EZ-OTG / EZ-Host Boot Code Design

Overview

The EZ-OTG and EZ-Host tools are a port of the Gnu development suite to the EZ-OTG / EZ-Host platform. The compiler's default output format is ELF. ELF is a loadable file format suitable for use on a general-purpose computer system such as a Linux installation. However it is not generally suitable for use in an embedded system because of the overhead required to load it and because of the general-purpose code that is linked in by default. This extra, general-purpose code, adds several kilobytes of often-unused code to the executable image. By eliminating the default startup code, and the general-purpose code, it is possible to drastically reduce the size of the executable image.

Default Gnu Startup Code

The default Gnu startup code is generally contained within crt0.s. The functions within this file link to general purpose memory management routines that are rarely used in a small embedded environment, as well as atexit() and similar functionality that is never used in a small embedded environment. This code can usually be stripped away which results in a smaller executable image with no loss of functionality.

EZ-OTG / EZ-Host Development Kit Startup Code

The EZ-OTG / EZ-Host Development Kit startup code has been reduced to the bare minimum necessary to establish a C operating environment and call the main function. No cleanup occurs if main ever returns (it shouldn't for an embedded system), a "hang" loop will be entered and execution will remain in this loop until reset or power down.

All that is required to establish a C operating environment is to call the main function. We will share the stack space with the BIOS so no code is required to establish a stack. Our code does not have any data in the .bss (uninitialized data) section, so nothing is required to zero out the .bss. If your project requires a .bss section, you must add a small amount of code to the startup code (prior to calling the main function) that zeroes out the .bss section. The address and size of the .bss section can be determined from linker variables in the linker script.

Using Custom Startup Code

To use the custom startup code contained within startup.s, you must do two things. First tell the compiler/linker that you will not be using the default startup code. If you use `cy16-elf-gcc` (the compiler driver) to perform the link phase, you must add the `–nostartfiles` argument on the `cy16-elf-gcc` command line. Next, you must tell the linker to use the startup.o file as the first file to be linked. Do this simply by naming startup.o as the first object file on the command line.

Listing

Here is a listing of the startup code for the OTG-Host DVK Design Examples:

```
; File: startup.s

.include "fwxcfg.inc"
.include "memmap.inc"

; Declare the main function.

.global main
.section .text

; Declare the start symbol

.global _start
      ; Call the main function.
      call  main

.global _exit
_exit:
      mov   r1, 0x0
      mov   r2, 0xdead
      mov   r3, 0xbeef

      ; Enter a hang loop.

hang:
      mov   [hang_data], r2
      mov   r0, r1
      mov   [hang_data], r3
      inc   r1
      jmp   hang

.section .data
      hang_data : .short 0

; End of file: startup.s
```