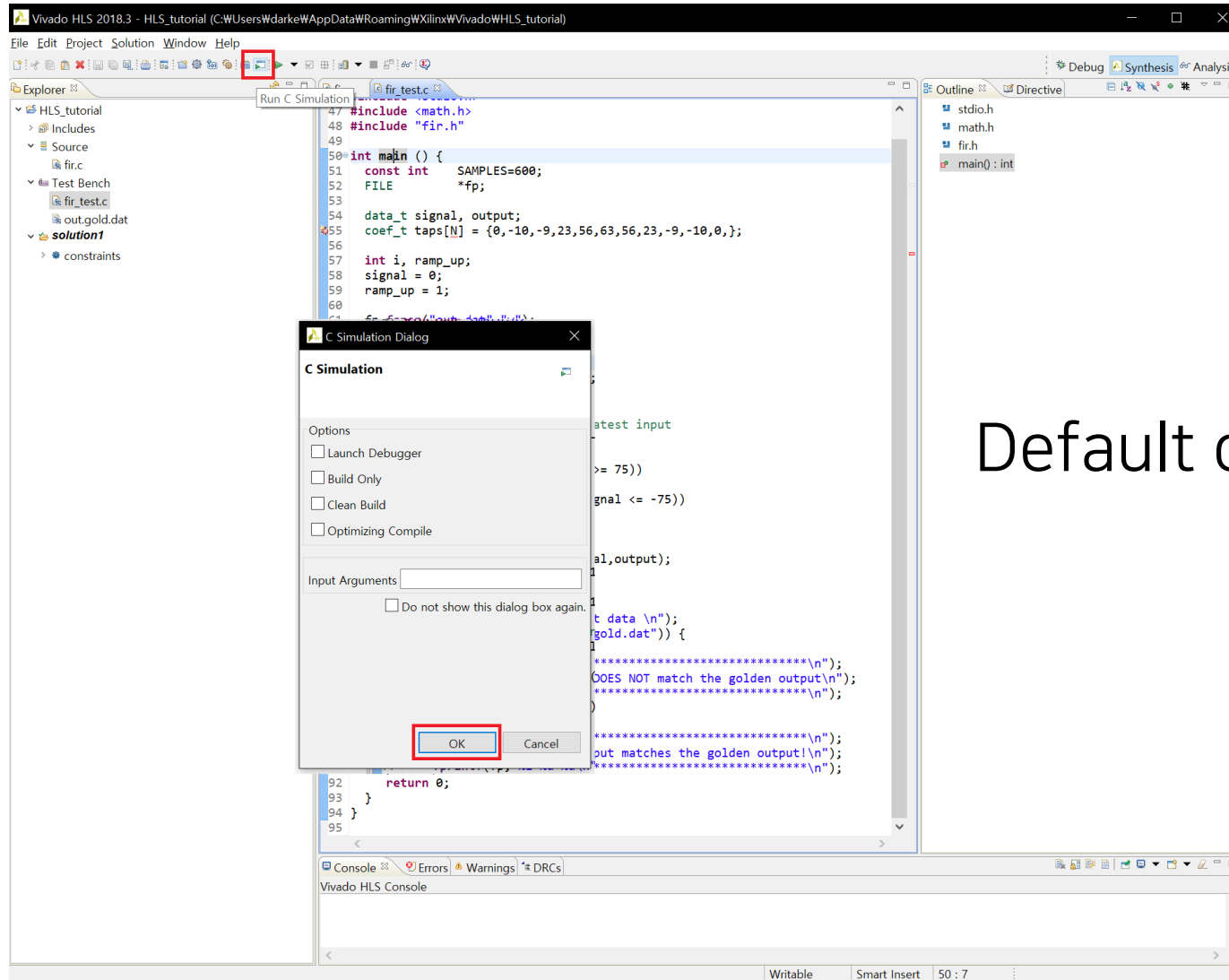
fir_test.c(test bench)

```
fir.c  fir_test.c
47 #include <math.h>
48 #include "fir.h"
49
50 int main () {
51     const int    SAMPLES=600;
52     FILE         *fp;
53
54     data_t signal, output;
55     coef_t taps[N] = {0,-10,-9,23,56,63,56,23,-9,-10,0,};
56
57     int i, ramp_up;
58     signal = 0;
59     ramp_up = 1;
60
61     fp=fopen("out.dat","w");
62     for (i=0;i<=SAMPLES;i++) {
63         if (ramp_up == 1)
64             signal = signal + 1;
65         else
66             signal = signal - 1;
67
68         // Execute the function with latest input
69         fir(&output,taps,signal);
70
71         if ((ramp_up == 1) && (signal >= 75))
72             ramp_up = 0;
73         else if ((ramp_up == 0) && (signal <= -75))
74             ramp_up = 1;
75
76         // Save the results.
77         fprintf(fp,"%i %d %d\n",i,signal,output);
78     }
79     fclose(fp);
80
81     printf ("Comparing against output data \n");
82     if (system("diff -w out.dat out.gold.dat")) {
83
84         fprintf(stdout, "*****\n");
85         fprintf(stdout, "FAIL: Output DOES NOT match the golden output\n");
86         fprintf(stdout, "*****\n");
87         return 1;
88     } else {
89         fprintf(stdout, "*****\n");
90         fprintf(stdout, "PASS: The output matches the golden output!\n");
91         fprintf(stdout, "*****\n");
92         return 0;
93     }
94 }
```

main() function

- calls fir(), saves output as out.dat
- Golden reference와 out.dat 비교
 - Match 시 0 return
 - Not match 시 1 return

Launch simulation



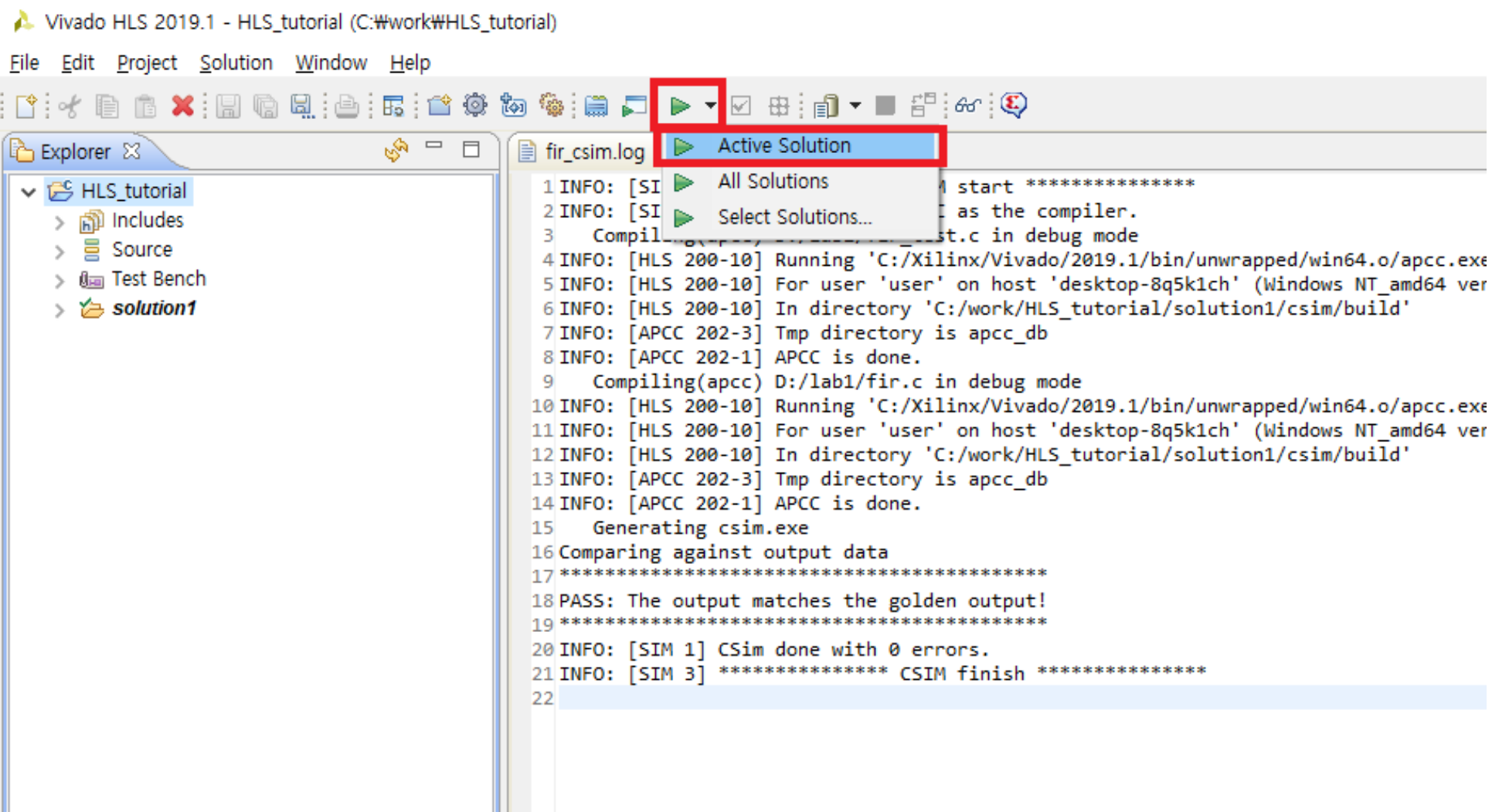
Default option으로 simulation launch

C-Simulation result

fir_csim.log

```
1 INFO: [SIM 2] ***** CSIM start *****
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3   Compiling(apcc) D:/lab1/fir_test.c in debug mode
4 INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2019.1/bin/unwrapped/win64.o/apcc.exe'
5 INFO: [HLS 200-10] For user 'user' on host 'desktop-8q5k1ch' (Windows NT_amd64 version 6.2) on Thu J
6 INFO: [HLS 200-10] In directory 'C:/work/HLS_tutorial/solution1/csim/build'
7 INFO: [APCC 202-3] Tmp directory is apcc_db
8 INFO: [APCC 202-1] APCC is done.
9   Compiling(apcc) D:/lab1/fir.c in debug mode
10 INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2019.1/bin/unwrapped/win64.o/apcc.exe'
11 INFO: [HLS 200-10] For user 'user' on host 'desktop-8q5k1ch' (Windows NT_amd64 version 6.2) on Thu J
12 INFO: [HLS 200-10] In directory 'C:/work/HLS_tutorial/solution1/csim/build'
13 INFO: [APCC 202-3] Tmp directory is apcc_db
14 INFO: [APCC 202-1] APCC is done.
15   Generating csim.exe
16 Comparing against output data
17 *****
18 PASS: The output matches the golden output!
19 *****
20 INFO: [SIM 1] CSim done with 0 errors.
21 INFO: [SIM 3] ***** CSIM finish *****
22 |
```

Synthesis to RTL



Synthesis Report

Resource

Synthesis(solution1)(fir_csynth.rpt) X

Performance Estimates

[-] Timing (ns)

[-] Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.510	1.25

[-] Latency (clock cycles)

[-] Summary

Latency		Interval		
min	max	min	max	Type
56	56	56	56	none

[-] Detail

[-] Instance

N/A

[-] Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- Shift_Accum_Loop	55	55	5	-	-	11	no

Execution Timing,
Latency

Utilization Estimates

[-] Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	85	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	0	-	64	6	0
Multiplexer	-	-	-	116	-
Register	-	-	177	-	-
Total	0	3	241	207	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0

[+] Detail

Interface

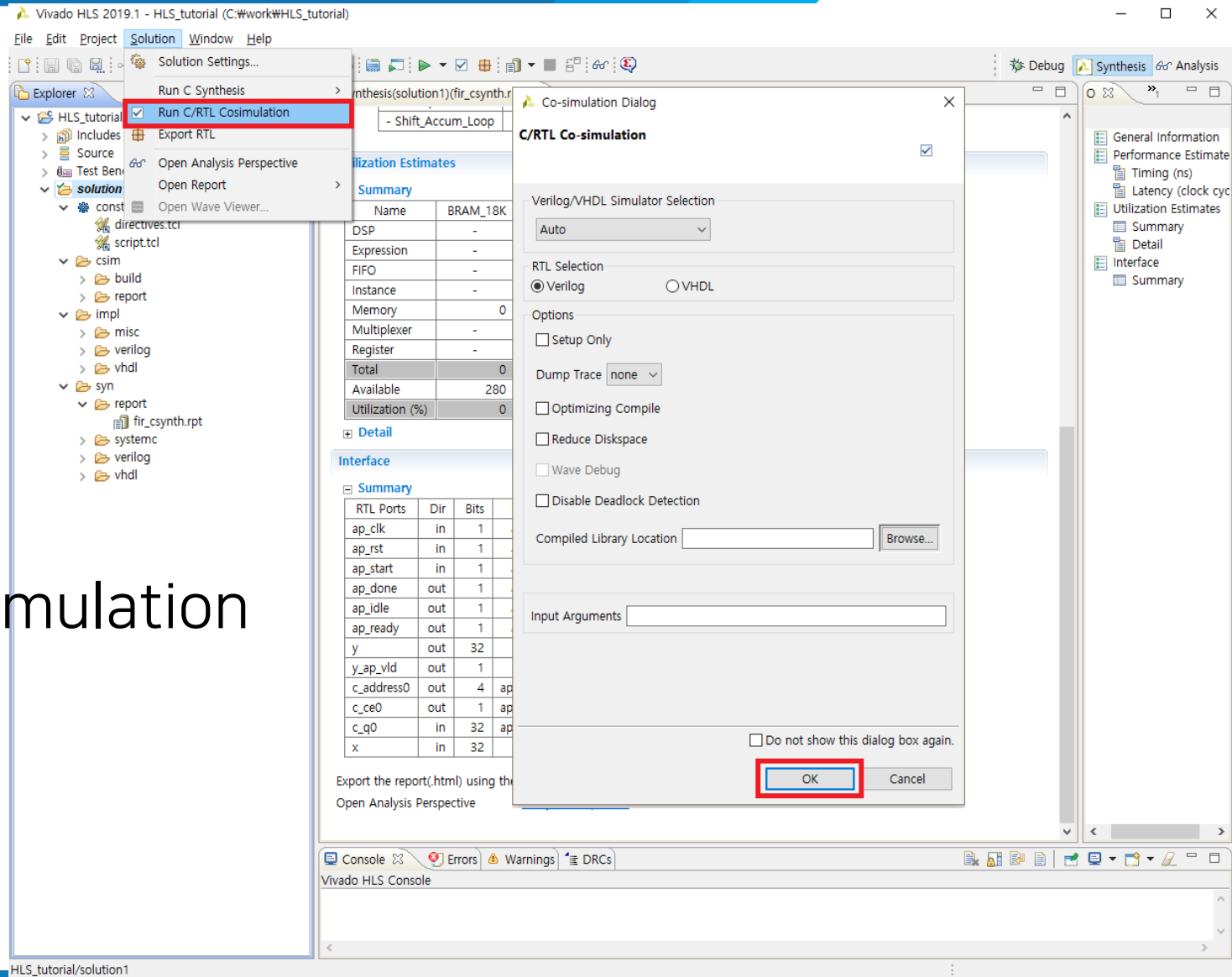
interface

[-] Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir	return value
ap_rst	in	1	ap_ctrl_hs	fir	return value
ap_start	in	1	ap_ctrl_hs	fir	return value
ap_done	out	1	ap_ctrl_hs	fir	return value
ap_idle	out	1	ap_ctrl_hs	fir	return value
ap_ready	out	1	ap_ctrl_hs	fir	return value
y	out	32	ap_vld	y	pointer
y_ap_vld	out	1	ap_vld	y	pointer
c_address0	out	4	ap_memory	c	array
c_ce0	out	1	ap_memory	c	array
c_q0	in	32	ap_memory	c	array
x	in	32	ap_none	x	scalar

RTL Verification

C/RTL Co-simulation



Co-Simulation Result

Synthesis(solution1)(fir_csynth.rpt) Simulation(solution1)(fir_csim.rpt)

Cosimulation Report for 'fir'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	56	56	56	57	57	57

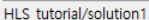
Export the report(.html) using the [Export Wizard](#)

C Test Bench

1. Creates Input Vector
2. Simulates RTL Design
3. Check RTL Output

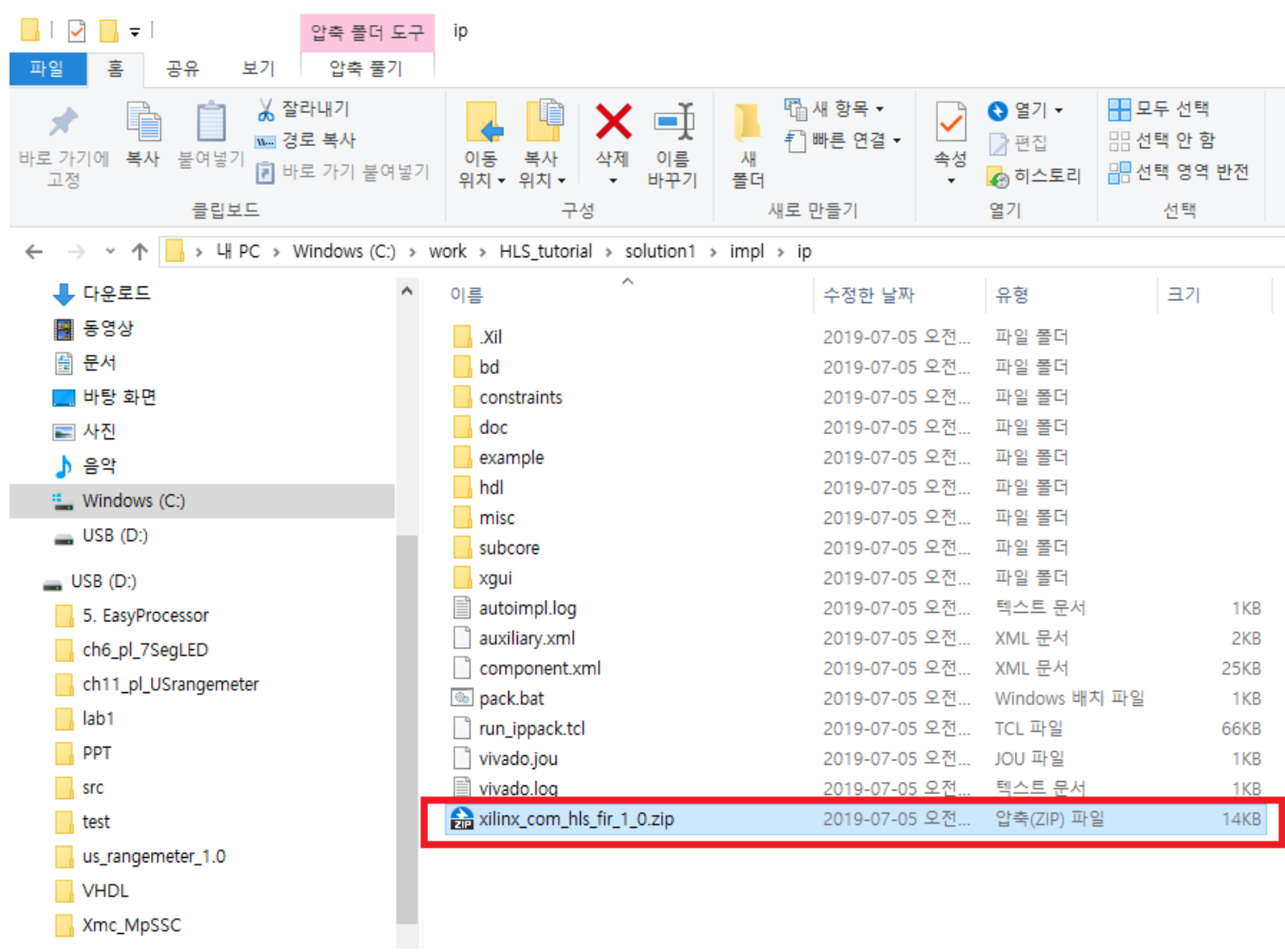
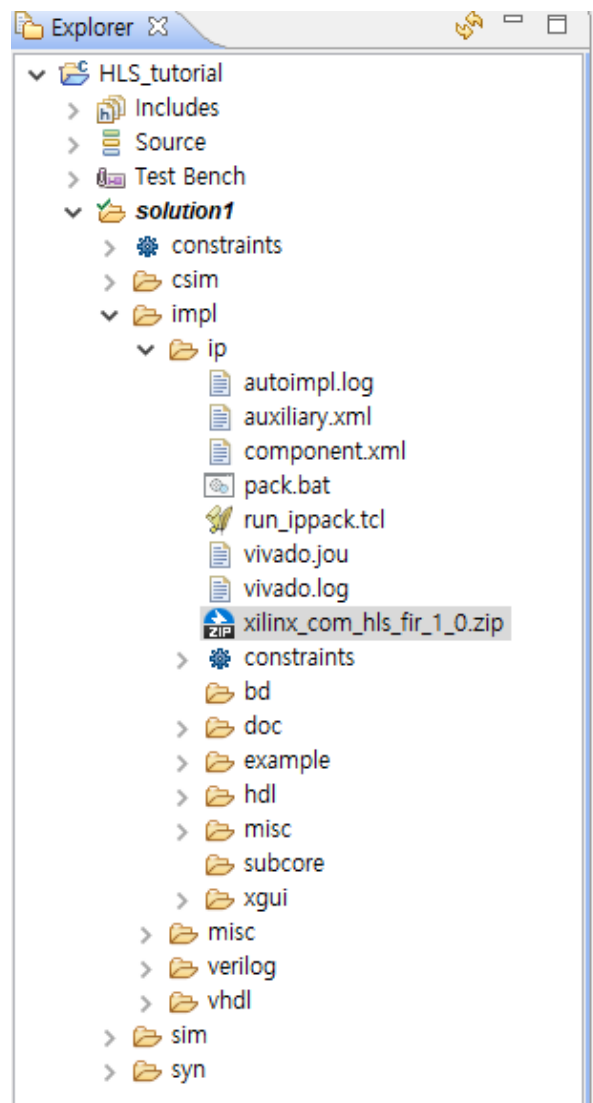
```
Console Errors Warnings DRCs
Vivado HLS Console
// RTL Simulation : 577 / 601 [100.00%] @ "329025000"
// RTL Simulation : 578 / 601 [100.00%] @ "329595000"
// RTL Simulation : 579 / 601 [100.00%] @ "330165000"
// RTL Simulation : 580 / 601 [100.00%] @ "330735000"
// RTL Simulation : 581 / 601 [100.00%] @ "331305000"
// RTL Simulation : 582 / 601 [100.00%] @ "331875000"
// RTL Simulation : 583 / 601 [100.00%] @ "332445000"
// RTL Simulation : 584 / 601 [100.00%] @ "333015000"
// RTL Simulation : 585 / 601 [100.00%] @ "333585000"
// RTL Simulation : 586 / 601 [100.00%] @ "334155000"
// RTL Simulation : 587 / 601 [100.00%] @ "334725000"
// RTL Simulation : 588 / 601 [100.00%] @ "335295000"
// RTL Simulation : 589 / 601 [100.00%] @ "335865000"
// RTL Simulation : 590 / 601 [100.00%] @ "336435000"
// RTL Simulation : 591 / 601 [100.00%] @ "337005000"
// RTL Simulation : 592 / 601 [100.00%] @ "337575000"
// RTL Simulation : 593 / 601 [100.00%] @ "338145000"
// RTL Simulation : 594 / 601 [100.00%] @ "338715000"
// RTL Simulation : 595 / 601 [100.00%] @ "339285000"
// RTL Simulation : 596 / 601 [100.00%] @ "339855000"
// RTL Simulation : 597 / 601 [100.00%] @ "340425000"
// RTL Simulation : 598 / 601 [100.00%] @ "340995000"
// RTL Simulation : 599 / 601 [100.00%] @ "341565000"
// RTL Simulation : 600 / 601 [100.00%] @ "342135000"
// RTL Simulation : 601 / 601 [100.00%] @ "342705000"
////////////////////////////////////
$finish called at time : 342745 ns : File "C:/work/HLS_tutorial/solution1/sim/verilog/fir.autotb.v" Line 332
## quit
INFO: [Common 17-206] Exiting xsim at Fri Jul 5 05:46:01 2019...
INFO: [COSIM 212-316] Starting C post checking ...
Comparing against output data
*****
PASS: The output matches the golden output!
*****
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
Finished C/RTL cosimulation.
```

52

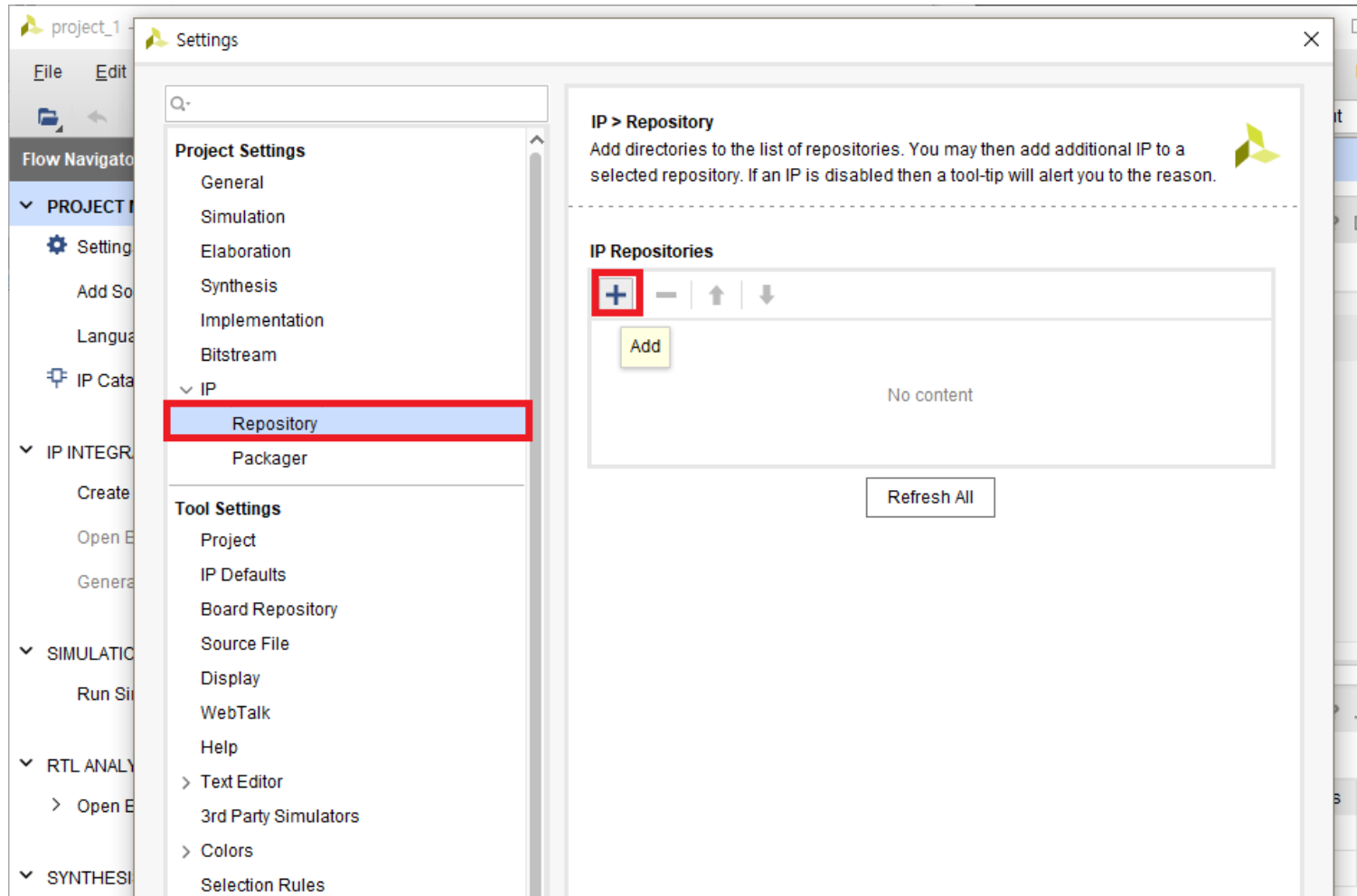


HUINS
Human Intelligent System

Exported IP



Use HLS in Vivado



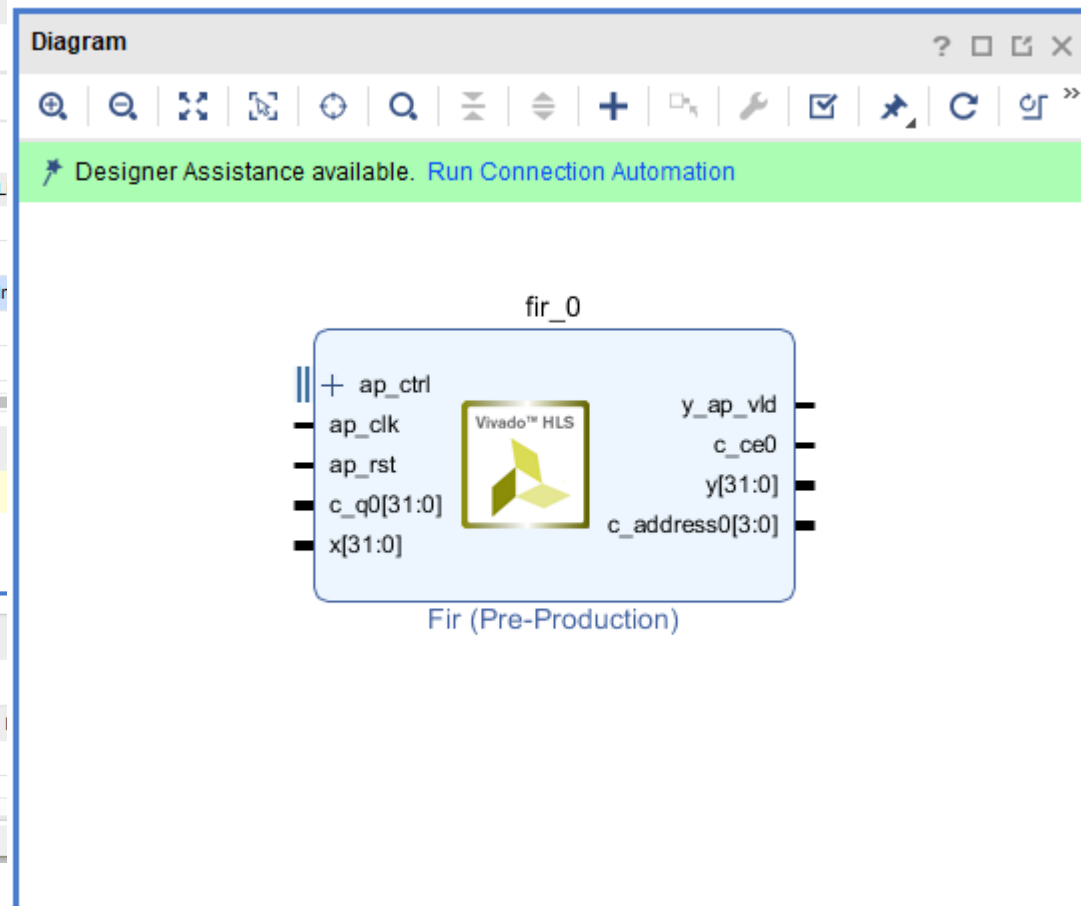
Unzip HLS IP
Add IP Repository

Import in Vivado

The screenshot displays the Vivado 2019.1 software interface. The top menu bar includes File, Edit, Flow, Tools, Reports, Window, Layout, View, and Help. Below the menu is a toolbar with various icons. The main workspace is divided into several panes:

- Flow Navigator:** Located on the left, it shows the project hierarchy with sections for PROJECT MANAGER, IP INTEGRATOR, SIMULATION, RTL ANALYSIS, SYNTHESIS, and IMPLEMENTATION.
- PROJECT MANAGER - project_1:** This pane is active and shows the project's sources and properties. The 'Sources' tab is selected, displaying a tree view of Design Sources, Constraints, Simulation Sources (sim_1), and Utility Sources. The 'IP Properties' tab is also visible, showing details for the 'Fir' IP, including its version (1.0), description, status (Pre-Production), and license (Included).
- IP Catalog:** This pane is open, showing a search bar and a list of IP cores. The 'Fir' core is highlighted, and its details are shown in the 'Details' tab, including a warning that 'IP has duplicates'.
- Design Runs:** At the bottom, a table shows the status of design runs. The 'synth_1' run is listed with a status of 'Not started'.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
synth_1	constrs_1	Not started									
impl_1	constrs_1	Not started									



Synthesis Report

Resource

Synthesis(solution1)(fir_csynth.rpt) X

Performance Estimates

[-] Timing (ns)

[-] Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.510	1.25

[-] Latency (clock cycles)

[-] Summary

Latency		Interval		
min	max	min	max	Type
56	56	56	56	none

[-] Detail

[-] Instance

N/A

[-] Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- Shift_Accum_Loop	55	55	5	-	-	11	no

Execution Timing,
Latency

Utilization Estimates

[-] Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	85	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	0	-	64	6	0
Multiplexer	-	-	-	116	-
Register	-	-	177	-	-
Total	0	3	241	207	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0

[+] Detail

Interface

interface

[-] Summary

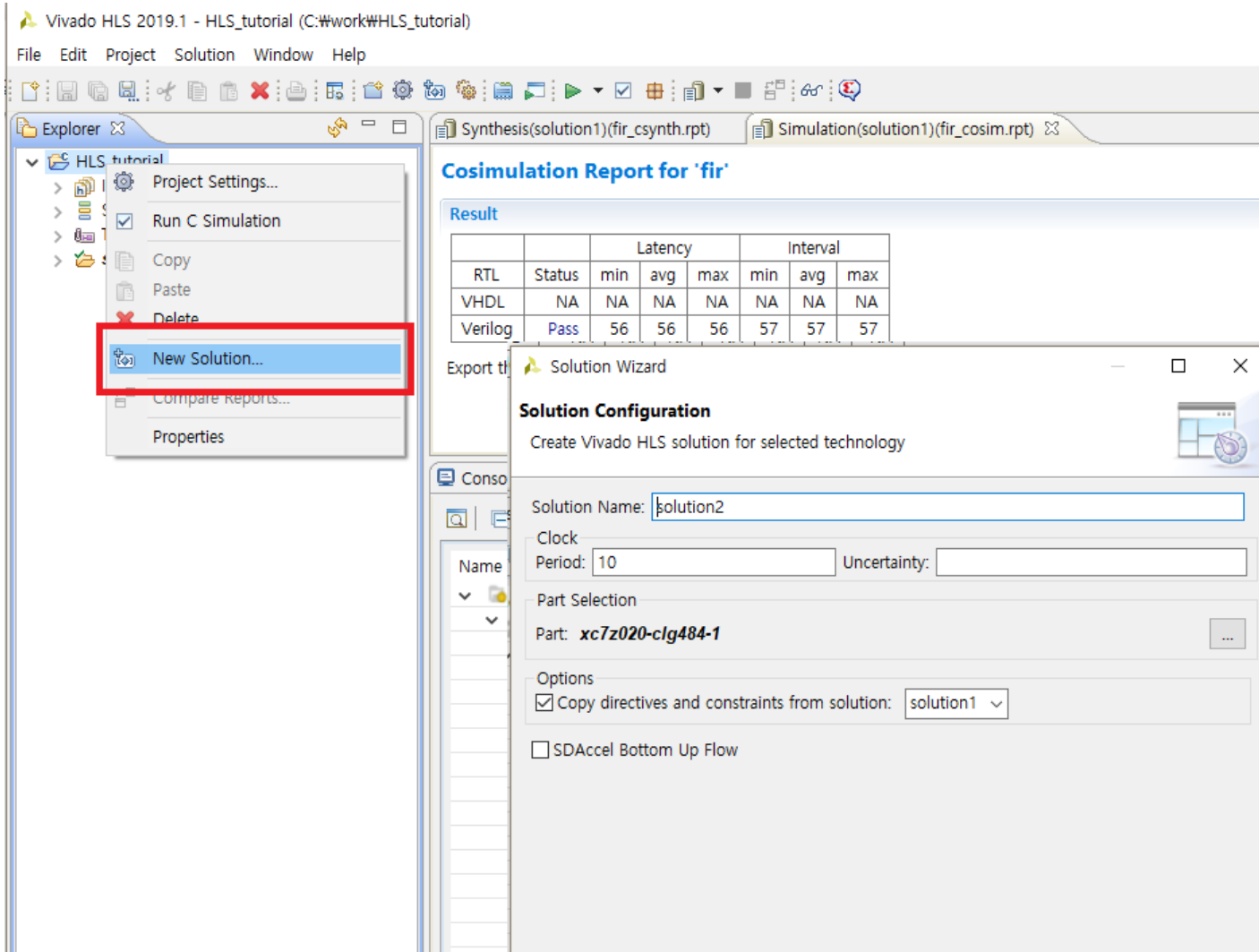
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir	return value
ap_rst	in	1	ap_ctrl_hs	fir	return value
ap_start	in	1	ap_ctrl_hs	fir	return value
ap_done	out	1	ap_ctrl_hs	fir	return value
ap_idle	out	1	ap_ctrl_hs	fir	return value
ap_ready	out	1	ap_ctrl_hs	fir	return value
y	out	32	ap_vld	y	pointer
y_ap_vld	out	1	ap_vld	y	pointer
c_address0	out	4	ap_memory	c	array
c_ce0	out	1	ap_memory	c	array
c_q0	in	32	ap_memory	c	array
x	in	32	ap_none	x	scalar

Optimize Port to

Target

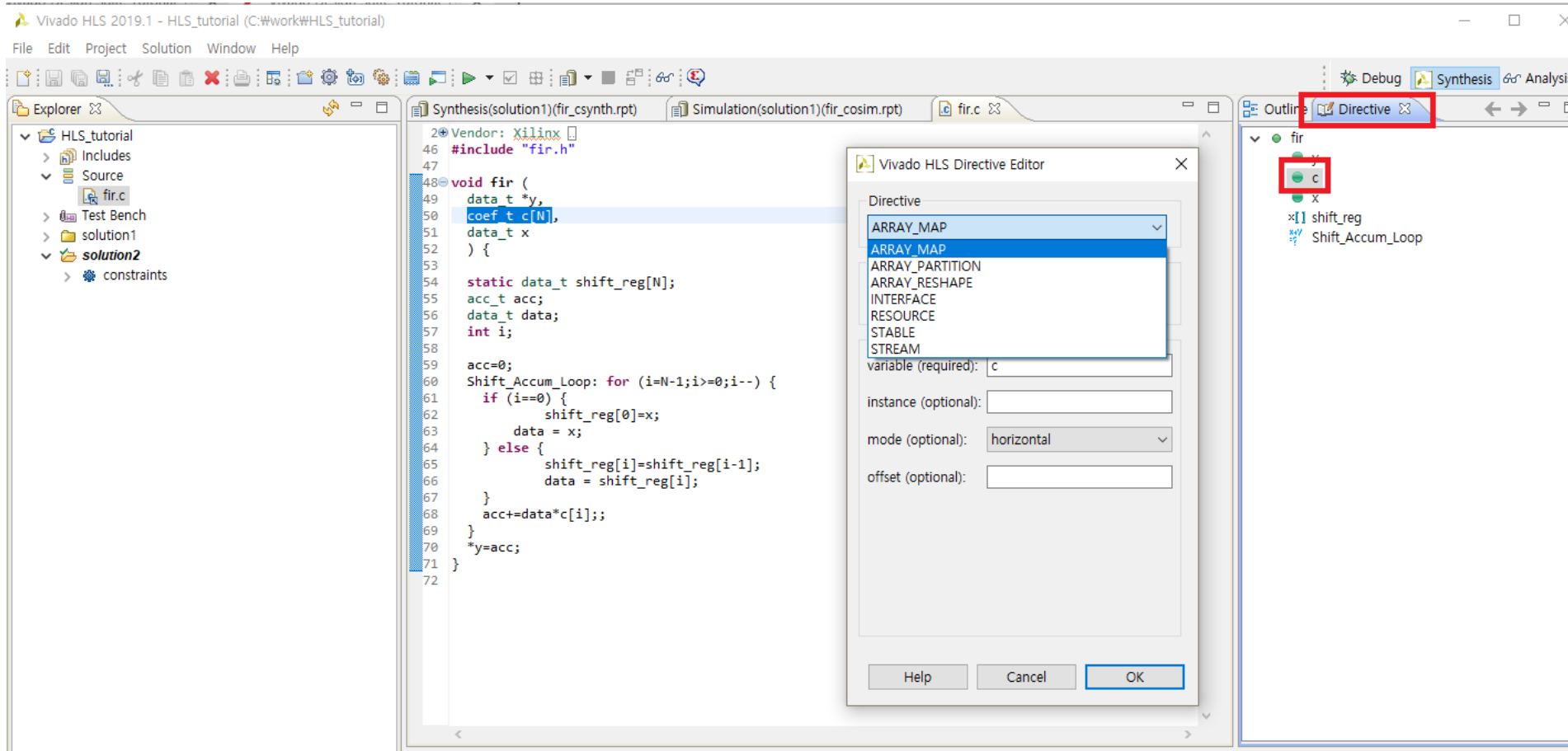
- Port C(input array): single port RAM access
- Port X(input data): input data valid signal
- Port Y(output data): output data valid signal

Create New Solution

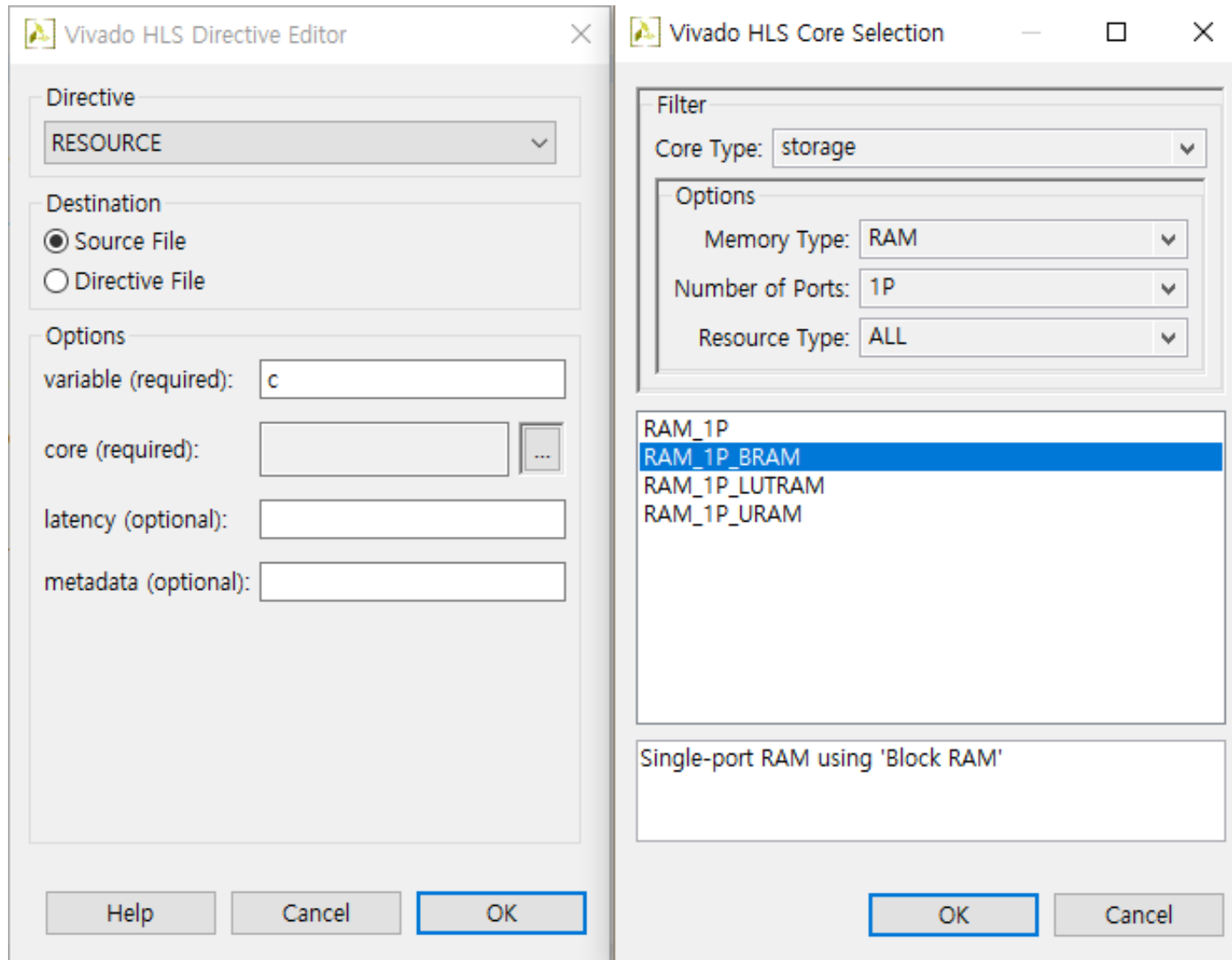


최적화를 통한 성능 향상이
목표이므로, 다른 것 변경하
지 않고 새 솔루션 생성

Directives



Directives for Port C



The image shows two overlapping windows from the Vivado HLS tool. The 'Vivado HLS Directive Editor' window on the left has 'Directive' set to 'RESOURCE', 'Destination' set to 'Source File', and 'variable (required)' set to 'c'. The 'Vivado HLS Core Selection' window on the right has 'Filter' set to 'storage', 'Memory Type' set to 'RAM', 'Number of Ports' set to '1P', and 'Resource Type' set to 'ALL'. A list of RAM options is shown with 'RAM_1P_BRAM' selected. Below the list, it says 'Single-port RAM using 'Block RAM''.

Vivado HLS Directive Editor

Directive: RESOURCE

Destination: ☒ Source File ☐ Directive File

Options:

variable (required): c

core (required): ...

latency (optional):

metadata (optional):

Buttons: Help, Cancel, OK

Vivado HLS Core Selection

Filter: Core Type: storage

Options:

Memory Type: RAM

Number of Ports: 1P

Resource Type: ALL

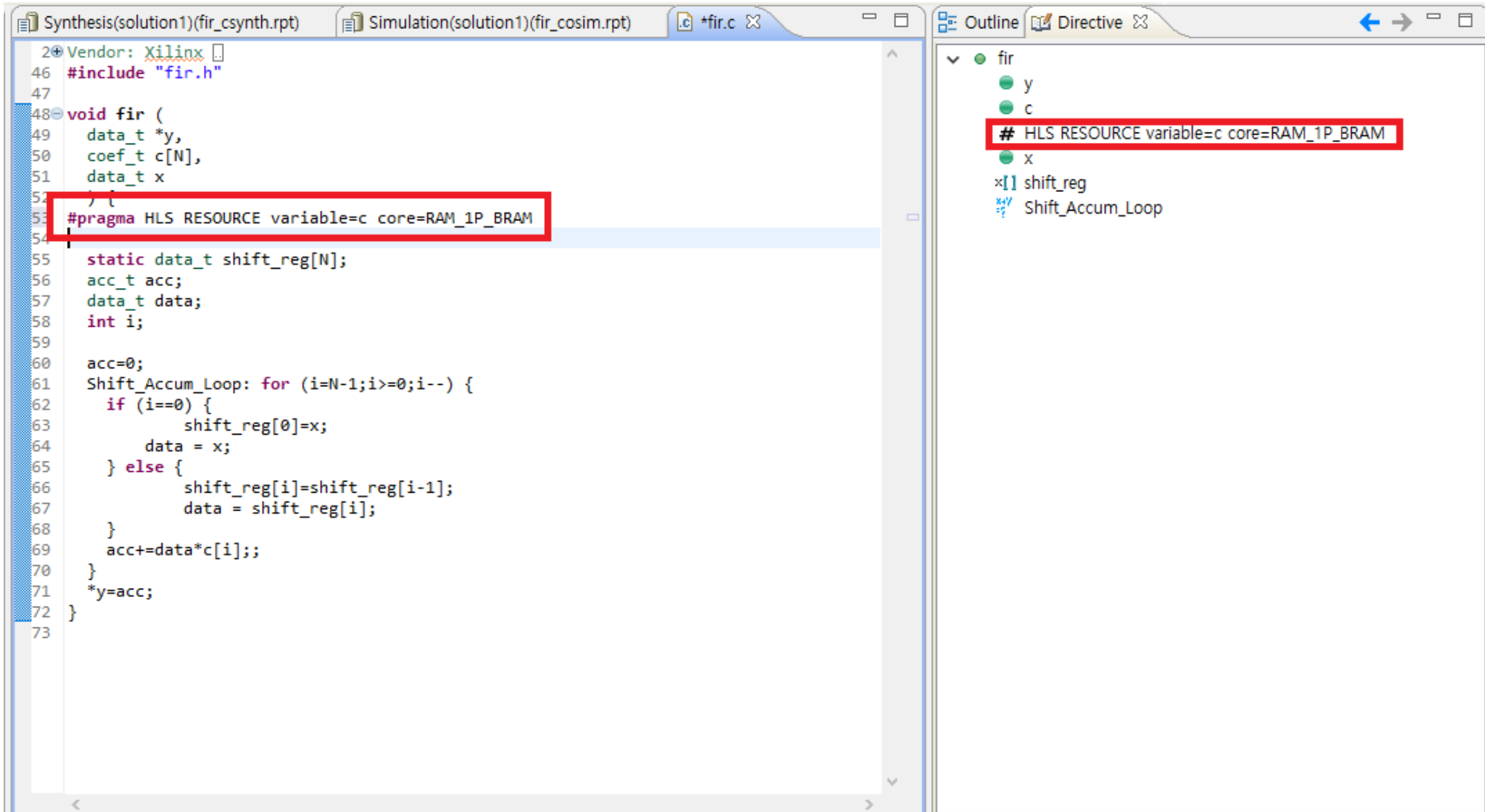
RAM_1P
RAM_1P_BRAM
RAM_1P_LUTRAM
RAM_1P_URAM

Single-port RAM using 'Block RAM'

Buttons: OK, Cancel

Port C: Single Port RAM

Directives as #pragma



The screenshot displays a code editor with two tabs: 'Synthesis(solution1)(fir_csynth.rpt)' and 'Simulation(solution1)(fir_cosim.rpt)'. The active file is '*fir.c'. The code defines a function 'fir' that takes a pointer to a data type 'data_t' and a coefficient array 'c[N]'. A red box highlights the pragma directive: `#pragma HLS RESOURCE variable=c core=RAM_1P_BRAM`. The function body includes static variables for a shift register, accumulator, and data, and a loop labeled 'Shift_Accum_Loop' that processes the input data and coefficients. The right-hand pane shows the 'Outline' view of the function 'fir', listing its parameters (y, c, x), a shift register array, and the 'Shift_Accum_Loop' block. A red box in the Outline pane highlights the pragma directive: `# HLS RESOURCE variable=c core=RAM_1P_BRAM`.

```
2 Vendor: Xilinx
46 #include "fir.h"
47
48 void fir (
49     data_t *y,
50     coef_t c[N],
51     data_t x
52 ) {
53     #pragma HLS RESOURCE variable=c core=RAM_1P_BRAM
54
55     static data_t shift_reg[N];
56     acc_t acc;
57     data_t data;
58     int i;
59
60     acc=0;
61     Shift_Accum_Loop: for (i=N-1;i>=0;i--) {
62         if (i==0) {
63             shift_reg[0]=x;
64             data = x;
65         } else {
66             shift_reg[i]=shift_reg[i-1];
67             data = shift_reg[i];
68         }
69         acc+=data*c[i];
70     }
71     *y=acc;
72 }
73
```

Outline

- fir
 - y
 - c
 - # HLS RESOURCE variable=c core=RAM_1P_BRAM
 - x
 - ×[1] shift_reg
 - Shift_Accum_Loop

Directives for X, Y

The image displays two instances of the Vivado HLS Directive Editor dialog box, illustrating the configuration for the 'INTERFACE' directive.

Left Dialog (Port Y):

- Directive: INTERFACE
- Destination: ☒ Source File
- Options:
 - mode (optional): ap_vld
 - register (optional): ☐
 - depth (optional):
 - latency (optional):
 - port (required): y
 - name (optional):

Right Dialog (Port X):

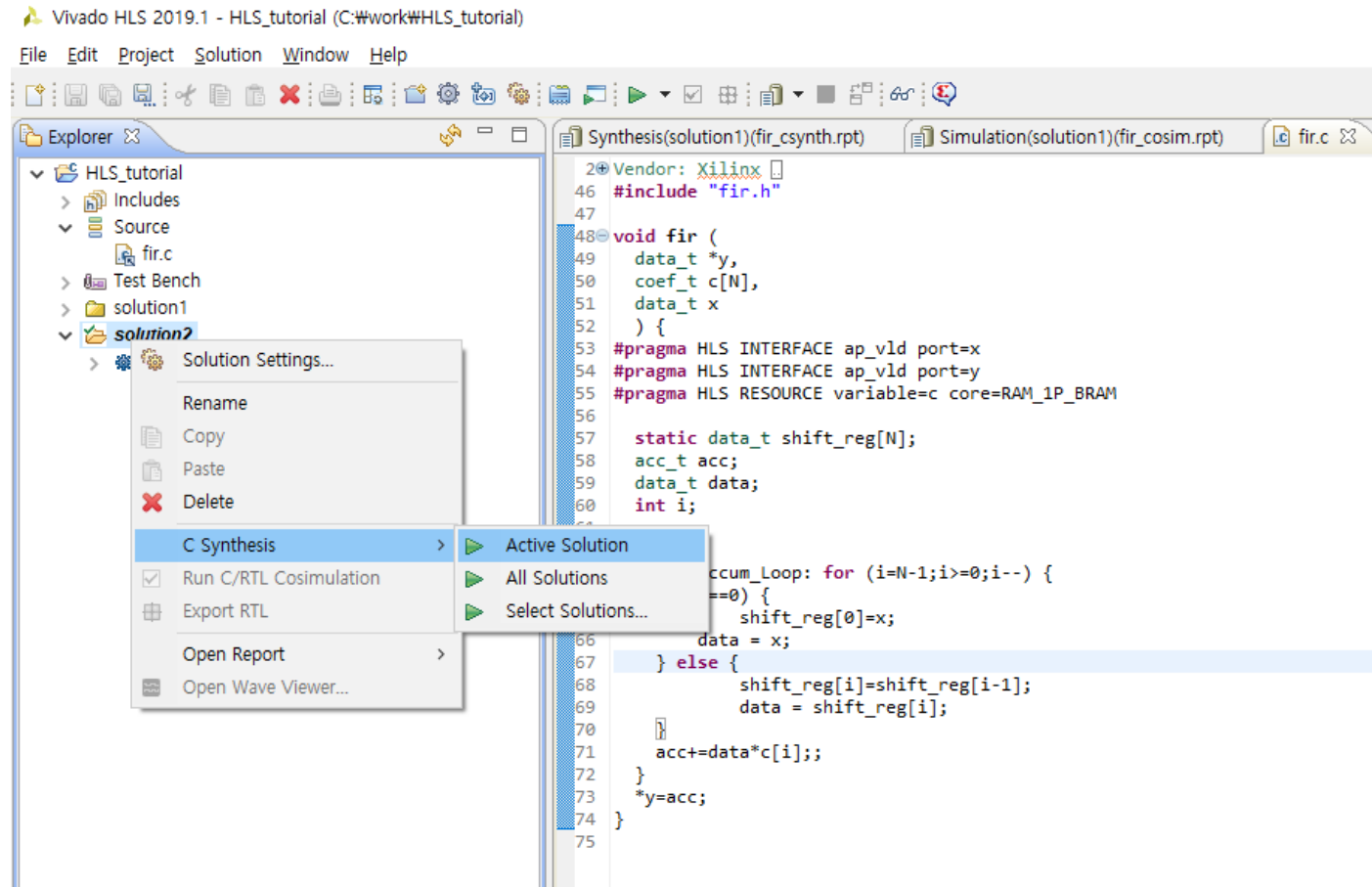
- Directive: INTERFACE
- Destination: ☒ Source File
- Options:
 - mode (optional): ap_vld
 - register (optional): ☐
 - depth (optional):
 - latency (optional):
 - port (required): x
 - name (optional):

- Port X(input data): valid
- Port Y(output data): valid

Inserted #pragma

```
Synthesis(solution1)(fir_csynth.rpt) Simulation(solution1)(fir_cosim.rpt) *fir.c
2+ Vendor: Xilinx
46 #include "fir.h"
47
48 void fir (
49     data_t *y,
50     coef_t c[N],
51     data_t x
52 ) {
53     #pragma HLS INTERFACE ap_vld port=x
54     #pragma HLS INTERFACE ap_vld port=y
55     #pragma HLS RESOURCE variable=c core=RAM_1P_BRAM
56
57     static data_t shift_reg[N];
58     acc_t acc;
59     data_t data;
60     int i;
61
62     acc=0;
63     Shift_Accum_Loop: for (i=N-1;i>=0;i--) {
64         if (i==0) {
65             shift_reg[0]=x;
66             data = x;
67         } else {
68             shift_reg[i]=shift_reg[i-1];
69             data = shift_reg[i];
70         }
71         acc+=data*c[i];
72     }
73     *y=acc;
74 }
75
```

C Synthesis(solution2)



Synthesis Report - Interface

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir	return value
ap_rst	in	1	ap_ctrl_hs	fir	return value
ap_start	in	1	ap_ctrl_hs	fir	return value
ap_done	out	1	ap_ctrl_hs	fir	return value
ap_idle	out	1	ap_ctrl_hs	fir	return value
ap_ready	out	1	ap_ctrl_hs	fir	return value
y	out	32	ap_vld	y	pointer
y_ap_vld	out	1	ap_vld	y	pointer
c_address0	out	4	ap_memory	c	array
c_ce0	out	1	ap_memory	c	array
c_q0	in	32	ap_memory	c	array
x	in	32	ap_none	x	scalar

Solution1

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir	return value
ap_rst	in	1	ap_ctrl_hs	fir	return value
ap_start	in	1	ap_ctrl_hs	fir	return value
ap_done	out	1	ap_ctrl_hs	fir	return value
ap_idle	out	1	ap_ctrl_hs	fir	return value
ap_ready	out	1	ap_ctrl_hs	fir	return value
y	out	32	ap_vld	y	pointer
y_ap_vld	out	1	ap_vld	y	pointer
c_address0	out	4	ap_memory	c	array
c_ce0	out	1	ap_memory	c	array
c_q0	in	32	ap_memory	c	array
x	in	32	ap_vld	x	scalar
x_ap_vld	in	1	ap_vld	x	scalar

Solution2

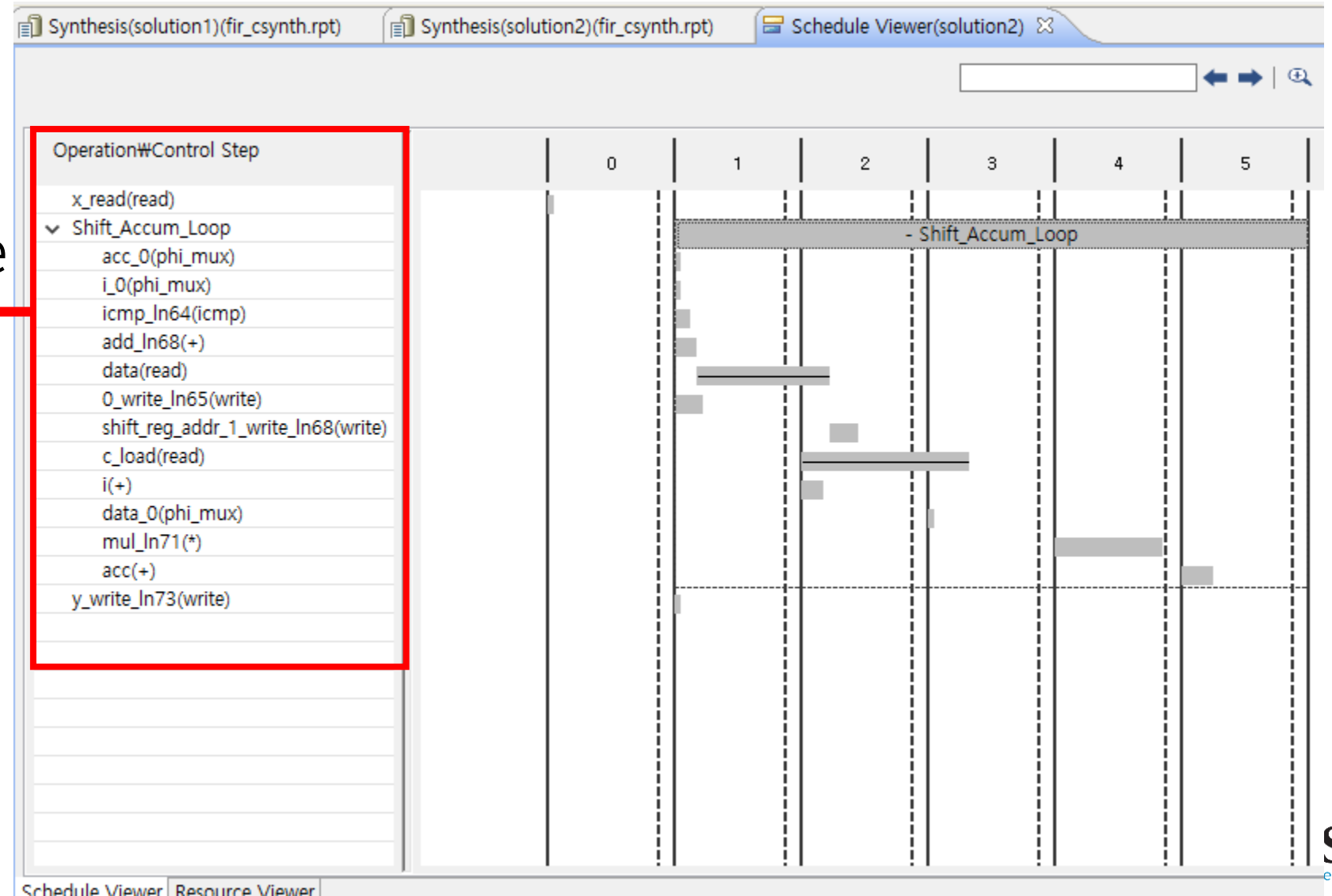
Analyze

Export the report(.html) using the [Export Wizard](#)

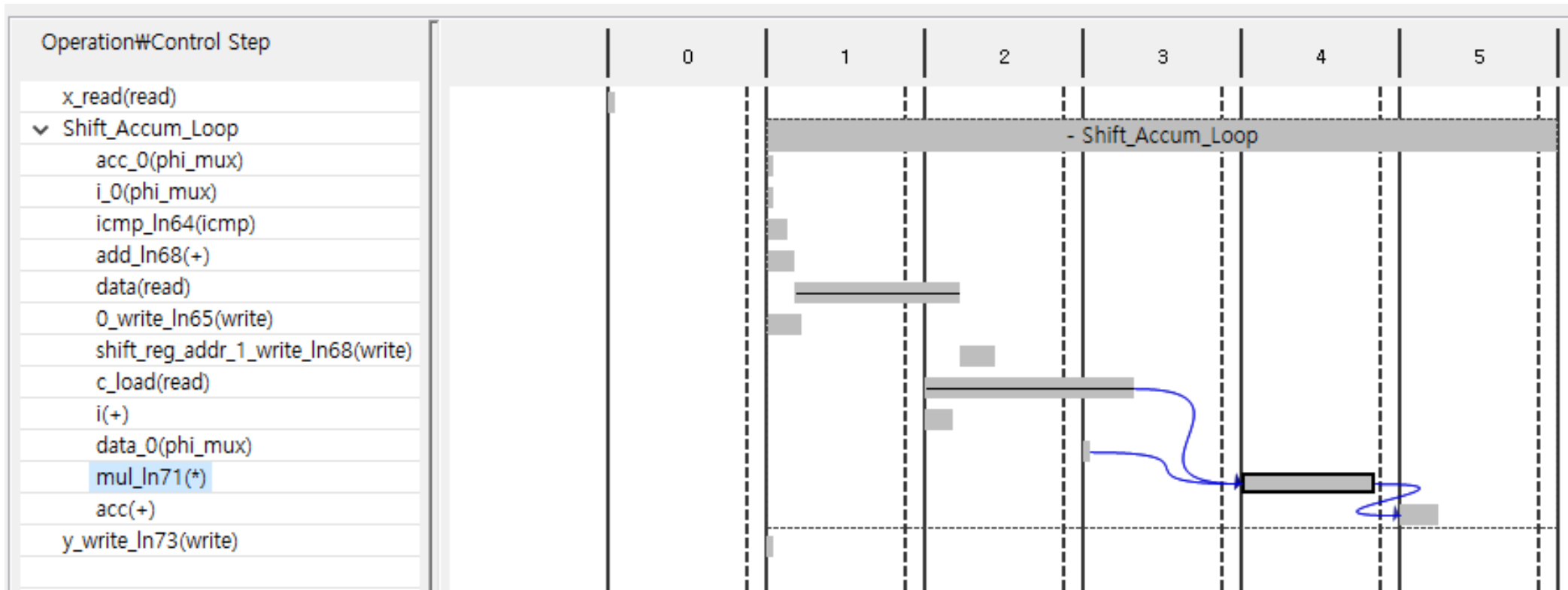
Open Analysis Perspective

[Analysis Perspective](#)

RTL Hierarchy module
operation



Operation Flow



`x_read()` - `Shift_Accum_Loop`
operation

Analysis - Resource

Operation#Control Step	0	1	2	3	4	5
<div> <div>▼ [+]I/O Ports</div> <div> <div>x</div> <div>y</div> <div>c(p0)</div> </div> </div> <div> <div>▼ [+]Memory Ports</div> <div> <div>shift_reg(p0)</div> <div>c(p0)</div> </div> </div> <div> <div>▼ [+]Expressions</div> <div> <div>grp_fu_138</div> <div>acc_0_phi_fu_107</div> <div>i_0_phi_fu_120</div> <div>icmp_ln64_fu_157</div> <div>data_0_phi_fu_131</div> <div>mul_ln71_fu_176</div> <div>acc_fu_181</div> </div> </div>	read	write	read			
		write	write	read		
		+	+			
		nhi mux				
		nhi mux				
		icmp		nhi mux	*	
						+

Higher optimization

- 루프
 - Default → 루프 연산 수행
 - 하나의 루프를 재사용 → Dependency, 리소스 최적화
 - Unroll Loop(병렬화)
- block RAM, register
 - Array shift_reg[] 구현에 BRAM이 사용됨
 - Default: BRAM
 - Shift Register로 구현하기 위해 BRAM → 개별 레지스터로 분할

Create New Solution

Vivado HLS 2019.1 - HLS_tutorial (C:\work\HLS_tutorial)

File Edit Project Solution Window Help



Explorer

- ▼ HLS_tutorial
 - > Includes
 - ▼ Source
 - fir.c
 - > Test Ben
 - > solution
 - > solution

Project Settings...

☒ Run C Simulation

Copy

Paste

☒ Delete

☒ New Solution...

Compare Reports...

Properties

Solution Wizard

Solution Configuration

Create Vivado HLS solution for selected technology

Solution Name: solution3

Clock

Period: 10

Uncertainty:

Part Selection

Part: xc7z020-clg484-1

Options

☒ Copy directives and constraints from solution:

solution2

☐ SDAccel Bottom Up Flow

Finish

Cancel

Unroll the Loop

The screenshot shows the Vivado HLS Directive Editor dialog box. The 'Directive' dropdown is set to 'UNROLL'. The 'Destination' section has 'Directive File' selected. The 'Options' section includes 'skip_exit_check (optional):', 'factor (optional):', and 'region (optional):'. A red box highlights the 'Directive' and 'Destination' sections, and a red arrow points from the 'Directive File' option to the text 'Solution 3'.

Code Snippet (fir.c):

```
20 Vendor: Xilinx
46 #include "fir.h"
47
48 void fir (
49     data_t *y,
50     coef_t c[N],
51     data_t x
52 ) {
53     #pragma HLS INTERFACE ap_vld port=x
54     #pragma HLS INTERFACE ap_vld port=y
55     #pragma HLS RESOURCE variable=c core=RAM_1P_BRAM
56
57     static data_t shift_reg[N];
58     acc_t acc;
59     data_t data;
60     int i;
61
62     acc=0;
63     Shift Accum Loop: for (i=N-1;i>=0
64         if (i==0) {
65             shift_reg[0]=x;
66             data = x;
67         } else {
68             shift_reg[i]=shift_reg[
69             data = shift_reg[i];
70         }
71         acc+=data*c[i];;
72     }
73     *y=acc;
74 }
75
```

Outline:

- fir
 - y
 - # HLS INTERFACE ap_vld port=y
 - c
 - # HLS RESOURCE variable=c core=RAM_1P_BRAM
 - x
 - # HLS INTERFACE ap_vld port=x
 - shift_reg
 - x[] shift reg
 - Shift_Accum_Loop

Solution 3

- Constraints
- directives.tcl

Partition Array

The screenshot displays the Vivado HLS IDE interface. On the left, the 'fir.c' file is open, showing a C function 'void fir' that processes an array 'shift_reg'. The line 'static data_t shift_reg[N];' is highlighted. In the center, the 'Vivado HLS Directive Editor' dialog is open, showing the 'Directive' dropdown set to 'ARRAY_PARTITION'. The 'Destination' section has 'Directive File' selected. The 'Options' section includes 'variable (required): shift_reg', 'type (optional): complete', 'factor (optional):', and 'dimension (optional): 1'. On the right, the 'Outline' pane shows the project structure, including 'fir', 'y', 'c', 'x', 'shift_reg', and 'Shift_Accum_Loop'.

```
20 Vendor: Xilinx
46 #include "fir.h"
47
48 void fir (
49     data_t *y,
50     coef_t c[N],
51     data_t x
52 ) {
53     #pragma HLS INTERFACE ap_vld port=y
54     #pragma HLS INTERFACE ap_vld port=x
55     #pragma HLS RESOURCE variable=c core=RAM_1P_BRAM
56
57     static data_t shift_reg[N];
58     acc_t acc;
59     data_t data;
60     int i;
61
62     acc=0;
63     Shift_Accum_Loop: for (i=N-1;i>=0;i--)
64         if (i==0) {
65             shift_reg[0]=x;
66             data = x;
67         } else {
68             shift_reg[i]=shift_reg[i-1];
69             data = shift_reg[i];
70         }
71     acc+=data*c[i];
72 }
73 *y=acc;
74 }
75
```

Vivado HLS Directive Editor

Directive: ARRAY_PARTITION

Destination: ☐ Source File ☒ Directive File

Options:

variable (required): shift_reg

type (optional): complete

factor (optional):

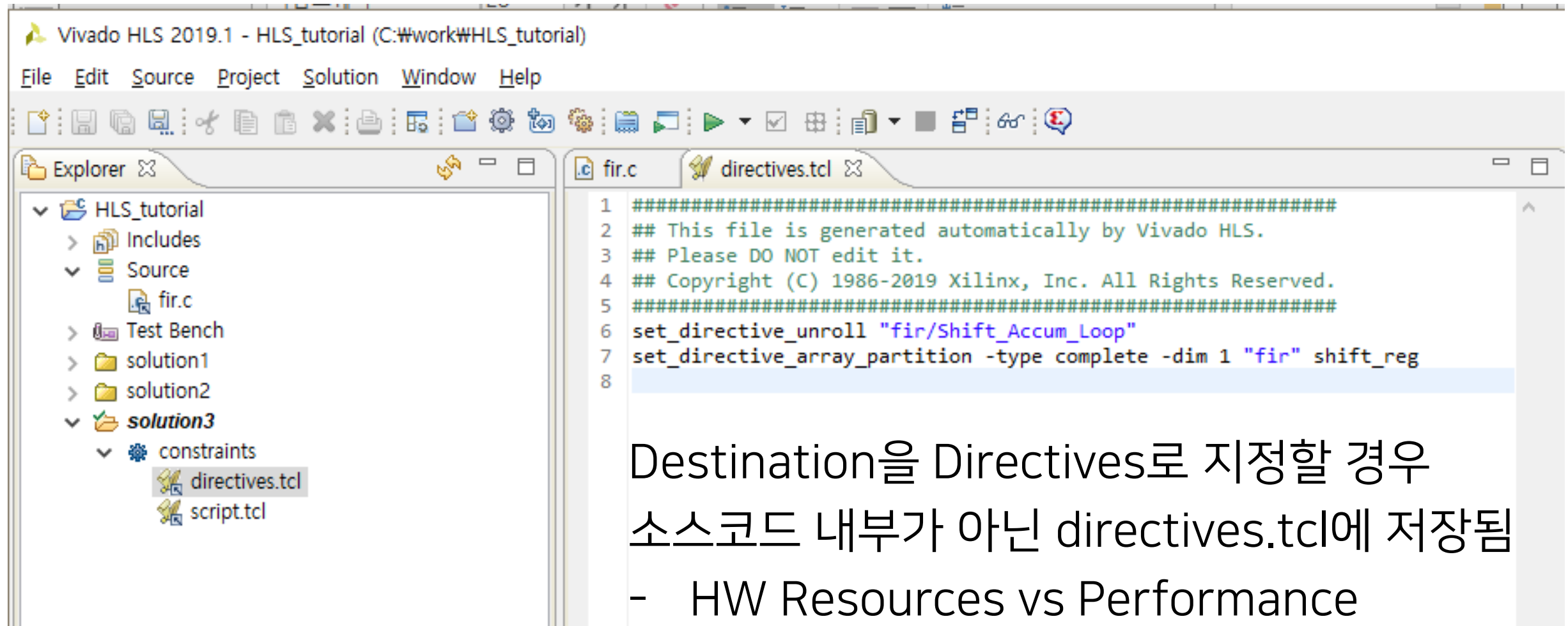
dimension (optional): 1

Help Cancel OK

Outline Directive

- fir
 - y
 - # HLS INTERFACE ap_vld port=y
 - c
 - # HLS RESOURCE variable=c core=RAM_1P_BRAM
 - x
 - # HLS INTERFACE ap_vld port=x
 - shift_reg
 - x[1] shift_reg
 - Shift_Accum_Loop
 - % HLS UNROLL

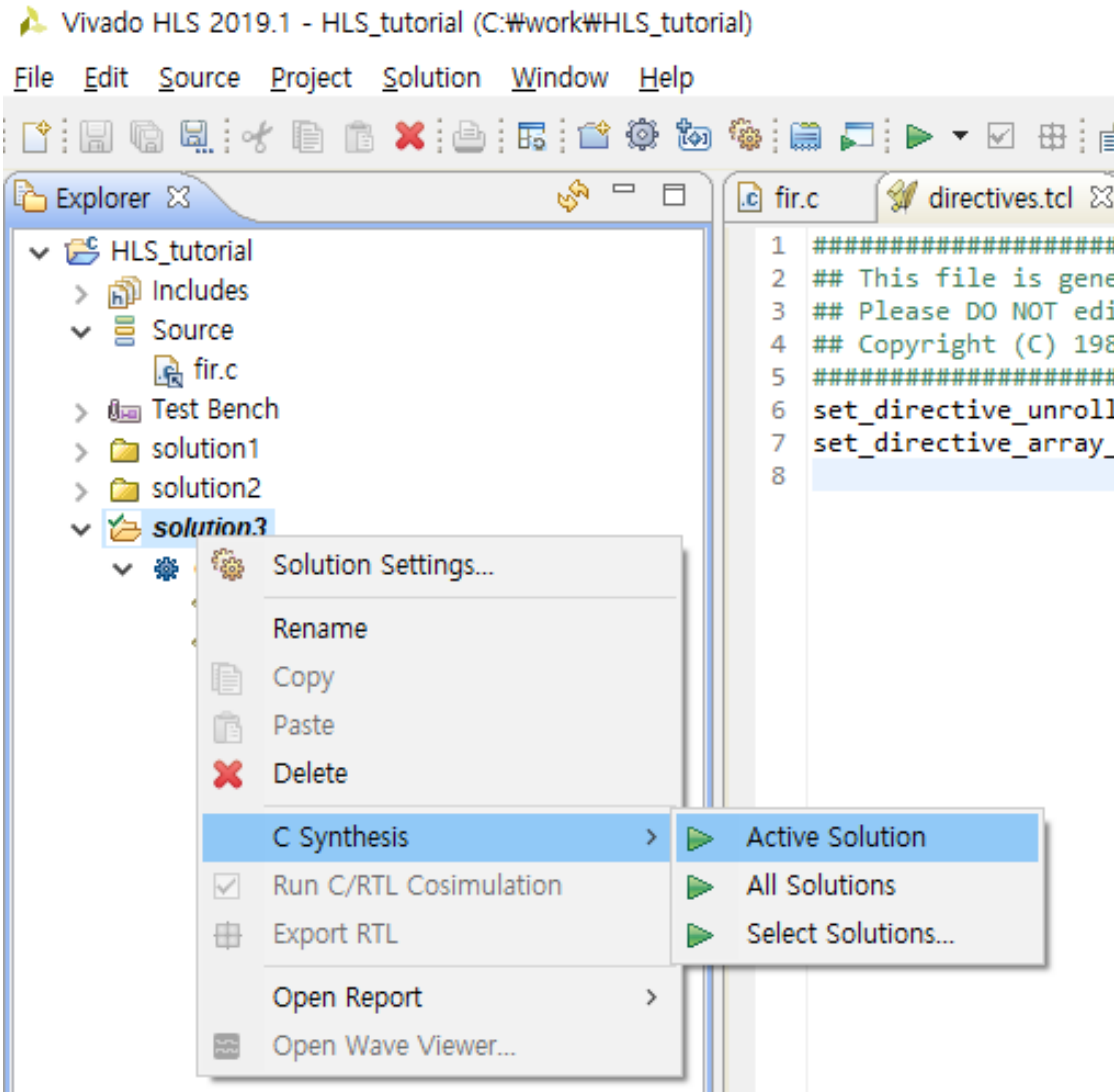
directives.tcl



Destination을 Directives로 지정할 경우
소스코드 내부가 아닌 directives.tcl에 저장됨

- HW Resources vs Performance
- 여러 버전의 최적화 정도 비교 시 사용

Synthesis



Synthesis Report

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.510	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
56	56	56	56	none

Detail

Instance

N/A

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.742	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
14	14	14	14	none

Detail

Instance

Loop

x4 faster

Summary

- HLS
 - C/C++ 작성된 SW를 HW IP로 변환, 테스트
 - Data Types, I/O Ports
 - HLS를 이용한 최적화

What's Next?

HLS

- C/C++ → HW IP 설계
- FPGA / ZYNQ 모두 사용

SDSoC(Software Defined System-On-a-Chip)

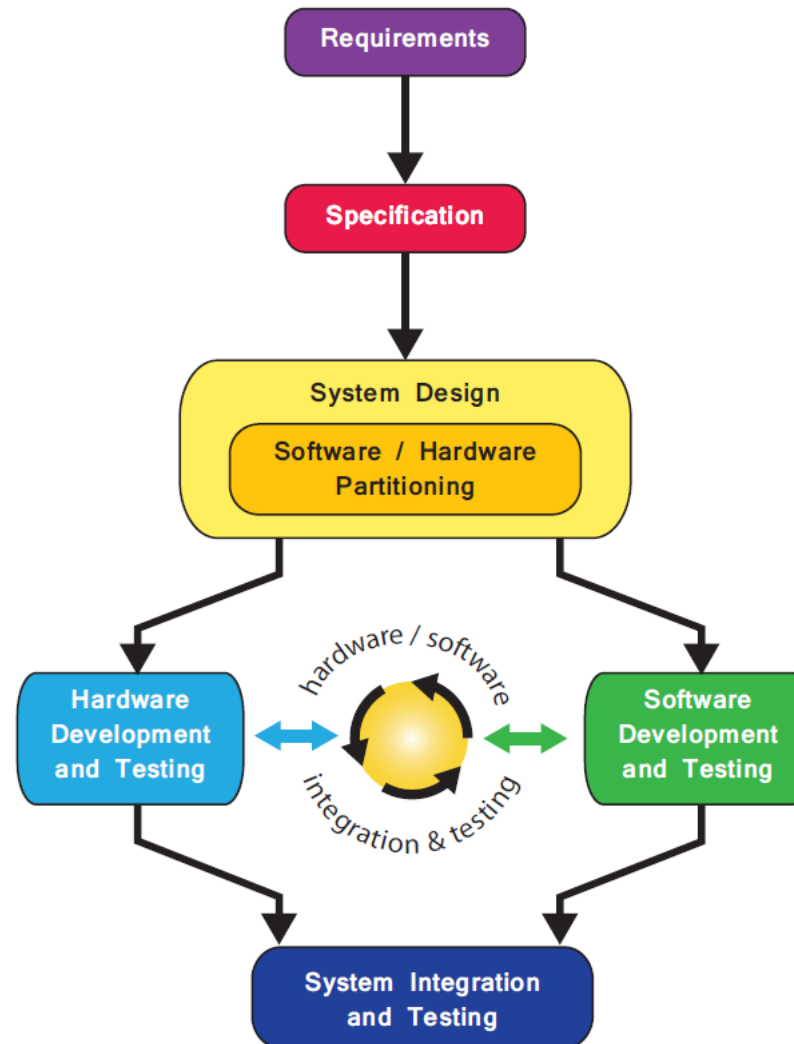
- Zynq용 HW Acceleration system
- System Profiling, SW/HW Partitioning, HLS, Compile 동시에

Zynq SoC 설계(HDL, HLS)

Xilinx VIVADO

VHDL/Verilog
HLS(SDSoC)

AXI HW IP



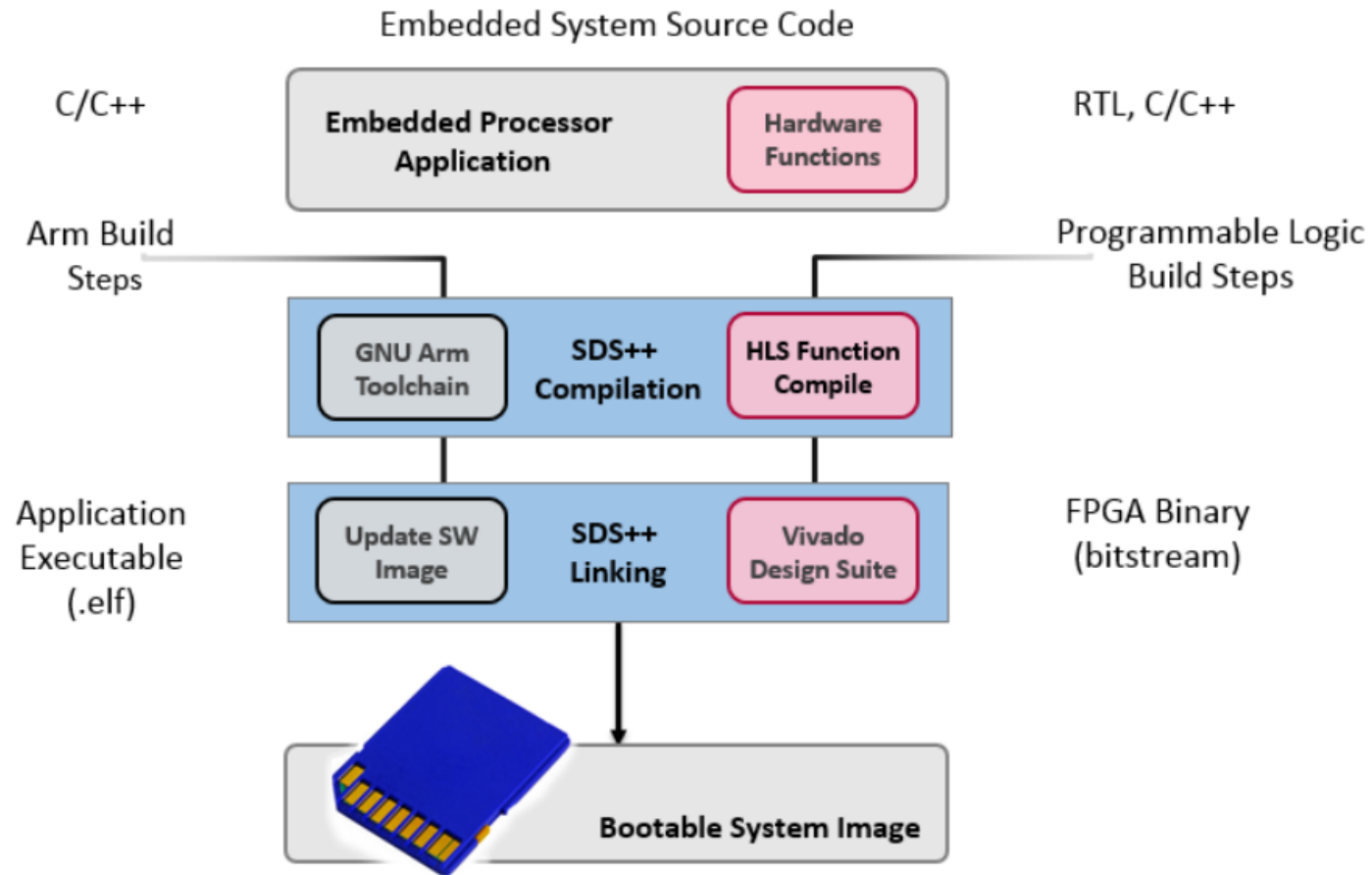
Xilinx SDK

Xilinx, ARM
라이브러리

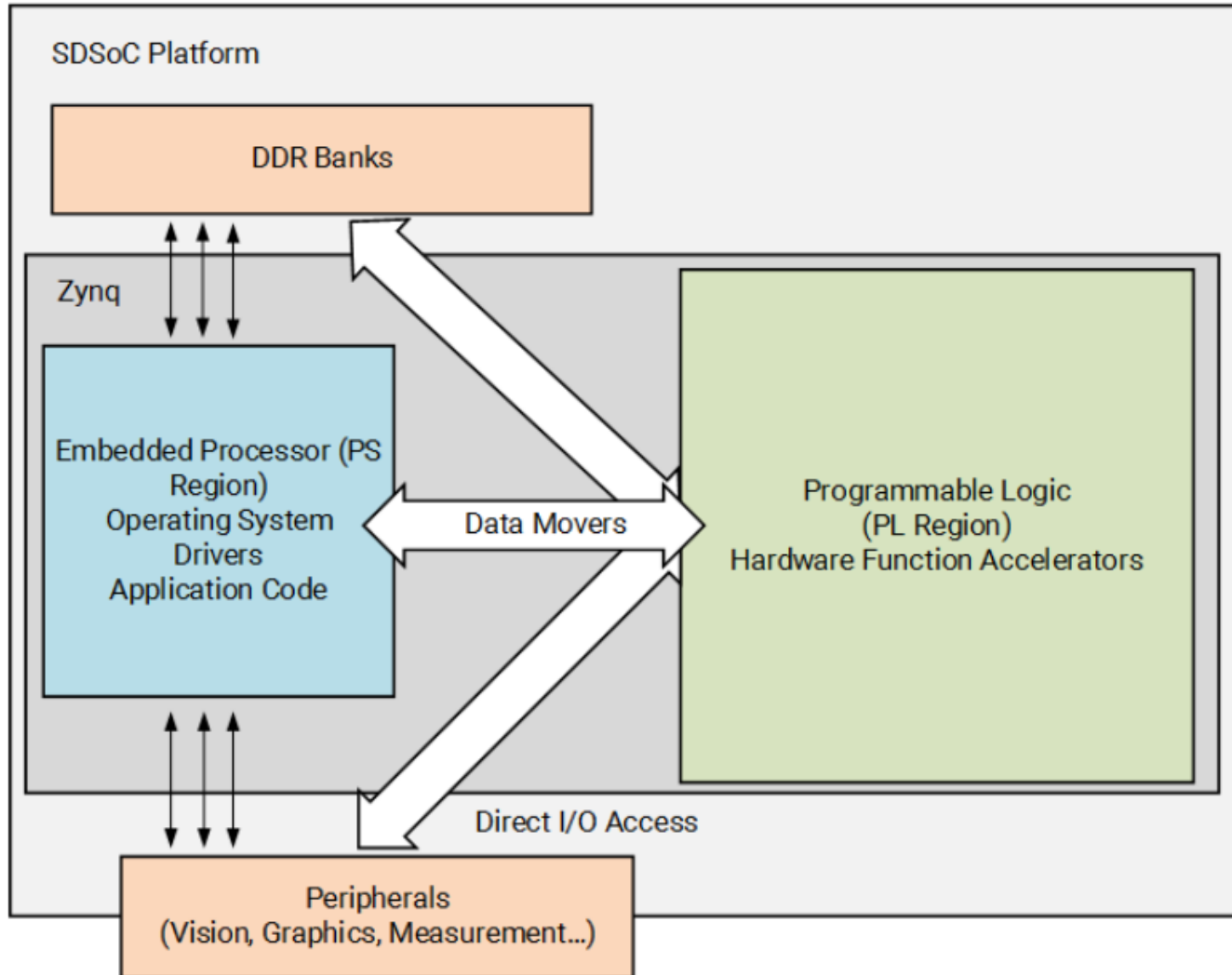
C/C++ SW IP

Bare Metal App
freeRTOS, Linux

SDSoC



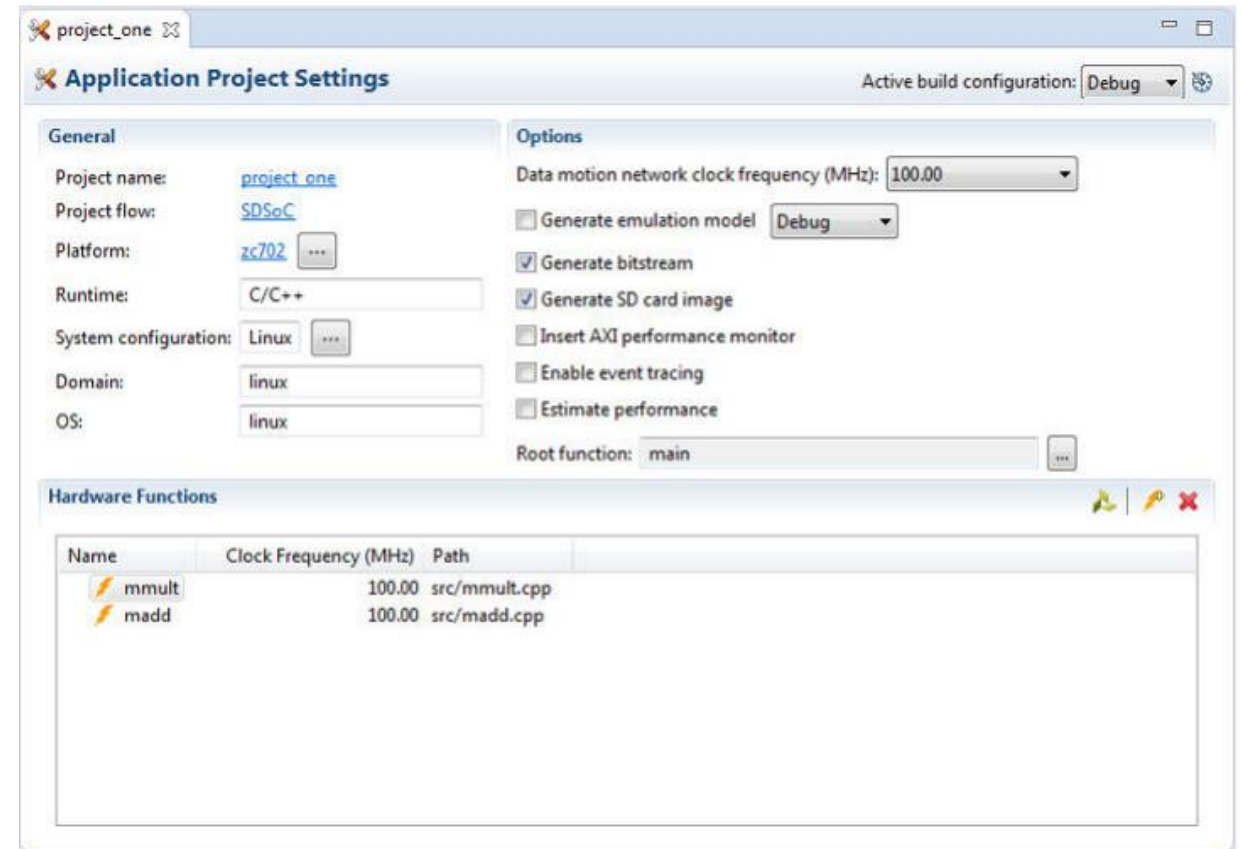
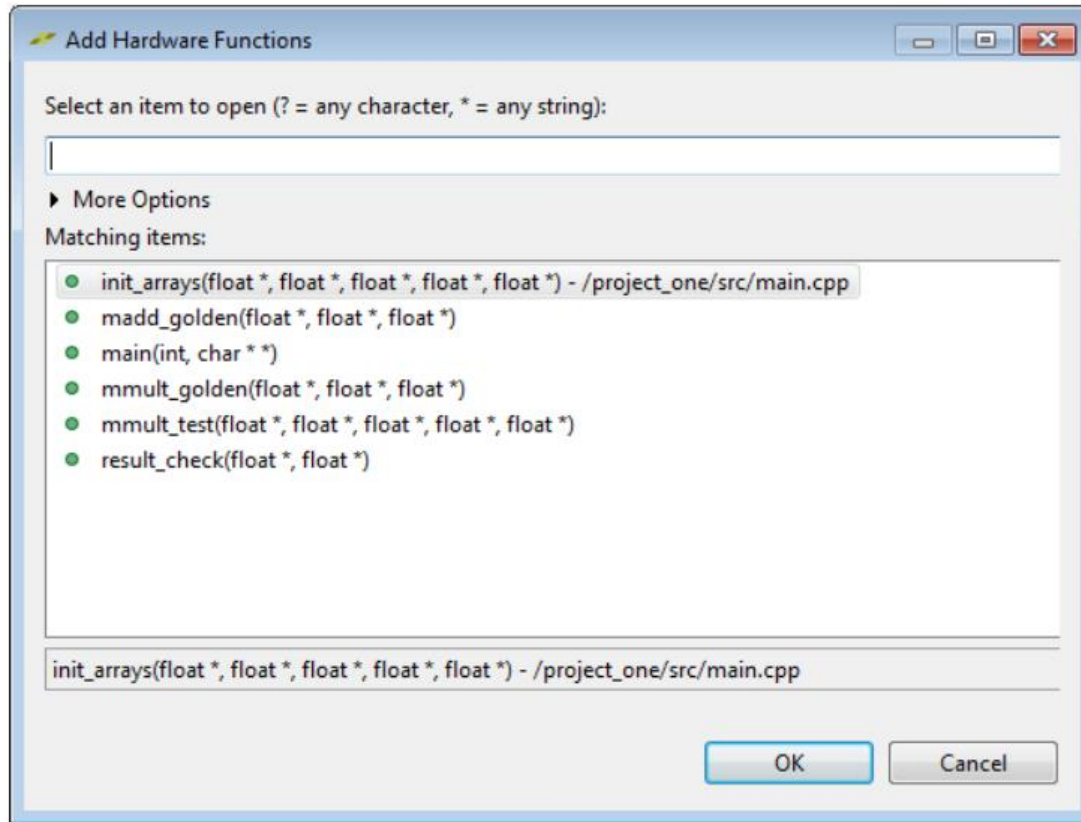
SDSoC System



SDSoC System

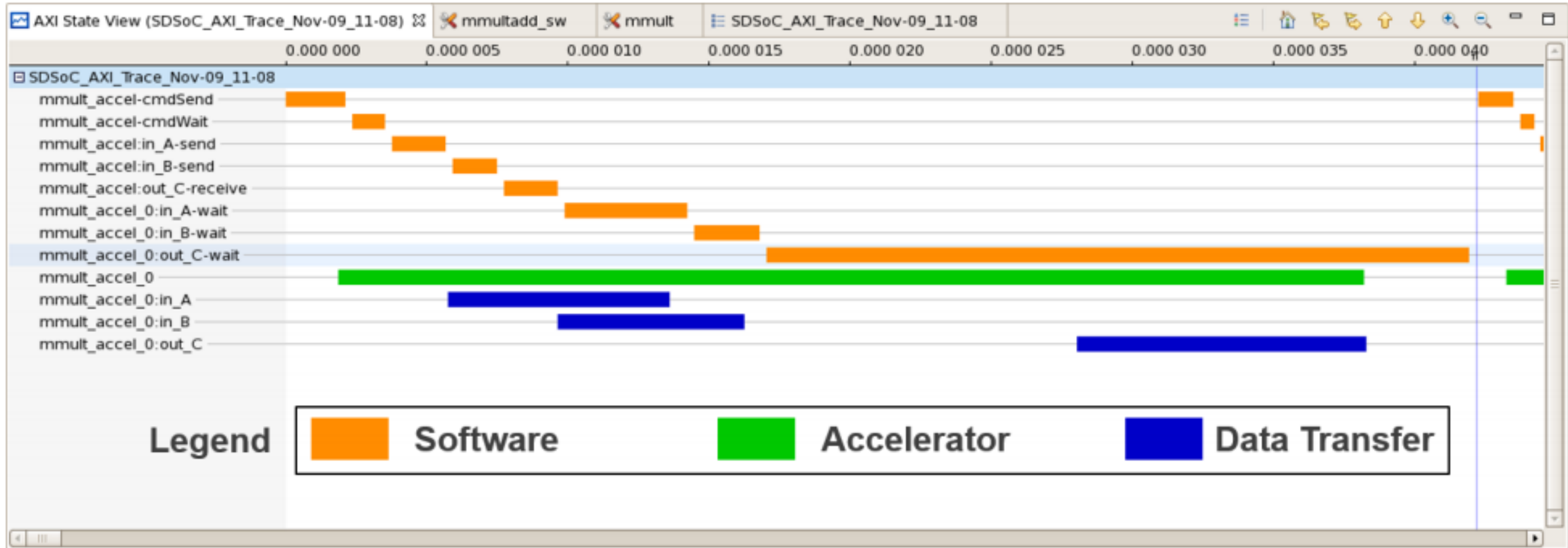
- 기본 PS Configuration
 - DRAM
 - Peripherals
 - MIO
- PS <-> PL Port

SW/HW Partitioning in SDSoC



- 소스코드의 전체 함수 중 선택한 함수를 HW 변환

System Profiling in SDSoC



References

ZYNQ

- THE ZYNQ BOOK(<http://www.zynqbook.com>)
- ZYNQ Technical Reference Guide(UG-585)

HLS

- Vivado-high-level-synthesis(ug871)
- Vivado-high-level-synthesis(UG902)
- Vivado-Intro-FPGA-Design-HLS(UG998)

SDSoC

- SDSoC-optimization-guide(UG1235)