PEARL-News

Ausgabe 2 November 1999

Mitteilungen der GI-Fachgruppe 4.4.2 Echtzeitprogrammierung PEARL

ISSN 1437-5966

Impressum

Herausgeber	GI-Fachgruppe 4.4.2 Echtzeitprogrammierung PEARL URL: http://www.real-time.de
Sprecherin	Dr. Birgit Scherff ATR Industrie-Elektronik GmbH & Co. KG Textilstraße 2 D-41751 Viersen Telefon: 02162/485-363 Telefax: 02162/485-863 E-Mail: u.terporten@atrie.de
Stellvertreter	Dr. P. Holleczek Universität Erlangen-Nürnberg Regionales Rechenzentrum Martensstraße 1 D-91058 Erlangen Telefon: 09131/85-7817 Telefax: 09131/30 29 41 E-Mail: holleczek@rrze.uni-erlangen.de
Redaktion	Prof. Dr. Dr. W. A. Halang FernUniversität Fachbereich Elektrotechnik D-58084 Hagen Telefon: 02331/987-372 Telefax: 02331/987-375 E-Mail: wolfgang.halang@fernuni-hagen.de
ISSN	1437-5966

Redaktionell abgeschlossen am 2. November 1999

Einreichung von Beiträgen: Beiträge zu dieser Zeitschrift sind jederzeit hochwillkommen. Je früher und "rechtzeitiger" sie eintreffen und je humorvoller sie sind, umso glücklicher ist der Redakteur. Einreichungen "auf den letzten Drücker" und das Textverarbeitungsprogramm Word verleiten ihn hingegen zu Wutausbrüchen. Darum bittet er höflich um Übersendung der Beiträge per E-Mail in reinem ASCII und nicht codiert — eben ohne jeden Firlefanz — oder noch besser in LaTeX.

Inhalt

1 Editorial

Diese Ausgabe macht ganz deutlich, daß PEARL keine rein deutsche Angelegenheit mehr ist. So ist es dem Arbeitskreis 5 gelungen, PEARL dem Normungsgremium SC22 des gemeinsamen Technischen Komitees JTC1 der beiden internationalen Normierungsorganisationen ISO und IEC auf seiner Berliner Plenartagung im September durch drei Präsentationen vorzustellen und dabei auf große Resonanz zu stoßen.

Dem internationalen Interesse an PEARL ist auch der erste Schwerpunkt dieser Ausgabe gewidmet. An der Technischen Fakultät der Universität Maribor in Slowenien wird PEARL seit drei Jahren im Rahmen zweier Kurse — einer vor dem Bachelor-Grad und ein postgradualer — über Echtzeitsysteme verwendet. Dabei kommt das kostenfrei verfügbare PEARL für Linux von Werum[1] zum Einsatz, das auf einem Labor-Server installiert ist und von den Studierenden via Telnet benutzt wird. Diese arbeiten in kleinen Gruppen von bis zu 10 Personen an Projekten, die sie entweder aus einem Katalog auswählen oder selbst vorschlagen. Thematisch geht es dabei i.w. um die Programmierung paralleler Prozesse und deren Zuteilung. Zur Lösung der Aufgaben müssen die Studierenden PEARL erlernen und ihre Projekte darin implementieren.

Ebenfalls an der Universität Maribor schlossen am 18. Oktober zwei Herren ihre Promotionen ab, die von Herrn Prof. Colnarič und mir betreut wurden. Beide Dissertationen beschäftigen sich in großen Teilen mit PEARL. Die PEARL-bezogenen Ergebnisse sind in den folgenden beiden Abschnitten dieser Ausgabe zusammengefaßt. Domen Verber macht in seiner Arbeit Vorschläge zur Erweiterung von PEARL im Hinblick auf Objektorientierung. Er führt Klassen und Objekte ein und stellt traditionelle Echtzeitsprachelemente objektorientiert dar. Für seine Sprache ObjectPEARL hat er einen lauffähigen Übersetzer entwickelt. Roman Gumzej hat einen Ansatz zur simultanen Entwicklung der Hard- und Software von Echtzeitsystemen entwickelt. Darin nimmt eine Spezifikations- und Konfigurationsbeschreibungssprache auf der Grundlage von Mehrrechner-PEARL eine zentrale Stellung ein, von der auch eine graphische Version definiert wurde.

Im zweiten Schwerpunkt dieser Ausgabe stellen dann zwei Mitarbeiter des Instituts für Regelungstechnik der Universität Hannover mit dem Mikrocontroller MPC555 aus der PowerPC-Familie und dem Multitasking-Echtzeitbetriebssystem RTOS-UH eine Plattform vor, die die Realisierung anspruchsvoller mechatronischer Anwendungen auf flexible Weise erlaubt.

Wolfgang A. Halang

2 ObjectPEARL — object oriented extensions to PEARL

The object oriented approach is the most widely accepted paradigm today. However, the process of integrating this methodology into real-time systems is relatively slow. Therefore, we extent the object oriented paradigm to be successfully used in safety-critical hard real-time systems. To substantiate theoretical findings, an object oriented programming language is designed. It is based on PEARL and called ObjectPEARL. Not only classes are introduced in the syntax of the language, but also traditional elements of real-time systems are represented in it by means of object oriented notation. Parallel to the programming language design, a compiler was implemented, which contains an integrated execution time analyser due to the need for schedulability analysability. The benefits of object oriented description can be utilised when there is already a large set of pre-defined data types (classes), and when components are divided into specialised groups (e.g., periodic and aperiodic tasks). On the other hand, the object oriented approach usually requires a slightly different point of view. This can be a problem if it is used by persons who know traditional (i.e., structured) developments techniques, only.

2.1 Introduction of a Class as extension to standard structures

In introducing classes into the language a similar approach as in C++ was used. In both cases a class is defined as an extension to a structure. For simplicity of code generation and execution time analysis, it

[1] Die Universität Maribor möchte sich dafür bei der Firma Werum an dieser Stelle sehr herzlich bedanken.

was decided not to use multiple inheritance of classes. Therefore, a class can be a descendent of only one parent class. The basic syntax for a class in ObjectPEARL reads:

```
TypeClass ::=
  CLASS [ClassHeritage] '['
    ClassComponent {,ClassComponent}
  ']'

ClassHeritage ::=
  '(' Identif#TypeClass ')'

ClassComponent ::=
  [ClassVisibility]
    ( ClassField  | Property    |
      ClassMethod | Constructor |
      Destructor ) ;
```

A class is defined as a sequence of class components. In standard PEARL only simple variables are allowed in a structure. In ObjectPEARL there are also methods and virtual variables (properties). Class inheritance is defined with the `ClassHeritage` clause. In this case all components of the predecessor class are also implicitly included.

Similar to C++ and some other object oriented programming languages, it is possible to define visibility for each class component:

```
ClassVisibility ::=
  PRIVATE | PROTECTED | PUBLIC
```

Private components can be accessed only inside methods of the current class. Protected components can be used inside the current class and in methods of all derived classes. Public components can also be accessed from outside of the class methods.

Class variables are declared in the same way as variables in the structures of standard PEARL. Special kinds of variables are so-called virtual variables or properties. Properties behave as ordinary variables with two additional methods. The first method is used for reading the variable, and the second one is used when a property value should be changed. Properties operate in a similar manner as those used in the COM Automation model:

```
Property :=
  PROPERTY Identif#NameOfProp DataType
       [READ Identif#GetMethod]
       [WRITE Identif#SetMethod]
       [SPECPROP]
```

A method declared with a `READ` attribute must be a function without parameters returning a result of the same data type as the property. A method declared with a `WRITE` must be a procedure with only one parameter of the same type as the property. Both methods must be declared within the same class or within one of its predecessors. Instead of the method, an ordinary variable defined in the class can be used. In this case, when accessing the property this variable is actually used. By leaving out the `READ` or `WRITE` attribute the property becomes write-only or read-only. With the `SPECPROP` attribute a property is marked as specification-only. This means that this property is declared for specification purposes, only, and no code is associated with it.

Class methods have a similar description as procedure specifications in standard PEARL:

```
ClassMethod ::=
  IdListPack [:] PROCEDURE
  [ListOfFormalParameters]
  [ResultAttribute][VirtualAttribute]
```

```
VirtualAttribute ::=
  ABSTRACT | VIRTUAL | OVERRIDE
```

A method can be virtual to implement polymorphism in the object oriented paradigm. First, the method is declared with the attribute `VIRTUAL`. Later, in the ascendant class it can be re-defined with the `OVERRIDE` attribute. With the `ABSTRACT` attribute abstract virtual methods are defined. Those methods have no implementation within the current class; they must be overridden in the derived class. Within the method the implicitly defined variable `SELF` can be used as a reference to the current object.

A special kind of methods are constructor and destructor. The constructor is called when a specific object of a class is declared. This is usually the case upon entry into a procedure. Objects declared as global variables are initialised before a program is started. If virtual methods are used within the class, the constructor is also responsible for initialisation of the polymorphism mechanism. There can be more than one constructor. The programmer can use one of them with the `INIT` clause within object declaration. There is also a default constructor that is called when no specific one is defined. There can be only one destructor. It is used to de-initialise an object when it leaves its scope. Here is an example of class definition:

```
TYPE
  BaseClass CLASS [
  PRIVATE
    Var1 FIXED;
    GetP : PROCEDURE RETURNS (FIXED);
    SetP : PROCEDURE (Value FIXED);
  PROTECTED
    Var2 FIXED;
  PUBLIC
    Const : CONSTRUCTOR(InitValue FIXED);
    Destr : DESTRUCTOR;
    Foo : PROCEDURE((x,y) FIXED) VIRTUAL;
    PROPERTY Prop FIXED READ GetP WRITE SetP;
  ];

  InheritedClass CLASS(BaseClass) [
  PRIVATE
    Var3 FIXED;
  PUBLIC
    Foo : PROCEDURE((x,y) FIXED) OVERRIDE;
  ];
```

In the first class two variables, constructor, destructor and one virtual method are declared. Also, a property with associated read and write methods is specified. The other class implicitly inherits all components of the base class and re-defines the virtual method. Variable `Var1` can be accessed only within methods of the base class, variable `Var2` can be used within methods of both classes, and variable `Var3` can be accessed inside the inherited class, only.

An object can be declared as follows:

```
DCL
  O1 BaseClass INIT(Const(3));
  O2 InheritedClass;
```

Object `O1` is declared as variable of the base class with explicit constructor, object `O2` is declared as a variable of the inherited class. Since there is no `INIT` attribute, the default contructor is used (i.e., all variables are set to zero or `NIL`). Here are examples of the usage of objects in a program:

```
O1.Foo(1,2);
O2.Foo(1,2);
O1.Prop := 4;
X := O1.Prop;
```

The second call of `Foo` is performed with the method from the inherited class. When the property appears in the left side of an assignment, the `SetP` method is used, otherwise the `GetP` method is called.

2.2 Extension of the object model to other elements of the language

The benefits of using the object oriented approach can be much greater if traditional elements of real-time systems are also represented as objects. Therefore, a way to represent tasks, semaphores, bolts, signals etc. was studied.

Tasks in PEARL may be considered as a special kind of procedures, which can be executed in parallel under control of an operating system. For this a set of well defined tasking operations is used. On the other hand, objects are traditionally considered as extensions to static data types. Objects do not have direct control over their execution, and there is usually no provision for tasking. A unified views can be achieved if a task is considered as an object with a method that represents the main task behaviour and with several methods for tasking.

```
TYPE
  TASK CLASS [
  PRIVATE
    FContext TASKCONTEXT;
    SaveContex PROCEDURE
       (ProcContext REF TASKCONTEXT);
    RestoreContext PROCEDURE
       (ProcContext REF TASKCONTEXT);
    SetState PROCEDURE
       (Value TASKSTATE);
    GetState PROCEDURE
       RETURNS(TASKSTATE);
    ...
  PROTECTED
    MainProc PROCEDURE
       VIRTUAL ABSTRACT MAIN;
  PUBLIC
    Activate PROCEDURE
       (Schedule SCHEDULE,
        Deadline DURATION) VIRTUAL;
    Terminate PROC VIRTUAL;
    Suspend PROC VIRTUAL;
    ...
    PROPERTY State TASKSTATE
       READ GetState WRITE SetState;
    PROPERTY CurrSchedule SCHEDULE
       READ GetCurrSchedule;
    PROPERTY Deadline DURATION
       READ GetDeadline;
    ...
  ];
```

Data types like `TASKCONTEXT` and `TASTKSTATE` are system-defined classes representing the context of a task, its current state etc. The variable `FContext` and the methods `SaveContex` and `RestoreContex` are declared as private components of the class and are only used by the operating system. A set of `Get` and `Set` methods is declared in a similar way and is used to access an internal state of the task and to perform low-level tasking operations. MainProc is the main task execution method. Tasking operations are declared as public to allow access from outside of the object.

Similar to this, other elements of real-time systems can be represented as objects. For instance, a semaphore can be declared as:

```
TYPE
  SEMA CLASS [
  PRIVATE
    FCount FIXED;
    FSemaState SEMASTATE;
  PUBLIC
    Preset CONSTRUCTOR(InitCount FIXED);
    Request PROC(TimeOut DURATION);
    Release PROC;
    Try PROC RETURNS(BIT(1));
  ];
```

As in tasks, the private variables `FCount` and `FSemaState` are used for internal implementation of a semaphore. Again, private variables and implementation of methods are under direct control of an operating system. Due to the need for temporal predictability of the programs written in ObjectPEARL, there is an additional parameter associated with the `Request` method that serves as deadline on waiting for the semaphore to be released.

2.3 Compiler for ObjectPEARL

To support our basic research, a compiler for ObjectPEARL was implemented. With commercially available compilers and tools this was not possible. Also, none of them has provision for execution time analysis. Furthermore, a compiler for our experimental platform that was used for different projects of our research team was also needed. Using experiences gathered in the course of developing a compiler for MiniPEARL, we decided to develop a compiler for ObjectPERARL from scratch. The primary goals in building the compiler were:

Modularity and flexibility Since the compiler was used as a practical tool to support our research, it had to be modular and flexible to be easily and quickly adaptable to all changes. The compiler is built of separate components. When new syntactic elements are added to the language, new modules are built and dynamically linked to the compiler. Similar to this, modules for code generation and execution time analysis were designed.

Target system independence Hardware platforms for hard real-time systems are often specially designed and can comprise different sets of microprocessors. Thus, a compiler usable for only one type of platform would be too limited. The differences between processor architectures are too big to use a universal approach. To cope with this problem we decided to divide code generation into a system-independent part and supporting routines tailored for specific processora. Here we utilise the object oriented approach. The core of the code generator is designed as a set of well-defined classes, which can be re-defined through inheritance to satisfy specific needs.

Generation of efficient code Although the generation of efficient code is of secondary importance for hard real-time systems in comparison to time correctness, smaller and faster code can increase performance reserves in a system to overcome transitional overloads. To achieve this, several optimisation techniques were included into the compiler.

Realistic estimation of task execution times Too pessimistic estimations of task execution times could let appear a system infeasible and require unnecessary effort to improve it. On the other hand, too optimistic estimations cause deadline misses when applications are executed on real systems. Similar to code generation, execution time analysis is tightly dependent on the used microprocessor. Therefore, execution time analysis of parts of code should be integrated into a low-level code generator. Thus, each object for code generation has an associated method for execution time estimation of generated code. High-level analysis (i.e., on the level of loops, decisions, procedure calls etc.) can be performed separately.

The compiler for ObjectPEARL was implemented with the Delphi development environment and runs on Windows 9x or Windows NT platforms. In its current version it generates code for Motorola's ColdFire microcontroller. To be execution time analysable the target object code is directly generated by the compiler. Translation of ObjectPEARL code into standard PEARL was also studied, but there were

several semantic differences between both languages (e.g., visibility of class members) that cannot be easily handled by using simple translation techniques.

Dr. Domen Verber
University of Maribor
Technical Faculty
ul. Smetanova 17
SLO-2000 Maribor
domen.verber@uni-mb.si

3 Embedded Systems Architecture Co-Design

Since most automation and real-time processing applications require the programming of distributed, fault-tolerant multiprocessor systems, PEARL has been extended by constructs for the programming of multiprocessors. In PEARL for distributed systems (DIN 66253 Teil 3), the language is enhanced with constructs which allow for the abstract description of hardware and software. These enable real-time embedded systems to be co-designed in order to increase their dependability and quality. They are not translated into machine code. Instead, they are used as directives for system programs (e.g., configuration management programs, loaders etc.).

Based on PEARL for distributed systems a specification language is proposed featuring:

- constructs to describe hardware configurations,
- constructs to describe software configurations,
- constructs to specify communication and its characteristics (peripheral and process connections, physical and logical connections, transmission protocols), as well as
- constructs to specify both conditions and methods for dynamic reconfigurations in cases of failure.

In the course of a research project carried out at the University of Maribor[2] a platform for embedded real-time system was developed. Its architecture has dedicated processors which execute the operating system. As programming language we selected a subset of PEARL ensuring maximum predictability of execution times. The syntax of PEARL for distributed systems was extended to allow for the description of the operating system kernel processor's attributes. To enable this, the processor description was extended. To specify properties of peripheral devices, a new division was added which is dedicated to their description. Properties were added to ease later execution time analysis.

3.1 Specification PEARL

The proposed extension "Specification PEARL", composed of selected language constructs of PEARL for distributed Systems and of natural-language comments, has the following characteristics, usually required for specification languages:

- abstraction, i.e., insignificant details are suppressed, the conceptual world of the application domain is supported, and no implementation is referred to; application concepts and structures, relations and sequences are easily recognisable,
- easy readability, but nevertheless precise notation,
- provision for inambiguous and complete descriptions of requirements and design,
- support for effective communication between clients, designers and users about the systems to be developed,
- possibility of easily extending specifications into executable prototypes, and

[2]For details cp. M. Colnarič, D. Verber, R. Gumzej and W.A. Halang: Implementation of Real-Time Embedded Control Systems, *Real-Time Systems* 14, 3, 293 – 310, 1998

- systematic integration of the specification method into the entire development process.

A textual system architecture description consists of divisions, which describe different associated layers of a system design in considerable detail: station division, configuration division, net division, system division and peripheral division. They have an equivalent representation in a graphical form, which will be outlined later.

Station division In the station division the processing nodes of a system are introduced stating their most important characteristics. Stations are treated as black boxes with connectors, called "ports", by the other layers of the architecture description. To allow for the design of multiprocessor nodes, a "compound station" was defined as a set of stations which are logically and/or physically strongly connected (they share the same connections with other stations or peripheral devices). The basic components of a station are its processing elements, workstores and devices. There may be more stations in a system, so each one of them is uniquely identified. Each station in a system is associated with the state information.

Several types of stations were defined. The default type is the BASIC station, being a general purpose processing node. To enable describing asymmetrical architectures, we defined two additional types of processing nodes: TASK for task processors and KERNEL for operating system kernel processors. Since (intelligent) peripheral devices are very important in embedded system design, the PERIPHERAL station type was defined. A multiprocessor node is introduced by the PART OF attributes of the constituent processing nodes.

Processing elements (PROCTYPE) have their IDs and speed descriptors, indicating the clock generator's frequency. In general, it is possible to mix processors with different clock rates within the same system, although it may not always be a wise decision. Beside the general attributes, the processing elements may also have additional attributes, depending on the nature of a station. Kernel station processors have properties which are specific for them and are useful to the system designer (e.g., scheduling strategy specification, maximum number of tasks handled by the processor, maximum number of synchronisers, events, queued events and schedules which they support, real-time clock resolution).

Workstores are described by sizes and memory maps (they show the purpose of different areas of memory). The wait-cycles associated with the workstore areas may also be specified (on-chip, RAM or ROM memories usually have different access times). This information is used by the compiler to determine the maximum execution times of tasks, which are loaded in these memory areas or access them during their execution.

Devices are identified by IDs (like stations, but they may be assigned a logical name for easier reference). The device types may vary and have different attributes assigned depending on their nature. Currently, interfaces, timers and shared variables are supported. The use of specific standard devices is supported through the generic device specification.

Configuration division The configuration division is dealing with software architectures. The largest piece of software associated with a station and its state is a "collection" of "modules". Modules consist of "tasks" which may communicate through "ports". Each program part has its unique name for reference. Modules are further described by their import and export parts, in which it is stated which data structures and tasks are shared with other modules.

Tasks are described by their trigger conditions and response times. Task alternatives are given, which serve the purpose of increasing fault-tolerance and to enhance the feasibility of task scheduling (during scheduling a task with shorter run time or longer requested response time can be scheduled in order to maintain the feasibility of a schedule).

Collections of modules are loaded to stations. It is also possible to specify under which conditions certain collections are removed from a station and which collections are loaded instead. These conditions are station state dependent.

The connections between the ports are described by their directions and line attributes. Line attributes state which connections are always installed (VIA attribute) and which ones can be chosen from a list based on the PREFER attribute.

Net division Any net topology of a distributed system can be described by point to point connections. The net division describes the physical PORT to PORT connections between the stations of a system by their logical names and directions.

System division A system division encapsulates the hardware description and the assignment of symbolic names to hardware devices. The described components from the station and net divisions are used.

Peripheral division The peripheral division gives the details about the peripheral devices, which are attached to a system. Peripheral devices are identified by names. Their connections to the devices in the system are described by the attributes of the interfaces used for communication (e.g., the direction of data flow, the protocol used, and any additional signals which may be necessary for communication). To support schedulability analysis, every signal can be associated with its minimum inter-arrival time.

3.2 Graphical Specification PEARL

The graphical equivalent of the textual Specification PEARL language was devised in order to ease the exchange of ideas, and to design specifications graphically. It also represents a good foundation for the construction of a graphical design tool for system architecture specifications in the syntax of Specification PEARL. During design, the constructs are picked from tool-boxes, and are drawn on a specification panel. Right-clicking the mouse, pointing to a construct, may open a window with its properties, while double-clicking may open its lower layer specification if one exists.

There are three layers in the graphical representation of a system architecture:

1. global layer: STATIONs and PERIPHERALs with their PORT to PORT connections,
2. detailed layer: PROCTYPES, WORKSTORES and DEVICES with their internal and global interconnections, and
3. software configuration layer: COLLECTIONS of MODULES and TASKS which are associated with stations on the global layer.

The global and software configuration layers represent the high-level specifications, whereas the detailed layer and collection and module component layers represent lower-level specifications.

Global layer Here the stations with their interconnections are given. All stations share the general properties. The special types of stations have some additional properties, which are listed after the general properties.

STATION (general properties): Name, type (BASIC, COMPOSITE, KERNEL, TASK, PERIPHERAL), super station name (unless the station itself is COMPOSITE), states of operation (NORMAL, EXCEPTION, CRITICAL,...).

PERIPHERAL station: Interface name and description (detailed layer), minimum time between signals.

TASK station: Supervisor name (KERNEL), port name (connection with the supervisor).

KERNEL station: Real-time clock resolution, scheduling strategy, MAXTASK, MAXSEMA, MAXEVENT, MAXQEVENT, MAXSCHEDEVENT.

PORT: Port ID, data flow direction, single transfer unit (smallest item being transferred in a packet) or DMA, synchronisation mechanism.

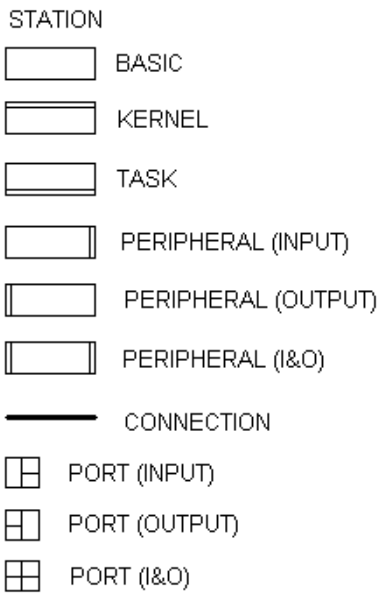

Figure 1: Global layer constructs

Detailed layer The components of the stations with their properties are described on the detailed layer.

Proctype: Processor ID, processor speed.

Workstore: Memory area size, access type (READ/WRITE, READ, EXECUTE), number of wait cycles to access the area.

Device: can be one of the following:

- Interface: Interface ID, driver ID and start address of the driver, data transfer direction, transfer speed, package size or DMA, interrupt vector and level.
- Timer: Timer ID (driver or device), timer activation time, period between signals and duration of activity, timer resolution.
- Shared variable: Shared variable ID or address, signal trigger condition (value change or logical condition), comparison register address.

Standard devices are identified by their identifier, only. Their behaviour is assumed to be known. As in Full PEARL it is assumed here that we have a database of standard devices with their relevant properties.

Software configuration layer The software configuration constructs are given here with their mapping to the global layer stations and PORTs.

Collection: Station ID (residence), state (station state), assignment of logical names to connections between collections (tasks) and stations.

Module: Module ID, collection ID, properties and methods, which the module exports/imports to/from another modules.

Task: Task ID, module ID, collection ID, trigger condition (on demand, timer, interrupt, signal, shared variable), deadline, alternative task ID (scheduled instead if the schedule becomes infeasible).

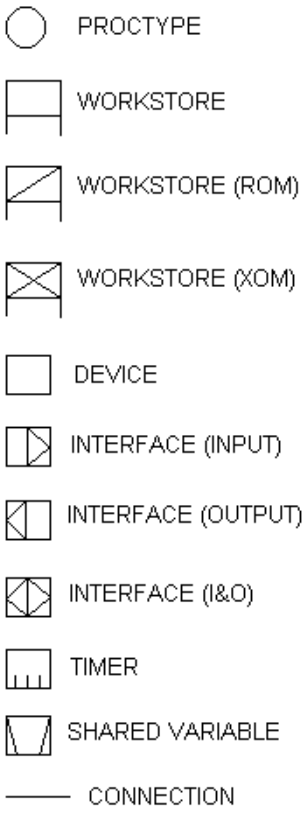

Figure 2: Detailed layer constructs

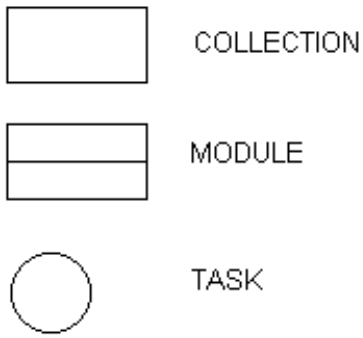

Figure 3: Software configuration layer constructs

3.3 Evaluation

The expressive power of Specification PEARL is considerably greater as compared with similar methodologies. It enables early reasoning about system integration, but at the same time the hierarchical structure of the tool also enables top-down stepwise refinement in design.

The designer first sets up the logical structure of a system, which is being detailed later as well as are implementation parameters. When at least the logical hardware architecture is determined, software units (collections) may be associated with it. The shell of the hardware architecture and the interconnections are sufficient to logically map the software onto hardware. A design can also be started from the software point of view, and the mapping can then be done subsequently, when the stations are present.

Compatibility of parameters is checked instantaneously or while creating the architecture description in the Specification PEARL syntax. A modeled system is checked for completeness and parameter compatibility, since for subsequent design steps a coherent, inambiguous description is needed.

The method for real-time systems' co-specification based on Specification PEARL enables parallel design of the hardware and of software parts, and offers textual as well as graphical notations. For PEARL programmers it provides a good way to design the SYSTEM parts of their programs. On the other hand, it is not strictly bound to PEARL, and its output can be used as input to a configuration manager program or a loader.

Dr. Roman Gumzej
University of Maribor
Technical Faculty
ul. Smetanova 17
SLO-2000 Maribor
roman.gumzej@uni-mb.si

4 RTOS-UH für das Multitalent MPC555

4.1 Motivation

Moderne mechatronische Systeme, wie sie beispielsweise im Bereich der Robotik oder der Fahrzeug-/Fahrwerktechnik zu finden sind, benötigen zur Realisierung komplizierter Regelungsstrategien oftmals ein hohes Maß an Rechenleistung. Häufig ist dabei schnelle Verarbeitung trigonometrischer Funktionen erforderlich. Auf Grund dieser Anforderungen erscheint zur Realisierung auf einem Einplatinenrechner eine Floating Point Unit (FPU) unumgänglich zu sein. Weiterhin werden zur Ansteuerung mechatronischer Komponenten diverse Zeitfunktionen benötigt. Als Beispiel mögen pulsweitenmodulierte Signale zur Regelung von Servomotoren dienen. Mit Ausnahme des Vorhandenseins einer FPU erfüllt beispielsweise der MC68376 die oben genannten Kriterien. Das Manko des MC68376 wird Motorola bald durch einen ähnlichen Prozessor (MPC555) auf Basis der PowerPC-Familie beseitigen. Nach mehrmaligen Zeitverschiebungen soll er nun in Kürze erhältlich sein. Uns stand er auf einem Development Board zur Verfügung. Das Betriebssystem RTOS-UH ließ sich problemlos portieren. Im folgenden sollen die Eigenschaften des MPC555 näher erläutert werden. Dazu werden einige Benchmarks angewendet, die die Leistungsfähigkeit gegenüber bekannten Prozessoren aufzeigen sollen.

4.2 MPC555

Zunehmend können die üblicherweise peripheren Bausteine auf Prozessorchips integriert werden. So verfügt beispielsweise das neueste Mitglied der MPC500-PowerPC-RISC-Mikrocontroller-Familie von Motorola, der MPC555 (Abbildung 4), neben einem PowerPC-Kern und einer 64-bit-FPU über zahlreiche Module zur Anwendung in der Automatisierungstechnik. Die aus der 683xx-Reihe bekannte Time Processor Unit (TPU) liegt in einer weiterentwickelten Form mit 32 frei programmierbaren Timer-Kanälen vor. Neben den 'festvergossenen' Timer-Funktionen (Pulsweitenmodulation, Schrittmotoransteuerung, Quadraturdecoder, Frequenzmessung u.v.m.) lassen sich auch selbst codierte Funktionen einsetzen. Die TPU besitzt eine eigene 'micro engine' und arbeitet somit vollständig autonom, ohne den Prozessor zu belasten.

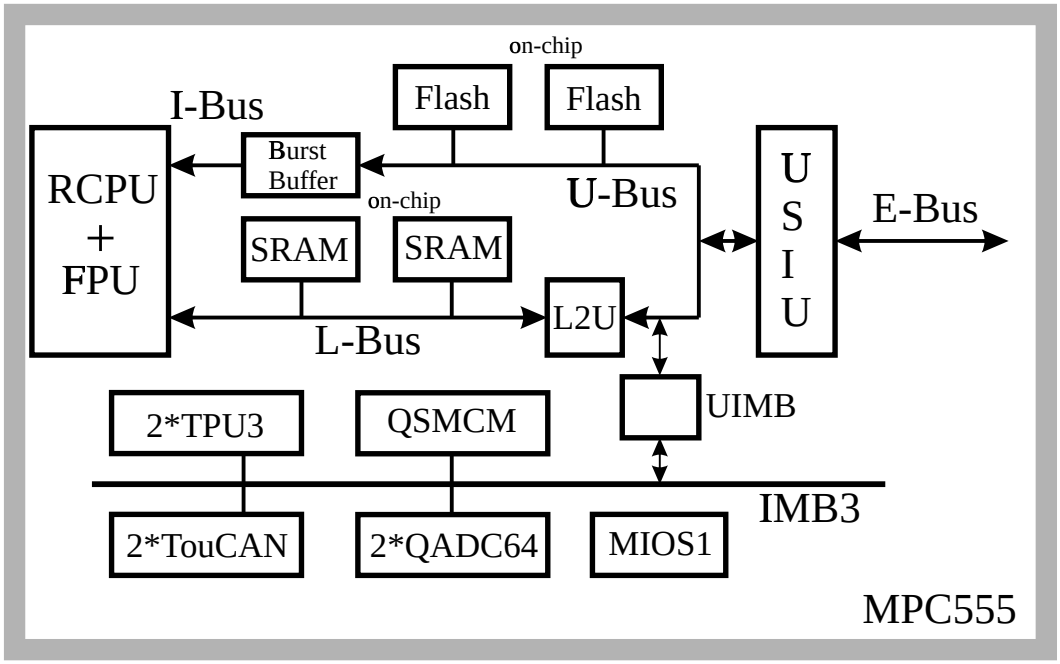

Abbildung 4: Schema der Prozessorarchitektur des MPC555 (Quelle: Motorola Product Preview)

Die TPU der dritten Generation läßt sich intern mit Frequenzen bis hin zur halben Systemfrequenz takten. Zusätzliche Timer-Funktionen (18 Kanäle) sind mit Hilfe des modularen MIOS1-Moduls realisierbar. Dieses erlaubt auch den Aufbau von zwei Parallelports. Das QADC64-Modul bietet die Möglichkeit, bis zu 82 analoge Eingangskanäle mit einer Auflösung von 10 Bit zu digitalisieren. Als Kommunikationsschnittstellen stehen außerdem zwei serielle und zwei CAN-Anschlüsse (TOUCAN) zur Verfügung. Weiterhin kann eine serielle Übertragungstrecke mit einer Datenrate bis zu 10 Mbit/s aufgebaut werden (QSMCM). Nicht zuletzt sind noch das 26 KByte große On-chip-SRAM und das 448 KByte On-chip-Flash-EEPROM zu nennen, das genügend Raum zur Unterbringung eines leistungsfähigen Betriebssystems bietet.

Das interne Flash-EEPROM ist nicht direkt ansprechbar; es wird über sogenannte 'page buffer' adressiert. 'Read page buffer' dienen zum Einlesen von Daten und Befehlen; 'programming page buffer' beinhalten Daten, die auf den Flash-Baustein zu programmieren sind. Bei einem Lesezugriff wird zunächst untersucht, ob der entsprechende Adreßbereich des Flash-Speichers in einem 'page buffer' vorliegt. Ist dies der Fall, so beträgt die Zugriffs- bzw. die Ausführungszeit eines Befehls genau einen Prozessortakt. Anderenfalls beträgt sie zwei Taktzyklen. Zum Vergleich erfordert das Abarbeiten eines Befehls in einem externen SRAM-Baustein ohne 'waitstates' stets zwei Prozessortakte. Ein Nachteil ist der fehlende Cache, den man sonst in nahezu allen PowerPC-Prozessoren findet. Mit den internen Speicherbausteinen wird dies jedoch in vielen Anwendungen kompensiert. Der Prozessor selbst ist mit 40 MHz betreibbar. Alle nachfolgend vorgestellten Benchmarks beziehen sich auf diese Taktfrequenz. Der externe Quarz auf dem Development Board liefert eine Frequenz von 4 MHz, die mit Hilfe der integrierten Phase Locked Loop (PLL) auf die Betriebsfrequenz erhöht wird. Die PLL mit nachgeschalteten Teilern liefert ebenso den Takt für die seriellen Schnittstellen, die TPU und den Dekrementer, der für die Zeiterfassung und die zeitliche Aktivierung und Einplanung von Tasks zuständig ist. Unter RTOS-UH beträgt das Zeitatom 50 msec, d.h. eine Regelung mit Abtastfrequenzen bis hin zu 20 kHz ist möglich.

4.3 Das Betriebssystem RTOS-UH

Um das Potential des neuen Prozessors tatsächlich ausschöpfen zu können, benötigt man nun noch ein geeignetes Echtzeitbetriebssystem, auf dem die Anwendungs-Software aufsetzen kann. Mit der Eigenentwicklung RTOS-UH verfügt das Institut für Regelungstechnik der Universität Hannover hier über ein adäquates Werkzeug, das sich seit vielen Jahren in vielen tausend Kopien sowohl im industriellen als auch im wissenschaftlichen Einsatz bewährt hat. In seiner gesamten Konzeption ist RTOS-UH gezielt für die Anforderungen optimiert, die bei der Regelung und Automatisierung realer Prozesse auftreten. Besonderes Augenmerk gilt dabei stets den Reaktionszeiten auf externe Ereignisse und den auftreten-

Tabelle 1: Ergebnisse der Benchmarks; Hinweise zu * und ** finden sich im Text

CPU	Takt	FPU	INT	REAL	TRIGLOG	STRING
MPC821	40	aus	0,038	0,666	1,540	0,318
MC68376	21	aus	0,415	3,290	5,380	3,340
MC68060	50	aus	0,041	0,361	0,554	0,266
MC68060	50	ein	0,041	0,058	0,066	0,266
PPC604	300	aus	0,006	0,107	0,163	0,041
PPC604	300	ein	0,006	0,010	0,006	0,041
MPC555	40	aus	0,097	0,730	1,320	0,893
MPC555	40	ein	0,097	0,115	0,127	0,893
MPC555*	40	ein	0,083	0,087	0,095	1,007
MPC555**	40	ein	0,053	0,067	0,093	0,637

den Task-Wechselzeiten. Für die Programmierung von Anwendungen wird hauptsächlich die Echtzeithochsprache PEARL 90 (DIN 66253-2) verwendet. Sie enthält alle notwendigen Elemente zur Ein- bzw. Ausplanung, zur Unterbrechung, Fortsetzung und Beendigung einzelner Tasks sowie Vorkehrungen zur Synchronisation mehrerer Tasks. RTOS-UH wurde auf nahezu alle Motorola-Prozessoren der 68k- und der PowerPC-Familie portiert. Weitere Informationen zum Multitasking-Echtzeitbetriebssystem RTOS-UH sind unter dem URL http://www.rtos.irt.uni-hannover.de zu finden.

4.4 Benchmarks

Ziel der Benchmarks war es, die Eignung des Gespanns aus MPC555 und RTOS-UH hinsichtlich der unterschiedlichen Einsatzgebiete festzustellen. Mit Hilfe von Schleifenoperationen wurden im speziellen das Verhalten bei ganzzahligen Operationen (INT), Gleitkommaoperationen (REAL), bei trigonometrischen und logarithmischen Funktionen (TRIGLOG) und bei Stringoperationen (STRING) untersucht (siehe Tabelle 1).

Die angegebenen Resultate sind die Ausführungszeiten in Sekunden. Die Zahlenwerte sind aber insofern unbedeutend, als daß es lediglich auf das Verhältnis zu den Resultaten der anderen Prozessoren ankommt. Die Programme sind in der Echtzeithochsprache PEARL 90 unter RTOS-UH codiert. Der PEARL 90-Compiler erlaubt durch Übersetzerdirektiven die Erzeugung für das jeweilige Zielsystem optimierten Codes. Um die Ergebnisse überschaubar zu halten, wurde nur von der Option FPU ein/aus Gebrauch gemacht. Die restlichen Einstellungen waren jeweils auf das entsprechende Zielsystem hin ausgerichtet. Als Vergleichsprozessoren wurden verwendet:

MBX-Board mit PowerPC MPC821: Dieser Prozessor wurde vornehmlich für die Bildverarbeitung konzipiert. Er stellt einen guten Vergleich dar, da er ebenfalls über einen mit 40 MHz getakteten PowerPC-Kern verfügt. Der MPC821 besitzt keine FPU, dafür aber einen jeweils 8 Kbyte umfassenden Daten- und Instruktions-Cache.

MC68376 (CPU32): Der MC68376 ist der direkte Konkurrent zum MPC555 aus der 68k-Familie zum Einsatz in regelungstechnischen Anwendungen. Seine Architektur umfaßt u.a. eine TPU (16 Kanäle), ein AD-Modul (bis zu 41 Kanäle zu 10 Bit), einen CAN-Controller, ein CTM-Modul (9 zusätzliche vordefinierte Timer-Funktionen), 4 Kbyte On-chip-SRAM und einen loop-instruction-cache mit drei Langworten Kapazität. Die derzeit empfohlene höchste Systemfrequenz beträgt 21 MHz. Eine Variante mit FPU ist bisher nicht verfügbar.

MC68060: Der Vergleich mit dem MC68060 erlaubt, die architektonischen Merkmale zu verdeutlichen, und gibt damit gleichzeitig über die Leistungsfähigkeit der RISC-Technologie gegenüber den CISC-Prozessoren Aufschluß. Er stellt den leistungsfähigsten Prozessor der 68k-Familie dar. Untersucht wurde eine VME-Bus-Karte (MVME 177), die mit 50 MHz getaktet ist. Es gibt jedoch auch eine 60 MHz-Version.

MTX-Board mit PowerPC PPC604: Der PPC604 mit 300 MHz stand als Referenzsystem aus der Klasse der modernen Hochleistungsprozessoren mit mehreren hundert MHz Taktfrequenz zur

Verfügung. Im Versuch soll er zudem die übliche Leistungssteigerung bei Hinzunahme der FPU für einen PowerPC-Kern aufzeigen.

Für den MPC555 wurden vier Betriebsarten untersucht. Das Betriebssystem befand sich stets im internen Flash-EEPROM.

1. Programm und Daten im externen SRAM (0 waitstates); Compiler-Direktive 'ohne FPU'
2. wie Ziffer (1), jedoch 'mit FPU'
3. Programm und Daten im internen SRAM; Compiler-Direktive 'mit FPU' (in Tabelle 1 mit * gekennzeichnet)
4. Programm im externen SRAM (0 waitstates); Task-Workspace (Daten) der einzelnen Benchmark-Tasks im internen SRAM, 'mit FPU' (in Tabelle 1 mit ** gekennzeichnet)

Bei der Compiler-Direktiven 'ohne FPU' werden die Gleitkommaoperationen emuliert.

4.5 Ergebnisse und Interpretation

- Zur Einschätzung der Verluste, die sich durch den fehlenden Cache ergeben, eignet sich die Gegenüberstellung mit dem MPC821, der den gleichen Prozessorkern besitzt und auch mit der gleichen Taktrate betrieben wird. Vergleicht man die Ergebnisse der Integer-Operationen mit dem MPC555 ohne FPU-Option, so ist der MPC821 um einen Faktor von ca. 2,55 besser. Ein ähnliches Ergebnis ergibt sich auch beim STRING-Benchmark. Bei der Gleitkommarechnung beträgt der Vorteilsfaktor lediglich 1,1 und beim TRIGLOG schneidet bereits der MPC555 um den Faktor 1,17 besser ab. Die Erklärung für dieses Phänomen findet sich in der Emulation der Gleitkommaarithmetik, die im Betriebssystem stattfindet. Da die FP-Emulation beim MPC555 somit im internen Flash-EEPROM stattfindet, sind die erforderlichen Operationen sehr schnell durchführbar. Die Berechnung trigonometrischer Funktionen beruht systemintern auf einer Potenzreihenentwicklung, die vornehmlich in Schleifenkonstrukten realisiert ist. Die Wahrscheinlichkeit, den nächsten Befehl im 'read page buffer' des aktuellen Adreßbereichs des Flashs zu finden, ist dementsprechend groß und eine Vielzahl von Befehlen kann innerhalb eines Systemtaktes des MPC555 ausgeführt werden. Bei den trigonometrischen und logarithmischen Funktionen scheint dieser Effekt soweit zu überwiegen, daß der fehlende Cache mehr als nur kompensiert wird.
- Eine interessante Feststellung gelingt durch die Untersuchung des Leistungsvorteils bei eingeschalteter FPU. Für den MPC555 beträgt diese für die Gleitkommaoperationen 6,35 und für den TRIGLOG-Benchmark 10,4. Die entsprechenden Werte beim PPC604 betragen 10,7 beziehungsweise 27,2 und weisen somit auf eine etwa um den Faktor 2 höhere Ausnutzungsmöglichkeit der FPU hin. Die Ursache hierfür liegt in der Datenbusbreite der Prozessoren. Der MPC555 besitzt lediglich einen 32 Bit breiten Datenbus, beim PPC604 ist er 64 Bit breit, so daß nur ca. halb so viele Zugriffe notwendig sind, um die 64-Bit-FPU zu versorgen.
- Ein zunächst unerwartetes Phänomen ist die Verbesserung von der Einstellung * zur Einstellung ** (Programmcode des Benchmarks wird vom internen ins externe SRAM verlagert). Schließlich liegen bei ersterem sowohl die Daten als auch die Instruktionen im schnelleren internen SRAM. Bei der Einstellung ** befinden sich die Instruktionen im externen SRAM und nur die jeweiligen Task-Workspaces, d.h. die Task-Variablen, im internen SRAM. Um eine Erklärung zu finden, bedarf es einer näheren Untersuchung der Prozessorarchitektur (Abbildung 4). Die CPU wird über den L-Bus mit Daten und über den I-Bus mit Instruktionen versorgt. Das interne SRAM befindet sich direkt auf dem L-Bus, so daß Daten sehr schnell ein- und ausgelesen werden können. Instruktionen müssen jedoch den Umweg über das L2U-Interface zunächst auf den U-Bus machen, auf den die externen Speicherbausteine quasi direkt über die Unified System Interface Unit USIU) zugreifen können. Dies deutet darauf hin, daß Datenspeicherzugriffe schneller auf dem internen SRAM und Instruktionszugriffe schneller vom externen SRAM durchführbar sind.

Tabelle 2 zeigt die Leistung relativ zu den untersuchten Vergleichsprozessoren. Es wurden die jeweils optimalen Übersetzereinstellungen zu Grunde gelegt. Die erhoffte Leistungssteigerung im Gleitkommabereich im Vergleich zum MC68376 wurde bestätigt und brachte den "satten" Faktor 50 hervor. Auch

Tabelle 2: Relative Leistung des MPC555** gegenüber den anderen Prozessoren

CPU	Takt	FPU	INT	REAL	TRIGLOG	STRING
MPC555	40 MHz	ja	100%	100%	100%	100%
MC68376	21 MHz	nein	12,8%	2,0%	1,7%	19,2%
MPC821	40 MHz	nein	143%	10,1%	6,0%	200%
MC68060	50 MHz	ja	125%	111%	143%	250%

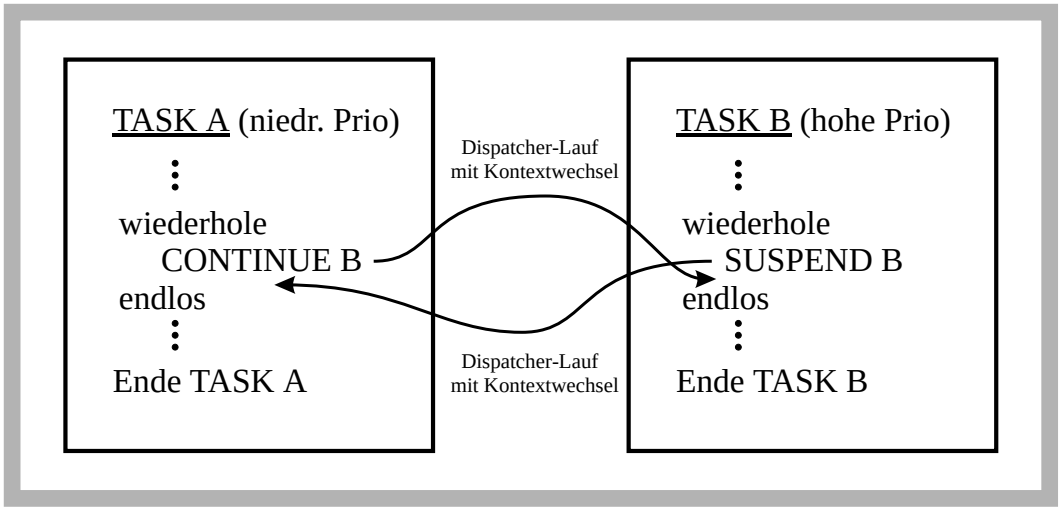

Abbildung 5: Continue/Suspend-Szenario zur Ermittlung der Task-Wechselzeiten

mit dem leistungsstärksten Vertreter der 68k-Familie (MC68060) kann sich der MPC555 messen. Bezieht man beide Prozessoren auf die gleiche Taktrate, so besitzt der MPC555 bei Gleitkommaoperationen sogar einen kleinen Vorteil. Bei den Ganzzahl- und String-Operationen kann er dem MPC821 mit Cache nicht gefährlich werden. Dafür überragt er auch hier hinsichtlich der Gleitkommafunktionen.

4.6 Task-Wechselzeiten

Für die Reaktivität eines Prozeßrechensystems sind die Task-Wechselzeiten von zentraler Bedeutung. Sie geben an, wie schnell der Dispatcher die Prozessorleistung der Task mit der aktuell höchsten Priorität zuteilen kann, wenn diese den Zustand 'lauffähig' erhält. Es ist dann ein Kontextwechsel durchzuführen, der die Registerbelegung der zur Zeit laufenden Task speichert, um sie später im aktuellen Zustand weiterlaufen zu lassen. Weiterhin ist ggf. der alte Zustand der neuen 'lauffähigen' Task zu laden. Sie kann ja zuvor durch eine Selbstblockierung oder eine Fremdsuspendierung in den Zustand 'blockiert' versetzt worden sein. Als Vergleichsmaßstab möge das Continue/Suspend-Szenario dienen. Die Abbildung 5 verdeutlicht den Ablauf. Sowohl Task A (niedrige Priorität) als auch Task B (höhere Priorität) befinden sich in einer Dauerschleife. Task B führt in ihrer Dauerschleife eine Selbstsuspendierung durch. Die Aufgabe der Task A besteht in der Fortsetzung der Task B. Über einen Zeitraum von 30 Sekunden wird jeweils die Anzahl der A-B-A-Zyklen gezählt. Bei den Versuchen befanden sich 30 Tasks im Dispatcher-Ring. Task B nahm in dieser prioritätsorientierten Kette die achte Position ein. Für Task A wurden die Task-Wechselzeiten für die Positionen 9 und 19 im Dispatcher-Ring gemessen. Die Ergebnisse sind der Tabelle 3 zu entnehmen.

Um die Ergebnisse richtig interpretieren zu können, muß man die Arbeitsweise des Dispatchers kennen. Der Dispatcher selbst ist Bestandteil des Betriebssystems und übernimmt immer dann die Kontrolle, wenn sich der Zustand einer Task ändert. Im sogenannten Dispatcher-Ring befinden sich alle Tasks, System-Tasks und User-Tasks, die sich nicht im Zustand 'ruhend' befinden. In diesem Ring sind sie nach Prioritäten sortiert. Der Dispatcher untersucht nun den Dispatcher-Ring solange, bis er eine Task im Zustand 'lauffähig' findet und startet diese sodann. Hieraus erklären sich die unterschiedlichen Messungen in Tabelle 3. Ein längerer Dispatcher-Lauf (für die Konstellation 8/19) geht folglich mit einer höheren

Tabelle 3: Task-Wechselzeiten für A-B-A-Zyklus

Pos. von B/A im Dispatcher-Ring	PowerPC604 300 MHz	MPC821 40 MHz	MC68060 50 MHz	MC68376 21 MHz	MPC555 40 MHz
8/9	1,96 μsec	16,88 μsec	13,49 μsec	110,00 μsec	26,30 μsec
8/19	2,06 μsec	18,38 μsec	14,60 μsec	123,80 μsec	28,17 μsec

Tabelle 4: Task-Wechselzeiten für A-B-A-Zyklus im 'worst case'

Pos. von B/A im Dispatcher-Ring	PowerPC604 300 MHz	MPC821 40 MHz	MC68060 50 MHz	MPC555 40 MHz
8/9	3,03 μsec	23,13 μsec	20,00 μsec	26,30 μsec
8/19	6,14 μsec	30,00 μsec	31,42 μsec	28,17 μsec

Task-Wechselzeit einher.

Die verwendeten Vergleichsprozessoren aus der PowerPC-Familie verfügen, wie bereits erwähnt, über einen Cache. Sie sind daher im gewählten Szenario insofern im Vorteil, als daß sich nach einem A-B-A-Zyklus sämtliche Befehle der beiden Tasks einschließlich des Dispatchers selbst und insbesondere die beim Durchsuchen des Dispatcher-Ringes betrachteten Datenbereiche im Cache befinden. Dadurch kann der Dispatcher-Lauf sehr schnell ausgeführt werden. Die Ergebnisse stellen somit eine theoretische Untergrenze für die erreichbare Task-Wechselzeit dar. Ein weiterer Test soll den 'worst case' untersuchen. Dazu werden beide Tasks vor der Continue- bzw. Suspend-Anweisung um einen speziellen Trap-Befehl erweitert, der den gesamten Cache löscht. Die um den Lauf des Trap-Befehls bereinigten Ergebnisse sind in Tabelle 4 zusammengestellt.

Auch dieses Experiment gibt nur bedingt das tatsächlich auftretende Verhalten wieder, da bei normalem Betrieb die bei jedem Task-Wechsel ausgeführte Routine fast immer auf Cache-Informationen zurückgreifen kann. Bei völlig leerem Cache holt der Prozessor zudem bei jeder Befehlsabarbeitung eine Cache-Line aus dem Speicher, so daß er hauptsächlich mit dem Anlegen des Caches beschäftigt ist. Die in realen Anwendungen auftretenden Werte liegen zwischen den in den Tabellen 3 und 4 dargestellten Ergebnissen. Die Werte aus Tabelle 3 entsprechen jedoch eher den im normalen Betrieb auftretenden Gegebenheiten.

4.7 Fazit

Der MPC555 erweist sich als ein sehr leistungsstarker Prozessor für eingebettete Systeme. Seine Gleitkommaleistung kann sich mit der des stärksten Vertreters der 68k-Familie messen. Auf Grund der vielfältigen E/A-Möglichkeiten eignet er sich insbesondere für regelungstechnische Anwendungen, die dennoch keine hohe Rechenleistung vermissen lassen möchten. Bei den Task-Wechselzeiten sind jedoch im Vergleich zu anderen PowerPC-Prozessoren Abstriche hinzunehmen.

Amos Albert
albert@irt.uni-hannover.de
http://www.irt.uni-hannover.de/~albert/
+49-511-762-4518
Institut für Regelungstechnik
Appelstraße 11
Universität Hannover
30167 Hannover
Fax: +49-511-762-4536

Björn Wolter
wolter@irt.uni-hannover.de
http://www.irt.uni-hannover.de/~wolter/
+49-511-762-4521

5 PEARL in der Ausbildung und Sprachpflege von PEARL: Bericht des Arbeitskreises 5

5.1 Web-Site des AK 5

Seit Mitte des Jahres hat der AK 5 eine eigene Web-Site (vgl. PEARL-News 1/99, Latest News), die über die PEARL-Homepage erreichbar ist. Der Autor, Prof. Müller von der FH Furtwangen, und der gesamte AK 5 würden sich sehr über weitere Vorschläge hierzu freuen.

5.2 Präsenz von PEARL auf dem "12th Plenary Meeting" des ISO/IEC JTC1/SC22

Vom 20. bis 23. September tagte in Berlin das Normungsgremium SC22 des gemeinsamen Technischen Komitees JTC1 der beiden internationalen Normierungsorganisationen ISO und IEC. Der AK 5 nahm an diesen Sitzungen teil und konzentrierte sich mit seinen Aktivitäten auf die von Prof. Giere (Frankfurt, Obmann des DIN NI-22) angeregte und vom SC22 (Vorsitzender R.H. Follett) in die offizielle Agenda aufgenommene "DIN Presentation of PEARL", die am 22. und 23. September 1999 stattfand und aus einem Vortrag und zwei Rechnerpräsentationen bestand. Der Vortrag "Problem-Oriented Real-Time Programming of Embedded Systems with PEARL 90" sowie die Poster zur Präsentation "PEARL auf Linux" sind über die Web-Site des AK 5 verfügbar. Der Vortrag wurde an die mehr als 40 Delegierten aus Dänemark, Deutschland, Estland, Finnland, Frankreich, Großbritannien, Japan, Kanada, den Niederlanden, Norwegen und den USA zusätzlich in gedruckter Form verteilt. Vom "Convener" der SC22/WG9 (Ada), J.W. Moore, wurde der Vortrag außerdem an die Mitglieder der WG9 sowie vom AK 5 an Toshiaki Kurokawa, Mitglied der japanischen Delegation, nachträglich elektronisch verteilt. Die Präsentation "PEARL for Embedded Systems" wurde von der Firma esd, Hannover, vertreten durch Herrn Krause, am 22., die Präsentation "PEARL auf Linux" von Prof. Müller, Furtwangen, am 22. und 23. September durchgeführt. Beide Präsentationen fanden reges Interesse der Delegierten. In einer der Schlußresolutionen des SC22 wurde die "interesting and informative presentation of PEARL" gewürdigt. In seinem "National Activity Report" bezeichnete Prof. Giere die WG NI-22.01 PEARL als "very active".

Als Fazit kann man sagen, daß die Präsentation auf der SC22-Plenartagung eine ausgezeichnete Möglichkeit war, PEARL auch international bekannt zu machen.

5.3 Recherche zu PEARL in der Ausbildung

Es ist geplant, die Liste der Institutionen, die PEARL in der Ausbildung verwenden, ins Englische zu übersetzen. Außerdem werden alle Mitglieder und Nicht-Mitglieder der FG 4.4.2 gebeten, diese, soweit Information vorhanden sind, weiter zu vervollständigen.

5.4 Treffen des AK 5

Das nächste (neunte) Treffen des AK 5 findet wieder — wie auch in der Einladung zum Workshop PEARL 99 und auf der AK 5-Web-Site angekündigt — am Rande des Workshops am Donnerstag, dem 25. November 1999, um 11 Uhr in Boppard statt.

5.5 Mitglieder

Ideen und Mitarbeit jedweder Art, sowohl von Mitgliedern wie von Nicht-Mitgliedern, sind jederzeit sehr willkommen.

Prof. Georg Thiele
Universität Bremen, FB1
Kufsteiner Straße
28359 Bremen

Tel.: (0421) 218-2442
Fax : (0421) 218-4707
E-Mail: thiele@iat.uni-bremen.de

6 Exponate auf dem Workshop PEARL '99

Auf dem diesjährigen Workshop PEARL '99 wird es erstmalig nicht nur Vorträge, sondern auch Vorführungen von Austellungsstücken geben. Zwei dieser Exponate finden Sie am Ende dieser Ausgabe auf jeweils einer Seite kurz beschrieben.

7 Der PEARL-Kurs der FernUniversität

Unter diesem Titel war bereits in der letzten Ausgabe der PEARL-News ein Beitrag zu finden. Da leider jedoch das darin erwähnte Bestellformular im Eifer des Gefechts beim Drucken der Ausgabe unter die Räder gekommen ist, wiederholen wir hier noch einmal das Wesentliche und fügen diesmal natürlich das Formular bei.

Der an der FernUniversität entwickelte Kurs *02417 Realzeitprogrammiersprache PEARL* (Kursnummer 02417) ist mittlerweile voll verfügbar. Der Kurs stellt eine umfassende Einführung in die Echtzeitprogrammierung an Hand der Sprache PEARL dar. Er ist multimedial und wird auf einer CD-ROM ausgeliefert. Zur Ausführung sind ein PC mit 8 MB Hauptspeicher, CD-ROM-Laufwerk, mindestens VGA-Graphikkarte und Sound-Karte sowie Windows in mindestens der Version 3.0 erforderlich. Neben dem eigentlichen Lernprogramm umfaßt der Kurs eine komplette Programmentwicklungsumgebung mit integrierten vollständigen Dokumentationen.

Da man Echtzeitprogrammierung nicht losgelöst von realer Hardware und zu automatisierenden Prozessen lernen kann, wird als Experimentierplattform ein Einplatinenrechner mitgeliefert, der mit dem PC über eine serielle Schnittstelle kommuniziert und an den Sensoren und Aktoren, und somit beliebige Prozesse wie Modelle, elektrische Eisenbahnen, Kaffeemaschinen usw., angeschlossen werden können. Vom PC aus können Prozeßsteuerungsprogramme in den Einplatinenrechner geladen und dort unter Kontrolle des bewährten Echtzeitbetriebssystems RTOS-UH ausgeführt werden, das von Herrn Professor Gerth für diese Lernplattform freundlicherweise zur Verfügung gestellt wurde. In dieser Betriebsart übernimmt der PC die Funktion der Bedienkonsole.

Studierende und Gasthörer der FernUniversität können den PEARL-Kurs so wie jeden anderen Kurs belegen und so CD-ROM und Einplatinenrechner beziehen. Nur am PEARL-Kurs Interessierten wird dieser zum Zwecke der persönlichen Weiterbildung auch unabhängig von einer Einschreibung angeboten. Zum Bezug ist nur das dieser Ausgabe beigefügte Formular auszufüllen und einzusenden.

8 Die Witzseite

Unter der Rubrik “Embedded Systems” und dem Titel “Echtzeitfähiges Java spezifiziert” war in einer Oktober-Ausgabe der Computer-Zeitung folgende Notiz zu lesen:

> Auf den Vorstoß einer Gruppe um Hewlett-Packard und Microsoft, ein eigenes Java für eingebettete Systeme festzulegen, hat Sun mit Echtzeitspezifikationen für ihr Java geantwortet. Die jetzt veröffentlichten Schnittstellenfestlegungen werden die Hersteller von Embedded Systems in die Lage versetzen, zeitkritische Java-Anwendungen zu entwickeln. Die neuen Sun-Spezifikationen entstanden in Abstimmung mit der *Realtime Expert Group*, zu der Hersteller sowohl von Mobilfunkgeräten wie auch aus der Industrieautomation zählen.

Wenn man dies liest, erhebt sich die Frage, warum Java jetzt erst für eingebettete Systeme “aufgebohrt” werden soll — hat man uns Java nicht seinerzeit als **die** Sprache für eben diesen Anwendungsbereich verkauft? Da der Hype-Sprache Java die Echtzeitfähigkeiten noch fehlen, bleibt der dreizigjährige Entwicklungsvorsprung von PEARL erst einmal unangetastet.

Wie charakterisierte Herr Gerth die offensichtlich selbsternannte und in Fachkreisen völlig unbekannte “Realtime Expert Group” doch so treffend mit der Bemerkung: “Die haben die Probleme noch nicht einmal erkannt, die wir vor 20 Jahren schon gelöst haben.”