Searching RC5-Keys with Distributed Reconfigurable Computing

Dirk Koch, Matthias Körber, and Jürgen Teich
Department of Computer Science 12
University of Erlangen-Nuremberg
Email: {dirk.koch, teich}@cs.fau.de

Abstract

Distributed Computing projects such as SETI@home or distributed.net have demonstrated how supercomputer performance can be achieved by the use of thousands of linked computers contributing their idle time. The goal of this paper is to combine traditional Distributed Computing concepts with reconfigurable hardware, where free resource area can be used to achive computation speed-ups. Our approach is demonstrated on the base of a RC5 brute force key search analog to the distributed.net project. To support this, we implemented I) a platform-independent, scalable, and efficient hardware key searcher which can be easily used with various types of FPGA boards. In addition, we II) present how these computing nodes can be managed in reconfigurable runtime systems and integrated into a large scale network.

1 Introduction

Projects like *SETI@home* from the Univ. of California, Berkeley, *Folding@Home* from the Stanford Univ. and the RC-5 project from *distributed.net* [4] are only some examples for large scale Distributed Computing systems. Common for these projects is a high ratio of computation time to communication time. This can be best observed in the successful finished RC5-64 key searching project from distributed.net, where the enormous amount of $15.8 \cdot 10^{18}$ keys out of the complete keyspace of $18.4 \cdot 10^{18}$ keys were checked in 1757 days by more than three hundred thousand individual participants. Each project attendee ran a client software communicating with a centralized server managing the keyspace. Software clients optimized for various computer systems allowed heterogeneous computing nodes. The main information necessary to be exchanged between the clients and the server is the fraction of the keyspace to be tested next on a client and a potential hit.

The contribution of this paper is a) to merge and combine Reconfigurable Computing together with the concepts of large scale Distributed Computing. In order to illustrate our methodology, we b) analyzed the distributed.net project and implemented a comparable system to be capable to use a wide spectrum of different available FPGA-boards to accelerate the breaking of RC-5 encrypted messages on the Internet. Analog to the distributed.net software client that uses *idle CPU time*, our hardware version allows *idle FPGA resource area* to be used for searching RC5 keys. In detail, we cover the following points:

- *VHDL implementation* of a *generic* and therefore for different RC5 parameters easily adjustable hardware key searcher that needs to be *high-performance* and *platform-independent* for easy FPGA board adaptation. In addition, the design needs to be *scalable* in order to make *efficient* use of different FPGA types with a high degree of device utilization.
- *Network integration* of a PC based client-server system for managing the keyspace and distribution of the workload.
- *Interfacing* of the FPGA boards to distributed PC clients.
- Managing of the key searcher instances by the use of partial runtime reconfiguration.

Successful work in the field of Reconfigurable Computing has been published several times before [3]. An often seen problem with respect to a large scale distributed Reconfigurable Computing system is the *specificity* of the proposed solution. This can be seen with respect to the used platform, the scalability or the adaptability that is addressed in this paper. In [6], a related project is presented where eight PCs each equipped with a Virtex XCV2000E board are used to break RC5. The performance of the complete cluster with 13 MKeys/second is much below our results achieved on a single board equipped with an FPGA of lower density. In addition, we are capable to integrate various boards consisting of different FPGA families together to a large scale network.

The paper continues as follows: Section 2 reveals the RC5 Algorithm with respect to an implementation of a key searcher presented in Section 3. Section 4 discusses the integration of the key searchers into a large scale network environment. Our experimental results are presented in Section 5. Finally, the paper is concluded in Section 6.

This work was partly supported by DFG (Deutsche Forschungsgemeinschaft) under grant Te163/10-2

2 The RC5 Algorithm

The RC5 cryptographic symmetric block cipher algorithm [7] is designed to be easily implementable in software as well as in hardware. As compared to DES, it is highly parametrizable to satisfy different security demands. A specific RC5 algorithm is specified as RC5-$w/r/b$ with w denoting the word size in bits (16, 32, 64), r the number of encryption rounds (0...255), and b the key size in bytes (0...255).

The algorithm is based on the simple operations ADD/ SUB modulo 2^w, bitwise eXclusive OR ($\oplus$) and bitwise rotation ($\lll$, $\ggg$). The two main functions in RC5 are the round key generator and the encryption round computation (the decryption round, respectively). Let A and B be registers of size w. Let S and L be arrays initialized with some predefined values. Then, the main work of the round key expansion is given by the following algorithm:

Algorithm 1. The RC5 round key setup.

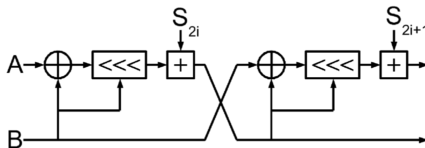
$i = j = 0$;
$A = B = 0$;
do $3 * \max(2(r+1), 8b/w)$ times:
  $A = S[i] = (S[i] + A + B) \lll 3$;
  $B = L[j] = (L[j] + A + B) \lll (A + B)$;
  $i = i + 1 \bmod (2(r+1))$;
  $j = j + 1 \bmod (8b/w)$;

The result of the key expansion S is used by the encryption round. For an input block of two words stored in two registers A and B each of size w, the encryption round performs the following algorithm:

Algorithm 2. The RC5 encryption round.

$A = A + S[0]$;
$B = B + S[1]$;
for $i = 1$ to r do
  $A = ((A \oplus B) \lll B) + S[2 * i]$;
  $B = ((B \oplus A) \lll A) + S[2 * i + 1]$;

This algorithm is depicted in the following scheme



with S denoting the key expansion supplied from the round key generator. As the decryption round has to accomplish the inverse functions in opposite order, it can be easily derived from the encryption round.

Both, the encryption as well as the decryption round has to execute two times the following operations: a barrel-shift, a bitwise XOR and an ADD (SUB, respectively). Especially for the implementation of RC5 on low density FPGAs, a half round consisting of only the half set of operations is suitable.

2.1 Attacking RC5

In order to test the security of cryptographic algorithms, it is common to stage various attacks on them. For symmetric block ciphers like RC5, attacks can be divided into the following classes [8]:

- Ciphertext Only – The attacker has only encrypted messages for the attack available.
- Known Plaintext – Beside the encrypted message, the attacker has additional information of the unencrypted data. This can be the data itself but also metadata like the file format of the plaintext data can reveal enough information to allow these kind of attacks.
- Chosen Plaintext – In this methodology, the attacker stimulates the encryption algorithm with defined data in order to draw conclusions about the key by analyzing the cipher output.
- Chosen Ciphertext – equivalent to Chosen Plaintext, but this time the decryption algorithm is stimulated.

More advanced attacks have been developed in the last two decades, e. g., differential cryptanalysis which can be classified as a chosen plaintext attack. Indeed, it has been shown that RC5 is susceptible to this kind of attack using 2^{44} chosen plaintexts [1].

With respect to the RSA competition, however, the only known way to break RC5 is by trying all possible keys with a brute force attack because the plaintext information is too short to allow the use of other crypto-analytical methodologies.

The difference between such a brute force key searcher and a (de-)cipher is basically that a key searcher processes only a limited number of input blocks for each key while a (de-)cipher can reuse the expanded round keys until the secret key changes. Therefore, focus has to be put onto the computation of the round key expansion when a key searcher has to be efficiently implemented, as shown in the following.

3 Searching RC5 Keys in Hardware

For a known plaintext brute-force attack on RC5, it is possible to encrypt the known plaintext and compare it with the ciphered message as well as to decipher the encrypted message and perform the comparison on the plaintext side. As shown in the algorithm for the encryption and decryption round, the amount of hardware needed is equal but the index range to access the array S containing the expanded round keys differs. As the entries of the array S are determined ascending from $S[0]$, it is advantageous to encrypt the plaintext, because the results of S can be used in the same order they were computed and have to be accessed only once.

Beside the design objectives flexibility and portability, we demand performance and efficient implementation

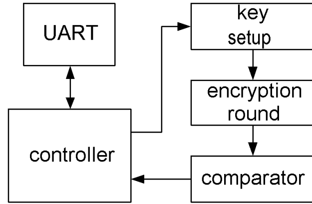


Figure 1. Block diagram of the key searcher.

of the key searcher. Therefore, we cannot accept large amounts of logic with a low utilization factor. So, we have to pay attention to find a good partitioning of the algorithm onto the FPGA resources. If we analyze Fig. 1, we see that the result of the key setup is supplied to the encryption round. The algorithm describing the round key expansion shows that we need four distinct additions, two bitwise rotations and two memory accesses per iteration for each round key. We can parallelize some of the operations by creating a pipeline with five stages. Thus, we are able to compute round keys corresponding to five different secret keys that have to be tested. In addition, we know that the key setup takes three iterations. Therefore, we use three instances of the key setup in parallel delaying the start of the second and third instance by the duration of one and two iterations, respectively. If we examine again the encryption round, we see that two consecutive round keys are consumed in every iteration. This leads to two possible alternatives: we can double the amount of key setup instances, or we can use a so called *half round* with only one adder, XOR and barrel shifter that is used two times once for A and once for B. As the first approach gives a better overall device utilization ratio, mainly because of the control and communication overhead, it is favored for the implementation (Fig. 2).

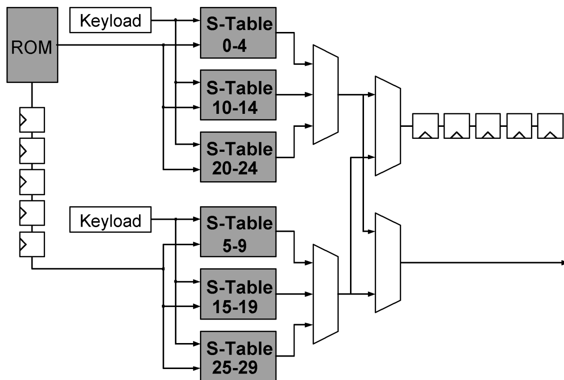


Figure 2. Key setup; the numbers in the S tables depict the key offsets computed simultaneously.

4 Network Integration

As the title of this paper denotes, the integration of distributed computing nodes to a large scale system is a center point in this work. This demands a suitable network topology described next as well as adequate software support described at the end of this chapter.

In practice, it is rather common to use an FPGA board in combination with a PC than running it completely standalone when a LAN access is demanded (as we always assume in this paper). As we want to keep as much FPGA-resources for the key search as possible, we use a hierarchical approach for the communication of the hardware / software RC5 clients with the Server running the keyspace manager. We divided the computing node integration in three levels. On the highest level, PC clients communicate with a centralized keyspace-manager. In the next lower hierarchy, the FPGA boards communicate with the overlaying PC clients using serial RS232 links. Finally, on the lowest hierarchy, we have the FPGA device level where the hardware key search instances communicate by the use of configured links inside a FPGA.

As displayed in Fig. 3, we use a TCP/IP connection between the RC5 Server hosting the keyspace-manager and the client PCs. Each client PC has two functions. First, it contributes to the key search by running a software version of the key searcher. The second function of the PC client is its gateway role for the hardware key searching instances running on the FPGA boards. These instances are connected to the client PC via RS232 serial links. Note that a single FPGA can be used to accommodate more than one instance of the hardware key searcher if there are sufficient resources available.

4.1 PC Client to Board Integration

As mentioned before, the FPGA boards are not communicating with the RC5 server directly. They only see their PC client they are connected with by the use of an RS232 serial link. This demands a tiny UART in every instance and will therefore produce a small overhead when running more than one instance per board on the one side. On the other side, this allows an extreme flexible composition of the instances in the target system where it is of course possible to run the key searchers simultaneously with other hardware modules on the same FPGA.

Even autonomous dynamic changes are possible on boards that support partial runtime reconfiguration. In this case, free FPGA resource areas left over by the runtime system can be used for the key search in hardware as presented in the next section. This is analogue to the RC5 software client from distributed.net that uses only idle time on the CPU. So far, we haven't discussed what data is sent between the RC5 server and the clients. The keyspace manager allocates an individual 32 bit sub-keyspace to every known client. The server does only distinguish between software

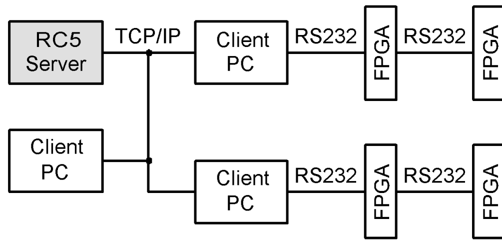


Figure 3. Example of a hierarchical network where the HW instances running on the FPGAs communicate through the client PCs with the server.

and hardware key searching instances for statistical monitoring purpose. If a new client registers for the first time at the RC5 server, it gets back a package with one block of the known plaintext, the according ciphered block (a block has a size of two words) and the allocated keyspace. Thereupon, the client will start the search process, where the plaintext is encrypted and compared with the ciphered block. If a client finds a match, it sends the current key to the server and continues the search. After all 2^{32} keys of the sub-keyspace have been processed, the client requests the next sub-keyspace from the server. In addition, each client sends periodically an alive message to the server, because the processing time for one sub-keyspace can take in the minute to hour range. So we see that we can count the necessary bandwidth for this protocol in bytes per hour. Therefore, we do not demand high speed links making the interfacing simple.

Figure 3 shows that at every of the right two client PCs, two additional FPGAs were connected in a daisy-chained fashion. This allows the connection of many hardware instances to a single COM port of the client PC. For linking hardware key searchers among themselves, it is sufficient to use standard I/O-pins from the FPGA without additional drivers when different boards are connected together. If the instances are on the same FPGA, the linking is trivially done on-chip. Each node in a chain is addressed by its order inside the chain.

All instances are equal and demand no extra pins for setting addresses. Therefore, the user only has to plug a chain of the wanted amount of key searching units to the COM port of a PC client in order to start the search. As shown in Fig. 4, this does not need an additional UART for routing messages over the chain. In order to selectively address a specific node inside the chain, it is necessary to modify the address field inside the package, because all HW instances are completely the same. The package format is kept simple to maintain the hardware overhead acceptable. The first byte of the package contains the address field. Therefore, 256 individual nodes can be selected. The rest of the data (the payload) is passed to the addressed key searching instance. Only one instance in a chain can be accessed at a point of time. We demand that the key searcher keeps passive on reset not sending any data without a request from outside. Figure 4 reveals some details of the hardware that performs the modification and decoding of addresses as well as the adoption to the key searching unit. The according control FSM is presented in Fig. 5 The incoming data from the preceding and succeeding nodes is passed by a sample and hold register. Upon reset, the FSM is in the state *Init* where no incoming data is passed to the key searcher or to the succeeding nodes. If no traffic occurs, the FSM goes into the state *Idle* and is now able to accept incoming data. This concept ensures that the alignment to the serial input stream is set to the beginning of a package.

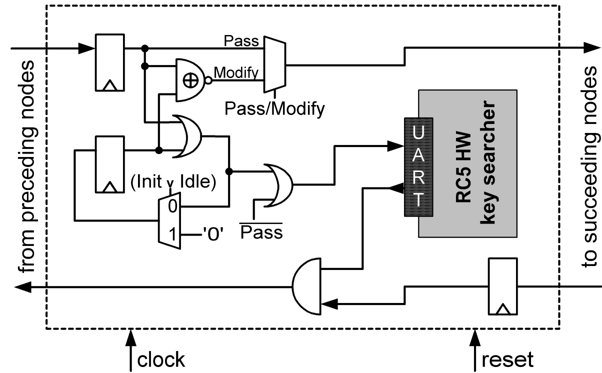


Figure 4. Decoder and modifier for addressing the HW-instances.

If in the following, the client PC sends a package to the chain, the first byte representing the address is decremented by 1 and passed to the following nodes in the chain performing the same operation (state *Modify*). The node that gets the address 0 is the one that is selected inside the chain. Other stages stay passive. The payload depicting the rest of the data, is passed in the state *Pass* unmodified to the successors. The address decrementation is done in a sequential subtracter (Fig. 4). As the representation of the -1 in the two's complement contains only a '1' at every bit position, the result bit can be determined by the use of a simple XNOR gate and the carry (in a subtracter it is often called a borrow bit) is just an OR of the present incoming bit of the address field and the carry bit result from the previous iter-

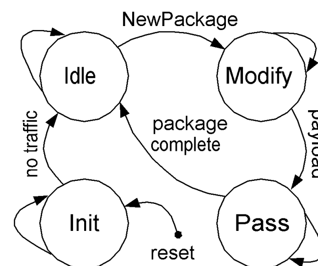


Figure 5. Control FSM for address decoder and modifier.

ation. If the register storing the carry bit is disabled during the state *Pass*, then its value is only '0' when the address field was 0, therefore we need no additional address comparator.

The length of the packages can be variable. It is only important that the control FSM remains in the state *Pass* for the time the largest payload needs to be sent by the client PC. In addition, the client PC is disallowed to send out packages only with a rate small enough that the control FSM reaches the state *Idle* before the next package arrives.

So far, we focused on how data is sent from the client PC to the attached hardware instances. In the following we describe how the instances can send data back to the PC. If a RS232 transceiver is passive and not sending data, as we demand if no request is sent to a node, it sends out a constant 1. When now, for example, the last node in the chain is answering a request (e. g., a ping request), its value passes an AND gate in every stage on the way towards the client PC. All distributed AND gates in the backward path of the chain form logically one large gate. It has the transceiver outputs of the key searching units as inputs. The output of this AND gate is connected to the receive pin of the PC. This semantics can also be seen as a distributed OR for zeros.

An interesting aspect is the capability of *hot plugging* of key searchers into a chain. This is possible as a client is not allowed to initialize a transfer by itself. If the board should be able to support hot plugging we have to ensure that the input pins for the backward path to the PC client are in a passive state by adding pull up resistors (with respect to the RS232 logic) to the pins. Some FPGA families allow this as a configuration feature. In this case, no board modification is necessary for hot plug support. As FPGAs are driving their pins tristate after power up, it is possible to connect boards to the chain before they are configured.

4.2 Online FPGA Management

On the lowest level, the key search instances are integrated in a FPGA runtime system where FPGA resources are used in a time-variant manner. In this section, we present an online placement and communication method that is compatible with most approaches for operating systems for reconfigurable hardware.

The key searching instances will be constrained to a rectangle shape of minimal width to fit one complete instance. The only global routing resource that is allowed to be used by the key searcher module is the clock. This allows to use global routing links for the communication with the I/O-pins without an influence to runtime reconfigured parts of the FPGA. The key searching instances on one FPGA are connected among themselves by the use of two identical communication macros for the left and right module edges. Therefore, two instances can be linked together by communicating directly across their borders. These communication macros must be placed in such a way that different key searchers will be integrated to a chain that is compatible to the chain known from the board level integration if the instances are placed side by side. This is achieved by placing the two communication macros in the same horizontal plane.

Whenever another module not being a key searcher is placed on an resource location, it must accommodate the same communication macros at the same horizontal position as the one used by the key searchers. These macros are connected from one side to the other trough local routing links. A reconfiguration process may infer a running communication. Therefore, we need a robust communication strategy that is presented in the following paragraph.

4.3 Software integration

The communication between the PC and the instances on the FPGA boards is always initialized by the PC playing the role of a master. Therefore, we need a software to deal with the FPGA boards on the one side and the RC5 server on the other. Note that a client PC contributes to the key search by running a software instance of the client that is functionally equivalent to the hardware implementation.

The master PC can send four different commands to a hardware instance that can be distinguished by the slave through a command byte that is always the first byte of the payload:

> $0x01$: `Setup`; if this command is sent to the slave, it knows that four w bit words will follow containing one plain text block and another ciphered one for the comparison. An eventually running key search is aborted.
>
> $0x02$: `Set new sub-keyspace`; after this command, a base address with $b - 32$ bits representing the MSBs of the sub-keyspace will follow. An eventually running key search is aborted and restarted from the beginning by the use of the newly assigned sub-keyspace. The least 32 bits are incremented automatically inside the key searcher.
>
> $0x04$ `Set identifier`; this command writes an identifier to a key search instance. This identifier will be included in all messages that are send back to the master PC.
>
> $0x08$: `Ping and state request`.

The master has to arbitrate the different slaves in a cyclic fashion in order to avoid conflicts on the shared communication medium. The arbitration is done by sending the `ping and state request` command to a specific slave. If a slave is scheduled to send data to the master, it responds first with an identifier. The identifier has to be set uniquely for each hardware instance inside a chain and is used to recognize changes to the chain. Such changes can be issued by excising or including boards to the chain or when instances are removed or added by the use of partial runtime reconfiguration. Note that changes can happen anytime and at

every position inside the chain. Therefore, the communication between the master PC and the hardware instances is designed to be fault tolerant. In the case of write operations to the hardware modules inside the chain, an operation is assumed to take place successful if we do not detect changes to the chain just before and after the write operation. In the case of read operations, all data is requested two times and tested on consistency. A failed communication process will be repeated for a limited time before a hardware instance will be assumed to be disconnected or deleted by reconfiguration, respectively.

After the identifier, a hardware key searcher instance will reply upon a `ping and state request` with a state byte allowing to distinguish between different answers. After this state byte, optional data may follow. In the case of the RC5 keyspace tester, a client can answer with three separate states encoded in one bit each allowing an overlay of the states. In detail, the states are:

> $0x01$: Found a potential key. If this bit is set, a 32 bit value containing the least significant bits will follow. Based on this information, the server can determine the complete key, as it knows which individual key searcher got a specific keyspace assigned.
>
> $0x02$: Answer a ping request.
>
> $0x04$: Current sub-keyspace is finished, request new keyspace.

If the hardware has finished a block, it waits for a new `set new sub-keyspace` command. If a potential key is found by the hardware, the current search is paused until the candidate is sent to the master.

Beside the periodic arbitration of the slaves, the client PC sends an additional ping request to the address once behind the last known node in the chain. If a new node is found, the client notifies the RC5 server that includes the new instance into his present list of key-searchers. In the following, the new hardware instance will get all necessary data to contribute to the search.

This protocol allows the hot plug and play from the software side. If nodes are removed from the chain, their temporal results get lost and the absence is detected by the client PC that will inform the RC5 server in the following. The server will assign the sub-keyspaces of removed key searchers to the remaining pool.

5 Experimental Results

We divided this section in three parts. First, we present some synthesis results of the RC5 keysearcher. Then, we reveal an implementation of a dynamical runtime reconfigurable system while focusing in the end onto the network integration.

5.1 RC5 Keysearcher Implementation

In order to demonstrate the flexibility of our VHDL implementation, we integrated it on different FPGA boards each with two chained instances of the key searcher. This test setup runs on all boards with 100 MHz achieving a computation power of 15,3 MKeys per second and board. The consumption of the logic resources is shown in the following table:

Board	Altera Nios II Kit Cyclone	AVNET V4LX25	Digilent XUP-V2Pro
Chip	EP1C20F400	XC4VLX25	XC2VP30
Synth. Tool	Quartus 4.2	ISE 6.3	ISE 6.3
Logic usage	13870 LEs (69%)	9388 slices (87%)	9215 slices (60%)

The amount of logic occupied by the communication interface for external data transfer by the UART and internal transfer between the building blocks was for all FPGA types less than 6% of the overall logic. The power consumption from the power grid was in all cases in the range of 15 watts. The maximum performance of the boards is limited by the cooling capacity. Crypto algorithms have a low level of correlation between the data of different loop iteration resulting in an extremely high flip-flop toggle rate. The aggressive pipelining helped not only to get clock rates with more than 100 MHz, it also reduces the power consumption [9]. The memory for storing the round keys was defined device-independent. It was necessary to include registers on both sides of the memory in order to get all used synthesis tools to automatically utilize the appropriate FPGA-specific RAM blocks out of the same VHDL description. The only board-specific work needed to be done manually before starting the compilation is to define the number of instances and the clock frequency (for adjusting the baud rate), assign two times two pins for the serial communication, one pin for the reset, and finally one pin for the clock. In addition, the RC5-specific parameters can be modified. Here, we kept the default settings that are compatible with the distributed.net RC5-72 project.

The key searching software on the client PCs was not optimized for speed. In order to make a fair comparison, we present the performance results of the distributed.net client installed on two different PC systems. This software client is optimized individually for two different CPU architectures

	AMD Athlon	Intel Pentium IV
frequency	1.7 GHz	3.2 GHz
MKeys/sec.	6,6	7,2
power	118 W	160 W

We can identify that the 32 times higher clocked Pentium has half the performance of the hardware implementation while consuming about 10 times the power of an FPGA board. Note that there are plenty of FPGA resources left over for other work.

5.2 Online Runtime Configuration Management

We used the ESM-Platform [2] to demonstrate the configuration management on a Virtex2-6000 device. This platform is especially designed for runtime reconfigurable systems and provides us with various types of I/O communication. However, the methodologies are transferable to other platforms and FPGA devices. In the prototype implementation, we provide four resource areas each offering 23% of the logic resources ($slot_0 \ldots slot_3$ in Fig. 6). These areas can be used for any hardware module including an RC5 key searcher instance, as shown for left and right slot in Fig. 6. The remaining 8% of the resources are used for interface purpose.

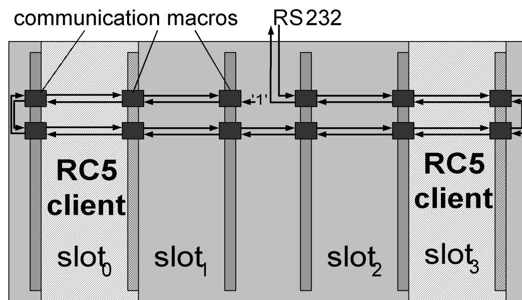


Figure 6. VirtexII-6000 FPGA implementation.

We used an incremental design flow and started from a basic design that contains a global interconnect to some I/O pins, the clock networks and all communication macros. These macros have been designed to communicate across the FPGAs internal RAM columns. Therefore, it is possible that a module uses one of the neigbor memory columns together with its logic cells. Based on this infrastructure, one hardware key searcher is designed into one resource area and connected to the communication macros. This design step is repeated for all other hardware modules with the only extension that they accommodate links between the communication macros belonging to the key searchers. The configuration designed for one resource area can be used by the other ones just by modifying the destination addresses of the configuration data [5]. In the prototype implementation, we used an PC to manage and download the adjusted configurations via the FPGAs JTAG port.

5.3 Network Integration

The RC5 server and the client software runs under Linux. Different FPGA boards have been connected and removed to the client PCs showing a stable behavior. In the largest test case, we connected 15 hardware instances by the use of 4 software clients together to the RC5 keyspace manager. In order to test the RC5 server under high load conditions, we started one hundred thousand PC clients at the same time. The test RC5 server is based on a 3.2 GHz Pentium IV PC. The RC5 keyspace manager used 40% of the CPU time and 8 megabyte of the memory when the equivalent of 100 000 key searching instances have been integrated to the system. If we want to expand our approach to much larger networks, we can easily split the whole keyspace into fractions for numerous RC5 servers.

6 Conclusions and Future Work

In this paper, we demonstrated how the huge computing problem of breaking RC5 can be tackled by linking together PCs and FPGA boards. Analog to the distributed.net project, where idle CPU time have been used to break RC5 we have pointed out, how unused resource areas on FPGAs can be utilized for reconfigurable distributed computing. In order to allow as many instances to contribute to the key search as possible, we implemented a straightforward portable VHDL module that is nevertheless high-performance and easily scalable. A communication infrastructure based on a hierarchical approach beginning from the LAN level over the board-level down to the FPGA device-level was presented. Beside a concept for an online system using partial runtime reconfiguration, we have further demonstrated our approach by a working prototype implementation.

The concepts will be transferred to further applications where such networks can be used for monitoring and reconfiguration purpose in heterogeneous distributed control systems.

References

[1] A. Biryukov and E. Kushilevitz. Improved Cryptanalysis of RC5. In *Advances in Cryptology – Eurocrypt 98*, pages 85–99. Springer, 1998.

[2] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich. Increasing the Flexibility in FPGA-Based Reconfigurable Platforms: The Erlangen Slot Machine. In *IEEE 2005 Conf. on Field-Progr. Techn. (FPT)*, pages 37–42, Singapore.

[3] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.

[4] distributed.net. *RC5-72 project home page*, 2005. http://www.distributed.net/.

[5] H. Kalte, G. Lee, M. Porrmann, and U. Rckert. REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems. In *Proc. of the 19th Int. Parallel and Distributed Processing Symposium (IPDPS05)*, 2005.

[6] J. P. Morrison, P. J. O'Dowd, and P. D. Healy. Searching RC5 Keyspaces with Distributed Reconfigurable Hardware. In *ERSA*, Las Vegas, Nevada, USA, June 2003.

[7] R. L. Rivest. The RC5 Encryption Algorithm. In *Proceedings of the 1994 Leuven Workshop on Fast Software Encryption*, pages 86–96. Springer, 1995.

[8] B. W. Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[9] S. Wilton, S.-S. Ang, and W. Luk. The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays. In *Proceedings of Int. Conf. on Field-Programmable Logic and Applications (FPL)*, volume 3203 of *(LNCS)*, page 719 ff., Antwerp, Belgium, Aug. 2004. Springer.