Bridge of Life Education

Advance SOC
Final Project Proposal
Team 2

Project Title:
Implementation of the Falcon Algorithm: Applying High-Level Synthesis to Post-Quantum Cryptography

Content of Final Project Proposal

- Team: Leader + Members
- Problem statement
- Project scope
- Project plan
- Reference

Team

- Leader:劉祐瑋
- Members: 陳昇達、劉佩雯

Problem Statement

- Context: PQC algorithm – Falcon https://falcon-sign.info/

Github link: https://github.com/tprest/falcon.py/tree/master

Background Introduction

- In response to the emergence of quantum computers, which pose a significant threat to existing cryptographic standards due to their potential to easily break them, post-quantum cryptography (PQC) has emerged as a critical area of research.
- Falcon stands for Fast Fourier Lattice-based Compact Signatures over NTRU. This scheme is not only a candidate in NIST's post-quantum cryptography standardization process but also one of the frontrunners, aiming to set new benchmarks for efficiency and security in the era of quantum computing.

Keygen

Algorithm 1 FALCON: Key generation

Input : No Input Required.
Output : Secret Key $sk = (f, g, F, G)$ and Public Key $pk = (h)$.

1: **procedure** KEY GENERATION ▷ crypto_sign_keypair(sk, pk)
2: Generate seed for hash function using AES. ▷ randombytes
3: Initialize the hash function using the seed. ▷ falcon_keygen_set_seed
4: **while** (1) **do**
5: Generate polynomial f and g. ▷ poly_small_mkgauss
6: Check the norm of the polynomial. ▷ poly_small_sqnorm
7: Compute the orthogonalized vector norm. ▷ "Multiple functions"
8: **if** (vector norm < 16822) **then** ▷ fpt_lt
9: "Continue"
10: **end if**
11: **if** (Fail to generate h) **then** ▷ falcon_compute_public
12: "Continue"
13: **end if**
14: **if** (Fail to generate F and G) **then** ▷ solve_NTRU
15: "Continue"
16: **end if**
17: "Break"
18: **end while**
19: Encode secret key. ▷ falcon_encode_small
20: Encode public key. ▷ falcon_encode_12289
21: **return** ($sk = (f, g, F, G)$, $pk = (h)$)
22: **end procedure**

Sign

Algorithm 2 FALCON: Signature generation

Input : Message m, Message Length $mlen$, Secret Key $sk = (f, g, F, G)$.
Output : Signature $sm = (sig\_len$, nonce, message, signature), Signature Length $smlen$

1: **procedure** SIGNATURE GENERATION ▷ crypto_sign($sm, smlen, m, mlen, sk$)
2: Generate seed for hash function using AES. ▷ randombytes
3: Initialize the hash function using the seed. ▷ falcon_sign_set_seed
4: Decode and pre-computation of secret key. ▷ falcon_sign_set_private_key
5: $c \leftarrow H(r, m)$. ▷ falcon_sign_start, falcon_sign_update
6: $s_1 + s_2 h = c \ mod \ q$.
7: Encode s_2. ▷ falcon_sign_generate
8: Copy nonce and encoded s_2 to sm. ▷ memcpy
9: **return** $sm = (sig\_len$, nonce, message, encoded $s_2)$.
10: **end procedure**

Vrfy

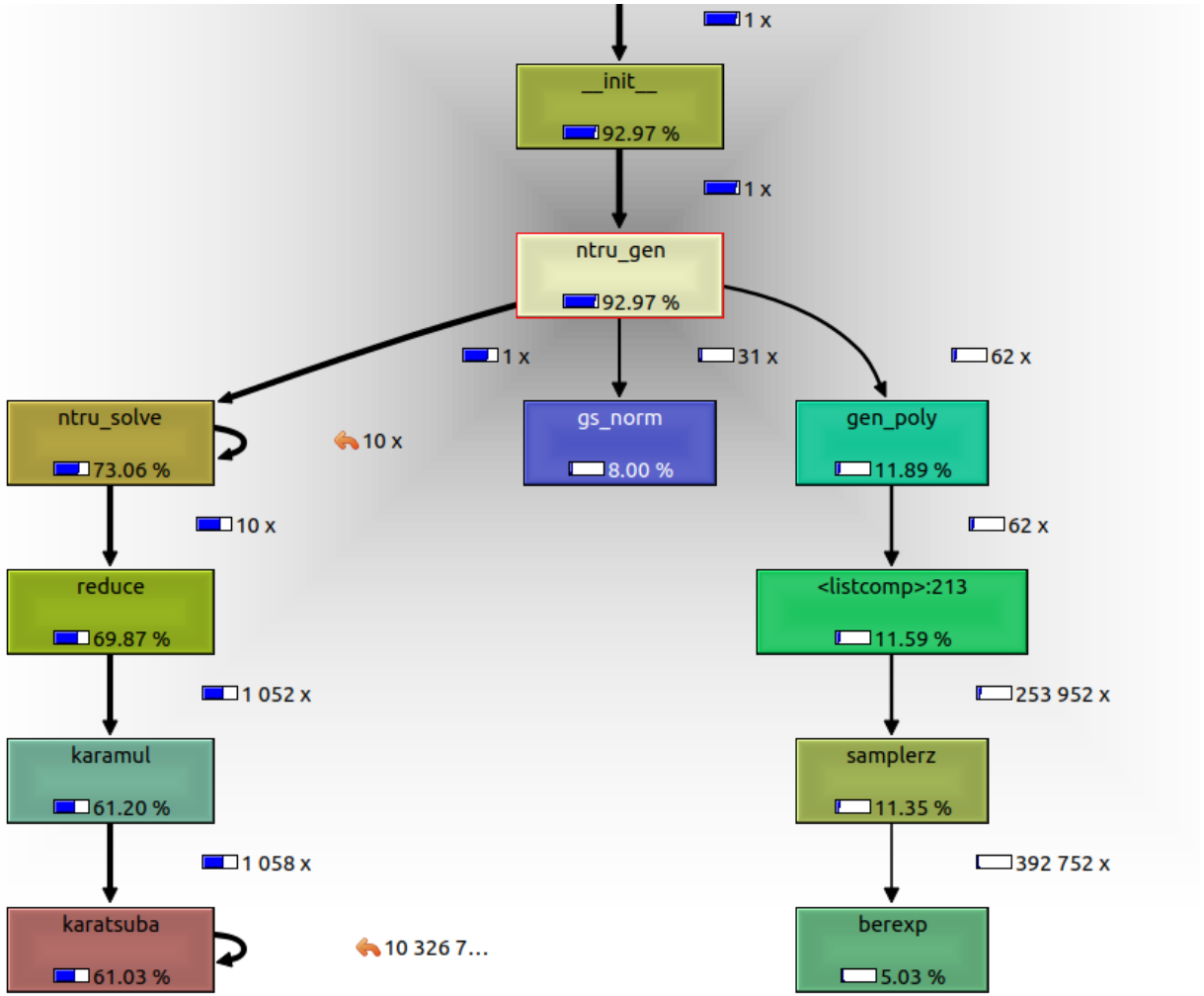
Algorithm 3 FALCON: Signature verification

Input : Signature $sm = (sig\_len$, nonce, message, encoded $s_2)$, Signature length = $smlen$, Public Key $pk = (h)$.

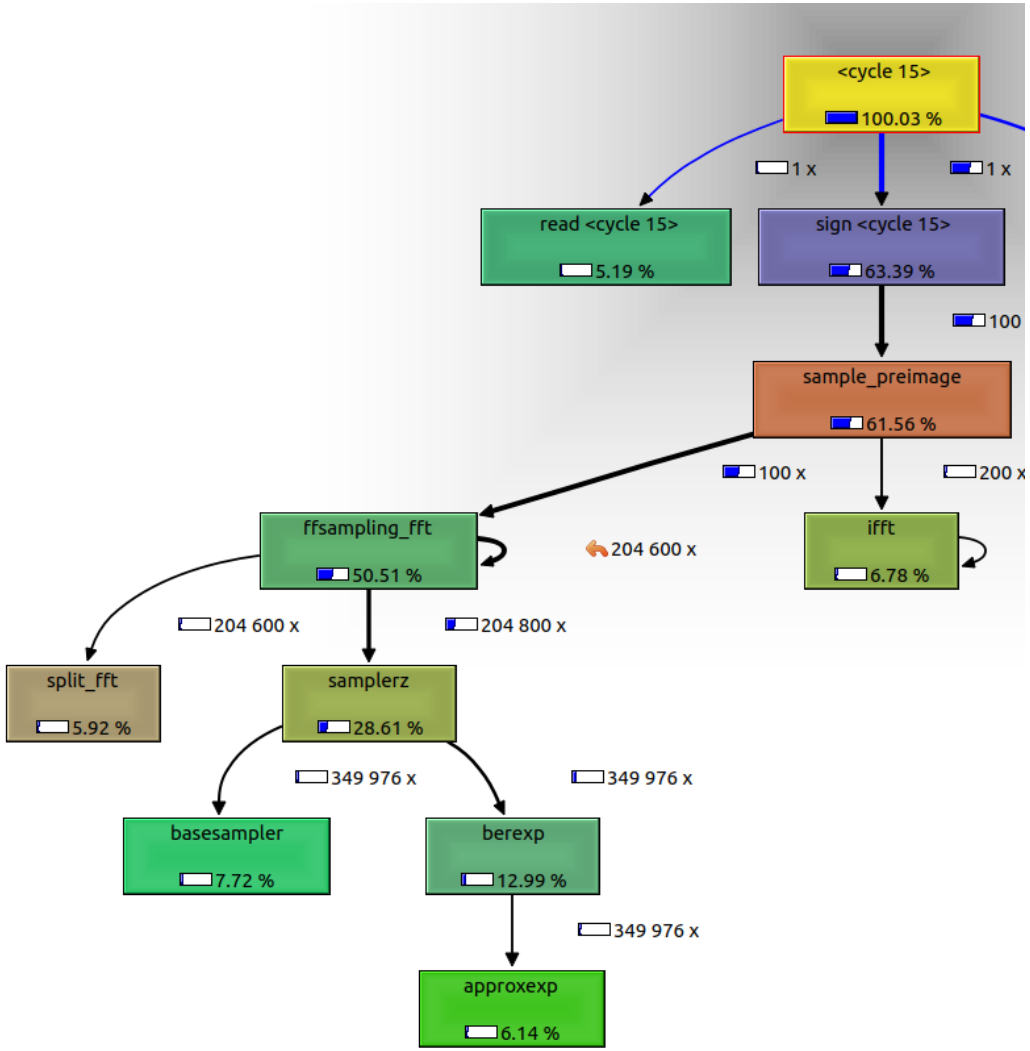
Output : "Accept"/"Reject", Message m, Message Length $mlen$.

1: **procedure** SIGNATURE VERIFICATION ▷ crypto_sign_open($m, mlen, sm, smlen, pk$)
2: Decode public key. ▷ falcon_vrfy_set_public_key
3: **if** $(smlen < (2+\text{PARAM\_NONCE}) \| sig\_len > (smlen - (2+\text{PARAM\_NONCE})))$ **then**
4: **return** "Reject"
5: **end if**
6: Initialize the hash function. ▷ falcon_vrfy_start
7: $c \leftarrow H(r, m)$ ▷ falcon_vrfy_update
8: $s_2 \leftarrow$ Decode(signature). ▷ falcon_decode_small
9: $s_1 \leftarrow c - s_2 h \ mod \ q$ ▷ falcon_vrfy_verify_raw
10: **if** $(\|s_1, s_2\| > \beta)$ **then** ▷ falcon_is_short
11: **return** "Reject"
12: **end if**
13: **return** {"Accept", m, $mlen$}
14: **end procedure**

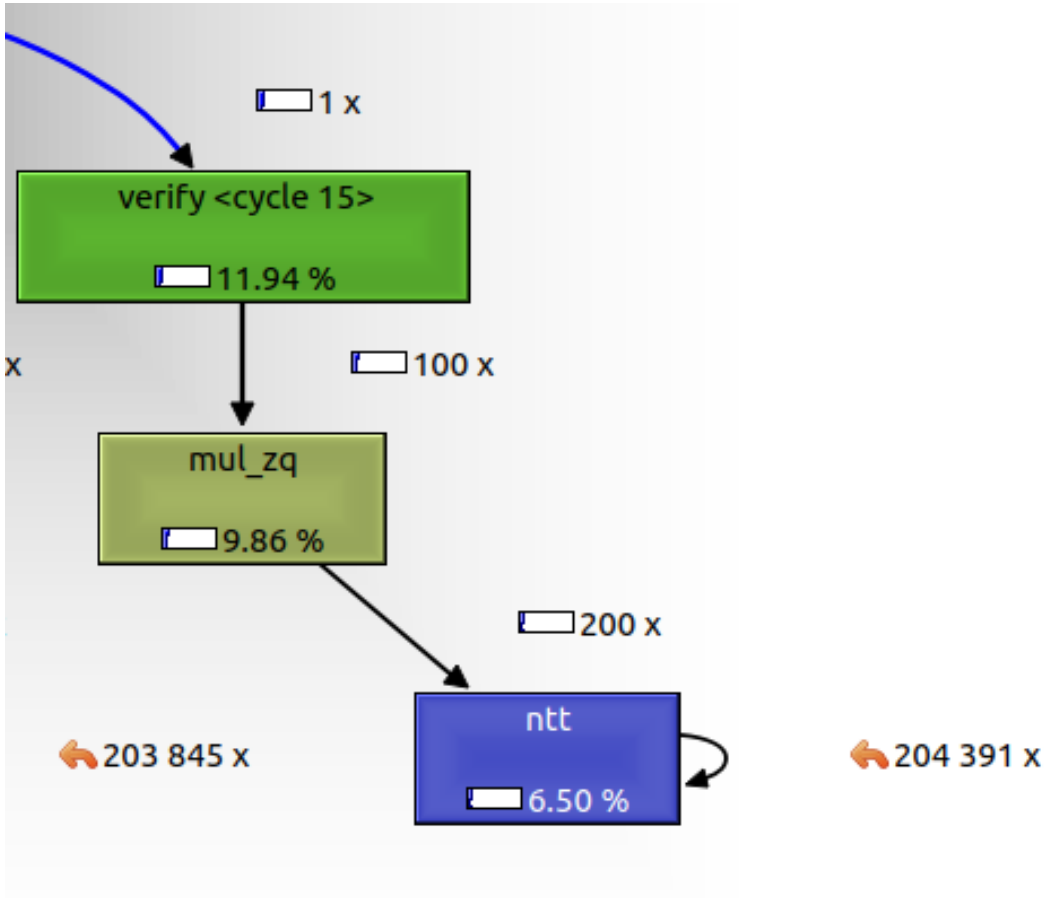
Keygen Callgraph



Sign Callgraph



Vrfy Callgraph



Problem Statement

- Issue: Takes long time looping with some critical functions

variant	keygen (ms)	keygen (RAM)	sign/s	verify/s	pub size	sig size
FALCON-512	8.64	14336	5948.1	27933.0	897	666
FALCON-1024	27.45	28672	2913.0	13650.0	1793	1280

```
Test battery for n = 1024
Test FFT              : OK          (20.706 msec / execution)
Test NTT              : OK          (22.937 msec / execution)
Test NTRUGen          : OK       (17707.189 msec / execution)
Test ffNP             : OK          (135.42 msec / execution)
Test Compress         : OK           (3.292 msec / execution)
Test Signature        : OK         (102.022 msec / execution)
```

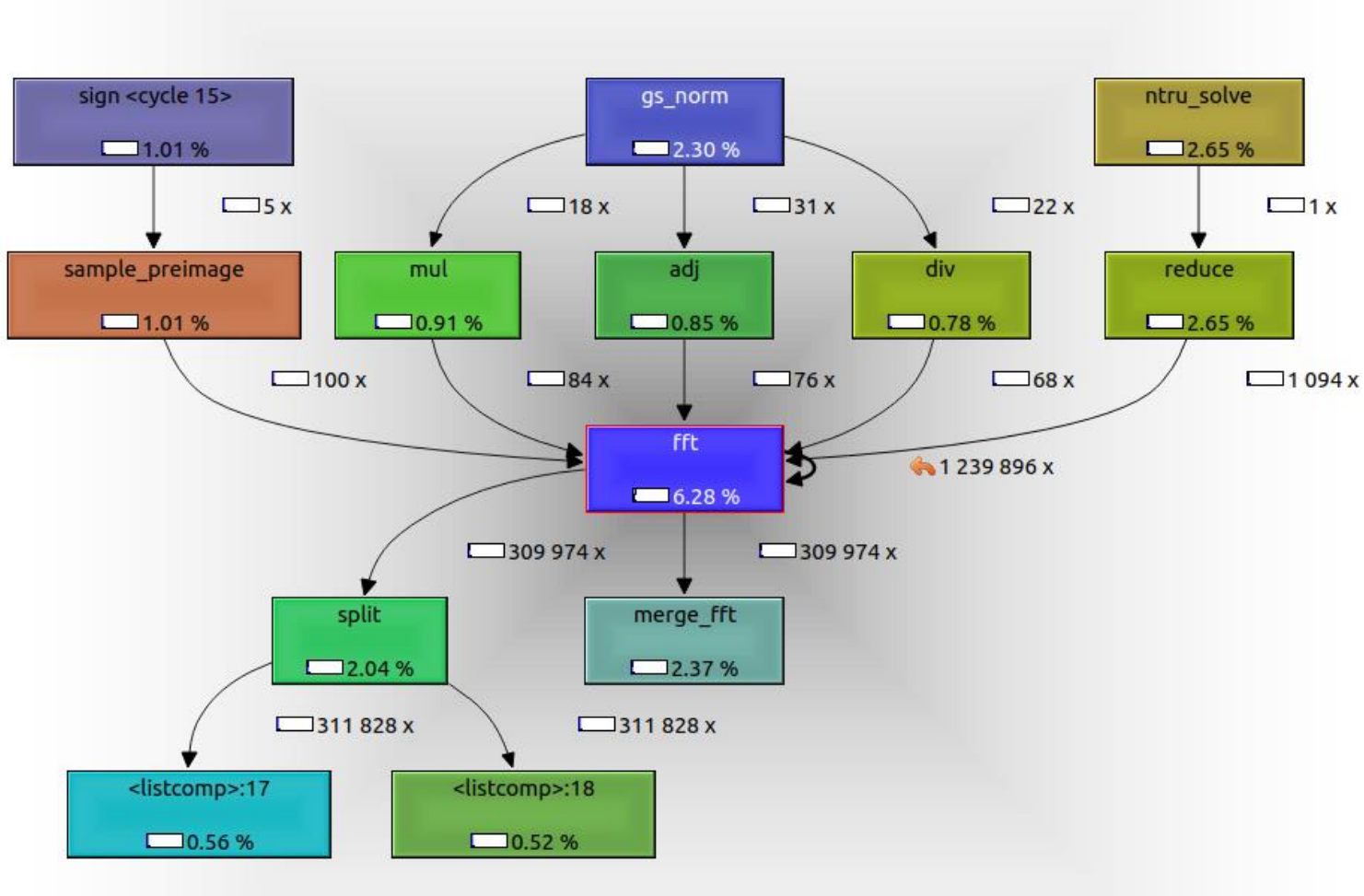
Problem Statement

- Objective: Replace those critical functions with hardware accelerators
  - Ex: FFT / iFFT / NTT / iNTT

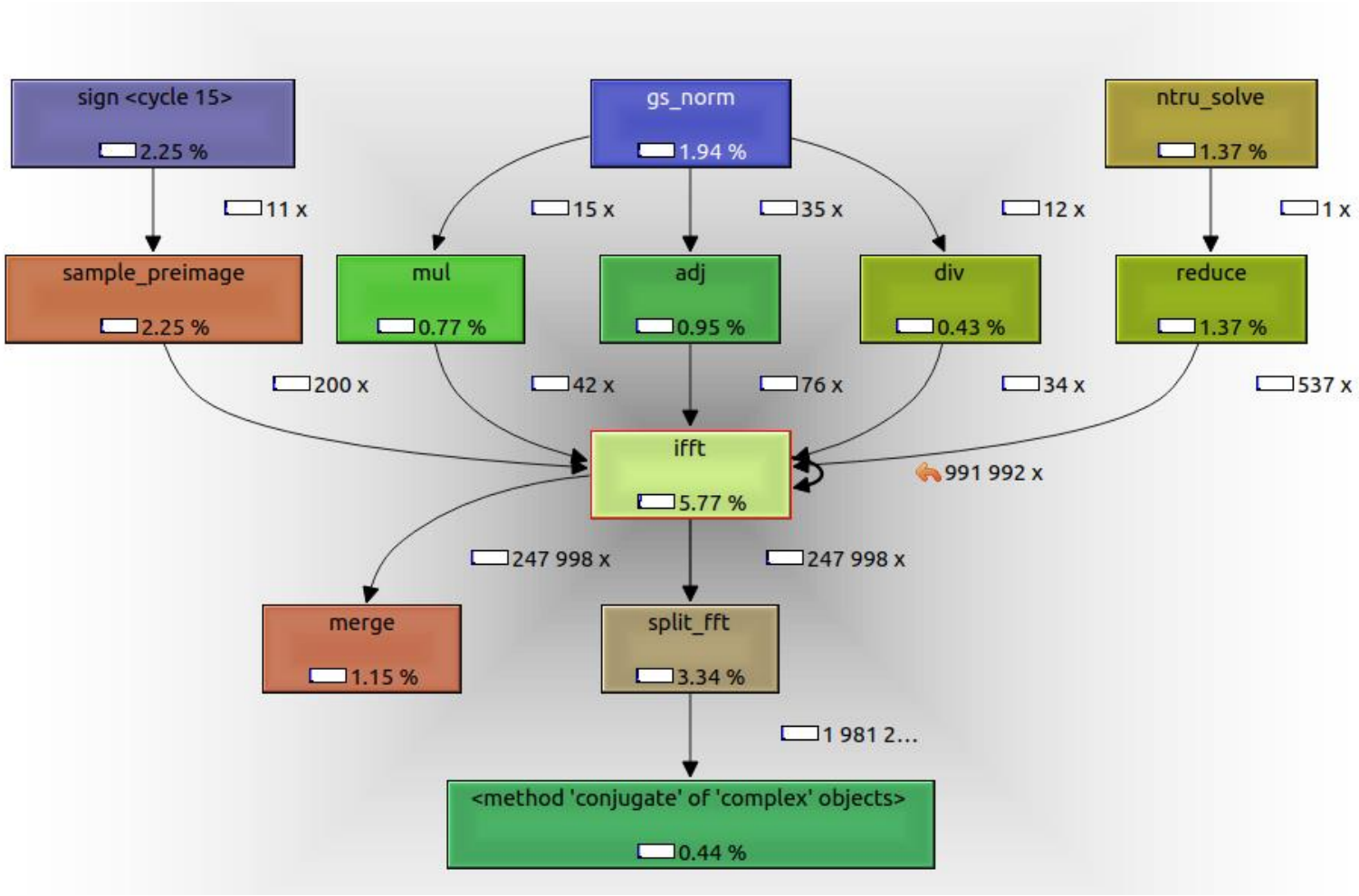
```
Test battery for n = 1024
Test FFT              : OK        (20.706 msec / execution)
Test NTT              : OK        (22.937 msec / execution)
Test NTRUGen          : OK     (17707.189 msec / execution)
Test ffNP             : OK       (135.42 msec / execution)
Test Compress         : OK        (3.292 msec / execution)
Test Signature        : OK      (102.022 msec / execution)
```

Which execute many times while looping in Falcon

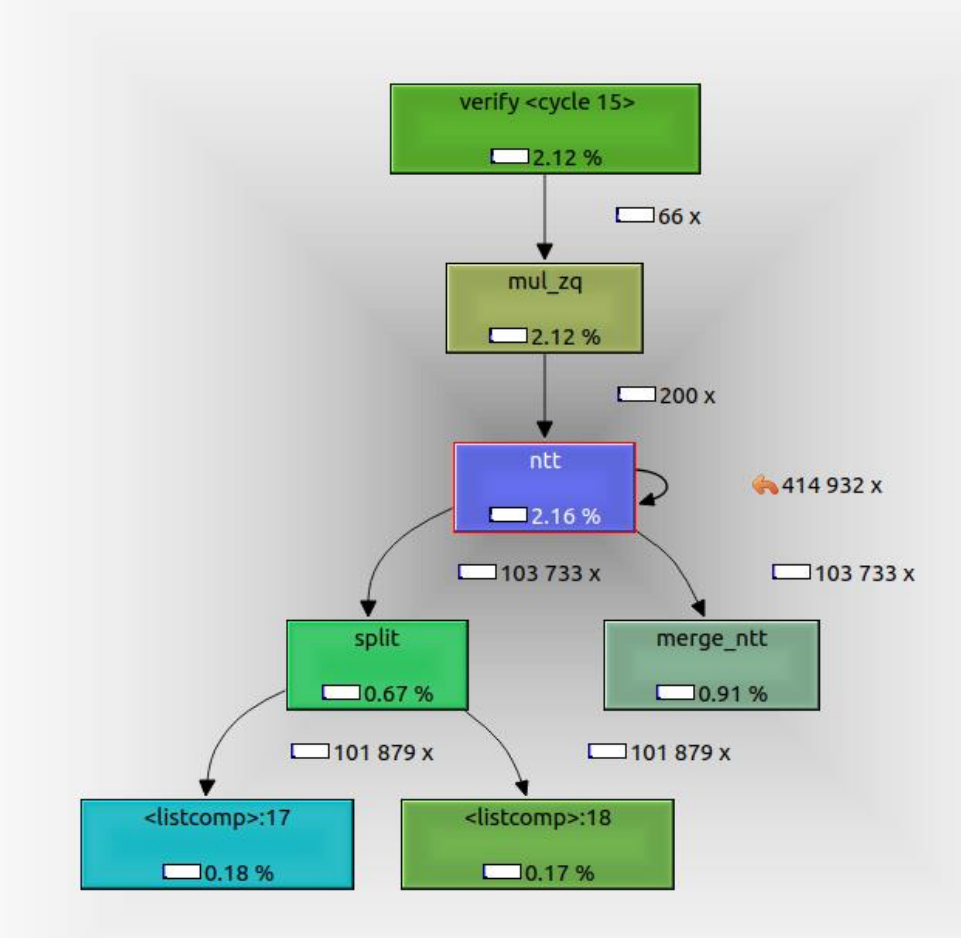
Kernel function-FFT



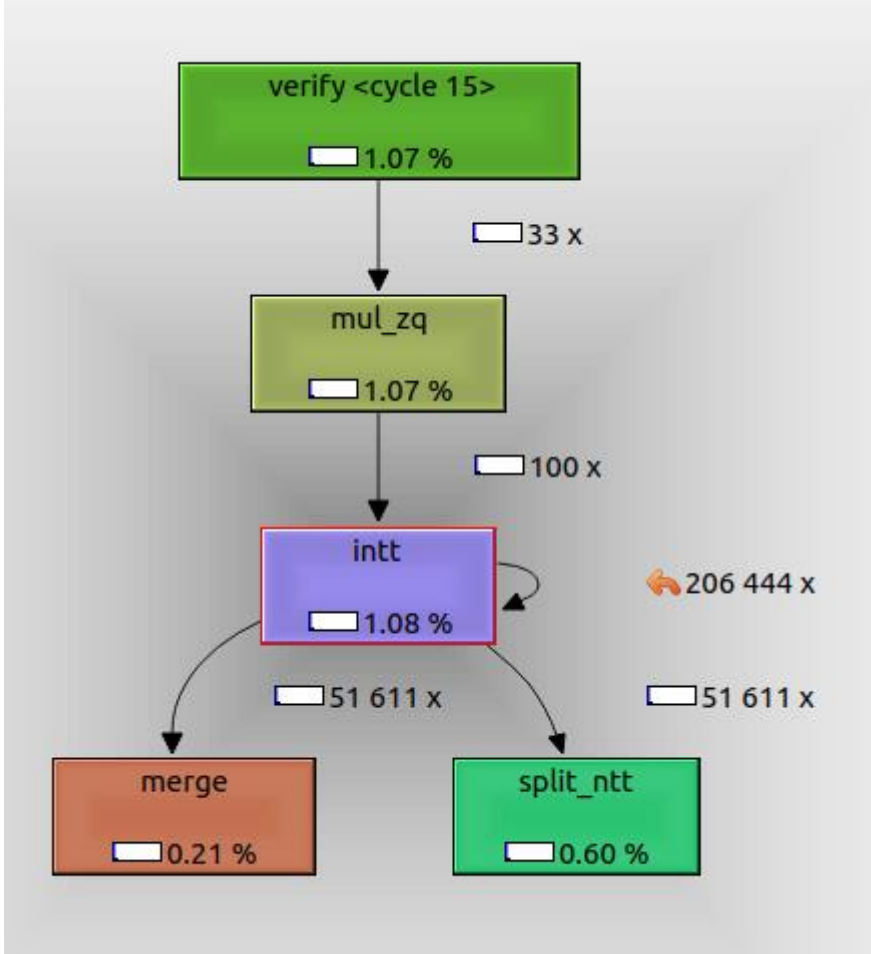
Kernel function-iFFT



Kernel function-NTT



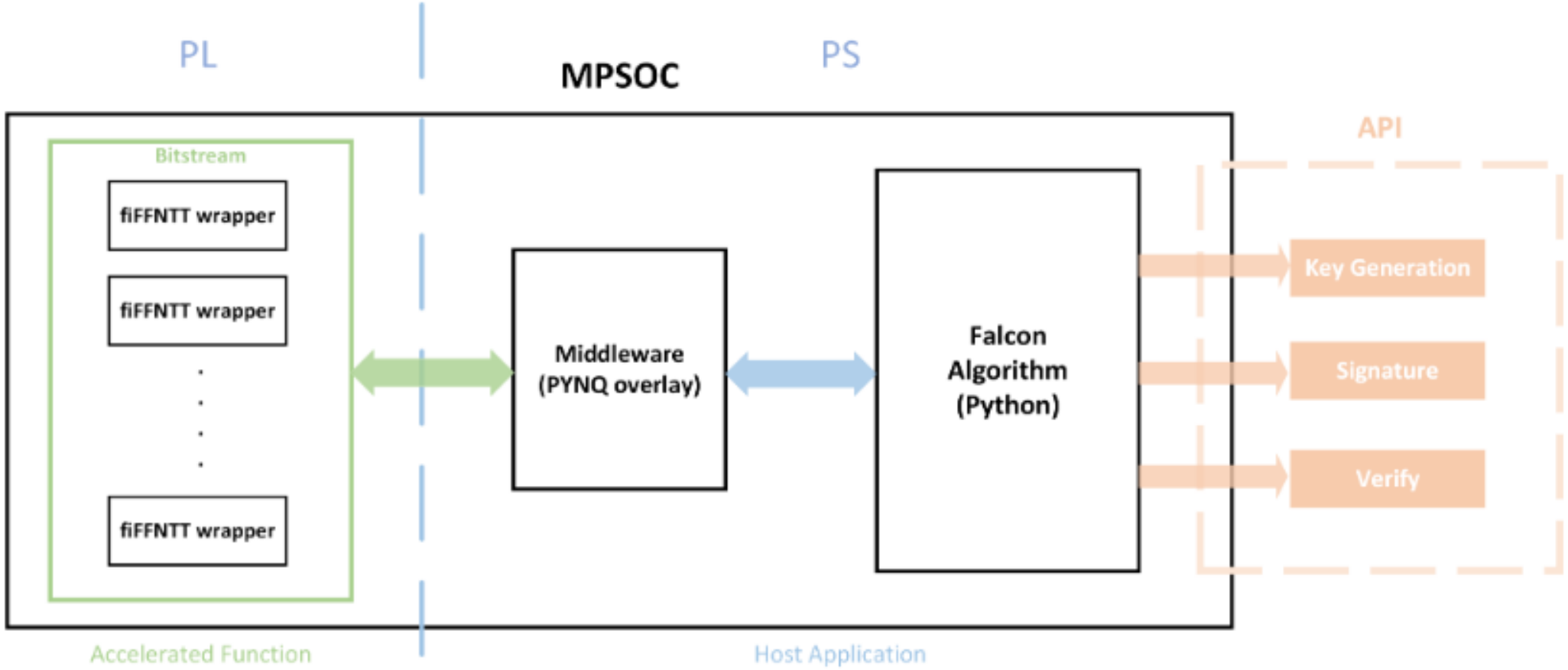
Kernel function-iNTT



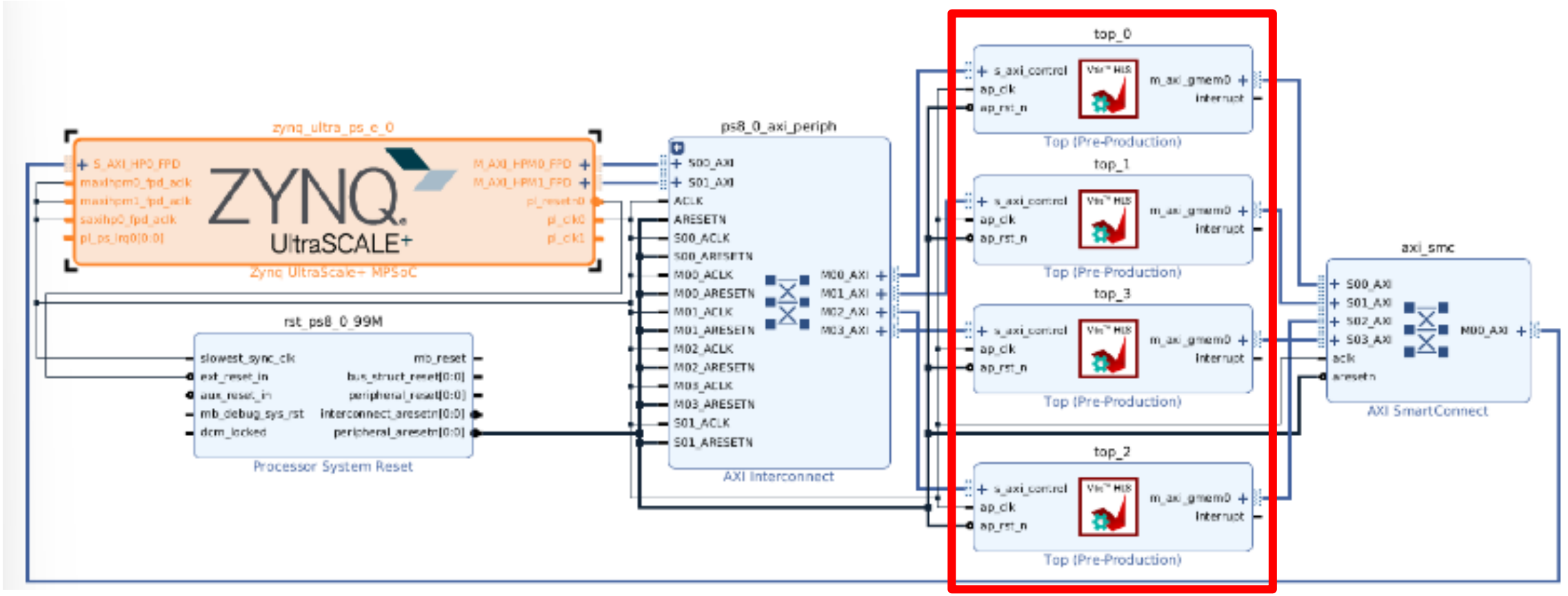
Project Scope

- System block diagram, and its operation flow
- Implement on KV260

A Brief System Block Diagram



Implement on KV260



Our hardware accelerators

Project Plan

- Identify algorithm C-source code - Done
  - self-contained, no library function call
  - Identify test dataset
  - Partition host + kernel
- Run C-sim in Vitis environment Partition - Done
  - run through dataset -> check correctness
- Kernel (team2) and middleware (team3) HLS implementation-2w
  - Turn Vitis HLS into Catapult
- Individual Kernel FPGA validation/integration test -1w
  - Validate with Falcon Algorithm on Host Program/integrate in Caravel FSIC
- Kernel and Host Optimization - 1w

Reference

- List of Papers for reference:

  https://falcon-sign.info/

- Identify open-source to use :
  https://github.com/tprest/falcon.py/tree/master