AUTOMOTIVE EMBEDDED SYSTEMS

1. Introduction

1.1. Opening
1.2. The V Model
1.3. Emerging concepts
  1.3.1. Embedded software and telematics
  1.3.2. Functional safety, software architecture and testing
  1.3.3. Process assessment
1.4. Structure of the course. Practicalities

2. Embedded software

2.1. Embedded software design principles
  2.1.1. Algorithm design and coding practices
  2.1.2. Advanced I/O techniques
    2.1.2.1. DMA-handled I/O
    2.1.2.2. Interrupt-handled I/O
  2.1.3. MISRA-C design rules and good practices
2.2. RTOS
  2.2.1. Introduction
  2.2.2. Kernel
  2.2.3. Tasks, multitasking and multithreading
  2.2.4. Scheduler
  2.2.5. Inter-process communication
2.3. The CAN communication protocol
  2.3.1. Introduction
  2.3.2. Bus topology
  2.3.3. CAN messages
  2.3.4. Physical layer
  2.3.5. Bit Timing
  2.3.6. Error handling
  2.3.7. Protocol versions (2.0A, 2.0B, Open)
2.4. Laboratory sessions
  2.4.1. Introduction to the laboratory and the design tools
  2.4.2. Design of a standalone software application
  2.4.3. Design of a software application based on an RTOS

3. Telematics

3.1. Short range communications
  3.1.1. Remote keys. Tire monitoring
  3.1.2. RFID NFC and applications
  3.1.3. Bluetooth 5.0
3.2. V2X communications
  3.2.1. WLAN (IEEE 802.11p)
  3.2.2. C-V2X
3.3. Location and positioning
  3.3.1. GNSS: GPS, GLONASS and Galileo
  3.3.2. Assisted GPS and Dead Reckoning
  3.3.3. European eCall initiative
3.4. Embedded Linux on automotive telematics

3.4.1. Linux kernel architecture: essential points for adapting the kernel to a custom embedded platform
3.4.2. Techniques for right-sizing the system to meet project constraints
3.4.3. Yocto Distribution: Cross development environment for embedded projects.
3.4.4. Bootloaders. Focus on U-Boot and Android Fastboot
3.4.5. Flash storage and file systems
3.4.6. Developing and debugging applications for the embedded system
3.5. Laboratory sessions
3.5.1. Develop a Linux application for launching and interact with a Qualcomm Linux modem

4. Autosar

4.1. Reference architectures and their role in software Systems
4.2. AUTOSAR: a software reference architecture for the automotive industry
4.2.1.Goals
4.2.2.Chronology. Releases
4.2.3.Partnership
4.3. Background
4.3.1.Automotive communication protocols: CAN, LIN, Flexray
4.3.2.Diagnostics. UDS ISO 14229. Adaptation of UDS to CAN
4.4. Constituent elements of AUTOSAR
4.4.1.The layers.
4.4.1.1. Basic software. Dependencies
4.4.1.2. Runtime Environment and its configuration
4.4.1.3. Application layer
4.4.2.The Virtual Functional Bus
4.4.3.Interfaces
4.5. AUTOSAR methodology
4.5.1.Defining the architecture
4.5.2.Development processes
4.5.3.Software production. Code generation (model-based)
4.5.4.Data interchange
4.5.5.Tool support
4.6. Conclusions

5. Verification and validation

5.1. Introduction
5.2. Test levels (unit testing, system testing, integration testing, ...)
5.3. Test methods (black box, white box, grey box, ...)
5.4. Test automation – otherwise, traceability
5.5. Test-driven development
5.6. Conclusions

6. Functional safety

6.1. Introduction
6.1.1. What does functional safety mean?
6.1.2. Definitions.
6.1.3. ISO26262 structure.
6.1.4. Hazard & Risk analysis and determination of ASILs
6.1.5. System-level architectures and examples.

6.2. Software safety
    6.2.1. Software safety process overview
    6.2.2. Specification and requirements.
    6.2.3. Architectural description for functional safety.
    6.2.4. Patterns in software architecture.
    6.2.5. Freedom from interference.
    6.2.6. Software safety analysis.
    6.2.7. Software verification and validation methods.
    6.2.8. Autosar and safety.

7. SPICE methodology

7.1. Introduction
7.2. Process Maturity Models. CMM. SPICE
7.3. Automotive SPICE
    7.3.1.Process Groups
    7.3.2.Work Products
    7.3.3.Maturity Levels
7.4. Conclusions