디지털 FPGA설계

REAL
8-Point FFT
Calculator Design

20222947 전민서
20202840 한정호

2025-12-11

목차

Contents

역할 분담

전민서

- Top-level Design & Integration
- Input/Output System 설계
- FFT Core 알고리즘 구현 및 최적화
- Simulation & Timing Analysis

한정호

- FSM Control Logic 설계 (UI Controller)
- Keypad Debouncing & Decoding Logic 구현
- 7-Segment Multiplexing Display Driver 구현
- BCD Conversion Algorithm (Double Dabble) 설계

개발 환경

OS & Software

Win 11
Operating System

Vivado 2024.1
IDE / Synthesis Tool

Target Hardware

JFK-200A Board
* Note: Synthesized using Vivado for Xilinx equiv. testing

Language

Verilog HDL
IEEE 1364-2005 Standard

구현 시나리오

3단계 처리 프로세스

시스템은 사용자 입력부터 결과 표시까지 명확하게 분리된 3단계 처리 프로세스로 구성됩니다.

Phase 1: 입력 처리

계산기식 실수 입력

- 입력 범위: ±999.999
- 정수부 3자리 + 소수부 3자리

데이터 변환

- BCD 입력 관리
- ×1000 스케일링
- 8개 샘플 입력 완료 시 FFT 트리거

Phase 2: FFT 연산

8-Point Radix-2 DIF FFT

- 3-Stage 버터플라이 구조
- 입력: 8×21-bit 실수
- 출력: 8×32-bit 복소수

최적화 기법

- Twiddle factor 근사 (shift-add)
- i_start 펄스 → o_busy → o_done

Phase 3: 출력 표시

결과 선택 및 관리

- 실수부/허수부 토글
- 인덱스 이동 (X[0]~X[7])

BCD 변환

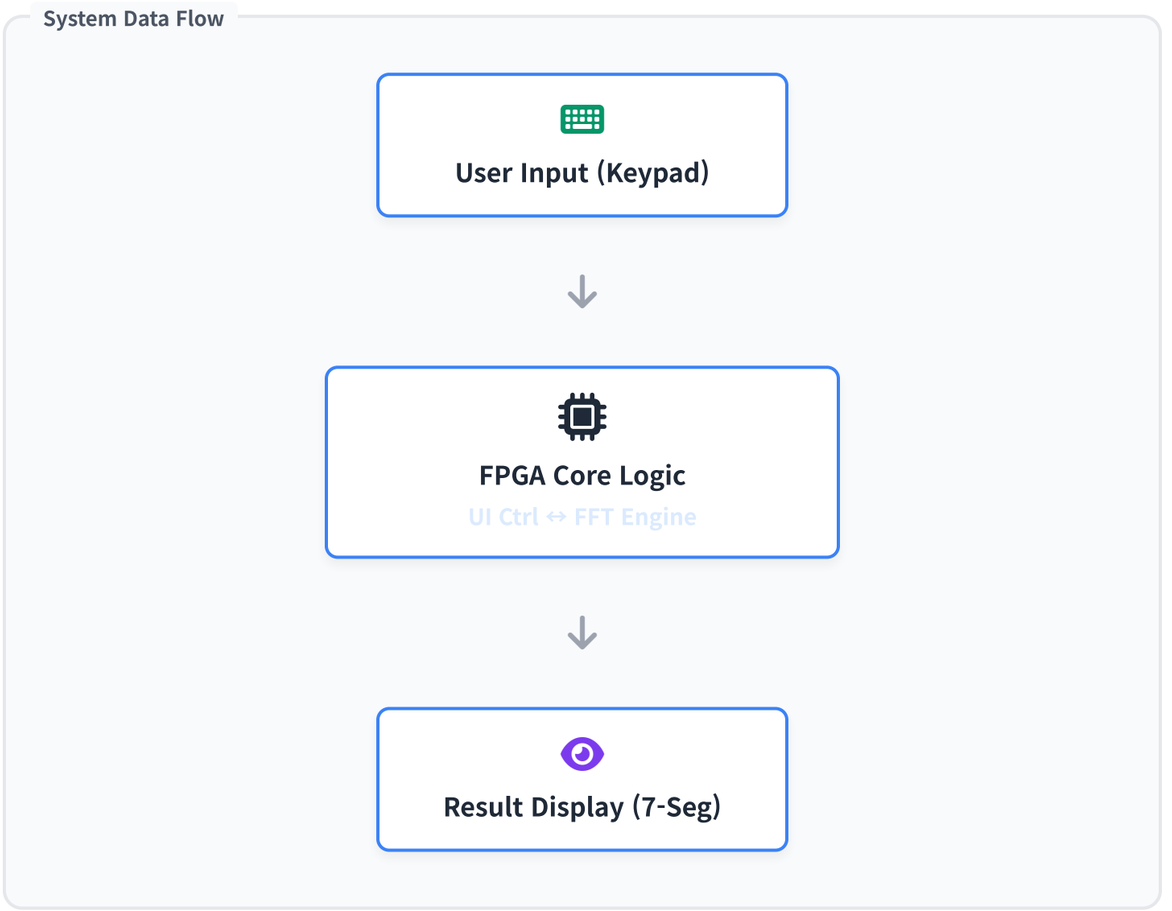
- Double-dabble 알고리즘
- 24 사이클 변환 시간
- 8자리 7세그먼트 표시

목표 및 특징

본 프로젝트는 **Radix-2 DIF 알고리즘**을 하드웨어로 구현하고, 사용자가 계산기처럼 숫자를 입력하여 주파수수 성분을 분석할 수 있는 **FFT 프로세서** 구현을 목표로 합니다.

Key Features

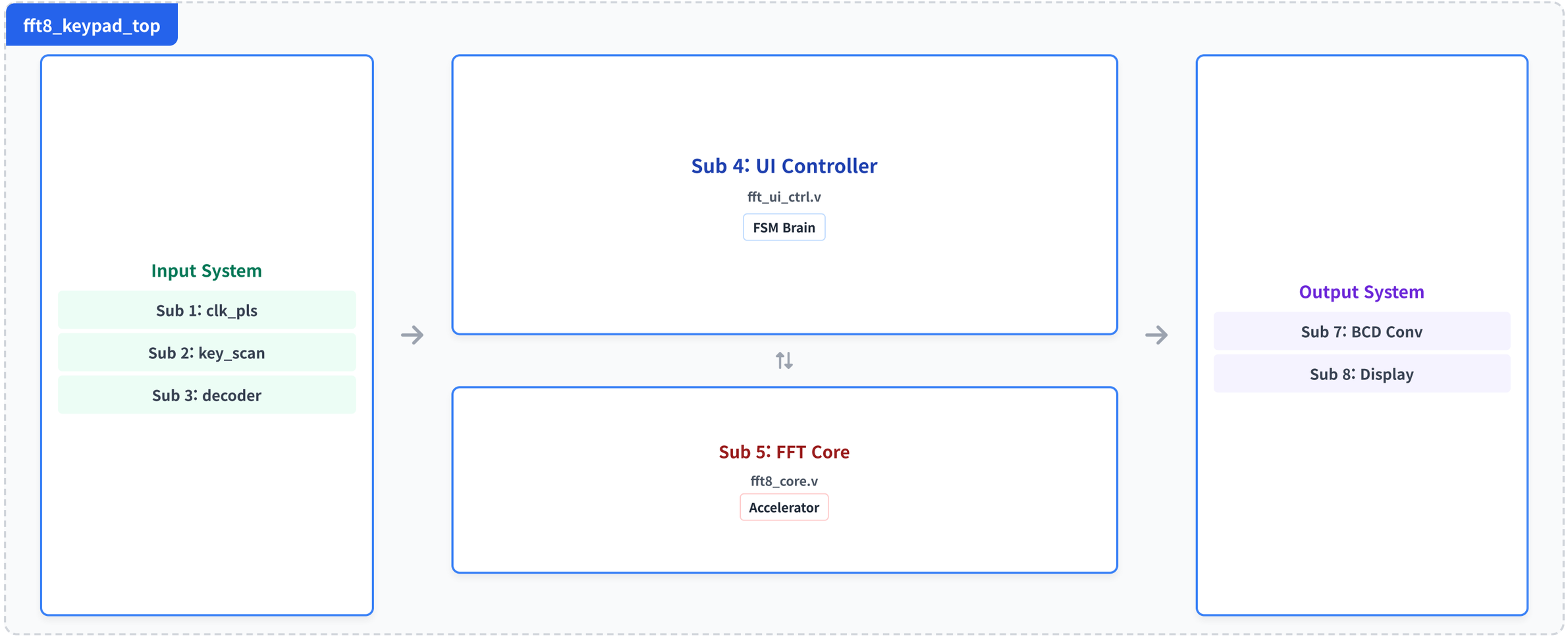
- **Hierarchical Design:** Top-down 방식의 모듈화 설계
- **Interactive UI:** 소수점/음수 입력 및 수정 기능
- **Optimization:** 고정소수점 연산 및 자원 절약형 Butterfly 구조
- **Visual Output:** 7-Segment Multiplexing 디스플레이, LED 시각화



Top-Level Design

Top Module Hierarchy

`fft8_keypad_top` 모듈을 최상위로 하여 기능별로 서브모듈을 계층적으로 인스턴스화하였습니다.

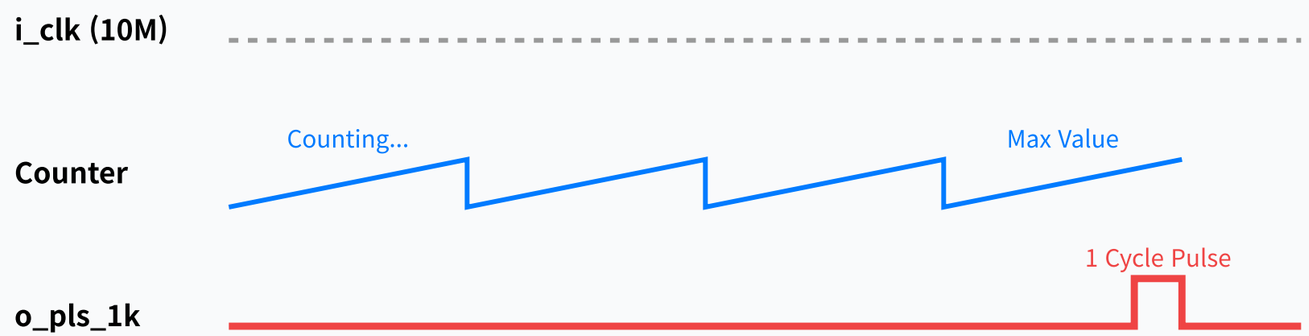


Module: clk_pls.v

시스템 클럭을 분주하여 제어용 펄스를 생성합니다.

- **Input:** 10MHz System Clock (`i_clk`)
- **Output:** 1kHz Pulse (`o_pls_1k`)
- **Logic:** Counter (0 ~ N-1)
- **Role:** Keypad Debouncing 및 7-Segment Multiplexing 타이밍 기준 신호 제공

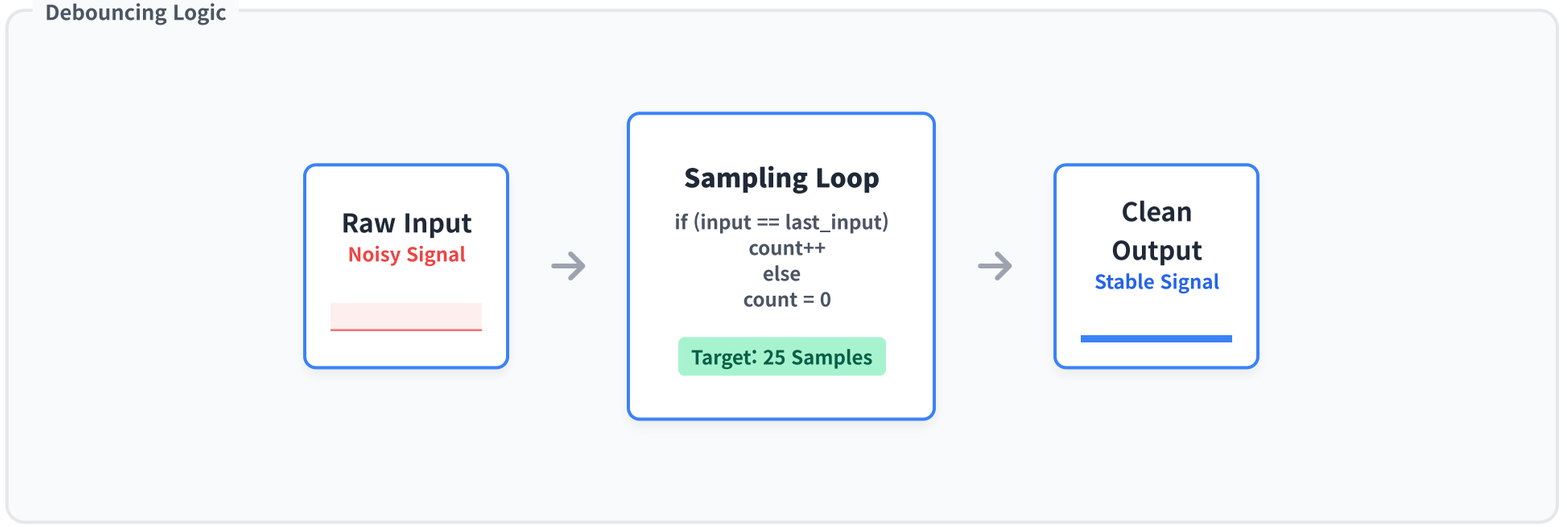
Timing Diagram



Module: key_scan.v

물리적 키 입력을 스캔하고 디바운싱(Debouncing) 처리합니다.

- **Input:** 5-bit Raw Key Data (`i_key_in`)
- **Control:** 1kHz Pulse (Enable Signal)
- **Output:** Valid Key Code & Valid Flag
- **Algorithm:** 25회 연속 동일 값 입력 시 유효한 키로 인정 (Chattering 제거)



Module: decoder.v

스캔된 키 코드를 기능별 신호로 매핑합니다.

- **Input:** 5-bit Code (1 ~ 20)
- **Output:** One-hot Encoded Flags
- **Mapping:** 물리적 위치에 따른 기능 부여

Physical Keypad Mapping & User Guide

Key Usage Guide

DATA ENTRY

0 ~ 9	Numeric Input
. / -	Decimal / Sign
Ent	Confirm / Next

SYSTEM CONTROL

Esc	Reset (To Input)
F1	Backspace

RESULT NAVIGATION

F2 / F3	Next / Prev
F4	Real / Imag

Module: fft_ui_ctrl.v

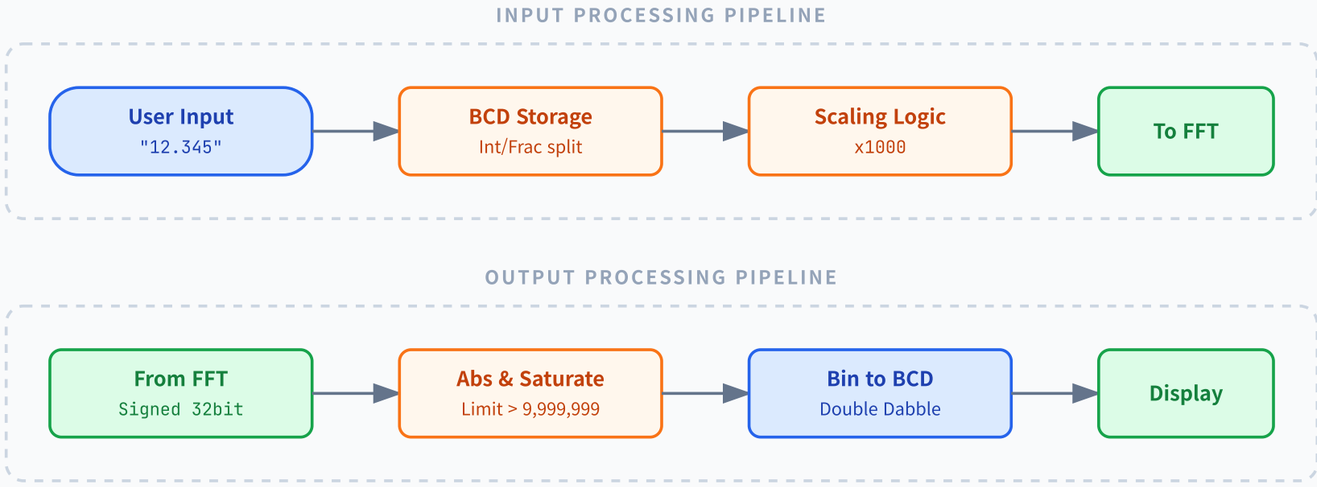
전체 시스템의 상태(FSM)를 관리하고 데이터 흐름을 제어하는 모듈입니다.

Internal Logic

- **Input Parsing:** 입력된 BCD Digit를 정수부/소수부로 구분하여 저장.
- **Fixed-Point Conv:** 내부 연산을 위해 입력을 정수형(x1000)으로 변환.
- **FSM:** Input -> Wait(Calc) -> Output 제어.

1. FSM Flow

2. Data Pipeline



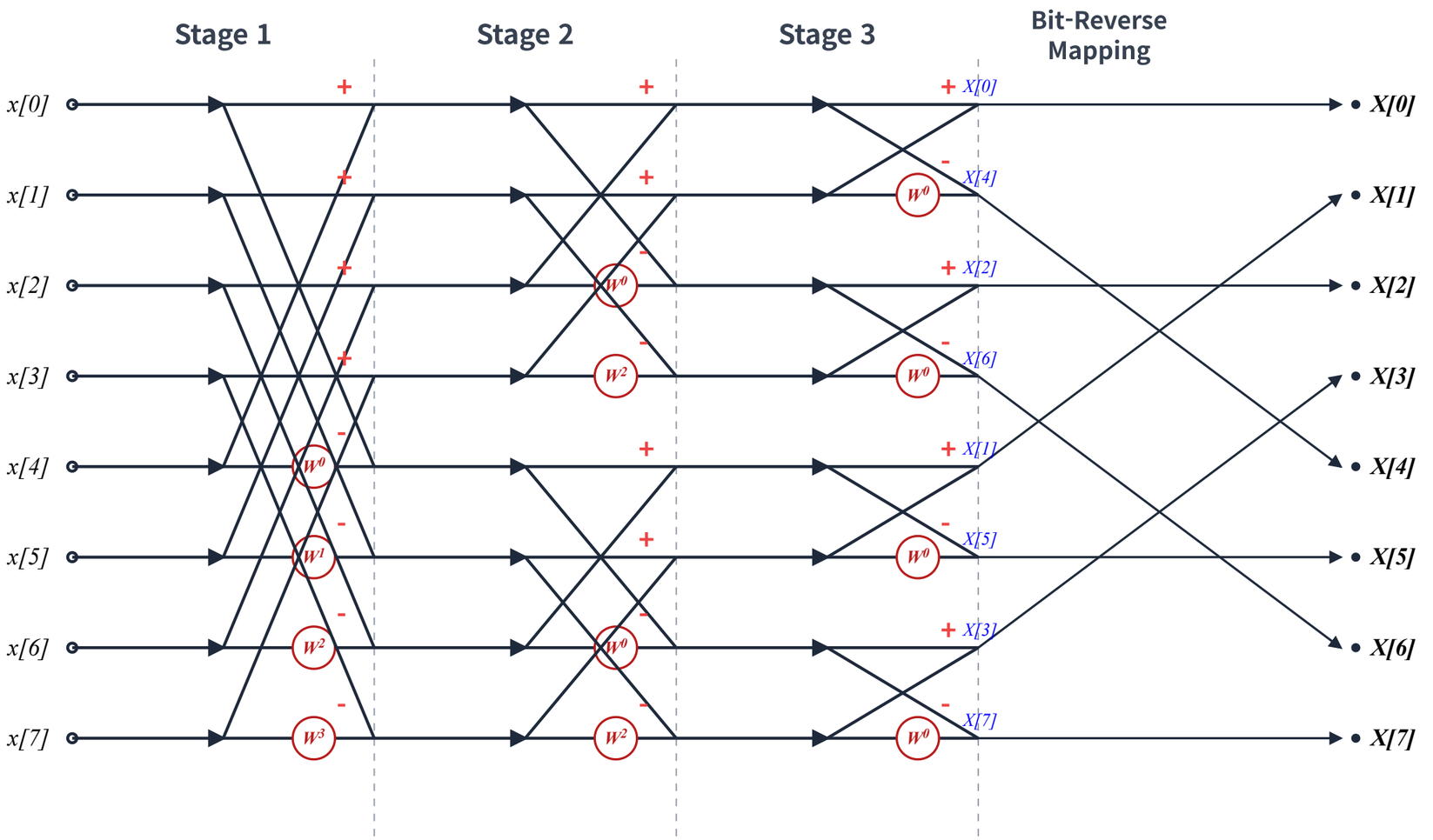
Sub-module 5: FFT Core (Top)

Module: fft8_core.v

Radix-2 DIF 알고리즘을 수행하는 모듈입니다.

Architecture

1. **Pipeline:** 3-Stage Butterfly 구조.
2. **Handshaking:** Start/Busy/Done 프로토콜 사용.
3. **Data Path:** 21-bit Input -> 32-bit Complex Output.



Arithmetic Logic Unit

1. Fundamental Theory

COMPLEX BUTTERFLY

Upper: $A' = A + B$

Lower: $B' = (A - B) \cdot W_N^k$

2. HW Optimization

곱셈기(Multiplier) 사용을 피하기 위해 상수 곱셈을 **Shift-Add 방식**으로 대체합니다.

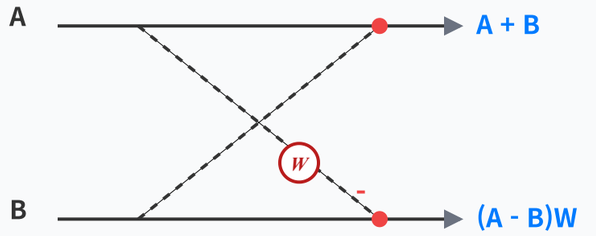
```
Target:  1/√2 ≈ 0.707106
Approx:  181/256 ≈ 0.707031
Error:   0.01%
```

3. Computational Complexity (8-Point 기준)

FFT는 DFT 대비 **연산량이 대폭 감소된다.**

구분	DFT $O(N^2)$	FFT $O(N \log_2 N)$	감소율
복소수 곱셈	64회 N^2	12회 $\frac{1}{2}N \log_2 N$	81% ↓
복소수 덧셈	56회 $N(N-1)$	24회 $N \log_2 N$	57% ↓

Radix-2 Butterfly Diagram



8-Point Twiddle Factors (W_8^k)

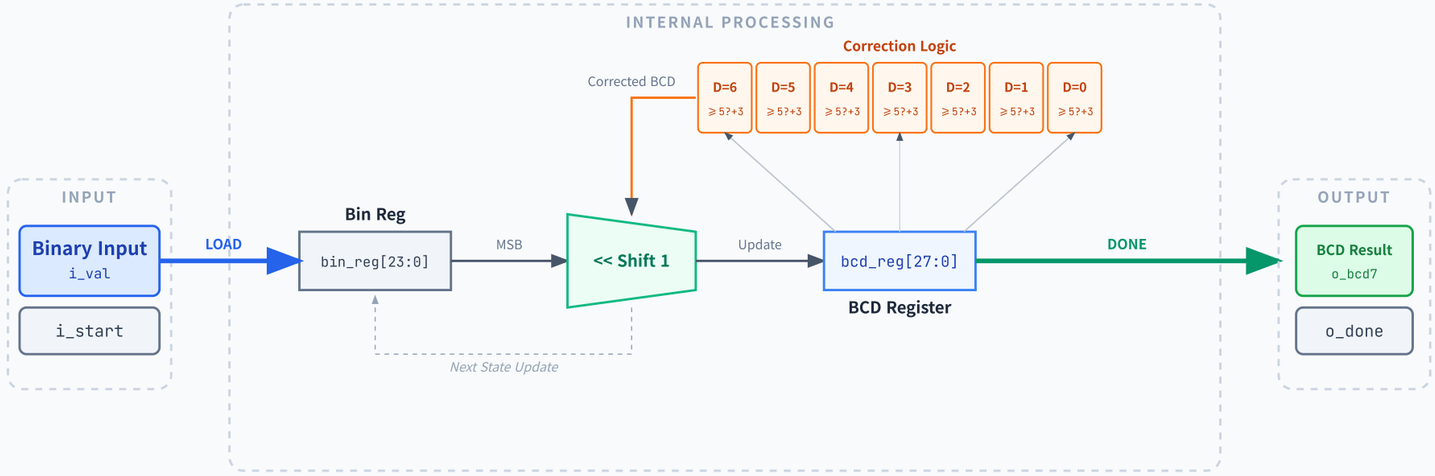
k	Formula	Value (Complex)	Implementation
0	W^0	1 + j0	No Op (Pass Through)
1	W^1	0.707 - j0.707	Shift-Add (x181 >> 8)
2	W^2	0 - j1	Swap Re/Im & Negate
3	W^3	-0.707 - j0.707	Shift-Add & Negate

Module: bin_to_bcd7.v

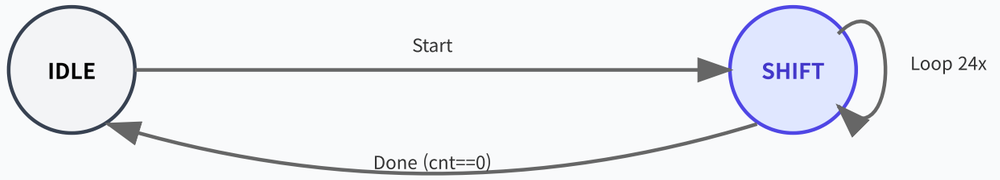
2진수 결과를 10진수로 변환하기 위해 **Double Dabble** 알고리즘을 사용합니다.

- **Input:** 24-bit Binary Integer
- **Output:** 28-bit BCD (7 Digits)
- **Logic:** Sequential Shift & Correction
- **Latency:** 24 Clock Cycles

Double Dabble Algorithm



Control FSM



Sub-module 8: Display Driver

Module: seg8digit.v

8자리 7-Segment를 시분할 제어하여 하나의 디스플레이처럼 표시합니다.

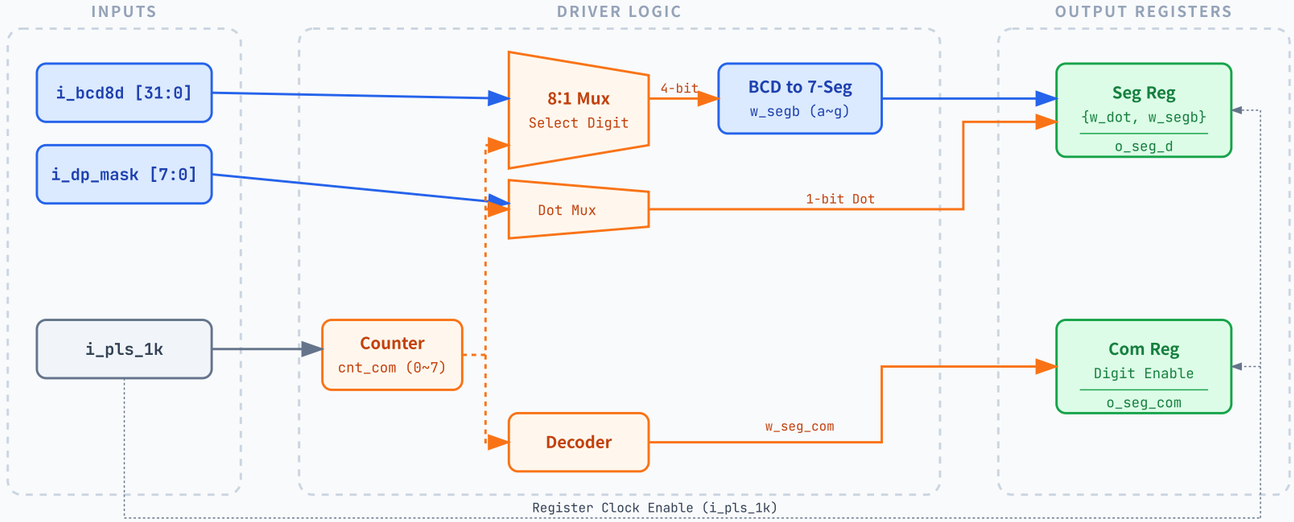
Logic Details

Refresh: 1kHz Pulse 사용 (1ms per digit)

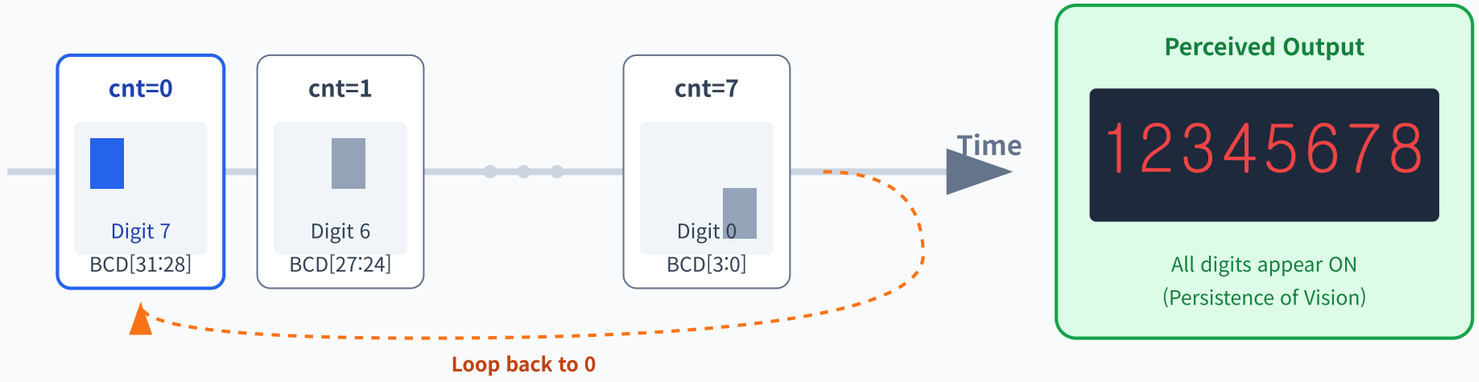
Counter: 0~7 순환 카운터로 Common 단자 선택

Decoder: 현재 Digit의 BCD 값을 a~g 신호로 변환

1. Internal Data Path & Multiplexing Logic



2. Multiplexing Timing Diagram



Input System & Control Flow Verification

Testbench 입력: 키패드로 1~8 순차 입력 → Enter 키로 FFT 실행 → 전체 흐름 검증

검증 시나리오

Test Case: 키패드를 통해 x[0]~x[7] 입력 (1~8) 후 Enter 키로 FFT 연산 시작

Phase 1: 데이터 입력

- **key_push[4:0]:** 키 입력 시퀀스 검증
- **i_key_valid:** Debouncing 완료 신호
- **i_is_digit & i_digit:** 숫자 디코딩 정확성
- **i_is_next:** 샘플 인덱스 증가 제어

Phase 2: FFT 실행

- **i_is_ent:** Enter 키 감지
- **o_fft_start:** 1-cycle 시작 펄스
- **i_fft_busy:** 연산 진행 상태
- **i_fft_done:** 연산 완료 신호

검증 결과

✓ 키 스캔 & 디코딩 정상 동작
✓ 입력 시퀀스 제어 정확
✓ Handshaking 프로토콜 완벽
✓ 입력 {1,2,3,4,5,6,7,8} 성공

Phase 1: Data Input Sequence

Fig 1-1. 데이터 입력 시퀀스 (x[0]~x[7]: 1→2→...→8)
숫자 키 입력 → Next 키 → 다음 숫자 입력 반복 (key_push: 07→0f→08→0f...)

Phase 2: FFT Execution (Start→Busy→Done)

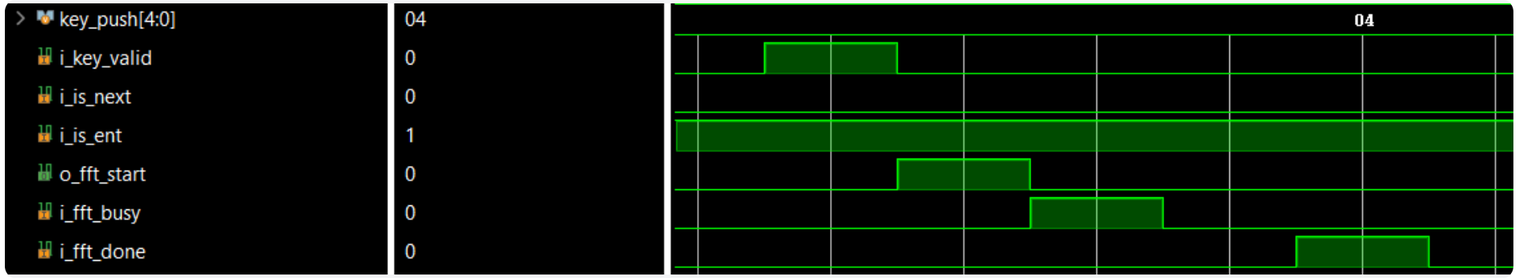


Fig 1-2. FFT 연산 실행 및 Handshaking 프로토콜 검증
Enter(04) → o_fft_start 펄스 → i_fft_busy → i_fft_done 시퀀스

FFT Accuracy Test Results

검증 시나리오

Test Case: x[n] = [1,2,3,4,5,6,7,8] 입력 후 Python FFT 결과와 비교

Test Input

- **입력 샘플:** [1,2,3,4,5,6,7,8]
- **데이터 타입:** 실수 (Real)
- **비트폭:** 21-bit Fixed Point

Verification Method

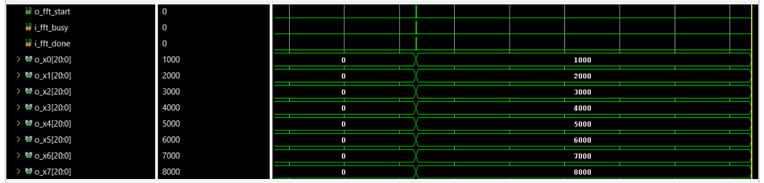
- **Reference:** Python numpy.fft
- **비교 대상:** Real & Imaginary Parts
- **측정 지표:** 상대 오차율 (%)

검증 결과

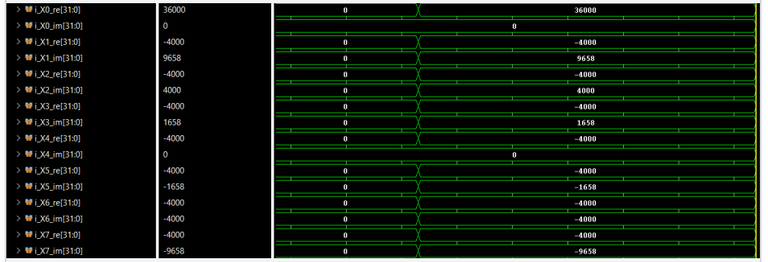
✓ 실수부 평균 오차: 0.00000%
✓ 허수부 평균 오차: 0.02701%
✓ 전체 평균 오차: **0.01157%**
✓ 하드웨어 구현 정확도 검증 완료

FFT Input/Output Data

Input Data



Output Data



FFT 연산 결과 비교 (Reference vs User Implementation)

Index	Real Part (실수부)			Imaginary Part (허수부)		
	Ref	User	Err(%)	Ref	User	Err(%)
X[0]	36.000	36.000	0.00%	0.000	0.000	0.00%
X[1]	-4.000	-4.000	0.00%	9.657	9.658	0.01%
X[2]	-4.000	-4.000	0.00%	4.000	4.000	0.00%
X[3]	-4.000	-4.000	0.00%	1.657	1.658	0.07%
X[4]	-4.000	-4.000	0.00%	0.000	0.000	0.00%
X[5]	-4.000	-4.000	0.00%	-1.657	-1.658	0.07%
X[6]	-4.000	-4.000	0.00%	-4.000	-4.000	0.00%
X[7]	-4.000	-4.000	0.00%	-9.657	-9.658	0.01%

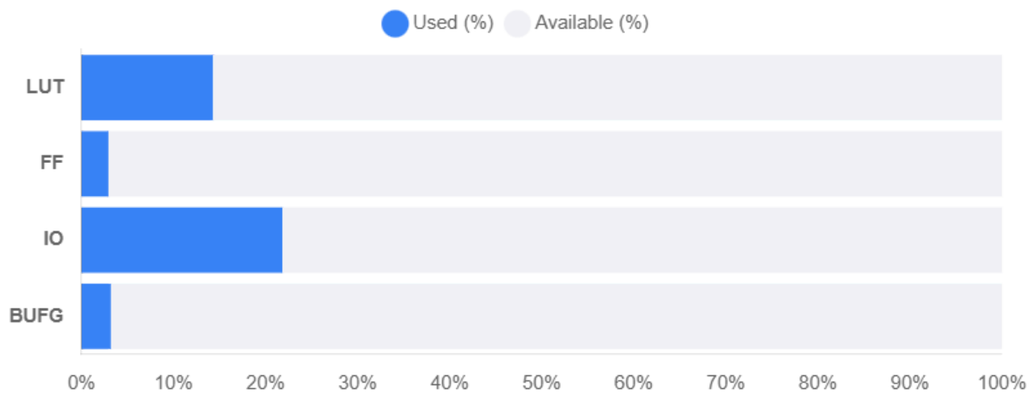
요약 결과 (0인 항목 제외)

실수부	허수부	전체
0.00000%	0.02701%	0.01157%

Hardware Implementation

Synthesis & Timing Results

FPGA Resource Utilization



LUT	FF	IO	BUFG
14.2%	2.9%	21.8%	3.1%
2956/20800	1190/41600	37/170	1/32

Timing Report

Timing Constraints MET
All user specified timing constraints are met.

Setup		Hold		Pulse Width	
WNS:	80.501 ns	WHS:	0.044 ns	WPWS:	49.500 ns
TNS:	0.000 ns	THS:	0.000 ns	TPWS:	0.000 ns
Failing:	0	Failing:	0	Failing:	0
Total:	2287	Total:	2287	Total:	1191

No timing violations were found. The design meets all setup and hold requirements with significant margin.

Conclusion & Future Work

HW 설계부터 HW 구현 및 검증까지 전체 시스템 구현 완료

Project Outcome

- ✓ **계층적 설계:** 모듈 간 의존성 최소화 및 재사용성 증대
- ✓ **자원 최적화:** Multiplier연산을 ADD, Shift 연산으로 대체, 연산 횟수 개선
- ✓ **완성도:** 실시간 입력 및 즉각적인 결과 확인 가능

Future Plan

- **Floating Point:** IEEE 754 도입으로 정밀도 향상
- **Pipelining:** 메모리 기반 순차 처리로 대규모 FFT 지원
- **복소수 입력:** 실수 입력에서 복소수 입력으로 확장
- **N-Point FFT:** 1024-point 등 대규모 FFT로 확장

💡 주요 과제 및 해결 방법

⚠ Challenge 1: 타이밍 위반

FFT 연산 구현 중 타이밍 위반(Timing Violation)이 발생하여 클럭 제약 조건을 만족하지 못함

✓ Solution

3-Stage 파이프라인 구조로 설계를 분할하여 각 스테이지의 Critical Path를 단축시킴으로써 타이밍 제약 만족

⚠ Challenge 2: 코드 가독성 및 구현 복잡도

$1/\sqrt{2}$ 곱셈 연산이 반복적으로 등장하여 코드 가독성이 떨어지고 구현이 복잡해짐

✓ Solution

Verilog Function을 활용하여 Twiddle Factor 곱셈 로직을 모듈화하고 재사용성을 높여 코드 간결화 및 유지보수성 향상

Thank You

Q & A

디지털 FPGA 설계

전민서 | 한정호