Design Reuse by Modularity: A Scalable Dynamical (Re)Configurable Multiprocessor System

Rolf Drechsler Nicole Drechsler Elke Mackensen Tobias Schubert Bernd Becker

Institute of Computer Science
Chair of Computer Architecture
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{drechsle,ndrechsl,mackense,schubert,becker}@informatik.uni-freiburg.de

Abstract

We present a scalable, low cost multiprocessor system, which is used in the area of measurement, regulation, controlling and soft computing. The system is an example of extremely modular hardware/software design. Modular in this context means that we can easily change or optimize individually the components in the system for a given problem. One of the most important aspects of our multiprocessor system is the fact, that the user can dynamically change the communication topology by software 'on-the-fly'. Thus, the hardware can be (re)configured during operation in less than 1ms. By this, the CPUs in the multiprocessor system can communicate directly without any invention of another processor preventing the typical bottleneck in parallel systems. First applications have been implemented in an embedded controller for a vote counting machine with emphasis on high data security and high throughput.

1 Introduction

Parallel hardware is getting more and more popular, since many hard problems require large computational power. Many CPUs in parallel offer new opportunities to handle large problem instances.
One of the major drawbacks of most parallel systems is that the communication topology is fixed (see e.g. Transputers). On the other hand different problems often work only efficiently with different hardware requirements [2, 3]. In recent studies it has been shown by simulations that for varying applications different communication topologies should be preferred (see e.g. [1]). Thus, there is need for parallel systems that can be dynamically adapted to a given problem description.
In this paper we present a dynamical (re)configurable, low cost multiprocessor system. The system is based on the HSB bus definitions [1]. Small RISC type CPUs, the so-called Processor Nodes (PNs), are the basic computing units of the system. The PNs are put on an carrier board. As a carrier board we used a long PC ISA card. Thus the multiprocessor system can be used as a plug-in in each PC, but it can also be run stand alone. Up to 9 PNs can be used on the carrier board. Several cards can be run in parallel on a single PC. Coupling carrier boards across PC boundaries is also possible. The communication channels between the PNs can dynamically be switched by a Field Programmable Interconnection Device (FPID) realizing a crosspoint switch. The controlling of this FPID is done by a Communication Processor (CP), that is also placed on the carrier board. It is important to notice that the CP controls the information exchange between the PNs in the multiprocessor system, but does not actively take part. PC-based software tools have been developed that allow in a comfortable way to program the multiprocessor board. As programming languages assembler and C can be used. A *Win95* based environment is available for software development and debugging. The user can flexibly change the communication topology and download programs to the PNs. The system is a very good example for design reuse based on modularity. The board can be reconfigured dependent on the application. The multiprocessor system finds application in the area of measurement, regulation and controlling. Due to the execution speed the multiprocessor system is also suitable for real-time-applications. We discuss an industrial application based on these principles that has already been implemented successfully.

The paper is structured as follows. The definition of the HSB buses, that is the underlying bus structure in the system, is given in Section 2. In Section 3 the architecture of the multiprocessor system is introduced and the main components are presented. The communication structure is described in Section 4. An industrial application is presented in Section 5. Finally, in Section 6 we summarize the main results.

2 Modular Hardware

In this section, we describe the *Hierarchical System of Busses* (HSB), which builds the core structure of the hardware and determines the modularity of the system by definition. The HSB standard originally has been defined to support embedded control systems in small to medium volume applications. Major design goals were the reuseability of hardware by means of an extremely modular design, the support for multiple bus protocols and multiple processors in one system, and finally, small module sizes to complement the ongoing decrease in size of electronic circuitry, while its complexity increases.

HSB definitions arrange the overall structure of a computing system into four hierarchically ordered layers, each with its own bus definition:

- **Layer 1:** HSB-AB, the local CPU bus,
- **Layer 2:** HSB-BB, the system bus,
- **Layer 3:** HSB-CB, the controller bus, and
- **Layer 4:** HSB-DX, the driver 'bus'.

The local multiprotocol CPU bus, HSB-AB, is suitable for multiplexed or non-multiplexed 8-32 bit processors. HSB CPU modules are credit card sized (95 * 55 mm^2).
Any of the standard bus definitions available on the market can be used for the system bus HSB-BB (e.g. AT96, PC-ISA, STD, STE, or VMEbus, to name only a few). In embedded control applications the system bus may even be omitted, thus giving room to additional IO pins. According to the system bus chosen, HSB carrier boards may vary in board size and shape.
The controller bus HSB-CB is capable of running multiple bus protocols also. It uses an 8 bit wide multiplexed data/address path. Single width HSB controller modules measure 94 * 23 mm^2. Double width HSB controllers allow for 16 bit transfers. Programmed IO, interrupts, and DMA transfers as well as IO coprocessors are all supported by the HSB-CB controller bus, as are auxiliary serial busses like I^2C-, SPI-Bus, or Microwire. A common trigger signal can be used for synchronizing all clocks or AD converters in the system.
The driver 'bus', HSB-DX, is no real bus in most instances, but allows for the unified connection of HSB controllers to their respective HSB driver modules. This can be done in a daisy chain or party line manner, or any other scheme appropriate for the application. Thus, a multichannel serial controller can be equipped with different driver modules on a channel by channel basis.

In a configured HSB system the local CPU and the controllers can concurrently run different bus protocols in an arbitrary mix. Incompatibilities of the various bus protocols used are resolved by a small PLD on the HSB carrier board. If the PLD is being replaced by a more sophisticated FPGA, the HSB definitions allow for autoconfiguration, also.

3 Multiprocessor System Architecture

In this section, we describe the most important hardware components of our multiprocessor system (see Figure 1). The system is characterized by its modularity. It mainly consists of three elements, i.e. the Carrier Board (CB), the Processor Nodes (PNs) and the Communication Processor (CP), that will be described below.

Before these components are explained some general properties of the system are discussed:

- Each processor, i.e. the PNs and the CPs, is located on separate boards and for this can easily be exchanged or in the case of PNs even be removed. Dependent on the application other modules can be used and by this it is allowed to reuse large parts of existing hardware also in new applications/projects.
- The communication topology of the whole system can be reconfigured 'on-the-fly' in less than 1ms. Notice that this is not possible with most commonly used parallel computers, like e.g. Transputers. Furthermore, we make use of a Field Programmable Interconnection Device (FPID) realizing a crosspoint switch (the device is located below the CP on the CB). The FPID allows to establish a real hardware connection between PNs that want to exchange information. Thus, the typical bottleneck of parallel computers is prevented (the communication between the PNs is described in more detail in Section 4).
- The systems is dimensioned such that up to 81 processors (corresponding to nine CBs) can be run in parallel.
- For providing a comfortable user interface the CB can be plugged in a PC, but it can also be run in a stand alone mode.

3.1 Carrier Board (CB)

A long PC ISA slot card serves as the CB. The CB is the core of the multiprocessor system and it is used for *commu-*

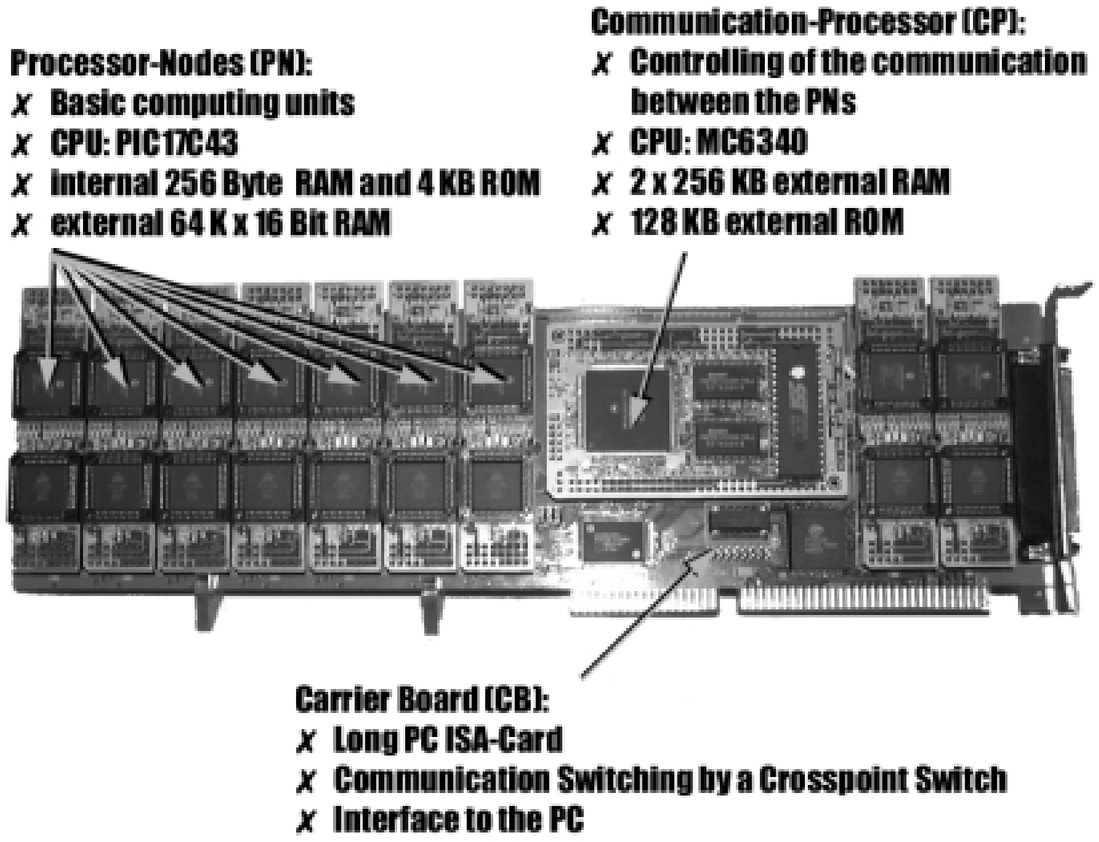

Processor-Nodes (PN):
✗ Basic computing units
✗ CPU: PIC17C43
✗ Internal 256 Byte RAM and 4 KB ROM
✗ external 64 K x 16 Bit RAM
Communication-Processor (CP):
✗ Controlling of the communication between the PNs
✗ CPU: MC6340
✗ 2 x 256 KB external RAM
✗ 128 KB external ROM
Carrier Board (CB):
✗ Long PC ISA-Card
✗ Communication Switching by a Crosspoint Switch
✗ Interface to the PC

Figure 1. Picture of the Multiprocessor System

nication switching. Furthermore, during software development all programs are downloaded using the interface to the PC. Besides the CP up to nine PNs fit onto one board (see Figure 1). Notice that the CB can also be run with less PNs, e.g. in applications where less are sufficient.
A dual port RAM on the CB serves for connecting the local bus of the CB to the PC ISA bus. In our hardware environment we use a local bus also for connecting different CBs. This results from our applications, where the multiprocessor system is also used in real-time applications, but the PC ISA bus cannot guarantee any timing behavior due to the specification.

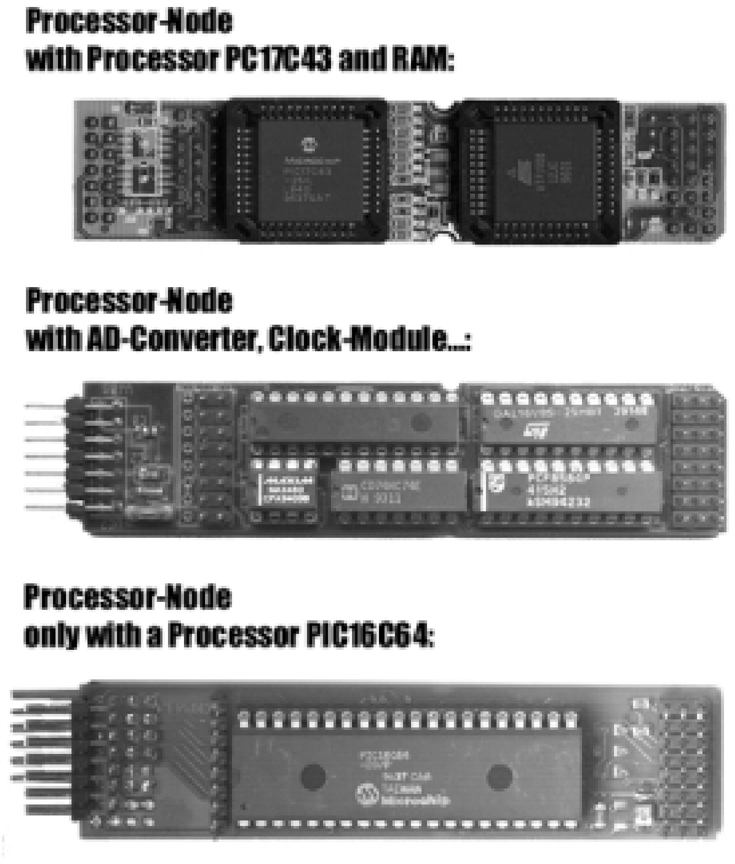
The connection between PNs is established by a crosspoint switch. In our case we use the ICUBE IQ160, a Field Programmable Interconnection Device (FPID) [4, 5]. Off board channel connections are made through 3 IDC cables running between the carrier boards. This cables are placed on the top of the CB.
The internal routing of the CB has been done in such a way that the complete layout can be transferred also to a short PC ISA card. This allows to reuse not only the hardware, but also gives the opportunity to reuse most of the design information.
For all components, i.e. the CP and the PNs, on a CB a communication protocol is specified that allows to exchange the different components (see below).

3.2 Processor Node (PN)

The PNs are the basic computing units. First, we give one more detailed example of a PN and then outline some alternatives.

The PN used in Figure 1 (see also top of Figure 2) has the following main characteristics:

- Microchip PIC 17C43 CPU, i.e. a pipelined RISC type Harvard architecture [7].
- 256 Byte local RAM and 4 KB local EPROM.
- External 64 K x 16 Bit RAM.



Processor-Node
with Processor PC17C43 and RAM:
Processor-Node
with AD-Converter, Clock-Module...:
Processor-Node
only with a Processor PIC16C64:

Figure 2. Different PN Modules

The PIC 17C43 CPU from Microchip was chosen for our purposes, since it is a low cost and for our applications satisfying module (see Section 5). The external RAM is reserved for the application program of the PN, while the EPROM contains basic functionality, like downloading programs and a simple operating system. The PN is equipped with one serial communication channel, capable of transferring data at 5 Mbit/s. Dependent on the application this channel can be used in two different ways:

1. In the field of *Evolutionary Algorithms* for example the serial ports of all PNs are connected to the FPID device to establish the communication between the PNs.
2. For regulation and measurements the serial port of each PN can be used for communication with the 'outside world'. This further underlines the flexibility of the system.

The PNs interface to the CP is implemented very flexibly, i.e. it can easily be replaced by other modules. Two more examples are given in Figure 2:

1. In the middle an analog/digital converter is shown. This allows to also input analog data into the system.
2. A Microchip PIC16C64 CPU, i.e. a smaller CPU than described above without external memory, for applications where this is sufficient.

Notice that PNs of different type can easily be run in parallel on the same CB. Especially the combination of different PNs shows the advantages of the modular design. For completely different applications still the same CB can be used and only some components have to be designed. This allows effective hardware design and fast time-to-market also with highly optimized components.

3.3 Communication Processor (CP)

The CP serves for handling the requests for communication issued by the PNs, for arbitrating with the CP on other CBs, and for controlling the channel switching FPID on its own CB.

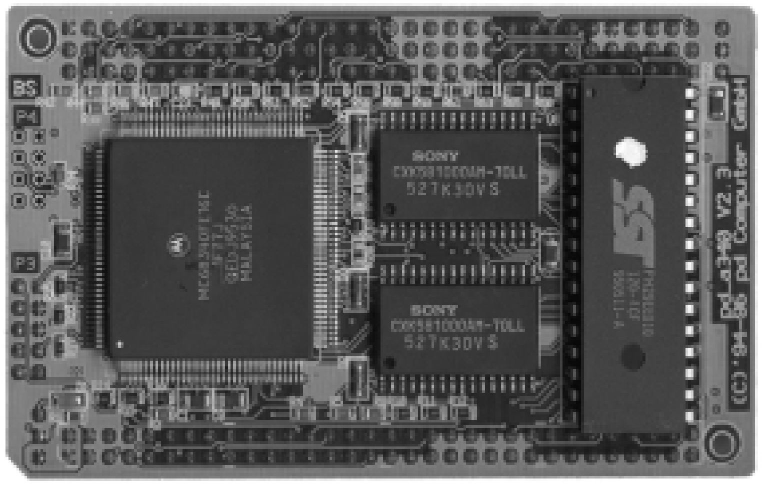


Figure 3. CP with Motorola MC68340 CPU

The CP consists of the following (see Figure 3):

- Motorola MC68340, i.e. a CISC type CPU of the 68-family tuned for data exchange [8].
- 256 KB RAM.
- 128 KB EPROM.

In our software environment the CP also handles the communication with the PC: It receives the application programs for the PNs from the PC, and then the CP is responsible for downloading the programs into the PNs. Since the crosspoint switch on the CB realized by an FPID is used, the programs can also be distributed in parallel (broadcasting). Due to the modular design of our system also the CP can be replaced by other CPU types.

4 Communication Structure

So far we have only focussed on the hardware of the system. In this section the communication between PNs is explained in more detail. By this, the modularity and the close interaction between hardware and software will be demonstrated. An example is given to show how the PNs can communicate *without* the influence of the CP during data exchange.

In Figure 4 the principle of the communication process is shown. The communication between the PNs takes place by the serial communication ports of the PNs. The serial ports of all PNs on the CB are directly connected to the on-board crosspoint switch ICUBE IQ160. The channel switching is left to this device. A crosspoint switch can rapidly switch connections on and off, thus decreasing the switching overhead to almost meaningless figures ($<$ 1ms). The FPID used to realize the crosspoint switch allows for one-to-one, as well as one-to-many connections. With this feature we can realize different communication topologies as star, ring, and cubes, to name only a few.

The user can choose (on-line) a desired communication topology by software. The CP manages the present communication topology by a RAM-list. The CPs as master processors accept the request for communication issued by the PNs and control the crosspoint switch, but they do not participate in the communication between the PNs. Consequently, the active communication tasks do not slow down the channel assignments, and vice versa. The communication channels are completely separated from the CP bus, thus reducing the hardware requirements considerably and preserving the CP bus bandwidth.

To illustrate the communication principle we give a brief example (see Figure 4): If PN 1 wants to exchange data with an other PN, it gives a signal to the local CP. The CP looks up in the RAM-list, where the actual communication topology and the switched channels are tagged. A free channel at the crosspoint switch has to be found and then is assigned. This one has to be flagged in the RAM-list as being in use now. The assignment has to be reported to the participants of the communication, in our example PN 1 and PN 3, and then the CP drives the crosspoint switch to route the chosen channels (dotted lines). The CP uses the FPID like a memory device and (dis)connecting two channels is done by writing a ’1’ (’0’) to the corresponding address.

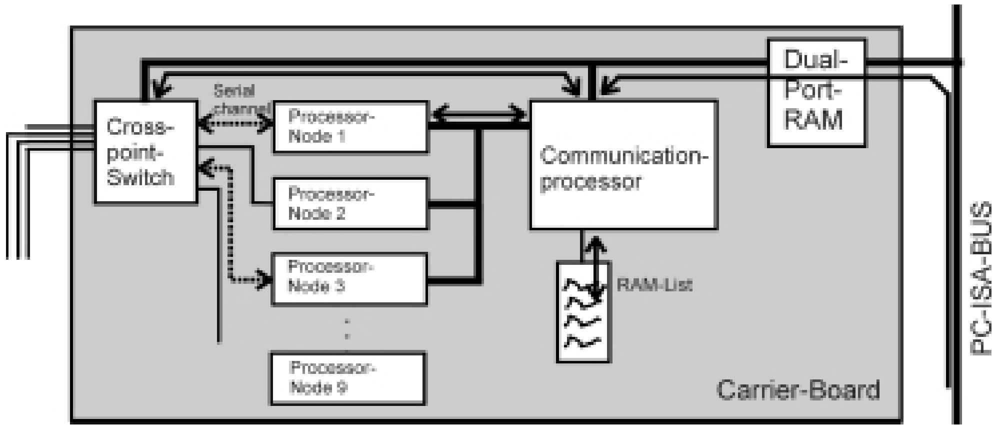


Figure 4. Principle of the Communication Structure

The participants are now allowed to exchange their message(s). Upon completion the channel assignment is released, the assignment flags are cleared, and the release is reported to the participants of the communication. Notice once more that the CP is only responsible for establishing the connection, but not for transferring the data.

5 Modular Design in Real Applications

The HSB bus has in the meantime been used as a backbone structure in other applications, e.g in embedded control. In the following we briefly report on a successful implementation, that has frequently been used in industry. We start with a short definition of the embedded control task:

The HSB definitions have first been used in the embedded control system of a fast batch reader for barcode cards, which was specified to read at a rate of 800-1000 cards/min. The data collected by a mark/sense and a 2-channel barcode read head had concurrently to be gathered, interpreted, approved by validity check, and merged, before being formatted, buffered, and sent to the host processor. A control panel incorporating a distribution of concurrent software tasks onto different processors simplified the software by eliminating the need for a fast real time operating system. Thus, the design became less error prone and featured greatly enhanced testability, both, from a hardware and software point of view.

By modularity the design invited to delegate the implementation of different design tasks to different persons. Hardware and software could concurrently be developed. As the hardware structure was based on the highly modular HSB bus definitions, design prepartitioning, which is considered to be a mission-critical issue, did not show up as an explicit and demanding step in the design flow. Moreover, most of the tests, necessary to verify proper system operation, were restricted to the narrow scope of the modules themselves. A common over-all system test proved to be quite unambitious.

Meanwhile, these barcode batch readers are being used for vote counting on large shareholders assemblies by nine of the fifteen largest shareholder companies in Germany. This is a critical application, demanding a very high standard of reliability and data validity.

6 Conclusion

We presented a modular, low cost multiprocessor system. Each PN consists of a RISC type Harvard architecture CPU. The system can dynamically be (re)configured during runtime, i.e. the communication topology can be adapted to the given application.
Based on the use of crosspoint switches it is possible to switch the communication topology, i.e. to simulate different communication structures like chains, arrays, or cubes. Even more exotic ones are possible, e.g. rings, spirals, and helixes, to name only a few. Thus, the method is well suited for education and research purposes. Additionally, we discussed software and hardware aspects with special emphasis on modularity.
The system has been especially designed for the area of measurement, regulation, controlling and experimenting with different parallel algorithms in real-time environments.

A first embedded control application in a fast batch reader for barcode cards, which utilizes a 4-processor structure, has successfully demonstrated the merits of the highly modular design described.

Acknowledgement

The authors like to thank Peter Biermann (†), Horst Reinhardt, and Kai Reidelbach from *pd Computer GmbH* for their help with the hardware implementation.

References

[1] P. Biermann, R. Drechsler, and B. Becker. Modularity as key element in modern system design - a case study for industrial application of parallel processing. In *European Design & Test Conf. User Forum*, 1997.

[2] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM3 user's guide and reference manual*. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, September 1994.

[3] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, 1984.

[4] I-Cube Inc. Optimizing performance in a multistage network. In *Application Note No. 3*, 1994.

[5] I-Cube Inc. Using FPID devices in FPGA-based prototyping. In *Application Note No. 2*, 1994.

[6] Microchip Technology Inc. *Embedded Control Handbook*, 1994.

[7] Microchip Technology Inc. *Microchip Data Book*, 1994.

[8] Motorola Inc. *MC68340 Integrated Processor with DMA - User's Manual*, 1992.

[9] T. Schubert, E. Mackensen, N. Drechsler, R. Drechsler, B. Becker. Specialized hardware for implementation of Evolutionary Algorithms. *Technical Report*, Albert-Ludwigs-University of Freiburg, 2000.

[10] N. Sitkoff, M. Wazlowski, A. Smith, and H. Silverman. Implementing a genetic algorithm on a parallel custom computing machine. In *International Symposium On FPGAs for Custom Computing Machines*, pages 180-187, 1995.