Instrumentation and Control Engineering

Interdisciplinary Minor

Embedded Automotive

Sem	Course Type	Course Code	Course Name	L	T	P	S	Cr	Evaluation Scheme				
									Theory			Laboratory	
									MSE	TA	ESE	ISE	ESE
III	IDP-01	*<tbd>*	Foundations of Automotive Embedded Systems	2	-	2	1	3	30	20	50	50	50
IV	IDP -02	*<tbd>*	Embedded Programming	2	-	4	1	**4**	30	20	50	50	50
V	IDP -03	*<tbd>*	Embedded Linux	2	-	4	1	**4**	30	20	50	50	50
VI	IDP -04	*<tbd>*	Automotive Communication Networks	2	-	4	1	**4**	30	20	50	50	50
VII	IDP -05	*<tbd>*	Embedded System Design with V-Model	0	-	06	1	**3**	--	--	--	50	50
			Total	**08**	**-**	**20**	**05**	**18**					

***This minor is also allowed to Instrumentation & Control Department students.**

Foundations of Automotive Embedded Systems						
Course Code			**Examination Scheme**			
Teaching Scheme	2-0-2-1		**Theory**	TA: 20	MSE: 30	ESE: 50
Credits	3		**Laboratory**	CIE: 50	ESE: 50	--

Course Outcomes: Students will be able to:

1. Explore tools and techniques of writing efficient and optimized codes.
2. Understand computer systems organization and architecture.
3. Describe the principles of computer system architecture.
4. Understand the fundamentals of system software and role of operating system in achieving parallel processing.
5. Understand the ways of Power and Performance Optimization in a computer system.

Coding Practices , Standards and GNU Tool Chain: **[4 Hrs]**

Coding standards - MISRA and code analysis, writing efficient and optimized codes, Introduction to GNU Toolchain (GCC, GDB, Make files), use of GenAI tools for writing, debugging and testing codes

Software Engineering: **[4 Hrs]**

Introduction, Software Development Life Cycle, Requirement gathering and analysis, Use Case Modeling. Software Testing and Quality Assurance – levels of testing, testing techniques.

System Organization: **[6 Hrs]**

Overview of computer system organization, Brief history of computer development; Hardware Components - CPU: structure, function, and types, Memory - main memory, virtual memory, caching, I/O devices -peripherals, interfaces; Memory Management – memory organization, Virtual memory - concept, advantages, and disadvantages, Caching - concept, types, and cache hierarchies; Input/Output Systems – peripherals and Interfaces.

System Architecture: **[6 Hrs]**

Overview and history of computer system architecture, CISC, RISC, and ARM Architectures, Bus Architecture - Bus types: system bus, I/O bus, Bus protocols - synchronous and asynchronous buses; Parallel Processing and Multi-Core Architecture.

System Software: **[6 Hrs]**

Introduction to Operating System - Operating System Concept, Roles, and Operations, process, thread, and memory management, process scheduling, memory allocation, and I/O management, booting, login, and system calls, types of operating system.

Textbooks:

[1] Kernighan, B. W., & Ritchie, D. M. (1988). The C programming language. prentice-Hall.

[2] Harris, S., & Harris, D. (2021). Digital Design and Computer Architecture, RISC-V Edition. Morgan Kaufmann.

[3] Deitel, H. M., Deitel, P., & Choffnes, D. R. (2003). Operating systems. Prentice Hall.

Reference Books:

[1] Stallman, R., Pesch, R., & Shebs, S. (1988). Debugging with GDB. Free Software Foundation.

[2] Bast, R., & Di Remigio, R. (2018). CMake Cookbook: Building, testing, and packaging modular software with modern CMake. Packt Publishing Ltd.

[3] Hayes, J. P. (2022). Computer architecture and organisation. Tata McGraw Hill.

Foundations of Computer Systems Lab	
Course Outcomes: Students will be able to:	
1. Apply coding guidelines and best practices to write high-quality code 2. Use GDB to identify and fix errors in C/C++ programs 3. Understand the basics of CMake and its applications in building C/C++ projects 4. Use the Google Test framework to write and execute unit tests for C/C++ code. 5. Utilize Generative AI tools to improve code writing, debugging and testing efficiency. 6. Explore simulation tools to model and analyze computer systems and architectures	
Topic No.	**Description**
1	Writing C/C++ programs considering coding guidelines and code analysis. Use of Generative AI tools for efficient code writing.
2	Debugging code with GDB. Use of GenAI tools for efficient code debugging.
3	Understanding CMake for cross-platform project configuration and builds.
4	Performing unit testing with Google Test framework. Use of GenAI tools for testing.
5	Simulate CPU execution, memory management, multi core task execution and bus communication using tools like Multi2Sim etc.
6	Development of Minor project in group adhering to the learned concepts.
Textbooks/ Reference Books:	
[1]	Kernighan, B. W., & Ritchie, D. M. (1988). The C programming language. prentice-Hall.
[2]	Deitel, P., & Deitel, H. (2016). C++ how to Program. Pearson.
[3]	Stallman, R., Pesch, R., & Shebs, S. (1988). Debugging with GDB. Free Software Foundation.
[4]	Bast, R., & Di Remigio, R. (2018). CMake Cookbook: Building, testing, and packaging modular software with modern CMake. Packt Publishing Ltd.
[5]	GoogleTest User's Guide: https://google.github.io/googletest/
[6]	CMake: A Powerful Software Build System: https://cmake.org/
[7]	Multi2Sim A Heterogeneous System Simulator: http://www.multi2sim.org/

Embedded Programming						
Course Code			**Examination Scheme**			
Teaching Scheme	2-0-4-1		**Theory**	TA: 20	MSE: 30	ESE: 50
Credits	4		**Laboratory**	CIE: 50	ESE: 50	--

Course Outcomes: Students will be able to:

1. Understand the fundamentals of embedded systems and interpret technical documents.
2. Understand and explain the build and development tools used in embedded systems design.
3. Understand the function and program the microcontroller peripherals.
4. Write interrupt service routines while using microcontroller peripherals.
5. Understand RTOS internals and performance metrics.

Introduction to Embedded Systems:: **[4 Hrs]**

What are Embedded Systems and characteristics, building blocks of Embedded Systems, various applications of Embedded Systems, Difference between Desktop Vs Embedded application development process, role of microprocessor and microcontroller in Embedded System, Understanding Block Diagram, Schematic Diagram, Datasheet, User manual, Application Notes

Embedded Systems Development Tool Chain: **[4 Hrs]**

Build and development tools, Build Process, role of Linker, understanding object files, Elf Generation / Compilation/ Linker Error Resolution, tools used to extract binary code from elf, converting to different formats like plain binary, S-record, Concept of Make file and Complex Make file, Big Endian vs Little Endian, Memory mapping, understanding Startup code, Reset Code, Bootloader.

Microcontroller Peripherals and Programming I: **[4 Hrs]**

Understanding and programming STM32 microcontroller for GPIO, UART, Timer,.

Microcontroller Peripherals and Programming II: **[4 Hrs]**

Understanding and programming STM32 microcontroller for PWM, ADC, I2C and SPI Protocol

Microcontroller Peripheral Interrupts and Programming: **[6 Hrs]**

Interrupt Vector Table, Interrupt Priorities, Nested Interrupts, Writing ISR in assembly, and Embedded C, interrupt latency, Write interrupt service routines for various peripherals and operations.

Real Time Operating System: **[6 Hrs]**

Introduction to Real-Time Concepts, RTOS Internals & Real Time Scheduling, Performance Metrics of RTOS, Task Specifications, Schedulability Analysis, Application Programming on RTOS, Configuring and Porting of RTOS.

Textbooks:

[1] Gay, Warren,"Beginning stm32" *Beginning STM32* (2018).

[2] Amos, Brian. *Hands-On RTOS with Microcontrollers: Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools*. Packt Publishing Ltd, 2020.

Reference Books:

[1] K.V.K Prasad. (2003), *Embedded / Real-Time Systems: Concepts, Design and Programming Black Book*, Dreamtech Press.

[2] STM32F401 - PDF Documentation: https://www.st.com/en/microcontrollers-microprocessors/stm32f401/documentation.html

[3] FreeRTOS Documentation: https://www.freertos.org/Documentation/00-Overview

Embedded Programming Lab	
Course Outcomes: Students will be able to:	
1. Interpret and Analyze Technical Documents. 2. Utilize build and development tools and processes to design, develop, and implement projects using the STM32 microcontroller. 3. Write efficient C code to program and interface with various STM32 peripherals, including GPIO, UART, Timer, PWM, ADC, I2C, and SPI protocols. 4. Develop interrupt service routines to handle peripheral operations and events, ensuring efficient and reliable system performance. 5. Design and implement multitasking operations using the FreeRTOS.	
Topic No.	**Description**
1	Interpret and understand various technical documents, including Block Diagrams, Schematic Diagrams, Datasheets, User Manuals, and Application Notes related to the STM32.
2	Understanding build and development tools and processes involved in working with the STM32.
3	Program the STM32 microcontroller to utilize its peripherals, such as GPIO, UART, Timer, PWM, ADC, I2C, and SPI protocols.
4	Write interrupt service routines to handle peripheral operations and events.
5	Perform multitasking operations using FreeRTOS, by creating tasks and using inter process communication and synchronization mechanisms.
6	Development of Minor project in group adhering to the learned concepts.
Textbooks/ Reference Books:	
[1]	STM32F401 - PDF Documentation: https://www.st.com/en/microcontrollers-microprocessors/stm32f401/documentation.html
[2]	Free RTOS Documentation: https://www.freertos.org/Documentation/00-Overview

Embedded Linux						
Course Code			**Examination Scheme**			
Teaching Scheme	2-0-4-1		**Theory**	TA: 20	MSE: 30	ESE: 50
Credits	4		**Laboratory**	CIE: 50	ESE: 50	--

Course Outcomes: Students will be able to:

1. Understand Linux OS Fundamental.
2. Design and Develop POSIX Thread-Based Applications.
3. Understand the principles and terminologies of Embedded Linux based on Yocto.
4. Apply the BSP philosophy to configure and develop Yocto Project for application development.
5. Develop and Debug Embedded Linux Device Drivers.

Introduction to Linux: **[4 Hrs]**

Introduction to Embedded Operating Systems and Linux, Bootloader, Kernel, Root File System, Process Management, Inter-process Communication & Synchronization, Memory Management, I/O subsystem.

POSIX Thread Programming: **[4 Hrs]**

POSIX Thread Programming, Debugging and Testing of Multi-threaded Applications, Socket Programming.

POSIX IPC and Synchronization: **[4 Hrs]**

POSIX Semaphores, Mutexes, Conditional Variables, Message Queues, Shared Memory

Embedded Linux based on Yocto: **[4 Hrs]**

Introduction to the Yocto Project, setting and configuration of build environment, Yocto Project architecture and components, OpenEmbedded Build System, BitBake Build Engine, Poky.

Board Support Packages and Application Development **[6 Hrs]**

The Embedded Linux Software Eco-System, Linux Kernel Modules and Module Programming, Char Device Drivers, Kernel Internals: Dynamic memory allocations, Handling Delays, Timers, Synchronization, Locking, I/O Memory and Ports, Interrupts, Deferred Executions, Driver Debugging Techniques, Drivers for GPIO, I2C, and SPI, Pseudo Filesystems (procfs, sysfs).

Embedded Device Drivers: **[6 Hrs]**

Overview and history of computer architecture, CISC, RISC, and ARM Architectures, Bus Architecture - Bus types: system bus, I/O bus, Bus protocols - synchronous and asynchronous buses; Parallel Processing and Multi-Core Architecture.

Textbooks:

[1] Kerrisk, Michael. The Linux programming interface: a Linux and UNIX system programming handbook. No Starch Press, 2010.

[2] Streif, R. J. (2016). Embedded Linux Systems with the Yocto Project. Prentice Hall Press.

Reference Books:

[1] Hallinan, C. (2011). Embedded Linux primer: a practical real-world approach. Pearson Education India.

[2] Drivers, L. D. (3). Linux Device Drivers. Greg Kroah Hartman, Alessandro Rubini.

Embedded Linux Lab	
Course Outcomes: Students will be able to: 1. Demonstrate Proficiency in understanding and applying Linux Commands. 2. Design and Implement POSIX Thread-Based Applications 3. Configure and Customize Embedded Linux using Yocto 4. Develop, Build, and Deploy embedded Linux applications on Embedded Hardware. 5. Understand the process of developing character and GPIO drivers.	
Topic No.	**Description**
1	Gain a comprehensive understanding of various Linux commands and their applications in different scenarios.
2	Learn to harness the power of POSIX threads and Inter-Process Communication (IPC) mechanisms to demonstrate multi-threaded operations, ensuring efficient and synchronized execution of tasks.
3	Set up and configure Embedded Linux using the Yocto Project.
4	Develop, build, and launch Embedded Linux applications on QEMU and BeagleBone Hardware modules, showcasing the integration of software and hardware components in embedded systems.
5	Leveraging kernel internals create character device drivers and develop drivers for GPIO.
6	Development of Minor project in group adhering to the learned concepts.
Textbooks/ Reference Books:	
[1]	Hallinan, C. (2011). Embedded Linux primer: a practical real-world approach. Pearson Education India.
[2]	Streif, R. J. (2016). Embedded Linux Systems with the Yocto Project. Prentice Hall Press.
[3]	Drivers, L. D. (3). Linux Device Drivers. Greg Kroah Hartman, Alessandro Rubini.

Automotive Communication Networks						
Course Code			**Examination Scheme**			
Teaching Scheme	2-0-4-1		**Theory**	TA: 20	MSE: 30	ESE: 50
Credits	3		**Laboratory**	CIE: 50	ESE: 50	--

Course Outcomes: Students will be able to:

1. Explain the importance and evolution of automotive networks.
2. Understand the Controller Area Network (CAN), Local Interconnect Network (LIN) protocols and their role in automotive networks.
3. Understand the role of the internet protocol and ethernet in advanced automotive systems.
4. Understand the role of Diagnostic over IP and SOME/IP in optimizing performance in automotive systems.

Introduction to Automotive Protocols: **[4 Hrs]**

Overview of automotive networks and their importance, History and evolution of automotive networks, Types of automotive networks - CAN, LIN, FlexRay, MOST, Ethernet, Network topology and architecture, Introduction to OSI layers.

Controller Area Network: **[4 Hrs]**

Introduction to CAN protocol, CANFD, CAN in automotive systems, CAN frame structure and message formatting, CAN bus topology and wiring, CAN error handling and fault tolerance. Introduction to CANXL. Introduction to CAN Database with DBC file.

Local Interconnect Network: **[4 Hrs]**

Introduction to LIN protocol, LIN applications in automotive systems, LIN frame structure and message formatting, LIN bus topology and wiring, handling and fault tolerance.

Ethernet in Automotive Systems: **[4 Hrs]**

Introduction to Ethernet protocol, Ethernet applications in automotive systems, Ethernet network topology, IEEE 100 BASE T1, IEEE 1000 BASE T1, IEEE 10 BASE T1S, Ethernet Frame, VLAN.

Internet Protocols: **[6 Hrs]**

Internet Protocol basics and properties, IPv4, IPv6, network and host address, TCP and UDP,

Ethernet Applications: **[6 Hrs]**

Understanding and implementation of Diagnostic over IP, SOME/IP

Textbooks:

[1] Matheus, K., & Königseder, T. (2021). Automotive ethernet. Cambridge University Press.

[2] Emmelmann, Marc, Bernd Bochow, and Christopher Kellum, eds. Vehicular networking: Automotive applications and beyond. John Wiley & Sons, 2010.

Reference Books:

[1] Kosch, Timo, Christoph Schroth, Markus Strassberger, and Marc Bechler. Automotive internetworking. John Wiley & Sons, 2012.

[2] Bradner, Scott O., and Allison Mankin, eds. IPng, Internet protocol next generation. Vol. 63395. Addison-Wesley Professional, 1996.

Automotive Communication Networks Lab	
Course Outcomes: Students will be able to:	
1. Design and implement a CAN-based system. 2. Implement a LIN-based system 3. Integrate CAN communication with Enhanced Ethernet Switch. 4. Use Python scripts for CAN communication in ANDi. 5. Send and receive SOME/IP messages using Media converter.	
Topic No.	**Description**
1	Implement a CAN-based system. Investigate the effects of errors and faults on the CAN network and implement fault-tolerance mechanisms.
2	Implement a LIN-based system.
3	CAN communication using Enhance Ethernet Switch.
4	CAN communication using Python Script in ANDi.
5	Sending and Receiving TCP/UDP/ SOME/IP messages using Media converter.
6	Development of Minor project in group adhering to the learned concepts.
Textbooks/ Reference Books:	
[1]	CAN Protocols: https://www.bosch-semiconductors.com/products/ip-modules/can-protocols/
[2]	LIN standards and specifications: https://www.lin-cia.org/standards/
[3]	Technica Hardware and Software Document ,ANDi Reference Document

Embedded System Design with V-Model						
Course Code			**Examination Scheme**			
Teaching Scheme	0-0-6-0		**Theory**			
Credits	3		**Laboratory**	CIE: 50	ESE: 50	--

Course Outcomes: Students will be able to:

1. Understand the V-Design Cycle and its application in embedded systems development
2. Learn to define requirements and specifications for embedded systems
3. Apply design principles and methods to develop embedded systems
4. Understand the importance of verification and validation in the V-Design Cycle
5. Develop skills in testing and validation of embedded systems
6. Learn to implement and maintain embedded systems

Topic No.	Description
1	Requirements Definition and System Design for project/case study.
2	Component Design and Implementation of project/case study.
3	Unit Testing of project/case study.
4	Integration and deployment of project/case study.
5	Validation of project/case study.
6	Documentation of the project/case study.

Textbooks/ Reference Books:

[1]	Gajski, D. D., Abdi, S., Gerstlauer, A., & Schirner, G. (2009). Embedded system design: modeling, synthesis and verification. Springer Science & Business Media.
[2]	Gerardus Blokdyk, (2022). V-Model The Ultimate Step-By-Step Guide. 5STARCooks
[3]	https://www.code-intelligence.com/blog/everything-about-v-model-and-testing-embedded-software