Ingenieurbüro für Echtzeitprogrammierung

Installation des Echtzeit-Betriebssystems RTOS-UH auf dem phyCORE-MPC5*x*5

Dok-Rev. 1.4 vom 04.10.2007

Software-Rev. 1.2 vom 15.04.2004

EVALUIERUNGSVERSION

IEP GmbH • Am Pferdemarkt 9c • D-30853 Langenhagen • Tel.: +49 (511) 70832-0 • Fax: +49 (511) 70832-99 • E-Mail: info@iep.de

Inhaltsverzeichnis

1 Allgemeine Hinweise....4
1.1 Handhabung 4
1.2 Installation 4
1.3 Erklärung 4

2 Übersicht....6
2.1 CD-Inhalt 7

3 Einführung in das Echtzeitbetriebssystem RTOS-UH....8
3.1 Eigenschaften von RTOS-UH 8
3.1.1 Unterstützte Prozessoren und Hardware....9
3.1.2 Verfügbare Systemkomponenten....9
3.1.3 Nutzerschnittstelle (Shell)....9
3.1.4 Ausführen von Anwenderprogrammen....9
3.1.5 Tasks....10
3.1.6 Systemgröße und Skalierbarkeit....10
3.1.7 I/O-System....11
3.1.8 Prozesssynchronisation und Interprozesskommunikation....11
3.1.9 Echtzeitdaten für Hochsprachanwenderprozesse....11
3.1.10 Technische Daten....12
3.2 Einschränkungen des Testsystems 12
3.3 Kurzbeschreibung der benötigten Utilitys 13
3.3.1 OCD-Debugger....13
3.3.2 RTERM32....13

4 Inbetriebnahme....14
4.1 Voraussetzungen 14
4.1.1 Installation des OCD-Debuggers....14
4.1.2 Installation von RTERM32....14
4.2 Anschluß des EVA-Boards 15
4.3 Brennen des RTOS-UH ins FLASH 19
4.4 EVA-Board mit externem FLASH (nur bei MPC555) 21
4.5 Serielle Schnittstellen 23
4.6 Ethernet 23

5 Programmierung....24
5.1 PEARL 90 24
5.2 CREST-C 27
5.3 CoDeSys 30

5.4 Debugger 35

6 Programmierung der CAN-Schnittstelle.....37

Revisionsliste:

Rev.	Datum	Na.	Änderung
1.0	07.03.2003	Kr	Erstellung
1.1	15.04.2004	Kr	Modifikation Int-Vectoren
1.2	07.03.2005	Ko	Beschreibung für EVA-Boards
1.3	22.02.2007	Ko	CAN-Beschreibung ergänzt
1.4	04.10.2007	Kr	MPC565 Besonderheiten

1 Allgemeine Hinweise

1.1 Handhabung

1. Lesen Sie bitte zuerst sorgfältig diese Dokumentation sowie die Dokumentation des Evalutionsboard für den phyCORE-MPC5*x*5 der Firma Phytec, bevor Sie die Hardware auspacken und einschalten. Sie sparen Zeit und vermeiden Probleme.
2. Beachten Sie bitte die Vorsichtsmaßnahmen bei der Handhabung elektrostatisch gefährdeter Hardware.
3. Wenn die Hardware Batterien enthält, legen Sie sie nicht auf elektrisch leitfähige Unterlagen. Die Batterie könnte kurzgeschlossen werden und Schäden verursachen.
4. Achten Sie bitte darauf, daß der spezifizierte Temperaturbereich nicht verlassen wird.

1.2 Installation

1. Überprüfen Sie mit Hilfe der Dokumentation, ob alle Jumper entsprechend Ihrer Anwendung gesetzt sind.
2. Schalten Sie die Spannungsversorgung ab, bevor Sie eine Verbindung herstellen.
3. Wenn Sie sicher sind, daß alle Verbindungen korrekt hergestellt sind, schalten Sie die Spannungsversorgung ein.

1.3 Erklärung

Wir behalten uns das Recht vor, Änderungen, die einer Verbesserung der Schaltung oder des Produktes dienen, ohne besondere Hinweise vorzunehmen. Trotz sorgfältiger Kontrolle kann für die Richtigkeit der hier gegebenen Daten, Schaltpläne, Programme und Beschreibungen keine Haftung übernommen werden. Die Eignung des Produktes für einen bestimmten Einsatzzweck wird nicht zugesichert.

BITTE BEACHTEN SIE:

ALLE AUF DIESER CD VORHANDENEN PROGRAMME DÜRFEN, SOWEIT NICHT AUSDRÜCKLICH ANDERS ERWÄHNT, NUR FÜR EVALUIERUNGSZWECKE EINGESETZT WERDEN! INSBESONDERS IST EINE KOMMERZIELLE NUTZUNG UNTERSAGT UND WIRD STRAFRECHTLICH VERFOLGT!

Wenn Sie Interesse am Einsatz von RTOS-UH haben, nehmen Sie bitte mit uns Kontakt auf.

http://www.iep.de

info@iep.de

2 Übersicht

Dieses Manual erklärt die Installation und den Einsatz des Echtzeitbetriebssystems RTOS-UH auf den PowerPC-Boards der Firma Phytec. Es werden die beiden Boards phyCORE-MPC555 und phyCORE-MPC565 unterstützt. Die Größe des Speichers wird automatisch erkannt, ebenso die bestückte Peripherie. Sie bekommen einen Eindruck von der Leistungsfähigkeit des Systems. Weiterhin erhalten Sie eine Einführung in die Programmierung des Systems sowie das Debuggen von Programmen unter RTOS-UH. Die Compiler laufen auf PCs mit Betriebssystem ab Windows98.

Die Beschreibung gilt für beide EVA-Boards. Die Testprogramme lassen auf dem phyCORE-MPC565 nur eine LED blinken, auf dem phyCORE-MPC555 zwei. Diese Beschreibung wurde mit dem phyCORE-MPC555 erstellt, wo signifikante Abweichungen zum phyCORE-MPC565 bestehen, wird darauf eingegangen.

Auf den folgenden Seiten erwartet Sie:

1) Eine "Einführung in das Echtzeitbetriebssystem RTOS-UH" mit einem Überblick über die Möglichkeiten der Programmierung. Weiterhin werden die für die Übertragung des Systems auf das phyCORE-Board nötigen Utilitys kurz vorgestellt.
2) Die "Inbetriebnahme" beschreibt, wie das System in den Flashes der phyCORE-Boards abgelegt werden kann. Die Grundlagen der Bedienung des RTOS-UH werden vermittelt.
3) Im Kapitel "Programmierung" können Sie lernen, wie ein Beispielprogramm zu übersetzen ist und auf dem System exekutiert werden kann. Der Einsatz des Debuggers wird erläutert.

Bitte beachten Sie, dass dieses Manual nur einen sehr kurz gehaltenen Überblick geben kann. Es wird in den entsprechenden Kapiteln immer wieder auf die ausführliche Original Dokumentation verwiesen, auf deren Studium Sie leider nicht verzichten können, wenn Sie die volle Leistungsfähigkeit des Systems ausschöpfen wollen.

Die Programme wurden so auf der CD abgelegt, dass nur das CoDeSys installiert werden muß, alle anderen Programme können Sie ohne Installation oder Pfad setzen ausprobieren. Dadurch bedingt ist eine gewisse Unübersichtlichkeit in den jeweiligen Ordnern, die sich leider nicht vermeiden läßt. Sie können natürlich jederzeit die Compiler und Utility-Programme in ein eigenes Verzeichnis legen, für das Sie dann einen Pfad eintragen müssen.

2.1 CD-Inhalt

Sie finden auf dieser CD:

MPC5x5Demo	
Dokumentation	
CoDeSys	CoDeSys Handbücher
CREST-C	Handbuch als PDF
CrestManualHtml	Handbuch in html
PEARL	PEARL90 Sprachreport in englisch, Buch über PEARL
VS	Einbindung in das Visual Studio
RTOS-UH	RTOS-UH Systemhandbuch, Einführung und FAQ
MPC555	
RTOSInstall	Macros und Batchdateien
System	RTOS-UH für phyCORE-MPC555
MPC565	
RTOSInstall	Macros und Batchdateien
System	RTOS-UH für phyCORE-MPC565
Programierung	
CoDeSys	setup.exe um CoDeSys zu installieren
CoDeSysINI	
Gateway	
MFCDLLs	
Targets	
demoppc	
CANDRV	
fpu	
IOLIB	
IEPdemo IO-Module	
CREST-C	Compiler/Linker/Make und Beispielprogramm
h	Header-Dateien
Debugger	Debugger
Beispiel	PEARL-Beispielprogramm
RTOS	RTOS-UH Laufzeitsystem für Debugger
PEARL90	PEARL-Compiler und Beispielprogramm

Weiterhin finden Sie im Ordner `Dokumentation` die Beschreibung der Netzwerkschnittstelle (Ordner `Netzwerk`), des CAN-Bus (`stcanV3.pdf`) und des Flash-Utilities (`rt-flash.pdf`).

3 Einführung in das Echtzeitbetriebssystem RTOS-UH

Komplexe Prozessoren wie der MPC555/565 erfordern bei der Programmierung einen hohen Einarbeitungsaufwand. Mehr als 1000 Register sind aufzusetzen, erst dann kann die eigentliche Anwendungssoftware entwickelt und laufen gelassen werden. Kommen dazu noch Netzwerkanbindung (Ethernet, TCP/IP) und Bussysteme (CANOpen, Profibus, InterBus-S usw.) lohnt sich eine Eigenentwicklung nur noch in den seltensten Fällen.

Mit dem Einsatz des Echtzeitbetriebssystems RTOS-UH umgehen Sie alle oben geschilderten Probleme. Sie bekommen ein echtzeitfähiges Multitaskingsystem und können sofort mit der Programmierung Ihrer Anwendung beginnen. Alle Schnittstellen des MPC555/565 werden unterstützt, Beispielprogramme erleichtern den Einstieg.

Wenn wir die Assemblerprogrammierung außen vor lassen, stehen Ihnen 3 verschiedene Programmiersprachen/Systeme zur Verfügung:

1) ANSI-C

   Die CREST-C Programmierumgebung stellt neben dem C-Compiler, Linker, Librarys auch einen Debugger zur Verfügung. Der Compiler enthält Erweiterungen, um das Multitasking des Betriebssystems zu unterstützen. Auch hardwarenahe Programmierung von z.B. Interrupts wird unterstützt.

2) PEARL90

   Die Programmiersprache PEARL wurde gegen Ende der 70er Jahre in Deutschland von Ingenieuren für Ingenieure entwickelt. Multitasking und Echtzeitbehandlung sind integraler Bestandteil der Sprache. Ein klares Modulkonzept ermöglicht die übersichtliche Programmierung auch größerer Programmpakete.

3) CoDeSys

   Das CoDeSys-Programmiersystem ist eine integrierte Entwicklungsumgebung nach der IEC 61131-3. Alle 5 Programmiersprachen des Standards werden unterstützt. Damit kann das System wie eine SPS programmiert werden. Eine integrierte Visualisierung ermöglicht die einfache Erstellung von HMI-Oberflächen. Die Offlinesimulation erleichtert ebenso wie die umfangreichen Debuggingmöglichkeiten die Programmerstellung.

Welche Programmiersprache Sie auch einsetzen wollen, zuerst einmal müssen Sie das Betriebssystem auf den MPC5x5 bringen. Wie Sie dies machen, wird Kapitel 4 ff. beschrieben. Das Betriebssystem wird ins FLASH gebrannt und steht dann permanent zur Verfügung. Ihr Anwendungsprogramm kann dann einfach ins RAM geladen und laufen gelassen werden, RTOS-UH stellt alle dafür nötigen Mechanismen zur Verfügung.

3.1 Eigenschaften von RTOS-UH

RTOS-UH wurde am Institut für Regelungstechnik der Universität Hannover speziell für die Belange der Automatisierungs- und Regelungstechnik entworfen. Ende der 70er Jahre entstand der Wunsch, Regelungsaufgaben mit Hilfe von Kleinstrechnern zu lösen. Geeignete Betriebssysteme und Compiler waren nicht vorhanden oder zu teuer.

So wurde durch Ingenieure der Regelungstechnik mit den hervorragenden intellektuellen Möglichkeiten an einer Universität ein erstes Konzept erarbeitet und realisiert. RTOS-UH wurde kontinuierlich weiterentwickelt und strukturell verbessert, um den Anwendungswünschen optimal entsprechen zu können.

3.1.1 Unterstützte Prozessoren und Hardware

RTOS-UH unterstützt Prozessoren der Firma Freescale (ex Motorola). Es wird die komplette MC68xxx-Familie unterstützt. Weiterhin werden die PowerPC-Familien von Freescale und AMCC (ex IBM) unterstützt.

Treiber für gängige I/O-, Netzwerk- und Prozessdateninterfacebausteine sind vorhanden. Eine modulare Scheibenstruktur ermöglicht die Einbindung eigener Treiber für Sonderhardware.

3.1.2 Verfügbare Systemkomponenten

- Shell
- Shellinterpreter gemäß Bourne-Syntax
- Festplatten- und Floppyverwaltung
- Streamer- und CD-ROM-Unterstützung
- Netzwerkunterstützung und verteiltes Filesystem
- Multiwindow-Editor
- Remote Login und verteiltes Filesystem
- Telnet-, ftp- und Mail-Programme sowie Telnet- und ftp-Server, SAMBA-Server
- Feldbustreiber z.B. für CAN, InterBus-S, Profibus-DP
- Verschiedene Compiler und Assembler

3.1.3 Nutzerschnittstelle (Shell)

- Übliche Philosophie mit Stdin, Stdout, Stderror.
- Die Shell lässt sich um residente und transiente, in Universalassembler oder Hochsprache codierte, Befehle beliebig erweitern.
- Tasks namentlich überprüfbar und manipulierbar.
- Adress- und Zeilenstop taskorientiert.
- Speicher- und Tasksituation direkt einsehbar, kein Hardwaredebugger nötig.

Alle Übersetzer, der Linker, die Editoren, und der Shellspracheninterpreter sind wiedereintrittsfest, d.h. der Code wird für beliebig viele Prozesse nur einmal im RAM oder ROM benötigt.

3.1.4 Ausführen von Anwenderprogrammen

- RTOS-UH Programme lassen sich aus dem EPROM/FLASH exekutieren oder von Festplatte bzw. Floppy laden.

- Das Laden über das Netzwerk oder eine serielle Schnittstelle ist möglich.
- Das Laden aus RAM-Disks, deren Speicher batteriegepuffert ist, ermöglicht sowohl leichte Änderungen an der Anwendersoftware als auch ein schnelles Anfahren bei SBCs mit Netzwerkanschluss.
- Bei Systemen mit mehreren Prozessorkarten ermöglicht eine gemeinsam genutzte RAM-Disk von Master- und Slave-Prozessor das Nachladen auf die Slavekarte.
- Im laufenden Betrieb lassen sich beliebig viele Anwendungen nachladen, sofern der Speicherplatz es erlaubt.
- RTOS-UH lässt sich einfach um beliebige Anwenderkomponenten erweitern (s.u.).

3.1.5 Tasks

RTOS-UH ist völlig frei von "Prozesstabellen" oder ähnlichen statischen Limitierungen.

- Die Anzahl der Tasks ist bei RTOS-UH nur durch den vorhandenen Speicher begrenzt.
- Die Verwaltung einer Task ohne FPU-Benutzung benötigt ca. 220 Byte auf einem 68xxx-Rechner, 300 Byte bei einem Power-PC.
- Die Verwaltung einer Task mit FPU-Benutzung benötigt ca. 500 Byte auf einem 68xxx-Rechner, 750 Byte bei einem Power-PC.
- RTOS-UH kann Tasks sowohl laden als auch dynamisch beim Systemanlauf oder bei der Ausführung von Bedienbefehlen generieren.
- Tasks, die sich beendet haben, müssen nicht aus dem Speicher entfernt werden, was einen sehr schnellen Wiederanlauf ermöglicht.
- Wird eine Task erneut aktiviert, während sie noch läuft, puffert RTOS-UH die Aktivierung. Dieses fehlertolerante Verhalten führt selbst bei hoher Systemauslastung zu korrekter, wenn auch verzögerter Abarbeitung.

3.1.6 Systemgröße und Skalierbarkeit

RTOS-UH ist durch seinen modularen Aufbau sehr gut in seiner Größe skalierbar.

- Das kleinste Laufsystem mit allen Systemdiensten und einem Treiber für eine serielle Schnittstelle benötigt bei einem 68xxx Prozessor 23 KByte und bei einem PowerPC 52 KByte.
- Das komplette Laufzeitsystem mit Shell, Filesystem und Ethernet erfordert ca. 512 KByte (68xxx) / 1024 KByte (PowerPC) Speicherplatz.

RTOS-UH ist komplett in portablem Assembler codiert. Dieses wirkt sich neben der Kompaktheit auch in der Geschwindigkeit aus, weil

- Interrupt-Serviceroutinen nur die Register retten, die sie auch wirklich benötigen,
- die Register viel besser als bei einer Hochsprachrealisierung ausgenutzt werden,
- Tasks des Betriebssystems wichtige Zeiger auch über Prozeduren hinweg in einem Register halten, was zu einer sehr schnellen Adressierung aller Objekte führt.

3.1.7 I/O-System

Das I/O-System gilt allgemein als wunder Punkt von Echtzeitbetriebssystemen. RTOS-UH verwendet schon seit seiner Anfangszeit hier die einzige echtzeitkonforme Methode der I/O-Dämonen:

- Ein I/O-Dämon ist eine Task, die jeweils eine Rechnerschnittstelle, wie z.B. Festplatte, Floppy, Netzwerk, serielle oder parallele Schnittstelle, betreut. Dadurch lässt sich das gesamte I/O auf Nutzerebene abarbeiten. Durch niederpriorisierte Tasks angestoßenes I/O beeinträchtigt daher die Echtzeitreaktivität nur minimal.
- Tasks, die Daten einlesen oder ausgeben wollen, setzen den entsprechenden Auftrag an RTOS-UH ab, welches den Auftrag prioritätsgerecht in die Warteschlange des zugehörigen I/O-Dämons einreiht.
- Eine variable Priorität der I/O-Dämonen verhindert Prioritätsinversionsprobleme.

Das gesamte I/O kann auf Slaveprozessoren ausgelagert werden.

3.1.8 Prozesssynchronisation und Interprozesskommunikation

- Innerhalb eines Moduls können Tasks auf gemeinsame Variablen zugreifen.
- Für Anwendungsprogramme, die sich aus mehreren Modulen zusammensetzen, ist die Deklaration globaler Variablen möglich, auf die alle Tasks des Programmpaketes zugreifen können.
- Benambare Pipes in beliebiger Zahl.
- Zählende und binäre Semaphore in beliebiger Zahl.
- Extrem schnelle Boltvariable in beliebiger Zahl.
- Message passing.

Das leistungsfähige Werkzeug der Boltvariablen ist bisher vermutlich in kaum einem anderen Echtzeitbetriebssystemkern realisiert (in RTOS seit 1992).

3.1.9 Echtzeitdaten für Hochsprachanwenderprozesse

- Echte Preemption durch prioritätsgerechten Abbruch auch innerhalb von Systemfunktionen (Traps).
- Der interne Takt für die zeitliche Einplanung von Tasks hat eine Auflösung von 50 µs. Für langsamere Prozessoren, die durch einen alle 50 µs auftretenden Timer-Interrupt zu sehr belastet werden, ist die Auflösung in Vielfachen von 50 µs möglich.
- Gemessene Zeit vom äußeren Ereignis bis zum Start der Anwendertask beim 50 MHz 68060: 8.0 µs typ., 20.0 µs max.
- Gemessene Zeit vom äußeren Ereignis bis zum Start der Anwendertask beim 100 MHz Power PC 604: 5.0 µs typ., 10.2 µs max.
- Das RTOS-UH-eigene Floating-Point-Package zur Berechnung trigonometrischer und Exponentialfunktionen läuft im User-Mode des Prozessors und behindert dadurch die Echtzeitreaktivität nicht.

Bei einem Systemvergleich beachte man bitte: Wie bei anderen Systemen, so kann man auch bei RTOS-UH natürlich eigene Interrupthandler hinzufügen, die die gesamte Echtzeitdatenverarbeitung durchführen. Damit erreicht man sehr viel kürzere Reaktionszeiten. Leider geben einige Systemanbieter derartige Daten als "typische Reaktionswerte" an. Obige Werte beschreiben die für den seriösen, sicherheitssensiblen Nutzer wichtigen Zeiten, bei denen die Interrupthandler lediglich Nutzertasks fortsetzen oder aktivieren. Diese führen dann die eigentliche Echtzeitdatenverarbeitung aus.

3.1.10 Technische Daten

Allgemeine Daten	68xxx	PowerPC
Kernel-ROM (minimum)	23 KByte	52 KByte
Kernel-ROM (maximum)	318 KByte	800 KByte
Kernel-RAM (minimum)	5 KByte	24 KByte
Kernel-RAM (maximum)	430 KByte	450 KByte
Minimaler Platzbedarf pro Task	218 Byte	300 Byte
Minimaler Platzbedarf pro Queue	90 Byte	90 Byte
Anzahl der Prioritätenstufen	32768	
Maximal erlaubte Anzahl von Tasks	Nur durch Speicher begrenzt	
Zeitliche Auflösung der Planungsuhr	50 µs	
Prozessorabhängige Daten	68060/50MHz	604/100MHz
Durchschnittliche Reaktionszeit	8.0 µs	5.0 µs
Maximale Reaktionszeit	20.0 µs	10.2 µs
Durchschnittliche Taskwechselzeit	5.5 µs	2.7 µs

3.2 Einschränkungen des Testsystems

Da es sich bei diesem System um ein Testsystem zur Evaluierung handelt, gibt es ein paar Einschränkungen in der Funktionalität:

1) Das RTOS-UH System läuft nur zeitlich beschränkt. Nach ca. 2 Stunden wird das System angehalten. Damit das System neu startet, muß die Spannungsversorgung ab und wieder angeschaltet werden.
2) Der mit den beiliegenden Entwicklungsumgebungen erzeugte Code läuft nur auf dem Evaluierungssystem.

Bei Problemen nehmen Sie bitte Kontakt mit uns auf unter info@iep.de.

3.3 Kurzbeschreibung der benötigten Utilitys

3.3.1 OCD-Debugger

Der OCD-Debugger ermöglicht den Download des RTOS-UH-Betriebssystems auf das phyCORE-MPC5*x*5. Nach dem Download wird das Betriebssystem automatisch gestartet. Ab dann ist der OCD-Debugger nicht mehr nötig.

3.3.2 RTERM32

Das Terminalprogramm RTERM32 stellt über die serielle Schnittstelle die Verbindung zum RTOS-UH her. Sie können auch jedes andere Terminalprogramm nehmen, RTERM32 stellt aber zusätzliche Funktionen bei der Arbeit mit RTOS-UH zur Verfügung. Sie können z.B. mit einem Mausklick Dateien senden oder Programme laden.

4 Inbetriebnahme

4.1 Voraussetzungen

Es werden folgende Systemvoraussetzungen für die Evaluierungsinstallation des RTOS-UH benötigt:

Bezeichnung	Hersteller	URL
EVA-Board	PHYTEC GmbH	http://www.phytec.de
PhyCORE-MPC555 PhyCORE-MPC565	PHYTEC GmbH **mit externem FLASH**	
OCD-Debugger	MacRaigor Systems	http://www.macraigor.com
RTERM32	IEP GmbH	http://www.iep.de/Downloads/programme/rtw.zip
RS232 Kabel 9pol	PHYTEC GmbH	
Parallelkabel	PHYTEC GmbH	
CD RTOS-5x5	IEP GmbH	http://www.iep.de
PC ab WIN98		

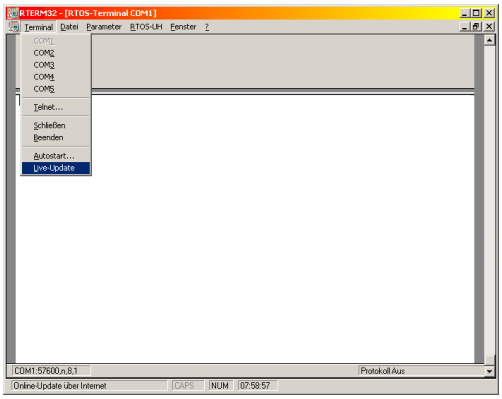
Wie Sie aus der Tabelle entnehmen können, werden nur EVA-Boards mit externem Flash unterstützt. RTOS-UH ist auch für Boards verfügbar, die kein externes Flash besitzen, dann ist allerdings eine "abgespeckte" Version nötig. Bei Bedarf sprechen Sie uns bitte an.

4.1.1 Installation des OCD-Debuggers

Der OCD-Debugger kann von der CD installiert werden. Die aktuellste Version gibt es unter http://www.macraigor.com. Auf der CD finden Sie den Ordner `MacRaigor`. In diesem finden Sie das Programm `ocd_dbgr.exe`. Starten Sie das Programm und folgen Sie der Installationsanweisung.

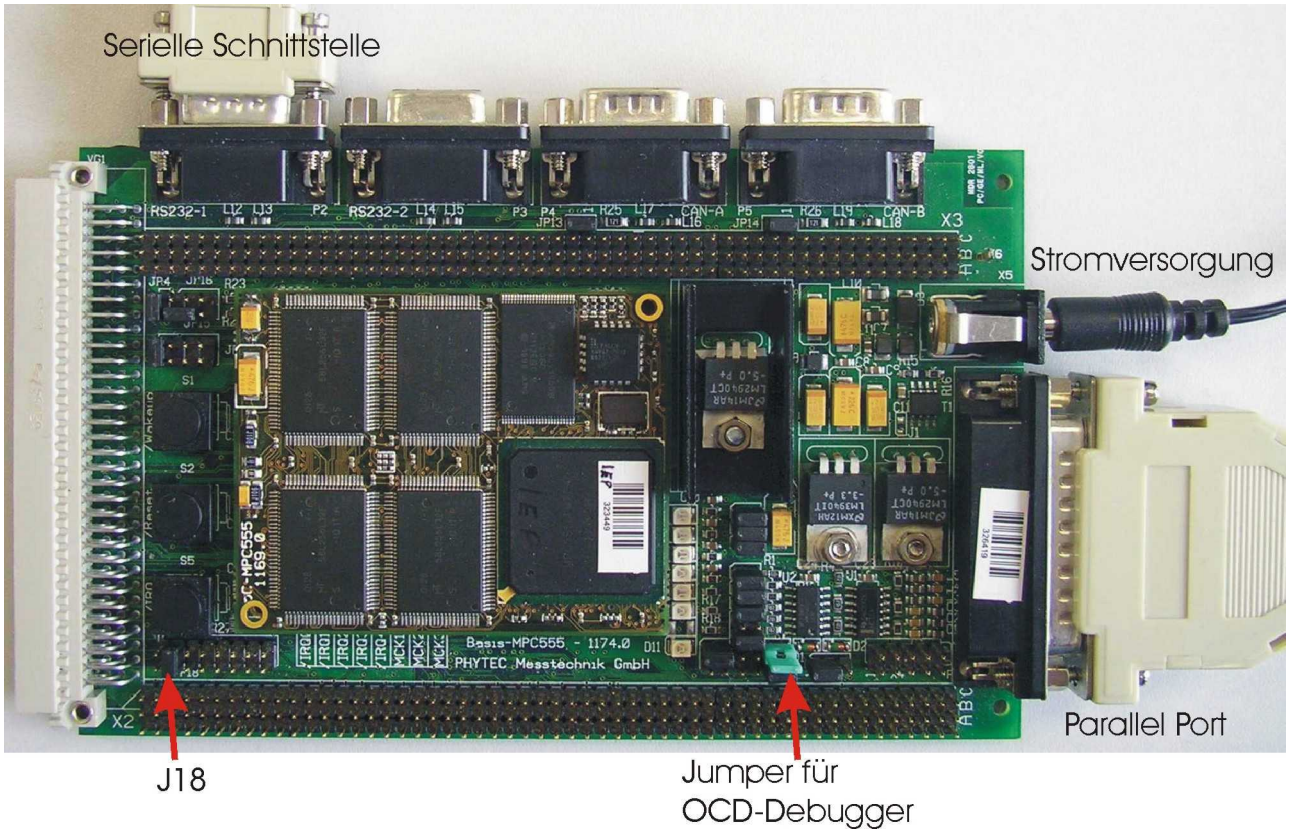
4.1.2 Installation von RTERM32

Eine Installation von RTERM32 ist nicht nötig. Es reicht, wenn Sie das Programm starten. Zur Arbeitserleichterung können Sie sich ein Icon in der Schnellstartleiste ablegen. Nutzen Sie die automatische Update-Funktion, um das Programm auf den aktuellen Stand zu bringen:

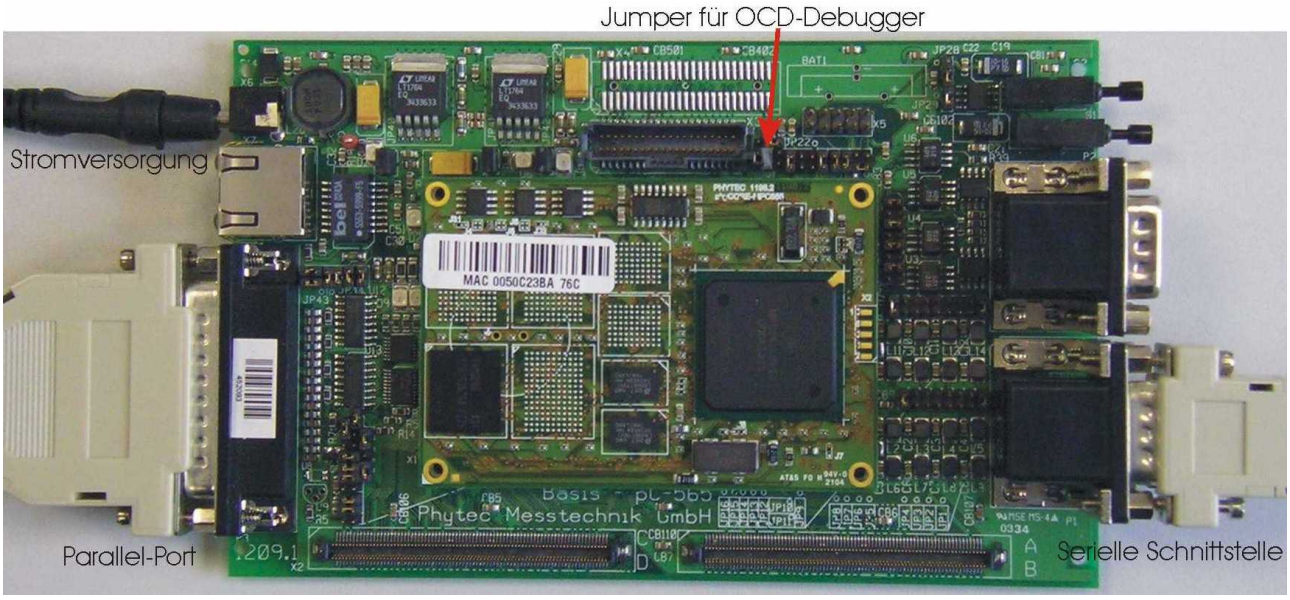


4.2 Anschluß des EVA-Boards

Bitte schließen Sie das EVA-Board folgendermaßen an:

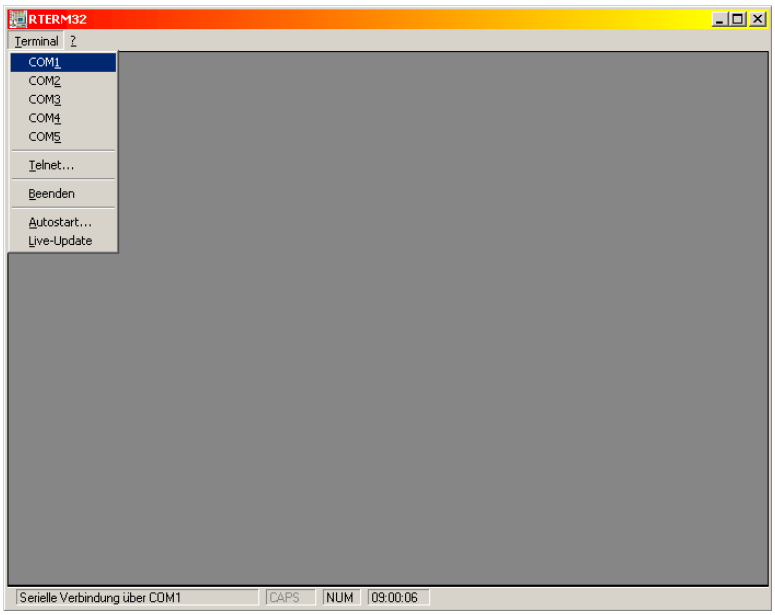


oder

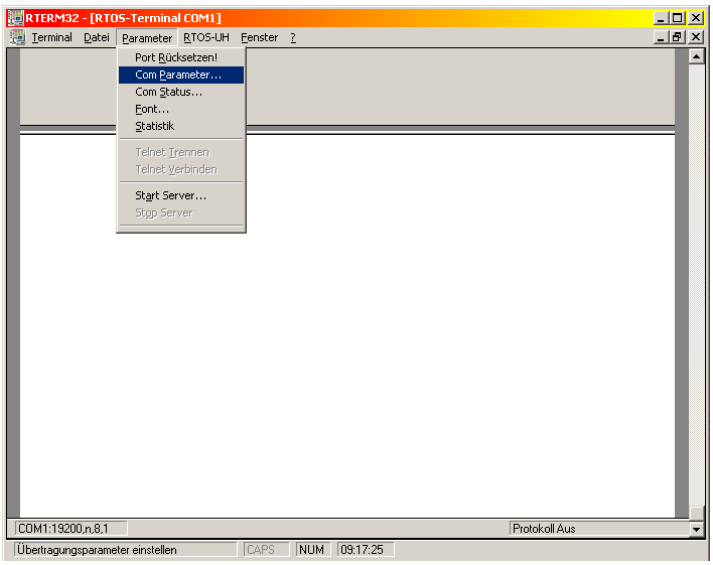


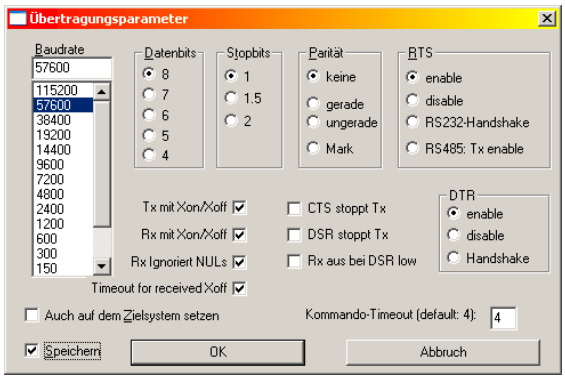
Verbinden Sie die serielle Schnittstelle RS232-1 (oben beim phyCORE-MPC565) mit einem freien COM-Port Ihres PCs. Starten Sie RTERM32. Öffnen Sie das COM-Port, an

dem die serielle Schnittstelle des EVA-Boards angeschlossen ist, in diesem Beispiel ist es COM1:

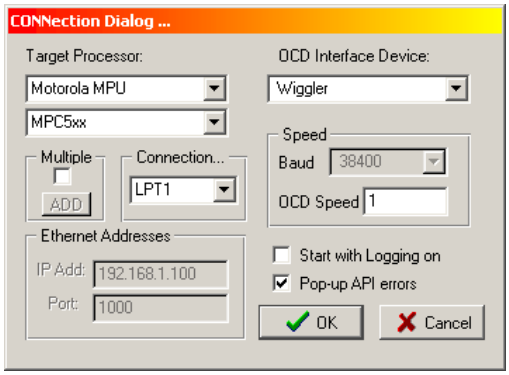


Stellen Sie die Baudrate auf 57000 Baud entsprechend der folgenden Bilder:

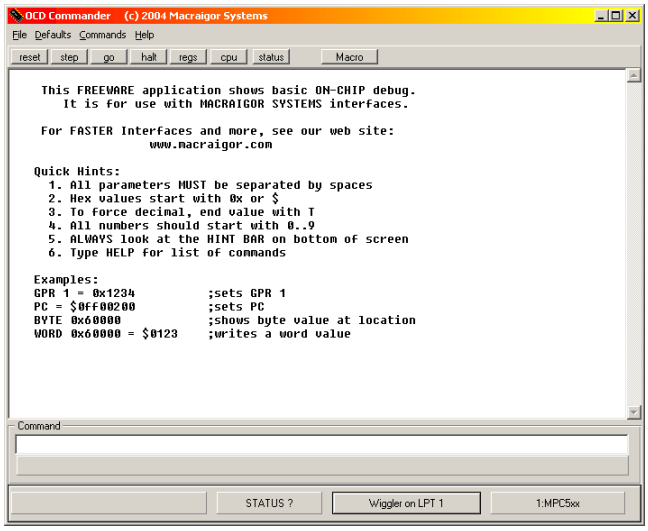




Nun starten Sie den OCD-Debugger. Bitte beachten Sie, dass der "Jumper für OCD-Debugger" gesteckt ist, sonst kann der Debugger keine Verbindung zum EVA-Board herstellen! Es erscheint die folgende Startmeldung, nehmen Sie die dargestellten Einstellungen vor:

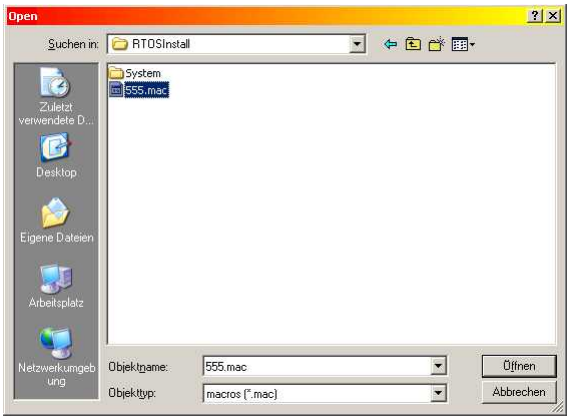


So sollte der Startbildschirm aussehen:

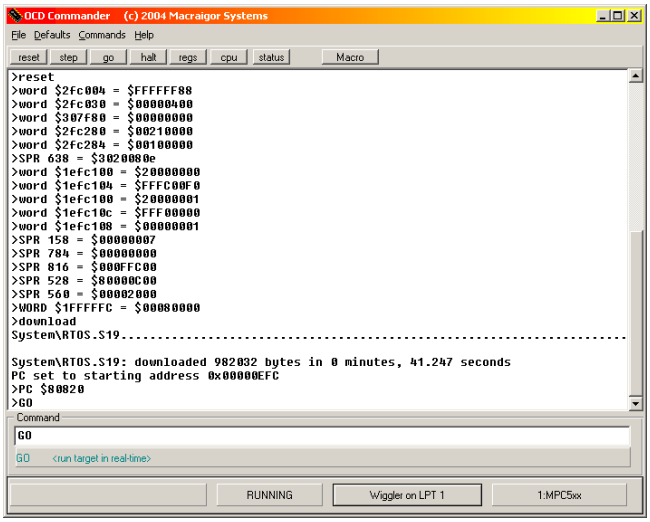


Nach Druck auf den

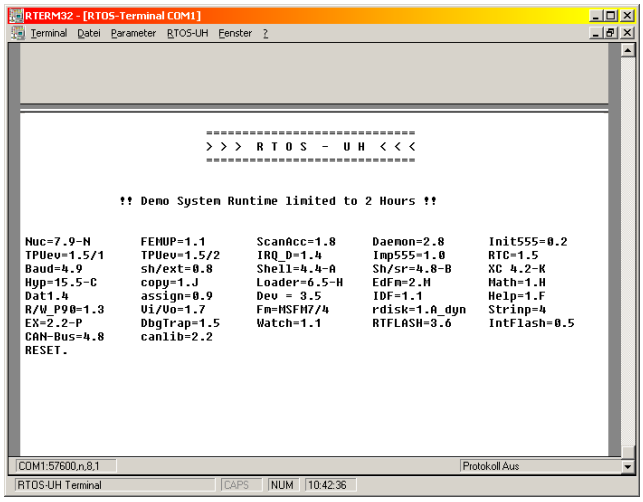
–Button öffnet sich folgende Box:



Wechseln Sie auf die CD, dort in den Ordner `MPC555` bzw. `MPC565` und weiter nach `RTOSInstall`. Öffnen Sie das Macro `555.mac` bzw. `565.mac`. Damit werden ein paar Register aufgesetzt und das RTOS-UH System auf den MPC555/565 übertragen. Sie erkennen den Fortschritt an den Punkten.

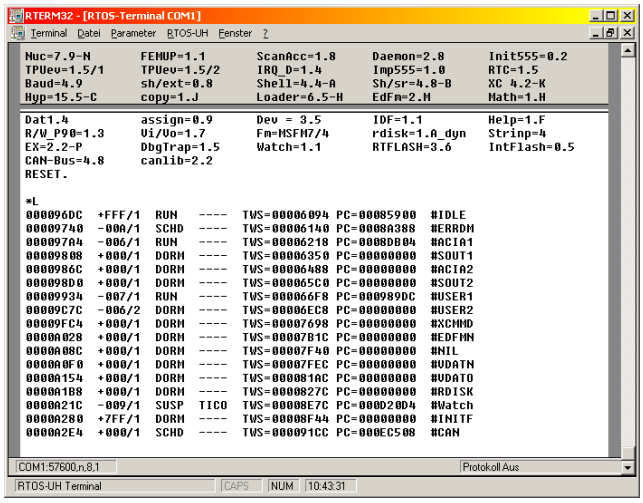


Wenn der GO erscheint, sollte sich das RTOS-UH auf dem Terminal melden:



Wenn Sie diese bzw. eine ähnliche Überschriftsmeldung sehen, läuft RTOS-UH.

Sie können jetzt Ihr erstes Kommando eingeben, dazu müssen Sie dem System einen Prompt entlocken, aus Sicht des Echtzeitbetriebssystems sind Sie als Anwender auch nur ein (störendes) externes Ereignis. Um die Aufmerksamkeit des Systems auf sich zu ziehen, geben Sie bitte `Cntr/Strg A` ein, d.h. Sie drücken gleichzeitig die `Strg`- und die A-Taste. Haben Sie alles richtig gemacht, so sollten Sie als Reaktion vom System ein '*' als Prompt bekommen. Nun können Sie den gewünschten Befehl eingeben, wenn Sie noch keinen kennen, könnten Sie z.B. ein 'L' eingeben. Sie sollten dann die folgende Ausgabe erhalten:



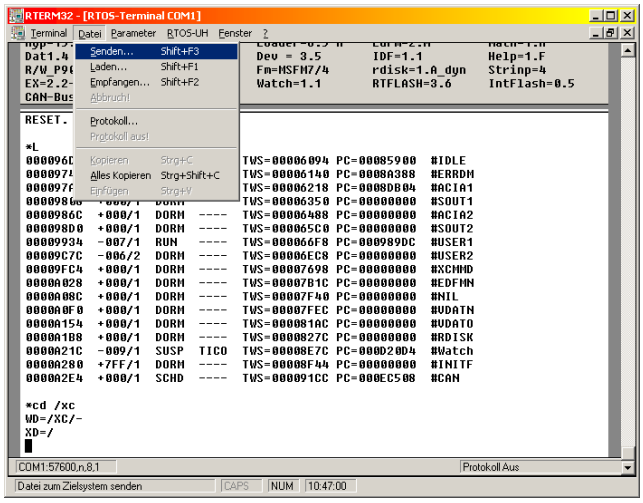
Die Liste, die der `L`-Befehl ausgibt, zeigt Ihnen alle im Betriebssystem vorhandenen Tasks an, eine ausführliche Erklärung zu allen Befehlen finden Sie im RTOS-UH Systemhandbuch. An dieser Stelle reicht es erst einmal, dass das RTOS-System läuft und Sie es bedienen können, jetzt kommt der nächste Schritt.

4.3 Brennen des RTOS-UH ins FLASH

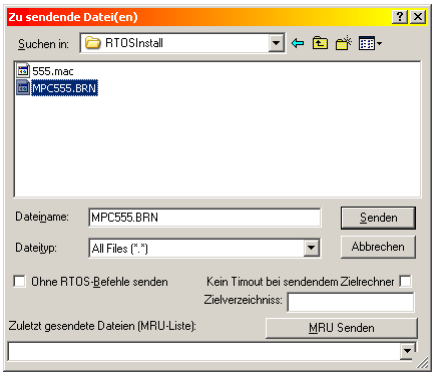
Damit das Betriebssystem dauerhaft läuft, muß es im internen Flash des MPC5*x*5 abgelegt werden. Dazu müssen Sie wieder ein paar Befehle eingeben, beginnen Sie – nach der Eingabe von `Strg A` – mit:

`*CD /ED` *Der '*' ist der Prompt*

Damit haben Sie das Working-Directory auf ein Filesystem gestellt, das im RAM des Rechners gehalten wird. Nun können Sie eine Datei auf den MPC5X5 übertragen:



In der Dateiauswahlbox wählen Sie die Datei `RTOSInstall\MPC555.BRN` bzw. `RTOSInstall\MPC565.BRN`:

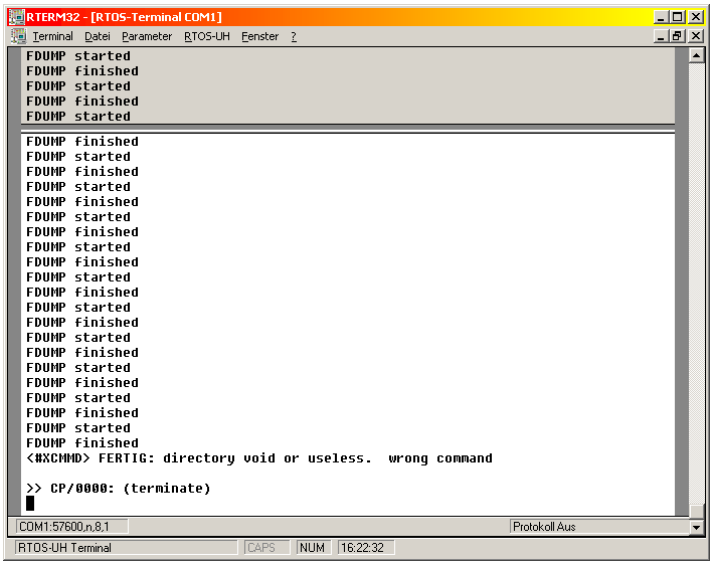


Während die Datei übertragen wird, sehen Sie unten im RTERM32 den Fortschrittsbalken. Hier können Sie den Vorteil beim Einsatz des RTERM32 sehen, die Befehle die auf der RTOS-UH Seite nötig sind, um die Datei zu übertragen, werden automatisch in der richtigen Reihenfolge mit abgesetzt.

Nachdem die Übertragung abgeschlossen ist, geben Sie bitte den nächsten Befehl ein:

`*cp MPC555.BRN>/xc` bzw. `cp MPC565.BRN>/xc`

An das `Strg A` vor der Eingabe des Befehls haben Sie sich sicher schon gewöhnt? Nun wird das interne Flash gebrannt, Sie können sich erst einmal entspannt zurücklehnen und sich erklären lassen, was der obige Befehl eigentlich bewirkt. Mit der Eingabe von `cp` wird ein COPY-Befehl gestartet. Der Befehl erwartet die Eingabe der Quelle, in unserem Fall ist das die Datei `/ED/MPC5`*x*`5.BRN`. Diese Datei ist eine Batchdatei, d.h. in ihr stehen Befehle an das Betriebssystem. Sie hätten diese Befehle auch alle von Hand eingeben können, so geht es aber schneller und fehlerfreier. Die Befehle werden zu der Datenstation `/xc` kopiert, dies ist genau so ein Kommandoprozessor, wie Sie ihn mit `Strg A` aufwecken. Inzwischen sollten schon einige Ausgaben auf dem Bildschirm erschienen sein, das Flash ist gebrannt, wenn Sie folgende Ausgaben im Terminalfenster sehen:



Nun ist das Basissystem RTOS-UH im MPC5*x*5 abgelegt. Um zu testen, ob alles geklappt hat, ziehen Sie bitte die Stromversorgung, das Parallelport-Kabel und den "*Jumper für OCD-Debugger*" ab. Wenn Sie nun die Stromversorgung wieder anschließen, sollte sich

das RTOS-UH wieder auf der seriellen Schnittstelle melden. Es leuchten nur die oberen 3 grünen LED's auf dem EVA-Board. Leuchtet auch die rote LED L11, so haben Sie den Jumper nicht entfernt. Meldet sich das RTOS-UH immer noch nicht, wiederholen Sie bitte alle Schritte sorgfältig.

4.4 EVA-Board mit externem FLASH (nur bei MPC555)

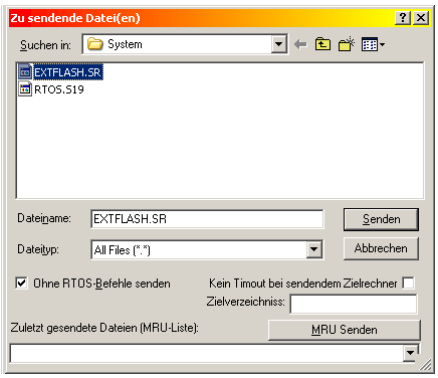
! *Auf dem Phytec MPC565 Board ist es leider nicht möglich, das externe Flash als Programmspeicher direkt zu nutzen, da Phytec sein CPLD so programmiert hat, das keine Codeexekution aus dem Flash möglich ist. Um dieses zu ändern wäre es notwendig das CPLD erst umzuprogrammieren*.

Haben Sie ein Board mit zusätzlichem externen FLASH, so können Sie das Basissystem ergänzen. Es stehen Ihnen dann das Laufzeitsystem für die IEC 61131-3 Programmierumgebung CoDeSys und die Treiber für Netzwerkunterstützung (TCP/IP) zur Verfügung.

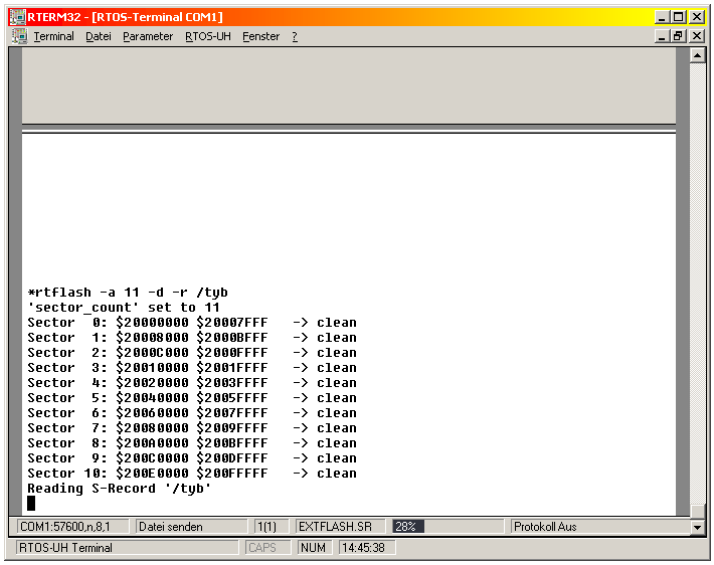
Dazu rufen Sie das Programm RTFLASH mit folgenden Parametern auf:

```
rtflash -a 11 -d -r /tyb
```

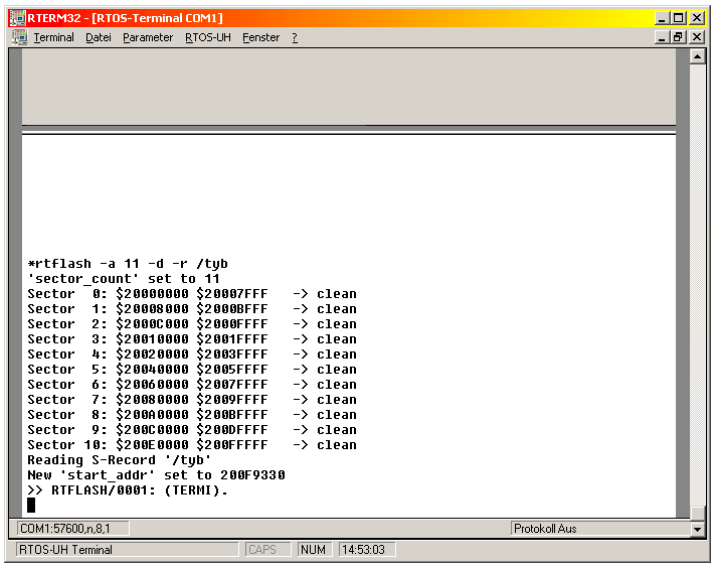
Dieser Aufruf löscht die ersten 11 Blöcke im Flash und wartet dann auf Daten von der Schnittstelle. Sie müssen dann die Datei `EXTFLASH.SR` senden, wichtig ist, dass die Checkbox "*Ohne RTOS-Befehle senden*" an ist:



Nun ist wieder etwas Zeit um z.B. einen Kaffee oder Tee zu trinken, die Übertragung dauert ein Weilchen:



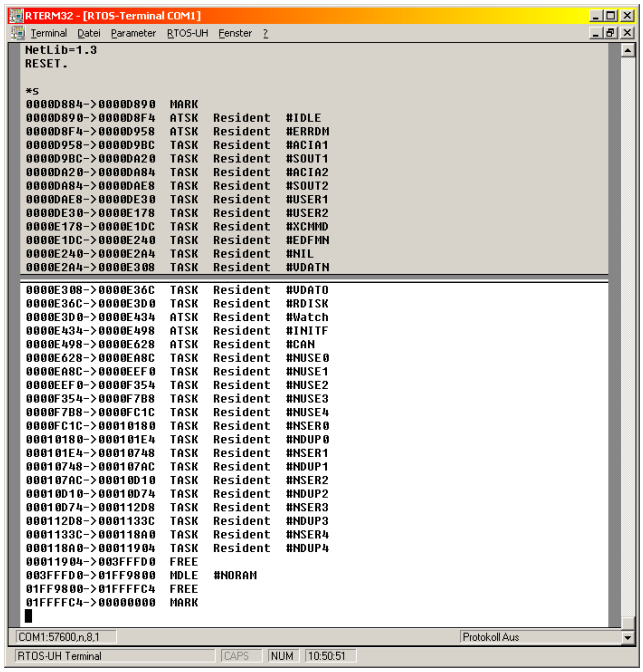
Sie können natürlich auch den Fortschrittsbalken beobachten und warten, dass das Flash fertig programmiert ist. Wenn der Fortschrittsbalken verschwunden ist, müssen Sie dem RTFLASH noch mitteilen, dass Sie nicht gedenken weitere S-Records zu senden und er seine Arbeit beenden kann. Dies können Sie mit einen `Strg D` erreichen, Sie wissen schon, beide Tasten gleichzeitig drücken. Daraufhin sollten Sie die Abschlußmeldung des RTFLASH bekommen:



Danach kann ein RESET ausgelöst werden und die Installation des RTOS-UH ist komplett. Mit dem Kommando

`*S`

können Sie sich einen Überblick über die Speicherbelegung verschaffen:



Nun ist es an der Zeit sich näher mit dem System zu beschäftigen. Dazu können Sie den Über- und Einblick Rtstart.pdf lesen. Sie können sich natürlich auch eine Programmiersprache aussuchen und die Beispiele ausprobieren.

4.5 Serielle Schnittstellen

Es stehen 2 serielle Schnittstellen mit 9 poligen SubD-Buchsen zur Verfügung.

Unter RTOS-UH ist die mit RS232-1 bezeichnete serielle Schnittstelle das /A1/-Port. Die Standardbaudrate ist 57600 Baud, 8 Datenbits, 1 Stopbit, ohne Parität.

Auf dem Stecker RS232-2 ist das /A2/-Port verfügbar. Die Parameter der Schnittstelle entsprechen denen der Schnittstelle /A1.

4.6 Ethernet

Wenn auf Ihrer Hardware eine Ethernetschnittstelle ist, können Sie darüber Kontakt zum EVA-Board aufnehmen. Lesen Sie strtlan.pdf, um mehr über das Netzwerkpaket unter RTOS-UH zu erfahren. Um das Netzwerk zu starten, geben Sie folgende Kommandos ein:

```
*netio.NETIO -I=192.168.1.123
*TEL_SERVER
```

Die Ethernetadresse muß von Ihnen natürlich entsprechend Ihrem Netz angepasst werden. Sie können dann z.B. mit dem RTERM32 eine Telnet-Verbindung aufbauen.

5 Programmierung

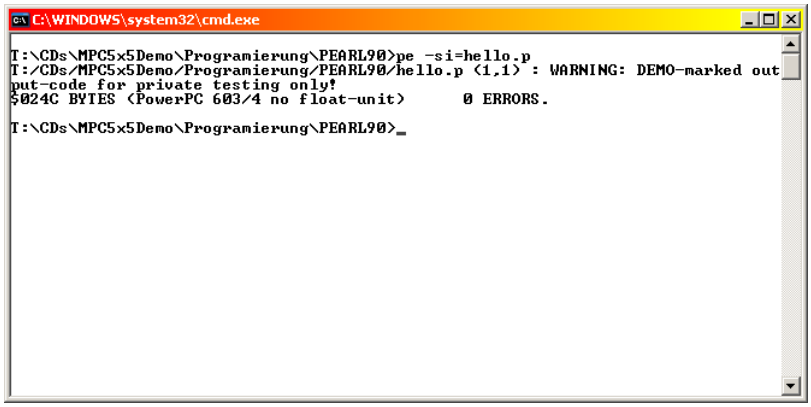
Sie finden in diesem Kapitel jeweils ein Beispiel für die verschiedenen Programmierumgebungen.

5.1 PEARL 90

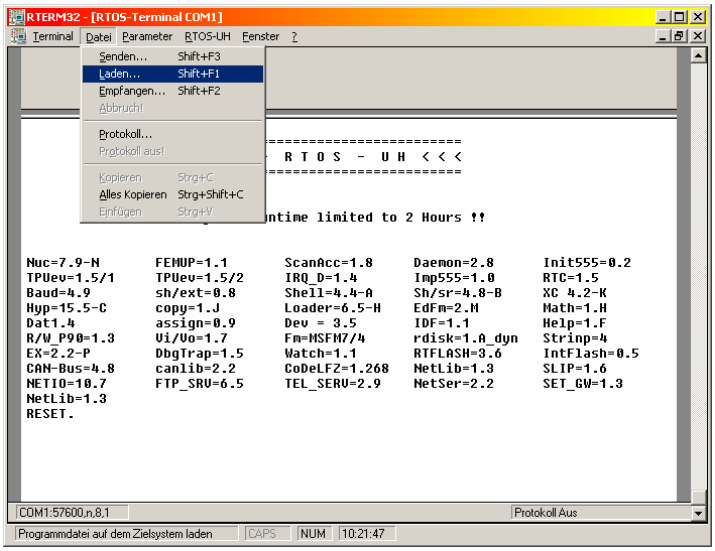
Sie haben mehrere Möglichkeiten, den PEARL-Compiler zu nutzen. Die komfortabelste Variante ist die Integration in das Visual Studio. Lesen Sie dazu bitte das Dokument PEARL\VS\VS_Einbindung.pdf. Hier ein Bild, wie der Quelltext des Programms `hello.p` aussieht:

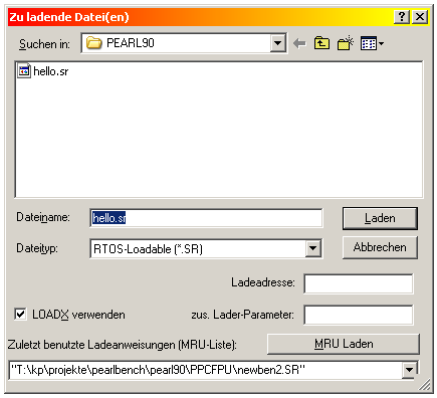
```
MODE=DEBUG ;
/*+M*/
MODULE M; /* Hier steht der Name des Moduls */

SYSTEM ; /* Im Systemteil werden die Verbindungen zur Hardware hergestellt */

   TY: A1. ; /* TY ist die serielle Schnittstelle A1 */

   /* MPIO Register, Zugriff über BU-Dations */
   MPIOSMDR:  BU(1F06100,2)<->; /* MPIOSM Data Register           */
   MPIOSMDDR: BU(1F06102,2)<->; /* MPIOSM Data Direction Register */

PROBLEM ; /* Der Problemteil ist hardwareunabhängig */

   SPC TY DATION INOUT ALPHIC CONTROL( ALL ) ;      /* Hier wird die serielle Schnittstelle spezifiziert */
   SPC ( MPIOSMDR, MPIOSMDDR ) DATION INOUT BASIC ; /* Zugriff direkt                                    */


Hello: TASK ; /* Dies ist eine Task, ein eigenständig lauffähiges Programm */

   PUT 'Hello World' TO TY BY A, SKIP ; /* Ausgabe eines Textes */

END ; /* TASK Hello */


Blink: TASK PRIO 10 ; /* Diese Task hat eine höhere Priorität */

   SEND '0003'B4 TO MPIOSMDDR ; /* Die unteren beiden Ausgänge als Ausgang schalten */
   SEND '0003'B4 TO MPIOSMDR  ; /* Beide LED's ausschalten                          */

   REPEAT /* Endlosschleife */
      SEND '0001'B4 TO MPIOSMDR ; /* rote LED an */
      AFTER 0.5 SEC RESUME ;    /* Für eine halbe Sekunde schlafen legen */

      SEND '0002'B4 TO MPIOSMDR ; /* grüne LED an */
      AFTER 1 SEC RESUME ;      /* Für eine Sekunde schlafen legen */
   END ;

END ; /* TASK Blink */


MODEND ;
```

Die Beschreibung im folgenden geht davon aus, dass Sie kein Visual Studio haben und den Systemeditor nutzen. Um nun mit dem Programmieren anfangen zu können, nehmen Sie den gesamten Ordner `PEARL90` und kopieren ihn auf Ihre Festplatte. Sie brauchen nichts weiter zu installieren, solange Sie nur in diesem Ordner arbeiten. Öffnen Sie eine Eingabeaufforderung (DOS-Box) in diesem Ordner und rufen Sie den PEARL-Compiler mit `pe -si=hello.p` auf:



Das Programm wurde mit 0 Fehlern übersetzt, Sie finden die Liste `hello.lst` und den SRecord `hello.sr`. Laden Sie den SRecord:





Nach dem Laden setzen Sie ein 'S'-Kommando ab, Sie sehen, dass das Modul `M` mit den beiden Task `Hello` und `Blink` im Speicher liegt. Mit dem Kommando `LU` können Sie den Zustand der beiden Task kontrollieren, Sie sehen vorne die Taskkopfadresse, gefolgt von der Priorität (`Hello` hat die Priorität 48 (Hex 30), `Blink` die Priorität 10 (Hex A). Der Zustand beider Task ist `DORM`, d.h. die Tasks laufen nicht, also brauchen sie auch keinen TaskWorkSpace und keinen PC.

```
*SD /TY/ 0B --LOADX.Ld14000F  /TYB/
>> Ld14000F: (terminate)
*SD /TY 33
*S
0000D884->0000D890  MARK
0000D890->0000D8F4  ATSK  Resident  #IDLE
0000D8F4->0000D958  ATSK  Resident  #ERRDM
000118A0->00011904  TASK  Resident  #NDUP4
00011904->0001196C  MDLE  M
0001196C->00011A28  TASK  Hello
00011A28->00013904  TASK  Blink
00013904->003FFFD0  FREE
*lu
0001196C  +030/1  DORM  ----  TWS=00000000 PC=00000000  Hello
00011A28  +00A/1  DORM  ----  TWS=00000000 PC=00000000  Blink
```

Nun können Sie eine Task aktivieren, dazu brauchen Sie nur den Namen eingeben. Bitte achten Sie auf die Groß-/Kleinschreibung, bei Kommandos ist diese egal. Zuerst aktivieren Sie `Hello`:

```
*Hello
Hello World
```

Die Task `Hello` macht nichts anderes als `Hello World` auf der Schnittstelle `A1` auszugeben. Wenn Sie nun noch einmal ein `LU` absetzen sehen Sie, dass `Hello` wieder `DORM` ist. Jetzt aktivieren Sie die Task `Blink` und geben wieder ein `LU` ein:

```
*Blink

*lu
0001196C  +030/1  DORM  ----  TWS=00000000 PC=00000000  Hello
00011A28  +00A/1  SUSP  TICO  TWS=01FFFAA0 PC=01C1DC08  Blink
```

Mit einem Blick auf das EVA-Board sollten Sie jetzt eine rote und eine grüne LED (die grüne LED nur auf dem phyCORE-MPC555) im Wechsel blinken sehen. Bei der `LU`-Ausgabe sehen Sie, dass `Blink` nicht mehr `DORM` ist, sondern im Regelfall `SUSP` – es könnte aber auch sein, dass Sie `Blink` mal im `RUN` erwischen. Mit einem Blick ins Programm läßt sich feststellen, dass in der `TASK Blink` 2 Zeilen mit einem `AFTER` *x*`.5 SEC RESUME` stehen. Dies sind eigentlich 2 Befehle, zum ersten wird die Task suspendiert (deshalb der Zustand `SUSP`), zum zweiten wird die Task auf eine zeitliche Einplanung gesetzt (deshalb der Eintrag `TImeCOntinue`).

Nun können Sie selber einmal die Task `Hello` zeitlich einplanen. Dazu geben Sie den Befehl:

```
all 1 sec Hello
```

ein.

```
*all 1 sec Hello
Hello World
Hello World

*lu
0001196C  +030/1  SCHD  CYAC  TWS=00000000 PC=00000000  Hello
00011A28  +00A/1  SUSP  TICO  TWS=01FFFAA0 PC=01C1DC08  Blink
Hello World
Hello World
```

Jede Sekunde wird `Hello World` ausgegeben. Mit dem `LU` sehen Sie, dass `Hello` jetzt `SCHeDuled` (eingeplant) ist und zwar zyklisch.

Sie haben jetzt einen kleinen Eindruck von den Möglichkeiten des RTOS-UH bekommen. Jetzt sollen Sie noch eine kleine Änderung am Programm vornehmen und so einen Entwicklungszyklus durchführen. Ändern Sie den Text in der Task Hello auf Hallo Welt. Lassen Sie dann wie oben beschrieben den Compiler laufen. Wenn Sie alles richtig gemacht haben, kann der Compiler Ihre Datei fehlerfrei übersetzen. Bevor Sie den SRecord jetzt laden müssen Sie das Programm, das jetzt im Rechner ist entfernen. Eine mögliche Variante wäre den Reset-Knopf drücken, dann hat der Rechner alles bisherige vergessen. In der Praxis kann diese Methode etwas roh sein, da vielleicht noch andere Programme laufen. Zum entladen gibt es den Befehl `unload`. Sie können damit entweder eine einzelne Task, ein Modul oder ein Modul mit allen seinen Task entladen. Sie wollen das Modul M mit allen zugehörigen Task entladen. Dazu geben Sie ein:

```
unload M*
```

Das nebenbei noch die Ausgaben von Hello auf dem Bildschirm erscheinen braucht Sie nicht zu stören.

```
Hello World

*unload M*
Hello World
Hello World
Hello World
Hello World
Hello World
```

Je nach dem, wie schnell Sie tippen, erscheinen die Ausgaben auch noch nach dem Unload, sie wurden schon während Ihrer Eingabe erzeugt, konnten aber nicht ausgegeben werden, da die Schnittstelle blockiert war. Auch das Blinken der LED's hat aufgehört, der

Zustand der LED's ist so, wie er zum Zeitpunkt des Entladens war. Nun ist ein Entwicklungszyklus komplett, Sie können Ihre neue Programmversion laden und testen. Wenn Sie den Unload einmal vergessen, wundern Sie sich, warum Ihre Änderungen des Programms nicht greifen. Mit einem `LU` können Sie schnell feststellen, dass Ihr Programm 2 mal im Speicher ist und immer die alte Version läuft.

Zum Abschluß noch zwei Befehle, die gerade beim Programmieren von Hardware sehr nützlich sind. Sie können damit vom Betriebssystem direkt auf eine oder mehrere Hardwareadressen zugreifen. Vorher noch eine Warnung, schreiben Sie nur in den Speicher, wenn Sie sicher sind das richtige zu tun, Sie können u.U. sowohl das RTOS-UH "abschießen" als auch Hardware zerstören. Der folgende Befehl `dm-w` liest wortweise aus dem Speicher und zwar min. 3 Byte. Die Adresse kennen Sie schon aus dem PEARL-Programm, es sind das Daten- und das DataDirection Register. Der Befehl `sm-w` schreibt die angegebenen Wort an die entsprechende Adresse, hier schreiben Sie eine `0` ins Datenregister, damit sollten beide LED's gleichzeitig leuchten.

```
*dm-w 1f06100 3
01F06100:  FFFE  0003

*sm-w 1f06100 0000
```

5.2 CREST-C

Sie haben mehrere Möglichkeiten, den CREST-C-Compiler zu nutzen. Die komfortabelste Variante ist die Integration in das Visual Studio. Lesen Sie dazu bitte das Dokument PEARL\VS\VS_Einbindung.pdf, dort wird erklärt, wie Sie externe Tools einbinden können. Hier ein Bild, wie der Quelltext des Programms `hello.c` aussieht:

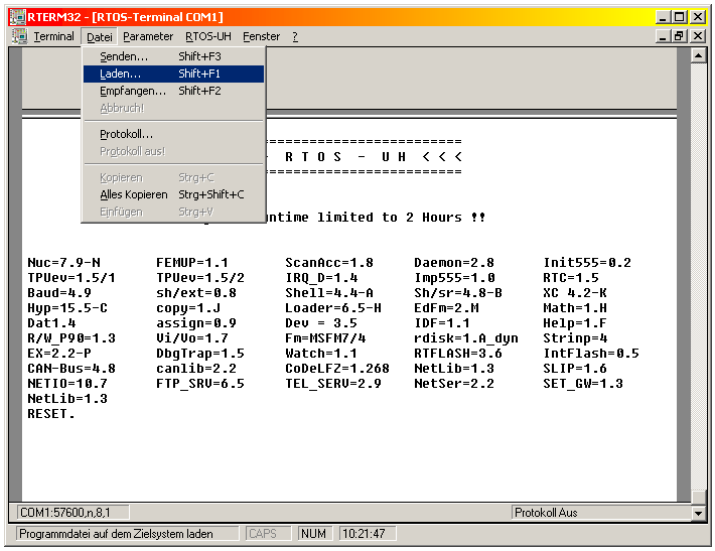
```
#include <h\stdlib.h>
#include <h\stdio.h>
#include <h\rtos.h>

#define MPIOSMDR  (WORD*)0x01F06100  /* MPIOSM Data Register           */
#define MPIOSMDDR (WORD*)0x01F06102  /* MPIOSM Data Direction Register */

/***********/
/*  Blink  */
/***********/

#pragma TASK USE_FUNCTION_NAME NO_TASKSTART

   void

Blink( void )

{
   WORD *pMPIOData    = MPIOSMDR  ;
   WORD *pMPIODataDir = MPIOSMDDR ;

   *pMPIODataDir = 0x0003 ; /* Die unteren beiden Ausgänge als Ausgang schalten */
   *pMPIOData    = 0x0003 ; /* Beide LED's ausschalten                          */

   for (;;)
   {
      *pMPIOData = 0x0001 ;    /* rote LED an  */
      rt_resume_after( 500 ) ; /* 500 ms Pause */

      *pMPIOData = 0x0002 ;     /* grüne LED an  */
      rt_resume_after( 1500 ) ; /* 1500 ms Pause */
   }
} /* Task Blink */

/***********/
/*  main  */
/***********/

   void

main( void )

{
   Blink() ;
   printf( "Hello, world!\n" ) ;
   rt_suspend() ; /* Hier muß main warten, da Blink auf dem gleichen Speicher arbeitet */

   system( "unload Blink" ) ; /* Auch Blink entladen */
} /* Task main */
```

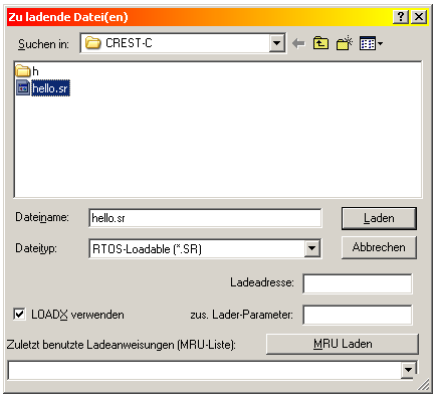
Die Beschreibung im folgenden geht davon aus, dass Sie kein Visual Studio haben und den Systemeditor nutzen. Um nun mit dem Programmieren anfangen zu können, nehmen

Sie den gesamten Ordner `CREST-C` und kopieren ihn auf Ihre Festplatte. Sie brauchen nichts weiter zu installieren, solange Sie nur in diesem Ordner arbeiten. Öffnen Sie eine Eingabeaufforderung (DOS-Box) in diesem Ordner und rufen Sie den CREST-C-Compiler mit `cmake hello.mak -pva` auf (das a diente nur der unbedingten Übersetzung):

```
C:\WINDOWS\system32\cmd.exe

T:\CDs\MPC5x5Demo\Programierung\CREST-C>cmake hello.mak -pva
Crest CMAKE                    Version for WINDOWS NT x86
Copyright (C) 1987-2003        by IEP GmbH
Release 2.653                  Oct 11 2004  09:11:12

*** Reading (hello.mak)
cccppc hello.c hello.obj
Crest C Compiler CCC Demo      PPC-Version for WINDOWS NT x86
Copyright (C) 1987-2003        by IEP GmbH
Release 2.653                  Oct 11 2004  09:05:24

Object file          :  hello.obj
Generated code for   :  PowerPC without FPU

Errors               :        0          Warnings           :        0
Number of C-lines    :     3372          Compile time       :  00:00:01
Code segment size    :      1A4 (hex)    Number of ASM-lines:      187
Data segment size    :        0 (hex)    Global symbols     :        2
Bss segment size     :        0 (hex)    Local symbols      :        6
Common segment size  :        0 (hex)    References         :        6
Local segment size   :        0 (hex)



echo sstart.obj hello.obj stdplong.lib > hello.lcr

clnppc hello.lcr hello.sr -M -N=HELLO
Crest S-Record-Linker CLN      PPC-Version for WINDOWS NT x86
Copyright (C) 1987-2003        by IEP GmbH
Release 2.653                  Oct 11 2004  09:10:53


Output file          : hello.sr
Map File             : hello.map

Text section size    :    45128          LOAD Relocs...
Data section size    :      292          Text Relocations :        11
Bss section size     :        0          Data Relocations :         0
Local section size   :     1568          Runtime Relocs...
Common section size  :        0          Data Relocations :         0
Stack size           :    16384          Local Relocations:         0
FILE structures      :       32
Link time            :  00:00:01


CMAKE completed: hello.mak

Errors               :        0
Number of commands   :        3          Used time          :  00:00:02

T:\CDs\MPC5x5Demo\Programierung\CREST-C>_
```

Das Programm wurde mit 0 Fehlern übersetzt, es wurde das MAP-File und der SRecord `hello.sr` erzeugt. Laden Sie den SRecord:





Nach dem Laden setzen Sie ein 'S'-Kommando ab, Sie sehen das Modul `HELLO` ohne weitere Task im Speicher liegt. Das Kommando `LU` zeigt keine Tasks. Die Prozedur `main`, die den Namen `hello` tragen soll, wurde nicht als Task, sondern als Bedienbefehl kodiert. Damit ist es möglich Parameter an das Programm zu übergeben. Mit dem `?`-Befehl werden alle Bedienbefehle ausgegeben, am Ende der Liste finden Sie den Befehl `HELLO`:

```
000118A0->00011904  TASK  Resident  #NDUP4
00011904->0001CA68  MDLE  HELLO
0001CA68->003FFFD0  FREE
003FFFD0->01FF9800  MDLE  #NORAM
01FF9800->01FFFFC4  FREE
01FFFFC4->00000000  MARK

*lu

*?
RTOS_Lexikon  (short information)
SB            SET_BAUD      DM            SM
DL            A             AFTER         ALL
C             DIR           DD            DL
SLIP          NETIO         FTPSRV
HELLO
```

Nun können Sie dieses Kommando aktivieren, dazu brauchen Sie nur den Namen eingeben. Die Groß-/Kleinschreibung spielt bei Kommandos keine Rolle, Task müssen so geschrieben werden, wie sie im Speicher stehen. Bitte geben Sie `hello` ein:

```
*hello
Hello, world!

*lu
003FEDF0  +014/1  DORM  ----  TWS=00000000 PC=00000000  Blink
01FFFC90  +014/1  SUSP  ----  TWS=01FFAAC8 PC=0001C590  HELLO/0000
```

Das Kommando `hello` macht nichts anderes als `Hello World!` auf der Schnittstelle `A1` auszugeben – dazu wird die Task `HELLO/0000` erzeugt. Die Zahl am Ende ist eine laufende Nummer, die an eine temporär erzeugte Task angehängt wird. Weiterhin wird die Task `Blink` erzeugt, die aber noch nicht losläuft. Wenn Sie nun noch einmal ein `LU` absetzen sehen Sie, dass `Hello` im `SUSPend` steht, so wie es programmiert ist. Sie sehen vorne die Taskkopfadresse, gefolgt von der Priorität (`Hello` hat die Priorität 20 (Hex 14), `Blink` die Priorität 20 (Hex 14). Der Zustand von `Blink` ist `DORM`, d.h. die Task läuft nicht, also braucht sie auch keinen TaskWorkSpace und keinen PC. Jetzt aktivieren Sie die Task `Blink` und geben wieder ein `LU` ein:

```
*Blink

*lu
003FEDF0  +014/1  SUSP  TICO  TWS=003FD7D0 PC=0001C60C  Blink
01FFFC90  +014/1  SUSP  ----  TWS=01FFAAC8 PC=0001C590  HELLO/0000
```

Mit einem Blick auf das EVA-Board sollten Sie jetzt eine rote und eine grüne LED (die grüne LED blinkt nur auf dem phyCORE-MPC555) im Wechsel blinken sehen. Bei der `LU`-Ausgabe sehen Sie, dass `Blink` nicht mehr `DORM` ist, sondern im Regelfall `SUSP` – es könnte aber auch sein, dass Sie `Blink` mal im `RUN` erwischen. Mit einem Blick ins Programm läßt sich feststellen, dass in der `TASK Blink` 2 Zeilen mit einem `rt_resume_after(` *x* `)` stehen. Dies sind eigentlich 2 Befehle, zum ersten wird die Task suspendiert (deshalb der Zustand `SUSP`), zum zweiten wird die Task auf eine zeitliche Einplanung gesetzt (deshalb der Eintrag `TImeCOntinue`).

Nun können Sie die Task `HELLO/0000` fortsetzen. Dazu geben Sie den Befehl:

```
c HELLO/0000
```

ein.

Damit wird zuerst die Task `Blink` entladen und dann `HELLO/0000` beendet. Mit dem `LU` sehen Sie, dass keine Task mehr vorhanden ist.

Sie haben jetzt einen kleinen Eindruck von den Möglichkeiten des RTOS-UH bekommen. Jetzt sollen Sie noch eine kleine Änderung am Programm vornehmen und so einen Entwicklungszyklus durchführen. Ändern Sie den Text in der Task `main` auf `Hallo Welt`.

Lassen Sie dann wie oben beschrieben den Compiler laufen. Wenn Sie alles richtig gemacht haben, kann der Compiler Ihre Datei fehlerfrei übersetzen. Bevor Sie den SRecord jetzt laden müssen Sie das Programm, das jetzt im Rechner ist entfernen. Eine mögliche Variante wäre den Reset-Knopf drücken, dann hat der Rechner alles bisherige vergessen. In der Praxis kann diese Methode etwas roh sein, da vielleicht noch andere Programme laufen. Zum entladen gibt es den Befehl `unload`. Sie können damit entweder eine einzelne Task oder ein Modul entladen. Sie wollen das Modul `HELLO` entladen, die Task sind nach der Fortsetzung von `HELLO/0000` schon nicht mehr im Speicher. Dazu geben Sie ein:

```
unload HELLO
```

Nun ist ein Entwicklungszyklus komplett, Sie können Ihre neue Programmversion laden und testen. Wenn Sie den Unload einmal vergessen, wundern Sie sich, warum Ihre Änderungen des Programms nicht greifen. Mit einem `LU` können Sie schnell feststellen, dass Ihr Programm 2 mal im Speicher ist und immer die alte Version läuft.

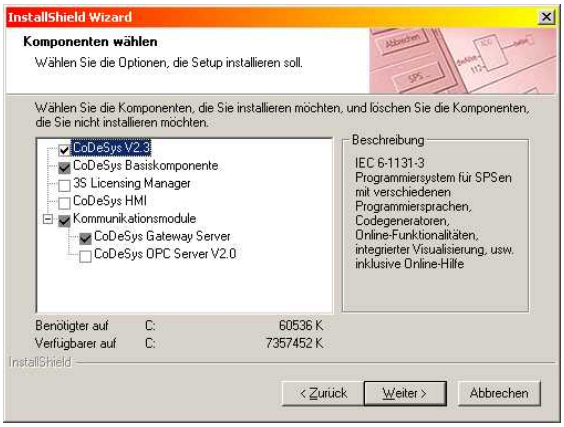
Zum Abschluß noch zwei Befehle, die gerade beim Programmieren von Hardware sehr nützlich sind. Sie können damit vom Betriebssystem direkt auf eine oder mehrere Hardwareadressen zugreifen. Vorher noch eine Warnung, schreiben Sie nur in den Speicher, wenn Sie sicher sind das richtige zu tun, Sie können u.U. sowohl das RTOS-UH "abschießen" als auch Hardware zerstören. Der folgende Befehl `dm-w` liest wortweise aus dem Speicher und zwar min. 3 Byte. Die Adresse kennen Sie schon aus dem PEARL-Programm, es sind das Daten- und das DataDirection Register. Der Befehl `sm-w` schreibt die angegebenen Wort an die entsprechende Adresse, hier schreiben Sie eine `0` ins Datenregister, damit sollten beide LED's gleichzeitig leuchten.

```
*dm-w 1f06100 3
01F06100:  FFFE  0003

*sm-w 1f06100 0000
```

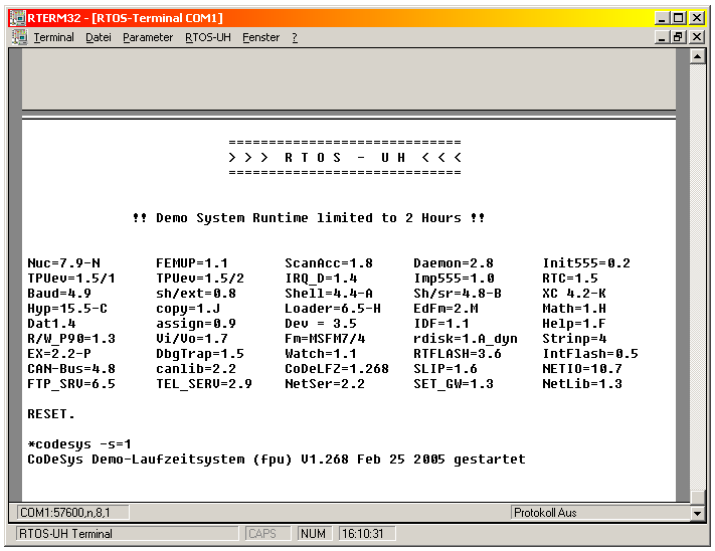
5.3 CoDeSys

! *Für die Nutzung des CoDeSys auf dem PhyCore 565 ist es notwendig das CoDeSys-Laufzeitsystem erst in das RTOS-UH nachzuladen, das es nicht im Flash abgelegt ist. Dazu ist mittels des RTERM und der Funktion „Datei Laden“ die folgende Datei \MPC5x5-Demo\MPC565\RTOSinstall\CODESYS.FPU nachzuladen.*

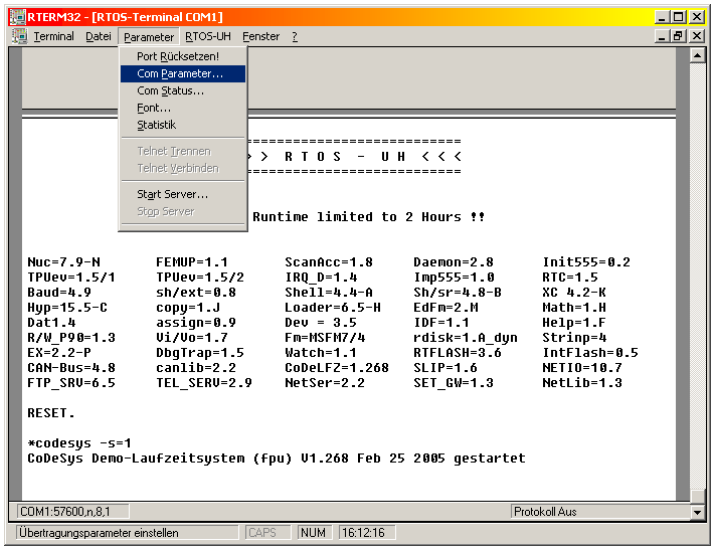
Das CoDeSys müssen Sie auf Ihrem Rechner installieren. Dazu starten Sie die Datei `Setup.exe` im Ordner `Programmierung\CoDeSys`. Sie müssen nicht alle Komponenten installieren, so sähe die minimale Installation aus:

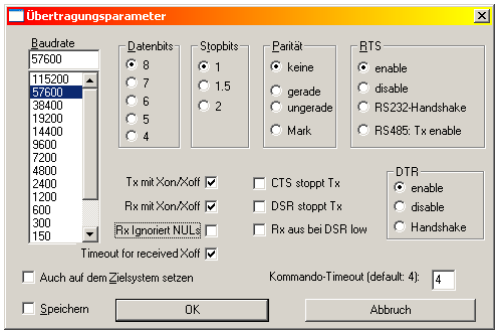


Bestätigen Sie alle sonstigen Eingaben. Wenn die Installation abgeschlossen ist, starten Sie auf den RTOS-UH das Laufzeitsystem mit:



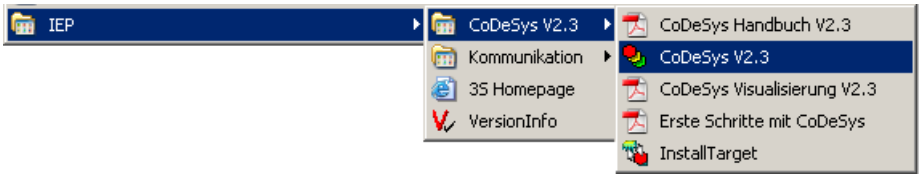
Bevor Sie jetzt den RTERM32 schließen müssen Sie noch die folgende Einstellung vornehmen:



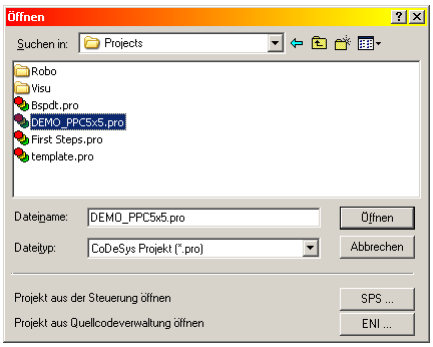


Schalten Sie "`Rx Ignoriert NULs`" auf jeden Fall aus! Sonst kommt keine Kommunikation zustande! Schließen Sie RTERM32.

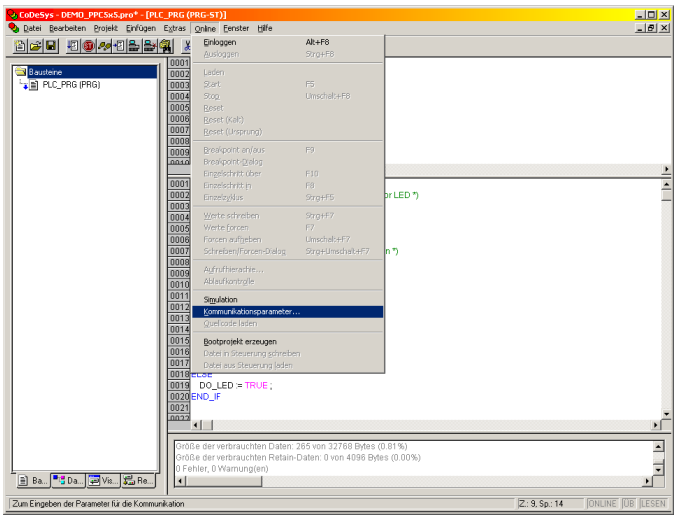
Nun starten Sie das CoDeSys:

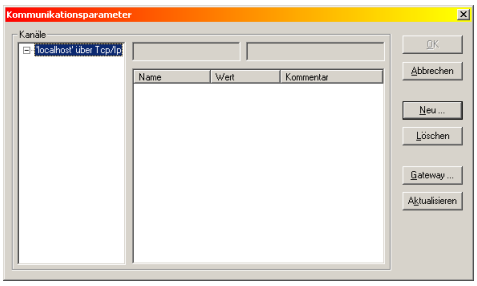


Kopieren Sie das Projekt `Programmierung\CoDeSys\DEMO_PPC5x5.pro` in Ihr Installationsverzeichnis (im Regelfall `C:\Programme\IEP\CoDeSys V2.3\Projects`). Öffnen Sie das Projekt:

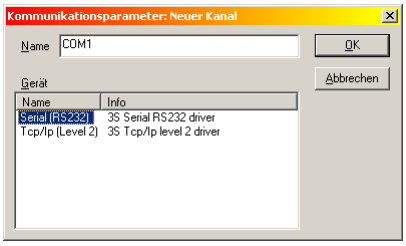


Nach dem Öffnen des Projektes müssen Sie die Kommunikationsparameter einstellen:

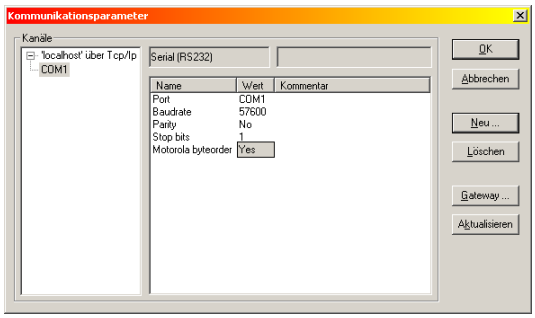




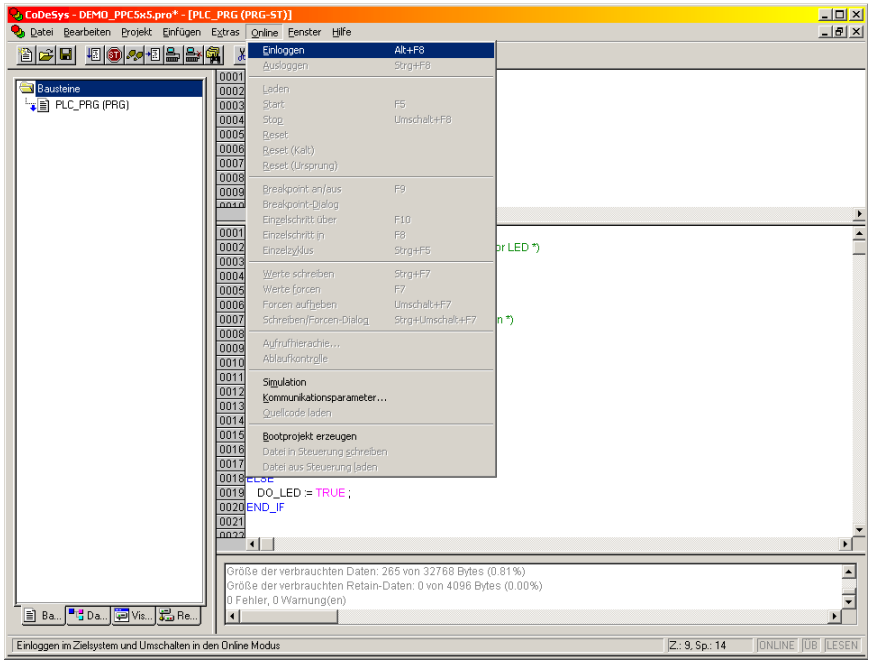
Klicken Sie auf `Neu`:



Wählen Sie `Serial(RS232)`, Sie können auch einen eigenen Namen eintragen. Dann können Sie durch Doppelklick auf die Einträge in der Spalte "Wert" die richtigen Werte einstellen:

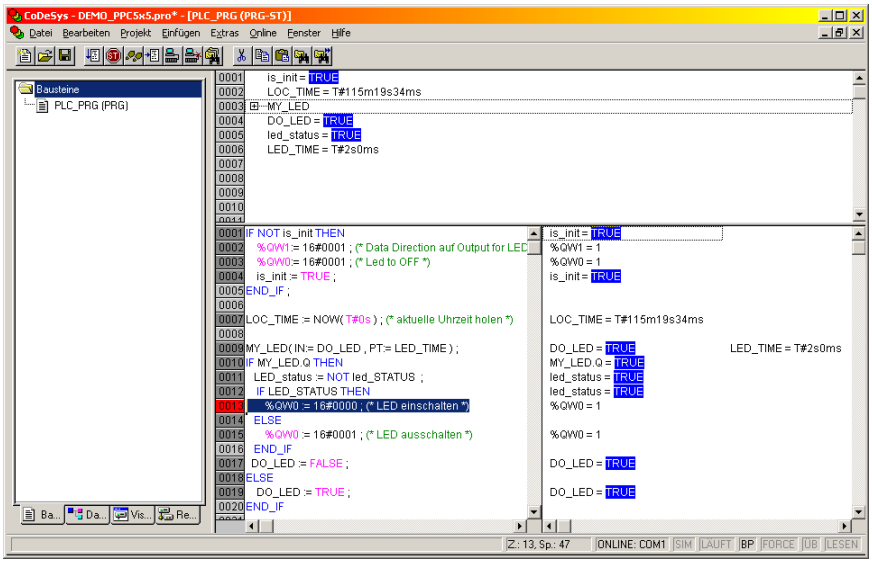


Die "*Baudrate*" müssen Sie auf 57600 stellen und "*Motorola byteorder*" auf Yes. Schließen Sie das Fenster. Nun können Sie Verbindung zum Zielsystem herstellen:



Sie sollten die folgende Meldung mit Ja beantworten:

Daraufhin wird das Programm auf das MPC5x5-Board übertragen und gestartet. Nun können Sie das Programm bei der Arbeit beobachten und z.B. einen Breakpoint setzen, indem Sie in Zeile 13 klicken. Die Zeilennummer wird erst blau um den Breakpoint anzuzeigen, wenn das Programm diese Zeile abarbeiten will, wird der Breakpoint rot und das Programm hält an:

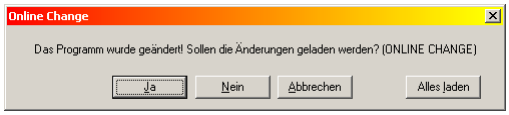


Mit Drücken von F8 können Sie im Single-Step durch das Programm laufen und die Variablenwerte beobachten. Sie können den Breakpoint wieder entfernen und das Programm mit F5 weiter laufen lassen.

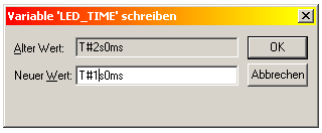
Loggen Sie sich aus dem Zielsystem aus, Sie werden feststellen, dass die LED weiter blinkt. Nun kommentieren Sie die Zeile 7, in der die Zeit geholt wird, aus:

```
0007 (* LOC_TIME := NOW( T#0s ) ; (* aktuelle Uhrzeit holen *) *)
```

Gehen Sie wieder Online:



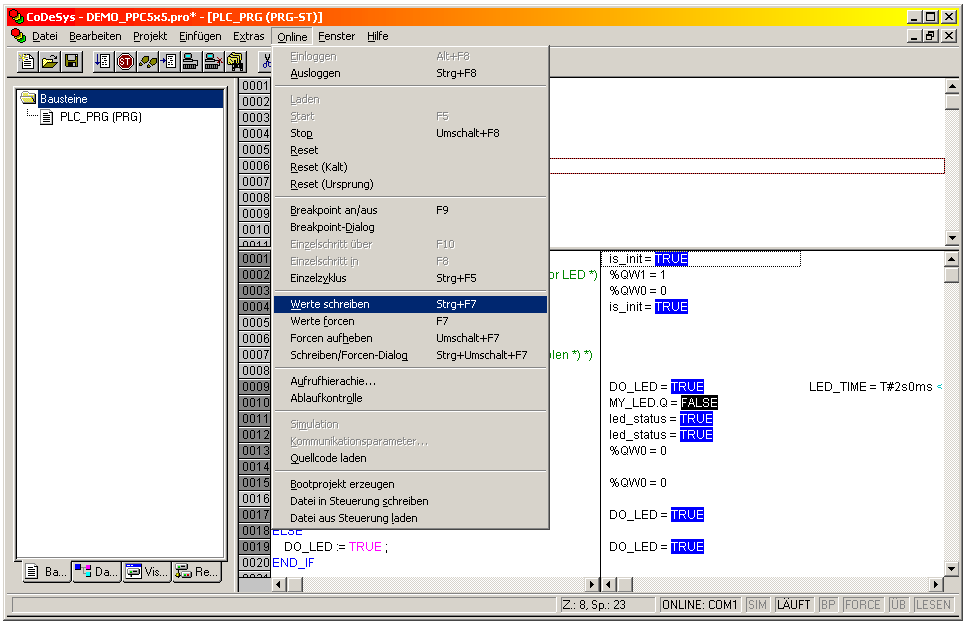
und bestätigen Sie den Online Change. Nun läuft das geänderte Programm. Jetzt soll noch eine Variable geändert werden. Machen Sie einen Doppelklick auf `LED_TIME`. Es öffnet sich folgendes Fenster, in das Sie z.B. die Zeit auf 1 sec verkürzen:



Die Änderung wird nicht sofort übernommen, sondern erst einmal nur angezeigt:

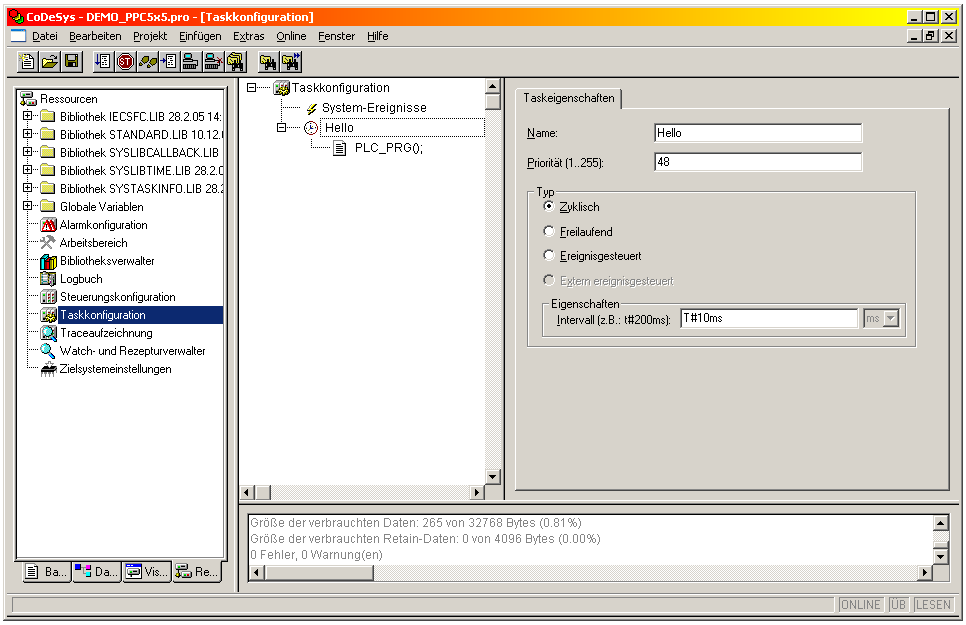
```
0006 LED_TIME = T#2s0ms <:= T#1s0ms>
```

Damit die Änderung wirksam wird, müssen Sie die Werte schreiben:



Damit wird die Änderung wirksam und die LED blinkt schneller.

Mit dem rechten Reiter im linken Fenster (Re...) kommen Sie in die Taskkonfiguration. Dort können Sie die Parameter der Task einstellen, weitere Tasks anlegen usw.:



Sie sehen, dass die Task `Hello` heißt, auf Priorität 48 läuft und zyklisch alle 10ms eingeplant ist. Alle im CoDeSys angelegten Tasks laufen im RTOS-UH als eigenständige, voneinander unabhängige Prozesse.

5.4 Debugger

Der Debugger kann sowohl für PEARL90- als auch für CREST-C-Programme eingesetzt werden. Sie benötigen eine TCP/IP Verbindung – wenn Ihr EVA-Board keinen Ethernetan-

schluß besitzt, können Sie auch über eine serielle SLIP-Verbindung debuggen. Das weitere Vorgehen wird im Dokument rtdebug.pdf beschrieben. Sie müssen den Debugger nicht installieren, starten Sie Ihn einfach.

6 Programmierung der CAN-Schnittstelle

Im Ordner CANDemo finden Sie folgende Dateien:

candemo.c	Beispielprogramm in C
candemo.lcr	Link-Datei (intern)
CANDEMO.map	Linker Map
candemo.obj	OBJ-File des Compilers
CANDEMO.SR	Ladbarer S-Record
CANLIB.SR	Library für PEARL
canlibS.obj	Library für C
canproto.h	Header für C
makefile	Makefile

Im ersten Schritt könnten Sie die beiden CAN-Schnittstellen verbinden, das Programm `CANDEMO.SR` laden und mit `*candemo` starten. Es werden 4 Tasks erzeugt, für jeden CAN-Kanal eine Sende- und eine Empfangstask. Alle Sekunde wird ein CAN-Paket verschickt und empfangen.

Wenn Sie Ihre C-Entwicklungsumgebung eingerichtet haben, reicht der Aufruf von `cmake`, um das Programm zu übersetzten. Ggf. müssen Sie im `makefile` noch den Pfad für die Library anpassen.

Sollten Sie in PEARL programmieren, müssen Sie vor dem Laden Ihres PEARL-Programms die Datei `CANLIB.SR` laden, damit die CAN-Library für PEARL zur Verfügung steht.