A Massively Parallel RC4 Key Search Engine

K.H. Tsoi, K.H. Lee and P.H.W. Leong
{khtsoi,khlee,phwl}@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT Hong Kong

Abstract

A massively parallel implementation of an RC4 key search engine on an FPGA is described. The design employs parallelism at the logic level to perform many operations per cycle, uses on-chip memories to achieve very high memory bandwidth, floorplanning to reduce routing delays and multiple decryption units to achieve further parallelism. A total of 96 RC4 decryption engines were integrated on a single Xilinx Virtex XCV1000-E field programmable gate array (FPGA). The resulting design operates at a 50 MHz clock rate and achieves a search speed of 6.06×10^6 *keys/second, which is a speedup of 58 over a 1.5 GHz Pentium 4 PC.*

1 Introduction

Field programmable gate array (FPGA) devices have for many years been identified as a suitable technology for the implementation of general purpose cryptanalysis devices with the potential for improved performance over traditional microprocessor based approaches. To date, there have been much activity in the design of high speed FPGA based encryption devices for embedded applications. However, their application to cryptanalysis has been very limited due to their high cost and low density.

A brute force key search can be used to determine the key used to encrypt a message by trying every possible key to decrypt the message. Such a key search is trivially parallelizable and successful key searches using loosely coupled microprocessors in a distributed computing approach have successfully been applied to the 56-bit DES algorithm and the 56-bit RC5 algorithm.

Application specific integrated circuits (ASICs) have also been used by the Electronic Frontier Foundation (EFF) to implement a DES key search engine, called "Deep Crack", which could search 88 billion keys per second [1]. The machine solved the "Blaze Challenge" and the RSA Laboratories DES–III challenge, the latter on January 1999 in 22 hours [2]. One limitation of an ASIC based implementation is that they are hardwired for specific problems.

Recently, FPGA based devices have been shown to offer much higher performance for encryption than microprocessor based implementations (e.g. [3, 4, 5, 6]). Thus, it would appear that they would make excellent key search engines. The main problem to date is that, although many organizations have access to hundreds or more personal computers (PCs) and workstations, FPGA boards are specialized equipment and equivalent FPGA resources are not generally available. Another issue is that capacity limitations of the FPGA serve to limit the amount of parallelism which can be achieved on the device. This, coupled with the fact that the clock frequencies of modern PCs are approximately one order of magnitude higher than that of a typical FPGA design, made their performance benefits limited.

It is our contention that FPGAs are approaching a level of density and integration where the above issues are being resolved, and that they are practical for brute force key search applications. In this paper, we present an implementation of the alleged RC4 cipher which achieves significant performance improvement over a microprocessor implementation. RC4 is used for encryption in products such as the secure sockets layer (SSL) protocol, the secure shell (SSH) protocol, the wired equivalent privacy (WEP) algorithm (part of the IEEE 802.11b wireless LAN security standard), Lotus Notes, Oracle Secure SQL, Microsoft Office and Adobe Acrobat (Acrobat 4.x or older). Furthermore, in the past, the key size was often limited to 40 bits due to US export restrictions.

There have been two previously reported FPGA based RC4 key search machines. In 1996, Goldberg and Wagner proposed an RC4 search engine using an

Altera RIPP10 board which had 8 FLEX8000 chips and four static RAM chips [7]. Their design could perform 4 parallel searches and each unit required 1286 cycles per key. Kundarewich *et. al.* proposed a key search engine using a single Altera EPF10K20 complex programmable logic device (CPLD). In their implementation, each search unit required 1304 cycles per key and 5 parallel searches could be made at 10 MHz [8].

The RC4 implementation described in the paper integrates the key search controller and 96 parallel RC4 decryption engines on a single Xilinx Virtex XCV1000E FPGA (much larger FPGA devices are already available). Although the RC4 implementation operates at a clock frequency which is an order of magnitude lower than that of the latest microprocessors, the FPGA implementation achieves a significant speedup due to the following features:

- Parallelism in the implementation of the RC4 core allows several operations to be completed in a single cycle.
- On-chip resources were used to achieve a very low latency, high bandwidth memory interface
- The memory used was dual-ported, allowing for higher memory transfer efficiency.
- Floorplanning was used to minimize interconnect delays
- A large number of the decryption cores were used in parallel.

In our implementation, each search unit requires approximately 800 cycles per key and 96 such units are integrated on a single FPGA.

The rest of the paper is organized as follows: in Section 2, the RC4 and key search algorithms are described. Section 3 describes the architecture of the RC4 implementation. Implementation details are presented in Section 4, and performance measurements are presented in Section 5. Finally, conclusions are drawn in Section 6.

2 Algorithms

2.1 RC4

RC4 is a stream cipher designed by Ron Rivest and was originally proprietary to RSA Data Security [9]. The algorithm was leaked anonymously to the Cypherpunks mailing list in 1994. The RC4 algorithm generates a key dependent pseudorandom number sequence of arbitrary length.

In the description below, two 256 byte arrays are used, namely the *K-block*, K and the *S-block*, S. Note that the *K-block* does not change during the encryption process.

The RC4 algorithm can be divided into 2 phases: a key scheduling phase and the pseudorandom number generator (PRNG) phase. Both phases must be performed for every new key.

In the key scheduling phase, a scrambling process is used to produce a key dependent permutation of $0, 1, \ldots 255$ in the S array. In the initialization stage, the S array is set to the identity permutation using the formula $S[i] = i (i = 0, 1 \ldots 255)$ and the K array is set to the *key*, repeating as necessary to fill the array. The S array is scrambled by selecting two indices i and j and then swapping $S[i]$ and $S[j]$. In pseudocode form, the key schedule is computed as follows:

```
keyschedule()
{
    /* initialization */
    for i = 0 to 255
        s[i] = i;

    /* scrambling */
    j = 0;
    for i = 0 to 255
    {
        j = j + K[i] + S[i];
        swap S[i] and S[j];
    }
}
```

The PRNG phase is similar to the key schedule. Indices i and j are selected and $S[i]$ and $S[j]$ swapped. The output of the PRNG is the value of the S array indexed by $S[i] + S[j]$ (i.e. $S[S[i] + S[j]]$).

Encryption or decryption is achieved by performing an exclusive-OR of the pseudorandom number output with the plaintext or ciphertext respectively. The pseudocode below shows the process for encryption of the plaintext in the *pt* array, the result being written to the ciphertext array *ct*:

```
prng()
{
    i = 0;
    j = 0;
    while not end of stream
```

```
    {
        i = (i + 1) mod 256;
        j = (j + S[i]) mod 256;
        swap S[i] and S[j];
        t = S[i] + S[j];
        ct[i] = pt[i] xor S[t];
    }
}
```

2.2 Key Search

The design described in this paper performs a known plaintext attack via a key search [9]. In a known plaintext attack, it is assumed that the ciphertext as well as the corresponding plaintext is available and one wishes to deduce the key used for encryption. The same architecture, with additional filtering logic (e.g. to detect if the message is 7-bit ascii) could be used for a ciphertext only attack.

If the plaintext and ciphertext are known and n bytes in length, checking that the ciphertext, ct, when decrypted using a key k is the same as the plaintext pt, is equivalent to checking if the first n bytes of the RC4 PRNG produces the sequence pt xor ct.

If N RC4 key search units are available, i is an index used to identify each RC4 key search unit, and $rc4(cxp, k)$ checks to see if the PRNG produced with key k gives cxp, the key search procedure can be described in pseudocode form as:

```
keysearch()
{
    k = 0;
    cxp = pt xor ct;
    forever
    {
        for i = 0 to N-1 (in parallel)
        {
            found = rc4(cxp, k + i)
            if (found(i))
                return k + i;
        }
        k = k + N;
    }
}
```

3 System Architecture

3.1 RC4 Cell Design

The datapath of a single RC4 cell is shown in Figure 1. The core component of the RC4 cell is the S-block for the S array, which is implemented using a 4096-bit on-chip Block RAM [10], configured as an 8-bit wide dual port memory. Since the RC4 algorithm requires only $8 \times 256 = 2048$ bits of memory for the S array, the Block RAM is divided into two halves via the most significant bit of the address. As the key scrambling phase for a new key is being computed in one half of the RAM, initialization for the next key is done in the other half. This scheme saves 256 cycles and hereafter, this combined initialization and scrambling step will be referred to as the key schedule phase.

Each iteration of the key schedule phase requires 3 clock cycles as shown in Figure 2. In the first clock cycle, i is passed into port A of the Block RAM as an address, and the initialization of S for the next key is done at the same time via port B. In the second clock cycle, the value of S[i] becomes available and j is computed. In the last clock cycle, S[j] is available and the contents of S[i] and S[j] are swapped and written back to S.

The PRNG phase (see Figure 3) also requires 3 clock cycles per iteration, hence a total of $768 + 3n$ cycles are required to test each key (for an n byte long ciphertext). Operations in this phase are similar to those of the key schedule phase except that S does not require initialization. The t value is ready (as the *t_pre* signal) in the first clock cycle of the next iteration. The output, *s[t]*, is read and compared with the *cxp* value in following cycle.

A possible memory contention problem exists in the last clock cycle of each iteration in the key schedule and PRNG phases, since it is possible that both ports attempt to write the same data to the same address, producing unpredictable results [10]. To avoid this conflict, a comparator is added to the RC4 cell (not shown in the schematic) so that if i and j are equal, the write enable to one of the ports is disabled.

Finally, a latch called *found* in the RC4 cell is used to indicate whether the key being tested matches the plaintext. This latch is cleared if the byte produced by a decryption does not match *cxp* (as described in Section 2.2). Should the latch remain high after all bytes of the plaintext have been tested, the key being tested is the desired key.

3.2 Key Search

The top level block diagram of the design is shown in figure 4. All RC4 cells are identical. Each cell accepts a key input and sets a flag if the input is a valid key. There is one global key register which is initialized by the host and routed to all RC4 cells. A local key is computed in each cell by summing the

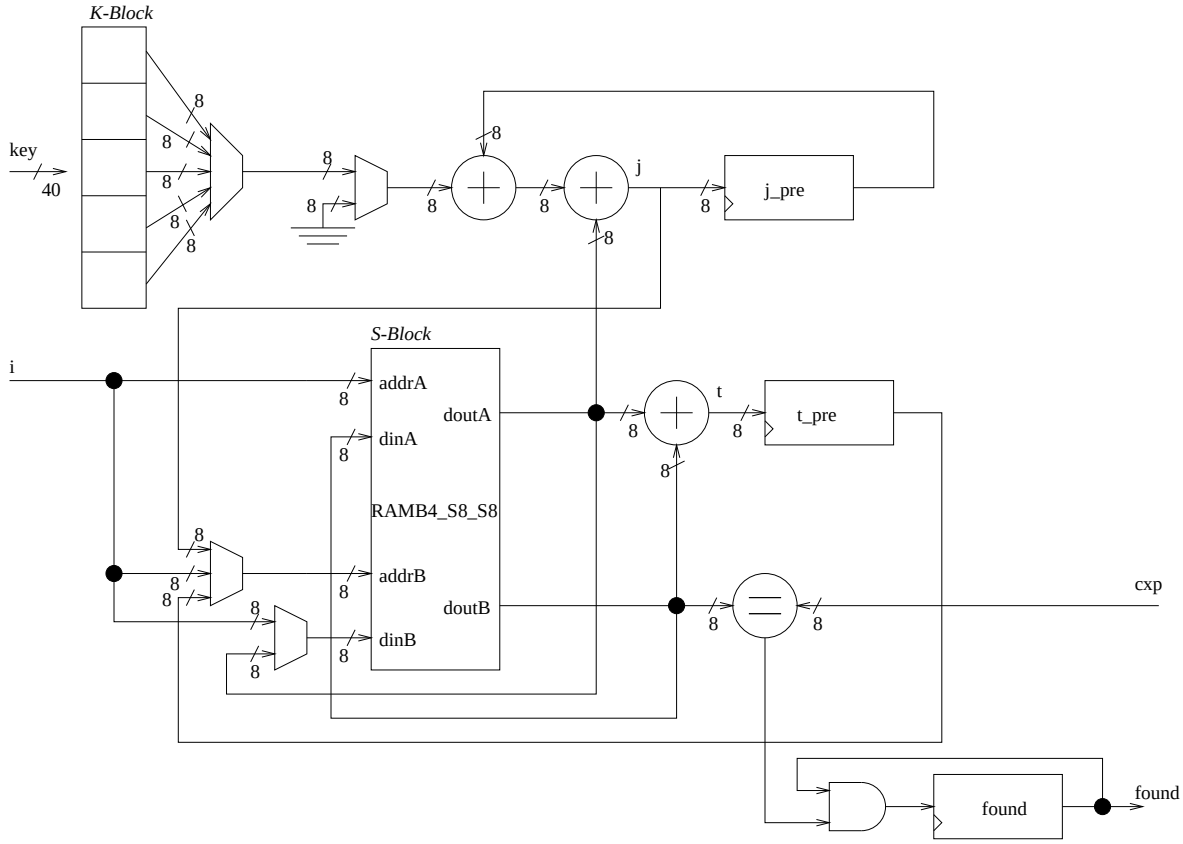

Figure 1: Datapath of the RC4 cell.

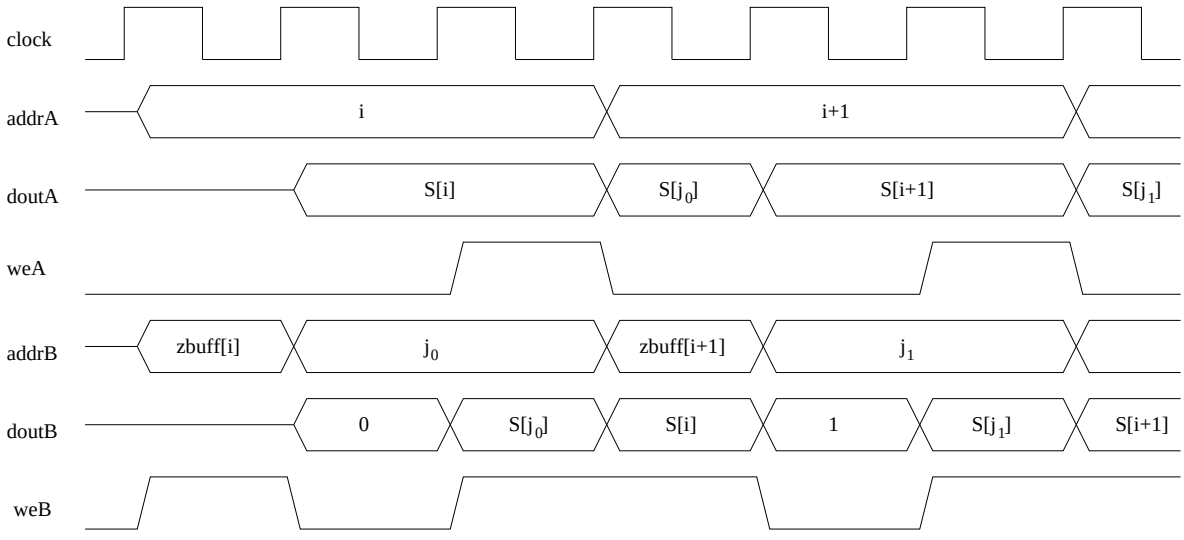

Figure 2: Timing diagram of the block RAM during the key schedule phase.

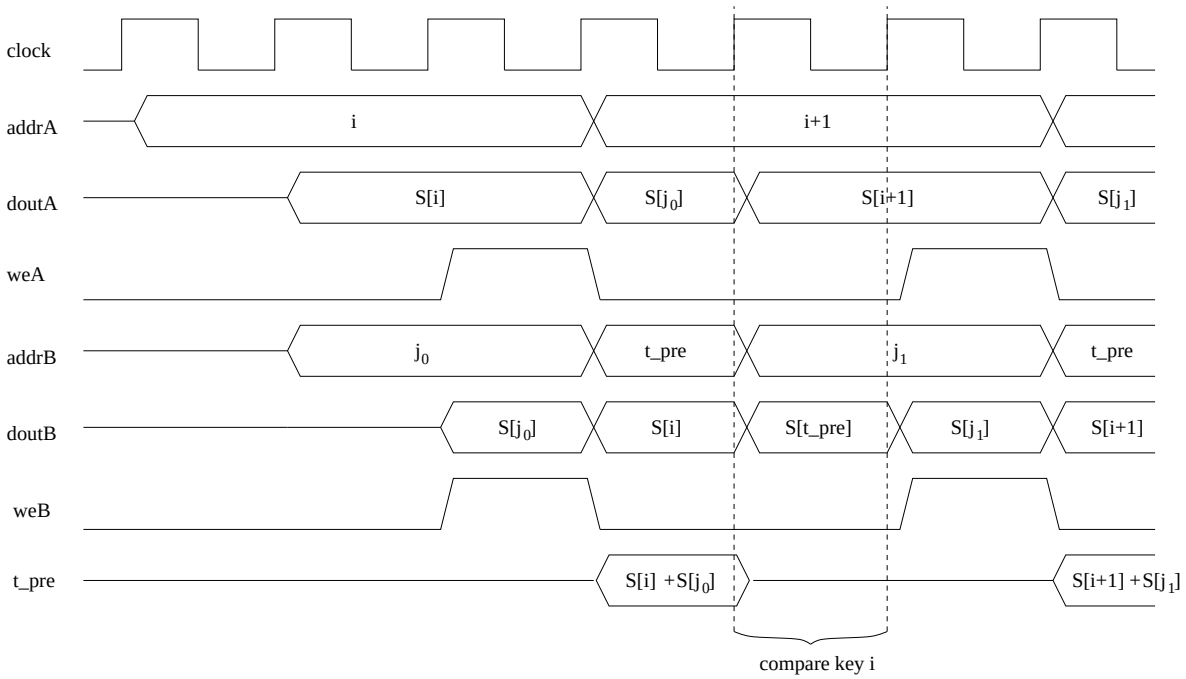

Figure 3: Timing Diagram of Block RAM in PRNG Phase

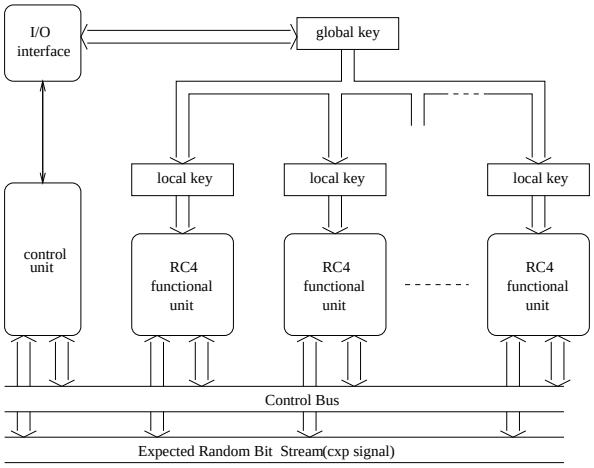

Figure 4: Block diagram of parallel RC4 key search machine.

global key with a cell offset, which is a unique value ranging from 0 to 95. By using this scheme, the RC4 cell array can process 96 different keys in parallel, after which, 96 is added to the global key.

All RC4 cells share a common control unit, implemented as a simple finite state machine (FSM). This unit controls the state of all the RC4 cells, updates the global key and also provides the interface to a host computer (discussed in Section 3.3).

3.3 Interface

In the host/key search engine interface protocol, the host must download the expected PRNG sequence *cxp* and then the start key value for the key search. After the search engine receives the start key, it works independently of the host, testing new keys until it detects that the *found* flag of an RC4 cell has been asserted (in which case the FSM halts). The host then can download the global key and offset which produces *cxp*. The host and key search engine communicate via a set of 3 64-bit read and 2 64-bit write registers.

The interface protocol is detailed in Table 1. Write registers are used by the host to send the start key (w0) and expected PRNG sequence, cxp (w1) to the key search engine. After the key has been found, the host can read back the global key value (r0), and the offset of the RC4 cell which asserted the *found* flag (r1, r2).

Table 1: Host/key search engine handshaking protocol.

	Action	Register	State
1	Host writes expected PRNG sequence (cxp)	w1	idle
2	Host writes start key	w0	start
3	Host polls flag registers	r1, r2	searching
4	Search engine writes 96-bit offset	r1, r2	end
5	Host reads global key	r0	idle
6	Host reads offset	r0	idle

Table 2: Components inside an RC4 cell.

name	function
D_unit	8-bit 2-to-1 MUX select portB data input
A_unit	8-bit 3-to-1 MUX select portB address input
F_unit	8-bit compare and registers generate *found signal*
W_unit	8-bit compare detect Block RAM address conflict
I_unit	combinational logic to control MSB of portB address
J_unit	two 8-bit adders with registers compute the *j value*
T_unit	8-bit address with registered outputs compute the *t value*
K_unit	5-to-1 8-bit mux (using tristate buffers) select byte from K-block

4 Implementation

The design is modularized and floorplanning was done to reduce implementation time as well as improve the maximum frequency of the design. In this section, details of the implementation are presented.

4.1 RC4 cell

There are 8 major components inside an RC4 cell, the dual port RAMs and the 40-bit local key registers being excluded from the RC4 cell for reasons described in Section 4.2. The names of the components and their functions are listed in Table 2.

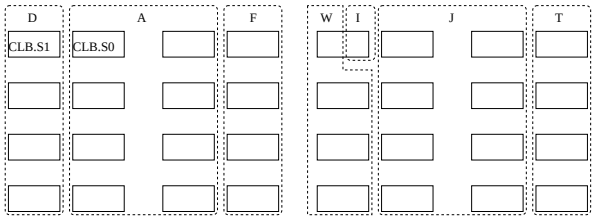

Figure 5: Block diagram showing component placement within an RC4 cell.

The RC4 cell was designed to fit into a 4 *row* × 6 *column* Virtex-E configurable logic block (CLB) array. All components are structural HDL descriptions containing only primitives provided by the Xilinx library. The physical placement of components were fixed using relative location (RLOC) attributes. The complete cell is a RPM (Relationally Placed Macro) which can be instantiated multiple times in the top level design. The block diagram in Figure 5 shows the layout of components within the RC4 cell. In the figure, the small rectangular boxes represent a slice (two logic cells, where each logic cell contains a 4 input lookup table) and two adjacent slices form a Virtex-E CLB. This scheme ensures low local routing delays.

The multiplexer for the *K_unit*, which is used to select a byte from a 40-bit key, is implemented using tristate buffers (TBUFs) and do not use CLB resources. This scheme replaces the large multiplexer in Figure 1 and reduces both logic and routing resources.

4.2 Floorplan

On the XCV1000E FPGA, the 96 Block RAMs are grouped into 6 columns. Adjacent Block RAMs are separated by 4 rows of CLBs. The RC4 cell described in Section 4.1 was designed to have the same pitch as the Block RAM and hence, each of the 96 RC4 cells are placed adjacent to a Block RAM which is used for the *S-block*.

The 40-bit local key is another module used in the design. This module is a 40-bit adder with registered outputs and is used to latch the sum of the global key and the offset of the RC4 cell. To avoid breaking the fast carry chain, this module is implemented as a column which is 20 slices (or 5 RC4 cells) high (see Figure 6). Five local keys are grouped together and placed perpendicular to their corresponding RC4 cells as shown in Figure 6). Since the local key modules have no direct connections to the Block RAMs, placing them away from the Block RAM column does

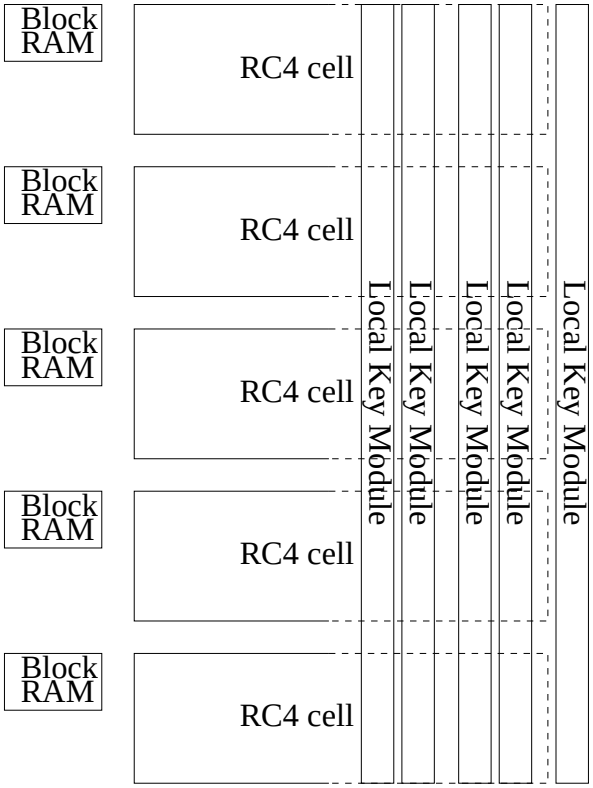

Figure 6: Block RAM, RC4 Cell and local key module placement.

not increase the routing delay. Since the TBUFs and CLBs are independent, the RC4 cell overlaps with the local key module in a section where the RC4 cell only uses TBUFs and the local key module only uses the CLBs.

Figure 7 shows the floorplan of the completed design. It can be seen that the RC4 cells and local key modules are placed close to the Block RAM columns. The control unit is located in the center where the distance to all RC4 cells is minimized.

5 Results

An implementation of the RC4 key search engine was synthesized and implemented using Synopsys FPGA Compiler and Xilinx Alliance respectively. The FPGA platform used was a Pilchard FPGA card (Figure 8) [11] which uses the SDRAM bus instead of the PCI bus used in conventional FPGA boards. The design was successfully tested on the Pilchard platform with a Xilinx XCV1000E-6 FPGA by performing key searches on randomly generated 40-bit keys. The performance was compared with an optimized software implementation on various general purpose microprocessors.

Table 3: Device utilization summary.

DLLs	1	out of	4	25%
BLOCKRAMs	96	out of	96	100%
SLICEs	5178	out of	12288	42%
TBUFs	4608	out of	12544	36%

Table 4: RC4 Encryption Speed on Different Platforms.

Platform	Frequency MHz	Time us	Normalized Time
Sun Ultra IIi	400	49456	299
SGI R12000A	400	11318	68.6
Intel P4	1500	9618	58.3
This work	**50**	**165**	**1**

The RC4 engine containing 96 RC4 cells was designed for 50 MHz operation as reported by the Xilinx timing analyzer. The system RAM bus interface operates at 100 MHz. Resource utilization as reported by the implementation tools are listed in Table 3. In our design, less than half of the slices are used, and it may be possible to implement more RC4 cells on the FPGA using slices in place of the Block RAM as the memory elements.

Since each RC4 core requires $768+3n$ cycles to test a key (Section 3.1) and $n = 8$ was used, a single RC4 key is tested in 792 cycles (15 μS). Hence the average time to test a key when all 96 cells operate in parallel is 165 ns.

An optimized software implementation of the RC4 algorithm was used to compare the speed of the RC4 key search engine with that of a contemporary microprocessor. The key is generated and stored in memory and the size of expected pseudo random bit stream was 8 bytes. The speed measurements (for searching 1000 keys) only consider the computation time and involve no I/O operations. The GNU GCC compiler v2.9 was used to compile the program source using the '-O3' optimization flag. The speed of the microprocessor based implementation is compared with that of the FPGA implementation in Table 4. The 50 MHz FPGA implementation is approximately 60 times faster than the 1.5 GHz Pentium 4 implementation.

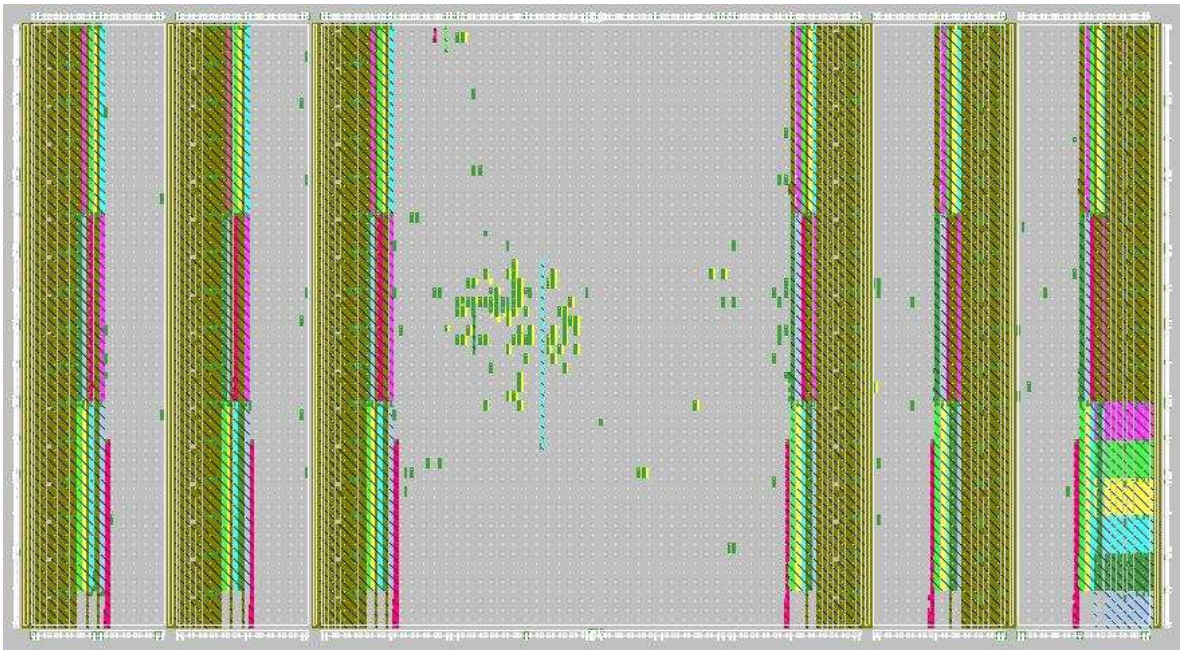

Figure 7: Floorplan of the completed design.

Figure 8: Photograph of the Pilchard board.

Table 5: Time required for an RC4 key search.

Platform	40-bit key hours	56-bit key years
Sun Ultra IIi	15084	113007
SGI R12000A	3451	25861
Intel Pentium 4	1361	10269
This work	**50**	**377**

Table 5 shows the time required to search a complete 40-bit and 56-bit RC4 key space. Since FPGA chips with more logic resources and faster clock rate are already available, the performance of the FPGA RC4 key search engine can be further improved. A Xilinx XCV3200E has double the number of block RAMs, and the XC2V8000 can contain 672 RC4 engines.

6 Conclusion

A highly parallelized RC4 key search engine based on an FPGA device was presented. In this system, a single Xilinx XCV1000E-6 chip can search more than 6,000,000 keys per second, and is approximately 60 times faster than a 1.5 GHz Pentium 4. Using this implementation, a complete 40-bit RC4 key space can be tested in 50 hours. The design can be easily adapted to systems containing multiple FPGA cards or chips. It can also be improved by using larger FPGA devices.

This particular application is a good demonstration of a system where the differences in architecture between an FPGA implementation over a microprocessor based implementation are highlighted. Despite the fact that the Pentium 4 operates at a clock speed which is 30 times faster than the FPGA implementation, the FPGA implementation is approximately 60 times faster. The microprocessor based implementation is limited by relatively low memory subsystem throughput and data dependencies in the algorithm. The FPGA implementation benefits from a high degree of parallelism, both at the algorithmic level and due to the integration of a large number of RC4 cores and memory on a single device.

References

[1] Electronic Frontier Foundation., *Cracking DES*. O'Reilly, 1998.

[2] RSA Labs, "DES III Challenge," *http://www.rsa.com/rsalabs/des3/*, 1999.

[3] A. Elbirt and C. Paar, "An FPGA implementation and performance evaluation of the Serpent block cipher," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 33–40, 2000.

[4] C. Patterson, "High Performance DES Encryption in Virtex FPGAs using JBits," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 113–121, 2000.

[5] S. Trimberger, R. Pang, and A. Singh, "A 12Gbps DES Encryptor/Decryptor core in an FPGA," in *Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES)*, pp. 156–163, Springer, 2000.

[6] O.Y.H. Cheung, K.H. Tsoi, K.H. Leung, P.H.W. Leong, and M.P. Leong, "Tradeoffs in parallel and serial implementations of the international data encryption algorithm IDEA," in *Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES)*, pp. 333–347, LNCS 2162, Springer, 2001.

[7] I. Goldberg and D. Wagner, "Architectural considerations for cryptographic hardware," http://www.cs.berkeley.edu/~iang/issac/hardware/, 1996.

[8] P. Kundarewich, S. Wilton, and A. Hu, "A CPLD–based RC4 cracking system," in *IEEE Proceedings of the Canadian Conference on Electrical and Computer Engineering*, pp. 397–401, 1999.

[9] B. Schneier, *Applied Cryptography*. Wiley, 2nd ed., 1996.

[10] Xilinx, "Using the Virtex Block SelectRAM+ Features," *Applications Note XAPP130*, 2000.

[11] P.H.W. Leong, M.P. Leong, O.Y.H. Cheung, T. Tung, C.M. Kwok, M.Y. Wong, and K.H. Lee, "Pilchard – a reconfigurable computing platform with memory slot interface," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) – to appear*, 2001.