2016年夏计算机系统综合实验
基于MIPS32和Linux的计算机系统设计

作者信息

姓名	学号	电子邮箱	Git 账户
张宇翔	2013011352	zz593141477@gmail.com	https://git.net9.org/zhangyx13
王邈	2013011329	shankerwangmiao@gmail.com	https://git.net9.org/u/shanker

1 项目概述

1.1 项目背景

在秋季学期的计算机组成原理课程上，我们开发了32位的MIPS CPU，代号为NaiveMIPS。其在设计时参考了MIPS32规范，实现的功能为MIPS32 Release1（以下简称MIPS32R1）规范的子集。在春季学期的操作系统课程上，我们把Linux内核成功移植至NaiveMIPS上，使得Linux内核可以在NaiveMIPS上启动，支持串口、GPIO等基本外设，并能够运行用户态程序。作为挑战性课程的一环，在暑期的综合实验课程上，我们希望在前两门课程实现的软硬件基础上，添加硬件外设与驱动程序支持，进一步完善设计，实现一套更加全面的计算机系统。此外，我们还将移植编译原理课程提供的Decaf编译器，使得用Decaf语言编写的应用程序能够交叉编译为MIPS32 Linux下的原生应用程序。

1.2 设计基础

项目已有的软件资源包括32位的NaiveMIPS CPU、Linux内核源码、GCC交叉编译工具链、QSYS IP核、Decaf编译器等。其中NaiveMIPS为我们自己开发的项目，其余为自行配置或修改的开源软件。硬件资源为友晶DE2i-150开发板。

1.3 项目计划

本项目选择的DE2i-150平台为CPU+FPGA架构，我们只使用其中FPGA部分，其技术参数如下：

组件	数量	型号/参数
FPGA	1	Altera CycloneIV EP4CGX150DF31C8
SSRAM	2	两片总共 1M × 32bits (4MB)
SDRAM	2	两片总共 32M × 32bits (128MB)
Flash	1	4M × 16bits (64MB)
串口	1	
以太网	1	10/100/1000 Mbps (Phy Only)
G-Sensor	1	ADXL345 (I^2C接口加速度计)
SD Card	1	存储卡槽
数码管	8	
LED	18	
拨码开关	18	
按钮开关	4	
晶振	1	50MHz

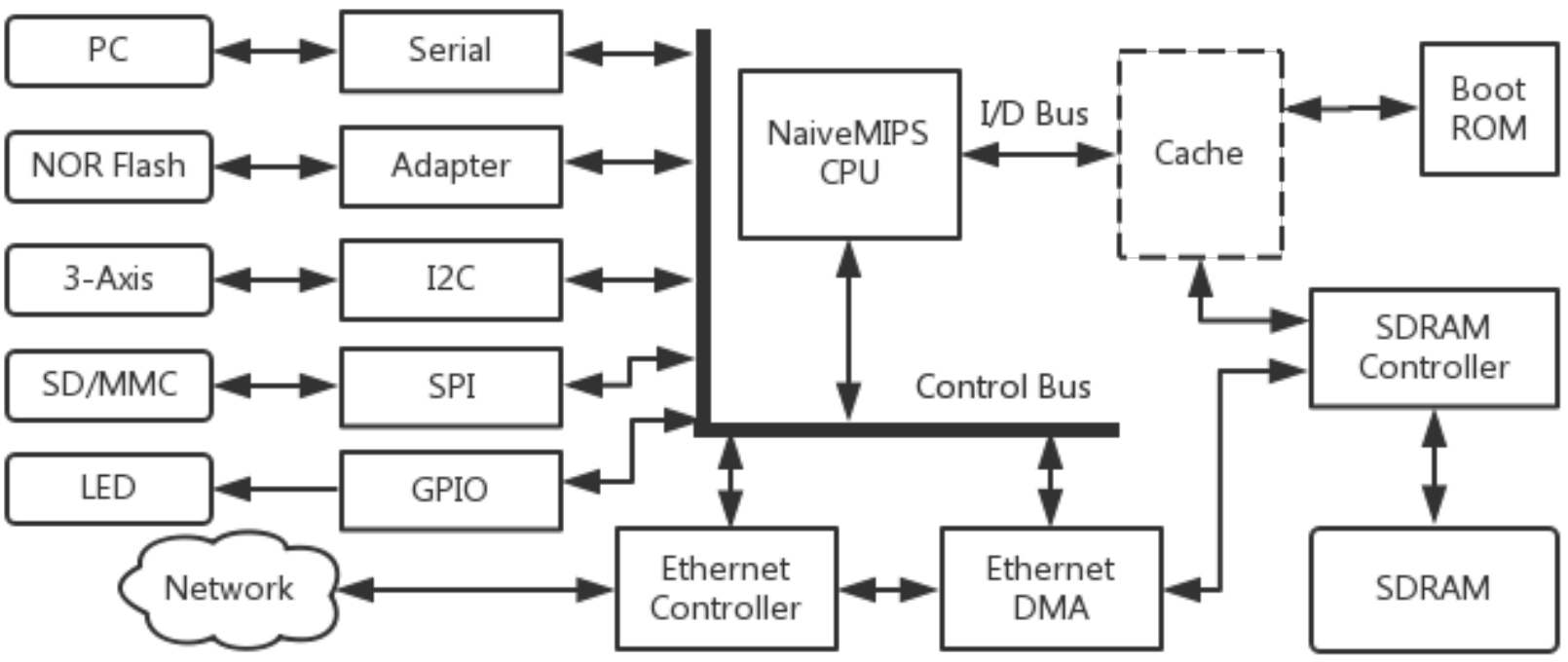
相比于Thinpad，DE2i提供了128MB的DRAM，使得在Linux上运行网络等复杂应用成为可能。在已有Linux核心功能的基础之上，实现更多的硬件支持变得相对容易，因此我们希望尽可能多地将开发板上的资源应用起来，实现功能更加全面的计算机系统设计。

计划使用到的硬件资源包括SDRAM、Flash、串口、以太网、G-Sensor、SD Card和GPIO（LED及开关）。软件层面配置相应的驱动程序，并在Linux中运行一些应用程序用于演示硬件的工作效果。另外，由于使用了DRAM，随机访存的周期较长，如果不使用Cache，系统性能会很差，因而我们将设计一个简单的Cache模块用于访存加速。

2 硬件及驱动程序开发

在DE2i平台上，我们基于Altera提供的QSYS工具搭建了SoC。该工具可以基于配置，自动生成互联总线逻辑，简化开发工作，同时使得各个组件更加模块化、规范化。SoC中的核心为NaiveMIPS CPU，外设为Altera提供的IP核以及其他的开源的IP核。整个QSYS工程文件位于HDL代码库下的`altera/naive_mips_soc.qsys`（不带cache）文件或`altera/naive_mips_soc_cache.qsys`（带cache）文件。

硬件框图如下：



为了使Linux内核知道实际的硬件配置（如总线地址映射）并驱动硬件，需要实现板级设备描述代码。这些代码分3部分：

- 早期初始化代码：位于`arch/mips/thinpad/init.c`，实现板级的早期初始化，同时提供早期调试信息输出函数`prom_putchar`。在串口驱动加载之前，`printk`打印的调试信息最终均通过`prom_putchar`函数输出。
- 设备树描述：位于`arch/mips/boot/dts/thinpad/`目录中的`naivemips_de2i.dts`。设备树（dts）文件用于描述硬件连接关系（时钟、地址、中断号）等信息，各个条目含义参考`Documentation/devicetree/bindings/`目录中对应各个驱动程序的说明。设备树中还提供了缺省的内核启动参数。
- 内核配置：用于在编译时选择需要包含的驱动程序和feature。默认内核配置位于`arch/mips/configs/naivemips_de2i_defconfig`文件，详见“使用说明”一节。

2.1 CPU

在原始设计中，NaiveMIPS CPU有数据和指令两个总线接口，包含地址、数据、读写使能、暂停等信号，其时序与Avalon-MM规范兼容，可以直接封装成为QSYS元件。为了兼容Cache，我们额外增加了控制总线接口，该接口不经过Cache，与各外设的控制寄存器相连。

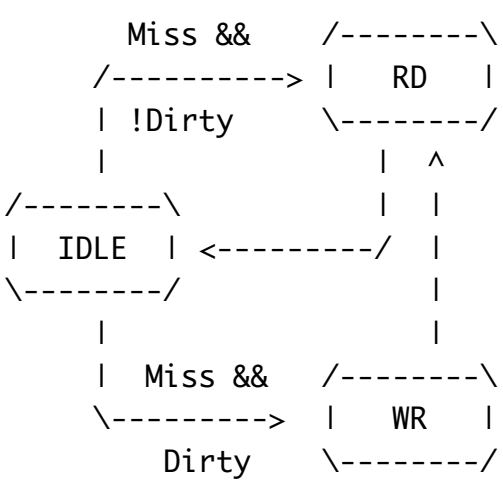
CPU元件用QSYS中的元件创建向导创建，向导中需要填写接口的信号、属性等信息。最终创建的元件wrapper文件位于HDL代码库下的`altera/cpu_qsys_comp.v`，元件描述文件位于`altera/naive_mips_cpu_hw.tcl`。在QSYS中可以直接选择该元件添加至SoC中，无可配置参数。

2.2 Cache

Cache模块同样实现为一个QSYS元件，包含一个只读Slave、一个读写Slave和一个支持突发传输的Master。在QSYS项目中，CPU指令Master与cache的只读Slave相连，CPU数据Master与cache的读写Slave相连，cache的Master与DRAM相连。在cache miss时，cache会通过Master以突发传输读取一个cache行，减少总数据延迟，从而提高了内存访问的效率。由于指令和数据共用一块cache空间，避免了指令、数据一致性的问题。

为了增强灵活性，Cache的QSYS元件有三个可配置参数，分别为地址线宽度 a、Cache Line 宽度 c 、Cache Tag 宽度 t 。这意味着，Cache Line 大小为 2^c 字节，Cache Line 个数为 2^{a-c-t}，Cache 总大小为 2^{a-t} 字节。

该 Cache 分为两部分，一部分由组合逻辑构建，用于从 Cache 中直接读写数据，另一部分主要由时序逻辑构建，用于从支持突发传输的 Master 接口中读写数据。当 Cache 元件接受到一个读／写请求时，会立刻使用组合逻辑电路判断请求的地址是否已经在 Cache 内。若存在，则在同一周期内给出读取结果／写入 Cache。若不在，则触发 `CacheMiss` 信号，启动读写支持突发传输的 Master 的时序逻辑，同时挂起该次读写请求。



时序逻辑部分的状态机有三个状态，分别是空闲、读、写。一般情况下，该状态机处在空闲状态。当注意到发生了 `CacheMiss` 时，若发生 Miss 的 Cache Line 处于 Dirty 状态，则状态机进入写状态，将该 Cache Line 的内容从支持突发传输的 Master 接口写回，写回完毕后，状态机进入读状态，从支持突发传输的 Master 接口取出数据，填实 Cache Line；若发生 Miss 的 Cache Line 并不处在 Dirty 状态，则直接进入读状态。当读出完毕后，状态机恢复空闲状态，同时 `CacheMiss` 信号解除，由组合逻辑电路部分负责继续完成该读写请求。

然而，在实现该 Cache 时，要注意如下的边界情况：

- 当两个 Slave 接口同时发生 Miss 时，应当有相应电路进行优先排序，先后处理两个 Miss。
- 在处理低优先级的 Slave 接口的 Miss 过程中，如果高优先级的 Slave 接口出现 Miss，需要保持当前处理的 Miss 请求不变，屏蔽后到来的 Miss 信号，待处理完前一个 Miss 时再处理下一个。
- 在读 Master 过程中，应当在首个 Word 的读结果到来时，将相应的 CacheLine 置为无效状态，以免在同时发生不期望的读命中。当全部数据都装入完毕后，再置为有效状态。
- 在将读取 Master 的结果写入 CacheLine 时，若同时发生了命中的写入，要注意防止内部总线出现冲突。

2.3 串口控制器

本系统使用了Altera UART IP核作串口控制器。串口控制器配置为基本模式，对外接口只有发送、接收两条信号线，不支持硬件流控。串口控制器有一个Avalon-MM Slave，其挂载在CPU的控制总线上，所有寄存器访问均通过该总线完成。

驱动程序的内核选项为 `CONFIG_SERIAL_ALTERA_UART`，在设备树文件中，增加以下条目描述硬件：

```
    serial0: serial@0xbfd003E0 {
        device_type = "serial";
        compatible = "altr,uart-1.0";
        reg = <0xbfd003E0 0x20>;
        current-speed = <115200>;
        clock-frequency = <30000000>;
        interrupt-parent = <&cpuintc>;
        interrupts = <4>;
    };
```

在操作系统中，串口体现为一个终端 `/dev/ttyAL0`。

2.4 I2C控制器及加速度计

I2C总线用于和DE2i板上的加速度传感器通信。I2C控制器来自OpenCores开源项目（见参考资料），位于HDL代码库

`altera/opencores_i2c/`目录下。在QSYS中添加I2C控制器后，无需配置参数。

I2C控制器驱动的配置条目为`CONFIG_I2C_OCORES`，设备树中添加对应的设备描述：

```
    i2c0: ocores@0xbc010000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "opencores,i2c-ocores";
        reg = <0xbc010000 0x20>;
        interrupt-parent = <&cpuintc>;
        interrupts = <5>;
        clocks = <&ext>;
        clock-frequency = <100000>; /* i2c bus frequency 100 KHz */

        reg-shift = <2>;    /* 32 bit registers */
        reg-io-width = <1>; /* 8 bit read/write */

        adxl345@53 {
            compatible = "adi,adxl345";
            reg = <0x53>;
            interrupt-parent = <&button_pio>;
            interrupts = <31 IRQ_TYPE_LEVEL_HIGH>;
        };
    };
```

与I2C控制器连接加速度传感器同样在设备树中声明，因其挂载在I2C总线下，所以设备树中声明为与I2C控制器的子节点。加速度传感器的驱动选项名为`CONFIG_INPUT_ADXL34X`和`CONFIG_INPUT_ADXL34X_I2C`。

在操作系统中，加速度计通过Input子系统映射到`/dev/input/event0`设备，读取该设备可以收到加速度计输出的数据。

2.5 以太网控制器

DE2i平台上板载了以太网Phy芯片，没有MAC，需要在FPGA中实现MAC。由于以太网协议较负载，因此我们使用了Altera 10/100/1000M Ethernet MAC IP核实现MAC。该IP核还需要两个配套的DMA元件用于接收、发送的数据通路。

MAC元件和两个DMA元件的控制寄存器接口均挂载在控制总线上，由CPU控制。以太网数据通路为MAC的数据接口至DMA，再通过DMA的Master到DRAM。工作时，网络数据可以直接经由DMA与内存交互，不经过CPU，提高运行性能。

MAC元件的对外接口为RGMII和MDIO，其中MDIO为两线总线用于配置Phy芯片，而RGMII为双倍数据率的数据接口，用于传输以太网数据。

在QSYS中添加MAC和DMA元件后，有较多的可配置参数，对此我们参考了Altera NiosII Max10开发板所用的参数。特别需要注意的一点是，Linux中以太网控制器对应驱动程序可能向DMA传递非4字节对齐的内存地址，因此需要在DMA配置参数中启用非对齐访问支持。

Altera的MAC在Linux下的驱动程序选项为`CONFIG_ALTERA_TSE`，在内核配置中打开网卡驱动程序后，还需要启用基本的网络协议栈支持。内核设备树中以太网控制器的设备描述包含了各个元件的地址及基本参数，其内容为：

```
    rgmii_0_eth_tse_0: ethernet@0xbc000000 {
        compatible = "altr,tse-msgdma-1.0";
        reg = <0xbc000000 0x00000400>,
            <0xbc001800 0x00000020>,
            <0xbc001900 0x00000020>,
            <0xbc001a00 0x00000008>,
            <0xbc001b00 0x00000020>,
            <0xbc001c00 0x00000020>;
        reg-names = "control_port", "rx_csr", "rx_desc", "rx_resp", "tx_csr", "tx_desc";
        interrupt-parent = <&cpuintc>;
        interrupts = <2>, <3>;
        interrupt-names = "rx_irq", "tx_irq";
        rx-fifo-depth = <8192>;
        tx-fifo-depth = <8192>;
        address-bits = <48>;
        max-frame-size = <1518>;
        local-mac-address = [00 00 00 00 00 00];
        phy-mode = "rgmii-id";
        phy-handle = <&phy0>;
        rgmii_0_eth_tse_0_mdio: mdio {
            compatible = "altr,tse-mdio";
            #address-cells = <1>;
            #size-cells = <0>;
            phy0: ethernet-phy@16 {
                //MARVELL_PHY_ID_88E1111 0x01410cc0 "ethernet-phy-id0141.0cc0",
                compatible = "ethernet-phy-ieee802.3-c22";
                reg = <16>;
                device_type = "ethernet-phy";
            };
        };
    };
```

在操作系统启动时，MAC驱动会首先尝试经MDIO访问Phy，访问成功即认为网络控制器正常。在进入用户态后，查看网络接口时见到eth0接口即为板载以太网。

2.6 Flash控制器

板载Flash使用了标准的并口时序，因此可由Altera的通用三态控制器IP核控制。再QSYS中添加通用三态控制器后，需要根据Flash芯片手册配置控制信号、延迟等参数。通用三态控制器仅进行时序转换，不处理通信内容，故Flash将被直接映射至CPU总线的地址空间，所有Flash控制指令均由软件产生。

在本项目中，Flash仅用于存储操作系统本身，没有作为文件系统。因此Flash仅被Bootloader访问，在系统启动后就不再使用，不需要在Linux中作任何配置。

2.7 DRAM控制器

为了获得较大的RAM空间，我们使用了板载的SDRAM芯片。由于SDRAM控制较为复杂，因而需要SDRAM控制器实现SDRAM的读、写、刷新操作。我们使用了Altera提供的SDRAM控制器IP，在QSYS中添加该元件后，需要根据SDRAM芯片手册配置容量、延迟等参数。由于内存工作在150Mhz频率上，对于I/O速度的要求较高，因而需要通过调整时钟补偿I/O上的延迟。实际中我们用PLL对SDRAM芯片的时钟增加了-100°的相位差，否则在突发内存访问时可能出现数据错误的情况，将影响以太网DMA和Cache的正常工作。

2.8 GPIO控制器

GPIO控制器作为普通输入、输出引脚到CPU的桥梁，在本系统中用于驱动LED和读开关状态。QSYS中共添加了两个GPIO控制器，其中一个配置为仅支持输出，连接至LED；另一个配置为仅输入，与开关连接。配置为输入的GPIO控制器同时启用了输入中断捕获功能，可作为中断控制器使用，系统中的部分外设（如加速度计）中断信号就连接在GPIO上。

Altera的GPIO在Linux下的驱动程序选项为`CONFIG_GPIO_ALTERA`。内核设备树中条目的内容为：

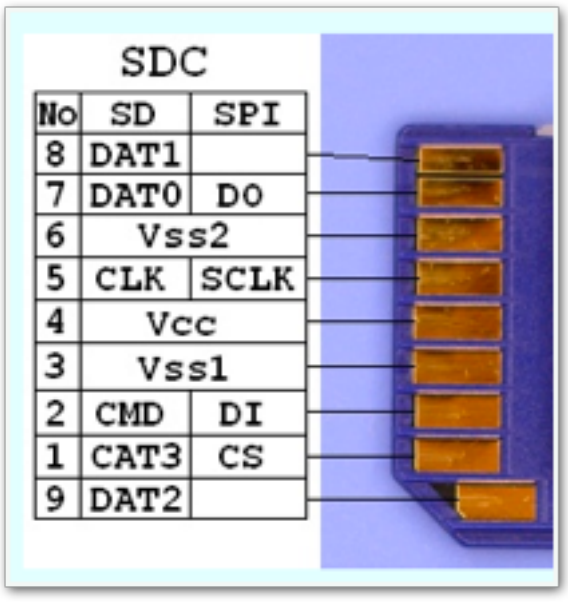
```
    led_pio: gpio@0xbfd00400 {
        compatible = "altr,pio-1.0";
        reg = <0xbfd00400 0x10>;
        altr,ngpio = <32>;
        resetvalue = <0xaa>;
        #gpio-cells = <2>;
        gpio-controller;
    };

    button_pio: gpio@0xbfd00410 {
        compatible = "altr,pio-1.0";
        reg = <0xbfd00410 0x10>;
        altr,ngpio = <32>;
        resetvalue = <0>;
        #gpio-cells = <2>;
        gpio-controller;
        interrupt-parent = <&cpuintc>;
        interrupts = <6>;
        altr,interrupt-type = <IRQ_TYPE_LEVEL_HIGH>;
        #interrupt-cells = <2>;
        interrupt-controller;
    };
```

在用户态下，所有GPIO引脚均可通过`/sys/class/gpio/`目录下的文件访问，访问方法遵循Linux GPIO子系统的标准流程。

2.9 SD卡控制器

DE2i平台板载了SD卡槽，并将所有SD卡的信号线与FPGA相连，这使得我们可以在FPGA中实现SD卡控制器来访问SD卡。SD有多种访问协议，其中最简单（但低效）的访问方式为SPI串行协议，该协议仅需使用SD卡中部分引脚：



在QSYS中，可以直接用Altera提供的3线SPI控制器IP作为SPI主机，使得CPU能够访问存储卡。SPI协议参数可以直接使用默认值。

SPI控制器在Linux下的驱动程序选项为`CONFIG_SPI_ALTERA`，为了使Linux支持由SPI访问存储卡，还需要启用选项`CONFIG_MMC_SPI`。除此之外，为了支持文件系统，还需要在内核配置中打开块设备子系统、文件系统的相关选项。

在内核设备树中，支持存储卡需要添加的条目的内容为：

```
    spi: spi@0xbc030000 {
        #address-cells = <1>;
        #size-cells = <0>;
        cell-index = <0>;
        compatible = "altr,spi-1.0";
        reg = <0xbc030000 0x20>;

        mmc-slot@0 {
            compatible = "fsl,mpc8323rdb-mmc-slot",
                     "mmc-spi-slot";
            reg = <0>;
            voltage-ranges = <3300 3300>;
            spi-max-frequency = <4000000>;
        };
    };
```

系统启动后会不断轮询检查存储卡插入，当检测到存储卡并初始化成功后，将出现`/dev/mmcblk0`设备。用户态程序及文件系统可以通过该设备访问存储卡内容。

2.10 内存地址映射

组件	起始地址	结束地址
DRAM	0x00000000	0x07ffffff
Flash	0x10000000	0x17ffffff
Ethernet Control	0x1c000000	0x1c0003ff
Ethernet Rx CSR	0x1c001800	0x1c00181f
Ethernet Rx Desc	0x1c001900	0x1c00191f
Ethernet Rx Resp	0x1c001a00	0x1c001a07
Ethernet Tx CSR	0x1c001b00	0x1c001b1f
Ethernet Tx Desc	0x1c001c00	0x1c001c1f
I2C Host	0x1c010000	0x1c01001f
SPI	0x1c030000	0x1c03001f
BootROM	0x1fc00000	0x1fc01fff
UART	0x1fd003e0	0x1fd003ff
GPIO Output	0x1fd00400	0x1fd0040f
GPIO Input	0x1fd00410	0x1fd0041f

3 应用程序

3.1 Busybox根文件系统

我们编译了Busybox工具集作为用户态的根文件系统，该工具集中包含shell及基本的命令行工具实现。编译好的Busybox放置于Linux代码的`busybox-net/`目录下，并在内核配置中指定`CONFIG_INITRAMFS_SOURCE="busybox-net"`，可以使得编译时将busybox打包到内核中，并在内核引导结束后自动挂载至根目录，并运行初始化程序`/sbin/init`。这样不要额外的磁盘设备或者文件系统支持，就可以在运行用户态程序。

3.2 Decaf编译器移植

Decaf编译器的移植工作沿用了编译原理挑战性课程任务的成果，主要修改了libdecaf运行时库的实现及编译参数。

3.2.1 运行时库libdecaf

本运行时库由"decafIo"和"decafCall"两部分组成，其中，"decafCall"用汇编书写，用以实现 Decaf 程序运行时库函数，以供 Decaf 程序调用；"decafIo"用 C 语言书写，用以实现相关库函数的具体逻辑。

库函数实现位于`libdecaf/decafIo.c`文件中，下面给出各个库函数的设计。

- _Alloc 分配内存。调用C语言的malloc函数。
- _StringEqual 比较两个字符串。直接使用strcmp函数比较字符串，strcmp返回0则说明相等，返回true，否则返回false。
- _ReadLine 读取一行字符串。直接用fgets函数实现。
- _ReadInteger 读取一个整数。直接用scanf函数实现。
- _PrintInt 打印一个整数。直接用printf函数实现。
- _PrintString 打印一个字符串。直接用printf函数实现。
- _PrintBool 打印一个布尔值。根据值转为"true"或"false"，用printf函数打印。
- _Halt 结束程序。调用C语言的exit函数，结束当前进程。

3.2.2 交叉编译工具

我们基于GCC-MIPS和libdecaf整合了完整的交叉编译工具`mips-linux-decaf`，使用时可以用如下命令行直接从decaf源文件生成可执行文件。

```
mips-linux-decaf <source.decaf> <executable>
```

该工具内部工作流程为：

1. 调用decaf编译器将decaf源代码编译为汇编
2. 调用as将汇编代码编译为.o目标文件
3. 调用gcc将目标文件和libdecaf链接为可执行文件

3.3 演示程序

为了演示综合实验的成果，我们开发、移植了两个应用程序。

3.3.1 NaiveFTP

NaiveFTP是计算机网络原理课程大实验中实现的一个简单FTP服务器。在本项目中，我们对其功能进行了少量修改，去掉了对多线程的依赖（因为系统缺少libpthread）。源代码位于linux代码库的`userland-sample/naive_ftp`目录。

将移植好的NaiveFTP打包进根文件系统后，在网络连接正常的情况下，用`naiveftp -p 22`命令即可启动FTP服务器。

3.3.2 加速度计演示程序

由于硬件上实现了加速度计传感器接口并启动了相关驱动程序，用户态程序可以读取到加速度数据。该演示程序将读取X轴重力加速度数据，随后驱动LED以温度计码的形式展现，实现倾角测量。程序源代码位于linux代码库的`userland-sample/accelerometer.c`。

4 使用说明

4.1 交叉编译工具链

编译NaiveMIPS的Linux内核需要GCC编译工具链，经过验证的GCC版本为5.3.0，目标为mipsel-unknown-linux-gnu-gcc。如果自行编译GCC，可以按如下参数配置：

`../configure --target=mipsel-unknown-linux-gnu --enable-languages=c`

实验中可能用到binutils用于反编译、提取二进制代码等，如果自行编译binutils，可以按如下参数配置：

`./configure --disable-debug --disable-dependency-tracking --disable-werror --enable-interwork --enable-multilib --enable-64-bit-bfd --enable-targets=all`

其中`--enable-targets=all`可以使编译出的binutils支持所有平台（包括MIPS）。

4.2 配置和编译Linux内核

下载内核源码源码后，进入代码顶层目录，用如下命令生成适用于NaiveMIPS的初始内核配置：

`make ARCH=mips CROSS_COMPILE=mipsel-unknown-linux-gnu- naivemips_de2i_defconfig`

上述命令将产生`.config`内核配置文件。如需编辑配置，可以使用命令：

`make ARCH=mips CROSS_COMPILE=mipsel-unknown-linux-gnu- menuconfig`

配置完成后，开始编译：

`make ARCH=mips CROSS_COMPILE=mipsel-unknown-linux-gnu- -j4`

其中`-j4`是指用4线程编译，可以根据实际情况调整。

编译成功后将产生`vmlinux`文件，即整个内核的elf格式可执行文件。将该文件用开发板工具写入Flash后，将SW0置于1位置，复位即可引导操作系统。开发过程中可将SW0置于0位置，进入串口模式，通过NaiveBootloader上位机工具从串口引导内核。

5 附录

5.1 实验代码仓库

- CPU HDL Project
- Linux Kernel Source
- NaiveBootloader
- Decaf Compiler
- libdecaf

5.2 参考资料

- MIPS32规范-指令集: Mips_vol2_InstructionSetReference.pdf
- MIPS32规范-特权资源：MIPS_Vol3.pdf
- 《Linux设备驱动开发详解：基于最新的Linux 4.0内核》
- 内核文档（内核源码树的Documentation目录）
- Altera Avalon 总线规范
- Altera IP 手册
- OpenCores I2C 控制器

5.3 指令集

最终使用到的指令（含伪指令）共74条：

```
addiu
addu
and
andi
b
beq
beqz
bgez
bgtz
blez
```

```
bltz
bne
bnez
cache
div
divu
ehb
eret
j
jal
jalr
jr
lb
lbu
lh
lhu
li
lui
lw
lwl
lwr
madd
maddu
mfc0
mfhi
mflo
move
movn
movz
mtc0
mthi
mtlo
mul
mult
multu
negu
nop
nor
or
ori
pref
sb
sh
sll
sllv
slt
slti
sltiu
sltu
sra
srav
srl
srlv
ssnop
subu
sw
swl
swr
sync
tlbr
tlbwi
wait
xor
```

```
xori
```