

National Tsing Hua University

# FINAL PROJECT SPEC

2025/5/13

Kuan-Hsi (Vic) Chen

## Hardware Design

- Function Specification
- Design Specification
- Architecture Framework
- Configuration Register Access Protocol
- Testbench Specification

## Software & firmware

- Middleware
- Falcon Host

# Hardware Design

# Function Specification

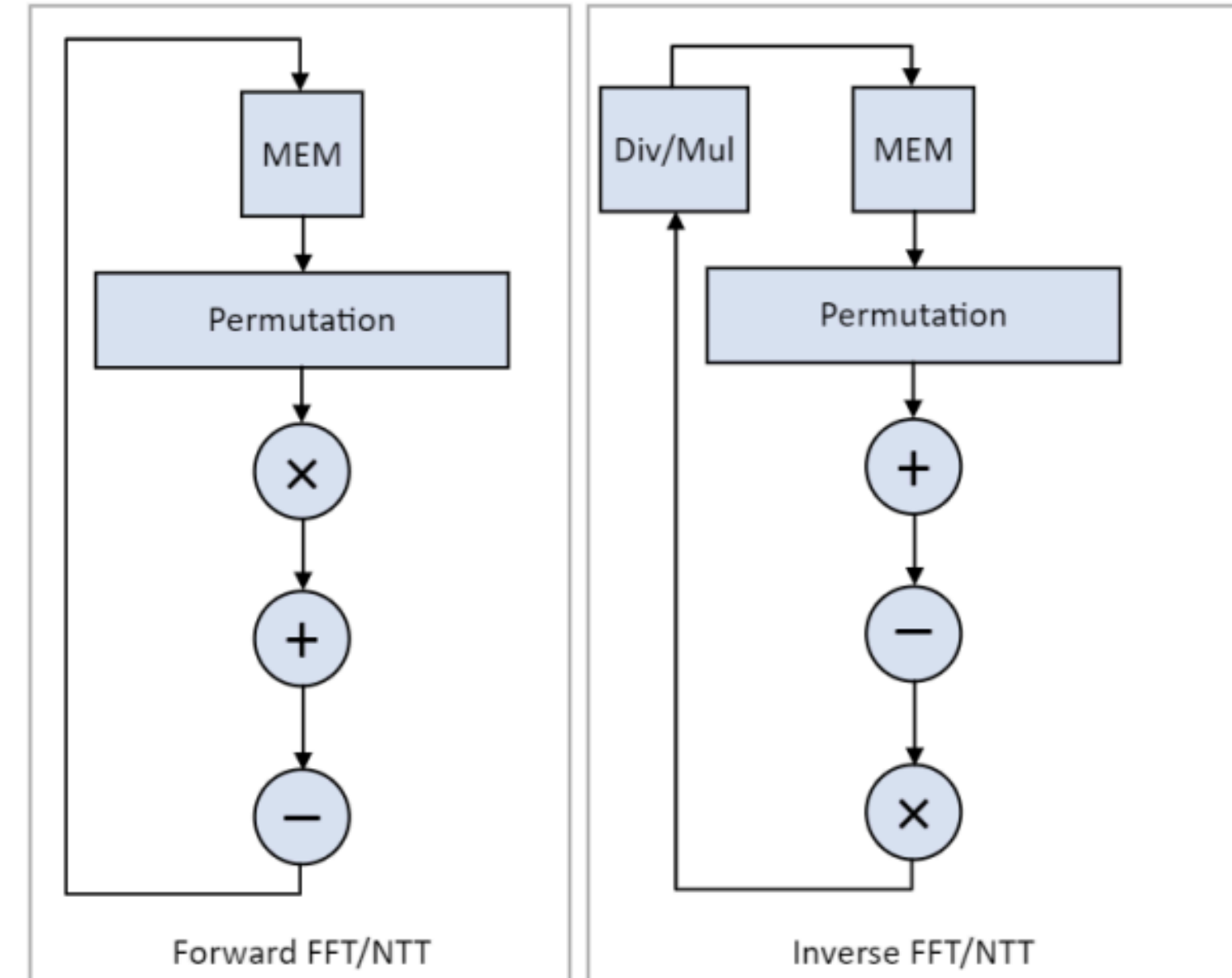
- DFT

$$\text{DFT : } X[k] = \sum_{i=0}^{n-1} x[i] e^{\frac{-j2\pi ik}{n}}, k = 0, 1, 2, \dots, n-1$$

- NTT

$$\text{NTT : } \tilde{a}[i] = \sum_{j=0}^{n-1} x[j] \omega^{ij} \bmod q, \text{ for } i = 0, 1, \dots, n-1$$

- Implement DFT & NTT in  $O(N \log N)$



# Design Specification

## Data Width

- Stream-in / out 32-bit
- Internal (IOP <-> BPE) data stream: 128-bit (64-bit real + 64-bit imag)

Data Num - Based on the first data DMA stream-in (kernel mode)

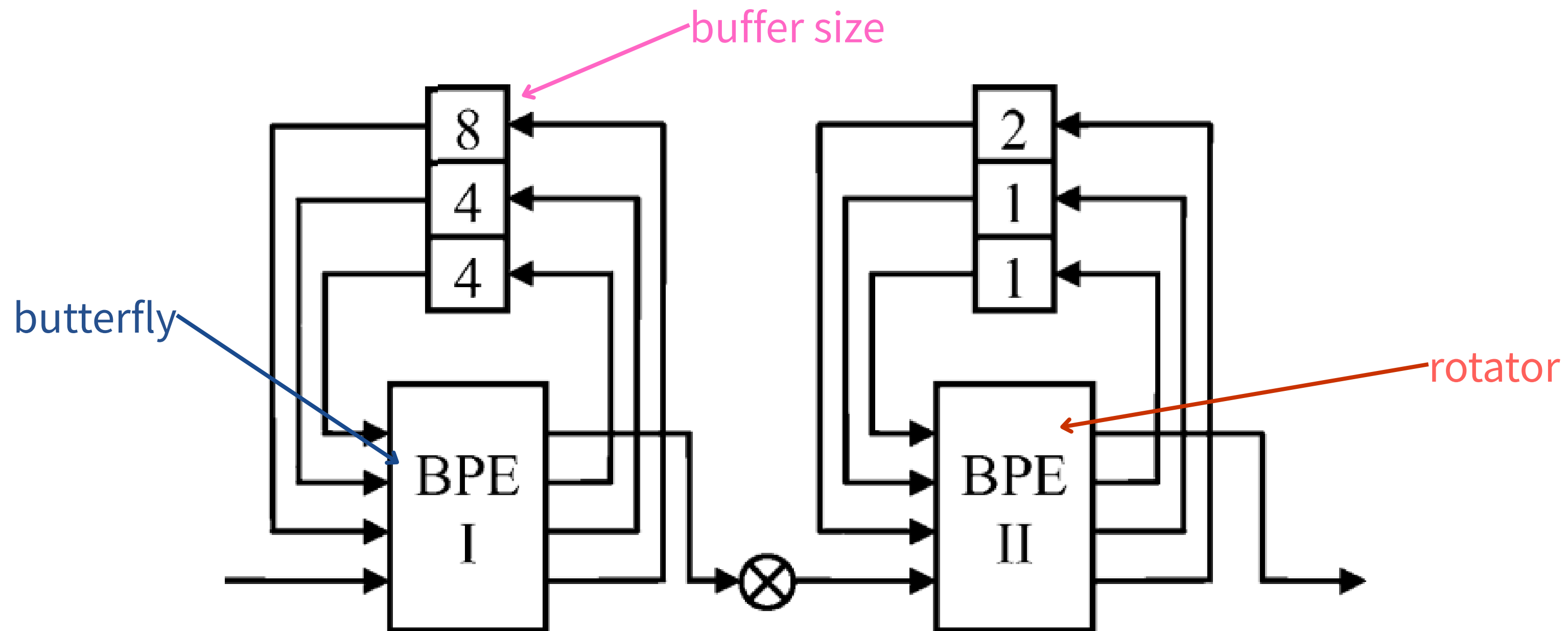
## Interface

- AXI-Lite: for status reading / write **coef\_done** after coefficients are initialized
- AXI-Stream: Stream-in  $f[i]/n[i]$  , Stream-out  $F[i], N[i]$

## Coefficient implemented with SRAM

- F\_RAM: 1024 DW, increase the bandwidth to 128-bit (read [real](#) + [imag](#) part in 1 T)
- N\_RAM: 1024 DW, [16-bit unsigned integer](#)
- iN\_RAM: 1024 DW, [16-bit unsigned integer](#)

# SDF Deep-Feedback scheme



R2SD<sub>2</sub>F pipeline architecture for 16-point DFT.

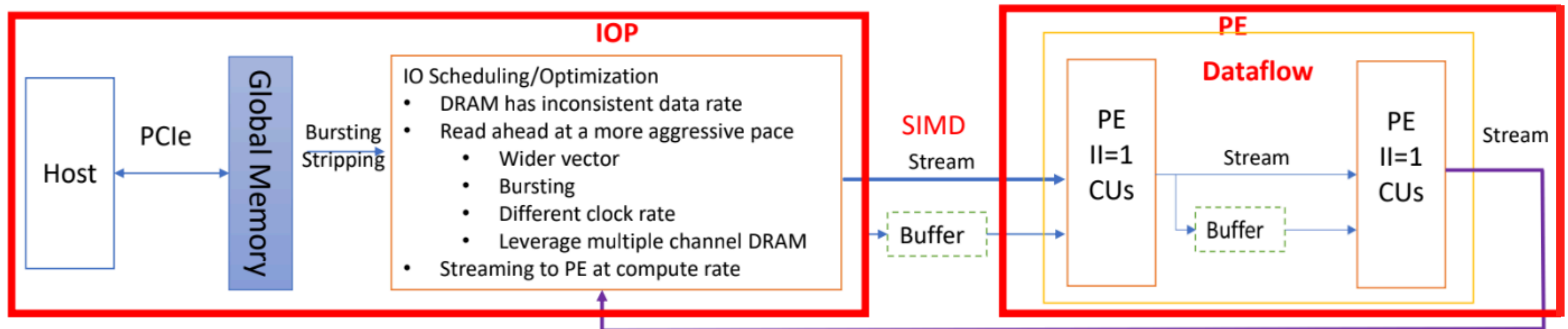
# Architecture Framework (1/2)

## IOP (stage\_top)

- Get stream-in data, memory partition, and stream-out data to PE
- Control the parameter for each stage

## BPE (butterfly processing element)

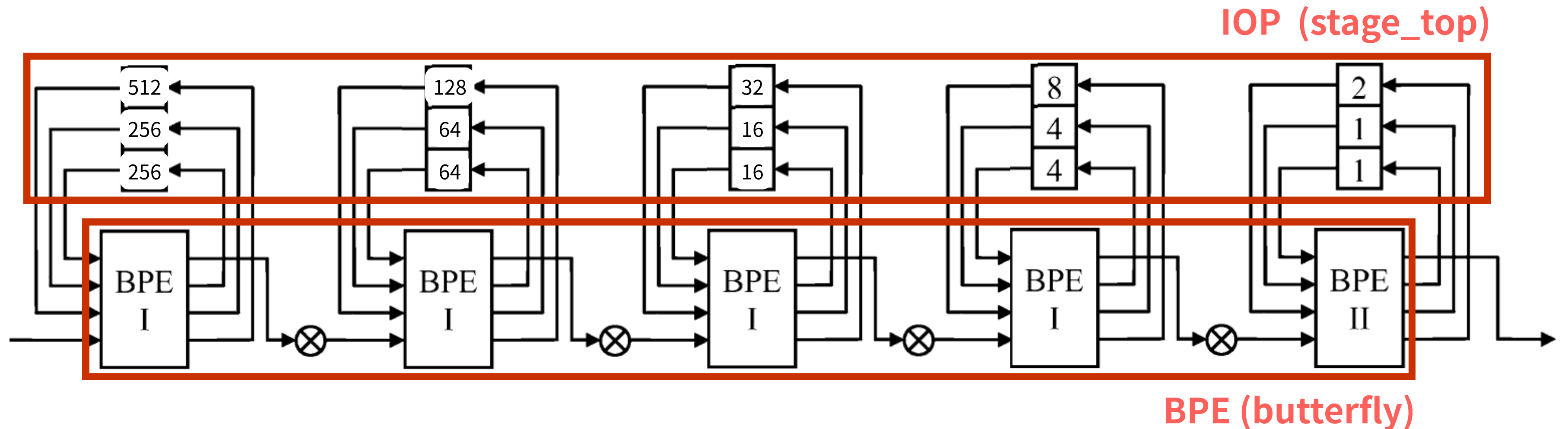
- Complex / Montgomery ADD / SUB / MUL



# Architecture Framework (2/2)

## fiFFNTT

- 10 → 5 BPEs
  - 100% butterfly usage compared to the original SDF





# Mapping algorithm into R2SD2F scheme

## FFT / iFFT (512 complex points => 1024 num)

- Complex num (first 512: real part, second half: imaginary part)
- 64-bit double precision floating point
- The first stage is completed, see the [link](#)
- Start from the second stage (total 9 stages)
- Operators in BPE: [complex mul](#), [complex add](#), [complex sub](#)
- For the *dividing by N* in inverse FFT, use the [shifting technique](#)

## NTT / iNTT (1024 real num points)

- 16-bit integer
- 10 stages
- Operators in BPE: [mq montymul](#), [mq add](#), [mq sub](#)

# Complex Multiplication

Original:  $Z_r = X_r \cdot Y_r - X_i \cdot Y_i$   
 $Z_i = X_r \cdot Y_i + X_i \cdot Y_r$       4 mul, 2 add

Optimize:  $Z_r = X_r \cdot (Y_r - Y_i) + Y_i \cdot (X_r - X_i)$   
 $Z_i = X_i \cdot (Y_r + Y_i) + Y_i \cdot (X_r - X_i)$       3 mul, 5 add

=> reduce the area

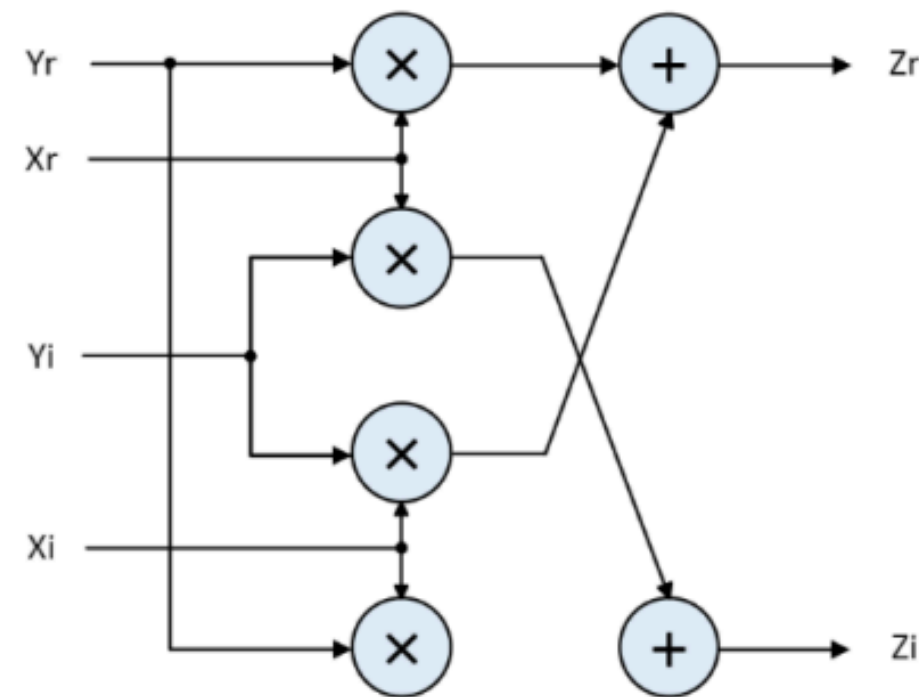


Fig. 2-10 Direct complex multiplier

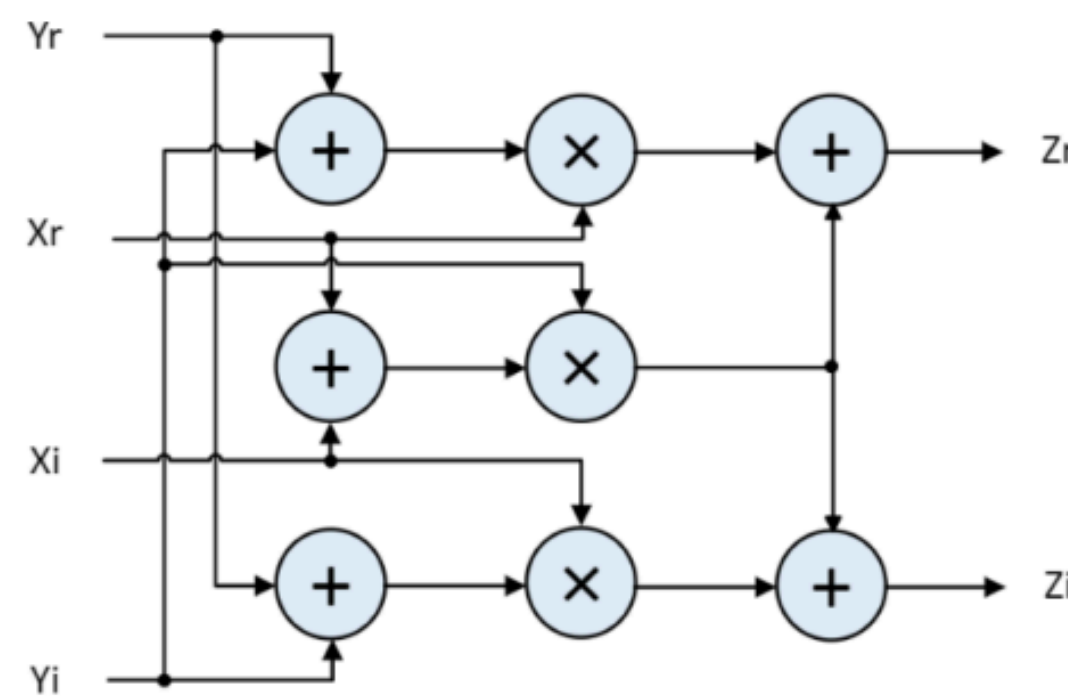
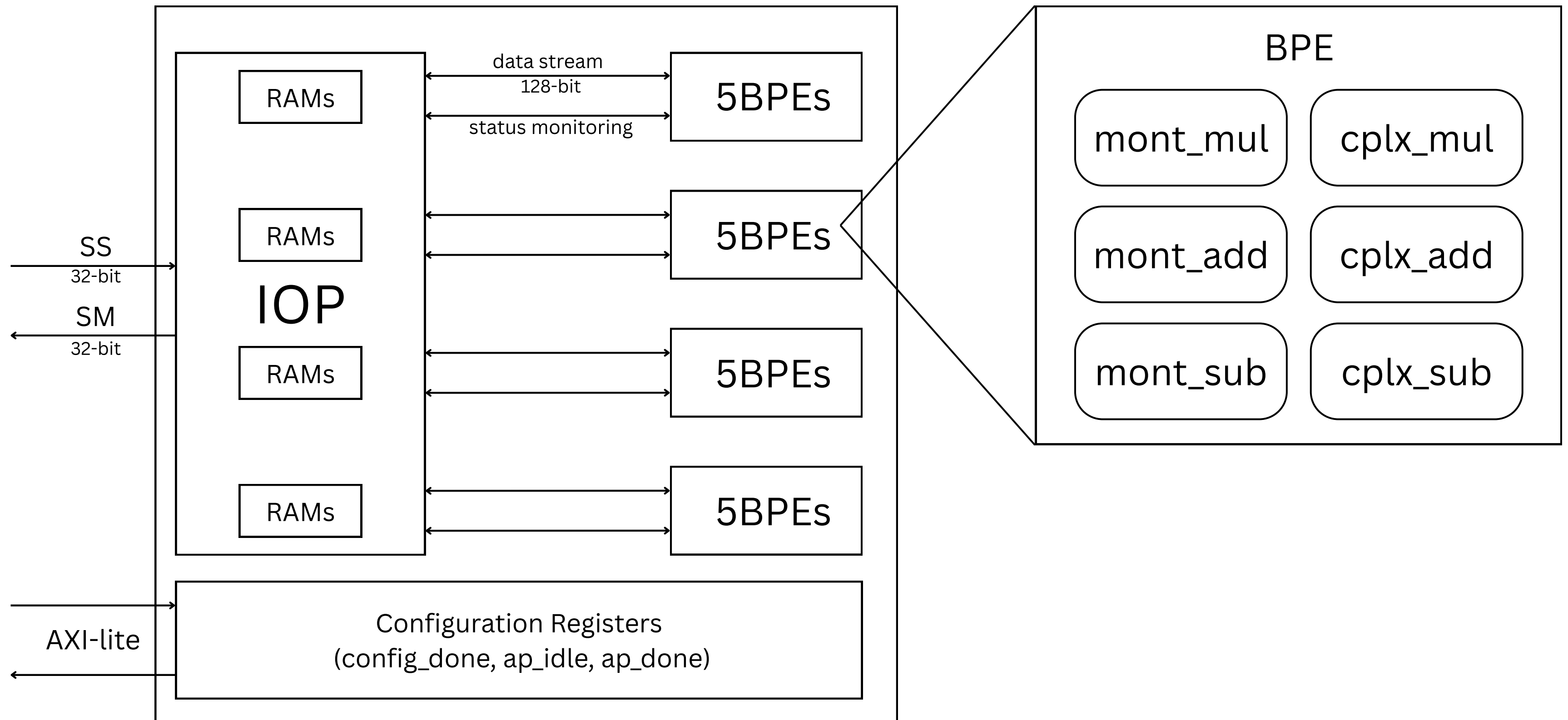


Fig. 2-11 Improved complex multiplier

# Block Diagram



# Folder Structure / Interface Ports

- Top: composition script (makefile, file.txt)
  - rtl
    - fiffNTT.v
      - stage\_top.v (IOP)
      - butterfly.v (BPE)
        - umul16
        - uadd16
        - usub16
        - fmul64
        - fadd64
        - fsub64
  - sim
    - falcon.v (after “source concat.sh”)
    - tb.v

# Other Optimization Techniques

Can refer to our previous report:

- <https://drive.google.com/file/d/1rCroW8wXty7tp5wPHU3g-swqUZEjEUsX/view?usp=sharing>

# Configuration Register Access Protocol for MailBox scheduling

# Configuration Register Address map

## **0x00 -**

[0] - **ap\_done** status

Middleware (RISC-V CPU) will read the status of each kernel, and write to MB

[1] - **ap\_idle** status

De-asserted when DMA streams the first data. (no ap\_start)

## **0x10 - coef\_done**

1: Indicate fiFFNTT coef are initialized; 0: stream-in data are for coef

## **First stream data - kernel mode**

Kernel is configured by the first stream-in data

# ap\_done protocol and implementation

1. Read clear register
2. ap\_done is asserted when the engine completes the last data processing, and data is transferred.
3. ap\_done is reset in the following condition
  - a. reset signal is asserted
  - b. after a task is complete, the ap\_done is cleared by reading address 0



# ap\_idle protocol and implementation

1. ap\_idle is set to 1 when reset
2. ap\_idle is set to 0 when DMA stream-in the first data (kernel mode)
3. ap\_idle is set to 1 when the engine processed the last data and last data is transferred

# Testbench Specification

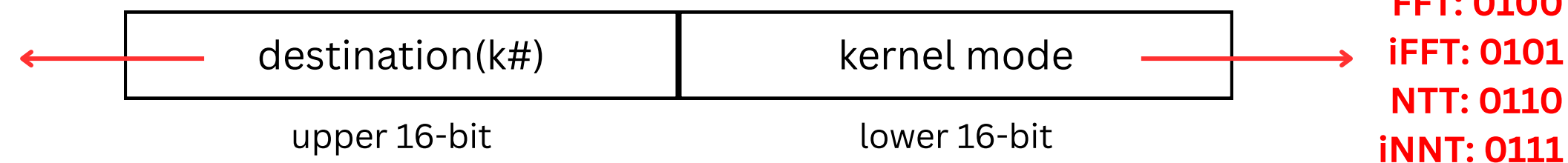
# Develop testbench - simulate DMA&FW behavior (1/2)

- Setup phase
  - a. Define 4 fiFFNTTs and give each an ID (Ex: k1, k2, k3, k4)
  - b. Load datafile and stream in coefficients (FFT/iFFT/NTT/iNTT constants) - DMA
  - c. Configure **0x10** (coef\_done) - Host
  - d. Define and Initialize 4 MailBoxs (write 0x3a3a3a3a) w.r.t k1, k2, k3, k4 in Testbench - FSIC/FPGA
- Execution phase
  - a. Check 4 MBs if there is any 0x3a3a3a3a => decide use which kernel - Host
  - b. Write 0x5a5a5a5a (means kernel is used) to MB w.r.t the idle kernel ID - Host
  - c. Stream-in the first data (ap\_idle<=0) which will decide the kernel mode and destination - DMA ([link](#))

```
// First data: program kernel mode - 0: FFT, 1: iFFT, 2: NTT, 3: iNTT
```

```
out_val.data_filed = (kernel_mode == 0)? 4/*{27'b0, 01, 00}*/: (kernel_mode == 1)? 5/*{27'b0, 01, 01}*/:  
    (kernel_mode == 2)? 6/*{27'b0, 01, 10}*/: (kernel_mode == 3)? 7/*{27'b0, 01, 11}*/: 0;
```

k0: 0100  
k1: 0101  
k2: 0110  
k3: 0111



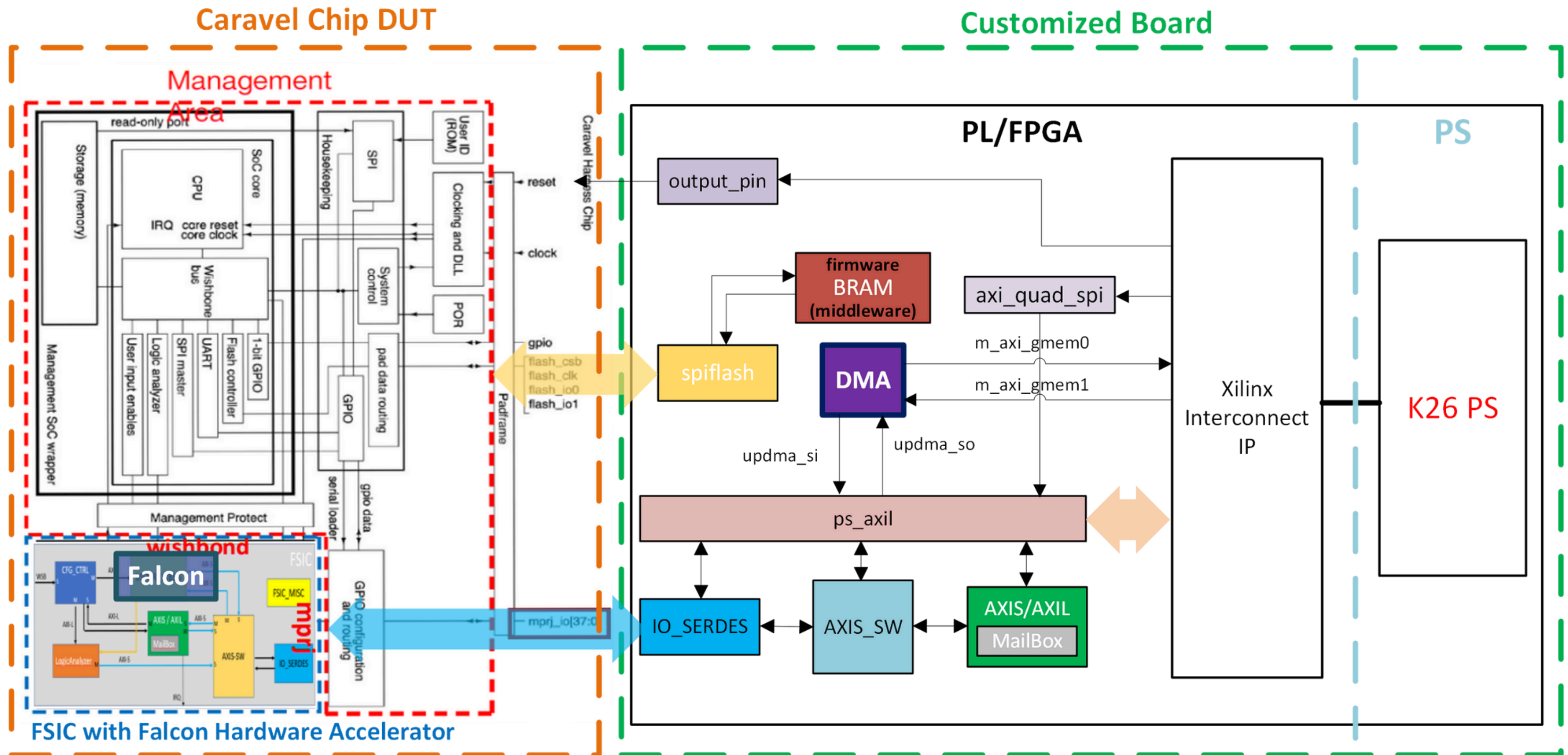
# Develop testbench - simulate DMA&FW behavior (2/2)

- d. Start latency timer
- e. Fork the following operations, run concurrently
  - i. Stream-in  $f[n]$  (refer to [tb.sv](#)) - DMA
    - 1. For FFT/iFFT, split 64-bit data to upper/lower 32-bit
    - 2. For NTT/iNTT, stream {16'd0, 16-bit data}
  - ii. Stream-out  $F[n]$  - DMA
  - iii. Polling `ap_done` and write 0x3a3a3a3a to MB w.r.t the done kernel - Middleware (FW)
- Checking phase
  - a. Report latency
  - b. Compare output with golden data

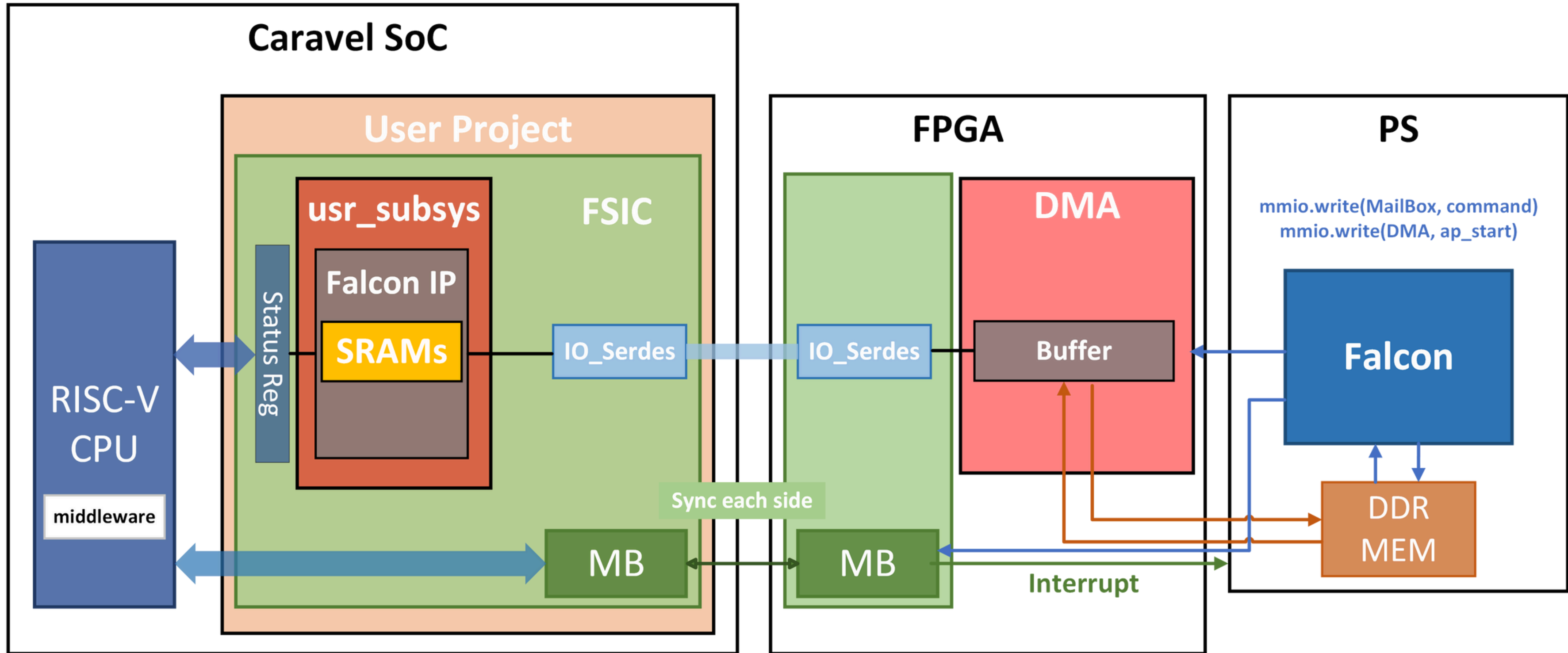
**Repeat for 4 times (FFT, iFFT, NTT, iNTT)**

# Software / Firmware

# System View



# Brief Block Diagram



# Definition

- **Middleware**

- Polling ap\_done of each kernel => write pattern(0x3a3a3a3a) into MB (0x3000\_2000~3000\_201f)  
addr[4:2] decide which MB to store (we have 8, use 4)

- **Falcon Host (on-board)**

- Enable MailBox irq using mmio.write(PL\_AA) `## Write aa_mb_irq_en ##  
mmio.write(0x2100, 0x01)`
- Replace SW FFT/iFFT/NTT/iINTT to call our HW accelerator ([refer to on-board validation](#))
  - Check MB if there is any free kernel (0x3a3a3a3a) and write 0x5a5a5a5a to that MB
  - Check DMA status, and configure DMA for kernel mode decision
  - Load data into AXI-M buffer => ap\_start
  - If receive irq (interrupt) => return value (answer)
- Write kernel function as deferrable functions
- Modify host code to expose 3 APIs (KeyGen, Sign, Vrfy) to user

- **DMA**

- Get data from host memory (AXIM) and stream into kernel
- Get stream-out data from kernel, store into host memory



# DMA SPEC

- FFT/iFFT **m2s** (DMA stream out 2049 data into kernel)
  - First data: kernel\_mode
  - 2048 data: upper 32-bit and lower 32-bit of 1024 “double precision floating point” data
- NTT/iNTT **m2s** (DMA stream out 1025 data into kernel)
  - First data: kernel\_mode
  - 1024 data: 16-bit of 1024 “uint\_16” data
- FFT/iFFT **s2m** (stream in 2048 data)
  - upper 32-bit and lower 32-bit of 1024 “double precision floating point” data
- NTT/iNTT **s2m** (stream in 1024 data)
  - 1024 16-bit “uint\_16” data