Embedded C Programming

Hands-On C Programming on Embedded Systems

Objectives

- Use the C features that matter in embedded applications, following the C language standard
- Understand each step of the toolchain and configure a linker script to place code and data in memory
- Use the debug features to follow a program running on the target
- Follow the microcontroller boot sequence from the reset vector to main, and understand the memory map
- Use types, operators, pointers, arrays and structures to access memory-mapped registers and to model peripherals in C
- Write functions, understand how they use the stack, and check for stack overflows
- Write interrupt handlers in C, set their priorities, and share data safely between interrupt handlers and the main program
- Implement data structures such as FIFOs and linked lists
- Understand dynamic allocation in an embedded context and the different memory allocators
- Analyse compiler optimisation, write optimised code and interface C with assembly

Course environment

- Theoretical course
  - PDF slides and lab guides, with lab code and solutions, sent to each participant before the course
  - Live online course on Microsoft Teams, or on the Skillnet's preferred platform
  - The trainer answers questions during the course and provides technical and pedagogical assistance
- Practical activities
  - Practical activities represent from 40% to 50% of course duration
  - 16 labs: most labs give a program skeleton to complete, then participants check the result on the target with the debugger
  - Extra challenges, without a skeleton, for some topics and for participants who finish a lab early
- Lab boards and environment
  - Virtual boards in the Zazu Simulator, developed by AC6: the same source code runs on the simulator and on a physical board
  - Optional physical board: AC6 can send participants coupons to order an NXP FRDM board
  - The complete development environment opens in a web browser, through a link sent one week before the course

Prerequisites

- Programming experience in another language
- Knowledge of binary arithmetic

Duration

- Total: 3 days
- 7 hours each, from 9:00 to 17:00 Irish time, with a one-hour lunch break

Course Outline

Day 1

Analysing the toolchain elements (2.5 h)

- Cross-compilation
- Compiler, assembler and linker
- Structure of a C program
- The evolution of the C standard, from C89/C90 through C99, C11 and C17 to C23
- The preprocessor: #define and #include, writing macros and header files with care
- Object file sections
- Library inclusion
- GCC compiler options
- Configuring the linker to place code and data in memory and to execute code from RAM
- Makefile and CMake

Lab environment (0.5 h)

- Creating a project from scratch
- Debug probe
- Communicating with the target
- Debugger windows: C source and disassembly, memory, stack, variables, registers
- Breakpoints

Microcontroller memory map and boot sequence (1 h)

- The memory map (Flash, RAM, peripheral registers and system area) and buses
- The core registers and the stack pointers
- The boot sequence from reset to main: vector table, initial stack pointer and reset handler
- The startup file: copying initialised data to RAM, clearing uninitialised data and calling main
- The standard microcontroller software interface (CMSIS) and the device header files

Types and operators (3 h)

- Storage classes (static, automatic, register, extern), location and lifetime of variables in memory
- Local and global declarations and their scope
- Scalar, fixed-width and derived types
- Signed versus unsigned
- Constants and literals
- Strings
- Type conversion and casting

- The volatile attribute for memory-mapped registers
- Logical, arithmetic, relational and bitwise operators, with masks to set, clear and test register bits
- Operator priority and expression evaluation

Day 2

Control structures (0.5 h)

- if/else, switch/case
- while, do/while and for loops
- break, continue and goto

Pointers and arrays (2 h)

- Pointer definition, initialisation, access and operations
- Pointers to constants, constant pointers and volatile pointers
- Accessing memory-mapped registers through pointers to fixed addresses
- The restrict attribute
- One- and multi-dimensional arrays
- Array initialisation, access and operations
- Arrays of pointers

Structures and unions (2 h)

- Structure and structure pointer declarations
- Field access
- Modelling a peripheral's registers with a structure placed at its base address, as in the vendor device header files
- Designing a simple driver API for a peripheral
- Padding and #pragma pack
- Big- and little-endian formats
- Bit-fields and masks for register fields
- Arrays of structures
- typedef and enum
- Unions

Functions (2 h)

- Prototypes, definition, declaration and visibility
- Function pointers, callback tables and state machines
- Function calls and parameter passing
- Stack operation, stack frames and the call stack
- Recursion and the stack
- Macros versus functions
- Pipeline and branches
- Function inlining
- Interfacing C and assembler

- Stack usage analysis and stack overflow checks with Puncover

Standard library overview (0.5 h)

- stdio
- getchar and putchar
- memcpy
- printf and scanf
- Redirecting the standard input and output to a serial UART
- File access functions

Day 3

Interrupts, exceptions and the memory model (2.5 h)

- The microcontroller exception model: vector table and vector installation, entry and return, tail-chaining and pre-emption
- Processor modes and privilege levels
- The interrupt controller (NVIC) and interrupt priorities
- The system control registers (SCB)
- The system timer (SysTick)
- Level- and pulse-triggered interrupts and interrupt clearance
- Writing interrupt handlers in C
- The memory model: where code, data, stack and heap are placed in Flash and RAM
- Normal and device memory, memory latency and cache, access ordering and memory barriers
- Sharing data between interrupt handlers and the main program: volatile, atomic accesses and critical sections

RTOS concepts (2 h)

- Basic real-time concepts
- Real-time constraints
- Multi-tasking and real time
- Tasks and task descriptors
- Context switch
- Task scheduling and pre-emption
- Scheduling systems and schedulability proof: fixed-priority scheduling, rate monotonic analysis (RMA) and earliest deadline first (EDF) scheduling

Data structures (0.5 h)

- FIFOs
- Singly and doubly linked lists

Dynamic allocation (1 h)

- malloc and free
- The sizeof operator

- Dynamic versus static allocation in embedded systems
- Stack versus heap
- The different memory allocators: buddy system, best fit and first fit, memory pools
- Memory management errors

Compiler hints and tips for microcontrollers (1 h)

- Compiler optimisations
- Mixing C and assembly
- The architecture standard for converting C function calls to assembly (AAPCS)
- Function inlining
- Unaligned accesses and padding
- Local and global data, alignment and structures

Labs

- **First program:** Set up the development environment, build, flash and debug a first program, print on the serial console
- **Build steps:** Run each build step (preprocessor, compiler, assembler, linker) with GCC and a Makefile
- **Conditional compilation:** Select code at build time with #define and #if
- **Linker script:** Define Flash and RAM regions, place code and data in them, run a function from RAM, check the map file
- **Boot sequence:** Follow the boot in the debugger from the reset vector and the initial stack pointer to main, and check the data copied to RAM by the startup file
- **Types and operators:** Storage classes, casting, operators and the volatile attribute, comparing the generated code from -O0 to -O3, then drive an LED by setting and clearing GPIO register bits
- **Control structures:** Iterative factorial with an overflow check
- **Pointers and arrays:** Exchange variables through pointers, copy one- and multi-dimensional arrays with array and pointer notation, then read and write peripheral registers through pointers
- **Structures:** Model a timer peripheral with a structure and program it, check the memory layout
- **Functions:** Pass parameters by value and by address, follow the stack frames of a recursive function in the debugger
- **Stack analysis:** Measure the stack usage of each call path with Puncover and check for stack overflows
- **Interrupts:** Configure the system timer (SysTick) and timer interrupts, set priorities and observe nesting
- **Context switch:** Implement a context switch between two tasks and follow each task's saved context in the debugger
- **Linked list:** Implement a sorted singly linked list
- **Dynamic allocation:** Use malloc and free, and find memory management errors
- **C and assembly:** Call assembly functions from C following the architecture calling standard (AAPCS), observe inlining and padding