# A Prototyping System for Verification and Evaluation in Hardware-Software Cosynthesis

Th.Benner, R.Ernst, I.Könenkamp, P.Schüler, H.-C.Schaub
Technische Universität Braunschweig
Institut für Datenverarbeitungsanlagen
Hans-Sommer-Str. 66, D 38112 Braunschweig, Germany
benner@ida.ing.tu-bs.de

**Abstract**

*We present a system emulator for rapid prototyping of small embedded HW/SW- systems with hard timing constraints generated by a HW/SW cosynthesis system. It consists of a standard core processor and an application specific coprocessor, which is emulated by XILINX FPGAs. A byte slice architecture allows to emulate rather complex coprocessors. The system emulator supports the prototyping, debugging and time measurement in a comfortable way.*

## 1 Introduction

The field of applications for embedded control such as automotive electronics, office automation and telecommunication is growing rapidly. Market pressure requires shorter design cycles from the specification to the final product. As a consequence, the design methods are changing. While in the past hardware and software were designed separately, it becomes now reasonable to start with a common description. The partitioning into hardware and software is performed in a later stage of the design process. It becomes possible to compare different design alternatives without changing the input description.

Embedded system verification is a complex task. While simulation on RT level may be acceptable even for complex designs, the simulation times are quite impracticable on the gate level. A system emulation makes it possible to prove the design much closer to real timing conditions.

In order to be correct, a hard real time system has to fulfill two conditions:

- It has to generate the correct output dependent on the input stimuli.
- All hard timing constraints have to be met.

In order to verify the design, a system emulator should support the validation of both conditions in a comfortable way. Therefore it should be equipped with suitable debugging and timing measurement facilities.

Several commercial emulator systems are on the market [In93] [Qu], which are quite flexible. Based on reconfigurable FPGAs and programmable interconnections, they can be adapted to various designs. In order to fulfill the requirements to an embedded real-time system verification they have to be extended by a processor core and by comfortable high-level debugging features. The resulting prototyping system becomes quite costly and itś development is time consuming.

The programmable interconnection of commercial prototyping systems allows the emulation of a wide field of system architectures but it also causes disadvantages. One major disadvantage is the delay of the connection elements. The programmable interconnection also aggravates the netlist partitioning problem.

The system prototyper which is presented in this paper does not require programmable interconnections between the FPGAs. As a consequence, it allows the emulation of systems with a clock rate of 1-10Mhz. The emulator serves as a verification tool for our experimental system COSYMA, which we present in the next section. Section 3 derives the requirements to a suitable system emulator, while section 4 introduces the system architecture. Section 5 gives an overview on the design flow. We will conclude with an example and a summary.

## 2 COSYMA

COSYMA (CoSYnthesis of embedded Micro Architectures) is one of the first systems for hardware-software cosynthesis [Er93]. It is targeted to the design of small embedded controllers. Given an input description in a superset of C, $C^x$, consisting of one or more tasks with timing constraints, and given a fixed core processor, COSYMA tries to map as much of the system as possible to software. When the time constraints cannot be met, it automatically partitions parts of the system description to an application specific coprocessor such that the timing constraints are met with minimized hardware overhead. The partitioning process regards and minimizes communication overhead. Significant speedups have been observed for real examples with more than 1000 lines of C code and a 33MHz SPARC as core processor. Figure 1 outlines the design flow. Partitioning is done on the basic block level using simulated annealing based on speedup, communication time and hardware overhead estimation. The software part is enhanced by a communication protocol, translated to C and compiled to object code. The hardware part is generated by high-level synthesis. The approach is flexible enough to permit the use of different high-level synthesis systems; currently these are OLYMPUS [Mi90] from Stanford taking a HardwareC description as input and our own synthesis system BSS [Ho93] taking a CDFG as input. A run time analysis is able to accurately estimate the execution time of the resulting hardware-software system on a clock cycle basis [Ye93] and thus shows if the timing constraints are met. Currently, we are integrating an approach to adapt the estimation in hardware-software partitioning to the actual results[He94]. Once the timing constraints are met, the high-level synthesis system generates a hierarchical netlist on the logic level consisting of the controller and the data path in SLIF format (Stanford Logic Intermediate Format) or Verilog. In order to verify this netlist, we originally used HW/SW cosimulation only. It turned out that logic level simulation of coprocessor and communication hardware is only possible for small examples in acceptable computation time, even though the processor is simulated at the register-transfer level. So we decided that for hardware-software co-verification, a hardware prototyping system would be very helpful.

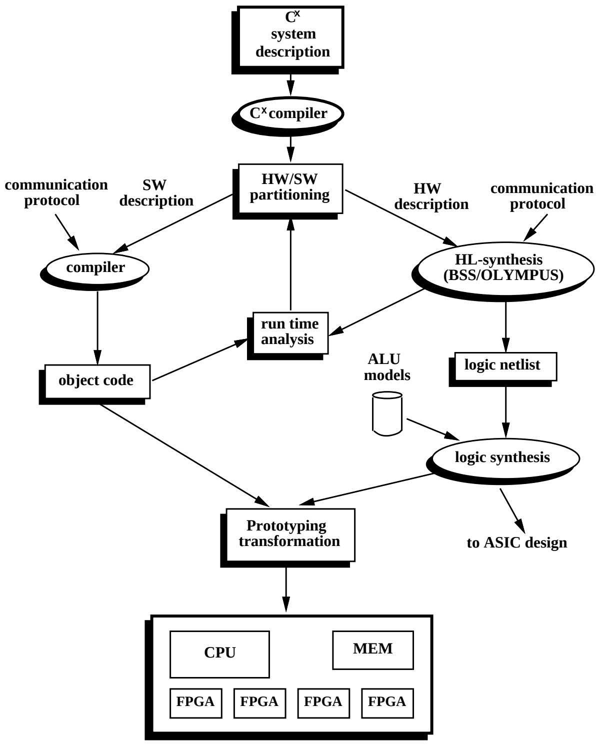


Figure 1: The COSYMA experimental system

## 3 Prototyping System Requirements

Because the generated hardware-software system uses memory coupling, it seemed reasonable not to use a very expensive commercial prototyping system, which would have to be extended by a SPARC core anyway, but to develop a specialized system. As the coprocessor is typically rather small, we concluded that it should be possible to design a hardware-software prototyping system that runs close to the real time (33MHz SPARC clock) and could be used for system evaluation in the real environment. Furthermore, symbolic debugging features with software breakpoints were considered essential for COSYMA verification. A major requirement was that the whole design flow from the high-level synthesis output should work with as little manual interaction as possible.

## 4 The Architecture of the Prototyping System

The prototyping system consists of three different synchronously clocked boards. Figure 2 gives an overview of the system. The LSI SPARC processor [LSI92] and the coprocessor board are memory coupled by RAM A and RAM B of the coprocessor board. The arbiter separates the processor bus from the system bus, which enables the processor to fetch an instruction from its instruction RAM, while the coprocessor is fetching one or two 32 bit words in the same clock cycle.

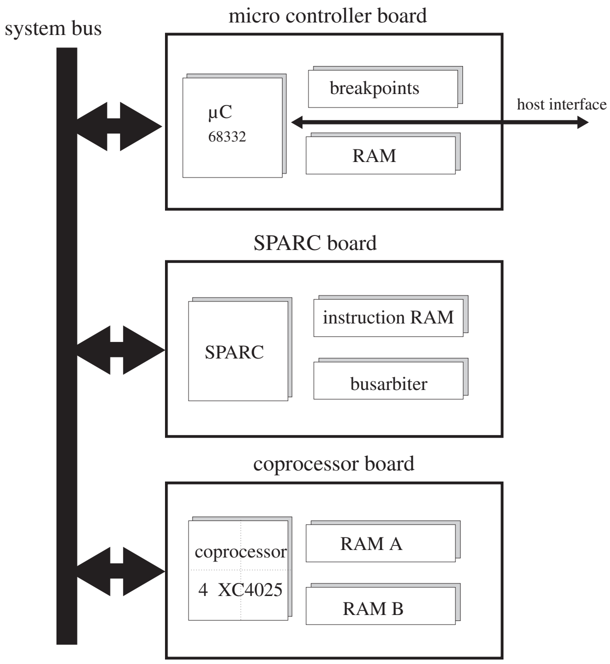


Figure 2: The architecture of the prototyping system

The current FPGA board contains four XILINX XC4010/4025[1] [Xi94]. We chose this family of reprogrammable circuits because of its size. For multiplications, we added an AMD 29C323.

[1] A 4025 board is under development.

For the purpose of time measurement, debugging and the connection to the host computer, a Motorola MC 68332 [Mo90] is used.

## 4.1 The FPGA Board

The major function of the prototyping system is the emulation of an application specific coprocessor. A typical design generated by COSYMA could include up to five ALUs and 20 registers. This exceeds the capacity of one single FPGA and we decided to spend four XILINX XC 4010/4025. When partitioning the design onto the four FPGAs the restricted amount of IO-Pins ($\leq$ 260 for the XC4025) has to be taken into consideration.

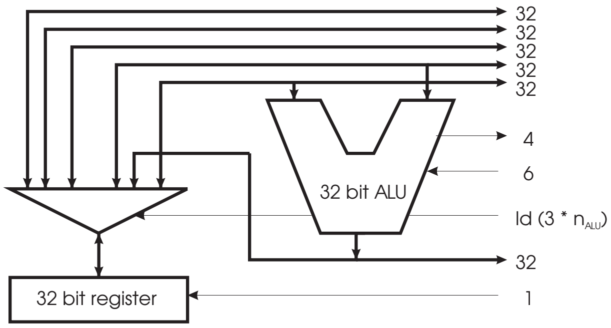


Figure 3: Partial model of a coprocessor.

Due to a high degree of parallelism of the application specific coprocessor, each ALU should be connectable to each register. In order to estimate the number of I/O Pins, we assume that a typical design consists of more registers than ALUs and come to the following simplified model. The coprocessor (fig. 3) consists of 32 bit ALUs and registers. Every external device like a RAM port or a multiplier is buffered by a register, which is also connected to each ALU. For each ALU, 3 * 32 bit have to be routed between the FPGAs. So the number of pins we have to spend for the data path is

$$n_{data} = n_{ALU} * 96,$$

while $n_{ALU}$ is the number of ALUs.

Additional to the data path the control signals are exchanged between the chips. We spent 10 pins per ALU and one pin per register. In order to connect the ALUs with the registers, multiplexer are required where the amount of select signal depends on the number of ALUs. For each ALU, both the input ports and the output port can be connected to each register in the worst case. So the number of control pins $n_{cont}$ is

$$n_{cont} = n_{reg} + 10 * n_{ALU} + \lceil ld(3 * n_{ALU}) \rceil.$$

Additional pins are required for the communication of the controller, which is also partitioned onto the four FPGAs.

Table 1 gives the sum of $n_{data}$ and $n_{cont}$. The numbers in braces are the required pins if the ALUs and

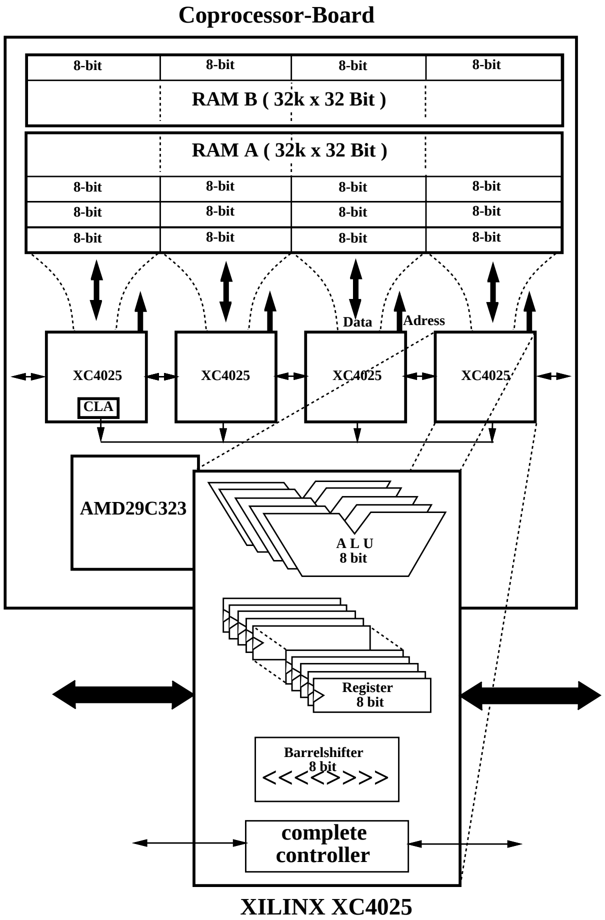


Figure 4: Architecture of the FPGA board

registers are not fully connected but only 20 %. In this case only the multiplexer select pins are saved. The table shows that a typical design consisting of five ALUs and 10-15 registers can not be partitioned in this way.

Instead of mapping complete ALUs and registers onto one single FPGA we chose a byte slice architecture (fig. 4). The FPGA board consists of four XILINX XC 4010/4025, each handling an eight bit slice of five ALUs and registers. So every FPGA holds up to five 8-Bit-ALU slices and an application dependent number of 8 bit register slices.

In order to implement a fast ALU, we compared a conventional CLA with XBLOX macros for adders, subtracters and comparators enhanced by logical operations. In [Xi94] the propagation delay of an N bit adder for a device with 5ns block delay is specified as

$$t_{pda}(N) = T_{INCY} + \frac{N}{2} * T_{BYP} + T_{SUM}$$

while $T_{INCY}$ is the delay from a CLB input to the

| | 1 ALU | 3 ALUs | 5 ALUs | 7 ALUs | 9 ALUs |
|---|---|---|---|---|---|
| 1 reg. | 108 (107) | 322 (319) | 534 (532) | 747 (745) | 959 (957) |
| 5 reg. | 116 (111) | 338 (323) | 550 (540) | 767 (757) | 979 (969) |
| 10 reg. | 126 (116) | 358 (328) | 570 (550) | 792 (772) | 1004 (984) |
| 15 reg. | 136 (121) | 378 (333) | 590 (560) | 817 (787) | 1029 (999) |
| 20 reg. | 146 (126) | 398 (338) | 610 (570) | 842 (802) | 1054 (1014) |

Table 1: Required number of I/O pins.

CARRY chain (4ns for 4010-5) and $T_{SUM}$ is the delay from a CLB input to the CARRY chain (6ns for 4010-5).

As a result, an 8 bit adder can be implemented on a single FPGA with a propagation delay of 16ns. An output propagation delay $t_{opd}$ amounts to 7ns while an input signal is delayed 3ns ($t_{ipd}$). The carry must be rippled through three IOBs (4 byte slices $\cong$ 32 bit). As a result, neglecting the external wires and internal interconnect delays, the propagation delay of a sliced 32 bit ALU can be estimated as

$$\begin{aligned} t_{pd} &= 4 * t_{pda}(8)ns + 3t_{opd} + 3t_{ipd} \\ &= 64 + 3 * 7ns + 3 * 3ns = 94ns. \end{aligned}$$

A whole ALU including the adder, subtracter, comparator and logical operations consists of 41 CLBs.

As an alternative, a second less area consuming ALU is included in our module library, based on a VHDL description of the 74HC181 which was synthesized using the Synopsys FPGA compiler. This version needs only 32 CLBs. Another advantage is that only pure logic but no hard macros are used which makes it possible to use the flipflops in the CLBs for registers. By spending a carry look ahead logic a higher performance is reached. For the evaluation of the generate and the propagate signals, $t_{gp} = 26ns$ are required while the CLA itself needs $t_{cla} = 12ns$. Neglecting the interconnection on a single device again, the propagation delays could be estimated as

$$\begin{aligned} t'_{pd} &= t_{gp} + t_{cla} + 2t_{opd} + 2t_{ipd} \\ &= 26ns + 12ns + 2 * 7ns + 2 * 3ns = 58ns. \end{aligned}$$

Another speedup could be reached by combining the XBLOX modules and a CLA. Caused by the spare recourses, we chose the conventional 74HC181 for the XC4010 board.

In order to handle the large number of control signals, a copy of the whole controller is included in each slice. This ensures that all control signals are generated on each slice. The carry signal of the ALUs is handled by a carry-look-ahead logic on one of the slices (no. 3). Additional flags for comparison reduce the controller overhead.

On every slice there are two RAM ports. Each of the ports is able to read or write 8 bits of a 32 bit word. The memory control signals are generated on one of the slices (no. 2). The memory consists of two banks, whose address spaces can be mapped in two different ways. The address space of port B can be mapped following port A. Another possibility is to address port A and port B alternately in steps of four bytes for interleaving. The address modes are configured by the microcontroller.

### 4.2 Communication Between the SPARC and the FPGA Board

Currently, processor and coprocessor communicate through shared memory with mutual exclusive access. In order to increase the parallelism in the future, every slice uses two separate memory ports. The coprocessor is able to fetch two 32 bit values concurrently. The two memory banks are mapped into the SPARC address space. A separate instruction memory is inserted, allowing the SPARC to fetch an instruction during a data fetch of the coprocessor.

### 4.3 Time Measurement and Debugging

As mentioned above, a microcontroller is used for time measurement and debugging. One interrupt channel of the MC68332 is triggered by a tag RAM, which contains the addresses of up to eight breakpoints. Whenever one of eight maskable tag breakpoints appears on the address bus, the microcontroller receives an interrupt while the processor and the coprocessor are stopped in the same cycle. Then the user is able to read the memory map.

The hardware breakpoints are also used for time measurement by starting and stopping one of the MC68332 timers at specified tag RAM entries. Each breakpoint can be used either for time measurement or for debugging. The state of the system bus at a breakpoint event is stored in a register which can be read by the microcontroller.

At the beginning, the microcontroller loads the memory of the SPARC and the coprocessor, while the FPGAs are simply programmed by a XCHECKER cable.

## 5 Design Flow

The result of the partitioning process in COSYMA consists of software for a processor core, and a coprocessor. For the software parts, COSYMA generates ANSI-C code which is then compiled.

As an output of high-level synthesis [Mi90, Ho93], the coprocessor is described as a logic level netlist (SLIF). Because tri state signals are a very spare resource on a XILINX, the registers and the ALUs are connected by multiplexers.

| | No. | CLBs p. item | $\sum$ CLBs |
|---|---|---|---|
| ALU | 3 | 29 | 87 |
| CLA | 3 | 10 | 30 |
| Register (8 bit) | 16 | 4 | 64 |
| Controller (synthesized allone) | | | 100 |
| Controller (real allocated resources) | | | 67 |
| Allocated Resources | | | 279 |

Table 2: Resources on Slice 3

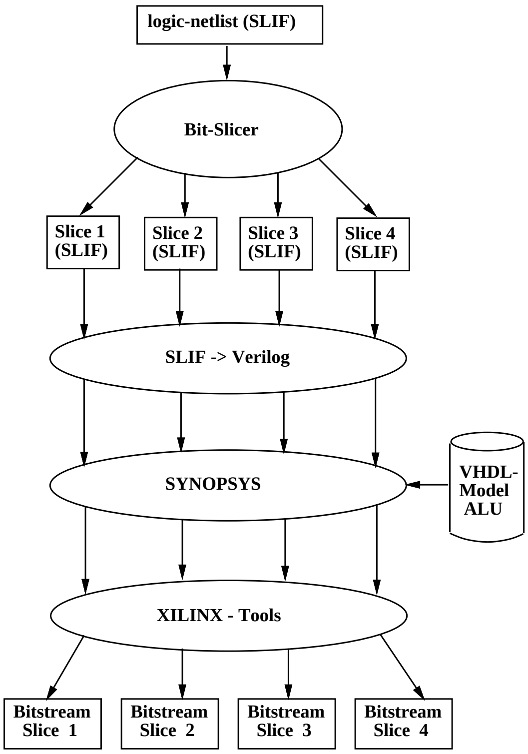


Figure 5: The design flow

Given the predefined byte slice architecture, partitioning is straight forward and can be performed with a "perl" script (fig. 5). After partitioning and translation to Verilog, logic optimization is done by the Synopsys$^{TM}$ system on each of the FPGA-netlists separately. At this point VHDL models of the ALUs and the barrel shifter are included. The synthesized slices are mapped to the FPGAs. The output of the Xilinx tools is a bitstream, which is then downloaded to the FPGAs controlled by the MC68332.

## 6 A practical example

For demonstration we present a practical example which is partitioned by the COSYMA system. The

```
/* ============= area 30c ============= */
for (i = cr1; i <= cr2; i++) {

  ihilf = labs(cr - i) + labs(cb - vtab[i]);
  if (ihilf < iabsv) {
     iabsv = ihilf;
  }
}  /*of for i*/

iabsh = 512;   /* = 256+256 */

/* ============ area 30d ============ */
for (i = cr1; i <= cr2; i++) {
  ihilf = labs(cr - i) + labs(cb - htab[i]);
  if (ihilf < iabsh) {
    iabsh = ihilf;
  }
}  /*of for i*/
```

Figure 6: Loops in chroma-key algorithm.

example is a chroma-key algorithm for high-definition television studio equipment. The desired response time was 1 second. On a Sparc 1+ it needs 3 seconds. The program has 1400 lines of C code, but about 90 % of the computation time is spent in the loops 30c and 30d of figure 6.

The high level synthesis, BSS, uses three ALUs in order to reach the required speedup of 3.0. Table 2 gives an overview of the allocated resources on slice 3, which is the largest slice, because it holds the carry look ahead logic (CLA) of the ALUs. For each ALU the function tables of 29 CLBs are allocated, while the CLA takes function tables of 10 CLBs. Each register uses the flip flops of four CLBs, but they can share the CLBs with the ALUs. We give two different values for the controller. If it is synthesized alone, it allocates one hundred CLBs, but due to resource sharing it takes only 67 ALUs for the controller, if the whole design is synthesized. The sum of all resources including multiplexer for the interconnections is 279 CLBs. The worst case pad to pad delay is 133 ns.

In order to save I/O pins, a copy of the whole controller is installed on each slice. The controller multiplication overhead of 20-30 % of the design area is the price we have to pay for the byte slice architecture.

## 7 Current status and conclusion

We have implemented a first wire-wrap prototype, which can be clocked at 1-10 Mhz depending on the design. Because the XC4025 are not yet available, we used the XC4010 instead. By replacing them by four XC 4025, an internal multiplier could be implemented on the chips and a large number of registers.

Due to the byte slice architecture and the fixed connections between the FPGAs, clock rates up to 10 MHz can be reached. The partitioning is quite simple. Several scripts allow a design flow from the unpartitioned netlist down to the boards with few interactions of the designer. The whole synthesis of a correct specification takes typically 4-10 hours depending on the design.

## 8 Acknowledgement

J. Herrmann and P. Meier were responsible for the mechanical construction of the prototyping system and H.-O. Leilich, N. Serafimov and F. Rabe gave very helpful hints when we debugged the hardware.

## References

[Er93] R. Ernst, J. Henkel and Th. Benner, Hardware/Software Co-Synthesis for Microcontrollers, IEEE Design & Test of Computers, pp. 64–75, Dec. 1993.

[He94] D.Herrmann, J.Henkel, R.Ernst, *An Approach to the Adaption of Estimated Cost Parameters in the COSYMA System,* Third International Workshop on Hardware/Software Codesign, Grenoble, Sep. 22-24, 1994.

[Ho93] U. Holtmann, Hierarchical Behavioural Representation in the Braunschweig Synthesis System BSS, IFIP Workshop on Application of Synthesis and Simulation, Lenggries, 25-28.8.1993.

[In93] InCA Ltg, VS-II Logic Emulator, 1993.

[LSI92] LSI Logic Corporation, L64831 - SPARC Integrated IU/FPU Technical Manual, Milpitas, Calif. 1992.

[Mi90] G. De Micheli et al., The Olympus Synthesis System, IEEE Design & Test of Computers, pp. 37–53, Oct. 1990.

[Mo90] Motorola Inc., MC68332 User's Manual, Phoenix, Arizona,1990.

[Qu] Quickturn Design Systems Inc., Enterprise Emulation System

[Xi94] Xilinx, Inc., The Programmable Logic Data Book, San Jose, Calif., 1994.

[Ye93] W. Ye, R. Ernst, Th. Benner, J. Henkel, Fast Timing Analysis for Hardware-Software Co-Synthesis, Proc. of ICCD 1993, Cambridge, pp. 452–457 , 1993.