C
for Embedded Systems

Complete Course Roadmap

Learn C by Seeing It Execute on ARM Cortex-M4 via Renode

PLATFORM
Renode Emulator + ARM Cortex-M4 (Transparent execution - see every register/memory change)

NO HARDWARE
Everything in Docker Container One command: zero setup, runs anywhere

VISUALIZATION
Live PC/SP/LR tracking, Memory map, Stack frames, Heap, Register dashboard

Visit website: electro4U

COURSE DETAILS

COURSE DETAILS

Course Name: C for Embedded Systems

Mode: Online Live Training

Level: Beginner to Professional Level

Duration: 2 Months

- **A fully practical, hands-on course focused on line-by-line C code debugging for ARM Cortex-M4 processors using Renode.**
- **Lifetime Access to Materials.**
- **Certificate of Completion.**

COURSE FEES

Price: 499/-

INSTRUCTOR

Name: Yashwanth Kumar VG

Experience: 2+ Years

Current Role: Embedded Software Engineer at **Texas Instruments.**

WHY THIS COURSE

Stop learning C like a desktop programmer. Most courses hide the hardware. This course trains you to write bare-metal firmware the way industry professionals do.

Zero Hardware, Zero Setup: Run a complete **ARM-M4 environment** instantly via a single Docker command.

100% Visual Debugging: See exactly how your C code changes registers, stack frames, and memory line-by-line using **Renode**.

Interview-Ready Concepts: Master bare-metal essentials like volatile, **Linker scripts**, and **Startup code**.

Industry Mentorship: Learn the exact workflows used by an Embedded Engineer in actual Industry

WHO CAN ENROLL

Engineering Graduates & Students: Aspiring embedded engineers looking to build industry-ready portfolios for placements.

Freshers & Job Seekers: Anyone aiming to master bare-metal C programming and transition from desktop programming to firmware development.

Working Professionals (ECE / EEE / Embedded): Engineers wanting to master advanced debugging, register-level manipulation.

Aspiring Firmware Engineers: Professionals preparing for technical interviews and roles targeting Embedded Firmware positions.

1 PHASE 1: FOUNDATIONS & BIT MANIPULATION

Chapters 1-2 | Syntax, Data Types, Expressions, Bitwise Operations | ~7 hours

Getting Started

Chapter 1

1.1 Introduction, Overview of the Course.

1.2 Software Tools used in the Course, Setting up the Toolchain.

1.3 Getting Started - Hello World, C-Syntax, Template.

Data Types, Operators & Expressions

Chapter 2

2.1 Variable Names

2.2 Data Types & Sizes - char, int, float, double

2.3 Constants - integer, float, character, string

2.4 Declarations - implicit int, initialization

2.5 Arithmetic Operators + - * / %

2.6 Relational & Logical Operators < > <= >= == != && || !

2.7 Type Conversions - implicit, explicit (cast) KEY FOR EMBEDDED

2.8 Increment/Decrement Operators ++ -- (prefix vs postfix)

2.9 Bitwise Operators & | ^ ~ << >> KEY FOR EMBEDDED

2.10 Assignment Operators &= |= ^= <<= >>=

2.11 Conditional Expression ?: ternary operator

2.12 Precedence & Order of Evaluation KEY FOR EMBEDDED

Bitwise Deep Dive - The Language of Hardware KEY FOR EMBEDDED

Section 2.9

2.9.1 Bitwise AND (&) - masking, checking bits

2.9.2 Bitwise OR (|) - setting bits, combining flags

2.9.3 Bitwise XOR (^) - toggling bits, parity

2.9.4 Bitwise NOT (~) - inverting, creating masks

2.9.5 Left Shift (<<) - building masks, multiplication by 2^n

2.9.6 Right Shift (>>) - division by 2^n, sign extension

2.9.7 Compound Assignment &= |= ^= <<= >>=

2.9.8 Common Idioms: flags, bitfields, register definitions

2 PHASE 2: CONTROL FLOW & FUNCTIONS

Chapters 3-4 | Branching, Loops, Functions, Program Structure, Dynamic Memory Allocation | ~7 hours

Control Flow

Chapter 3

3.1	Statements & Blocks - compound statements { }	
3.2	If-Else - conditional branching	
3.3	Else-If - chained conditions	
3.4	Switch - jump table vs if-chain	
3.5	Loops - While & For	
3.6	Loops - Do-While (guaranteed one iteration)	
3.7	Break & Continue - loop control	
3.8	Goto & Labels - error cleanup, state machines	

Functions & Program Structure

Chapter 4

4.1	Memory Model in C	KEY FOR EMBEDDED
4.2	Basics of Functions - declaration, definition, call	
4.3	Functions Returning Non-integers - float, double, pointers	
4.4	External Variables - global scope, file scope	
4.5	Scope Rules - block, file, function, prototype	
4.6	Header Files - .h guards, declarations only	
4.7	Static Variables - internal linkage, persistent locals	KEY FOR EMBEDDED
4.8	Register Variables - hint to compiler (deprecated)	KEY FOR EMBEDDED
4.9	Block Structure - nested scopes (C99+)	
4.10	Initialization - static vs automatic, constant expressions	KEY FOR EMBEDDED
4.11	Recursion - stack usage, tail-call optimization	
4.12	Dynamic Memory Allocation in C	
4.13	The C Preprocessor - #define, #include, #ifdef	KEY FOR EMBEDDED

3 PHASE 3: POINTERS, ARRAYS & STRUCTURES

Chapters 5-6 | Pointers, Arrays, Structures, Unions, Bit-fields | ~9 hours

Pointers & Arrays

Chapter 5

5.1	Pointers & Addresses - &, *, pointer = address	KEY FOR EMBEDDED
5.2	Pointers & Function Arguments - pass by reference	KEY FOR EMBEDDED
5.3	Pointers & Arrays - a[i] == *(a+i), decay to pointer	KEY FOR EMBEDDED
5.4	Address Arithmetic - p+1 advances by sizeof(type)	KEY FOR EMBEDDED
5.5	Character Pointers & Functions - strings, strcpy, strcmp	KEY FOR EMBEDDED
5.6	memcpy(), memcmp(), memset() functions usage in <string.h>	KEY FOR EMBEDDED
5.7	Pointer Arrays; Pointers to Pointers - argv, jump tables	
5.8	Multi-dimensional Arrays - row-major layout	
5.9	Initialization of Pointer Arrays - string tables	
5.10	Pointers vs Multi-dim Arrays - different memory layout	
5.11	Command-line Arguments - argc, argv on stack	
5.12	Pointers to Functions - callbacks, state machines	KEY FOR EMBEDDED
5.13	Complicated Declarations - cdecl.org, spiral rule	KEY FOR EMBEDDED

Structures, Unions & Bit-fields

Chapter 6

6.1	Basics of Structures - struct keyword, members, sizeof	
6.2	Structures & Functions - pass by value vs pointer	
6.3	Arrays of Structures - contiguous memory	
6.4	Pointers to Structures - -> operator, linked lists	KEY FOR EMBEDDED
6.5	Self-referential Structures - linked lists, trees	
6.6	Table Lookup - binary search, jump tables	
6.7	Typedef - aliases, opaque types	KEY FOR EMBEDDED
6.8	Unions - shared memory, type punning	KEY FOR EMBEDDED
6.9	Bit-fields - packed flags, register mapping	KEY FOR EMBEDDED

4 PHASE 4: EMBEDDED-SPECIFIC C ESSENTIALS

Beyond C | Required Daily on Real Hardware | ~4 hours

Embedded-Specific C Essentials

VERY IMPORTANT FOR EMBEDDED

Beyond C Ch 1-6

7.1 volatile Keyword - prevents optimization on hardware registers

7.2 Fixed-width Types (stdint.h) - int8_t, uint32_t, intptr_t

7.3 const & Flash/ROM Placement - .rodata section, read-only

7.4 Interrupt Service Routines (ISRs) - no args, no return, context save

7.5 Linker Sections (.text/.data/.bss) - memory layout control

7.6 Startup Code & Reset Handler - .data copy, .bss zero, main()

7.7 Struct Packing & Alignment - __attribute__((packed)), alignment

7.8 Endianness & Byte Order - little-endian Cortex-M, network order

7.9 Compiler Attributes/Pragmas - __attribute__, #pragma, sections

7.10 Inline Functions & Intrinsics - __inline__, __builtin_, CMSIS

COURSE STRUCTURE: 9 WEEKS

27 Hours | 8 Labs | Progressive Complexity

Wk	Topic	Hrs	Phase	Key Focus
1-2	Foundations & Bit Manipulation (Ch 1-2)	**6**	**1**	*Variables, types, operators, bitwise*
3-4	Control Flow & Functions (Ch 3-4)	**6**	**2**	*Branching, loops, functions, preprocessor*
5-6	Pointers, Arrays & Structures (Ch 5-6)	**6**	**3**	*Pointers, arrays, structs, unions, bit-fields*
7	Embedded Essentials (Beyond C)	**3**	**4**	*volatile, stdint.h, ISR, linker, startup*
8	Debugging: GDB, Renode, Visualization	**3**	**All**	*Watchpoints, memory maps, profiling*

Ready to See C Execute?

One command starts your journey

```
docker pull electro4u/c-embedded:latest
docker run -it --rm -v ${PWD}:/workspace electro4u/c-embedded:latest
```

Visit website : https://electro4u.net