Embedded Systems

Implementation and Testing

Development Tools

- It is through which the development and integration of an embedded system's various hardware and software components are made possible
- Provide everything from loading software into the hardware to providing complete control over the various system components.
- Embedded systems require at least one other computer system connected to the embedded platform to manage the development of that platform
- In short, a development environment is typically made up of a target (the embedded system being designed) and a host (e.g. a PC)

Types of Development Tools

- The key development tools are:
  - ✓Utility tool
    - Example- editors (for writing source code), VCS (Version Control Software) that manages software files, ROM burners that allow software to be put onto ROMs
  - ✓Translation tool
    - Convert code (that a developer intends for the target) into a form the target can execute
  - ✓Debugging tools
    - Can be used to track down and correct bugs in the system

The Main Software Utility Tool: Editor or IDE

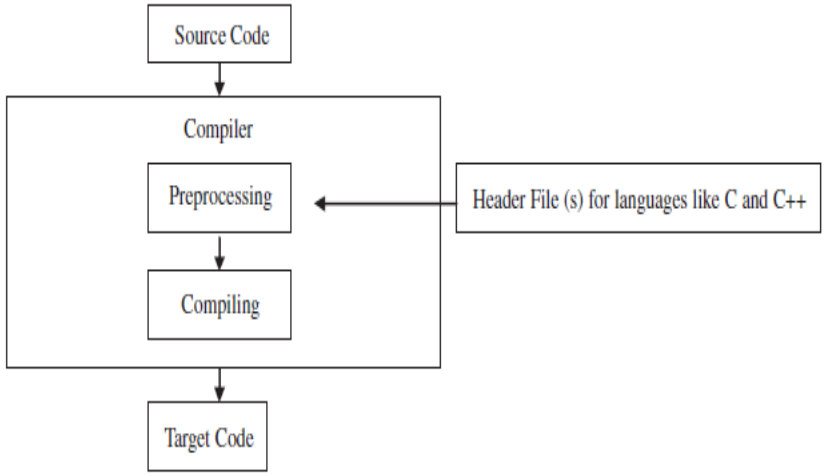
- Source code is typically written with a tool such as a standard ASCII text editor, or an Integrated Development Environment (IDE) located on the host platform
- An IDE is a collection of tools, including an ASCII text editor, integrated into one application user interface
- Any ASCII text editor can be used to write any type of code, independent of language and platform,
- However an IDE is specific to the platform and is typically provided by the IDE's vendor, a hardware manufacturer, OS vendor, or language vendor (Java, C, etc.).

Computer-Aided Design (CAD)

- Commonly used by hardware engineers to simulate circuits at the electrical level in order to study a circuit's behavior under various conditions before they actually build the circuit
- Given a complex set of circuits in a processor or on a board, it is very difficult to perform a simulation on the whole design, so a hierarchy of simulators and models are typically used.

Translation Tools

- After the source code has been written, it needs to be translated into machine code, since machine code is the only language the hardware can directly execute
- This mechanism usually includes one or some combination of preprocessing, translation, and/or interpretation machine code generation techniques.

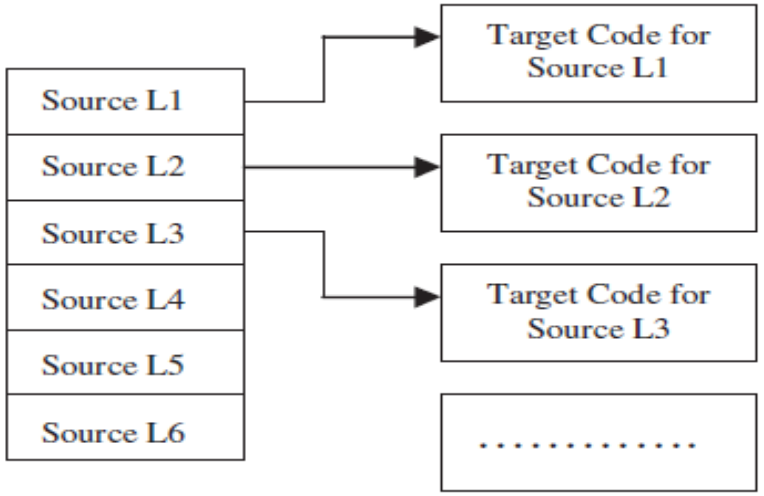


Preprocessors

- Preprocessing is an optional step that occurs either before the translation or interpretation of source code,
- Its functionality is commonly implemented by a preprocessor.
- The preprocessor’s role is to organize and restructure the source code to make translation or interpretation of this code easier.
- The preprocessor can be a separate entity, or can be integrated within the translation or interpretation unit.

Interpreters

- The IDE, preprocessors, compilers, linkers reside on the host development system,
- However some languages, such as Java and scripting languages, have compilers or interpreters located on the target.
- An interpreter translates source code into object code one instruction at a time
- The resulting object code is then executed immediately.
- The process is called interpretation.



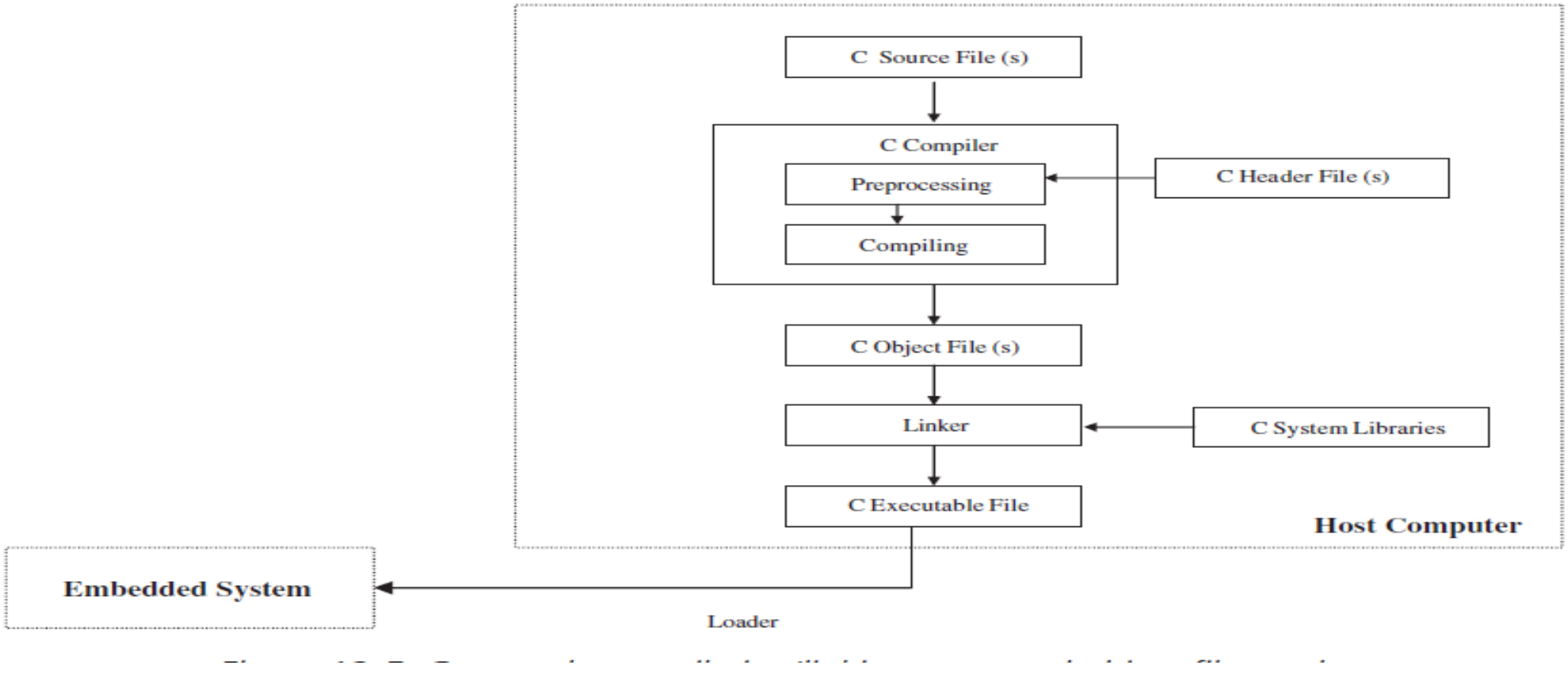
Compilers and Linkers

- A compiler typically translates all of the source code to a target code at one time
- Most compilers are located on the programmer's host machine
- Generate target code for hardware platforms that differ from the platform the compiler is actually running on.
- These compilers are commonly referred to as cross-compilers
- After all the compilation on the programmer's host machine is completed, the remaining target code file is commonly referred to as an **object file**,
  - It can contain anything from machine code to Java byte code, depending on the programming language used

Cntd..

- linker integrates this object file with any other required system libraries,
- Creating what is commonly referred to as an executable binary file
- either directly onto the board’s memory or ready to be transferred to the target embedded system’s memory by a loader.

Compilation/Linking steps and object file results



Debugging Tools

- Task of locating and fixing errors within the system
- This task is made simpler when the programmer is familiar with the various types of debugging tools available and how they can be used
- Debugging tools reside on the host, and/or on the target board

Quality Assurance and Testing of the Design

- Among the goals of testing and assuring the quality of a system are finding bugs within a
- design and tracking whether the bugs are fixed. Quality assurance and testing is similar to debugging,

Debugging vs Quality and testing

- Goals of debugging are to actually fix discovered bugs
- Debugging typically occurs when the developer encounters a problem in trying to complete a portion of the design, and then typically tests-to-pass the bug fix (meaning tests only to ensure the system minimally works under normal circumstances)
- With testing, on the other hand, bugs are discovered as a result of trying to break the system, including both testing-to-pass and testing-to-fail, where weaknesses in the system are probed.

Testing techniques

- The types of bugs encountered in testing depend on the type of testing being done.
- In general, testing techniques fall under one of four models:
  - static black box testing,
  - static white box testing,
  - dynamic black box testing, or
  - dynamic white box testing
- Black box testing occurs with a tester that has no visibility into the internal workings of the system (no schematics, no source code, etc.).
- Black box testing is based on general product requirements documentation,
- In white box testing (also referred to clear box or glass box testing) the tester has access to source code, schematics, and so on.
- Static testing is done while the system is not running, whereas dynamic testing is done when the system is running.

Cntd..

	Black Box Testing	White Box Testing
Static Testing	Testing the product specifications by: 1. looking for high-level fundamental problems, oversights, omissions (i.e., pretending to be customer, research existing guidelines/standards, review and test similar software, etc.). 2. low-level specification testing by insuring completeness, accuracy, preciseness, consistency, relevance, feasibility, etc.	Process of methodically reviewing hardware and code for bugs without executing it.
Dynamic Testing	Requires definition of what software and hardware does, includes: • *data testing*, which is checking info of user inputs and outputs • *boundary condition testing*, which is testing situations at edge of planned operational limits of software • *internal boundary testing*, which is testing powers-of-two, ASCII table • *input testing*, which is testing null, invalid data • *state testing*, which is testing modes and transitions between modes software is in with state variables i.e., race conditions, repetition testing (main reason is to discover memory leaks), stress (starving software = low memory, slow cpu slow network), load (feed software = connect many peripher-als, process large amount of data, web server have many clients accessing it, etc.), and so on.	Testing running system while looking at code, schematics, etc. Directly testing low-level and high-level based on detailed operational knowledge, accessing variables and memory dumps. Looking for data reference errors, data declaration errors, computation errors, comparison errors, control flow errors, sub-routine parameter errors, I/O errors, etc.