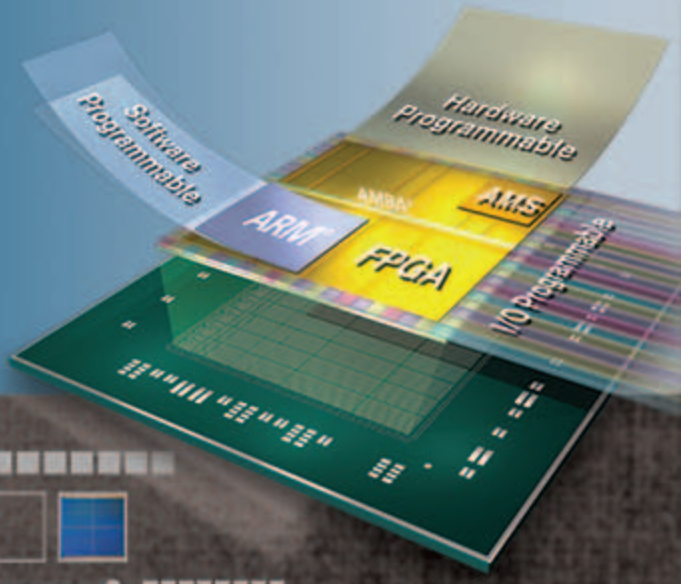
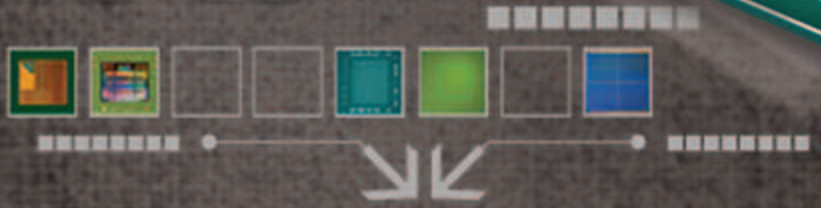
XILINX
ALL PROGRAMMABLE™

XILINX专家大讲堂

—Zynq SoC篇

ZYNQ





赛灵思市场部制作

2013年秋季

Zynq All Programmable SoC

适用于市场中的所有应用，是针对各类系统设计问题的最智能解决方案

Xilinx Zynq®-7000 AI SoC 通过硬件、软件和 I/O 可编程性实现了扩展式系统级差异、集成和灵活性。通过 Zynq-7000 平台，您可以设计更智能的系统，控制和分析部分利用灵活的软件、紧密配合擅长实时处理的硬件，辅之以优化的系统接口 - 这样，BOM 成本可大幅削减、NRE 成本更低、设计风险减少、加快上市时间。。

主要文档

- Zynq-7000 A Generation Ahead 背景资料
- Zynq-7000 All Programmable SoC 简介
- Zynq-7000 技术文档
- 查看所有关于 Zynq™-7000 技术文档

快速链接

- Zynq-7000 Soc 开发板与套件
- Zynq-7000 SoC 产品选型表
- Zynq-7000 SoC 解决方案中心
- Zynq-7000 Linux 方案
- 嵌入式培训

目 录

XILINX ALL PROGRAMMABLE™

在 Zynq 上用 MIG 扩展内存(1)-XPS 篇

作者：Haoliang Qin，Xilinx 处理器专家

硬件平台：ZC706 开发板

软件工具：XPS & SDK 14.4

labfiles.zip 41 KB

MIG(Memory Interface Generator)的基本配置：

AXI 接口: 200MHz, 32bit

Memory 接口: 800MHz, 64bit

Step 1: 创建工程

启动 XPS 14.4。用器件 XC7Z045(FFG900, -2)创建一个新的工程。创建工程时不要选择 ‘AXI Reset Module’。

Step 2: 配置 Zynq

按照 labfiles 里面的 Zynq-PS-DDR-Configuration.png 配置 PS DDR3 的参数。

将 CPU 的频率设置为 733MHz

取消 ‘Enable Programmable Clock and reset to PL’

取消 ‘Enable PL Interrupts to PS and vice versa’

取消所有外设，仅仅保留 UART。UART1 使用 MIO 48..49

Step 3: 配置 Clock Generator

CLKIN：Frequency=200000000

CLKOUT0：Frequency=800000000, Phase=337.5, Group=PLLE0, Buffered=FALSE

CLKOUT1：Frequency=800000000, Phase=0, Group=PLLE0, Buffered=FALSE

CLKOUT2：Frequency=50000000, Phase=10, Group=PLLE0, Buffered=FALSE

CLKOUT3：Frequency=200000000, Phase=0, Group=PLLE0, Buffered=TRUE

CLKOUT4：Frequency=200000000, Phase=0, Group=PLLE0, Buffered=TRUE

注意: CLKOUT2 是为 axi_7series_ddrx_0::sync_pulse 提供时钟的，必须是 CLKOUT0 (axi_7series_ddrx_0::freq_refclk)的 1/16。

Step 4: 配置 MIG

从 IP Catalog 里面添加 ‘AXI 7 Series Memory Controller(DDR2/DDr3)’ 到当前设计

配置 PHY to Controller Clock Ratio 为 4:1

配置 Memory Type=SODIMMS; Memory Part=MT8JTF12864HZ-1G6

更改 AR/AW/B/R/W 寄存器的状态为 ‘AUTOMATIC’

确认 RTT 为 RZQ/4

选中 ‘DCI Cascading’

从 labfiles\zc706_ddr3_sodimm_pinout.ucf 中导入 DDR3 的管脚配置

将 axi_7series_ddrx_0 的内存大小修改为 1GB

在所有 axi_7series_ddrx_0:: (IO_IF)memory_0 端口(除了 parity)上单击右键 ,选择 Make external。

Step 5: 建立 IP 之间的连接

axi_7series_ddrx_0::clk_ref <-> clock_generator_0::CLKOUT3
axi_7series_ddrx_0::mem_refclk <-> clock_generator_0::CLKOUT1
axi_7series_ddrx_0::freq_refclk <-> clock_generator_0::CLKOUT0
axi_7series_ddrx_0:: pll_lock <-> clock_generator_0::LOCKED
axi_7series_ddrx_0::sync_pulse <-> clock_generator_0::CLKOUT2
axi_7series_ddrx_0:: S_AXI::clk <-> clock_generator_0::CLKOUT4

processing_system7_0::M_AXI_GP0::M_AXI_GP0_ACLK <-> clock_generator_0::CLKOUT4

axi_interconnect_1::INTERCONNECT_ACLK <-> clock_generator_0::CLKOUT4
axi_interconnect_1::INTERCONNECT_aresetn <-> clock_generator_0::LOCKED (Done in column Net)

在 ‘clock_generator_0::RST ‘上单击右键，选择 Make external。将 External Port 下面的 ‘clock_generator_0_RST_pin’ 名字更改为 ‘RESET’ 类匹配相应的 ucf 约束。

Step 6：GUI 之外的更改

关闭当前工程。

用文本编辑器打开 system.mhs，找到 CLKOUT2 并添加 DUTY_CYCLE

PARAMETER C_CLKOUT2_FREQ = 31250000
PARAMETER C_CLKOUT2_PHASE = 10
PARAMETER C_CLKOUT2_DUTY_CYCLE = 0.0625
PARAMETER C_CLKOUT2_GROUP = PLLE0
PARAMETER C_CLKOUT2_BUF = FALSE

用 labfiles\ system.ucf 替换 ‘data’ 目录下的同名文件

Step 7：生成 BitStream

重新打开工程，电机 Generate BitStream 生成.bit 文件，然后 Export Design to SDK。

在 SDK 里面，可以用模板 “Memory Tests” 创建一个工程，测试确认 MIG 工作正常。

Zynq PL 侧的 DDR PHY 的最高速率为 1866Mbps。如果配置 MIG 的’ PHY to Controller Clock Ratio’ 为 4:1 ,MIG 的 AXI 端口的最高工作频率只能到 233.33MHz。如果 PL 里面的 IP 对 MIG 的访问数据量比较大，这种配置有优势。如果 CPU 通过 MIG 访问扩展内存比较频繁，就需要提高 MIG 的 AXI 端口的工作频率。

以下面的 MIG 配置为例:

AXI 接口: 250MHz, 32bit

Memory 接口: 500MHz, 64bit

在上面的基础上，要做以下修改：

Step 3: 配置 Clock Generator:

CLKIN：Frequency=200000000
CLKOUT0：Frequency=500000000, Phase=337.5, Group=PLLE0, Buffered=FALSE
CLKOUT1：Frequency=500000000, Phase=0, Group=PLLE0, Buffered=FALSE
CLKOUT2：Frequency=31250000, Phase=10, Group=PLLE0, Buffered=FALSE
CLKOUT3：Frequency=200000000, Phase=0, Group=PLLE0, Buffered=TRUE
CLKOUT4：Frequency=250000000, Phase=0, Group=PLLE0, Buffered=TRUE

Step 4: 配置 MIG

配置 PHY to Controller Clock Ratio 为 2:1

在 Ports Tab 页面单击右键，使能 Net 列的显示。将 axi_7series_ddrx_0:: (IO_IF)memory_0 下所有的 net 的名字删除掉前缀 ‘axi_7series_ddrx_0_’ ，然后将 External Ports 下 MIG 对应的信号的名字也删除前缀。这可以帮助工具完成时序收敛。

在新的配置下，CPU 通过 MIG 访问扩展 DDR3 内存的吞吐量会得到一定的提升。通过分析 Timing Analyzer 发现，MIG 的工作频率在 250MHz 的基础上还有小幅的提升空间。

在 Zynq 上用 MIG 扩展内存(2)-Vivado 篇

作者：Haoliang Qin，Xilinx 处理器专家

硬件平台：ZC706 开发板

软件工具：Vivado 2013.2

labfiles.zip 41 KB

Step 1: 创建工程

启动 Vivado 2013.2,创建一个新的工程 zc706_mig。选中 Create project subdirectory。

选择 RTL Project

一路 Next，在 Default Part 页面选择 ZC706 开发板。

Step 2: 配置 Zynq

在左面的 Flow Navigator 窗口，单击 Create Block Design，Design Name 填写 zynq。

在 Diagram Tab 页里面添加 IP ‘ZYNQ7 Processing System’。

双击 processing_system7_1，打开配置界面。取消所有外设，仅仅保留 UART。UART1 使用 MIO 48..49。关闭 FCLK_CLK0 的输出。

Step 3: 配置 MIG

在 Diagram Tab 页里面添加 IP ‘MIG 7 Series’。双击 mig_7series_1，打开配置界面。

在 Memory Selection 页，选择 DDR3

在 Controller Options 也 配置 Clock Period 为 1250ps 配置 Memory Type=SODIMMS; Memory Part=MT8JTF12864HZ-1G6。

在 AXI Parameter 页，配置 Data Width=32， ID width=12。

在 Memory Options 页，配置 Input Clock Period=5000ps(200MHz).

在 FPGA Options 页，配置 System Clock=Differential，Reference Clock=Use System Clock，System Reset Polarity=ACTIVE HIGH。

在 Extended FPGA Options 页，选中 DCI Cascade。

在 IO Planning Options 页，选择 Fixed Pin Out，然后导入 labfiles\zc706_mig_pinout.ucf，单击 Validate，忽略 warnings，单击 Next。

在 System Signals Selection 页，单击 Next。

在 Summary 页，单击 Next。

在 Simulation Options 页，选择 Accept，单击 Next。

在 PCB Information 页，单击 Next。

在 Design Notes 页，单击 Generate。

Step 4: 建立 IP 之间的连接

在 Diagram Tab 页，单击窗口上部的 Run Connection Automation，选择/mig_7series_1/S_AXI，系统将自动添加 IP 并建立部分连接。

单击窗口上部的 Run Connection Automation，选择/mig_7series_1/sys_rst。

连接/mig_7series_1/ui_clk 到 processing_system7_1/M_AXI_GP0_ACLK

连接/mig_7series_1/aresetn 到/proc_sys_rest/peripheral_aresetn[0:0]

单击窗口上部的 Run Block Automation，选择 processing_system7_1

单击/mig_7series_1/SYS_CLK 前面的加号，展开这个接口。

选中/mig_7series_1/sys_clk_p ,单击右键 ,选择 Create Port。在弹出的窗口里面更改 Type 为 Clock，填写 Frequency(MHz)为 200。

对/mig_7series_1/sys_clk_n 执行同样的操作。

选中/mig_7series_1/DDR3 接口，单击右键，选择 Make External

单击 Diagram Tab 页左边最下面的 Regenerate Layout 按钮，由工具自动重新排布。生成的结果如下。是不是感觉非常漂亮？

system diagram

Step 5: 创建约束

命名为 system，将以下内容拷贝到约束文件中：

```
set_property LOC G9 [ get_ports sys_clk_n]
set_property IOSTANDARD DIFF_SSTL15 [ get_ports sys_clk_n]
set_property LOC H9 [ get_ports sys_clk_p]
set_property IOSTANDARD DIFF_SSTL15 [ get_ports sys_clk_p]
set_property LOC A8 [ get_ports reset]
set_property IOSTANDARD LVCMOS15 [ get_ports reset]
# additional constraints
#
create_clock -name sys_clk_pin -period "5.0" [get_ports "sys_clk_p"]
```

Step 6：设计验证

在 Block Design 窗口里面，在 zynq.bd 上面单击右键，选择 Generate Output Products，然后再单击右键，选择 Create HDL Wrapper。

在左面的 Flow Navigator 窗口，单击 Generate Bitstream。

在我的计算机上，大约 20 分钟后，bit 文件生成。

然后在 Vivado 中，单击 File->Export->Export Hardware for SDK，选中 Launch SDK。

在 SDK 里面，可以用模板“Memory Tests”创建一个工程，测试确认 MIG 工作正常。

以此为基础，开发者可以灵活的调整 MIG 的 AXI 端口和 Memory 端口的工作频率，完成贴合自己应用的嵌入式设计。

嵌入式系统的性能测试(1) – lmbench 篇

作者：Haoliang Qin，Xilinx 处理器专家

Zynq All Programmable SoC

适用于市场中的所有应用，是针对各类系统设计问题的最智能解决方案

Xilinx Zynq®-7000 AI SoC 通过硬件、软件和 I/O 可编程性实现了扩展式系统级差异、集成和灵活性。通过 Zynq-7000 平台，您可以设计更智能的系统，控制和分析部分利用灵活的软件、紧密配合擅长实时处理的硬件，辅之以优化的系统接口 - 这样，BOM 成本可大幅削减、NRE 成本更低、设计风险减少、加快上市时间。。

主要文档

- Zynq-7000 A Generation Ahead 背景资料
- Zynq-7000 All Programmable SoC 简介
- Zynq-7000 技术文档
- 查看所有关于 Zynq™-7000 技术文档

快速链接

- Zynq-7000 Soc 开发板与套件
- Zynq-7000 SoC 产品选型表
- Zynq-7000 SoC 解决方案中心
- Zynq-7000 Linux 方案
- 嵌入式培训

要评价一个系统的性能，通常有不同的指标，相应的会有不同的测试方法和测试工具。既有比较成熟的商业测试软件，也有许多优秀的开源工具来完成这个任务。本文简要介绍如何使用 lmbench 来完成系统综合性能测试。

Lmbench 用 C 语言编写的，是一套具有较好可移植性的，简易的，符合 ANSI/C 标准为 UNIX/POSIX 而制定的微型测评工具。一般来说，它衡量两个关键特征：反应时间和带宽。Lmbench 旨在使系统开发者深入了解系统关键操作的基础成本。

Lmbench 是个多平台软件，因此能够对同级别的系统进行比较测试，反映不同系统的优劣，通过选择不同的库函数我们就能够比较库函数的性能；更为重要的是，作为一个开源软件，lmbench 提供一个测试框架，假如测试者对测试项目有更高的测试需要，能够通过少量的修改源代码达到目的（比如现在只能评测进程创建、终止的性能和进程转换的开销，通过修改部分代码即可实现线程级别的性能测试）。

lmbench 是一个用于评价系统综合性能的软件，主要测试内容包括：

- 存储器延迟计算结果
  - 存储器延迟测试展示了所有系统（数据）的缓存延迟，包括 L1/L2 cache 以及主内存
- 带宽测评工具
  - 读取缓存文件
  - 拷贝内存
  - 读内存
  - 写内存
  - 管道
  - TCP
- 反应时间测评工具
  - 上下文切换
  - 网络： 连接的建立，管道，TCP，UDP 和 RPC hot potato
  - 文件系统的建立和删除

- 进程创建
- 信号处理
- 上层的系统调用
- 内存读入反应时间

- 其他
  - 处理器时钟比率计算

Lmbench 的测试内容有很多，实在是 benchmarking 领域的“瑞士军刀”啊。

Lmbench 的成功应用案例包括：

- Sun 公司和 SGI 公司已经使用这种测评工具以寻找和补救存在于性能上的问题。
- Intel 公司在开发 P6 的过程中，使用了它们。
- Linux 在 Linux 的性能优化中使用了它们。

1. 下载：

Lmbench 当前的最新版本为 3.0。

lmbench 的主站：

http://www.bitmover.com/lmbench/

lmbench 3 版本的下载链接：

http://www.bitmover.com/lmbench/lmbench3.tar.gz

2. 编译：

下载解压后运行 make build 后会出现以下错误：

make[2]: *** No rule to make target `../SCCS/s.ChangeSet', needed by `bk.ver'. Stop.

make[2]: Leaving directory `/home/wave/xilinx/lmbench3/src'

make[1]: *** [lmbench] Error 2

make[1]: Leaving directory `/home/wave/xilinx/lmbench3/src'

make: *** [build] Error 2

问题来源：

SCCS 是 Sun 的版本管理工具。lmbench 这个项目是 Sun 资助的，而且作者 Larry McVoy 也曾经是 Sun 的员工，就用了这玩艺。

- 解决办法 1：修改 src/Makefile，将 231 行的 bk.ver 去掉就可以了。编译完成后在 bin 下有 benchmarking 需要的的 binary。
- 解决办法 2：在 lmbench3/SCCS 目录下创建一个工程要的文件 s.ChangeSet，骗过 make。

交叉编译：

本人在 Xilinx ZC706 开发板上尝试用 PetaLinux 2013.04 附带的交叉编译工具链进行了编译，可以在 lmbench3 目录下使用以下命令完成编译：

```
make OS=armv7l-linux-gnu CC=arm-xilinx-linux-gnueabi-gcc AR=arm-xilinx-linux-gnueabi-ar build
```

如果使用其他 pre-built 的交叉编译工具 ,需要修改 CC 和 AR 的内容。编译完成后 ,会在 lmbench3\bin 目录下生成 OS 指定的目录，可执行文件放在这个目录下。这个 OS 名称适用于 ZC706 开发板，在其他嵌入式开发板上可能会不同。一个简单的工程上取得合适名字的办法：如果不是这个名字，在下一步配置完成后会提示无法保存配置文件到某个位置，看系统提示修改就好了。

注意：使用 PetaLinux 2013.04 交叉编译工具链之前，要先’source settings.sh’初始化环境变量。

3. 配置：

把 Host 上编译好的 lmbench 转移到嵌入式 Linux 中有很多种办法，本人采用以下办法，在 Host 上把 lmbench3 目录打成一个 tar ball，然后拷贝到 SD 卡中，嵌入式 Linux 从 SD 卡启动。Linux 启动后，将 SD 卡 mount 到文件系统中：

```
mount /dev/mmcblk0p1 /mnt
```

在嵌入式 Linux 中创建临时文件系统，这里假定开发者是用 root 帐号登录到 Linux 的：

```
mkdir -p /home/root/ramfs
mount -t ramfs none /home/root/ramfs -o maxsize=32768
```

注意：缺省情况下，Ramfs 被限制最多可使用内存大小的一半。可以通过 maxsize(以 kbyte 为单位)选项来改变。

用以下命令启动配置过程：

```
cd /home/root/ramfs
tar xvf /mnt/lmbench3.tar
export PATH=$PATH:/home/root/ramfs/lmbench3/bin/armv7l-linux-gnu
cd lmbench3/scripts
./config-run
```

本人使用的配置为：

```
MULTIPLE COPIES [default 1]: 1
Job placement selection [default 1]: 1 > Allow scheduler to place jobs
MB [default 84]: 8
SUBSET (ALL|HARWARE|OS|DEVELOPMENT) [default all]: all
FASTMEM [default no]: yes
SLOWFS [default no]:
DISKS [default none]
REMOTE [default none]
Processor mhz [default 498 MHz, 2.0080 nanosec clock] 733
FSDIR [default /tmp]
Status output file [default /dev/tty]
Mail results [default yes] no
```

关于测试项的解释：

1) 在多核 CPU 上并行多少个 lmbench。作者说这个目前还是实验性质的，结果可能不正确并有随机性。更重要的是，整个 benchmark 的速度会变得非常慢（100 倍）。既然这样，当然用缺省的数值 1 了。
2) 测试任务的分配方式。按回车选择缺省的方案 1。
3) 有些测试需要一段内存。内存至少是 cache size 的 4 倍，最大不超过物理内存的 80%。内存越大，测试的结果越准确，不过需要的时间也越长。
4) 选择测试项目。分别是全部测试、硬件测试、OS 测试和操作系统开发阶段的测试。缺省选择为全部测试。
5) 内存延迟测试。测试需要的时间可能比较长。在 cache line size 超过 128 字节并希望确定 cache line size 的时候比较有用。
6) 测试文件系统的延迟。在一些老的文件系统(UFS, FFS 等)上会比较慢 ,在 Linux ext2fs 和 Sun tmpfs 上会比较快。可以用来测试 ramdisk 的性能，间接测试 RAM 的性能。
7) 测试磁盘的性能。需要提供磁盘的位置，例如：/dev/sda。同时需要提供一行磁盘的文字描述。
8) 网络测试。需要有另外一台中间不经过网关的机器，需要 rsh 访问权限。
9) 软件会自动分析 CPU 的工作频率。如果软件无法检测，需要提供 CPU 的工作频率。
10) 需要一个位置保存临时文件。要求不能是内存驻留文件系统。
11) lmbench 的输出设备。缺省为/dev/tty，在大多数场合都适用。
12) 是否将结果上传给作者。

4. 运行

首先要确认 Embedded Linux 已经配置了合适的 hostname ,这个 hostname 将作为测试结果 raw 文件的文件名。

用以下命令开始性能测试：./results

在本人的计算机上，大约运行了 9 分钟后结束。生成的结果在 lmbench3/results/$OS 目录里面。

如果 Embedded Linux root fs 里面有 perl，可以直接在开发板上看测试结果的 summary。如果没有也不打紧，可以把生成的测试结果 raw 文件拷贝到 host 上，在 host 上生成 summary。

./getsummary ./zynq.0

Summary 是对测试结果的高度精简。如果感兴趣，开发者可以用 Text Editor 打开测试结果 raw 文件，找到更多更细致的有用信息。

嵌入式系统的性能测试(2) – iozone 篇

作者：Haoliang Qin，Xilinx 处理器专家

Zynq All Programmable SoC

适用于市场中的所有应用，是针对各类系统设计问题的最智能解决方案

Xilinx Zynq®-7000 AI SoC 通过硬件、软件和 I/O 可编程性实现了扩展式系统级差异、集成和灵活性。通过 Zynq-7000 平台，您可以设计更智能的系统，控制和分析部分利用灵活的软件、紧密配合擅长实时处理的硬件，辅之以优化的系统接口 - 这样，BOM 成本可大幅削减、NRE 成本更低、设计风险减少、加快上市时间。。

主要文档

- Zynq-7000 A Generation Ahead 背景资料
- Zynq-7000 All Programmable SoC 简介
- Zynq-7000 技术文档
- 查看所有关于 Zynq™-7000 技术文档

快速链接

- Zynq-7000 Soc 开发板与套件
- Zynq-7000 SoC 产品选型表
- Zynq-7000 SoC 解决方案中心
- Zynq-7000 Linux 方案
- 嵌入式培训

iozone 是一个文件系统性能评测工具，可以测试 Read, write, re-read,re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write 等不同模式下不同文件系统的读写性能。本文介绍了它的各种功能，如何针对 ARM 进行交叉编译，以及如何配置合理参数进行评测。通过在 RAMFS 上运行 iozone，亦可测试内存子系统的性能。

1. 介绍

iozone 的 web site 位于：http://www.iozone.org/

本文中使用的软件为：
http://www.iozone.org/src/current/iozone3_414.tar

iozone 的主要测试内容：

Write: 测试向一个新文件写入的性能。当新文件被写入时，除了文件中的数据，还有被称作“元数据”的额外信息也需要被存储。这些额外信息包括目录信息，所分配的空间和一些与该文件有关但又并非该文件所含数据的其他数据。因为这些额外信息，Write 的性能通常会比 Re-write 的性能低。

Re-write: 测试向一个已存在的文件写入的性能。因为此时元数据已经存在。Re-write 的性能通常比 Write 的性能高。

Read: 测试读一个已存在的文件的性能。

Re-Read: 测试读一个最近读过的文件的性能。因为操作系统通常会缓存最近读过的文件数据，Re-Read 性能会高些。

Random Read: 测试读一个文件中的随机偏移量的性能。

Random Write: 测试写一个文件中的随机偏移量的性能。

Random Mix: 测试读写一个文件中的随机偏移量的性能。在随机访问的时候，许多其他因素可能影响测试结果，例如：操作系统缓存的大小，磁盘数量，寻道延迟和其他。

Backwards Read: 测试使用倒序读一个文件的性能。尽管不常见，但事实上确实有些应用这么干，例如 MSC Nastran。

Record Rewrite: 测试写与覆盖写一个文件中的特定块的性能。在跨越 L1 cache、L2 cache 和操作系统缓存边界时，测试结果会发生突然变化。

Strided Read: 测试跳跃读一个文件的性能。例如：每间隔 200Kbytes,读 4Kbytes 并重复这个模式。文件中使用了数据结构并且访问这个数据结构的特定区域的应用程序常常这样做。

Fwrite: 测试调用库函数 fwrite()来写文件的性能。这个测试是针对新文件，所以包括元数据的写入。

Frewrite:测试调用库函数 fwrite()来写文件的性能。类似 Re-write 操作，因为是针对已存在的文件，无元数据操作，测试的性能会高些。

Fread:测试调用库函数 fread()来读文件的性能。

Freread: 这个测试与上面的 fread 类似，类似 Re-Read，因为操作系统缓存了文件数据会导致测试结果比较高。

几个特殊测试:

Mmap: 这个测试就是测量使用 mmap()机制完成 I/O 的性能。许多操作系统支持 mmap()的使用来映射一个文件到用户地址空间。映射之后,对内存的读写将同步到文件中去。这对一些希望将文件当作内存块来使用的应用程序来说很方便。一个例子是内存中的一块将同时作为一个文件保存在于文件系统中。mmap 文件的语义和普通文件略有不同。如果发生了对内存的存储，并不是立即发生相应的文件 I/O 操作。使用 MS_SYNC 和 MS_ASYNC 标志位的 msyc()函数调用将控制内存和文件的一致性。调用 msync()时将 MS_SYNC 置位将强制把内存里的内容写到文件中去并等待直到此操作完成才返回。而 MS_ASYNC 置位则告诉操作系统使用异步机制将内存刷新到磁盘，这样应用程序可以直接返回而不用等待此操作的完成。

Async I/O: 这个测试测量 POSIX 异步 I/O 机制的性能。许多操作系统支持的另外一种 I/O 机制是 POSIX 标准的异步 I/O。本程序使用 POSIX 标准异步 I/O 接口来完成此测试功能。例如： aio_write(), aio_read(), aio_error()。

2. 针对 ARM 交叉编译：

iozone 对交叉编译的支持算是比较好的。打开 iozone3_414/src/current/Makefile 找到 CC 和 GCC 的定义并修改成如下内容。

CC = arm-xilinx-linux-gnueabi-gcc

GCC = arm-xilinx-linux-gnueabi-gcc

注意：这里使用的是 PetaLinux 2013.04 的 tool chain，使用前要先到 PetaLinux 目录下'source settings.sh'

然后用命令'make linux-arm' 即可完成编译。编译成功后 生成可执行文件 iozone。

3. 在 zc706 上运行 iozone

可以用以下命令看 iozone 的详细参数列表和解释：./iozone –h

iozone 有很多参数。在这里我们关注的是如何用 iozone 通过 ramfs 来测试和比较内存性能 ,在 zc706 上使用的命令如下：

./iozone -Raz -b out.xls -i 0 -i 1 -i 2 -S 512 -g 8M -+r

常用参数说明如下：

-R: 产生 EXCEL 格式的报告

-a: 全自动模式。生成包括所有测试操作的报告，使用的块 大小从 4k 到 16M，文件大小从 64k 到 512M。

-f filename: 用来指定测试时使用的临时文件的文件名。

-z: 和-a 一起，指定测试所有可能的 record size

-S: 指定 process cache 的大小，单位是 Kbytes。

-g -n: 指定 file size 的最大值和最小值。文件越大测试时间越长。测试文件的大小一定要大过 cache，否则会使数值非常不真实。

-+r:在打开文件时的 flag 中包含 O_RSYNC 和 O_SYNC，即同步读和同步写，保证数据真正写到硬件上和真正从硬件上读数据。

可以用以下命令创建 ram file system 以供测试

```
mkdir -p /home/root/tmpfs
mount tmpfs /home/root/tmpfs -t tmpfs -O size=32M
mkdir -p /home/root/ramfs
mount -t ramfs none /home/root/ramfs -o maxsize=32768
```

注意:缺省情况下，Ramfs 被限制最多可使用内存大小的一半。可以通过 maxsize(以 kbyte 为单位)选项来改变。

```
mkdir -p /home/root/ramdisk
mke2fs /dev/ram1 -L "ramdisk" -b 1024 -m 0
mount -t ext2 /dev/ram1 /home/root/ramdisk
```

注意: ramdisk 的大小在配置 Linux kernel 的时候被指定。

进入到相应的目录，执行以下命令完成测试：

```
tar xvf /mnt/iozone3_414.tar
cd iozone3_414/src/current/
./iozone -Raz -b out.xls -i 0 -i 1 -i 2 -S 512 -g 8M -+r
```

生成的 xls report 里面，行是记录大小，列是测试文件大小，单位为 Kbytes。表格中的数据是传输速度，单位为 Kbytes/s。

从测试结果来看，有如下结论：

- 因为是对 ram file system 测试，是否打开数据同步对性能影响不大。
- 对 Write 操作，性能 ramfs>tmpfs>>ramdisk; 对 Read 操作，性能 ramfs>ramdisk>tmfs。可以认为 ramfs 的开销更小 ,更能反映 main memory 的性能。值得注意的是 ramdisk 的写性能很差，只有 ramfs 和 tmpfs 的 1/4 左右。

4. 测试 PL MIG 的性能：

Linux 的 memory pool 缺省是从高地址开始分配的。如果要测试 PL MIG 的性能，只需要简单的修改 Linux 的 kernel command line 即可。在 U-BOOT 里面运行命令：

setenv bootargs console=ttyPS0,115200

root=/dev/ram rw ip=192.168.1.10

earlyprintk mem=2048M

run sdboot

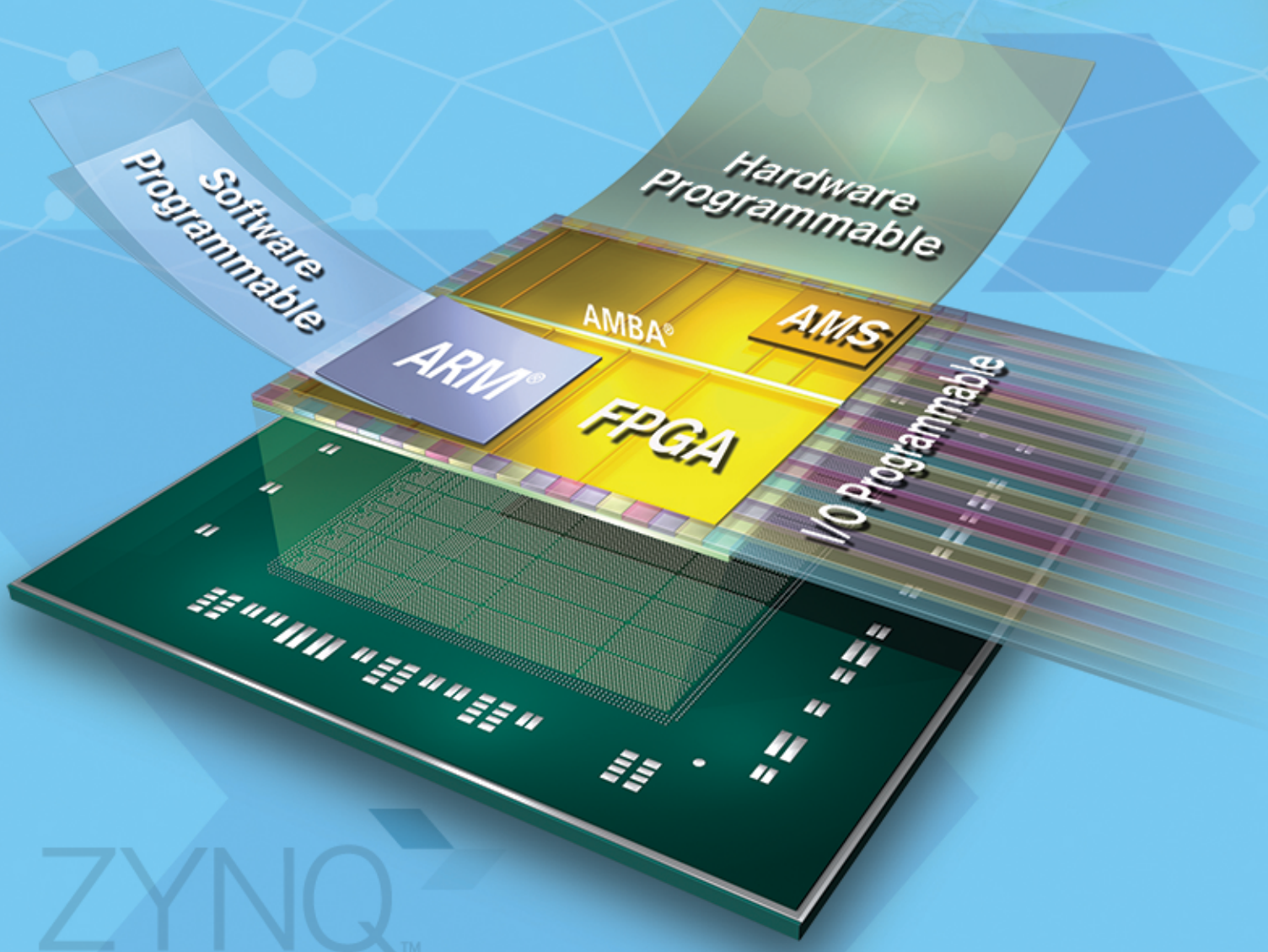
启动后 cat /proc/meminfo 可以看到 Linux 确实使用了 2GB 的内存。

重新运行 benchmarker，即可得到 PL DDR 的性能数据。

注意：文件大小要大于 L2 cache size(512KB)才有意义，否则数据都是在 L1/L2 cache 里面转来转去。

A Generation Ahead

The First
All Programmable SoC



ZYNQ™

XILINX
ALL PROGRAMMABLE™