

Improving Resource Utilization

Innova Lee(이상훈)
gcccompil3r@gmail.com

Introduction

이 Lab 에서는 Vivado HLS 에서 Area 및 Resource 사용은 물론 Design 성능을 향상시키는데 사용할 수 있는 다양한 기법과 지침을 소개한다. 고려중인 Design 은 8 x 8 Block 의 데이터에 대해 DCT(Discrete Cosine Transformation) 을 수행한다.

Objectives

이 Lab 을 완료한 이후 아래를 수행할 수 있다:

- * Design 에 Directives 를 추가한다.
- * PIPELINE Directive 를 사용하여 성능 향상
- * DATAFLOW Directive 와 구성 명령 기능을 구별한다.
- * Memory Partitions 기술을 적용하여 Resource 사용을 향상

Procedure

이 Lab 은 세부적인 지침에 대한 정보를 제공하는 일반적인 개요부터 단계별로 구성된다. Lab 을 진행하려면 이 세부 지침을 따르라.

이 Lab 은 9 가지 기본 단계로 구성된다:

Vivado HLS 명령 프롬프트에서 Design 의 유효성 검사,
Vivado HLS GUI 를 사용하여 새 프로젝트 만들기,
Design 합성, RTL Simulation 실행, 성능 개선을 위한 PIPELINE 지시문 적용,
PARTITION 지시문 적용을 통한 Memory Bandwidth 개선,
DATAFLOW 지시문 적용,
INLINE 적용,
마지막으로 RESHAPE 지시문을 적용한다.

General Flow for this Lab

1. Design 검사
2. 새로운 프로젝트 생성
3. Design 합성
4. RTL Simulation 을 수행
5. PIPELINE 지시어 적용
6. Memory Bandwidth 를 향상
7. DATAFLOW 지시어 적용
8. INLINE 지시어 적용
9. RESHAPE 지시어 적용

Validate the Design from Command Line

1-1. Vivado HLS Command Line 에서 Design 검증하기

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 lab3 으로 변경한다.

- 1-1-3. 자체 점검 프로그램(dct_test.c)가 제공된다.
이것을 사용하여 Design 을 검증할 수 있다.
Makefile 도 제공된다.
Makefile 을 사용하여 필요한 Source 를 컴파일하고 컴파일 된 프로그램을 실행할 수 있다.
Vivado HLS Command Prompt 에서 make 를 입력하여 Program 을 컴파일하고 실행한다.

```
c:\xup\hls\labs\lab3>make
gcc -ggdb -w -I/c/Xilinx/Vivado/2017.4/include -c -o dct.o dct.c
gcc -lm -lstdc++ dct.o dct_test.o -o dct
./dct
    xxx xxx xxx xxx
    Results are good
    xxx xxx xxx xxx

c:\xup\hls\labs\lab3>
```

소스 파일(dct.c 및 dct_test.c) 가 컴파일 된 이후 dct 실행 프로그램이 작성 되고 그 다음에 실행되었음을 유의하라)
프로그램은 설계를 테스트하고 결과는 good 메시지에 해당한다.

- 1-1-4. exit 를 입력하여 명령 프롬프트 창을 닫는다.

Create a New Project

- 2-1. Vivado HLS GUI 에서 XC7Z020CLG484-1 (ZedBoard) 또는 XC7Z010CLG400-1 (Zybo)을 대상으로 새 프로젝트를 만든다.
- 2-1-1. Vivado HLS 시작: 시작 > 모든 프로그램 > 자일링스 디자인 도구 > Vivado 2017.4 > Vivado HLS > Vivado HLS 2017.4 를 선택
- 2-1-2. Vivado HLS GUI 에서 File > New Project 를 선택
New Vivado HLS 프로젝트 마법사가 열린다.

2-1-3. Browse ... 버튼을 클릭하고 lab3 로 이동한 이후 OK 를 클릭한다.

2-1-4. 프로젝트 이름에 dct.prj 를 입력한다.

2-1-5. Next 를 클릭한다.

2-1-6. Source 파일에 대한 파일 추가/제거에서 함수 이름으로 dct 를 입력한다.
(제공된 소스 파일에는 dct 라는 합성된 함수가 포함되어 있다)

2-1-7. Add Files ... 버튼을 클릭하고 lab3 에 dct.c 파일을 선택한 다음 Open 을 클릭한다.

2-1-8. Next 를 클릭한다.

2-1-9. testbench 에 대한 파일 추가/제거에서 Add Files ... 버튼을 클릭하고 lab3 폴더에서 dct_test.c, in.dat, out.golden.dat 파일을 선택하고 Open 을 클릭하라.

2-1-10. Next 를 클릭한다.

2-1-11. 솔루션 구성 페이지에서 솔루션 이름 필드를 solution1 로 두고 Clock Period 를 10 (ZedBoard 의 경우) 또는 8 (Zybo 의 경우) 로 설정한다.
불확실성 필드를 비워두면 ZedBoard 의 기본값으로 1.25, Zybo 의 기본값으로 1.25 가 설정 된다.

2-1-12. Part's Browse 버튼을 클릭하고 Parts Specify 옵션을 사용하여
xc7z020clg484-1 (ZedBoard) 또는 xc7z010clg400-1 (Zybo)을 선택하고 확인을 클릭하여 아래 필터를 선택하라:

Family: Zynq

Sub-Family: Zynq

Package: clg484 (ZedBoard) 혹은 clg400 (Zybo)

Speed Grade: -1

2-1-13. Finish 를 클릭한다.

2-1-14. 원본 폴더 아래의 dct.c 를 두 번 클릭하여 Information Pane 에서 해당 내용을 Open 한다.

```
78 void dct(short input[N], short output[N])
79 {
80
81     short buf_2d_in[DCT_SIZE][DCT_SIZE];
82     short buf_2d_out[DCT_SIZE][DCT_SIZE];
83
84     // Read input data. Fill the internal buffer.
85     read_data(input, buf_2d_in);
86
87     dct_2d(buf_2d_in, buf_2d_out);
88
89     // Write out the results.
90     write_data(buf_2d_out, output);
91 }
```

최상위 함수 dct 는 78 행에서 정의 된다.

이것은 먼저 1D DCT 를 통해 입력 배열의 각 행을 처리 한 다음 동일한 1D DCT 를 통해 결과 배열의 열을 처리하여 2D DCT 알고리즘을 구현한다.
read_data, dct_2d 및 write_data 함수를 호출한다.

read_data 함수는 54 행에서 정의되며 두 개의 Loop(RD_Loop_Row 및 RD_Loop_Col)로 구성된다.

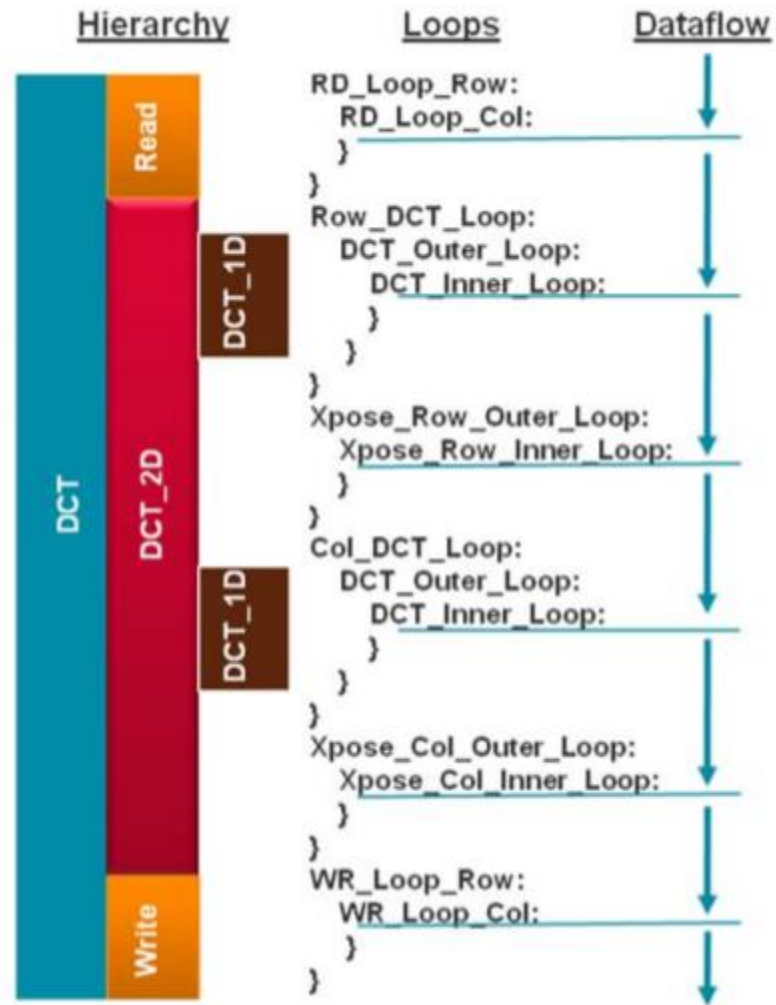
write_data 함수는 66 행에 정의되며 결과 쓰기를 수행하는 두 개의 Loop 로 구성된다.

23 행에 정의된 dct_2d 함수는 dct_1d 함수를 호출하고 전치를 수행한다.

마지막으로 4 행에 정의된 dct_1d 함수는 dct_coeff_table 을 사용하고 1D Type-II DCT 알고리즘의 기본 반복 형식을 구현하여 필요한 기능을 수행한다.

다음 그림은 왼쪽에서 함수 계층 구조, 실행중인 순서의 Loop 를 보여주고

오른쪽에서는 데이터 흐름을 보여준다.



Synthesize the Design

3-1. 기본값을 사용하여 설계를 합성한다.

합성 결과를 보고 이 단계의 세부 절차에 나열된 질문에 답해보도록 한다.

3-1-1. Solution > Run C Synthesis > Active Solution 을 선택하거나 버튼을 클릭하여 합성 절차를 시작한다.

3-1-2. 합성이 완료되면 여러 개의 보고서 파일에 접근할 수 있고 합성 결과가 정보 창에 표시된다.

Explorer View 의 Synthesis Report 섹션에는 dct_1d.rpt, dct_2d.rpt 및 dct.rpt 항목만 표시된다.

read_data 및 write_data 함수 보고서는 나열되지 않는다.

이 두 함수가 inline 되기 때문이다.

Vivado HLS Console View 로 Scroll 하여 확인하라.

```
INFO: [XFORM 203-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.  
INFO: [XFORM 203-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.  
INFO: [HLS 200-111] Finished Checking Synthesizability Time (s): cpu = 00:00:02 ; elapsed  
INFO: [XFORM 203-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.  
INFO: [XFORM 203-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
```


3-1-3. Synthesis Report 는 설계의 예상 대기 시간뿐만 아니라 성능 및 자원 예측도 표시한다(Zybo 의 경우 Target Period 가 8.00 일 것이다)
설계가 최적화되지 않았거나 파이프라인화 되어 있지 않다.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.51	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
3959	3959	3959	3959	none

Detail

Instance

Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- RD_Loop_Row	144	144	18	-	-	8	no
+ RD_Loop_Col	16	16	2	-	-	8	no
- WR_Loop_Row	144	144	18	-	-	8	no
+ WR Loop Col	16	16	2	-	-	8	no

3-1-4. 오른쪽에 Scroll Bar 를 사용하여 아래로 Scroll 하여 보고서에 들어가 아래 부분을 확인해보도록 한다.

Question 1

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. 이 보고서에는 도구로 생성된 최상위 인터페이스 신호도 표시된다.

Interface					
Summary					
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	dct	return value
ap_rst	in	1	ap_ctrl_hs	dct	return value
ap_start	in	1	ap_ctrl_hs	dct	return value
ap_done	out	1	ap_ctrl_hs	dct	return value
ap_idle	out	1	ap_ctrl_hs	dct	return value
ap_ready	out	1	ap_ctrl_hs	dct	return value
input_r_address0	out	6	ap_memory	input_r	array
input_r_ce0	out	1	ap_memory	input_r	array
input_r_q0	in	16	ap_memory	input_r	array
output_r_address0	out	6	ap_memory	output_r	array
output_r_ce0	out	1	ap_memory	output_r	array
output_r_we0	out	1	ap_memory	output_r	array
output_r_d0	out	16	ap_memory	output_r	array

ap_clk, ap_rst 가 자동으로 추가 된 것을 볼 수 있다.
ap_start, ap_done, ap_idle 및 ap_ready 는 설계가
다음 계산 명령(ap_idle)을 받을 수 있는지,
다음 계산이 시작될 때(ap_start),
계산이 완료 될 때를 나타내는
Hand Shaking 신호로 사용되는 최상위 신호다(ap_done).
최상위 함수는 입력 및 출력 배열을 가지므로
각각에 대해 ap_memory 인터페이스가 생성된다.

- 3-1-6. Explorer View 를 사용하거나 정보보기의 dct.rpt 하단에 하이퍼링크를 사용하여 dct_1d.rpt 및 dct_2d.rpt 파일을 연다.
dct_2d 에 대한 보고서는 Desing Cycles(3668)의 대부분이 행 및 열 DCT 를 수행하는데 소비된다는 것을 분명하게 나타낸다.
또한 dct_1d 보고서는 Latency(대기 시간)이 209 Clock Cycle((24 + 2) * 8 + 1)임을 나타낸다.

Run Co-Simulation

- 4-1. Co-Simulation 을 실행하여 Verilog 을 선택한다.
시뮬레이션이 통과하는지 확인한다.
- 4-1-1. Solution > C/RTL Co-simulation 실행을 선택하거나 버튼을 클릭하여 Dialog Box 를 열면 원하는 시뮬레이션을 실행할 수 있다.
C/RTL Co-Simulation Dialog Box 가 열린다.
- 4-1-2. Verilog 옵션을 선택하고 확인을 클릭하여 XSIM 시뮬레이터를 사용하여 Verilog 시뮬레이션을 실행한다.
RTL Co-simulation 이 실행되어 여러 파일을 생성 및 컴파일 한 다음 설계를 시뮬레이션한다.
Console Window 에서 진행 상황과 테스트가 통과되었다는 메시지를 볼 수 있다.

Cosimulation Report for 'dct'

Result							
RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	3959	3959	3959	0	0	0

INFO: [\[COSIM 212-316\]](#) Starting C post checking ...

*** **

Results are good

*** **

INFO: [\[COSIM 212-1000\]](#) *** C/RTL co-simulation finished: PASS ***

Finished C/RTL cosimulation.

Apply PIPELINE Directive

5-1. 이전 솔루션 설정을 복사하여 새 솔루션을 만든다.

DCT_Inner_Loop, Xpose_Row_inner_Loop, Xpose_Col_Inner_Loop, RD_Loop_Col 및 WR_Loop_Col 에 PIPELINE 지시어를 적용한다.
솔루션을 생성하고 출력을 분석한다.

5-1-1. Project > New Solution 을 선택하거나 Tool Bar 에서  버튼을 클릭한다.

5-1-2. Solution Configuration Dialog Box 가 나타난다.

Finish 버튼을 클릭하라(Solution1 에서 복사가 선택됨)

5-1-3. 정보창에 dct.c 소스가 열려 있는지 확인하고 Directive 탭을 클릭하라.

5-1-4. Directive Pane 에서 dct_1d 함수의 DCT_Inner_Loop 을 선택하고 마우스 오른쪽 버튼으로 클릭 한 다음 Insert Directive ... 를 선택한다.

5-1-5. 다양한 지시어를 나열하는 Pop-Up 메뉴가 표시된다.

PIPELINE 지시어를 선택하라.

5-1-6. Vivado HLS 가 매 Clock Cycle 마다 II = 1, 하나의 새로운 입력을 시도 할 때 II(Initiation Interval)를 비워둔다.

5-1-7. 확인을 클릭하라.

5-1-8. 마찬가지로 dip_2d 함수의 Xpose_Row_Inner_Loop 및 Xpose_Col_Inner_Loop 및 read_data 함수의 RD_Loop_Col 및 write_data 함수의 WR_Loop_Col 에 PIPELINE 지시어를 적용한다. 이 시점에서 Directive 탭은 아래와 같이 보인다.

```
└─ ● dct_1d
    └─ ×[1] dct_coeff_table
    └─ └─ ×[1] DCT_Outer_Loop
        └─ └─ ×[1] DCT_Inner_Loop
            └─ % HLS PIPELINE
└─ ● dct_2d
    └─ ×[1] row_outbuf
    └─ ×[1] col_outbuf
    └─ ×[1] col_inbuf
    └─ ×[1] Row_DCT_Loop
    └─ └─ ×[1] Xpose_Row_Outer_Loop
        └─ └─ ×[1] Xpose_Row_Inner_Loop
            └─ % HLS PIPELINE
    └─ ×[1] Col_DCT_Loop
    └─ └─ ×[1] Xpose_Col_Outer_Loop
        └─ └─ ×[1] Xpose_Col_Inner_Loop
            └─ % HLS PIPELINE
└─ ● read_data
    └─ └─ ×[1] RD_Loop_Row
        └─ └─ ×[1] RD_Loop_Col
            └─ % HLS PIPELINE
└─ ● write_data
    └─ └─ ×[1] WR_Loop_Row
        └─ └─ ×[1] WR_Loop_Col
            └─ % HLS PIPELINE
└─ ● dct
```

5-1-9. Synthesis 버튼을 클릭하라.

5-1-10. Synthesis 가 완료되면 Project > Compare Reports ... 선택하거나 클릭하여 두 가지 솔루션을 비교한다.

5-1-11. 사용 가능한 보고서에서 Solution1 및 Solution2 를 선택하고 Add >> 버튼을 클릭한 다음에 OK 를 누른다.

5-1-12. Latency 가 3959 에서 1851 Clock Cycle(ZedBoard)로 감소하고 3959 에서 1855(Zybo)로 감소함을 관찰하라.

Performance Estimates			
[-] Timing (ns)			
Clock		solution2	solution1
ap_clk	Target	10.00	10.00
	Estimated	8.23	6.51
[-] Latency (clock cycles)			
		solution2	solution1
Latency	min	1851	3959
	max	1851	3959
Interval	min	1851	3959
	max	1851	3959

(a) ZedBoard

Performance Estimates			
[-] Timing (ns)			
Clock		solution2	solution1
ap_clk	Target	8.00	8.00
	Estimated	6.51	6.51
[-] Latency (clock cycles)			
		solution2	solution1
Latency	min	1855	3959
	max	1855	3959
Interval	min	1855	3959
	max	1855	3959

(b) Zybo

- 5-1-13. 비교 보고서를 아래로 Scroll 하여 Resource Utilization 을 본다.
FFs 및/또는 LUT 활용도는 증가했지만 BRAM 및 DSP48E 는 동일하게 유지된다.

Utilization Estimates		
	solution2	solution1
BRAM_18K	5	5
DSP48E	1	1
FF	256	278
LUT	1286	982

(a) ZedBoard

Utilization Estimates		
	solution2	solution1
BRAM_18K	5	5
DSP48E	1	1
FF	298	278
LUT	1298	982

(b) Zybo

Apply PIPELINE Directive

- 5-2. Analysis Perspective 를 열고 대부분의 Clock Cycle 이 소비되는 곳, 즉 대기 시간이 긴 곳을 결정한다.

- 5-2-1. Analysis Perspective 버튼을 클릭하라.

- 5-2-2. Module Hierarchy 에서 dct 항목을 선택하고 RD_Loop_Row_RD_Loop_Col 및 WR_Loop_Row_WR_Loop_Col 항목을 관찰한다.
이것은 두 개의 중첩된 Loop 를 Flatten 하게 만들고 내부 Loop 이름을 Outer Loop 이름에 추가하여 새로운 이름을 만든다.
Console View 메시지를 보고 이를 확인 할 수도 있다.

```
INFO: [HLS 200-10] Checking synthesizability ...
INFO: [XFORM 203-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.
INFO: [XFORM 203-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
INFO: [HLS 200-111] Finished Checking Synthesizability Time (s): cpu = 00:00:02 ; elapsed = 00:00:08 . Memory (MB):
INFO: [XFORM 203-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.
INFO: [XFORM 203-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:03 ; elapsed = 00:00:09 . Memory (MB): peak = 117.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' (dct.c:38:1) in function 'dct_2d'.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' (dct.c:49:1) in function 'dct_2d'.
WARNING: [XFORM 203-542] Cannot flatten a loop nest 'DCT_Outer_Loop' (dct.c:13:67) in function 'dct_1d' :
WARNING: [XFORM 203-542] the outer loop is not a perfect loop because there is nontrivial logic in the loop latch.
INFO: [XFORM 203-541] Flattening a loop nest 'RD_Loop_Row' (dct.c:59:67) in function 'dct'.
INFO: [XFORM 203-541] Flattening a loop nest 'WR_Loop_Row' (dct.c:71:67) in function 'dct'.
```

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
• dct	5	1	256	1286	1851	1852	none	
• dct_2d	3	1	195	843	1718	1718	none	
• dct_1d2	0	1	117	232	97	97	none	

Performance Profile					Resource Profile			
	Pipelined	Latency	Initiation Interval	Iteration Latency				
• dct	-	1851	1852	-				
• RD_Loop_Row_RD_Loop_Col	yes	64	1	2				
• WR_Loop_Row_WR_Loop_Col	yes	64	1	2				

(a) ZedBoard

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
• dct	5	1	298	1298	1855	1856	none	
• dct_2d	3	1	215	849	1720	1720	none	
• dct_1d2	0	1	117	232	97	97	none	

Performance Profile					Resource Profile			
	Pipelined	Latency	Initiation Interval	Iteration Latency				
• dct	-	1855	1856	-				
• RD_Loop_Row_RD_Loop_Col	yes	65	1	3				
• WR_Loop_Row_WR_Loop_Col	yes	65	1	3				

(b) Zybo

5-2-3. Module Hierarchy 탭에서 `dct > dct_2d` 를 확장하라.
대부분의 대기 시간은 `dct_2d` 함수에서 발생한다.

5-2-4. Module Hierarchy 탭에서 `dct_2d` 모듈에
여전히 계층 구조가 있음을 확인하라.
`dct > dct_2d > dct_1d2` 를 확장하고 `dct_1d` 항목을 선택하라.

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
▲ ● <code>dct</code>	5	1	256	1286	1851	1852	none
▲ ● <code>dct_2d</code>	3	1	195	843	1718	1718	none
● <code>dct_1d2</code>	0	1	117	232	97	97	none

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
▲ ● <code>dct_1d2</code>	-	97	97	-	-
▲ ● <code>DCT_Outer_Loop</code>	no	96	-	12	8
● <code>DCT_Inner_Loop</code>	yes	9	1	3	8

(a) ZedBoard

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
▲ ● <code>dct</code>	5	1	298	1298	1855	1856	none
▲ ● <code>dct_2d</code>	3	1	215	849	1720	1720	none
● <code>dct_1d2</code>	0	1	117	232	97	97	none

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
▲ ● <code>dct_1d2</code>	-	97	97	-	-
▲ ● <code>DCT_Outer_Loop</code>	no	96	-	12	8
● <code>DCT_Inner_Loop</code>	yes	9	1	3	8

(b) Zybo

5-2-5. Performance Profile 탭에서 DCT_Inner_Loop 항목을 선택하고
Performance View 의 C3 상태에 있는 node_60(쓰기) Block 을 마우스 우클릭 한 다음 Goto Source 를 선택한다.
DCT_Outer_Loop 의 병합을 방지하는 19 행이 강조 표시된다.

Current Module : dct > dct 2d > dct 1d2

	Operation\Control Step	C0	C1	C2	C3	C4
1	dst offset read(read)					
2	src offset read(read)					
3	DCT Outer Loop					
4	k(phi mux)					
5	tmp(icmp)					
6	k 1(+)					
7	tmp 16(+)					
8-...	DCT Inner Loop					
19	tmp 12(+)					
20	node 60(write)					

PerformanceResource

PropertiesWarningsC Source

File: C:\xup\hls\labs\lab3\dct.c

12 DCT_Outer_Loop:
13 for (k = 0; k < DCT_SIZE; k++) {
14 DCT_Inner_Loop:
15 for(n = 0, tmp = 0; n < DCT_SIZE; n++) {
16 int coeff = (int)dct_coeff_table[k][n];
17 tmp += src[n] * coeff;
18 }
19 dst[k] = DESCALE(tmp, CONST_BITS);
20 }
21 }

5-2-6. Synthesis Perspective 로 전환한다.

5-3. 이전 솔루션 설정을 복사하여 새 솔루션을 만든다.

내부 Loop 에서 dct_1d 의 외부 Loop 로 PIPELINE 지시어를 이동하며

이동시킨 PIPELINE 지시어를 사용하여 dct_1d 의 내부 Loop 에 대한 곱셈 및 덧셈 연산을 수행하는 Fine-Grain Parallelism 을 적용한다.
Solution 을 생성하고 출력을 분석한다.

5-3-1. Project > New Solution 을 선택한다.

5-3-2. Solution Configuration Dialog Box 가 나타난다.

Finish 버튼(Solution2 가 선택된 상태)를 클릭하라.

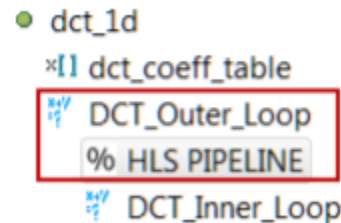
5-3-3. 지시어 창에서 dct_1d 함수의 DCT_Inner_Loop 의 PIPELINE 지시어를 선택하고 마우스 우클릭 버튼을 눌러 다음 지시어 제거를 선택한다.

5-3-4. 지시어 창에서 dct_1d 함수의 DCT_Outer_Loop 를 선택하고 마우스 우클릭 버튼으로 클릭한 다음 Insert Directive ... 를 선택한다.

5-3-5. 다양한 지시어를 나열하는 Pop-Up 메뉴가 표시된다.

PIPELINE 지시어를 선택한다.

5-3-6. OK 를 클릭한다.



5-3-7. Synthesis 버튼을 클릭하라.

5-3-8. Synthesis 가 완료되면 Project > Compare Reports ... 를 선택하여 두 가지 솔루션을 비교하라.

5-3-9. 사용 가능한 보고서에서 Solution2 및 Solution3 을 선택하고 Add >> 버튼을 클릭한 다음 OK 를 클릭한다.

5-3-10. ZedBoard 에 대한 Latency 는 1851 에서 875 Clock Cycle(Zybo 의 경우 1855 ~ 895)로 줄었는지 확인하라.

Performance Estimates			
Timing (ns)			
Clock		solution3	solution2
ap_clk	Target	10.00	10.00
	Estimated	9.40	8.23
Latency (clock cycles)			
		solution3	solution2
Latency	min	875	1851
	max	875	1851
Interval	min	875	1851
	max	875	1851

(a) ZedBoard

Performance Estimates			
Timing (ns)			
Clock		solution3	solution2
ap_clk	Target	8.00	8.00
	Estimated	9.40	6.51
Latency (clock cycles)			
		solution3	solution2
Latency	min	895	1855
	max	895	1855
Interval	min	895	1855
	max	895	1855

(b) Zybo

5-3-11. Comparison Report 를 아래로 Scroll 하여 Resource 사용률을 본다.
BRAM 을 제외한 모든 Resource 의 사용률이 증가했는지 확인한다.
DCT_Inner_Loop 가 전개되었기 때문에 병렬 계산에는 8 개의 DSP48E 가 필요하다.

Utilization Estimates		
	solution3	solution2
BRAM_18K	5	5
DSP48E	8	1
FF	678	256
LUT	1484	1286

(a) ZedBoard

Utilization Estimates		
	solution3	solution2
BRAM_18K	5	5
DSP48E	8	1
FF	736	298
LUT	1496	1298

(b) Zybo

5-3-12. dct_1d 보고서를 열고 pipeline initiation interval(II)로 기대되는 것은 1 이 아닌 4 Cycle 이며 이제 계수 테이블에 8 개의 BRAM 이 사용됨을 관찰하라. Synthesis Log 를 면밀히 살펴보면 계수 테이블이 자동으로 분할되어 8 개의 개별 ROM 이 생성된다는 것을 알 수 있다. 이렇게 하면 풀어헤쳐진 Loop 를 유지하여 Latency 를 줄이는데 도움이 되지만 dct_1d 함수의 입력 배열은 자동으로 분할되지 않았다. II 가 8 개가 아닌 4 개가 예상되는 이유는 Vivado HLS 가 자동으로 듀얼 포트 RAM 을 사용하기 때문에 작업 예약에 도움이 되기 때문이다.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.40	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
36	36	36	36	none

Detail

Instance

Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- DCT_Outer_Loop	34	34	7	4	1	8	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	8	-	-
Expression	-	-	0	247
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	119	16
Multiplexer	-	-	-	125
Register	-	-	420	-
Total	0	8	539	388
Available	280	220	106400	53200
Utilization (%)	0	3	~0	~0

INFO: [XFORM 203-502] Unrolling all sub-loops inside loop 'DCT_Outer_Loop' ([dct.c:13](#)) in function 'dct_1d' for pipelining.
 INFO: [XFORM 203-501] Unrolling loop 'DCT_Inner_Loop' ([dct.c:15](#)) in function 'dct_1d' completely.
 INFO: [XFORM 203-102] Partitioning array 'dct_coeff_table' in dimension 2 automatically.
 INFO: [XFORM 203-602] Inlining function 'read_data' into 'dct' ([dct.c:85](#)) automatically.
 INFO: [XFORM 203-602] Inlining function 'write_data' into 'dct' ([dct.c:90](#)) automatically.

```
INFO: [HLS 200-10] -- Implementing module 'dct_1d2'
INFO: [HLS 200-10] -----
INFO: [SCHED 204-11] Starting scheduling ...
INFO: [SCHED 204-61] Pipelining loop 'DCT_Outer_Loop'.
WARNING: [SCHED 204-69] Unable to schedule 'load' operation ('src_load_5', dct.c:17) on array 'src' due to limited memory ports
INFO: [SCHED 204-61] Pipelining result: Target II: 1, Final II: 4, Depth: 7.
WARNING: [SCHED 204-21] Estimated clock period (9.4ns) exceeds the target (target clock period: 10ns, clock uncertainty: 1.25ns,
WARNING: [SCHED 204-21] The critical path consists of the following:
    'mul' operation ('tmp_7_7', dct.c:17) (3.36 ns)
    'add' operation ('tmp7', dct.c:19) (3.02 ns)
    'add' operation ('tmp6', dct.c:19) (3.02 ns)
```


5-4. Analysis Perspective 로 전환하고
dct_1d Performance View 를 보고 Design Analysis 를 수행한다.

5-4-1. Analysis Perspective 로 전환하고
Module Hierarchy 항목을 확장 한 다음 dct_1d 항목을 선택하라.

5-4-2. 필요한 경우 Profile 탭 항목을 확장하고
DCT_Outer_Loop 가 파이프라인화 되고
DCT_Inner_Loop 항목이 없는 것을 확인한다.

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ ● dct	5	8	678	1484	875	876	none	
▲ ● dct_2d	3	8	617	1032	742	742	none	
● dct_1d2	0	8	539	388	36	36	none	

Performance Profile					
Resource Profile					
	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
▲ ● dct_1d2	-	36	36	-	-
● DCT_Outer_Loop	yes	34	4	7	8

(a) ZedBoard

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ ● dct	5	8	736	1496	895	896	none	
▲ ● dct_2d	3	8	653	1038	760	760	none	
● dct_1d2	0	8	555	388	37	37	none	

Performance Profile					
Resource Profile					
	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
▲ ● dct_1d2	-	37	37	-	-
● DCT_Outer_Loop	yes	35	4	8	8

(b) Zybo

5-4-3. Module Hierarchy 탭에서 dct_1d 항목을 선택하고 DCT_Outer_Loop 가 Performance View 의 8 가지 상태에 걸쳐 있음을 관찰하라.

Current Module : dct > dct 2d > dct 1d2									
	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1	dst offset read(read)								
2	src offset read(read)								
3	tmp 18()								
4	tmp 20()								
5	tmp 22()								
6	tmp 24()								
7	tmp 26()								
8	tmp 28()								
9	tmp 30()								
1...	⊕DCT Outer Loop								

Performance

Resource

5-4-4. Resource 탭을 선택하고 Memory Ports 항목을 확장하고 BRAM src 의 메모리 접근이 매 Clock Cycle 마다 최대 값으로 사용되는지 확인한다.
 (대부분 BRAM 은 이중 포트가 될 수 있으며 두 포트가 모두 사용됨)
 이는 설계가 Memory Resource 에 의해 Bandwidth 가 제한 될 수 있음을 나타내는 좋은 예다.

Current Module : dct > dct 2d > dct 1d2

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1	[-]I/O Ports								
2	dst offset	read							
3	src offset	read							
4	src(p0)		read	read	read	read			
5	src(p1)		read	read	read	read			
6	dst(p0)								write
7	[-]Memory Ports								
8	src(p0)		read	read	read	read			
9	dct coeff table 3...		read						
10	src(p1)		read	read	read	read			
11	dct coeff table 1...		read						
12	dct coeff table 6...			read					
13	dct coeff table 5...			read					
14	dct coeff table 7...			read					
15	dct coeff table 4...			read					
16	dct coeff table 0...			read					
17	dct coeff table 2...			read					
18	dst(p0)								write
1...	[+]Expressions								
Performance Resource									

5-4-5. Synthesis Perspective 로 전환한다.

Improve Memory Bandwidth

6-1. 이전 솔루션(Solution 3) 설정을 복사하여 새 Solution 을 만든다.

병목 현상이 dct_1d 함수의 src 포트와 dct_2d 의 col_inbuf 에 있었기 때문에 dct 의 buf_2d_in 에 ARRAY_PARTITION 지시어를 적용한다.
(dct_2d 함수의 in_block 을 통해 전달되었으며, dct_2d 함수의 buf_2d_in 을 통해 전달되었다)
솔루션을 생성한다.

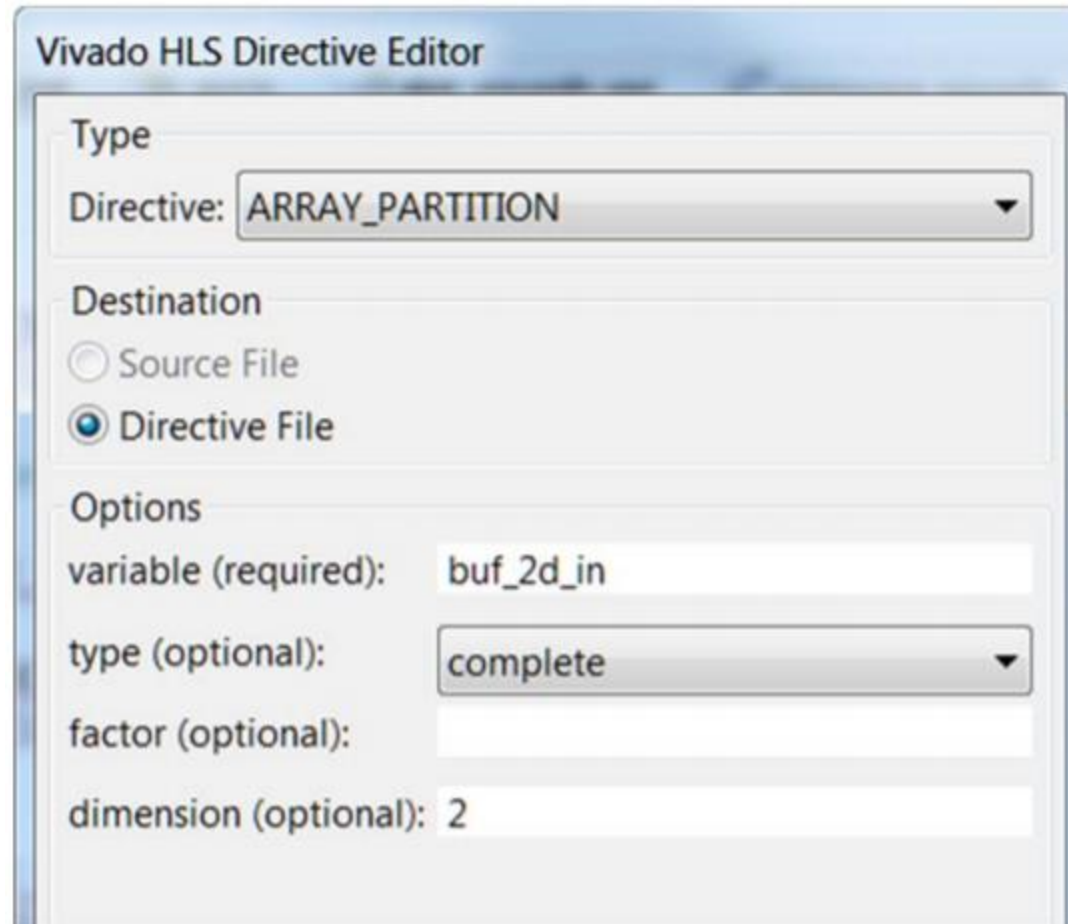
6-1-1. Project > 새로운 솔루션을 선택하여 새 솔루션을 만든다.

6-1-2. Solution Configuration Dialog Box 가 나타난다.
Finish 버튼(Solution3 이 선택된 상태)을 클릭하라.

6-1-3. dct.c 를 열고 지시어 창에서 buf_2d_in 배열 함수를 선택하고 마우스 우클릭을 한 이후 Insert Directive ... 를 선택한다.
dct_2d 함수의 in_block 을 통해 전달된 dct_1d 함수의 src 포트의 병목 현상이 있었기 때문에 buf_2d_in 배열이 선택되었다.
이 함수는 dct 함수의 buf_2d_in 을 통해 전달되었다.

6-1-4. 다양한 지시어를 나열하는 Pop-Up 메뉴가 표시된다.
ARRAY_PARTITION 지시어를 선택한다.

- 6-1-5. Type 이 완전한지 확인한다.
Dimension 필드에 2 를 입력하고 OK 를 클릭한다.



The image shows the 'Vivado HLS Directive Editor' dialog box. It has three main sections: 'Type', 'Destination', and 'Options'. In the 'Type' section, the 'Directive' dropdown is set to 'ARRAY_PARTITION'. In the 'Destination' section, the 'Directive File' radio button is selected. In the 'Options' section, the 'variable (required)' field contains 'buf_2d_in', the 'type (optional)' dropdown is set to 'complete', the 'factor (optional)' field is empty, and the 'dimension (optional)' field contains '2'.

Vivado HLS Directive Editor	
Type	
Directive:	ARRAY_PARTITION
Destination	
<input type="radio"/> Source File	
<input checked="" type="radio"/> Directive File	
Options	
variable (required):	buf_2d_in
type (optional):	complete
factor (optional):	
dimension (optional):	2

- 6-1-6. 마찬가지로 dimension 이 2 인 ARRAY_PARTITION 지시어를 col_inbuf 배열에 적용한다.

- 6-1-7. Synthesis 버튼을 클릭한다.

6-1-8. Synthesis 가 완료되면 Project > Compare Reports ... 를 선택하여 두 가지 솔루션을 비교하라.

6-1-9. 사용 가능한 보고서에서 Solution3 및 Solution4 를 선택하고 Add >> 단추를 클릭하라.

6-1-10. ZedBoard 의 경우 Latency 가 875 ~ 509 Clock Cycle(Zybo 인 경우 895 에서 529) 로 감소했는지 확인하라.

Performance Estimates			
Timing (ns)			
Clock		solution4	solution3
ap_clk	Target	10.00	10.00
	Estimated	9.40	9.40
Latency (clock cycles)			
		solution4	solution3
Latency	min	509	875
	max	509	875
Interval	min	509	875
	max	509	875

(a) ZedBoard

Performance Estimates			
Timing (ns)			
Clock		solution4	solution3
ap_clk	Target	8.00	8.00
	Estimated	9.40	9.40
Latency (clock cycles)			
		solution4	solution3
Latency	min	529	895
	max	529	895
Interval	min	529	895
	max	529	895

(b) Zybo

6-1-11. Comparison 보고서를 아래로 Scroll 하여 Resource 사용률을 본다.
FF Resource 사용량의 증가(거의 두 배)를 관찰하라.

Utilization Estimates		
	solution4	solution3
BRAM_18K	3	5
DSP48E	8	8
FF	1294	678
LUT	2014	1484

(a) ZedBoard

Utilization Estimates		
	solution4	solution3
BRAM_18K	3	5
DSP48E	8	8
FF	1436	736
LUT	2026	1496

(b) Zybo

6-1-12. dct.rpt 항목에서 Loop 항목을 확장하고 Pipeline II 가 이제 1 임을 관찰하라.

6-2. Analysis Perspective 로 전환하고 dct Resource Profile View 를 보고 Resource Analysis 를 수행한다.

6-2-1. Analysis Perspective 로 전환하고 Module Hierarchy Entry 를 확장하고, dct 엔트리를 선택한다.

6-2-2. Resource Profile Tab 을 선택하라.

6-2-3. Memories and Expressions 항목을 확장하고 대부분의 자원이 Instance 에 의해 소비됨을 관찰하라.
buf_2d_in 배열은 여러 개의 메모리로 분할되며 대부분의 연산은 덧셈 및 비교 작업으로 수행된다.

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
└─ ● dct	3	8	1294	2014	509	510	none	
└─ ● dct_2d	2	8	974	1267	374	374	none	
└─ ● read_data	0	0	28	174	66	66	none	

Performance Profile Resource Profile								
	BRAM	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	
└─ ● dct	3	1294	2014					
└─ I/O Ports(2)				32				
└─ Instances(2)	2	1002	1441					
└─ Memories(9)	1	256	16	144			9	
└─ buf_2d_out_U	1	0	0	16			1	
└─ buf_2d_in_6_U	0	32	2	16			1	
└─ buf_2d_in_5_U	0	32	2	16			1	
└─ buf_2d_in_4_U	0	32	2	16			1	
└─ buf_2d_in_3_U	0	32	2	16			1	
└─ buf_2d_in_7_U	0	32	2	16			1	
└─ buf_2d_in_2_U	0	32	2	16			1	
└─ buf_2d_in_1_U	0	32	2	16			1	
└─ buf_2d_in_0_U	0	32	2	16			1	
└─ Σ Expressions(11)	0	0	117	39	44	8		
└─ Registers(11)		36		36				
└─ Channels(0)	0	0	0	0			0	
└─ Multiplexers(33)	0	0	440	69			0	

(a) ZedBoard

Apply DATAFLOW Directive

7-1. 이전 솔루션(Solution4) 설정을 복사하여 새 솔루션을 만든다.
처리량을 향상시키려면 DATAFLOW 지시어를 적용하라.
솔루션을 생성하고 출력을 분석하라.

7-1-1. Project > New Solution 을 선택하라.

7-1-2. Solution Configuration Dialog Box 가 나타난다.
Finish 버튼을 클릭하라(Solution4 가 선택된 상태)

7-1-3. Project > Close Inactive Solution Tabs 를 선택하여 모든 비활성 솔루션 창을 닫으라.

7-1-4. Directive Pane 에서 dct 함수를 선택하고 우클릭 한 다음 Insert Directive ... 를 선택하라.

7-1-5. 처리량을 향상시키려면 DATAFLOW 지시어를 선택하라.

7-1-6. Synthesis 버튼을 클릭하라.

7-1-7. Synthesis 가 완료되면 Synthesis Report 가 자동으로 열린다.

7-1-8. dataflow type pipeline 처리량이 Performance Estimates 에 나열되어 있는지 확인하라.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.40	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
508	508	375	375	dataflow

(a) ZedBoard

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	9.40	1.00

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
528	528	393	393	dataflow

(b) Zybo

- * Dataflow pipeline 처리량은 각 입력 읽기 집합간의 Clock Cycle 수를 나타낸다(interval 파라미터)
이 값이 Design Latency 보다 작으면 전류 입력 데이터가 출력되기 전에 설계가 새로운 입력을 처리하기 시작할 수 있음을 나타낸다.
- * Dataflow 는 최상위 수준의 함수 및 Loop 에 대해서만 지원되며 Design Hierarchy 구조에는 영향을 주지 않는다.
Design 의 최상위에 노출된 Loop 및 함수만 Dataflow 최적화의 이점을 얻는다.

7-1-9. Area Estimates 쪽 아래로 Scroll 하면 최상위 수준에서 필요한 BRAM_18K 의 수가 3 으로 유지된다.

Utilization Estimates					Utilization Estimates				
Summary					Summary				
Name	BRAM_18K	DSP48E	FF	LUT	Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-	DSP	-	-	-	-
Expression	-	-	0	152	Expression	-	-	0	152
FIFO	-	-	-	-	FIFO	-	-	-	-
Instance	2	8	1036	1681	Instance	2	8	1178	1693
Memory	1	-	256	16	Memory	1	-	256	16
Multiplexer	-	-	-	72	Multiplexer	-	-	-	72
Register	-	-	8	-	Register	-	-	8	-
Total	3	8	1300	1921	Total	3	8	1442	1933
Available	280	220	106400	53200	Available	120	80	35200	17600
Utilization (%)	1	3	1	3	Utilization (%)	2	10	4	10

(a) ZedBoard

(b) Zybo

7-1-10. Console View 를 보고 dct_coeff_table 이 차원 2 에서 자동으로 분할 된 것을 확인하라.

이전 실행에서 지시어를 적용한 것처럼 buf_2d_in 및 col_inbuf 배열이 분할되었다.

Dataflow 는 최상위 함수 read_data, dct_2d 및 write_data 사이의 채널을 만든 최상위 레벨에 적용된다.

```
INFO: [XFORM 203-712] Applying dataflow to function 'dct' (dct.c:78), detected/extracted 3 process function(s):
      'read_data'
      'dct_2d'
      'write_data'.
INFO: [XFORM 203-11] Balancing expressions in function 'dct_1d' (dct.c:4)...8 expression(s) balanced.
INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:01 ; elapsed = 00:00:06 . Memory (MB): peak =
INFO: [XFORM 203-541] Flattening a loop nest 'WR_Loop_Row' (dct.c:71:67) in function 'write_data'.
INFO: [XFORM 203-541] Flattening a loop nest 'RD_Loop_Row' (dct.c:59:67) in function 'read_data'.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' (dct.c:38:1) in function 'dct_2d'.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' (dct.c:49:1) in function 'dct_2d'.
INFO: [HLS 200-111] Finished Architecture Synthesis Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB)
INFO: [HLS 200-10] Starting hardware synthesis ...
INFO: [HLS 200-10] Synthesizing 'dct' ...
```

7-2. Analysis Perspective 로 전환하고 dct Performance Profile View 를 보고 Performance Analysis 를 수행한다.

7-2-1. Analysis Perspective 로 전환하고 Module Hierarchy 항목을 확장 한 다음 dct_2d 항목을 선택하라.

7-2-2. Performance Profile Tab 을 선택하라.

대부분의 Latency 및 Interval(처리량)은 dct_2d 기능으로 인해 발생한다.

최상위 함수 dct 의 interval 은 이들이 병렬로 작동한다는 것을 나타내는

read_data, dct_2d 및 write_Data 함수의 interval 의 합보다 작으며 dct_2d 가 제한 요소다.

Performance Profile Tab 에서 Row_DCT_Loop 및 Col_DCT_Loop 가 pipeline 되지 않았으므로 dct_2d 는 완전히 병렬로 작동하지 않는 것을 볼 수 있다.

The screenshot displays two windows from the ZedBoard IDE. The top window, titled 'Module Hierarchy', shows a tree view where the 'dct' module is expanded, revealing sub-modules: 'dct_2d', 'write_data', and 'read_data'. Below this, the 'Performance Profile' tab is active, showing a table of performance metrics for the selected 'dct_2d' module and its sub-loops.

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	3	8	1300	1921	508	375	dataflow
dct_2d	2	8	975	1284	374	374	none
write_data	0	0	32	206	66	66	none
read_data	0	0	29	191	66	66	none

	Pipelined	Latency	Initiation Interval
dct_2d	-	374	374
Row_DCT_Loop	no	120	-
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	yes	64	1
Col_DCT_Loop	no	120	-
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	yes	64	1

(a) ZedBoard

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ dct	3	8	1442	1933	528	393	dataflow	
▶ ● dct_2d	2	8	1094	1290	392	392	none	
● write_data	0	0	41	209	67	67	none	
● read_data	0	0	43	194	67	67	none	

Performance Profile				Resource Profile			
		Pipelined	Latency	Initiation Interval			
▲ ● dct_2d		-	392	392			
● Row_DCT_Loop		no	128	-			
● Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop		yes	65	1			
● Col_DCT_Loop		no	128	-			
● Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop		yes	65	1			

(b) Zybo

Dataflow 최적화의 한계 중 하나는 최상위 Loop 와 함수에서만 작동한다는 것이다.
dct_2d 에서 Block 을 병렬로 처리하는 한 가지 방법은 전체 함수를 Pipeline 화 하는 것이다.
그러나 이것은 모든 Loop 를 풀어 때로는 큰 면적 증가로 이어질 수 있다.
다른 방법으로는 dct_2d Hierarchy 를 제거하여 dct_2d 함수를 inline 하여
Dataflow 최적화를 적용할 수 있는 Hierarchy 의 최상위 수준까지 이러한 Loop 를 발생시키는 것이다.

7-2-3. Synthesis Perspective 로 전환하라.

Apply INLINE Directive

8-1. 이전 솔루션(Solution5) 설정을 복사하여 새 솔루션을 만든다.
dct_2d 에 INLINE 지시어를 적용하라.
솔루션을 생성하고 출력을 분석하라.

8-1-1. Project > New Solution 을 선택하라.

8-1-2. Solution Configuration Dialog Box 가 나타난다.
Finish 버튼을 클릭하라(Solution5 가 선택됨)

8-1-3. 지시어 창에서 dct_2d 함수를 선택하고 우클릭 버튼으로 클릭 한 다음 Insert Directive ... 를 선택한다.

8-1-4. 다양한 지시어를 나열하는 Pop-Up 메뉴가 표시된다.
INLINE 지시어를 선택하라.
INLINE 지시어는 적용되는 함수가 inline 되도록하여 해당 Hierarchy 를 해체한다.

8-1-5. Synthesis 버튼을 클릭하라.

8-1-6. Synthesis 가 완료되면 Synthesis Report 가 열린다.

8-1-7. ZedBoard(Zybo 의 경우 513 에서 449 Clock Cycle)에서 Latency 가
508 에서 495 Clock Cycle 로 감소하고 Dataflow 파이프라인 처리량이
375 에서 114 Clock Cycle(Zybo 의 경우 514 에서 114 Clock Cycle) 로 크게 감소했음을 관찰하라.

8-1-8. Synthesis Log 를 검사하여 자동으로 적용된 변환을 확인한다.

* dct_1d 함수 호출은 Loop 가 호출되는 Loop 에 자동으로 inline 되기 때문에 Loop 중첩이 자동으로 병합 될 수 있다.

* DSP48E 사용량이 두 배로 증가 했다(8 에서 16).

이전에는 dct_1d 의 단일 Instance 가 행 및 열 처리를 수행하는데 사용되었기 때문이다.

행과 열 Loop 가 동시에 실행되고 있기 때문에 더 이상 적용할 수 없으며 dct_1d 의 복사본이 두 개 필요하다.

Vivado HLS 는 영역을 늘려도 Clock 수를 최소화하려고 한다.

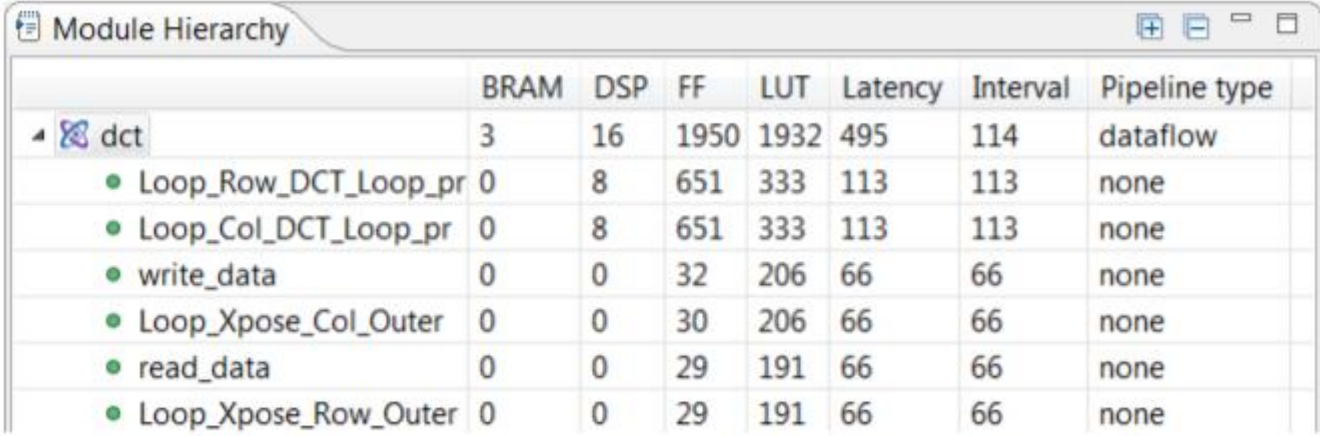
* 더 많은 Dataflow 절차 간의 ping-pong 버퍼링으로 인해 BRAM 사용이 다시 한 번 증가했다(4 에서 6)

```
INFO: [XFORM 203-712] Applying dataflow to function 'dct' (dct.c:78), detected/extracted 6 process function(s):
    'read_data'
    'Loop_Row_DCT_Loop_proc'
    'Loop_Xpose_Row_Outer_Loop_proc'
    'Loop_Col_DCT_Loop_proc'
    'Loop_Xpose_Col_Outer_Loop_proc'
    'write_data'.
INFO: [XFORM 203-602] Inlining function 'dct_1d' into 'Loop_Row_DCT_Loop_proc' (dct.c:33->dct.c:87) automatically.
INFO: [XFORM 203-602] Inlining function 'dct_1d' into 'Loop_Col_DCT_Loop_proc' (dct.c:44->dct.c:87) automatically.
INFO: [XFORM 203-11] Balancing expressions in function 'Loop_Row_DCT_Loop_proc' (dct.c:13:61)...8 expression(s) balanced.
INFO: [XFORM 203-11] Balancing expressions in function 'Loop_Col_DCT_Loop_proc' (dct.c:13:61)...8 expression(s) balanced.
```


8-1-9. Analysis Perspective 로 전환하고, Module Hierarchy 엔트리를 확장하고, dct 엔트리를 선택하라.

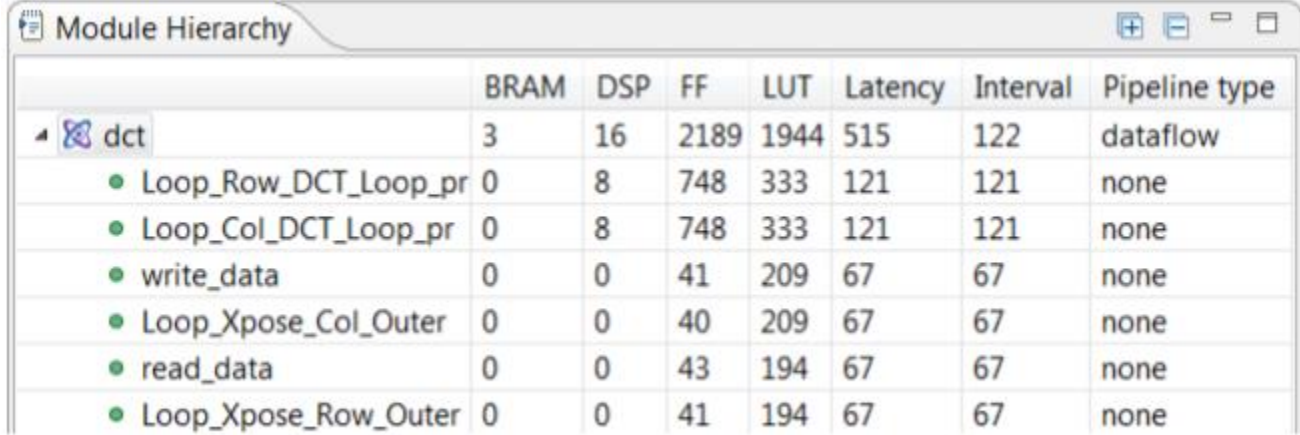
dct_2d 함수가 inline 되기 때문에 dct_2d 항목이 dct_Loop_Row_DCT_Loop_proc, dct_Loop_Xpose_Row_Outer_Loop_proc, dct_Loop_Col_DCT_Loop_proc 및 dct_Loop_Xpose_Col_Outer_Loop_proc 으로 바뀐다.

또한 모든 기능이 병렬로 작동하여 106 Clock Cycle 의 최상위 함수 Interval(처리량)을 산출한다.



	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
└─ dct	3	16	1950	1932	495	114	dataflow
└─ Loop_Row_DCT_Loop_pr	0	8	651	333	113	113	none
└─ Loop_Col_DCT_Loop_pr	0	8	651	333	113	113	none
└─ write_data	0	0	32	206	66	66	none
└─ Loop_Xpose_Col_Outer	0	0	30	206	66	66	none
└─ read_data	0	0	29	191	66	66	none
└─ Loop_Xpose_Row_Outer	0	0	29	191	66	66	none

(a) ZedBoard



	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
└─ dct	3	16	2189	1944	515	122	dataflow
└─ Loop_Row_DCT_Loop_pr	0	8	748	333	121	121	none
└─ Loop_Col_DCT_Loop_pr	0	8	748	333	121	121	none
└─ write_data	0	0	41	209	67	67	none
└─ Loop_Xpose_Col_Outer	0	0	40	209	67	67	none
└─ read_data	0	0	43	194	67	67	none
└─ Loop_Xpose_Row_Outer	0	0	41	194	67	67	none

(b) Zybo

8-1-10. Synthesis Perspective 로 전환하라.

Apply RESHAPE Directive

9-1. 이전 솔루션(Solution6) 설정을 복사하여 새 솔루션을 만든다.
RESHAPE 지시어를 적용하라.
솔루션을 생성하고 출력을 이해하라.

9-1-1. Project > New Solution 을 선택한다.

9-1-2. Solution Configuration Dialog Box 가 나타난다.
Finish 버튼을 클릭하라(Solution6 이 선택된 상태)

9-1-3. Directive Pane 에서 dct 함수의 buf_2d_in 배열에 적용된 PARTITION 지시어를 선택하고 우클릭 한 다음 Modify Directive 를 선택한다.
ARRAY_RESHAPE 지시어를 선택하고 치수로 2 를 입력 한 다음 OK 를 클릭한다.

9-1-4. 마찬가지로 Directive Pane 에서 dct_2d 함수의 col_inbuf 배열에 적용된 PARTITION 지시어를 차원이 2 인 ARRAY_RESHAPE 로 변경한다.

9-1-5. dct_coeff_table 배열에 차원 2 의 ARRAY_RESHAPE 지시어를 할당한다.

```
└─ ● dct_1d
    *11 dct_coeff_table
    % HLS ARRAY_RESHAPE variable=dct_coeff_table complete dim=2
    ▷ *11 DCT_Outer_Loop
└─ ● dct_2d
    % HLS INLINE
    *11 row_outbuf
    *11 col_outbuf
    *11 col_inbuf
    % HLS ARRAY_RESHAPE variable=col_inbuf complete dim=2
    *11 Row_DCT_Loop
    ▷ *11 Xpose_Row_Outer_Loop
    *11 Col_DCT_Loop
    ▷ *11 Xpose_Col_Outer_Loop
▷ ● read_data
▷ ● write_data
└─ ● dct
    % HLS DATAFLOW
    ● input
    ● output
    *11 buf_2d_in
    % HLS ARRAY_RESHAPE variable=buf_2d_in complete dim=2
    *11 buf_2d_out
```

9-1-6. Synthesis 버튼을 클릭하라.

9-1-7. Synthesis 가 완료되면 Synthesis Report 가 자동으로 열린다.

9-1-8. ZedBoard 의 경우 495 에서 559, Zybo 의 경우 515 에서 657 로 증가한 Latency 와 ZedBoard 의 경우 114 에서 131, Zybo 의 경우 122 에서 134 로 증가한 Dataflow 파이프라인 처리량이 모두 저하되었음을 관찰하라. BRB Resource 사용률은 ZedBoard 의 경우 3 에서 7 로 증가했지만 Zybo 의 경우 3 에서 11 로 증가했다.

* Synthesis Log 를 검토하면 몇 가지 단서가 제공된다.
read_data 에 대한 Scheduling 단계에서 II = 1 을 달성 할 수 없다는 경고가 있다.
실제로 read_data 는 읽기 및 쓰기 작업의 충돌에 대해 불평한다.

* 여기서 문제는 재구성 된 배열의 단일 요소를 업데이트 할 때
전체 WORD 를 읽고 단일 요소를 업데이트하고 전체 WORD 를 다시 작성해야 한다는 것이다.
재구성 된 배열에는 읽기 - 수정 - 쓰기 Cycle 이 필요하다
(Vivado HLS 는 쓰기에 바이트 마스크를 구현하지 않는다).

* 이 작업은 이러한 배열의 최대 쓰기 대역폭에 부정적인 영향을 미친다.

9-1-9. 따라서 지시어를 신중하게 적용해야 한다는 것을 알 수 있다.

9-1-10. File > Exit 를 선택하여 Vivado HLS 를 닫는다.

Conclusion

이 Lab에서는 Performance를 향상시키고 Resource 사용의 균형을 맞추기 위한 다양한 기법을 배웠다.

PIPELINE 지시어를 바깥 쪽 Loop에 적용하면 내부 Loop가 자동으로 풀린다.

Loop가 펼쳐지면 작업이 동시에 완료될 때 Resource 활용도가 증가한다.

메모리를 Partitioning하면 Performance는 향상되지만 BRAM 사용률은 증가한다.

함수에 INLINE 지시어를 적용하면 하위 수준 Hierarchy가 자동으로 해제된다.

DATAFLOW 지시어를 적용하면 ping-pong 유형의 기본 메모리 버퍼가 최상위 함수와 Loop 사이에 자동으로 삽입된다.

RESHAPE 지시어는 BRAM에 여러 번 접근할 수 있지만 전체 요소에 대해 읽기 - 수정 - 쓰기 작업이 발생하므로 단일 요소를 수정해야 하는 경우 주의해야 한다.

Analysis Perspective와 Console Log는 현재 일어나고 있는 일에 대한 통찰력을 제공할 수 있다.

Answers

1. Answer the following questions for dct:

Estimated clock period: 6.51 ns

Worst case latency: 3959 clock cycles

Number of DSP48E used: 1

Number of BRAMs used: 5

Number of FFs used: 278

Number of LUTs used: 982

References

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