The Kernel Accelerator Device

Ludwig Jaffé

www.openhardware.de

Ludwig Jaffé is 32 years old and currently living in Munich. He already holds a degree in Electrical Engineering and has some years of experience in designing computer boards in the architectures x86, sparc and power-pc. At present he is preparing for a master's degree in Systems Engineering. He used Linux as his favorite operating system since about Slackware 2.1.

Abstract

The KAD is a hardware based accelerator card which accelerates computers by executing recurring time consuming tasks in hardware. The KAD-hardware is a PCI-slot card with at least one reconfigurable FPGA (Field Programmable Gate Array) on it. An additional FPGA is used as PCI-bridge which is needed to handle the communication and the FPGA reconfiguration tasks. Depending on the task which is to be accelerated, the kernel-module to be used will load the appropriate open-source FPGA-firmware into the device (configuration).

For example if one wants to accelerate AES drive encryption she simply loads the kad_loop-aes kernel-module which does the computation intensive parts in the KAD. So the CPU has more time for other things.

KAD is an interdisciplinary development project which consists of kernel driver programming, FPGA programming (VHDL/Verilog) and hardware development. There is no working KAD at the moment, but our aim is to develop such a device under the GPL with much support from everyone who likes to do so.

The development will be carried out from bottom up by integrating existing components into the KAD. The first part of the Lecture will present the Architecture and the Technologies behind the KAD. The second part will discuss the concept and possible alternatives and variants.

INTRODUCTION

Why Do We Want The KAD?

Many things we like are too slow. This is because of the limitations of modern CPUs because they do all jobs in sequence – even if we use a concurrent operating system :-)
The following Examples illustrate the pain we feel without the use of reconfigurable computing.

Drive Encryption

Wanting a performant server one buys High-Speed(TM) SCSI-Drives with 15000rpm and U360-SCSI interface. Being paranoid, we encrypt everything on our server using loop-aes. The result: Although having a dual processor machine running at 2GHz each, file access becomes deadly slow. This is because of the software implementation of the AES-Algorithm. It is slow - even being coded in assembler.
And the worst thing is: we need to execute the algorithm for every little file transaction. Wanting to accelerate our file server we invest in Mips an buy a dual Opteron Board or the like. This involves around 200Watts of heat generated by the CPUs alone. Having a reconfigurable FPGA-card we can use an under clocked Athlon CPU which totals together with the FPGA-card to < 10 Watts. So an enterprise having lots of valuable Data saves much energy by using a cryptography accelerator.

Image Processing

The University of the Bundeswehr in Munich did a research project about a driver less van that finds it way through the roads by means of image processing. The van was equipped with a 19 inch rack full of Pentium-III class computers.
Properly scaled, a FPGA-Board could do the time critical processing and save much space and Energy. Maybe, one can use a single PC and some FPGAs on a PCI-card to control a VW Golf without the need of a driver.

In Short Words:

Every CPU-time-consuming recurrent task can be significantly accelerated by the KAD.
What is the KAD?

The KAD is a reconfigurable computer subsystem. As a reconfigurable computer it does its jobs in dedicated hardware.

How Do We Tell It Our FPGAs?

Programming a FPGA is rather complicated opposed to programming a CPU.

If we program a CPU we first write some code in C, run a compiler and a linker as result we get an executable that runs on the computer.

In the domain of FPGAs we write our code in a hardware description language which helps us to think in parallel processes. Commonly used languages are VHDL and Verilog.

To get executable code for FPGAs, a special tool-set is needed. These tools compile hdl, simplify logic structures, place the structures in the FPGA resources and route the signal lines inside the FPGA to the resources. Finally, a download file is generated which is needed to load the FPGA. This loading is called “configuring”.

Usable Tools are mostly expensive. But the FPGA-vendors (e.g. Altera, Lattice, Xilinx) provide free (as in free beer) tools which produce download files from VHDL or Verilog source. Unfortunately, these tools are intended to be used with a rather unstable and insecure operating system provided by a company in Redmond.

As we want to accelerate Linux and we want the KAD to be use able by any Linux user, the following concept was developed:

Ease Of Use Concept

The KAD is abstracted from the user land by using special kernel modules.

The normal use of the KAD is to simply load a special kernel module, which is designed to do the desired job. The module does all the transactions with the KAD. Depending on the application, the job done by the kernel modules is partitioned in an appropriate matter to use both domains time – the CPU and space – the FPGAs to do the job efficiently.

To talk to user land, the KAD-Kernel-Module uses all the technologies that conventional kernel modules use.

As Kernel-Modules are written by specialists to provide a widely used functionality like a file system or a SCSI-driver the KAD-firmware, which is the FPGA-download file generated by the FPGA programming tools, should be written by hdl-specialists.

The pre-generated firmware is then packaged together with the source code of the kernel modules to be compiled and used by anybody – just like conventional kernel modules.

As with the Linux kernel the hdl-sources are free (as in freedom) and anybody who wants to, can compile them using the free (as in free beer) FPGA-tools by the FPGA-vendors and the design documentation which comes with the KAD.

Versioning of the hdl-code is done by versioning the kernel modules. So if the hdl code is touched, it will be transformed into a firmware file. The old kernel module will be repackaged and announced as a new version. This is done to ensure that the firmware is always used with a compatible kernel module. Otherwise finding the right firmware image to the used kernel module may frustrate the common user. This is because of the additional layer of complexity involved with the reconfigurable hardware.

In Short Words:

Using the KAD is as easy as using a conventional kernel module.

How Does It Work?

To use the KAD, a special KAD-Kernel-Module has to be loaded. Then the module sets up communication with the KAD via PCI. Therefore the KAD has a PCI bridge which interfaces from PCI to the internal bus of the KAD. The internal bus will be implemented as wishbone bus (see www.opencores.org). The FPGA containing the PCI-bridge will not be user-reconfigurable as it is reserved for administrative functions inside the KAD. Reconfiguring the bridge-FPGA could easily lead into a dead lock situation as one can easily imagine.

After the PCI-communication is set up the kernel module reads the configuration and the status from the KAD status registers. Here the kernel module get also information if the KAD is used by another module and which resources are free. Also future hardware variants will be marked inside the status registers of the KAD.

If everything is fine, the KAD-Kernel-Module decides, that the KAD-card is compatible with the payload - the contained firmware image. It then starts the reconfiguration engine of the KAD and transmits the firmware image via the PCI-bus to the FPGA which is fee to be used. Obviously, a KAD-Kernel-Module can contain more than one firmware image to use more than one FPGA for the job. The other way round, a firmware image can be shared by two or more KAD-Kernel-Modules if they belong to a group of compatible

KAD- Kernel-Modules with only small FPGA usage.

Now having the firmware installed in the user FPGA(s) we can start using the KAD's power. Depending on the application the CPU-load can scale from simple memory mapped data transfers on PCI to quite heavy load if we need the CPU for additional tasks. If ever possible, we try to do all the dirty computing intensive work inside the FPGAs so we will find low CPU load in most cases. The user will talk to the kernel module like he does with conventional kernel modules.

If we for example have a KAD-Kernel-Module for loop-AES drive encryption, the user will load this module and mount the drives with user land programs like mount that came with loop-AES. The feeling talking to an encrypted loop-device will be different though: Because of hardware encryption the device will be nearly as fast as an ordinary device. When the user does not need the services of the KAD-Kernel-Module he unloads it by saying rmmod. Upon unloading the KAD-Kernel-Module the kernel-module sets up the registers of the KAD to mark the formerly used resources as free. If the resource is used by other KAD-Kernel-Modules the KAD will know it because of its register content. In this case the resource stays in the locked state and can only be freed, if all KAD-Kernel-Modules have been unloaded that need the resource.

If the modules did something security relevant like encryption it will wipe the keys which were in the KADs SRAM before marking the FPGA resources as free. Optionally the FPGAs can be overwritten with some dirt, but in normal operation the FPGAs are erased before configuration whenever a KAD-Kernel-Module is loaded that needs the resource.

What Does Reconfigurable Computing?

This paragraph will compare general-purpose and reconfigurable computing.

General-purpose Computers

have a fixed component, also known as the CPU, that does all the things the user wants it to do. Being a von Neumann machine the CPU sequentially fetches instructions and data from memory, processes them sequentially and stores the results back into memory.

The following example illustrates this with simple assembler code:

```
MOV     A,23        ;Load Register A
MOV     B,42        ;Load Register B
ADD     A,B         ;The sum is in Register A
MOV     Result,A    ;Store the Result
```

If we want to add (a+b and c+d) we need to execute the program twice, which means that we need twice the time for the calculation.

We see that general-purpose-computers step through a set of instructions in the *dimension of time*

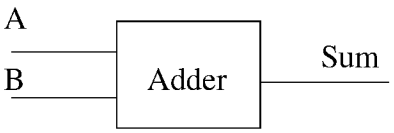
Reconfigurable Computers

program a programmable logic component (FPGA) to do all the things the user wants it to do. Being a piece of configured hardware the reconfigurable computer uses configured functional units and interconnects to do the job. In other words the reconfigurable computer has been configured to be a machine specialized on the problem to solve.

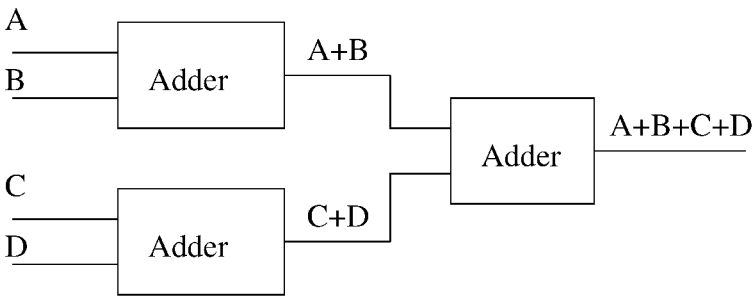
If there are other problems to solve, the machine has to be "re-specialized" for the new tasks. This "re-specializing" is called reconfiguration.

As long as there are configurable hardware-resources available the reconfigurable computer can be configured to compute in parallel. Though being a parallel machine, reconfigurable computers need a clock signal. This is because the should be - as every good design - implemented as synchronous logic. Using some state machines on our reconfigurable computer we can have best of both worlds: sequential and parallel operation, depending on our specific needs.

The following example shows our adder as parallel logic design depicted as schematic diagram:



If we want to add (a+b and c+d) the same time, we need two adders, which means that we need twice the space. To add a+b+c+d we need three adders which results in triple space:



The corresponding Assembler for a CPU is:

```
MOV     A,NumberA ;Load Register A
MOV     B,NumberB ;Load Register B
ADD     A,B       ;The sum is in Register A
MOV     RES1,A    ;Store the Result
MOV     A, NumberC
MOV     B, NumberD
ADD     A,B
MOV     B, RES1   ;We already have Result2 in A
ADD     A,B
MOV     Result, A ;Store A+B+C+D
```

So we need 3 addition steps in sequence. The time consumed by moving data depends on the Register Count of the CPU. If we have 4 registers we could save time.

But this example illustrates us the difference between *general-purpose-computing which happens in the domain of time* and *reconfigurable computing which happens in the domain of space.*

What Is So Special About FPGAs?

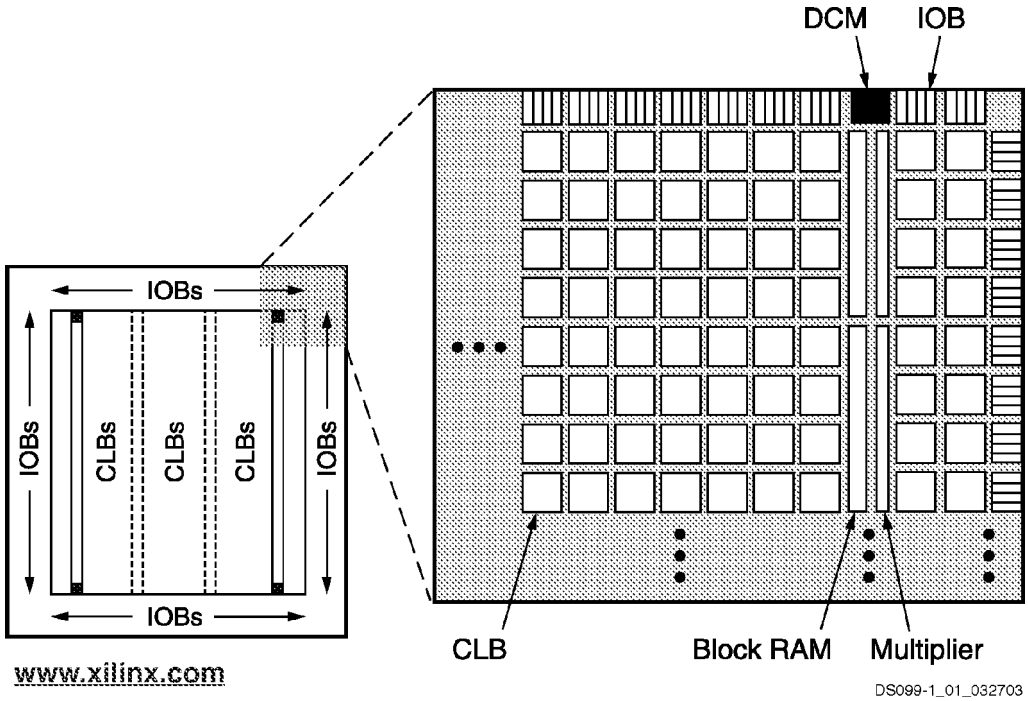
FPGA means **F**ield Programmable **G**ate **A**rray. Xilinx, a FPGA-vendor of the first time, called them LCA=Logic Cell Array in former times. The latter name depicts the principle of the FPGA better.

The FPGA consists of Logic Blocks and a programmable routing fabric to interconnect them. For the interconnection to the surrounding world a FPGA has a ring of I/O-Cells. The Logic Blocks have different names and granularity depending on the FPGA-vendor. Altera speaks of "Logic Elements (LE)" while Xilinx believes in Configurable Logic Blocks (CLB) which consist of four slices. A slice is equivalent to a Logic Block.

I/O-Cells are responsible for interfacing to various I/O-standards. They have sometimes additional features like feedback of the output inside the routing fabric or simply a output register. These features are vendor depended and for the most applications not that important as the vendors claim that they are.

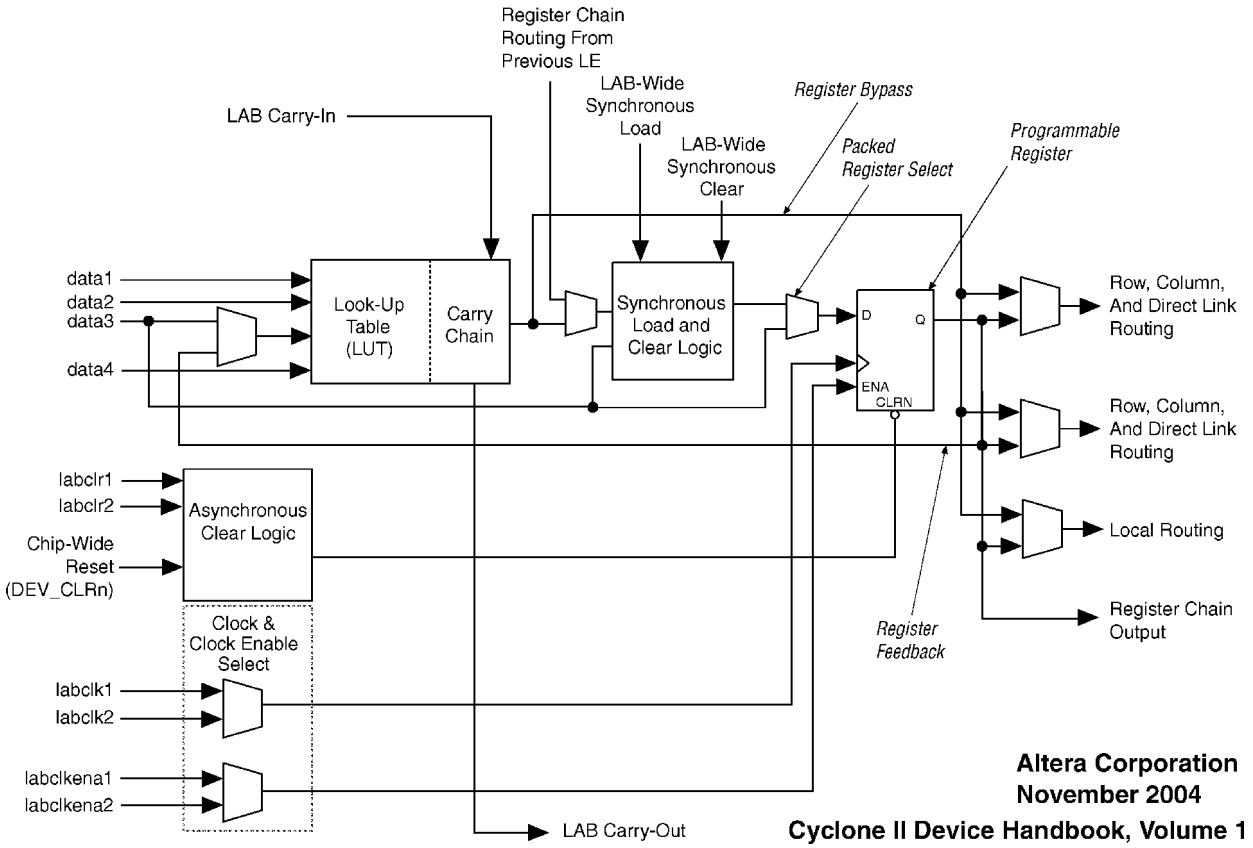
Additionally the vendors differentiate their products by adding extra functions such as PLLs, dedicated RAM or dedicated functions such as Multipliers. PLLs are useful to generate clock signals, while dedicated RAM-blocks are useful to build dual ported RAM or simply to provide a look-up-table. Multipliers are used for DSP-operations.

This picture shows the Structure of a Xilinx FPGA:



What Does A Logic Element Look Like?

The following picture shows the structure of an Altera Logic-Element:



As can be seen a Logic Element looks rather complex. But its main operation is quite simple: To store states it has a programmable Register. A clock allows the register to switch state depending on the clock and the input signals. The Lookup Table provides combinatoric logic.

The following example illustrates an AND function using a Lookup Table:

Input 1	***Input 0***	***Output***
0	0	0
0	1	0
1	0	0
1	1	1

We can see the Output-Collumn as a RAM, which is addressed by both Input-Collumns. So every RAM can represent combinatoric logic. But without a clock it will never

represents a state. With combinatoric logic and clocked registers any type of digital system can be build – even a CPU. Have a look at www.opencores.org to see many different library functions, so called cores. They even have open source cores of 32-Bit RISC-CPUs.

If you want to know more about the FPGA-structures of different FPGAs refer to the vendor's web pages. For example: www.xilinx.com and www.altera.com.

The Architecture Of The KAD

The System-View

The KAD has 4 layers, where two of them reside in hardware and two of them are software layers:

Layer 1: User configurable FPGAs (do the job: DSP-Functions, encryption etc...)

Layer 2: PCI-Bridge and PCI-Bus (transports the data to and from the FPGAs)

Layer 3: KAD-Kernel-Module (configures FPGAs and supports the FPGA operations)

Layer 4: User land application (load, unload kernel module, application specific stuff)

The Hardware-View

The payload of the KAD are the User-FPGAs. They are configured with user specific firmware to do the intended task. The User-FPGAs are interconnected using the Wishbone-SOC-Bus. The Wishbone-Bus was created by www.opencores.org as enabler for modular design reuse of open source cores. Because of this the Whishbone-Bus was chosen as Interconnect-Structure because the enables us to plug together a lot available cores. Furthermore the Wishbone-Bus is good documented, and well tested.

Another already available core is the PCI2Whishbone-Bridge which comes good documented with a hdl-configuration tool. (I think you got the philosophy already: We simply plug together great things to form an even greater one – The KAD.)

The PCI2Whishbone bridge is bus-master capable and interfaces the KAD to the computer via PCI-Bus. Being bus-master capable, the KAD can initiate Transfers to memory and also other devices on the PCI-Bus like NICs and Hard disk controllers without the help of the CPU. This feature enables the KAD to be an intelligent preprocessor for other PCI-Devices like SCSI-Controllers or NICs.

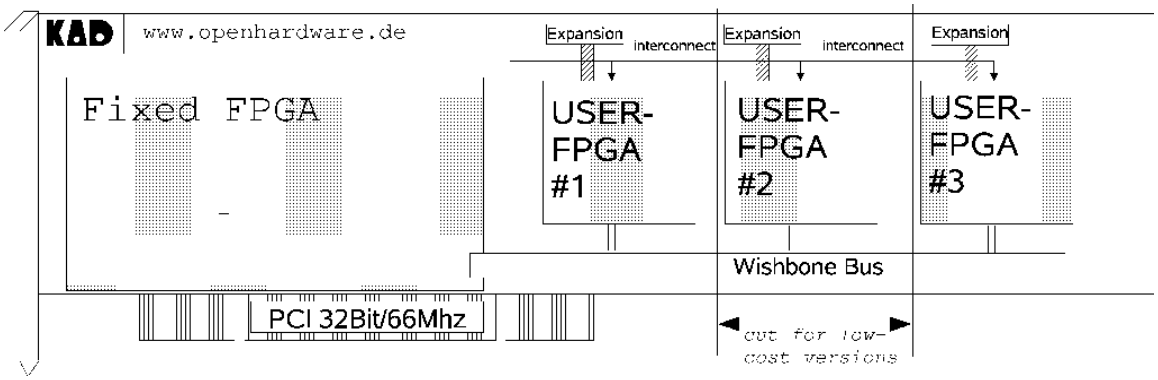
(Maybe, we will have a KAD-Kernel-Module to transform cheap Hard disk-Controllers into crypto-enabled RAID5-capable devices.)

Another Interconnect is the expansion-header, which every User-FPGA has. The Expansion can be interconnected via a pluggable Bus-Board to allow for Inter-FPGA-communication without utilizing the Whisbone-Bus. Furthermore, Plug-In-Modules can be connected to each FPGA's expansion bus to further enhance flexibility.

These Modules can be Hardware-Random generators, dedicated SRAM or even A/D-Converters and D/A-Converters, which allows the KAD to be not limited to crypto things. It can also be used as Measurement-Instrument, or Signal processor for software defined radio. You see, the KAD is a universal card that stimulates your creativity.

To scale costs and complexity, the KAD can be scaled by simply cutting unwanted FPGAs from the PCB. On the other hand we can route the Whisbone-Bus through a FPGA to the Expansion-connector which then allows us to connect slave cards with lot of further FPGAs via a fast serial link cable using LVDS.

The following picture illustrates the architecture of the KAD-card:



The Operating System-View

The User land-Interface to the KAD only depends on the intended application. Special Kernel-Modules abstract the KAD from the user. To do a certain job, the user only needs to know the user land-interface of the used kernel-module. In most cases the kernel-module will come together with an appropriate user land application.

To prevent chaotic versions, each version of a KAD-kernel-module is shipped together

with

a pre-compiled FPGA-Firmware that it will load into the FPGAs of the KAD. So every change either in FPGA-Firmware or Kernel-Module code will find its representation in the version of the KAD-Kernel-Module. This ensures compatibility, and stability of the whole system.

The VHDL-View:

The KAD uses free (mostly free as in freedom) IP-Cores to provide a fast time-to-“market” for the KAD-project by lowering project risk at the same time.
The cores are glued to the Whisbone-Bus with additional logic if they do not support the Whishbone-Bus already. Additional HDL-Code will be needed to create the reconfiguration-engine inside the PCI2Whishbone-FPGA. Some additional glue logic will help us to debug the KAD.

What About The Tools – They Are Not Free, Aren't They?

Yes, the tools are not free as in freedom, but there are tools that are free as in free beer. These are prove tools provided by the FPGA-vendors that unfortunately require an unstable non-free operating-system from Redmond. I think that we can live with this drawback for the sake of the project result :-)
The hardware-design will be done with GNU-EDA and GNU PCB because both tools are really free.

Open Source Hardware

Why the hell do we need open source hardware? Open source software should be enough. Well, having free hardware designs we can reuse the design for other projects. The conventional closed-source hardware can only be used for the purpose the vendor intended it for. The need for open source hardware can be seen in the tremendous effort people spend in reverse-engineering PDAs, GSM-phones and even scanners because the manufacturers refuse to give any little bit of useful documentation to the end user. Sometimes they provide a little bit of shit they call “users' guide”.
If they give useful information, they require you to sign a Non-Disclosure-Agreement, which stops the information from being freed.
As with GNU/Linux Open Source Hardware wants the people to engineer together new

solutions that suit their needs.

Resources About Open-Source-Hardware Can Be Found Here:

www.openhardware.de (The KAD on my open hardware page)

www.opencores.org (Many free IP-Cores)

http://sourceforge.net/projects/blowfishvhdl (a free blowfish core written in vhdl)

www.opecollector.org (Information about open source hardware)

http://www.geda.seul.org (GNU-EDA-Tool used for schematic circuit design)

http://pcb.sourceforge.net (GNU-PCB-Tool used for PCB-layout)