XAPP482 (v2.0) 2005 年 6 月 27 日

MicroBlaze Platform Flash/PROM 引导加载器和用户数据存储

作者：Shalin Sheth

提要

本应用指南讲述一种实用的 MicroBlaze™ 系统，用于在非易失性 Platform Flash PROM 中存储软件代码、用户数据和配置数据，以简化系统设计和降低成本。另外，本应用指南还介绍一种可移植的硬件设计、一个软件设计以及在实现流程中使用的其他脚本实用工具。

简介

许多 FPGA 设计都集成了使用 MicroBlaze 和 PowerPC™ 处理器的软件嵌入式系统，这些设计同时使用外部易失性存储器来执行软件代码。使用易失性存储器的系统还必须包含一个非易失性器件，用来在断电期间存储软件代码。大多数 FPGA 系统都在电路板上使用 Platform Flash PROM（在本文中称作 PROM），用于在上电时加载 FPGA 配置数据。另外，许多应用还可能使用其他非易失性器件（如 SPI Flash、Parallel Flash 或 PIC）来保存 MAC 地址等少量用户数据，因此导致系统电路板上存在大量非易失性器件。

本应用指南演示了如何减少系统电路板上所需非易失性器件的数量，以及如何使用一个 PROM 来存储 FPGA 配置数据、软件代码和用户数据。本应用指南论及以下概念：

- 从 PROM 中加载应用软件
- 介绍在 PROM 中存储多数据块的方法，如图 1 所示
- 为引导加载等应用构建最小 MicroBlaze 存储器系统
- 介绍使用 C 代码动态重写复位、中断和异常向量的方法
- 定义将软件和用户数据加入 PROM 文件的软件流程

本应用指南是针对低成本 MicroBlaze 嵌入式处理器核编写的，但也可用于任何使用通用输入 / 输出端口的 8 位、16 位或 32 位微控制器。

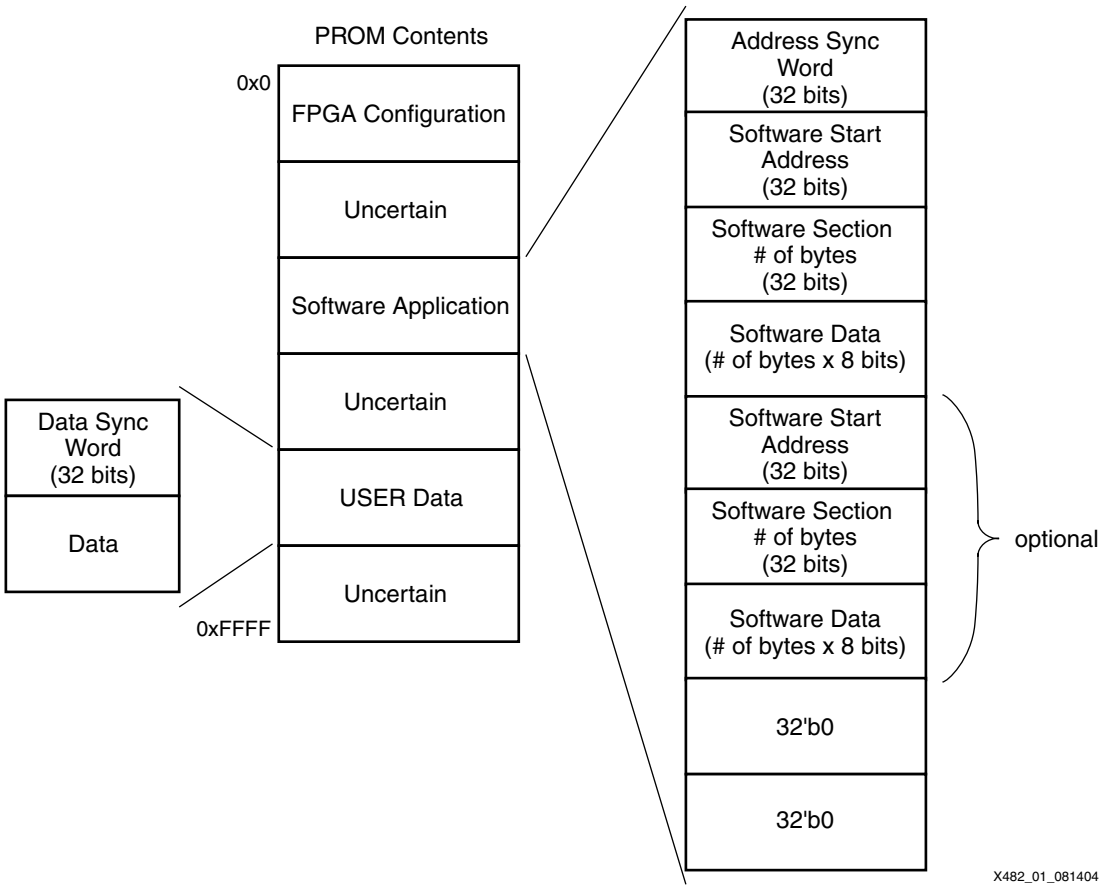

图 1: **在 PROM 中存储多数据段的方法**

图 1 所示为 PROM 在存储多数据段时的内容。软件应用段可处在 PROM 中的任意位置，用地址同步字标识。跟在地址同步字后面的是一个 32 位软件起始地址、32 位软件段（指定后面软件数据的字节数），然后是实际的软件数据。软件起始地址、字节数和其他软件数据可以在同一软件应用中多次重复。软件应用段的末尾用两个等于零的 32 位字标识。用户数据段由用户同步字定义，同步字之后紧随用户数据。由于介于任意 FPGA 配置数据、软件应用或用户数据之间的数据不确定，因此需要使用同步字。

电路板考虑事项

为了能在配置 FPGA 之后读 PROM，设计系统电路板时必须考虑某些要求。本部分介绍主串配置连接，并说明使用必要连接的原因。

图 2 所示为主串配置方法中所需的电路板连接。有关详情，请参阅 XAPP694，其中的电路板考虑事项与本参考设计的相同。

Xilinx Spartan™–3 入门套件电路板提供了一个实用示例，其中有这些电路板的考虑事项。UG130 中有此电路板的说明及其原理图。

有关 FPGA 配置的更多信息，请参阅 XAPP501 和 XAPP138。

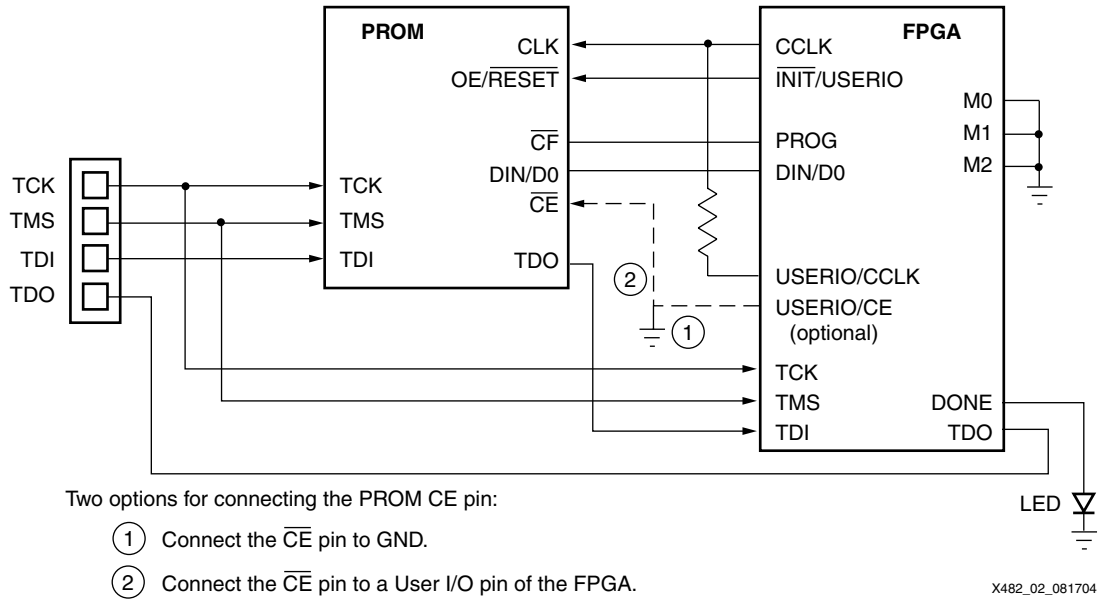

图 2: **电路板考虑事项**

PROM 的 $\overline{CE}$ 引脚

PROM 的 $\overline{CE}$ 引脚通常与 FPGA 的 DONE 引脚连接，用于在配置 FPGA 之后将 PROM 保持在待机模式。用户可以用此引脚启用或禁用 PROM，并且在不准备访问 PROM 时降低功耗。不过，如果连接了 DONE 引脚和 PROM 的 $\overline{CE}$ 引脚，就不能在配置 FPGA 之后读 PROM。

关于如何连接 PROM 的 $\overline{CE}$ 引脚，有两种可选方案：

1. 将 $\overline{CE}$ 引脚接地。
2. 将 $\overline{CE}$ 引脚连接 FPGA 的一个用户 I/O 引脚。此选项需要为解决方案附加一个 I/O 引脚，但允许将 PROM 置为待机模式以降低功耗。软件通过驱动此引脚来启用或禁用 PROM。Platform Flash 的最大待机电流是 1 mA，而其最大有功电流是 10 mA（有关详情，请见 DS123）。

当 FPGA 的 DONE 引脚未连接 PROM 的 $\overline{CE}$ 引脚时，可以将 FPGA 的 DONE 引脚连接至外部 LED，以显示 FPGA 配置完成的状态（请见图 2）。

PROM 的 CLK 引脚

连接 FPGA 的一个附加用户 I/O 引脚，用来驱动 PROM 的 CLK 输入。本参考设计必须使用此连接，因为在任何主配置模式下 FPGA 生成的配置时钟 CLK 都会在成功配置 FPGA 之后停止翻转，以防止 PROM 的地址计数器的操作超出 PROM 中存储的 FPGA 设计。当在配置 FPGA 之后读 PROM 时，附加的用户 I/O 为 PROM 的 CLK 引脚提供时钟。此连线上有一个 390Ω 的电阻，以避免 CLK 信号的两个可能的驱动器之间发生冲突。

PROM 的 OE/$\overline{RESET}$ 引脚

将 PROM 的 OE/$\overline{RESET}$ 引脚连接至 FPGA 的 $\overline{INIT}$ 引脚，以便在配置过程发生 CRC 错误时让 FPGA 重新开始配置。$\overline{INIT}$ 引脚在配置后成为一个用户 I/O，因此可以配置成输出逻辑 “高” 信号，以使 PROM 的输出保持有效。

PROM 的 DIN/D0 引脚

将 FPGA 的 DIN/D0 引脚连接 PROM 的 DIN/D0 引脚，以便将 PROM 中的数据读入 FPGA。这不是此应用的专用连接，DIN 引脚在配置后不可用作用户 I/O。

硬件设计

为了实现此参考设计，在嵌入式开发套件 (EDK) 中使用了一个 MicroBlaze 系统。硬件核构建在一个简单的片上外设总线 (OPB) 通用输入 / 输出 (GPIO) 核上，该 GPIO 核用来控制 $\overline{\text{INIT}}$、$\overline{\text{CE}}$、OE 和 DIN 引脚，“电路板考虑事项”中叙述了这些引脚。图 3 所示为硬件系统框图。

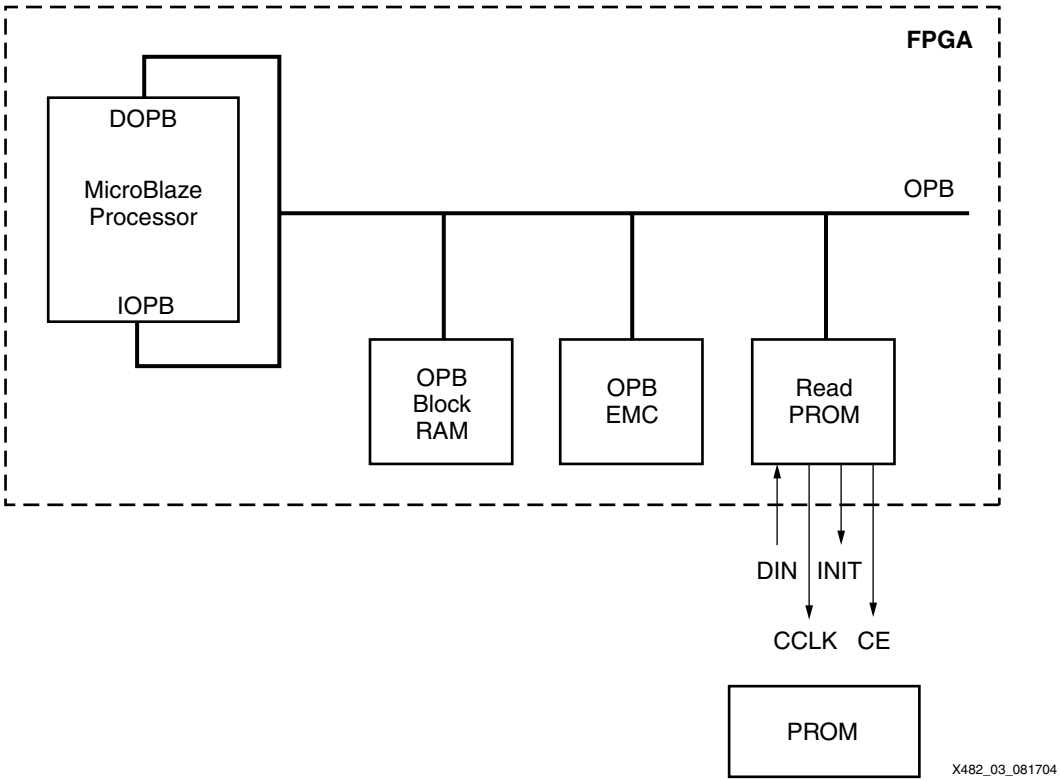

图 3: **MicroBlaze 硬件系统框图**

PromRead GPIO 核使用 Spartan–3 器件中的 26 个四输入查找表 (LUT) 和 61 个触发器。另外，此参考设计还使用一个定制 OPB Block RAM 接口控制器核，其中只用一个 Block RAM 来说明最小系统。EDK 7.1i 中的最小系统始终使用四个 Block RAM。在不需要这么多块存储器的优化系统（如引导加载器）中，用一个定制 Block RAM 接口控制器核来创建具有一个 Block RAM 的系统。

固件设计

本应用的复杂性在于固件设计。PROM 的控制是通过 C 软件程序进行处理。配置 PROM 是不可寻址存储元件，其中所有数据都是串行输出并由软件系统读取。

驱动软件基本原理

promread 函数处理与 PROM 的交互操作：

```
Xuint32 promread (Xuint8 read)
```

表 1 列出了读 PROM 的两种模式。在地址段读中，该函数将软件代码从 PROM 复制到 PROM 中的地址同步字（默认为 0x9F8FAFBF）之后描述的存储器位置。在数据字读中，读取数据同步字（默认为 0x8F9FAFBF）之后的第一个 32 位数据字。

表 1: **promread 函数描述**

函数	输入（读）	输出（返回）
地址段读取	0x1	始终是 0x0
数据字读取	0x0	数据同步字之后的 32 位数据字

图 4 所示为这两种函数的用法示例。

```
#define DATAREAD 0x0
#define ADDRREAD 0x1

//to copy the contents from the PROM to a memory location
  promread(ADDRREAD);

//to return a 32-bit word stored after the sync word.
  xil_printf("\n\rData %x\n\r", promread(DATAREAD));
```

图 4: **promread 函数的用法示例**

定制实用工具 `xapp482.exe` 和 `xapp694.exe` 用来填充 MCS 文件（MCS 是 Xilinx 的文件扩展名，表示 Intel 扩展十六进制格式文件）。有关 Perl 脚本和更新实用工具的更多信息，请见“用法 / 流程”。“驱动软件细节”部分讨论如何构建 PROM 数据。

驱动软件细节

理解固件设计，关键是理解从 PROM 串行输出的 PROM 内容。第 2 页图 1 所示为如何在 PROM 中存储数据。

地址读

当指示地址读（promread 输入 = 0x1）时，软件每次读完 PROM 中的 32 个位，直到发现与地址同步字匹配的 32 位字。有关如何从 PROM 中读取数据的信息，请见“读 PROM”。默认的地址同步字是 0x9F8FAFBF；不过，可以在 `promread.h` 头文件和用来填充 MCS 文件的 Perl 脚本中对其进行修改。在创建 PROM 文件时，可以确认同步字的唯一性（有关更多信息，请参见“用法 / 流程”）。

图 5 所示为如何在 PROM 中查找数据。找到地址同步字之后，紧跟其后的是一个 32 位字，在处理器的存储器映射中表示软件代码存储位置的起始地址。此地址后面是起始地址之后存储的字节数。软件从 PROM 中如数读取字节数，并且将数据复制到在该块起始处指定的地址。第一个地址段复制完成后，软件读取两个 32 位字。如果这两个字之一的值大于零，则第一个 32 位字是下一个软件数据段的起始地址，而第二个字是下一个地址段中的字节数。软件继续读地址段，直到达到两个 32 位字都等于零的地址结束序列，如第 2 页图 1 中所示。promread 函数继续读 PROM 并且如本段落所述复制地址段，直到从 PROM 中读出 0xFFFFFFFF 处的

END_PROM 字。然后，promread 函数返回 0x0。0xFFFFFFFF 的值表示已经达到 PROM 中的空白数据的起点。

```
9F8FAFBF  80180000  00007100  BA101056
--------  --------  --------  --------
address   memory    number    data
sync      mapped    of        …
word      starting  bytes
          address
```

图 5: **PROM 的软件段内容示例**

数据读

当指示数据读（promread 输入 = 0x0）时，软件每次读完 PROM 中的 32 位，直到发现与数据同步字匹配的 32 位字。默认的数据同步字是 0x8F9FAFBF；不过，可以在 `promread.h` 头文件和用来填充 MCS 文件的 Perl 脚本中对其进行修改。在创建 PROM 文件时，可以确认同步字的唯一性（有关更多信息，请参见 “用法 / 流程”）。一旦找到数据同步字，则 promread 函数返回数据同步字后面的第一个 32 位字。如果还需要其他数据，就必须根据您的要求修改 promread 函数的数据检索段。

读 PROM

为了在软件中读 PROM，MicroBlaze GPIO 翻转 PROM 的 CLK 引脚。$\overline{\text{INIT}}$ 引脚必须为 “高”，而 CE 引脚为 “低”，才能读 PROM。PROM 输出的每个字节都经过位交换。

```
//clock the PROM to output data
XIo_Out32(XPAR_PROMREAD_BASEADDR, OE_HIGH | CCLK_HIGH | CE_LOW);
XIo_Out32(XPAR_PROMREAD_BASEADDR, OE_HIGH | CCLK_LOW | CE_LOW);
```

图 6 所示为如何对每个字节进行位交换。加载 PROM 的 Perl 脚本在将位元载入 PROM 之前先进行位交换，这样才能按正确顺序从 PROM 中读回数据。

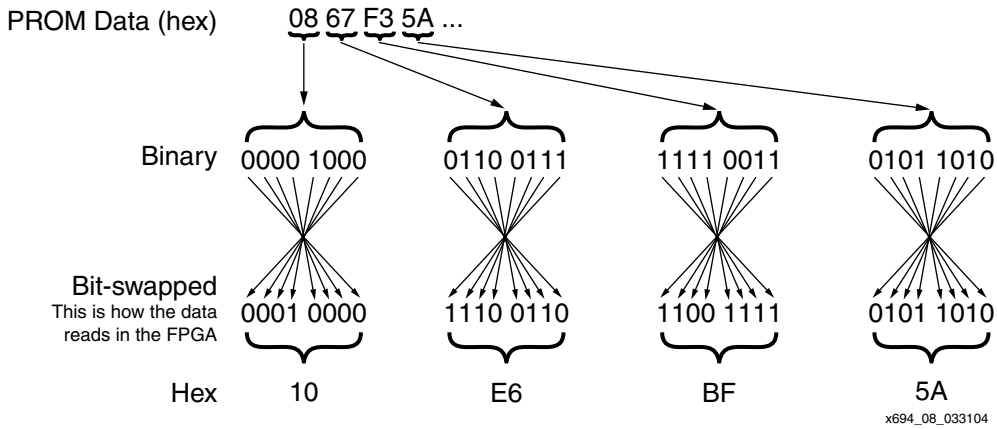

图 6: **对 PROM 的输出进行位交换**

请注意，如果将 MCS 文件与从 PROM 读入的内容进行比较，则在 MCS 文件中进行位交换。

表 2 是真值表，说明 PROM 控制输入如何从 PROM 中获得数据输出。

表 2: **PROM 控制输入真值表**

控制输入		内部地址	输出
OE/$\overline{\text{RESET}}$	$\overline{\text{CE}}$		DATA
高	低	如果地址 <= TC：递增 如果地址 > TC：不变	有效
低	低	保持复位	高 –Z
高	高	保持复位	高 –Z
低	高	保持复位	高 –Z

固件性能

引导操作的性能缓慢，因为数据是串行读取，而且 PROM 的时钟是在软件中生成。访问 PROM 的时间还受到 PROM 中所存储比特流大小的影响。如果 PROM 中存储的比特流较大，promread 函数就要花较长时间来全程解析 PROM，直到在 PROM 中找到软件或用户数据段。

在以 50 MHz 运行的 Spartan–3 入门套件电路板上进行的实时性能标准测试发现，对于 1 Mb 比特流，全程解析 PROM 大约需要两秒。

promread 函数的软件代码大小是 0x344 （即 836）字节。

双软件工程

在许多嵌入式系统中，设计人员使用链接器脚本将软件代码段分入不同存储器。另一种方法是根据所执行代码使用多个软件工程。本参考设计使用双软件工程概念划分引导加载器软件和应用软件。引导加载器软件在 Block RAM 中驻留并从中执行，而应用软件在 SRAM 中驻留并从中执行。必须将 SRAM 中的程序设置成不会初始化到 Block RAM 中。在引导时，引导加载器将数据从 PROM 复制到 SRAM。复制完成后，引导加载器跳转到 SRAM 的起始处，开始执行应用软件。向 SRAM 的跳转是用函数指针完成，如 “修改 C 程序中的程序计数器 (PC)”中所述。

图 7 所示为在 EDK 7.1i 中设置双软件工程。

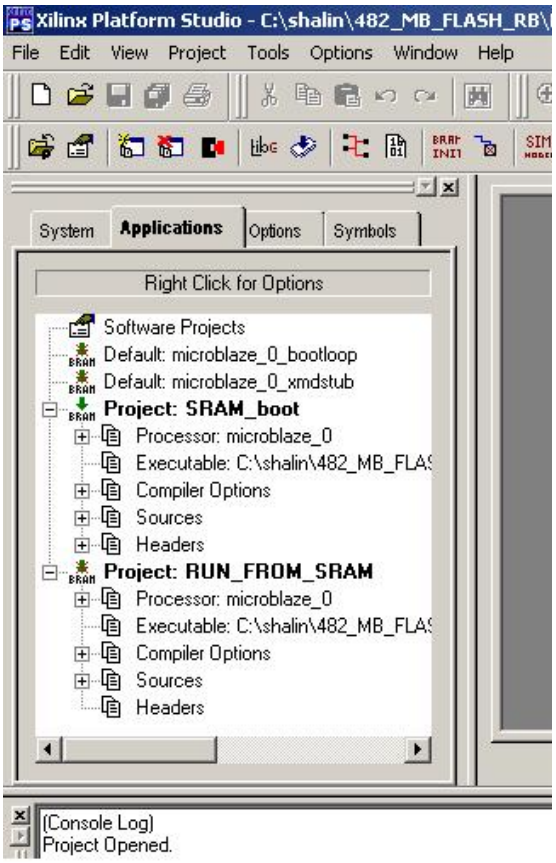

图 7: **双软件工程方法**

修改 C 程序中的程序计数器 (PC)

在引导加载器末尾处，通常是从引导程序空间跳转到另一地址位置，以开始执行应用程序指令。在这样的示例中，使用汇编指令即可修改 PC；不过，如果您的软件是用 C 语言编写的，这样修改就会导致语言混杂。简单的解决办法是使用以 C 语言编写的函数指针，如图 8 所示。必须在引导加载器中将 PROG_START_ADDR 设置到应用软件的起始地址。

```
//declare before main()
// Function point that is used at the end of the program
// to jump to the address location stated by PROG_START_ADDR
#define PROG_START_ADDR 0x80180000
int (*func_ptr) ();


// declare after main()
// function point that is set to point to the address of
// PROG_START_ADDR
func_ptr = PROG_START_ADDR;
// jump to start execution code at the address
// PROG_START_ADDR
func_ptr();
```

图 8: **用来创建汇编语言跳转指令的 C 代码函数指针**

图 9 所示为如何将图 8 中的 C 代码转换成 MicroBlaze 汇编语言。

```
  84     func_ptr = PROG_START_ADDR;
- 0xe4 <main+16>:     imm    -32744
- 0xe8 <main+20>:     addik  r3, r0, 0
- 0xec <main+24>:     swi    r3, r0, 1808// 0x710 <func_ptr>
  86     func_ptr();
- 0xf0 <main+28>:     brald  r15, r3
- 0xf4 <main+32>:     or     r0, r0, r0
```

图 9: **函数指针的反汇编**

通过修改引导代码来缩小代码（可选）

本部分所述缩小代码的方法是删除由 EDK 工具为名为 SRAM_boot 的引导加载器软件工程插入的默认 C 运行时例程 (CRT) 文件。此优化是 *可选步骤*，且仅当要缩小软件代码时才需要使用，它要求引导加载过程中不需要中断和异常。如果使用优化，则中断和异常处理程序仍可保留在 RUN_FROM_SRAM 软件工程中，一经引导加载便可从 SRAM 运行。不过，如 “重新设计复位、异常和中断处理程序”中所述，还需要附加步骤，以便将处理器设置成访问这些处理程序。有关软件初始化文件的更多信息，请参阅《嵌入式系统工具指南》。

此参考设计中附带初始化文件 `init.s`。要修改此初始化文件，请按以下步骤进行：

1. 将文件 `init.s` 添加到软件工程。
2. 添加链接器脚本 `bootlinker.scr`。
3. 将编译器选项修改成禁用初始化文件的自动插入功能。

请注意，将文件 `init.s` 添加到软件工程之后，如果在链接器脚本和编译器选项中设置了特定选项，则工具会自动使用汇编器和链接器来编译和链接文件。

参考设计中包括链接器脚本 `bootlinker.scr` 的一个示例，可将其用于引导加载器软件工程。在提供的链接器脚本中，初始化文件保留 2 KB (0x7FF) 存储器空间。此示例占用 2 KB（0x0 到 0x7FF）存储器空间。必须根据设计的微处理器硬件规范 (MHS) 文件修改此存储器空间的大小，使其与每个具体设计中的存储器空间相匹配。在此链接器脚本中，.boot 段的内容放置在 0x0 到 0x7FF 存储器空间中。

需要对编译器进行以下两项设置，以使其能捕捉对初始化文件的修改：

1. `-nostartfiles` 设置项告诉汇编器不要包括默认的初始化文件。如果忽略此选项，在将该等初始化文件包括在软件工程中之后，链接器内就会出现一个多段声明。
2. `-save-temps` 设置项允许链接器从汇编文件 `init.s` 中获取处理程序。

图 10 所示为如何设置编译器选项。

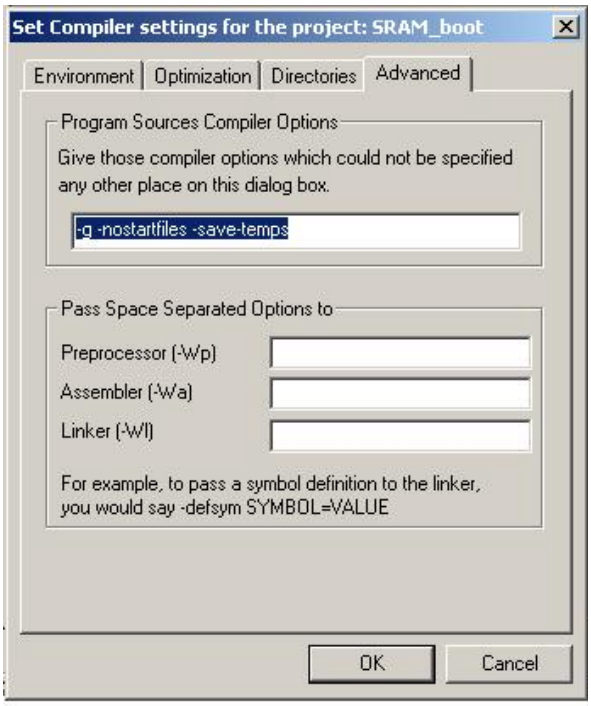

图 10: **用来替换 CRT 文件的编译器修改**

重新设计复位、异常和中断处理程序

在双软件工程系统中，将软件工程移入 SRAM 之后，您可能希望访问 SRAM 中存储的中断和异常处理程序。在 MicroBlaze 系统中，通常 Block RAM 是映射到 0x0 的存储器，而 SRAM 则驻留在存储器映射中的其他某个位置。不过，在默认情况下，MicroBlaze 系统会在出现某个复位时跳转到地址 0x0，或者在出现某个中断时跳转到地址 0x10。MicroBlaze 的默认处理程序地址如下：

- 复位：0x0
- 异常：0x8
- 中断：0x10

要解决此问题，可以用汇编语言在默认的 MicroBlaze 处理程序处编写跳转例程，以跳转到 SRAM 中软件处理程序的位置，如图 11 所示。获取软件处理程序，然后在 0x0、0x8 或 0x10 处（分别对应复位、异常和中断处理）插入跳转指令。请注意，这种方法需要动态修改指令，这对于某些软件设计人员可能是无法接受的；不过，这是保持引导加载器和软件代码分离的一种解决方法。对于本应用指南中所述方法，其好处是引导加载器不需要对应用软件有任何了解；不过，如果引导加载器知道中断和异常处理程序在 SRAM 中的位置，则不必在应用软件中动态修改软件。

```
//insert before main()
extern int _start;
extern int _exception_handler;
extern int __interrupt_handler;

//=====================================================

//insert after main() and after variable declarations
int x = &_start;
*(int*)(0x0) = 0xb0000000 | (((x-1) & 0xFFFF0000) >> 16 );
*(int*)(0x4) = 0xb8000000 | (((x-1) & 0xFFFF));

x = &_exception_handler;
*(int*)(0x8) = 0xb0000000 | (((x-1) & 0xFFFF0000) >> 16 );
*(int*)(0xB) = 0xb8000000 | (((x-1) & 0xFFFF));

x = &__interrupt_handler;
*(int*)(0x10) = 0xb0000000 | (((x-1) & 0xFFFF0000) >> 16 );
*(int*)(0x14) = 0xb8000000 | (((x-1) & 0xFFFF));
```

图 11: **为 MicroBlaze 系统重新编写复位、异常和中断处理程序的动态软件**

用法 / 流程

创建 Platform Flash/PROM 引导加载器需要使用定制脚本和流程。本部分介绍用来完成此任务的脚本的流程和用法。本部分还讲述如何用可执行链接格式 (ELF) 文件的内容填充 MCS 文件 (以使软件从 SRAM 运行)，或者用用户数据填充 MCS 文件。表 3 列出了可以通过参考设计中提供的定制脚本使用的功能。

表 3: **所提供实用工具的功能**

格式内容	MEM 地址和数据
大小 / 16 字节数据 → 存储的二进制代码	16 字节
寻址块开销	8 字节
存储的起始地址	否
存储的地址位置	是
校验和	否
引导代码	中
支持多 ELF 文件	是
流程	三步： gcc → Data2MEM → xapp482.exe

创建 MCS 文件

所有流程均以 MCS 文件开始。可以用 iMPACT 或 promgen 创建 MCS 文件。请参阅有关如何创建 MCS 文件的相应技术文档。

向 MCS 文件添加软件段

图 12 所示为向 PROM 文件添加代码的软件流程。

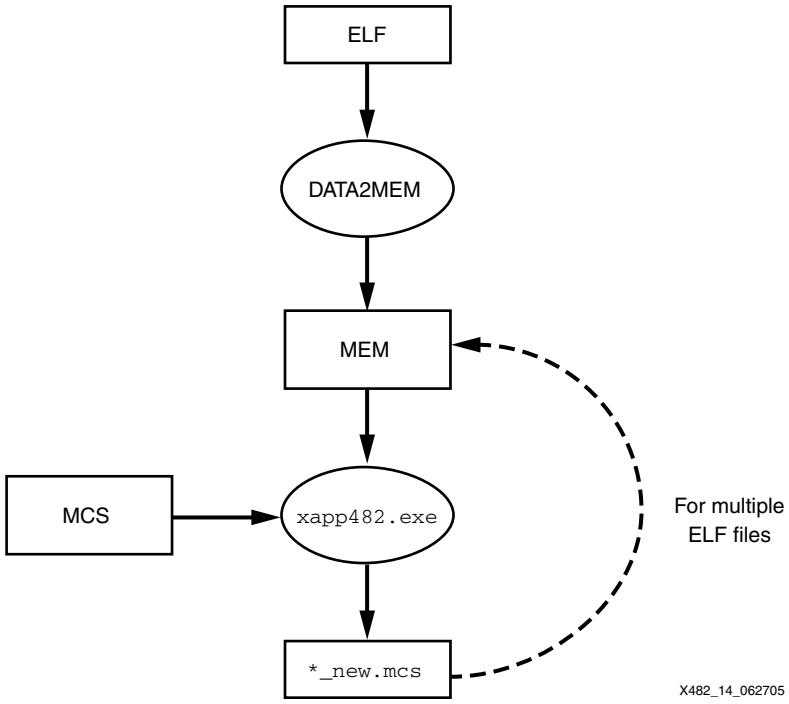

图 12: **向 PROM 文件添加应用软件的流程**

代码编译成从 SRAM 执行之后，将 ELF 文件输入 Data2MEM，以输出一个 MEM 文件。经加密的 ELF 文件格式转换成十六进制 MEM 文件，供所提供的 Perl 脚本使用。从 ELF 文件创建 MEM 文件的命令行是：

```
Data2MEM -bd *.elf -d -o m *.mem
```

有关运行 Data2MEM 的详情，请参阅《开发系统参考指南》。

下一步是使用所提供的实用工具将 MEM 文件的内容与 MCS 文件合并。

```
xapp482 *.mem *.mcs new*.mcs [syncword]
```

以上命令行的输出是 `new*.mcs`，可用此输出为 PROM 编程。如果未指定同步字，则使用默认的同步字 0x9F8FAFBF。可以重复以上各步骤来向 MCS 文件添加其他地址段。如果在输入的 MCS 中发现同步字的实例，实用工具会发出警告。

向 MCS 文件添加用户数据段

向 MCS 文件添加用户数据段的命令行是：

```
xapp694 user_data.txt input.mcs output.mcs [-noswap]
```

用户必须填充文件 `userdata.txt`，并且要保持符合以下具体要求：

1. 各数据行必须是 16 字节长。
2. 各数字必须以十六进制码表示。
3. 要添加注释，请在注释行前插入一个 “#”。

4. 请在数据段起始处放置一个同步字。在以下示例中，默认的同步字是 0x8F9FAFBF。

```
#This is data block 0
#The sync pattern is 8F9FAFBF
#The data is ASCII code for:
#XAPP 694 DATA BLOCK 0
#01234567890123456789012345678901
8F9FAFBF58415050203639342044415 4
4120424C4F434B203000000000000000
```

请注意， xapp694 实用工具不检查同步字。默认情况是先交换用户数据，然后填充输出的 MCS 文件，如第 6 页图 6 所示。要禁用交换，用户必须启用 -noswap 开关。

MCS 更新实用工具注意事项

上面讲述了 MCS 更新实用工具的使用模型。请务必注意，不要向 PROM 添加过多用户定义的数据，否则配置工具会拒绝 PROM 文件。要选择能同时存储 FPGA 配置数据和用户定义数据的 PROM，只需将用于 FPGA 配置数据的位数加上用户定义数据、软件代码和同步字开销的位数。可以在相应的 FPGA 数据手册中查到用于 FPGA 配置数据的位数。

结论

本应用指南讲述了为了在配置 FPGA 之后读 PROM 所需的板级修改、在 PROM 中保存多数据流的方法、用来从 PROM 中读取用户数据的软件、软件系统的引导加载方法以及针对引导加载器优化 MicroBlaze 硬件和软件系统的方法，最后还讲述了允许将软件和用户数据加入 PROM 文件的软件流程。这些方法都用来帮助降低已部署 MicroBlaze 系统的总体系统成本。

设计资源

可以通过以下链接下载本应用指南所述参考设计：

http://www.xilinx.com/cn/bvdocs/appnotes/xapp482.zip

参考文献

下列 Xilinx 技术文档可作为辅助资料与本应用指南配套使用：

1. XAPP694：“从配置 PROM 读取用户数据”
2. XAPP501：“配置快速入门指南”
3. XAPP138：“Virtex FPGA 系列配置和读回”
4. UG130：《Spartan–3 入门套件板用户指南》
5. UG111：《嵌入式系统工具指南 》
6. 《开发系统参考指南》
7. DS099：《Spartan–3 FPGA 系列完整数据手册》
8. 《MicroBlaze 处理器参考指南》
9. DS123：Platform Flash 在系统可编程配置 PROM

修订历史

下表说明此技术文档的修订历史。

日期	版本	修订
2004 年 8 月 19 日	1.0	Xilinx 最初版本。
2005 年 6 月 27 日	2.0	参考设计中新增引导初始化文件和 MCS 填充实用工具。