Project Checklist

Modules

For our minimum commitment we will implement the following modules/components:

- C implementation of functions for algorithm design and verification

We will use the OpenSSL C library to implement each of the modules defined below for CPUs. We will test our C implementation by comparing to the outputs produced by the high-level EC operations provided by OpenSSL to our own implementations. The C implementation will also be used to benchmark our algorithms versus OpenSSL's and how the CPU implementation performance compares to the FPGA translation.

- Big Number ALU

The big number module includes the following math cores, with data flow and execution order controlled by a dispatch unit. The functionality of each submodule can be tested by comparing it to our reference implementation in C, as well as by checking against tests used for well-known cryptography libraries. The dispatch unit is tested in a similar way, by observing how it passes data between submodules and verifying its output. Each submodule has 2 64-byte inputs and outputs either a point or a boolean.

    - Modular Add
    - Modular Subtract
    - Modular Multiply
        - Modular Square
    - Modular Inverse
    - Equality

- Elliptic Curve Point ALU

This module interfaces with the math cores listed above to implement different elliptic curve operations. We can demonstrate its functionality through tests where we compare its output given the same input as our C implementation to check that both agree and are valid. Each function has inputs and outputs of the same size as the big number ALU.

    - Point add
        - Point double
    - Scalar-point multiply

    - Point on curve
    - Point equality

- “Virtual machine” for sequencing multiple EC/BN operations

Since a typical ECDSA signature validation requires 10s of EC operations and 100s of BN operations, a virtual machine-type bytecode language will be required to feed-in data and sequence the operations. We will test the VM by implementing it in C and creating test fixtures that can be compared against the Verilog implementation.

The goal:

- UART communications for I/O

The end goal for our project is to have a host machine pass the FPGA EC operations to perform over UART, and for the FPGA to return the result, effectively offloading these operations to the FPGA. We have seen from previous labs how to test UART send and receive modules, and this goal includes the implementations of a small program on the host to send the appropriate data to the FPGA.

Stretch goal:

- Integration with Bitcoin Core

The ultimate goal here is to enable much faster (and/or low-power) validation of the Bitcoin blockchain. Therefore as a stretch goal we would like to modify Bitcoin Core (the Bitcoin reference implementation) to use our accelerator for signature validation rather than the CPU. This could be tested for correctness by providing blocks with transactions that have invalid signatures to Bitcoin Core and ensuring they are rejected by the accelerator whereas valid signatures are approved and the block/transaction is accepted.