Non-Wafer-Scale Sieving Hardware for the NFS: Another Attempt to Cope with 1024-bit

Willi Geiselmann[1] and Rainer Steinwandt[2]

[1] IAKS, Fakultät für Informatik, Universität Karlsruhe (TH), Am Fasanengarten 5, 76128 Karlsruhe, Germany, geiselma@ira.uka.de

[2] Department of Mathematical Sciences, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA, rsteinwa@fau.edu

Abstract. Significant progress in the design of special purpose hardware for supporting the Number Field Sieve (NFS) has been made. From a practical cryptanalytic point of view, however, none of the published proposals for coping with the sieving step is satisfying. Even for the best known designs, the technological obstacles faced for the parameters expected for a 1024-bit RSA modulus are significant.

Below we present a new hardware design for implementing the sieving step. The suggested chips are of moderate size and the inter-chip communication does not seem unrealistic. According to our preliminary analysis of the 1024-bit case, we expect the new design to be about 2 to 3.5 times slower than TWIRL (a wafer-scale design). Due to the more moderate technological requirements, however, from a practical cryptanalytic point of view the new design seems to be no less attractive than TWIRL.

Keywords: RSA, cryptanalytic hardware, factoring integers, NFS

1 Introduction

Even for the best known factoring algorithms, coping with the complexity of a factorization of a 1024-bit RSA modulus looks extraordinary challenging. In an attempt to bring such a record factorization closer to what is currently feasible, various hardware designs to support implementations of the Number Field Sieve (NFS) have been made. While theoretical advances in the design of factoring algorithms are more desirable, at the moment these special purpose designs for speeding up time-critical computations in the NFS seem to be the most promising approach for practically challenging a 1024-bit RSA modulus. After a series of works on the linear algebra step of the NFS [Ber01,LSTT02,GS03b,GKST05,GSST05], one may adopt the position that the linear algebra step expected for a 1024-bit factorization is by now close to or in reach of current technology.

On the other hand, none of the suggested designs implementing the sieving step of the NFS is really satisfying:

– TWINKLE [Sha99,LS00] builds on an opto-electronic hybrid design where no promising parameter set for the 1024-bit has been proposed.
– For designs building on a mesh architecture, no promising specification for the 1024-bit case is known (cf. [Ber01,GS03a,GS04,IKOS06]).
– SHARK [FKP+05] imposes the use of an elaborate butterfly transport system, whose implementation is far from trivial.
– TWIRL [ST03,LTS+03] seems to be the currently best-explored design. Unfortunately, it is a wafer-scale design building on a quite complex layout.

In an attempt to reduce the layout complexity, Geiselmann et al. [GJK+06] recently proposed to combine a modified TWIRL with an "ECM engine": For 1024-bit parameters of interest, [GJK+06] argues that an optimized implementation of the Elliptic Curve Method (ECM) can efficiently compute all factorizations of (semi-)smooth norms occurring in the sieving step. The idea is that in this way the circuitry for TWIRL's "diary part", which stores large prime factors of norms, can be removed. The design we present below also relies on this idea: We do not store any prime factors encountered during relation collection and assume a postprocessing of the sieving output with an ECM engine as described in [GJK+06]. However, unlike TWIRL, the device proposed below is a non-wafer-scale design.

After having recalled some facts on the sieving step in the NFS in Section 2, in Section 3 we describe our design that builds on ideas of several published proposals: Like the mesh-based proposals, we implement a version of line sieving where each sieving line is split into consecutive subintervals. To overcome the need of a wafer-scale design, we distribute (the majority of) the factor bases on moderately sized chips. The circuitry on these chips produces the arithmetic progressions needed for sieving and is inspired by TWIRL. Eventually, to combine the sieving contributions of the different factor base elements, we use a central unit whose structure reminds of the linear algebra design proposed in [GSST05]. In our preliminary analysis of the 1024-bit case, for the ease of comparability we adopt the technological parameters and the NFS parameters from [ST03]. Summarizing, we expect our device to be about 2 to 3.5 times slower than TWIRL. On the other hand, the maximal chip size involved is 493 mm^2 and also the interconnection circuitry among these chips does not seem utopian. From a practical point of view, this new design appears to be no less attractive than the existing hardware designs for implementing the sieving step.

2 Preliminaries: Sieving in the NFS

For the purposes of this paper, it is sufficient to recall the basic set-up of so-called line sieving in the NFS. For a more thorough discussion of the NFS we refer to the standard reference [LHWL93].

2.1 Line sieving

In a precomputation phase of the NFS two univariate polynomials $f_1(x), f_2(x) \in \mathbb{Z}[x]$ with integer coefficients are determined that have a root m modulo n in common:

$$f_1(m) \equiv f_2(m) \equiv 0 \pmod{n}$$

A typical choice is to have $f_1(x)$ of degree $d \geq 5$ and $f_2(x)$ to be monic and linear, i. e., $f_2(x) = x - m$. By setting $F_1(x, y) := y^d \cdot f_1(x/y)$ resp. $F_2(x, y) := y \cdot f_2(x/y)$, two homogeneous polynomials $F_1(x, y), F_2(x, y) \in \mathbb{Z}[x, y]$ are derived. Now everything related to the polynomial $f_1(x)$ resp. $F_1(x, y)$ is said to belong to the *algebraic side*, whereas everything related to the polynomial $f_2(x)$ resp. $F_2(x, y)$ is referred to as belonging to the *rational side*. In particular, for given smoothness bounds $B_1, B_2 \in \mathbb{N}_0$ the sets

$$P_i := \{(p, r) : f_i(r) \equiv 0 \pmod{p},\ p \text{ prime},\ p < B_i,\ 0 \leq r < p\} \subseteq \mathbb{N}^2 \quad (i = 1, 2)$$

are known as algebraic and rational *factor base*, respectively.

Throughout the relation collection step, pairs of integers $(a, b) \in \mathbb{Z} \times \mathbb{N}$ with $\gcd(a, b) = 1$ are to be found, so that the values $F_1(a, b)$ and $F_2(a, b)$ are *smooth*. This means that the values $F_1(a, b)$ and $F_2(a, b)$ both factor over the primes $< B_1$ resp. $< B_2$, except for a small number of prime factors. At this, the precise number of 'extra' prime factors on the rational and algebraic side is not necessarily identical. The actual computation of (a, b)-pairs where both $F_1(a, b)$ and $F_2(a, b)$ are smooth can be performed by means of a sieving process, e. g., over a rectangular region $-A \leq a < A$, $0 < b \leq B$ with $A, B \in \mathbb{N}$. For organizing this sieving process, different

$$
\begin{array}{l}
b \leftarrow 0 \\
\textbf{repeat} \\
\quad b \leftarrow b+1 \\
\quad \textbf{for } i \leftarrow [1,2] \\
\quad\quad s_i(a) \leftarrow 0 \quad (\forall a:\ -A \leq a < A) \\
\quad\quad \textbf{for } (p,r) \leftarrow P_i \\
\quad\quad\quad s_i(br+kp) \leftarrow s_i(br+kp) + \log_{\sqrt{2}}(p) \quad (\forall k:\ -A \leq br+kp < A) \\
\quad \textbf{for } a \leftarrow \{-A \leq a < A : \gcd(a,b)=1,\ s_1(a) > T_1, \text{ and } s_2(a) > T_2\} \\
\quad\quad \text{check if both } F_1(a,b) \text{ and } F_2(a,b) \text{ are smooth} \\
\textbf{until} \text{ enough pairs } (a,b) \text{ with both } F_1(a,b) \text{ and } F_2(a,b) \text{ smooth are found}
\end{array}
$$

Fig. 1. Line sieving

techniques are known, and for our purposes we focus on simple *line sieving* as outlined in Figure 1. At this, the thresholds T_i correspond to the bitlength of the remaining cofactor on the algebraic and rational side, respectively. The T_i-values are to be updated several times throughout the sieving. For the sake of efficiency, in an an actual implementation the values $\log_{\sqrt{2}}(p)$ are usually replaced by an integer approximation. Also the use of base 2-logarithms is certainly not mandatory. In analogy to [ST03], in the sequel we will use a 10-bit counter for summing up approximations $\lceil \log_{\sqrt{2}}(p) \rceil$. It is worth noting that testing the norms $F_1(a,b)$, $F_2(a,b)$ for smoothness and in case of smoothness recovering their prime factors is computationally nontrivial. For the device proposed below, we rely on a design as presented in [GJK+06], which uses an optimized ECM implementation to perform the required norm factorizations in connection with a TWIRL-based realization of the sieving step.

2.2 Choice of 1024-bit parameters

Deducing a reliable estimate for the NFS parameters suitable for a factorization of a 1024-bit RSA modulus is a non-trivial problem in its own and outside the scope of this paper. Already for the sake of comparability, here we adopt parameters from [ST03]. Summarizing, for the sequel the following parameter choices are of interest:

- On the algebraic side, the smoothness bound $B_1 = 2.6 \cdot 10^{10}$ is used.
- On the rational side, the smoothness bound $B_2 = 3.5 \cdot 10^9$ is used.
- The sieving region $-A < a \leq A, 0 < b \leq B$ uses $A = 5.5 \cdot 10^{14}$ and $B = 2.7 \cdot 10^8$.
- The algebraic and rational polynomials are chosen of degree 5 and 1, respectively, as specified in [ST03, Appendix B.2].

For further details and a discussion on how to identify suitable NFS parameters, we refer to [LTS+03]. With the mentioned parameters, the factor bases are of size $|P_1| \approx 1.134 \cdot 10^9$ and $|P_2| \approx 1.673 \cdot 10^8$, respectively.

3 The Proposed Design: Main Components

For the sake of clarity, in this section we only discuss the basic structure of our design. Parameter choices we made for the case of a 1024-bit factorization are indicated in double brackets $\langle\!\langle \cdot \rangle\!\rangle$, but a discussion of implementation details is postponed to Section 4. The basic organization of the sieving process is analogous to [GS03a,GS04]. Namely, we divide each sieving line in subintervals of $S \langle\!\langle = 2^{26} \rangle\!\rangle$ consecutive sieve locations. Switching to the next subinterval within one sieving line can be done with local operations only. However, to switch to a different sieving line, i. e., to increase the b-value, new data is to be loaded into the device, and our running time analysis has to take this into account.

At a high level, the architecture of our design relies on two types of components, which we detail in the sequel: a) a collection unit that is in charge of updating the rational and algebraic sieving counters and b) stations that compute the arithmetic progressions needed for updating the counters.

3.1 Collection Unit

For each value in the current sieving interval, this part of our device hosts an algebraic and a rational DRAM counter for summing up the respective $\log_{\sqrt{2}}(p)$-values. Each of these counters has a size of $b\langle\!\langle = 10\rangle\!\rangle$ bit, and the counters are distributed onto a number $c\langle\!\langle = 2^{14}\rangle\!\rangle$ of identical processors. We refer to these processors as *counting units*, and each counting unit is in charge of $S/c\langle\!\langle = 2^{12}\rangle\!\rangle$ consecutive sieve locations. It is not necessary to place all counting units on a single chip, and we distribute them onto a small number $\gamma\langle\!\langle = 4\rangle\!\rangle$ of chips.

These γ chips are all organized in the same manner: we arrange the respective counting units in two-dimensional arrays of size $\sigma \times \sigma\langle\!\langle = 2^5 \times 2^5\rangle\!\rangle$, yielding a total number of $c/(\gamma\sigma^2)\langle\!\langle = 4\rangle\!\rangle$ arrays per chip. Each array is organized as depicted in Figure 2: The counting units in each row are connected through a circular bus, whereas the counting units within a column are connected through a unidirectional bus, originating in an *input part*. This structure is reminiscent of the linear algebra design in [GSST05].

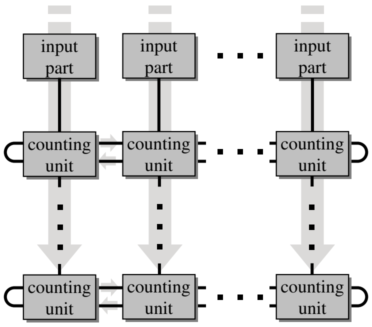

Fig. 2. Organization of one array of counting units

The input parts receive $(\log_{\sqrt{2}}(p), r)$-values from external *stations* (see below) with the r-value indicating to which of the counters in the array the respective $\log_{\sqrt{2}}(p)$-value is to be added. In each clock cycle, a received $(\log_{\sqrt{2}}(p), r)$-values passes (along with an algebraic/rational flag) on to the next row over the vertical bus. Then each counting unit checks whether the pair received on the vertical bus is to be handled in that row. If yes, the packet is removed from the vertical bus and via the circular horizontal bus transported to the correct counting unit. The latter then removes the received packet from the horizontal bus and adds the $\log_{\sqrt{2}}(p)$-value to the appropriate counter.

3.2 Computing the Arithmetic Progressions

Similarly, as in [ST03], to handle the arithmetic progressions for the $(\log_{\sqrt{2}}(p), r)$-pairs we use different types of circuits, and refer to these as *stations*. In dependence on the size of the prime number p, we distinguish four types of stations, whose structure is reminiscent of the stations in TWIRL.

Largish stations. These are in charge of primes p greater than a bound $B_{\text{largish}}\langle\!\langle = 1.5 \cdot 10^8 \rangle\!\rangle$, where $B_{\text{largish}} > S$. The majority of primes in the factor base is handled in this way. They "hit" no more than once per sieving interval, and the design of largish stations reflects this. Each such station handles a certain number $n_{\text{largish}}\langle\!\langle = 10^5 \rangle\!\rangle$ of factor base elements, which are stored in a sequence of DRAM banks as sketched in Figure 3. Each of the memory banks is operated as a stack, random access is not needed.

First, we initialize the sieving line defined through a specific b-value (starting with $b = 1$): For each factor base element (p, r) we replace r with $br \mod p$ (see Figure 1). This precomputation is performed on an external PC and the modified (p, r)-pairs are then loaded in the mentioned series of DRAM banks. The first DRAM bank holds all (p, r)-pairs that "hit" in the first sieving interval of size S, the second DRAM all those with an r-value indicating a hit in the second subinterval of size S, etc. The number n_{banks} of DRAM banks we need is $n_{\text{banks}} = \lceil p_{\max}/S \rceil + 1\langle\!\langle \leq 389 \rangle\!\rangle$ with $p_{\max}$ being the maximal prime handled by the station.[1] The number of entries $n_{\text{entries}}\langle\!\langle \leq 10^5 \rangle\!\rangle$ per DRAM bank has to suffice for holding all "hits" that can occur in a single subinterval of size S.

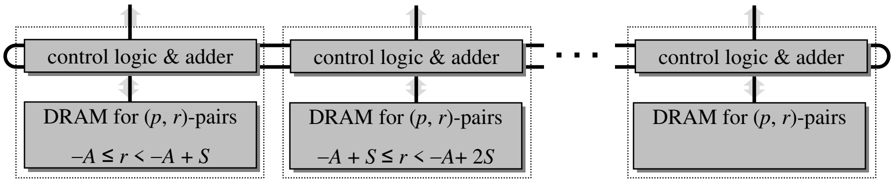

Fig. 3. Largish station

Now, for sieving the first subinterval, the first DRAM bank is read sequentially (or in small blocks of $\ell_{\text{largish}}\langle\!\langle = 2 \rangle\!\rangle$ values). The $\log_{\sqrt{2}}(p)$-approximations are constant within one unit, as the prime numbers handled in a unit are of approximately equal size. Along with the r-values (and a rational/algebraic flag), the approximation for $\log_{\sqrt{2}}(p)$ is sent over a unidirectional[2] bus to the appropriate array of the collection unit. Further on, an updated entry is written into the DRAM bank that handles the subinterval where the next "hit" for this progression occurs. More specifically, we proceed as follows: With the adder residing next to each DRAM bank, we compute the new r-value as $r \leftarrow r + p$. Now, choosing $S\langle\!\langle = 2^{26} \rangle\!\rangle$ as power of 2, the most significant bits of the new r-value can serve as counter indicating the number of "hops", that the updated (p, r)-pair has to travel among the cyclically connected DRAM banks. Once the pair has arrived at its destination DRAM, which handles the subinterval for the next "hit", the control logic associated to that DRAM bank removes the packet from the cyclic bus and appends it at the end of the entries currently stored in that DRAM. If there is no space left in this DRAM, the pair is deleted and lost for the entire sieving line. This never happened in our simulations.

Once a complete subinterval (i. e., a DRAM bank) has been processed, the unit proceeds to the (cyclic) successor of that DRAM and processes it in the same manner. In this way, the complete sieving line is processed.

Medium stations. For prime numbers that are smaller than the sieving interval size S, the respective arithmetic progressions may encounter several hits within one subinterval. For some bound $B_{\text{medium}}\langle\!\langle = 2^{13} \rangle\!\rangle$, we handle the primes $B_{\text{medium}} < p < B_{\text{largish}}$ as follows.

[1] Using one more DRAM bank than $\lceil p_{\max}/S \rceil$ avoids the problem of having to read and write from one DRAM bank at the same time.

[2] Only for initializing a new sieving line this bus is operated in the opposite direction.

In one station $n_{\mathrm{medium}}\langle\!\langle\approx 10^5\rangle\!\rangle$ pairs (p, r) are stored in a DRAM bank. As for the largish stations, to start a new sieving line, the r-value is to be initialized according to Figure 1. Unlike for largish primes, now we have only one DRAM bank in the station, and in order to save memory—or rather chip area—we sort the (p, r)-pairs according to p: In this way, storing the difference between primes is sufficient to recover the next p-value. As sketched in Figure 4, next to the DRAM bank and control logic, we also have an adder unit. The latter consists of an array of $\ell_{\mathrm{medium}}\langle\!\langle= 128\rangle\!\rangle$ adders.

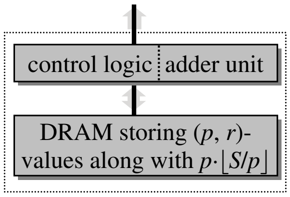

Fig. 4. Medium station

The DRAM will be processed sequentially in blocks of $\ell_{\mathrm{medium}}\langle\!\langle= 128\rangle\!\rangle$ entries. After reading such a block of (p, r)-values, it is forwarded to the addition unit, where the needed primes p are reconstructed from the stored differences. Also, similarly as in the largish stations, the needed $\log_{\sqrt{2}}(p)$-approximations are determined here. The adders now compute all values $r + k \cdot p$ that are relevant for the current subinterval, i. e., p is added as long as the obtained value is still smaller than $S\langle\!\langle= 2^{26}\rangle\!\rangle$, which for S being a power of 2 can be tested by observing a single bit. The respective $r + k \cdot p$-values are transmitted to the appropriate array of the collection unit—together with the $\log_{\sqrt{2}}(p)$-approximation and a rational/algebraic flag. In parallel to the computation of these $\ell_{\mathrm{medium}}\langle\!\langle= 128\rangle\!\rangle$ arithmetic progressions, the r-values stored in the DRAM are updated for the next subinterval. To this aim, along with each (p, r)-entry we also store the (precomputed) value $p \cdot \lfloor S/p\rfloor$ in the DRAM. Knowing this value, updating an r-value for the next sieving interval reduces to computing $r \leftarrow r + \lfloor S/p\rfloor \cdot p$. If this value does not exceed S yet, p has to be added. Eventually, we subtract S from the obtained new r-value.

To keep the number of pins of the chips holding the collection unit within acceptable boundaries, the medium stations will be hosted on the same chips as the collection unit. If the collection unit is distributed over several chips, we have to duplicate the medium stations accordingly. Also, as the medium stations are expected to produce relations at a very high rate, we equip the (unidirectional) buses into the collection unit's arrays with a "panic feedback flag". This allows the collection unit to put a medium station on hold until the buses and buffers can cope with new $(\log_{\sqrt{2}}(p), r)$-pairs again.

Smallish stations. We refer to factor base elements (p, r) with $p \leq B_{\mathrm{medium}}$ as *smallish*, and handle them in basically the same type of stations as just discussed. However, we do without a difference coding here. Progressions computed by smallish stations produce several hits within a subinterval, even within one array of the collection unit. Consequently we duplicate the smallish stations, so that on each chip of the collection unit all smallish primes can be handled locally.

4 Performance and Parameters for the 1024-bit Case

In this section we discuss more details of our design, when dealing with NFS parameters for a 1024-bit factorization as described in Section 2.2. For choosing and optimizing the design parameters specified below, we relied on simulations by means of a computer algebra system [BCP97] and a heuristic approach. We did not invoke a rigorous mathematical optimization and do not claim that our parameter choices are "the best possible".

In addition to the NFS parameter choices mentioned in Section 2.2, we fix the subinterval size $S := 2^{26}$ that specifies the number of consecutive sieve locations processed by our device at once. As outlined in the previous section, the progressions corresponding to the different types of primes are generated in different types of stations. Below we first describe the structure of the stations and their placement within the device. Section 4.2 details the structure of the collection unit.

4.1 Stations for the 1024-bit Case

To keep the amount of inter-chip communication at a reasonable level, we subdivide the stations for largish primes into three types. While the first type describes stations that are placed on a chip different from the chips hosting the collection unit, the other two types reside on the same chips as the collection unit. Similarly, we use two different types of medium stations, both residing on the same chips as the collection unit. For the sake of comparability with [ST03], for estimating the space complexities we assume a 0.13 μm process with a DRAM bit occupying about 0.2 μm^2 and a transistor occupying about 2.8 μm^2 of silicon.

Largish Stations

Type I. We use this type of stations for largish primes $> 1.5 \cdot 10^8$. As described before, the factor base elements are distributed onto different DRAMs, so that all primes of this station relevant for the processed size $S = 2^{26}$ subinterval are stored in one DRAM. For the chosen subinterval size $S = 2^{26}$, we choose the DRAM bank large enough to store up to $100,000$ (p, r)-pairs. For each such pair (p, r) the respective prime $p < 2^{35}$ and r-value $\pmod{2^{26}}$ are stored, yielding a total of 34+26 bit per DRAM entry.[3]

The DRAM is read sequentially, on average reading two pairs per clock cycle. Each r-value is sent—together with the 4-bit value $\lceil \log_{\sqrt{2}}(p) \rceil - 55$ that is chosen to be constant for the whole station—to a small *routing network* on the same chip (see below). An adder (with input widths 35 and 26 bit) calculates the next hit for p in the current sieving line. As described in Section 3.2, the pair $(p, r + p)$ is forwarded—through one of the two cyclic buses connecting all DRAM banks of the station—to the DRAM bank in charge of the subinterval where p hits next. More specifically, we send the value $(p, (r + p) \bmod 2^{26})$ to the DRAM bank that is $(r + p)$ div 2^{26} "hops" away. To implement this routing operation, adjacent to each DRAM two adders (shared among four DRAMs), a decrement and compare unit, and the memory cells for the two buses of width up to 69 each[4] are needed.

All in all, we estimate that this logic can be realized with 4000 transistors per DRAM bank. Together with 6,000,000 bits of storage space, the size of one DRAM bank with update logic is estimated to have a size of 1.2 mm^2. To handle both the algebraic and the rational factor base elements with $p > 1.5 \cdot 10^8$, we use 256 largish stations of Type I (156 algebraic and 99 rational ones). The number of DRAM banks per station varies from 4 up to 389, yielding a total of $13,440$ DRAM banks. We distribute the Type I stations on 32 chips, each holding 8 stations with ≈ 420 DRAM banks.

On each chip one *routing network* collects the 16 outputs of the 8 stations and distributes them to the correct array of size 2^{22} on the collection unit. This routing network is realized through a butterfly network with 16 inputs. Each of the four stages of the butterfly network has 16 buffers to store up to 30 pairs $(r, \lceil \log_{\sqrt{2}}(p) \rceil - 55)$. If one of the buffers is full, a panic flag

[3] The least significant bit of p is known to be 1.

[4] We need up to 9 bit for the "hop counter".

informs the parent nodes to stop sending data. The panic flags of the input nodes of the network stop the corresponding station from producing further pairs. According to our simulations, one station outputs on average 98,000 $(r, \lceil \log_{\sqrt{2}}(p) \rceil - 55)$-pairs per subinterval and all the pairs of the 8 stations on one chip are routed to the correct destination within about $52,000$ clock cycles. The butterfly network requires $4 \times 16 \times 30$ buffers for 30-bit values (realized as latches) and the routing and control logic. We estimate that 300,000 transistors with an area of less than 1 mm^2 should be sufficient. The output of the butterfly network—16 pairs comprised of $r \bmod 2^{22}$ and the 4-bit encoding of the corresponding $\lceil \log_{\sqrt{2}}(p) \rceil$-value—is sent (across chip borders) to the correct array of size 2^{22} of the collection unit.

Summarizing, the largish stations of Type I can produce the needed progressions for the primes $p > 1.5 \cdot 10^8$ in approximately $52,000$ clock cycles. For this, we need 32 chips, each having a size of ≈ 472 mm^2 and each outputting $16 \cdot 28 = 448$ bit per clock cycle. Each of the outputs has a fixed destination—an array of size 2^{22} in the collection unit where the hit is processed.

Type II. To keep the amount of inter-chip communication at a reasonable level, we introduce a slightly different type of largish stations, which are in charge of all factor base elements with primes $4 \cdot 10^7 < p < 1.5 \cdot 10^8$. These Type II stations are placed on the chips holding the collection unit: Our collection unit will be distributed onto 4 chips, so we need 4 copies of each of these Type II largish stations. As one chip of the collection unit handles only one quarter of the total subinterval of size $S = 2^{26}$, the Type II largish stations are designed for a sieving interval size of 2^{24}. The overall structure is identical to the Type I case just discussed. However, reflecting the reduced subinterval size 2^{24}, the number of (p, r)-pairs per DRAM is reduced to $50,000$. Finally, the calculation of the next hit has to be modified, so that the subsequent three subintervals of size 2^{24} are skipped.

With this strategy, the needed arithmetic progressions for the primes $4 \cdot 10^7 < p < 1.5 \cdot 10^8$ can be generated by $4 \cdot 44$ stations with a total of $4 \cdot 290$ DRAMs of size 0.6 mm^2 each. To route the outputs of these 44 stations *on the same chip* to the correct array of the collection unit, four truncated butterfly networks are used. In each of the two stages of the 16 input network, buffers of size two are sufficient to cope with the 11 inputs per clock cycle on average.

Type III. To handle the primes in the range $1.5 \cdot 10^7 < p < 4 \cdot 10^7$, we use a third type of largish stations. The overall structure is the same as for Type I and II. As for the Type II station, the number of factor base elements per DRAM is 50,000 and Type III stations are placed on the same chips as the collection unit. However, the size of the sieving subinterval handled by Type III stations is reduced to 2^{23}. Consequently, we need in total 8 copies (i. e., 2 per chip) of each Type III largish station, and each of these largish stations is in charge of two arrays of the collection unit.

To process all the primes $1.5 \cdot 10^7 < p < 4 \cdot 10^7$, we use $8 \cdot 16$ largish stations of Type III with $8 \cdot 76$ DRAMs of size 0.6 mm^2 each. The outputs of these 16 stations are sent to the correct one of the two related arrays of the collection unit. A switching unit (butterfly network with depth 1) with 16 inputs and $16 \cdot 8$ buffers can handle this.

Medium Stations

Type I. This type of medium stations is in charge of primes in the range $2^{20} < p < 1.5 \cdot 10^7$. In analogy to the largish stations of Type II, each medium station of Type I handles a sieving subinterval of size 2^{24}. Consequently, there are four copies of each medium station of Type I—one on each chip of the collection unit. In total, each chip of the collection unit hosts 20 medium stations of Type I, where each stations is equipped with $4.0 \cdot 10^6$ bit of DRAM. The first prime hitting the respective subinterval is stored in full, and for the remaining primes a simple difference coding is used. Storing the difference between successive primes instead of the primes itself allows us to reduce the memory required for a factor base element to 44 bit. On average, from each DRAM, two factor base elements are read per clock cycle.

For each of the respective primes, all relations within the subinterval of the chip (of size 2^{24}) are calculated using several adders, and the hits are reported to the relevant array of the collection unit. Additionally, the corresponding hit in the next subinterval of the device (of size $S = 2^{26}$) is produced and written back into the DRAM. To perform this operation, along with a (p, r)-pair the value $p \cdot \lfloor S/p \rfloor$ is stored in the DRAM. Applying a difference coding as for the p values, 12 bit suffice for encoding $p \cdot \lfloor S/p \rfloor$—this includes a flag to indicate a new "starting value". To implement the arithmetic for updating the r-values, two adders (with inputs of (7/24) and (11/24) bit) are used that derive the p- and $p \cdot \lfloor S/p \rfloor$-value from the difference encoding, and two 24-bit adders are used to update the r-value for the subsequent sieving interval of size $S = 2^{26}$. As we want to process the factor base elements at a rate of *two* pairs per clock cycle, for each station, we need two quadruples with the mentioned adders. They perform the necessary update within one clock cycle (in a pipeline structure). In total, we estimate the logic for the updating to require no more than $10,000$ transistors per DRAM bank. Together with the $4.0 \cdot 10^6$ bit of DRAM, this amounts to a silicon area of $\approx 0.83\,\mathrm{mm}^2$.

The adders mentioned so far are only in charge of updating the DRAM entries. To determine the hits within the subinterval of size 2^{24} handled by a station, the (p, r)-pairs of each station travel, through two cyclic buses, along a chain of 8 adders (of width 24 bit). The first free adder removes the pair from the bus and calculates all the hits of p in the current subinterval of size 2^{24}. The buses of two adjacent chains of 8 adders are connected; if the workload of the two adder chains is not balanced, (p, r)-pairs will change the station. On average, we expect medium stations of Type I to emit 32 pairs $(\lceil \log_{\sqrt{2}}(p) \rceil, r)$ per clock cycle (and a maximum of 40). The outputs of two adjacent stations (at a maximum 16 per clock cycle, on average 12) are sent not only to the correct array of the collection unit, but even to the correct quarter of it. This is performed by a butterfly network with 16 inputs; in each node of the network 6 buffers are enough to cope with the inputs.

Type II. The arithmetic progressions for the factor base elements (p, r) with $2^{13} < p < 2^{20}$ are stored in a similar way as in the medium stations of Type I. However, for the medium stations of Type II, *two* DRAM banks of the same size as before are used to store the $\approx 162,000$ pairs representing the first hit *within each subinterval of size* 2^{24}. The update into the next sieving interval (of size S) is realized in the same way as for the Type I stations. Differing from the handling of the primes $> 2^{20}$, however, only the pair (p, r) for the first hit in each array of the collection unit is sent to the collection unit. The other hits are calculated there, i. e., within the collection unit.

Smallish Stations

For each array of the collection unit, the pairs (p, r) with primes $p < 2^{13}$ are stored in a separate DRAM together with the value $\ell \cdot p$, so that $r + \ell \cdot p$ or $r + \ell \cdot p + p$ is the first hit in the next subinterval of size $S = 2^{26}$, and two more numbers for an update to the next row of the array (interval size 2^{17}) and to the next processor (interval size 2^{12}). The update to the next sieving interval is performed with one adder within 3 clock cycles and the first hit for the subinterval is sent to the array of the collection unit for further processing in the same way as the primes $2^{13} < p < 2^{20}$ handled by the medium stations of Type II. We have four smallish stations on each chip, and they easily fit on a silicon area of 0.4 mm^2; there are some 2050 smallish primes. In total, one smallish station requires no more than 1,500 transistors and $2.9 \cdot 10^6$ bit of DRAM. It fits on a silicon area of ≈ 0.06 mm^2.

4.2 Collection Unit for the 1024-bit Case

The main part of the collection unit consists of 128×128 processors, each in charge of a subinterval of the sieving region of size 2^{12}. This set of processors is split into 16 arrays of 32×32 processors, and distributing the collection unit onto four chips means to place four of these arrays on each chip. The processors within an array are connected through horizontal and vertical buses to transport the $\log_{\sqrt{2}}(p)$-approximations and the index r to the processor in charge. In addition, an algebraic/rational flag is needed, so that we know which of the two counters per sieving location is to be be updated. Each processor stores the algebraic and rational counters in a DRAM holding 2^{12} words of 20 bit each—10 bit for the algebraic and 10 bit for the rational counter.

Array Structure

As in Section 3.1, we refer to the individual processors within an array as *counting units*. The counting units within one array are connected through vertical and cyclic horizontal buses, basically as indicated in Figure 2. More specifically, in each column of the array, we place one vertical bus, that is running top to bottom for columns with an even number and bottom to top otherwise.

Handling data of largish stations. At the top of each of the 32 columns we have an input unit that is connected to one of the 32 chips holding largish stations of Type I. The input unit translates received $(r, \lceil \log_{\sqrt{2}}(p) \rceil - 55)$-values into pairs $(r, \lceil \log_{\sqrt{2}}(p) \rceil)$. Moreover, the two pairs at top of column $2 \cdot i$ and $2 \cdot i + 1$ (for $0 \leq i < 16$) are exchanged if the distances of both pairs to their target row are larger than 15. The resulting values are put onto the vertical buses.

The outputs of the largish stations on the same chip (Type II and III) are put onto the vertical buses after/before row 16. These outputs (on average 20 per clock cycle; 24 as a maximum) are put onto a bus, so that the distance to the target row is at most 16. The pairs are stored in a buffer of size 4 if the appropriate bus is not free. If the buffer is full, a "panic flag" stops the corresponding node of the butterfly network to produce outputs. According to simulations, a panic flag is set in some 2000 cases and delays the output of the largish stations of Type II and III by a few hundred clock cycles.

The target address of the packets on the vertical bus are compared with the actual row number and removed from the vertical bus if they are equal. The $(r, \lceil \log_{\sqrt{2}}(p) \rceil)$-values are then transferred to one of the two cyclic horizontal buses running in opposite directions. Using a buffer of size 4 here seems to be sufficient (in our simulations, less than 0.4 pairs were lost in a sieving interval of size $S = 2^{26}$). The counting unit reads the addresses on both horizontal buses, transfers the pair to its own buffer and removes it from the bus, if a packet has reached its target processor, i. e., the correct counting unit. If there is no space left in the buffers of the processor, it is possible to leave the entry on the bus—it will return to the same position after 32 clock cycles.

Handling data of medium stations. The progressions of the medium stations are input at the left and right side of the array, directly into the correct row. The routing to the correct row is performed by an extra structure, adjacent to the array.

- Progressions output by the medium stations of Type I are sent to the correct quarter of the array by the station. In each quarter of the array (8 rows) at most 10 inputs arrive (8 on average) per clock cycle. On either side of the array, an 8 input/8 output butterfly network distributes 5 inputs to the correct row.
- Progressions output by medium stations of Type II are stored in two DRAMs, one on the left and one on the right side of the array. On either side, two 48 bit buses transport

$(p, r, \lceil\log_{\sqrt{2}}(p)\rceil)$-values to the correct rows. Along each of theses buses, in each row, the data is forwarded unchanged to the next row if the target of the r-value is not in this row. If the data has reached a suitable row, it is checked if $p > 2^{17}$ (then, there is only one hit per row). In this case, $(r, \lceil\log_{\sqrt{2}}(p)\rceil)$ is sent to its destination via one of two horizontal "medium prime buses", and $(p, r+p, \lceil\log_{\sqrt{2}}(p)\rceil)$ is forwarded to the next row. The pairs for primes $p < 2^{17}$ are transferred to the adder unit of this row to produce all hits within this row and feed them into the array. When the adder unit has finished with the prime p, the data is forwarded to the next row through one of the two vertical buses.

Handling data of smallish stations. The hits for smallish primes are counted in a separate array of processors and DRAMs: The slow access time of DRAM (6 clock cycles) does not allow to store all hits of a subinterval of size 2^{12} in one DRAM. Therefore we double all DRAM counters, so that while processing the smallish progressions for the current sieving interval, the medium and largish progressions of the next sieving interval can already be processed in the other DRAM bank. We switch the role of the two DRAM banks for each sieving interval, so that effectively the smallish progressions are always "one sieving interval ahead". More specifically, instead of one array of 32×32 processors, we now have two such arrays, which are merged so that each processor in one array is adjacent to one of the other array. One of these arrays contains the logic needed for handling the smallish primes. In each row of this array, the 16 processors on the left side of the array are connected through one cyclic bus with one input node at the left side. The same connection is established for the right half of the processors of this row.

The smallish primes are split into two types (Type I: $1024 \leq p < 2^{13}$ and Type II: $p < 1024$). Both types are stored together in one DRAM as described in Section 4.1. For each p, we also store the value $\lceil\log_{\sqrt{2}}(p)\rceil$. The data is distributed to the left and right half of the array and sent on either side of the array through a vertical (56 bit) bus to 32 *progression generators*. All but the first of these progression generators calculate $r_0 := r + \lfloor 2^{17}/p \rfloor \cdot p$. If $r_0 > 2^{17}$, then $(r_0 \pmod{2^{17}}, p)$ is the first pair to be reported in the row of this progression generator, otherwise p is added to r_0 to obtain the first element to be reported in this row. This value is used within the actual row and in addition forwarded to the progression generator of the subsequent row. For smallish primes of Type I, each progression generator calculates the first hit in each processor (using a 12 bit adder and the value $\lfloor 2^{12}/p \rfloor \cdot p$) and sends the triple $(p, r, \lceil\log_{\sqrt{2}}(p)\rceil)$, along with an algebraic/rational flag and a 1-bit flag indicating the type of the smallish prime, through a cyclic 36 bit bus to the processors of its half of its row. All the progressions of primes of Type II within each half of one line are generated by the corresponding progression generator, and the value $(r, \lceil\log_{\sqrt{2}}(p)\rceil)$ along with the algebraic/rational flag is sent to the target processor through the horizontal 36 bit bus.

Each target processor stores the reported $(r, \lceil\log_{\sqrt{2}}(p)\rceil)$-pairs of Type II in one of its 4 buffers and adds the $\lceil\log_{\sqrt{2}}(p)\rceil$-values into its DRAM. For smallish primes of of Type I, on average every 16 clock cycles a hit can be reported, and these $\lceil\log_{\sqrt{2}}(p)\rceil$-values are added to the DRAM with higher priority than for the smallish primes of Type II. Therefore a buffer of size 2 is sufficient for the smallish primes of Type I.

Area Estimate

Each counting unit requires ≈ 2800 transistors for the largish and medium sized primes and ≈ 1500 transistors for the smallish primes plus two times 82,000 bit of DRAM. The input units for the largish primes require some 1250 transistors per column of one array, and the units for the input of the medium primes 8750 transistors per row. To generate the smallish primes, in addition, some 4400 transistors per row are necessary. Thus, the total area of one array of 32×32 counting units is approximately 26 mm^2 for the medium and largish primes and 22 mm^2 for the smallish primes. Summarizing, each of the four chips holding collection units has a size of 493 mm^2 and consists of:

– 44 largish stations of Type II (180 mm^2),
– $2 \cdot 16$ largish stations of Type III ($2 \cdot 46$ mm^2),
– 20 medium stations of Type I (20 mm^2),
– $4 \cdot 2$ medium stations of Type II ($4 \cdot 2$ mm^2),
– $4 \cdot 1$ smallish stations ($4 \cdot 0.06$ mm^2),
– 4 arrays of collection units ($4 \cdot 48$ mm^2).

One subinterval of size $S = 2^{26}$ is processed within 53,000 clock cycles.

4.3 Combination of the Chips for the 1024-bit Case

One complete sieving device is comprised of 36 chips: 32 chips (each of size 472 mm^2) holding the largish stations of Type I plus 4 chips (each of size 493 mm^2) hosting the collection unit. Each chip holding largish stations of Type I, per clock cycle sends 16 pairs (with 28 bit each) to one of the 16 arrays of counting units distributed over the four chips holding the collection unit. The collection unit as a whole, i. e., totaling all four chips, receives $4 \cdot 32$ pairs (3584 bit) per clock cycle. The 36 chips can be placed in a regular, grid-like structure, so that the maximum distance any pair has to travel is 5 times the distance between adjacent chips. Implementing this communication across chip borders is non-trivial, but does not appear utopian. The necessary wiring still seems significantly easier to realize than SHARK's transport system [FKP+05]. Finally, as we do not store factors found during sieving, the sieving reports output by our device are fed into an ECM engine as described in [GJK+06]. In this way, the needed norm factorizations can be obtained without affecting the sieving time in a relevant manner. Including one ECM chip for computing and factoring the norms, the silicon area needed for one complete device is about 172 cm^2.

One sieving line is split into $16.4 \cdot 10^6$ subintervals of size 2^{26} and at a clocking rate of 600 MHz can be processed in less than 25 minutes. The time needed for switching to the next sieving line, i. e., loading new pairs into the DRAMs, requires some 0.035 seconds and is negligible. Similarly, the time needed for outputting the (candidate) relations identified in the completed sieving line is not significant. Using the same 33% saving as in TWIRL [ST03, Appendix A.5], with 8300 of the above devices, the sieving step for a 1024 bit number can be expected to be completed within one year. Comparing the sieving time/chip area of our design and TWIRL, we see that our device requires by a factor of 3.5 more silicon area than TWIRL. Unlike TWIRL, however, our design is not wafer-scale.

Optimizing parameters. More research is needed for finding optimal parameters for our design: For instance, after a simple modification, the largish units of Type I can output the pairs of two DRAM banks (two consecutive subintervals of size 2^{26}) within $52,000$ clock cycles. If the four chips holding the collection units are doubled, the silicon area for this modified device increases by roughly 20 cm^2 and halves the processing time. Using this simple modification, the sieving for a 1024 bit number can be expected to be completed within one year using only 2.0 times the silicon area of TWIRL.

5 Conclusion and Future Work

The hardware design proposed above uses only chips of moderate size (493 mm^2 and 472 mm^2) without paying for this in a drastic loss of performance: Compared to TWIRL, only a factor 2–3.5 in performance is lost. The inter-chip communication required is non-trivial, but still seems doable and easier to realize than the transport system for SHARK. Thus, from a practical cryptanalytic point of view, the new design seems to deserve a more detailed exploration.

So far we did not explore the cost of a prototype for, say, 512 bit or 768 bit numbers, which seems a worthwhile next step. The results achieved so far also seem to justify a closer look at our design when allowing more advanced fab technology, say involving a 90 nm process.

References

[BCP97] Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *Journal of Symbolic Computation*, 24:235–265, 1997.

[Ber01] Daniel J. Bernstein. Circuits for Integer Factorization: a Proposal. At the time of writing available electronically at `http://6 k2a 02g rg. olli eefood.rest/papers/nfscircuit.pdf`, 2001.

[FKP+05] Jens Franke, Thorsten Kleinjung, Christof Paar, Jan Pelzl, Christine Priplata, and Colin Stahlke. SHARK: A Realizable Special Hardware Sieving Device for Factoring 1024-Bit Integers. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems; CHES 2005 Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 2005.

[GJK+06] Willi Geiselmann, Fabian Januszewski, Hubert Köpfer, Jan Pelzl, and Rainer Steinwandt. A Simpler Sieving Device: Combining ECM and TWIRL. In *Proceedings of ICISC 2006*, volume 4296 of *Lecture Notes in Computer Science*. Springer, 2006. Preprint available at `http://      ugw9  2emm  . olli eefood.rest/2006/109`.

[GKST05] Willi Geiselmann, Hubert Köpfer, Rainer Steinwandt, and Eran Tromer. Improved Routing-Based Linear Algebra for the Number Field Sieve. In *Proceedings of ITCC '05 – Track on Embedded Cryptographic Systems*. IEEE Computer Society, 2005.

[GS03a] Willi Geiselmann and Rainer Steinwandt. A Dedicated Sieving Hardware. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 2003.

[GS03b] Willi Geiselmann and Rainer Steinwandt. Hardware for Solving Sparse Systems of Linear Equations over GF(2). In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems; CHES 2003 Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 51–61. Springer, 2003.

[GS04] Willi Geiselmann and Rainer Steinwandt. Yet Another Sieving Device. In Tatsuaki Okamoto, editor, *Topics in Cryptology — CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2004.

[GSST05] Willi Geiselmann, Adi Shamir, Rainer Steinwandt, and Eran Tromer. Scalable Hardware for Sparse Systems of Linear Equations, with Applications to Integer Factorization. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems; CHES 2005 Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2005.

[IKOS06] Tetsuya Izu, Noboru Kunihiro, Kazuo Ohta, and Takeshi Shimoyama. Analysis on the Clockwise Transposition Routing for Dedicated Factoring Devices. In Jooseok Song, Taekyoung Kwon, and Moti Yung, editors, *Information Security Applications: 6th International Workshop, WISA 2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2006.

[LHWL93] Arjen K. Lenstra and Jr. Hendrik W. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer, 1993.

[LS00] Arjen K. Lenstra and Adi Shamir. Analysis and Optimization of the TWINKLE Factoring Device. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2000.

[LSTT02] Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, and Eran Tromer. Analysis of Bernstein's Factorization Circuit. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2002.

[LTS+03] Arjen K. Lenstra, Eran Tromer, Adi Shamir, Wil Kortsmit, Bruce Dodson, James Hughes, and Paul C. Leyland. Factoring Estimates for a 1024-Bit RSA Modulus. In Chi-Sung Laih, editor, *Advances in Cryptology — ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 55–74. Springer, 2003.

[Sha99] Adi Shamir. Factoring Large Numbers with the TWINKLE Device. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems. First International Workshop, CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 1999.

[ST03] Adi Shamir and Eran Tromer. Factoring Large Numbers with the TWIRL Device. In Dan Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2003.