Project proposal for Cpr E 583

Blake Vermeer, Piriya Kris Hall, and Rohit Zambre

***Abstract*—The abstract goes here.**

***Index Terms*—IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

THE discovery and verification of prime numbers is crucial for security systems.

II. Core Problems

After researching the topic of prime number generation and verification, we have identified three main core problems that need to be solved:

A. Generation of Cunningham Chains

A Cunningham chain is a chain or sequence of prime numbers such that the following property is true:

$$a_i = (2^i)P_{initial} \pm (2^i - 1)$$

Where P is a prime number seed.

B. Generation of Prime Numbers

The problem of generating prime numbers ultimately boils down to determining if a given number has any factors apart from 1 and the number itself. Over the years, multiple factorisation algorithms have been developed to tackle this problem. Alternatively, sieve techniques don't require complex mathematics and are used to find prime numbers within a certain range. Since an immense amount of parallelization can be achieved with a seive-based technique, they are ideal for hardware implementation.

1) The sieve of Eratosthenes: The sieve of Eratosthenes is an ancient and simple algorithm to find all prime numbers up to any given limit. The following describes the algorithm:

1) Create a list of consecutive integers from 2 through n.
2) Let p equal to 2, the first prime number.
3) Starting from p, mark all the multiples of p in the list by counting to n in increments of p.
4) Find the number greater than p in the list that is NOT marked. If there was such a number, stop. Otherwise, let p equal this new number which is the next prime number and repeat from step 3.

2) Related work: Using dynamic reconfigurable logic M. Saleeba and R. Pose from Monash University in Australia implemented a sieve-based algorithm to generate prime numbers using dynamically reconfigurable logic. This project was primarily implemented on the DRAMA research tool to demonstrate the tool's powerful features including runtime reconfigurability. In the algorithm of the implementation, prime numbers are essentially generated by checking each successive integer for divisibility by each prime number less than equal to its square root. Any integer with no primes is added to the list of primes. A re-sizeable logic array is used to append units that check divisibility of the number-in-test by prime factors.

The algorithm consists of two stages: candidate generation and candidate verification. In the candidate generation stage, algorithm counts upwards through the integers and tests the primality of a new integer on each clock cycle. At a given instance, primality is checked against prime factors that have been discovered by the algorithm thus far. With the detection of every new prime factor, a new unit that tests for divisibility by the new prime factor is added to the logic array during run time. The prime factors in the logic array are termed as ”low-value primes.” When a prime number candidate is generated, it enters the candidate verfication stage in which the candidate is tested against groups of ”high-value primes.” The hardware for each of the stages has been described in the paper.

For RSA encryption/decryption C. Torng and Y. Lee from Cornell University implemented a prime nunmber generator and RSA encrypter/decrypter using the Altera DE2 FPGA. The prime number generation was based on the Miller-Rabin primality test to search for prime numbers. They fed in pseudo-randomly generated odd numbers to the test to ”generate” prime numbers. We have described the Miller-Rabin test in the following section.

C. Verification of Generated Prime Numbers

Since the prime number generators work on the principal of generating numbers that has a reasonably high probability of being prime, the numbers must be verified to insure that they are actually prime. This is a very resource intensive problem since it requires testing if the potential prime number number is evenly divisible by a set of numbers. Since it is prohibitively expensive to test if large prime numbers are definitively prime, probabilistic methods have been developed that will determine if a number is prime with a high probability. One of these probability methods of determining if a number is prime is called the Rabin-Miller primality test.

1) Rabin-Miller Primality Test: The Rabin-Miller primality test is a probabilistic method of determining if a number is prime. The algorithm is based around the idea of testing if a number in question is divisible by a subset of small primes. If it is not divisible by any of the small primes in the set then it is a prime number with a probability that is determined by the number of small primes in the set. The larger the set of small prime numbers is allows for the primality of the number in question to be determined with greater confidence. Since the probability of correctly identifying a prime number is directly related to the number of small primes tested again the test number, the accuracy of the Rabin-Miller primality test can be easily customized. [1]

Now to dive into the theory of the Rabin-Miller primality test. The Rabin-Miller theory is based on the contrapositive of Fermat's little theorem which states that for a prime number n:

$$a^{n-1} \equiv 1 \pmod{n}$$

Therefore, the Rabin-Miller primality test can show that a number is not positive of for a number n if we can find a number a such that:

$$a^{d} \not\equiv 1 \pmod{n}$$

and

$$a^{2^{r}d} \not\equiv -1 \pmod{n}$$

for all $0 \leq r \leq s-1$, then n is not prime. [3]

The Rabin-Miller Primality test is a good test to implement on hardware since the test can be parallelized by testing many small primes again the number in question concurrently and the math operations involved would be fairly expense in terms of cycles if implemented in software because of its sequential nature. Since we are attempting to implement this design in hardware let's first examine how to do the modulus operation in hardware.

One way to implement a modulus operation in hardware is to use the Montgomery algorithm to perform a modulo-multiply. These modulo-multiply operations can then be repeatedly run to perform an modular exponentiation. As it turns out there are multiple ways to implement a Montgomery algorithm. The two major approaches are performing the multiplication first and then taking the modulus of the result or the other option is to do the multiplication and module operations in parallel. Since we are trying to implement this as efficiently as possible in hardware we will just focus on the designs that perform the multiplication and module operations in parallel.

III. CONCLUSION

The conclusion goes here.

REFERENCES

[1] Cheung, R., Brown, A., Luk, W., Cheung, P.: A scalable hardware architecture for prime number validation. In: IEEE Int. Conf. on Field-Programmable Technology, pp. 177-184 (2004)

[2] A. Daly and W. Mamane. Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. In *Tenth ACM International Symposium on Field-Programmable Gate Arrays* pages 40-49, February 2002.

[3] Miller-Rabin Primality Test. On: *Wikipedia*, 27 Oct. 2014. <http://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test>

[4]

[5]

[6]

[7]

[8]

[9]

[10]