h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt

– Fachbereich Informatik –

Konzeptionierung und prototypische Umsetzung einer Echtzeitbasierten Restbussimulation mittels Mikrocontrollerunterstützung

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

vorgelegt von

Sven Eisenhauer

Referent	Prof. Dr. Joachim Wietzke
Korreferent	Prof. Dr. Ralf S. Mayer

Ausgabedatum	14.08.2011
Abgabedatum	14.02.2012

Zusammenfassung

Eingebettete Systeme nehmen einen immer größeren Stellenwert in allen Bereichen unseres täglichen Lebens ~~ein. So~~ auch im Automobilbereich. Die zunehmende Leistungsfähigkeit eingebetteter Systeme beflügelt hier die Entwicklung immer komplexerer Anzeige- und Assistenzsysteme. Diese Systeme bestehen aus leistungsfähiger Hardware und äußerst komplexer Software. Bei bestimmten Systemen handelt es sich sogar um sicherheitsrelevante Funktionen. Daraus ergeben sich sehr hohe Qualitätsanforderungen an die Software. Aktuell sollen sehr gut definierte Entwicklungsprozesse und ausgiebige Softwaretests dazu beitragen, diese Anforderungen zu erfüllen. Diese eingebetteten Systeme arbeiten dabei nicht autark, sondern benötigen oftmals Daten von anderen Systemen im Fahrzeug. Während der Entwicklungs- und Testphasen besteht häufig das Problem, dass die Steuergeräte, von denen das aktuell zu entwickelnde System abhängt, ~~noch nicht bestehen oder~~ nicht verfügbar sind. Im Rahmen dieser Entwicklungs- und Testmaßnahmen einzelner Softwarekomponenten finden deshalb sehr oft sog. Restbussimulationen Verwendung, die bestimmte Komponenten eines Fahrzeugs, die aktuell physisch nicht verfügbar sind, simulieren. So lässt sich eine einzelne Komponente entwickeln und prüfen, ohne ein komplettes Fahrzeug zu benötigen.

Diese Arbeit beschreibt ~~nun~~ eine Lösung, wie eine speziell dafür entwickelte Hardware bestimmte Teile einer solchen Restbussimulation ausführt. Die Definition und Steuerung dieser Simulation erfolgt dabei durch den Benutzer auf seinem ~~bestehenden~~ Personalcomputer. Die Arbeit beschreibt ein Konzept zur Verteilung der notwendigen Softwarekomponenten und stellt eine prototypische Umsetzung vor. Konzept und Umsetzung orientieren sich dabei an den spezifischen Eigenschaften der beteiligten Systeme. Weiterhin berücksichtigen sie besondere Anforderungen der Systeme an Konzept und Implementierung. Abschließend zeigt die Arbeit anhand der Auswertung der Ergebnisse des Prototyps die Vorteile einer solchen Verteilung auf.

Abstract

English abstract here

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 4. Dezember 2011 *Sven Eisenhauer*

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CAN	Controller Area Network
CPU	Central Processing Unit
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution
ELF	Executable and Linkable Format
GUI	Graphical User Interface
LIN	Local Interconnect Network
MFC	Microsoft Foundation Classes
MOST	Media Oriented Systems Transport
PC	Personal Computer
RAII	Resource Acquisition Is Initialization

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagung

Danke an alle

Kapitel 1.

Einleitung

> Ein Text ist nicht dann vollkommen, wenn man nichts mehr hinzufügen kann, sondern dann, wenn man nichts mehr weglassen kann.
>
> *(Antoine de Saint-Exupery)*

Dieses Kapitel beschreibt zuerst die Motivation zur Erstellung dieser Arbeit. Der anschließende Abschnitt formuliert die Problemstellung, um die sich die Arbeit dreht. Abschließend liefert dieses Kapitel eine Übersicht über die nachfolgenden Kapitel und deren Inhalt.

1.1. Motivation

Komplexe Software im Auto Vernetzte Steuergeräte Große Datenmenge

Hoher Entwicklungs- und Testaufwand Analyse [SSH11]

Problem der isolierten Entwicklung einzelner Komponenten, die aber von anderen, nicht verfügbaren, Komponenten abhängen. Das Verhalten dieser Komponenten ist bekannt, womit sie sich simulieren lassen.

Zur Simulation vernetzter Steuergerät hat die Schleißheimer GmbH die Software CanEasy entwickelt. Dabei handelt es sich um eine Software für Personalcomputer mit Betriebssystem Windows. Die Bedienung der Software erfolgt über eine grafischer Benutzerschnittstelle zur Definition der Busse mit Steuergeräten. Jedes Steuergerät versendet einzelne Nachrichten mit bestimmten Daten, sog. Signalen. CanEasy versendet dabei Nachrichten von simulierten Steuergeräten und zeichnet Nachrichten von real existierenden Steuergeräten auf. Bei Tests und der Fehlersuche ist die zeitliche Abfolge von bestimmten Ereignissen auf dem Bus besonders wichtig, um bestimmte Situationen her zu stellen. Zur Kommunikation mit den realen Steuergeräten benötigt CanEasy immer separate Hardware für die verwendete Bus-Technologie.

Die Firma x2e stellt die Hardware XORAYA Connect, im Folgenden kurz Xoraya genannt, her, die über vielfältige Hardwareschnittstellen für alle gängigen Bus-Technologien verfügt. x2e stellt dem Benutzer eine Umgebung zur Entwicklung eigener Software für die Xoraya bereit. Allerdings verfügt die Xoraya über keine grafische Benutzerschnittstelle.

1.2. Problemstellung und Zielsetzung

Diese Arbeit befasst sich mit der Integration von CanEasy und der Xoraya. Diese Integration soll die komfortable Benutzung von CanEasy mit der Leistungsfähigkeit der Xoraya verbinden. Als Synergie soll ein komfortabel zu bedienendes und leistungsfähiges Gesamtsystem entstehen. Da der CAN-Bus heute und in naher Zukunft die Bus-Technologie zur Anbindung von Kombiinstrumenten ist und die Schleißheimer GmbH hauptsächlich Software für Kombiinstrumente entwickelt, liegt der Fokus der Arbeit auf der Nutzung der CAN-Schnittstellen der Xoraya.

1.3. Übersicht

Kapitel 2 beschreibt CanEasy und Xoraya detailliert und geht auf die spezifischen Eigenschaften beider Komponenten ein. Ebenso stellt es die verwendeten Bus-Technologien vor.
Kapitel 3 entwickelt ein Konzept zur Erreichung des Ziels. Es berücksichtigt dabei die spezifischen Eigenschaften der jeweiligen Systeme. An manchen Stellen bieten sich mehrere Alternativen zur Lösung einer konzeptionellen Fragestellung. Hier erfolgt eine Abwägung der Optionen, falls sich auf konzeptioneller Ebene schon eine bestimmte Entscheidung begründen lässt. Andernfalls verlagert sich die Entscheidung in spätere Kapitel. Weiterhin stellt Kapitel 3 ein Konzept zur Bewertung der Ergebnisse der prototypischen Implementierung vor.
Darauf erfolgt in Kapitel 4 eine detaillierte Beschreibung der einzelnen Softwarekomponenten, die für die verschiedenen Systeme im Rahmen der prototypischen Implementierung entwickelt wurden. Neben einer Beschreibung des Gesamtsystems geht dieses Kapitel auf bestimmte Details der Implementierung ein. Es läutert deren besondere Bedeutung für das Gesamtsystem und verdeutlicht die Design- und Implementierungsentscheidungen.
Kapitel 5 bietet eine Auswertung der Ergebnisse des in Kapitel 4 beschriebenen Prototyps.
Kapitel 6 fasst die Erkenntnisse der Arbeit zusammen und liefert eine kritische Betrachtung. Es analysiert, welche Teile der Zielsetzung die Arbeit erreichen konnte und welche nicht. Es listet weiterhin Fragen auf, die im Rahmen dieser Arbeit nicht geklärt werden konnten oder sich im Verlauf der Arbeit ergaben. Abschließend betrachtet es verschiedene Richtungen, in die basierend auf dieser Arbeit in Zukunft weiter gearbeitet werden kann.

Kapitel 2.

Grundlagen

Der echte Schüler lernt aus dem Bekannten das Unbekannte entwickeln und nähert sich dem Meister.

(Johann Wolfgang von Goethe)

2.1. CanEasy

2.1.1. Datenmodell - Die Datenbasis

Busse
Steuergeräte
Nachrichten
Signale

2.1.2. GUI

Datenbasis in Baumstruktur
Plots zur Analyse des zeitlichen Verlaufs [SSH11] von Signalwerten
Traces

2.1.3. Plugin-Konzept

Flexible Erweiterungsmöglichkeiten

2.2. XORAYA Connect

2.2.1. CPU

Powerpc
500MHz
256 MB RAM

2.2.2. Hardware-Schnittstellen

Ethernet
CAN (Highspeed und Lowspeed)
LIN
FlexRay
MOST25
MOST150

2.2.3. SDK

Zugriff auf spezielle Hardware
GNU cross-toolchain unter MS Windows

2.3. Echtzeit, Windows vs. realtime Linux

Definition: Vorhersagbarkeit des Systems

2.4. Scheduling

Pseudo-Parallelität
Eine Betriebssystemkomponente, der Task-Scheduler, entscheidet, welche Task als nächstes die CPU verwenden darf. Diese Entscheidung hängt maßgeblich vom Scheduling-Verfahren, der Priorität der Tasks und den Zuständen der Tasks ab.
Taskzustände nach [Tan92]

Running Genaue eine Task pro CPU befindet sich im Zustand Running. Sie führt gerade ihren Code auf der CPU aus.

Ready Eine Task in diesem Zustand ist bereit ihre Anweisungen auf der CPU auszuführen und wartet auf Zuteilung

Blocked In diesem Zustand wartet eine Task auf ein Ereignis, z. B. Abschluss eine IO-Operation oder Freigabe einer Ressource wie z. B. eines Mutex.

2.4.1. Zeitscheiben

Task mit höherer Priorität bekommt CPU, wenn die Zeitscheibe der vorigen Task abgelaufen ist oder die vorige Task die CPU abgibt.

2.4.2. Preemption

Höher-priore Task verdrängt sofort die laufende nieder-priore Task, sobald die höher-priore Task in den Zustand Ready geht.

2.5. Ethernet

CSMA/CD
Nicht geeignet für Echtzeit

2.6. CAN

Nach [Law97] ...
CSMA/CR
CSMA/CA
Arbitrierung, Priorität

Kapitel 3.

Konzept

> Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT.
>
> *(Linus Torvalds)*

3.1. Architektur des Gesamtsystems

3.1.1. Anforderungen

Xoraya soll autark z. B. im Fahrzeug arbeiten können. Deshalb muss die Simulation ohne Verbindung zu CanEasy laufen können.

3.1.2. Verteilung der Komponenten auf die Systeme

Bild der Gesamtarchitektur
GUI für Definition und Anzeige auf dem PC.
Senden und Empfangen autark auf dem Target.

3.2. Konzept zur Übertragung der Datenbasis vom Host auf das Target

3.2.1. Alternativen

Als textbasierte Beschreibung z. B. in XML, Nachteil: Parsen auf Target sehr aufwändig.
Binär, Vorteil: Kein Parsen auf dem Target notwendig, direkter Zugriff auf die Daten möglich.

3.2.2. Cross compilation

3.2.3. Dynamic loading and linking

[Lev99]

3.3. Targetseitige Integration

3.3.1. Komponenten

Zentrale Steuerung
Netzwerkkommunikation
Sendesteuerung
Aufzeichnung

3.3.2. Konzept der Target-Plugins

3.3.3. Threading

Nur ein CPU-Kern
Deshalb möglichst wenige Threads, da Threadwechsel vergleichsweise lange dauern.
Hardwarezugriffe auf Bus-Schnittstellen nicht blockierend durch Sende- und Empfangspuffer.

3.3.4. Steuerung

Ein Thread
Zentrale Steuerung: Start, Stop etc.
Sendesteuerung für Nachrichten unter Berücksichtigunng ihrer Priorität

3.3.5. Aufzeichnung

Auslesen der Empfangspuffer im Thread der Steuerung
Eigener Thread zum Beschreiben der Logdatei auf die Festplatte
Synchronisationspunkte

3.3.6. Netzwerk

Eigener Thread
Synchronisationspunkte

3.4. Hostseitige Integration

3.4.1. CanEasy-API

3.4.2. Code-Generierung

3.4.3. Anzeige des Logs

3.5. Verifikation

Atmel AT32UC3C-EK verschickt CAN-Botschaften mit definierter Zykluszeit auf Basis eines Hardware-Timers. Bietet so eine möglichst genaue Referenz zur Verifikation der Genauigkeit der Empfangszeitstempel auf der Gegenseite. Je geringer die Abweichung der Differenz der Empfangszeitstempel von der definierten Zykluszeit ist, desto genauer sind die Empfangszeitstempel der Gegenseite.

Kapitel 4.

Prototypische Implementierung

> Es ist nicht genug, zu wissen, man muß auch anwenden; es ist nicht genug zu wollen, man muß auch tun.
>
> *(Johann Wolfgang von Goethe)*

4.1. Implementierungsgrundsätze

Keine Verwendung von Heap-Speicher bei embedded systems
boost-Bibliotheken zur Netzwerkkommunikation, Smart Pointer, Intrusive Container zur Vermeidung der impliziten Speicherallokation in STL-Containern
Memory locking: Verhindert das auslagern von Speicherkacheln
Stack pre-faulting
Echtzeit-Prio thread
Priority inversion Problem [Ree97]
Lösung: Priority Inheritance [SRL90], implementiert durch x2e::Mutex bzw. PTHREAD_PRIO_INHERIT
RAII eines Mutex durch boost::lock_guard
Keine shared ressources halten, wenn IO operation.

4.2. Xoraya

4.2.1. Plugin-API

Beschreibung des Interfaces, das Plugins implementieren müssen.
Plugin API

4.2.2. Steuerung

boost intrusive set zur impliziten Beachtung der Priorität der Nachrichten

4.2.3. Aufzeichnung

boost circular buffer zwischen Auslesen der Hardware-Message-Queue im Hauptthread und dem Schreiben der Log-Datei im Nebenthread mit niedriger Priorität

4.2.4. Netzwerk

4.3. CanEasy

4.3.1. GUI

4.3.2. Code-Generierung, Cross-Compiler

Vorteil von Initializer-Listen nach C++x0: Keine Heap-Allokation und alle Nachrichten direkt von Sendesteuerung zugreifbar

4.3.3. Logconverter

4.4. Verifikationswerkzeug Atmel

4.4.1. main loop

4.4.2. timer interrupt

Kapitel 5.

Ergebnisse

> Ein jeder lernt nur, was er lernen kann;
> Doch der den Augenblick ergreift, das
> ist der rechte Mann.
>
> *(Johann Wolfgang von Goethe, Faust I)*

5.1. Genauigkeit der Zeitstempel beim Empfang

Ansatz: Versand einer CAN-Nachricht mit dem Atmel Board in einem genau definierten Zeitintervall. Vergleich der Abweichung der relativen Empfangszeitstempel vom Sendeintervall. Abweichung klein bedeutet hohe Genauigkeit.

5.1.1. CanEasy mit Softwarezeitstempel

5.1.2. CanEasy mit Hardwarezeitstempel

5.1.3. XORAYA Connect

5.2. Genauigkeit der Sendesteuerung der XORAYA Connect

Verifikation mit CanEasy mit Hardwarezeitstempel.

5.3. Verhalten der XORAYA Connect unter hoher Prozessorlast

Ansatz: Erzeugung von hoher CPU-Last und Analyse der TX-Genauigkeit bei unterschiedlichen Prioritäten des Steuerthreads

5.3.1. Steuerthread auf RT-Priorität

RX- und TX-Genauigkeit der XORAYA Connectwenn der Steuerthread auf höchster Priorität mit FIFO-Scheduler läuft.

5.3.2. Steuerthread auf normaler Priorität

RX- und TX-Genauigkeit der XORAYA Connectwenn der Steuerthread auf normaler Priorität mit normalem Scheduler läuft.

5.4. Bewertung der Ergebnisse

RX-Zeitstempel sehr genau auf der XORAYA Connect
Hohe TX-Genauigkeit durch Hardware-Timer
RT-Linux ermöglicht hohe zeitliche RX- und TX-Genauigkeit auch unter hoher Prozessorlast durch andere Anwendungen

Kapitel 6.

Zusammenfassung und Ausblick

Viele Wege führen zum Gipfel eines
Berges,
doch die Aussicht bleibt die gleiche.

(Chinesische Weisheit)

Konzept für Zielerreichung erstellt
Proof Of Concept mit Prototyp belegt
Zeitlich genaues Versenden und Empfangen von CAN-Botschfaten
XORAYA Connectautark einsetzbar. Sowohl als Logger zur späteren Analyse in CanEasy als auch als autarke Sendeeinheit.
Ausblick: Persistenz von geänderten Botschaftsdaten in CanEasy auf der XORAYA Connect.
Verlagerung weiterer CanEasy-Funkionen wie z. B. erweiterte Sendesteuerung oder Formel-Plugin auf die XORAYA Connect.
Anbindung der weiteren Hardware-Schnittstellen der XORAYA Connectan CanEasy.

Literaturverzeichnis

[Bra10] Brands, G.: *Das C++ Kompendium*. 2. Auflage. Springer, 2010 (eXamen.press Series). – ISBN 9783642047862

[But11] Buttazzo, G. C. ; Stankovic, J. A. (Hrsg.): *Hard Real-Time Computing Systems*. Third Edition. Springer, 2011 (Real-Time Systems Series)

[Law97] Lawrenz, W.: *CAN Controller Area Network.: Grundlagen und Praxis*. Hüthig, 1997 http://books.google.com/books?id=_kFSSQAACAAJ. – ISBN 9783778525753

[Lev99] Levine, John R.: *Linkers and Loaders*. 1st. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999. – ISBN 1558604960

[Lov07] Love, R.: *Linux System Programming*. O'Reilly Media, 2007. – ISBN 9780596009588

[Ree97] Reeves, G. E.: *What really happened on Mars ?* Website. http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Authoritative_Account.html. Version: December 1997

[Sch11a] Schick, F.: *XORAYA Connect Embedded-SDK reference*. Website. http://dev.xoraya.com/embedded/index.html. Version: November 2011. – Last visited: 03.12.2012

[Sch11b] Schick, F.: *XORAYA Connect Wiki*. Website. http://wiki.kandel.x2e.de/doku.php?id=connect:public:start. Version: November 2011. – Last visited: 03.12.2012

[SRL90] Sha, L. ; Rajkumar, R. ; Lehoczky, J. P.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization. In: *IEEE TRANSACTIONS ON COMPUTERS* 39 (1990), SEPTEMBER, Nr. 9, S. 1175–1185

[SSH11] Sedlmair, M. ; Schraut, M. ; Hintermaier, W.: Visualisierung von Busdaten. In: *AUTOMOBIL ELEKTRONIK* (2011), Oktober, Nr. 4, S. 21–23

[Tan92] Tanenbaum, A. S.: *Modern Operating Systems*. Prentice-Hall, 1992

[TIS95] *Executable and Linkable Format ELF*. Tool Interface Standards TIS, 1995 (Portable Formats Specification, Version 1.1)

[WT05] Wietzke, J. ; Tran, M.T.: *Automotive Embedded Systeme: Effizientes Framework-vom Design zur Implementierung*. Springer, 2005 (Xpert. press Series). http://books.google.com/books?id=bUfQ0h8FgPoC. – ISBN 9783540243397

Stichwortverzeichnis

Anhang A.

Listings

A.1. Plugin API

```
/*
 * plugin_api.h
 *
 *  Created on: 31.08.2011
 *      Author: Eisenhauer
 */

#ifndef PLUGIN_API_H_
#define PLUGIN_API_H_

#include "global.h"

struct tstCanTxMessage
{
  uint32_t  u32TxCycleMilliseconds;
  uint32_t  u32MillisecondCounter;
  uint32_t  u32CanId;
  uint8_t   au8Data[8];
  uint8_t   u8Dlc;
};
typedef tstCanTxMessage* tpstCanTxMessage;

class IPlugin
{
public:
  virtual void vRun( void ) = 0;
  virtual tpstCanTxMessage pxGetCanTxMessage( uint32_t u32MsgIndex ) = 0;
  virtual uint32_t u32GetNumOfCanTxMessages( void ) = 0;
  virtual int32_t i32GetCanInterfaceHandle( void ) = 0;
  virtual bool boAutoload( void ) = 0;
  virtual bool bolsLogger( void ) = 0;
  virtual tenRetCodes enGetLog(void*) = 0;
  virtual void vInit(void* pxIfMan, int32_t i32Interface, uint32_t
      u32PluginId) = 0;
};
```

```
35
typedef IPlugin* tpxCreatePlugin(
37      void* pxIfMan,
        int32_t i32Interface,
39      uint32_t u32PluginId
        );
41 typedef void tvDestroyPlugin(IPlugin*);

43 extern "C"
   {
45   tpxCreatePlugin pxCreatePlugin;
     tvDestroyPlugin vDestroyPlugin;
47 }

49 #endif /* PLUGIN_API_H_ */
```

../../src/common/inc/plugin_api.h

Anhang B.

Inhalt der beiliegenden CD-ROM

In den Unterverzeichnissen der CD-ROM befinden sich

src Quellcode des realisierten Prototypen

www Abzüge der verwendeten Webseiten im PDF-Format

thesis Diese Arbeit im PDF-Format