A SCALABLE SYSTEM-ON-A-CHIP ARCHITECTURE FOR PRIME NUMBER VALIDATION

Ray C.C. Cheung and Ashley Brown
Department of Computing, Imperial College London, United Kingdom

Abstract

This paper presents a scalable SoC architecture for prime number validation which targets reconfigurable hardware. The primality test is crucial for security systems, especially for most public-key schemes. The Rabin-Miller Strong Pseudoprime Test has been mapped into hardware, which makes use of a circuit for computing Montgomery modular exponentiation to further speed up the validation and to reduce the hardware cost. A design generator has been developed to generate a variety of scalable and non-scalable Montgomery multipliers based on user-defined parameters. The performance and resource usage of our designs, implemented in Xilinx reconfigurable devices, have been explored using the embedded PowerPC processor and the soft MicroBlaze processor. Our work demonstrates the flexibility and trade-offs in using reconfigurable platform for developing System-on-a-Chip applications. It is shown that, for instance, a 1024-bit Primality test can be completed in less than a second, and a low cost XC3S2000 FPGA chip can accommodate a 32k-bit scalable primality test with 64 parallel processing elements.

1 Introduction

For centuries, the problem of validating prime numbers, the Primality test, has posed a great challenge to both computer scientists and mathematicians [6]. The problem of identifying Prime and Composite numbers is known as one of the most important problems in arithmetic. Many security applications also involve large numbers: while it is easy to multiply prime numbers to get a product, the reverse process of recovering the primes is much more difficult.

Field programmable gate arrays offer a rapid-prototyping platform for the verification of embedded systems.FPGAs are also used as active components in security systems, such as Firewall processors [12]. As the threat of attacks appears to be increasing, many existing systems do not seem to be secure enough. One of the reasons is due to the use of weak key generation. It has been recognised that strong prime number generation is important, and the prime validation is an intrinsic part of the generation. Here we define validation as the process of performing prime testing on a given number.

This paper presents a scalable SoC architecture for prime number validation which targets reconfigurable hardware such as FPGAs. In particular, users are allowed to select predefined scalable or non-scalable modular operators for their designs [4]. Our main contributions include: (1) Parallel designs for Montgomery modular arithmetic operations (Section 3). (2) A scalable design method for mapping the Rabin-Miller Strong Pseudoprime Test into hardware (Section 4). (3) An architecture of RAM-based Radix-2 Scalable Montgomery multiplier (Section 4). (4) A design generator for producing hardware prime number validators based on user-specified parameters (Section 5). (5) Implementation of the proposed hardware architectures in FPGAs, with an evaluation of its effectiveness compared with different size and speed tradeoffs (Section 6). (6) Evaluation of a SoC approach using customised IP blocks with both hard [5] and soft processor (Section 7).

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 presents a design flow for mapping scalable prime number validators into hardware. Section 4 covers a scalable architecture for modular multiplication. Section 5 presents a design generator $GenDesign$ which produces architectures with different design trade-offs, based on user-specified parameters. Section 6 provides experimental results of one single IP-block. Section 7 describes the SoC-based framework using embedded processors. Finally, Section 8 summarises our current and future research.

2 Background and Related work

Primality test is essential for prime number generation. Cryptography often uses large prime numbers. For instance, RSA public key algorithm requires 512-bit, 1,024-bit for key generation [17]. Much research work has been done on primality test and generation, and most algorithms are based on factorization [16]. In [2], the primality test has been proved to be solvable in polynomial time. However, it requires a log operation which is expensive for hardware implementation; it also requires knowledge of the primality of preceding numbers, which is impractical for arbitrary prime testing. Joye et al [10] has proposed an efficient prime number generation scheme based on pseudo-random number generation using a smart-card implementation. Lu et al [13] has further developed the RSA key generation for smart card using different prime number algorithms.

```
Input: The number under primality test: p

1. Random choose a number b in [1, p-1] ∈ Z
2. Let p = 2^(q×m)+1 where m is an odd integer
3. IF either
4.    Case 1: b^m = 1 (mod p) or
5.    Case 2: there is an integer i in [0, q-1]
6.       such that b^(m×2^i) = -1 (mod p)
7.    RETURN "Inconclusive"
8. ELSE
9.    RETURN "Composite"
```

Figure 1: The Rabin-Miller Probabilistic Primality Test

There is a special class of very large prime numbers which have the representation of $2^n - 1$. They are called Mersenne prime numbers. The 40^{th} Mersenne prime number $2^{24,036,583} - 1$ has been discovered in May 2004. A clustered workforce GIMPS [9] in the Internet is currently dedicated on locating the next Mersenne number.

2.1 Rabin-Miller Primality Test

In this paper, we have selected the Rabin-Miller probabilistic primality test algorithm [15] (Figure 1) as the core for the primality test. The algorithm can quickly determine the primality of the given large number with a controllably small probability of error [1]. It requires a number of small primes for repeatedly testing the input number. The probability of falsely identifying a composite as a prime decreases with every additional small prime used. This method enables the tradeoff between accuracy (more small primes) and efficiency (fewer small primes) for different applications. As shown in the algorithm, "`inconclusive`" implies the number p maybe prime. The selection of number b is based on a set of small primes such as 2, 3, 5, ... With the use of first eight small primes, the 100% accuracy of primality test can be achieved for numbers up to 3.4×10^{10} [1].

2.2 Scalable Montgomery Algorithm

The fundamental operation of the RSA public key cryptosystem is modular exponentiation, achieved by repeated modular multiplications. The Montgomery modular multiplication [14] has been widely used to speed up the multiplication, squaring and exponentiation. There are many different extended algorithms [11] and software implementation based on the Montgomery algorithm, and the high-radix Montgomery modular exponentiation has been implemented in reconfigurable hardware [3].

The reusability and scalability of the Montgomery multiplier have been investigated in [18, 19] such that the design is no longer restricted to a fixed precision. The input operands are partitioned into multiple words. Figure 2 shows the algorithmic description of the Radix-2 Montgomery multiplication. The operand Y (multiplicand) is scanned word-by-word and the operand X (multiplier) is scanned bitwise. The loop index n is the bit width of the prime number and e is the number of partitions in the operand Y. The scalable design of Montgomery multiplication has been implemented as a coprocessor in reconfigurable RSA System-on-Chip building block in [8]. Furthermore, a high-radix design of scalable modular multiplier has also been discussed. We observe that the scalable design is particularly useful for very large precision such as the prime test.

```
Input: X * Y (mod M), Output: S
1.  S = 0
2.  for i = 0 to n - 1
3.     (Ca,S^(0)) = x_i*Y^(0) + S^(0)
4.     if S_0^(0) = 1 then
5.        (Cb,S^(0)) = S^(0) + S^(0)
6.        for j = 1 to e
7.           (Ca,S^(j)) = C_a + x_i*Y^(j) + S^(j)
8.           (Cb,S^(j)) = C_b + M^(j) + S^(j)
9.           S^(j-1) = (S_0^(j),S_(w-1..1)^(j-1))
10.       end for
11.    else
12.       for j = 1 to e
13.          (Ca,S^(j)) = C_a + x_i*Y^(j) + S^(j)
14.          S^(j-1) = (S_0^(j),S_(w-1..1)^(j-1))
15.       end for
16.    end if
17.    s^(e) = 0
18. end for
```

Figure 2: The Multiple Word Radix-2 Montgomery Multiplication Algorithm

3 Design Flow

In this section we present the design flow for our scalable prime number validator. The input to the system includes user-specified constraints, such as the bit-width of the input prime number and the word-width which is required for the scalable multiplier, while the output from the system is a synthesizeable hardware block that can be embedded in different cryptographic designs. This flexible design flow facilitates the ability to upgrade existing security systems with very small overhead.

The major features of the proposed validator include: (1) A variable compile-time prime number validator in which user can easily update the complexity of the system by controlling the prime width. (2) The variable input small prime numbers determine the accuracy and performance of the system. (3) The user-specified word-width controls the modular operation taken in the hardware. (4) Users are able to select different Montgomery architectures for hardware implementation and thus enhance rapid prototyping.

The Rabin-Miller algorithm is first implemented using standard multiplication and exponentiation with a sequential modulo-multiply. We observe that one of the bottlenecks of the Rabin-Miller algorithm is the modular exponentiation in testing the two valid checks in Section 2. Montgomery algorithm is widely used for modular multiplication and exponentiation. It requires a constant at the start of algorithm to facilitate conversion between standard and Montgomery space. The constant c is $(2^{(2\times(width(p)+2))} \% p)$ where $width(p)$ is the bit-width of the prime number p under test. In this paper, this constant c has been precomputed and saved in hardware. Note that

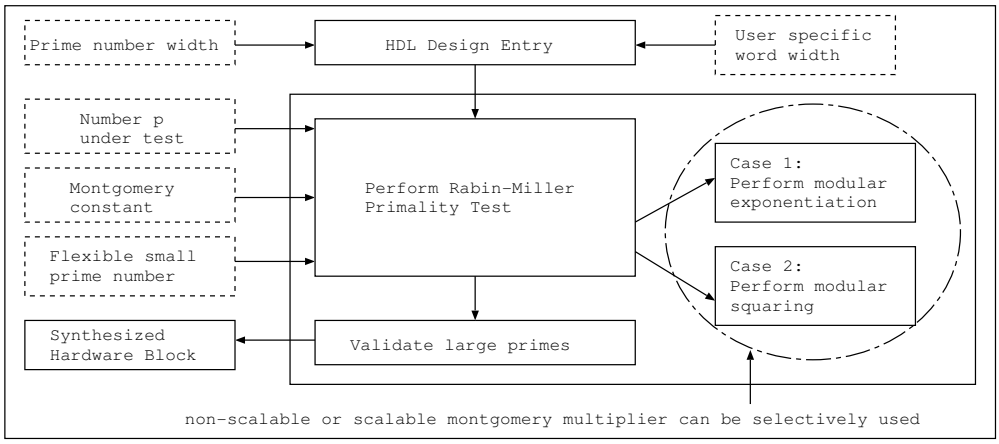

Figure 3: The design flow of the scalable prime number validator

```
Algorithm for computing S = A × B mod M

    parallel {
1.      S = 0
2.      j = width(A) + 2
    }
3.  FOR i = j to 0
        parallel {
4.        q = S + (B0? A : 0)
5.        S = shiftRight(q + (q0? M : 0))
6.        B = shiftRight(B)
        }
7.  END FOR
8.  RETURN S
```

Figure 4: Generating Montgomery modular multiplication S

the statements embraced by $parallel\{...\}$ mean that they are executed concurrently in hardware.

3.1 Montgomery modular multiplication

Current modular multiplication approaches are mostly based on the Montgomery algorithm [14]. The simpler combinational logic used in this design has shortened the critical path and thus accelerates the calculation. An efficient hardware implementation has been presented in [7]. The basic idea of Montgomery's algorithm is to multiply two integers modulo M in other words, $(A \times B \bmod M)$ without division by M. We first use the generated Montgomery constant c to transform the integers in m-residues and compute the multiplication with these m-residues. Finally, we transform this result back to the normal representation. Note that modular multiplication is used in modular exponentiation, since it is beneficial if we compute a series of multiplications in the transformed domain, the Montgomery space. Figure 4 shows the pseudo code of the hardware description. For example, "$q = S + (B_0 \ ?\ A : 0)$" checks if the LSB of B is true, then "$q = S + A$" else "$q = S$".

3.2 Montgomery modular exponentiation

Figure 5 shows the algorithm for calculating the modular exponentiation using the Montgomery algorithm. This algorithm is not limited to the input bit-width and is suitable for replacing the standard sequential modulo-multiplier. The X and 1 value is first transformed into Montgomery space by using the Montgomery constant c. Since there is no data dependency between the modular squaring and multiplying operation in line 6 and line 7, both operations are put into separated hardware and execute in parallel. The final result is transformed back to the standard domain for the Rabin-Miller primality test.

```
Algorithm for computing S = X^E mod M

1. P[2] = 0, Z[2] = 0;
   parallel {
    // Apply two parallel multipliers
2.  P[0] = P[1] = MontgomeryModularMulti[0](c, 1)
3.  Z[0] = Z[1] = MontgomeryModularMulti[1](c, X)
4.  j = width(E) - 1
   }
5. FOR i = j to 0
     parallel {
6.    P[!i0] = E0?
          MontgomeryModularMulti[0](P[i0],Z[i0]):P[i0]
7.    Z[!i0] = MontgomeryModularMulti[1](Z[i0], Z[i0])
     }
8.   E = shiftRight(E);
9. END FOR
10.RETURN S = MontgomeryModularMulti[0](1, P[i0])
```

Figure 5: Generating Montgomery modular exponentiation S (both scalable and non-scalable)

The datapath of the modular exponentiation unit is depicted in figure 6. Two parallel multipliers have been deployed in this unit together with four temporary storages, P_0, P_1, Z_0 and Z_1. The inputs to this unit are X and E which are stored in registers or memory depending on the architecture of the multiplier. The pre-stored Montgomery constant c is used for the first step calculation in figure 5. The control path, other temporary storage and memory decoding unit are not shown in this figure.

4 Scalable modular multiplier

The previous section presents the general design flow of primality test using non-scalable multiplier. In this section,

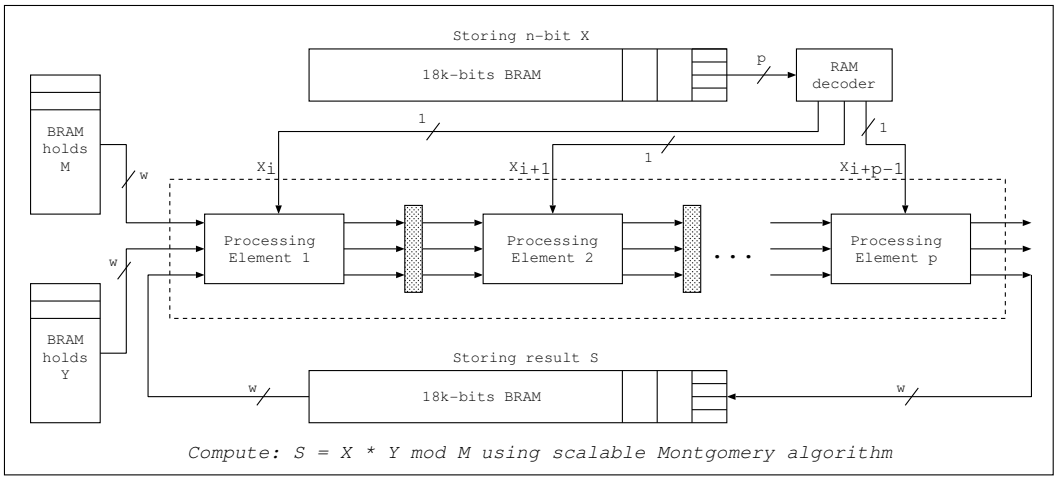

Figure 7: The architecture of the scalable Montgomery multiplier

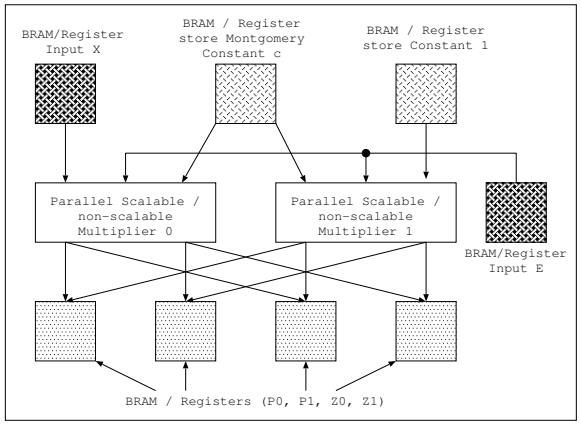

Figure 6: The architecture of the Exponentiation unit using two parallel Montgomery multipliers and multiple Block-RAMs

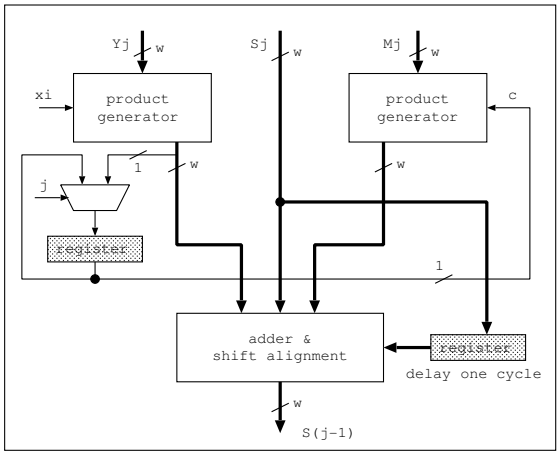

Figure 8: The architecture of a single Processing Element

we present the mapping of scalable Montgomery multiplication into technology independent hardware. The technology dependent such as FPGA and the optimisation techniques are presented in the next subsection.

The scalable Montgomery algorithm MWR2MM is shown in figure 2 for multiplying X and Y. The general idea is to repeatedly multiply X_i, the i^{th} bit of X, with Y. The parallelism of the design can be explored by applying multiple processing elements (PEs) as the data dependency can be resolved between different X_i values, i.e. each single bit of X's calculation. The datapath of the multiplier using p PEs is shown in figure 7. First, the RAM decoder produces the p-bits, $X_i, X_{i+1}, \ldots X_{i+p-1}$ and feeds these bits into p PEs. These signals are valid for j cycles and the memory decoder then extracts the next p-bits from the memory storing X.

For the scalable version, the modular multiplier is replaced by the one described in figure 2. Note that the number of clock cycle and the performance of the scalable MWR2MM algorithm depends on the number of bit of the number under test and the user-specified partition size which is the word-length. The words extracted from Y, M are serially put into the first PE and pipeline to the next and other PE elements.

In the cycle when $j = 0$, the PE element determines the addition of M for the next j cycles within this PE element. We can refer to line 4-10, and line 12-15 in figure 2 more details. The exact datapath of a single PE is described in figure 8. We use a multiplexor to select and store $S_0^{(0)}$, the bit-0 of the lowest word in S, in a register only when $j = 0$. The input S_j is latched for computing the S_{j-1} in the next cycle as shown in line 9 and 14 in figure 2.

Memory exploration in FPGA

We retarget the scalable Montgomery multiplier in FPGA and explore the available resources such as embedded memory. For instance, if we use shift register for n-bit data, the hardware usage is linearly proportional to the input bit-width. In order to save area, our design stores the n-bit data into Block-RAMs (BRAM) which holds up to 18k-bit data. Together with all the temporary storage, each design takes 20 BRAMs which is half of the available BRAMs in XC2V1000 FPGA chip. In figure 7, each data bit in BRAM-X is decoded for a PE while in each cycle, the j^{th} data in BRAM-Y and BRAM-M are stored in the first PE and propagated in the pipeline registers. The outputs of $PE_1, \ldots PE_{p-1}$ are stored in the intermediate registers, so that in each cycle only the last PE, PE_p, is responsible to update the content in BRAM-S.

5 Hardware Design Generator

This section presents the $GenDesign$ tool which we develop for transforming user-constraints into Hardware Description Language (HDL) format, as well as implementing hardware designs. $GenDesign$ is coded in C. This generator enhances designer productivity and exploits available resources from the rapid-prototyping platform. The objective of this design generator is to accelerate the overall design flow from specification to final implementation. Using this tool, designers can specify architectural-specific description of their designs. The predefined libraries consist of non-scalable Montgomery operators, scalable version and the Rabin-Miller algorithm. The existing libraries are developed in Handel-C, and with slightly modification, the design generator is able to support other HDL output format for synthesis.

$GenDesign$ enables customisable designs such that users can have fully control to the output HDL file. Two major operations in the design generator are the architectural selection routine and RAM decoding routine. In our design, the width of most internal signals are dependent on the prime number bit width. Second, user-specified word width controls the iteration of the inner while loop in the scalable Montgomery multiplier (the value e in figure 2). Third, besides the predefined small prime numbers, users are able to add or remove small prime numbers of the Rabin-Miller algorithm in the final HDL output. $GenDesign$ is then used to expand the design into p parallel PEs.

6 Single IP-block results

Our single prime testing IP block can be implemented on any FPGA-based prototyping platform. For small size problems, we select the RC200E development board as the testing platform which contains the Xilinx XC2V1000-4FG456C FPGA chip. For large size problems, the RC2000 development board which has Virtex II XC2V6000-4FF1152 FPGA has been selected for implementation. We also place and route our designs on a low cost Spartan XC3S2000-4-FG900 chip. The generated designs are synthesized by using Celoxica DK2 and PDK tools. This design facilitates parametrised input bit-widths.

Table 1 shows the performance and area comparisons between non-scalable and scalable Montgomery multiplication. From the collected results, the non-scalable designs produce the fastest execution time with a linearly increase of area penalty, while the scalable designs achieve relatively small and stable increase in both resource usage and critical delay path as the design scale grows. In the same table, we observe that by adopting multiple PE elements, the total number of cycle taken for the primality test has been much reduced while there is a slight effect to the delay path. Our experimental results also confirm the speed improvement by using more parallel processing elements. Note that for smaller problems such as the 128-bit design, the control logic of the scalable design dominates the major resource and causes the larger area than the non-scalable design. The scalable design is best suitable for area-limited embedded applications.

7 System-on-a-Chip Framework

We also investigate a general design framework for prime number validation that makes use of an embedded microprocessor, a fast On-Chip Peripheral Bus (OPB) or Processor Local Bus (PLB) and programmable user-logic in reconfigurable hardware. In this framework, a divide and conquer technique is applied to prime number generation. The circuit in previous sections is used to validate one long prime number. The generated designs are first synthesized into VHDL and connected to the PLB/OPB bus through the predefined bus interface as programmable slave bus modules. An embedded microprocessor such as the PowerPC or the soft-processor MicroBlaze is used to generate high quality random numbers and to interface between user and on-chip validators. The OPB/PLB bus interface provides a high performance interface between the microprocessor and the reconfigurable logic. We have chosen Xilinx ML300 as the prototyping platform which contains a Virtex-II Pro FPGA. Our result shows that the design is highly scalable and can accommodate up to 8 slave modules using non-scalable multipliers for 256-bit prime generation in an XC2VP125 device operating at 20MHz. By using the same device, we can have 16 slave modules using scalable multipliers with 8 parallel processing elements at 33MHz. In this SoC system, the microprocessor can be either PowerPC which operates at 300MHz or MicroBlaze which operates at 150MHz. The major tradeoff is that MicroBlaze consumes around 450 slices and enables multiple on-chip processors, while the largest Virtex-II Pro chip comes with 4 embedded hard-processors.

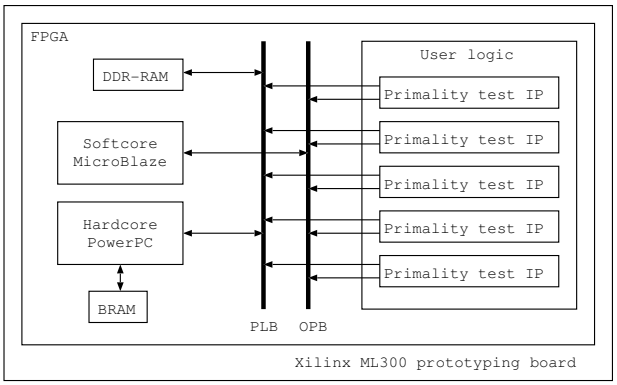

Figure 9: A system overview of prime number validators in a single FPGA

In a traditional RISC system, the clock speed of the microprocessor system is limited by the customised instruction block; the longer the critical path of this block, the slower the processor speed. MicroBlaze provides a Fast Simplex Link (FSL) channel for including customisable IP blocks such as our prime test block. In this case, the microprocessor and the IP block operate at two different clock speeds while there is no degradation to the MicroBlaze speed. The FSL channel enables the data to be passed between hardware block and processor using C-macros, with the processor able to execute other instruction while waiting for the result. The proposed customisable MicroBlaze processor now has the primality test ability which is very flexible and can be used in various SoC and embedded security system.

	Virtex II - XC2V1000 / XC2V6000				*Spartan - XC3S2000*			
Prime size	128-bit	256-bit	512-bit	1,024-bit	128-bit	256-bit	512-bit	1,024-bit
	Non-scalable design							
Area (slices)	2,630	5,140	10,153	20,131	2,630	5,140	10,153	20,131
Clock cycle (ns)	36.75	49.51	80.72	122.34	43.67	62.35	95.01	162.24
Performance (ms)	0.64	3.34	21.49	129.28	0.76	4.21	25.29	171.45
	Scalable design (8 PE, word-size:w = 8-bit)							
Area (slices)	2,651	2,726	2,768	2,872	2,622	2,700	2,736	2,842
Clock cycle (ns)	31.66	30.78	31.07	32.62	40.38	34.72	39.68	37.75
Performance (ms)	20.69	109.37	734.39	5478.14	26.39	123.36	937.82	6338.95
	Scalable design (32 PE, word-size:w = 8-bit)							
Area (slices)	9,064	9,144	9,192	9,333	9,019	9,092	9,146	9,283
Clock cycle (ns)	30.17	29.81	29.47	31.03	40.87	40.61	38.94	39.03
Performance (ms)	12.70	56.07	286.18	1776.80	17.20	76.38	378.11	2235.08

Table 1: Primality test comparison

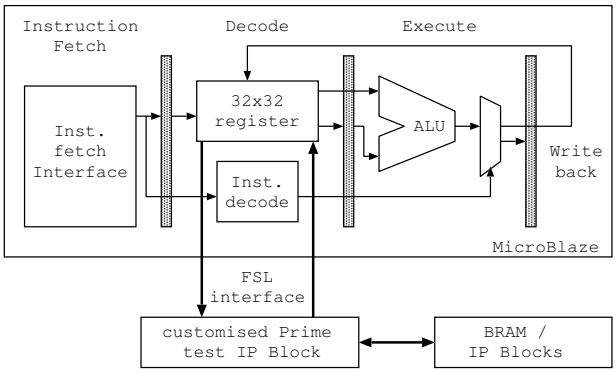

Figure 10: Connecting customised IP block to MicroBlaze

8 Conclusions

This paper presents a scalable SoC architecture for prime number validation which targets reconfigurable hardware. Two significant algorithms have been discussed: Rabin-Miller and Montgomery algorithms. The Rabin-Miller Strong Pseudoprime Test has been successfully mapped into hardware. Several modular multipliers have been developed as libraries in the design generator, $GenDesign$. In particular, the Montgomery modular multiplication is the core of other modular operations used in the primality test. The scalability and parallelism of this design have been explored for very large prime numbers. We systematically implement the design for different bit lengths on reconfigurable devices using both scalable and non-scalable multipliers, and study the tradeoff amongst different designs. Our architectures and tools appear to be useful for exploring efficient designs for use in SoC and embedded systems. We evaluate our designs as a replicable IP block for the use in a SoC framework with both hard and soft processors, and describe the integration with MicroBlaze using the FSL channel. On-going and future developments include the extension of this scalable architecture across multiple reconfigurable hardware device and applying Run-Time Reconfiguration (RTR) techniques for different primality test algorithms.

Acknowledgment

The support of Celoxica, Xilinx, the Croucher Foundation and UK EPSRC (Grant number GR/R 31409, GR/R 55931, GR/N 66599) is gratefully acknowledged.

References

[1] F. Arnault, *Rabin-Miller Primality Test: Composite Numbers Which Pass It*, Math. Comput. 64, 355-361, 1995.

[2] M. Agrawal et al, *Primes in P*, ITT Kanpur, August, 2002.

[3] T. Blum and C. Paar, *High Radix Montgomery Modular Exponentiation on Reconfigurable Hardware*, IEEE Transactions on Computers, vol. 50, no. 7, pp.759-764, 2001.

[4] R. Cheung, A. Brown, W. Luk, and P. Cheung, *A Scalable Hardware Architecture for Prime Number Validation*, to appear IEEE Conference on Field Programmable Technology, 2004.

[5] R. Cheung, *A System on Chip Design Framework for Prime Number Validation using Reconfigurable Hardware*, to appear IEEE Conference on FPL (PhD Forum), 2004.

[6] R. Crandall and C. Promerance, *Prime Numbers - A Computational Perspective*, Springer, 2001.

[7] S. E. Eldridge and C. D. Walter, *Hardware Implementation of Montgomery's Modular Multiplication Algorithm*, IEEE Transactions on Computers, vol. 42, no. 7, pp. 693-699, July, 1993.

[8] V. Fischer and M. Drutarovsky, *Scalable RSA Processor in Reconfigurable Hardware - a SoC Building Block*, in Proceedings of XVI Conference on Design of Circuits and Integrated Systems (DCIS), pp. 327-332, November, 2001.

[9] *Internet Mersenne Prime Search*: "http://www.mersenne.org/"

[10] M. Joye, P. Paillier and S. Vaudenay, *Efficient Generation of Prime Numbers*, Cryptographic Hardware and Embedded Systems CHES, pp. 340-354, 2000.

[11] C.K. Koc, T. Acar and B. S. Kaliski, *Analyzing and Comparing Montgomery Multiplication Algorithms*, IEEE Micro, vol. 16, no. 3, pp. 26-33, 1996.

[12] T. K. Lee et al., *Compiling policy descriptions into reconfigurable firewall processors*, IEEE Symposium on FCCM, pp. 39-48, 2003.

[13] C. Lu, A. L. M. dos Santos and F. R. Pimentel, *Implementation of fast RSA key generation on smart cards*, in Proceedings of the ACM symposium on Applied computing, pp. 214-220, 2002.

[14] P. Montgomery, *Modular Multiplication without Trial Division*, Math. of Computation, vol. 44, pp. 519-521, 1985.

[15] M. O. Rabin, *Probabilistic Algorithm for Primality Testing*, Journal of Number Theory, Vol. 12, pp. 128-138, 1980.

[16] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Progress in Mathematics, vol. 126, Birkhauser, 1994.

[17] B. Schneier, *Applied Cryptography*, John Wiley and Sons, 1996.

[18] A. F. Tenca and C. K. Koc, *A Scalable Architecture for Montgomery Multiplication*, CHES, pp. 94-108, 1999.

[19] A. F. Tenca and C. K. Koc, *A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm*, IEEE Transactions on Computers, Vol 52, No. 9, pp. 1215-1221, 2003.