PEARL-News

Ausgabe 2 November 2004

Mitteilungen der GI-Fachgruppe 'EP' Echtzeitsysteme und PEARL

ISSN 1437-5966

Impressum

Herausgeber	GI-Fachgruppe 'EP' Echtzeitsysteme und PEARL URL: http://www.real-time.de
Sprecherin	Prof. Dr.-Ing. B. Vogel-Heuser Bergische Universität, Fachbereich E Lehrstuhl Automatisierungstechnik / Prozeßinformatik Rainer-Gruenther-Str. 21, D-42119 Wuppertal Telefon: 0202/439-1848 Telefax: 0202/429-1944 E-Mail: bvogel@uni-wuppertal.de
Stellvertreter	Dr. P. Holleczek Universität Erlangen-Nürnberg, Regionales Rechenzentrum Martensstraße 1, D-91058 Erlangen Telefon: 09131/85-27817 Telefax: 09131/30 29 41 E-Mail: holleczek@rrze.uni-erlangen.de
Redaktion	Prof. Dr. R. Müller FH Furtwangen, Fachbereich Computer- & Electrical Engineering Robert-Gerwig-Platz 1, 78120 Furtwangen Telefon: 07723/920-2416 Telefax: 07723/920-2610 E-Mail: mueller@fh-furtwangen.de Prof. Dr. R. Müller HTWK Leipzig, Fachbereich Elektrotechnik und Informationstechnik Wächterstraße 13, D-04107 Leipzig E-Mail: mueller@fbeit.htwk-leipzig.de
ISSN	1437-5966

Redaktionell abgeschlossen am 17. November 2004

Einreichung von Beiträgen
Diese Zeitschrift soll nicht nur Mitteilungsblatt sein, sondern auch eine Plattform für den Informations- und Meinungsaustausch zwischen allen an den Fragen der Echtzeitprogrammierung Interessierten bilden. Diskussionsstoff bzw. offene Fragen gibt es auf unserem Gebiet reichlich.

Wir möchten Sie, liebe Leserinnen und Leser, daher ausdrücklich ermuntern, auch in Zukunft die PEARL-News durch Ihre Beiträge mit zu gestalten. Für ein ausgewogenes Bild der News sollten Beiträge nicht länger als 5 Seiten sein.

Rainer Müller (Furtwangen)
Rolf Müller (Leipzig)

Inhalt

1 Kooperation mit der GMA

Die Fachgruppe Echtzeitsysteme und PEARL (REAL-TIME) wird auch weiterhin in Kooperation mit der GMA (Gesellschaft für Meß-und Automatisierungstechnik) als FG 5.12 geführt. Bisher wurde dort die FG 4.4.1 Echtzeitsysteme geführt. Der Webauftritt im Rahmen der GMA wird auf dem Workshop im November neu diskutiert werden müssen. Vorschläge werden dankend angenommen

B. Vogel-Heuser
bvogel@uni-wuppertal.de

2 Berichte aus den Arbeitskreisen

Viele Arbeitskreise der Fachgruppe existieren schon eine lange Zeit. Aus einigen Arbeitskreisen wird jedes Jahr auf der PEARL-Tagung berichtet — um andere Arbeitskreise ist es still geworden. Anläßlich des Jubiläums ist es sicher angebracht die Arbeitskreise noch einmal vorzustellen. Mit der Absicht, die gesetzten Aufgaben, die Ziele und evtl. auch die erreichten Ergebnisse der Arbeitskreise einmal zusammen darzustellen habe ich die AK-Leiter um einen kurzen Bericht gebeten. Meine Hoffnung dabei war auch, die Stille in einigen Ecken zu brechen. Die Gebiete sind sicherlich nach wie vor interessant. Nachstehend sind die eingegangenen Berichte.

R. Müller
mueller@fh-furtwangen.de

2.1 AK: „Embedded Systems“

Der Schwerpunkt des Arbeitskreises „Embedded Systems“ der GI-Fachgruppe „Echtzeitsysteme und PEARL“ liegt in der praktischen Anwendung der Echtzeitprogrammiersprache PEARL 90 auf eingebetteten Systemen, Mikrocontrollern und Prozessrechensystemen insbesondere unter dem speziell als Laufzeitumgebung für PEARL entwickelten Echtzeitbetriebssystem RTOS-UH.

Der Arbeitskreis findet sich einmal jährlich zu einem Treffen zusammen, das gemeinsam mit der seit 1986 bestehenden PEARL-User-Group stattfindet. An diesem Treffen nehmen neben RTOS-UH/PEARL-Anwendern aus der Industrie und aus Forschungseinrichtungen auch Entwickler und Mitarbeiter von Ingenieurbüros teil, um sich über Weiterentwicklungen des PEARL 90 Compilers und des Laufzeitsystems oder über neue Portierungen auszutauschen. Darüber hinaus erfolgen Erfahrungsberichte der einzelnen Nutzer aus ihren Anwendungsbereichen und über spezielle Applikationen sowie den damit verbundenen Problemen. Im Anschluss an das offizielle Treffen besteht die Gelegenheit zu vertiefenden Diskussionen in kleinerer Runde. Neben dem Informationsaustausch können daher auch neue Kontakte zu Entwicklern oder anderen Anwendern geknüpft werden.

Neue Mitglieder sind in dem Arbeitskreis jederzeit willkommen.

Prof. Dr.-Ing. W. Gerth
gerth@irt.uni-hannover.de

2.2 AK: „PEARL und Echtzeitbetriebssysteme“

Für mich selbst unerwartet und nicht ganz freiwillig musste ich Ende des Jahres 2001 fast alle Aktivitäten an der UniBw München aus gesundheitlichen Gründen aufgeben. Dadurch dass ich keine Vorlesungen halten konnte, schwand der Kontakt mit den Studenten, und durch eine größere Umorganisation der Fakultät auf Grund mehrerer frei werdender Professorenstellen war auch kein direkter Nachfolger da, der meine Arbeiten hätte fortführen können. In der Zwischenzeit erfährt in der Regel keiner der Studenten etwas über die Programmiersprache PEARL.

Dies führte dazu, dass spezielle PEARL- Aktivitäten nicht mehr im Rahmen von Studien- und Diplomarbeiten unterstützt werden konnten. Nur die Arbeiten zur Echtzeitfähigkeit von Linux konnten fortgesetzt werden, da ein Mitarbeiter, den ich noch eingestellt habe, seine Dissertation damit bestreiten will. Durch dessen Kontakte zu den Studenten konnten auch einige Implementierungen mit Studien- und Diplomarbeiten unterstützt werden. Die Schwerpunkte der Untersuchungen waren dabei:

1. Analyse der Linux- Kernel 2.0 bis 2.4 und 2.6
2. Einfügen von Preemption- Points in den Kernel
3. Entwurf für unterbrechbare Kernel- Funktionen: Möglichkeiten, Probleme und Limitierungen
4. Techniken zur Verkürzung der Interrupt- Latenzzeiten
5. Verbessern der Zeitauflösung bei minimalem zusätzlichen Overhead

Die Arbeiten wurden zum Teil in Kontakt mit der Firma MontaVista durchgeführt, die auch verschiedene Ansätze in ihre Arbeiten aufgenommen hat. Es wurden auch Ansätze im UTIME Patch der Kansas University aufgegriffen und weiterentwickelt. Zu erwähnen ist noch, dass der Preemption Patches, der von Linus Torvalds unter der Kerneloption „Preemptible Kernel“ in den Kernel 2.6 aufgenommen wurde, auf diesen Ideen beruht. In den News des Heise-Verlags findet sich am 12.10.04 unter dem Stichwort „Echtzeit-Initiative für Linux von Montavista“ auch ein Hinweis auf diese Arbeiten an der UniBw München. Die entwickelten Kernel- Patche sind auch unter `http://inf3-www.unibw-muenchen.de/research/linux` verfügbar (GPL-2- Lizenz). Die oben erwähnte Dissertation wird voraussichtlich Anfang nächsten Jahres abgeschlossen sein. Dort können dann die erreichbaren Ergebnisse in den Einzelheiten nachgelesen werden.

Meine eigenen Aktivitäten enden damit auch. Da die Arbeiten weitgehend dokumentiert sind, sind alle Interessenten eingeladen, diese aufzugreifen. Zu tun gibt es noch genügend.

Prof. Dr. Helmut Rzehak
rz@informatik.unibw-muenchen.de

2.3 AK: „Modellierung“

Der AK Modellierung wurde im Herbst 2001 gegründet. Während des ersten Treffens an der Universität Wuppertal wurde ein ambitioniertes Programm aufgestellt, welches das Ziel verfolgte, die speziellen Anforderungen der Modellierung von Echtzeitsystemen mit Blick auf die industrielle Anwendung unter anderem in der Automatisierungstechnik und dem Automobilbereich zusammenzustellen und Lösungsvorschläge zu erarbeiten. Diese Lösungsvorschläge sollten in „Patterns“ fixiert und zur Verfügung gestellt werden. Der AK traf sich ca. alle 2 – 3 Monate in Hamburg, Koblenz, Stuttgart und Wuppertal. Die Interessenten stammten aus der Hochschule (HAW Hamburg, TU Braunschweig, Uni Duisburg, IPA Stuttgart, FeU Hagen, TU Ilmenau, Uni Koblenz, Uni Wuppertal) und der Industrie (ATR, Daimler Chrysler, Bosch). Während des ersten Jahres erfolgt ein sehr fruchtbarer Gedankenaustausch der Erfahrungen mit der Modellierung an sich sowie bezüglich der eingesetzten Werkzeuge. Das Treffen in Hamburg diente insbesondere der Klärung von Konzepten in verschiedenen objektorientierten Programmiersprachen. In Koblenz wurden anhand von drei Vergleichsbeispielen die Vor- und Nachteile der verschiedenen Modellierungstechniken (SA/RT, UML, UML-RT) untersucht (ein Kopierer, eine Heizungsregelung, Pressenregelung). Die aktuellen Ideen der UML-Standards und der OMG zur Echtzeitverarbeitung wurden jeweils vorgestellt (Uni Duisburg). Die Idee der Festlegung einer objektorientierten Modellierung, welche die Anforderungen von Echtzeitsystemen berücksichtigt (Zeitverhalten, Priorisierung, Scheduling-Verfahren, Abbildung der Hardwareaspekte, etc.) wurde auf Basis eines Thesenpapiers von Herrn Kaltenhäuser voran getrieben. Nach einer längeren Phase der Begriffsklärung konnten einige Mechanismen erarbeitet und mit Automaten beschrieben werden. Die Arbeiten ruhen zur Zeit, werden aber hoffentlich bald wieder aufgenommen. Während dieser Zeit entstanden zwei Dissertationen, der am AK beteiligten Teilnehmer (T. Licht, T. Heverhagen).

Ansprechpartner: Prof. Vogel-Heuser, Prof. Kaltenhäuser

Prof. Dr. B. Vogel-Heuser
bvogel@uni-wuppertal.de

2.4 AK „Echtzeit in der Ausbildung und PEARL, PEARL-Sprachpflege“

PEARL in der Ausbildung

Eine der ersten Aufgaben des Arbeitskreises AK „PEARL in der Ausbildung“ nach seiner Einrichtung war die Erstellung einer Dokumentation aller Institutionen (Universitäten, Fachhochschulen, Berufs-Akademien, Schulen, andere), die in der Ausbildung PEARL oder Konzepte von PEARL verwenden. Die aktuelle Liste dieser Institutionen weist 21 Universitäten, 23 Fachhochschulen und 6 Berufsakademien aus. Der Ansatz zu einer Ausbildung auch an einer Schule konnte leider nicht in die Realität umgesetzt werden. Es ist interessant, dass zur Zeit die Anzahl der Fachhochschulen größer als die der Universitäten ist, was einem bereits seit einiger Zeit zu beobachtenden Trend entspricht. Überlegungen des AK, selbst PEARL-Kurse durchzuführen, wurden vor diesem Hintergrund zurückgestellt, da auch entsprechende Angebote keine große Resonanz zeigten.

PEARL-Sprachpflege

Da die Verbreitung von PEARL auch wesentlich von der Existenz bzw. Fortschreibung einer gültigen Norm abhängt und der AK die regelmäßige Vertretung der PEARL-Fachgruppe im DIN übernahm, war es naheliegend, dass er sich maßgeblich an der Normung von PEARL 90 (DIN 66252-2, 1998) beteiligte. Dies war auch der Anlass, die „PEARL-Sprachpflege“ zusätzlich in den „Titel“ des AK mit aufzunehmen. Um PEARL auch international zugänglich und damit den Inhalt des Standards bekannt zu machen, wurde als nächster Schritt ein englischer Sprachreport auf der Basis des deutschen Standards zusammen mit dem AK „Embedded Systeme“ (Prof. Gerth) und der FernUniversität Hagen (Prof. Halang) entwickelt und im Internet veröffentlicht (`http://www.real-time.de`).

Liste exemplarischer Projekte und Veröffentlichungen

Zur Linie der bereits genannten Aktivitätsbereiche des AK 5 gehört auch die Herausgabe einer Liste von exemplarischen Projekten mit PEARL im Internet. Dazu zählt auch die Erstellung einer Liste über Veröffentlichungen, die sich auf PEARL beziehen oder die auf PEARL Bezug nehmen. Auf Anregung des DIN wurde des weiteren eine „Success-Story“ von PEARL aufgeschrieben und vom DIN im Internet veröffentlicht.

Vorträge und Exponate

Im Rahmen seiner Mitgliedschaft im DIN NI-22 (Normungsausschuss Informationstechnik: Programmiersprachen, Spiegelgremium des ISO/IEC JTC 1 SC 22) stellte der AK die Realzeit-Sprache PEARL in einem ausführlichen Vortrag vor. In einer Kurzfassung wurde PEARL auch auf dem SC 22 Plenary Meeting 1999 in Berlin, das vom DIN als Gastgeber ausgerichtet wurde, vorgestellt, zusammen mit Exponaten der FH Furtwangen (Prof. Müller) und der Firma esd (Hr. Krause), die auch über die Homepage des AK zugänglich und in den „Resolutions“ des SC 22 aufgeführt sind. Außerdem wurde PEARL auf dem IMACS-Welt-Kongress 1997 in Berlin im Rahmen eines wissenschaftlichen Beitrags vorgestellt (Problem-oriented Real-Time Programming of Embedded Systems with PEARL 90). Weiterhin war der AK auf diversen EchtZeit-Tagungen mit Exponaten präsent.

Projekt-Unterstützung

In neuerer Zeit unterstützt der AK insbesondere Realzeit-Projekte, in die PEARL als Entwurfssprache bzw. als konzeptionelle Basis eingebracht wird, insbesondere zum Thema „Realzeit-Software-Entwurf und Verlässliche Automatisierung mit Funktionsblock-Diagrammen nach IEC 61131“. Hierzu gibt es zwei Veröffentlichungen in den Proceedings der jährlichen PEARL-Workshops (z.B. „Entwurf und Implementierung von zertifizierbaren Verlässlichkeits-Funktionen für die Fuzzy-Regelung eines chemischen Prozesses mit analytischer Redundanz“, 2002) sowie zwei weitere Beiträge auf den internationalen Konferenzen SAFEPROCESS 2003 in Washington und Control Systems Design CSD 2003 in Bratislava („Verifiable-by-inspection fault-tolerant control software for a dependable chemical process using analytical redundancy“).

Zukunft des AK

Die sich abzeichnende wachsende Bedeutung der Ausprägung von PEARL als konzeptionelle Basis für den Realzeit-Software-Entwurf führte kürzlich zu einer Änderung des Titels, nämlich in „Echtzeit-Ausbildung und PEARL, PEARL-Sprachpflege. Überlegungen, PEARL über die Existenz von impliziten Klassen zur Realzeit-Programmierung hinaus dem allgemeinen Trend der Programmiersprachen nach Erweiterungen um Konstrukte zur Objekt-Orientierung anzupassen, werden noch kontrovers diskutiert und daher bisher noch nicht umgesetzt. Interessant könnte die weitere Verfolgung z.B. der Entwicklung der Sprache Ada sein, zu der inzwischen die internationale Normung eines „Profiles“ auf dem Wege ist, um „sichere Software“ u.a. durch Begrenzung der vorhandenen Sprachmittel zu garantieren (ISO/IEC JTC 1 N7499 - Guide for the use of the Ada Ravenscar Profile in High Integrity Systems, 23.7.2004).

Mitglieder

Der AK hat in den Einladungen zu seinen jährlichen Sitzungen in den PEARL-News stets um neue Mitglieder geworben, allerdings bisher ohne sichtbaren Erfolg. Um Ideen wie z.B. Erweiterungen in Richtung Objekt-Orientierung, Vergleich existierender PEARL-Konzepte mit den aktuellen Entwicklungen anderer Realzeitsprachen und Schlussfolgerungen im Sinne eines „Guide to Real-time Software Design“ zu erarbeiten und (im Internet) zu publizieren, wäre eine Vergrößerung der Anzahl der aktiven Mitglieder des AK 5 wünschenswert, allerdings auch dringend erforderlich.

Prof. Dr. G. Thiele
thiele@iat.uni-bremen.de

2.5 AK: „Bedienoberflächen von Echtzeitsystemen“

Dieser Arbeitskreis wurde im November 1995 in Boppard am Rande der PEARL Tagung Echtzeitsysteme ins Leben gerufen. Er entstand aus der Idee, die Arbeit aus der Normierung von CAD Systemen zu nutzen, um Standards für die Bedienoberflächen von Echtzeitsystemen zu schaffen. Der Arbeitskreis stand unter der Leitung von Dr. Heinecke.

Im Januar 1996 fand eine Sitzung des Arbeitskreises in der Fachhochschule Hamburg (der heutigen HAW Hamburg) statt. Sie wurde gefolgt von weiteren Sitzungen in Lüneburg bei der Firma Werum und in der Fachhochschule Dortmund, sowie von Sitzungen am Rande der PEARL Tagungen in Boppard.

Bie diesen Treffen wurde die Normen im CAD Bereich gesichtet und analysiert, welche Teile für Standards im Echtzeitbereich genutzt werden können. Schnell wurde erkannt, dass einerseits die Ziele präziser gefasst werden mussten und neue Mitarbeiter gewonnen werden mussten. Im November 1998 beendete der Leiter des Arbeitskreises seine Aktivitäten. Der Autor dieses Beitrags übernahm den Vorsitz kommissarisch in der Hoffnung, dass neue Mitglieder für neue Belebung sorgen.

Prof. Dr. R. Baran
baran@cpt.fh-hamburg.de

3 Automatische PEARL-Code-Generierung zur Emulation verschiedener Schedulingverfahren

Abstract

Der vorliegende Artikel zeigt, wie sich mit einem Interrupt-gesteuerten Echtzeit-Betriebssystem unterschiedliche Scheduling-Verfahren emulieren lassen. Dazu erfolgt eine automatische Code-Generierung mit Hilfe eines Perl-Skriptes, das eine Task-Tabelle aus einer Datei auswertet und einen entsprechenden Scheduler/Dispatcher-Code für das gewünschte Schedulingverfahren in PEARL90 [PEA95] implementiert. Die Implementierung der Emulation findet mit Hilfe des ereignisgesteuerten (Interrupt-gesteuerten)

Echtzeit-Betriebssystems RTOS-UH [Ger04] statt und wird mit Hilfe von Oszilloskopaufnahmen verifiziert. Zu den bisher realisierten Emulationen gehören das Schedulingverfahren STT (Static Time-Triggered Scheduling), RMS (Rate Monotic Scheduling), TTOS (Time-Triggered OS nach OSEK/VDX-Spezifikation [OSE01]) und in ersten Ansätzen die Methode EDF (Earliest Deadline First).

3.1 Einleitung

Unter Scheduling versteht man die Planung der Zuteilung der verfügbaren Rechnerleistung auf die durchzuführenden Aufgaben des Gesamtsystems (Tasks). Dabei unterscheidet man im Allgemeinen zwischen statischen und dynamischen Verfahren [Kop97, LG99]. Bei statischen Verfahren findet das Scheduling im Vorfeld, bei dynamischen Verfahren hingegen zur Laufzeit statt. Um ein statisches Scheduling durchführen zu können, ist die vollständige Kenntnis über die Taskmenge erforderlich. Dies beinhaltet für alle Tasks dieser Menge die Taskattribute, wie etwa die maximale Laufzeit oder die Deadline. Die Realisierung statischer Schedulingverfahren ist generell einfacher und es ist auch weniger Rechenaufwand für Betriebssystemfunktionen zur Laufzeit erforderlich, um Schedulingentscheidungen zu treffen. Demgegenüber bieten dynamische Verfahren eine höhere Flexibilität und häufig auch eine höhere Ausnutzung der Rechnerleistung; die Realisierung ist jedoch aufwändiger.
Schedulierbarkeitsanalysen liefern notwendige und/oder hinreichende Kriterien für die Schedulierbarkeit (Ablaufsteuerbarkeit) einer Taskmenge. Sie sind für den Anwender von höchster Bedeutung, um zu überprüfen, ob alle Deadlines eingehalten werden. Darüber hinaus können die Analysen auch Hinweise für Optimierungspotential geben. Auf eine eingehende Darstellung der Schedulierbarkeitsanalysen wird jedoch an dieser Stelle verzichtet und etwa auf [MG01] verwiesen.

Die unterschiedlichen Schedulingverfahren lassen sich nicht ohne Weiteres in eine Reihenfolge bringen; vielmehr hängt die Eignung der einzelnen Schedulingverfahren maßgeblich von der Anwendung ab. Echtzeit-Betriebssysteme unterstützen jedoch im Allgemeinen nur eine sehr eingeschränkte Menge an Schedulingverfahren. Es kann daher vorkommen, dass ein für die vorliegende Anwendung geeignetes Scheduling vom Betriebssystem, das zur Anwendung kommen soll, nicht unterstützt wird.
Wie der vorliegende Beitrag zeigt, gelingt es unter gewissen Performanceverlusten, die jedoch bei vielen Anwendungen akzeptabel sind, das Scheduling selbst auf die Userebene zu verlagern. Den größten Freiraum bietet dabei ein asynchrones, interrupt-getriebenes Betriebssystem, bzw. ein 'highest priority first'-Konzept, das sich aus der PEARL-Spezifikation ableiten lässt. Der Artikel gliedert sich im weiteren wie folgt:
Abschnitt 3.2 erläutert zunächst das realisierte Perl-Skript zur automatischen Codegenerierung sowie dessen Anwendung. Danach findet in den Abschnitten 3.3 bis 3.6 die Beschreibung der durchgeführten Emulation statt. D.h. das Verhalten unterschiedlicher Schedulingverfahren wird mit einem System nachgebildet, das originär für eine anderes Scheduling konzipiert ist. Ferner wird mit Hilfe von Oszilloskopaufnahmen die Eignung der Emulation verifiziert. Abschließend gibt Abschnitt 3.7 eine Zusammenfassung.

3.2 Perl-Skript zur automatischen Codegenerierung

Die im Rahmen der vorliegenden Untersuchung durchgeführte Emulation findet exemplarisch am Beispiel des RTOS-UH [Ger04] statt. Das Scheduling von RTOS-UH ist als ein prioritätsgesteuertes, preemptives Verfahren zu klassifizieren. Das Scheduling selbst findet zur Laufzeit statt, wobei die Ausführung der Tasks stets in der Reihenfolge 'highest-priority-first' stattfindet. Eine explizite Schedulierbarkeitsanalyse erfolgt im Allgemeinen nicht.
Bis auf einige wenige Ausnahmen sind die dargestellten Lösungen reine PEARL-Lösungen und hängen nicht explizit von RTOS-UH ab. Andernfalls findet eine kurze Kennzeichnung im Text in der Form `<special RTOS-UH>` statt.

3.2.1 Aufruf und Übergabeparameter

Für die exemplarische Emulation wurde das Perl-Skript `schedu_gen.pl` realisiert, das eine automatische PEARL90-Code-Generierung durchführt[1]. Der Aufruf (aus dem Verzeichnis, in dem sich das Skript befindet) sieht wie folgt aus:

```
perl schedu_gen.pl <emu_mode> [options]
```

Mit dem Parameter `<emu_mode>` bestimmt man das gewünschte Schedulingverfahren. Zulässige Parameter sind im Augenblick

- `STT` für 'Static Time-Triggered Scheduling',
- `RMS` für 'Rate Monotic Scheduling',
- `EDF` für 'Earliest Deadline First' und
- `TTOS` für 'Time-Triggered OS'.

Die optionalen Parameter `[options]` ermöglichen eine weitere Anpassung des erzeugten Programm-Codes an das vorliegende System.

- Die Option `'-c'` generiert einen Code für ein $50\mu s$ System (Zeitatom für zyklische, absolute und relative Zeiteinplanungen). Ohne diesen Parameter findet eine Generierung für das Default $1ms$ System statt. <`special RTOS-UH`>
- Mit der Option `'-s'` wird ein Synchronisationslayer aktiviert, wie es etwa für die Untersuchung zeitgesteuerter Systeme notwendig sein kann. Mit Hilfe einer entsprechenden IR-Serviceroutine findet dann eine Synchronisation der Betriebssystem-Zeit auf einen externen Zeitgeber statt. Im Allgemeinen wird dies ein zeitgesteuerter Bus sein.
- Die Option `'-a'` kann sowohl bei `TTOS`, als auch bei `EDF` Anwendung finden und führt lediglich zu einer alternativen Code-Variante, ohne dabei die Funktion zu verändern.
  Bei `TTOS` und `EDF` ist es unter PEARL nicht möglich, das gewünschte Zeitverhalten lediglich durch eine adäquate Reihenfolge der Taskaktivierungen mit entsprechenden Prioritäten einmalig zu Beginn zu realisieren. Vielmehr muss zur Laufzeit eine Scheduler-Task die Zuteilungen vornehmen. Die Option `'-a'` führt dann zu einer alternativen Realisierung dieser Scheduler-Task.

3.2.2 Deklaration von Konstanten

Einige weitere benötigte Parameter werden nicht durch die Kommandozeile übergeben, sondern vom Anwender direkt als Konstanten im Perl-Programm definiert. Der entsprechende Teil im Perl-Programm sieht etwa wie folgt aus:

```
#####################################################################
## Edit these constants if necessary:                              ##
#####################################################################

my $INPUT_FILE  = "schedule.txt";
my $OUTPUT_FILE = "> c:/rtosuh/os_emu/code.pq";
my $BUFF_GLOBAL_1  = 0.002;   # BUFF_GLOBAL for 1ms   system
my $BUFF_LOCAL_1   = 0.001;   # BUFF_LOCAL  for 1ms   system
my $BUFF_GLOBAL_50 = 0.00015; # BUFF_GLOBAL for 50mus system
my $BUFF_LOCAL_50  = 0.00010; # BUFF_LOCAL  for 50mus system
my $MASTER_CYCLE_DURATION = 0.032;

#####################################################################$
```

Die Datei `INPUT_FILE` beinhaltet die Task-Tabelle (Scheduling-Tabelle). Deren Syntax ist in Abschnitt 3.2.3 erläutert. Die Datei `OUTPUT_FILE` ist der Zielort für den generierten Code. Die Buff-Variablen bestimmen Zeitdauern, die den korrekten Ablauf des Schedulers garantieren. Deren Bedeutung wird in den nachfolgenden Abschnitten erläutert, sodass an dieser Stelle auf weiterführende Anmerkungen verzichtet wird.

[1] Das Perl-Skript kann gerne beim Autor per Email angefordert werden.

3.2.3 Datei mit Scheduling-Tabelle

Die Datei `INPUT_FILE` beinhaltet alle notwendigen Angaben zu den Tasks. Eine explizite Schedulierbarkeitsanalyse zu diesen Angaben erfolgt durch die automatische Code-Generierung jedoch nicht! Es wird vorausgesetzt, dass eine entsprechende Analyse im Vorfeld stattgefunden hat, die Angaben somit konsistent und zur Laufzeit erfüllbar sind. Eine Beispieldatei sieht wie folgt aus:

```
#task name   start time(sec)  period(sec)   running time(s)  priority
#-------------------------------------------------------------------
T1               0.000           0.006           0.002           20
T2               0.002           0.008           0.002           15
T3               0.010           0.012           0.003           10
```

Kommentarzeilen beginnen mit dem `#-Zeichen.` Leere Zeilen werden nicht beachtet. Jede gültige Zeile ist wie folgt aufgebaut:

`<task_name> <start_time(sec)> <period(sec)> <running_time(sec)> <priority>`

Dabei sind einzelne Parameter durch mindestens ein Leerzeichen oder einen Tabulator zu trennen. Wird ein bestimmter Parameter nicht benötigt, so ist statt seiner ein Platzhalter zu verwenden, z.B. `------`.

- Der Parameter `<task_name>` ist obligatorisch. Bei `STT` und `RMS` darf jeder Taskname nur einmal vorkommen. Andernfalls erfolgt die Ausgabe einer Fehlermeldung.
- Der Parameter `<start_time>` ist ebenfalls obligatorisch. Bei `STT` und `RMS` steht die Startzeit für die zeitliche Verschiebung gegenüber dem Gesamtstartzeitpunkt. Bei `TTOS` und `EDF` gibt die Startzeit die Verschiebung gegenüber dem Startzeitpunkt jedes so genannten Master-Zyklus (vgl. Erläuterung in den Abschnitten 3.5 und 3.6) an.
- Der Parameter `<period>` ist obligatorisch für die Schedulingverfahren `STT`, `RMS` und `EDF`. Bei `EDF` steht der unter `<period>` angegebene Wert für die Zeitspanne zwischen dem (gewünschten) Startzeitpunkt und der Deadline.
- Der Parameter `<running_time>` wird zur Zeit nicht berücksichtigt. Für die Untersuchung des Einflusses der Tasklaufzeiten sind entsprechende Vorkehrungen bei der Implementation der Tasks selbst vorzunehmen.
- Der Parameter `<priority>` ist nur beim Schedulingverfahren `STT` von Bedeutung. Bei den anderen drei implementierten Schedulingverfahren erfolgt die Vergabe geeigneter Prioritäten (relative Prioritäten der Tasks untereinander) automatisch.

Nachfolgend sind einige Ergebnisse der automatischen Code-Generierung für die unterschiedlichen Schedulingverfahren dargestellt. Die implementierten Tasks führen dabei keinen allzu sinnvollen Code aus und dienen lediglich der Überprüfung der korrekten Umsetzung. Eine Task hat dabei typischerweise den folgenden Aufbau:

```
/************************************************************/
Task1: TASK PRIO 22 RESIDENT GLOBAL;
/************************************************************/
DCL WERT FIXED;
     SEND Task_gestartet TO AUSGANG_T1;
      WERT = ROUND(1365.2*2.5);
      FOR I TO WERT REPEAT
         /* 2.5 ms in der Warteschleife */
      END;
     SEND Task_beendet TO AUSGANG_T1;
END; ! TASK Task1
/************************************************************/
```

Die erste Zeile stellt an einem digitalen Pin (`AUSGANG_T1`) des verwendeten Mikrocontrollers einen High-Pegel ein; die letzte Zeile entsprechend einen Low-Pegel. Damit ist es möglich, die Bearbeitung der Task mit einem Oszilloskop sichtbar zu machen. Ansonsten führt die Task selbst lediglich eine einfache Schleifenoperation aus. Die Anzahl der Schleifendurchläufe wird so vorgegeben, dass die Gesamtlaufzeit der Task (ohne Unterbrechungen) einem vorher definierten Wert entspricht. Bei der hier zur Anwendung kommenden MPC555-Karte sind etwa 1365.2 Schleifendurchläufe je Millisekunde Tasklaufzeit erforderlich. Beim vorliegenden Beispiel der Task `Task1` ist somit eine Tasklaufzeit von $2.5ms$ beabsichtigt. Soll die Tasklaufzeit Jitter unterworfen sein und beispielsweise eine Gleichverteilung im Bereich $[2.5ms, 4.7ms]$ aufweisen, so wäre die entsprechende Zeile zur Berechnung der Schleifendurchläufe durch

```
WERT = ROUND(1365.2*(2.5+2.2*RANF(ZUFALL1,ZUFALL2)));
```

zu ersetzen.
Zur Unterscheidung der einzelnen Tasks in den Oszilloskop-Aufnahmen findet für jede Task eine unterschiedliche Skalierung am Oszilloskop statt. Damit zeigt die Oszilloskop-Aufnahme eine Art 'Prozess-Zeit-Diagramm', aus dem sich das Systemverhalten entlang der Zeitachse ablesen lässt.

3.3 Static Time-Triggered Scheduling (STT)

Unter der Bezeichnung STT wird im Folgenden ein statisches, tabellengesteuertes Schedulingverfahren für periodische Tasks verstanden. Hierbei handelt es sich im Grunde um kein echtes Schedulingverfahren; es wird lediglich eine vorgegebene Taskmenge aus einer Textdatei Eins-zu-Eins umgesetzt. Dieser einfache Mechanismus soll hier als Einstieg dienen und damit die automatische Code-Generierung und die folgenden Oszilloskopdarstellungen etwas besser verdeutlichen.
Für die Methoden STT und RMS (Abschnitt 3.4) kommt die folgende Task-Tabelle zur Anwendung:

Tabelle 1: Taskmenge $\mathcal{T}_1$ mit drei Tasks zur Darstellung der Emulationsmethoden STT und RMS

#task name	start time(sec)	period(sec)	running time(s)	prio
T1	0.000	0.006	0.002	20
T2	0.000	0.008	0.002	15
T3	0.000	0.012	0.003	10

Es sind somit drei periodische Tasks mit unterschiedlichen Wiederholfrequenzen definiert. Zum Zeitpunkt $t = 0.0$ findet die gleichzeitige Aktivierung aller Tasks statt. In der Hochsprache PEARL gilt für die Prioritäten: Je kleiner der Wert, desto höher die Priorität. Somit besitzt die Task T3 die höchste und T1 entsprechend die niedrigste Priorität.
Mit dem Aufruf `perl C:/.../schedu_gen.pl STT` entsteht der folgende Code:

```
!STT (Static time-triggered scheduling)
!Simple cyclic scheduling of periodic tasks.
P=MPC604+FPU;
/*************************************************************/
MODULE SCHEDULER;
SYSTEM;

PROBLEM;

DECLARE START_TIME CLOCK;
DECLARE BUFF INV DURATION INIT(0.001 SEC);

SPECIFY HW_INIT    TASK GLOBAL;
SPECIFY (T1,T2,T3) TASK GLOBAL;
/*-----------------------------------------------------------*/
START: TASK PRIO 5;
/*-----------------------------------------------------------*/
  ACTIVATE HW_INIT;
  START_TIME = NOW + BUFF;
  AT START_TIME + 0.000 SEC ALL 0.006 SEC ACTIVATE T1;
  AT START_TIME + 0.000 SEC ALL 0.008 SEC ACTIVATE T2;
```

```
  AT START_TIME + 0.000 SEC ALL 0.012 SEC ACTIVATE T3;
END;
/*-----------------------------------------------------------*/
MODEND;
/*************************************************************/
```

Die Tasks `T1`, `T2` und `T3` sind in einem anderen Modul (vom Anwender) implementiert und werden daher lediglich spezifiziert. Zusätzlich sei angenommen, dass eine Task mit dem Namen `HW_INIT` vorliegt, die sämtliche Initialisierungsroutinen für die Hardware beinhaltet. Um einen ausführbaren Code zu erzeugen, ist der compilierte Code dann mit den entsprechenden Hex-Files (, die die Tasks `T1`, `T2`, `T3` und `HW_INIT` beinhalten) zu linken. Das Aufsetzen des Systems erfolgt durch Aufruf der Task `START`. Diese Task initialisiert zunächst die Hardware und führt dann sämtliche zyklischen Aktivierungen durch.
Bild 1 zeigt eine Oszilloskopaufnahme, aus der sich das Zeitverhalten ablesen lässt. Die Skalierung der einzelnen Eingänge wurde so vorgenommen, dass die Taskprioritäten direkt mit den Amplituden korrespondieren. Es entsteht somit eine Art 'Prozess-Zeit-Diagramm'. Zu jedem Zeitpunkt findet die Ausführung derjenigen Task statt, die die höchste Amplitude aufweist.

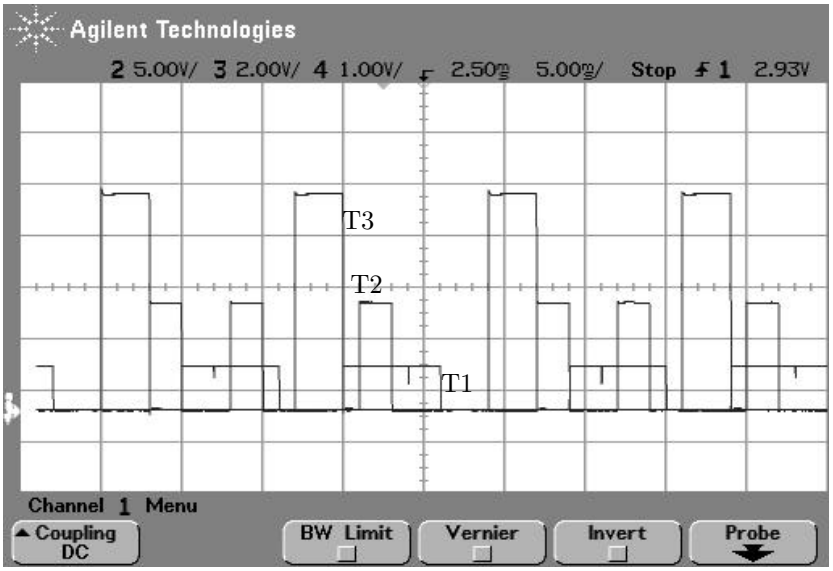

Abbildung 1: 'Prozess-Zeit-Diagramm' bei Anwendung der automatischen Codegenerierung auf die Taskmenge aus Tabelle 1. Als Emulationsmodus kam STT zum Einsatz. Es läuft stets die Task, der die höhere Amplitude zugeordnet ist.

Das kleinste gemeinsame Vielfache der Taskperioden beträgt hier $24ms$. Dies stellt den globalen Zyklus dar, in dem sich das Verhalten wiederholt. Alle Tasks besitzen den gleichen Aktivierungszeitpunkt, sodass der globale Zyklus mit der Abfolge T3 $\rightarrow$ T2 $\rightarrow$ T1 beginnt.
In Bild 1 fallen Störungen bei dem zur Task T1 korrespondieren Signal auf. Diese Ticks zeigen an, dass die Task T1 nach ihrer Ausführung (das entsprechende Signal springt auf den Wert Null) sofort erneut gestartet wird. Demzufolge hat in der Zwischenzeit eine erneute Aktivierung von T1 stattgefunden. Wäre das verwendete Betriebssystem nicht in der Lage, Einplanungen zu puffern, käme es an dieser Stelle zu Verlusten von Einplanungen. Da bei periodischen Tasks im Allgemeinen die Deadline gleich der Periodendauer ist, wären hier somit Deadlines verpasst worden.
Die Prozessorauslastung η lässt sich einfach berechnen, indem man die Verhältnisse der Ausführungszeiten C_i zu den Periodendauern T_i für alle N Tasks berechnet und alles aufsummiert[2]. Im vorliegenden Beispiel ergibt sich

$$\eta = \sum_{i}^{N} \frac{C_i}{T_i} = \frac{0.002}{0.006} + \frac{0.002}{0.008} + \frac{0.003}{0.012} = \frac{1}{3} + \frac{1}{4} + \frac{1}{4} = \frac{10}{12} = 83.3\% \quad .$$

Da die Prozessorleistung keineswegs ausgeschöpft ist, kann man erwarten, dass geeignetere Schedulingstrategien existieren. Insbesondere sollte es möglich sein, keine Deadlines zu verpassen.
Bei verteilten Regelungssystemen ist es im Allgemeinen erforderlich, alle Teilnehmer auf eine gemeinsame, globale Zeitbasis zu beziehen. Die Realisierung dieses Synchronisationslayers erfolgte im Rahmen der Untersuchung in einer Interrupt-Serviceroutine. Um auch den Start des gesamten Schedulings auf ein

[2]Dazu erfolgt der Einfachheit halber die Vernachlässigung der Taskwechselzeiten, des Zeitbedarfs für den Timer-Interrupt und weiterer Betriebssystemfunktionen.

externes Startsignal zu realisieren, ist der Perlskript-Aufruf um die '-s'-Option zu erweitern. Mit der Befehlszeile `perl C:/.../schedu_gen.pl STT -s` ergibt sich:

```
!STT (Static time-triggered scheduling)
!Simple cyclic scheduling of periodic tasks.
P=MPC604+FPU;
/*************************************************************/
MODULE SCHEDULER;
SYSTEM;

PROBLEM;

DECLARE START_TIME CLOCK;
DECLARE BUFF INV DURATION INIT(0.001 SEC);
DECLARE BOLT_SYNC BOLT GLOBAL;

SPECIFY HW_INIT TASK GLOBAL;
SPECIFY (T1,T2,T3) TASK GLOBAL;
/*-----------------------------------------------------------*/
START: TASK PRIO 5;
/*-----------------------------------------------------------*/
  ACTIVATE HW_INIT;
  FREE BOLT_SYNC;
  RESERVE BOLT_SYNC;
  RESERVE BOLT_SYNC;
  START_TIME = NOW + BUFF;
  AT START_TIME + 0.000 SEC ALL 0.006 SEC ACTIVATE T1;
  AT START_TIME + 0.000 SEC ALL 0.008 SEC ACTIVATE T2;
  AT START_TIME + 0.000 SEC ALL 0.012 SEC ACTIVATE T3;
END;
/*-----------------------------------------------------------*/
MODEND;
/*************************************************************/
```

Nach der Initialisierung der Hardware gibt die Task `START` die BOLT-Variable `BOLT_SYNC` zunächst frei (`FREE BOLT_SYNC`) und reserviert (`RESERVE BOLT_SYNC`) sie anschließend wieder. Damit ist `BOLT_SYNC` nun belegt und die Task steht auf der zweiten Zeile `RESERVE BOLT_SYNC` fest. Die Freigabe von `BOLT_SYNC` erfolgt nun durch die Interrupt-Serviceroutine, die auf das Synchronisationsereignis hin gestartet wird. Dieses externe Ereignis muss die Zeitdauer `BUFF` vor dem Startzeitpunkt `START_TIME` erfolgen.

3.4 Rate Monotic Scheduling (RMS)

Beim RMS findet nur die Berücksichtigung von periodischen Tasks statt. Für die Zuteilung der (statischen) Prioritäten kommt folgende Regel zur Anwendung:

> Je kürzer die Periode der Task, desto höher ist die Priorität.

Das Verfahren RMS genießt eine große Verbreitung. Grund hierfür ist die Verfügbarkeit einfach anzuwendender Schedulierbarkeitsanalysen. Weist eine Taskmenge etwa eine Prozessorauslastung von weniger als 69.3% auf, so garantiert der RMS Algorithmus stets die Schedulierbarkeit [LL73, MG01]. Dieser Wert lässt sich aus einem hinreichenden Kriterium ableiten, wonach eine Taskmenge mit N Tasks schedulierbar ist, wenn für die Prozessorauslastung

$$\eta \leq N(2^{1/N} - 1) \tag{1}$$

gilt. Der Wert 69.3% ergibt sich durch eine Grenzwertbetrachtung für $N \to \infty$. Dieses Kriterium ist hinreichend, jedoch nicht notwendig. Für die Taskmenge aus Tabelle 1 liefert das Kriterium keine Aussage, da die Auslastung 83.3% beträgt und somit der Wert 69.3% überschritten wird. Selbst die Auswertung der genaueren Formel 1 liefert lediglich 77.9% und daher auch keine Aussage. Dennoch ist die Schedulierbarkeit erfüllt, wie man Bild 2 entnehmen kann. Hier kam das RMS für die Taskmenge aus Tabelle 1 zum Einsatz.
Bei Anwendung des hinreichenden Kriteriums führt das RMS zu einer relativ schlechten Auslastung bei beliebigen Periodendauern. Es existieren aber auch exakte (notwendige und hinreichende) Verfahren, die jedoch etwas komplizierter in der Anwendung sind. Sind die Perioden der Tasks ganzzahlige Vielfache voneinander, so lässt sich mit dem RMS stets eine Auslastung von 100% erreichen.
Mit dem Aufruf `perl C:/.../schedu_gen.pl RMS -s` entsteht der folgende Code:

```
!RMS (Rate monotonic scheduling)
!Static scheduling of periodic tasks. The shorter the period,
!the higher the priority.
P=MPC604+FPU;
/*************************************************************/
MODULE SCHEDULER;
SYSTEM;

PROBLEM;

DECLARE START_TIME CLOCK;
DECLARE BUFF INV DURATION INIT(0.001 SEC);
DECLARE BOLT_SYNC BOLT GLOBAL;

SPECIFY HW_INIT TASK GLOBAL;
SPECIFY (T1,T2,T3) TASK GLOBAL;
/*-----------------------------------------------------------*/
START: TASK PRIO 5;
/*-----------------------------------------------------------*/
  ACTIVATE HW_INIT;
  FREE BOLT_SYNC;
  RESERVE BOLT_SYNC;
  RESERVE BOLT_SYNC;
  START_TIME = NOW + BUFF;
  AT START_TIME + 0.000 SEC ALL 0.006 SEC ACTIVATE T1 PRIO 10;
  AT START_TIME + 0.000 SEC ALL 0.008 SEC ACTIVATE T2 PRIO 11;
  AT START_TIME + 0.000 SEC ALL 0.012 SEC ACTIVATE T3 PRIO 12;
END;
/*-----------------------------------------------------------*/
MODEND;
/*************************************************************/
```

Bild 2 zeigt das erhaltene 'Prozess-Zeit-Diagramm'. Um einen Vergleich mit Bild 1 zu vereinfachen, wurde die Skalierung der Ausgänge beibehalten. Da aber nun die Prioritäten in der Reihenfolge T1 → T2 → T3 vergeben sind, korrespondiert eine niedrige Amplitude mit einer hohen Priorität und umgekehrt. Task

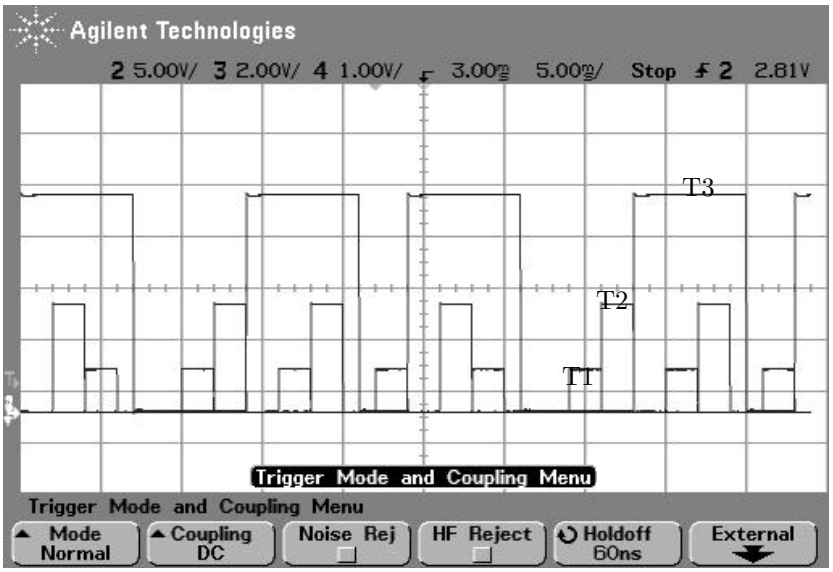

Abbildung 2: 'Prozess-Zeit-Diagramm' bei Anwendung der automatischen Codegenerierung auf die Taskmenge aus Tabelle 1. Als Emulationsmodus kam RMS zum Einsatz. Es läuft stets die Task, der die niedrigere Amplitude zugeordnet ist.

T2 wird hier immer dann von Task T1 verdrängt, wenn für beide eine gleichzeitige Aktivierung vorliegt. Task T3 wird stets durch T1 und T2 unterbrochen; der Verlust einer Deadline tritt nun nicht mehr auf.

3.5 Time-Triggered OS nach OSEK Spezifikation (TTOS)

Rein zeitgesteuerte Betriebssysteme arbeiten nach einem fest definierten Zeitplan, der im Allgemeinen die Bearbeitung der Tasks ohne gegenseitige Unterbrechungen ermöglicht. Zur Erstellung eines effizienten Zeitplanes ist eine WCET-Analyse (**W**orst **C**ase **E**xecution **T**ime) durchzuführen, die angibt, welche

maximalen Ausführungszeiten für die einzelnen Tasks vorliegen. Das Problem liegt hier meist darin, möglichst kleine Werte zu finden, d.h. eine möglichst kleine obere Schranke zu ermitteln, die sicher nicht überschritten wird. Mit Hilfe dieser Analyse lassen sich dann Zeitscheiben für die Tasks und die Datenübertragungen in adäquater Weise festlegen. Die gesamte Planung (das Scheduling) findet somit statisch statt; zur Laufzeit lassen sich keine Veränderungen mehr vornehmen und der Dispatcher muss lediglich die bei der Planung entstandene Tabelle umsetzen. Änderungen während der Entwicklungsphase erfordern automatisch einen Neuentwurf, wobei im Allgemeinen auch eine erneute WCET-Analyse[3] durchzuführen ist.

Tabelle 2: Taskmenge $\mathcal{T}_2$ mit drei Tasks zur Darstellung der Emulationsmethode TTOS

```
#task name      start time(sec) period(sec)     running time(s) prio
#-------------------------------------------------------------------
T1              0.000           0.010           0.0025          10
T3              0.003           0.010           0.0005          15
T2              0.005           0.010           0.0015          20
T3              0.007           0.010           0.0005          15
```

Die Anwendung der automatischen Codegenerierung auf Tabelle 2 führt mit dem Aufruf `perl C:/.../schedu_gen.pl TTOS -s` auf:

```
!TTOS (OSEK standard) - A scheduled task is guaranteed to start execution at
!its start time (decreasing priorities). This version supports several instances
!of the same task with different priorities within the same master cycle.
P=MPC604+FPU;
/************************************************************/
MODULE SCHEDULER;
SYSTEM;

PROBLEM;

DECLARE MASTER_CLOCK CLOCK;
DECLARE MASTER_CYCLE_DURATION INV DURATION INIT(0.01 SEC);
DECLARE BUFF_LOCAL  INV DURATION INIT(0.001 SEC);
DECLARE BUFF_GLOBAL INV DURATION INIT(0.001 SEC);
DECLARE FIRST_TIME  BIT(1) INIT('1'B1);
DECLARE BOLT_SYNC   BOLT GLOBAL;

SPECIFY HW_INIT     TASK GLOBAL;
SPECIFY MASTER_TASK TASK;
SPECIFY (T1,T2,T3)  TASK GLOBAL;
/*-----------------------------------------------------------*/
START:TASK PRIO 5;
/*-----------------------------------------------------------*/
  ACTIVATE HW_INIT;
  FREE BOLT_SYNC;
  RESERVE BOLT_SYNC;
  RESERVE BOLT_SYNC;
  MASTER_CLOCK = NOW + BUFF_GLOBAL;
  AT MASTER_CLOCK + 0.000 SEC ACTIVATE T1 PRIO 1000;
  AT MASTER_CLOCK + 0.003 SEC - BUFF_LOCAL ACTIVATE MASTER_TASK;
  FIRST_TIME='1'B1;
END;
/*-----------------------------------------------------------*/
MASTER_TASK:TASK PRIO 5;
/*-----------------------------------------------------------*/
! One call to MASTER_TASK corresponds to one master cycle.
! The task re-schedules itself upon termination.

  IF FIRST_TIME THEN
    FIRST_TIME='0'B1;
  ELSE
    AT MASTER_CLOCK + 0.000 SEC ACTIVATE T1 PRIO 1000;
    AT MASTER_CLOCK + 0.003 SEC - BUFF_LOCAL RESUME;
```

[3]Selbst die einfache Verschiebung von Code-Abschnitten kann zu veränderten WCETs führen, da dann beispielsweise Burst-Speicherzugriffe, Cache-Aktionen und dergleichen anders ausfallen können.

```
  FIN;
  AT MASTER_CLOCK + 0.003 SEC ACTIVATE T3 PRIO 999;
  AT MASTER_CLOCK + 0.005 SEC - BUFF_LOCAL RESUME;
  AT MASTER_CLOCK + 0.005 SEC ACTIVATE T2 PRIO 998;
  AT MASTER_CLOCK + 0.007 SEC - BUFF_LOCAL RESUME;
  AT MASTER_CLOCK + 0.007 SEC ACTIVATE T3 PRIO 997;

  MASTER_CLOCK = MASTER_CLOCK + MASTER_CYCLE_DURATION;
  AT MASTER_CLOCK - BUFF_GLOBAL ACTIVATE MASTER_TASK;
END;
/*-----------------------------------------------------------*/
MODEND;
/*************************************************************/
```

Der Start des Gesamtsystems erfolgt, wie schon beim RMS in Abschnitt 3.4 gezeigt, über die Task `START` und das externe Synchronisationssignal. `START` selbst plant jedoch lediglich die Task `MASTER_TASK` ein, die anschließend die Aufgabe des Laufzeit-Schedulers/Dispatchers übernimmt.
Innerhalb des Master-Zyklus führt die Task `MASTER_TASK`[4] jeweils nach Einplanung der nächsten Taskaktivierung eine Eigensuspendierung bis zu dem um `BUFF_LOCAL` verminderten Startzeitpunkt der nächsten eingeplanten Task durch. Dieser Zeitpuffer ermöglicht es der Mastertask, die als nächstes anstehende Task einzuplanen und sich selbst für eine definierte Zeit zu suspendieren. Um das Anlaufen einer Task zum Zeitpunkt 0 relativ zum Start des Master-Zyklus zu ermöglichen, findet der Start des Master-Zyklus bereits zum Zeitpunkt t + `MASTER_CYCLE_DURATION` - `BUFF_GLOBAL` statt.
Die beiden erforderlichen Zeitpuffer `BUFF_LOCAL` und `BUFF_GLOBAL` hängen lediglich vom verwendeten System (Hardware und 'Zeitatom') ab und sind unabhängig von der Anzahl der Tasks oder der Taskstruktur. Die Zeitpuffer sind als Konstanten im Perl-Skript definiert (siehe Abschnitt 3.2.2). Kommt die Compileranweisung `MODE=CLOCK50` zur Anwendung, d.h. beträgt das 'Zeitatom' für zeitliche Einplanungen $50\mu s$, so erhält `BUFF_LOCAL` den Wert `BUFF_LOCAL_50` und `BUFF_GLOBAL` den Wert `BUFF_GLOBAL_50`. Bei einem $1ms$-System erfolgt entsprechend die Übergabe der Werte `BUFF_LOCAL_1` und `BUFF_GLOBAL_1`. Die Bestimmung adäquater Zeitpuffer für das jeweils verwendete System findet einmalig mit Hilfe von Experimenten statt. Die Konstante `MASTER_CYCLE_DURATION` definiert die Periode des globalen Master-Zyklus und wird ebenfalls im Perl-Skript definiert.

Im regulären Betrieb sollten bei entsprechender Zeitplanung eigentlich keine Ressourcenkonflikte zwischen Tasks auftreten. Steht beim Start einer Task entsprechend dem Zeitplan jedoch noch eine andere Task in der Ausführung, so sieht die TTOS OSEK/VDX-Spezifikation die Unterbrechung dieser Task vor [OSE01]. D.h., die Aktivierung einer Task unterbricht (preempted) stets die gerade laufende Task.
Um dieses Verhalten mit einem prioritätsgesteuerten Betriebssystem zu emulieren, besitzen die Tasks in der Ablaufsteuerung der Mastertask aufsteigende Prioritäten (absteigende Prioritätenwerte). Die Fehlerbehandlung bei einer Deadline-Überschreitung, wie sie in [OSE01] gefordert wird, findet bei dieser Emulation jedoch nicht statt. Das Ziel der vorliegenden Arbeit ist die Untersuchung qualitativer Eigenschaften. Dafür erscheint es ausreichend, wenn die Emulation ein identisches Verhalten entlang der Zeitachse aufweist.

Bild 3 zeigt das 'Prozess-Zeit-Diagramm' bei Anwendung der automatischen Codegenerierung für die Taskmenge aus Tabelle 2. Die Tasklaufzeiten entsprechen exakt den Werten aus Tabelle 2, sodass beim vorliegenden Zeitplan keine Ressourcenkonflikte zwischen Tasks auftreten.
Die Laufzeiten der Tasks T1 und T2 werden nun zufallsmäßig variiert. Dies soll etwa einen Code simulieren, bei dem die Bearbeitung zustandsabhängig und variabel ist. Bild 4 zeigt das Zeitverhalten. Die Startzeitpunkte der Tasks werden strikt eingehalten. Eventuell dann noch laufende Task werden stets unterbrochen.

3.6 Earliest Deadline First (EDF)

Das Schedulingverfahren EDF ist vom Prinzip her dynamisch und erlaubt auch die Berücksichtigung von sporadischen und aperiodischen Tasks. Die Prioritätszuteilung erfolgt nach der Regel:

Je unmittelbarer die Deadline, desto größer die Priorität.

[4] Sie besitzt die höchste User-Priorität im System.

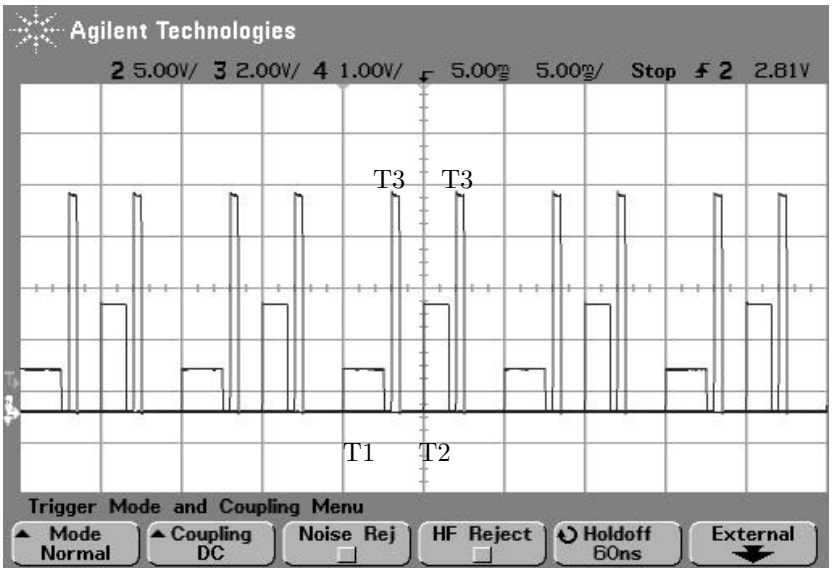

Abbildung 3: 'Prozess-Zeit-Diagramm' bei Anwendung der automatischen Codegenerierung auf die Taskmenge aus Tabelle 2. Als Emulationsmodus kam TTOS zum Einsatz. Die Tasklaufzeiten entsprechen den Werten aus Tabelle 2.

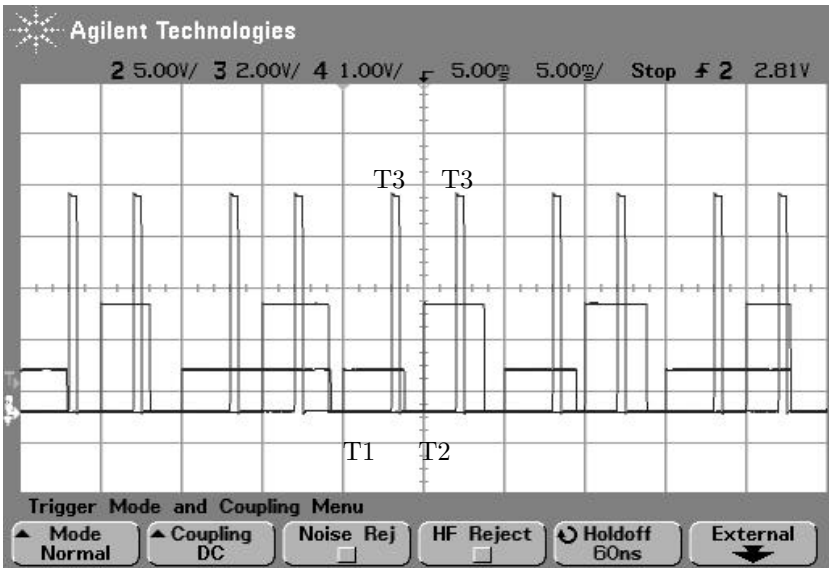

Abbildung 4: 'Prozess-Zeit-Diagramm' bei Anwendung der automatischen Codegenerierung auf die Taskmenge aus Tabelle 2. Als Emulationsmodus kam TTOS zum Einsatz. Zu den Tasklaufzeiten aus Tabelle 2 wurde bei T1 und T2 ein Zufallswert addiert.

Bei aperiodischen Tasks ist die Prioritätszuteilung zur Laufzeit durchzuführen. Eine vollständige Emulation ist somit relativ aufwändig. Die Emulation vereinfacht sich jedoch erheblich, wenn lediglich die Berücksichtigung von einer bekannten Taskmenge stattfindet, die in einem globalen Zyklus definiert ist. Unter dieser Annahme kann das Scheduling statisch erfolgen.
Tabelle 3 definiert eine Taskmenge mit vier Tasks für die Untersuchung der Methode EDF. Die Kon-

Tabelle 3: Taskmenge $\mathcal{T}_3$ mit vier Tasks zur Darstellung der Emulationsmethode EDF

#task name	start time(sec)	period(sec)	running time(s)	prio
T1	0.000	0.009	0.0040	xx
T3	0.003	0.005	0.0015	xx
T2	0.004	0.006	0.0015	xx
T4	0.006	0.003	0.0005	xx
T3	0.007	0.002	0.0015	xx

stante `MASTER_CYCLE_DURATION` im Perl-Skript definiert die Periode des globalen Master-Zyklus. Der

unter <period(sec)> eingetragene Wert gibt hier die relative Deadline, bezogen auf den Startzeitpunkt an. Über die Startzeitpunkte der Tasks ist der Zyklus damit eindeutig definiert. Mit dem Wert MASTER_CYCLE_DURATION=10ms ergibt der Aufruf `perl C:/.../schedu_gen.pl EDF` folgenden Code:

```
!EDF (Earliest deadline first)
!Priorities are assigned according to the tasks' deadlines. The
!closer the deadline, the higher the priority.
P=MPC604+FPU;
/*************************************************************/
MODULE SCHEDULER;
SYSTEM;

PROBLEM;

DCL MASTER_CLOCK CLOCK;
DCL MASTER_CYCLE_DURATION INV DURATION INIT(0.01 SEC);
DCL BUFF_LOCAL  INV DURATION INIT(0.001 SEC);
DCL BUFF_GLOBAL INV DURATION INIT(0.001 SEC);

SPC HW_INIT     TASK GLOBAL;
SPC MASTER_TASK TASK;
SPC (T1,T3,T2,T4) TASK GLOBAL;
/*-----------------------------------------------------------*/
START:TASK PRIO 5;
/*-----------------------------------------------------------*/
  ACTIVATE HW_INIT;
  MASTER_CLOCK = NOW + 2*BUFF_GLOBAL;
  AT MASTER_CLOCK - BUFF_GLOBAL ACTIVATE MASTER_TASK;
END;
/*-----------------------------------------------------------*/
MASTER_TASK:TASK PRIO 5;
/*-----------------------------------------------------------*/
! One call to MASTER_TASK corresponds to one master cycle.
! The task re-schedules itself upon termination.
  AT MASTER_CLOCK + 0.000 SEC ACTIVATE T1 PRIO 12;
  AT MASTER_CLOCK + 0.003 SEC - BUFF_LOCAL RESUME;
  AT MASTER_CLOCK + 0.003 SEC ACTIVATE T3 PRIO 11;
  AT MASTER_CLOCK + 0.004 SEC - BUFF_LOCAL RESUME;
  AT MASTER_CLOCK + 0.004 SEC ACTIVATE T2 PRIO 15;
  AT MASTER_CLOCK + 0.006 SEC - BUFF_LOCAL RESUME;
  AT MASTER_CLOCK + 0.006 SEC ACTIVATE T4 PRIO 13;
  AT MASTER_CLOCK + 0.007 SEC - BUFF_LOCAL RESUME;
  AT MASTER_CLOCK + 0.007 SEC ACTIVATE T3 PRIO 14;
  MASTER_CLOCK = MASTER_CLOCK + MASTER_CYCLE_DURATION;
  AT MASTER_CLOCK - BUFF_GLOBAL ACTIVATE MASTER_TASK;
END;
/*-----------------------------------------------------------*/
MODEND;
/*************************************************************/
```

Bild 5 zeigt die entsprechende Oszilloskopaufnahme. In dieser Aufnahme sind die Amplituden von T1 (kleinste Amplitude) nach T4 (größte Amplitude) sortiert. Mit dem EDF lässt sich stets eine Prozessorauslastung von 100% erreichen, unabhängig davon, ob die Perioden ganzzahlige Vielfache voneinander sind oder nicht.

3.7 Zusammenfassung

Dieser Bericht zeigte in einer ersten Grundsatzuntersuchung, wie sich mit einem prioritätsgesteuerten, preemptiven Betriebssystem unterschiedliche Schedulingverfahren emulieren lassen. Insbesondere ist dabei auch die funktionelle Nachbildung eines zeitgesteuerten Betriebssystemkonzeptes nach der OSEK/VDX-Spezifikation [OSE01] möglich, wie es für sicherheitskritische Anwendungen vielfach gefordert wird. So lässt sich durch die Nachbildung entsprechender Systemfunktionen auch mit einem ereignisgesteuerten Betriebssystem eine zeitgesteuerte Architektur aufrecht erhalten. Das Produkt der Betriebssystem-Emulation lässt sich als ein gemischt ereignis- und zeitgesteuertes Betriebsystem betrachten. Durch die Prioritätenvergabe haben zeitgesteuerte Tasks immer Vorrang und unterbrechen stets ereignisgesteuerte Tasks. Letztere laufen im Hintergrund, sodass sich das System nach Außen konform zur OSEK/VDX-Spezifikation verhält. Die Spezifikation erlaubt beliebige Aktionen innerhalb der Idle-Phase der Zeitsteue-

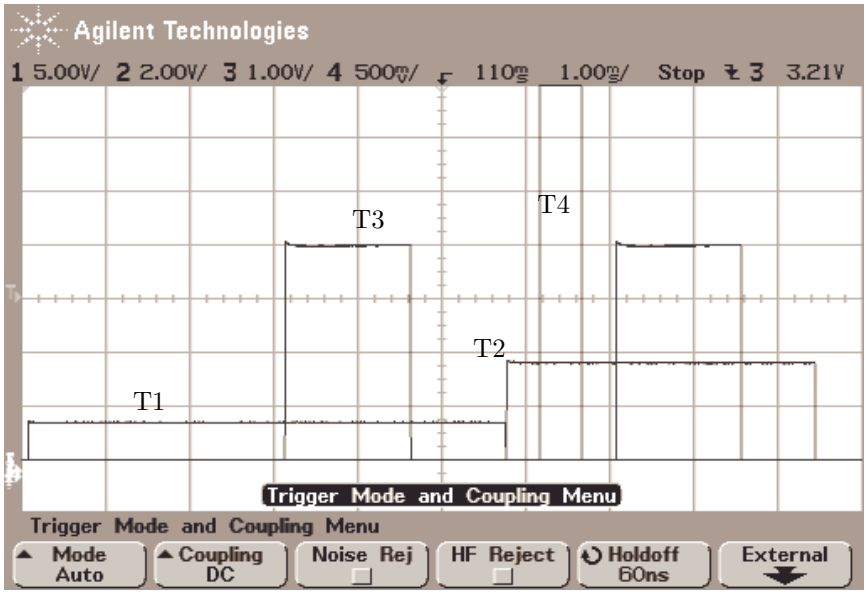

Abbildung 5: 'Prozess-Zeit-Diagramm' bei Anwendung der automatischen Codegenerierung auf die Taskmenge aus Tabelle 3. Als Emulationsmodus kam EDF zum Einsatz.

rung, insbesondere damit auch den Lauf eines OSEK-konformen Betriebssystems [OSE01]. Mit anderen Worten: Dem zeitgesteuerten Betriebssystem lässt sich ein ereignisgesteuertes Betriebssystem unterlagern. Die hier dargestellte Emulation verfolgt den umgekehrter Weg und überlagert dem ereignisgesteuerten Betriebssystem ein zeitgesteuertes.
Mit dem erstellten Hilfsmittel lassen sich etwa unterschiedliche Schedulingverfahren bzw. Betriebssystemkonzepte qualitativ untersuchen. Interessante Fragestellung betreffen das Latenz- und Jitterverhalten der einzelnen Schedulingverfahren sowie der Einfluss auf die Regelgüte bei Anwendung an verteilten Regelkreisen.

Literatur

[Ger04] W. Gerth. *RTOS-UH Handbuch*, Institut für Regelungstechnik, Universität Hannover, aktuelle Internetversion 5.3: http://www.rtos.irt.uni-hannover.de/, 2004.

[Kop97] H. Kopetz. *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers Boston/Dordrecht/London, 1997.

[LG99] R. Lauber und P. Göhner. *Prozeßautomatisierung Band 1*. Springer Verlag Berlin Heidelberg New York, 3. Auflage, 1999.

[LL73] C.L. Liu und J.W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment*, *Journal of the ACM*, 20(1):46–61, Januar 1973.

[MG01] C. Siva Ram Murthy und G.Manimaran. *Resource Management in Real-Time Systems and Networks*, The MIT Press, 2001.

[OSE01] OSEK/VDX, *Time-Triggered Operating System*. http://www.osek-vdx.org, Version 1.0, 2001.

[PEA95] GI-Fachgruppe 4.4.2 Echtzeitprogrammierung PEARL, *PEARL90, Sprachreport*, 1995.

Amos Albert
Robert Bosch GmbH
amos.albert@de.bosch.com

Sam Schneider
sam.schneider@speedgrafix.com