

Running Code Out of the PPC405 Caches

The PowerPC 405 Core, included in Virtex-II Pro, contains 16KB Instruction and 16KB Data Cache. A common usage of these caches is pre-loading them with the software application, which runs completely in the cache and has no external memory requirements. This document explains the process of creating an ELF file and downloading that ELF file to the cache.

This document contains the following sections:

- Creating the Software Application
- Downloading to the PowerPC Caches
- XMD.INI File Example
- Creation of bootloop.elf

Creating the Software Application

Creating a software application to run out of the PPC405 caches requires the use of a linker script. A linker script must be created to build a segmented ELF file. A segmented ELF consists of three sections, a boot, data, and an instruction section. The following steps outline the creation of a linker script and the segmented ELF file.

- 1) Copy the linker script, found below, to the project directory.

```
/* XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
// SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR
// XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION
// AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION
// OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS
// IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
// AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
// FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY
// WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
// IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
// REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
// INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
// FOR A PARTICULAR PURPOSE.
//
// (c) Copyright 2003 Xilinx, Inc.
// All rights reserved.
//
//-----*/

_STACK_SIZE = DEFINED(_STACK_SIZE) ? _STACK_SIZE : 4k;
_HEAP_SIZE = DEFINED(_HEAP_SIZE) ? _HEAP_SIZE : 4k;

MEMORY
{
    icache : ORIGIN = 0x70000000, LENGTH = 16k - 4
    boot   : ORIGIN = 0x70003ffc, LENGTH = 4
    dcache : ORIGIN = 0x78000000, LENGTH = 16k
}

ENTRY(_boot)
STARTUP(boot.o)
GROUP(libxil.a libc.a)
```

```

SECTIONS
{
    .vectors :
    {
        . = ALIGN(64k);
        *(.vectors)
    } > icache

    .text :
    {
        *(.boot0)
        *(.text)
    } > icache

    .data :
    {
        *(.data)
        *(.got2)
        *(.rodata)
        *(.fixup)
    } > dcache

    /* small data area (read/write): keep together! */
    .sdata :
    {
        *(.sdata)
    } > dcache

    .sbss :
    {
        . = ALIGN(4);
        *(.sbss)
        . = ALIGN(4);
    } > dcache
    __sbss_start = ADDR(.sbss);
    __sbss_end   = ADDR(.sbss) + SIZEOF(.sbss);

    /* small data area 2 (read only) */
    .sdata2 :
    {
        *(.sdata2)
    } > dcache

    .bss      :
    {
        . = ALIGN(4);
        *(.bss)
        *(COMMON)
        . = ALIGN(4);
        __bss_end = .;

        /* add stack and align to 16 byte boundary */
        . = . + _STACK_SIZE;
        . = ALIGN(16);
        __stack = .;
    }
}

```

```

    /* add heap and align to 16 byte boundary */
    __heap_start = .;
    . = . + _HEAP_SIZE;
    . = ALIGN(16);
    __heap_end = .;
} > dcache
__bss_start = ADDR(.bss);

/* .boot must be at the end and it must be mapped to the last
   address in the instruction cache. Writing to the last address
   triggers the tools (GDB, svf2elf) to commit the previous write
   accesses.
*/

.boot    :
{
    *(.boot)
} > boot
}

```

- 2) There are several things to note about this linker script:
- a) The addresses defined in the MEMORY section do NOT apply to any memory in the processor system. These addresses will be used when the application is downloaded to specify cache memory.

```

MEMORY
{
    icache : ORIGIN = 0x70000000, LENGTH = 16k - 4
    boot   : ORIGIN = 0x70003ffc, LENGTH = 4
    dcache : ORIGIN = 0x78000000, LENGTH = 16k
}

```

- b) The .boot section must be placed at the end of icache memory space. Writing to the last address causes GDB, XMD, and SVF2ELF to reset the PC to this address, which is the location of the boot code.
- 3) In XPS, select the processor to which the linker script will be applied.
- 4) Select **Options -> Compiler Options**
- 5) In the Compiler Options dialog shown in figure 1, select the linker script just created.

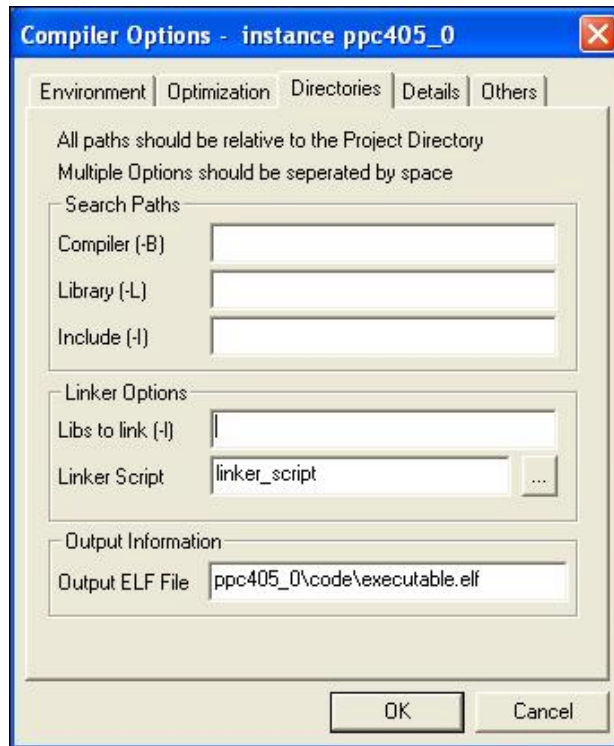


Figure 1. Compiler Options for the PPC405 Instance

- 6) Click **Ok**.
- 7) Select **Tools -> Compile Program Sources**. This will generate the segmented ELF file.

Downloading to the PowerPC Caches

The default XMD options do NOT include the use of cache memory. Applications using cache memory require changes to the XMD options. The following steps outline the commands used to download to the cache memory. These commands can also be placed in a xmd.ini file. XMD will read any xmd.ini file located in the directory where XMD is started.

- 1) Program the FPGA using iMPACT.
- 2) In XPS, select **Tools -> XMD**.
- 3) Enter the following command:

```
ppcconnect -cable type xilinx_parallel4 port lpt1 \  
-configdevice devicenr 1 partname System_ACE irlength 8 idcode 0x0a001093 \  
-configdevice devicenr 2 partname XC2VP7 irlength 10 idcode 0x0124a093 \  
-debugdevice devicenr 2 icachestartadr 0x70000000 dcachestartadr \  
0x78000000
```

- 4) There are several things to note about this command line:
 - a) The JTAG chain must be modified to match the JTAG chain on the board being used.
 - b) The icachestartadr and dcachestartadr must match the addresses specified in the linker_script. These options tell XMD to place the instructions and data in the PPC405 caches.
- 5) The following is an explanation of these switches and their associated options:

-configdevice : used to define a specific device on the JTAG chain. A separate “-configdevice” switch is required for each device in the chain. The “-configdevice” switch supports the following options:

- **partname** <devicename>
 - Name of the device. (Optional)
- **devicenr** <device position>
 - Position of the device in the JTAG chain. (Required)
- **irlength** <length of the JTAG Instruction Register>
 - Length of the IR register of the device. This information can be found in the device BSDL file. (Required for non-Xilinx devices)
- **idcode** <device idcode>
 - JTAG Idcode of the device. This information can be found in the device BSDL file. (Required for non-Xilinx devices)

-debugdevice : used for configuration and debug specifications of a specific CPU. Only one “-debugdevice” switch is allowed. It defines the device containing the CPU as well as the CPU number being debugged. The “-debugdevice” switch supports the following options:

- **devicenr** <PowerPC device position>
 - Position of the VirtexIIPro device containing the PowerPC, in the JTAG chain
- **cpunr** <CPU Number>
 - ID of the specific PowerPC to be debugged in a Virtex-II Pro device containing multiple processors

PowerPC processors

The following options allow users to map special PowerPC features like ISOCM, Caches, TLB, DCR registers, etc. to unused memory addresses. The debugger can then access these special functions using these unused memory addresses. This is helpful for reading and writing to these registers/memory from GDB or XMD. These addresses are also used to preload the OCM memory or PPC caches.

Note that, these options **do not** create any real memory mapping in hardware.

- **icachestartadr** <I-Cache start address>
 - Start address for reading or writing the instruction cache contents
- **dcachestartadr** <D-Cache start address>
 - Start address for reading or writing the data cache contents
- **itagstartadr** <I-Cache tag start address>
 - Start address for reading or writing the instruction cache tags
- **dtagstartadr** <D-Cache tag start address>
 - Start address for reading or writing the data cache tags
- **isocmstartadr** <ISOCM start address>
 - Start address for the ISOCM
- **isocmsize** <ISOCM size>
 - Size of the ISBRAM memory connected to the ISOCM interface
- **isocmdcrstartadr** <ISOCM DCR address>
 - DCR address corresponding to the ISOCM interface specified using the TIEISOCMDCRADDR signals on PowerPC
- **tlbstartadr** <TLB start address>
 - Start address for reading and writing the Translation Look-aside Buffer
- **dcrstartadr** <DCR start address>
 - Start address for reading and writing the Device Control Registers. Using this option, the entire DCR address space (2^{10} addresses, one word per DCR address) can be mapped to addresses starting from <dcrstartadr> for debugging purposes from XMD and GDB

Table 2, contains the irLength for each Virtex-II Pro device used in defining the JTAG chain. For additional devices included in the JTAG chain, the irLength can be found in the BSDL file for the associated device.

Device	# PPC Cores	PPC405 IR Length
XC2VP2	0	6
XC2VP4	1	10
XC2VP7	1	10
XC2VP20	2	14
XC2VP30	2	14
XC2VP40	2	14
XC2VP50	2	14
XC2VP70	2	14
XC2VP100	2	14
XC2VP125	4	22

Table 2. irLength for PPC405 Cores

For additional information regarding each of these options, refer to the Embedded Systems Tool Guide chapter 13 included in the EDK Installation.

- 6) Once a connection to the processor has been made, the ELF file can be downloaded using the following command:

```
dow ppc405_i/code/executable.elf
```

- 7) Upon completion of the download, begin execution of the program by entering the following command:

```
run
```

XMD.INI File Example

```
puts " "  
puts "**** Download an ELF File to the Cache ****"  
puts " "  
  
ppcconnect -cable type xilinx_parallel4 port lpt1 \  
-configdevice devicenr 1 partname System_ACE irlength 8 idcode 0x0a001093 \  
-configdevice devicenr 2 partname XC2VP7 irlength 10 idcode 0x0124a093 \  
-debugdevice devicenr 2 icachestartadr 0x70000000 dcachestartadr 0x78000000  
  
puts " "  
puts " *** Reset the PPC ***"  
puts " "  
rst  
  
puts " "  
puts " *** Download the ELF file ***"  
puts " "  
dow ppc405_i/code/executable.elf  
  
puts " "  
puts " *** Run the code *** "  
puts " "  
run
```

