

Improving Performance

Innova Lee(이상훈)
gcccompil3r@gmail.com

Introduction

이 Lab에선 Vivado HLS 에서 설계 성능을 향상시키는데 사용할 수 있는 다양한 기법과 지침을 소개한다.

고려중인 설계(사용자 정의)는 RGB 형식의 Image 를 받아 Y'UV 색상 공간으로 변환하고 Y'UV 이미지에 필터를 적용한 다음 다시 RGB 로 변환하는 것이다.

Objectives

이 Lab 을 완료하면 아래를 수행할 수 있다:

- * 설계에 Directives(지시어) 추가
- * INLINE 지시어의 효과 이해
- * PIPELINE 지시어를 사용하여 성능 향상
- * DATAFLOW 지시어와 Configuration 명령 기능을 구별

Procedure

이 Lab 은 일반적인 개요로 구성된 단계로 구분된다.

아래 나오는 세부 지침에 대한 정보를 제공한다.

Lab 을 진행하려면 이 세부 지침을 따르라.

이 Lab 은 6 가지 기본 단계로 구성된다:

Vivado HLS 명령 프롬프트를 사용하여 새 프로젝트를 만들고 생성 된 프로젝트와 생성 된 솔루션을 분석하고 인라인 기능을 끄고

TRIPCOUNT, PIPELINE 및 DATAFLOW 지시어와 Command Configuration 을 적용한 다음 마지막으로 Design 을 Export 하고 구현한다.

General Flow for this Lab

1. CLI 를 사용하여 Project 를 생성한다.
2. Project 를 분석하고 결과를 확인한다.
3. TRIPCOUNT 지시문을 적용한다.
4. PIPELINE 지시문을 적용한다.
5. DATAFLOW 지시문을 적용한다.
6. Design 을 Export 하고 구현한다.

Create a Vivado HLS Project from Command Line

- 1-1. Vivado HLS Command Line 을 사용하여 Design 의 유효성을 검사한다.
Command Line 에서 새로운 Vivado HLS Project 를 생성하라.
- 1-1-1. Vivado HLS 를 띄운다.
`/opt/Xilinx/Vivado_HLS/2017.1/bin/vivado_hls`
- 1-1-2. Vivado HLS Command Prompt 에서 디렉토리를 lab2 로 변경한다.

1-1-3. 자체 점검 프로그램(yuv_filter_test.c)가 제공된다.

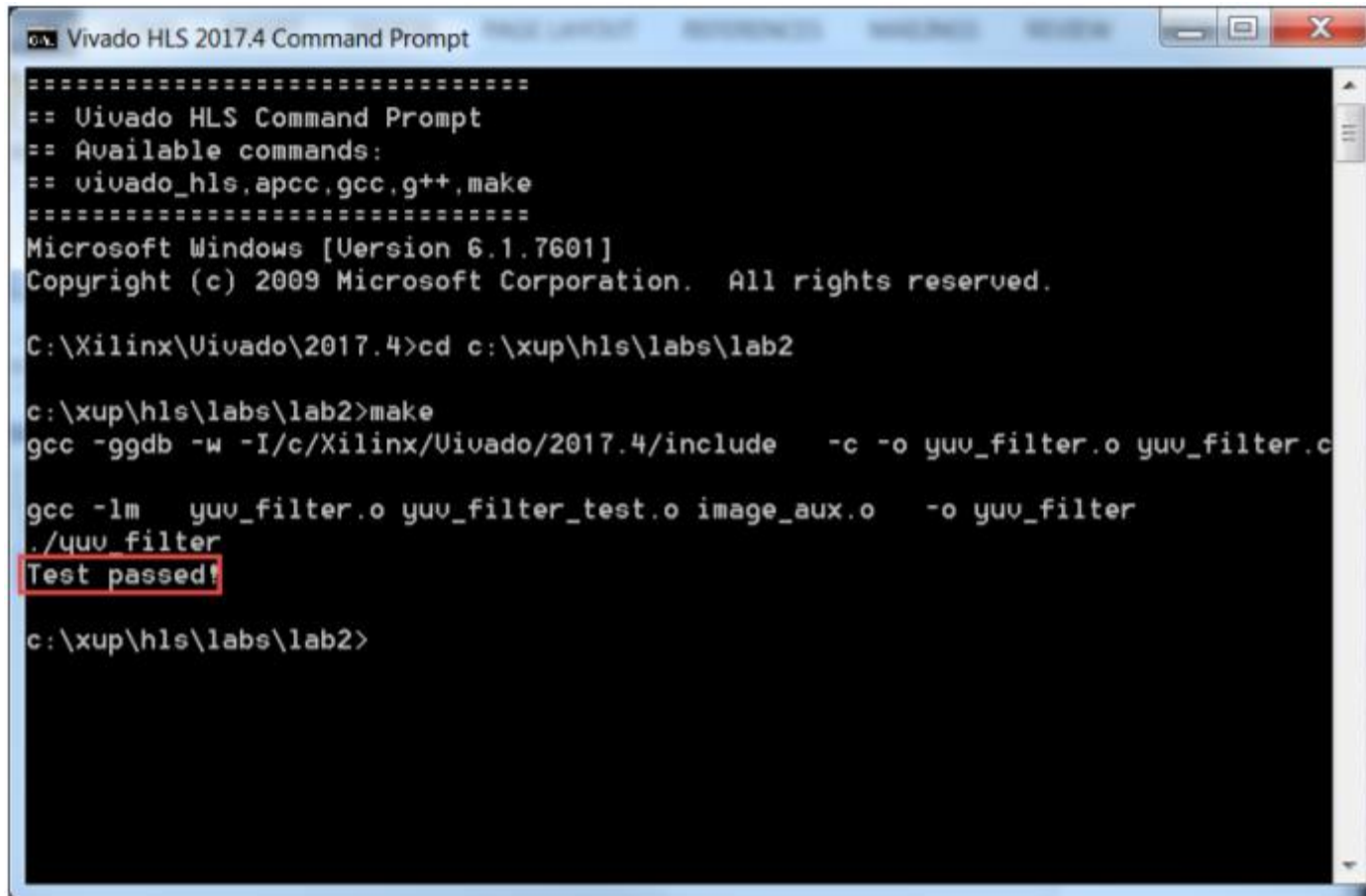
이것을 사용하여 Design 을 검증할 수 있다.

Makefile 도 제공된다.

Makefile 을 사용하여 필요한 Source 파일을 컴파일하고 컴파일된 프로그램을 실행할 수 있다.

이러한 파일과 Project 디렉토리의 내용을 검사할 수 있다.

Vivado HLS 명령 프롬프트에서 make 를 입력하여 Program 을 컴파일하고 실행한다.



```
=====  
== Vivado HLS Command Prompt  
== Available commands:  
== vivado_hls,apcc,gcc,g++,make  
=====
```

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

```
C:\Xilinx\Uivado\2017.4>cd c:\xup\hls\labs\lab2  
  
c:\xup\hls\labs\lab2>make  
gcc -ggdb -w -I/c/Xilinx/Uivado/2017.4/include -c -o yuv_filter.o yuv_filter.c  
  
gcc -lm yuv_filter.o yuv_filter_test.o image_aux.o -o yuv_filter  
./yuv_filter  
Test passed!  
  
c:\xup\hls\labs\lab2>
```

Source 파일(yuv_filter.c, yuv_filter_test.c 및 image_aux.c)이
컴파일 된 다음 yuv_filter 실행 가능 프로그램이 작성된 다음 실행되었다.
Program 은 Design 을 테스트하고 통과 테스트 메시지를 출력한다.

1-1-4. Vivado HLS tcl Script File(yuv_filter.tcl)이 제공되며
Vivado HLS Project 를 만드는데 사용할 수 있다.

1-1-5. Vivado HLS Command Prompt 창에서 vivado_hls -f zed_yuv_filter.tcl 를 타입하여 ZedBoard 를 Target 으로 Project 를 만들거나
Vivado HLS Command Prompt 창에서 vivado_hls -f zybo_yuv_filter.tcl 을 타입하여 Zybo 를 Target 으로 Project 를 만든다.

Project 가 생성될 것이고 Vivado HLS.log 파일이 생성될 것이다.

1-1-6. Text 편집기를 사용하여 lab2 에서 vivado_hls.log 파일을 열고 아래 섹션을 참조하라.

- * yuv_filter.prj 라는 디렉토리와 Project 를 생성하고,
Project 에 Design 파일을 추가하고,
Solution 이름을 solution1 로 설정하고,
Target Device(ZedBoard 의 경우 Zynq-z020
혹은 Zybo 의 경우 Zynqz010)을 설정하고,
원하는 Clock 주기를 10 ns(ZedBoard) 혹은 8 ns(Zybo)로 설정하고
Design 및 Test Bnech 파일 가져오도록 한다.

- * 각 기능과 하위 기능의 Scheduling 과
Binding 을 포함하는 Design 을 합성(생성)한다.

- * SystemC, Verilog 및 VHDL Language 에서 각 함수 및 하위 함수의 RTL 생성

```
1
2 ***** Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC v2017.4 (64-bit)
3      **** SW Build 2086221 on Fri Dec 15 20:55:39 MST 2017
4      **** IP Build 2085800 on Fri Dec 15 22:25:07 MST 2017
5      ** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
6
7 source C:/Xilinx/Vivado/2017.4/scripts/vivado_hls/hls.tcl -notrace
8 INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2017.4/bin/unwrapped/win64.o/vivado_hls.exe'
9 INFO: [HLS 200-10] For user 'parimalp' on host 'xsjparimalp31' (Windows NT_amd64 version 6.1)
10 INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab2'
11 INFO: [HLS 200-10] Creating and opening project 'C:/xup/hls/labs/lab2/yuv_filter.prj'.
12 INFO: [HLS 200-10] Adding design file 'yuv_filter.c' to the project
13 INFO: [HLS 200-10] Adding test bench file 'image_aux.c' to the project
14 INFO: [HLS 200-10] Adding test bench file 'yuv_filter_test.c' to the project
15 INFO: [HLS 200-10] Adding test bench file 'test_data' to the project
16 INFO: [HLS 200-10] Creating and opening solution 'C:/xup/hls/labs/lab2/yuv_filter.prj/solution1'
17 INFO: [HLS 200-10] Cleaning up the solution database.
18 INFO: [HLS 200-10] Setting target device to 'xc7z020clg484-1'
19 INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
20 INFO: [HLS 200-10] Analyzing design file 'yuv_filter.c' ...
21 INFO: [HLS 200-10] Validating synthesis directives ...
22 INFO: [HLS 200-111] Finished Checking Pragmas Time (s): cpu = 00:00:01 ; elapsed = 00:00:09 .
23 INFO: [HLS 200-111] Finished Linking Time (s): cpu = 00:00:01 ; elapsed = 00:00:12 . Memory (0
24 INFO: [HLS 200-10] Starting code transformations ...
```



```
24 INFO: [HLS 200-10] Starting code transformations ...
25 INFO: [HLS 200-111] Finished Standard Transforms Time (s): cpu = 00:00:01 ; el
26 INFO: [HLS 200-10] Checking synthesizability ...
27 INFO: [HLS 200-111] Finished Checking Synthesizability Time (s): cpu = 00:00:0
28 INFO: [XFORM 203-602] Inlining function 'yuv_scale' into 'yuv_filter' (yuv_fil
29 INFO: [XFORM 203-401] Performing if-conversion on hyperblock from (yuv_filter.
30 INFO: [XFORM 203-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c
31 INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:02 ; elapsed
32 INFO: [HLS 200-111] Finished Architecture Synthesis Time (s): cpu = 00:00:02 ;
33 INFO: [HLS 200-10] Starting hardware synthesis ...
34 INFO: [HLS 200-10] Synthesizing 'yuv_filter' ...
35 INFO: [HLS 200-10] -----
36 INFO: [HLS 200-42] -- Implementing module 'rgb2yuv'
37 INFO: [HLS 200-10] -----
38 INFO: [SCHED 204-11] Starting scheduling ...
39 WARNING: [SCHED 204-21] Estimated clock period (10.283ns) exceeds the target (
40 WARNING: [SCHED 204-21] The critical path consists of the following:
41     'mul' operation ('tmp_25', yuv_filter.c:57) (3.36 ns)
42     'add' operation ('tmp3', yuv_filter.c:57) (3.02 ns)
43     'add' operation ('tmp_26', yuv_filter.c:57) (3.9 ns)
44 INFO: [SCHED 204-11] Finished scheduling.
45 INFO: [HLS 200-111] Elapsed time: 15.045 seconds; current allocated memory: 9
46 INFO: [BIND 205-100] Starting micro-architecture generation ...
47 INFO: [BIND 205-101] Performing variable lifetime analysis.
48 INFO: [BIND 205-101] Exploring resource sharing.
49 INFO: [BIND 205-101] Binding ...
50 INFO: [BIND 205-100] Finished micro-architecture generation.
51 INFO: [HLS 200-111] Elapsed time: 0.105 seconds; current allocated memory: 95
52 INFO: [HLS 200-10] -----
53 INFO: [HLS 200-42] -- Implementing module 'yuv2rgb'
54 INFO: [HLS 200-10] -----
55 INFO: [SCHED 204-11] Starting scheduling ...
56 WARNING: [SCHED 204-21] Estimated clock period (10.8454ns) exceeds the target
57 WARNING: [SCHED 204-21] The critical path consists of the following:
58     'mul' operation ('tmp_12', yuv_filter.c:101) (3.36 ns)
59     'add' operation ('tmp1', yuv_filter.c:101) (3.02 ns)
60     'add' operation ('tmp_14', yuv_filter.c:101) (2.14 ns)
61     'icmp' operation ('icmp9', yuv_filter.c:101) (0.959 ns)
62     'select' operation ('p_phitmp2', yuv_filter.c:101) (0 ns)
63     'select' operation ('G', yuv_filter.c:101) (1.37 ns)
64 INFO: [SCHED 204-11] Finished scheduling.
```

```

1
2 ***** Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC v2017.4 (64-bit)
3 **** SW Build 2086221 on Fri Dec 15 20:55:39 MST 2017
4 **** IP Build 2085800 on Fri Dec 15 22:25:07 MST 2017
5 ** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
6
7 source C:/Xilinx/Vivado/2017.4/scripts/vivado_hls/hls.tcl -notrace
8 INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2017.4/bin/unwrapped/win64.o/vivado_hls.exe'
9 INFO: [HLS 200-10] For user 'parimalp' on host 'xsjparimalp31' (Windows NT_amd64 version 6.1)
10 INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab2'
11 INFO: [HLS 200-10] Creating and opening project 'C:/xup/hls/labs/lab2/yuv_filter.prj'.
12 INFO: [HLS 200-10] Adding design file 'yuv_filter.c' to the project
13 INFO: [HLS 200-10] Adding test bench file 'image_aux.c' to the project
14 INFO: [HLS 200-10] Adding test bench file 'yuv_filter_test.c' to the project
15 INFO: [HLS 200-10] Adding test bench file 'test_data' to the project
16 INFO: [HLS 200-10] Creating and opening solution 'C:/xup/hls/labs/lab2/yuv_filter.prj/solution'
17 INFO: [HLS 200-10] Cleaning up the solution database.
18 INFO: [HLS 200-10] Setting target device to 'xc7z020clg484-1'
19 INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
20 INFO: [HLS 200-10] Analyzing design file 'yuv_filter.c' ...
21 INFO: [HLS 200-10] Validating synthesis directives ...
22 INFO: [HLS 200-111] Finished Checking Pragmas Time (s): cpu = 00:00:01 ; elapsed = 00:00:09 .
23 INFO: [HLS 200-111] Finished Linking Time (s): cpu = 00:00:01 ; elapsed = 00:00:12 . Memory (I
24 INFO: [HLS 200-10] Starting code transformations ...

```

```

129 INFO: [RTGEN 206-100] Finished creating RTL model for 'yuv_filter'.
130 INFO: [HLS 200-111] Elapsed time: 0.623 seconds; current allocated memory: 98.680 MB.
131 INFO: [RTMG 210-278] Implementing memory 'yuv_filter_p_yuv_hbi_ram (RAM)' using block RAM
132 INFO: [HLS 200-111] Finished generating all RTL models Time (s): cpu = 00:00:04 ; elapsed
133 INFO: [SYSC 207-301] Generating SystemC RTL for yuv_filter.
134 INFO: [VHDL 208-304] Generating VHDL RTL for yuv_filter.
135 INFO: [VLOG 209-307] Generating Verilog RTL for yuv filter.
136 INFO: [HLS 200-112] Total elapsed time: 21.239 seconds; peak allocated memory: 98.680 MB.
137 INFO: [Common 17-206] Exiting vivado_hls at Tue Feb 13 19:33:43 2018...

```


1-1-7. Vivado HLS 명령 프롬프트 창에서 `vivado_hls -p yuv_filter.prj` 를 입력하여 생성된 Project(GUI 모드에서)를 연다.

Analyze the Created Project and Results

2-1. Source 파일을 열고 3 가지 기능이 사용됨에 주의하라.
결과를 보고 Latency 가 정의되지 않았음을 관찰하라(? 로 표기)

2-1-1. Vivado HLS GUI 의 Explorer View 에서 원본 폴더를 확장하고
`yuv_filter.c` 를 더블 클릭하여 내용을 살펴본다.

* Design 은 `rgb2yuv`, `yuv_scale` 및 `yuv2rgb` 의 3 가지 기능으로 구현된다.

* 이러한 각 Filter 함수는 전체 Source Image
(`image_aux.h` 에 지정된 최대 크기를 가짐)를 반복하여
결과 Image 에 Pixel 을 생성하기 위해 단일 Source Pixel 이 필요하다.

* Scale 함수는 Y'UV 구성 요소에
최상위 인수로 제공된 개별 Scale 인수를 간단히 적용한다.

* 대부분의 변수는 사용자 정의(`typedef`) 및 집계(예: 구조, 배열) 유형이다.

* 원본 소스가 `malloc()` 을 사용하여 내부 Image Buffer 에
동적으로 Storage 를 할당했음을 주목하십시오.
`malloc()` 은 SW 에서 큰 자료 구조에 적합하지만
합성할 수 없으며 Vivado HLS 에서 지원하지 않는다.

* 가능한 해결 방법은 `__SYNTHESIS__` 매크로를 사용하여
조건부로 코드를 컴파일 한다.
Vivado HLS 는 Code 를 읽을 때 `__SYNTHESIS__` 매크로를 자동으로 정의한다.
이렇게하면 원본 `malloc()` 코드가 `synthesis` 의 외부에서 사용되지만
Vivado HLS 는 합성할 때 해결 방법을 사용한다.

2-1-2. Explorer 보기에서 syn -> report 폴더를 확장하고 yuv_filter_csynh.rpt 항목을 더블 클릭하여 합성 보고서를 연다.

2-1-3. 이 Design 의 각 Loop 에는 가변 범위가 있다.

width 와 height 는 입력 타입 image_t 의 멤버에 의해 정의된다.

변수 범위가 Loop 에 있을 때 Loop 의 총 대기 시간을 결정할 수 없다.

이는 report 를 사용하여 분석을 수행하는 능력에 영향을 미친다.

따라서 다양한 Latencies 에 대해 "?" 가 보고된다.

☐ Latency (clock cycles)

☐ Summary

Latency		Interval		Type
min	max	min	max	
?	?	?	?	none

Apply TRIPCOUNT Pragma

3-1. 소스 파일을 열고 pragma 라인을 주석 해제하고, 재합성하고

예상 대기 시간뿐만 아니라 사용된 자원을 관찰한다.

이 단계의 세부 항목에 나열된 질문에 답하라.

3-1-1. Loop 대기 시간 추정을 돕기 위해 Vivado HLS 는

TRIPCOUNT 지시문을 제공하여 사용자가 변수 범위를 제한할 수 있도록 한다.

이 Design 에서 이러한 지시문이 #pragma 문 형식으로 Source 에 포함되었다.

3-1-2. Loop 경계를 정의하고 파일을 저장하려면

#pragma lines(50, 53, 90, 93, 130, 133) 의 주석 처리를 제거하라.

3-1-3. Solution -> Run C Synthesis -> Active Solution 을 선택하여 Design 을 합성한다.
Process 가 완료되면 합성 보고서를 살펴본다.

▢ Latency (clock cycles)

▢ Summary

Latency		Interval		Type
min	max	min	max	
841205	51621125	841205	51621125	none

(a) ZedBoard

▢ Latency (clock cycles)

▢ Summary

Latency		Interval		Type
min	max	min	max	
1001205	61451525	1001205	61451525	none

(b) Zybo

3-1-4. 보고서를 살펴보고 아래 질문에 답해보자.

Estimated Clock Period:

Worst case Latency:

Number of DSP48E used:

Number of BRAMs used:

Number of FFs used:

Number of LUTs used:

3-1-5. Console 창을 Scroll 하면 yuv_scale 함수가 yuv_filter 함수에 자동으로 inline 된다.

```
INFO: [HLS 200-10] Checking synthesizability ...
INFO: [HLS 200-111] Finished Checking 5synthesizability Time (s): cpu = 00:00:01 ; elapsed = 00:00:11 . Memory (MB): peak = 104.156 ; gain = 47.289
INFO: [XFORM 203-602] Inlining function 'yuv scale' into 'yuv filter' (yuv_filter.c:24) automatically.
INFO: [XFORM 203-401] Performing if-conversion on hyperblock from (yuv_filter.c:92:33) to (yuv_filter.c:92:27) in function 'yuv2rgb'... converting 7 basic blocks.
INFO: [XFORM 203-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c:30)...11 expression(s) balanced.
INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:01 ; elapsed = 00:00:12 . Memory (MB): peak = 125.930 ; gain = 69.063
```

3-1-6. Explorer View 의 syn 보고서 폴더 아래에 rgb2yuv.rpt, yuv_filter.rpt 및 yuv2rgb.rpt 라는 3 가지 항목이 있음을 확인하라.
함수가 yuv_filter 함수로 inline 되었기 때문에 yuv_scale.rpt 에 대한 항목이 없다.

최상위 보고서의 구성 요소(Utilization -> Details -> Component)
혹은 Project Explorer 의 보고서 컨테이너에서 아래로 이동하여 하위 Module 의 보고서에 접근할 수 있다.

3-1-7. Loop 대기 시간 요약을 확장하고 yuv_scale 기능의 대기 시간 및 trip 횟수를 확인하라.
YUV_SCALE_LOOP_Y Loop Latency 는 지정된 TRIPCOUNT 의 7 배이며 Loop 의 각 반복에 대해 7 Cycle 이 사용됨을 의미한다.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
841205	51621125	841205	51621125	none

Detail

Instance

Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- YUV_SCALE_LOOP_X	240400	14749440	1202 ~ 7682	-	-	200 ~ 1920	no
+ YUV_SCALE_LOOP_Y	1200	7680	6	-	-	200 ~ 1280	no

(a) ZedBoard

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
961205	58993925	961206	58993926	none

Detail

Instance

Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- YUV_SCALE_LOOP_X	280400	17207040	1402 ~ 8962	-	-	200 ~ 1920	no
+ YUV_SCALE_LOOP_Y	1400	8960	7	-	-	200 ~ 1280	no

(b) Zybo

3-1-8. Analysis Perspective View 를 열고 YUV_SCALE_LOOP_X 항목을 확장한 다음 YUV_SCALE_LOOP_Y 항목을 확장하여 이를 확인할 수 있다.

Current Module : vuv filter										
	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8
1	in width read(read)									
2	in height read(r...									
3	rgb2yuv(function)									
4	V scale read(read)									
5	U scale read(read)									
6	Y scale read(read)									
7	YUV SCALE LOOP X									
8	x i(phi mux)									
9	exitcond1 i(icmp)									
10	x(+)									
11	tmp 2(+)									
12	YUV SCALE LOOP Y									
13	y i(phi mux)									
14	exitcond i(icmp)									
15	y(+)									
16	tmp 3(+)									
17	Y(read)									
18	U(read)									
19	V(read)									
20	tmp 7 i(*)									
21	tmp i(*)									
22	tmp 8 i(*)									
23	node 89(write)									
24	node 91(write)									
25	node 93(write)									
26	yuv2rgb(function)									
27	node 102(write)									
28	node 104(write)									

3-1-9. Report Tab 에서 Utilization Estimates 의 Detail -> Instance 섹션을 확장하고 grp_rgb2yuv_fu_244(rgb2yuv) 항목을 클릭하여 보고서를 연다.

3-1-10. rgb2yuv 함수와 관련된 아래 질문에 답하라.

Estimated Clock Period:

Worst case Latency:

Number of DSP48E used:

Number of FFs used:

Number of LUTs used:

3-1-11. 마찬가지로 yuv2rgb 보고서를 연다.

3-1-12. yuv2rgb 함수와 관련된 아래 질문에 답하라.

Estimated Clock Period:

Worst case Latency:

Number of DSP48E used:

Number of FFs used:

Number of LUTs used:

3-1-13. rgb2yuv 함수의 경우 ZedBoard 의 경우 최악의 대기 시간은 17207040 Clock Cycle 로 보고된다.
보고된 대기 시간은 아래와 같이 추정할 수 있다.


- * RGB2YUV_LOOP_Y 총 Loop Latency = $8 \times 1280 = 10240$ Cycles
- * Loop 에 대한 1 entry 및 1 exit Clock RGB2YUV_LOOP_Y = 10242 Cycles
- * RGB2YUV_LOOP_X Loop 본문 Latency = 10242 Cycles
- * RGB2YUV_LOOP_X 총 Loop Latency = $10242 \times 1920 = 19664640$ Cycles

3-1-14. rgb2yuv 함수의 경우 Zybo 의 경우 최악의 대기 시간은 19664640 Clock Cycle 로 보고된다.
보고된 대기 시간은 아래와 같이 추정할 수 있다.

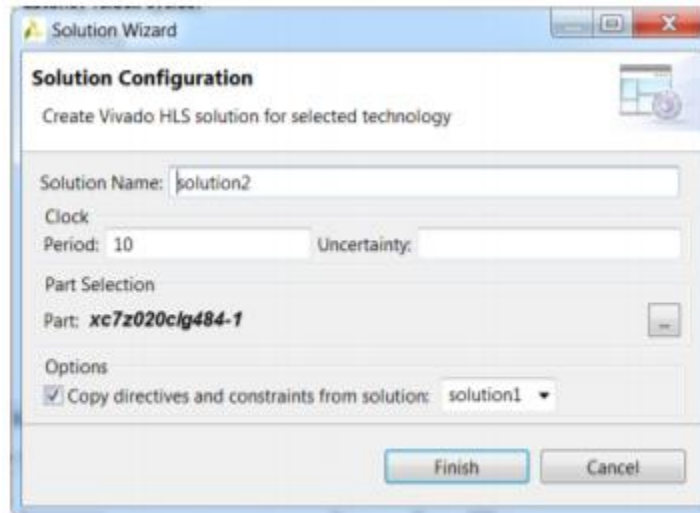
- * RGB2YUV_LOOP_Y 총 Loop Latency = $8 \times 1280 = 10240$ Cycles
- * Loop 에 대한 1 entry 및 1 exit Clock RGB2YUV_LOOP_Y = 10242 Cycles
- * RGB2YUV_LOOP_X Loop 본문 Latency = 10242 Cycles
- * RGB2YUV_LOOP_X 총 Loop Latency = $10242 \times 1920 = 19664640$ Cycles

Turn OFF INLINE and Apply PIPELINE Directive

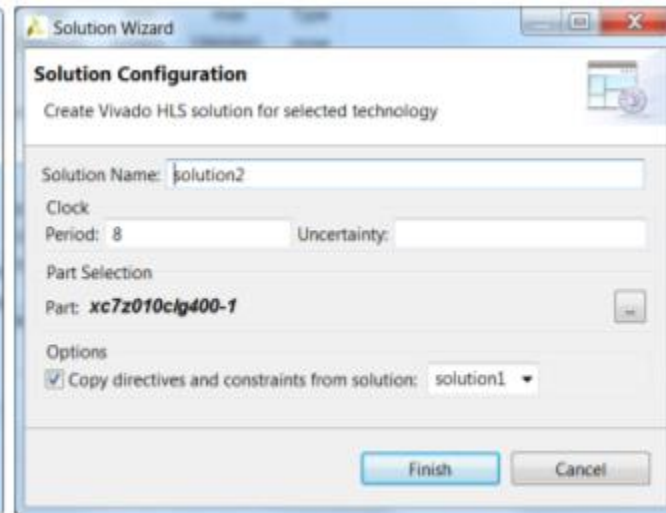
- 4-1. 이전 Solution 설정을 복사하여 새로운 Solution 을 만든다.
자동 INLINE 을 방지하고 PIPELINE 지시문을 적용하라.
Solution 을 생성하고 출력을 이해하라.

- 4-1-1. Project -> New Solution 을 선택하고
 형태로 생긴 마름모에 눈빛 번쩍 아이콘을 Tool Bar 에서 찾아 누른다.

- 4-1-2. Solution Configuration Dialog Box 가 나타날 것이다.
Solution1 에서 기존 지시문 복사 및
사용자 지정 제약 지시문 복사의 확인란이 선택되어 있는지 확인한다.
마침 버튼을 클릭하여 기본 설정으로 새 솔루션을 만든다.



(a) ZedBoard



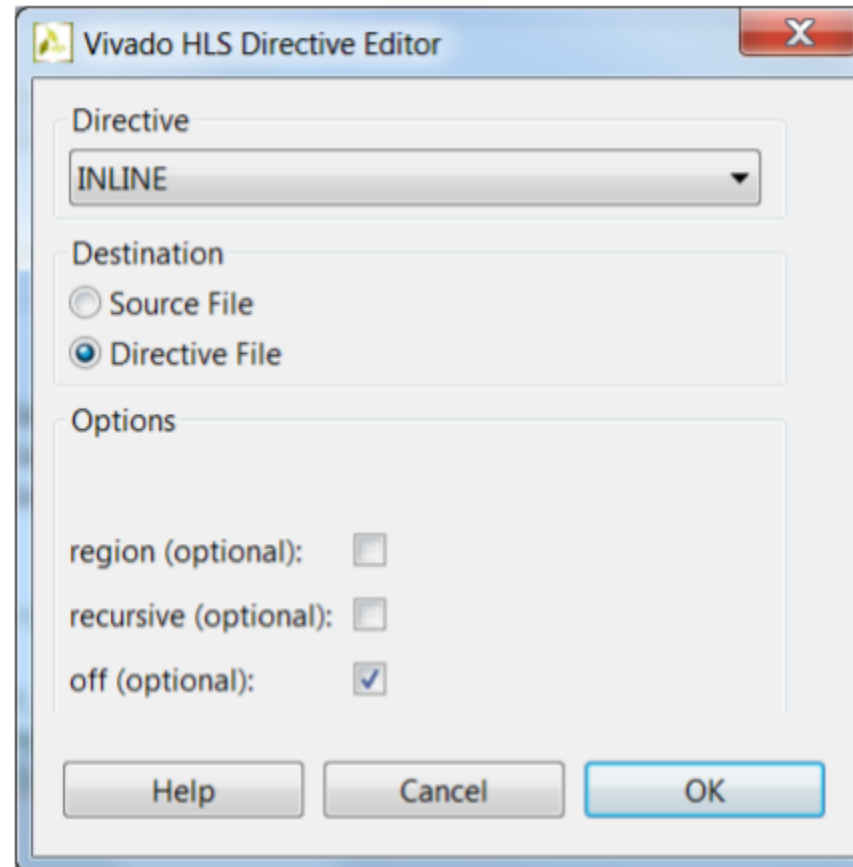
(b) Zybo

4-1-3. yuv_filter.c 소스가 열리고 Information Pane 에 표시되는지 확인한 다음 지시문 탭을 클릭하라.

4-1-4. Directives Pane 에서 yuv_scale 함수를 선택하고 마우스 오른쪽 버튼으로 클릭한 다음 Insert Directive... 를 선택한다.

4-1-5. Directive 필드의 Drop-Down 버튼을 클릭하라.
다양한 지시문을 나열하는 Pop-Up 메뉴가 표시된다.
INLINE 지시문을 선택한다.

4-1-6. Vivado HLS Directive Editor Dialog Box 에서 off 옵션을 클릭하여 자동 inlining 을 끈다.
Directive File 이 선택되었는지 확인하라.
OK 를 누른다.



- * Object(함수 혹은 루프)가 Pipeline 될 때,
그 아래의 모든 Loop 가 계층 구조를 통해 자동으로 풀린다.
- * Loop 를 풀기 위해 고정된 경계선이 있어야한다:
이 Design 의 모든 Loop 는 입력 변수를 사용하여
최상위 함수에 정의된 변수 범위를 갖는다.
- * Loop 의 TRIPCOUNT 지시어는 report 에만 영향을 주며
합성 경계를 설정하지 않는다는 점에 유의하라.
- * 이 예에서 최상위 함수도 Pipeline 화 되어있지 않다.
- * Pipeline 지시문은 각 함수의 가장 안쪽 Loop 에 적용되어야 한다.
가장 안쪽의 Loop 에는 내부에 가변 경계 Loop 가 없으며
외부 Loop 에 Data 가 공급되도록 유지한다.

4-1-7. Directive Tab 에서 yuv_scale 을 확장하고
YUV_SCALE_LOOP_Y 객체를 마우스 오른쪽 버튼으로 클릭한 다음
insert directives ... 을 선택하고 지시문으로 PIPELINE 을 선택한다.

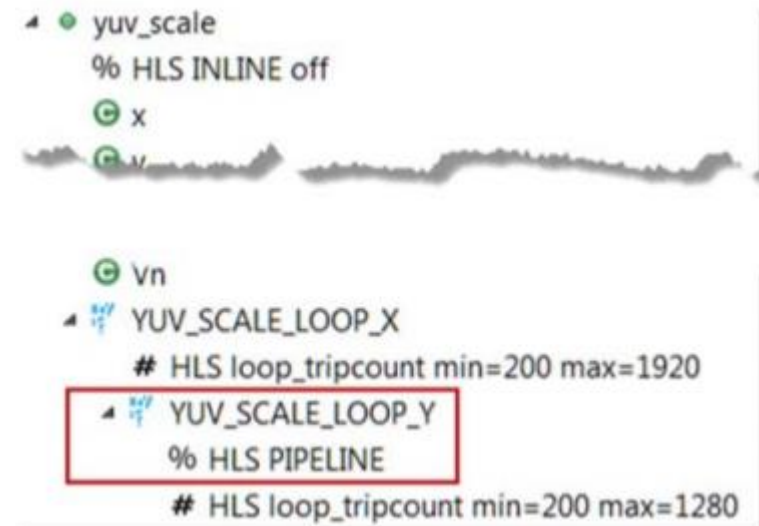
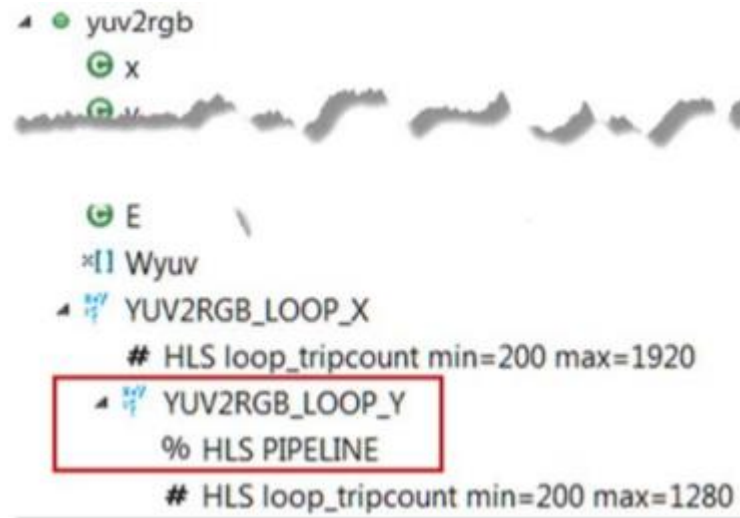
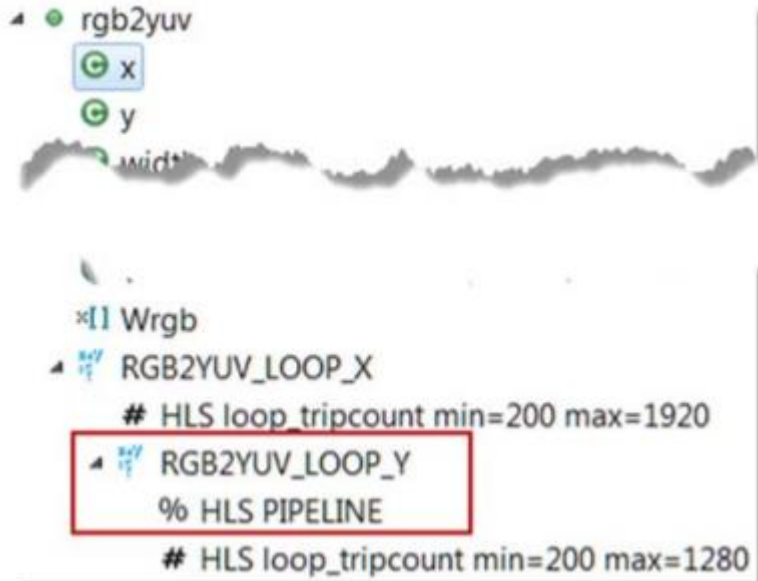
4-1-8. Vivado HLS 가 매 클럭주기마다 $II = 1$,
하나의 새로운 입력을 시도할 때 II (Initiation Interval)를 비워둔다.

4-1-9. OK 를 클릭한다.

4-1-10. 마찬가지로 PIPELINE 지시문을

YUV2RGB_LOOP_Y 및 RGB2YUV_LOOP_Y 객체에 적용한다.

이 시점에서 Directive Tab 이 보여지는 것을 볼 수 있다.



4-1-11. Synthesis 버튼을 클릭한다.

4-1-12. Synthesis 가 완료되면 Project -> Compare Reports ... 를 선택하거나

창과 창 사이에 이중 연결리스트 아이콘을 클릭하여 2 개의 결과를 비교할 수 있다.

4-1-13. Available Reports 에서 Solution1 과 Solution2 를 선택하고 Add >> 버튼을 클릭한다.

4-1-14. Latency 가 56536325 에서 7372831 (ZedBoard) 로, 58993925 에서 7372828 (Zybo) 까지 감소한 것을 관찰하라.

All Compared Solutions

[solution2:](#) xc7z020clg484-1

[solution1:](#) xc7z020clg484-1

Performance Estimates

Timing (ns)

Clock		solution2	solution1
ap_clk	Target	10.00	10.00
	Estimated	10.85	10.85

Latency (clock cycles)

		solution2	solution1
Latency	min	120028	841205
	max	7372828	51621125
Interval	min	120028	841205
	max	7372828	51621125

(a) ZedBoard

All Compared Solutions

[solution2:](#) xc7z010clg400-1

[solution1:](#) xc7z010clg400-1

Performance Estimates

Timing (ns)

Clock		solution2	solution1
ap_clk	Target	8.00	8.00
	Estimated	9.63	8.71

Latency (clock cycles)

		solution2	solution1
Latency	min	120035	1001205
	max	7372835	61451525
Interval	min	120035	1001205
	max	7372835	61451525

(b) Zybo

Solution1 에서, 가장 안쪽에 있는 Loop 의 전체 Loop Latency 는 $\text{loop_body_latency} \times \text{Loop 반복 횟수}$ 였지만
Solution2 에서는 가장 안쪽 Loop 의 새로운 전체 Loop Latency 가 $\text{loop_body_latency} + \text{Loop 반복 횟수}$ 다.

4-1-15. Comparison Report 에서 아래로 스크롤하여 Resource 사용률을 본다.
FFs, LUT 및 DSP48E 사용률은 증가했지만 BRAM 은 동일하게 유지된다.

Utilization Estimates		
	solution2	solution1
BRAM_18K	12288	12288
DSP48E	9	6
FF	1297	688
LUT	2051	1482

(a) ZedBoard

Utilization Estimates		
	solution2	solution1
BRAM_18K	12288	12288
DSP48E	9	6
FF	1593	785
LUT	2083	1494

(b) Zybo

Apply DATAFLOW Directive and Configuration Command

5-1. 이전 Solution2 설정을 복사하여 새 솔루션을 만든다.
DATAFLOW 지시문을 적용하라.
솔루션을 생성하고 출력을 이해하라.

5-1-1. Project -> New Solution 을 선택한다.

5-1-2. Solution Configuration Dialog Box 가 나타날 것이다.
Finish 버튼(Solution2 에서 사본이 선택됨)을 클릭한다.

5-1-3. Project -> Close Inactive Solution Tabs 를 선택하여 모든 비활성 솔루션 창을 닫는다.

5-1-4. Information Pane 에서 yuv_filter.c 소스가 열려 있는지 확인하고 Directive 탭을 선택하라.

5-1-5. Directives Pane 에서 yuv_filter 함수를 선택하고 마우스 오른쪽 버튼으로 클릭한 다음 Insert Directive... 를 선택한다.

5-1-6. 다양한 지시문을 나열하는 Pop-Up 메뉴가 표시된다.
DATAFLOW 지시문을 선택하고 확인을 클릭하라.

5-1-7. Synthesis 버튼을 클릭한다.

5-1-8. Synthesis 가 완료되면 Synthesis Report 가 자동적으로 열린다.

5-1-9. Performance Estimates 섹션에서 추가 정보, Dataflow Type 을 확인하라.

Performance Estimates

[-] **Timing (ns)**

[-] **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	11.02	1.25

[-] **Latency (clock cycles)**

[-] **Summary**

Latency		Interval		Type
min	max	min	max	
120025	7372825	40009	2457609	dataflow

(a) ZedBoard

Performance Estimates

[-] **Timing (ns)**

[-] **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	9.63	1.00

[-] **Latency (clock cycles)**

[-] **Summary**

Latency		Interval		Type
min	max	min	max	
120031	7372831	40011	2457611	dataflow

(b) Zybo

- * Dataflow Pipeline 처리량은 각 입력 집합 사이의 Cycle 수를 나타낸다.
이 처리량 값이 설계 대기 시간보다 작으면 전류 입력 데이터가 출력되기 전에 설계가 새로운 입력을 처리하기 시작할 수 있음을 나타낸다.
- * 전반적인 대기 시간은 크게 변하지 않았지만 데이터 흐름 처리량은 설계가 이론적인 한계($1920 * 1280 = 2457600$)에 근접할 때마다 클럭 주기마다 한 픽셀을 처리할 수 있음을 보여준다.

5-1-10. Utilization Estimates 로 스크롤하여 필요한 BRAM 수가 2배로 증가했는지 확인한다.
 이는 기본 데이터 흐름 ping-pong Buffering 때문이다.

Utilization Estimates					Utilization Estimates				
Summary					Summary				
Name	BRAM_18K	DSP48E	FF	LUT	Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-	DSP	-	-	-	-
Expression	-	-	0	200	Expression	-	-	0	200
FIFO	0	-	35	172	FIFO	0	-	35	172
Instance	0	11	1188	1834	Instance	0	11	1426	1834
Memory	12288	-	96	0	Memory	12288	-	96	0
Multiplexer	-	-	-	90	Multiplexer	-	-	-	90
Register	-	-	10	-	Register	-	-	10	-
Total	12288	11	1329	2296	Total	12288	11	1567	2296
Available	280	220	106400	53200	Available	120	80	35200	17600
Utilization (%)	4388	5	1	4	Utilization (%)	10240	13	4	13

(a) ZedBoard

(b) Zybo

- * DATAFLOW 최적화가 수행되면 Memory Buffer 가 함수 사이에 자동으로 삽입되어 이전 함수가 완료되기 전에 다음 함수가 작업을 시작할 수 있다.
 기본 Memory Buffer 는 가장 큰 생산자 혹은 소비자 배열을 완전히 수용할 수 있는 크기의 ping-pong Buffer 다.
- * Vivado HLS 는 Memory Buffer 가 기본 ping-pong Buffer 혹은 FIFO 가 되도록 한다.
 이 Design 에는 완전히 순차적인 data 접근이 있기 때문에 FIFO 를 사용할 수 있다.
 FIFO 를 사용하는 또 다른 이점은 FIFO 의 크기를 직접 제어할 수 있다는 것이다.
 (임의 Access 가 허용되는 ping-pong Buffer 에서는 불가능함)

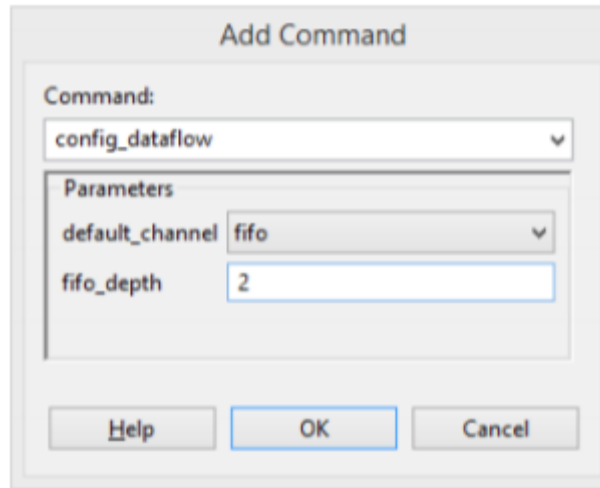
5-1-11. Vivado HLS Configuration Command 를 사용하여 Memory Buffer 타입을 선택할 수 있다.

5-2. Dataflow 구성 명령을 적용하고 솔루션을 생성하며 향상된 리소스 사용률을 관찰한다.

5-2-1. Solution -> Solution Settings... 를 선택하거나  톱니바퀴 아이콘을 클릭하여 구성 명령 설정에 접근한다.

5-2-2. Configuration Settings Dialog Box 에서 General 을 선택하고 Add... 버튼을 클릭한다.

5-2-3. Drop-Down 버튼과 FIFO 를 default_channel 로 사용하여 config_dataflow 를 명령으로 선택하라.
fifo_depth 로 2 를 입력하라.
확인을 클릭하라.



5-2-4. OK 를 다시 누른다.

5-2-5. Synthesis 버튼을 클릭한다.

5-2-6. Synthesis 가 완료되면 Synthesis Report 가 자동으로 열린다.

5-2-7. Performance 매개 변수는 변경되지 않았다;

그러나 자원 추정에 따르면 Design 에 BRAM 이 사용되지 않고 다른 리소스(FF, LUT) 사용량도 줄었다.

Utilization Estimates					Utilization Estimates				
Summary					Summary				
Name	BRAM_18K	DSP48E	FF	LUT	Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-	DSP	-	-	-	-
Expression	-	-	0	8	Expression	-	-	0	8
FIFO	0	-	65	292	FIFO	0	-	65	292
Instance	0	11	855	1786	Instance	0	11	1116	1818
Memory	-	-	-	-	Memory	-	-	-	-
Multiplexer	-	-	-	-	Multiplexer	-	-	-	-
Register	-	-	-	-	Register	-	-	-	-
Total	0	11	920	2086	Total	0	11	1181	2118
Available	280	220	106400	53200	Available	120	80	35200	17600
Utilization (%)	0	5	~0	3	Utilization (%)	0	13	3	12

(a) ZedBoard

(b) Zybo

Export and Implement the Design in Vivado HLS

6-1. Vivado HLS 에서 VHDL 을 언어로 선택하여 Design 을 내보내고 Evaluate 옵션을 선택하여 구현을 실행한다.

6-1-1. Vivado HLS 에서 Solution -> Export RTL 를 선택하거나

Java 패키지 모양의 아이콘을 클릭하여 Dialog Box 를 열면 원하는 Implementation 을 실행할 수 있다.

Export RTL Dialog Box 가 열릴 것이다.

6-1-2. Evaluate Generated RTL 필드의 Drop-Down 버튼을 클릭하고 VHDL 을 언어로 선택하고 아래의 확인란을 클릭하라.

- 6-1-3. OK 를 클릭하면 구현 실행이 시작된다.
 Vivado HLS Console 창에서 진행 상황을 볼 수 있다.
 실행이 완료되면 Implementation Report 가 Information Pane 에 표시된다.

Export Report for 'yuv_filter'

General Information

Report date: Wed Feb 14 07:41:30 -0800 2018
 Project: yuv_filter.prj
 Solution: solution3
 Device target: xc7z020clg484-1
 Implementation tool: Xilinx Vivado v.2017.4

Resource Usage

	VHDL
SLICE	315
LUT	782
FF	700
DSP	11
BRAM	0
SRL	68

Final Timing

	VHDL
CP required	10.000
CP achieved post-synthesis	8.918
CP achieved post-implementation	9.227

Timing met

(a) ZedBoard

Export Report for 'yuv_filter'

General Information

Report date: Sat Feb 24 08:37:33 -0800 2018
 Project: yuv_filter.prj
 Solution: solution4
 Device target: xc7z010clg400-1
 Implementation tool: Xilinx Vivado v.2017.4

Resource Usage

	VHDL
SLICE	340
LUT	772
FF	826
DSP	11
BRAM	0
SRL	69

Final Timing

	VHDL
CP required	8.000
CP achieved post-synthesis	7.158
CP achieved post-implementation	7.099

Timing met

(b) Zybo

Implementation 은 ZedBoard 의 경우 성공했지만 Zybo 의 경우 실패했다.

- 6-1-4. File -> Exit 를 선택하여 Vivado HLS 를 닫는다.

Conclusion

이 Lab 에서 이 Design 이 최상위 수준에서 Pipeline 될 수 없지만 개별 Loop 를 Pipelining 한 다음 Dataflow 최적화를 사용하여 기능을 병렬로 작동시키는 전략은 Clock 당 하나의 Pixel 을 처리하면서 동일한 높은 처리량을 달성할 수 있었다.

DATAFLOW 지시문을 적용하면 기본 Memory Buffer(ping-pong 유형)가 함수 사이에 자동으로 삽입된다.

Design 이 순차적(Streaming) Data 접근만 사용한다는 사실을 이용하여

Dataflow 최적화와 관련된 값 비싼 Memory Buffer 를

Dataflow 명령 구성을 사용하는 간단한 2 요소 FIFO 로 대체할 수 있었다.

Answers

1. Answer the following questions for yuv_filter:

Estimated clock period:	<u>10.85 ns (ZedBoard) 8.71 ns (Zybo)</u>
Worst case latency:	<u>51621125 (ZedBoard) 61451525 (Zybo) clock cycles</u>
Number of DSP48E used:	<u>6</u>
Number of BRAMs used:	<u>12288</u>
Number of FFs used:	<u>688 (ZedBoard) 785 (Zybo)</u>
Number of LUTs used:	<u>1482 (ZedBoard) 1494 (Zybo)</u>

2. Answer the following questions for rgb2yuv:

Estimated clock period:	<u>10.28 ns (ZedBoard) 6.42 ns (Zybo)</u>
Worst case latency:	<u>17207041 (ZedBoard) 22122241 (Zybo) clock cycles</u>
Number of DSP48E used:	<u>3</u>
Number of FFs used:	<u>203 (ZedBoard) 249 (Zybo)</u>
Number of LUTs used:	<u>514 (ZedBoard) 520 (Zybo)</u>

3. Answer the following questions for yuv2rgb:

Estimated clock period:	<u>10.85 ns (ZedBoard) 8.71 ns (Zybo)</u>
Worst case latency:	<u>19664641 (ZedBoard) 22122241 (Zybo) clock cycles</u>
Number of DSP48E used:	<u>3</u>
Number of FFs used:	<u>195 (ZedBoard) 221 (Zybo)</u>
Number of LUTs used:	<u>438 (ZedBoard) 441 (Zybo)</u>

References

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