SHARK
A Realizable Special Hardware Sieving Device for Factoring 1024-bit Integers

Jens Franke[1], Thorsten Kleinjung[1], Christof Paar[2], Jan Pelzl[2],
Christine Priplata[3], Colin Stahlke[3]

[1] University of Bonn, Department of Mathematics, Beringstraße 1, D-53115 Bonn, Germany
{franke,thor}@math.uni-bonn.de
[2] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{cpaar,pelzl}@crypto.rub.de
[3] EDIZONE GmbH, Siegfried–Leopold–Straße 58, D-53225 Bonn, Germany
{priplata,stahlke}@edizone.de

Abstract

Since 1999 specialized hardware architectures for factoring numbers of 1024 bit size with the Generalized Number Field Sieve (GNFS) have attracted a lot of attention ([Ber], [ST]). Concerns about the feasibility of giant monolytic ASIC architectures such as TWIRL have been raised. Therefore, we propose a parallelized lattice sieving device called SHARK, which completes the sieving step of the GNFS for a 1024-bit number in one year. Its architecture is modular and consists of small ASICs connected by a specialized butterfly transport system. We estimate the costs of such a device to be less than US$ 200 million. Because of the modular architecture based on small ASICs, we claim that this device can be built with today's technology.

Keywords: integer factorization, lattice sieving, RSA 1024 bit, special hardware.

1 Introduction

The Generalized Number Field Sieve (GNFS) is asymptotically the best known algorithm to factor numbers with large factors. In practice it seems to be the best algorithm for both software and hardware for factoring 1024-bit numbers, such as they appear in RSA based cryptographic protocols. The GNFS has two expensive parts: the sieving part and the matrix step. This paper describes SHARK, a specialized hardware architecture which completes the sieving step of the GNFS for a 1024-bit number in one year. It is much cheaper than general purpose hardware that solves the same problem (e.g. personal computers). The architecture consists of 2300 identical isolated machines sieving in parallel. In the following we describe one of these machines.

We estimate the costs of one machine to be US$ 70 000. It uses lattice sieving. The actual sieving is done in very fast accessible memory ("cache"). If this memory would be extremely cheap, we could construct a machine that sieves in some extremely large memory chip. Since this kind of memory is expensive we only use 32 MB of sieving cache memory.

The sieving area is split into many small parts such that each part fits in the sieving cache. After the sieving of one small part is completed, the machine moves on to the next part until the whole sieving area has been scanned.

The tricky part is to sort the sieving contributions such that all sieving contributions for a certain part are loaded into the sieving cache just before the sieving of that part starts. To achieve this, the data produced by the lattices corresponding to the larger primes of the factor base are sent through a specialized transport system with butterfly topology.

The output of the sieve consists of potential sieving reports that still need to be checked for smoothness. This is done (after a quick compositeness test) by special hardware devices using the Elliptic Curve Method (ECM). The algorithm has been adapted for hardware implementations (see [FKPPPSS]). The use of ECM in special hardware is preferable for lowering the costs of the machine. However, in this paper we use a choice of parameters with a moderate ECM support in order to focus on the sieving part of the machine. There are better choices with much more ECM, as indicated at the end of Section 3. Notice that the importance of using special hardware for factoring the potential sieving reports grows with the bit length of the number to be factored.

The estimated costs of computing power for factoring 1024-bit numbers have been derived from software experiments. Together with the experience from recent factoring records in software (see [RSA576]), this leads to a realistic choice of parameters and good estimates for the amount of computing power and storage needed by each part of the machine.

Section 2 summarizes the necessary background on the GNFS and, in particular, on lattice sieving. It also discusses parameter choices. The SHARK architecture is introduced in Section 3 and an overview of the whole machine is given. A detailed description of the hardware modules and a cost estimate is presented in Section 4. We finish with some conclusions and remarks in Section 5.

2 The General Number Field Sieve and Lattice Sieving

In GNFS we are given two homogeneous polynomials $F_i \in \mathbb{Z}[X, Y]$, $i = 1, 2$, satisfying certain conditions. The task of the sieving step is to collect sufficiently many coprime pairs of integers (a, b), $b > 0$, such that both integers $F_i(a, b)$ decompose into prime factors smaller than a given bound L. Such pairs (a, b) are also called relations. The number of relations needed depends on the bound L. Collecting $2\pi(L) \approx \frac{2L}{\log L}$ relations is usually far more than enough. For more details on GNFS see [LL].

The collection of relations is usually done by a combination of a sieving technique and a method for factoring smaller numbers, e.g. ECM or MPQS. For this purpose we choose two factor bases $\mathcal{F}_i$ each consisting of pairs (p, r), where $p < B_i$ is a prime and r an integer such that p divides $F_i(a, b)$ whenever $p \mid a - br$. The sieving technique identifies pairs (a, b) such that both values $F_i(a, b)$ are divisible by many primes $< B_i$. The cofactors ($F_i(a, b)$ divided by all prime factors $< B_i$) are subsequently handled by a factoring method for small numbers. If both decompose into prime factors $< L$ a relation is found.

Our proposed sieving device will carry out the collection of relations by lattice sieving in the way described in [FK]. Let the dimensions of the sieving rectangle be $I \times J$ and let (q, s) be a special q. That means that we consider the lattice associated to (q, s), calculate a reduced basis (a_1, b_1), (a_2, b_2) and define the sieving rectangle to consist of the points $i(a_1, b_1) + j(a_2, b_2)$ for $-\frac{I}{2} \leq i < \frac{I}{2}$ and $0 < j \leq J$.

The factor base elements (p, r) with $p \leq I$ have to be adapted to the lattice given by (q, s), yielding $(p, \tilde{r})$. Then we proceed with $(p, \tilde{r})$ the same way as in line sieving. The factor base elements (p, r) with $p > I$ are handled differently. First, we transform the elements to obtain vectors v and w which allow us to quickly identify the points of the intersection of the sieving rectangle and the lattice corresponding to (p, r). This is done by starting at the point $(0, 0)$ and continuing from there by a sequence of additions of either v or w or $v + w$ by a simple rule as described in [FK]. At each of these locations we have to add a contribution of $\log p$ to the sieving array. We are interested in those points of the sieving array where the sum of all contributions is bigger than some bound.

In GNFS we have to perform two sieves, an algebraic sieve and a rational sieve. Moreover, we perform a trial division sieve which is a modification of [GLM] described in [FK].

For estimating the costs of a factorization of a 1024-bit number we use the following parameters which are based on the polynomial pair of degree 5 and 1 of [ST]. The factor base bounds are $B_1 = 4 \cdot 10^{10}$ on the algebraic side ($1.7 \cdot 10^9$ prime ideals) and $B_2 = 2 \cdot 10^{10}$ on the rational side ($9 \cdot 10^8$ prime ideals). The size of the sieving rectangle is $2^{20} \times 2^{19}$. If a point of the sieving rectangle passes both sieves and both cofactors are at most 2^{125}, we check for smoothness (and aborting as soon as it fails) by quick compositeness tests and ECM. If this is successful and all factors are at most $L = 2^{42}$ we obtain a relation. We will do lattice sieving for all $3.7 \cdot 10^9$ special q in $[4 \cdot 10^{10}, 1.33 \cdot 10^{11}]$ which we estimate to yield $2.7 \cdot 10^{11}$ relations. The last number was obtained by integrating smoothness probabilities over sieving rectangles. In the whole process, about $1.7 \cdot 10^{14}$ numbers are processed by ECM.

If one desires a smaller matrix, more relations are needed. In this case, we propose to do lattice sieving for all $4.4 \cdot 10^9$ special q in $[4 \cdot 10^{10}, 1.5 \cdot 10^{11}]$ which we estimate to yield $3.1 \cdot 10^{11}$ relations. This increases the number of machines needed for the sieving from 2300 to 2800.

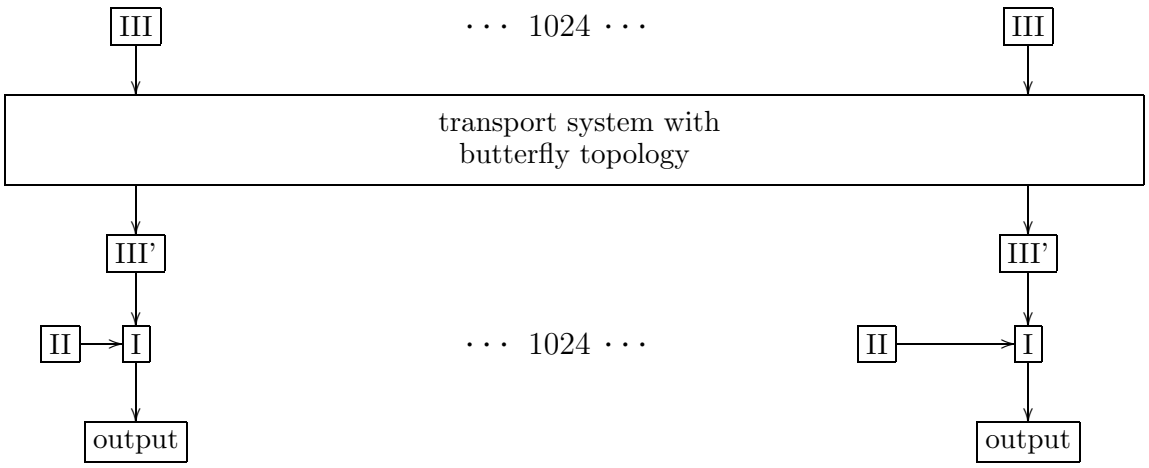
As for cost estimates for other sieving devices the costs will be reduced if one spends more effort in finding a good polynomial pair.

3 SHARK – Architectural Overview

The SHARK machine consists of parts I, II, III and a transport system (see Figure 1). The sieving area is split into small parts consisting of 2^{14} lattice points. For the sieving process one byte per point has a sufficient precision to sum up the logarithms of the primes. Therefore, sieving one part is done in 16 kB of fast accessible memory (comparable to the first level cache of a general purpose CPU).

We split the factor base into small, medium and larger primes which will be dealt with in the three different parts of the machine. Part III of the machine takes care of the larger primes, extracts the necessary data for the sieving process and sends it through a specialized transport system with butterfly topology. The transport system sends the data only to that part of the machine where it is needed. Part III has 1024 small units working in parallel, each dealing with just 1/1024 of the sieving area. Therefore, the transport system has 1024 inputs.

Figure 1: High-Level Schema of the SHARK Sieving Machine



Part II of the machine processes the medium primes. Since the lattices corresponding to these primes are much denser, their data do not need to be sent to all parts of the machine, but can be sorted locally. As visible in Figure 1, part II consists of 1024 small parts, each dealing locally with a small part of the sieving area. These 1024 parts do not communicate among each other.

Part I of the machine consists of 1024 small local units that do not communicate among each other. It generates the very dense lattices for the small primes of the factor base and sieves with this data on 2^{14} lattice points. Additionally, part I collects the sieving data from part II and part III that are necessary for the sieving on the 2^{14} lattice points and sieves with these data. The survivors of this small part of the sieving area are potential sieving reports, and they are sent as output to an ECM unit to be checked for smoothness. Then, part I turns to the next 2^{14} lattice points.

Within one year, 2300 such machines will output about $1.7 \cdot 10^{14}$ potential sieving reports that need to be tested for smoothness, e.g. with the Elliptic Curve Method (ECM). This could be done by conventional PCs within the required time.

Changing the parameters of SHARK and using special hardware for ECM (see [FKPPPSS]) we can save up to 50% of the costs for SHARK. E.g. increasing the bound for cofactors from 2^{125} to 2^{163} we only need 1300 machines producing $1.3 \cdot 10^{16}$ potential sieving reports to be processed by ECM.

4 Description of the SHARK Modules

The key to the modular architecture is the partitioning of the sieving area and of the factor base. This algorithmic aspect of the sieving is explained in the first subsection, whereas the three parts of the machine (I, II and III), reflecting the partitioning of the factor base, are described subsequently.

4.1 Sieving

In GNFS we have to perform two sieves: an algebraic sieve and a rational sieve. Notice that we do not need to choose a linear polynomial, the following will also work with two polynomials of degree > 1. These two sieving tasks are almost identical except that for the second sieve we only consider the surviving points of the first sieve. Since we want to know the factorizations of the polynomial values for the surviving points, we also perform a trial division sieve to recover the factors found by the sieves.

We divide a sieving task into three phases: the generation of sieving contributions, the actual sieving, and the evaluation of the sieving area. The first phase is the generation of triples $(p, \log p, e)$, where p is a prime, $\log p$ the (scaled) logarithm of p, and e a position in the sieving area. If a prime ideal has a contribution to a sieve location, a corresponding triple is produced. In the second phase the contributions are summed up. A sieving array is initialized by zero and for each triple, $\log p$ is added at position e, i.e., for each e the sum

$$\sum_{(p_i,\log p_i,e_i) \text{ with } e_i=e} \log p_i$$

is calculated. The evaluation phase isolates those sieving locations where the contribution exceeds a certain bound (also depending on the location). For these survivors we can perform a trial division sieve, creating for each survivor a list of its prime divisors, in the following way. We clear the sieving array and fill the positions of the survivors with different identifiers $(1, 2, 3, \ldots)$. Afterwards, for each triple the prime p is stored in the list given by the identifier at position e (if the identifier is not zero). Note that the generation of triples is only done once while they are used twice: $\log p$ and e for the actual sieving and p and e for the trial division sieve.

We will use lattice sieving which means that we often change the lattice corresponding to a special q. At every change we have to carry out initializations for all elements of the factor base (see [FK]). These initializations amount to roughly one inversion and one half of an extended gcd per factor base element. They are done locally at the places of the machine where the factor base elements are stored. The machine is divided into (roughly) three parts: Part I deals with the small elements (p, r) of the factor base $(1 < p < 2^{14})$, part II with the medium elements and part III processes the large elements $(2^{22} < p)$.

We now describe the general structure of the components of the machine and their interaction over time. Our sieving area has size $2^{20} \times 2^{19}$. Since we omit those pairs for which both coordinates are even we will sieve over three subareas of size $2^{19} \times 2^{18}$. We divide these subareas into 32 parts, each of size $2^{19} \times 2^{13}$. These are called *ranges* and have the

following meaning: During a certain period of time all parts of the machine with the exception of part I will prepare data for the algebraic sieve for the n-th range. In the next period of time these parts will do the same for the rational sieve for the n-th range while part I will complete the algebraic sieve for range n using the data prepared in the previous period of time. The rational sieve for range n will be completed in the next period of time by part I while the other parts prepare data for the algebraic sieve for range $n+1$ etc. Hence there is a need to buffer the prepared data for two sieves over a range.

Each range is divided into 1024 parts of size $2^{19} \times 2^3$ which we will call *domains*. There are also 1024 identical parts of the machine (one for each domain), which will handle sieving contributions of prime ideals of type I and II. The contributions of prime ideals of type III are processed in a different way. These prime ideals are split into 1024 parts and for each part all contributions for a range are prepared and sent to the correct part of the machine. This sorting will be done by a transport system with butterfly topology.

Sieving for a domain is done in 256 steps handling 2^{14} points each. For this purpose, data for prime ideals of type II and III (which have to be stored anyway) are written to the correct array out of 256 arrays. Data for prime ideals of type I are generated on the fly and combined with the data from the corresponding array.

We now describe the individual parts in more detail.

4.2 Part III

This part generates triples for prime ideals of type III. It consists of 1024 identical units each containing 64 MB DRAM and a generation unit. The DRAM is used to hold the factor bases and related information. For each element (p, r) of the factor bases we store an 8-tuple $(p, r, \log p, v_x, v_y, w_x, w_y, e)$ where $\binom{v_x}{v_y} = v$ and $\binom{w_x}{w_y} = w$ are vectors used to update the contribution location and e is the next contribution location for this prime ideal. For our choice of parameters we can store such an 8-tuple in 25 byte using 36 bit for p and r, 8 bit for $\log p$, 20 bit for v_x, v_y, w_x, w_y and 40 bit for e.

The generation unit has two tasks. After changing a special q it calculates for each prime ideal the values v_x, v_y, w_x and w_y for this lattice and sets e to the first location where the prime ideal contributes. During the sieving phase it reads all 8-tuples one by one, generates the triples for all locations in the sieved domain where this prime ideal has a contribution, and writes the 8-tuple back to memory (actually only e will change). The generated triples are sent to the transport system.

For the initialization task the generation unit has to perform calculations of the complexity of an extended gcd. The actual generation of triples requires only simple instructions such as conditional additions or load/store operations.

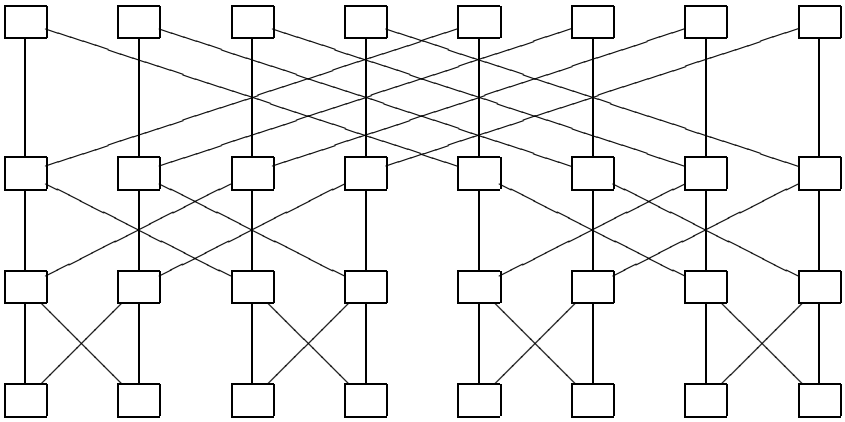
4.3 Transport System

The transport system has 1024 input channels and 1024 output channels. The purpose of the transport system is to deliver each triple $(p, \log p, e)$ from an input channel to a certain output channel determined by 10 bits of e. Triples have a size of at most 80 bit

and may arrive simultaneously at different input channels. We will tolerate a small loss of triples arising from data collisions. For instance, the loss of one triple out of 2^{40} will at most affect one potential sieving report per special q.

We now describe a structure which will comply to the requirements above (see Figure 2). It consists of $11 \cdot 1024$ simple nodes connected in a butterfly topology, i.e., nodes $N_{i,j}$ and $N_{i',j'}$ ($0 \le i, i' < 11$, $0 \le j, j' < 1024$) are connected if $i' = i - 1$ and either $j' = j$ or $j' = j$ xor $2^{i'}$. Data always flow from nodes $N_{i,j}$ with a higher i to those with a lower i. A typical node consists of two input lines where each is connected to a small buffer such that they can simultaneously receive triples, two output lines and a logic which reads a triple from an input buffer, examines a certain bit of e and delivers this triple to the corresponding output line. For the nodes $N_{i,j}$ in the top layer (i.e. $i = 10$) one of the input lines is an input channel of the transport system and the other input line is not connected. The nodes in the bottom layer (i.e. $i = 0$) send all output to one output line which is the output channel of the transport system.

Figure 2: Butterfly Topology of Width 8



For a physical realization it is not necessary to manufacture a separate chip for each node or to strictly adhere to the topology. We might also group several nodes on a chip or implement a different sorting structure as long as the performance is not worse than that of the butterfly topology. Grouping several nodes will also reduce the costs for connecting them.

4.4 Part III'

This part also consists of 1024 identical units each of which handles triples for one domain of the processed range. Each unit connects directly to an output channel of the transport system and receives triples which are to be sorted and stored in a double buffer via a 64 kB cache. The double buffer has a size of $2 \cdot 16$ MB DRAM and each half is used to store triples from prime ideals of type III generated for one domain. They are stored in one of 256 arrays according to 8 bit of e. Since at this stage 18 bit of e are fixed we can omit them and store a triple in 7 byte. Therefore it is possible to store 9300 triples per array which is far more than the expected 7700 triples per array on average.

The two halves of the double buffer are written alternately by this part while the other half is read by part I of the machine (see below).

4.5 Part II

Part II again consists of 1024 identical units. Each unit is responsible for the generation of triples for prime ideals of type II for one domain of the processed range. Since it will generate triples in a line sieve like fashion, it is essentially a simplified version of part III and part III'.

This unit consists of 8 MB DRAM, a generation unit, a sorter and a 64 kB-cached double buffer of size 2·12 MB DRAM. The 0.6 million factor base elements of type II (size between 2^{14} and 2^{22}) can be stored in 8 MB, each using 14 byte. These 14 byte take into account some auxiliary data needed for line sieving and the change from one domain of a range to the corresponding domain in the next range. The generation unit has a slightly easier initialization task than that of part III but the actual generation tasks are comparable. It sends the generated triples to a sorting unit which stores them via a 64 kB cache in one half of the double buffer. For prime ideals of type II 5 byte per triple are sufficient such that each array can hold 9800 triples which is more than the 7400 needed on average.

4.6 Part I

This part again appears in 1024 identical units. Each unit has more complex tasks than the units in the other parts of the machine. It generates triples for prime ideals of type I, adds up these contributions, adds up the contributions from prime ideals of type II and III generated by the other parts of the machine, combines these sums and evaluates them. This process is now described in more detail.

In this part the sieving for a domain will be done in $2 \cdot 256$ steps: first, 256 algebraic sieves, each over an area of 2^{14}, and then 256 rational sieves over the same areas. Each of these sieving steps consists of several phases: first, an initialization of the sieving caches, then the actual summation of the contributions, then an evaluation, and finally the trial divison sieve.

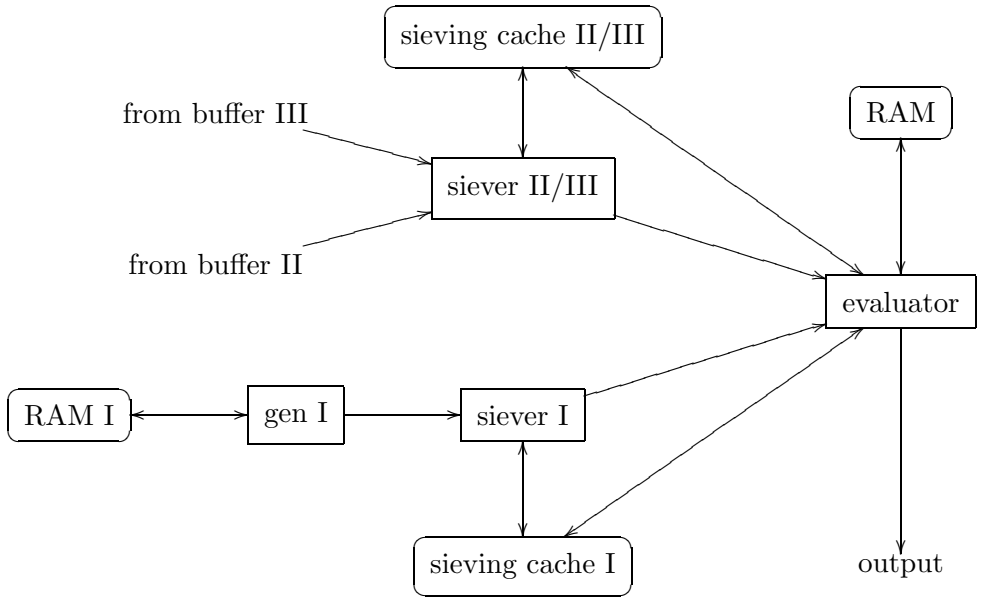
The prime ideals of type I together with auxiliary data are stored in less than 50 kB DRAM. A generation unit comparable to that of part II accesses this memory and generates triples for a sieving area of size 2^{14}. These triples are directly sent to a sieving unit which performs the actual sieving in a cache of 2^{14} byte. Since there is no buffering of the triples they have to be generated a second time during the trial division sieve. The initialization of the cache with zeros is also done by the sieving unit.

At the same time another sieving unit which also controls a cache of 2^{14} byte reads the triples generated by parts II and III of the machine and does the actual sieving in this cache. Since parts II/III and part I are processing on different sieving sides (i.e., algebraic/rational) there will be no conflict in accessing the double buffers. Reading the triples will also be fast since triples for an area of size 2^{14} are stored in one array.

Apart from those units described so far there is a more complex evaluation processor which has 8 MB DRAM. It is also connected to the two sieving units and to their caches (see Figure 3). During the actual sieving phase it computes thresholds for the evaluation phase. After all triples have been processed by the sieving units, the processor evaluates the sieving area by adding up corresponding bytes of the two sieving caches and comparing

the result to a previously computed threshold. Whenever the sum surpasses the threshold the position is marked in both sieving caches, otherwise it is set to zero (in a sieve on the rational side we also set to zero a position which has not survived the corresponding algebraic sieve). When this has been done for the whole area of size 2^{14}, the trial division phase begins. The sieving units read (resp. receive) again triples and send those triples which correspond to a marked position in the sieving area to the evaluation processor which stores them in its DRAM. After a trial division sieve on the rational side has been finished, the evaluation processor outputs the survivors and all data obtained from the trial division sieves for this sieving area.

Figure 3: Block Diagram of Part I



4.7 Cost Estimates

The width of the transport system is crucial for the costs of the whole machine. We first give a simple analysis of the behaviour of costs for varying widths. The total costs consist of the costs for the transport system, the costs for memory and the costs for the ASICs outside the transport system. The third summand remains constant since doubling the width of the transport system will double the number of these ASICs but also halve the time spent for one special q. Furthermore the total memory of the machine remains constant which has the consequence that doubling the width of the transport system will decrease the costs as long as standard memory chips of smaller size get cheaper. Notice that we want to use standard memory chips, because we assume these to be cheaper than customized memory ASICs. The first summand always grows when doubling the width, since a transport system of width 2^{n+1} consists of two systems of width 2^n and its top layer together with all connections of the top layer. There will be a certain width for which minimal costs will be attained. In our design this will probably be bigger than 1024 but this is technically more demanding. Therefore we chose a width of 1024.

Apart from a few PCs for controlling the sieving process and collecting the output, one machine consists of 136 GB DRAM, 160 MB cache and various ASICs. Most of the ASICs only perform quite simple tasks. Only the evaluator needs a considerable area. We estimate that they occupy a quarter of the area of a 300 mm wafer. Even taking into account a whole wafer, the silicon costs including memory are less than US$ 30 000. Doubling this number for overhead (packaging, cooling, ...) and adding US$ 10 000 for the PCs and special ECM hardware we obtain US$ 70 000 per machine. Remark: The cost for the ECM hardware for this choice of parameters is just a few dollars and thus negligible.

At a clock frequency of 1 GHz one machine takes around 20 s per special q such that 2300 machines are needed for $3.7 \cdot 10^9$ special q. This amounts to production costs of US$ 160 million (without development). Considering the ASIC area, we estimate that each machine has a power consumption of at most 30 kW which induces a power bill of US$ 60 million per factorization.

5 Conclusions and Remarks

Conclusions. SHARK appears to be the first proposal for an architecture for sieving a 1024-bit number within a year which is realizable with conventional technology and costs less than a thousand million US$. The main difference to other proposed architectures is (in contrast to a giant monolytic ASIC) its modular design composed of small ASICs connected by conventional data busses. The modularity is achieved by dividing the factor base into several parts and sorting the sieving data with a butterfly transport system. All choices of parameters are a result of intense software experiments with a complete implementation of the GNFS for factoring large numbers.

Remarks. Our architecture permits many reasonable modifications: the size of the transport system could be smaller or larger, the partition of the factor base in three parts could vary, ECM could be used more intensely to permit less sieving, many other parameters could be changed. This permits using the architecture also for other bit lengths. 768-bit numbers can be sieved by a similar architecture. While scaling the system for larger numbers, the role of an efficient hardware (like ECM in ASICs, see [FKPPPSS]) to factorize the cofactors becomes more and more important. The transport system has to become very large and at some point the complexity of the connections between the layers will be practically impossible.

Future work. The efficiency of the machine heavily depends on the different processing of factor base elements of different size. We will analyse different methods for processing very large elements, small prime powers and different classifications of sizes in more than three categories. Some initializations and choices of parameters can still be optimized. A crucial point for the scalability to larger numbers than 1024 bit will be the size of the butterfly transport system. We will investigate different realizations and try to make it larger than 1024 channels. A large butterfly transport system can also be used for solving the matrix in GNFS. We will analyse how to optimize the matrix step in this way and how to lower the size of the butterfly transport system needed for solving the matrix.

References

[Ber] D. J. Bernstein, *Circuits for Integer Factorization: A Proposal*, Manuscript, November 2001. http://cr.yp.to/papers.html#nfscircuit

[FK] J. Franke and T. Kleinjung, *Continued Fractions and Lattice Sieving*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[FKPPPSS] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, C. Priplata, M. Šimka, C. Stahlke, *An Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[GS] W. Geiselmann and R. Steinwandt, *Yet another sieving device*, CT-RSA 2004, LNCS **2964**, Springer, 2004, 278-291.

[GLM] R. A. Golliver, A. K. Lenstra and K. S. McCurley, *Lattice sieving and trial division*, in: Algorithmic Number Theory (ed. by L. M. Adleman, M.-D. Huang), LNCS **877**, Springer, 1994, 18–27.

[LL] A.K. Lenstra and H.W. Lenstra, Jr. (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Math. **1554**, Springer, 1993.

[LTS+] A. K. Lenstra, E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes and P. Leyland, *Factoring Estimates for a* 1024*-bit RSA Modulus*, in: Proc. ASIACRYPT 2003, LNCS **2894**, Springer, 2003, 55-74.

[RSA576] J. Franke, T. Kleinjung et al., *RSA*-576, Email announcement, 2003. http://www.crypto-world.com/announcements/rsa576.txt

[ST] A. Shamir and E. Tromer, *Factoring Large Numbers with the TWIRL Device*, in: Proc. Crypto 2003, LNCS **2729**, Springer, 2003, 1–26. http://www.wisdom.weizmann.ac.il/~tromer/papers/twirl.ps.gz