Developing Software for Embedded Systems*

3/7/01 Lecture #12 16.070

- Handyboards have been distributed -- Welcome to the world of programming embedded systems!
- Embedded system
  - Computer hardware, software, other parts designed to perform a specific function
  - Component within larger system - in cars, air/spacecraft
- Embedded software in almost every electronic device
  - Watches, VCRs, cellular phones, microwaves, thermostats
  - In US, ~8 μprocessor-based devices for every person
- Each embedded system is unique, with specialized hardware and specialized software

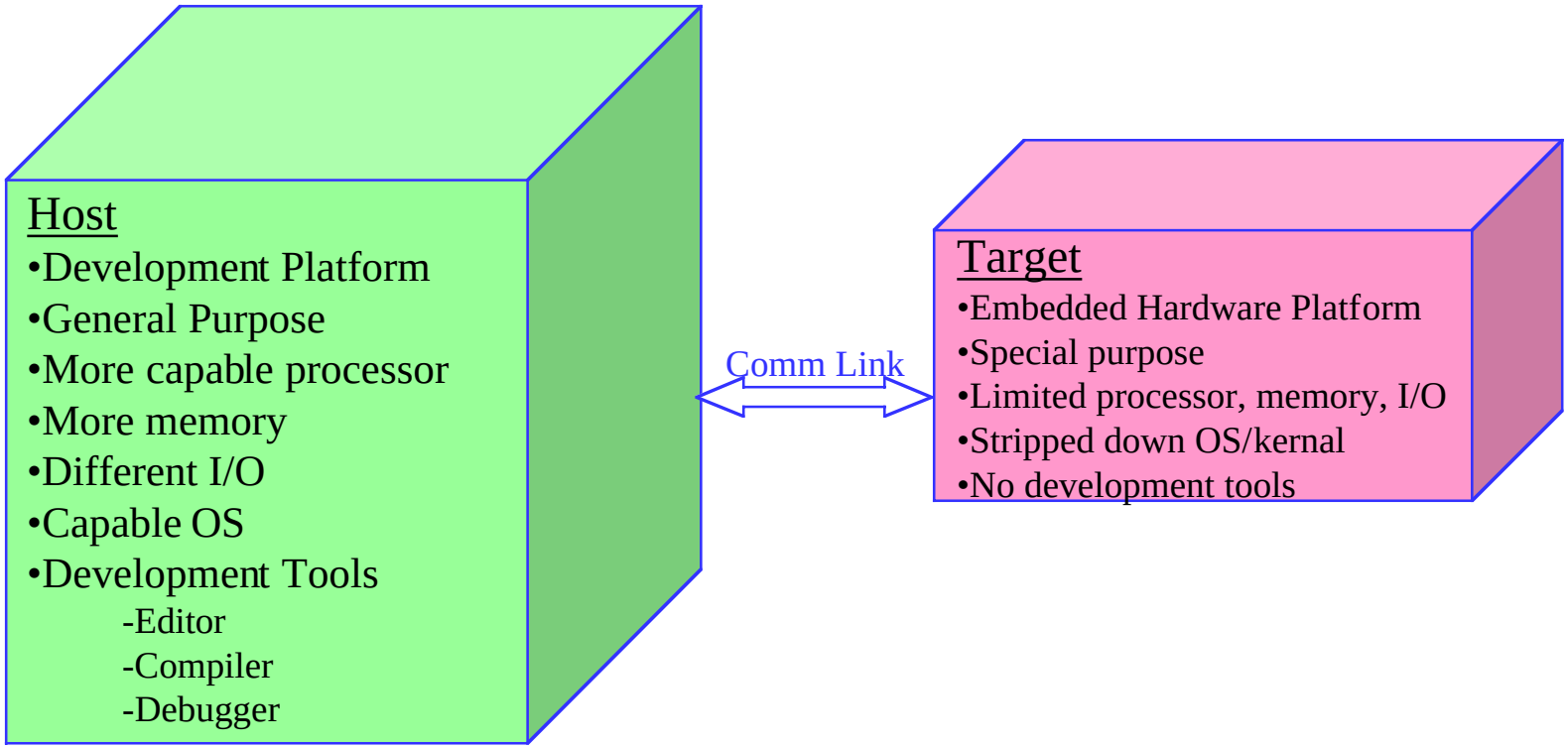
*Reference: "Programming Embedded Systems," by Michael Barr, in A/A Library

Developing Software for Embedded Systems

- Steps involved in preparing embedded software similar to general programming
  - Follow five steps in Software Design Process
  - Use Spiral Model
  - Decide on programming language (C)
  - Know capabilities/limitations of processor/compiler
- Main difference is that target hardware platform is unique
  - Leads to additional software complexity
  - Software Engineer must be aware of software build process

Embedded Software Development Process

- Software Development is performed on a Host computer
  - –Compiler, Assembler, Linker, Locator, Debugger
  - –Produces executable binary image that will run on Target Embedded System



Programming Embedded Systems

- Embedded Systems Programming requires a more complex software build process
  - Target hardware platform consists of
    - Target hardware (processor, memory, I/O)
    - Runtime environment (Operating System/Kernel)
  - Target hardware platform contains only what is needed for final deployment
  - Target hardware platform does not contain development tools (editor, compiler, debugger)
- Target hardware platform is different from development platform
  - Development platform, called the Host Computer, is typically a general-purpose computer
  - Host computer runs compiler, assembler, linker, locator to create a binary image that will run on the Target embedded system

Process for Developing Embedded Software

- To develop software for a General Purpose Computer
  - Create source file
  - Type in C code
  - Build: compile and link
  - Execute: load and run
- To develop software for an embedded system
  - Create source file (on Host)
  - Type in C code (on Host)
  - Compile/Assemble: translate into machine code (on Host)
  - Link: combine all object files and libraries, resolve all symbols (on Host)
  - Locate: assign memory addresses to code and data (on Host)
  - Download: copy executable image into Target processor memory
  - Execute: reset Target processor

Compiling Embedded Systems

- Compiler translates program written in human-readable language into machine language
  - Source Code --> Object file
  - Object file is binary file that contains set of machine-language instructions (opcodes) and data resulting from language translation process
- Machine-language instructions are specific to a particular processor
- Can the host computer's compiler be used to compile a program to run on the target computer?
- A Native-compiler runs on a computer platform and produces code for that same computer platform
- A Cross-compiler runs on one computer platform and produces code for *another* computer platform

Assemblers/Interpreters for Embedded Systems

- In some cases, a compiler is not used
  - Assembler performs one-to-one translation from human-readable assembly language mnemonics to equivalent machine-language opcodes
  - Interpreter constantly runs and interprets source code as a set of directives
    - » Performs syntax checking as program is typed in
    - » Result is slow performance - can be ~1000x slower than an equivalent compiled language
    - » Interactive capability provides more feedback -- easier to learn

Handyboard Interpreter

- Handyboard runs a C interpreter called Interactive C
  - C code is compiled into custom language
  - Custom language is interpreted by p-code
  - In interactive mode, syntax checked in IC console window
    - » E.g., To clear HB screen, type in following line:

```
printf (“\n”);
```

Linking Embedded Systems

- The Linker combines object files (from compiler) and resolves variable and function references
  - Source code may be contained in >1 file, which must be combined
  - Resolve variables which may be referenced in one file and defined in another file
  - Resolve calls to library functions, like *sqrt*
  - May include operating system
- Linker creates a “relocatable” version of the program
  - Program is complete, except no memory addresses assigned

Locating Embedded Systems

- A Locator is the tool that performs the conversion from relocatable program to executable binary image
- The Locator assigns physical memory addresses to code and data sections within the relocatable program
- The Locator produces a binary memory image that can be loaded into the target ROM
- In contrast, On General Purpose Computers, the operating system assigns the addresses at load time

Downloading and Executing Your Program

- Once a program has been successfully compiled, linked, and located, it must be moved to the target platform
- Download the binary image to the embedded system
  - Executable binary image is transferred and loaded into a memory device on target board
    - Can be loaded into ROM via a device programmer, which "burns" a chip that is then re-inserted into the embedded system
    - Handyboard must be put into bootstrap download mode first, then data can be transferred via serial port into memory
- Your program will then execute when you reset the processor, or apply power to the embedded system

Review Process for Developing Embedded Software

- To develop software for an embedded system
  - Create source file (on Host)
  - Type in C code (on Host)
  - Compile/Assemble: translate into machine code (on Host)
  - Link: combine all object files and libraries, resolve all symbols (on Host)
  - Locate: assign memory addresses to code and data (on Host)
  - Download: copy executable image into Target processor memory
  - Execute: reset Target processor

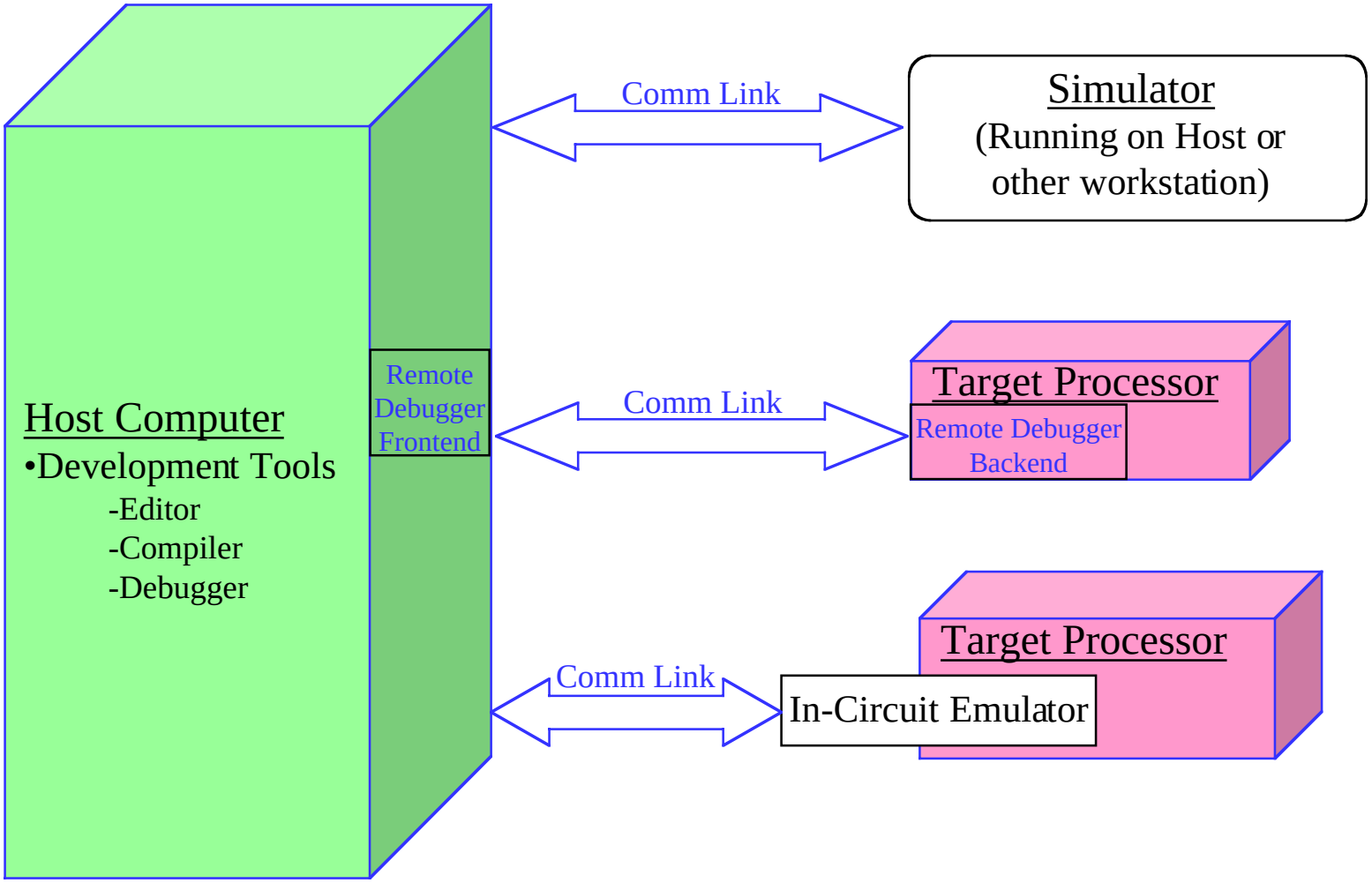
Process for Developing Handyboard Software

- To develop software for the Handyboard
  - Create source file (on Host)
  - Type in C code (on Host)
  - Download file:
    - » Compile: translate into custom interpreted language (on Host)
    - » Locate: assign memory locations (on Host)
    - » Download: transfer interpreted language via serial board to HB
  - Execute:
    - » Reset HB which runs “main” function
    - » P-code interprets custom language

Debugging Embedded Software

- Now that the software has been downloaded to the target processor, how do we know if it is working?
- Run-time errors are not as obvious
  - Most embedded systems do not have a “screen”
  - When a program fails, usually causes the processor to crash or lock-up
- Logic errors
  - If program runs, is it performing the correct steps?

Debugging Tools



Debugging with Simulators

- Simulator is host-based program that simulates functionality and instruction set of target processor
  - Front-end has text or GUI-based windows for source code, register contents, etc
  - Valuable during early stages of development
  - Disadvantage: only simulates processor, not peripherals

Debugging with Remote Debuggers

- Remote Debuggers used to monitor/control embedded SW
  - Used to download, execute and debug embedded software over communications link (e.g., serial port)
  - Front-end has text or GUI-based windows for source code, register contents, etc
  - Backend provides low-level control of target processor, runs on target processor and communicates to front-end over comm-link
  - Debugger and software being debugged are executing on two different computer systems
  - Supports higher level of interaction between host and target
    - » Allows start/restart/kill, and stepping through program
    - » Software breakpoints ("stop execution if instruction X is fetched")
    - » Read/write registers or data at specified address
  - Disadvantage: Requires target processor to run more than final software package

Debugging with In-Circuit Emulators

- In-Circuit Emulators (ICE)
  - Take the place of (i.e., emulates) target processor
  - Contains copy of target processor, plus RAM, ROM, and its own embedded software
  - Allows you to examine state of processor while program is running
  - Uses Remote debugger for human interface
  - Has more capability than target processor
    - » Supports software and hardware breakpoints (stop execution on memory and I/O read/write, interrupts) "Stop on write to variable `num`"
    - » Real-time tracing
      - = Stores information about each processor cycle that is executed
      - = Allows you to see what order things happened
  - Disadvantage: Expensive!

Debugging on the Handyboard

- None of these debugging tools is available for the Handyboard
- What is available?
  - "Simulate" program by running in Visual C first, using Visual Studio Development Environment
  - Use Handyboard screen (unusual capability for a target processor!)
  - Use beeps/LEDs to identify points reached in program
  - Watch heartbeat