Class Project Challenge(CPC) Presentation

Digital System Design

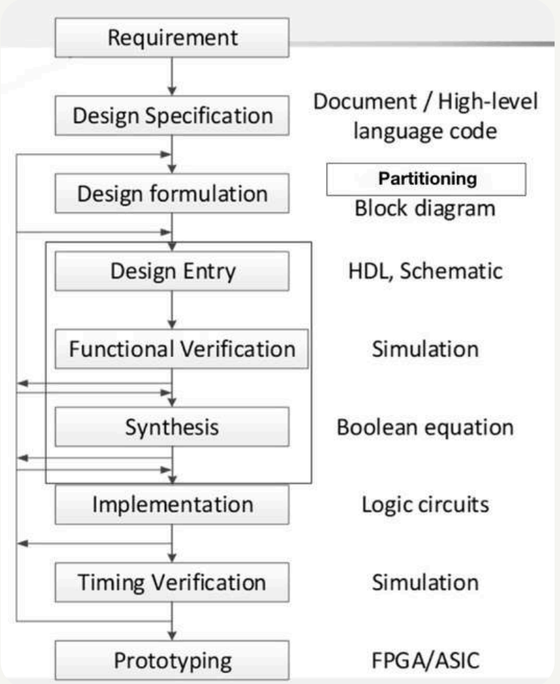
FIR filter를 이용한 다중처리 FIR filter 시스템 구성

Team. Edge Runner

20192490 김현우 / 20192484 김주성
20191888 박소현 / 20192513 윤종민

Contents

What we learned in class



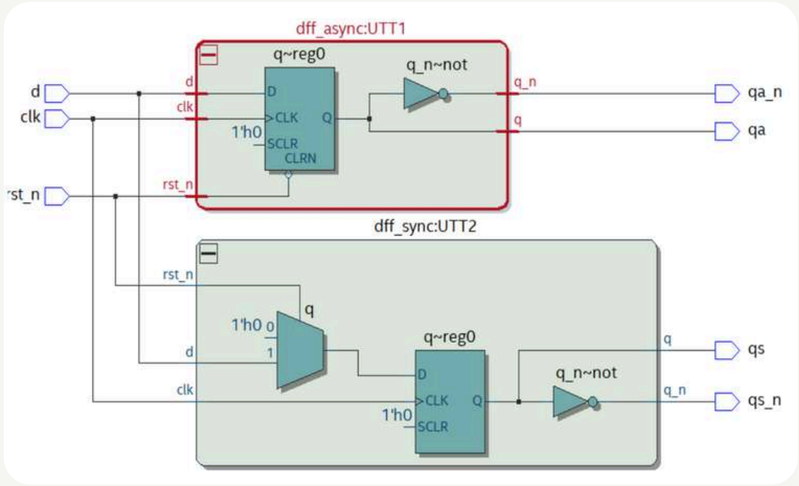
디지털시스템설계 Process

요구사항 분석, 설계, 구현, 검증, 테스트 단계를 통해 디지털 회로(시스템)를 개발하는 과정을 학습

```
// The first example uses continuous
wire out;
assign out = sel ? a : b;

// the second example uses a procedu
// to accomplish the same thing.

reg out;
always @(a or b or sel)
  begin
    case(sel)
      1'b0: out = b;
      1'b1: out = a;
    endcase
  end

// Finally – you can use if/else in
// procedural structure.
reg out;
always @(a or b or sel)
  if (sel)
    out = a;
  else
    out = b;
```

Verilog HDL

Verilog HDL의 구조와 문법을 학습



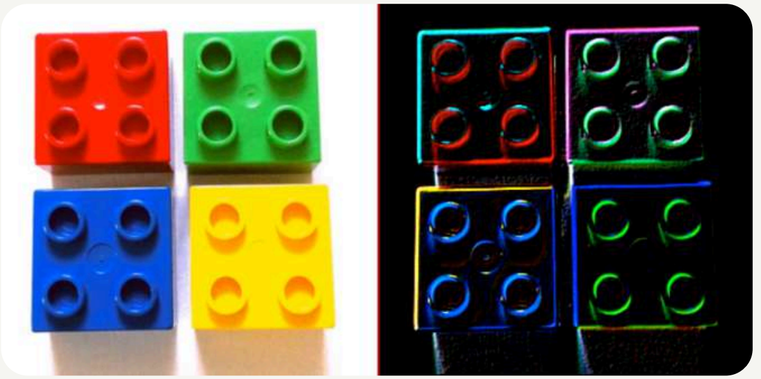
RTL Circuit 설계 및 테스트

Verilog HDL을 이용해 작성한 코드를 Systhesis 하고 테스트벤치를 제작하여 Digital Logic 검증

Project outline and objective

프로젝트 목표 : FIR Filter을 이용한 FHD 이미지 필터링 성능 개선

FIR Filter를 설계한 후 Filter의 작동을 확인하기 위해 320x320 해상도의 이미지에서 Edge detection을 진행한다. (Edge detection을 위한 filter kernel을 이용) 이후 FHD 해상도의 이미지에서도 동일하게 필터링을 진행하고 이와 비교하여 더 빠르게 필터링(최대 64배)이 진행되게 할 수 있는 방법에 대해서 연구한다.



FIR Filter를 이용한 320x320 해상도의
이미지 필터링 결과 (Edge detection)

MAC 연산기를 여러개 사용

하나의 FIR Filter 내에 여러 개의 MAC 연산 처리기를 두어 필터링을 병렬적으로 처리한다.

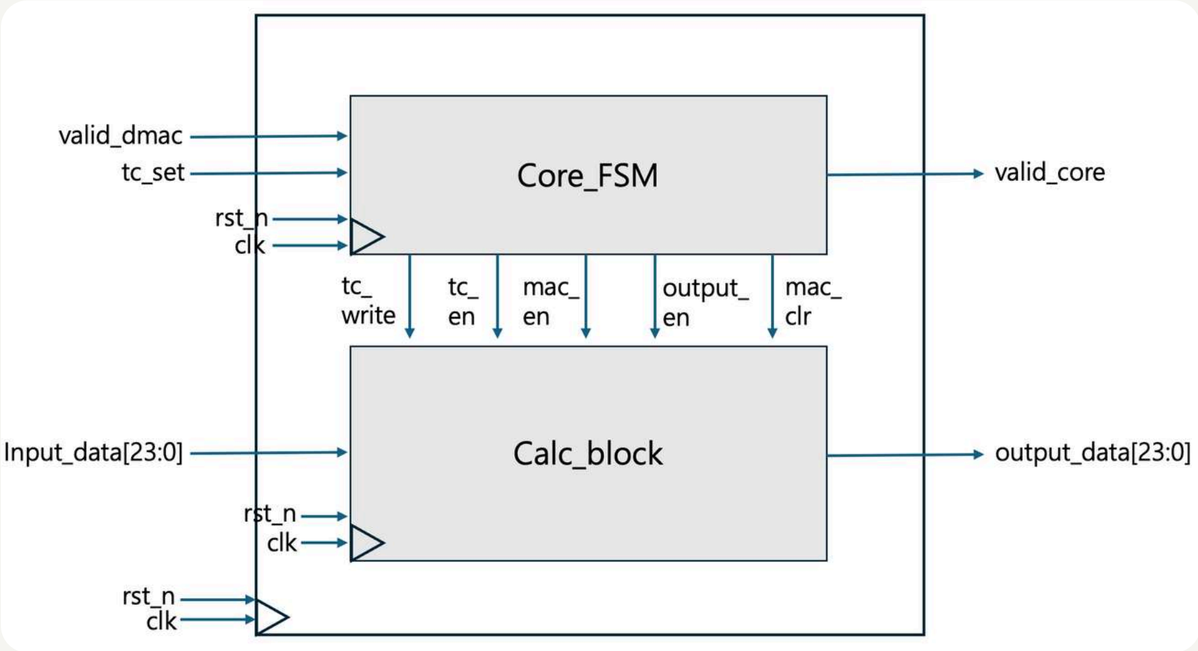
Filter를 여러개 사용

FIR Filter 여러 개를 병렬적으로 배치하여 각각이 이미지의 부분별 필터링을 동시에 진행하게 한다.

Filter Kernel의 값 조정

Filter kernel의 값을 바꾸어 보면서 필터링 성능과 속도의 변화를 체크한다.

Design an FIR Filter



FIR Filter의 전체 구조

FIR Filter의 작동 방식

FIR Filter는 내부의 **2개의 모듈**을 통해 필터링을 진행한다. **Core_FSM**이 각 state 별로 적절한 출력을 (5개의 출력 중) 만들어 내면 이 출력값을 입력으로 하여 **Calc_block**을 제어한다. Calc_block 내에서는 image pixel과 filter kernel 값의 분리, filter kernel 값 저장, MAC 연산, 후처리의 작업을 진행한다.

Core_FSM input

- valid_dmac, tc_set

Core_FSM output

- tc_write, tc_en, mac_en, output_en, mac_clr, valid_core

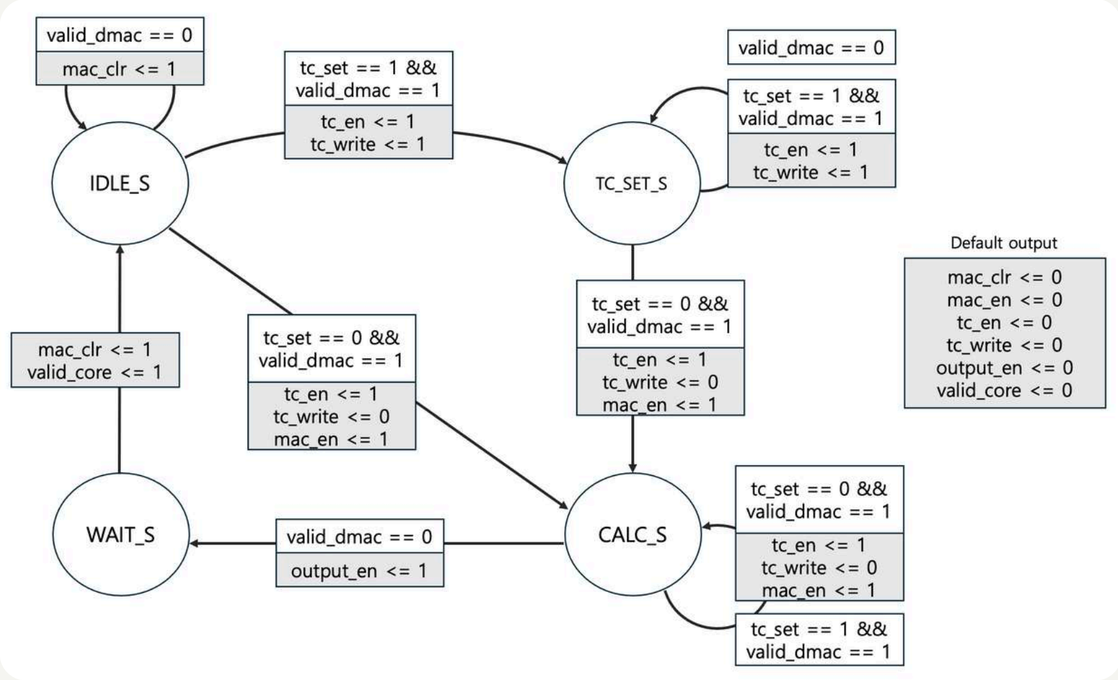
Calc_block input

- input_data[23:0], tc_write, tc_en, mac_en, output_en, mac_clr

Calc_block output

- output_data[23:0]

Design an FIR Filter



Core_FSM의 State Diagram

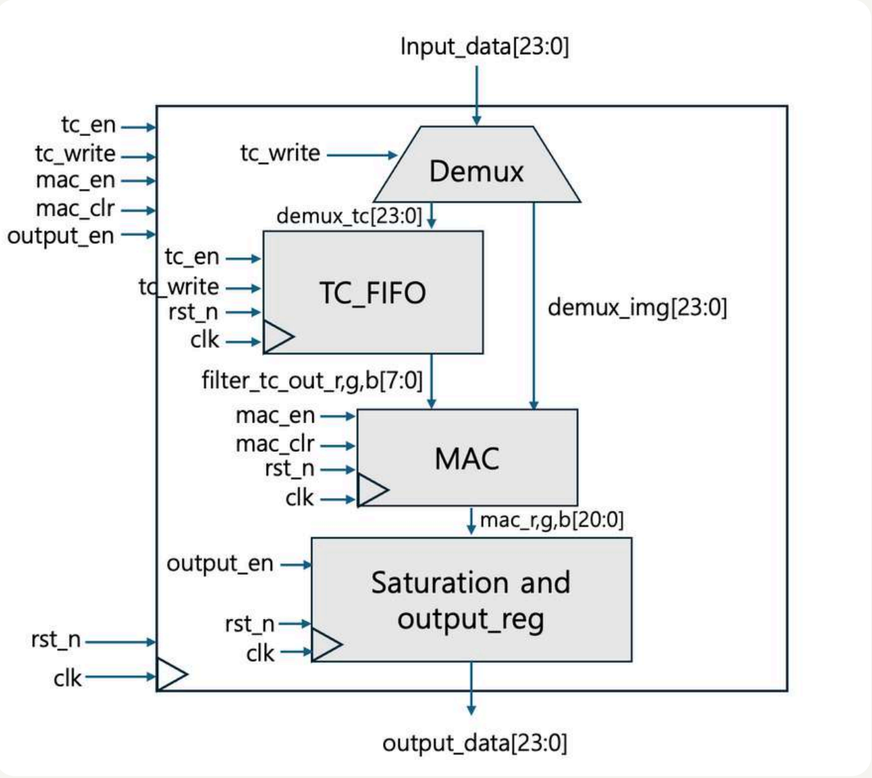
x means 'don't care'

	Input			Output					
ps (present state)	tc_set	valid_dmac		ns (next state)	mac_en	mac_clr	tc_en	tc_write	output_en	valid_core
IDLE_S	1	1		TC_SET_S	0	0	1	1	0	0
	0	1		CALC_S	1	0	1	0	0	0
	x	0		IDLE_S	0	1	0	0	0	0
TC_SET_S	1	1		TC_SET_S	0	0	1	1	0	0
	0	1		CALC_S	1	0	1	0	0	0
	x	0		TC_SET_S	0	0	0	0	0	0
CALC_S	0	1		CALC_S	1	0	1	0	0	0
	1	1		CALC_S	0	0	0	0	0	0
	x	0		WAIT_S	0	0	0	0	1	0
WAIT_S	x	x		IDLE_S	0	1	0	0	0	1

Core_FSM의 input / ouput Table

FSM의 state, input의 따른 state 변화와 output을 다음과 같이 그림과 표로 나타내었다.

Design an FIR Filter



Calc_block의 내부 구조

Demux

tc_write 값에 따라 tap coefficient값 또는 image pixel 값을 구분하여 출력으로 내보낸다. tc_write값이 0일 때는 들어오는 입력값을 하나의 이미지 픽셀 데이터로 tc_write값이 1일 때는 들어오는 입력값을 tap coefficient 로 처리한다.

TC_FIFO

TC_FIFO 내에 9개의 데이터를 저장할 수 있는 3개의 (R, G, B 성분) Rotation Queue가 있고 이 Queue들 내에 3x3 filter kernel의 tap coefficeient 값들을 저장한다. tc_en과 tc_write의 값에 따라 tap coefficient의 값을 Queue에 저장하거나 저장된 값을 연산을 위해 출력값으로 내보낸다.

MAC

mac_en의 값이 1, mac_clr의 값이 0 일 때, 이미지 픽셀의 R, G, B, 성분에 각각 tap coefficient를 곱하고 누적하여 더한다. 9번의 MAC 연산을 수행 후 각 R, G, B 성분의 연산 결과값을 출력한다.

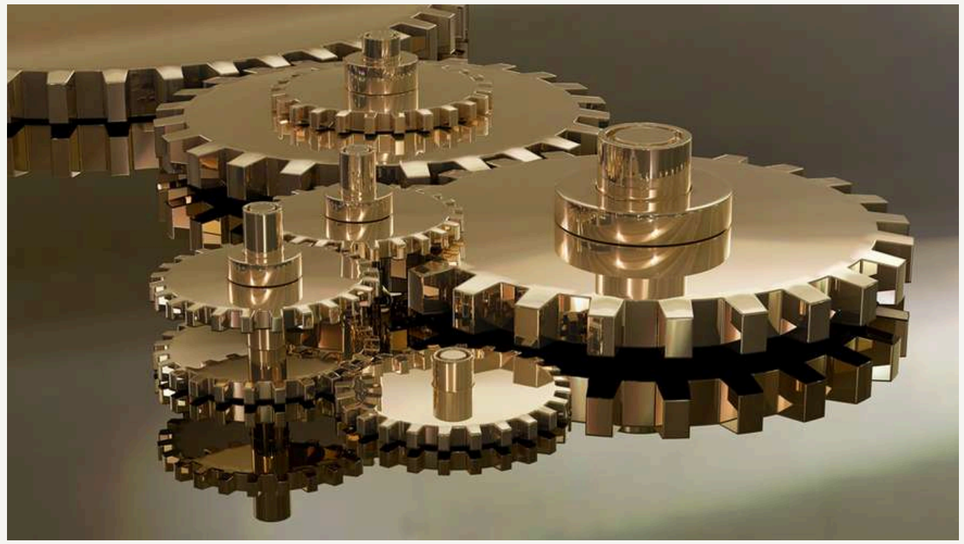
Saturation and output_reg

output_en의 값이 1 일 때, mac_r,g,b 입력 값들을 각각 확인 후 음수가 되는 부분은 0으로 saturation 하고 255가 넘어가는 부분은 255로 saturation 한 후 다시 하나의 RGB 픽셀로 조합하여 최종적으로 필터링된 24bit의 output_data를 출력한다.

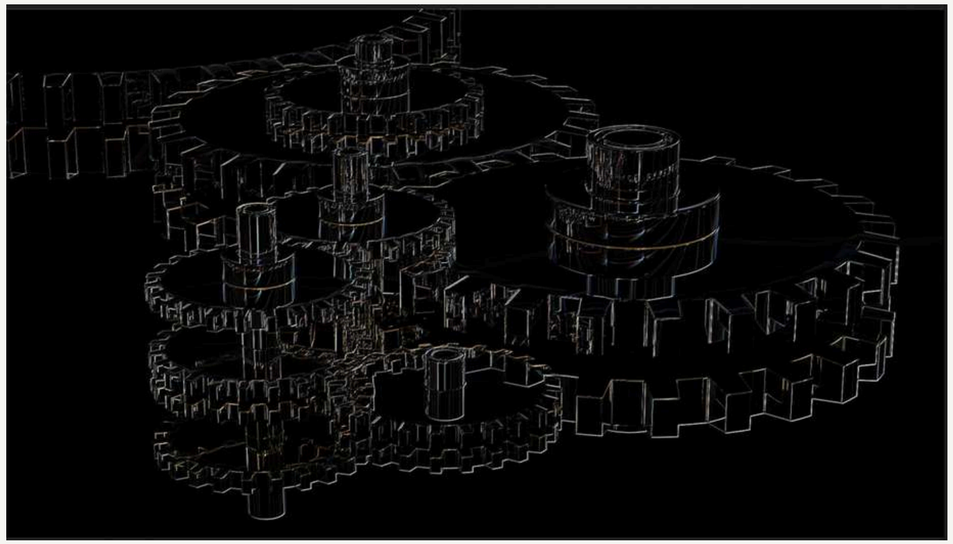
Design a testbench

Testbench

제작한 filter를 검증하기 위해 Testbench를 이용하여 filtering한다. Clock에 맞춰서 이미지의 pixel값을 입력하면 output으로 filtering된 값이 나오고, 이것들을 다시 이미지로 재구성한다.



filtering할 FHD(1920 X 1080)이미지



filtering된 FHD(1920 X 1080)이미지

Design a testbench

Basic operation of testbench

이미지 Load

불러온 FHD 이미지를 픽셀의 개수인 1920x1080 size의 1차원 배열로 저장

1차원 이미지 데이터를 2차원 배열로 변환 (이미지 모양으로)

모든 픽셀 대상으로 MAC연산을 진행할 9개의 픽셀값을 순서대로 가져와 재구성한다.

이미지 가장자리 픽셀의 padding 문제 해결

총 1980 * 1080 * 9 (전체 입력 데이터) + 9 (filter kernel 값) 개의 데이터를 1차원 배열로 저장

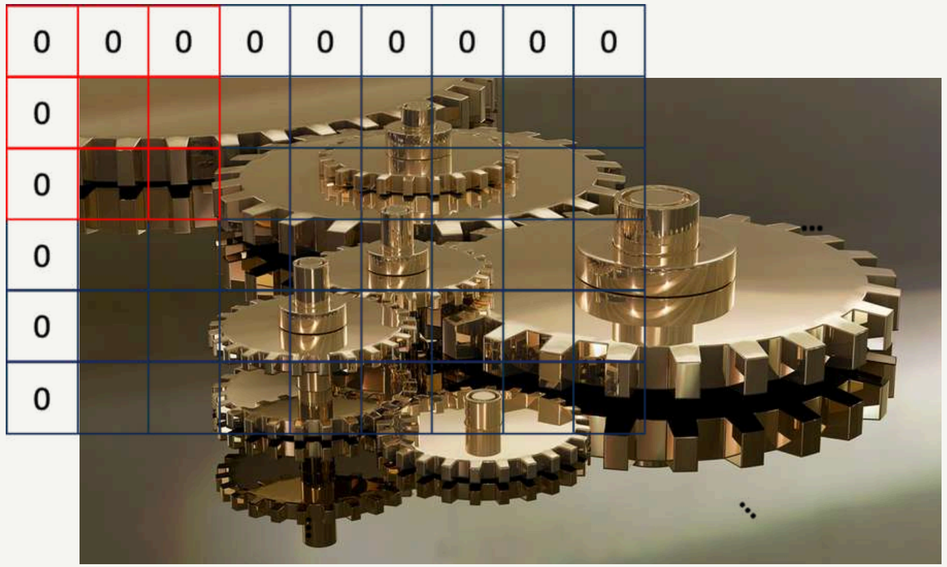
tap coefficient부터 순서대로 FIR_filter_2d 모듈의 input_data로 입력 (필터마다 입력 방식 차이)

최종 필터링된 출력 데이터를 1920x1080 1차원 배열로 변환해 .rgb 파일로 저장

Design a testbench

Zero padding

실제로 이미지에 zero padding을 하기보다는 pixel 하나를 필터링 하기 위해 그 pixel과 주변 8개의 pixel을 가져오는 과정에서 커널이 벗어나는 가장자리 부분이 있으면 0을 가져오도록 알고리즘을 만들어 가장자리 padding 문제를 해결하였다.



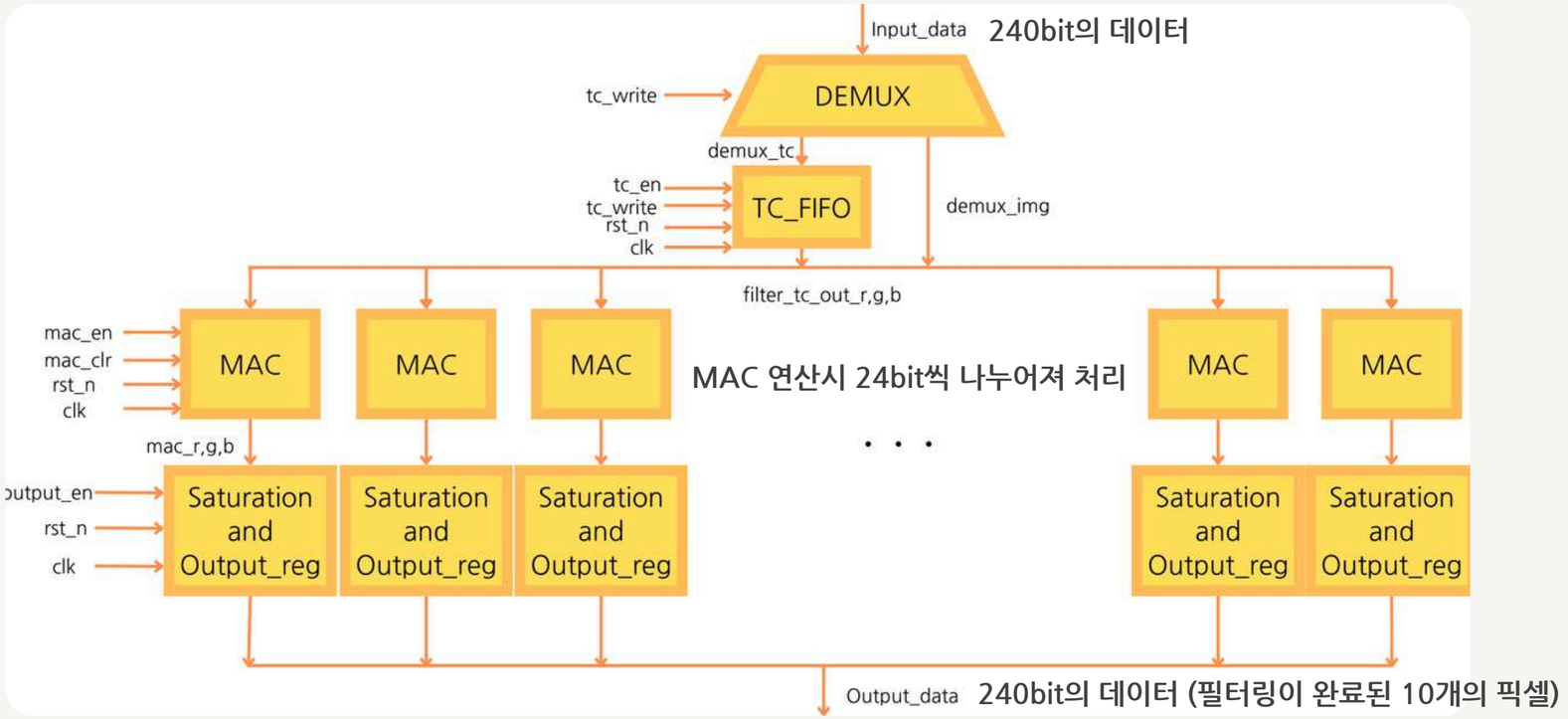
Zero padding 예시

0	0	0
0	7e6b43	7d6a42
0	7e6b43	7d6a42

첫 번째 pixel과 주변 8개 픽셀값을 가져오는 예시

Operation

한 필터 내에서 MAC & Saturation block을 다수(10개) 사용하여한 clock cycle에 10번의 MAC 연산을 수행한다.



10개의 MAC 연산 & saturation 모듈을 사용한 FIR_filter Block diagram

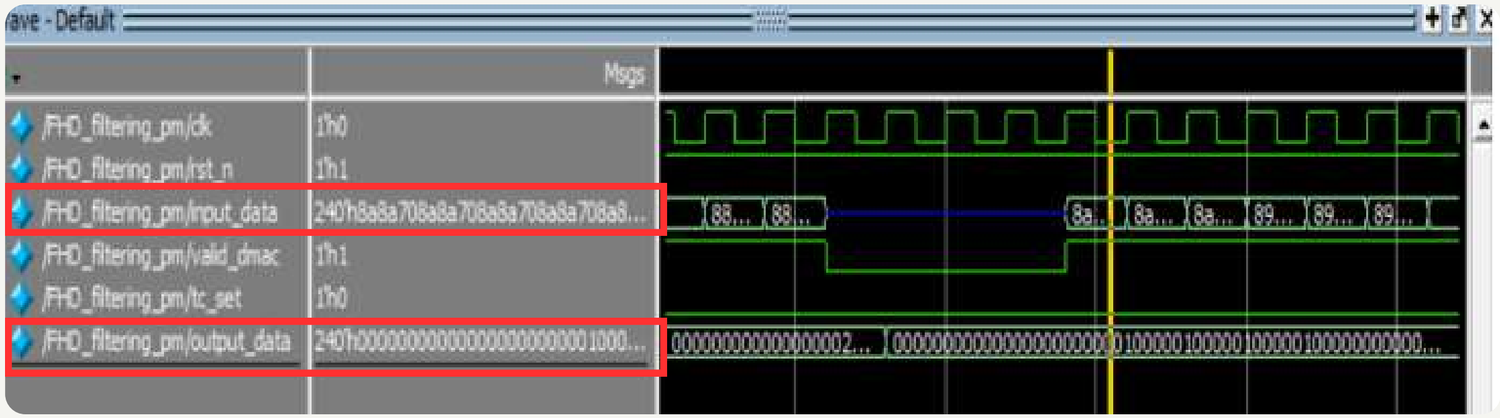
Testbench

동시에 10번의 MAC 연산을 위해 한번에 240bit(10*24bit)의 같은 커널 위치 값을 넣어 주는 방식으로 수정하였다. 예를 들어, 맨 처음은 0번에 해당하는 값들을 10개 묶어서 filter로 넣어주고, 그 다음에는 1번에 해당하는 값들을 묶어서 넣어주는 방식으로 Testbench를 수정하였다.

커널 번호

0	1	2
3	4	5
6	7	8

filtering을 진행한 결과 다음과 같이 240bit의 데이터가 입력되고 출력되는 것을 확인할 수 있다.



Result

성능 평가 결과		
	Initial	MAC*10
시뮬레이션 [ps]	5.39E+11	5.39E+10
실제 측정 시간 [s]	583	226
시뮬레이션 성능 개선 (n배)		9.98
실제 성능 결과 (n배)		2.58

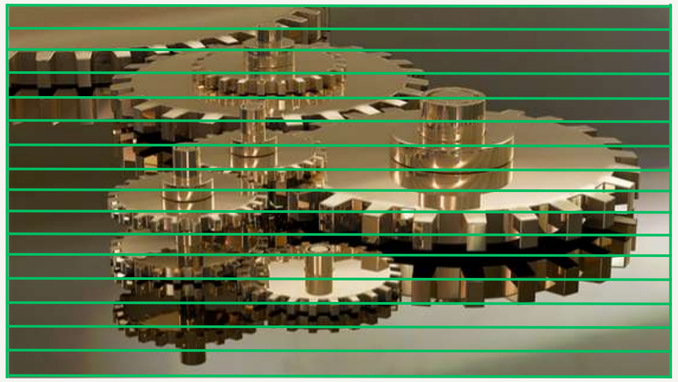
<filtering 결과>

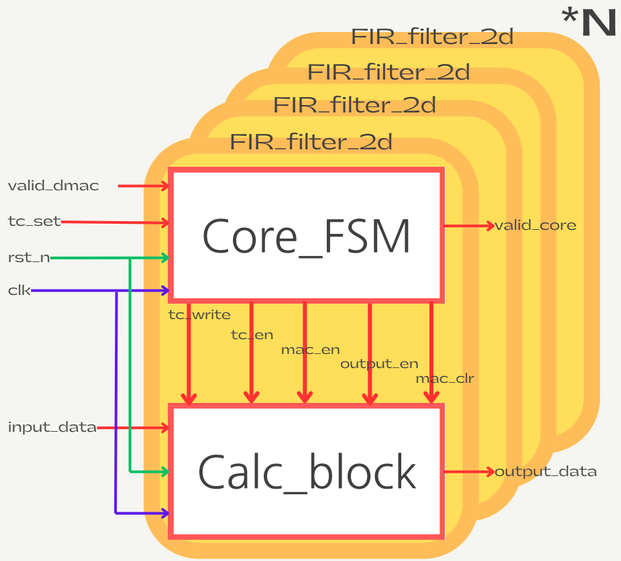
시뮬레이션 상에서 필터링을 완료하는 데 소모된 시간은 기존의 필터에 비해 약 10배 빨랐고 이는 이론적이고 이상적인 성능 개선의 결과와 일치했다.

하지만 실제 필터링 시간을 측정한 결과 필터링은 기존의 모델보다 약 2.6배 빠르게 완료됐다. 이는 컴퓨터 성능에 따라 바뀔 수 있다.

Operation

또 다른 방법으로는 이미지를 여러 영역으로 나누고, 각 영역의 연산을 동시에 진행시킨다면 더 짧은 시간 내에 필터링을 완료할 수 있을 것이라 생각하였다.

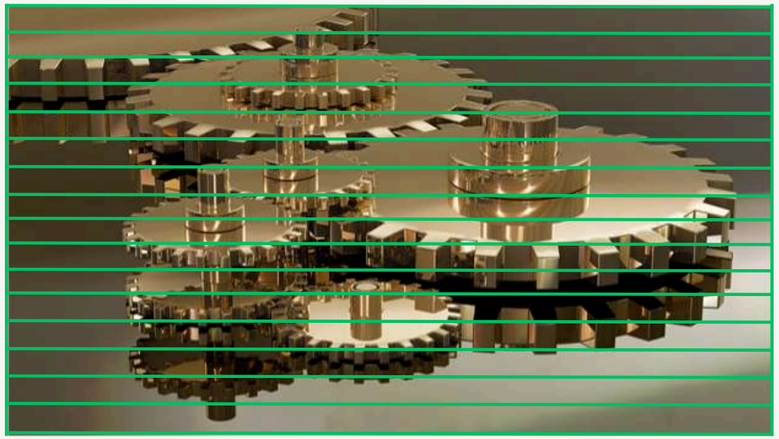


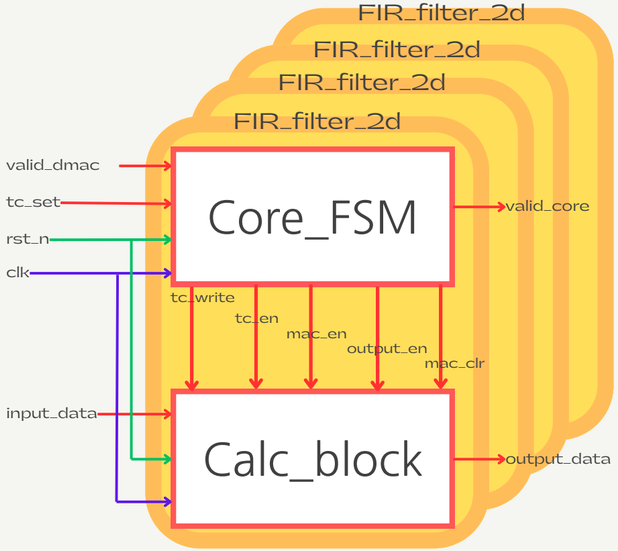


Process Speed ↑ * N ??

- 이미지를 분할 후 처리하기 때문에 Top Module에 여러 개의 Filter를 Instantiate 한다.
- 각 Filter는 주어진 데이터(분할된 이미지)만 사용한 연산을 진행한다.
- 병렬화한 Filter의 개수를 늘리고 시뮬레이션 하는 작업을 반복 시행, 검증해본다.

Testbench





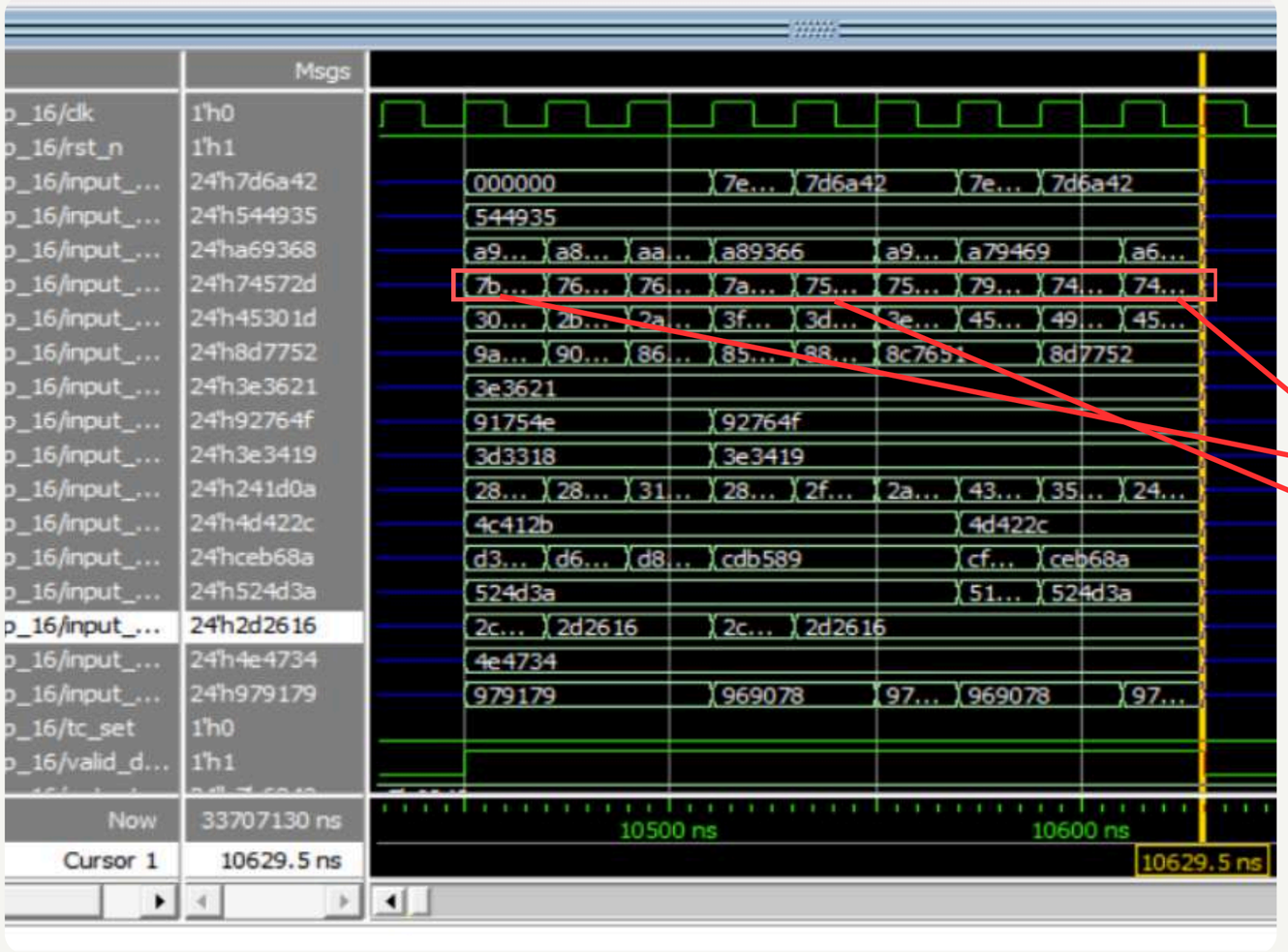
Testbench에서 입력은 이미지를 분할하여 필터에 입력하는 것과 동일한 상황을 만들기 위해 18,662,400개의 데이터를 사용하려 하는 필터의 개수만큼 분할했고, 각 필터의 input으로 하나씩 입력하여 주는 방식으로 프로그램을 작성하였다. 결과적으로는 input_data의 개수가 줄었을 뿐, 기본 모듈로 이미지를 필터링 하는 경우와 동일한 프로세스를 진행한다

Simulation

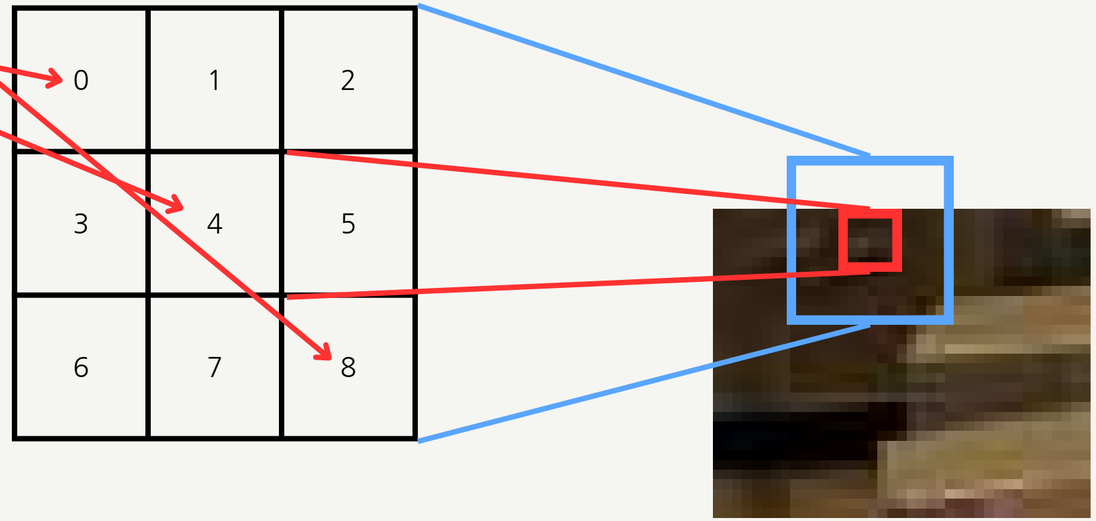
시뮬레이션을 진행하였을 때,16개의 Filter가 정상적으로 작동하는 것을 확인할 수 있다.

tc_set의 값이 1이 되었을 때에 clock의 Rising Edge에서 16개 모듈 모두에 동일한 Tap Coefficient가 입력된다.

Simulation



clock의 Rising Edge에 맞춰서 16개의 Filter에 각각 다른 값들이 입력되는 것을 확인할 수 있다. 하나의 픽셀 출력에 대해 9개의 픽셀에 대한 연산을 진행하므로 9번의 입력을 받는다.



Simulation

16개의 Filter들이 각각의 필터링 결과값을 출력하는 모습을 확인할 수 있다.

Result

성능 평가 결과				
	FIR_Filter * 1 (Initial)	FIR_Filter * 4	FIR_Filter * 16	FIR_Filter * 64
시뮬레이션 [ps]	5.39E+11	1.35E+11	3.37E+10	8.44E+09
실제 측정 시간 [s]	583	330	252	174
시뮬레이션 성능 개선 (n배)		4.00	15.99	63.90
실제 성능 결과 (n배)		**1.63**	**2.31**	**3.35**

〈 Filtering 결과 〉

시뮬레이션 상에서 필터링을 완료하는 데 소모된 시간은 기존의 필터에 비해 모듈의 개수만큼 빨랐고 이는 이론적이고 이상적인 성능 개선의 결과와 일치했다.

하지만 실제 필터링 시간을 측정한 결과 필터링은 기존의 모델보다(16개의 경우) 약 2.31배 빠르게 완료됐다. 이는 컴퓨터 성능에 따라 바뀔 수 있다.

Conclusion

Third idea - diverse kernel values

여러 Kernel(mask)값에 따른 필터링 결과

1. Mask 변화에 따른 출력 품질 차이

mask B는 edge를 두껍게 표현하였고 이와 비교하였을 때 A, C, D는 edge를 얇게 표현함을 확인할 수 있다. 또한 mask D에서 검출되지 않은 edge 영역이 mask B에서는 선명함을 확인할 수 있다.

2. Mask 변화에 따른 출력 속도 차이

mask A를 이용했을 때 필터링 소요 시간은 53s이고 mask B, C, D는 각각 49s, 47s, 47s를 기록하였다. 이에 따라 mask 값에 따른 출력 속도 차이가 존재함을 확인하였다.

∴ **filter kernel 값에 따라 성능 및 속도의 변화를 확인할 수 있다.**

→ **검출하고자 하는 특징에 따라 적절한 mask 선택 필요**

Conclusion

Performance evaluation

	Initial	MAC X 10	FIR_Filter X 4	FIR_Filter X 16	FIR_Filter X 64
시뮬레이션 [ps]	5.39E+11	5.39E+10	1.35E+11	3.37E+10	8.44E+09
성능 개선 효과	-	9.98	4	15.99	63.9

- MAC연산 구조를 병렬화해 데이터를 연산하도록 구조화한 결과
  MAC 연산기 개수에 비례하는 9.98배의 성능 개선 효과

- FIR_Filter 구조를 병렬화해 이미지를 분할 처리하도록 구조화한 결과
  이미지 분할 횟수에 비례하는 성능 개선 효과

∴ 성능 개선 효과는 데이터 처리 경로를 분할한 구조에 비례

	Initial	MAC X 10	FIR_Filter X 4	FIR_Filter X 16	FIR_Filter X 64
실제 출력 시간[s]	583	226	330	252	174
출력 개선 효과	-	2.58	1.63	2.31	3.35

- MAC연산 구조를 병렬화해 데이터를 연산하도록 구조화한 결과
  226초로 **약 2.58배의 출력 시간 단축 효과**

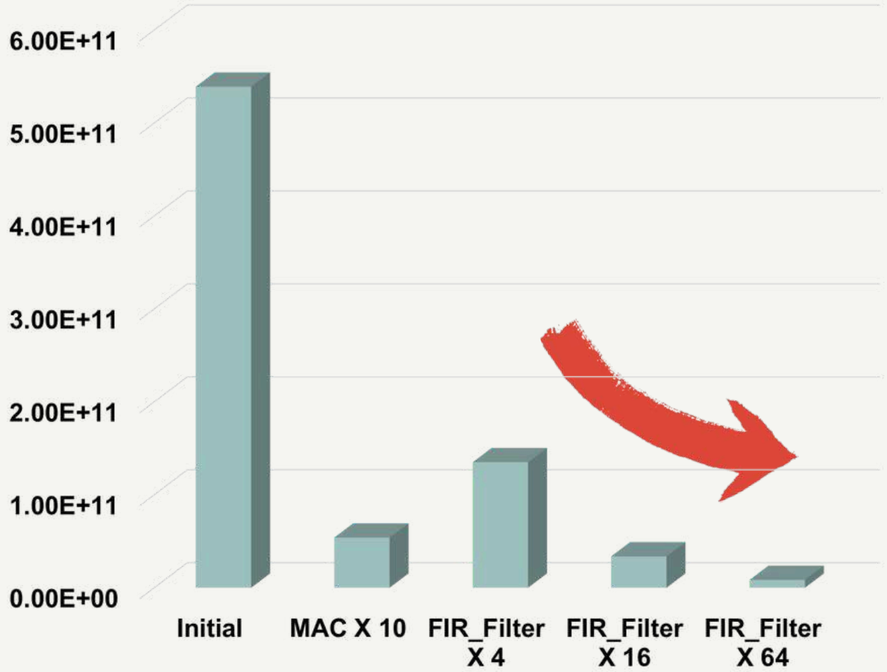
- FIR_Filter 구조를 병렬화해 이미지 분할 처리하도록 구조화한 결과
  각 330초. 252초, 174초로 **1.63배~3.35배의 출력 시간 단축 효과**

∴ 출력 시간 단축 효과는 병렬 처리 구조에 비례했으나, 그 효율은 점차 감소

Conclusion

Performance evaluation

시뮬레이션 동작 측정



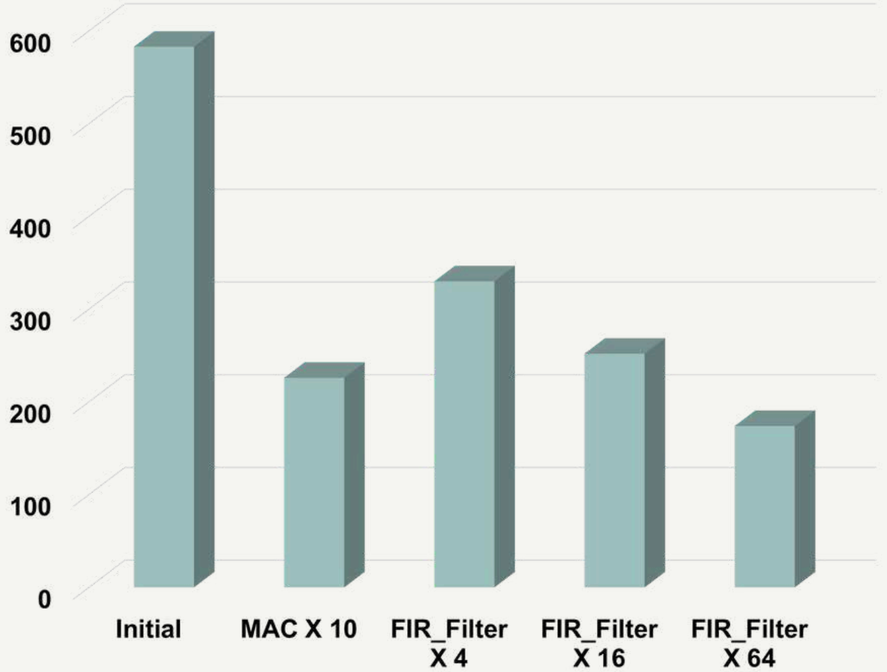
시뮬레이션 상에서의 필터링 속도는 설정한 Clock 주기를 따르기에

속도 판단의 척도로 세우기엔 어려움이 있으나,

시각화 하였을 때 **이론상 지수함수 꼴로 시간이 감소**하는걸 확인할 수 있다.

✓ **이론상 정상적으로 작동함을 확인 가능**

실제 출력 시간 측정



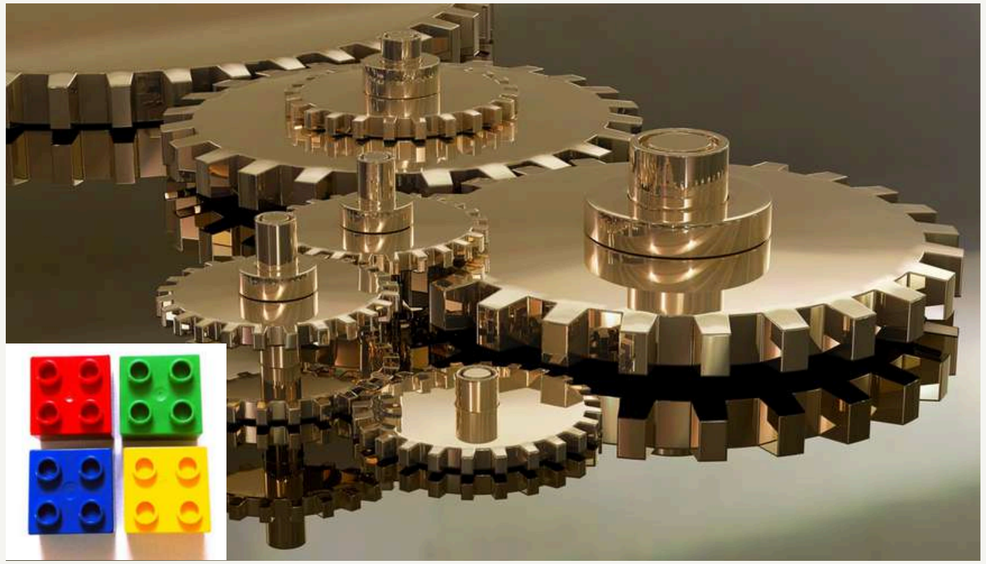
실제 필터링 시간을 스톱워치를 이용해 측정한 결과이다.

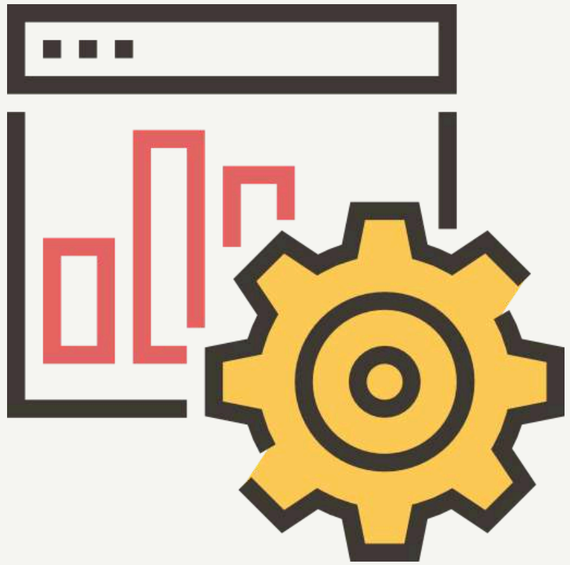
예상한대로, 이미지 **병렬 처리구조에 비례해 출력시간이 감소함**을

확인할 수 있다.

Conclusion

Project Review





Resolution Expansion

320 X 320 해상도 이미지에서

1920 x 1080 해상도 이미지 필터링으로 범위 확장

Optimization

필터링 품질과 작동 시간 최적화를 위한

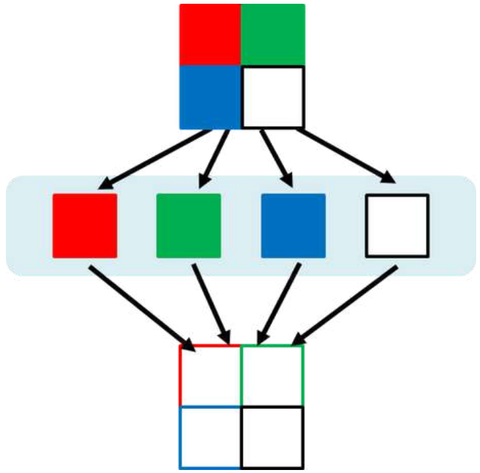
데이터 처리 구조 및 필터링 조건 연구

Conclusion

Project Review

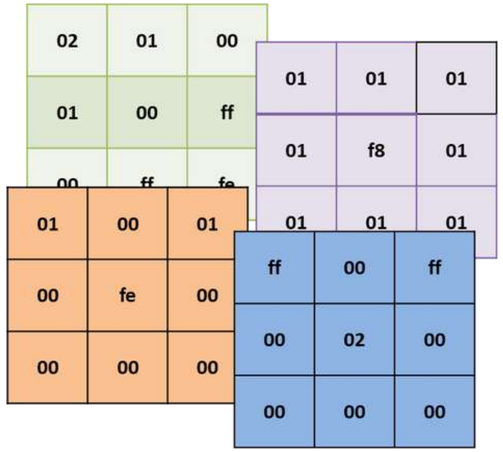
이미지 처리 효율성

FHD 해상도의 이미지를 신속하고 효율적으로 필터링할 수 있는 방법을 제시함으로써, 고해상도 이미지 처리에 필요한 시간과 자원을 절약할 수 있다.



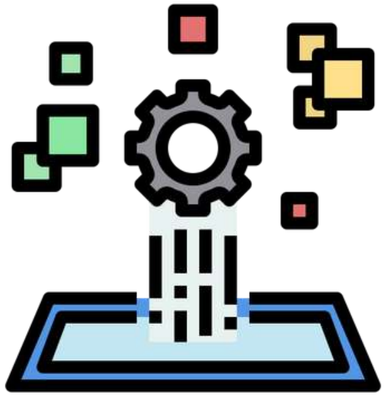
병렬 처리 기법의 응용

병렬 처리 기법을 적용함으로써 이미지 필터링 속도를 개선하는 것은 이미지 처리 및 실시간 고속 영상 처리 시스템에 응용될 수 있다.



다양한 필터링 기법 검증

여러 가지 필터 커널을 활용한 것은 검출하고자 한 특징에 최적화된 필터링 방법을 찾는 데 도움이 될 수 있다.



기술 확장성

본 프로젝트에서 구상한 방법론은 이미지 필터링 외에도 다양한 디지털 신호 처리 분야에 적용되어 타 프로젝트나 연구 개발의 기반이 될 수 있다.