EREIGNISGESTEUERTE SYSTEME PRÜFUNGSÜBERSICHT SS 2015

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Be Prepared to Answer

- 20 Questions- taken from these subjects:
  - Event Driven System Basics
  - Programing Code Models
  - UML StateCharts
  - State Machines
  - State Machine Implementations
  - Real Time frameworkS / RTOS Execution Environments
  - RT FRAMEWorks / RTOS CPU Architecture

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Prüfungskatalog

EVENT DRIVEN SYSTEM BASICS

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Real Time Event Driven Systems

- Event Driven Paradigma
  - Event Capture
  - Event Dispatch
  - Event Processing

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Real Time Event Driven Systems

- Real Time Systems are „event driven“ when Program control is a function of an event occuring in the system and ending with a determined response
- Real Time Event Driven System Architecture is based on 3 logical levels
  - Event Capture
  - Event Dispatch
  - Event Processing

Real Time System Types

- Regardlesss of the trigger – two basic approaches
  - Event Driven – event type determine the state change of the RT Entity
  - Time Triggered – periodic time slices determine the state change of the RT Entity

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Event Types

- Predictable Events
  - Function of physical activity
    - Pressure in vessel exceeds a certain limit
  - Deterministic, hence resource allocation and reservation is integral part of system design
- Chance Events
  - Event occurance is random
  - Non Deterministic
    - Implementation based on statistical data

Prüfungskatalog

PROGRAMING CODE MODELS

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Programing Models

- Sequential Code Flow Model:
  - structured to flow from start to end:
  - first do this,
  - then do that
  - to achieve a desired RESULT!
- Asynchronous Event Handling
  - Events are abstracted away from app logic
  - Execution Context Control is key element

Concepts of Sequential Programs

- Processing flow is accomplished via:
  - calls to subroutines, functions
  - return to main control loop
- Program behaviour is captured
  - in global variables (access in the subroutines)
  - stack operations (arguments and return values)
- Execution control is a function
  - of the sequential progress
  - the main procedure is always in control
- Current Execution Context is always a
  - Point in a progression from A to B

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Asynchronous Event Model

- Program logic is determined by external events (interrupts) , or internal exceptional conditions (Blinky: – AD Ints + Internal Trans)
- Processing is accomplished by mapping an event-handling routine to an event
- Program behaviour is captured in static, event specific variables and event handlers
- Execution control is NOT a structured progression – seems to be chaotic

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Inversion of Control – Key to Event Processing

The Hollywood Principle **Don‘t call us, we call you**

Register the interface/handler **When the time is right**

Avoiding Spaghetti

- The Tools
  - UML – maintain overview
  - Object Oriented Techniques - consistent environment
- State Machines and
  - Inversion of Control
    - Inversion of Control defies human logic
    - Is NOT suited for Flow Charts??
    - It is easy to loose sight of system purpose

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Prüfungskatalog

UML STATECHARTS

The Role of UML State Charts

- UML State Charts enable:
  - Blueprint for defining encapsulated states
  - Representation of transitions between states
  - Concept for controlling Inversion of Control
  - Ultimately:
    - EXECUTION CONTEXT CONTROL
- UML State Charts are:
    - NOT FLOW CHARTS
    - NOT Code Generators

UML State Chart Extensions

- Separate the event object (variable) from the state
  - The state handler is the qualitative component
  - The variable is the quantitative component
  - Extended State Variables combine quantitative + qualitative aspects to comprise the complete state
- Add concept of nested states

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Prüfungskatalog

STATE MACHINES

Generic State Machine Theory

- Finite State Machines are defined as a 5-tupel
  - (S, Σ, s, F, δ)
  - Where S = States (Setting, Timing)
  - Where *Σ = Events – UP / DOWN ARM TICK*
  - Where *s = Initial state*
  - Where *F =Set of ending states F={s(i)}*
  - Where *δ = Transition function determining next state*

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Transition Mapping

- Transition Mapping is key element in definition
  - Efficient mapping of Events to Handler
  - Requires compact & resource effective paradigm
- The Event Action-Handler Paradigm
  - Capture Event
  - Map Event to Handler (c/c++ function call)
  - Dispatch Event

Prüfungskatalog

STATE MACHINE IMPLEMENTATIONS

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

State Machine Implementations

- There are certain "standard" implementation methods :
  - Nested Switch
  - State Tables
  - Object Oriented Design Patterns
- ALL have limitations:
  - Nested switch -> spaghetti code via multi levels
  - State Tables -> difficult to initalise and maintain
  - OO requires OO Language – bad for small µP

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Implementation Models

- State Machine may be:
  - Nested Switch Case
  - Function Pointer State Tables
- Nested Switch Case
  - Simple to Implement
  - Low Memory Usage
  - Depends on Procedure Model
  - Complexity increases with number of events
- Functions Pointer
  - Efficent Mapping of Events to Handlers
  - 1st Step toward generic event processing
  - Complex initialisation and maintaince issues

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Opaque Pointers

- Opaque Pointers
  - Enforce information hiding
  - Special case of „opaque data type“
- Play a fundamental role in Object Oriented Programming by enabling:
  - Encapsulation
  - Polymorphism
- Available in Ada / C / C++

Object Oriented Principles applied to State Machines

- Encapsulation is achieved through
  - Separate event variables – the quantitative component
  - Separate handler variables- the qualitative component
- Inheritance is achieved through
  - Extended State Variables
  - Opaque pointers
- Polymorphism is achieved through
  - Late binding of events to handlers via opaque pointers and function pointers

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Function Pointers

- Function Pointers enable opaque or late binding of state handle to Event Processor
- Function Pointers are „life line“ of state machines
- On Havard/RICS Machines, Function Pointers enable „low cost“ task switching in micro kernel environments

Hierarchical HSM Implementation

- UML Introduces State Nesting - Advantages
  - Helps control state explosion
  - Promotes reuse of behaviour
  - Enables inheritance in sub states from features in super state
  - Presumes however an OO CASE Environment
- Vast majority of RT Embedded Systems
  - Resource Constrained – minimal memory and CPU
  - Applications are very HW – Aware
  - Written usually in ASM and C combinations

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Advantages of HSM

- Support decomposition of Problem
- Reduce redundant code implementation
- Superstate – Substate Architecture
  - Excellent Inheritance properties
  - Good Encapsulation
- Concept of “top-state” logically bounds scope of reactive possibilities

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Prüfungskatalog

REAL TIME FRAMEWORKS / RTOS EXECUTION ENVIRONMENTS

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Frameworks vs. Toolkits

- Tookits
  - Provide APIs called by programer‘ logic
  - Main program logic is in controll

- Frameworks
  - Provide main which calls progamer‘s logic
  - Provide „glue“ to underlying RTOS resources
  - Enable abstraction and encapsulation of RTOS

Ideal Real Time Event Processor

– Follows UMS State Chart Rules
– Based on Formal Definition
– Desired Attributes :
  - Uses Nested Switch efficiently (single level)
  - Eliminated State Tables und associated overhead
  - Is Object-based but does not require OOLanguage
  - Implemented as a Framework Component

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

RT Framework Implementation Goals

- Small footprint suitable for µ Controlers
- Highly Portable and Maintainable
- Flexible – does not need an OO Language
- Demonstrate that CASE is not necessary
- Based on a RT Framework that enforces
  - Execution Control
  - Inversion of Control
- Integrates easily with RTOS (QNX, Nucleus+, etc)
- Allows Execution Context Control via Active Object

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Transparent Inversion of Control

- Abstracts Event Source (i.e., interrupts)
- Separates Event Occurrence and Handling
  - control resides in the event manager
  - application handler is called only when event arrives and runs to completion
- Co-ordinates Execution Context

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Soft RTOS…

- In a soft real-time system, it is considered undesirable, but not catastrophic, if deadlines are occasionally missed.
- Also known as “best effort” systems
- Most modern operating systems can serve as the base for a soft real time systems.
- Examples:
  - multimedia transmission and reception,
  - networking, telecom (cellular) networks,
  - web sites and services
  - computer games.

Hard RTOS…

- A hard real-time system has time-critical deadlines that must be met; otherwise a catastrophic system failure can occur.
- Absolutely, positively, first time every time
- Requires formal verification/guarantees of being to always meet its hard deadlines (except for fatal errors).
- Examples:
  - air traffic control
  - vehicle subsystems control
  - Nuclear power plant control

RT Frameworks – RTOS

- OO RT Frameworks abstracts the RTOS Resouces to provide generic intefaces for:
  - Seperate Event Capture
  - Seperate Event Dispach
  - Seperate Event Processing / Handling
- Generic Interfaces help to realise the OO goals:
  - Ecapsulation
  - Reusability of code

RTOS –Active Objects

- Active Objects Abstract Execution of Control
  - Event Queue Management
  - State Handler Management
  - Task Control Management
- Into a generic task management object
- Helps realise the OO Goals:
  - Ecaplsulation
  - Inheirtance
  - Resuability

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Prüfungskatalog

CPU ARCHITECTURE & KERNEL MODELS

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Comparison of CISC - RISC

Ciscc Architectuer

- Multi CLK Cycle
- Load / Store via Mem to Mem
- High Clock Cycles
- Large # of transisitors
- Pipeline difficult
- Closed BUS

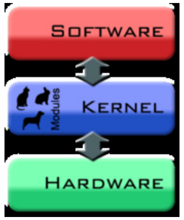
RISC

- 1 Inst per CLK Cycle
- Load/Store REG to REG
- Low Clock Cycles
- Small Num Transistors
- Pipelineing easy
- Open BUS - SOC

Kernel Architectures

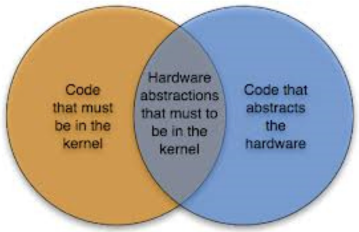
Monlithic

- Entire OS is in Kernel Space
- All Services run in Supervisor Mode
- Architecture developed for Von Neuman / CISC and General Purpose Register CPUS
- Unix / Linux / OS360



Micro

- Only Basic Services in Kernel
- Seperate Address space for
  - Kernel Services
  - User Service
- „Perfect Fit“ for opaque pointers on HAVARD / RISC CPUs
- Nucleuss / QNX / FreeRtos



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Active Objects /CPU / Task Control

- RISC Register to Registers Operations enables:
  - AO Task Control Block (TBC) to be loaded in single operation
  - Calling Active Object State Machine Handler
    - Code via
      - opaque fuction pointers
    - Events via
      - Opaque ponter to event object in state handleer signature
    - …in a single register operation
  - All necessary Pointers are
    - Preloaded in Registers
    - Code and Data are acessed in paralled (Havards machine)