Embedded Applications

COMP595EA
Embedded Applications
Development
Embedded Development

- There exists a tension inherent to embedded development:
  - Embedded hardware and sensors require testing on target platform
  - Limited resources and human interfaces prevent programming and development on embedded platforms
- The development is split between host and target machines.
Host

- A typical, general purpose workstation used for:
  - programming
  - compiling
  - (testing/emulation)
- Vast computing resources support sophisticated IDEs, compilers and debugging environments.
Target

• The Target machine is the embedded platform selected and supports:
  – testing
  – integration
  – Calibration
  – execution
  – deployment
Native vs Cross-compilers

- **Native tools:**
  - Compiler, assembler, linker and debugger
  - Take source and produce executables that run on the same machine the compilation took place on.

- **Cross-Compiler:**
  - Produce instruction codes and formats for execution on a foreign platform.

- **Collection of tools is called a “tool chain”**
Compilation Problems

- Code that compiles natively may not cross-compile for the target system.
  - usage of libraries or system calls that do not exist.
  - memory or other resources may not permit it.
Linkers / Locators

• Native Linkers
  - produce objects designed to be loaded at runtime
  - produce relative or symbolic memory locations
  - resolution of addresses is performed by the loader

• Locators (linkers for embedded platforms)
  - produce stand alone executable
  - Don't have the advantage of a loader (ever)
  - Handle allocation and placement of memory.
Locator Complexities

- Locators have to handle memory
  - Determine what goes in ROM
  - Determine what goes in RAM
- Most tool chains divide memory into segments
  - segments are allocated to either ROM or RAM
  - segments can be placed independently of each other
- Locators must ensure the placement of the first instruction and handle start-up code placement.
Cross-Compiler vs Cross-Assembler

• Cross-Compiler
  – usually handles all of the locator complexities automatically with reasonable default behavior

• Cross-Assembler
  – Programmer is god.
  – What you write is what you get
    • (even if it's a platypus or sea cucumber)
Initialized Data

- Initialized Data presents a problem for embedded locators concerning memory placement

- Take for example:
  ```c
  static int freq=2410;
  ```

  - Where should the data be placed?
    - ROM: because it has to persist and re-initialize across reboots?
    - RAM: because it needs to be changed or written to?
Initialized Solution

- Most Locators solve the problem by creating “shadow” segments in ROM
- Initialized values are written to the shadow segment
- Code at start-up copies the shadow segment to the target segment in RAM
  - Locator must produce addresses for the RAM locations and not the ROM locations where the data starts.
Initializing Memory

- Extra Care should be taken on embedded platforms to zero all data used.
- Most typically memory values start with whatever random values were in memory when the system started.
- Some tool-chains take care of preinitializing data; some do not.
Constant Strings

- Constant strings are also a problem similar to pre-initialized data.
  - `char *Msg = “Reactor is melting!”;`

- ROM or RAM again?

- Array boundaries!!
  - Thousands of problems wouldn't exist if programmers would test their array boundaries.

- Array boundaries!! (In case you missed that point)
• No, really...
  – Array boundaries!!
Locator Maps

• Unlike native tools, Locators typically produce a “map” of where it placed things in memory

• Useful for:
  - verifying that the tool-chain produced a suitable executable format.
  - debugging
Executing out of RAM

• RAM is typically faster than ROM
• Executing code residing in ROM is therefore slower.
  – Many microprocessors don't suffer this penalty due to ROM being faster then the processor
• If code was executed from RAM a performance increase can be achieved.
RAM Execution

• This can be achieved by:
  – Use a small start-up code routine to copy contents of ROM to RAM
  – Switch execution pointer to RAM location

• Locator needs to produce proper addresses referring to RAM locations.

• Code can be compressed / use less space
Transferring to Target

• Once the executable is created by the Cross-platform tool chain it needs to be transferred to the target machine.

• Several method exist for accomplishing this
PROM programmers

- Chip is inserted into special equipment
- Host machine transfers executable to parallel/serial port to equipment
- Equipment electrically programs the chip
- Chip is reinserted / soldered to target platform

- Very inexpensive, time consuming
- Difficult to upgrade program later.
ROM Emulators

• ROM emulator
  – An electronic device hooked up to both the host and target machines.
  – Host machine transfers program to ROM emulator device
  – ROM emulator device acts just like ROM attached to the target platform

• more costly

• Fast turn around times / better debugging
Flash

- Programmable read only memory that can be programmed in place without a dedicated programmer.

- Can programmed from a serial port / software

- Allows field upgrades

- costly components / complicated programming

- Flexible / upgradable
Monitors

- **Monitor**
  - A small piece of software or hardware that listens to a serial port and installs any incoming program into RAM and then begins execution
- Useful for debugging.
- Programs do not survive power cycles.
- Not suitable for production