

Lehrveranstaltung

GSP - Grundlagen der Systemprogrammierung

Version: 2 | Letzte Änderung: 16.09.2019 10:26 | Entwurf: 0 | Status: vom verantwortlichen Dozent freigegeben

^ Allgemeine Informationen

Langname	Grundlagen der Systemprogrammierung
Anerkennende LModule	<u>GSP BaTIN</u>
Verantwortlich	Prof. Dr. Lothar Thieling Professor Fakultät IME
Niveau	Bachelor
Semester im Jahr	Sommersemester
Dauer	Semester
Stunden im Selbststudium	60
ECTS	5
Dozenten	Prof. Dr. Lothar Thieling Professor Fakultät IME
Voraussetzungen	grundlegende prozedurale Programmierkenntnisse Grundlegende Funktionsweise eines Von-Neumann-Rechners Grundlagen der Digitaltechnik Automaten u. Zustandsüberföhrungsdiagramme
Unterrichtssprache	deutsch
separate Abschlussprüfung	Ja

Abschlussprüfung

Details

Die Studierenden sollen in einer schriftlichen Klausur folgende Kompetenzen nachweisen: 1.) Sicherer Umgang mit grundlegenden Begrifflichkeiten, Mechanismen und Konzepten. 2.) Programmierung unter C. 3.) Entwicklung von einfachen Hardwaretreibern. 4.) Entwicklung von Problemlösungen aus dem Bereich Messen-Steuern-Regeln unter Verwendung eines Mikrocontrollers nebst Echtzeitbetriebssystem.

Mindeststandard

Mindestens 50% der möglichen Gesamtpunktzahl.

Prüfungstyp

Die Studierenden sollen in einer schriftlichen Klausur folgende Kompetenzen nachweisen: 1.) Sicherer Umgang mit grundlegenden Begrifflichkeiten, Mechanismen und Konzepten. 2.) Programmierung unter C. 3.) Entwicklung von einfachen Hardwaretreibern. 4.) Entwicklung von Problemlösungen aus dem Bereich Messen-Steuern-Regeln unter Verwendung eines Mikrocontrollers nebst Echtzeitbetriebssystem.

^ Vorlesung / Übungen

Lernziele

Kenntnisse

Grundlagen der C-Programmierung

Konstanten, Variablen, Datentypen

Ausdrücke, Anweisungen, Kontrollstrukturen

Präprozessoranweisungen

Zeiger und Zeigerarithmetik

Strukturierte Datentypen (Felder, Strukturen)

Funktionen

Standardbibliotheken

Aufbau von Mehrdateienprogrammen mit Zugriff auf Bibliotheken

Software-Entwicklungsumgebung

Compiler

Linker

Debugger

Simulator

hardwarenahe I/O-Programmierung in C

Aufbau digitaler I/O-Ports

Erreichbarkeit von I/O-Ports

Memory-Mapped-I/O

seperater I/O-Adressbereich

Zugriff auf I/O-Ports mittels Zeiger

Zugriff auf I/O-Ports mittels Treiberbibliotheken

Implementierung von Treiberbibliotheken in C

Bitbasierte Ein-Ausgabe und Auswertung von Daten mittels C

Programmierung von Aufgaben des Messens, Steuerns und Regeln in C

Realisierung von Moore- und Mealy-Automaten in C

Optimierung von zyklischen Zugriffen auf I/O-Daten

Aufbau und Funktionsweise eines Echtzeitbetriebssystems

Anforderungen und Vergleich zu "normalen" Betriebssystemen

Multitasking kooperativ und preemptive

Priorität und Zustände einer Task

Mutex, Semaphoren

ereignisgesteuertes Multitaskig

Intertask-Kommunikation mittels Queues

Deadlocks und Race-Conditions

I/O-Schnittstellen eines Rechnersystems und deren Nutzung mittels C (am Beispiel des dedizierten Kleinrechnersystems)

digitale Ports (siehe oben)

Timer/Counter (inkl. Digital-Analog-Wandlung mittel Pulsweitenmodulation)

Analog-Digital-Wandler

serielle Schnittstelle

Nutzung der I/O-Schnittstellen aus C heraus

Interrupts

Interrupt-Quellen und -Arten (extern, intern, hardware, software)

Interruptsverwaltung

Interrupt-Vektor-Tabelle

Interrupts-Service-Routine

zeitlicher Ablauf der Interruptsbearbeitung

Mechanismen zur Bearbeitung konkurrierender Interrupts

Priorisierung

Unterbrechung

Problemspezifischer Einsatz dieser Mechanismen

Nutzung von I/O-Schnittstellen mittels Interrupt unter C

Laufzeitsystem für C-Programme

Unterprogrammaufruf in Assembler

Stack und Assemblerbefehle zur Stackmanipulation

Programmuzustandsrettung und -wiederherstellung mittels Stack

C-Funktions-Parameterübergabe mittels Stack

Verwaltung lokalen C-Variablen mittels Stack

dynamischer Stackauf und -abbau bei geschalteten C-Funktionsaufrufen

manuelle Interpretation des Stackinhalts mittels Debugger

Fertigkeiten

Verstehen und erläutern der Arbeitsweise eines Mikrocontroller-System (Hardware und Echtzeitbetriebssystem)

Detaillierten technischen Spezifikationen von I/O-Schnittstellen interpretieren, so dass zielgerichtete sinnvolle Konfigurationen erstellt werden können

Erstellen von Treiberbibliotheken in C für verschiedene I/O-Schnittstellen mit Unterstützung ihrer Interruptfähigkeit

digitale Ports

Timer/Counter

Analog-Digital-Wandler

serielle Schnittstellen

Systemverhalten aus spezifizierenden Texten herleiten

technische Texte erfassen

implizite Angaben erkennen und verstehen

fehlende Angaben

erkennen

ableiten

erfragen

Erarbeitung von Problemlösungen aus dem Bereich Messen-Steuern-Regeln, die sich mit C-Programmen realisieren lassen

Systemverhalten aus spezifizierenden Text herleiten

Auswahl und Konfiguration der benötigten I/O-Schnittstellen

Erarbeitung eines Softwarekonzeptes

Aufteilung der Aufgaben in Prozesse

Kommunikationskonzept

Interruptskonzepte

Aufstellen des Zustandsüberförungsdiagramms

Auswahl der geeigneten Spezifikationsform (Moore versus Mealy)

Bewertung der Spezifikation

Vollständigkeit

Determiniertheit

Lebendigkeit

Implementierung mittels C

Laufzeitsystem für C-Programme analysieren und beschreiben

dynamischer Stackauf und -abbau bei geschachtelten C-Funktionsaufrufen ermitteln und beschreiben

dynamischen Stackaufbau mittels Debugger analysieren und beschreiben

Aufwand Präsenzlehre

Typ	Präsenzzeit (h/Wo.)
Vorlesung	2
Übungen (ganzer Kurs)	1
Übungen (geteilter Kurs)	1
Tutorium (freiwillig)	0

Separate Prüfung

keine

^ Praktikum

Lernziele

Fertigkeiten

siehe Fertigkeiten, die unter "Vorlesung/Übung->Lernziele->Fertigkeiten" aufgeführt sind

zielgerichtetes Handhaben der Software-Entwicklungsumgebung

komplexere Aufgaben in einem Kleinteam bewältigen

Erarbeitung von komplexeren Problemlösungen aus dem Bereich Messen-Steuern-Regeln, die sich mit C-Programmen unter Verwendung eines Echtzeitbetriebssystems realisieren lassen

Aufwand Präsenzlehre

Typ	Präsenzzeit (h/Wo.)
Praktikum	1
Tutorium (freiwillig)	0

Separate Prüfung

keine