PEARL-News

Ausgabe 2/2010

Mitteilungen
der GI-Fachgruppe Echtzeitsysteme

ISSN 1437-5966

Impressum

Herausgeber	GI-Fachgruppe Echtzeitsysteme (RT) URL: http://www.real-time.de
Sprecher	Dr. P. Holleczek Universität Erlangen-Nürnberg, Regionales Rechenzentrum Martensstraße 1, D-91058 Erlangen Telefon: 09131/85-27817 Telefax: 09131/30 29 41 E-Mail: peter.holleczek@rrze.uni-erlangen.de
Stellvertreter	Prof. Dr. Dr. W. Halang FernUniversität in Hagen Universitätsstraße 27 - PRG D-58084 Hagen Telefon: 02331/987-372 Telefax: 02331/987-375 E-Mail: wolfgang.halang@fernuni-hagen.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@hs-furtwangen.de
ISSN	1437-5966

Redaktionell abgeschlossen am 31. Oktober 2010

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.

Ich möchte 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)

Inhaltsverzeichnis

1 Vernetzung von Mikrocontrollern: Schnelle Datenübertragung mit TCP/IP

Im zunehmenden Maße werden auch in kleineren Mikrocontrollern Hochgeschwindigkeitsschnittstellen wie Ethernet oder USB implementiert. Neben den bisherigen Vernetzungen mit z. B. I2C oder CAN-Feldbussen lassen sich nun ganz andere Möglichkeiten der Datenübertragung realisieren. Dauerte z. B. ein Upload eines Programmupdates mit einem Umfang von 512 kBytes bei Verwendung eines CAN-Busses noch mindestens 8 Sekunden, so kann mit Ethernet die Aufgabe in 0,05 Sekunden abgeschlossen sein. USB hat mittlerweile eine sehr hohe Verbreitung gefunden. Allerdings ist eine hierauf basierende Vernetzung wenig flexibel, da ein zentraler Host notwendig ist. Eine Client-zu-Client Kommunikation muss immer indirekt über den Host erfolgen. Zudem ist eine Übertragung von zeitkritischen Daten nur durch hohen Aufwand im Host möglich. Ein weiteres Problem der USB-Schnittstelle ist die nur aufwändig zu realisierende galvanische Trennung. Daher wird man in vielen Anwendungen der Ethernetschnittstelle den Vorzug geben. Die heute üblicherweise implementierten Ethernet-Controller bieten in der Regel Datenübertragungsraten bis zu 100 MBit/s. Hier stellt sich die Frage, ob diese Geschwindigkeit überhaupt richtig ausgenutzt werden kann. Zum einen haben Mikrocontroller nur eine geringe Rechenleistung, zum anderen steht meist nur wenig Speicher zur Verfügung. Am Beispiel eines ARM-basierten Mikrocontrollers der LPC2000 Familie des Herstellers NXP sollen in diesem Bericht die Möglichkeiten der Ethernetvernetzung aufgezeigt werden. Der in den Experimenten verwendete LPC2378 Mikrocontroller (ARM 7 TDMI) kann mit einer Taktrate von 72 MHz betrieben werden. Er besitzt 512 kByte Flashspeicher und 64 kByte RAM.

In den Mikrocontrollern der LPC2000-Familie ist nur die Ethernet-Media-Access-Control-Schicht (MAC) implementiert. Zur Anbindung an das Netzwerk sind, wie die Abbildung 3 zeigt, ein externer PHY-Baustein, der die pysikalische Ethernet-Schicht beinhaltet, und ein Übertrager erforderlich. Der Datentransfer von und zum Netzwerk wird mittels DMA abgewickelt. Hierfür steht ein eigener 16 kByte Speicher zur Verfügung, wobei der Transfer über einen von der CPU getrennten Bus erfolgt. Dies hat den enormen Vorteil, dass auch während des Empfangs und des Sendens die CPU mit voller Geschwindigkeit weiter arbeiten kann. Nachteilig ist allerdings, dass auf Grund dieser Trennung der Netzwerkspeicher auf 16 kByte begrenzt ist, da der DMA-Controller den Hauptspeicher nicht erreichen kann. Jedoch kann der Ethernet-Speicherbereich bei Nichtbenutzung des USB-Controllers um dessen Speicher erweitert werden.

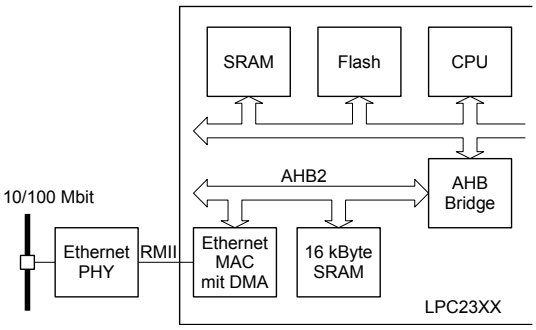


Abbildung 1: Ausschnitt aus dem Blockschaltbild der LPC23xx-Familie

1.1 TCP/IP-Implementierungen für Mikroprozessoren

Etliche Hersteller bieten speziell für Mikroprozessoren angepasste TCP/IP-Software an, z. B. die Firmen Interniche [1] oder Sevenstax GmbH [2]. Adam Dunkels [3] hat einen TCP/IP-Stack für sehr kleine Mikroprozessoren entwickelt (μIP). Hierfür gibt es mittlerweile etliche Anpassungen für unterschiedliche Plattformen. Im Rahmen einer Diplomarbeit [4] wurden an der μIP-Software, angepasst an die LPC2000-Familie, von Stefan Diercks Performancemessungen durchgeführt. Die Ergebnisse waren ziemlich ernüchternd. Eine Ursache ist sicherlich, dass μIP sehr kleine Puffer für die Datenübertragung verwendet. Ein weiterer Grund ist, dass die Software die Möglichkeiten der LPC2000-Familie nicht voll ausnutzen kann. Deswegen wurde entschieden, für die Untersuchungen einen eigenen Stack mit der Zielsetzung zu entwickeln, bei einer möglichst hohen Datenrate eine gute Ausnutzung der CPU-Ressourcen zu erreichen.

1.1.1 Anwendungsschnittstelle der Netzwerksoftware

Die klassische Socket-Schnittstelle lässt sich auf einem Mikroprozessor nicht effizient implementieren. Hierfür gibt es mehrere Ursachen. Zum einen wird die Anwendung in den *send*- und *recv*-Funktionen der Socket-Schnittstelle typischerweise blockiert. Dies setzt die Funktionalität eines Multitasking-Betriebs-

systems voraus. Neben der Voraussetzung des Betriebssystems ergibt sich in der Regel der zusätzliche Nachteil, dass bei jedem Sende- oder Empfangsvorgang Taskwechsel durchgeführt werden müssen, die die Leistungsfähigkeit der Netzwerksoftware deutlich reduzieren können. Der zweite Grund, der für die Verwendung einer speziellen Anwendungsschnittstelle spricht, ist, dass send und recv Pufferspeicher in der Anwendung voraussetzen. Das bedeutet, dass z. B. bei einem Sendevorgang die Daten aus diesem Puffer in den Socket-internen Speicher kopiert werden müssen. Neben dem hierfür erforderlichen zusätzlichen Speicherbedarf ergibt sich daraus eine unnötige CPU-Belastung.

Ziel der eigenen Implementierung war es, dass als Datenspeicher nur der spezielle Bereich des Ethernet-Controllers vom LPC2378 verwendet wird. Daten aus der Anwendung sollen direkt in diesen Speicher hinein kopiert werden. Beim Abarbeiten der Protokollschichten werden sie nur noch um die notwendigen Protokollköpfe ergänzt und können anschließend direkt vom DMA-Controller ohne zusätzliche CPU-Unterstützung versendet werden. Dieses Vorgehen erfordert allerdings, dass im Gegensatz zur Socket-API, die Anwendung vor einem Sendevorgang zunächst Speicher anfordern muss. Dabei kann es passieren, dass kein Speicher mehr zur Verfügung steht oder dass das vom Empfänger signalisierte TCP-Fenster zu klein ist. Die Anwendung bekommt dann keinen Speicher zugewiesen und muss warten. Die Freigabe des Speichers erfolgt nach dem Sendevorgang automatisch durch die TCP-Schicht, sobald die zugehörige Bestätigung eingetroffen ist. Falls diese nicht rechtzeitig erfolgt, wird von der TCP-Schicht selbstständig ein erneutes Senden durchgeführt. Das Empfangen gestaltet sich einfacher. Auch hier werden die Daten direkt aus dem DMA-Bereich an die Anwendung durchgereicht. Nach Bearbeitung der Daten kann die Anwendung den Speicher direkt zum Senden nutzen oder muss ihn wieder frei geben. Es wurde darauf geachtet, dass notwendige Bestätigungen mit eventuell zu versendenden Daten zusammengefasst werden.

Die vorliegende Implementierung benutzt kein Betriebssystem. Die gesamte Protokollbearbeitung erfolgt im Pollingverfahren. Es ist keine Interruptroutine vorhanden. In der Hauptschleife des Mikroprozessorprogramms muss lediglich regelmäßig die TCP-Software aufgerufen werden. Alle Schichten sind nicht blockierend auf Basis von Zustandsautomaten aufgebaut. Jede Anwendung muss die Ereignisfunktionen *init*, *tick*, *open*, *receive* und *close* zur Verfügung stellen. Mittels *open* und *close* wird die Anwendung über den Auf- und Abbau der TCP-Verbindung informiert. receive leitet empfangene Pakete an die Anwendung weiter. In *tick* kann bei Bedarf zusätzliche Anwendungslogik eingebaut werden. Die Anwendung ist durch die genannten Funktionen ereignisgesteuert. Es bietet sich daher an, auch die Anwendungslogik als Zustandsautomat zu implementieren. Die Verbindungen zwischen den Anwendungen und der Netzwerksoftware werden mit einem Konfigurationsmodul statisch festgelegt. Dies hat den Vorteil, dass hierfür kein zusätzlicher Datenspeicher benötigt wird. Über das Konfigurationsmodul wird auch die genaue Aufteilung des DMA-Speichers und die maximale Anzahl der gleichzeitig zugelassenen Verbindungen pro Server festgelegt. Auf diesem Wege konnte die Verwendung von dynamischen Speicher vermieden werden, der gesamte Speicherbedarf kann während der Programmerstellung ermittelt werden.

Bislang wurden folgende Serveranwendungen realisiert: Telnet-Zugang, einfacher Webserver, CAN-Ethernet-Bridge, netzwerkfähiger JTAG-Adapter, Bootloader, Datenrekorder (siehe unten). Die Server sind in voneinander isolierten Modulen implementiert und können in beliebiger Kombination gleichzeitig betrieben werden.

1.2 Performancemessungen

Zum Ausmessen der Leistungsfähigkeit der Software standen einfache Anwendungen zur Verfügung. Zum Überprüfen der Empfangsrichtung wurde eine Applikation auf einem PC gestartet, die mit maximaler Geschwindigkeit per TCP/IP versucht, Daten an das Mikroprozessorsystem zu senden. In dem Mikroprozessor wurden die Daten ohne Prüfung der TCP-Checksumme an die Serveranwendung übergeben. Das Auslassen der Prüfung kann akzeptiert werden, da bereits auf der Ethernet-Schicht eine gute Kontrolle auf Basis einer 32-Bit CRC-Prüfsumme erfolgt ist. Eine wichtige Optimierungsgröße ist die Wahl des TCP-Fensters. Um Speicher zu sparen, sollte ein möglichst kleines Fenster gewählt werden. Andererseits bedeutet ein zu kleines Fenster einen Performance-Nachteil, da dann auf Grund der Round-Trip-Time (RTTI) keine vollständige Ausnutzung des Mediums möglich ist. Zusätzlich reduzieren die Betriebssysteme bei sehr klein eingestellten Fenstern die maximale Größe der TCP-Segmente. Die Messungen zeigten, dass z. B. Linux erst ab einer Fenstergröße von 3000 eine maximale Ausnutzung der Segmente vornimmt. Um den Einfluss der RTTI auf den Durchsatz zu demonstrieren, wurden die Messungen mit unterschiedlicher Anzahl Switches zwischen PC und Mikroprozessor durchgeführt. Die Switches arbeiteten nach dem Store-und-Forward-Verfahren, so dass jede Switch die RTTI um die Serialisierungszeit eines Paketes verlängert (bei voller Paketgröße ca. 120 μsec). Abbildung 2 zeigt die Messergebnisse für zwei verschie-

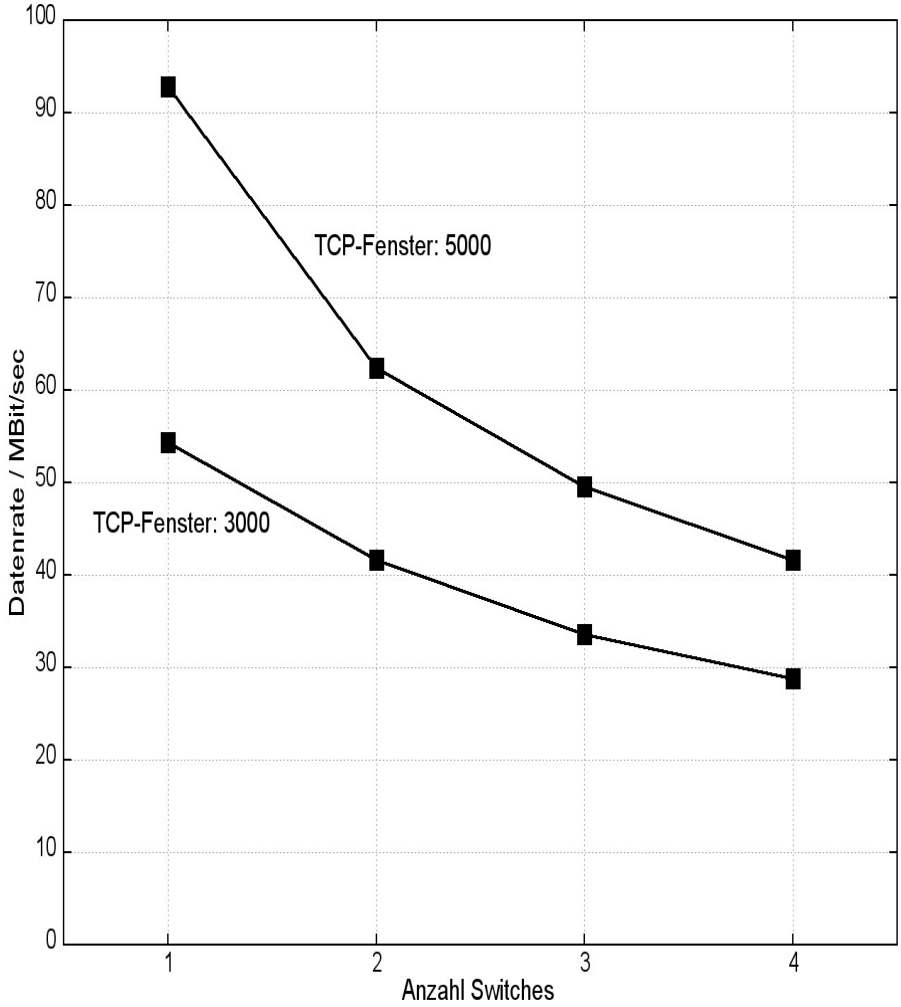


Abbildung 2: Durchsatz der TCP/IP-Übertragung von einem PC zum Mikroprozessorsystem in Abhängigkeit von der Anzahl der Switches. Ein weiterer Einfluss ergibt sich aus der Größe des verwendeten TCP-Fensters.

dene Fenstergrößen. Sie verdeutlicht, dass die Netzwerksoftware auf dem Mikroprozessor tatsächlich eine nahezu 100 % Auslastung des Netzwerks ermöglicht, allerdings nur bei einer Fenstergröße von 5000 und es darf nur eine Switch zwischen Mikroprozessor und PC geschaltet sein. Deutlich sieht man die Performancereduktion auf Grund der RTTI-Erhöhung durch Einfügen von weiteren Switches in den Datenpfad.

In der Senderichtung versucht der Mikrocontroller mit maximaler Geschwindigkeit Daten an den PC zu senden. Bei dieser Datenübertragung wird die Fenstergröße vom PC festgelegt und sollte für den Aufbau im Labor immer ausreichend sein. Allerdings wird der Mikrocontroller in der Regel dieses vorgegebene Fenster gar nicht ausnutzen können, da ihn der dafür notwendige Speicher fehlt. Eine wesentliche Optimierungsgröße ist daher die Menge Speicher, die der Server auf den Mikrocontroller für das Senden zur Verfügung hat. Im einfachsten Fall reicht dieser Speicher für genau ein Segment aus. In diesem Falle muss die Anwendung nach dem Versenden dieses Segmentes warten, bis die Bestätigung eingetroffen ist. Hier würde man vermuten, dass ca. eine RTTI gewartet werden muss. Leider ist das nicht der Fall. Auf Grund des Piggyback-Algorithmus, der dafür sorgt, dass Bestätigungen mit eventuell zu sendenden Antwortdaten zusammengefasst werden, kommt es je nach Betriebssystem zu erheblichen Verzögerungen. So sind unter Linux 40 msec zu beobachten und unter Windows sogar ca. 200 msec. Das bedeutet, das unter Linux nur alle 40 msec ein Segment mit 1460 Bytes und unter Windows sogar nur alle 200 msec versendet werden kann. Die sich daraus ergebenden Datenraten sind unter Linux 290 kBit/sec und unter Windows 58 kBit/sec! Um zu akzeptablen Raten zu kommen, ist also Speicher für mindestens 2 Segmente vorzusehen.

Abbildung 3 zeigt die Messergebnisse. Variiert wurde die Anzahl der Switches auf dem Weg vom Mikro-

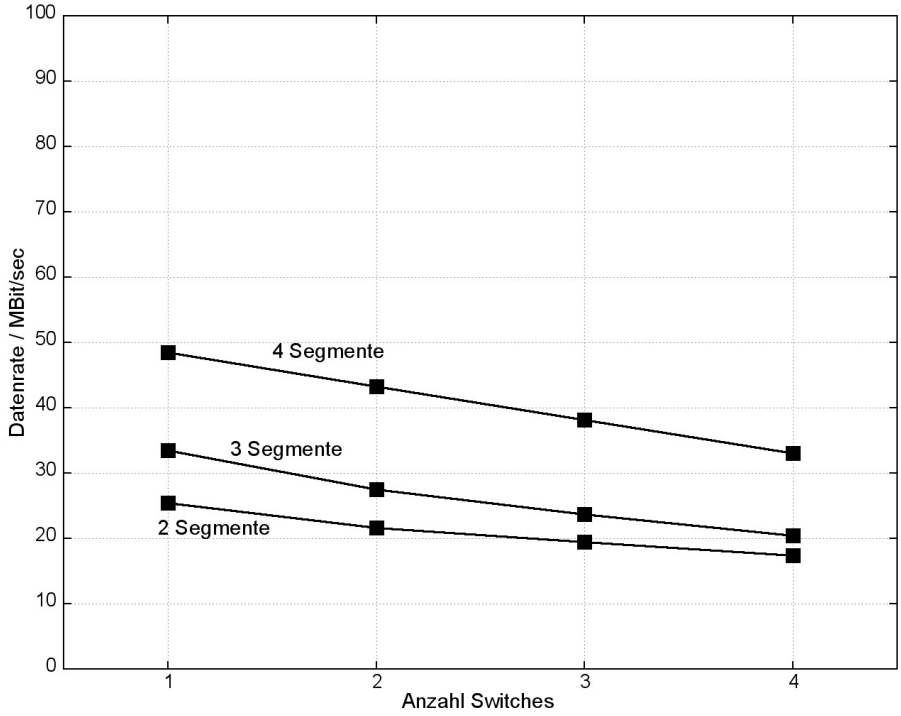


Abbildung 3: Gemessener Durchsatz beim Senden über TCP/IP in Abhängigkeit von der Anzahl Switches, die zwischen Sender und Empfänger geschaltet wurden. Der Durchsatz ist weiterhin abhängig von der Anzahl der im Sender zur Verfügung stehenden Puffer (Segmente).

prozessorsystem zum PC und die Größe des Speichers, die der Sender zum Senden nutzen konnte. In dieser Datenrichtung kann im günstigsten Fall die Datenrate nur zur Hälfte ausgenutzt werden. Ursache ist, dass jetzt die CPU-Leistung nicht mehr ausreicht, da beim Senden die TCP-Checksumme berechnet werden muss. Dies nimmt einen erheblichen Teil der CPU-Zeit in Anspruch. Auch in dieser Messung ergibt sich eine deutliche Abhängigkeit des Durchsatzes von der RTTI. Hierbei ist anzumerken, dass die Experimente nur im Labor durchgeführt worden sind. Beim Überbrücken von großen Entfernungen würden die dann auftretenden Round-Trip-Zeiten die Datenrate ganz erheblich reduzieren.

Neben dem Durchsatz wurden auch die dabei auftretenden CPU-Belastungen durch Auswertung der Durchläufe in der Idle-Schleife ermittelt. Wie erwartet, ist die CPU-Belastung nur von der aktuell erreichten Datenrate abhängig. Es ergibt sich ein direkter linearer Zusammenhang (siehe Abbildung 4). Aus dieser Abhängigkeit lässt sich die benötigte Rechenzeit pro Sende- oder Empfangsvorgang bestimmen. Beim Senden beträgt die Rechenzeit etwa 160 μsec. Circa ein Drittel der Zeit wird benötigt, um die zu sendenden Daten in den Sendepuffer zu kopieren. Der wesentliche Teil der restlichen Zeit verbraucht die Checksummenberechnung. Hierzu muss bei jedem Sendevorgang die Summe über alle Daten eines Segments gebildet werden. Die entsprechende Routine ist bislang in C implementiert. Durch Verwendung einer Assemblerroutine könnte man zu noch günstigeren Ergebnissen kommen. Die Auslastung während des Empfangsvorgangs ist deutlich geringer, da hier keine Checksummenberechnung durchgeführt wurde. Außerdem wurden die Daten in der Anwendung nicht kopiert. Somit ist hier nur die Zeit für die Abarbeitung des TCP-Protokolls enthalten.

1.3 Datenrekorder

Zur Demonstration der Leistungsfähigkeit der Netzwerkimplementierung wurde eine Anwendung zum kontinuierlichen Aufzeichnen von digitalen Eingangssignalen entwickelt. Mit dieser Beispielanwendung sollen insbesondere die Erfordernisse bei der Übertragung von zeitkritischen Daten aufgezeigt werden. Ziel ist eine kontinuierliche Speicherung von Bussignalen (z. B. I2C oder CAN) über 24 Stunden / 7 Tage, die mit einer Rate von 1 Million Aufzeichnungen pro Sekunde erfasst werden. Mit Hilfe der gespeicherten Daten sollen selten auftretende Übertragungsprobleme, die z. B. durch Störungen oder Softwareprobleme

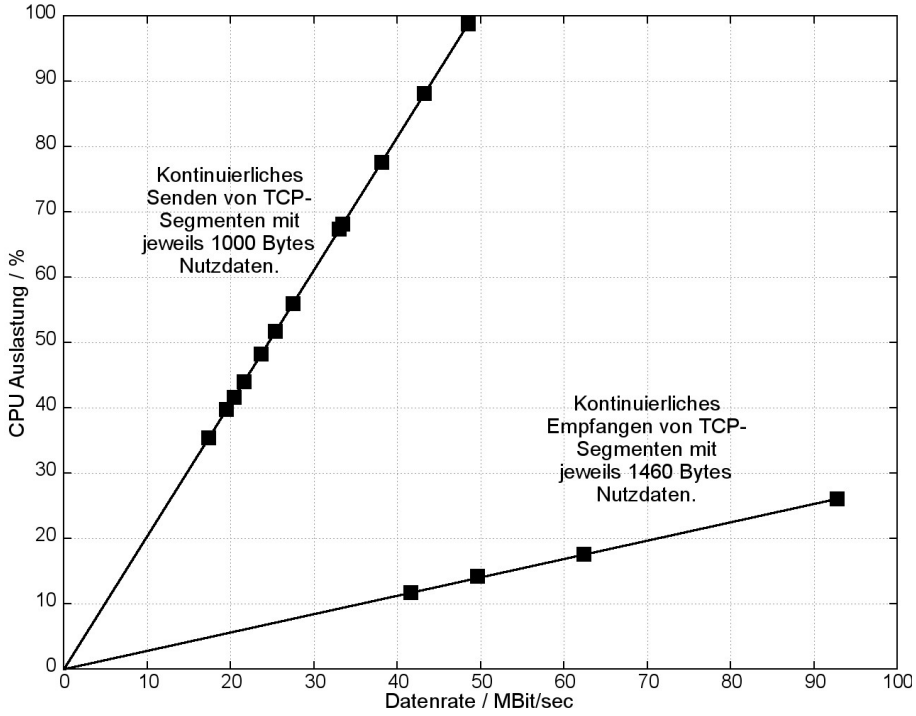


Abbildung 4: CPU-Auslastung als Funktion der Datenrate beim Senden und Empfangen. Während des Sendens werden die Daten in der Anwendung kopiert und die Checksumme berechnet. Beim Empfangen wird die Checksumme nicht ausgewertet und die Daten werden nicht kopiert.

ausgelöst worden sind, untersucht werden können. Insgesamt werden die Daten von 4 Eingangskanälen erfasst, über das Netzwerk zu einem PC übertragen und dort auf die Festplatte gespeichert. Bei den genannten Anforderungen ergibt sich eine kontinuierliche Datenrate von 4 MBit/sec. Auf dem PC müssen pro Tag ca. 43 GByte gespeichert werden.

Die eigentliche Datenerfassung erfolgt in einer Interruptserviceroutine mit einer Aktivierungsrate von 1 MHz. Hierfür wurde der Fast-Interrupt der ARM-CPU genutzt. Diese enorm hohe Interruptrate ist machbar, da die CPU 8 zusätzliche Register besitzt, die ausschließlich für die Verwendung im Fast-Interrupt reserviert sind. Die Aufgabe der Interruptroutine ist die Erfassung der digitalen Werte vom Parallelport, die Umwandlung in Bytes und Transfer in den Zielpuffer. Zusätzlich ist ein einfaches Puffermanagement notwendig, das dafür sorgt, dass ein vollständig aufgefüllter Puffer der TCP-Software gemeldet wird und eine Umschaltung auf den nächsten leeren stattfindet. Die Puffer wurden direkt in dem DMA-Bereich des Ethernet-Controllers angelegt, so dass die Daten ohne zusätzliches Kopieren versendet werden können. Die TCP/IP- und Ethernet-Schichten müssen nur noch die notwendigen Kopfdaten hinzufügen. Bei dem gewählten Aufbau wurden pro TCP-Segment 1000 Bytes verschickt, so dass jeweils alle 2 msec ein Segment versendet werden muss.

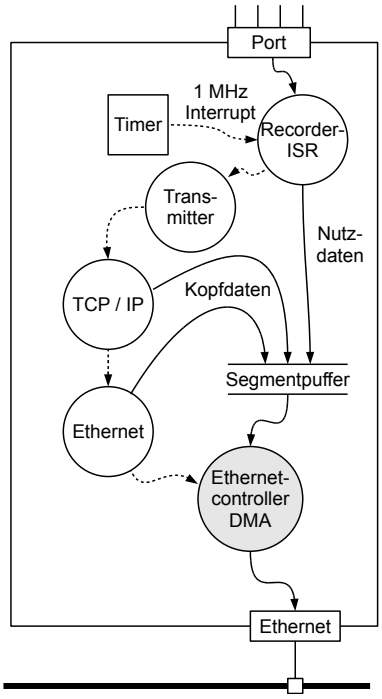


Abbildung 5: Soft- und Hardwarestruktur des Datenrekordes

Abbildung 5 zeigt die Struktur der Anwendung. Normalerweise würde man bei einer derartigen Anforderung die Daten per UDP versenden. Bei Verwendung von TCP muss der Empfänger die Bestätigung unbedingt innerhalb von 2 msec zurücksenden, ansonsten kommt es sofort zum Überlauf des Puffers. Das ist für übliche PC-Betriebssysteme eine harte Anforderung, die kaum zu erfüllen ist. Trotzdem hat sich gezeigt, dass auch bei längerem Betrieb der Anwendung nur sehr selten Datenverluste aufgetreten sind. Im Gegensatz zum UDP können solche Datenverluste durch den Bestätigungsmechanismus der TCP-Schicht

erkannt werden.

1.4 Zusammenfassung

In dieser Arbeit wurde der Aufbau einer Netzwerksoftware gezeigt, die unter Ausnutzung der besonderen Eigenschaften des zu Grunde liegenden Mikrocontrollers LPC2378 auf hohen Datendurchsatz und geringer CPU-Belastung optimiert wurde. Anhand einfacher Messungen wurde gezeigt, dass der erreichbare Durchsatz wesentlich durch die Verfügbarkeit von Speicher gegeben ist. Wesentlicher Einflussfaktor ist die Round-Trip-Time (RTTI). Bereits einige zusätzliche Switches oder Router, die im Prinzip nur eine geringe RTTI-Verlängerung bewirken, können die Datenrate deutlich reduzieren. Der Mikrocontroller selber ist in der Lage, eine Ethernetverbindung mit 100 MBit/s weitestgehend auszunutzen. Die hohe Performance wird dadurch erreicht, dass das Puffermanagement ausschließlich auf dem Speicher erfolgt, auf den der Ethernet-Controller direkt per DMA zugreifen kann. Durch den ereignisgesteuerten Aufbau der Software kann auf ein Betriebssystem verzichtet werden. Die diversen Serverimplementierungen zeigen, dass trotzdem eine einfache Programmierung der Anwendungen möglich ist.

1.5 Literatur

1. Interniche Technologies, Inc., Networking for Embedded Processors, http://www.iniche.com/.
2. Seventstax GmbH, http://www.sevenstax.de.
3. Adam Dunkels, Full TCP/IP for 8-bit architectures, Proceedings of the 1st international conference on Mobile systems, applications and services, San Francisco, California, 2003.
4. Stefan Diercks, Design und Implementierung einer Bibliothek für Webbrowser-basierte Bedienoberflächen von Mess- und Regelanwendungen auf Ethernet-fähigen Mikroprozessorsystemen, Diplomarbeit, Hochschule für Angewandte Wissenschaften, Hamburg, 2008.

Hans Heinrich Heitmann
Hochschule für Angewandte Wissenschaften, Hamburg
heitmann@informatik.haw-hamburg.de

2 Preisträger 2010

Die Fachgruppe Echtzeitsysteme veranstaltet seit 2007 einen Wettbewerb für Nachwuchsarbeiten im Bereich Echtzeitsysteme. Die Preisverleihung findet im Rahmen des diesjährigen Workshops „Echtzeit 2010“ statt.

Aus zahlreichen Einreichungen wurden die diesjährigen Preisträger ausgewählt:

Lino Schmid wurde für seine Masterthesis **„GPS Navigation on Ski Slopes“** an der ETH Zürich ausgezeichnet.

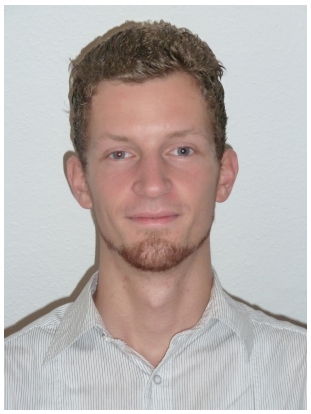
Abstract:
In this master thesis, we introduce RidersGuide, an autonomous navigation system for snow sports athletes. The system supports winter athletes during their ride on the ski slope. It consists of a mobile computer and a GPS receiver. By providing auditive and visual feedback to the athlete, RidersGuide leads skiers and snowboarders down the ski slope on a safe path. In addition, the system warns the athlete of potential hazards as the overrun of the slope edges, obstacles on the trail, slope crossings or steep areas. We present algorithms which calculate a safe route from a high-precision digital elevation model of the slope. We applied the model DHM25 with a mesh size of 25m and the model DOM with a mesh size of about 2 m. With the lower price model DHM25, we could create just as good routes as with DOM. The system was tested by 23 persons of various ages and different riding skills on the slope and in the backcountry. The tests were carried out in the Swiss ski resorts Savognin and Grindelwald. Afterwards, the test persons rated the system. On a scale of 1 to 10, where 10 is the top grade, the probands benchmarked the system with the grade of 9.25. The aural warnings of the safety- critical areas were assessed to be the most helpful. In foggy conditions, all participants would prefer to be on the way with rather than without RidersGuide.

Timon Kelter wurde für seine Diplomarbeit **„Superblock-basierte High-Level WCET-Optimierungen“** an der TU Dortmund ausgezeichnet.

Abstract:
Die Diplomarbeit stellt einen neuen Ansatz im Bereich der compilerbasierten Minimierung der WCET (*Worst Case Execution Time*) vor. Die WCET besitzt im Kontext sicherheitskritischer Echtzeitsysteme eine besonders hohe Bedeutung, da ihre Kenntnis für Echtzeit-Schedulingverfahren unabdingbar ist. Während zur Bestimmung der WCET bereits eine Vielzahl von Ansäzen bekannt ist, ist ihre Minimierung, insbesondere die automatische Minimierung in einem Compiler, bisher weniger stark untersucht worden.

Die Diplomarbeit üerträgt das aus der Compiler-Optimierung bekannte Konzept der Superblöcke auf die *High-Level*-Ebene, hier konkret am Beispiel der Sprache C, und wendet es dort zur Verringerung der WCET an. In Rahmen der Arbeit wurde die Superblockbildung selbst auf der *High-Level*-Ebene implementiert, zusammen mit einer *Superblock Common Subexpression Elimination* und einer *Superblock Dead Code Elimination*. Für die Superblockbildung wurden mehrere neue, an die ßem High-Level Darstellung und die WCET-Minimierung angepasste Strategien entwickelt: Die Pfade (*Traces*), die zu Superblöcken umgeformt werden, werden durch einen Graphalgorithmus auf Basis von WCET-Daten ausgewählt und unter Anwendung angepasster Techniken (*If-Else*-Neufaltung, *Switch*-Konvertierung) in die Superblockform überführt. Zur besseren Skalierbarkeit des vorgestellten Optimierungsverfahrens wurden in der Arbeit auch mehrere Hilfsmittel, wie z.B. die schnelle Neuberechnung der WCET-Informationen und eine *Def/Use*-Set Analyse entwickelt.

Die Superblock-Optimierungen wurden in den am Informatik-Lehrstuhl 12 der TU Dortmund entwickelten WCC (WCET-aware C Compiler) integriert und intensiv mit 60 Benchmarks überprüft.



Martin Kulas erhielt die Prämierung für seine Diplomarbeit **„Entwurf und Realisierung eines Adapters für UniLoG zur Lastgenerierung an IP-basierten Schnittstellen“** an der Universität Hamburg.

Abstract:
Im Rahmen dieser Arbeit wurde ein spezifischer Adapter für die IPv4(Internet Protocol IP, Version 4)-Schnittstelle des Internet konzipiert, prototypisch realisiert und validiert, der es ermöglicht, reale Auftragssequenzen an einer IP-Schnittstelle mithilfe des bereits existierenden UniLoG-Lastgenerators zu erzeugen. Wichtige Teilaufgaben des IP-Adapters betrafen dabei zum einen die Abbildung von zunächst abstrakt in einem Lastmodell (auf Basis sog. Benutzerverhaltensautomaten) spezifizierten Aufträgen auf die entsprechenden realen IP-Aufträge incl. der Auftragsübergabe an der IP-Schnittstelle sowie zum andern die Erfassung beobachtbarer Systemreaktionen und Rückmeldungen an der IP-Schnittstelle und deren anschließende Weiterverarbeitung im Lastgenerator. Bei der Implementierung waren überdies anspruchsvolle Echtzeitanforderungen an den Adapter zu berücksichtigen. Die echtzeitfähige Version von UniLoG wurde auch hinsichtlich des Grades ihrer erreichten Realitätsnähe beurteilt, d.h. es wurde insbesondere überprüft, ob und in wieweit die gestellten Echtzeitanforderungen bei unterschiedlichen Lastniveaus auch noch tatsächlich hinreichend gut erfüllt werden können.