**SSGRR 2001 - International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet**
L'Aquila, Italy; August 6 - 12, 2001

# Steps Towards Proven Secure Embedded Systems

**Ewald von Puttkamer**
Computer Science Department, University of Kaiserslautern,
D- 67663 Kaiserslautern, Germany
e-mail: puttkam@informatik.uni-kl.de

***ABSTRACT***

*An architecture for embedded systems is introduced using functional units with encapsulated functions on board. These units are strung together via busses in a multiprocessor system goverened by black board units for the control of data exchange. The black board units each control a bus. The semantics of data on a bus are standardized. Data exchange between functions at the same bus occurs via a black board inside the controlling unit. The software in the units is copy and tamper proof and part of the computer.*

## 1 MOTIVATION

Embedded systems creep into more and more technical systems. Although there are only a handful of really different functions to be realized, they are slightly different from implementation to implementation preventing reuse. There is virtually no parametrization. A lack of common interfaces and data structures hinders these functions to be chained together. The many different platforms around make it even more difficult. To prove a system distributed over many computers to be correct is extremely difficult. The complexity of possible interactions is simply too large. On the other hand more and more safety critical functions are implemented in embedded systems on computers with standard operating systems, wide open to viruses. As long as there is no suitable price to be gained as copying software is so simple no one wants to take the necessary effort for standardisation and the effort to produce fault tolerant software. It simply does not pay off. This hampers the development of software for embedded systems considerably and any way out should be tried.

### 1.1 Sketch of a solution

During the last years computing power has become cheap and computers of remarkable power have become available on credit card size boards: 16 bit platforms with 1 MB memory and 1MB PROM and a lot of interfaces ( Microcontroller ) or 32 bit computers with 2 MB RAM and 1 MB PROM [1]. They are equipped with interfaces on board to standard busses like CAN or SCSI [2] .

The idea is to implement the desired functions each on a computer, accessible only via a standard bus. Standardization of input and output includes here the standardization of the semantics of the data on the bus too, while the syntax follows a standard protocol for data transfer on the bus. The program itself realized inside the computer is inaccessible from the

bus.

There are steps into this direction already taken: motor controllers at a field bus [3] realize black box functions for the customer, but at the bus itself there is no higher level standardization.

## 2 FUNCTIONAL UNITS

The rough sketch is to be outlined now in further detail. A functional unit realizes a complex data processing task for an embedded system. Similar to devices attached to a GPIB bus [4] functional units recognize a repertoir of standard commands only, automatically executed once specified as a sub address of the functional unit. The only access to a functional unit is through these commands. Likewise to devices at a GPIB bus functional units are passive towards the bus: they have to be addressed from an external control and react to commands given to them.

In order to keep things as simple as possible there should be only a small number of these commands. Eight commands will do. They define the basic commands of an operating system on board of each computer.

### 2.1 Standard commands

- identify:

  outputs the name of the functional unit, number, names and parameter lists of functions implemented within the unit, maximal execution times ( allowing the automatic buildup of a schedule), necessary buffer sizes for data transfer, expected result of a self test, and a textual description of the functions

- execute a self test:

  the functional unit executes a self test with the result placed into a status register later to be read off from there.

- switch to a function

  the following byte transferred to the functional unit switches to the function named by that number. Any commands to a function now address this one. The name stays valid until the next “switch to a function” command

- transfer of input parameters

  the functional unit awaits to read in the input values of the parameters specified in the parameter list of the identification of the function switched to. The parameters could be complex data types like arrays or records

- start of function

  starts execution of the function at hand with the parameter values given

- send status

  the functional unit outputs the contents of its status word on the bus. Interrupts handled by a functional unit reflect themselves here as bits denoting an event. Other bits describe the status of an activated function or the validity of a self test.

• send results

the functional unit outputs the results of the function just executed to the bus including an error message and the number of bytes as specified in the identificat ion. This realizes a “call by value” mechanism.

• reset

the functional unit is reset into a definite state

The standard commands are made public and are the basis of data transfer between functional units. The results of one function may be input parameter values of annother one.

Figure 1 shows the data transferred to a functional unit and its response.

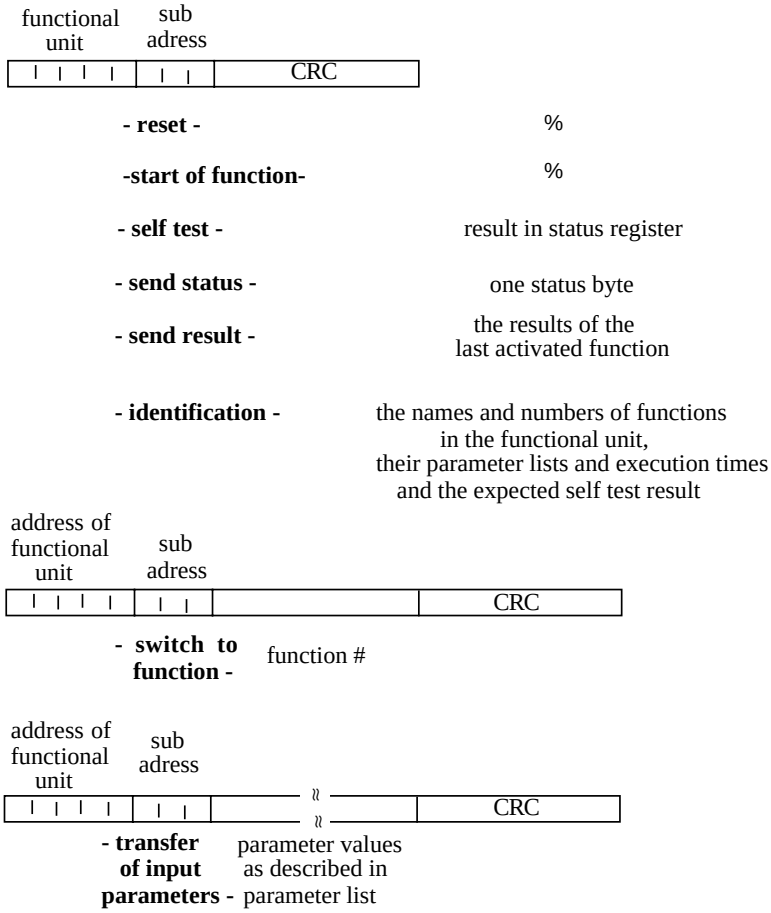


Fig.1. Commands understood by a functional unit

Aside from these commands there must be a synchronizing mechanism on the bus to let a functional unit distinguish between begin and end of a message from the controlling unit, to recognize an idle bus and a way to address the controlling unit once asked to send data. Furthermore there must be a way to detect errors on transmission and a possibility to ask for a retransmission. These are standard protocol functionalities on a bus and not further discussed here. They are specific for the bus system at hand. The general software layout of a functional unit is shown in fig.2. Square rectangles denote data processing parts, round objects denote data spaces ( variable declarations ). Hatched rectangles denote operating system functions in a functional unit.

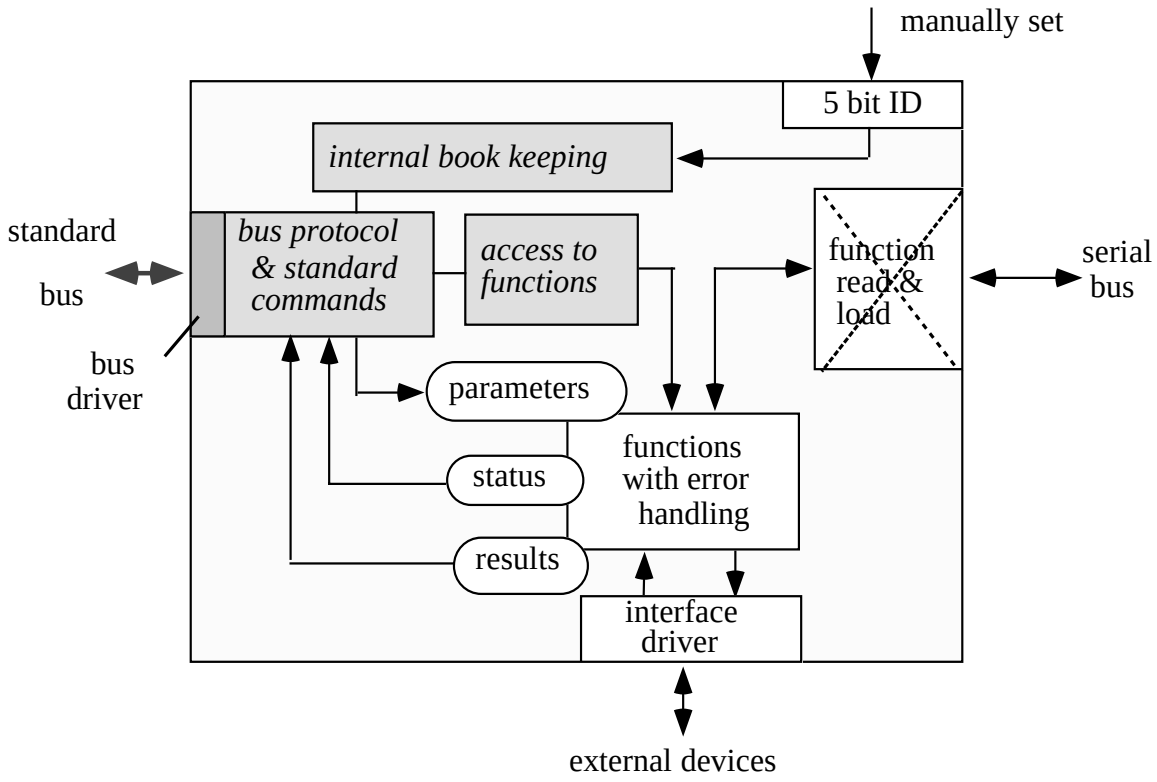


Fig.2 Functional unit - software layout

## 2.2 Hardware

Fig.3 shows a sketch of the hardware of a functional unit. It is a complete computer at a standard bus, possibly equipped with peripheral interfaces. Using i.e. an address length of 8 bits with 3 bits set aside to characterize the standard command within the functional unit 32 different units may be addressed. Set aside one address ( 0 ) for an idle bus and one address (31) for a reply to the control unit there remain 30 addresses for functional units at a bus. This should be sufficient for embedded systems at one bus. The addresses of the functional units in the embedded system at hand are set by address switches on board the functional units. At initialization an error is generated if two addresses are the same. Specified by one byte after a “switch-to-function” command a functional unit could have up to 256 different functions implemented. At initialization the correlation between addresses and functions for all units at a bus is established in the controlling (black board) unit of the bus. Functional units are passive towards their driving bus: they merely react to commands given via the bus.

As only the standard commands are understood, no alterations via the bus to the programs inside a functional unit are possible unless explicitely allowed by a program already in the program store of a functional unit. This fulfils one requirement for safe software: there is no pathway for viruses to creep into a functional unit.

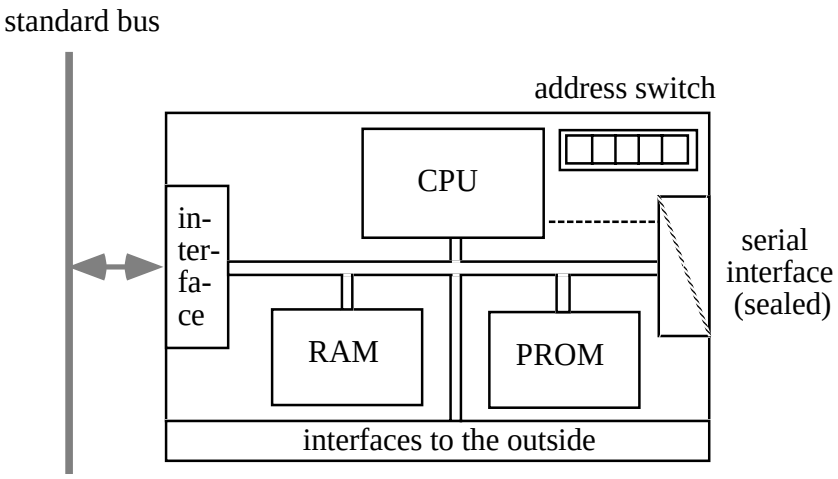


Fig.3 Functional unit - hardware

Access to the programs on board is either prohibited by an epoxy cover or only possible via a different input ( serial or SCSI ), normally covered by a seal. Breaking the seal ends any liabilities of the producer of the functional unit. There are common functions to access data and programs via this port. They form part of the operating system, too.

For testing or for special applications a user may write his own software into a blank functional unit. If his software finds customers he may seal the unit and sell it as a functional unit himself. The computers used may be 16 or 32 bit devices. In order to transfer larger amounts of data a fast bus may be used ( SCSI, USB or equivalent ), otherwise a CAN bus will do. Functional units realize abstract data types: the data stucture itself remains invisible inside the computer with access functions to write, read and modify the data from the outside. This could be a solution i.e. for the administration of maps in driver assistant systems in vehicles.

For the different types of data processing in an embedded system three different types of functional units are envisaged:

### 2.2 Pure functional units

They realise pure data processing functions and do not have interfaces to the outside. They are equipped with rather powerful machines and large memories for massive data processing tasks like picture processing or calculating integral transforms. They are connected to a fast bus.

### 2.3 I/O function units

These are functional units with peripheral interfaces on board for input from sensors and output to actuators, realized by microcontrollers. A closed loop control may be realized in their program. Once triggered, an I/O function unit may either repeat a control loop permanently or just execute the loop once. These are two different functions. In the latter case read out of sensor data and the synchronization of the control loop stays in the responsibility of the next higher stage. A third function could be to execute the control loop permanently and to output the latest sensor data as answer to a “send results” command.

For data logging a unit triggered from the bus can deliver preprocessed sensor values. Interrupts from sensors are dealt with inside the I/O units and made available to the outside as events signalled via the status register and with a service request signal line pulled down by the I/O function unit, provided such a mechanism is available at the bus. The unit in charge of this functional unit reads the status of all units attached to the bus to find out which event

occured.

As the hardware platform is typically a microcontroller and the data output limited, the bus could be a CAN bus or another slow serial bus. The necessary overhead in form of driving programs for the interfaces and the interpretation of standard functions is no problem any longer with megabyte RAM´s on board.

### 2.4 Black board unit

This special functional unit realizes the control and data flow on the level of its attached functional units. It is a functional unit like any other but connnected to two busses, a driving bus and a driven one. Seen from the driving bus it is a functional unit.
The second bus is controlled by the black board unit itself. There are black board units driving slow or fast busses. Fig 4 shows the general program structure of a black board unit.

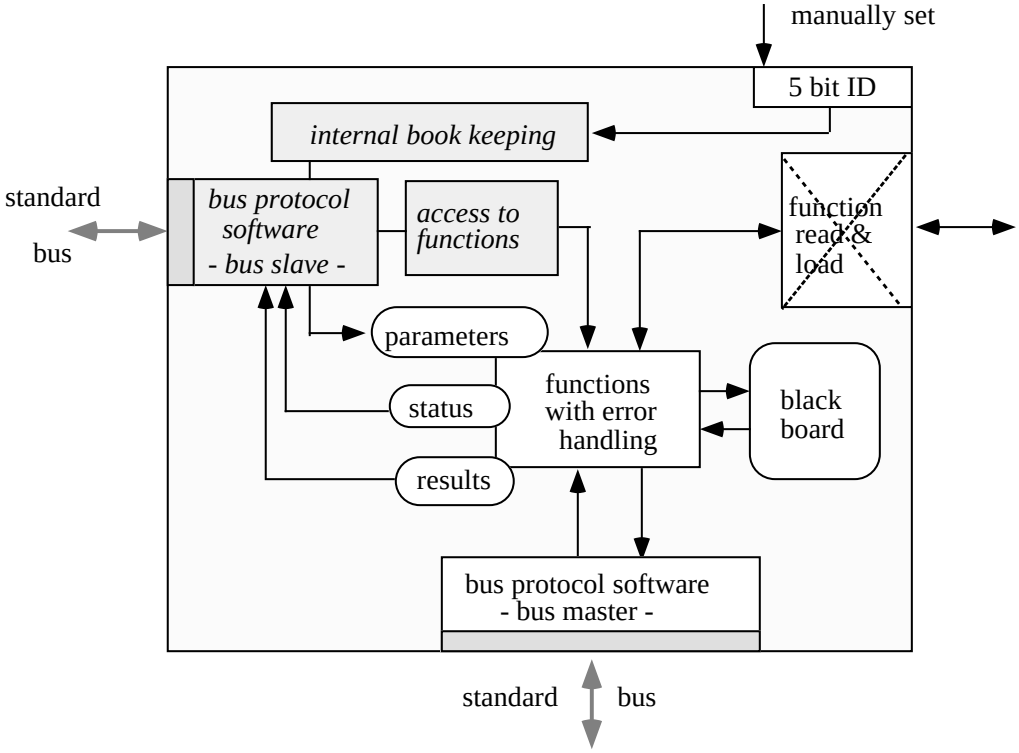


Fig.4 Black board unit - software layout

## 3 BLACK BOARD UNIT FUNCTIONS

The functions realized in a black board unit activate functional units at this bus, outputting standard commands, reading the results into a black board in the memory of the black board unit and transferring input parameter values from the black board to functional units at the bus. This realizes the data flow between functional units. The black board represents the momentary values of all relevant data in the system represented by the functional units at the bus controlled by the black board unit. An invisible buffer scheme and semaphores as standard software techniques guarantee data consistency in the black board. Consumers always see complete consistent data sets. As all data from and to the attached units at a bus run through the black board this is a point to set aside data for test purposes and/or later inspec-

tion with minor overhead.

The control flow is realized by function programs inside the black board unit. These functions govern the sequential activation of functions in the functional units at the bus and the data transfer via the black board. Conditional jumps are realized checking values from the black board or internal values in registers of this computer. The black board unit operates as the master for the bus driven by it. All data transfer is initiated from here. Up to 7680 different functions ( 30 x 256 ) may be evoked from one black board unit. The program inside a black board unit is as inaccessible from the outside like that in any other functional unit. There are special functions realized inside a black board unit like bus initialization and building a schedule. They form the kernel of an operating system for a multiprocessor system built up from a black board unit and its attached functional units at the bus driven by the black board unit. These functions are made public. They are part of the possible 256 different functions to be evoked at a black board functional unit.

The special functions in a black board unit are

### 3.1 Initialize bus

By starting this function in a black board unit a list of all available units at the controlled bus is set up. A check is performed, wether the function activated is itself a control program calling functions implemented in functional units at the controlled bus and wether all addresses are unequivocal. From the identification of the functions to be activated inside the units the necessary buffer regions in the black board are set aside for data transfer together with their pointers. Despite all activities to standardize formats once in a while the data delivered by one function have a different format from the data needed as input by another function. Then functions to translate between different data formats are evoked.

### 3.2 Scheduling

Cyclical activations are organized around a schedule calculated from execution times as given from the self identification of the attached functional units, transferred to the black board unit as result of an identify command. This guarantees hard real time behaviour of the program as a whole. The normal case in an embedded system will be a couple of cyclically activated functions with an overlay of asynchronous activations from the outside, slotted into suitable time slots of the general schedule. During these time slots possible events are polled reading the status registers from the attached I/O units and transferring data from and to these units.

### 3.3 Other operating system functions

There is no explicit operating system, only some kernel OS functions made public:

- an access procedure to functional units via standard commands
- bus initialization with automatic black board organisation in black board units
- schedule building in black board units from timing parameters
- a function to read and write via the extra serial link in each functional unit.

  This link is sealed to be used only in case of program failures to dump memory contents or to bring in new software from the side of the producer.
- a memory management function in each functional unit.

It manages the distribution of memory space to the different programs. This function is part of the cross compiler, too.

## 4 MULTIPROCESSOR SYSTEM

A black board unit and its attached bus form a multiprocessor system. The different functional units in this system each operate independently at their own pace in parallel to each other. The synchronization is done by the black board unit as bus master. Seen from the outside the black board unit is a functional unit, in most cases with input/output from/to the sensor/actuator hardware of the system. This allows the build up of a hierarchical system as shown in fig.5.

To the outside only the necessary data are made available, hiding all internal data processing. For the purpose of testing a switch over into a test function could be possible, delivering a lot more data from the black boards to the outside as result of an activation, by paying the price of slightly slower execution.

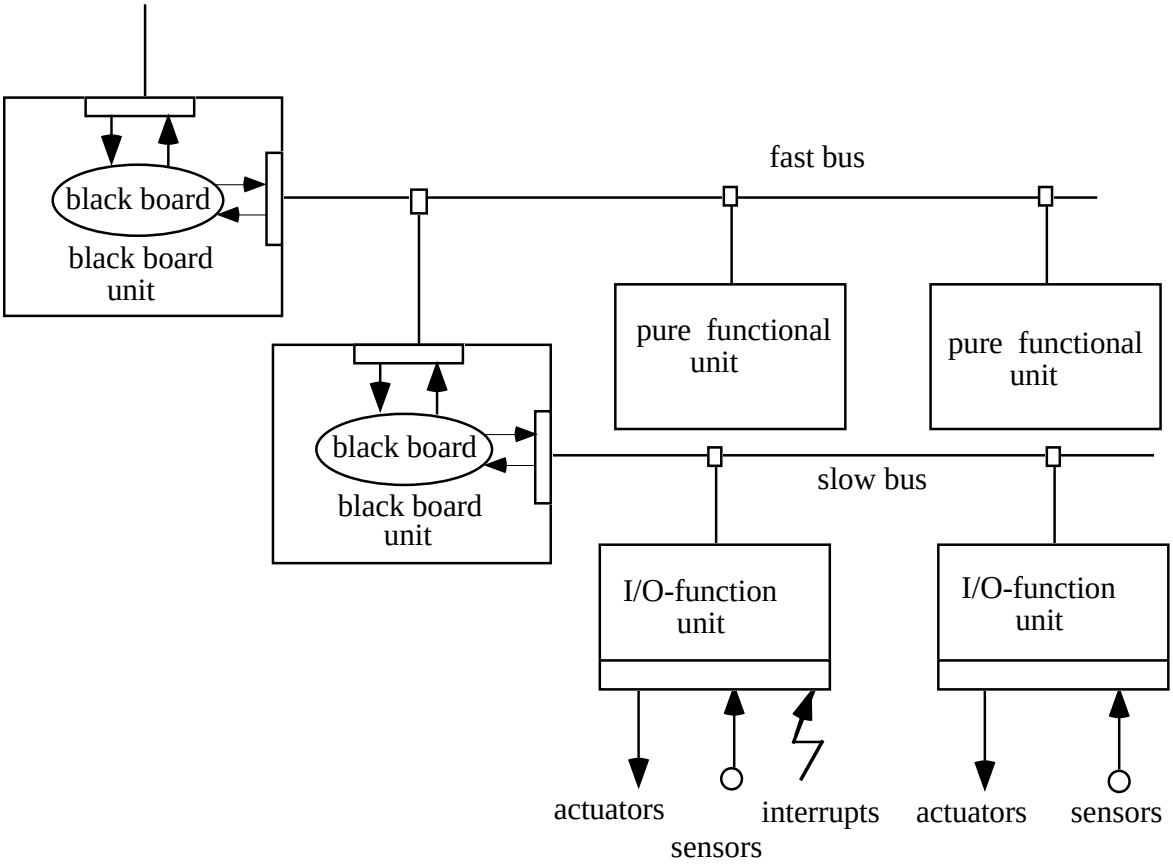


Fig.5 Hierarchy of functional units

### 4.1 Programming the units

Software development will take place on ordinary PC´s with cross compilers. The normal case will be an application programmer, writing his code for a black board unit and tranferring the code to the program store of that device via the extra serial link and sealing it with his seal. This states his responsibility towards the customer of the system.

Functions for functional units will be written the same way: the binary code transmitted by

the extra serial link to the functional unit and then this link sealed off. As an alternative the data are burnt into a PROM and inserted into place at the functional unit and the board coated with epoxy.
For sake of security there must be no functions implemented in a functional unit allowing to read in programs via the normal bus. This would allow in principle to bring in uncontrolled software, including viruses, and hurt the main advantage of this architecture that its software is secure from tampering.

### 4.2 Discussion

There are a lot of arguments in favour or against an approach as sketched:

#### 4.1.1 Pros

- by covering the board in epoxy the interior is inaccessible
- the software on board is copy safe if there is no read-program function implemented
- a function is offered to the customer: a computer with its program as a unity
- a guarantee can be given for this function
- a suitable price may be taken from a customer
- safety against failures may be high
- the system may be fast as functions on different computers are evaluated in parallel
- interrupts are handeled on board of the relevant functional unit and made public to the outside as events. Transforming an interrupt into an event takes a fixed amount of time.
- hard real time constraints can be met as a correct schedule is built up with help from the scheduling function in the black board unit.

#### 4.2.2 Cons

- the system software is realized on a multi processor system
- connecting the computers and coordinating them is an overhead effort
- the bus system used could become the system bottleneck, as all data exchanged between different functions have to be transported via the bus into and out of a blackboard even if they are produced and consumed at the same computer but in different functions.

## 5 SUMMARY

By using cheap hardware it becomes possible to encapsulate software functions in separate computers, prerequisite for giving guarantees to software and for copy proof systems. The functions are coupled via a common bus to a separate functional unit with a black board inside ( black board unit ) for data exchange. From this unit there is a connection to next higher stages via another bus. The program at this black board unit is as inaccessible from the outside like that on board any other functional unit. The different amounts of data to be transported will manifest themselves in different bus systems to be used.

This opens the path towards a whole series of functional units as building blocks for the

control structure of embedded systems, accelerating the development time markedly. In former times the power of FORTRAN was not the language itself but very large libraries of sophisticated functions for all branches of mathematics, physics, chemistry and engineering that simplified the task of writing application programs. Functional units might play the same role for embedded systems.

Selling hardware components - a computer with inaccessible software on board - the legal position of producers is much better. They may get fair prices for their products, thus allowing the large expenses for good software to be spent for the development of functional units. The customer gets complex function building blocks with a warranty from the producer in return. The common commands are the interface between them. At this interface the functionality may be proven. The producer of functional units gains from using a simple common standardized bus as described, throwing his efforts into producing sophisticated functions inside the functional units, simple to use externally as building blocks because of the standardized semantics on the bus.

This could be a possible solution for the software crisis, too.

## REFERENCES


[1] Phytec Meßtechnik GmbH, Robert-Kochstr.39, D-55129 Mainz, Germany
http://www.phytec.de

[2] D. del Corro, K. Kirrman, J.D. Nichould
Microcomputer Busses and Links
Academic Press, 1986

[3] G.Schnell
Bussysteme in der Automatisierungstechnik
2.Aufl. Vieweg, Braunschweig, 1996

[4] A.C. Caristi
IEEE-488 General Purpose Instruction Bus Manual
Academic Press, 1989