Advanced Embedded Systems

C language

- Introduction to C language
  - Features of C
  - History
  - Structure of C Program.
  - Keywords, Identifiers, Variables and Constants
- Data Types
  - Primitive Data Types.
  - Aggregated Data Types.
- Operators
  - Binary Operators
  - Unary Operators
  - Ternary Operators
  - Special Operators
  - Order of evaluation.
- Preprocessor features
- Control-Flow Statements
  - The Control-Flow Program Statements
  - Looping Statements
  - The Data-checking process
- Functions
  - Role of Functions
  - Passing arguments to functions
  - Returning values from functions
  - Recursive Functions
  - Callback functions
  - Implications on Stack
  - Pass by value/reference
- Arrays
  - Defining, initializing and using arrays
  - Multi Dimensional arrays
  - Arrays of Characters and Strings
  - Arrays and pointers
  - Passing arrays to functions

  - String handling and its library functions

- Storage Classes
  - Scope and lifetime of a variable
  - Internal
  - External/Global
  - Automatic
  - Static
  - Register
  - Volatile
- Structures & Unions
  - Usage of Structures
  - Declaration, initialization and accessing
  - Nested Structures
  - Array of structures
  - Allocation of memory and holes
  - Unions
- Bitwise Operators
  - AND (&), OR (|), XOR (^)
  - Compliment (~)
  - Left-shift (<<), Right-shift (>>)
  - Masking, Setting and Testing of Bit/Bits
- Pointers
  - The purpose of pointers
  - Defining pointers
  - The & and * operators
  - Pointer assignment
  - Pointers with functions
  - Pointer to Pointer
  - Pointers to Arrays
  - Arrays of Pointers
  - Void Pointers
  - Call By value and Call by reference
  - Advanced pointer types
  - Pointers to functions
  - Pointers and Strings

  - Pointers and Dynamic memory
- Command line arguments
- Dynamic Memory Allocation
  - Allocation (Malloc, Calloc, Realloc)
  - De-Allocation(Free)
- Variable Number of Arguments
  - Implementation of printf()
  - Implementation of scanf()
- Data Structures
  - Introduction
  - Linked Lists
  - Stacks & Queues
  - Stacks Using Arrays
  - Stacks Using Linked List
  - Queues using Arrays
  - Queues using Linked List.
  - Circular Queues
  - Single Linked List
  - Circular Linked List
  - Double Linked List
  - Infix, Prefix and Postfix Expressions
  - Trees
  - Binary Trees
  - Binary Search Trees
  - Graphs
  - Hashing
- Sorting and Searching Techniques
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Merge sort
  - Quick sort
  - Heap sort
  - Linear search
  - Binary search
- File Handling Concepts

- Concept of a FILE data type
- File Input, Output operations
- Sequential Files
- Random Access Files

- Standard I/O Library Functions
    - fopen,fread,fwrite,fclose,fseek
    - Relationship between file descriptor
    - and FILE pointer
    - Character at a time I/O
    - Line at a time I/O
    - Formatted I/O
- Reading and Writing Structures to Files
    - In Ascii format
    - In Binary format
    - Modifying a structure in the file

Development Tools & utilities:

- Linux commands useful in development
- Consumer Electronics
- Vi editor
- GCC compiler
- Functionality of Preprocessor
- Functionality of Compiler
- Functionality of Assembler
- Functionality of Linker
- Interrupting the Compiler
- Compiling a C Program
- Preprocessor Features
- Predefined Preprocessor Symbols
- Warnings and Extensions
- Optimization
- GDB debugger
- Archive Utility
- Make Utility
- Object File format

- Executable File Format

Operating System Concepts:

Learning of operating system concepts will help you in understanding Desktop, Embedded & Real-time Operating Systems easily in less time.

- Introduction
- Processes
- Threads
- CPU Scheduling
- Process Synchronization
- Deadlocks
- Memory management
- Virtual Memory.
- File management & Disk management

Linux System Programming:

Linux is used in almost all system domains (Networking/Telecom) and also most of the RTOS are very similar to Linux.

So learning of Linux programming will help you in understanding and work easily in system domain as well as in embedded systems.

- The GNU C Library and System Calls
  - Library Goals
  - Library Standards
  - GNU C Library - glibc
  - Library Functions vs. System Calls
  - Using System Calls
  - Handling Errors with errno
  - Making Sense of errno
  - Using strace
- Program Arguments and Environment
  - Program Startup

- Using argc/argv
- Handling Options with getopt()
- Handling Options with getopt_long()
- Environment
- Manipulating the Environment
- Program Exit
- Registering Exit Handlers

➢ Building Libraries

- Why Use Libraries?
- Static Versus Shared
- Static Library Benefits
- Shared Library Benefits
- Creating a Static Library
- Using Static Libraries
- Creating a Shared Library
- Using Shared Libraries
- Shared Library Management
- Library Locations
- ldconfig

➢ Time Functions

- When Does Time Begin?
- Time Data Types
- Determining Real Time
- Converting time_t
- Converting tm Structure
- Process Time
- Time arithmetic
- Second Resolution Timers
- Fine-Grained Timers
- Real Time Clock (RTC)

➢ Process Management

- What a Process Is
- Process Relationships
- Create a Child Process
- Doing Something Else
- Related execve() Functions

  - Wait For a Child
  - More Precise Waiting
  - Changing Priority/Nice
  - Real Time Priority
- Memory Operations
  - Allocating/Freeing Memory
  - Memory Alignment
  - Locked Memory
  - Memory Copy/Initialization
  - Memory Comparison/Search
- Debugging
  - What Is My Program Doing?
  - Source Level Debugging
  - Invoking gdb
  - Getting Started with gdb
  - Examining and Changing Memory
  - Debuginfo Libraries
  - Using gdb with a Running Process
  - Using gdb to Autopsy a Crash
  - Debugging Libraries - ElectricFence
  - Debugging with valgrind
  - Profiling for Performance
- Basic File Operations
  - Stream vs. System Calls
  - Opening/Closing Streams
  - Stream Input/Output Functions
  - Stream Status/Errors
  - Stream File Positioning
  - Stream Buffering
  - Temporary/Scratch Files
  - Opening/Closing File Descriptors
  - File Descriptor I/O
  - Repositioning File Descriptors
  - Stream/File Descriptor Conversions
  - cat using ANSI I/O
  - cat using POSIX I/O

- Communicating with Pipes
  - Introduction to Pipes
  - Standard I/O: popen()/pclose()
  - Using popen()/pclose()
  - System Call: pipe()
  - Using pipe()
  - Named Pipes
  - Using Named Pipes
  - For Further Reading
- Managing Signals
  - What Signals Are
  - Blocking/Checking Signals
  - Working with Signal Sets
  - Example of Blocking Signals
  - Handling Signals with sigaction()
  - igaction() Example
  - Handling Signals with signal()
  - Sending Signals
- Programming with Threads
  - Introducing Threaded Programming
  - Applications Suited to Threads
  - Building Threaded Programs
  - Creating Threads
  - Thread Identity
  - Synchronizing by Joining
  - Detaching Threads
  - Stopping Threads
  - Synchronizing with Mutexes
  - Using Mutexes
  - Read/Write Locks
  - Conditional Variables
  - Using Conditional Variables
- Advanced File Operations
  - Directory Operations
  - File System Operations
  - Multiplexed I/O with select()

  - Miscellaneous I/O Functions
  - Memory Mapped I/O
  - Using Memory Mapped I/O
  - File Locking
- Interprocess Communication
  - Interprocess Communication (IPC)
  - POSIX IPC Overview
  - POSIX Shared Memory
  - POSIX Semaphores
  - POSIX Message Queues
  - System V IPC Overview
  - System V IPC Shared Memory
  - System V IPC Semaphore Arrays

Introduction to Embedded Systems:

- Programmer's view of hardware
- CPU
- Types of CPUs
- CPU Characteristics
- CPU Bus/Machine Cycles
- Memory
- Memory types
- CPU Memory interface
- I/O
- I/O controllers
- CPU I/O interface
- I/O methods/techniques
- Polled I/O or Programmed I/O
- Interrupt Support
- DMA Support

Development Environment:

- Host-Target Environment
- Cross compilers
- Downloading methods

- Serial, Ethernet, Floppy, ROM
- Emulators
- Target based debugging
- Debug Monitors
- Host based source level debugging

Linux Kernel & Device Drivers:

After Gaining knowledge on Advanced Linux Programming you are ready to learn Device Drivers.

This is where you learn core of Linux Kernel and Device Drivers programming.

- Introduction to Linux kernel Programming
- Kernel Classifications
  - Monolithic Kernels
  - Micro Kernels
- The User space & Kernel space
- Tool Chains, Libraries, The Makefile
  - GNU toolchain
  - Creation of Static & Dynamic Libraries
  - Portability support in the kernel
- The Linux Kernel
  - Getting the sources
  - Configuring the kernel
  - Diff and Patching utilities
  - Compiling the kernel
  - Installing & Booting the kernel
- Step by Step demystification of Linux Boot Procedure
  - Module Programming
  - The HelloWorld Module
  - Module Stacking
  - Module Parameters
- System Calls
  - Registering a System Call
  - System Call Handler
  - Service Routines

Character Drivers

- Device Numbers
  - Major and Minor Numbers
  - Registering and Unregistering
  - Static and Dynamic allocations
- Important Structures
  - File Operations
  - File
  - Inode
- Character Devices
  - cdev structure
  - Adding, Allocating, Initializing and Deleting
  - User Space Applications and Device Driver mapping
  - Access methods within the driver, open, rea\d, write and close
  - Advanced Character Drivers
  - IOCTL
  - Wait
- Kernel Synchronization
  - Critical Sections, Race Conditions
  - Concurrency and its Sources
- Mechanisms for Kernel Synchronization
  - Semaphores
  - Reader/ Writer Semaphores
  - Spinlocks
  - Reader/ Writer Spinlocks
  - Completions
  - Sequential locks
  - Barriers
  - Read Copy Update
- Atomic Operations
- Memory Allocation in the kernel
- Debugging the Kernel
  - Printk, Traces and Watches
  - gdb, kgdb, kdb
  - User Mode Linux
  - Qemu
  - Proc & Sys File Systems

- Timers & Bottom Halves
  - HZ & Jiffies, Delays
  - Kernel Timers
  - Soft irqs
  - Tasklets
  - Work Queues
  - Kernel Data Types
- Interrupts
  - Handling I/O
  - I/O Architecture
- I/O Mapped I/O
- Memory Mapped I/O
- Interrupts & Registering Interrupt Handlers
- Interrupt Context vs Process Context

Block I/O Layer

Block Device Structure

Request queues

Block Driver

I/O Scheduling

Linux Device Drivers

- UART Driver
  - UART Protocol
  - UART Driver Layered Architecture
  - UART subsystems
  - Porting, Development & Validation of UART Driver
- I2C Driver
  - I2C Protocol
  - I2C subsystems
  - I2C Driver Layered Architecture
  - Porting, Development & Validation of I2C client Driver
  - Porting, Development & Validation of I2C platform Driver
- PCI Driver
  - PCI Architecture & Protocol

- PCI Regions & Direct Memory Access
- PCI subsystems
- PCI Driver Layered Architecture
- Porting, Development & Validation of PCI client Driver

➢ USB Driver

- USB Architecture & Protocol
- Types of Descriptors & URB
- USB subsystems
- USB Driver Layered Architecture
- Porting, Development & Validation of USB Gadget Driver

➢ Network Drivers

ARM (32-bit) Processor Architecture & Programming:

The ARM is a 32-bit reduced instruction set computer (RISC) instruction set architecture (ISA) developed by ARM Holdings. It was known as the Advanced RISC Machine. The ARM architecture is the most widely used 32-bit ISA in terms of numbers produced. The relative simplicity of ARM processors made them suitable for low power applications. This has made them dominant in the mobile and embedded electronics market, as relatively low cost, and small microprocessors and microcontrollers.

As of 2005, about 98 percent of the more than one billion mobilephones sold each year use at least one ARM processor. As of 2009, ARM processors account for approximately 90% of all embedded 32-bit RISC processors. ARM processors are used extensively in consumer electronics, including PDAs, mobile phones, digital media and music players, hand-held game consoles, calculators and computer peripherals such as hard drives and routers.

➢ Introduction to ARM (ARM7/ARM9)
➢ ARM processor architecture & Features
➢ ARM programming model (Instruction set and assembly language programming).
➢ RISC vs. CISC
➢ Pipelining concept
➢ Fundamentals of ARM
➢ Processor modes
➢ Exception Handling
➢ ARM versions
➢ Instruction Set & Addressing Modes

- ARM(32-bit) Instruction Set
- Thumb(16-bit) Instruction Set
- Pre & Post Indexed Addressing modes
- Stack Organization
- Memory Organization
- Mixed C and assembly programs
- System Design & Development Tools
- Case studies on ARM Controllers

Real-Time OS Introduction:

- What is RTOS?
- Desktop OS vs. RTOS
- RTOS Key Characteristics
- RTOS Services
- Task Management
    - Inter task Communication Methods
    - Synchronization Techniques
    - Interrupt handling
    - Timers
    - Signals and Events
    - Priority Inversion/Inheritance
- Embedded Linux
    - Benefits of using Linux and open source tools for embedded systems
    - Linux booting sequence
    - Components of Linux booting
    - Embedded Linux system architecture
- Cross Compiler Tool Chain
    - Need for cross tool-chain
    - Using pre-build cross tool-chain
    - Building our own cross tool-chain
- Boot-loader
    - Boot-loaders and its advantages
    - Overview of U-boot source
    - Building U-boot for target
    - Booting target with U-boot

- Kernel
  - Supported hardware architectures
  - Modifying Architecture Specific code
  - Cross-compiling the kernel for target
  - Understanding kernel boot arguments
- File System
  - Understanding NAND/NOR flash
  - Understanding Linux File system hierarchy
  - Busy Box & Build root
  - Cross-compiling applications and libraries
  - Creating File System Images(jffs, jffs2, yaffs, yaffs2)
- Board Bring Up
  - Flashing Boot-loader Image
  - Flashing Kernel Image
  - Flashing File system Image