Adapting and Extending Selected Open-Source Hardware Implementations of Post-Quantum Cryptography Algorithms with Support for the CERG Hardware API

ALEX OLIHOVIK
Outline

• Background
  • How will quantum computers affect cryptography?
  • What are post-quantum algorithms?

• The Problem: How can we compare new post-quantum algorithms?

• The Solution: Proposed hardware API

• Example: The NewHope Algorithm

• Development

• Implementation

• Results
Background

• Current cryptographic standards rely on the inability for modern computers to factor large prime products efficiently

• Quantum computers will factor large prime products efficiently, breaking current cryptographic standards

• Post-quantum (PQ) algorithms rely on problems still hard to solve for both modern and quantum computers

• Some PQ algorithms give little to no advantage to quantum computers

IBM’s Quantum Computer and Refrigeration Unit [1]
The Problem

• There is no current PQ standard for cryptographic functions
• Many different PQ algorithms compete to become the new standard
  • Interfaces vary between algorithms of the same class
  • Makes it hard to quickly implement and analyze for performance comparisons
• Implementations of the same algorithm must be compared fairly
The Solution

• Implement the proposed Hardware API by GMU CERG
• Standardizes input/output specification for classes of algorithms
  • Encryption/Decryption
  • Signature/Verification
  • Encapsulation/Decapsulation
• Design applicable outside of GMU
  • Covers all inputs/outputs for the above classes
  • Enables use with a universal testbench for cryptographic algorithms
• Provides fair benchmarking in hardware for PQC algorithms
The Hardware API (1)

- Similar inputs/outputs to the AXI-Streaming interface
  - “Valid” prefix indicates data is valid
  - “Ready” prefix indicates core is ready to accept new data
- Provides clearly designated ports for public, secret, and random data
- Common external circuits interfacing with core include FIFOs and other AXI Stream-enabled cores
The Hardware API (2)

Typical external circuits, such as AXI-Stream cores and FIFOs interfacing with PQC core [2]
The Hardware API (3)

- Specifies for how to provide input/output headers, system parameters, and keys

Input/Output of a PQC core [2]

Input/Output Data Formats [2]
Example: The NewHope Algorithm

- NewHope [3] is a PQ lattice-based algorithm, computationally hard for both classical and quantum computers to solve.
- Used as a key-exchange mechanism (KEM).
  - Sends data used to generate secret keys between users by first using public-key cryptography.

Poster for A New Hope [4]
Simplified NewHope Circuit Interface
PQC Wrapper Interface

Signals:
- **pdi_data**: 14 bits
- **pdi_valid**: 1 bit
- **pdi_ready**: 1 bit
- **rdi_data**: 8 bits
- **rdi_valid**: 1 bit
- **rdi_ready**: 1 bit
- **clk**: Clock signal
- **rst**: Reset signal
- **pdo_data**: 14 bits
- **pdo_valid**: 1 bit
- **pdo_ready**: 1 bit
- **sdo_data**: 8 bits
- **sdo_valid**: 1 bit
- **sdo_ready**: 1 bit

Components:
- **Register**
- **NewHope**
Example: The NewHope Algorithm (2)

- Reference implementation in VHDL [5]
  - Adapted to meet API interface and data formats

- Modifications for input
  - Designer left RDI input as stored data in core; should be taken out and set as input to wrapper to meet API

- Modifications for hardware reuse
  - Default implementation left client/server as separate entities; requires code changes to enable core to do both

```vhdl
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.numeric_std.all;

entity pqc_wrapper is
  generic(
    rdi: integer := 16; -- Width of public data input/output
    sdi: integer := 16; -- Width of secret data input
    rsi: integer := 16; -- Width of secret data output
    ssi: integer := 16; -- Width of memory address output
    msi: integer := 16; -- Width of memory data input/output
    mwi: integer := 16; -- Width of memory write output
  );
  port(
    clk : in std_logic;
    rdi_data : in std_logic_vector(rdi-1 downto 0);
    rdi_valid : in std_logic;
    rdi_ready : in std_logic;
    sdi_data : in std_logic_vector(sdi-1 downto 0);
    sdi_valid : in std_logic;
    sdi_ready : in std_logic;
    rdi_data : in std_logic_vector(rsi-1 downto 0);
    rsi_valid : in std_logic;
    rsi_ready : in std_logic;
    sdi_data : in std_logic_vector(ssi-1 downto 0);
    ssi_valid : in std_logic;
    ssi_ready : in std_logic;
    status_ready : out std_logic
  );
end pqc_wrapper;
```

PQC Wrapper Entity Declaration
Synthesis

- LUT increase by 35
  - 0.12% increase
- Negligible increase for pure wrapper implementation

Post-Synthesis Resource Utilization before wrapping NewHope core (above) and after (below)
Implementation

- LUT increase by 25
  - 0.12% increase
- Negligible increase for pure wrapper implementation

<table>
<thead>
<tr>
<th>Resource</th>
<th>Utilization</th>
<th>Available</th>
<th>Utilization %</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT</td>
<td>4770</td>
<td>20800</td>
<td>22.93</td>
</tr>
<tr>
<td>LUTRAM</td>
<td>16</td>
<td>9600</td>
<td>0.17</td>
</tr>
<tr>
<td>FF</td>
<td>4720</td>
<td>41600</td>
<td>11.35</td>
</tr>
<tr>
<td>BRAM</td>
<td>2</td>
<td>50</td>
<td>4.00</td>
</tr>
<tr>
<td>DSP</td>
<td>2</td>
<td>90</td>
<td>2.22</td>
</tr>
<tr>
<td>IO</td>
<td>55</td>
<td>106</td>
<td>51.89</td>
</tr>
<tr>
<td>BUFG</td>
<td>1</td>
<td>32</td>
<td>3.13</td>
</tr>
</tbody>
</table>

Post-Implementation Resource Utilization before wrapping NewHope core (above) and after (below)
Timing

- Time constraints met at same max clock frequency as unwrapped core

<table>
<thead>
<tr>
<th>Setup</th>
<th>Hold</th>
<th>Pulse Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Negative Slack (WNS):</td>
<td>0.270 ns</td>
<td>Worst Hold Slack (WHS):</td>
</tr>
<tr>
<td>Total Negative Slack (TNS):</td>
<td>0.000 ns</td>
<td>Total Hold Slack (THS):</td>
</tr>
<tr>
<td>Number of Failing Endpoints:</td>
<td>0</td>
<td>Number of Failing Endpoints:</td>
</tr>
<tr>
<td>Total Number of Endpoints:</td>
<td>11334</td>
<td>Total Number of Endpoints:</td>
</tr>
</tbody>
</table>

All user specified timing constraints are met.

Timing report after wrapping NewHope core
Power Consumption before wrapping NewHope core (left) and after (right)
Conclusions

• PQC Wrapper provides standardization for PQC algorithms
  • Allows PQC benchmarks to be more universal and easily adapted to new algorithms
  • Standardization provides fair benchmarking comparisons
• Negligible increases in resource utilization
• Provides interconnectivity with cores implementing AXI-Stream interface
References


