# A Prototyping Environment for Hardware/Software Codesign in the COBRA Project

Gernot Koch[1], Udo Kebschull[1], Wolfgang Rosenstiel[2]

[1]Forschungszentrum Informatik (FZI), Haid-und-Neu-Straße 10-14, D 76131 Karlsruhe, Germany
[2]FZI and University of Tübingen, Sand 13, D 72076 Tübingen, Germany

**Abstract**

*We present a prototyping environment with special benefit for hardware/software codesign which we use as target architecture in the COBRA project[1]. This architecture is very flexible, easy extensible, and provides a high gate complexity. It supports standard processor integration as well as processor emulation.*

## 1: Introduction

### 1.1: Prototyping in system design

In ASIC design since a long time there are many efforts to shorten the development process. In recent time, since programmable logic devices are known, these efforts include fast prototyping of ASIC's. Prototyping environments provide the possibility, to compile a specification very fast into an arrangement of programmable logic devices, to test it under realtime conditions in its real world environment, and to turn back to the specification for error correction. The result is a more reliable ASIC design in a shorter time.

The same evolution takes place in the domain of embedded system design. This domain includes the ASIC design as one part. Up to now an embedded system is specified in a non formal language. Then the designer decides in a heuristic way, what to implement in software or in hardware. Software and hardware design and synthesis are done independently. Weeks later, when the hardware is finished, the components are integrated and hopefully work together. Hardware/software codesign wants to integrate the hardware and the software synthesis in order to make the integration of the different parts easier. Our prototyping environment helps to shorten the synthesis process of an embedded system. Together with hardware/software codesign we get a more reliable design and a faster design cycle.

1. This project is sponsored by the European community under ESPRIT No. 8135

### 1.2: Scope of the COBRA project

The name COBRA is an acronym for COdesign Basic Research Action. This project is located in the domain of hardware/software codesign. It aims at the improvement of the knowledge in this domain. Therefore the main aims of this project are:

• Raising the design level of heterogeneous systems by raising the specification level from which the designer starts towards higher abstraction.
• The development of methods for the different design steps in the design of embedded systems, for the hardware/software partitioning, the synthesis of hardware and software parts, and the verification of a system level description.
• The develpment of a general design flow for hardware/software codesign.
• The modeling of the design flow with a framework environment and the integration of all COBRA tools into the framework.

The different objectives are depicted in figure 1. For the specification of a heterogeneous system three languages are considered:

• SA/VHDL, which is a combination of structured analysis (known from software engineering) and VHDL.
• UNITY and ST (synchronized transitions). Both are very similar and descibe a system as a set of concurrent assignments (or transitions).
• LOTOS, a language which allows to describe a system as a set of communicating processes. This language was developed for the specification of communication protocols.

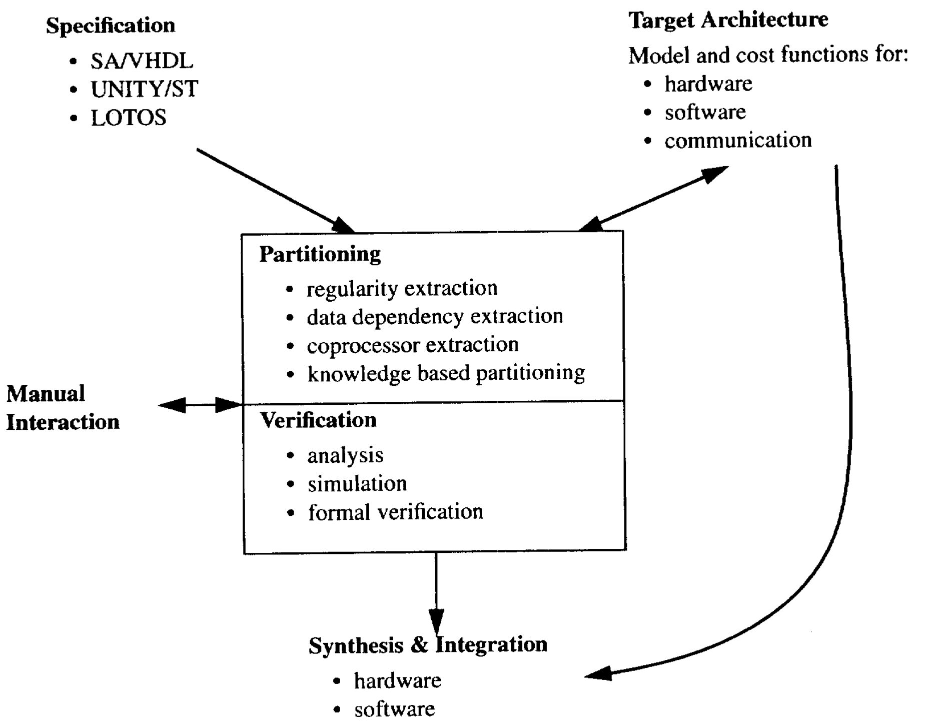

**Figure 1: Structure of the COBRA project [1]**

Unlike the existing hardware/software codesign approaches in this project a very flexible generic target architecture model is used. This model is described by several parameters and supports the partitioning process with cost functions for hardware, software and communication.

The hardware/software partitioning uses the specification to extract data dependencies and to define coprocessors for the application. The partitioning process is guided by the cost functions which are provided by the generic target architecture. Several distinct partitioning techniques exist and most of them are considered in COBRA:

• Partitioning by clustering: This approach defines a distance measure between UNITY elements and builts a clustering tree upon this distance measure. The cost functions are used to find a cut line in the clustering tree. The resulting clusters are assigned to hardware or software.[2]

• Partitioning by framework support: The partitioning is done manually but guided by framework assistance to enable an improved analysis and verification facility. The designer selects a partition and if it is found not to be satisfactory after analysis, a further partitition can be analyzed. In this way consecutive versions are obtained that are increasingly better. The total process is guided by a framework.[3]

• Co-synthesis for embedded heterogeneous multiprocessors: This method starts from the existing cosynthesis system COSYMA. The whole system is mapped to a single processor (software) and then hardware extraction is done to identify parts of the system to be partitioned to other processors or hardwired synthesized coprocessors. [4]

Verification aims at checking properties, e.g. functionality, timing, and resource requirements of a design prior to the realization. It is highly desirable to base as much as possible of the verification on high-level abstract descriptions. Three verification methods are taken into account in COBRA: Analysis goes down several steps in the design process and test with simulation there, whether the system meets its requirements or not. Simulation transforms the system level specification into a simulation model and stimulates this model. At the outputs of the model the designer can see if the system works correct. Both, analysis and simulation can only give an indication for the correctness of a system. A proof can be obtained by formal verification. This is done by mechanical verification tools.

The synthesis of the hardware and software parts is based on a model of the target architecture as well as on the output of the partitioning process. A very important point is given by the communication needs between the different parts. A communication model for the target architecture is needed.

## 1.3: Other approaches

Many different approaches for prototyping architectures exist. Most of them address the prototyping of hardware, not of embedded systems. Even if their gate complexity is sufficient for total systems, they have several disadvantages with respect to hardware/software codesign and prototyping of whole systems. First, until now there is no integration of microprocessors for the execution of the software parts supported. At least for the InCA system there is some work underway to integrate DSPs on daughterboards [5]. But for prototyping in hardware/software codesign the integration facilities are still not flexible enough and the interconnection scheme of InCA (and the others) makes it very difficult for the synthesis software to route buses which are usually introduced by the integration of microprocessors. Second the partitioning of the hardware parts isn't done efficient enough. The capacities of an architecture must be utilized in a better way. Third, the debugging facilities are not sufficient.

**General purpose environments:** The most important general purpose prototyping environments come from three companies. To each architecture a brief description is. They all have in common, that the interconnection is programmable and totally separated from the programmable logic. Most of them are extensible and allow the integration of application specific components.[6, 7]

• The **Aptix** company provides boards with up to 16 4010 Xilinx FPGAs for the configurable logic. The interconnection is done by a programmable interconnection chip which has 1024 pins and provides the ability to connect each pin with each other pin. The maximal usable number of gates is about 30K and the system can operate at frequencies up to 25 MHz. For debugging one has the possibilty to read out each pin of the interconnection chip. There is no chance to look into the configuration of the FPGAs.

• **InCA** uses Xilinx FPGAs for the programmable interconnection. The configurable logic is also located on Xilinx FPGAs and has a maximal size of 120K gates. The clockrate is maximal 15 MHz. The system can apply static test vectores and it provides the observation of probe points.

• **Quickturn** offers two different prototyping environments, the MarsIII system and the Enterprise system. Both provide a higher gate complexity than the other approaches. In both cases it is possible to combine several systems to a bigger system. The maximal complexity which can be achieved in this way is about 8M gates. Both use a programmable interconnection scheme and can be clocked with about 8 MHz. The MarsIII system has a probe capacity of 1024 channels and allows the observation of the internal shadow registers of the FPGAs.

**Special purpose environments:** Special purpose environments usually use a hardwired interconnection scheme and use the programmable logic devices for routing and logic. These systems are designed to implement hardware algorithms in a flexible way. They are not intended for the implementation of general purpose hardware but for the implementation of special things, i.e. coprocessors. Most of them include an amount of RAM and an interface to a host (VME, ISA, Sun SBus, etc.)

None of these environments is extensible or allows the integration of standard components. Especially for prototyping in the domain of hardware/software codesign the integration support of standard processors is missing. The most important examples are the PAM of the Research Laboratory of DEC, the SPLASH 2 of the IDA Supercomputing Research Center and the Virtual Computer of the Virtual Computer Corporation. [8, 9, 10]

## 1.4: Preliminary activities

At the University of Tübingen a FPGA based prototyping board called SPARROW is used [11]. This Board consists of:

• a 3195 Xilinx FPGA. This devices carries the configurable logic.

• 64 K RAM.

• 4 expansion slots for application specific expansions.

• standard interfaces like a centronics port, a DMA port, two serial ports and Xilinx programming and readback ports.

SPARROW is used for processor design and fast prototyping starting from VHDL descriptions. Thus it is a single board computer with a configurable microprocessor. It belongs to the group of special purpose prototyping environments. It is hardwired and not extensible in the sense of prototyping applications with a higher gate complexity than that provided by the Xilinx 3195 FPGA.

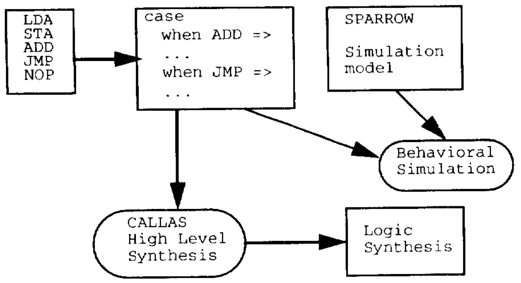

**Figure 2: SPARROW design flow**

The software environment of SPARROW includes a simulator model and a debugger. These tools allow to simulate an application before it is implemented on the SPARROW board. Also a configurable assembler and a simple multitasking kernel exist. The assembler can be configured for the particular processor which is located on the Xilinx chip

Figure 2 shows a part of the design flow with SPARROW. A processor description is given, which describes the instruction set of the processor. The designer has to transform this description into a behavioral VHDL description. This VHDL code is used together with the simulation model for a behavioral simulation. It is also the input of the CALLAS high level synthesis tool, which results in a register transfer level netlist. The RT netlist is compiled into the Xilinx chip via logic synthesis and technology mapping.

The design flow for the software part is as follows: The configurable assembler is configured for the particular processor, based on the VHDL description. Then the multitasking kernel is assembled by this assembler. All this is done on the host (a SPARCstation) with cross development tools.

The SPARROW has several properties which are important for a prototyping environment in the domain of hardware/software codesign. First, the configurable logic is used to implement processors. In that way it is possible to design a processor with an optimized instruction set for the application. This is very useful within hardware/software codesign, because in that way the processor can be designed in depencency of the software parts of the application. Second, it provides many standard components and expansion slots, which is necessary for prototyping arbitrary applications.

Although it has some interesting concepts, the SPARROW is not suitable for the COBRA project. The gate complexity which it provides is not sufficient for the design of entire embedded systems. The expansion capablilities are not suffient and not flexible enough.

## 2: Requirements on a Prototyping Environment

A prototyping system for hardware/software codesign must fulfill many requirements which exceed the normal requirements for a conventional prototyping system.

- It must be powerful and highly flexible. Otherwise it would imply constraints for the applications that it can carry. Only with a very flexible, extensible system it is possible to prototype arbitrary applications.
- It must provide a high gate complexity because the applications that are prototyped are entire embedded systems which would exceed the gate complexity of most of the conventional prototyping environments.
- It must be able to run as a stand alone system. If it carries an application, the possiblity is needed to test the application in its real world environment.
- It must be open for the integration of standard and application specific hardware components. It is not enough to provide interfaces. The supporting software must be able to include such extensions in the synthesis process.
- Arbitrary hardware debugging features are required. These include the ability to trace every signal and to read out the shadow registers of the FPGAs while the system is running in real time. One must be able to readback the configuration data of the FPGAs for analysis purpose.
- The prototyping system must be suitable for arbitrary logic applications and especially for processor design as a part of an embedded system. Processor design requires mainly software capabilities, but it needs also sufficient gate complexity.
- It must be possible to test an application which is prototyped in its real world environment. Therefore the prototyping system must provide a sufficient clock rate. The clock rate must at least be in the range of a few Megahertz.

Up to now no prototyping environment fulfills all these requirements. Although InCA and Quickturn provide enough gate complexity they do not really support the integration of application specific components or processor design as a part of the prototyped system. The special purpose environments are not flexible enough and do not provide enough gate capacity.

## 3: The COBRA Environment

The COBRA prototyping environment is designed especially for prototyping in the domain of hardware/software codesign. It is a modular and extensible system with high gate complexity. Because it fulfills the requirments on a conventional prototyping system it is also useful for hardware prototyping. The following contains a description of the hardware parts and the supporting software for this environment.

It uses a hardwired regular interconnection scheme. In that way less signals have to be routed through programmable devices, which results in a better performance. Nevertheless it will not always be possible to avoid routing of signal through the FPGAs. This must be minimized by the supporting software. For the interconnection of modules bus modules are provided. These offer 90 bit wide datapaths. This is enough even if 32 bit processors are integrated. Depending on the type of Xilinx FPGAs used, it provides 20K to 100K gates per board. Application specific components and standard processors can be integrated via their own boards which must have the same interface as the other modules.

For debugging it is possible to trace all signals at runtime, to read the shadow registers of the FPGAs and to readback the configuration data of any FPGA in the system.

A very interesting feature is the possibility to reconfigure particular FPGAs at the runtime of the system, without affecting the other parts of the architecture. Although this does not directly refer to prototyping, several publications in recent time have shown, that there is an interest in the domain of hardware reconfiguration at runtime. [12, 13, 14]

## 3.1: Hardware modules

The base module of the COBRA environment carries four Xilinx FPGAs for the configurable logic. On each side of the quadratic base module a connector with 90 pins is located. Each FPGA is connected with one of these connectors. Also every FPGA has a 75 bit link to two of its neighbors. A Control Unit is located on the base module. It does the programming and readback of the particular FPGAs. A separat bus comes to the control unit of each base module in the system. The programming data comes via that bus serially. This data is associated with address information for the base module and the FPGA on the base module. In that way the control unit can find out, if the programming data on the bus is relevant for its own base module and if so, it can forward the programming data to the FPGA for which it is intended. The readback of configuration data and shadow registers is done in the same way.

The ROM is used to store the configuration data for each FPGA of a base module. While startup the control unit reads out the ROM and programs each FPGA with its configuration data. Figure 3 shows the base module. Here it is equipped with Xilinx 4025 FPGAs which are not available yet. But in the same way it can be equipped with Xilinx 4013 FPGAs.

An I/O module provides a connection to a host. It supports the parallel Sparc S-bus. With this module the host works as I/O preprocessor which offers an interface for interaction to the user.

A RAM module with 4 MB static RAM can be plugged in for the storage of global or local data. It can be plugged in a bus module, so several modules have access to it in the same way, or it can be connected directly to a base module. Then this module has exclusive access to the memory. Requests of other modules must be routed through the FPGAs of the directly connected base module. This is possible, but time consuming.

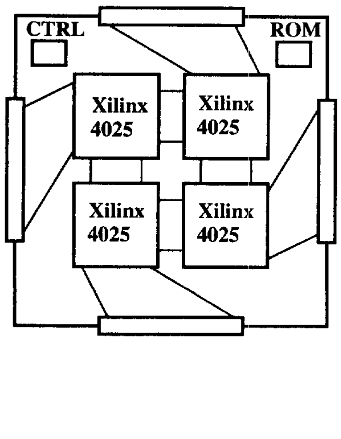

Figure 3: The base module

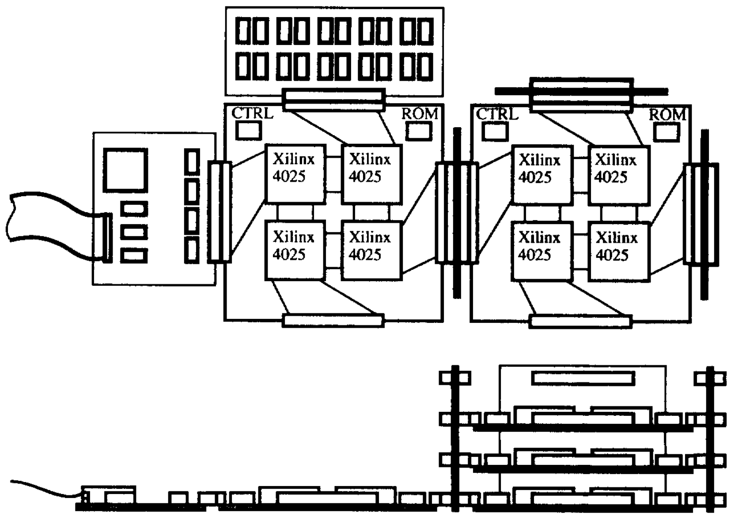

Figure 4: A more complex structure

The bus module makes it possible to plug modules in a bus oriented way together. With the possibility to have a bus on each side of the base module this architecture can be used to build multiprocessing systems with arbitrary structure.

For the integration of standard processors they must be located on their own modules which must meet the connection conventions of the other modules. A standard processor module for the Hyperstone 32 bit processor is already included in the set of module types.

Even though this list of module types is sufficient for many applications, there may be even more applications which require other types of modules like networking modules or DSP modules. One can think of any other type of modules and surely the list of existing module types will grow in the future.

With these modules arbitraty structures can be built. The structure depends on the application which is prototyped. So a running system contains all modules needed, but not more. That is important to reduce overhead and to keep the price low. An example of a more complex structure is depicted in figure 4. This picture shows an architecture in top view and in side view. As can be seen, the architecture is built in three dimensions and consists of four basic modules, a RAM module and an I/O module. On the right side is a tower of three basic modules which are connected via three bus modules. On the left side is another basic module with a local RAM module and an I/O module. The gross amount of gates in this example is about 400K. Thus it is an example which would be sufficient for many applications.

One may assume that the left basic carries an application specific software executing processor which works with the RAM on the RAM module and which communicates with software on a host for providing a user interface via the I/O module. The tower on the right side may carry some ASICs.

## 3.2: Supporting software

A very important part of the prototyping environment is the supporting software. The hardware is only the carrier of an application but the software has to solve all synthesis problems and management problems. Therefore different tools are needed:

• The hardware of the application must be partitioned to fit into the FPGAs of the architecture. The structure of the particular system must be an input parameter of the **partitioning software**. This tool was developed as a Phd work at the Computer Research Center of the University of Karlsruhe. [15]

• The structure which fits best for an application must be extracted from the specification. The **architecture synthesis** is responsible for the extraction of the information which module types an application requires and how they must be plugged together to get the best results.

• If a netlist is partitioned and a path between two different partitions must be routed through FPGAs, then a decision must be made, which of the possible ways the path should go. This **routing information** must be integrated in the programming of the FPGAs.

• The synthesis of the **FPGA configuration files** is done with conventional logic synthesis tools.

• After the hardware/software partitioning, the resulting behavioral hardware description must be synthesized via existing **high level synthesis tools**.

• The debugging is the job of the **hardware debugging software**. This software provides a user interface for the readback of configuration data, for signal tracing and for shadow register reading.

## 4: Results

The architecture is still in the development process. This holds especially for the software tools. Thus, it is not possible to give a full example for a running application. But the partitioning software is yet implemented and the design of the base module is ready, so it is possible to partition applications for this architecture and look at the results. We have mapped a VHDL description of a 16 bit processor to a base module. The processor consists of 5887 gates, 482 of them are latches and 5405 are combinational gates. Table 1 shows the results of the partitioning:

| | FPGA 1 | FPGA 2 | FPGA 3 | FPGA 4 |
|---|---|---|---|---|
| FPGA 1 | 336 | 58 | 0 | 67 |
| FPGA 2 | 58 | 229 | 30 | 0 |
| FPGA 3 | 0 | 30 | 271 | 16 |
| FPGA 4 | 67 | 0 | 16 | 476 |

**Table 1: Partitioning of a 16 bit processor on a base module.**

The diagonal cells show the amount of CLBs used in each FPGA. The other cells indicate the number of connections between the two FPGAs that denote the row and the column. The partitioning knows that the base module carries four FPGAs, so it doesn't try to push the total application into less FPGAs in order to save one though this would be possible. As can be seen it is easily possible to map this application on a base module with a hardwired interconnection structure.

## 5: Summary

In this paper we have presented a prototyping system which is especially useful for, but not restricted to, hardware/software codesign. In comparison to other environments it has the advantages that it is very flexible because of its modular architecture; it supports the integration of standard processors and the implementation of application optimized processors on FPGAs and it provides a non limited gate capacity.

The debugging facilities exceed those of other environments because they include signal tracing, configuration data readback and FPGA shadow register reading.

As an interesting side effect it is possible to change the configuration of particular FPGAs dynamically at runtime. This shows possibilities for further exploration and encouragement of the architecture.

## 6: References

[1] COBRA Project Synopsis, ESPRIT III, 1994

[2] E. Barros, *Hardware/Software Partitioning using UNITY,* PhD thesis, University of Tübingen, 1993

[3] J.C. Lopez, M. Jacome, S.W. Director, *Design Assistance for CAD Frameworks*, Proc. of the 1st European Design Automation Conference (EuroDAC), 1992

[4] R. Ernst, J. Henkel, Th. Benner, *Hardware-Software Cosynthesis for Microcontrollers*, IEEE Design & Test of Computers, Vol. 10, No. 4, 12. 1993

[5] S. Note, J. Van Ginderdeuren, P. Van Lierop, R. Lauwereins, M. Engels, B. Almond, B. Kiani, *Paradigm RP: A System for the Rapid Prototyping of Real-Time DSP Applications*, DSP Applications, Vol. 3, No. 1, 1. 1994

[6] D. Bittruf, Y. Tanurhan, *A Survey of Hardware Emulators*, Technical Note, ESPRIT Basic Research Project No. 8135, 1994

[7] H. Owen, U. Kahn, J. Hughes, *FPGA based ASIC Hardware Emulator Architectures*, School of Electrical and Computer Engineering, Georgia Institute of Technology, 1993

[8] P. Bertin, D. Roncin, J. Vuillemin, *Introduction to Programmable Active Memories*, DIGITAL Paris Research Laboratory, 1989

[9] J. Arnold, D. Buell, E. Davis, *Splash 2*, Proceedings of 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992

[10] S. Casselman, *Virtual Computing and the Virtual Computer,* IEEE Workshop on FPGAs for custom Computing Machines, 1993

[11] U. Kebschull, E. Schubert, P. Thole, W. Rosenstiel, *The Design and Implementation of an Educational Computer System Based on FPGAs*, Proc. of 4th Eurochip Workshop on VLSI Design Training, 1993

[12] X. Ling, H. Amano, *WASMII:A Data Driven Computer on a Virtual Hardware*, IEEE Workshop on FPGAs for custom Computing Machines, 1993

[13] P. Lysaght, J. Dunlop, *Dynamic Reconfiguration of Field Programmable Gate Arrays*, Field Programmable Logic Workshop, Oxford, 1993

[14] P. French, R. Taylor, *A self-reconfiguring processor,* IEEE Workshop on FPGAs for custom Computing Machines, 1993

[15] U. Weinmann, *FPGA Partitioning under Timing Constraints*, Field Programmable Logic Workshop, Oxford, 1993