Professional Elective-II: Embedded Systems (PECCSE601B)

Module - VI

The embedded system development environment

Prepared by – Dr. Susmita Biswas
Associate Professor
University of Engineering and Management, Kolkata

Sub topic

- The integrated development environment (IDE)
- Types of files generated on cross-compilation
- Disassembler/decompiler
- Simulators, emulators and debugging Target hardware
- debugging, boundary scan

Integrated Development Environment (IDE)

An **Integrated Development Environment (IDE)** is a powerful software tool that brings together various essential components for software development into a unified interface.

IDEs significantly improve productivity by combining common activities (editing code, building executables, and debugging) into a single application.

Functionality of embedded Integrated Development Environment (IDE)

1. **Code Editor**:
    1. IDEs typically include a **code editor** that assists developers in writing and editing code. These editors often provide features like syntax highlighting, auto-completion, and real-time error checking.
    2. They make code writing more efficient and help maintain code readability.
2. **Compiler**:
    1. A **compiler** translates human-readable code into machine-specific code that can be executed on different operating systems (such as Linux, Windows, or macOS).
    2. Most IDEs come with built-in compilers for the programming language they support.
3. **Debugger**:
    1. The **debugger** is a crucial tool for testing and debugging applications.
    2. It helps developers identify and fix errors by graphically pinpointing their locations within the code.

Functionality of embedded Integrated Development Environment (IDE)

4. **Built-in Terminal**:
    4. An IDE often includes a **built-in terminal** or console.
    5. Developers can directly run scripts or commands within the IDE, streamlining interactions with the operating system.

5. **Version Control**:
    1. IDEs facilitate **version control**, which helps manage changes to software code.
    2. Some IDEs even integrate with tools like Git, allowing users to track and manage code modifications effectively.

6. **Code Snippets**:
    1. IDEs support **code snippets**, which are reusable pieces of code for specific tasks.
    2. They reduce redundant work and enhance productivity.

Functionality of embedded Integrated Development Environment (IDE)

7. **Extensions and Plugins**:
    1. Developers can extend an IDE’s functionality by adding **extensions and plugins**.
    2. These enhance support for specific programming languages or provide additional features.

8. **Code Navigation**:
    1. IDEs offer tools like **code folding**, class/method navigation, and refactoring capabilities.
    2. These features simplify code analysis and exploration.

Disassembler

A **disassembler** is a tool that performs the reverse operation of an **assembler**. While an assembler translates **assembly language** into **machine code**, a disassembler does the opposite: it translates **machine code** back into **human-readable assembly language**.

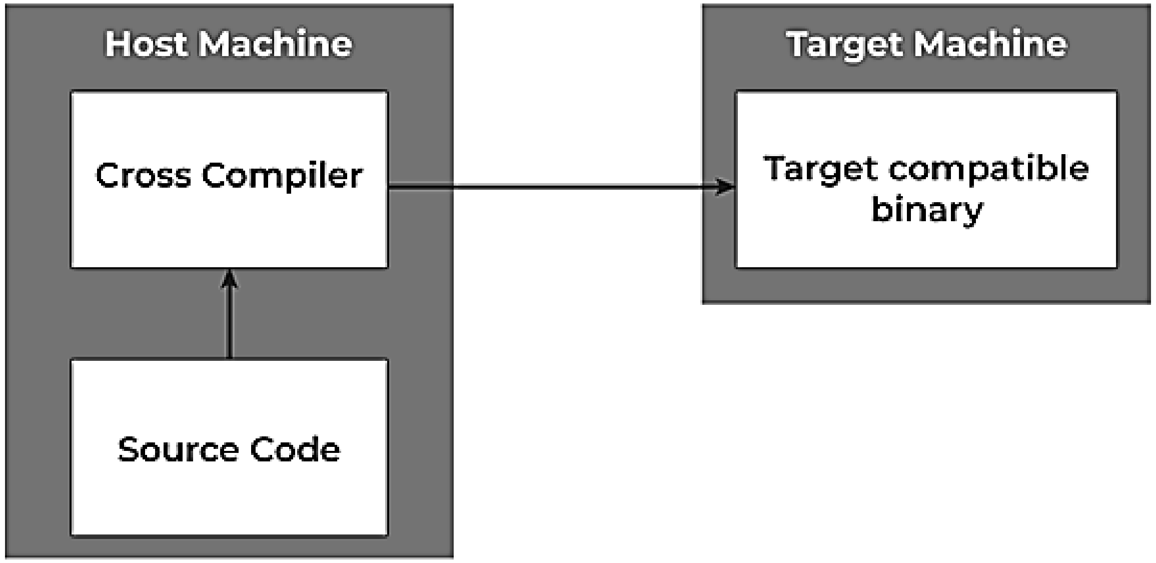
Disassembler in Embedded Systems

- **Reverse Engineering**: One of the primary use cases for disassemblers in embedded systems is **reverse engineering**. When dealing with proprietary products or closed-source software, reverse engineering helps uncover the inner workings of a system.
- **Understanding Proprietary Code**: In embedded systems, manufacturers often provide binary firmware or executables without revealing the original source code. A disassembler allows engineers to analyze and understand this proprietary code.
- **Security Analysis**: Disassemblers are essential for security professionals who need to assess the security of embedded devices. By examining the machine code, vulnerabilities can be identified and patched.
- **Debugging and Optimization**: During development or maintenance, disassemblers aid in debugging and performance optimization. Engineers can inspect the assembly code to identify bottlenecks, inefficiencies, or unexpected behavior.
- **Legacy Systems**: In legacy systems, where the original source code may be lost or unavailable, disassemblers allow developers to work with existing binaries.

Cross Compiler

Compilers are the tool used to translate high-level programming language to low-level programming language. The simple compiler works in one system only, but what will happen if we need a compiler that can compile code from another platform, to perform such compilation, the cross compiler is introduced.

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. For example, a cross compiler executes on machine X and produces machine code for machine Y.



Types of Files Generated on Cross- Compilation

The various files generated during the cross-compilation process are:

• **List file(.lst):** Contains information about the cross-compilation process.

a. Cross compiler details
b. Formatted source text
c. Assembly code generated from the source file
d. Symbol tables
e. Errors and warnings detected during cross-compilation

• **Hex file(.hex):** Hex file is the binary executable file created from the source code. The format of hex file varies across the family of processors/controllers. Intel HEX and Motorola HEX are the two commonly used hex file formats in embedded applications.

Types of Files Generated on Cross- Compilation

The various files generated during the cross-compilation process are:

• **Pre-processor output file:** Contains the pre-processor output for the pre-processor instructions used in the source file. The pre-processor output file is a valid C source file.

• **Object file(.obj):** List of some of the details stored in an object file

a. Reserved memory for global variables
b. Public symbol names
c. External symbol references
d. Library files with which to link
e. Debugging information to help synchronise source lines with object code

Types of Files Generated on Cross- Compilation

The various files generated during the cross-compilation process are:

• **Map file(.MAP):** Linking and locating of relocatable object files generate a list file called ‘linker list file’ or ‘map file’.

a. Map file contains information about the link/locate process.
b. Page header
c. Command line
d. CPU details
e. Input modules
f. Memory map
g. Program size
h. Warnings and errors