CAIRN 2: An FPGA Implementation of the Sieving Step in the Number Field Sieve Method*

Tetsuya Izu, Jun Kogure, and Takeshi Shimoyama

FUJITSU Limited
4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan
{izu,kogure,shimo-shimo}@jp.fujitsu.com

Abstract. The hardness of the integer factorization problem assures the security of some public-key cryptosystems including RSA, and the number field sieve method (NFS), the most efficient algorithm for factoring large integers currently, is a threat for such cryptosystems. Recently, dedicated factoring devices attract much attention since it might reduce the computing cost of the number field sieve method. In this paper, we report implementational and experimental results of a dedicated sieving device "CAIRN 2" with Xilinx's FPGA which is designed to handle up to 768-bit integers. Used algorithm is based on the line sieving, however, in order to optimize the efficiency, we adapted a new implementational method (the pipelined sieving). In addition, we actually factored a 423-bit integer in about 30 days with the developed device CAIRN 2 for the sieving step and usual PCs for other steps. As far as the authors know, this is the first FPGA implementation and experiment of the sieving step in NFS.

Keywords: Integer factorization, the number field sieve method (NFS), the sieving step, implementation, FPGA

1 Introduction

The integer factoring problem is one of the most fundamental problem in cryptology since the hardness of the problem assures the security of some public-key cryptosystems including RSA. Currently, the number field sieve method (NFS) [LLMP90] is the most efficient algorithm for factoring large composite integers. In fact, in 2005, Franke et al. established a world record by factoring a 663-bit integer (known as RSA200) by NFS implemented on large amount of PCs. Since the complexity of NFS grows subexponentially with regard to the size of input integer, it is widely and strongly believed that factoring 1024-bit integers (which RSA commonly uses in practice) is infeasible over the next several years by the same approach, namely, software implementation on PCs. Thus, it is very natural to consider a special-purpose device dedicated to integer factorization which might reduce the computing cost of NFS.

* A part of this research is financially supported by a contract research with the National Institute of Information and Communications Technology (NICT), Japan.

P. Paillier and I. Verbauwhede (Eds.): CHES 2007, LNCS 4727, pp. 364–377, 2007.

Among four major steps of NFS (namely, the polynomial selection step, the sieving or the relation finding step, the linear algebra step, and the square root step), the sieving and the linear algebra steps are dominant procedures theoretically and experimentally. Thus, these steps are main targets for the dedicated devices. In 2001, Bernstein proposed the hardware design for the linear algebra step based on a sorting algorithm with standard ASIC architectures [Ber01]. Then, Lenstra et al. enhanced the device by using a routing algorithm [LSTT02]. Geiselmann and Steinwandt applied these ideas to the sieving step and proposed two designs DSH and YASD [GS03, GS04]. Shamir and Tromer improved an optical sieving device TWINKLE [Sha99] into a novel ASIC-based hardware TWIRL [ST03]. Since the efficiency of TWIRL was not optimized, an improvement was proposed by Geiselmann et al. [GJK+06], and a combination of TWIRL and YASD was discussed by Geiselmann and Steinwandt [GS07]. On the other hand, Franke et al. proposed a sophisticated design SHARK by using a butterfly-sorting [FKP+05]. In order to accelerate the sieving step, FPGA implementations of the mini-factoring were discussed in [FKP+05, SPK+05, GKB+06]. In spite of these theoretical efforts, no implementational results of the whole sieving part on ASIC or FPGA have been known up to the present.[1] One of the reason may be that designing and manufacturing such dedicated devices require a large amount of money and time.

In this paper, we report implementational and experimental results of a dedicated sieving device "CAIRN 2" (Circuit Aided Intellegent Relation Navigator) which is designed to handle up to 768-bit integers. The developed device processes the core sieving based on the line sieving on FPGA (Xilinx's Virtex-4 XC4VLX200) and the primality test based on the Fermat and the Euler methods and the mini-factoring based on the ρ-method on a reconfigurable processor (IPFlex's DAPDNA-2). In order to optimize the efficiency, we develop a new implementational method (the pipelined sieving) for the core sieving. As an experiment of our device, we actually factored a 423-bit integer from the Cunningham project [Cun] (which was unfactored when the experiment was done) with the developed device for the sieving step and usual PCs for other steps. In the experiment, about 30 days are required for the sieving step, which is as fast as what our software on a PC based on the lattice sieving requires.

CAIRN 2 was developed in the CAIRN project financially supported by the National Institute of Information and Communications Technology (NICT), Japan, which lasted for 3 years. The goal of the project was to implement the sieving step on the dedicated device and to experiment a factorization on the device every year. In the first year, we implemented a naive sieving on the reconfigurable processor DAPDNA-2 (CAIRN 1) [IKS05, IKKN+06]. CAIRN 2 is a result of the project in the second year. Because of the time limitation, we used the line sieving rather than the lattice sieving which is potentially more efficient than the line sieving.

The rest of this paper is organized as follows: section 2 briefly introduces the number field sieve method, especially the sieving step by the line sieving.

[1] Kim et al. developed an FPGA-based siever for the quadratic sieve method [KM00], however, only simulational results were reported.

Alg. 1. Outline of the line sieving

```
for $b \leftarrow 1$ to $H_b$
  for $\bar{a} \leftarrow -H_a$ to $H_a - S$ step $S$
    for $j \leftarrow 0$ to $S - 1$
      set $R[j]$ to $\log_2 \lceil F(\bar{a} + j, b) \rceil$
    for prime $p \leftarrow 2$ to $B$
      compute the first sieving point $\bar{a} + r$
      while $r < S$
        $R[r] \leftarrow R[r] - \lceil \log_2 p \rceil$
        $r \leftarrow r + p$
```

Detailed descriptions of the developed device are figured in section 3. Finally, experimental results of factoring a 423-bit integer is shown in section 4.

2 Number Field Sieve Method

The number field sieve method (NFS) is known as the most efficient algorithm for factoring large integers currently [LLMP90]. NFS consists of 4 steps; the polynomial selection step, the sieving (or the relation finding) step, the linear algebra step, and the square root step. Among these steps, the sieving and the linear algebra steps are dominant theoretically and experimentally.

Let N be an integer to be factored by NFS. First of all, in the polynomial selection step, two univariate polynomials $f_{\mathrm{r}}(x),\ f_{\mathrm{a}}(x)$ and an integer m such that $f_{\mathrm{r}}(m) \equiv f_{\mathrm{a}}(m) \equiv 0 \pmod{N}$ are generated. Then these polynomials are converted to bivariate and homogeneous polynomials $F_{\mathrm{r}}(x, y),\ F_{\mathrm{a}}(x, y) \in \mathbb{Z}[x, y]$. Then, the sieving step finds a large number of *relations*, namely a set of integer pairs $\{(a, b)\}$ satisfying

- $\gcd(a, b) = 1,\ -H_a \le a \le H_a,\ 1 \le b \le H_b$,
- $F_{\mathrm{r}}(a, b)$ is B_{r}-smooth (namely, $F_{\mathrm{r}}(a, b) = \prod_{p_i \le B_{\mathrm{r}}} p_i^{e_i}$)[2],
- $F_{\mathrm{a}}(a, b)$ is B_{a}-smooth (namely, $F_{\mathrm{a}}(a, b) = \prod_{p_i \le B_{\mathrm{a}}} p_i^{e_i}$)[2].

Procedures corresponding to F_{r} (F_{a}) are sometimes called 'rational' ('algebraic'), respectively. Parameters $H_a,\ H_b$ determine the sieving region. In practice, the core sieving step picks up possible relations (called *candidates*), and an additional step checks whether it really is a relation via the primality test and the mini-factoring. After finding a set of relations, the Gaussian elimination over a matrix generated from obtained relations is computed in the linear algebra step. Then, a non-trivial factor of N is output by computing rational and algebraic square roots in the square root step. Further descriptions of NFS is found in [LL93].

[2] In order to collect relations as much as possible, we sometimes relax this condition in practice: $F_{\mathrm{r}}(a, b) = q \prod_{p_i < B_{\mathrm{r}}} p_i^{e_i}$ s.t. $B_{\mathrm{r}} < q \le B'_{\mathrm{r}}$ and $F_{\mathrm{r}}(a, b) = q \prod_{p_i < B_{\mathrm{a}}} p_i^{e_i}$ s.t. $B_{\mathrm{a}} < q \le B'_{\mathrm{a}}$ for additional threshold parameters $B'_{\mathrm{r}},\ B'_{\mathrm{a}}$ (the large prime variation). Similarly, two or three q's can be used.

Table 1. Specifications of the developed sieving device CAIRN 2

FPGA	Xilinx Virtex-4 XC4VLX200
	Logic Cell 200,448, Block RAM 336 × 18 Kbit, BGA 1513pin
	DDR SDRAM (1 GByte + 2 GByte) × 2 systems
Controller	CPU board: ADVANTECH's SOM-2353
	CPU: AMD Geode GX1 300 MHz (x86)
Output I/F	Direct I/O (50pin), 100 BaseTx, RS232C, VGA, KBD/Mouse
Frequency	133MHz (FPGA), 32bit/33MHz (PCI BUS), 83.4 MHz (Direct I/O)

Let us look at the sieving step in detail. We want to find relations as efficiently as possible. To do so, we use the following trick: when x is B-smooth, we have $\log_2 x - \sum_{p_i|x} e_i \log_2 p_i = 0$, where p_i is prime and $e_i = \mathrm{ord}_{p_i} x$ (the number how many times can x be divided by p_i). Since integer arithmetics are more efficient and $e_i = 1$ holds for large p_i, we use an approximation[3] $\lceil \log_2 x \rceil - \sum_{p_i|x} \lceil \log_2 p_i \rceil$. For a fixed value b, suppose we are going to find rational or algebraic relations from a subinterval with length S, namely S pairs of $(\bar{a}, b), \ldots, (\bar{a} + S - 1, b)$. Then we prepare S registers $R[0], \ \ldots, \ R[S-1]$. These registers are initialized by $\log_2 \lceil F(\bar{a} + j, b) \rceil$, respectively. For a prime $p < B$, if we have $p | F(\bar{a} + j, b)$, we subtract $\lceil \log_2 p \rceil$ from the corresponding register $R[j]$. After checking all primes less than B, we pick up (a, b) such that corresponding register almost equals 0.

Since polynomials $F(x, y)$ have a property that, if $p | F(a, b)$ holds, then we have $p | F(a + p, b)$. Thus, for a prime p, if we find an integer r such that $p | F(r, b)$, then we have $p | F(r + p, b), \ p | F(r + 2p, b), \ldots$. Once such r for each prime p is found, subtractions can be done very efficiently. The set of factor bases $\mathrm{FB_r}$ ($\mathrm{FB_a}$) consist of primes less than B_r (B_a) and corresponding r-values. The r-value is updated after $\lceil \log_2 p \rceil$ is subtracted by $r \leftarrow r + p$. When we go to the next subinterval, the first r-value for each p in the next interval is easily obtained from the old r-value (namely, $r \leftarrow r \bmod S$).

The above described sieving procedure is called the line sieving. A sample algorithm is shown in Algorithm 1. Note that in recent software implementations of NFS, the lattice sieving [Pol91] is used because of the efficiency. Since the main purpose of our implementation is to establish actually executable sieving device, we adopted the simpler one (the line sieving) rather than the complex one (the lattice sieving).

3 Implementational Details

3.1 Target Parameter

First of all, we have to consider data sizes of input/output. In our case, major input parameters for the sieving device are polynomials $f_\mathrm{r}(x), \ f_\mathrm{a}(x)$, sets of the factor bases $\mathrm{FB_r}, \mathrm{FB_a}$, sieving region parameters $H_a, \ H_b$. Since sizes of these

[3] If this value becomes negative, we treat the value as 0.

parameters can be determined from a target integer to be factored, it is enough to determine it.

Currently, 1024-bit integers are commonly used in practical RSA-based cryptosystems, so it is valuable to try factoring 1024-bit integers by the device. According to sample NFS parameters for factoring a 1024-bit integer (known as RSA1024) [LTS+03], it is estimated that about 36 GByte memory is required for the factor bases. A device with such huge memory is not visionary for the moment, however, controlling such huge memory space will be too complex to implement.

The next attractive integer will be 768-bit. Again, according to sample NFS parameters for factoring a 768-bit integer (known as RSA768) [LTS+03], it is estimated that 432 MByte memory is required for factor bases. Thus, we decided to design a sieving device which can handle up to 768-bit integers.

3.2 Platform

In previous papers, ASIC-based sieving devices are proposed to optimize the efficiency (namely, the AT product) [GS03, ST03, GS04, FKP+05, GJK+06, GS07]. However, no implementational results have been reported so far. One of the reason may that ASIC-based devices require a large amount of money and time. Thus we determined to use FPGA so that actually executable sieving device can be developed. Since the device requires huge amount of memory, we selected Xilinx's Virtex-4 XC4VLX200 [Xilinx] as a platform. Also, we used a reconfigurable processor DAPDNA-2 by IPFlex [IPFlex]. DAPDNA-2 has two processors DAP and DNA: DAP is a controller (a usual RISC processor), while DNA is a reconfigurable hardware with 376 fixed process elements which can be connected programably. The configuration can be changed in 1 clock without resetting the device. Comparing FPGA and DAPDNA-2, programming of DAPDNA-2 is much easier but the speed is slower. Thus, we use DAPDNA-2

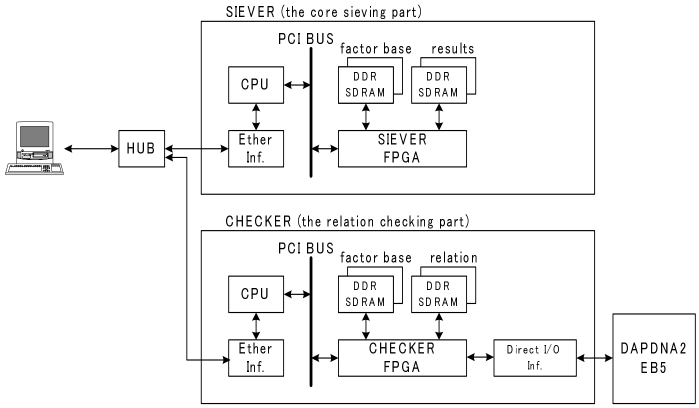

Fig. 1. Organization of the sieving device CAIRN 2

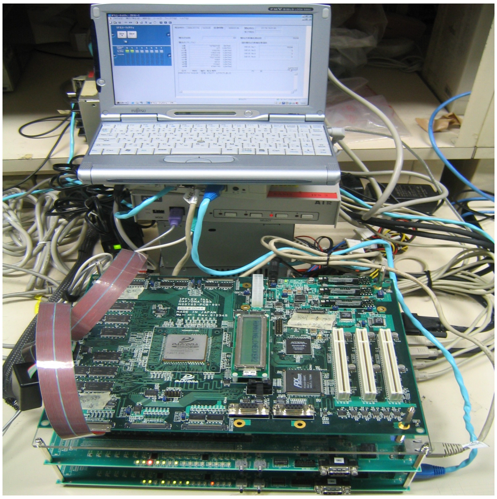

Fig. 2. Outlook of the sieving device CAIRN 2

for complex procedures (the primality test and the mini-factoring) and FPGA for simple procedures (the core sieving and the trial division).

The developed sieving device consists of 2 FPGA boards and 1 DAPDNA-EB5 board (which consists of DAPDNA and I/O interfaces) as in Figure 1. FPGA boards are connected via 100 BaseT ethernet, while the DAPDNA-EB5 board is connected to an FPGA board via the direct I/O. The device is connected to a control PC via 100 BaseT ethernet. Functionally, the sieving device consists of SIEVER and CHECKER. SIEVER processes the core sieving (on one FPGA board) while CHECKER processes the relation checking (the trial division on another FPGA board, the primality test and the mini-factoring on DAPDNA-EB5 board). In our design, a sieving device can handle a several SIEVERs and CHECKERs, however, we only implemented 1 SIEVER and 1 CHECKER in a device.

3.3 SIEVER (Core Sieving Part)

In this and next subsections, we describe SIEVER and CHECKER in detail. SIEVER is implemented on FPGA, which collects candidates $\{(a, b)\}$ from a given region by the line sieving (Algorithm 1). Since a main purpose of our first implementation is to establish actually executable sieving device, we adopted the naive line sieving (Algorithm 1).

Pipelined Sieving for Small Primes. Inputed a subinterval with length $S = 2^{19}$, SIEVER sieves the subinterval, and after the sieving, it sends candidates to SDRAM. In other words, SIEVER can sieve with only 1 prime. Since small primes (less than S) require much time for the sieving, we use the following pipelined implementation in order to optimize the efficiency.

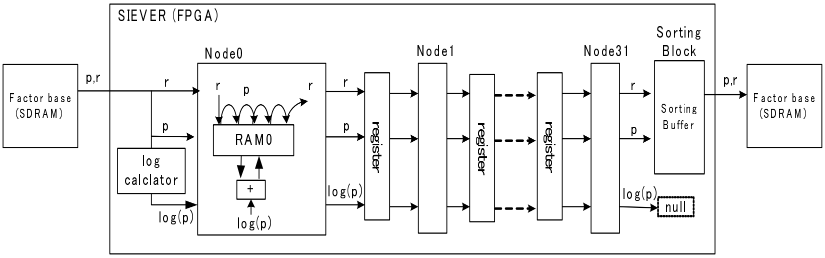

Fig. 3. Pipelined sieving

Table 2. Partitioned factor bases FB_i with $S = 2^{19}$

Partition	Condition on p	# of primes	Partition	Condition on p	# of primes
FB_0	$2 \le p < 2^{19}$	43,390	FB_6	$2^{24} \le p < 2^{25}$	985,818
FB_1	$2^{19} \le p < 2^{20}$	38,635	FB_7	$2^{25} \le p < 2^{26}$	1,894,120
FB_2	$2^{20} \le p < 2^{21}$	73,586	FB_8	$2^{26} \le p < 2^{27}$	3,645,744
FB_3	$2^{21} \le p < 2^{22}$	140,336	FB_9	$2^{27} \le p < 2^{28}$	7,027,290
FB_4	$2^{22} \le p < 2^{23}$	268,216	FB_{10}	$2^{28} \le p < 2^{29}$	13,561,907
FB_5	$2^{23} \le p < 2^{24}$	513,708	FB_{11}	$2^{29} \le p < 2^{30}$	26,207,278

We prepare 32 sieving nodes connected to each other in a pipelined manner as in Figure 3. Each sieving node is responsible for the mini-interval with length $S/32$ and has $S/32$ registers where log values are stored. These sieving nodes share 1 set of factor bases as in the following. When a prime p is sieved over the first sieving node Node0, p is handed to the next sieving node Node1 and a next prime p' is read from SDRAM simultaneously. In this implementation, a prime occupies a mini-interval with length $S/32$ only (rather than a subinterval with length S).

Partitioned Factor Bases for Large Primes. In our implementation, the length of the subinterval S is 2^{19}. Since most primes are larger than S, these primes are used for the sieving with low probability (remember that the probability that a prime p is used for the sieving in a subinterval with length S is just S/p). More worse, about $1 - S/\pi(B_\mathrm{r}) = 90.90\%$ factor bases in FB_r and $1 - S/\pi(B_\mathrm{a}) = 98.96\%$ factor bases in FB_a are not used in a interval. In order to avoid useless memory read and improve the efficiency for large primes (larger than S), we partition factor bases by according to p-values and r-values as in the followings: let S be the length of a subinterval. First, we partition a set of factor bases $\mathrm{FB} = \{(p,r) | p < B\}$ into

$$\mathrm{FB}_0 = \{(p,r) | 2 \le p < S\},$$
$$\mathrm{FB}_i = \{(p,r) | 2^{i-1} \cdot S \le p < 2^i \cdot S\} \ (i = 1, \ldots, 11)$$

according to p-values. Table 2 shows each partitioned factor bases FB_i and the number of included primes with $S = 2^{19}$. Then, we partition each FB_i into 2^i sub factor bases $\mathrm{SB}_i^{(j)}$ $(j = 1, \ldots, 2^i)$ by

$$\mathrm{SB}_i^{(j)} = \{(p, r')|(p, r) \in \mathrm{FB}_i, r' = r \bmod S,\ j \cdot S \leq r < (j+1) \cdot S\}$$

according to r-values. Table 3 shows some partitioned sub factor bases $\mathrm{SB}_i^{(j)}$ $(i = 0, 1, 2, 3)$ and the number of included primes with $S = 2^{19}$. By the above partitions, factor bases used in the first sieving for the first subinterval are included in $\mathrm{SB}_i^{(0)}$, and other primes (not included in $\mathrm{SB}_i^{(j)}$) are not used in the first sieving. Thus, useless reading is avoided in the first sieving.

Updating Factor Bases. The next task is to process the sieving with keeping a similar property, namely, in the t-th sieving, we want to collect factor bases used in the next interval in $\mathrm{SB}_i^{(t+1)}$. To do so, we implemented the core sieving as in Algorithm 2. Here, we are supposed to sieve the subinterval $(-H_a + t \cdot S, b), \ldots, (-H_a + (t+1) \cdot S - 1, b)$ $(t = 0, \ldots, 2H_a/S - 1)$ with length S and corresponding registers are $R[0], \ldots, R[S-1]$.

Alg. 2. Outline of the sieving with update

```
for $i \leftarrow 0$ to 11
  process the followings for each factor base $(p, r')$ included in $\mathrm{SB}_i^{(j)}$ $(j = t \bmod 2^i)$
    while $r' < S$
      $R[r'] \leftarrow R[r'] - \lceil \log_2 p \rceil$
      $r' \leftarrow r' + p$
    append a factor base $(p, r' \bmod S)$ to $\mathrm{SB}_i^{(j')}$ $(j' = j + \lceil r/S \rceil \bmod 2^i)$
```

Table 3. Partitioned sub factor bases $\mathrm{SB}_i^{(j)}$ with $S = 2^{19}$

Sub partition	Condition
$\mathrm{SB}_0^{(0)}$	$\{(p, r') \in \mathrm{FB}_0 \mid 0 \leq r < 2^{19}, r' = r\}$
$\mathrm{SB}_1^{(0)}$	$\{(p, r') \in \mathrm{FB}_1 \mid 0 \leq r < 2^{19}, r' = r\}$
$\mathrm{SB}_1^{(1)}$	$\{(p, r') \in \mathrm{FB}_1 \mid 2^{19} \leq r < 2 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_2^{(0)}$	$\{(p, r') \in \mathrm{FB}_2 \mid 0 \leq r < 2^{19}, r' = r\}$
$\mathrm{SB}_2^{(1)}$	$\{(p, r') \in \mathrm{FB}_2 \mid 2^{19} \leq r < 2 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_2^{(2)}$	$\{(p, r') \in \mathrm{FB}_2 \mid 2 \cdot 2^{19} \leq r < 3 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_2^{(3)}$	$\{(p, r') \in \mathrm{FB}_2 \mid 3 \cdot 2^{19} \leq r < 4 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_3^{(0)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 0 \leq r < 2^{19}, r' = r\}$
$\mathrm{SB}_3^{(1)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 2^{19} \leq r < 2 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_3^{(2)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 2 \cdot 2^{19} \leq r < 3 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_3^{(3)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 3 \cdot 2^{19} \leq r < 4 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_3^{(4)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 4 \cdot 2^{19} \leq r < 5 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_3^{(5)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 5 \cdot 2^{19} \leq r < 6 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_3^{(6)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 6 \cdot 2^{19} \leq r < 7 \cdot 2^{19}, r' = r \bmod 2^{19}\}$
$\mathrm{SB}_3^{(7)}$	$\{(p, r') \in \mathrm{FB}_3 \mid 7 \cdot 2^{19} \leq r < 8 \cdot 2^{19}, r' = r \bmod 2^{19}\}$

Table 4. Size of partitioned sub factor bases $\mathrm{SB}_i^{(j)}$

Algebraic FB

	# of $\mathrm{SB}_i^{(j)}$	Max. # of FB
$\mathrm{SB}_0^{(j)}$	1	65,536
$\mathrm{SB}_1^{(j)}$	2	32,768
$\mathrm{SB}_2^{(j)}$	4	32,768
$\mathrm{SB}_3^{(j)}$	8	32,768
$\mathrm{SB}_4^{(j)}$	16	32,768
$\mathrm{SB}_5^{(j)}$	32	32,768
$\mathrm{SB}_6^{(j)}$	64	32,768
$\mathrm{SB}_7^{(j)}$	128	32,768
$\mathrm{SB}_8^{(j)}$	256	24,576
$\mathrm{SB}_9^{(j)}$	512	24,576
$\mathrm{SB}_{10}^{(j)}$	1,024	24,576
$\mathrm{SB}_{11}^{(j)}$	2,048	24,576
Total		102,760,448
		822 MByte

Rational FB

	# of $\mathrm{SB}_i^{(j)}$	Max. # of FB
$\mathrm{SB}_0^{(j)}$	1	65,536
$\mathrm{SB}_1^{(j)}$	2	32,768
$\mathrm{SB}_2^{(j)}$	4	32,768
$\mathrm{SB}_3^{(j)}$	8	32,768
$\mathrm{SB}_4^{(j)}$	16	32,768
$\mathrm{SB}_5^{(j)}$	32	32,768
$\mathrm{SB}_6^{(j)}$	64	32,768
$\mathrm{SB}_7^{(j)}$	128	32,768
$\mathrm{SB}_8^{(j)}$	256	24,576
Total		14,680,064
		117 MByte
Buffer	32	32,768
Total		1,114,112 (9MByte)

Table 5. Overflow probabilities for $\mathrm{SB}_i^{(j)}$

	Average	σ	Pr.
$\mathrm{SB}_0^{(j)}$	43390.00	0.00	0
$\mathrm{SB}_1^{(j)}$	26856.52	86.19	2^{-3400}
$\mathrm{SB}_2^{(j)}$	25574.99	127.77	2^{-2293}
$\mathrm{SB}_3^{(j)}$	24382.34	141.33	2^{-2546}
$\mathrm{SB}_4^{(j)}$	23298.29	145.58	2^{-3059}
$\mathrm{SB}_5^{(j)}$	22309.65	145.95	2^{-3711}
$\mathrm{SB}_6^{(j)}$	21403.50	144.64	2^{-4461}
$\mathrm{SB}_7^{(j)}$	20559.50	142.57	2^{-5296}
$\mathrm{SB}_8^{(j)}$	19785.37	140.26	2^{-847}
$\mathrm{SB}_9^{(j)}$	19066.86	137.89	2^{-1158}
$\mathrm{SB}_{10}^{(j)}$	18397.14	135.54	2^{-1505}
$\mathrm{SB}_{11}^{(j)}$	17774.22	133.27	2^{-1885}

Buffer Estimation. Because of the update process, the number of factor bases in $\mathrm{SB}_i^{(j)}$ changes all time while in the sieving. Thus, the maximum number of factor bases in a partitioned sub factor bases $\mathrm{SB}_i^{(j)}$ should be considered.

In our device, sizes of each $\mathrm{SB}_i^{(j)}$ are set as in Table 4. With these parameters and the sieving implementation (Algorithm 2), we estimated the overflow probability from averaged numbers of factor bases in $\mathrm{SB}_i^{(j)}$ and their standard deviations σ observed from 2^{32} simulations on a PC. Results are summarized in Table 5, where we assumed that the number of factor bases in $\mathrm{SB}_i^{(j)}$ obey the normal distribution. Consequently, in our device, the overflow may not occur in practice because the probability is at most 2^{-848}.

Parallelized Buffers and Bucket Sorting. Since SDRAM memory access is done in 64-bytewise, appending a factor base to a new $\mathrm{SB}_i^{(j)}$ can be processed efficiently by using buffers. In the worst case, 2048 buffers are required (when $\mathrm{SB}_{11}^{(j)}$s are used) for the update, however, it is beyond the available memory.

Instead, by using the bucket sorting technique, we prepare only 32 buffers and store updated factor bases in them.

Computing Log Values. When the core sieving is processed, in addition to a prime p and the corresponding value r, a log value $\lceil \log_2 p \rceil$ is also required. Since $p,\ r$ are 32-bit and $\log_2 p$ is 8-bit, each factor base requires 72-bit memory. In our sieving device, each factor base is stored in DDR SDRAM outside the core sieving FPGA and supposed to be read successively. Thus it is desirable that the length of each factor base is multiple of 32. Keeping the log values is one solution, however, since memory amount is critical in our device, we give up keeping the log value in SDRAM. Instead, we compute $\lceil \log_2 p \rceil$ every time it is required. Strongly note that it is enough to know the highest bit position of p for computing $\lceil \log_2 p \rceil$, so overheads of the computation can be neglected.

3.4 CHECKER (Relation Check Part)

In this part, after obtained candidates in the core sieving, we check each candidate whether it really is a relation or not. CHECKER processes the trial division on the FPGA board, and the primality test and the mini-factoring on the DAPDNA board.

Trial Division. First, we directly check the divisivility by small primes (namely, up to 2^{30} for rational and 2^{27} for algebraic sievings) via the trial division. A candidate (a, b) is divisible by p when $a + b \cdot s \bmod p = 0$ for a factor base p and an integer s such that $f(s) = 0 \bmod p$. This test requires at most 64-bit multiplications and 32-bit divisions. We implement the trial division on an FPGA board different from SIEVER.

Primality Test. After the trial division, we check the primality of the cofactor by the Fermat test and the Euler method with base 2 only (this is enough since we do not require the perfect primality test). When the cofactor is resulted in composite, it is sent to the mini-factoring. Otherwise, it is treated as a relation if the cofactor is not larger than the threshold B'. This test is implemented on the DAPDNA-EB5 board, which is connected to the trial division FPGA via the direct I/O.

Mini-Factoring. Finally, we factor the cofactor by some factorization methods. Since the cofactor is relatively small (up to 128-bit) and small factors are already removed in the trial division, light methods are used in this mini-factoring. In our sieving device, we implemented the Pollard's ρ-method on DAPDNA-2.

4 Factoring a 423-Bit Integer

In this section, we show the experimental results of the integer factorization of a 423-bit integer with the developed sieving device CAIRN 2.

Table 6. Parameters for factoring a 423-bit integer

B_{r}	$3,000,000$	B'_{r}	6,000,000
B_{a}	$22,000,000$	B'_{a}	$30,000,000$
H_a	$2,300,000,000$	H_b	30,000

A target composite was selected from the Cunningham project [Cun] which is an Internet project to factor unfactored composites in the form $a^n \pm 1$ for small a and large n. We used a 423-bit integer N which is included in $7^{352}+1$ (989-bit) and had remained unfactored when we started the experiment:

$$\begin{aligned} N = {} & 1100292287\ 2496853405\ 9383191827\ 3088033131\ 3742514339\ 1686904758 \\ & 5356090653\ 2662764313\ 9824106278\ 4801654937\ 1557142696\ 9864417564 \\ & 88958657. \end{aligned}$$

Before factoring N by NFS, we applied ECM in advance for a while and failed. Thus we were convinced that N is a product of two primes with high probability. Parameters used in the factorization are summarized in Table 6.

Polynomial Selection Step. We used the Kleinjung and Franke's software for this step. After 34 hours computation on a PC (Pentium 4 Prescott, 3.8 GHz, Memory 2 GBytes, we use the same PC in the following), two polynomials

$$\begin{aligned} f_{\mathrm{r}}(x) &= 5175123296671x - 1362966569805857108976278, \\ f_{\mathrm{a}}(x) &= 2339280x^5 - 224252480052x^4 - 3621428496137064 6x^3 \\ & \quad +408360934897040026852x^2 + 1016360227410971377726 77441x \\ & \quad -263678243765181773090543855595, \end{aligned}$$

were obtained.

Sieving Step. The sieving step was processed on 1 set of the developed sieving device. The core sieving took about 30 days (42 days in calendar) and found

Table 7. Details of the linear algebra step

		# of Relations	# of Factor bases	Time
Filtering	In	2,698,117	2,792,081	8 Min.
	Out	1,313,971	1,312,969	
Gaussian Elimination	In	1,313,971	1,312,962	15 Min.
	Out	600,718	599,704	
Removing Heavy Weight Indices	In	184.148 MByte		1 Min.
	Out	132.876 MByte		
Lanczos Computation	In	$600,718 \times 599,480$ matrix		103 Min.
	Out	256 solutions		
Recovering Heavy Weight Indices	Out	32 solutions		5 Min.

2,828,755 relations. After removing 30,025 bad relations (which do not satisfy the desired format) and 100,613 duplicated relations, we obtained 2,698,117 relations for the next step. This procedure took about 19 minutes on the PC.

Linear Algebra Step. The linear algebra step and the square root step was processed on the PC. From 2,698,117 relations and 2,792,081 factor bases, we obtained 32 solutions in 132 minutes. Detailed data are summarized in Table 7.

Square Root Step. The first solution of 32 solutions brought an actual factorization. A rational square root R and an algebraic square root A are obtained from the first solution in 6 seconds and 18 minutes, respectively:

$$\begin{aligned} R = {} & 8264254310\ 5780678107\ 7260355319\ 7256576146\ 2501536747\ 5383577435 \\ & 6262381979\ 2641159252\ 9332800358\ 3963267372\ 5266446083\ 4479795292 \\ & 9881182, \\ A = {} & 5243143677\ 4850042977\ 2583462471\ 4452695094\ 3986687817\ 3054465838 \\ & 5952456214\ 0017654469\ 6893872943\ 7400764236\ 0303936118\ 0793853980 \\ & 039244. \end{aligned}$$

Here, the Nguyen's algorithm [Ngu98] is used. From these roots, we actually found a non-trivial factor of N by computing $\gcd(A - R, N)$ which took less than 1 second. Consequently, we obtained a complete factorization $N = P \times Q$ where

$$\begin{aligned} P \text{ (205-bit)} = {} & 4549363729\ 2816464852\ 0670147365\ 7133979231\ 5419859784 \\ & 2180768758\ 41, \\ Q \text{ (218-bit)} = {} & 2418563018\ 3133843753\ 7787898096\ 0626923598\ 1954330361 \\ & 9864074410\ 382977. \end{aligned}$$

Comparison with Software Implementation. As a comparison, we implemented the sieving step by the lattice sieving on a PC, which requires about a month for sieving the same region. Thus, the speed of the developed sieving device is comparable to the lattice sieving software implementation.

5 Concluding Remarks

In this paper, implementational results of the sieving step of NFS on Xilinx's FPGA are reported. Especially, we factored a 423-bit integer with the device “CAIRN 2” for the sieving and some PCs for other steps. The core sieving speed of the developed device is comparable to the lattice sieving software implementation. As far as the authors know, this is the first FPGA implementation and experiment of the sieving step (while implementational results of the mini-factoring and the linear algebra step have been reported [GKB+06, BMGG04]).

Because of the time limitation, we haven't experimented larger factorization with the device. Further experiments (especially factoring 663-bit or 768-bit integers) should be achieved. To do so, implementing the lattice sieving rather than the line sieving is indispensable. Moreover, manufactuarability and executability of previously proposed methods ([ST03], for example) should be evaluated via actual implementations.

Acknowledgements

First of all, the authors sincerely thank to Naoya Torii for his continuing support for the CAIRN project. On theoretical aspects of the research, the authors would like to thank the working group members: Kazumaro Aoki, Toshinori Fukunaga, Yuji Kida, Noboru Kunihiro, Tsutomu Matsumoto, Junji Shikata, Hiroaki Uede, Go Yamamoto and Kazuhiro Yokoyama. On implementational aspects, the authors thank to Akihiro Hayashi, Satoshi Nishimura, Kiyomitsu Katoh, Tomoharu Masuda and Tadashi Ishiwatari in Fujitsu Microelectronics Solutions, Kajiwara Takaharu, Toshihiro Yamanaka, Akihiro Mihata, Tatsuya Toyozumi, Jun-ichi Kugimiya, Kazunari Shiota and Kiichi Sugitani in FUJITSU Kyusyu Network Technologies, and Shunsuke Fueki in FUJITSU.

References

[AKSU04] Aoki, K., Kida, Y., Shimoyama, T., Ueda, H.: GNFS Factoring Statistics of RSA-100, 110,..., 150. Cryptology ePrint archive 2004/095, IACR (2004)

[AU03] Aoki, K., Ueda, H.: Sieving Using Bucket Sort. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 92–102. Springer, Heidelberg (2004)

[Ber01] Bernstein, D.: Circuits for integer factorization: a proposal. preprint (2001)

[BMGG04] Bajracharya, S., Misra, D., Gaj, K., El-Ghazawi, T.: Reconfigurable Hardware Implementation of Mesh Routing in the Number Field Sieve Factorization. In: FPT 2004, pp. 263–270. IEEE, Los Alamitos (2004)

[Cun] The Cunningham project: http://homes.cerias.purdue.edu/~ssw/cun/

[F+03] Franke, J., et al.: RSA-576. Email announcement (December 2003)

[FKP+05] Franke, J., Kleinjung, T., Paar, C., Pelzl, J., Priplata, C., Stahlke, C.: SHARK: A Realizable Special Hardware Sieving Device for Factoring 1024-bit Integers. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 119–130. Springer, Heidelberg (2005)

[GJK+06] Geiselmann, W., Januszewski, F., Köpher, H., Pelzl, J., Steinwandt, R.: A Simpler Sieving Device: Combining ECM and TWIRL. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296. Springer, Heidelberg (2006)

[GKB+06] Gaj, K., Kwon, S., Baier, P., Kohlbrenner, P., Le, H., Khaleeluddin, M., Bachimanchi, R.: Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 119–133. Springer, Heidelberg (2006)

[GS03] Geiselmann, W., Steinwandt, R.: A Dedicated Sieving Hardware. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 254–266. Springer, Heidelberg (2002)

[GS04] Geiselmann, W., Steinwandt, R.: Yet Another Sieving Device. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 278–291. Springer, Heidelberg (2004)

[GS07] Geiselmann, W., Steinwandt, R.: Non-Wafer-Scale Sieving Hardware for the NFS: Another Attempt to Cope with 1024-bit. In: EUROCRYPT 2007. LNCS, vol. 4515, pp. 466–481. Springer, Heidelberg (2007)

[IPFlex] IPFlex: DAPDNA Architecture. Available at `http://www.ipflex.com/en/E1-products/index.html`

[IKS05] Izu, T., Kogure, J., Shimoyama, T.: A Status Report: An Implementation of a Sieving Algorithm on a Dynamic Reconfigurable Processor (Extended Abstract). In: SHARCS 2005, ECRYPT (2005)

[IKKN+06] Izu, T., Katoh, K., Kogure, J., Nishimura, S., Shimoyama, T.: An Implementation of a Sieving Algorithm in the Number Field Sieve on a Dynamic Reconfigurable Processor (Extended Abstract). In: JWIS 2006 (2006)

[KM00] Kim, H.J., Mongione-Smith, W.: Factoring Large Numbers with Programmable Hardware. In: FPGA 2000, pp. 41–48. ACM Press, New York (2000)

[LL93] Lenstra, A., Lenstra, H.(eds.): The Development of the Number Field Sieve. Lecture Notes in Mathematics (LNM), vol. 1554. Springer, Heidelberg (1993)

[LLMP90] Lenstra, A., Lenstra, H., Manasse, M., Pollard, J.: The Number Field Sieve. In: STOC 1990, pp. 564–572. ACM Press, New York (1990)

[LS00] Lenstra, A., Shamir, A.: Analysis and Optimization of the TWINKLE Factoring Device. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 35–52. Springer, Heidelberg (2000)

[LTS+03] Lenstra, A., Tromer, E., Shamir, A., Kortsmit, W., Dodson, B., Hughes, J., Leyland, P.: Factoring Estimates for a 1024-bit RSA Modulus. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 55–74. Springer, Heidelberg (2003)

[LSTT02] Lenstra, A., Shamir, A., Tomlinson, J., Tromer, E.: Analysis of Bernstein's Circuit. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 1–26. Springer, Heidelberg (2002)

[Ngu98] Nguyen, P.: A Montgomery-like Square Root for the Number Field Sieve. In: Buhler, J.P. (ed.) Algorithmic Number Theory. LNCS, vol. 1423, pp. 151–168. Springer, Heidelberg (1998)

[Pol91] Pollard, J.: The Lattice Sieve, pp. 43–49 (1991) in [LL93]

[Sha99] Shamir, A.: Factoring Large Numbers with the TWINKLE Device (Extended Abstract). In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 2–12. Springer, Heidelberg (1999)

[SPK+05] Šimka, M., Pelzl, J., Kleinjung, T., Franke, J., Priplata, C., Stahlke, C., Drutarovský, M., Fischer, V., Parr, C.: Hardware Factorization Based on Elliptic Curve Method. In: FCCM 2005, pp. 107–116. IEEE, Los Alamitos (2005)

[ST03] Shamir, A., Tromer, E.: Factoring large numbers with the TWIRL device. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 1–26. Springer, Heidelberg (2003)

[Xilinx] Xilinx: Vertex-4 Multi-Platform FPGA, Available at `http://www.xilinx.com/silicon_solutions/fpgas/virtex/virtex4/index.htm`