Getting Started

COPACOBANA — Cost-optimized Parallel Code-Breaker

Chair for Communication Security

Department of Electrical Engineering and Information Sciences

Ruhr-Universität Bochum

Germany

Computer Engineering Section

Department of Computer Science

Christian-Albrechts University Kiel

Germany

`www.copacobana.org`

Revision: December 12, 2006

Table of Contents

1 Introduction to the COPACOBANA Platform

COPACOBANA[1] *(Cost-Optimized Parallel COde Breaker)* is a cost-optimized FPGA-based machine which is suitable for running cryptanalytical algorithms. A single machine is equipped with up to 120 FPGAs of type Xilinx Spartan3-1000 and fits into a 19" rack of three height units[2] (see Figure 1).

Figure 1: Front view: panel with power LED

COPACOBANA is suitable for computational problems which are parallelizable and have low communication requirements. An example of such a parallelizable problem is a brute-force attack on the *Data Encryption Standard* (DES): an exhaustive key search takes less than nine days on average with COPACOBANA. However, the machine can also be applied to other ciphers, and to other parallel computation problems.

[1]Yes, we know, Rio de Janeiro's famous beach is spelled slightly differently, Copacabana.

[2]Metric dimensions: width x depth x height = 45cm x 49.5cm x 13.5cm)

Figure 2 shows the substantial parts of COPACOBANA:

1. a *backplane*,
2. the *FPGA modules* (up to twenty per machine),
3. a *controller card* (the picture shows a new card which is under development — the beta version uses the Cesys card)
4. the *power supply*, and
5. *housing* with connectors and fans.

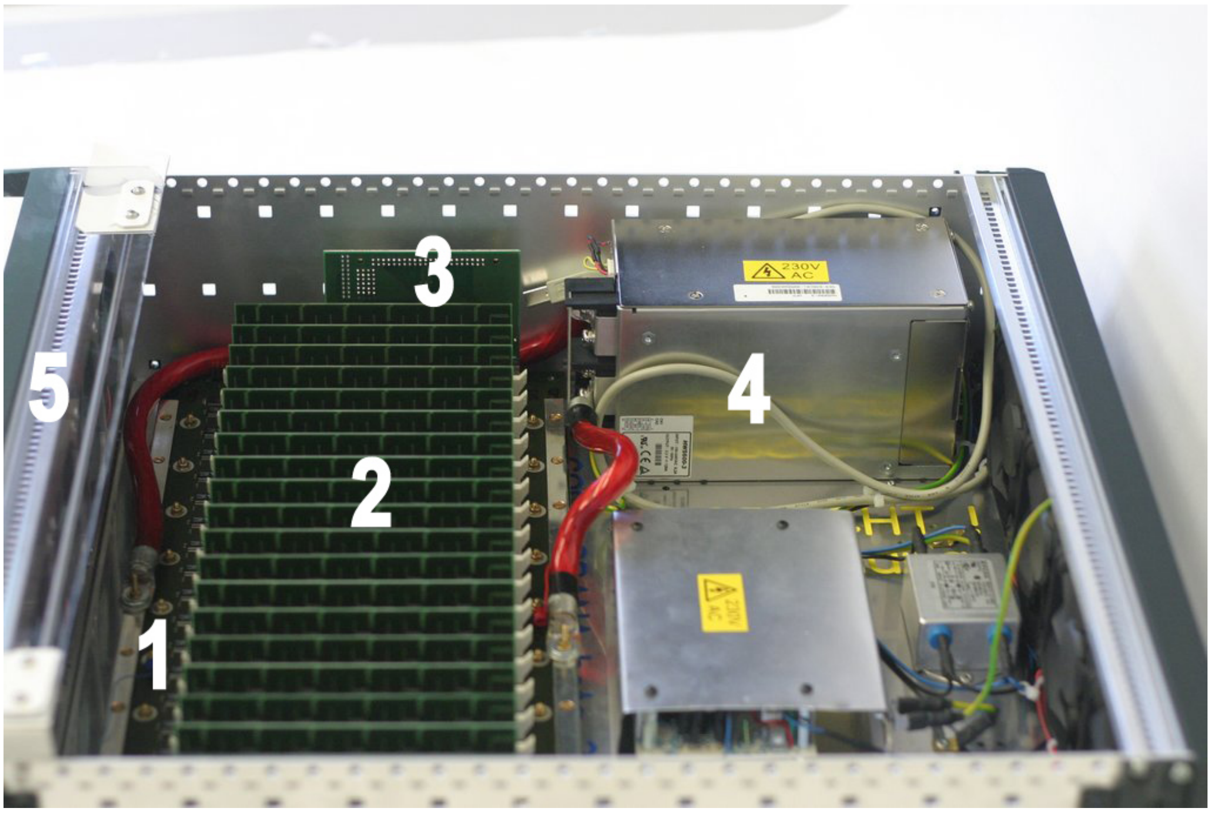


Figure 2: Main parts of COPACOBANA

1.1 Backplane

The backplane hosts the controller card which is connected via a 96-pin VG connector. 20 DIMM sockets allow for a maximum extension of 20 FPGA modules. The power supply supports COPACOBANA with 20 FPGA modules. All modules are connected by a 64-bit data bus and a 16-bit address bus. This single master bus is easy to control because no arbiter is required. Interrupt handling is totally avoided in order to keep the design as simple as possible. If the communication scheduling of an application is unknown in advance, the bus master will need to poll the FPGAs. Moreover, the power supply is routed to every FPGA module and the controller interface. The backplane distributes two clock signals from the controller card to the slots. Every FPGA module is assigned a unique hardware address, which is accomplished by *Generic Array Logic* (GAL) attached to every DIMM socket. Hence, all FPGA cores can have the same configuration and all FPGA modules can have the same layout. They can easily be replaced in case of a defect.

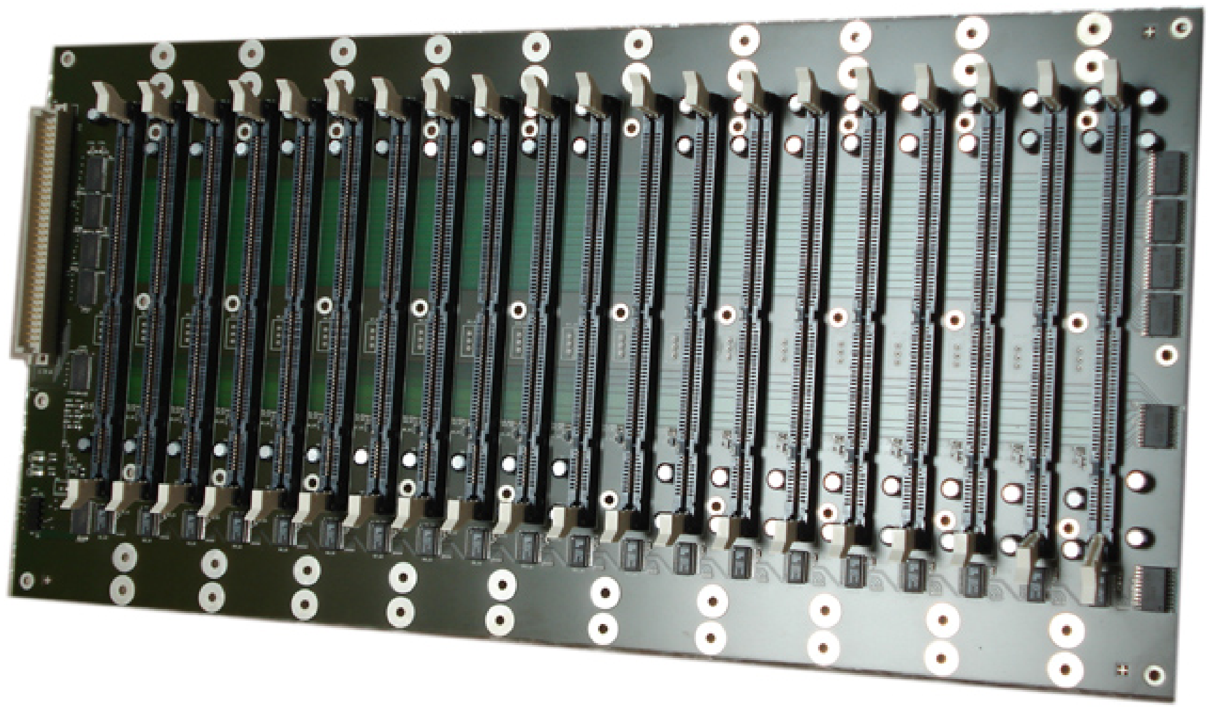


Figure 3: Plain Backplane

1.2 DIMM Modules

The FPGA modules contain six Xilinx Spartan3-1000 FPGAs (XC3S1000-4FT256). This comes with 1 million system gates, 17280 equivalent logic cells, 1920 Configurable Logic Blocks (CLBs) equivalent to 7680 slices, 120 Kbit Distributed RAM (DRAM), 432 Kbit Block RAM (BRAM), and 4 digital clock managers (DCMs). A step toward an extendable and simple architecture has been accomplished by the design of small pluggable FPGA modules. Figures 4 and 5 show the 4-layer custom made printed circuit boards. Each DIMM has its own power module, which generates the 1.2V core voltage for the FPGAs out of the supplied 3.3V.

The FPGAs are directly connected to a common 64-bit data bus on board of the FPGA module which is interfaced to the backplane data bus via transceivers with 3-state out-

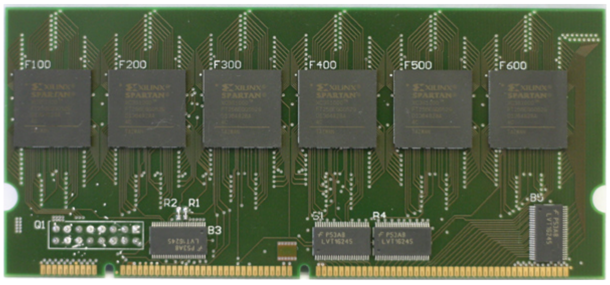


Figure 4: FPGA DIMM (front view)

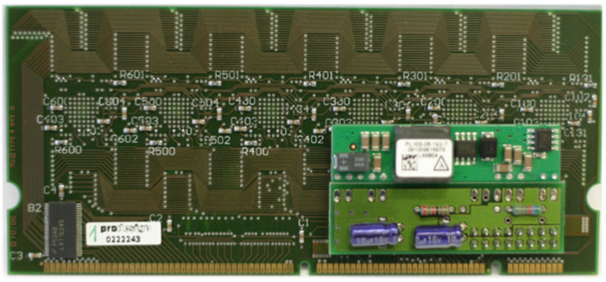


Figure 5: FPGA DIMM and power module (rear view)

puts. While disconnected from the bus, the FPGAs can communicate locally via the internal 64-bit bus on the DIMM module. The DIMM format allows for a very compact component layout, which is important to closely connect the modules by a bus.

WARNING: FPGAs can be destroyed when you do not use anti-static methods to protect the sensitive devices! We suggest the use of an ESD mat and wrist-strap.

1.3 Controller Card

Data transfer from and to the FPGAs and to the host-PC is accomplished by the control interface. In versions *Alpha* and *Beta*, COPACOBANA comes with a controller Board from *Cesys* [Ces06]. The development board comes with a Xilinx XC2S200 SPARTAN II FPGA (PQ208), an integrated USB controller (CYPRESS FX-2), and 1 MByte SRAM. Moreover, the board provides an easy-pluggable 96-pin VG connector which is used for the connection to the backplane. For more information on the hardware, refer to [Ces06]. The controller hardware has to handle the adaptation of different clock rates: The USB interface uses a clock rate of 24 MHz, the backplane is clocked with 33 MHz, and the controller itself is running at an internal clock of 133 MHz. The internal clock is generated by an external clock synthesizer, the system clock is derived from a digital clock manager (DCM) present on the FPGA. The main state machine of the control interface is used to decode and execute host commands received via USB, program the FPGAs via the data bus in slave parallel mode, initialize (write to) FPGAs and start the computation, and regularly poll the FPGAs and check for new results. Programming can be done for all FPGAs simultaneously, for a set of such, or for a particular one. Since the targeted cryptanalytic applications do not require different code on distinct FPGAs, a concurrent programming of all devices is very helpful.

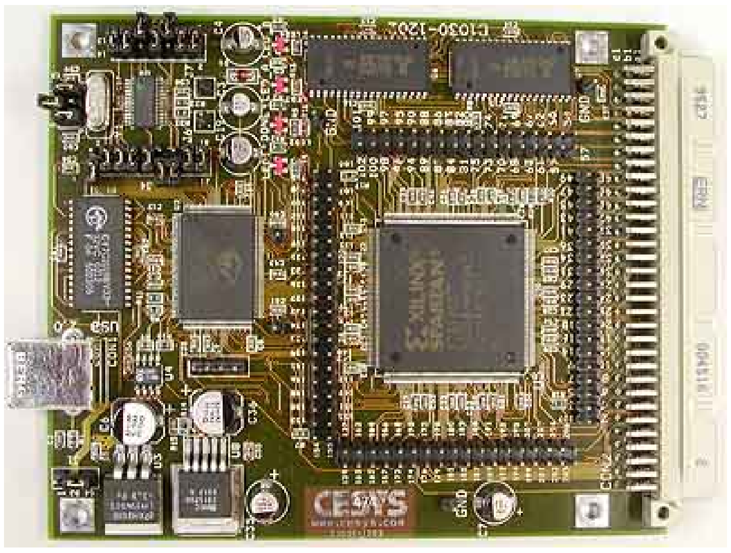


Figure 6: Cesys controller card

1.4 Housing and Power Supply

Housing is a standard 19" rack with three units of height. This way, multiple COPACOBANA units can be stacked easily. The maximum power dissipation is around 600 Watts per machine when the FPGAs are clocked internally at 100MHz and nearly 100% of the hardware resources are in use. Thus, heat dissipation is not a problem for COPACOBANA (unlike in many commercial super computers).

1.5 Host-PC

The top level entity of COPACOBANA is a host-PC which is used to program and control all FPGA implementations. For this purpose, the software issues commands to the USB connected controller card of COPACOBANA. As PC, you can essentially use any Windows PC with USB interface. Compiling VHDL sources, however, can be quite resource consuming (i.e., more than 1GHz strongly recommended). All software routines are based on the closed source library provided by the board manufacturer (CESYS). With the low-level functions, FPGAs can be addressed and data can be stored and read to/ from a particular FPGA. Further functions include the detection of the hardware and some configuration routines such as, e.g., a backplane reset. Higher-level functions comprise commands at application level. E.g., for the DES Cracker, we can store a certain plaintext in the DES units, check its status, etc.

1.6 Putting COPACOBANA into Place

If COPACOBANA is already equipped with FPGA modules (one up to twenty), the controller board, power supply, fans and further connections, simply attach the system to power plug (230V, 50Hz) and the host PC via the USB cable. Turn on the power switch which is located right below the power plug. The red LED (front panel) should light up and all fans should work properly. Please ensure that the airflow is sufficient.
Now, COPACOBANA is ready to be programmed with your favorite cryptanalytical application!

2 Setting up the Software Packages

Use the COPACOBANA CD to install the pre-compiled packages and install further software for compiling your own applications. Please insert the "COPACOBANA Resource CD" and let the setup program install all precomputed binaries, source code, and documentation.

As first step, you need to access COPACOBANA via the USB interface which requires the installation of the controller card drivers (see Subsection 2.1).
Optional further steps:

- In case you want to modify the host-PC program, you additionally need a C++ compiler to rebuild the source code of the host-PC application (see Subsection 2.3).
- For building custom FPGA applications, you also need Xilinx ISE to compile the VHDL or Verilog sources (see Subsection 2.2).

2.1 Cesys USB Driver

To install the required USB driver, simply power on COPACOBANA and connect COBACOBANA via USB to the host PC. When prompted for a driver, choose the driver from the corresponding installation directory of the COPACOBANA resources (`COPACOBANA/drivers`) and follow the installation instructions. You need to set up the driver accordingly. Currently, only the Windows operating system is supported. Alternatively, you can use the Cesys installation CD. Details regarding the Cesys platform an installation of drivers and additional software can be found in [Ces06].
A new controller board composed of an FPGA with a Xilinx MicroBlaze Softcore and Ethernet connection is under development.

2.2 Xilinx ISE

For building own images, we recommend Xilinx ISE. The images (controller and FPGA modules) for the DES cracker application were tested with ISE 6.3, ISE 7.1, and ISE 8.1. For the Spartan3 device, you can use a free version of ISE (Webpack), available at [Xil06].

WARNING: You might destroy the FPGAs when you do not use the framework. Never let more than one FPGA access the bus at the same time!

2.2.1 Compiling Code for the FPGA Modules

When building the images for the FPGA modules (DIMM), following settings have to be used:

- FPGA type: Spartan3, XC3S1000
- Packaging: FT256
- Speed-grade: -4
- The clock rate can be adjusted by four DCMs per FPGA, the input clock is 24 MHz
- Use the framework files from the example (toplevel entity and constraints file)
- Build *binary ASCII* file (not bit file, not simple binary file) and disable cyclic redundancy check when building the binary ASCII file

2.2.2 Compiling Code for the Controller Module (Cesys)

When building the images for the Cesys controller, following settings have to be used:

- FPGA type: Spartan2, XC2S200
- Packaging: PQ208
- Speed-grade: -5
- Use the framework files from the example (toplevel entity and constraints file)
- Build a *bit* file and use the Xilinx Impact tool to convert the bit file into a so-called *.exo* file. You can do this from ISE with the option "Generate PROM file".

2.3 C++ Compiler

For the host-PC application, essentially any compiler can be used. For our demo projects (DES brute-force and the memory test program), we used the Microsoft Visual C++ together with some libraries for the Cesys board and screen/ file input and output.

2.4 COPACOBANA Test Program

To check COPACOBANA's functionality, we can start with a simple test application which initializes the FPGAs with some memory blocks.

The test program simply programs the controller board with a pre-compiled image, programs all FPGAs with a pre-compiled image, and lets you access every single FGPA by

writing to some memory cells. The FPGA application is quite simple: every FPGA has 32 memory cells which can be written to and read from 64-bit values. You can choose between programming all slave FPGAs or only a subset by simply entering the DIMM number and FPGA number in the prompt. Values can then be written to any of the 32 internal registers in all FPGAs or in certain subsets of FPGAs. By reading out the memory cells, you can check the functionality of every single FPGA. There is also an automated check function, which writes and reads 64-bit random values to/ from all 32 registers of all FPGAs and reports potential errors.
If the other and more sophisticated applications do not work, this simple test program is a good point to start.

3 DES-Cracker Application

This application implements a brute-force attack on the Data Encryption Standard (DES). With a given pair of plaintext/ ciphertext, an average key search can be accomplished in 2^{55} steps where each step essentially is a DES encryption.

3.1 Short Description of the Implementation

COPACOBANA implements the DES encryptions on the FPGA modules and distributes the key space over all slave FPGAs. Each FPGA can do four keys per clock cycle (four DES engines which are fully pipelined), summing up to 480 keys per clock cycle for a single COPACOBANA machine.

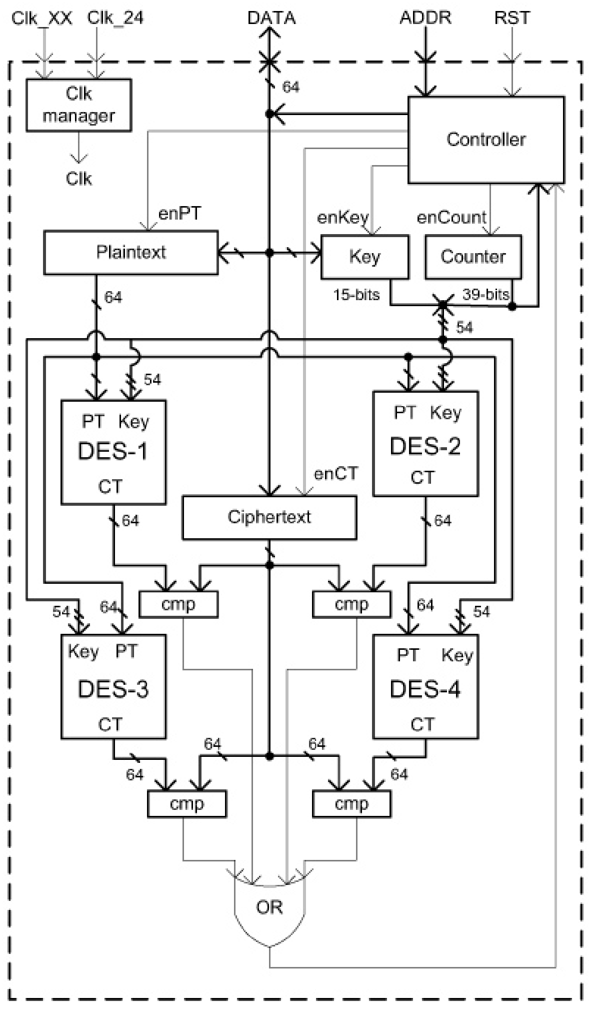


Figure 7: DES implementation on a slave FPGA

The control circuitry on each FPGA essentially consists of a 64-bit plaintext register, a 64-bit ciphertext register, a counter and some glue logic (see Figure 7). The key space is

allocated to each chip as the most-signifcant 15 bits of the key which is stored in the key register. The counter is used to run through the least signifcant 39 bits of the key. The remaining two bits of the 56-bit key determine the DES engine (we have $4 = 2^2$ engines) and is hardwired. Thus, for every such FPGA, a task is assigned to search through all the keys with the 15 most-signifcant bits fixed, that is 2^{41} different keys.
The partitioning of the key space is done by the host-PC such way that each chip takes approximately 90 minutes at 100 MHz to complete its allocated key subspace, thus, avoiding huge communication requirements. The generated cipher text (CT) is compared to that of the given cipher text stored in the register, using a comparator (cmp) block. The results of the four comparators are ORed and reported to the controller. If any of the DES engines provides a positive match, the controller reports the counter value to the host-PC, which computes the correct key and cross-checks with a software implementation of DES. The host-PC keeps track of the key range that is assigned to each of the FPGAs and, hence, can match the right key from a given counter value. If no match is found in some subspace, the FPGA reports completion of the task and a new key space is assigned by the host-PC. Since each FPGA can search through its key space totally independent of any other FPGA, only the host-PC needs to keep track of the number of FPGAs and the allocated key space.

The state machine of the slave FPGA implementation is quite simple and consists of four states (see Figure 8).

- RESET: in the beginning, every FPGA is set to the RESET state by applying the reset signal. All internal registers are being set to default values and the state machine goes to IDLE state.

- IDLE: in the IDLE state, the DES engines are not active and registers for the plaintext (PT), ciphertext (CT), and the key can be written. If the host-PC issues the start-DES command, the state machine goes to the RUNNING state.

- DONE: if the subspace is completed (indicated by a counter overflow), the key counter is stopped and the FPGA waits for a reset command to return to the RESET state.

- SUCCESS: in case a key is found in the current subspace, the counter is stopped and the FPGA stays in the SUCCESS state until a reset command is applied. In this state, the key register can be read from the controller.

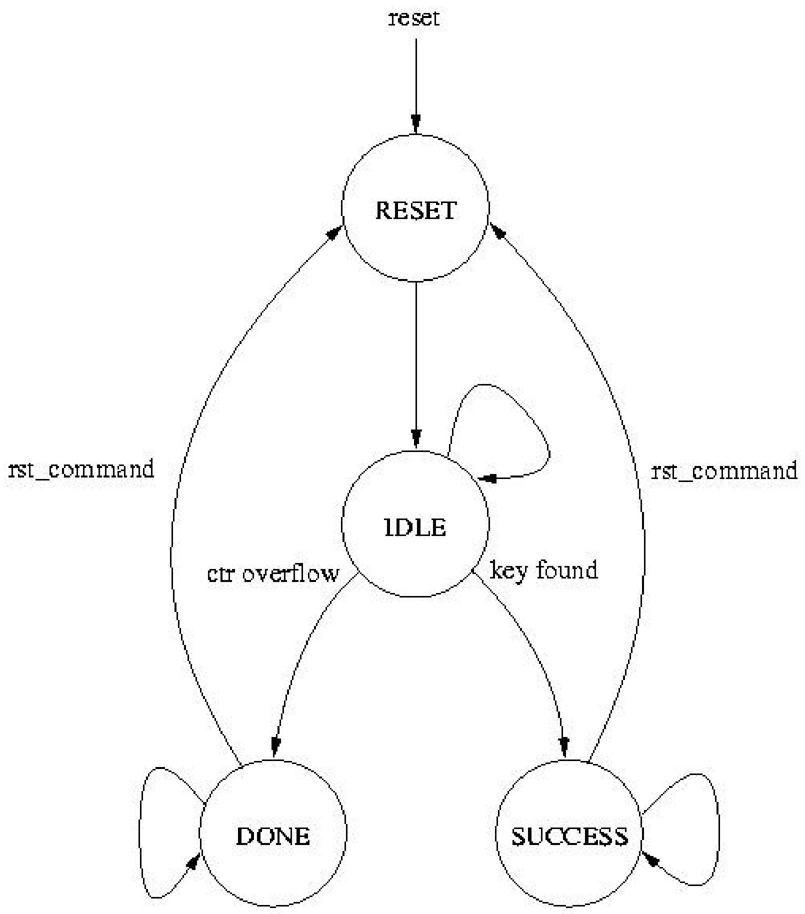


Figure 8: DES-cracker state machine

3.2 Set-up

The installation of the software (see Section 2) already created the required directories for the DES cracker application:

`COPACOBANA/DES-Cracker/Controller_VHDL`: VHDL source code for the controller (Cesys) and image file `cesys_ctrl.exo`.

`COPACOBANA/DES-Cracker/DES_CPP`: C++ source code for host-PC application and pre-compiled binary `Debug/des_host.exe` for WindowsXP. The binary is also linked to the respective entry in the start menu.

`COPACOBANA/DES-Cracker/DES_VHDL`: VHDL source code for the slave FPGAs and binary ASCII image file `toplevel_fpga.rbt`.

With these default directories and the binary files, the DES-cracker application can be started. In case you recompile (or modify and compile) one or more images, make sure that you either keep the same name and directory or change the entries in the `config.h`-file of the C++ implementation. You can also keep different working versions and change the `config.h`-file each time accordingly.

3.3 Usage

Power on COPACOBANA and plug in the USB connection. Then start the DES cracker application `des_host.exe` (accessible via the Start menu). The application automatically checks for a running COPACOBANA platform and will quit if none is found. If the COPACOBANA controller is unprogrammed, it will be programmed and restarted with the `cesys_ctrl.exo` image which takes a few seconds. Afterwards, the main menu offers you two major options:

- start key search
- resume key search

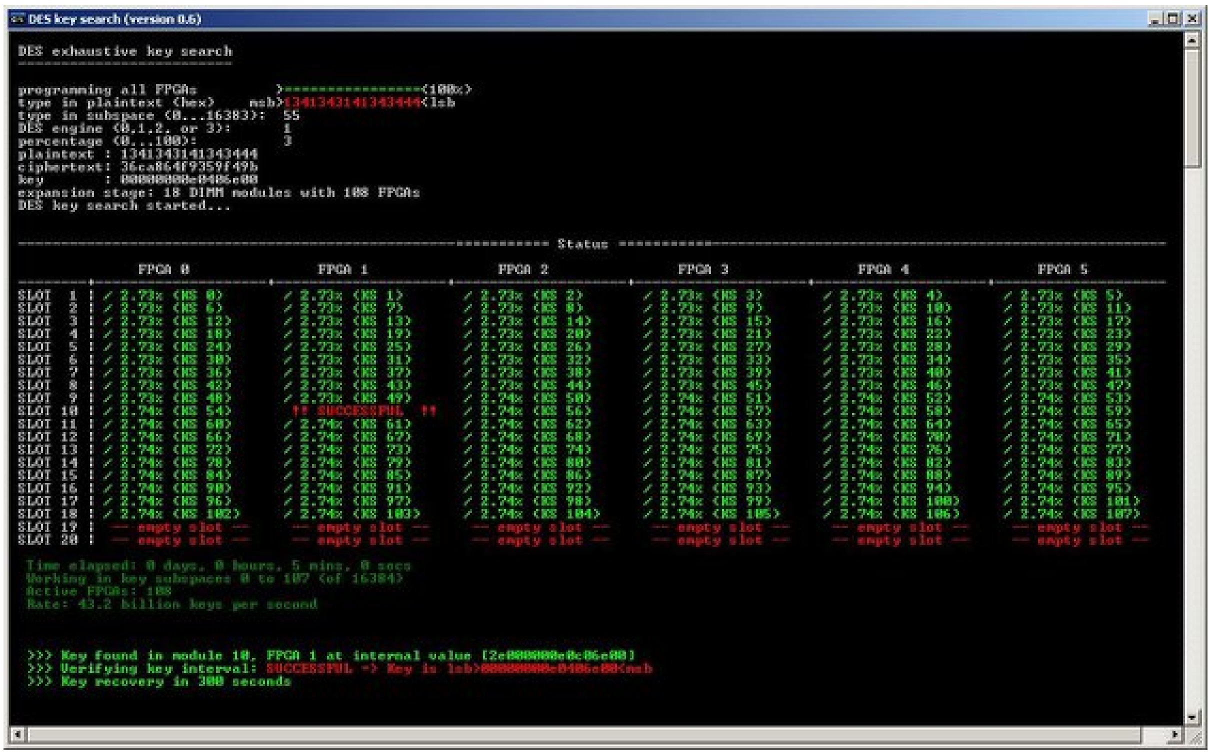
Besides, there are some low-level functions mainly for debugging purposes, which are briefly explained later in this section.

Once you start the application, the initial window will be too small. You can adjust the window size to allow for the application to fit perfectly (e.g., 200x60 character is a good adjustment).

3.3.1 Start Key Search

When starting a key search the first time, all COPACOBANA FPGAs (slaves) will be programmed with the `toplevel_fpga.rbt` image and wait for the first key subspace to be assigned. You are required to enter the plaintext, the ciphertext, the subspace to start (usually 0 when a new search starts), and a filename for storing the current project. Plaintext and ciphertext are entered by 16 hexadecimal characters, beginning with the MSB.

After the input, the application detects the expansion stage of COPACOBANA and assigns the key subspaces to all FPGAs which are available. The status is displayed on the console (see Figure 9). The displayed information comprises the entered plaintext and ciphertext and some progress information of every single FPGA and COPACOBANA as a whole. All FPGAs are polled every few ms and the displayed progress information is updated. The individual progress of every single FPGAs is displayed in an array from DIMM 1 (slot 1) up to DIMM 20 (slot 20) for FPGAs 1 to 6. After completion of a



```
DES key search (version 0.6)

DES exhaustive key search
==========================

programming all FPGAs          >==================<100%>
type in plaintext (hex)    msb>1341343141343444<lsb
type in subspace (0...16383):  55
DES engine (0,1,2, or 3):      1
percentage (0...100):          3
plaintext : 1341343141343444
ciphertext: 36ca864f9359f49b
key       : 00000000e0406e00
expansion stage: 18 DIMM modules with 108 FPGAs
DES key search started...

--------------------------------------------------========== Status ==========------------------------------------------------

                FPGA 0                  FPGA 1                  FPGA 2                  FPGA 3                  FPGA 4                  FPGA 5
---------+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------
SLOT  1 | √ 2.73% (KS 0)          √ 2.73% (KS 1)          √ 2.73% (KS 2)          √ 2.73% (KS 3)          √ 2.73% (KS 4)          √ 2.73% (KS 5)
SLOT  2 | √ 2.73% (KS 6)          √ 2.73% (KS 7)          √ 2.73% (KS 8)          √ 2.73% (KS 9)          √ 2.73% (KS 10)         √ 2.73% (KS 11)
SLOT  3 | √ 2.73% (KS 12)         √ 2.73% (KS 13)         √ 2.73% (KS 14)         √ 2.73% (KS 15)         √ 2.73% (KS 16)         √ 2.73% (KS 17)
SLOT  4 | √ 2.73% (KS 18)         √ 2.73% (KS 19)         √ 2.73% (KS 20)         √ 2.73% (KS 21)         √ 2.73% (KS 22)         √ 2.73% (KS 23)
SLOT  5 | √ 2.73% (KS 24)         √ 2.73% (KS 25)         √ 2.73% (KS 26)         √ 2.73% (KS 27)         √ 2.73% (KS 28)         √ 2.73% (KS 29)
SLOT  6 | √ 2.73% (KS 30)         √ 2.73% (KS 31)         √ 2.73% (KS 32)         √ 2.73% (KS 33)         √ 2.73% (KS 34)         √ 2.73% (KS 35)
SLOT  7 | √ 2.73% (KS 36)         √ 2.73% (KS 37)         √ 2.73% (KS 38)         √ 2.73% (KS 39)         √ 2.73% (KS 40)         √ 2.73% (KS 41)
SLOT  8 | √ 2.73% (KS 42)         √ 2.73% (KS 43)         √ 2.73% (KS 44)         √ 2.73% (KS 45)         √ 2.73% (KS 46)         √ 2.73% (KS 47)
SLOT  9 | √ 2.73% (KS 48)         √ 2.73% (KS 49)         √ 2.74% (KS 50)         √ 2.74% (KS 51)         √ 2.74% (KS 52)         √ 2.74% (KS 53)
SLOT 10 | √ 2.74% (KS 54)        !! SUCCESSFUL  !!        √ 2.74% (KS 56)         √ 2.74% (KS 57)         √ 2.74% (KS 58)         √ 2.74% (KS 59)
SLOT 11 | √ 2.74% (KS 60)         √ 2.74% (KS 61)         √ 2.74% (KS 62)         √ 2.74% (KS 63)         √ 2.74% (KS 64)         √ 2.74% (KS 65)
SLOT 12 | √ 2.74% (KS 66)         √ 2.74% (KS 67)         √ 2.74% (KS 68)         √ 2.74% (KS 69)         √ 2.74% (KS 70)         √ 2.74% (KS 71)
SLOT 13 | √ 2.74% (KS 72)         √ 2.74% (KS 73)         √ 2.74% (KS 74)         √ 2.74% (KS 75)         √ 2.74% (KS 76)         √ 2.74% (KS 77)
SLOT 14 | √ 2.74% (KS 78)         √ 2.74% (KS 79)         √ 2.74% (KS 80)         √ 2.74% (KS 81)         √ 2.74% (KS 82)         √ 2.74% (KS 83)
SLOT 15 | √ 2.74% (KS 84)         √ 2.74% (KS 85)         √ 2.74% (KS 86)         √ 2.74% (KS 87)         √ 2.74% (KS 88)         √ 2.74% (KS 89)
SLOT 16 | √ 2.74% (KS 90)         √ 2.74% (KS 91)         √ 2.74% (KS 92)         √ 2.74% (KS 93)         √ 2.74% (KS 94)         √ 2.74% (KS 95)
SLOT 17 | √ 2.74% (KS 96)         √ 2.74% (KS 97)         √ 2.74% (KS 98)         √ 2.74% (KS 99)         √ 2.74% (KS 100)        √ 2.74% (KS 101)
SLOT 18 | √ 2.74% (KS 102)        √ 2.74% (KS 103)        √ 2.74% (KS 104)        √ 2.74% (KS 105)        √ 2.74% (KS 106)        √ 2.74% (KS 107)
SLOT 19 |  -- empty slot --        -- empty slot --        -- empty slot --        -- empty slot --        -- empty slot --        -- empty slot --
SLOT 20 |  -- empty slot --        -- empty slot --        -- empty slot --        -- empty slot --        -- empty slot --        -- empty slot --
  Time elapsed: 0 days, 0 hours, 5 mins, 0 secs
  Working in key subspaces 0 to 107 (of 16384)
  Active FPGAs: 108
  Rate: 43.2 billion keys per second


  >>> Key found in module 10, FPGA 1 at internal value [2e00000000e0c06e00]
  >>> Verifying key interval: SUCCESSFUL -> Key is lsb>00000000e0406e00<msb
  >>> Key recovery in 300 seconds
```

Figure 9: DES-cracker console output

subspace, a new subspace is automatically assigned to the respective FPGA.

Once a key is found in some DES engine, the host-PC recomputes the retrieved value by a DES software reference and outputs the status.

You can always stop a running key search and continue (see next paragraph) at a later point in time. It is important that you first stop the application on the host-PC and then switch off COPACOBANA!

IMPORTANT: For running a key search, you (currently) cannot use COPACOBANA without the host-PC! The host-PC assigns the subspaces and keeps viable information of the key space, which can/ have to be reinitialized after stopping the key search.

3.3.2 Resume Key Search

At any time, a running key search can be be interrupted and continued at a later point in time. After entering the appropriate filename, the application takes care of initializing COPACOBANA to the state right before the interruption.

Remark: Before switching off COPACOBANA, you have to stop the host-PC application. The other way round will result in data loss, making the resume function fail!

3.3.3 Further Functions

Start demo key search

For testing COPACOBANA with a sample DES key search, a low-level function called `Start demo key search` is provided. This function basically allows you to generate a specific plaintext/ ciphertext pair which will be recovered at a particular (previously entered) point in time on a particular FPGA. For this purpose, you enter a plaintext and some specific data from which the application computes the corresponding ciphertext. This data consists of:

- A key subspace where the correct key should be allocated. Since the whole keyspace is partinioned in subspaces (by default, there are 16384 subspaces), you can easily estimate the time of a key recovery from the subspace where the key will be found. By taking the number of active FPGAs into account, the subspace number also determines the exact location *where* the key will be found (FPGA number, DIMM number).
- The relative position within the subspace (percentage, 0.0%...100.0%).
- The DES engine number (1,2,3, or 4) which shall find the correct key.

Once you are set, the application computes the correct key and the ciphertext accordingly and starts a brute-force key search as in the normal case. The only difference is that you already know the key for demonstration/ debugging purposes.

Example: Assume you want to let COPACOBANA (with 120 FPGAs, 100 MHz) run for 2 days and 7 hours to find a key in DES engine 2 on FPGA 3, DIMM 17.

- 2 days and 7 hours $= 55\ h = 3300\ min$, i.e., in 3300 minutes, you have already finished $\lfloor 3300/90 \rfloor = \lfloor 36,667 \rfloor = 36$ subspaces with each of the FPGAs.
- That makes $36 \cdot 120 = 4320$ subspaces in total.
- The subspace of interest shall be on FPGA 3, DIMM 17, which is subspace number $4425 + 3 + 17 \cdot 6 = 4425$. I.e., as subspace number enter 4425.
- Now, you have to specify the position within the subspace. Since you want to find a key in approximately 3300 minutes, you have to enter 66.7% as percentage (which is the remainder of 3300/90).
- As engine number, enter 2.

Now you are set! COPACOBANA will run for 3300 minutes or 2 days and 7 hours, respectively.

Reconfigure controller FPGA

If it is necessary to reconfigure the controller FPGA, this option reconfigures the controller FPGA with the standard image (`cesys_ctrl.exo`).

Reconfigure slave FPGAs

This function programs all slave FPGAs with the standard image (`toplevel_fpga.rbt`).

Display driver information

Use this function to displays details of the driver currently used to access COPACOBANA via the Cesys board.

Reset controller FPGA (Cesys)

Applies a reset to the controller FPGA. After resetting the controller FPGA, the slave FPGAs have to be reprogrammed.

Reset slave FPGAs

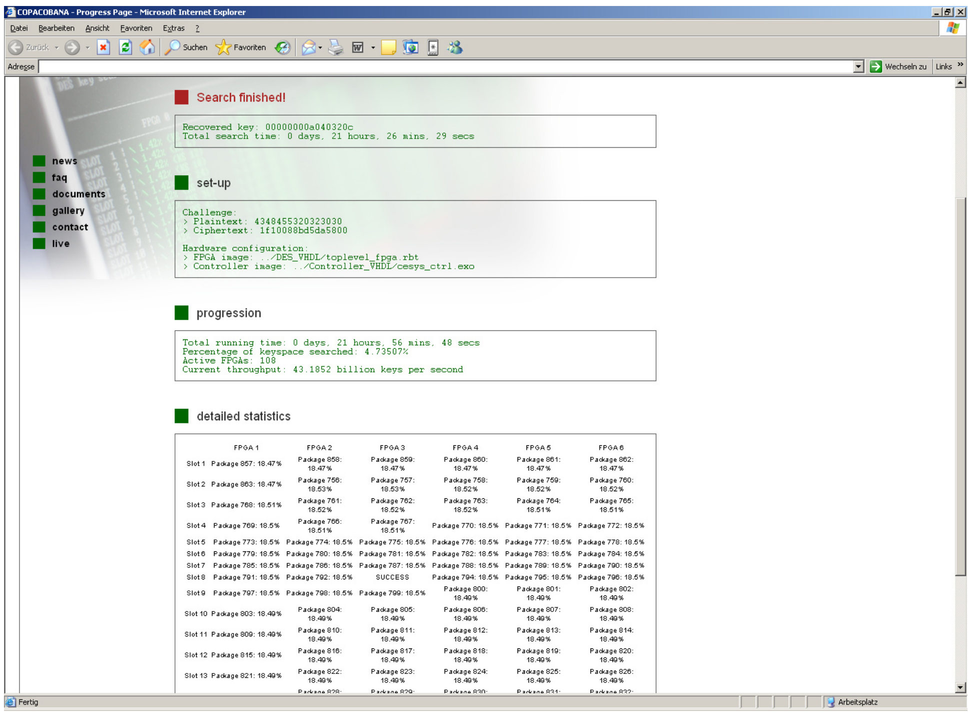
Applies a reset to all slave FPGAs and resets them to the IDLE state.

Show status of slave FPGAs

This function can be used to display the current status of all FPGAs. Do *not* use this function as a substitute of the key search function since data will not be saved in the project file and data loss may ooccur once switching off the host-PC. This function is for debugging purposes only.

3.3.4 User Interface

The actual user interface is very basic and redirects inputs and outputs from/ to a console in WindowsXP. For accessing the status of COPACOBANA remotely, you can use the HTML output function, which writes the current status to a HTML file as seen in Figure 10.
HTML output is enabled by default. To disable this function, you have to set the option (`HTML_OUTPUT 0`) in the `config.h` file. You can also provide a different (absolute/ relative) filename. By using some webserver (e.g., Apache), you can make this webpage accessible from the outside or use it additionally to the console output on the local PC (it does *not* replace the console output). You will need a style file (.css) for the output file to display correctly. The default style file `copa_progress.css` is installed in `COPACOBANA/DES-Cracker/html_output/`.


COPACOBANA - Progress Page - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Suchen
Favoriten
Adresse
Wechseln zu
Links
news
faq
documents
gallery
contact
live
Fertig
Arbeitsplatz

Search finished!

```
Recovered key: 00000000a040320c
Total search time: 0 days, 21 hours, 26 mins, 29 secs
```

set-up

```
Challenge:
> Plaintext: 4348455320323030
> Ciphertext: 1f10088bd5da5800

Hardware configuration:
> FPGA image: ../DES_VHDL/toplevel_fpga.rbt
> Controller image: ../Controller_VHDL/cesys_ctrl.exo
```

progression

```
Total running time: 0 days, 21 hours, 56 mins, 48 secs
Percentage of keyspace searched: 4.73507%
Active FPGAs: 108
Current throughput: 43.1852 billion keys per second
```

detailed statistics

	FPGA 1	FPGA 2	FPGA 3	FPGA 4	FPGA 5	FPGA 6
Slot 1	Package 857: 18.47%	Package 858: 18.47%	Package 859: 18.47%	Package 860: 18.47%	Package 861: 18.47%	Package 862: 18.47%
Slot 2	Package 863: 18.47%	Package 756: 18.53%	Package 757: 18.53%	Package 758: 18.52%	Package 759: 18.52%	Package 760: 18.52%
Slot 3	Package 768: 18.51%	Package 761: 18.52%	Package 762: 18.52%	Package 763: 18.52%	Package 764: 18.51%	Package 765: 18.51%
Slot 4	Package 769: 18.5%	Package 766: 18.51%	Package 767: 18.51%	Package 770: 18.5%	Package 771: 18.5%	Package 772: 18.5%
Slot 5	Package 773: 18.5%	Package 774: 18.5%	Package 775: 18.5%	Package 776: 18.5%	Package 777: 18.5%	Package 778: 18.5%
Slot 6	Package 779: 18.5%	Package 780: 18.5%	Package 781: 18.5%	Package 782: 18.5%	Package 783: 18.5%	Package 784: 18.5%
Slot 7	Package 785: 18.5%	Package 786: 18.5%	Package 787: 18.5%	Package 788: 18.5%	Package 789: 18.5%	Package 790: 18.5%
Slot 8	Package 791: 18.5%	Package 792: 18.5%	SUCCESS	Package 794: 18.5%	Package 795: 18.5%	Package 796: 18.5%
Slot 9	Package 797: 18.5%	Package 798: 18.5%	Package 799: 18.5%	Package 800: 18.49%	Package 801: 18.49%	Package 802: 18.49%
Slot 10	Package 803: 18.49%	Package 804: 18.49%	Package 805: 18.49%	Package 806: 18.49%	Package 807: 18.49%	Package 808: 18.49%
Slot 11	Package 809: 18.49%	Package 810: 18.49%	Package 811: 18.49%	Package 812: 18.49%	Package 813: 18.49%	Package 814: 18.49%
Slot 12	Package 815: 18.49%	Package 816: 18.49%	Package 817: 18.49%	Package 818: 18.49%	Package 819: 18.49%	Package 820: 18.49%
Slot 13	Package 821: 18.49%	Package 822: 18.49%	Package 823: 18.49%	Package 824: 18.49%	Package 825: 18.49%	Package 826: 18.49%
		Package 828:	Package 829:	Package 830:	Package 831:	Package 832:

Figure 10: DES-cracker html output

3.4 Source Code and Settings

All important settings can be changed in the application's main header file `config.h`. To make changes effective, the source code has to be recompiled. Simply use the sources from the directory `COPACOBANA/DES-Cracker/DES_CPP/`) and leave the compiled executable `des_host.exe` in the `COPACOBANA/DES-Cracker/DES_CPP/Debug`) directory. (This is the default setting for Visual C++.)

A complete documentation of the source code is given in [COS06a] in PDF format and in the `COPACOBANA/DES-Cracker/DES_CPP/doc/html` directory in HTML format.

3.4.1 The `config.h` Header File

Important settings can be made by setting constants in the
`COPACOBANA/DES-Cracker/DES_CPP/config.h`)
file accordingly and and by recompiling the executable. For a detailed and complete coverage, see [COS06a]. Most important constants subject to change are:

- `CONTROLLER_CONFIGURATION_FILE`: Relative or absolute path to the controller configuration file (.exo). The default setting is `''./Controller_VHDL/cesys_ctrl.exo''`.
- `SLAVE_CONFIGURATION_FILE`: Relative or absolute path to the slave FPGA image, used for all slaves. The default setting is `''../DES_VHDL/toplevel_fpga.rbt''`.

- `HTML_OUTPUT`: Enable or disable HTML output. The default setting is `1` (enabled).
- `HTML_FILE`: Output file for HTML output. The default setting is `../html_output/index.html`.
- `EXTENSION`: Number of DIMMs starting at slot 1. Please adopt to the hardware setting of COPACOBANA. The default setting is `20` (full extension).
- `KEY_SUBSPACES`: This number defines the granularity of the key search by setting the number of subspaces. This setting can only be changed when according changes are made to the bit size of the counter in the VHDL code. The default setting is `16384`.

4 Frequently Asked Questions

Frequently asked questions about the hardware, programming issues and commercial aspects of COPACOBANA. For updates, please also consult our web page [COS06b].

4.1 Applications

- *Q: Can COPACOBANA only break DES?*
  A: No, any symmetric cipher with up to (roughly) 64 key bits can be attacked with COPACOBANA. Moreover, cryptographically weak ciphers with higher key length can be broken, if the number of required steps is less than 2^{64}. This can be the case for ciphers with a weak key derivation function (e.g., AES with 8-character keys, etc...).

- *Q: Can I break AES or 3DES with COPACOBANA?*
  A: No in general. An exceptions is if part of the key is known or if the entropy of the key is less than roughly 2^{64}.

- *Q: Which mode of operations can be attacked?*
  A: Since the core modules of COPACOBANA can be reprogrammed, all possible modes can be attacked as long as the control logic does not require much memory. Simple modes such as ECB, CBC, CFB, and CTR are subject to attacks.

- *Q: Can I break RSA or Diffie-Hellman with COPACOBANA?*
  A: RSA and DH require quite different attack algorithms than symmetric ciphers. COPACOBANA is not the optimum machine for factoring large numbers or solving discrete logarithms. However, COPACOBANA can make sense as a co-processor for factoring attacks against RSA, as stated in [PŠK^{+}05].

- *Q: Can I break elliptic curve cryptosystems (ECC) with COPACOBANA?*
  A: In principle, the architecture of COPACOBANA is well suited for elliptic curve attacks such as the parallel Pollard rho method. However, COPACOBANA can only attack systems with up to roughly 112 bit (unless a part of the key is known). Most ECC implementation in practice use much longer keys, often between 160-256 bits.

- *Q: Can I run my own application on COPACOBANA?*
  A: Definitely! COPACOBANA is reprogrammable. Standard VHDL and Verilog code can be compiled and loaded onto COPACOBANA. If you buy a COPACOBANA machine, it comes with a complete communication framework for the FPGA modules, the controller card and the host-PC. Own applications can simply be plugged in the framework.

- *Q: Can each FPGA be programmed individually?*
  A: Yes and no: currently, programming can be accomplished only DIMM-wise. All DIMMs can be programmed simultaneously.

- *Q: How is the code (architecture) on the FPGAs generated?*
  A: We used VHDL and Xilinx tools to compile the bit file which is loaded onto the FPGAs.

- *Q: Can I use COPACOBANA for applications other than code breaking/ cryptography?*
  A: Yes, any problem that requires massive parallel computation can, in principle, be solved with COPACOBANA. One example is the Smith-Waterman algorithm for comparing DNA sequences (bio-informatics). However, the communication between the individual processors (FPGAs) is only moderately fast and has a relatively high latency. Many parallel computing problems require frequent interprocess communication, for which COPACOBANA is not optimum.

- *Q: Can I write my own application for COPACOBANA?*
  A: Yes! If you buy/ rent COPACOBANA, we provide a full framework for host-to-FPGA communication in which own applications and own code (VHDL for the FPGAs, C++ for the host PC) can be plugged in.

4.2 Hardware

- *Q: What was the objective when building COPACOBANA?*
  A: COPACOBANA was built with subject to minimum costs. By using off-the-shelf components such as FPGAs, non-recurring engineering costs (NRE) were drastically reduced compared to EFF's DeepCrack. Despite the high computational power of a single COPACOBANA machine (e.g., 48 billion decryptions per second), the power supply only requires approx. 600 Watts. DeepCrack requires much more than that. The physical dimension of COPACOBANA are pretty small. The beta-version fits into a standard 19" Rack (3 height units).

- *Q: How far is COPACOBANA different from EFF's DeepCrack?*
  A: Opposed to DeepCrack, COPACOBANA is entirely built of reprogrammable hardware. A single machine consists of 120 FPGAs (Xilinx Spartan3-1000) and a controller board (Xilinx FPGA with MicroBlaze) and can easily be reprogrammed with custom binary files, compiled from VHDL or Verilog.

- *Q: How much memory is available on COPACOBANA?*
  A: On the machine side, there is only the internal memory of the FPGAs and the memory of the controller board available. The usage of the FPGA's internal

memory is limited by the actual application running on it. Generally speaking, you will have a few 100Kbit per FPGA. To overcome the memory shortage, some of our architectures use the (global) memory of the host PC (hard disk, RAM, etc...) which has a high latency.

- *Q: Does COPACOBANA run stable?*
  A: Yes, it has been extensively tested. It runs at moderate frequencies (bus frequency up to 60MHz) in order to allow for a reliable communication.

- *Q: What is the power consumption of COPACOBANA?*
  A: A single machine consumes not more than 600 Watts. The actual power consumption depends on the application which runs on the FPGAs.

- *Q: Is heat a problem with COPACOBANA?*
  A: Not at all. The power dissipation is pretty low, the power supply is only 600 Watts. Currently, only standard fans are used to keep the FPGA's temperature low.

- *Q: What are the advantages and disadvantages of COPACOBANA compared to commercial super computers such as Cray or SG machines?*
  A: The optimum cost-performance ratio is the greatest advantage of COPACOBANA. However, not all computational hard problems can be fit on the COPACOBANA architecture since it is special (comparably low communication bandwidth, low local memory, global memory only on host PC).

- *Q: Can I run more than one COPACOBANA in parallel?*
  A: Yes! In the first version, COPACOBANA's host interface is USB and allows up to 127 COPACOBANA machines per USB interface. I.e., a multiple of 127 COPACOBANA machines per host-PC. In the revised version, COPACOBANA possesses an Ethernet interface and is, thus, not limited to 127 devices per USB interface.

- *Q: What kind of host PC is needed to controls COPACOBANA?*
  A: A standard PC running Windows or Linux is sufficient. For the first version of COPACOBANA we used Windows with a proprietary USB interface library. The second version can be used with any OS and a library for the network stack.

4.3 Miscellaneous

- *Q: Is COPACOBANA for sale?*
  A: Yes, please contact us for more details. You can also rent computing time on COPACOBANA.

References

[Ces06] Cesys. USB2FPGA development board. Available online: `http://www.cesys.com/resources/USB2FPGA.pdf`, 2006.

[COS06a] COSY - Chair for Communication Security, Horst Görtz Institute for IT-Security. Copacobana des-cracker reference manual, December 2006. Source code documentation for DES-Cracker.

[COS06b] COSY - Chair for Communication Security, Horst Görtz Institute for IT-Security. Copacobana website. `http://www.copacobana.org`, 2006. Check website for most recent updates on the COPACOBANA platform.

[PŠK$^+$05] J. Pelzl, M. Šimka, T. Kleinjung, J. Franke, C. Priplata, C. Stahlke, M. Drutarovský, V. Fischer, and C. Paar. Area-Time Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method. *IEE Proceedings Information Security*, 152(1):67–78, October 2005.

[Xil06] Xilinx. Xilinx webpack. `http://www.xilinx.com`, 2006.