EECE 4632 Project Report:

Outline:

1. Abstract/Introduction
   1. Purpose of this project and using FPGA for implementing algorithms on hardware.
   2. Overview of cyber security and the importance of RSA
2. RSA Overview
   1. Describe the RSA algorithm mathematics to encrypt and decrypt values.
   2. Review how the keys work and how they are generated.
   3. Review how secure this encryption is (good place to add research on cracking RSA).
3. PS Implementation
   1. Introduce how this code was developed on the PS and what worked best on Python.
   2. Start with the original design that used a single integer value to be encrypted.
      1. This solution worked for smaller values input and smaller prime values but ran into issues with larger values.
      2. This was caused by the calculation going into such a large number that it was represented by a float. As a result, taking the modulo of a float would return an inaccurate value.
   3. Next design was to consider using a string, converting the value to a decimal with ASCII and performing the algorithm.
      1. For this iteration to work, we needed to ensure that a key pair was created with a certain number of bits. To do this, the prime numbers were generated randomly with (x) number of bits and checked if prime. If not prime, the value is done again until two prime values are found. As a result, this took a lot of processing and resulted in a major delay of several seconds.
      2. Due to this issue, it was decided that using this method would not be feasible.
   4. The last version of this was one that took a string message and split the characters into an array of their ASCII values. This array then uses the encryption/decryption algorithm from prior to manipulate each of the values in the array.
      1. This was the best version of the code since it works most consistently with different prime numbers.
      2. It would furthermore be useful for testing hardware since we can scale up the input array size to explore speed-up.
4. PL Implementation: adding an AXI Stream feature to interact with hardware code to perform the encryption.
   1. The first version of building the RSA on C++ with Vitis was the RSA\_Simple project.
      1. This code only performed the encryption calculation of the values streamed into the hardware.
      2. Encryption and decryption keys were pre-generated and used in the calculation.
      3. To perform the RSA calculation, the modular exponent calculation was altered to the code (source from geekforgeeks).
      4. This limited the size of data when doing the calculations and prevented overflow from occurring.
      5. Show simple RSA output test bench on vitis and output test of overlay.
   2. Second version of building the RSA: adding key generator
      1. Added a function to generate the private and public key values from primes.
      2. For generating the public key, the key and the product of the totient needed to be prime, so a gcd function was generated. This function checked if the gcd is 1.
      3. For generating the private key, an inverse modular function was generated that uses the extended Euclidian algorithm.
   3. The third version of the RSA: adding prime value and encryption/decryption mode inputs to the overlay.
      1. This was a simple modification to the code that added three axi lite inputs for two integer prime values, and one Boolean value to determine if it will be encrypting or decrypting.
      2. Boolean is used to set the value of exponent for the modulo exponent to the private key or the public key value.
      3. PS Changes to test overlay
         1. Register map to the value to set the prime values and Boolean.
         2. Modifying the output\_buffer to an array and inputting it back.
5. Data verification and validation
   1. Show results of the PS implementation for “Hello, World!” with the primes 2027 and 3011.
      1. Results on Jupyter should show the values to the ASCII converted number array, the encrypted number array, the decrypted number array, and the original msg.
   2. Show results of the PL test bench performing the same encryption on the same array number with the same keys.
   3. Show the results of the PL operating on the overlay and converting the numbers back into the message.
6. Optimizations (WIP)
   1. Show Pareto graph of different optimization attempts.
   2. Compare the difference between the simple RSA and the fully functional one.
   3. Propose limitations of optimizations and capability to surpass software.
7. Conclusion
   1. Reiterate importance of the RSA algorithm.