Entwicklung und Optimierung von Display-Schnittstellen fuer embedded Linux Boards

-

Masterarbeit

im Fachgebiet Hard- und Softwareentwicklung

vorgelegt von:	Armin Schlegel
Studienbereich:	Fakultaet EFI
Matrikelnummer:	2020863
Erstgutachter:	Prof. Dr. Joerg Arndt
Zweitgutachter:	Peter Meier

Zusammenfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Quisque Natural-Programmierung blandit sed, hendrerit at, pharetra eget, dui. Sed lacus. Pellentesque malesuada. Cras gravida mi id sapien. Ut risus justo, fermentum non, scelerisque sit amet, lacinia in, erat. Proin nec lorem. Quisque porta, nisl at porta aliquam, felis libero consequat ipsum, vitae scelerisque dolor mi a odio. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis sollicitudin. Proin sollicitudin varius arcu. Morbi eleifend, metus sit amet placerat pharetra, dolor dui lobortis pede, vel imperdiet tellus eros imperdiet lorem. In hac habitasse platea dictumst. Curabitur elit mi, facilisis nec, ultricies id, aliquet et, magna. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam ac est. Mauris turpis enim, feugiat non, imperdiet congue, scelerisque non, purus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nullam dictum aliquet purus. Maecenas faucibus. Maecenas suscipit.

Abstract

Fusce neque est, tincidunt eu, nonummy nec, tempor iaculis, erat. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vestibulum egestas, velit a rhoncus gravida, metus dolor pulvinar diam, sit amet placerat risus dolor sit amet elit. Maecenas eget purus ut est mattis porta. Suspendisse ut mi et mauris lobortis malesuada. Vestibulum dapibus. Duis hendrerit, elit eu venenatis eleifend, sapien ante volutpat odio, ac condimentum tellus massa ut massa. Etiam dapibus imperdiet metus. Sed sapien arcu, pulvinar quis, laoreet quis, venenatis non, justo. Aliquam est ante, pulvinar nec, accumsan sed, auctor sed, augue.

Ut adipiscing ligula. In mattis. Ut varius. In nec nulla at eros molestie viverra. Duis dolor risus, lobortis vel, dictum a, pellentesque id, lectus. Sed suscipit orci ac ligula venenatis condimentum. Maecenas et sem lacinia tortor cursus tempus. Mauris pellentesque risus at nulla. In arcu. Curabitur mattis mi quis dolor. In leo. Vivamus ut libero.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

1.1. Motivation

In der heutigen Zeit treten eingebettete Systeme (engl. embedded systems) immer stärker in den Vordergrund. Gerade in den Bereichen der Industrie, Telekommunikation oder Multimedia wächst der Bedarf an Lösungen die durch Zuverlässigkeit, Energiesparsamkeit und kompakter Bauform bestechen.
Obwohl eingebettete Systeme meist für den Anwender unsichtbar ihren Dienst verrichten, sind sie doch inzwischen allgegenw"rtig. Im Bereich der Telekomunikation und Unterhaltungselektronik kommt ein solches System im prinzip nicht mehr ohne ein Display aus. Die Moeglichkeit zur Anzeige multimedialer Daten wird zur Kaufentscheidung. Auch hier gilt die Maxime: besser, schneller, groesser.
Im Sektor der eingebetten Systeme mit Betriebssystem spielt Linux neben diversen anderen Systemen wie beispielsweise RTOS, OSEK, QNX oder auch Windows eine sehr grosse Rolle. In Verbindung zeigen eingebettete Linuxsysteme mit Displays ein grosses Potential. Mit der beliebten ARM-Architektur lassen sich so kostenguenstige, leisttungsstarke Systeme aufbauen, die die gestellten Aufgaben gut erfuellen kann.

1.2. Ziel der Arbeit

Das Ziel dieser Arbeit ist zu zeigen, dass die Verwendung von Displays mit eingebetten Linux Systemen je nach Anforderung einfach oder ueber Umwege realisierbar ist.

1.3. Aufbau der Arbeit

Im ersten Teil der Arbeit werden theoretische Grundlagen gebildet, die fuer das Verstaendnis notig sind. Hier werden Standards wie z.B. HDMI bzw. DVI, LVDS und RGB behandelt. Es wird ein Ueberblick ueber ausgewaehlte embedded Linux Boards gegeben und diese Klassifiziert mit welchen Displayschnittstellen diese ausgestattet sind bzw. ausgestattet werden koennen. Der Zweite Teil behandelt das embedded

Linux Board 'Gnublin', welches von Haus aus keine Displayschnittstelle vorgesehen hat. Hier werden zwei Varianten zur Ansteuerung von Displays erarbeitet. Die Ansteuerung wird hierbei vom Prozessor erledigt, da das 'Gnublin' keine dedizierten Grafikcontroller besitzt. Im dritten Teil wird fuer leistungsstaerkere embedded Linux-Systeme mit HDMI-Schnittstelle eine Hardware entwickelt, RGB- oder LVDS-Panels anzuschliessen. Um die Displys ueber die entwickelte Hardware anzusteuern, wird der dedizierte Grafikcontroller der Boards verwendet.

1.4. Typographische Konventionen

2. Theoretische Grundlagen

2.1. Embedded Linux Boards

2.2. Video-Schnittstellen

2.2.1. DVI

2.2.2. HDMI

2.2.3. VGA

2.2.4. 8080-Interface

2.2.5. TV-OUT

3. Teil A

3.1. Display mit 8080-Interface

3.2. 8080-Interface mittels GPIO-Pins

3.2.1. Konzept

3.2.2. Hardwareverbindung zwischen GPIO-Pins und Display

3.2.3. User-Space-Treiber

3.2.3.1. Low-Level-Treiber

3.2.3.1.1. GPIO-Pin Frequenz erhoehen

3.2.3.1.2. GPIO-Treiber

3.2.3.1.3. Displaytreiber fuer SSD1963

3.2.4. Ansteuerung des Displays

3.3. 8080-Interface mittels SRAM-Interface

3.3.1. Konzept

3.3.2. MPMC - Multiport Memory Controller des NXP LPC313x

3.3.3. Hardwareverbindung zwischen SRAM-Interface und Display (Adapterplatine)

3.3.4. Software

3.3.4.1. Entwicklung eines Linux-Framebuffer-Treibers

3.3.4.2. Entwicklung eines User-Space-Treibers

3.3.4.3. ANpassung des APEX-Bootloaders zur Verwendung des Displays

3.3.4.4. Probleme bei der Entwicklung und Fehlersuche

3.3.4.4.1. Rolle des User-Space-Treibers

3.3.4.4.2. Debuggen mit Logik-Analyzer

3.3.4.4.3. Loesung des Problems

3.4. Vor- und Nachteile

4. Teil B

4.1. Konzept

4.2. Hardwareentwicklung

4.2.1. Spannungsversorgung

4.2.2. HDMI-RGB-Bridge

4.2.3. RGB-LVDS-Bridge

4.2.4. EDID-Daten

4.3. Software

4.3.1. EDID-Daten auf embedded Seite

4.3.1.1. Konzept

4.3.1.2. Low-Level-Treiber

4.3.1.2.1. UART-Treiber

4.3.1.2.2. I2C-Treiber

4.3.1.3. Programmablauf

4.3.2. EDID-Daten auf PC Seite

4.3.2.1. Konzept

4.3.2.2. GTK GUI mit Glade

4.3.2.3. Programmablauf

5. Zusammenfassung

Eidesstattliche Erklärung

Ich, Armin Schlegel, Matrikel-Nr. 2020863, versichere hiermit, dass ich meine Masterarbeit mit dem Thema

> *Entwicklung und Optimierung von Display-Schnittstellen fuer embedded Linux Boards -*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Masterarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Prüfungsamt der Ohm-Hochschule abzugeben oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Nuernberg, den 14. April 2014

Armin Schlegel

A. Anhang

A.1. Liste möglicher Aktivitäten der BPEL

Die folgenden Listen basieren auf [?, Abschnitt 10 u. 11].

Tabelle A.1.: Ausgewählte elementare BPEL-Aktivitäten

Aktivität	Beschreibung	Beispiel
invoke	Aufruf einer Operation eines Webservice. Dabei wird zwischen *One-way*- und *Request-response*-Kommunikation unterschieden. Eventuelle Input- und Output-Nachrichten werden hierbei angegeben. invoke-Elemente können weitere Elemente (wie z. B. faultHandler zur Fehlerbehandlung) beinhalten.	`<invoke partnerLink="PLName" portType="PTName" operation="OName" inputVariable="VarName" outputVariable="VarName">`
receive	Empfängt eine Nachricht von einem Partner. Dazu muss die Operation angegeben werden, die der Prozess anbietet um die Nachricht entgegenzunehmen. Die Nachricht kann in einer variable gespeichert werden.	`<receive partnerLink="PLName" portType="PTName" operation="OName" variable="VarName">`
reply	Antwortet auf die Nachricht eines Partners (nur sinnvoll bei *Request-Response*-Kommunikation).	`<reply partnerLink="PLName" portType="PTName" operation="OName" variable="VarName">`
assign	Zuweisung von Werten zu Variablen.	`<assign> <copy> <from> <literal> <![CDATA[Wert]]> </literal> </from> <to variable="myVar" /> </copy> </assign>`
throw	Signalisierung eines Fehlers (analog zu Programmiersprachen).	`<throw faultName="FName" faultVariable="VarName">`

wait	Lässt den Prozess eine gewisse Zeit lang warten.	`<wait>`
		`  <until>`
		`  '2002-12-24T18:00'`
		`  </until>`
		`</wait>`

exit	Beendet den Prozess sofort.	`<exit>`

Tabelle A.2.: Ausgewählte strukturierte BPEL-Aktivitäten

Aktivität	**Beschreibung**	**Beispiel**

sequence	Sequentielles Abarbeiten der angegebenen Aktivitäten.	`<sequence>`
		`  <invoke>...</invoke>`
		`  <invoke>...</invoke>`
		`</sequence>`

flow	Paralleles Abarbeiten der angegebenen Aktivitäten.	`<flow>`
		`  <invoke>...</invoke>`
		`  <invoke>...</invoke>`
		`</flow>`

if	Konditionale Abfragen (vergleichbar zur Programmierung).	`<if>`
		`  <condition>`
		`  ...`
		`  </condition>`
		`  <sequence>`
		`  ...`
		`  </sequence>`
		`  <elseif>`
		`  ...`
		`  </elseif>`
		`  <else>`
		`  ...`
		`  </else>`
		`</if>`

while	Wiederholung der angegebenen Aktivitäten (vergleichbar zur Programmierung).	`<while>`
		`  <condition>`
		`  $orderDetails > 100`
		`  </condition>`
		`  <scope>...</scope>`
		`</while>`

scope	Verändern des Kontextes in dem die angegebenen Aktivitäten ablaufen. So können z. B. neue Variablen deklariert oder eine andere Fehlerbehandlung definiert werden. Das scope-Element stellt eigentlich keine Aktivität dar, soll hier aber trotzdem erwähnt werden.	`<scope>`
		`  <faultHandlers>`
		`  ...`
		`  </faultHandlers>`
		`  <flow>`
		`  <invoke>...</invoke>`
		`  </flow>`
		`</scope>`