电子工程世界 eeworld.com.cn

ALTERA®



Altera SoC

深度体验之旅

前言

处理器和 FPGA 是大部分嵌入式系统中最关键的内核。SoC FPGA 器件在一个器件中同时集成了处理器和 FPGA 体系结构。由于其具有更高的集成度、更低的功耗、更小的电路板面积，以及处理器和 FPGA 之间带宽更大的通信以及丰富的外设、片内存储器、FPGA 类型的逻辑阵列，以及高速收发器等优点，应用 SoC FPGA 器件已成为嵌入式领域的趋势之一，成为许多产品的最佳解决方案。

本书旨在将 EEworld 网友参与 Altera SoC 深度体验活动（EEworld 与 Altera 联合举办）所产生的学习心得、评测文章、设计文档整理在册，以便给更多想学习和使用 SoC 的朋友们一些参考和实例。

本书中所有文章的撰写基于 Altera Cyclone V 系列 SoC FPGA（后文简称 Altera Cyclone V SoC 或 Altera SoC），具体的硬件平台分为四种：TerAsic DE1-SoC board、Embest SoC board、Arrow SoC Kit、Macnica Helio Kit（板卡资料可查看：http://www.eeworld.com.cn/huodong/201411Altera/prize.html）。书的内容按硬件平台的不同分为四部分，文章内容按照从入门到评测到设计的思路编排，由浅入深，即便是初学者也能从文章中获益。

Altera Cyclone V SoC 集成了双核 ARM Cortex-A9 处理器，同时具有 NEON 媒体处理引擎、单精度 / 双精度浮点单元、L1 和 L2 高速缓存、ECC 保护存储器控制器、ECC 保护高速暂存存储器，以及多种常用外设。器件利用了其 28nm 系列产品，在工艺技术、收发器技术、IO 资源和硬核 IP 等方面进行了创新。其器件体系结构具有提高系统性能、增强可靠性、更加灵活、降低系统成本、提高效能的优势，网友们也通过评测、数据分析、实际设计真切地体会到了 Altera SoC 的这些优势，获得了不俗的用户体验以及在 Altera SoC 开发方面，从开发环境的搭建、SoC 外设功能的使用，ARM 和 FPGA 的协同工作、操作系统、应用程序开发等等方面长足的进步：

“从工程师的角度来说，我觉得 SoC 最大的亮点是处理器和 FPGA 之间的高速互联，这种架构给了工程师更多选择和想象的空间，使得设计更加的灵活，缩短了产品开发周期，提升了产品开发效率，增强了产品的耦合度。”

——EEworld 网友 chenzhufly

“技术在不断更新，想以此生存必须更新自己的知识库。我觉得现在已经到了不得不学 SoC 的时候了，而且时间紧迫，因为它已经成为很多产品的最佳解决方案了。做为一个目前还不排斥新技术的人说，FPGA SoC 这么强大的处理能力，想想都觉得激动呢，因此抛开生存这个沉重话题，了解它、研究它、使用它何尝不是一种乐趣。”

——EEworld 网友 muhan9

“这次体验让我熟悉和了解了什么是 SoC,SoC 到底能给产品设计带来什么样的便利，同时在进行 SoC 开发的过程又会遇到什么样的问题，如何应对和解决。”

——EEworld 网友 chenzhufly

“一个执行力强的和一个想法多的人合作做一件事情，就像 FPGA 和 ARM 通过静态存储总线通信，虽然比孤军奋战强很多，但是还是沟通带宽有限。一个强大的团队，应该像 SoC 一样，既有灵活的思维，

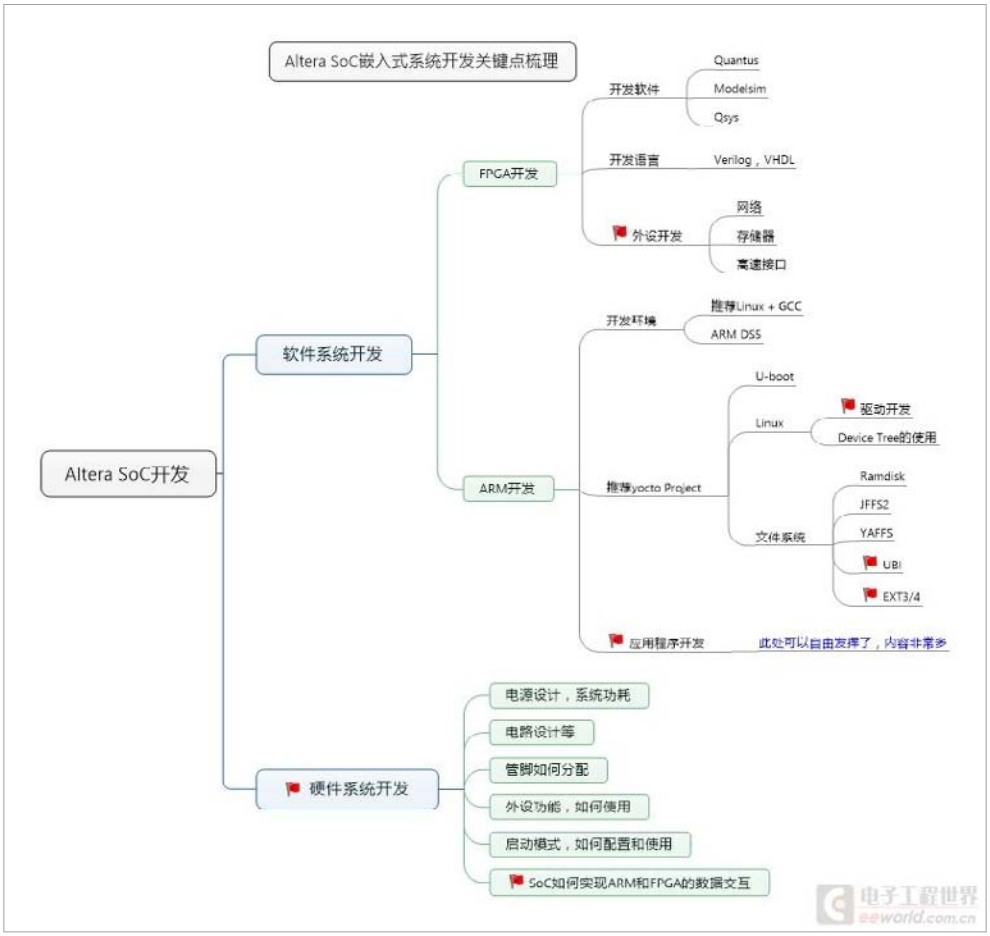
又有超强的执行力，同时还有超大的带宽方便沟通。”

——EEworld 网友小梅哥

“体验活动虽然结束了，但是 Altera SoC 探索之路还应该继续走下去”。

——EEworld 网友小梅哥

最后，附一张网友 chenzhufly 用思维导图软件整理的一些学习要点，给想学习 Altera SoC 的朋友们更直观的参考。



2015.9.20

EEworld 社区

目录

前言 2

一 、本章应用平台为友晶 DE1-SoC 7

1. 初见 DE1-SoC 7

1.1 友晶 DE1-SoC 官方资料，图片，参考设计 7

1.2 网友眼中的 DE1-SoC 10

1.2.1 开发环境的准备 10

1.2.2 亮起流水灯！ 10

2. 让 SoC 飞一会儿 18

2.1 Altera SoC 体验之 HPSFPGA 18

2.2 Altera SoC 体验之 linux+QT 26

2.3 DE1-SoC 之 LXDE 桌面系统 34

2.4 FPGA 与 ARM 的窃窃私语 38

2.5 Altera SoC 之地址映射 41

3. 设计与应用 43

3.1 DE1-SoC—人脸检测 43

3.2 基于 OPENCL 的卷积神经网络 48

3.2.1 开启 OpenCL 模式 48

3.2.2 DE1 上跑完整的卷积神经网络 63

4. 用户体验与感想 73

4.1 SoC 在系统集成方面的优势 73

4.2 SoCFPGA 之 DSP 进阶 ---NEON 高性能媒体引擎 75

4.2.1 前言 75

4.2.2 之一 77

4.2.3 之二 82

4.3 SoC 之从用户的角度谈谈使用体验 95

4.4 SoC 之从 GPU 程序员的角度谈谈使用体验 97

二、本章应用平台为：英蓓特的 lark board 99

1. 初见 lark board 99
1.1 官方资料、图，参考设计 99
1.2 网友眼中的 lark 101
1.2.1 Lark “冒烟” 测试 101
1.2.2 Lark 上 ARM 开发环境搭建和配置 106
1.2.3 Lark 上 ARM 开发之编译 u-boot 和内核 107
2. 让 SoC 飞一会儿 109
2.1 LWHPS2FPGA 深入浅出 - 点亮 LED 109
2.2 Enpirion Altera SoC 电源系统 113
2.3 Enpirion Altera SoC 电源上电时序 117
2.4 显示系统初探 121
3. 设计与应用 123
3.1 基于 Cyclone V SoC 的高速数据采集系统设计 123
3.1.1 概述 123
3.1.2 高速数据采集之 ADC（1） 126
3.1.3 高速数据采集之 ADC（2） 129
3.1.4 高速数据采集之 ADC（3） 131
3.1.5 高速数据采集之中断 136
3.1.6 高速数据采集之数据传输（1） 143
3.1.7 高速数据采集之数据传输（2） 146
3.1.8 高速数据采集之数据显示（1） 152
3.1.9 高速数据采集之数据显示（2） 156
3.2 多串口服务器设计 165
32.1 概述 165
3.2.2 多串口服务器之串口模块设计 168
4. 用户体验与感想 183
4.1 关于 SoC 系统应用 183
4.2 基于 iperf 的 Altera SoC 千兆网测试 184
4.3 使用 nbench 测试 Altera SoC 性能 188
4.4 主流平台的 nbench 性能测试大 PK 191

三、本章应用平台为 ARROW SoCkit 196

1. 初见 ARROW SoCkit 196
1.1 官方资料、图片、参考设计 196
1.2 网友眼中的 SoCkit 198
1.2.1 烧写和运行 linux 系统 198
1.2.2 Linux 系统初体验 199
1.2.3 So FPGA 配置与启动 202
1.2.4 Altera SoC 特点总结 204
2. 让 SoC 飞一会儿吧 206
2.1 Arrow SoC Kit 学习之拨码点灯 206
2.2 Arrow SoC Kit 学习之 LCD 显示 211
2.3 使用 System Console 验证硬件系统 214
2.4 利用 System Console 工具对 SoC 设计进行调试 215
2.5 对 ADC 采样数据进行 FFT 处理 221
3. 应用与设计 223
3.1 电源设计 223
3.2 Altera SoC 之移植 LINUX 系统 + 示波器界面 225
4. 用户体验与感想 231
4.1 对于 Altera SoC 用于高清相机控制分系统设计的感触 231

四 本章硬件平台为 Macnica Helio Kit 233
1. 初见 Macnica Helio Kit 233
1.1 官方资料、图片、考设计 233
1.2 网友眼中的 Helio Kit 234
1.2.1 Helio Cyclone V SoC 开发板应用之 FPGA 入门体验 234
1.2.2 Helio Cyclone V SoC 开发板应用之 linux 准备工作 238
2. 让 SoC 飞一会儿吧 241
2.1 Helio Cyclone V SoC 开发板应用之高速 DAC 波形发生一 241

附录 243

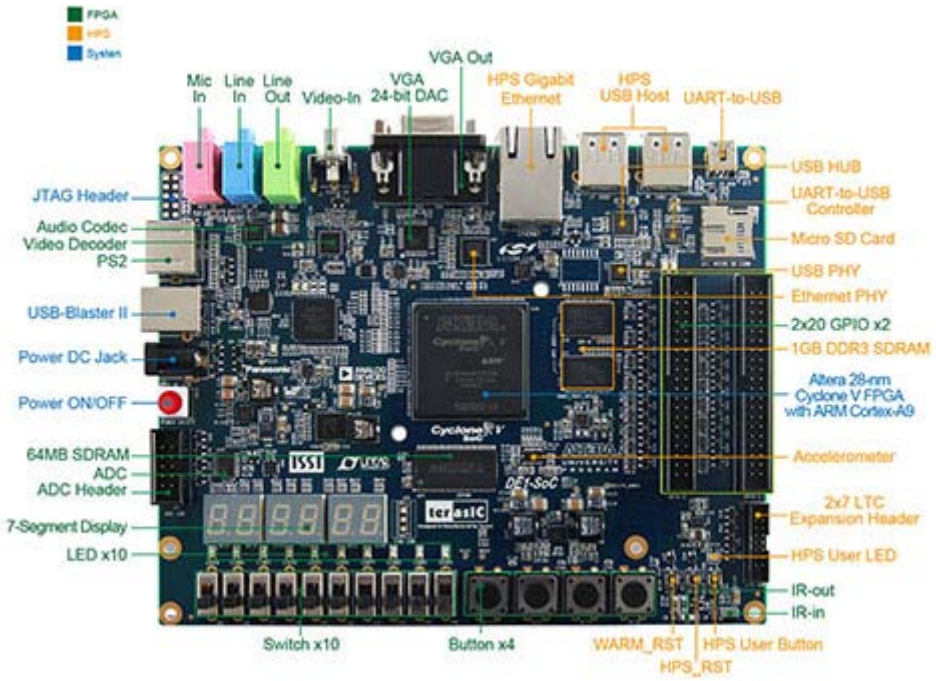
版权声明 244

一 、本章应用平台为友晶 DE1-SoC

1. 初见 DE1-SoC

1.1 友晶 DE1-SoC 官方资料，图片，参考设计

TerAsic DE1-SoC board





主芯片：

Cyclone V SoC 5CSEMA5F31C6 Device

Dual-core ARM Cortex-A9 (HPS)

85K Programmable Logic Elements

配置以及调试：

Quad Serial Configuration device – EPCQ256 on FPGA

On-Board USB Blaster II (Normal type B USB connector)

存储设备：

64MB (32Mx16) SDRAM on FPGA

1GB (2x256Mx16) DDR3 SDRAM on HPS

Micro SD Card SoCket on HPS

通讯：

两个 Port USB 2.0 Host (ULPI interface with USB type A connector)

USB to UART (micro USB type B connector)

10/100/1000 Ethernet

PS/2 mouse/keyboard

IR Emitter/Receiver

外扩接口：

两个 40-pin 扩展头 (voltage levels: 3.3V)

一个 10-pin ADC 输入头

一个 LTC 接口 (一个 Serial Peripheral Interface (SPI) Master , 一个 I2C 以及一个 GPIO)

Display：

24-bit VGA DAC

Video Input：

TV Decoder (NTSC/PAL/SECAM) and TV-in connector

开关，按键以及显示灯：

4 用户按键 (FPGA x4)

一个 10-pin ADC 输入头

11 用户 LEDs (FPGA x10 ; HPS x 1)

2 HPS Reset (HPS_RST_n and HPS_WARM_RST_n)

Six 7-segment displays

传感器：

G-Sensor on HPS

设计资源：

http://www.terasic.com.cn/cgi-bin/page/archive.pl?Language=China&CategoryNo=203&No=870&PartNo=4

参考视频：http://www.tudou.com/programs/view/C9uN-gYepT0/

详情请点击：http://www.eeworld.com.cn/huodong/201411Altera/prize.html



1.2 网友眼中的 DE1-SoC

1.2.1 开发环境的准备

昨天 DE1 到手，就准备装开发环境，因为 QII 之前有装是 13.1，所以打算装个 EDS 就可以开工了，到 Altera 下载了个 SoCEDS 14.1 版本，等下完安装的时候提示这个软件只能装在 64 位系统上，一下蛋疼了，现在用的 xp，装了诸多的软件，重装系统麻烦且相当费时，硬盘差不多也满了。果断到某宝买了个 1T 的硬盘，打算把笔记本的光驱搞成硬盘仓，原来的系统还在原来硬盘不变，买完硬盘之后感觉 64 位的系统我这 4G 内存有点小，于是又淘了个 4G 的内存条。心疼死了！

下完单想着这软件真蛋疼，然后打算试试低版本是不是可以装到 32 位系统上，又下载了个 13.1 版本的 SoCEDS，去嘞，这个居然能够装到 32 位上，看到文档上的版本也是 13.1，先凑合着用吧，等硬盘来装系统也得差不多一周的时间了。

简要说下这里要装的环境 quartusII， 这个版本不要太老，太老不支持 SoC，不要太新，最新的应该是没有和谐版本吧，虽然和谐方法都是一样的，但是不同版本的和谐文件应该是不能相互使用的。或者使用此次 Altera 提供的半年 license。

SoCEDS，Altera SoC Embedded Design Suite，这个是 DE-5 的 Altera 定制版，新版本只支持 64 位系统，使用 32 位操作系统的要注意了。还好 13.1 能够装在 32 位系统上。

友晶提供的文档上 QII 使用的是 13.1 版本，SoCEDS 使用的也是 13.1 版本。现在因为操作系统迫不得已跟文档上使用的一样了。哈哈。

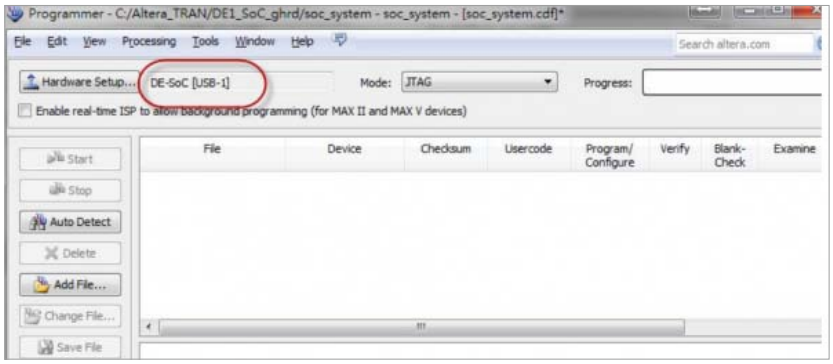
SoCEDS 的社区版是永久免费，Altera 版允许 30 天的免费评估，区别从以下地址找 http://ds.arm.com/zh-cn/Altera/SoCEDS 以下地址下载 https://www.Altera.com/download/software/SoC-eds

详情请点击：

http://bbs.eeworld.com.cn/thread-454246-1-1.html

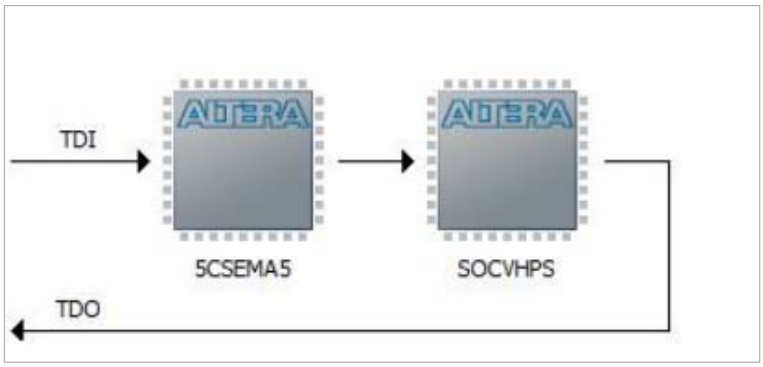
1.2.2 亮起流水灯！

至于 DE1_SoC 的板子简介我就不多说了，拿到板子首先按照培训资料（见附件）中的教程安装 usb blaster ii 的驱动，串口驱动等，然后按照教程下载测试例程，看到的是一个流水灯和数码管的 0~F 轮回计数。过程中想说的两点是，

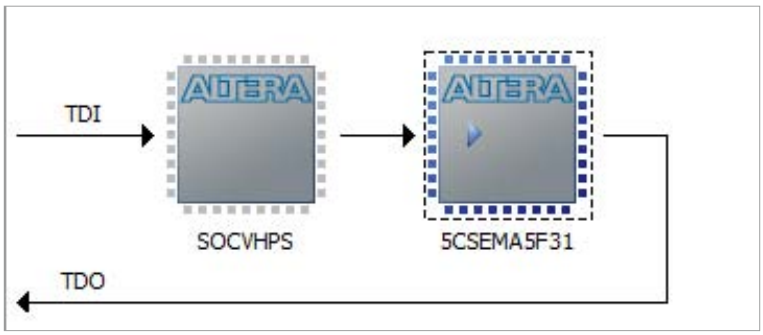


开始是 NO hardware，要稍微过一会才会有 DE-SoC[USB-1](感觉时间稍微有点长)，另外一个教

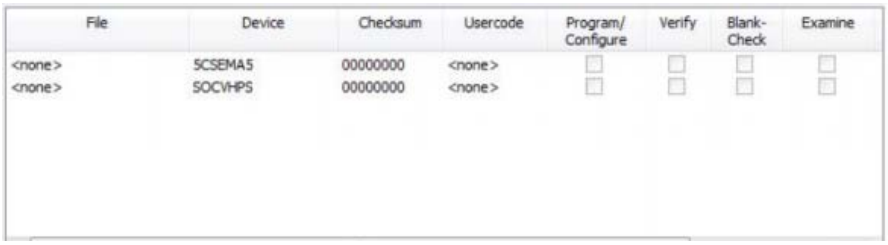
程中 programmer 面 Auto Detect 器件出现的是 5CSEMA5 然后是 SoCVHPS



而我的是



SoCVHS 在左，5CSEMA5 在右，开始选中了其中的 5CSEMA5 然后右键 up，两者位置交换如下图



但是 program 出现错误，又 Auto Detect 一次 program 正常。至于两者的位置不同有什么区别还不知道，以后再探究吧。

按照教程烧 SD 镜像，启动系统。

```
U-Boot SPL 2013.01.01 (Nov 04 2013 - 19:51:38)
BOARD : Altera SoCFPGA Cyclone V Board
SDRAM: Initializing MMR registers
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
Altera DWMMC: 0

U-Boot 2013.01.01 (Oct 24 2013 - 17:40:22)

CPU   : Altera SoCFPGA Platform
```

```
BOARD : Altera SoCFPGA Cyclone V Board
DRAM:  1 GiB
MMC:   Altera DWMMC: 0
In:    serial
Out:   serial
Err:   serial
Net:   mii0
Warning: failed to set MAC address

Hit any key to stop autoboot:  5  4  3  2  1  0
reading u-boot.scr
** Unable to read file u-boot.scr **
Optional boot script not found. Continuing to boot normally
reading zImage
3809104 bytes read in 1297 ms (2.8 MiB/s)
reading SoCFPGA.dtb
17119 bytes read in 14 ms (1.2 MiB/s)
FPGAintf
ffd08028: 00000000    ....
FPGA2sdram
ffc25080: 00000000    ....
axibridge
ffd0501c: 00000000    ....
## Flattened Device Tree blob at 00000100
   Booting using the fdt blob at 0x00000100
   Loading Device Tree to 03ff8000, end 03fff2de ... OK

Starting kernel ...

Booting Linux on physical CPU 0x0
Initializing cgroup subsys cpuset
Linux version 3.12.0-00307-g507abb4-dirty (root@matthew) (gcc version 4.6.3 (Sourcery
CodeBench Lite 2012.03-57) ) #2 SMP Mon Jan 6 19:54:56 CST 2014
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=10c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Altera SoCFPGA, model: Altera SoCFPGA Cyclone V
Memory policy: ECC disabled, Data cache writealloc
PERCPU: Embedded 8 pages/cpu @80fd1000 s11328 r8192 d13248 u32768
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 260096
Kernel command line: console=ttyS0,115200 root=/dev/mmcblk0p2 rw rootwait
PID hash table entries: 4096 (order: 2, 16384 bytes)
Dentry cache hash table entries: 131072 (order: 7, 524288 bytes)
Inode-cache hash table entries: 65536 (order: 6, 262144 bytes)
Memory: 1031484K/1048576K available (5637K kernel code, 253K rwdata, 1424K rodata, 343K
init, 256K bss, 17092K reserved)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xc0800000 - 0xff000000   (1000 MB)
    lowmem  : 0x80000000 - 0xc0000000   (1024 MB)
```

```
    modules : 0x7f000000 - 0x80000000   (  16 MB)
      .text : 0x80008000 - 0x806ed828   (7063 kB)
      .init : 0x806ee000 - 0x80743c40   ( 344 kB)
      .data : 0x80744000 - 0x807834f0   ( 254 kB)
       .bss : 0x807834f0 - 0x807c355c   ( 257 kB)
SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=2, Nodes=1
Hierarchical RCU implementation.
NR_IRQS:16 nr_irqs:16 16
sched_clock: 32 bits at 100MHz, resolution 10ns, wraps every 42949ms
Console: colour dummy device 80x30
Calibrating delay loop... 1594.16 BogoMIPS (lpj=7970816)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
ftrace: allocating 17704 entries in 52 pages
CPU0: thread -1, cpu 0, SoCket 0, mpidr 80000000
Setting up static identity map for 0x8051f018 - 0x8051f070
CPU1: Booted secondary processor
CPU1: thread -1, cpu 1, SoCket 0, mpidr 80000001
Brought up 2 CPUs
SMP: Total of 2 processors activated.
CPU: All CPU(s) started in SVC mode.
devtmpfs: initialized
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
NET: Registered protocol family 16
FPGA bridge driver
DMA: preallocated 256 KiB pool for atomic coherent allocations
L310 cache controller enabled
l2x0: 8 ways, CACHE_ID 0x410030c9, AUX_CTRL 0x32460000, Cache size: 512 kB
syscon fffef000.l2-cache: regmap [mem 0xfffef000-0xffffffff] registered
syscon ffd05000.rstmgr: regmap [mem 0xffd05000-0xffd05fff] registered
syscon ffc25000.sdrctl: regmap [mem 0xffc25000-0xffc25fff] registered
syscon ff800000.l3regs: regmap [mem 0xff800000-0xff800fff] registered
syscon ffd08000.sysmgr: regmap [mem 0xffd08000-0xffd0bfff] registered
hw-breakpoint: found 5 (+1 reserved) breakpoint and 1 watchpoint registers.
hw-breakpoint: maximum watchpoint size is 4 bytes.
Altera_hps2FPGA_bridge FPGAbridge.2: FPGA bridge [hps2FPGA] registered as device

hps2FPGA
Altera_hps2FPGA_bridge FPGAbridge.2: init-val not specified
Altera_hps2FPGA_bridge FPGAbridge.3: FPGA bridge [lshps2FPGA] registered as device

lwhps2FPGA
Altera_hps2FPGA_bridge FPGAbridge.3: init-val not specified
Altera_hps2FPGA_bridge FPGAbridge.4: FPGA bridge [FPGA2hps] registered as device

FPGA2hps
Altera_hps2FPGA_bridge FPGAbridge.4: init-val not specified
bio: create slab <bio-0> at 0
FPGA Mangager framework driver
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
```

```
usbcore: registered new device driver usb
pps_core: LinuxPPS API ver. 1 registered
pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
PTP clock support registered
Switched to clocksource timer1
NET: Registered protocol family 2
TCP established hash table entries: 8192 (order: 4, 65536 bytes)
TCP bind hash table entries: 8192 (order: 4, 65536 bytes)
TCP: Hash tables configured (established 8192 bind 8192)
TCP: reno registered
UDP hash table entries: 512 (order: 2, 16384 bytes)
UDP-Lite hash table entries: 512 (order: 2, 16384 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX SoCket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
hw perfevents: enabled with ARMv7 Cortex-A9 PMU driver, 7 counters available
arm-pmu arm-pmu: PMU:CTI successfully enabled
NFS: Registering the id_resolver key type
Key type id_resolver registered
Key type id_legacy registered
NTFS driver 2.1.30 [Flags: R/W].
jffs2: version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
msgmni has been set to 2014
io scheduler noop registered (default)
Serial: 8250/16550 driver, 2 ports, IRQ sharing disabled
ffc02000.serial0: ttyS0 at MMIO 0xffc02000 (irq = 194, base_baud = 6250000) is a 16550A
console [ttyS0] enabled
Altera_FPGA_manager ff706000.FPGAmgr: FPGA manager [Altera FPGA Manager] registered as
minor 0
brd: module loaded
cadence-qspi ff705000.spi: DMA NOT enabled
cadence-qspi ff705000.spi: master is unqueued, this is deprecated
m25p80 spi2.0: unrecognized JEDEC id ffffff
cadence-qspi ff705000.spi: Cadence QSPI controller driver
dw_spi_mmio fff01000.spi: master is unqueued, this is deprecated
CAN device driver interface
c_can_platform ffc00000.d_can: invalid resource
c_can_platform ffc00000.d_can: control memory is not used for raminit
c_can_platform ffc00000.d_can: c_can_platform device registered (regs=c08e4000, irq=163)
stmmac - user ID: 0x10, Synopsys ID: 0x37
Ring mode enabled
DMA HW capability register supported
Enhanced/Alternate descriptors
        Enabled extended descriptors
RX Checksum Offload Engine supported (type 2)
TX Checksum insertion supported
Enable RX Mitigation via HW Watchdog Timer
libphy: stmmac: probed
```

```
eth0: PHY ID 00221611 at 1 IRQ 0 (stmmac-0:01) active
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
Synopsys Designware Multimedia Card Interface Driver
dwmmc_SoCFPGA ff704000.dwmmc0: Using internal DMA controller.
dwmmc_SoCFPGA ff704000.dwmmc0: Version ID is 240a
dwmmc_SoCFPGA ff704000.dwmmc0: DW MMC controller at irq 171, 32 bit host data width,
1024 deep fifo
mmc_host mmc0: Bus speed (slot 0) = 12500000Hz (slot req 400000Hz, actual 390625HZ div
= 16)
dwmmc_SoCFPGA ff704000.dwmmc0: 1 slots initialized
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
mmc_host mmc0: Bus speed (slot 0) = 12500000Hz (slot req 12500000Hz, actual 12500000HZ
div = 0)
mmc0: new high speed SDHC card at address 1234
mmcblk0: mmc0:1234 SA16G 14.6 GiB
mmcblk0: p1 p2 p3
dwc2 ffb40000.usb: DWC OTG Controller
dwc2 ffb40000.usb: new USB bus registered, assigned bus number 1
dwc2 ffb40000.usb: irq 160, io mem 0x00000000
usb usb1: New USB device found, idVendor=1d6b, idProduct=0002
usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb1: Product: DWC OTG Controller
usb usb1: Manufacturer: Linux 3.12.0-00307-g507abb4-dirty dwc2_hsotg
usb usb1: SerialNumber: ffb40000.usb
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
oprofile: using arm/armv7-ca9
TCP: cubic registered
NET: Registered protocol family 10
sit: IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
NET: Registered protocol family 15
can: controller area network core (rev 20120528 abi 9)
NET: Registered protocol family 29
can: raw protocol (rev 20120528)
can: broadcast manager protocol (rev 20120528 t)
can: netlink gateway (rev 20130117) max_hops=1
8021q: 802.1Q VLAN Support v1.8
Key type dns_resolver registered
ThumbEE CPU extension supported.
Registering SWP/SWPB emulation handler
kjournald starting.  Commit interval 5 seconds
EXT3-fs (mmcblk0p2): using internal journal
EXT3-fs (mmcblk0p2): mounted filesystem with ordered data mode
VFS: Mounted root (ext3 filesystem) on device 179:2.
devtmpfs: mounted
Freeing unused kernel memory: 340K (806ee000 - 80743000)
```

```
usb 1-1: new high-speed USB device number 2 using dwc2

INIT: version 2.88 booting

usb 1-1: New USB device found, idVendor=0424, idProduct=2512
usb 1-1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
hub 1-1:1.0: USB hub found
hub 1-1:1.0: 2 ports detected
Starting Bootlog daemon: bootlogd.

Configuring network interfaces... eth0: device MAC address 6a:a9:48:f6:c3:f3
udhcpc (v1.20.2) started

Sending discover...
Sending discover...
Sending discover...
No lease, failing
Starting portmap daemon...
Sat Sep 28 04:38:00 UTC 2013
INIT: Entering runlevel: 5
Starting OpenBSD Secure Shell server: sshd
done.
Starting syslogd/klogd: done
Starting Lighttpd Web Server: lighttpd.
Starting blinking LED server
Stopping Bootlog daemon: bootlogd.
Poky 8.0 (Yocto Project 1.3 Reference Distro) 1.3
ttyS0
SoCFPGA login: root
root@SoCFPGA:~# p who i am
                                                                        am i
BusyBox v1.20.2 (2013-09-27 23:27:54 CDT) multi-call binary.
Usage: who [-a]
root@SoCFPGA:~# poweroffer
Broadcast message from root@SoCFPGA (ttyS0) (Sat Sep 28 04:39:11 2013):
The system is going down for system halt NOW!
INIT:
INIT: Sending processes the TERM signal
root@SoCFPGA:~#
INIT: Sending processes the KILL signal
Stopping OpenBSD Secure Shell server: sshdstopped /usr/sbin/sshd (pid 148)
.Stopping syslogd/klogd: stopped syslogd (pid 152)
stopped klogd (pid 155)
done
Deconfiguring network interfaces... ifdown: interface eth0 not configured
done.
Stopping Lighttpd Web Server: stopped /usr/sbin/lighttpd (pid 159)
lighttpd.
/etc/rc0.d/K90gsrd_init.sh: line 23: kill: (204) - No such process
Sending all processes the TERM signal...
```

```
Sending all processes the KILL signal...
Unmounting remote filesystems...
Stopping portmap daemon...
Deactivating swap...
Unmounting local filesystems...
reboot: System halted
```

启动过程中可以看到许多系统信息。

登陆系统：

```
SoCFPGA login: root
```

```
root@SoCFPGA:~#  who i am
```

```
 BusyBox v1.20.2 (2013-09-27 23:27:54 CDT) multi-call binary.
```

关机：

```
root@SoCFPGA:~# poweroff
```

详情请点击：

http://bbs.eeworld.com.cn/thread-455066-1-1.html



2. 让 SoC 飞一会儿

2.1 Altera SoC 体验之 HPSFPGA

一、感想：我也是开心 的快要哭了，整个人感觉都不好了。整体的对这个 SoC 的印象就是 ARM 和 FPGA 关于 AMBA4.0 的一段恋情。。。开发起来也是复杂的不行，首先你得会 ARM 和 FPGA，还得会用 AMBA。作为一个基本不知道 FPGA 是什么的东东的我来说确实有点难度。

二、CycloneV SE 简介

Cyclone Series Introduction

	Cyclone FPGA	Cyclone II FPGA	Cyclone III FPGA	Cyclone IV FPGA	Cyclone V FPGA
Year introduced	2002	2004	2007	2009	2011
Process technology	130 nm	90 nm	65 nm	60 nm	28 nm
Recommended for new designs	No	Yes	Yes	Yes	Yes

图 3-3：Cyclone 简介

Cyclone V 较 CycloneIV 的提升

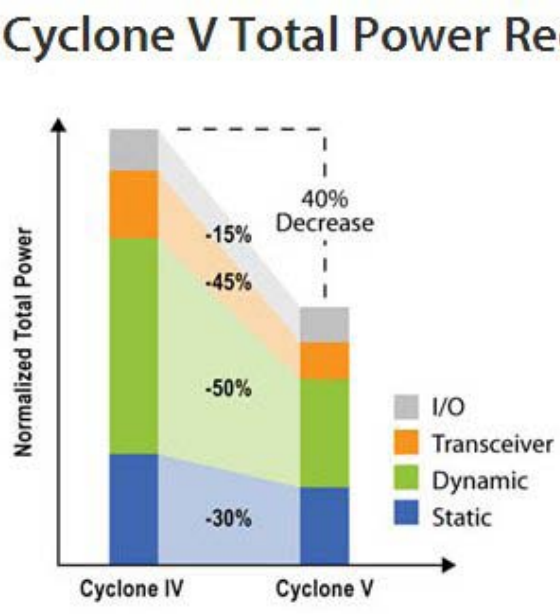


图 3-4：CycloneIV 和 V 的区别

Cyclone V 下的各种的种类也是各种的各种多。

Cyclone V Variants

Features	Lowest cost and power	3.125G transceivers	6.144G transceivers
	Optimized for lowest system cost and power for a wide spectrum of general logic and DSP applications	Optimized for lowest cost and power for 614 Mbps to 3.125 Gbps transceiver applications	FPGA industry's lowest cost and power for 6.144 Gbps transceiver applications(1)
FPGA	E Variant	GX Variant	GT Variant
Integrated ARM Cortex™-A9 MPCore™ Processor System	SE Variant	SX Variant	ST Variant

Notes:

Cyclone V GT devices support 6.144 Gbps CPRI protocol.

图 3-5：Cyclone V 种类

所以来讲个命名规则吧，因为要围绕 DE1-SoC 的板载芯片来讲，所以看看 CycloneV（SE、SX、ST）命名规则如下：

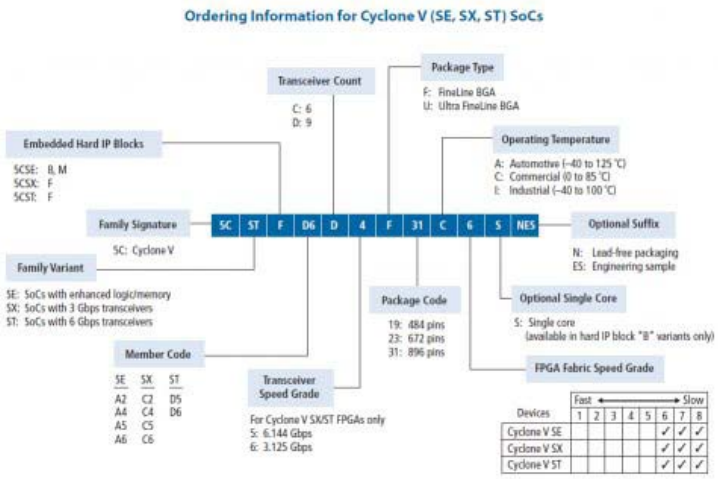


图 3-6：SE、SX、ST

DE1-SoC 板载一颗型号为：5CSEMA5F31C6N 的芯片，关于这个芯片的性能与功能，请参照图 3-3 自行脑补，或者参看下面的图 3-7、8、9

	Product Line	Cyclone V SE SoCs[1]			
		5CSEA2	5CSEA4	5CSEA5	5CSEA6
Resources	LEs (K)	25	40	85	110
	ALMs	9,434	15,094	32,075	41,509
	Registers	37,736	60,376	128,300	166,036
	M10K memory blocks	140	270	397	557
	M10K memory (Kb)	1,400	2,700	3,970	5,570
	MLAB memory (Kb)	138	231	480	621
	Variable-precision DSP blocks	36	84	87	112
	18 x 18 multipliers	72	168	174	224
	Processor cores (ARM Cortex-A9)	Single or dual	Single or dual	Single or dual	Single or dual
	Maximum CPU clock frequency (MHz)	925	925	925	925
	Global clock networks	16	16	16	16
	PLLs[2] (FPGA)	5	5	6	6
	PLLs (HPS)	3	3	3	3

图 3-7：5CESA5 功能

Device	Part Number	Package Type	Pins	Silicon	Package Description
5CSEMA5	5CSEMA5U23	UBGA	672	Production	672-Pin UBGA - Wire Bond - A:1.85
	5CSEMA5F31	FBGA	896	Production	896-Pin FBGA - Wire Bond - A:2.00

图 3-8：5CESA5F31 封装

Device	Package Type	Pins	θ_{JA} (° C/W) Still Air	θ_{JA} (° C/W) 100 ft./min.	θ_{JA} (° C/W) 200 ft./min.	θ_{JA} (° C/W) 400 ft./min.	θ_{JC} (° C/W)	θ_{JB} (° C/W)
5CSEMA5	UBGA	672	16.0	14.8	13.0	11.5	2.6	6.1
	FBGA	896	13.7	12.5	10.7	9.4	2.3	5.4

JEDEC JESD51-1 Thermal

图 3-9：5CESA5F31 JESD51-1

（上面这个测试，细节我也不懂，只不过好高大上，就贴在上了）

三、HPS---FPGA

DE1-SoC 开发板入门资料还是不错的，不过都是英文，四六级都没过的我，开着这些资料开心的我也是快要了。

看看下面这个图，可能对 Cyclone 的开发工具和 CycloneV 的内部结构有点了解，如下：

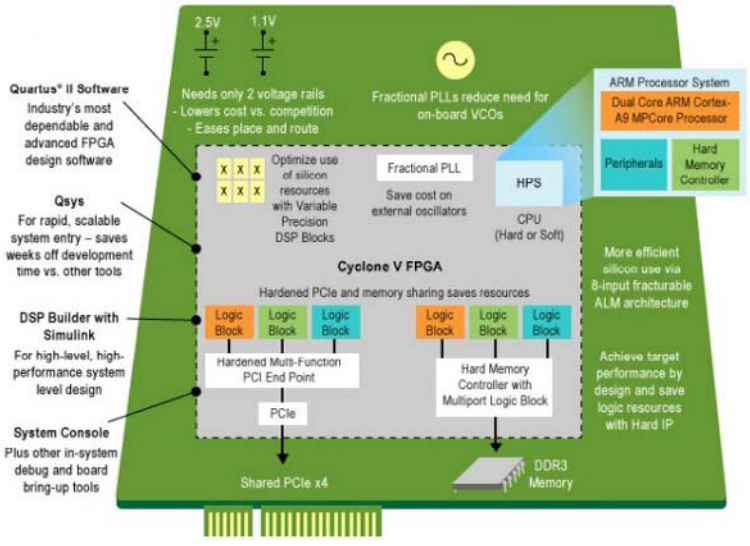


图 3-A：Cyclone V

首先看一下面的几个单词，其实用过 Cortex-M 的单片记得同学，除了 AXI 对下面的应该很熟悉的

```
AMBA（AdvancedMicrocontroller Bus Architecture）
AHB（theAdvanced High-performances Bus）
ASB（theAdvanced System Bus）
APB（theAdvanceed Peripheral Bus）
```

```
AXI（AdvancedeXtensible Interface）
AMBA® 4   :    AXI4™, AXI4-Lite™, or AXI4-Stream™ Protocol.
```

AMBA 是 ARM 公司设计的通信总线。AXI4.0 是对 3.0 的升级，可以通过它来进行 ARM 和 FPGA 的通信。具体的就参考 ARM 官网的用户手册吧。

ARM 和 FPGA 来直观的看一看，其实就是下面这个样子的

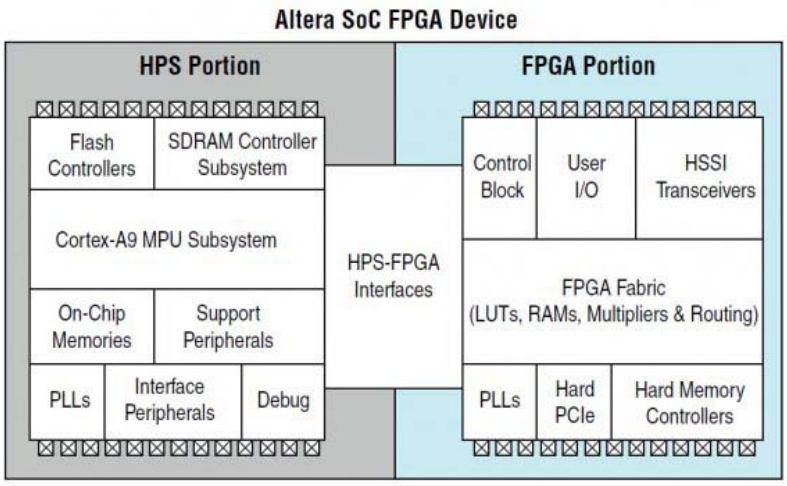


图 3-B：SoC

两个独立的东东，做到了一个芯片上，通过总线互联。

其实是这个样子的

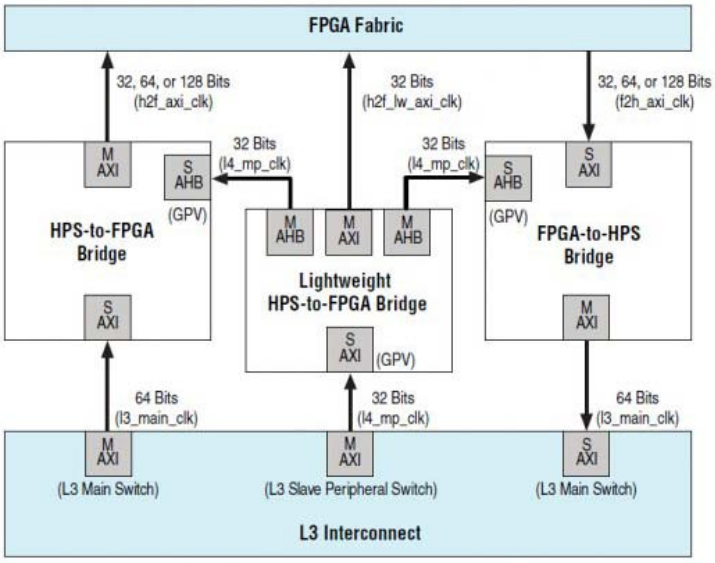


图 3-C：总线

M：master　　S： slave

FPGA-to-HPS bridge

HPS-to-FPGA bridge

Lightweight HPS-to-FPGA bridge

DE1-SoC 提供了 QuartusII 的一个基本的项目，DE1-SoC 板子的资源已经设置好，可以在这个基础上做一些开发。

例程是做了一个简单的流水灯的程序。

只需要做点小小的改动就可以完成对 HPS 和 FPGA 这块相应的简单的开发。在 Qsys 中添加好 PIO 模块之后，设置好系统总线。编译好项目之后，通过 JTAG 配置到 FPGA 中。

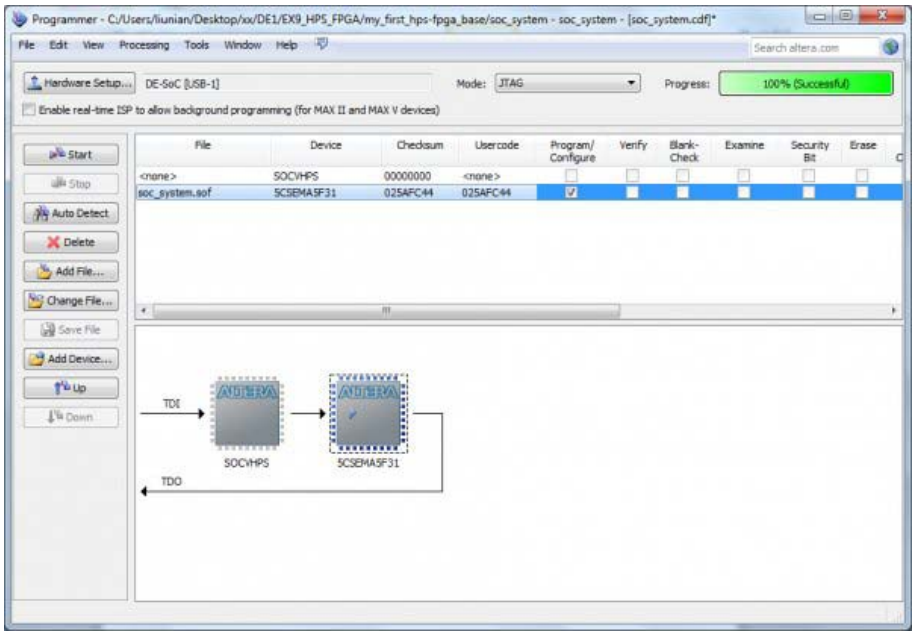


图 3-D：JTAG 调试

接着，将 IO 口地址虚拟映射到虚拟地址之后，通过“/dev/mem”就可以操作了，在 HPS 上执行的 C 语言代码如下：

```
<blockquote>#include <stdio.h>
分割线
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hwlib.h"
#include "SoCal/SoCal.h"
#include "SoCal/hps.h"
#include "SoCal/alt_gpio.h"
#include "hps_0.h"

#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

int main() {

        void *virtual_base;
```

```
        int fd;
        int loop_count;
        int led_direction;
        int led_mask;
        void *h2p_lw_led_addr;

        // map the address space for the LED registers into user space so we can
interact with them.
        // we'll actually map in the entire CSR span of the HPS since we want to
access various registers within that span

        if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
                printf( "ERROR: could not open \"/dev/mem\"...\n" );
                return( 1 );
        }

        virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_
SHARED, fd, HW_REGS_BASE );

        if( virtual_base == MAP_FAILED ) {
                printf( "ERROR: mmap() failed...\n" );
                close( fd );
                return( 1 );
        }

h2p_lw_led_addr=virtual_base + ( ( unsigned long  )( ALT_LWFPGASLVS_OFST + PIO_LED_BASE
) & ( unsigned long)( HW_REGS_MASK ) );

        // toggle the LEDs a bit
        loop_count = 0;
        led_mask = 0x01;
        led_direction = 0; // 0: left to right direction
        printf(" HelloWorld! \n");
        while( loop_count < 60 ) {

                // control led
                *(uint32_t *)h2p_lw_led_addr = ~led_mask;

                // wait 100ms
                usleep( 400*1000 );

                // update led mask
                if (led_direction == 0){
                        led_mask <<= 1;
                        if (led_mask == (0x01 << (PIO_LED_DATA_WIDTH-1)))
                                led_direction = 1;
                }else{
                        led_mask >>= 1;
                        if (led_mask == 0x01){
```

```
                                led_direction = 0;
                                loop_count+;
                        }
                }

        } // while

        // clean up or memory mapping and exit
        if( munmap( virtual_base, HW_REGS_SPAN ) != 0 ) {
                printf( "ERROR: munmap() failed...\n" );
                close( fd );
               return( 1 );
       }

        close( fd );

        return( 0 );
}
```

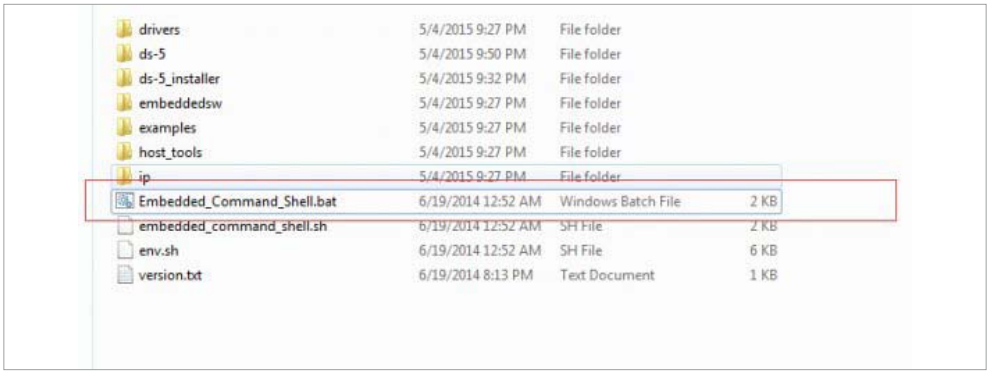


图 3-E：编译环境

感觉上面那个东东就是一个类似的 Cygwin 完美执行 linux 指令，开心的我也是不行。。

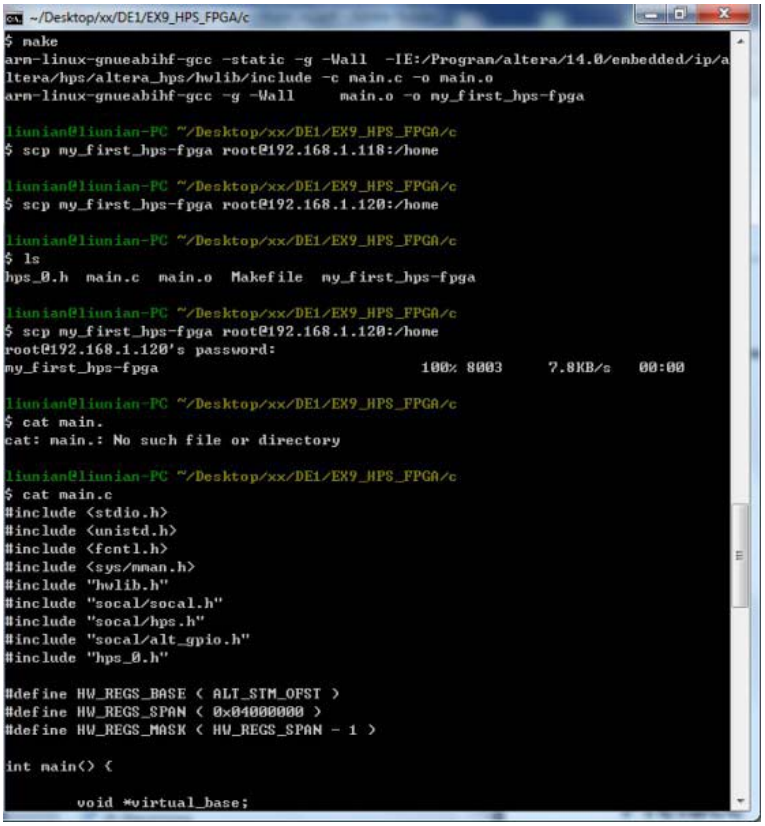


图 3-F：调试

编译之后将，可执行文件放到开发就可以发布运行了。

至此，也是结束了，开心的我也是不行。

总算有点明白了 ARM 与 FPGA 之间的这段 “奸情”  。

详情请点击：

http://bbs.eeworld.com.cn/thread-462426-1-1.html

2.2 Altera SoC 体验之 linux+QT

一、秀一个 Ubuntu

First，去网下载镜像文件，用 win32DiskImage，将镜像写入到内存卡中。

Linux BSP (Board Support Package): MicroSD Card Image

Title	Linux Kernel	Min. microSD Capacity	Size(KB)	Date Added	Download
Linux Console	3.12	4GB	66495	2014-01-14	
Linux Console with framebuffer	3.12	4GB	328524	2014-03-24	
Linux LXDE Desktop	3.12	8GB	1369526	2014-03-21	
Linux Ubuntu Desktop	3.12	8GB	1136075	2014-02-11	

图 1：OS 镜像各种

启动方式如下：

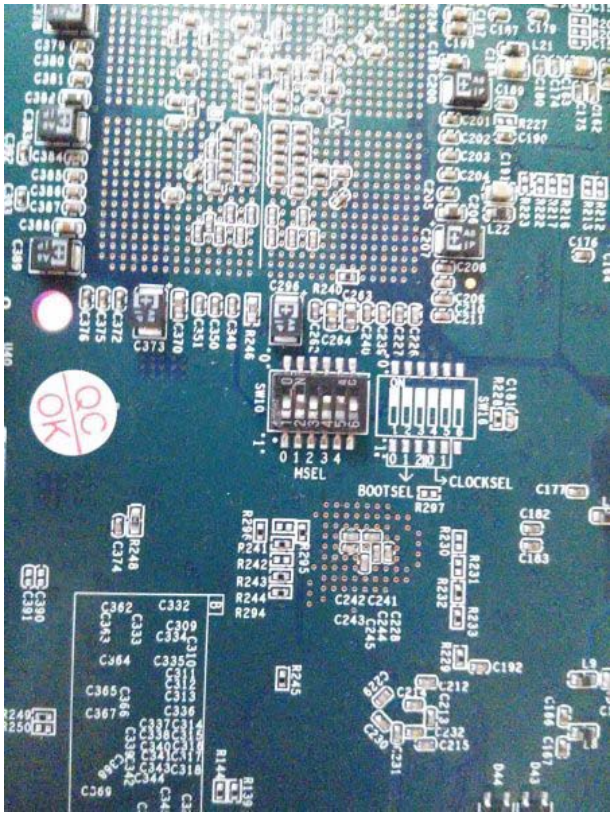


图 1—0：启动方式

插卡 ,插键鼠 ,VGA 接显示器 ,上电！！！！！！ 一个 Ubuntu 的小电脑 ,完美的呈现在你的眼前了。

Ubuntu 版本信息如下：



图 1-1：Ubuntu 版本信息

二、远程登陆

倘若没有 VGA 的显示器，可以使用远程登陆。

在 Ubuntu 中安装 xrp，执行代码如下：

```
$ sudo su
# apt-get install update
# apt-get install xrdp
```

如此并安装了 xrdp 包文件了。

启动 vncserver，如下示，设置密码即可：

```
# vnserver
```

在 windows 中搜素远程登陆的程序，如下图所示：

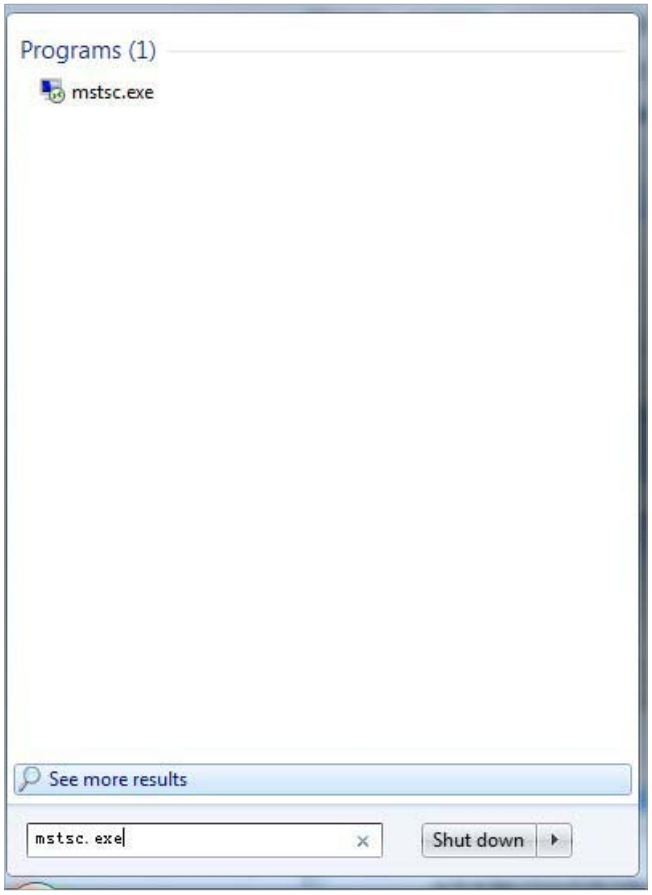


图 2：远程登陆

打开如下：输目标板（此处时 DE1-So）地址：

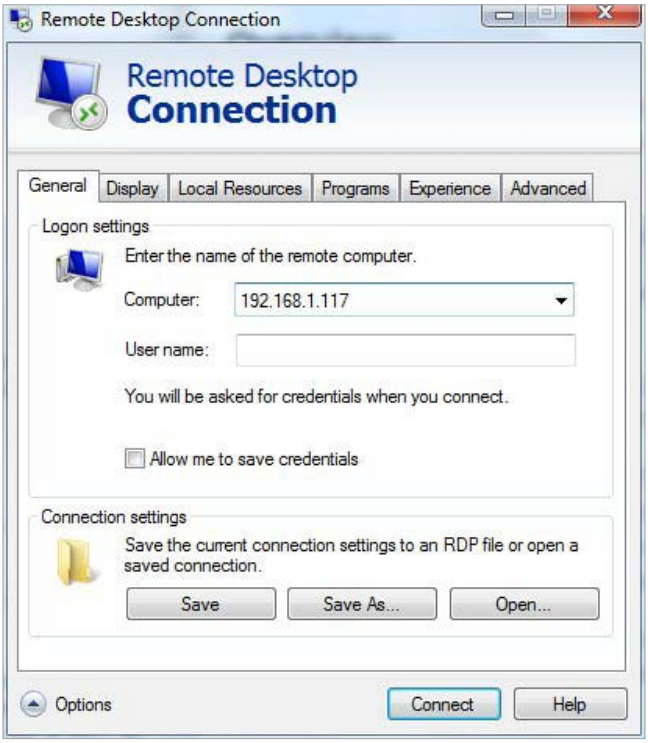


图 3：I 地址输入

输入目标板登陆名和密码

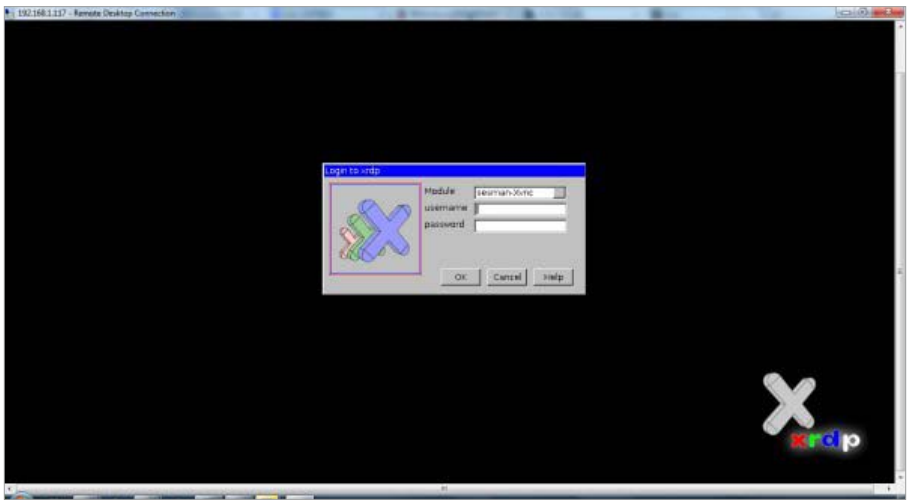


图 4: 用名和密码输入

登陆效果如下：



图 5：效果图

总体来说执行还行，有点略卡顿。。。但是 DE1-SoC 在 arm 这块性能并不是很好，不如树莓派 .....

打开网页，多点几下，直接闪退 .....

输入下面命令查看系统信息：

```
# gnome-system-monitor
```

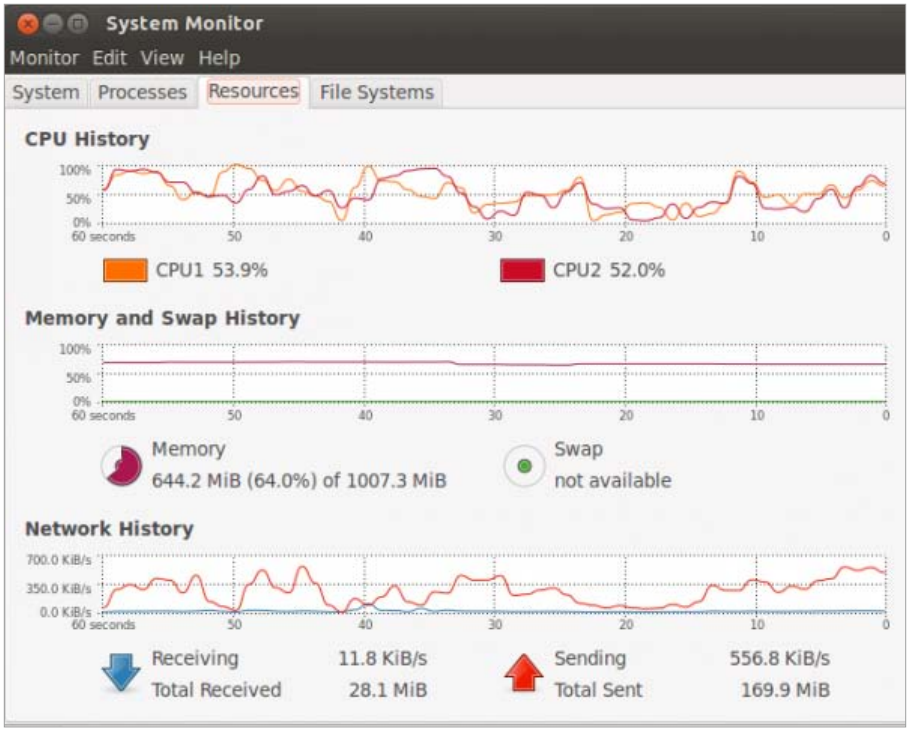


图 6：系统信息

总体来说，性能并不是很好，SoC 毕竟重点不在这里。

三、秀一个 linux+QT

安装另一镜像 Linux Console with framebuffer

启动如下：



图 7-1：登陆界面



图 7-2：VGA 显示

打开 QT 程序，下面是各个功能的不同的页面 ……



图 7-3：LED 控制



图 7-4：数码管



图 7-5：按键



图 7-6：三轴

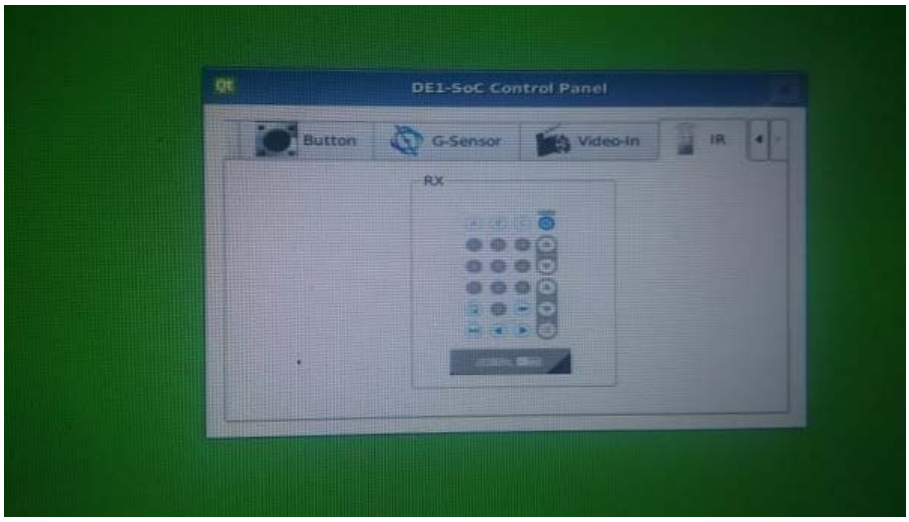


图 7-8：红外接收

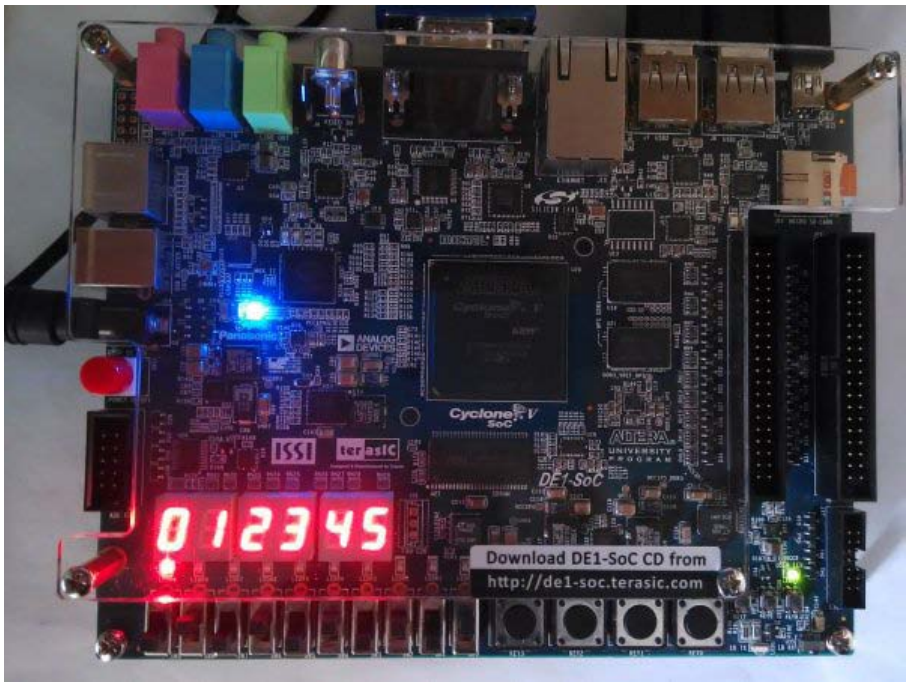


图 7-9：实物演示

这个系统还是蛮流畅的 ……

怎么编辑界面都是 html……

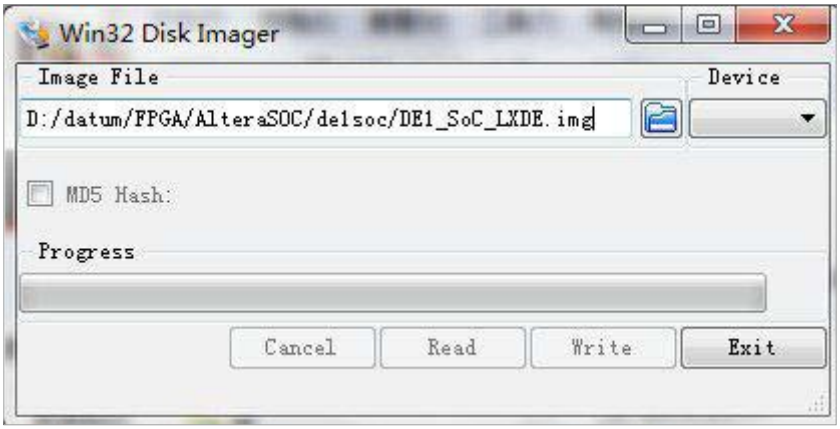
详情请点击：

http://bbs.eeworld.com.cn/thread-461650-1-1.html

2.3 DE1-SoC 之 LXDE 桌面系统

折腾了几天板子，受够了在终端敲指令，一点都不高端大气上档次 ，后面有几天很忙 ，趁着夜还没深多鼓捣鼓捣 ,看着别人的板子一个个都跑起来桌面 ,DE1-SoC 表示不服□ 虽然是几个板卡中便宜的，一点都不能小看它。以前使用过 zynq，所以上手 Altera 的 SoC 不是很困难，主是上次使用 SoC 是一年以前了 ，本来对 ARM 不是很熟，玩 zynq 的时候没有深入研究 ARM 有点后，想好好利用这次体验机会从基础上提升 ARM 的水平， 按照手册步来可能熟能生巧，但只有深入了解才能再以后的设中游刃有 。

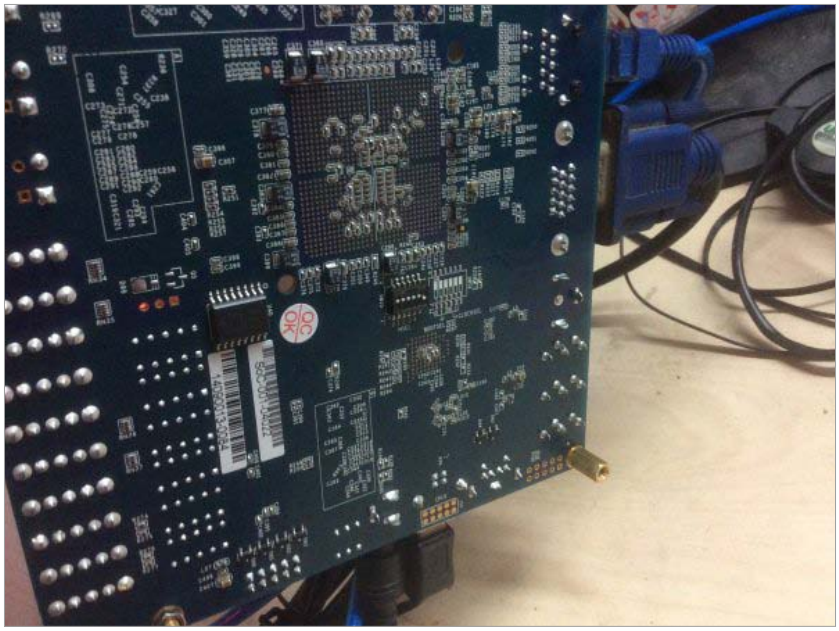
初步打算花一个月的时间好好巩固 linx ARM 的知识， 熟悉 cylone V SoC 体系结构与开发，为今后不到 3 个月的项目开发做准备。闲话少说，有了上次 HPSFPGA 运行 yocto 的经验，这次晚上只花了 10 分钟左右就给 DE1-SoC 装上了一个逼格够高的 linux 桌面系统。第一部分 制作可供开发板开机的 SD 卡下载官网 LXDE 的 img 和系统光盘　打开官网下载的系统光盘 tools 中的 Win32DiskImager 将读卡器插在 pc 上



点击 write 弹出 SD 卡

第二步

将开发板背后的 DIP 调整至如图所示的位置



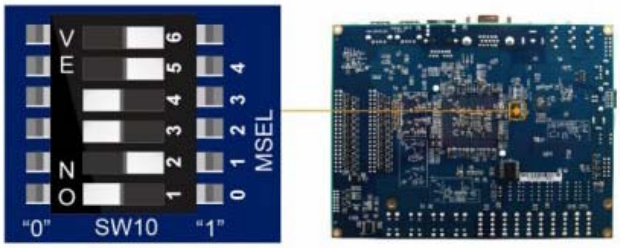


图 1-18 FPGA MSEL 开关

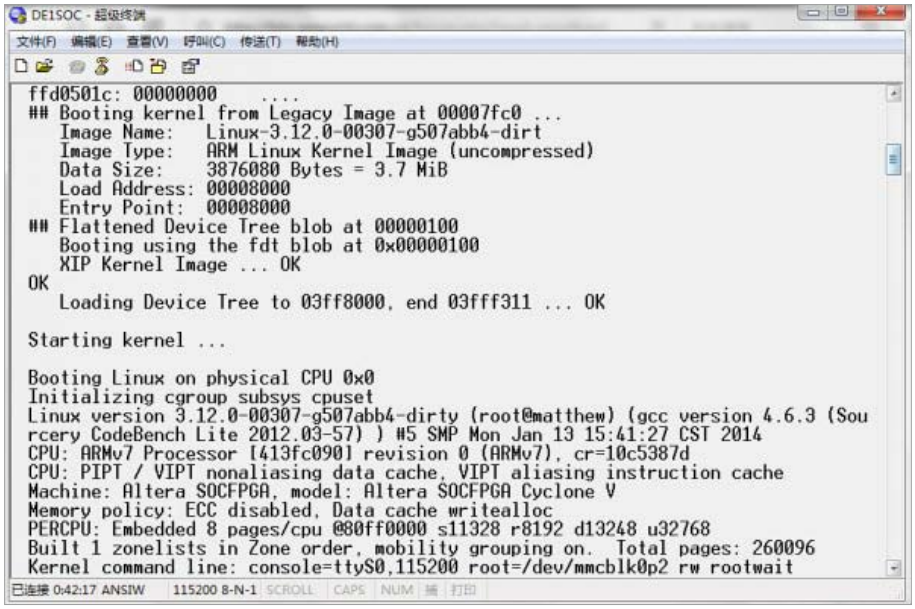
此开关用途为设定 FPGA 的配置来源。常用的设定如下表 ：

MSEL[4:0]	设定模式	描述
10010	AS	FPGA configured from EPCQ (default)
01010	FPPx32	FPGA configured from HPS software: Linux
00000	FPPx16	FPGA configured from HPS software: U-Boot, with image stored on the SD card, like LXDE Desktop or console Linux with frame buffer edition.

选择第三种模式

将 sd 卡插入开发板　连上串口 打开串口助手和开发板电源

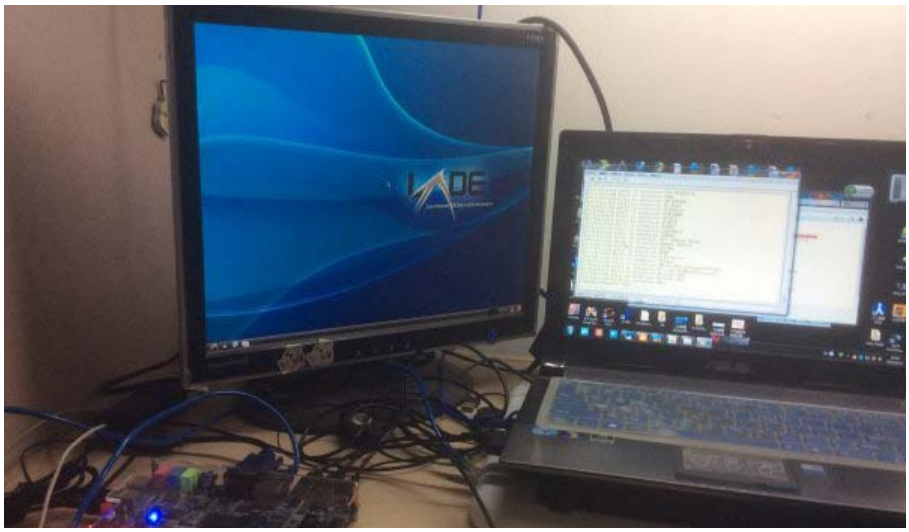
系统开始启动

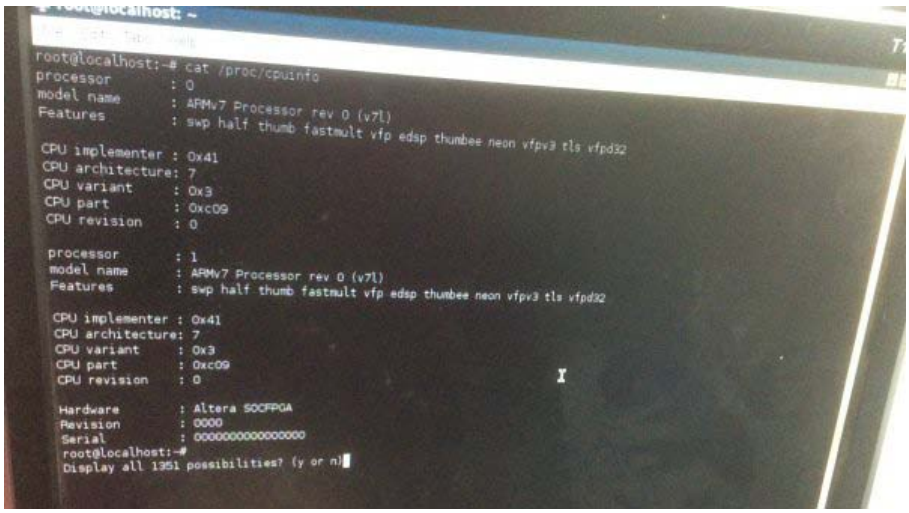


看看 VGA　2 只可爱的小企鹅出现　恕我没拍到图

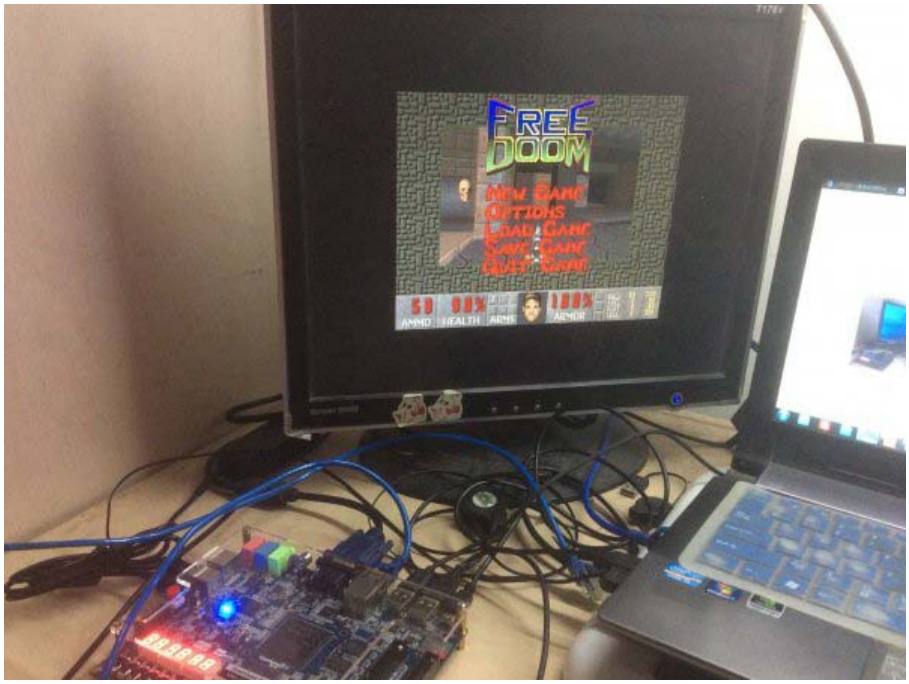


成功进入桌面系统！！！！！！！！





附带的游戏很带感噢 视频奉上



详情请查看：

http://bbs.eeworld.com.cn/thread-454429-1-1.html



2.4 FPGA 与 ARM 的窃窃私语

喜大普奔，公司要评估用 SoC 做产品，我就自然而然的被安排了学习和评估的工作，于是，每天的工作就是开始研究 SoC 了。其实，只要能静下心来学习，一切都还是能够弄出来的。

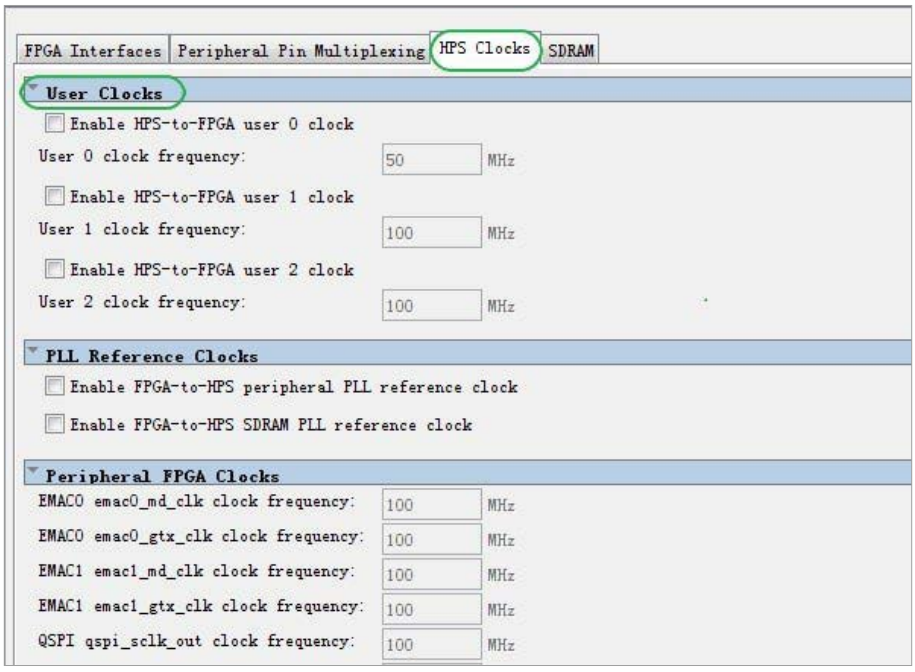
以前像个无头苍蝇一样到处乱撞，搞了两三个月没把 FPGA 和 arm 给联通起来，尽然只是因为教程用的 Quartus II13.1 软件和我自己电脑上装的 Quartus II14.1 软件在生成 dts 时调用的一个文件不一样导致的。当然，这个只是外在原因，真正内在的原因还是因为内在对 linux 的恐惧，因为是零 linux 基础，所以全部 linux 下命令都是第一次接触，于是各种错误各种不通，最后就放弃了。但是，心里总是不愿就这么放弃了，总想学会这门新技术。于是前两天又花钱买了周立功的 ARM9 开发板，然后跟着他提供的教程慢慢学习。终于是能够简单的进行一些 linux 下的命令操作了，昨天再次上手 DE1—SoC，终于成功的生成了 preloader 和 DTS，并通过 HPS 控制了 FPGA 端的 LED。

相信多参加此次活动或者自学 SoC 的筒子也都有过我以上这段经历这里，我愿意把我遇到的问题和解决方法提出来，希望那些还卡在这些问题的地方的筒子能够快速走出困境。

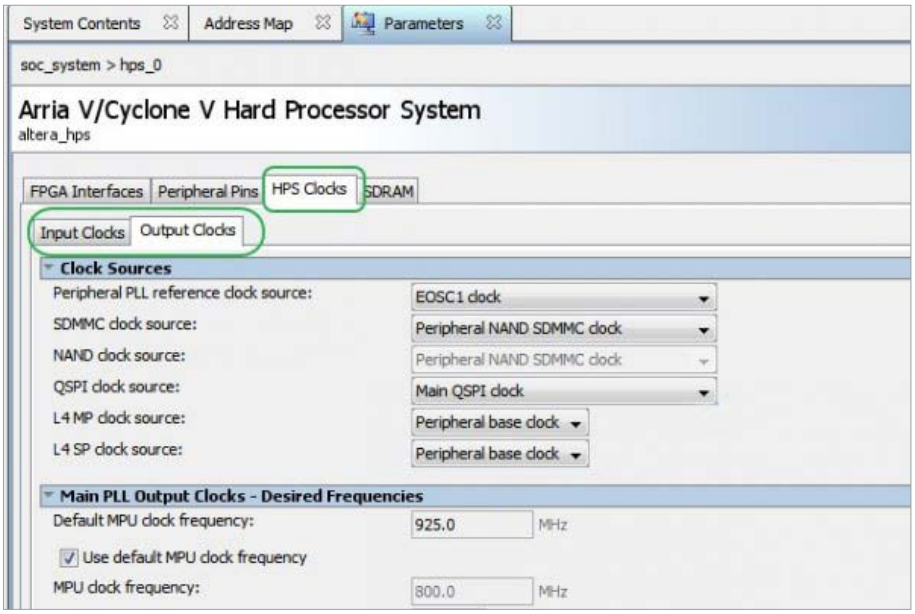
我此次学习是跟着友晶提供的培训教材一步一步来进行的，该培训教材中，使用的是 Quartus II13.1 的软件，而我在使用的时候，直接安装的 Quartus II 14.1。而也正是这样一个选择，让我被坑了好几天，最终放弃沉寂了好久，直到昨天，在网友“微末凡尘”的指导下，才执行成功。

首先，在教材的第 49 页，在 hps clock 选项卡上，需要进行一下更改，具体我也是看了骏龙科技的一篇文章《在 SoCEDS 环境下编译和更新 preloader 和 uboot 程序的方法》才知道的，原文如下：

相对于 13.0 主要变化是有了 HPS Clocks 窗口分为了 Input Clocks 和 Output Clocks ！对比如下图：



Quartus 13.0 HPS Clocks 窗口



Quartus 14.0 HPS Clocks 窗口

14.0 软件中 Cyclone V 器件 MPU 默认时钟为 925Mhz, 而 13.1 版为 800M。这里编不改变 MPU 的 Clock, 所以去掉 usedefault MPU clock frequency 选项！不用改其他的选项，Clock 即为 800MHz。重新 generate 关闭就好了。使用 Quarut 编译编译完成后会有 hps_isw_handoff 中的文件会更新，这些文件时连接 HPS 硬件配置与 preloaderr 软 件接口文件。

这里文中说的 14.0，但是 14.0 和 14.1 在这一点上没有区别，因此，直接按照文中的内容操作即可。

第 53 页最后一行，说要求将 pio_0 重命名为 led_pio，这里一定要注意并保持一致，否则生成 dts 时会报错。同理，button_pio 也是一样，名字最好不要错。个人感觉，初学，尽量不要标新立异，尤其是对环境不熟的，否则都不知道死在哪里。

第 83 页最后一句话，我直接按照文档中的命令内容敲进去，会报错，提示我少了 -hv 参数，如下图所示：

```
xiaomeige@xiaomeige-PC /cygdrive/c/Altera_TRAN/DE1_SoC_GHRD/software/spl_bsp
$ mkpimage -o preloader_with_header.img u-boot-spl.bin
Option "-hv (--header-version)" is required
```

这里我也不懂是怎么回事，既然说少了个参数，就加个参数呗，于是我将命令内容改为：

mkpimage –hv 0 –opreloader_with_header.img u-boot-spl.bin

然后就能成功了。

第 85 页，生成 DTS 的时候，这个地方是变化最大的一个地方，我就是因为这个地方，才被卡了很久。这页的最后一段，说生成 dts 需要的四个文件为

SoC_system.sopcinfo
hps_clock_info.xml
SoC_system_board_info.xml
Makefile

那么，如果我们直接使用这三个文件，用原来的 Makefile 文件来生成 dts 和 dtb，都会报错，dts 能够生成，但是有错误，dtb 直接无法生成。原因就在于，在 Quartus II14.1 的软件中，第二个使用的不再是 hps_clock_info.xml 文件，而是 hs_common_board_info.xml。这个文件我们可以在 SoCEDS 软件安装目录下的 examples\hardware\cv_SoC_devkit_ghrd 文件夹下找到，我的电脑中的绝对位置为

D:\Altera\14.1\embedded\examples\hardware\cv_SoC_devkit_ghrd

我们将其中的两个 xml 文件 hps_common_board_info.xml、SoC_system_board_info.xml 和一个 Makefile 复制出来，粘贴到我们自己的工程目录下，并将原来的三个文件删除即可。再次 make dts，就能够成功了，当然这个过程比较长，需要 2 到 5 分钟。然后，待 dts 生成完成后，再次 make dtb，就很快，也能成功了。

注意，如果按照第 86 页最后一段来自己敲命令进行生成 dts，则需要将命令中对应的

```
sopc2dts --input soc_system.sopcinfo --output soc_system.dts --board
soc_system_board_info.xml --board hps_clock_info.xml --bridge-removal
all
```

hps_clock_info.xml 替换为 hps_common_board_info.xml。为了简便推荐大家直接使用“makedts”命令来操作。后面生成 dtb 的也是一样，直接使用“make dtb”命令就行了，详见教材 88 页最后一段。

自此，后面的就没有其他问题了，对着该教材一步一步的进行，最终都能运行出正确的结果。只是，在做 de1_SoC_sw_lab3 中的实验时，原本提供的 main.c 文件里面使用的 FPGA 端的 led 的名字叫 pio_led，而我们的 Qsys 系统中命名为 led_pio，因此直接使用该 c 文件生成可执行文件时会报错，需要将其中的第 40 行的“PIO_LED_BASE”改为“LED_PIO_BASE”。第 59 行的“PIO_LED_DATA_WIDTH”改为“LED_PIO_DATA_WIDTH”。然后就能生成成功了。

执行实验 3 生成的可执行文件，能够看到开发板上 FPGA 端的 10 个红色 LED 灯流水 60 次，然后停止。

好了，就写到这里，文字不够详细，若有疏漏或错误，欢迎各位指出。具体效果，就不上图啦。

详情请点击：
http://bbs.eeworld.com.cn/thread-461330-1-1.html

2.5 Altera SoC 之地址映射

HPS 如何对 FPGA 外设进行操作？

hardware：在 Qsys 中将外设连接到 AXI bridge 上 software：映射外设物理地址到到应用程序可以操作的虚拟地址，应用程序通过得到的虚拟地址入口控制外设。

为什么要进行地址映射？

内存映射就是讲内核空间的一部分区域映射到用户空间，用户对这段内存空间的修改可以反映到内核空间。可以将内核空间的一段地址映射到多个进程，以实现线程间的内存通信。系统调用 mmap() 就是进行地址映射。mmap 是将一个文件（linux 下设备也被看做是文件）或其他对象映射进内存。munmap 执行相反的操作，删除特定地址区域的对象映射。采用共享内存进行通信的优点是效率高，直接读写内存不需要进行数据拷贝。

通常使用 mmap 有三种情况，1、提高 I/O 效率　2、匿名内存映射　3、共享内存进程通信。

mmap 用于内存映射的一种方式是打开或创建一个文件，然后调用 mmap().（另一种方式如下：进程 A 和进程 B 都将该页映射到自己的地址空间，当进程 A 第一次访问该页中的数据时产生一个缺页中断，内核此时读入这一页到内存并更新页表使之指向它，当进程 B 访问同一页发生缺页中断时，该页已经在内存中，内核只需要将进程 B 的页表登记项指向此页即可）

mmap 用法

```
#include <sys/mman.h>
void *mmap(void *addr,  size_t length,  int prot,  int flags,  int fd,  off_t offset);
int munmap(void *addr, size_t  length);
```

---

start：映射区的开始地址。
　　length：映射区的长度。
　　prot：期望的内存保护标志，不能与文件的打开模式冲突。是以下的某个值，可以通过 or 运算合理地组合在一起
　　PROT_EXEC //页内容可以被执行
　　PROT_READ //页内容可以被读取
　　PROT_WRITE //页可以被写入
　　PROT_NONE //页不可访问
　　flags：指定映射对象的类型，映射选项和映射页是否可以共享。它的值可以是一个或者多个以下位的组合体
　　MAP_FIXED //使用指定的映射起始地址，如果由 start 和 len 参数指定的内存区重叠于现存的映射空间，重叠部分将会被丢弃。如果指定的起始地址不可用，操作将会失败。并且起始地址必须落在页的边界上。
　　MAP_SHARED //与其它所有映射这个对象的进程共享映射空间。对共享区的写入，相当于输出到文件。直到 msync() 或者 munmap() 被调用，文件实际上不会被更新。
　　MAP_PRIVATE //建立一个写入时拷贝的私有映射。内存区域的写入不会影响到原文件。这个标志和以上标志是互斥的，只能使用其中一个。
　　MAP_DENYWRITE //这个标志被忽略。
　　MAP_EXECUTABLE //同上
　　MAP_NORESERVE //不要为这个映射保留交换空间。当交换空间被保留，对映射区修改的可能会得到保证。当交换空间不被保留，同时内存不足，对映射区的修改会引起段违例信号。

```
    MAP_LOCKED // 锁定映射区的页面，从而防止页面被交换出内存。
    MAP_GROWSDOWN // 用于堆栈，告诉内核 VM 系统，映射区可以向下扩展。
    MAP_ANONYMOUS // 匿名映射，映射区不与任何文件关联。
    MAP_ANON //MAP_ANONYMOUS 的别称，不再被使用。
  MAP_FILE // 兼容标志，被忽略。
    MAP_32BIT // 将映射区放在进程地址空间的低 2GB，MAP_FIXED 指定时会被忽略。当前这个标志只
在 x86-64 平台上得到支持。
    MAP_POPULATE //为文件映射通过预读的方式准备好页表。随后对映射区的访问不会被页违例阻塞。
    MAP_NONBLOCK // 仅和 MAP_POPULATE 一起使用时才有意义。不执行预读，只为已存在于内存中的
页面建立页表入口。
    fd: 有效的文件描述词。如果 MAP_ANONYMOUS 被设定，为了兼容问题，其值应为 -1。
    offset: 被映射对象内容的起点。
    返回说明:
    成功执行时，mmap() 返回被映射区的指针，munmap() 返回 0。失败时，mmap() 返回 MAP_FAILED[ 其
值为 (void *)-1]，munmap 返回 -1。errno 被设为以下的某个值
    EACCES: 访问出错
    EAGAIN: 文件已被锁定，或者太多的内存已被锁定
    EBADF: fd 不是有效的文件描述词
    EINVAL: 一个或者多个参数无效
    ENFILE: 已达到系统对打开文件的限制
    ENODEV: 指定文件所在的文件系统不支持内存映射
    ENOMEM: 内存不足，或者进程已超出最大内存映射数量
    EPERM: 权能不足，操作不允许
    ETXTBSY: 已写的方式打开文件，同时指定 MAP_DENYWRITE 标志
    SIGSEGV: 试着向只读区写入
    SIGBUS: 试着访问不属于进程的内存区
```

下面通过 HPS 控制 FPGA leds 实例来了解 HPS 如何进行 FPGA 周边设备的管理。

通过 open 和 mmap 映射到相应的虚拟基地址

```
if( ( fd(fd =open( "/d / " ( O RDWR | O SYNC ) ) )"/dev/mem" , (O_ RDWR|O_ SYNC))) == -1
) {){
printf( "ERROR: could not open \"/dev/mem\"...\n" );
return( 1 );1);
}
virtual_base =mmap( NULL, HW_REGS_SPAN, ( PROT_READ |
PROT WRITE_ ), MAP SHARED_, , fd, HW REGS BASE_ _ );
```

LWAXI 总线（light weight AXI）相对于其虚拟基地址的偏移

（ALT_LWFPGASLVS_OFST&(unsigned long)(HW_REGS_MASK)）

```
FPGA 外设相对于 LWAXI 的地址（PIO_LED_BASE）
h2_lw_led_addr= virtual_base + ( ( unsigned long)(
ALT_LWFPGASLVS_OFST+PIO_LED_BASE) & ( unsigned
long)(HW_REGS_MASK) )
```

可见使用地址映射来实现 HPS 对 FPGA 周边设备的管理非常方便且步骤简单、灵活性高，同时效率又非常高。

详情请点击：

http://bbs.eeworld.com.cn/thread-458492-1-1.html

3. 设计与应用

3.1 DE1-SoC—人脸检测

真不易呀，通过两天的挑灯夜战，走了好多的弯路，总算把 SimpleCV 跑出来了。官方的 Ubuntu 镜像烧写之后，主分区只有 2G 左右，系统占了 1.5G，只剩 500M 左右的空间不能装太多的软件。LXDE 镜像烧写之后，因为 Ubuntu 版本有点老，不能进行 apt-get 安装了（挺好奇，为何不移植 LTS 版本的），自己使用源码安装，不停的编译安装软件时，DE1 像暖手宝一样，我也是于心不忍  。

其实主要原因是自己太小白了。接下来这段美好的时光里，先暂且让他当做一个树莓派来鼓捣一下，委屈一小下 DE1-SoC 了。下面写下过程吧，给自己作笔记用，日后方便查找  。

一、Ubuntu

1. 镜像烧写

使用的是一枚 16G 的 SD 卡，用 win32DiskImager 将官方的镜像（DownloadDE1-SoC CD fromhttp://de1-SoC.terasic.com）烧写到内存卡中。

配置

MSEL[4:0]=00101

插上网线、插卡、上电、开机、不出意外 Ubuntu 就开机了。

2. 准备工作

（1）联网

执行

```
# ifconfig
```

发现并没有分配 IP 地址，此时并不能上网。需要配置，如下：

```
# gedit /etc/network/interfaces
```

添加如下代码：

```
auto eth0
iface eth0 inet dhcp
```

保存，执行

```
# /etc/init.d/networking restart
```

不出意外，此时执行

```
$ ifconfig
```

即可看到 IP 分配。

（2）gparted

```
#apt-get install gparted
```

安装这个工具，稍后分区会用。

3. 分区

（1）分区查看

执行

```
# gparted
```

打开 gpated 工具，如下所示（本文中使用的是 16G S card）：

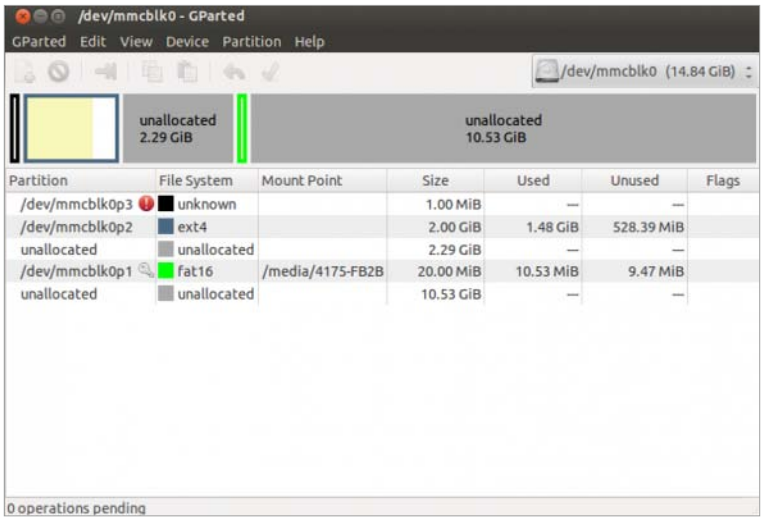


图 2-1：原始分区

可以调整分区，将右手边未分配的分区合并到左边。如果重新配置一个分区的时候，首先要 umont。

根目录分区只有个 2G，不能 umont，所以就不能使用 gparted 工具了，需要使用 fdisk 工具，打开终端窗口。

```
#df -h
```

如下图所示：

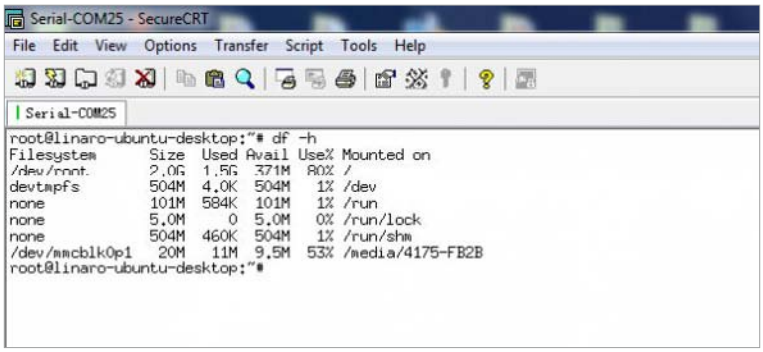


图 2-2: 磁盘空间

（2）fdisk

如下图所示，使用 fdisk 进行分区，没有输入参数的直接点选回车选择默认参数。

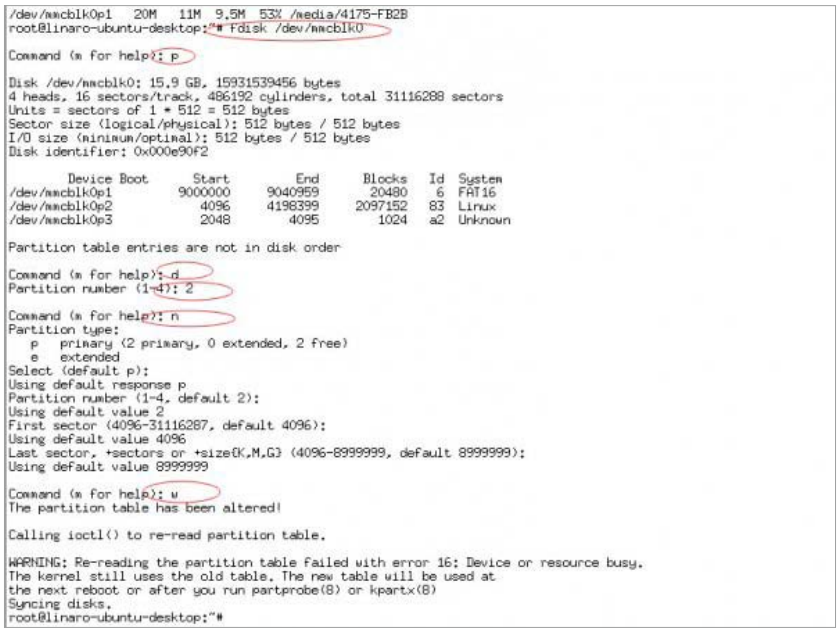


图 2-3：fdisk 分区操作

之后，重启 DE1-SoC。

执行

```
# df -h
```

并没有增加空间，还缺一条执行。执行如下命令

```
#  resize2fs /dev/mmcblk0p2
```

之后就可以看到根目录空增加，如下图所示：

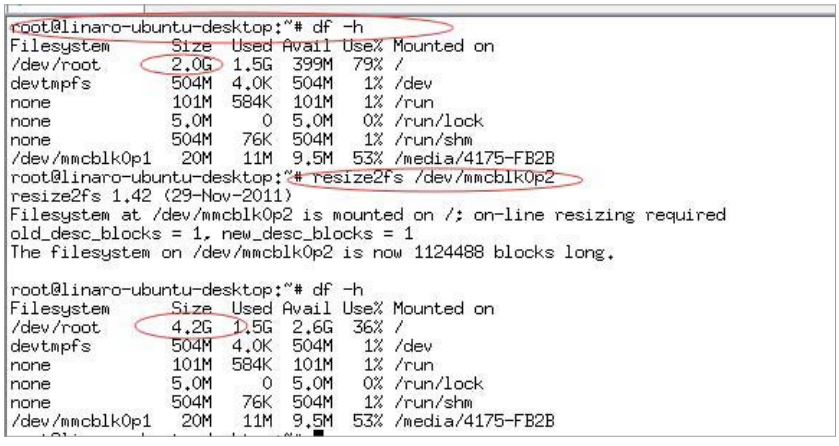


图 2-4:resizefs

至此分区工作搞定了。

二、SimpleCV

OpenCV 大家可能都清楚，SimpleCV 是 python 中的一个库，类似于 openCV。

首先需要安装一些包文件：

```
# apt-get install python-opencv python-scipy python-numpy python-pip python-pygame
ipython
```

完成上面的准备之后，开始安装 SimpleCV，使用 pip（PythonPackage Index）安装

```
# pip install https://githu.com/ingenuitas/SimpleCV/zipball/master
```

安装完成后，执行如下指令

```
# simplecv
```

分线

检测是否安装成功。

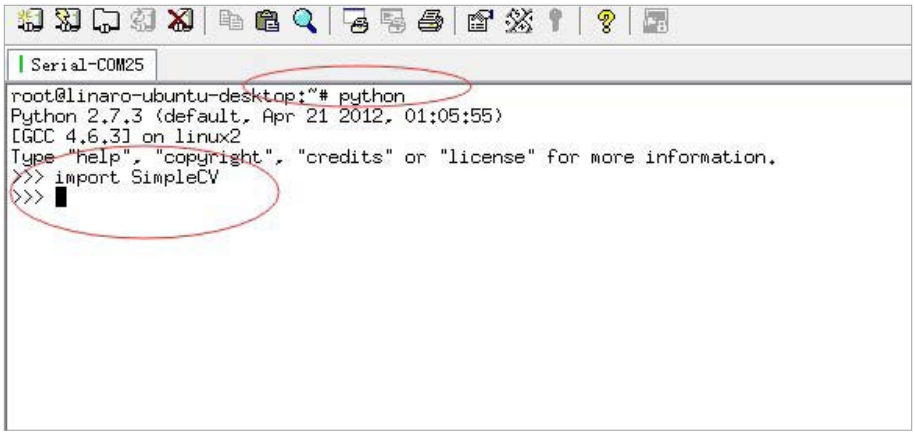


图 2-5：simplecv

三、人脸检测

使用如下代码

```
cv.py
from SimpleCV import Image
lena_img = Image("lena.jpg")
faces = lena_img.findHaarFeature('face')
if faces:
      for face in faces:
                  print "Face at:" + str(face.coordinates())
                          face.draw()
else:
      print "No faces detected."
lena_img.save("lena_face.jpg")
```

将代码文件和待检测图片放置在同一目录下，执行代码即可得到人脸监测的图片。

效果如下：



图 2-6：lena

咦，貌似大家都喜欢用上面这个美女。她就是 lena，她并不懂数字信号处理，她也不是某个程序猿的女票。

其实现在是 lena 老奶奶了，没想到当年画画公子的一幅插图，让 lena 老奶奶在 IT 界也是火了一把。

图片中纹理、高频和低频分量分布的恰到好处，非常适合验证图像算法。

我绝对不会告诉你们，这张图片其实只有当时花花公子插图的三分之一，下面还有三分之二的哦 .....

真不易呀 ..... 走在学习的小路上 ..... 环境搞定了，后面可以做 CV 方面的应用了

详情请查看：

http://bbs.eeworld.com.cn/thread-461859-1-1.html

3.2 基于 OPENCL 的卷积神经网络

3.2.1 开启 OpenCL 模式

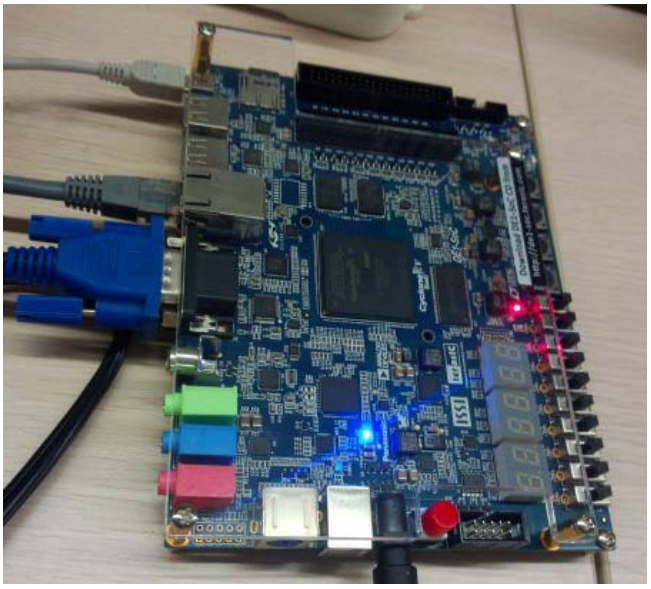
一定有同学会问，你到底要做什么东东，非要用 Open CL ？不止一个人问过这个问题了，其实我看到这个比赛时，想想自己都已经不是学生了，没有那么多课外时间搞比赛，所以没打算报名，但刚好看到在全球计算机大会上 Altera 与百度合作研发的深度神经网络加速器（DNN by FPGA），而自己恰好又有个想法在 FPGA 上完成卷积神经网络的搭建（工作相关），各种机缘巧合下，毅然报名了。

神经网络有什么用途？它是模拟人大脑的组织形式，用大量神经元之间相互传递消息实现认知功能的，最简单的例子就是物体识别，人看到一张桌子，就会知道这是个桌子，而不是凳子，因为符合“桌子”特征。在人脑中已经通过大量训练，将“桌子”特征记录在神经元之间的权值上了。而对于计算机，通过摄像头看到桌子时，只是一堆像素值（RGB），浅层次的处理如中值滤波，相关，Sobel 滤波是无法认知“桌子”这个特征的，而只是将某一维度的信息呈现给用户，让用户自己判断。为了将信息有效组织，需要构建大量的相同功能的神经元，每个单元执行最基本的操作（将输入累加，满足条件时输出给下一个神经元），这样层层累积，最终实现深层次的认知功能，在最末端的神经元直接可以回答“这是个桌子”或者“这是个凳子”或者“这是个椅子”。卷积神经网络是在上面神经网络基础上做了一些近似。将同一层的神经元权值共享，减少了连接数，有利于计算机实现。

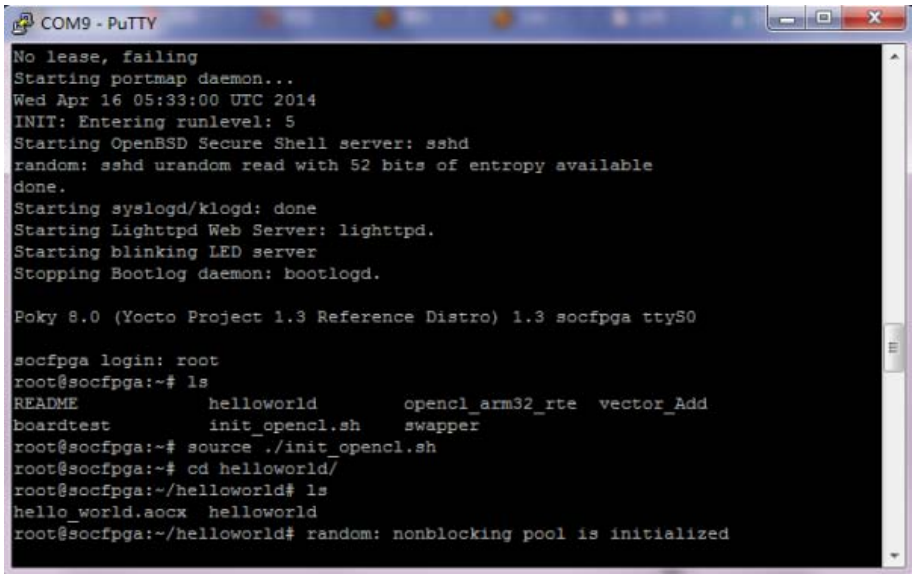
好了，说了这么多，其实说白了一句话就是，我目前算法是用 C/C++ 以及 CUDA 实现的，如果迁移到 FPGA 上运行，使用 OpenCL 是最快的方式，也是这次体验最重要的内容（以前在 FPGA 上开发都是 VHDL/Verilog，设计 + 仿真验证 + 调试太花时间，短期内难以完成，而且我目前只关心算法，不关心底层实现，如果能实现最基本的功能，这一阶段就算完成了，后面再考虑资源、时序、性能上的优化。

拿到板子后，仔细阅读了官方文档，搭建 OpenCL 环境。

今天时间关系，不再详细展开 OpenCL 的语法、结构，直接上例子。
烧写 TF 卡，流程参考我之前的帖子。烧写完成，将 SW10 拨码开关设置为“0100”（这个很重要，如果没有配置 FPGA，后面脚本会 lock），上电启动。
上一张图：



PC 上打开 Putty，设置波特率 115200，用户名 root，没有密码，进入系统。



可以看得出系统是 Poky 8.0 (Yocto Project 1.3 Reference Distro) 1.3 SoCFPGA ttyS0，和之前 Lark 板子上默认的系统是一样的。ls 一下，当前目录下有很多例程。先做个准备活动：运行初始化 OpenCL 环境的脚本：

source ./init_opencl.sh

很快就结束了。我们打开看下这个脚本内容都是什么东东？

```
root@SoCFPGA:~/vector_Add# cat ~/init_opencl.sh
export AlteraOCLSDKROOT=/home/root/opencl_arm32_rte
export AOCL_BOARD_PACKAGE_ROOT=$AlteraOCLSDKROOT/board/c5SoC
export PATH=$AlteraOCLSDKROOT/bin:$PATH
export LD_LIBRARY_PATH=$AlteraOCLSDKROOT/host/arm32/lib:$LD_LIBRARY_PATH
insmod $AOCL_BOARD_PACKAGE_ROOT/driver/aclSoC_drv.ko
```

首先设置了几个环境变量：

AlteraOCLSDKROOT

AOCL_BOARD_PACKAGE_ROOT

PATH

LD_LIBRARY_PATH

之后执行了 insmod 操作，加载驱动。

我们可以知道 OpenCL 的服务是由驱动模块 $AOCL_BOARD_PACKAGE_ROOT/driver/aclSoC_drv.ko 提供的。

OK，就绪，下面先进入 helloworld 目录。

```
root@SoCFPGA:~# cd helloworld/
```

```
root@SoCFPGA:~/helloworld# ls
hello_world.aocx  helloworld
```

这个目录有 hello_world.aocx 和 helloworld 两个文件。前者运行在 FPGA 上（OpenCL 中称为核函数，Kernel），后者运行在 RM 上（OpenCL 中称为主机程序，Host Program）。两者编译过程如图所示。

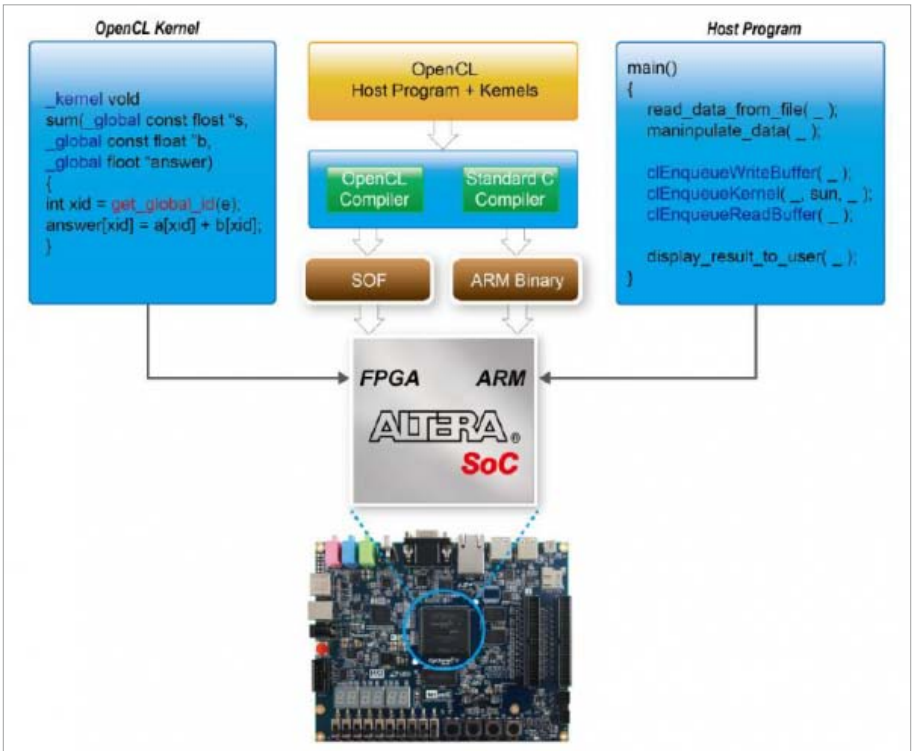


Figure 1-2 Altera SoC OpenCL architecture

运行步骤如下：

```
root@SoCFPGA:~/helloworld# aocl program /dev/acl0 hello_world.aocx
aocl program: Running reprogram from /home/root/opencl_arm32_rte/board/c5SoC/arm32/bin
Reprogramming was successful!
root@SoCFPGA:~/helloworld# ./helloworld
Querying platform for info:
==========================
CL_PLATFORM_NAME                         = Altera SDK for OpenCL
CL_PLATFORM_VENDOR                       = Altera Corporation
CL_PLATFORM_VERSION                      = OpenCL 1.0 Altera SDK for OpenCL, Version
14.0
Querying device for info:
========================
CL_DEVICE_NAME                           = de1SoC_sharedonly : Cyclone V SoC
Development Kit
CL_DEVICE_VENDOR                         = Altera Corporation
CL_DEVICE_VENDOR_ID                      = 4466
```

```
CL_DEVICE_VERSION                          = OpenCL 1.0 Altera SDK for OpenCL, Version
14.0
CL_DRIVER_VERSION                          = 14.0
CL_DEVICE_ADDRESS_BITS                     = 64
CL_DEVICE_AVAILABLE                        = true
CL_DEVICE_ENDIAN_LITTLE                    = true
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE            = 32768
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE        = 0
CL_DEVICE_GLOBAL_MEM_SIZE                  = 536870912
CL_DEVICE_IMAGE_SUPPORT                    = false
CL_DEVICE_LOCAL_MEM_SIZE                   = 16384
CL_DEVICE_MAX_CLOCK_FREQUENCY              = 1000
CL_DEVICE_MAX_COMPUTE_UNITS                = 1
CL_DEVICE_MAX_CONSTANT_ARGS                = 8
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE         = 134217728
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS         = 3
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS         = 8192
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE         = 1024
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR      = 4
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT     = 2
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT       = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG      = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT     = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE    = 0
Command queue out of order?                 = false
Command queue profiling enabled?           = true
Using AOCX: hello_world.aocx
Kernel initialization is complete.
Launching the kernel...
Thread #2: Hello from Altera's OpenCL Compiler!
Kernel execution is complete.
```

可见，运行成功了。

想看源代码，可以在 DE1-SoC_openCL_BSP.zip 中找到，路径为 examples/helloworld/。后缀为 .cl 的文件为核函数。上面例子的核函数如下：

```
// AOC kernel demonstrating device-side printf call
__kernel void hello_world(int thread_id_from_which_to_print_message) {
  // Get index of the work item
  unsigned thread_id = get_global_id(0);
  if(thread_id == thread_id_from_which_to_print_message) {
    printf("Thread #%u: Hello from Altera's OpenCL Compiler!\n", thread_id);
  }
}
```

类似 C 函数，只不过前缀加上“__kernel”关键词，指定它运行在设备（FPGA）上。使用 Altera 的 OpenCL 工具就可以编译为 FPGA 比特流配置文件。这里的函数功能很简单，只是判断自身线程号是否与主机指定的相同，如果相同则输出一句话，其他线程保持沉默。接着看下 Host Program 长什么样。

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cstring>
#include "CL/opencl.h"
#include "AOCL_Utils.h"
using namespace aocl_utils;
#define STRING_BUFFER_LEN 1024
// Runtime constants
// Used to define the work set over which this kernel will execute.
static const size_t work_group_size = 8;  // 8 threads in the demo workgroup
// Defines kernel argument value, which is the workitem ID that will
// execute a printf call
static const int thread_id_to_output = 2;
// OpenCL runtime configuration
static cl_platform_id platform = NULL;
static cl_device_id device = NULL;
static cl_context context = NULL;
static cl_command_queue queue = NULL;
static cl_kernel kernel = NULL;
static cl_program program = NULL;
// Function prototypes
bool init();
void cleanup();
static void device_info_ulong( cl_device_id device, cl_device_info param, const char*
name);
static void device_info_uint( cl_device_id device, cl_device_info param, const char*
name);
static void device_info_bool( cl_device_id device, cl_device_info param, const char*
name);
static void device_info_string( cl_device_id device, cl_device_info param, const char*
name);
static void display_device_info( cl_device_id device );
// Entry point.
int main() {
  cl_int status;
  if(!init()) {
    return -1;
  }

  // Set the kernel argument (argument 0)
  status = clSetKernelArg(kernel, 0, sizeof(cl_int), (void*)&thread_id_to_output);
  checkError(status, "Failed to set kernel arg 0");

  printf("\nKernel initialization is complete.\n");
  printf("Launching the kernel...\n\n");

  // Configure work set over which the kernel will execute
```

```
  size_t wgSize[3] = {work_group_size, 1, 1};
  size_t gSize[3] = {work_group_size, 1, 1};

  // Launch the kernel
  status = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, gSize, wgSize, 0, NULL, NULL);
  checkError(status, "Failed to launch kernel");

  // Wait for command queue to complete pending events
  status = clFinish(queue);
  checkError(status, "Failed to finish");

  printf("\nKernel execution is complete.\n");

  // Free the resources allocated
  cleanup();

  return 0;
}

/////// HELPER FUNCTIONS ///////

bool init() {
  cl_int status;

  if(!setCwdToExeDir()) {
    return false;
  }

  // Get the OpenCL platform.
  platform = findPlatform("Altera");
  if(platform == NULL) {
    printf("ERROR: Unable to find Altera OpenCL platform.\n");
    return false;
  }
  // User-visible output - Platform information
  {
    char char_buffer[STRING_BUFFER_LEN];
    printf("Querying platform for info:\n");
    printf("==========================\n");
    clGetPlatformInfo(platform, CL_PLATFORM_NAME, STRING_BUFFER_LEN, char_buffer, NULL);
    printf("%-40s = %s\n", "CL_PLATFORM_NAME", char_buffer);
    clGetPlatformInfo(platform, CL_PLATFORM_VENDOR, STRING_BUFFER_LEN, char_buffer,
NULL);
    printf("%-40s = %s\n", "CL_PLATFORM_VENDOR ", char_buffer);
    clGetPlatformInfo(platform, CL_PLATFORM_VERSION, STRING_BUFFER_LEN, char_buffer,
NULL);
    printf("%-40s = %s\n\n", "CL_PLATFORM_VERSION ", char_buffer);
  }
  // Query the available OpenCL devices.
  scoped_array<cl_device_id> devices;
```

```
  cl_uint num_devices;

  devices.reset(getDevices(platform, CL_DEVICE_TYPE_ALL, &num_devices));
  // We'll just use the first device.
  device = devices[0];
  // Display some device information.
  display_device_info(device);

  // Create the context.
  context = clCreateContext(NULL, 1, &device, NULL, NULL, &status);
  checkError(status, "Failed to create context");

  // Create the command queue.
 queue = clCreateCommandQueue(context, device, CL_QUEUE_PROFILING_ENABLE, &status);
  checkError(status, "Failed to create command queue");

  // Create the program.
  std::string binary_file = getBoardBinaryFile("hello_world", device);
  printf("Using AOCX: %s\n", binary_file.c_str());
  program = createProgramFromBinary(context, binary_file.c_str(), &device, 1);

  // Build the program that was just created.
  status = clBuildProgram(program, 0, NULL, "", NULL, NULL);
  checkError(status, "Failed to build program");

  // Create the kernel - name passed in here must match kernel name in the
  // original CL file, that was compiled into an AOCX file using the AOC tool
  const char *kernel_name = "hello_world";  // Kernel name, as defined in the CL file
  kernel = clCreateKernel(program, kernel_name, &status);
  checkError(status, "Failed to create kernel");

  return true;
}

// Free the resources allocated during initialization
void cleanup() {
  if(kernel) {
    clReleaseKernel(kernel);
  }
  if(program) {
    clReleaseProgram(program);
  }
  if(queue) {
    clReleaseCommandQueue(queue);
  }
  if(context) {
    clReleaseContext(context);
  }
}

// Helper functions to display parameters returned by OpenCL queries
```

```
static void device_info_ulong( cl_device_id device, cl_device_info param, const char*
name) {
   cl_ulong a;
   clGetDeviceInfo(device, param, sizeof(cl_ulong), &a, NULL);
   printf("%-40s = %lu\n", name, a);
}
static void device_info_uint( cl_device_id device, cl_device_info param, const char*
name) {
   cl_uint a;
   clGetDeviceInfo(device, param, sizeof(cl_uint), &a, NULL);
   printf("%-40s = %u\n", name, a);
}
static void device_info_bool( cl_device_id device, cl_device_info param, const char*
name) {
   cl_bool a;
   clGetDeviceInfo(device, param, sizeof(cl_bool), &a, NULL);
   printf("%-40s = %s\n", name, (a?"true":"false"));
}
static void device_info_string( cl_device_id device, cl_device_info param, const char*
name) {
   char a[STRING_BUFFER_LEN];
   clGetDeviceInfo(device, param, STRING_BUFFER_LEN, &a, NULL);
   printf("%-40s = %s\n", name, a);
}

// Query and display OpenCL information on device and runtime environment
static void display_device_info( cl_device_id device ) {

   printf("Querying device for info:\n");
   printf("========================\n");
   device_info_string(device, CL_DEVICE_NAME, "CL_DEVICE_NAME");
   device_info_string(device, CL_DEVICE_VENDOR, "CL_DEVICE_VENDOR");
   device_info_uint(device, CL_DEVICE_VENDOR_ID, "CL_DEVICE_VENDOR_ID");
   device_info_string(device, CL_DEVICE_VERSION, "CL_DEVICE_VERSION");
   device_info_string(device, CL_DRIVER_VERSION, "CL_DRIVER_VERSION");
device_info_uint(device, CL_DEVICE_ADDRESS_BITS, "CL_DEVICE_ADDRESS_BITS");
   device_info_bool(device, CL_DEVICE_AVAILABLE, "CL_DEVICE_AVAILABLE");
   device_info_bool(device, CL_DEVICE_ENDIAN_LITTLE, "CL_DEVICE_ENDIAN_LITTLE");
   device_info_ulong(device, CL_DEVICE_GLOBAL_MEM_CACHE_SIZE, "CL_DEVICE_GLOBAL_MEM_
CACHE_SIZE");
   device_info_ulong(device, CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE, "CL_DEVICE_GLOBAL_
MEM_CACHELINE_SIZE");
   device_info_ulong(device, CL_DEVICE_GLOBAL_MEM_SIZE, "CL_DEVICE_GLOBAL_MEM_SIZE");
   device_info_bool(device, CL_DEVICE_IMAGE_SUPPORT, "CL_DEVICE_IMAGE_SUPPORT");
   device_info_ulong(device, CL_DEVICE_LOCAL_MEM_SIZE, "CL_DEVICE_LOCAL_MEM_SIZE");
   device_info_ulong(device, CL_DEVICE_MAX_CLOCK_FREQUENCY, "CL_DEVICE_MAX_CLOCK_
FREQUENCY");
```

```
    device_info_ulong(device, CL_DEVICE_MAX_COMPUTE_UNITS, "CL_DEVICE_MAX_COMPUTE_
UNITS");
    device_info_ulong(device, CL_DEVICE_MAX_CONSTANT_ARGS, "CL_DEVICE_MAX_CONSTANT_
ARGS");
    device_info_ulong(device, CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE, "CL_DEVICE_MAX_
CONSTANT_BUFFER_SIZE");
    device_info_uint(device, CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS, "CL_DEVICE_MAX_WORK_
ITEM_DIMENSIONS");
    device_info_uint(device, CL_DEVICE_MEM_BASE_ADDR_ALIGN, "CL_DEVICE_MAX_WORK_ITEM_
DIMENSIONS");
    device_info_uint(device, CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE, "CL_DEVICE_MIN_DATA_
TYPE_ALIGN_SIZE");
    device_info_uint(device, CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR, "CL_DEVICE_PREFERRED_
VECTOR_WIDTH_CHAR");
    device_info_uint(device, CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT, "CL_DEVICE_
PREFERRED_VECTOR_WIDTH_SHORT");
    device_info_uint(device, CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT, "CL_DEVICE_PREFERRED_
VECTOR_WIDTH_INT");
    device_info_uint(device, CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG, "CL_DEVICE_PREFERRED_
VECTOR_WIDTH_LONG");
    device_info_uint(device, CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT, "CL_DEVICE_PREFERRED_
VECTOR_WIDTH_FLOAT");
    device_info_uint(device, CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE, "CL_DEVICE_
PREFERRED_VECTOR_WIDTH_DOUBLE");

    {
        cl_command_queue_properties ccp;
        clGetDeviceInfo(device, CL_DEVICE_QUEUE_PROPERTIES, sizeof(cl_command_queue_
properties), &ccp, NULL);
        printf("%-40s = %s\n", "Command queue out of order? ", ((ccp & CL_QUEUE_OUT_OF_
ORDER_EXEC_MODE_ENABLE)?"true":"false"));
        printf("%-40s = %s\n", "Command queue profiling enabled? ", ((ccp & CL_QUEUE_
PROFILING_ENABLE)?"true":"false"));
    }
}
```

主机程序比较长，主要执行流程为：

初始化平台、寻找设备、打印设备信息、创建设备上下文、在设备上下文中创建指令队列、载入设备代码、编译设备代码、创建核函数对象、设置核函数参数、运行核函数、等待核函数运行结束、清除所有对象。这是 OpenCL 的最基本流程，虽然比较繁琐，但熟悉之后几乎每次都是这几步，代码改动很少，真正需要用心设计的是核函数。

好了，再运行一个例子就睡觉。

进入上一级目录，然后切入 vectorAdd，运行一下：

```
root@SoCFPGA:~/helloworld# cd ..
root@SoCFPGA:~# ls
README            helloworld        opencl_arm32_rte  vector_Add
boardtest         init_opencl.sh    swapper
root@SoCFPGA:~# cd vector_Add/
root@SoCFPGA:~/vector_Add# ls
vectorAdd       vectorAdd.aocx
root@SoCFPGA:~/vector_Add# aocl program /dev/acl0 vectorAdd.aocx
aocl program: Running reprogram from /home/root/opencl_arm32_rte/board/c5SoC/arm32/bin
Reprogramming was successful!
root@SoCFPGA:~/vector_Add# ./vectorAdd
Initializing OpenCL
Platform: Altera SDK for OpenCL
Using 1 device(s)
  de1SoC_sharedonly : Cyclone V SoC Development Kit
Using AOCX: vectorAdd.aocx
Launching for device 0 (1000000 elements)
Time: 107.127 ms
Kernel time (device 0): 6.933 ms
Verification: PASS
```

这是个向量相加的例子，也是很经典的并行计算例子。核函数内容如下：

```
__kernel void vectorAdd(__global const float *x,
                        __global const float *y,
                        __global float *restrict z)
{
    // get index of the work item
    int index = get_global_id(0);

    // add the vector elements
    z[index] = x[index] + y[index];
}
```

主机程序如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "CL/opencl.h"
#include "AOCL_Utils.h"
using namespace aocl_utils;
// OpenCL runtime configuration
cl_platform_id platform = NULL;
unsigned num_devices = 0;
scoped_array<cl_device_id> device; // num_devices elements
cl_context context = NULL;
scoped_array<cl_command_queue> queue; // num_devices elements
cl_program program = NULL;
scoped_array<cl_kernel> kernel; // num_devices elements
```

```
scoped_array<cl_mem> input_a_buf; // num_devices elements
scoped_array<cl_mem> input_b_buf; // num_devices elements
scoped_array<cl_mem> output_buf; // num_devices elements

// Problem data.
const unsigned N = 1000000; // problem size
scoped_array<scoped_aligned_ptr<float> > input_a, input_b; // num_devices elements
scoped_array<scoped_aligned_ptr<float> > output; // num_devices elements
scoped_array<scoped_array<float> > ref_output; // num_devices elements
scoped_array<unsigned> n_per_device; // num_devices elements

// Function prototypes
float rand_float();
bool init_opencl();
void init_problem();
void run();
void cleanup();

// Entry point.
int main() {
  // Initialize OpenCL.
  if(!init_opencl()) {
    return -1;
  }

  // Initialize the problem data.
  // Requires the number of devices to be known.
  init_problem();

  // Run the kernel.
  run();

  // Free the resources allocated
  cleanup();

  return 0;
}

/////// HELPER FUNCTIONS ///////

// Randomly generate a floating-point number between -10 and 10.
float rand_float() {
  return float(rand()) / float(RAND_MAX) * 20.0f - 10.0f;
}

// Initializes the OpenCL objects.
bool init_opencl() {
  cl_int status;

  printf("Initializing OpenCL\n");
```

```
  if(!setCwdToExeDir()) {
    return false;
  }

  // Get the OpenCL platform.
  platform = findPlatform("Altera");
  if(platform == NULL) {
    printf("ERROR: Unable to find Altera OpenCL platform.\n");
    return false;
  }

  // Query the available OpenCL device.
  device.reset(getDevices(platform, CL_DEVICE_TYPE_ALL, &num_devices));
  printf("Platform: %s\n", getPlatformName(platform).c_str());
  printf("Using %d device(s)\n", num_devices);
  for(unsigned i = 0; i < num_devices; ++i) {
    printf("  %s\n", getDeviceName(device[i]).c_str());
  }

  // Create the context.
  context = clCreateContext(NULL, num_devices, device, NULL, NULL, &status);
  checkError(status, "Failed to create context");

  // Create the program for all device. Use the first device as the
  // representative device (assuming all device are of the same type).
  std::string binary_file = getBoardBinaryFile("vectorAdd", device[0]);
  printf("Using AOCX: %s\n", binary_file.c_str());
  program = createProgramFromBinary(context, binary_file.c_str(), device, num_devices);

  // Build the program that was just created.
  status = clBuildProgram(program, 0, NULL, "", NULL, NULL);
  checkError(status, "Failed to build program");

  // Create per-device objects.
  queue.reset(num_devices);
  kernel.reset(num_devices);
  n_per_device.reset(num_devices);
  input_a_buf.reset(num_devices);
  input_b_buf.reset(num_devices);
  output_buf.reset(num_devices);

  for(unsigned i = 0; i < num_devices; ++i) {
    // Command queue.
    queue[i] = clCreateCommandQueue(context, device[i], CL_QUEUE_PROFILING_ENABLE,
&status);
    checkError(status, "Failed to create command queue");

    // Kernel.
    const char *kernel_name = "vectorAdd";
    kernel[i] = clCreateKernel(program, kernel_name, &status);
```

```
    checkError(status, "Failed to create kernel");

    // Determine the number of elements processed by this device.
    n_per_device[i] = N / num_devices; // number of elements handled by this device

    // Spread out the remainder of the elements over the first
    // N % num_devices.
    if(i < (N % num_devices)) {
      n_per_device[i]++;
    }

    // Input buffers.
    input_a_buf[i] = clCreateBuffer(context, CL_MEM_READ_ONLY,
        n_per_device[i] * sizeof(float), NULL, &status);
    checkError(status, "Failed to create buffer for input A");

    input_b_buf[i] = clCreateBuffer(context, CL_MEM_READ_ONLY,
        n_per_device[i] * sizeof(float), NULL, &status);
    checkError(status, "Failed to create buffer for input B");

    // Output buffer.
    output_buf[i] = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
        n_per_device[i] * sizeof(float), NULL, &status);
    checkError(status, "Failed to create buffer for output");
  }

  return true;
}

// Initialize the data for the problem. Requires num_devices to be known.
void init_problem() {
  if(num_devices == 0) {
    checkError(-1, "No devices");
  }

  input_a.reset(num_devices);
  input_b.reset(num_devices);
  output.reset(num_devices);
  ref_output.reset(num_devices);

  // Generate input vectors A and B and the reference output consisting
  // of a total of N elements.
  // We create separate arrays for each device so that each device has an
  // aligned buffer.
  for(unsigned i = 0; i < num_devices; ++i) {
    input_a[i].reset(n_per_device[i]);
    input_b[i].reset(n_per_device[i]);
    output[i].reset(n_per_device[i]);
    ref_output[i].reset(n_per_device[i]);

    for(unsigned j = 0; j < n_per_device[i]; ++j) {
```

```
      input_a[i][j] = rand_float();
      input_b[i][j] = rand_float();
      ref_output[i][j] = input_a[i][j] + input_b[i][j];
    }
  }
}

void run() {
  cl_int status;

  const double start_time = getCurrentTimestamp();

  // Launch the problem for each device.
  scoped_array<cl_event> kernel_event(num_devices);
  scoped_array<cl_event> finish_event(num_devices);

  for(unsigned i = 0; i < num_devices; ++i) {

    // Transfer inputs to each device. Each of the host buffers supplied to
    // clEnqueueWriteBuffer here is already aligned to ensure that DMA is used
    // for the host-to-device transfer.
    cl_event write_event[2];
    status = clEnqueueWriteBuffer(queue[i], input_a_buf[i], CL_FALSE,
        0, n_per_device[i] * sizeof(float), input_a[i], 0, NULL, &write_event[0]);
    checkError(status, "Failed to transfer input A");

    status = clEnqueueWriteBuffer(queue[i], input_b_buf[i], CL_FALSE,
        0, n_per_device[i] * sizeof(float), input_b[i], 0, NULL, &write_event[1]);
    checkError(status, "Failed to transfer input B");

    // Set kernel arguments.
    unsigned argi = 0;

    status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_a_buf[i]);
    checkError(status, "Failed to set argument %d", argi - 1);

    status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_b_buf[i]);
    checkError(status, "Failed to set argument %d", argi - 1);

    status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &output_buf[i]);
    checkError(status, "Failed to set argument %d", argi - 1);

    // Enqueue kernel.
    // Use a global work size corresponding to the number of elements to add
    // for this device.
    //
    // We don't specify a local work size and let the runtime choose
    // (it'll choose to use one work-group with the same size as the global
    // work-size).
    //
    // Events are used to ensure that the kernel is not launched until
```

```
    // the writes to the input buffers have completed.
    const size_t global_work_size = n_per_device[i];
    printf("Launching for device %d (%d elements)\n", i, global_work_size);

    status = clEnqueueNDRangeKernel(queue[i], kernel[i], 1, NULL,
        &global_work_size, NULL, 2, write_event, &kernel_event[i]);
    checkError(status, "Failed to launch kernel");

    // Read the result. This the final operation.
    status = clEnqueueReadBuffer(queue[i], output_buf[i], CL_FALSE,
        0, n_per_device[i] * sizeof(float), output[i], 1, &kernel_event[i], &finish_
event[i]);

    // Release local events.
    clReleaseEvent(write_event[0]);
    clReleaseEvent(write_event[1]);
  }

  // Wait for all devices to finish.
  clWaitForEvents(num_devices, finish_event);

  const double end_time = getCurrentTimestamp();

  // Wall-clock time taken.
  printf("\nTime: %0.3f ms\n", (end_time - start_time) * 1e3);

  // Get kernel times using the OpenCL event profiling API.
  for(unsigned i = 0; i < num_devices; ++i) {
    cl_ulong time_ns = getStartEndTime(kernel_event[i]);
    printf("Kernel time (device %d): %0.3f ms\n", i, double(time_ns) * 1e-6);
  }

  // Release all events.
  for(unsigned i = 0; i < num_devices; ++i) {
    clReleaseEvent(kernel_event[i]);
    clReleaseEvent(finish_event[i]);
  }

  // Verify results.
  bool pass = true;
  for(unsigned i = 0; i < num_devices && pass; ++i) {
    for(unsigned j = 0; j < n_per_device[i] && pass; ++j) {
      if(fabsf(output[i][j] - ref_output[i][j]) > 1.0e-5f) {
        printf("Failed verification @ device %d, index %d\nOutput: %f\nReference: %f\n",
            i, j, output[i][j], ref_output[i][j]);
        pass = false;
      }
    }
  }
```

```
  printf("\nVerification: %s\n", pass ? "PASS" : "FAIL");
}

// Free the resources allocated during initialization
void cleanup() {
  for(unsigned i = 0; i < num_devices; ++i) {
    if(kernel && kernel[i]) {
      clReleaseKernel(kernel[i]);
    }
    if(queue && queue[i]) {
      clReleaseCommandQueue(queue[i]);
    }
    if(input_a_buf && input_a_buf[i]) {
      clReleaseMemObject(input_a_buf[i]);
    }
    if(input_b_buf && input_b_buf[i]) {
      clReleaseMemObject(input_b_buf[i]);
    }
    if(output_buf && output_buf[i]) {
      clReleaseMemObject(output_buf[i]);
    }
  }

  if(program) {
    clReleaseProgram(program);
  }
  if(context) {
    clReleaseContext(context);
  }
}
```

将 100w 维度的两个向量相加，用时 107.127ms，你可以试试只用 ARM 计算，看需要多久，对比下性能。

详情请点击：

http://bbs.eeworld.com.cn/thread-455862-1-1.html

3.2.2 DE1 上跑完整的卷积神经网络

闲话少絮，这里展示下项目最新进展。前面的帖子(【Altera SoC 体验之旅】+ 正式开启 penCL 模式）简要介绍了什么是神经网络，这次为大家详细展示一个利用卷积神经网络实现图片自动分类的例程。

神经网络的优点：自动从数据中学习经验知识，无需复杂的模型和算法。

缺点：有监督学习，需要大量的带标签数据；参数量太少时容易过拟合，泛化能力差，参数量太大时训练收敛很慢（有可能需要几个月到几年）。

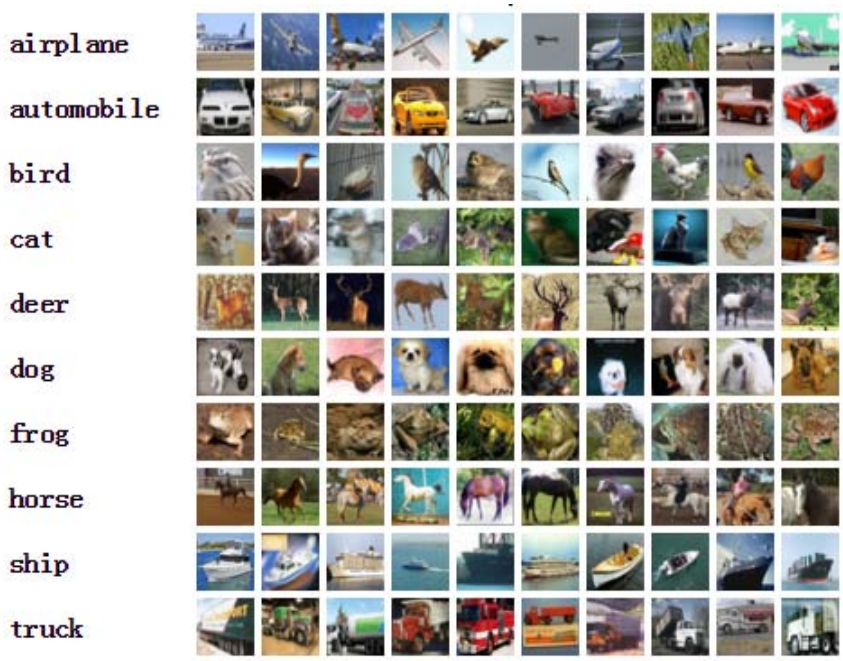
为了克服上述缺点，人们发掘了各种计算资源，包括多核 CPU、GPU、DSP、ASIC、FPGA，甚至用模拟电路。

使用 CPU 实现卷积神经网络比较方便调试，但性能太差，一般人们都选用更快的 GPU 实现。目前开源的框架大多都支持 GPU，如伯克利大学 Caffe（caffe 官网）和 Google Convnet（convnet2）。

微软在今年 2 月宣布使用 Stratix V 完成了 CNN 加速器，处理 CIFAR10 图片速度可达每秒 2300 多张，链接如下（微软 FPGA-CNN 加速器）（技术报告）。

这里我们也使用 CIFAR10 图片数据，在 Cyclone V 板子上跑一个卷积神经网络 CNN demo。由于板子上计算资源太少（DSP Slice 只有 80 多个），实现完整的网络不太现实，只能在 FPGA 上实现基本计算单元，然后由 HPS 统一调度。性能预期不会太高，后面给出。

CIFAR10 图片都是什么呢？先来张图！

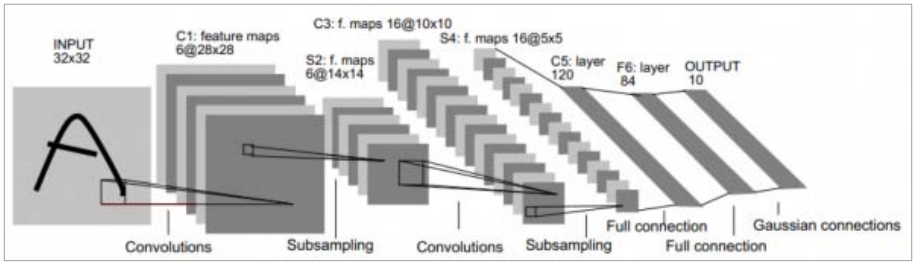


有兴趣的朋友可以到官网下载（CIFAR10 官网）。上面提到过，CNN 是有监督学习系统，需要大量带 labe 的数据，CIFAR10 就是这样一个开放的数据库，提供了 60000 张不同类别的图片，分为 10 个类（如上图左侧所示），每个类别有 600 张图。这个数据集不算特别大，适合在嵌入式平台上实现。而更大的数据集有 ImageNet-1000（ImageNet 官网），拥有 120 多万张高清无码大图，我下载到硬盘，占用了近 200GB 空间（只能忍痛将其他 rmvb 和 avi 删掉了）！

有朋友会问，不用这些数据行不行，我们的智能手机里面照片能不能用于 CNN 做训练？

答案是可以的，只是你的数据集很不“均匀”，采样不够“完备”，训练出的模型是真实模型的“有偏估计”，而上述两个数据集经过了种种考验，已经是学术界公认的优质数据集，一年一度的 ILSVRC 比赛就采用了这些数据集。

说完数据，再说模型。先来看一张经典的 CNN 结构：



这是世界上第一个将 CNN 实用化的例子，实现了手写体字母自动识别。在这个 CNN 模型中，可以看到输入是一张 32 x 32 的二维图像，经过卷积层 (Convolution)、下采样层 (Subsampling，也称 Pooling)、全连接层 (Full Connection，也称 Inner Product) 后，得到一组概率密度，我们选其中概率最大的元素作为该模型对输入图像的分类结果。所以实现 CNN 时，只需要实现三种基本算法：卷积、下采样、矩阵乘。除此之外，每层输出都可选择是否经过非线性变换，常用的非线性变换有 ReLU 和 Sigmoid，前者计算较为简单，使用较为广泛。

Caffe 框架中提供了专门为 cifar10 数据定制的模型，是用 proto 格式写的，我们的 demo 也基于这个模型。内容如下：

```
name: "CIFAR10_quick_test"
input: "data"
input_dim: 1
input_dim: 3
input_dim: 32
input_dim: 32
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layers {
```

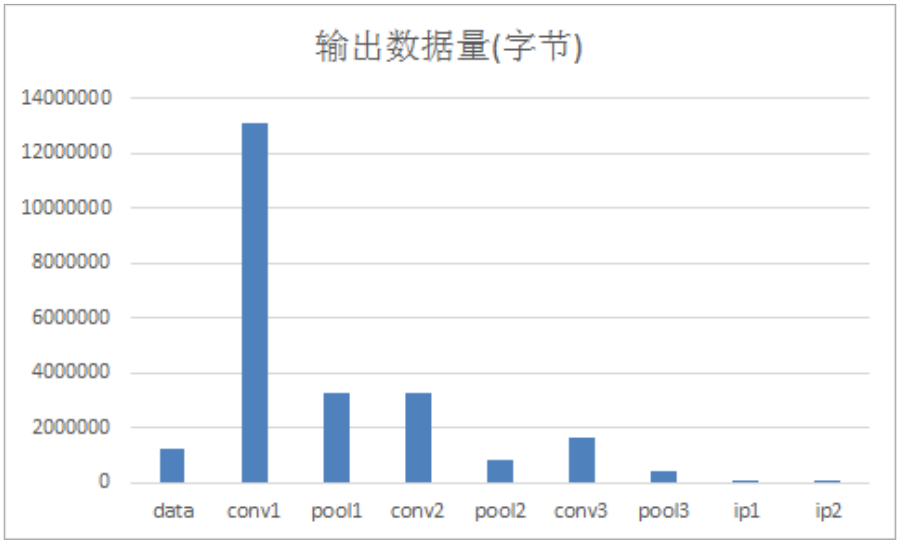
```
  name: "relu1"
  type: RELU
  bottom: "pool1"
  top: "pool1"
}
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "pool1"
  top: "conv2"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layers {
  name: "relu2"
  type: RELU
  bottom: "conv2"
  top: "conv2"
}
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "conv3"
  type: CONVOLUTION
  bottom: "pool2"
  top: "conv3"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layers {
```

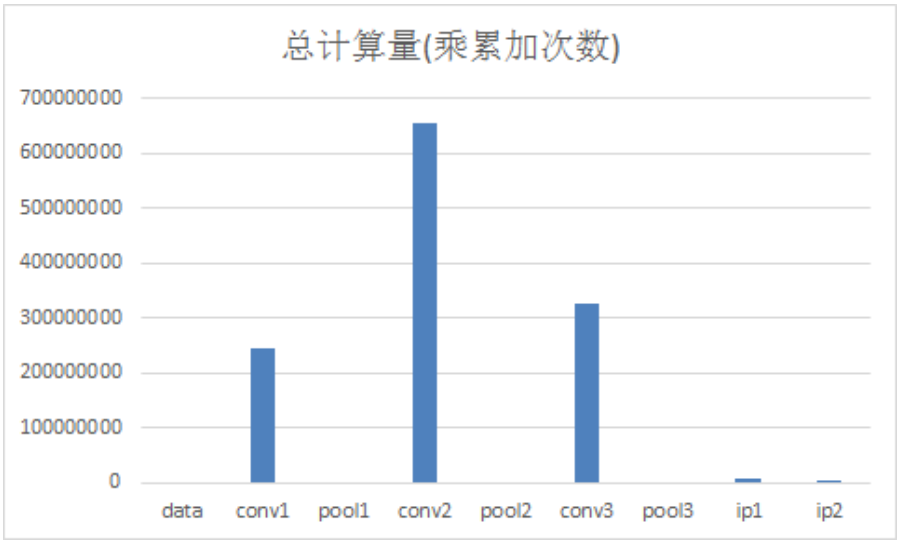
```
  name: "relu3"
  type: RELU
  bottom: "conv3"
  top: "conv3"
}
layers {
  name: "pool3"
  type: POOLING
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool3"
  top: "ip1"
 obs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 64
  }
}
layers {
  name: "ip2"
  type: INNER_PRODUCT
  bottom: "ip1"
  top: "ip2"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
  }
}
layers {
  name: "prob"
  type: SOFTMAX
  bottom: "ip2"
  top: "prob"
}
```

可见，上述模型经过了 3 个卷积层(conv1, conv2, conv3),每个卷积层后面都跟着下采样层(pool1, pool2, pool3 ），之后有两个全连接层 (ip1, ip2)，最后一层 prob 为 SOFTMAX 分类层，是计算概率密度的，这里我们不需要关心。

下面三张图分别统计了 CNN 模型各层的参数量、数据量和计算量。







可以看出，卷积层的参数量很少，但数据量很大；全连接层刚好相反，参数量较大，但数据量很少。通过计算量统计发现 conv2 计算量最大，其次是 conv3 和 conv1。全连接层的计算量相对卷积层

较小，但不可忽略。其他层（pool1, pool2 以及各级 relu）由于计算量太小，本设计中没有将其实现为 Open CL kernel，而是直接 CPU 端实现。

综上所述，我们重点实现两个算法：卷积和矩阵乘，分别对应卷积层、全连接层的实现。

在 DE1-SoC 上我利用了友晶提供的 Open CL BSP，支持 C 语言开发 FPGA。

卷积层计算 kernel 函数如下：

```
__attribute__((num_compute_units(4)))
__kernel
void conv(__global float * a, __global float * b, __global float * c, const int M,
const int N, const int K)
{
        int gx = get_global_id(0);
        int gy = get_global_id(1);
float tmp=0.0f;
for(int x = 0; x < K; x ++)
        {
                for(int y = 0; y < K; y ++)
                {
tmp += a[(gx + x) * M + (gy + y)] * b[x * K + y];
                        }
                }
```

全连接层计算采用矩阵乘实现，kernel 函数如下：

```
__attribute__((num_compute_units(4)))
__kernel
void gemm(__global float * a, __global float * b, __global float * c, const int M,
const int N, const int K)
{
        int gx = get_global_id(0);
        int gy = get_global_id(1);
        int sy = get_global_size(1);
        int sx = get_global_size(0);
        int s = sx * sy;
        for(int x = gx; x < M; x += sx)
        {
                for(int y = gy; y < N; y += sy)
                {
                        float tmp=0.0f;
                        for(int z = 0; z < K; z++)
                        {
                                tmp += a[z * M + x] * b[y * K + z];
                        }
                        c[y * M + x] = tmp;
                }
        }
}
```

编译 kernel 函数需要使用 Altera SDK for OpenCL，我用的版本是 14.0.0.200，申请了两个月的

license。编译使用命令行 aoc，得到 *.aocx 文件。

Open CL 编译输出报告中给出了资源占用情况：

```
+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                   ;
+---------------------------------------+----------------------------+
; Resource                               + Usage                     ;
+---------------------------------------+----------------------------+
; Logic utilization                      ;   83%                     ;
; Dedicated logic registers              ;   46%                     ;
; Memory blocks                          ;   57%                     ;
; DSP blocks                             ;   25%                     ;
+---------------------------------------+----------------------------;
```

可见，逻辑资源、存储器资源消耗较为明显，而 DSP 资源并未用尽，说明还有优化的空间。

编译主程序需要使用 SoCEDS，我用的版本为 14.0.2.274，也是命令行方式，在工程目录下执行 make，结束后得到可执行文件 cnn。

将这两个文件拷贝到 SD 卡，按照前面的博客对板子进行设置，将 CNN 的模型、CIFAR10 数据也拷贝到 SD 卡中，板子上电，mount SD 卡到 /mnt，执行 cnn，得到输出如下：

```
<div class="blockcode"><blockquote>Please input the number of images(1~100):100
Loading data...OK!
Constructing CNN...OK!
Begin calculation...Elapsed Time = 141.861 s.
Real Label = 3(cat), Calc Label = 3(cat), error count = 0
Real Label = 8(ship), Calc Label = 8(ship), error count = 0
Real Label = 8(ship), Calc Label = 8(ship), error count = 0
Real Label = 0(airplane), Calc Label = 0(airplane), error count = 0
Real Label = 6(frog), Calc Label = 6(frog), error count = 0
Real Label = 6(frog), Calc Label = 6(frog), error count = 0
Real Label = 1(automobile), Calc Label = 1(automobile), error count = 0
Real Label = 6(frog), Calc Label = 6(frog), error count = 0
Real Label = 3(cat), Calc Label = 3(cat), error count = 0
Real Label = 1(automobile), Calc Label = 1(automobile), error count = 0
Real Label = 0(airplane), Calc Label = 0(airplane), error count = 0
Real Label = 9(truck), Calc Label = 9(truck), error count = 0
Real Label = 5(dog), Calc Label = 5(dog), error count = 0
Real Label = 7(horse), Calc Label = 7(horse), error count = 0
Real Label = 9(truck), Calc Label = 9(truck), error count = 0
Real Label = 8(ship), Calc Label = 8(ship), error count = 0
Real Label = 5(dog), Calc Label = 5(dog), error count = 0
Real Label = 7(horse), Calc Label = 7(horse), error count = 0
Real Label = 8(ship), Calc Label = 8(ship), error count = 0
Real Label = 6(frog), Calc Label = 6(frog), error count = 0
Real Label = 7(horse), Calc Label = 7(horse), error count = 0
Real Label = 0(airplane), Calc Label = 2(bird), error count = 1
Real Label = 4(deer), Calc Label = 4(deer), error count = 1
Real Label = 9(truck), Calc Label = 9(truck), error count = 1
```

```
Real Label = 5(dog), Calc Label = 4(deer), error count = 2
Real Label = 2(bird), Calc Label = 3(cat), error count = 3
Real Label = 4(deer), Calc Label = 4(deer), error count = 3
Real Label = 0(airplane), Calc Label = 0(airplane), error count = 3
Real Label = 9(truck), Calc Label = 9(truck), error count = 3
Real Label = 6(frog), Calc Label = 6(frog), error count = 3
Real Label = 6(frog), Calc Label = 6(frog), error count = 3
Real Label = 5(dog), Calc Label = 5(dog), error count = 3
Real Label = 4(deer), Calc Label = 4(deer), error count = 3
Real Label = 5(dog), Calc Label = 5(dog), error count = 3
Real Label = 9(truck), Calc Label = 9(truck), error count = 3
Real Label = 2(bird), Calc Label = 3(cat), error count = 4
Real Label = 4(deer), Calc Label = 7(horse), error count = 5
Real Label = 1(automobile), Calc Label = 9(truck), error count = 6
Real Label = 9(truck), Calc Label = 9(truck), error count = 6
Real Label = 5(dog), Calc Label = 5(dog), error count = 6
Real Label = 4(deer), Calc Label = 4(deer), error count = 6
Real Label = 6(frog), Calc Label = 6(frog), error count = 6
Real Label = 5(dog), Calc Label = 5(dog), error count = 6
Real Label = 6(frog), Calc Label = 6(frog), error count = 6
Real Label = 0(airplane), Calc Label = 0(airplane), error count = 6
Real Label = 9(truck), Calc Label = 9(truck), error count = 6
Real Label = 3(cat), Calc Label = 5(dog), error count = 7
Real Label = 9(truck), Calc Label = 9(truck), error count = 7
Real Label = 7(horse), Calc Label = 7(horse), error count = 7
Real Label = 6(frog), Calc Label = 6(frog), error count = 7
Real Label = 9(truck), Calc Label = 9(truck), error count = 7
Real Label = 8(ship), Calc Label = 8(ship), error count = 7
Real Label = 0(airplane), Calc Label = 2(bird), error count = 8
Real Label = 3(cat), Calc Label = 3(cat), error count = 8
Real Label = 8(ship), Calc Label = 8(ship), error count = 8
Real Label = 8(ship), Calc Label = 8(ship), error count = 8
Real Label = 7(horse), Calc Label = 7(horse), error count = 8
Real Label = 7(horse), Calc Label = 7(horse), error count = 8
Real Label = 4(deer), Calc Label = 3(cat), error count = 9
Real Label = 6(frog), Calc Label = 3(cat), error count = 10
Real Label = 7(horse), Calc Label = 7(horse), error count = 10
Real Label = 3(cat), Calc Label = 5(dog), error count = 11
Real Label = 6(frog), Calc Label = 6(frog), error count = 11
Real Label = 3(cat), Calc Label = 3(cat), error count = 11
Real Label = 6(frog), Calc Label = 6(frog), error count = 11
Real Label = 2(bird), Calc Label = 2(bird), error count = 11
Real Label = 1(automobile), Calc Label = 1(automobile), error count = 11
Real Label = 2(bird), Calc Label = 2(bird), error count = 11
Real Label = 3(cat), Calc Label = 3(cat), error count = 11
Real Label = 7(horse), Calc Label = 9(truck), error count = 12
Real Label = 2(bird), Calc Label = 2(bird), error count = 12
Real Label = 6(frog), Calc Label = 6(frog), error count = 12
Real Label = 8(ship), Calc Label = 8(ship), error count = 12
Real Label = 8(ship), Calc Label = 8(ship), error count = 12
Real Label = 0(airplane), Calc Label = 0(airplane), error count = 12
```

```
Real Label = 2(bird), Calc Label = 2(bird), error count = 12
Real Label = 9(truck), Calc Label = 0(airplane), error count = 13
Real Label = 3(cat), Calc Label = 3(cat), error count = 13
Real Label = 3(cat), Calc Label = 2(bird), error count = 14
Real Label = 8(ship), Calc Label = 8(ship), error count = 14
Real Label = 8(ship), Calc Label = 8(ship), error count = 14
Real Label = 1(automobile), Calc Label = 1(automobile), error count = 14
Real Label = 1(automobile), Calc Label = 1(automobile), error count = 14
Real Label = 7(horse), Calc Label = 7(horse), error count = 14
Real Label = 2(bird), Calc Label = 2(bird), error count = 14
Real Label = 5(dog), Calc Label = 7(horse), error count = 15
Real Label = 2(bird), Calc Label = 2(bird), error count = 15
Real Label = 7(horse), Calc Label = 7(horse), error count = 15
Real Label = 8(ship), Calc Label = 8(ship), error count = 15
Real Label = 9(truck), Calc Label = 9(truck), error count = 15
Real Label = 0(airplane), Calc Label = 0(airplane), error count = 15
Real Label = 3(cat), Calc Label = 4(deer), error count = 16
Real Label = 8(ship), Calc Label = 8(ship), error count = 16
Real Label = 6(frog), Calc Label = 6(frog), error count = 16
Real Label = 4(deer), Calc Label = 4(deer), error count = 16
Real Label = 6(frog), Calc Label = 6(frog), error count = 16
Real Label = 6(frog), Calc Label = 6(frog), error count = 16
Real Label = 0(airplane), Calc Label = 2(bird), error count = 17
Real Label = 0(airplane), Calc Label = 0(airplane), error count = 17
Real Label = 7(horse), Calc Label = 7(horse), error count = 17
Classify Score = 83 %.
```

上面的执行流程是这样的，首先输入测试样本数目（1 到 100），由于 DE1 板子 FPGA 端 SDRAM 容量较小，难以加载全部测试数据（10000 张图片），故每次最多装入 100 张图片。之后载入数据到 HPS 内存，然后开始构建 CNN 模型，构建过程中也实现了 Open CL 的初始化。构建完毕，将输入图像依次通过 CNN，得到一系列分类结果，与标签进行对比，统计错误分类个数，计算分类准确率。经过测试，分类准确率达到 83%，与 Caffe 测试结果一致。

经过以上测试，可以得到结论：

（1）使用 Open CL 可以很方便地移植高级语言编写的算法；

（2）CNN 在移植过程中需要考虑实际硬件，定制合适的模型和数据；

（3）Cyclone 5 逻辑资源较少（85K，Open CL kernel 占用了 83%），如果希望进一步提高计算速度，一方面可以选用高性能器件（如 Stratix V、Arria 10），另一方面可以使用 RTL 自己搭建计算系统。

详情请点击：

http://bbs.eeworld.com.cn/thread-463088-1-1.html

4. 用户体验与感想

4.1 SoC 在系统集成方面的优势

以前一直使用 Altera 的 FPGA 器件。大部分情况下使用的是 verilog 自己写逻辑，涉及到需要 cpu 处理的时候，也使用 NIOS II 处理器。现在，Altera 推出了 SoC 器件，SoC 的开发流程和以前的 NIOS II 处理器比起来，有很大的相似之处，但也有了很大的不同。看论坛上各位大神都对 Altera 的各种性能进行了详细的测试，并与其他家相似的产品做了横向比较。我经验尚浅，没有接触过其他家类似的产品。但是因为以前有使用 NIOS II 进行开发的基础，所以今天，我就根据这段时间体验 SoC 的感受，来将 SoC 的开发流程与 RTL + NIOS II 的开发流程进行一个纵向的比较。通过纵向比较来展示 SoC 架构与传统的软核加 RTL 设计相比，有哪些相似之处，存在的优势，以及对于开发人员又提出了怎样的挑战。

首先，通过一个简单的千兆以太网传输的例子来对比两种实现方式实现单纯千兆以太网数据收发的开发流程：

RTL+NIOS II 软核方式：

1. 建立 Quartus II 工程；
2. 建立 Qsys 系统；
3. 在 QSYS 系统中添加三速以太网 IP 核；
4. 在 QSYS 系统中添加发送和接收 DMA 模块；
5. 在 QSYS 中添加其它可选组建；
6 生成 QSYS 系统；
7. 将 QSYS 生成的系统添加到 QuartusII 工程中来；
8. 编译 Quartus II 工程；
9. 在 NIOS II EDS 中建立网络应用的 C 工程，或者直接生成 webserver 模版系统；
10. 编译，调试。

SoC 方式：

1. 编写 Linux 系统的网络应用 C 程序；

2. 编译，调试；

以上为 FPGA 和 CPU（无论是软核 NIOS II 还是 HPS）无任何数据交互的过程的一个开发流程。说的再明白点，RTL+NIOS II 软核方式只是在 FPGA 内部搭建了一个处理器环境，然后实现了千兆以太网功能，实质还是处理器的事儿。SoC 的方式之所以相比 RTL+NIOS II 软核方式少了很多步骤，就是因为这些工作都已经被预先完成了，是以现成的方式提供给客户的。如果以 RTL+NIOS II 软核方式，在加入很多的功能 IP 后，其也能达到 HPS 那样的方便开发的效果，其开发也能像 SoC 的方式一样简单了。因此，单从简单的只使用处理器功能来对比，SoC 没有占到太多优势，可能唯一的优势就是 HPS 能够运行 linux 系统，而且性能比 NIOS II 要好吧。

上面说完了单纯的使用 NIOS II 或 HPS 实现千兆以太网传输，不涉及与 FPGA 进行交互，因此没能完全体现 SoC 架构的优势。

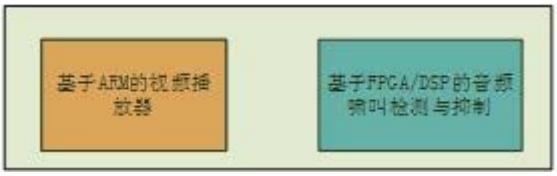
现在以另外一个卡拉 OK 系统的实际的例子来说明 SoC 在系统集成方面的巨大作用。

该系统需要实现两个主要功能如下：

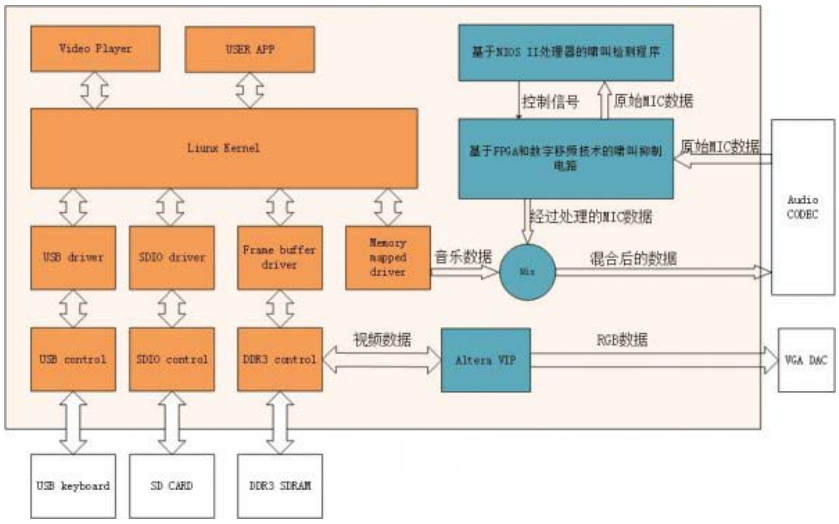
1. 播放视频媒体，如 MP4 格式的视频；

2. 实现扩音系统中存在的啸叫的消除或抑制；

以前这两部分分别由不同的器件完成，视频播放使用 PC 机或基于 ARM 的播放器；扩音系统的啸叫抑制由 DSP 系统通过数字信号处理的方式完成，当然也有使用 FPGA 系统实现的，其实质就是两套系统的拼接。



现在，有了 SoC 后，便可以使用一个 SoC 系统完成这所有的功能。使用 SoC 系统实现该系统的架构如下图所示：



基于数字移频的啸叫检测与抑制中，使用 NIOS II 处理器对麦克风采集到的音频信号进行快速傅里叶变换，以准确识别出啸叫点；使用 NCO 核产生 8Hz 的低频正弦信号，与麦克风采集到的音频信号进行乘法运算，再将所得结果使用 FIR 高通滤波器进行数字滤波，即可实现数字移频。

在这套系统中，充分发挥了 SoC 在系统集成方面的优势，HPS 被用来与视频媒体相关的处理工作，同时通过 FPGA 与 HPS 共享的 DDR3 SDRAM 存储器实现视频数据在显示器上的显示。通过此种结构，将以前需要两个甚至更多个系统配合才能完成的工作使用一块主芯片完成，节约了系统成本。

上述例子中，还是没有完全使用到 FPGA 与 HPS 间高带宽的优势，因此还是有很大的局限性，今天时间有限，暂时先写到这里。过几天再尝试一下用友晶的高速 ADDA 卡实现高速数据采集后送给 HPS 处理的实验，来更进一步体会两者在数据处理方面强大的协同工作能力。祝大家学习进步，工作顺利。

详情请查看：

http://bbs.eeworld.com.cn/thread-457890-1-1.html

4.2 SoCFPGA 之 DSP 进阶 ---NEON 高性能媒体引擎

4.2.1 前言

等等，SoC 不是 ARM 和 FPGA 搞基吗？和 DSP 有什么关系？其实感觉这个题目有点大，觉得自己驾驭不了，不过也是目前楼主思考过后的折中之取，因为目前所做过的工作中和 FPGA 与 DSP 打交道比较多，虽然手中很早就有 ARM 板，但是工作比较忙还是吃灰较多，板子还是别人的好啊。正题开始，从我刚接触电子的时候，从没有想过今后有机会接触到这样一种器件，同时含有 ARM,FPGA,DSP 三大器件的特性，Altera 的 SoC 器件让我们看到了这种可能性。



ARM 与 DSP 的融合好多年前就出来了，TI 的达芬奇系列就是这种结构，以前学生时代用过 TI 的经典板卡 beagleboard-XM，DM3730，达芬奇系列目前算上进入了一个困境，低端的专用用器件方案成本更低，而且 ARM 附带的媒体处理功能越来越强，对 DSP 中低端市场也行成了很大压力，在高端领域中，比如目前炒得火热的 4K 视频，即使是最高端的 davinci 系列也只能望洋兴叹，能满足这种方案的只有高端 FPGA。



然后讲讲 DSP 和 FPGA 吧，目前来说还没看到二种器件融合的产品，当然 FPGA 内部有 DSP 切片，一种是数字信号处理器一种是可编程逻辑，单从字面理解可能初学者很容易区分，但是当 FPGA 或 DSP 在信号处理领域相撞的时候，我想很多刚接触这 2 个器件的都会困惑，到底哪个器件更强，为什么要用 FPGA+DSP，单用 DSP 或 FPGA 不行吗？单从目前使用阶段来讲，秉着公平起见各打 50 大板。

1. 从研发角度来说 DSP 采用高级语言，FPGA 采用硬件描述语言，这加大了在 FPGA 上开发算法的成本，所以一般 DSP 负责复杂的算法。

2. 从结构上来说 DSP 是串行，FPGA 是并行，用 FPGA 来实现数字信号处理可以很好地解决并行性和速度问题。

DSP+FPGA 结构最大的特点是结构灵活，通用性强，同时其开发周期较短，系统易于维护和扩展，适合于实时数据采集和处理。传统 DSP 芯片的结构本质上是串行的，对于需要处理的数据量大、处理速度高，但是运算结构相对比较简单的底层信号处理算法来说，并没有优势可言，而这恰好是 FPGA 硬件的强项。而且针对 DSP 芯片不能直接与 PC 机进行异步串行通行的缺点，可以通过 FPGA 的相关通信协议转换来弥补。再结合 FPGA 的并行处理方式，使得 DSP+FPGA 非常适合超高速和实时信号处理领域。

DSP 和 FPGA 之间的区别主要在于成本、工具可用性、性能和设计灵活性。它们各有其优点和缺点，这取决于设计人员的经验和应用要求。然而，FPGA 领域近期的发展状况显示，DSP 和 FPGA 之间的差距正在缩小。

FPGA 与 DSP：竞合中缩小差距

FPGA 最新的发展成果

目前，最热门的技术在 FPGA 上上演：硬核浮点 DSP、OPENCL、SoC，FPGA 是否会成为武林盟主呢？

从处理响应速度来看，计算的处理响应速度从文本时代的秒级，到多媒体时代的百毫秒级，视频时代的十毫秒级，迅速推进到 5G 时代的 1 毫秒级。对海量数据在 1 毫秒内完成处理是未来 10 年电子系统设计工程师面临的难题，目前芯片工艺已经来到深纳米级别，每个节点的性价比幅度递减，流片成本越来越高，盈利的公司锐减，能存活的只有可编程器件！！！

拿 FPGA 进行 DSP 设计，我们用到的是 FPGA 的内部乘法器切片，而在 SoC 上，我们使用的是 ARM 附送的 NEON SIMD 高性能媒体引擎，同志门在用 SoC 可能把重心都放在 2 个 A9 核上，其实这个小姑娘也能独当一面的，反正 ARM 公司附送的，秉着不用白不用的原则，开始 NEON 学习之旅。

详情请查看：

http://bbs.eeworld.com.cn/thread-461338-1-1.html

4.2.2 之一

本小节对 NEON 做简单介绍

NEON 技术可加速多媒体和信号处理算法(如视频编码 / 解码、2D/3D 图形、游戏、音频和语音处理、图像处理技术、电话和声音合成) , 其性能至少为 ARMv5 性能的 3 倍 , 为 ARMv6 SIMD 性能的 2 倍。

通过干净方式构建的 NEON 技术可无缝用于其本身的独立管道和寄存器文件。

NEON 技术是 ARM Cortex ™ -A 系列处理器的 128 位 SIMD (单指令 , 多数据) 架构扩展 , 旨在为消费性多媒体应用程序提供灵活、强大的加速功能 , 从而显著改善用户体验。它具有 32 个寄存器 , 64 位宽 (双倍视图为 16 个寄存器 , 128 位宽。)

NEON 指令可执行“打包的 SIMD”处理 :

寄存器被视为同一数据类型的元素的矢量

数据类型可为 : 签名 / 未签名的 8 位、16 位、32 位、64 位单精度浮点

指令在所有通道中执行同一操作

使用 NEON 技术的 ARM Cortex ™ -A 系列处理器 , 以及 ARM 的 Mali 多媒体硬件解决方案可用于多媒体应用 , 范围从智能手机和移动计算设备到 HDTV。

NEON 是 ARM 公司的商标 ,也是高端 ARM 处理器集成的硬 IP ,类似 intel 的 MMX、SSE 硬件指令集。

NEON 的特征和优点 NEON 支持用于 Internet 应用程序的范围广泛的多媒体编解码器 :

* 许多软编解码器标准 : MPEG-4、H.264、On2 VP6/7/8、Real、AVS

* 对于各种格式的正常大小的“Internet 流”解码来说 , 是理想的解决方案

* 不仅仅针对编解码器 , 还适用于 2D 和 3D 图形和其他矢量处理

* 提供现有工具、操作系统支持和生态体系支持

所需周期减少 :

*NEON 可使复杂视频编解码器的性能提升 60-150%

* 单个简单 DSP 算法可实现更大的性能提升 (4 倍 -8 倍)

* 处理器可更快进入睡眠状态 , 从而在整体上节约了动态功耗

NEON 技术的大量元素能够提高性能并简化软件开发过程 , 如 :

* 通过对齐和非对齐数据访问 , 可对 SIMD 操作进行有效的矢量化。

* 清晰的指令集架构 , 设计用于自动矢量化编译器和手动编码。

* 有效访问打包数组 , 如 ARGB 或 xyz 坐标

* 支持整数和浮点操作 , 以确保适合从编解码器、高性能计算到 3D 图形等广泛应用领域。

* 与 ARM 处理器紧密结合 , 提供单指令流和内存的统一视图 , 从而能够提供一个具有更简单工具

流的开发平台目标。nbsp;

* 通过具有双 128 位 /64 位视图的大型 NEON 寄存器文件，可有效处理数据并尽可能减少对内存的访问，从而增加了数据吞吐量。

如何使用 NEONOpenMAX DL 库：

* 加速 AV 编解码器的建议方法

* 以源格式释放的库，在 ARM 网站上免费提供

* 支持以下格式：MPEG-4 简单配置文件、H.264 基准、JPEG、MP3、AAC

* 支持以下功能：FIR、IIR、FFT、点积、色彩空间转换、de-blocking.de-ringing、旋转、缩放、合成

矢量化编译器：

* 使用现有源代码自动搜索 NEON SIMD

* 受 ARM RealView 开发套件（v3.1 Pro 及更高版本）支持

* 在 2007q3 及更高版本中受 gcc 支持

C 内部函数：

*C 函数调用接口至 NEON 操作

* 支持 NEON 支持的所有数据类型和操作

* 在 ARM RealView 开发套件（3.1 及更高版本）和 gcc 2007q3 及更高版本中受支持

汇编器：

* 针对确实需要在最低级别进行优化的用户

* 在 ARM 的 RealView 开发套件（3.1 及更高版本）和 gcc 2007q3 及更高版本中受支持

开源社区中的 NEON 支持

当前，在以下开源项目中支持 NEON：

*Android – NEON 优化

使用 NEON，Skia 库 S32A_D565_Opaque 的速度加快了 5 倍

*Ubuntu 09.04 支持 NEON：

关键共享库的 NEON 版本

*Bluez – 官方 Linux 蓝牙协议堆栈

NEON SBC 音频编码器

*Pixman（Cairo 2D 图形库的一部分）

合成 /alpha 混合

X.Org、Mozilla Firefox、Fennec 和 Webkit 浏览器

例如，使用 NEON 后，fbCompositeSolidMask_nx8x0565neon 的速度提高了 8 倍

*ffmpeg - libavcodec

用于众多 Linux 分发版的 LGPL 媒体播放器

视频：MPEG-2、MPEG-4ASP、H.264 (AVC)、VC1

音频：Ogg Vorbis

* x264 – Google 2009 年度编程之夏

GPL h.264 编码器 – 例如，针对视频会议

DSP 扩展

ARM DSP 指令集扩展增加了高性能应用中 ARM 解决方案的 DSP 处理能力，同时通过便携式、电池电源设备提供所需的低能耗。DSP 扩展已经过优化，适用于众多软件应用（包括伺服马达控制、Voice over IP (VOIP) 和视频 / 音频编解码器），其中此扩展可增强 DSP 性能，使其能够有效处理所需任务。

特点

* 单周期 16x16 和 32x16 MAC 实现

* 与基于 ARM7 ™ 处理器的 CPU 产品相比，DSP 性能提高了 2-3 倍

* 零开销饱和扩展支持

* 用于加载和存储寄存器对的新指令，包含增强的寻址模式

* 新的 CLZ 指令改进了算术运算标准化，提高了除法性能

* 在 ARMv5TE、ARMv6 和 ARMv7 体系结构中完全受支持

应用

* 音频编码 / 解码（MP3：AAC、WMA）

* 伺服马达控制 (HDD/DVD)

*MPEG4 解码

* 语音和手写识别

* 嵌入式控制

* 位准确算法 (GSM-AMR)

用于 ARM 架构的编译器可以使用这些 DSP 扩展来改进标准 C 和 C++ 软件的代码生成过程，或

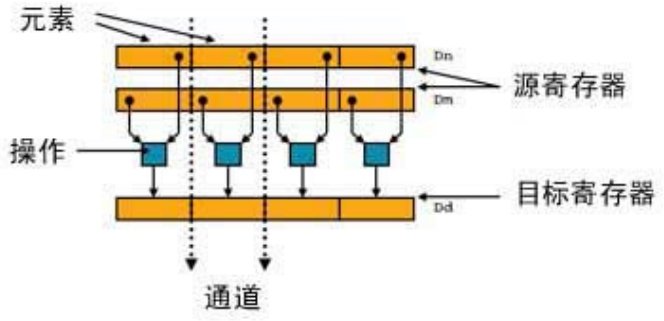
者允许软件开发人员要求通过内部函数或内联汇编代码显式使用这些扩展。

ARM DSP 扩展改进了 DSP 性能，且无需非常高的时钟频率。几乎不增加典型实现中的功耗即可获得此性能。DSP 扩展广泛应用于智能手机以及需要大量信号处理的类似嵌入式系统，从而避免使用其他硬件加速器。DSP 扩展可与 32 位 ARM 和 16 位 Thumb® 指令集完全兼容，从而确保所有现有操作系统和应用程序代码都可在支持 DSP 且基于 ARM 处理器的设备上重用。这些扩展广泛适用于大量细分市场，包括无线、大容量存储、汽车、消费娱乐和数字图像。

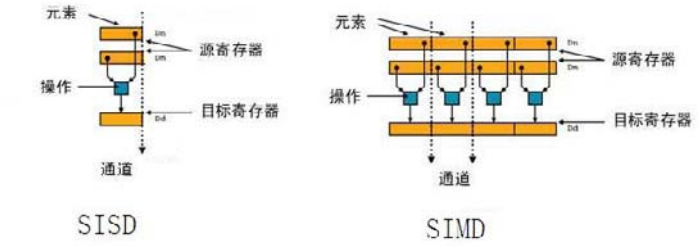
NEON 原理

NEON 指令可执行“打包的 SIMD”处理：

寄存器被视为同一数据类型的元素的矢量数据类型可以为：签名 / 未签名的 8 位、16 位、32 位、64 位单精度浮点 原理图如下：



下图通过 SISD（单指令单数据）和 SIMD（单指令多数据）的对比，来说明 NEON 的工作原理：



从图上可以看到，对于 SISD，每个指令只能处理一个数据，而 SIMD 一个指令可以处理多个数据，因为多个数据的处理是平行的，因此从时间来说，一个指令执行的时间，SISD 和 SIMD 是差不多的。由于 SIMD 一次可以处理 N 个数据，所以它的处理的时间也就缩短到 SISD 的 1/N。

需要指出一点，NEON 是需要硬件的支持的，需要有一块寄存器放到硬件上来处理这个的。

NEON 的使用

NEON 的函数是 C 语言风格，但是编译后成为汇编语句，这样执行的效率会提高。

NEON 代码的计算效率很高，举例 1 为实现信号增益的代码实例。

举例 1

标准 C 语言：

```
for ( i=0;i<BUFFERLENGTH;i++)
```

```
{
  output=input;
}
```

上述代码每次迭代仅能处理单一的值，通过这个展开循环可以进一步提高效率。

```
 for ( i=0;i<BUFFERLENGTH;i+=4)
{
  output=input;
  output[i+1]=input[i+1];
  output[i+2]=input[i+2];
  output[i+3]=input[i+3];
}
```

这样可以减少系统资源开销，提高性能，增加并发代码可能性。可是，乘法指令循环复杂的问题仍然存在，使用 NEON 内部函数，代码看起来会截然不同。

```
for ( i=0;i<BUFFERLENGTH;i+=4)
{
  // 加载
  samples_f32=vldlq_f32(input+i);
  // 执行乘法运算
  samples_f32=vmulq_f32(samples_f32,1.2);
  // 储存结果
  samples_f32=vstlq_f32(output+i,samples_f32);
}
```

例 2

下面是一个用 C 现实的将 GRB 转化成灰度图的算法

```
void reference_convert (uint8_t * __restrict dest, uint8_t * __restrict src, int n)
{
  int i;
  for (i=0; i<n; i++)
  {
    int r = *src++; // load red
    int g = *src++; // load green
    int b = *src++; // load blue

    // build weighted average:
    int y = (r*77)+(g*151)+(b*28);

    // undo the scale by 256 and write to memory:
    *dest++ = (y>>8);
  }
}
```

下面是一个用 NEON 现实的将 GB 转化成灰度图的算法

```
void neon_convert (uint8_t * __restrict dest, uint8_t * __restrict src, int n)
{
  int i;
  uint8x8_t rfac = vdup_n_u8 (77);
  uint8x8_t gfac = vdup_n_u8 (151);
  uint8x8_t bfac = vdup_n_u8 (28);
  n/=8;

  for (i=0; i<n; i++)
  {
    uint16x8_t  temp;
    uint8x8x3_t rgb  = vld3_u8 (src);
    uint8x8_t result;

    temp = vmull_u8 (rgb.val[0],      rfac);
    temp = vmlal_u8 (temp,rgb.val[1], gfac);
    temp = vmlal_u8 (temp,rgb.val[2], bfac);

    result = vshrn_n_u16 (temp, 8);
    vst1_u8 (dest, result);
    src  += 8*3;
    dest += 8;
  }
}
```

对以上函数进行说明如下：

```
uint8x8_t rfac = vdup_n_u8 (77);
uint8x8_t gfac = vdup_n_u8 (151);
uint8x8_t bfac = vdup_n_u8 (28);
```

把权重放入 NEON 寄存器

```
uint8x8x3_t rgb  = vld3_u8 (src);
```

一次入 8 pixls 三个 NOEN 寄存器，这个地方是重点可以说明 NEON 的高效率。

```
temp = vmull_u8 (rgb.val[0],      rfac);
temp = vmlal_u8 (temp,rgb.val[1], gfac);
temp = vmlal_u8 (temp,rgb.val[2], bfac);
```

计算结果

```
result = vshrn_n_u16 (temp, 8);

vst1_u8 (dest, result);
```

详情请查看：

http://bbs.eeworld.com.cn/thread-461459-1-1.html

4.2.3 之二

Neon 函数在头文件 arm_neon.h 中已经被定义，头件既定义内嵌函数，也定义一组向量类型，编译器将 NEON 内嵌函数视为普通函数调用。



基本的 DSP 处理模块

例程 1 实现从 0-99 的加法运算作为使用 NEON 函数的基本 DSP 处理模块。

```
--------------------------------------------------------------------------
#include<stdio.h>
#include<arm_neon.h>
void fill_array(int16_t *array,int size)
{
   int i;
   for(i=0;i<size;i++)
 {
     array=i;
 }
 }
int sum_array(int16_t *array,int size)
{
   int16_t *i;
   int16x4_t acc=vdup_n_s16(0);
int32x2_t acc1;
int64x1_t acc2;
for(i=array;i<array+size;i+=4)

                        {
```

```
                int16x4_t vec;
                vec=vld1_s16(i);
                acc=vadd_s16(acc,vec);
}
                 acc1=vpaddl_s16(acc);
                 acc2=vpaddl_s32(acc1);
                 return (int)vget_lane_s64(acc2,0);
}
int main()
{
 int16_t my_array[100];
 fill_array(my_array,100);
printf(“sum was %d/n”,sum_array(my_array,100));
 return 0;
}
```

例程 2 浮点向量运算，进行简单的浮点向量计算。VFP 除了对浮点数基本加减乘除，开方，比较，取反提供支持之外，其向量功能能够使它同时支持最多 8 组单精度、4 组双精度浮点运算。

```
-------------------------------------------------------------------------------------------
#include<stdio.h>
#include<arm_neon.h>
int main()
{
    float a[4]={2,4,8,16};
float b[4]={2,4,8,16};
    float c[4]={0,0,0,0};
   float32x4_t sample1,sample2,result;
   sample1=vldlq_f32(a);
sample2=vldlq_f32(b);
    result=vmulq_f32(sample1,sample2);
 vst1q_f32(c,result);
    printf(“Result:%f,%f,%f,%f”,c[0],c[1],c[2],c[3]);
    return 0;
}
```

例程 3 矩阵运算，用 NEON 优化矩阵乘法的实例，介绍如何用 NEON 有效地处理一个 4X4 的矩阵乘法运算，经常用于 3D 图形，我们认为这些矩阵在内存中是以列为主排列的。假设每列元素在 NEON 寄存器中表示为一个向量，那么矩阵乘法就是向量乘以标量的运算，而后续的累加也同样可以用向量乘以标量的累加指令实现。

```
#include<stdio.h>
#include<arm_neon.h>
int main()
{
          int i;
    uint16x4_t vectorT1,vectorT2,vectorT3,vectorT4;
    uint16x4_t A[4][4]={ 1,2,3,4,
                                  5,6,7,8,
                               9,10,11,12,
                               13,14,15,16};
uint16x4_t B[4][4]={16,15,14,13,
                                12,11,10,9,
                                8,7,6,5,
                                4,3,2,1};
        uint16_t C[4][4];
        uint16x4_t vectorB1,vectorB2,vectorB3,vectorB4;
        vectorB1=vld1_u16(b[0]);
           vectorB2=vld1_u16(b[1]); 搜索

           vectorB3=vld1_u16(b[2]);
         vectorB4=vld1_u16(b[3]);
                  for(i=0;i<4;i++)
                 {
                           vectorT1=vmul_n_u16( vectorB1,A[0]);
                           vectorT2=vmul_n_u16( vectorB2,A[1]);
                           vectorT3=vmul_n_u16( vectorB3,A[2]);
                           vectorT4=vmul_n_u16( vectorB4,A[3]);

                    vectorT1=vadd_u16( vectorT1, vectorT2);
                    vectorT1=vadd_u16( vectorT1, vectorT3);
                    vectorT1=vadd_u16( vectorT1, vectorT4);
                    vst1_u16(C,vectorT1);
                          printf("C[%d][0]=%ld\n",i,C[0]);
                          printf("C[%d][0]=%ld\n",i,C[1]);
                          printf("C[%d][0]=%ld\n",i,C[2]);
                          printf("C[%d][0]=%ld\n",i,C[3]);
            }
                return 0;
}
```

附录： ARM 和 NEON 指令

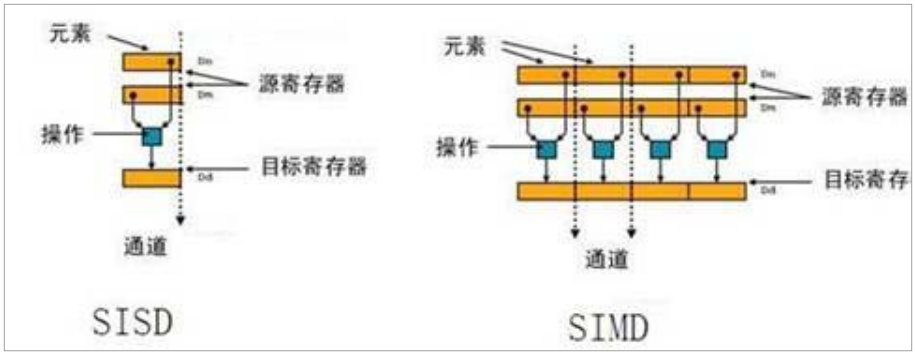
在移动平台上进行一些复杂算法的开发，一般需要用到指令集来进行加速。目前在移动上使用最多的是 ARM 芯片。

ARM 是微处理器行业的一家知名企业，其芯片结构有：armv5、armv6、armv7 和 armv8 系列。芯片类型有：arm7、arm9、arm11、cortex 系列。指令集有：armv5、armv6 和 neon 指令。关于 ARM 到知识参考：http://baike.baidu.com/view/11200.htm

最初的 ARM 指令集为通用计算型指令集 ,指令集都是针对单个数据进行计算 ,没有并行计算到功能。随着版本的更新，后面逐渐加入了一些复杂到指令以及并行计算到指令。而 NEON 指令是专门针对大规模到并行运算而设计的。

NEON 技术可加速多媒体和信号处理算法(如视频编码 / 解码、2D/3D 图形、游戏、音频和语音处理、图像处理技术、电话和声音合成），其性能至少为 ARMv5 性能的 3 倍，为 ARMv6 SIMD 性能的 2 倍。

关于 SIMD 和 SISD : Single Instruction Multiple Data ,单指令多数据流。之 SISD 是单指令单数据。以加法指令为例，单指令单数据（ SISD ）的 CPU 对加法指令译码后，执行部件先访问内存，取得第一个操作数；之后再一次访问内存，取得第二个操作数；随后才能进行求和运算。而在 SIMD 型的 CPU 中，指令译码后几个执行部件同时访问内存，一次性获得所有操作数进行运算。这个特点使 SIMD 特别适合于多媒体应用等数据密集型运算。如下图所示：



如何才能快速到写出高效的指令代码？这就需要对各个指令比较熟悉，知道各个指令的使用规范和使用场合。

ARM 指令有 16 个 32 位通用寄存器 ,为 r0-r15 ,其中 r13 为堆栈指针寄存器 ,r15 为指令计算寄存器。实际可以使用的寄存器只有 14 个。r0-r3 一般作为函数参数使用，函数返回值放在 r0 中。若函数参数超过 4 个，超过到参数压入堆栈。

有效立即数的概念：每个立即数采用一个 8 位的常数 (bit[7:0]) 循环右移偶数位而间接得到，其中循环右移的位数由一个 4 位二进制 (bit[11:8]) 的两倍表示。如果立即数记作 <immediate>，8 位常数记作 immed_8，4 位的循环右移值记作 rotate_imm，有效的立即数是由一个 8 位的立即数循环右移偶数位得到，可以表示成：

<immediate>=immed_8 循环右移 (2×rotate_imm)

如：mov r4 , #0x8000 000A　#0x8000 000A 由 0xA8 循环右移 0x2 位得到。

下面介绍一些比较常用到一些指令。

内存访问指令：

LDR 和 STR，有三种方式，比较容易搞混

LDR r0, [r1, #4]　r0 := mem[r1+4]　，#4 是直接偏移量，这时候只能在正负 4Kb 到范围内。也可以是寄存器偏移，用 +/- 表示。记住 r1 不进行偏移。

LDR r0, [r1, #4]!　r0 :=mem[r1+4]，r1 := r1 + 4，取值是取偏移量到值，并且 r1 进行偏移。

LDR r0, [r1], #4　r0 :=mem[r1]　，r1 := r1 +4，取值是取 r1 地方到值，取值后进行偏移。运算后自动加 4，后变址。

另外：LDRB 是无符号字节，SB 是有符号字节，H 无符号半字，SH 有符号半字。

存储器和寄存器数据交换：SWP，SWPB

如 SWP r0, r1, [r2]　r0 := mem[r2]，mem[r2] := r1

多寄存器数据传输：

LDMIA r1, {r0,r2,r5}　r0 = mem[r1], r2 = mem[r1+4], r5=mem[r1+8]

通用数据处理指令

第二操作数，常用到有 LSR，LSL 等，如 mov r1, r2, lsl #2 将 r2 左移 2 位然后赋值到 r1 中。

常用到操作有 ADD、SUB、AND、ORR、EOR、BIC、RN，如果加上了 S 则会更新条件标记。

MOV 移动，MVN 取反移动。MOV 可以是 R 寄存器，立即数以及接第二操作数。

REV：在字或半字内反转字节或位到顺序

MUL、MLA 和 MLS，乘法、乘加和乘减。MLA R1，R2，R3，R4 表示 R1=R2*R3+R4，还有有符号和无符号乘法等。

跳转指令

B：无条件跳转，BL：带链接到跳转，BX 跳转并交换指令集等。

BAL	Always	Always take this branch
BNE	Not equal	Comparison not equal or zero result
BMI	Minus	Result minus or negative
BEQ	Equal	Comparison equal or zero result
BHS	Higher or same	Unsigned comparison gave higher or same
BCC	Carry clear	Arithmetic operation did not give carry-out
BLT	Less than	Signed integer comparison gave less than
BPL	Plus	Result positive or zero
BLO	Lower	Unsigned comparison gave lower
BGE	Greater or equal	Signed integer comparison gave greater or equal
BGT	Greater than	Signed integer comparison gave greater than
BVS	Overflow set	Signed integer operation ; overflow occurred
BVC	Overflow clear	Signed integer operation ; no overflow occurred
BCS	Carry set	Arithmetic operation gave carry-out
BLS	Lower or same	Unsigned comparison gave lower or same
BLE	Less or equal	Signed integer comparison gave less than or equal

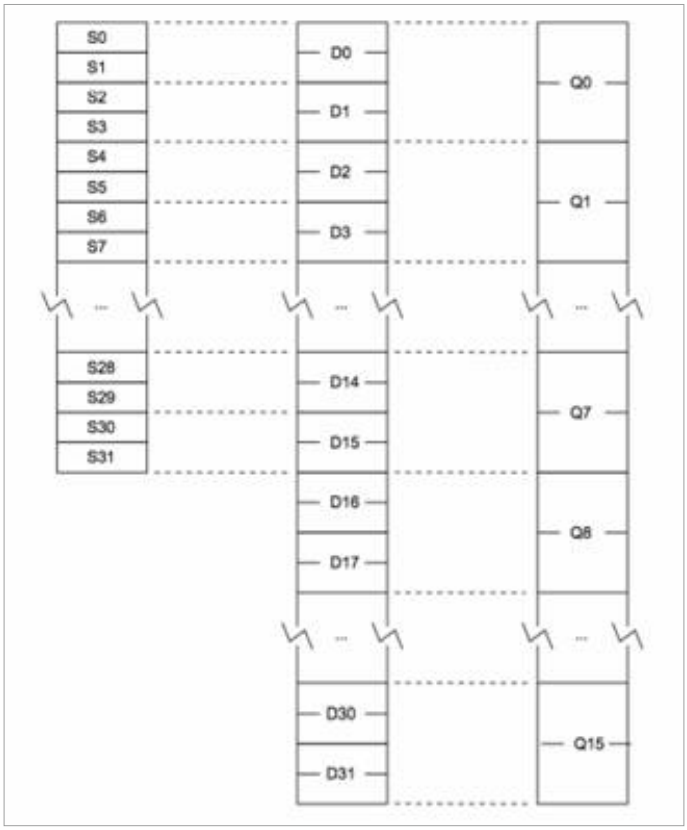
重点介绍一下 NEON 指令，目前使用较多。而且用难度也较大，很多文档上都没有比较详细到介绍，也没有给出相应到例子或者图示。

一、NEON 基本知识

NEON 的寄存器：

有 16 个 128 位四字到寄存器 Q0-Q15，3 个 64 位双子寄存器 D0-D31，两个寄存器是重叠的，在使用到时候需要特别注意，不小心就会覆盖掉。如下图所示：

两个寄存器的关系：Qn =D2n 和 D2n+1，如 Q8 是 d16 和 d17 的组合。



NEON 的数据类型：

注意数据类型针对到时操作数，而不是目标数，这点在写的时候要特别注意，很容易搞错，尤其是对那些长指令宽指令的时候，因为经常 Q 和 D 一起操作。

	8 位	16 位	32 位	64 位
无符号整数	U8	U16	U32	U64
有符号整数	S8	S16	S32	S64
未指定类型的整数	I8	I16	I32	I64
浮点数	不可用	不可用	F（或 F32）	不可用
{0,1} 上的多项式	P8	P16	不可用	不可用

NEON 中的正常指令、宽指令、窄指令、饱和指令、长指令

正常指令：生成大小相同且类型通常与操作数向量相同到结果向量

长指令：对双字向量操作数执行运算，生产四字向量到结果。所生成的元素一般是操作数元素宽度到两倍，并属于同一类型。L 标记，如 VMOVL。

宽指令：一个双字向量操作数和一个四字向量操作数执行运算，生成四字向量结果。W 标记，如 VADDW。

窄指令：四字向量操作数执行运算，并生成双字向量结果，所生成的元素一般是操作数元素宽度的一半。N 标记，如 VMOVN。

饱和指令：当超过数据类型指定到范围则自动限制在该范围内。Q 标记，如 VQSHRUN

二、NEON 指令

NEON 指令较多，下面主要介绍一些常见的指令用法。

复制指令：

VMOV：
两个 arm 寄存器和 d 之间
vmov d0, r0, r1：将 r1 的内容送到 d0 到低半部分，r0 的内容送到 d0 到高半部分
vmov r0, r1, d0：将 d0 的低半部分送到 r0，d0 的高半部分内容送到 r1
一个 arm 寄存器和 d 之间
vmov.U32 d0[0], r0：将 r0 的内容送到 d0[0] 中，d0[0] 指 d0 到低 32 位
vmov.U32 r0, d0[0]：将 d0[0] 的内容送到 r0 中
立即数：
vmov.U16 d0, #1：将立即数 1 赋值给 d0 的每个 16 位
vmov.U32 q0, #1：将立即数 1 赋值给 q0 的每个 32 位
长指令：VMOVL：d 赋值给 q
vmovl.U16 q0, d0：将 d0 的每个 16 位数据赋值到 q0 的每个 32 位数据中
窄指令：VMOVN：q 赋值给 d
vmovn.I32 d0, q0：将 q0 的每 32 位数据赋值到 q0 的每 16 位数据中
饱和指令：VQMOVN 等，饱和到指定的数据类型
vqmovun.S32 d0, q0：将 q0 到每个 32 位移动到 d0 中到每个 16 位中，范围是 0-65535
VDUP：
VDUP.8 d0, r0：将 r0 复制到 d0 中，8 位
VDUP.16 q0, r0：将 r0 复制到 q0 中，16 位
VDUP.32 q0, d2[0]：将 d2 的一半复制到 q0 中
VDUP.32 d0, d2[1]：将 d2 的一半复制到 d0 中
注意是 vdup 可以将 r 寄存器中的内容复制到整个 neon 寄存器中，不能将立即数进行 vdup，立即数只能用 vmov
逻辑运算：

VADD：按位与；VBIC：位清除；VEOR：按位异或；VORN：按位或非；VORR：按位或

移位指令：

VSHL：左移、VSHLL：左移扩展、VQSHL：左移饱和、VQSHLU：无符号移饱和扩展

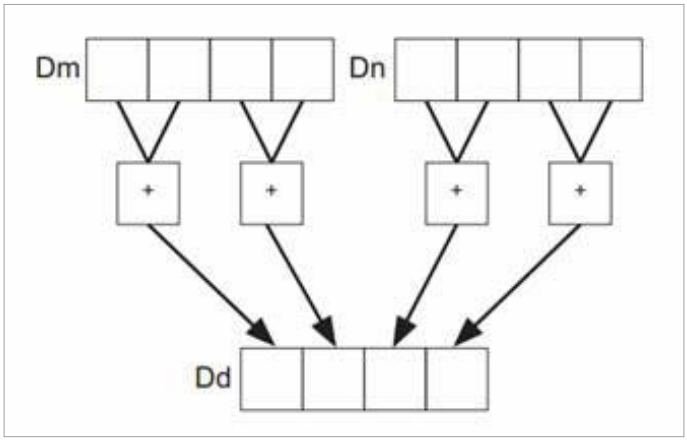
VSHR：右移、VSHRN：右移窄、VRSH：右移舍入、VQSHRUN：无符号右移饱和舍入

通用算术指令：

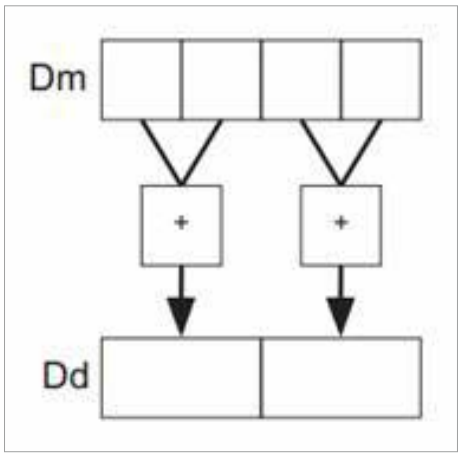
VABA：绝对值累加、VABD：绝对值相加、VABS：绝对值、VNEG 求、VADD、VADDWVADDL、VSUB、VSUL、VSUBW：加减

VADD：将两个向量的相邻元素相加

如 VPADI16 {d2}, d0, d1



VPADDL：VPADDL.S16 d0, d1

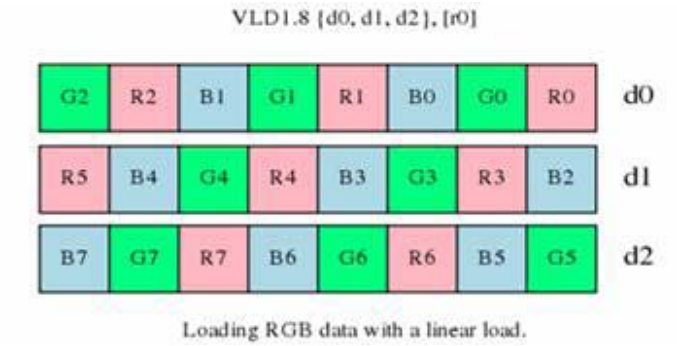


VMAX：最大值，VMIN：最小值

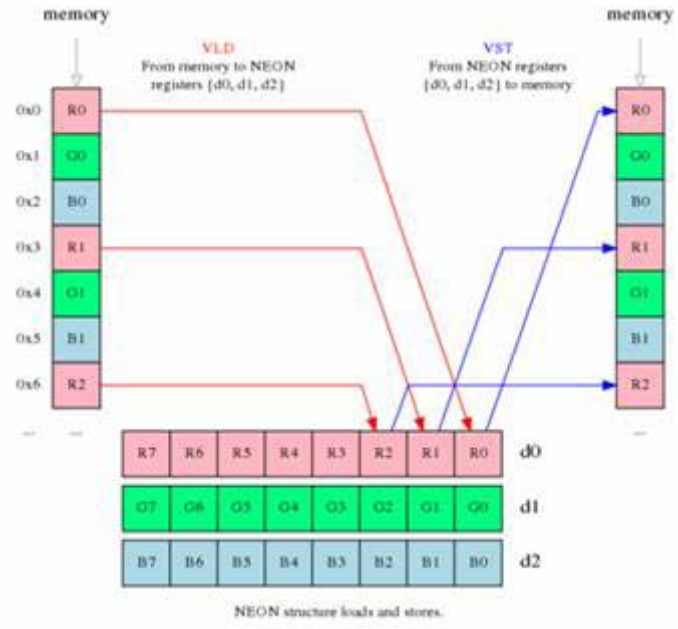
VMUL、VMULL、VMLA（乘加）、VMLS（乘减）、

加载存储指令：

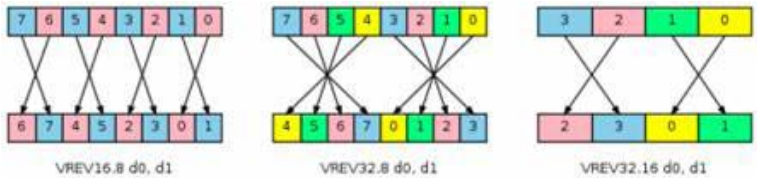
VLD 和 VST



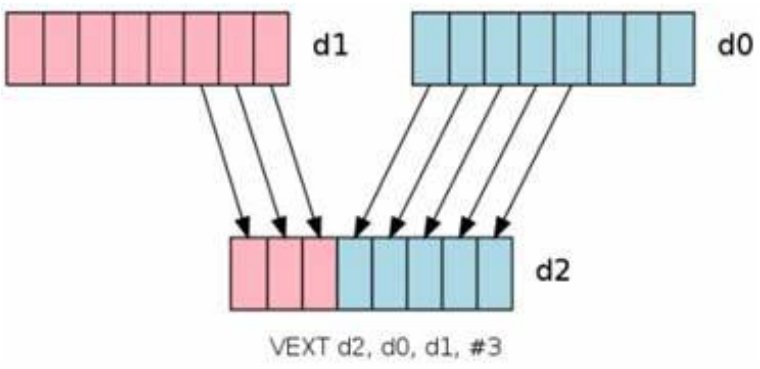
交叉存取的示意图：



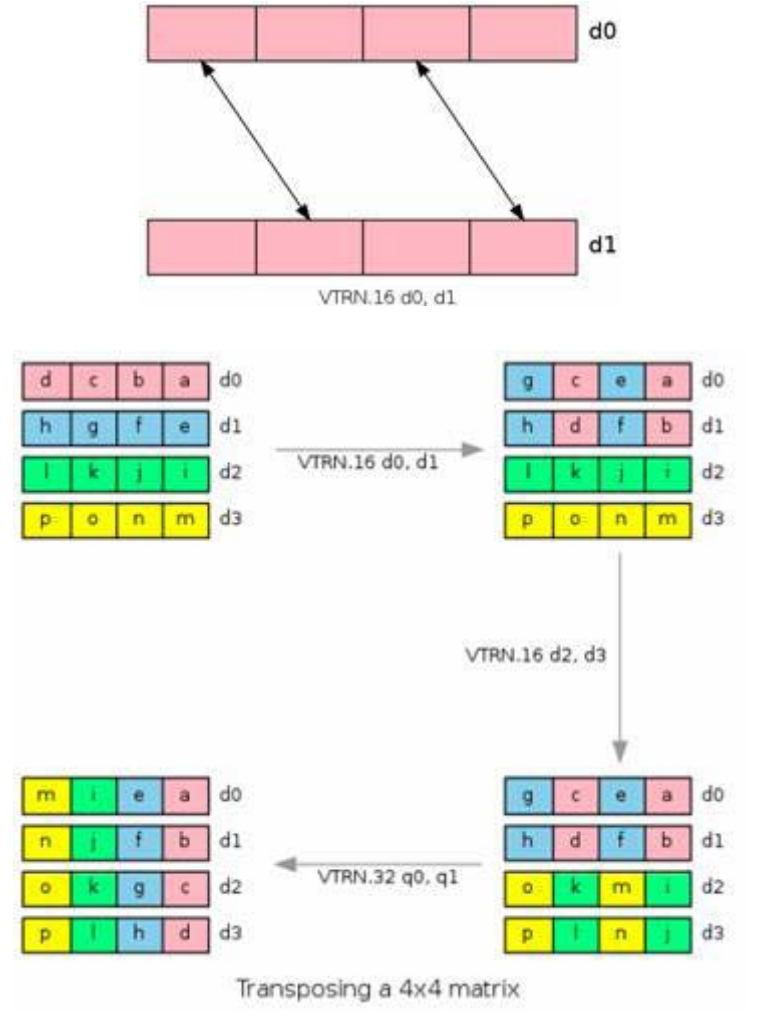
VREV 反转元素指令：



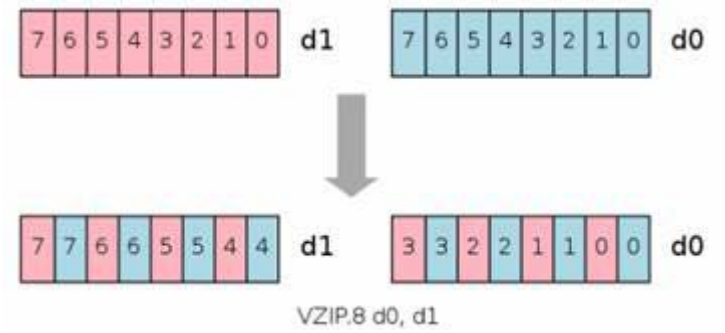
VEXT 移位指令：



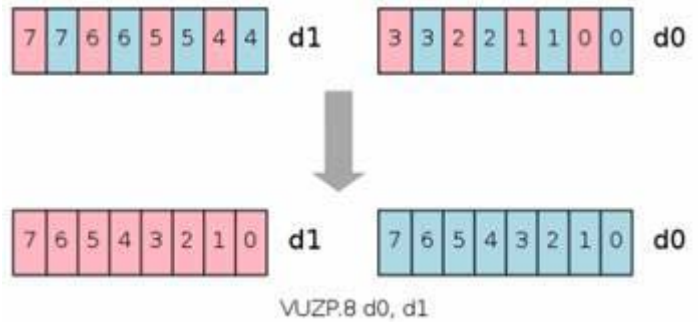
VTRN 转置指令：可以用于矩阵的转置



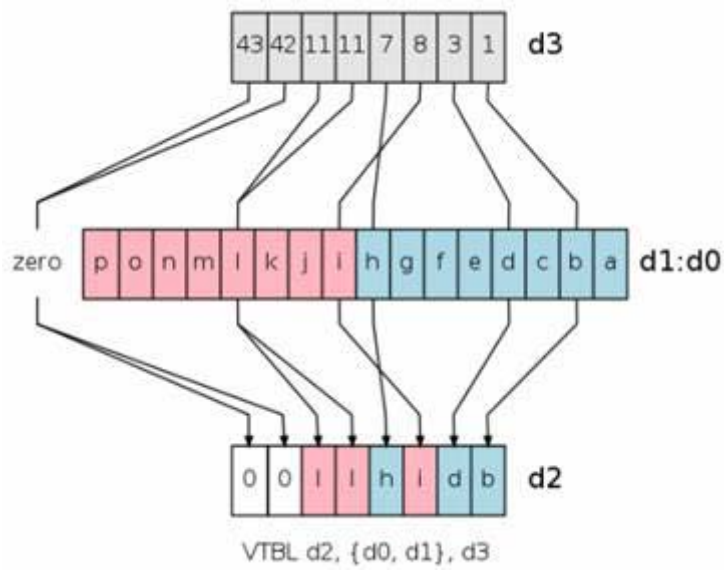
VZIP 指令：压缩，类似交叉存取



VUZP 指令：解压操作，类似交叉存取



VTBL 查表指令：从 d0，d1 中查找 d3 中的索引值，如果找到则取出，没有找到则为 0，存入 d2 中



三、需要注意的地方

load 数据的时候，第一次 load 会把数据放在 cache 里面，只要不超过 cache 的大小，下一次 load 同样数据的时候，则会比第一次 load 要快很多，会直接从 cache 中 load 数据，这样在汇编程序设计的时候是非常需要考虑的问题。

如：求取一个图像的均值，8*8 的窗口，先行求和，然后列求和出来均值，这时候会有两个函数，数据会加载两遍，如果按照这样去优化的话则优化不了多少。如果换成上面这种思路，先做行 16 行，然后再做列，这样数据都在 cache 里面，做列的时候 load 数据会很快。

在做 neon 乘法指令的时候会有大约 2 个 clock 的阻塞时间，如果你要立即使用乘法的结果，则就会阻塞在这里，在写 neon 指令的时候需要特别注意。乘法的结果不能立即使用，可以将一些其他的操作插入到乘法后面而不会有时间的消耗。

如：vmul.u16 q1, d3, d4

vadd.u32 q1, q2, q3

此时直接使用乘法的结果 q1 则会阻塞，执行 vadd 需要再等待 2 个 clock 的时间

使用饱和指令的时候，如乘法饱和的时候，在做乘法后会再去做一次饱和，所以时间要比直接做乘法要慢。

如： vmul.u16 q1, d3, d4

vqmul.u32 q1, q2, q3

后一个的时间要比第一个的时间要久。

在对 16 位数据进行 load 或者 store 操作的时候，需要注意的是字节移位。比如是 16 位数据，则 load 8 个 16 位数据，如果指定寄存器进行偏移，此时需要特别注意。

例如：vld1.64 {d0}, [r0], r1

详情请查看：

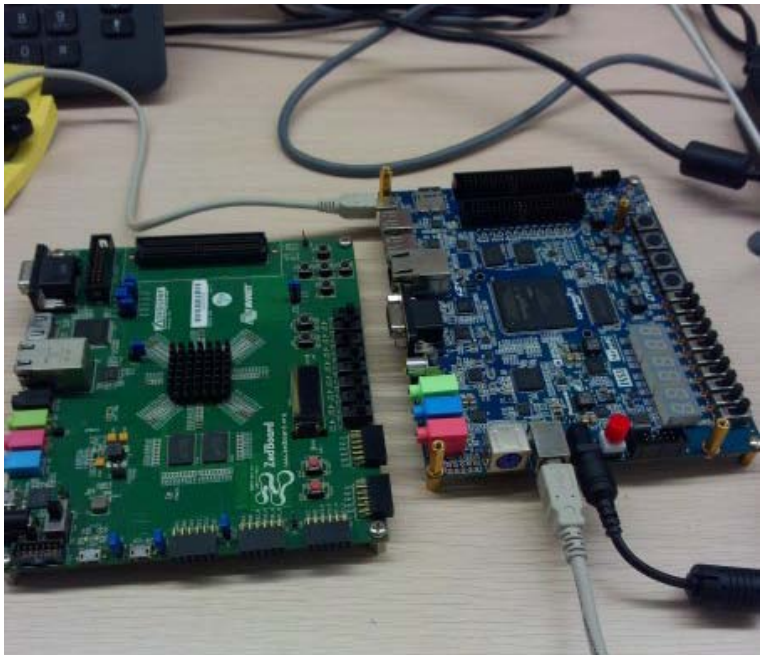
http://bbs.eeworld.com.cn/thread-462935-1-1.html

4.3 SoC 之从用户的角度谈谈使用体验

2012 年参加了 Xilinx 举办的 Open HW 2012 嵌入式设计大赛，得到一块 Zynq 7000 SoC 开发板，当时 Altera 还迟迟没有发布 Cyclone V SoC，而 Xilinx 已经大张旗鼓地宣传好一阵 All Programmable 概念了。

先上传两张照片，对比 Xilinx Zed Board 和 Altera DE1-SoC Board

正面



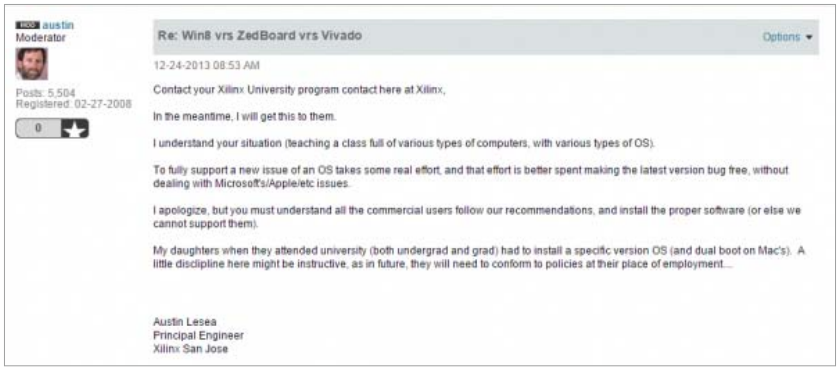
背面



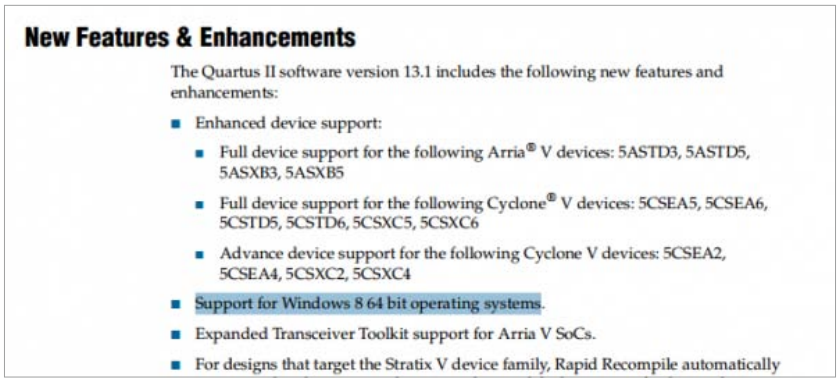
从板卡颜色来看，ZedBoard 是绿漆板，也就是生产 PCB 时最便宜的那种，而 DE1-SoC 是蓝漆板，属于较为高档的工艺。价格么，自行淘宝一下，发现 ZedBoard 价格居然是 DE1-SoC 的两倍！至于性能呢，@chenzhufly 之前的帖子 中有详细的对比参数，大家可以发现内部资源其实不相上下。这也是 Altera 的后发优势吧！

接着说一下软件使用。

我在 2012 年参加 Open HW 时使用的是自己的笔记本进行开发，当时安装的是 Windows 8 + Fedora 17 双系统，比较坑爹的是 Xilinx 软件对 Win8 支持非常差，折腾了半天放弃了，改用实验室电脑 Windows XP 开发。我们也可以从 Xilinx 的官方论坛看到一些吐槽帖子，比如这个。Xilinx 官方给出的解释图如下：



作出了很无奈的解释。大家注意时间是 2013 年 12 月 24 号（圣诞节前一天，估计急着休假去呢，哈哈）。而在差不多同一时期，我们看看 Altera 的软件做到了什么程度。大家可以看这个 Quartus II 13.1 软件发布文档。我把关键位置图如下：



注意到这个文档的发布时间为 2013 年 11 月，已经宣称支持 Windows 8 了，而且 Cyclone V, Arria V 器件也纷纷发布，进度明显比 Xilinx 快很多。我想从一个用户角度出发，首先考虑的是芯片的性价比，其次考虑的是开发环境、技术支持，而在这两个方面，无疑 Altera 做的更好些。

本文仅代表个人观点，且本人与 Altera、Xilinx 没有任何利益关系，如有言语失当，欢迎拍砖。

详情请查看：

http://bbs.eeworld.com.cn/thread-457784-1-1.html

4.4 SoC 之从 GPU 程序员的角度谈谈使用体验

说起 GPU，大家第一反应就是打游戏。是的，GPU 全称是 Graphics Processing Unit，即图形处理单元，是计算机系统中专门负责图形图像绘制的硬件加速器。在没有 GPU 之前，计算机上显示图形界面是非常昂贵的，因为需要 CPU 做大量重复性的工作。

随着 GPU 的进一步发展，计算能力与日俱增，远远超越了同时代的 CPU，于是很多学者提出了利用 GPU 进行通用计算的想法（GPGPU），后面经过 GPU 厂家的合纵连横，最终形成了以 CUDA 为代表的统一着色器架构，可以由用户控制 GPU 进行图形图像之外的通用并行计算。

OpenCL 则是更为通用的异构并行计算模型，不仅支持 CPU + GPU 的计算架构，还支持诸如多核 CPU、CPU + DSP、CPU + FPGA 等特定架构。

今天，我想从一个 GPU 程序员的角度谈谈这次使用 Altera SoC 的体验。

作为 GPU 程序员，首先要熟悉一种 GPU 计算语言，我之前是从 CUDA 开始，后面接触了 OpenCL。从效率上看，CUDA 远远超越 OpenCL；但从可移植性上讲，OpenCL 完虐 CUDA，可以这么说，使用 OpenCL 开发的 kernel 程序，可以一字不改地运行在上面所述的 CPU+GPU、多核 CPU、CPU+DSP、CPU+FPGA 等平台上。也就是说，真正做到了一次编写，处处运行（好像是 Java 的广告词）。随着硬件厂商如 Intel, AMD，NVIDIA，ARM, Xilinx, Altera 等对 OpenCL 的大力支持，后续性能方面也可能全面赶超 CUDA。

我们从一个最常用的 GEMM（矩阵乘）例子开始，分别在 CPU + GPU 平台、Altera SoC（ARM + FPGA）平台上运行同一个 OpenCL 程序。

OpenCL 中，为了便于平台管理，提供了一系列 API 函数，可以便于用户在异构计算平台选择合适的设备。打开 <CL/cl.h> 就能看到这些 API 和详细的参数。在我的工程中，为了方便管理，采用 C++ 面向对象的方式将这些 API 做了封装，掩盖了具体操作细节。

在 CPU + GPU 平台上运行果如下：

```
[yiting.zyk@Inspur-4U-gpu-131 gemm]$ ./test
open CL program started...
Found 1 platforms.
[0] Platform Name: NVIDIA CUDA
Please Choose a platform:0
OK. Your choice is valid.
On this platform there are 4 devices.
[0] Device Name: Tesla K80
[1] Device Name: Tesla K80
[2] Device Name: Tesla K80
[3] Device Name: Tesla K80
Please Choose a device:0
OK. Your choice is valid.
Detailed informations:
        Platform Version:       OpenCL 1.1 CUDA 6.0.39
        Driver Version:         340.65
        Global Memory Size:     11519 MB
        Global Cache Size:      0 MB
        Local Memory Size:      47 KB
        Max Clock frequency:    823 MHz
        Max Workgroup size:     1024
        Compute Units number:           13
        Max memory for allocating:      2879 MB

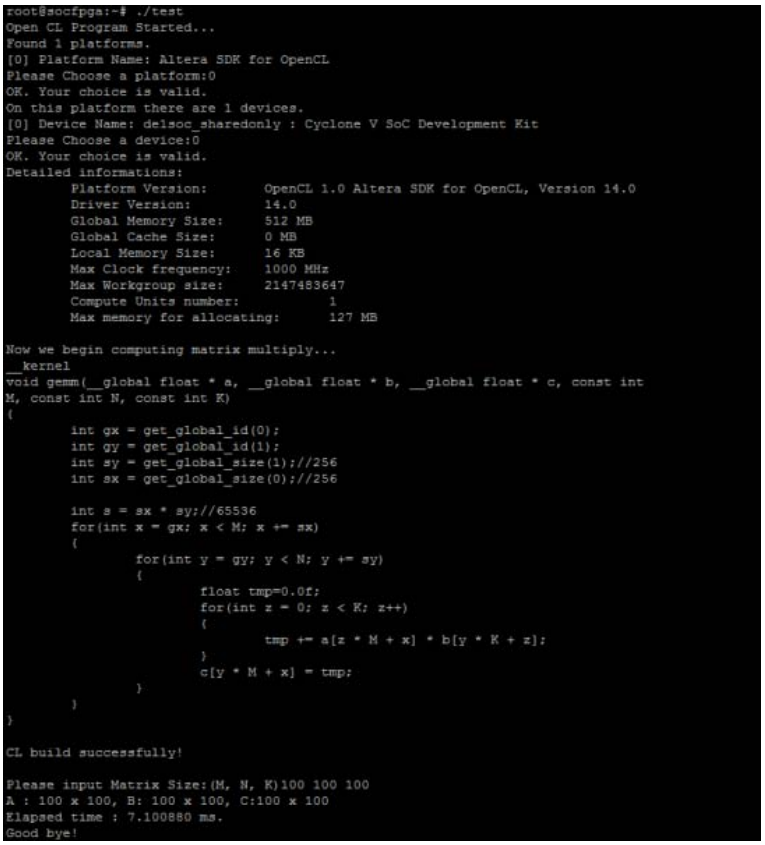
Now we begin computing matrix multiply...
__kernel
void gemm(__global float * a, __global float * b, __global float * c, const int M, const int N, const int K)
{
        int gx = get_global_id(0);
        int gy = get_global_id(1);
        int sy = get_global_size(1);//256
        int sx = get_global_size(0);//256

        int z = sx * sy;//65536
        for(int x = gx; x < M; x += sx)
        {
                for(int y = gy; y < N; y += sy)
                {
                        float tmp=0.0f;
                        for(int z = 0; z < K; z++)
                        {
                                tmp += a[z * M + x] * b[y * K + z];
                        }
                        c[y * M + x] = tmp;
                }
        }
}

CL build successfully!

Please input Matrix Size:(M, N, K)100 100 100
A : 100 x 100, B: 100 x 100, C:100 x 100
Elapsed time : 0.003816 ms.
Good bye!
```

在 Altera SoC 上运行的果如下：



```
root@socfpga:~# ./test
Open CL Program Started...
Found 1 platforms.
[0] Platform Name: Altera SDK for OpenCL
Please Choose a platform:0
OK. Your choice is valid.
On this platform there are 1 devices.
[0] Device Name: delsoc_sharedonly : Cyclone V SoC Development Kit
Please Choose a device:0
OK. Your choice is valid.
Detailed informations:
        Platform Version:       OpenCL 1.0 Altera SDK for OpenCL, Version 14.0
        Driver Version:         14.0
        Global Memory Size:     512 MB
        Global Cache Size:      0 MB
        Local Memory Size:      16 KB
        Max Clock Frequency:    1000 MHz
        Max Workgroup size:     2147483647
        Compute Units number:           1
        Max memory for allocating:      127 MB

Now we begin computing matrix multiply...
__kernel
void gemm(__global float * a, __global float * b, __global float * c, const int
M, const int N, const int K)
{
        int gx = get_global_id(0);
        int gy = get_global_id(1);
        int sy = get_global_size(1);//256
        int sx = get_global_size(0);//256

        int s = sx * sy;//65536
        for(int x = gx; x < M; x += sx)
        {
                for(int y = gy; y < N; y += sy)
                {
                        float tmp=0.0f;
                        for(int z = 0; z < K; z++)
                        {
                                tmp += a[z * M + x] * b[y * K + z];
                        }
                        c[y * M + x] = tmp;
                }
        }
}

CL build successfully!

Please input Matrix Size:(M, N, K)100 100 100
A : 100 x 100, B: 100 x 100, C:100 x 100
Elapsed time : 7.100880 ms.
Good bye!
```

从速度上来看，同样的参数，GPU（Tesla K80）大约需要 0.034816ms，而 FPGA 需要 7.1ms，前者速度是后者的 203 倍。差距如此之大，主要因为 GPU 的计算资源丰富（4992 个浮点计算核心）、主频高（820MHz），当然成本也很高（taobao 上单 K80 大约 38000RMB，此外还需要服务器、主板、电源等配套设施），功耗也高（300W，不包括服务器本身功耗）；而 FPGA 平台则具有速度与面积互换的灵活性，实现成本较低（以 DE1-SoC 为例，差不多 1500RMB，不需要额外的配套设施）、功耗较低（不到 40W，整机功耗）。

通过这个例子可以看出，一个 GPU 程序员可以利用 Altera 提供的 Open CL SDK 灵活地将已有的程序迁移到 FPGA 上。

当然，后续工作仍需针对 FPGA 平台做特定的裁剪以达到性能要求。

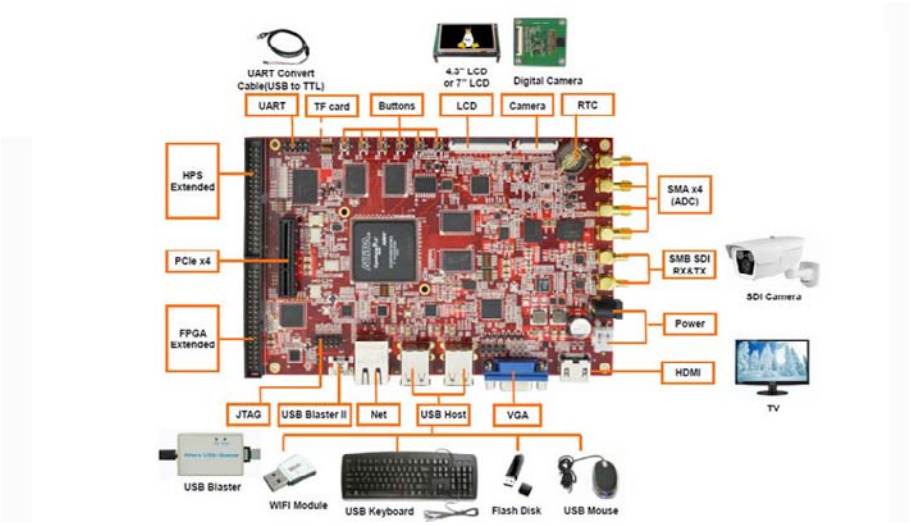
详情请点击：

http://bbs.eeworld.com.cn/thread-458116-1-1.html

二、本章应用平台为: 英蓓特的 lark board

1. 初见 lark board

1.1 官方资料、图，参考设计



Lark Board 采用的是 Altera Cyclone V SX 系列 SoC 芯片，核心资源包含三大部分：传统 FPGA 通用 IO 与逻辑、传统 ARM 通用 IO 及控制器、高速差分收发器。Lark Board 的外围接口也主要根据这三大部分进行定制，主要扩展和定制应用包括：

板载 USB Blaster II JTAG 在线调试

TF 卡及 eMMC 系统启动

HPS 与 FPGA 各 1GB DDR3 内存

SDI 高清输入和输出

VGA 及 HDMI 高清显示

可外接中小尺寸 LCD 触摸屏（7’’ ）

高带宽模拟前端及 ADC 采样

10/100/1000Mbps 以太网

4 个 USB 扩展

可外接数字摄像头扩展

PCIE x1/x4 扩展

RTC 时钟

HPS 部分扩展接口（QSPI、UART、SPI、I2C、GPIO 等）

FPGA 高速收发器（LVDS、RSDS、SLVS、mini-LVDS）扩展

Linux 内核版本是 3.13；u-boot 版本是 2013.01.01

参考资料（视频）：

2014 Altera SoC 合作伙伴研讨会 How to start a Linux based design on Altera SoC – part 1：

http://v.youku.com/v_show/id_XNzk5ODE1NzQw.html

2014 Altera SoC 合作伙伴研讨会 How to start a Linux based design on Altera SoC – part 2：

http://v.youku.com/v_show/id_XODA3MjgyOTg0.html

2014 Altera SoC 合作伙伴研讨会 Embest Demo 演示：

http://v.youku.com/v_show/id_XNzk5NTkxNTA0.html

2014 Altera SoC 合作伙伴研讨会 Embest AMP Reference Design：

http://v.youku.com/v_show/id_XNzk5MjcwOTI0.html

详情请见：
http://www.eeworld.com.cn/huodong/201411Altera/prize.html

1.2 网友眼中的 lark

1.2.1 Lark“冒烟”测试

打开包装盒后，清点完零件，将板子支柱、电源线、VGA 线、HDMI 线、USB 键盘鼠标等统统接上，准备第一次上电“冒烟”测试（Warm up）。

首先我们需要做一些设置。参考资料为 TF 卡中资料 \doc\Chinese\Lark Board 快速操作指南 _V1.0.pdf，设置拨码关如图所示：

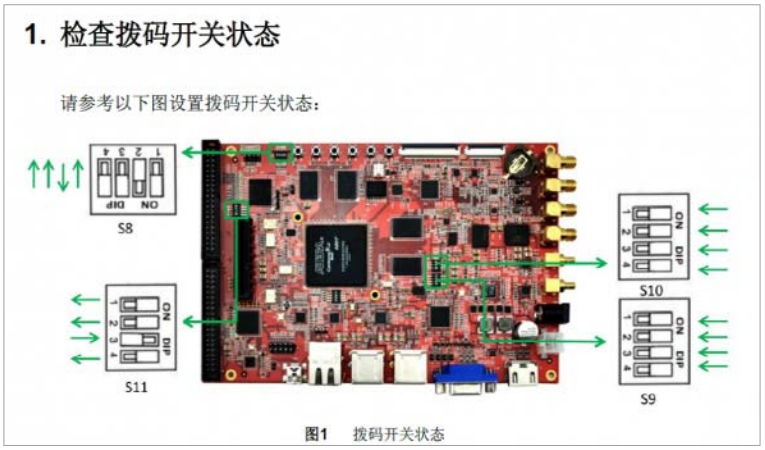


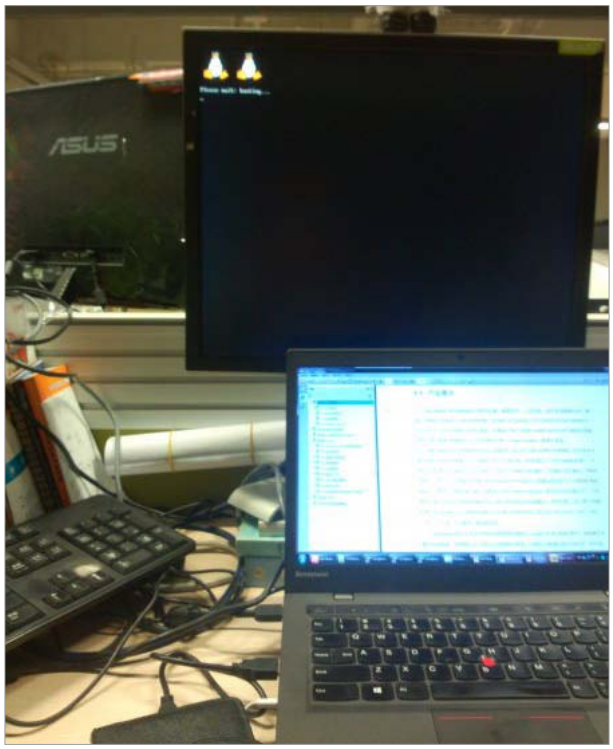
图1 拨码开关状态

我是按箭头指示方向拨的，这样才正常启动。如果这一步设置出错，启动后什么反应都没有。打开电源开关（晕，板子上居然没有开关，只好按下插座开关代替），看到板子一角的 LED 开始闪烁如图所示：

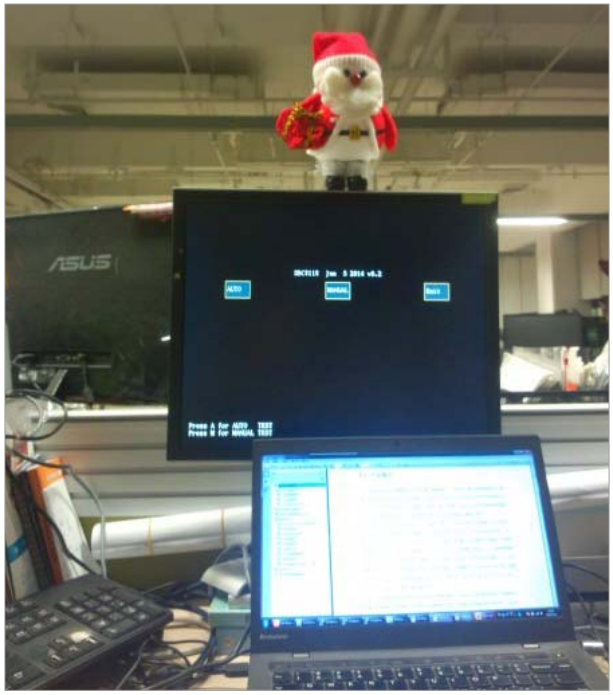


奇怪的是风扇无论如何都不转。

随后不久，VGA 连接的显示器上开始有动作，冒出两只企鹅（说明 Linu 起来了）：

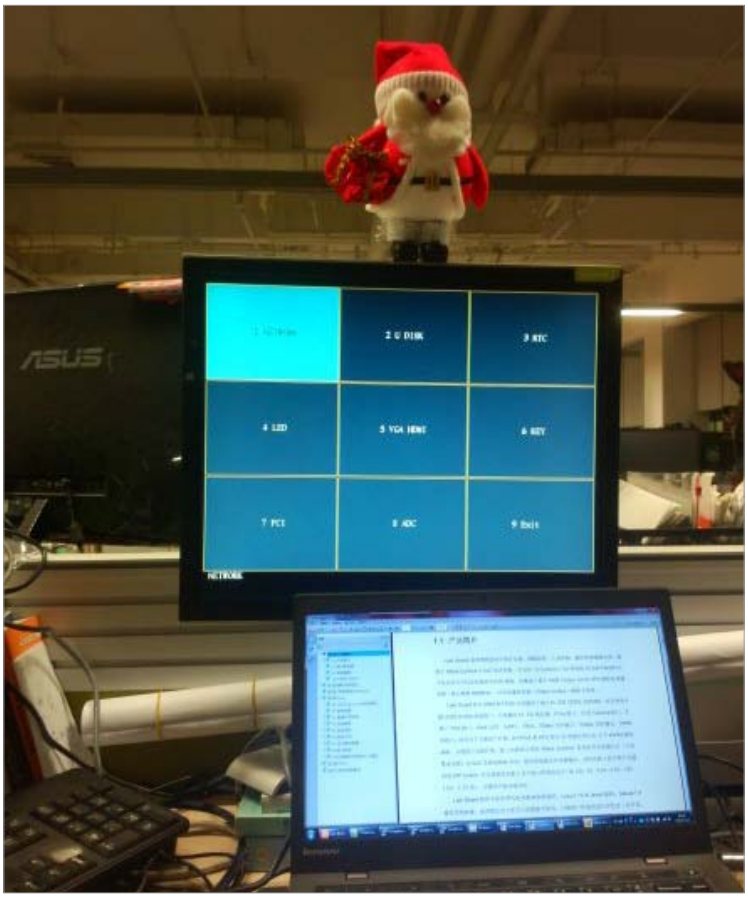


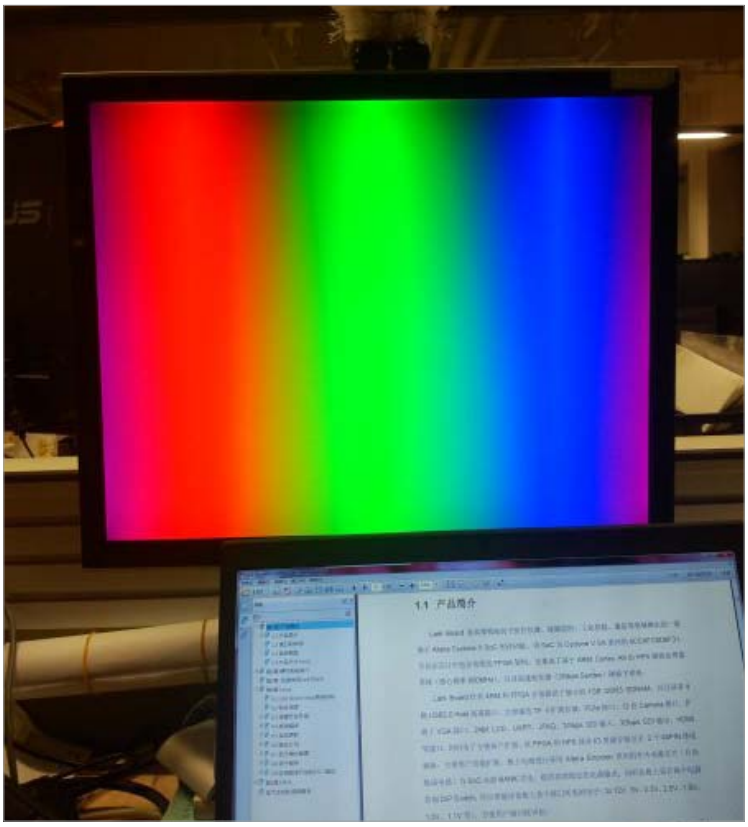
再接着，屏幕上出现三个选择菜单，下图所示：



这是板卡自检程序，需要使用板卡上接的 USB 键盘进行选择和控制。按下“A”选择自动测试模式，“M”为手动测试模式，“E”退出测试，进入系统。

下图为选择自动测试后输出况：

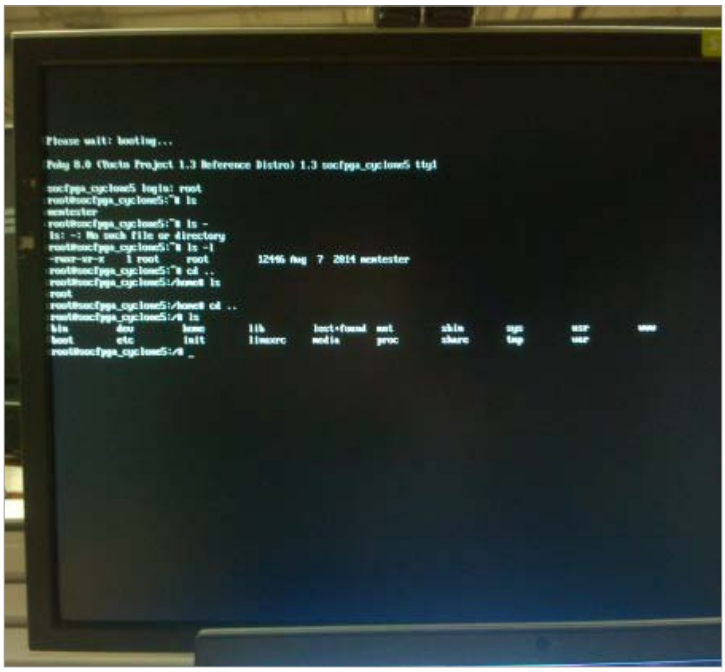




郁闷的是进入上图所示情况后，就再也无法退出，只能按下板卡上的 RESET 重启，选“Exit”退出

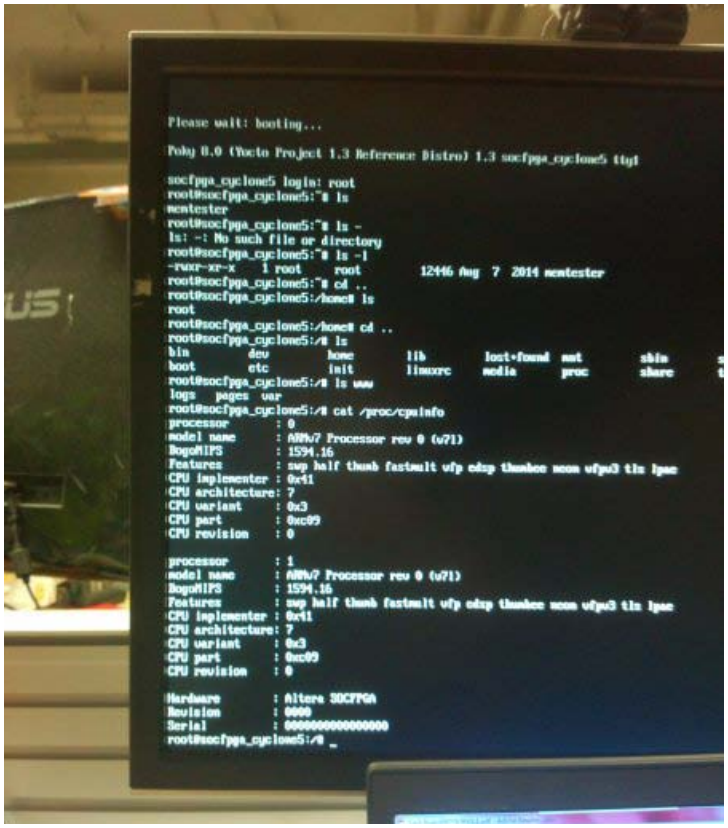
测试，进入系统。

我们使用用户名 root 登录，没有密，强势进入。

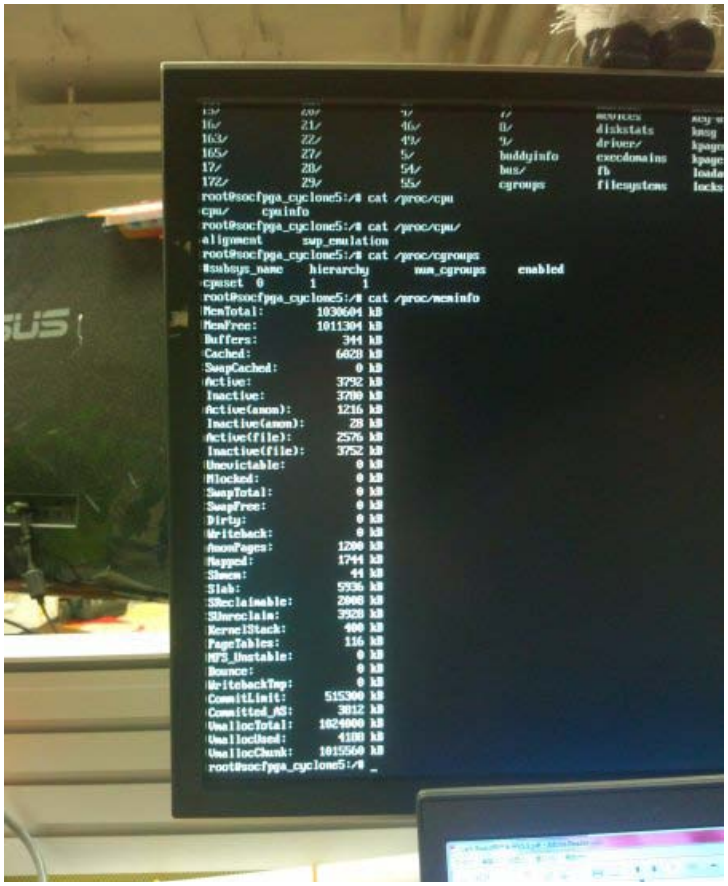


这是一个典型的 Linux 系统，我们可以运行诸如 cd, ls, top, mkdir 等命令。但我们系统与 PC 上有所不同，因为是运行在双核 Cortex A9+FPGA 上的 Linux 系统。

我们查看一下 CPU 信息，输入 cat /proc/cpuinfo 输出如下：



可见 CPU 型号为 ARMv7，支持 NEON 加速器。另外还看到 Altera SoCFPGA 硬件设备。进一步查看内存信息，输入 cat /proc/meminfo 输出如下图：



可以看到 1GB 内存空间，空闲 1GB，因为没有运行比较占内存的应用，还可以看下我们的磁盘分区情况，输入 cat /proc/partitions 输出如下图：



这里看到的是位于 eMMC Flash 中的磁盘分区，可以看到各个分区所占 block 数目。经过以上过程，对板子有了进一步了解，知道了板卡的启动过程和运行环境。在后面文章中我们会深入研究如何亲手制作这个系统及其驱动模块。

详情请查看：

http://bbs.eeworld.com.cn/thread-454241-1-1.html

1.2.2 Lark 上 ARM 开发环境搭建和配置

经过前面几篇文章的准备工作，我们初步熟悉了 LarkBoard。这篇文章主要介绍 ARM 开发环境的搭建，为后面软件工程的开发提供参考。

ARM 开发通常包括：基于嵌入式操作系统的应用开发，嵌入式操作系统驱动开发和硬件开发。这里关注的是应用开发。后面驱动开发会结合 HPS 外设、FPGA 等设备进行介绍。

本文使用开发环境为 Fedora 17 32bit 操作系统。交叉编译工具链是必不可少的。对于 Lark 开发板我们需要从 https://launchpad.net/linaro-too ... 21123_linux.tar.bz2 下载，大约 60 多 MB。下载后直接解压：

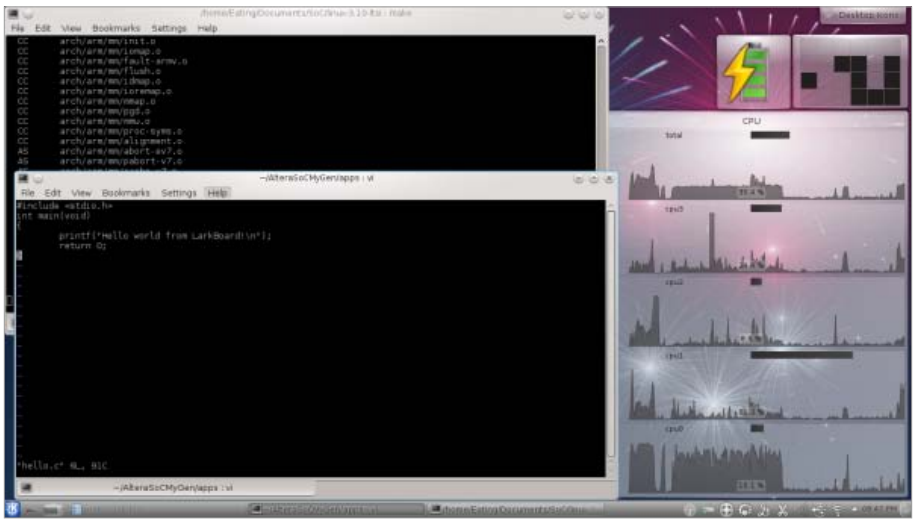
tar xjf gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux.tar.bz2

我的机器上解压目录为

/home/Eating/Documents/SoC/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux，为了便于书写命令，我们导出一个环境变量：export CROSS_COMPILE= /home/Eating/Documents/SoC/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/arm-linux-gnueabihf-
好了，以后我们编译程序可以直接用这个环境变量。

为了让这个环境变量每次开机后自动启用，我们将它写入 .bashrc：
echo export CROSS_COMPILE= /home/Eating/Documents/SoC/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/arm-linux-gnueabihf- >> ~/.bashrc

下面写个 Hello world 程序测试。用 vi 编辑一个 hello.c 文件，写入下内容：

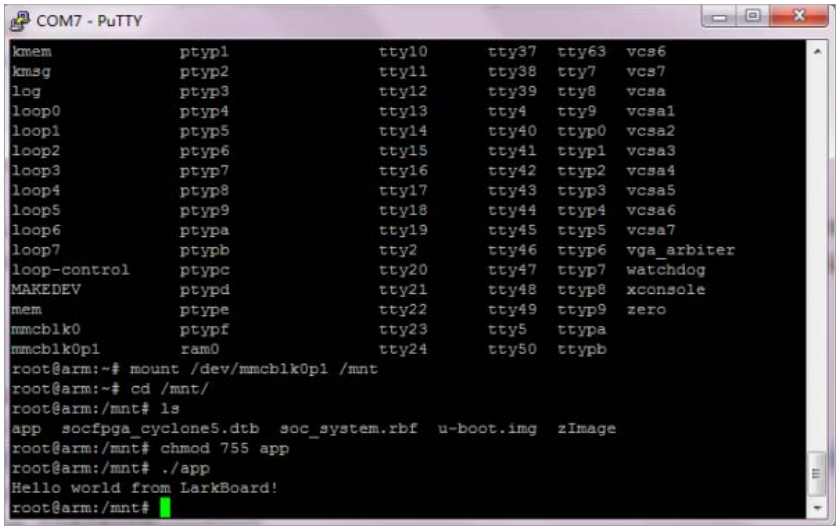


Fedora 17 界面很现代。

写完后保存，在命令行中输入如下命令进行编译：

${CROSS_COMPILE}gcc hello.c -o apps

生成了目标文件 apps。将它拷贝到我们的板子上（我直接放到了 TF 卡的 FAT 分区），行结果如下：



可见成功运行了。

详情请查看：

http://bbs.eeworld.com.cn/thread-454476-1-1.html

1.2.3 Lark 上 ARM 开发之编译 u-boot 和内核

这一部分比较枯燥，需要有耐心，输入命令有一点错误得到结果可能就不正确。

接着上文【Altera SoC 体验之旅】+ Lark 上 ARM 开发环境搭建和配置

http://bbs.eeworld.com.cn/forum. ... &fromuid=169743（出处：电子工程世界 - 论坛）

在此基础上，从 Embest 官网（http://www.embest-tech.cn/produc ... aluation-board.html）下载 u-boot、内核源代码到本地。

首先编译 u-boot：

tar xvjf u-boot-2013-lark-board.tar.bz2

cd u-boot-2013-lark-board

export CROSS_COMPILE=~/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/arm-linux-gnueabihf-

make mrproper

make SoCFPGA_cyclone5_config

make

最终在源代码根目录下生成 u-boot.img 文件；另外在源代码根目录 /spl/ 下生成 u-boot-spl.bin 文件。前者直接拷贝到 TF 卡的 FAT32 分区，覆盖原来的文件。后者用于更新 preloader.bin（本节先不关注这个）。

接下来编译 Linux 内核：

tar xvjf linux-3.10-ltsi.tar.bz2

cd linux-3.10-ltsi

export CROSS_COMPILE=~/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/arm-linux-gnueabihf-

make ARCH=arm lark_board_defconfig

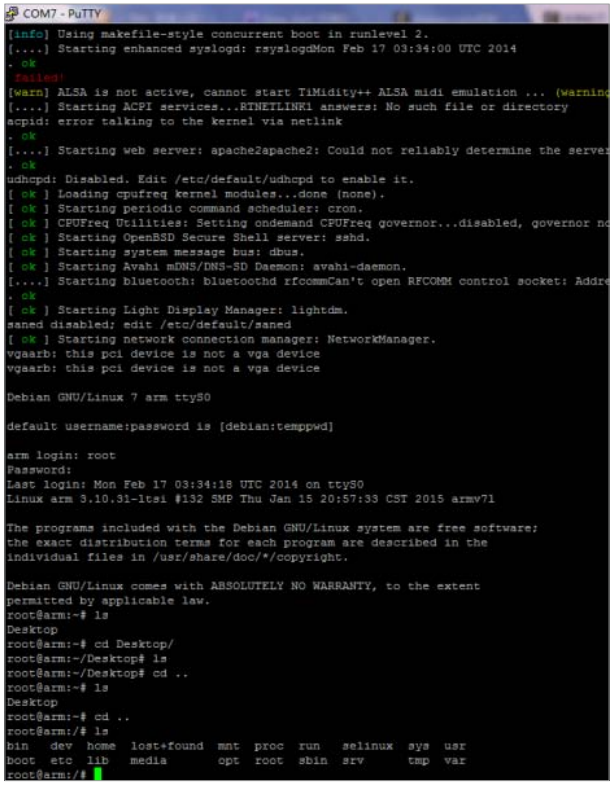
make ARCH=arm LOADADDR=0x8000

最终在 arch/arm/boot/ 目录下生成内核映像文件 zImage，在 arch/arm/boot/dts/ 目录下生成设备树文件 SoCFPGA_cyclone5.dtb。将这两个文件都复制到 TF 卡的 FAT32 分区，覆盖原来的文件。

注：目前 Altera 的 uboot_v2013.01.01 不支持 uImage；

单独 dtb 文件编译命令：make ARCH=arm dtbs

本节通过自己动手，更新了 TF 卡中三个文件：u-boot.img, zImage, SoCFPGA_cyclone5.dtb。将 TF 卡插入卡座，上电，果如图所示。



执行 startx，图形界面和之前的帖子【 Altera SoC 体验之旅】+ Lark 上的 Debian 桌面操作系统 http://bbs.eeworld.com.cn/forum. ... &fromuid=169743

详情请点击：http://bbs.eeworld.com.cn/thread-454479-1-1.html

2. 让 SoC 飞一会儿

2.1 LWHPS2FPGA 深入浅出 - 点亮 LED

本文主要涉及 lightweight HPS-to-FPGA bridge 的应用，SoC 中 ARM 通过 lightweightHPS-to-FPGA bridge 控制 FPGA 侧的 3 个 LED。

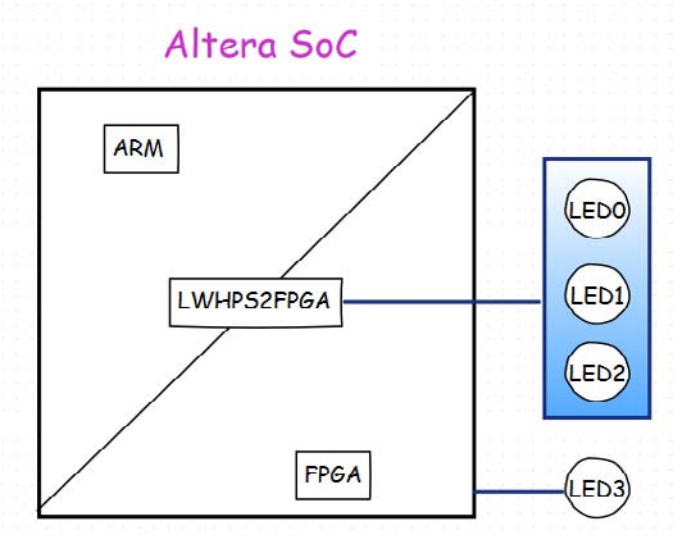
1、 硬件环境

硬件平台：Embest SoC --LarkBoard

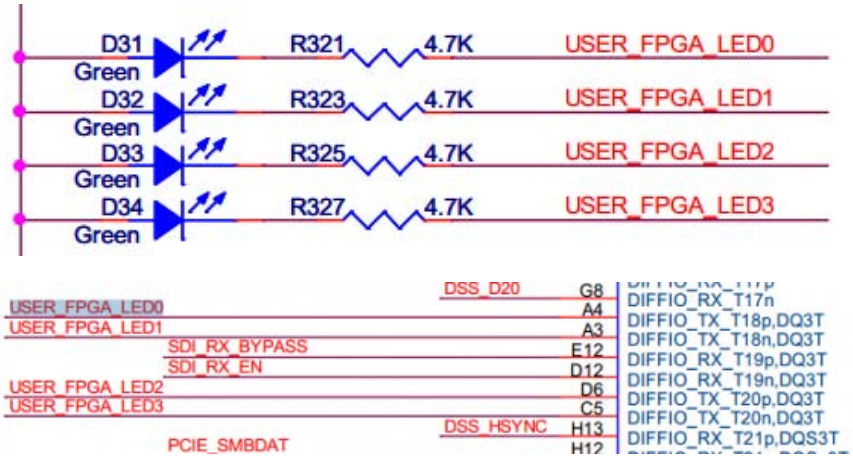
软件平台：开发板 -linux-3.10.31

Quartus 1.0

2、 系统设计框图

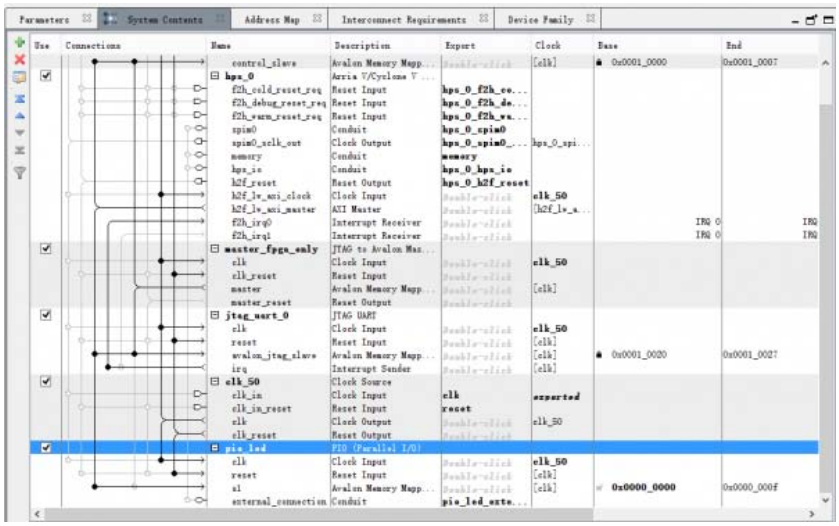


ARM 通过 lightweight HPS-to-FPGA bridge 控制 FPGA 侧的 3 个 LED，其中 Lark Board 中 FP 侧的 LED 硬件连接如下：

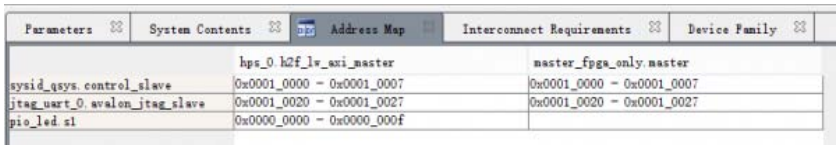


3、 SoC 中硬件系统设

硬件连接关系如下图所示：



其中 PIO_LED 的址分配我 0x0 - 0xf

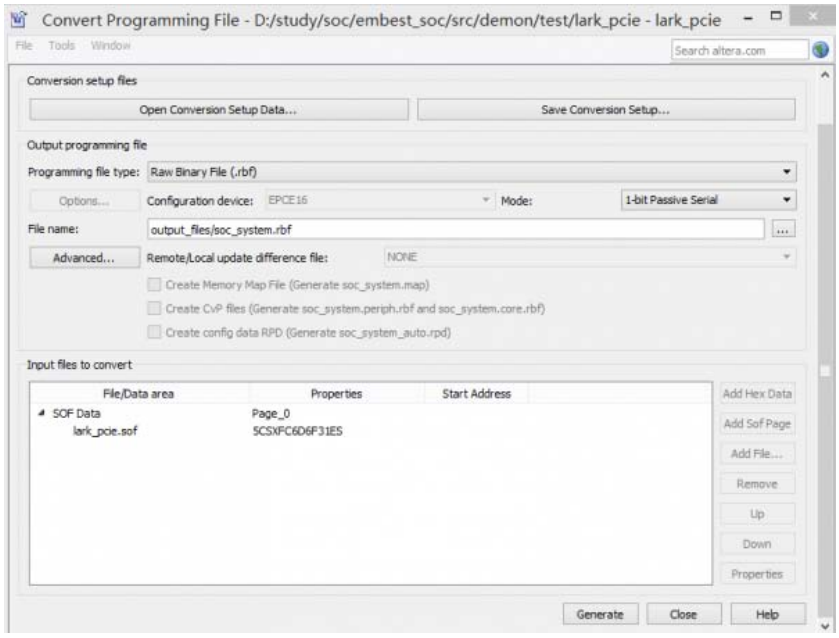


硬件系统例化文件如下：

.pio_led_external_connection_export (o_FPGA_led[2:0]),

复制代码

设计完毕后开始生成硬件系统信息，并在 Quartus 中编译生成 sof 文件，接着转成 rbf 文件，用来替换 TF 卡中的 rbf 文件



具体代码可以详见附件中的工程文件，在此不在赘述

4、ARM 侧代码设计

注意：lightweight HPS-to-FPGA bridge 在 ARM 中的地址映射为 0xff200000，通过上面的设计可以看出 LED 的偏移地址为 0，通过这个地址的操作就可以控制 FPGA 侧的 LED.

详细代码如下 blinker.c：

```
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>

#define PAGE_SIZE 4096
#define LWHPS2FPGA_BRIDGE_BASE 0xff200000
#define BLINK_OFFSET 0x0

volatile unsigned char *blink_mem;
void *bridge_map;

int main(int argc, char *argv[])
{
        int fd, ret = EXIT_FAILURE;
        unsigned char value;
        off_t blink_base = LWHPS2FPGA_BRIDGE_BASE;

        if (argc < 2) {
                fprintf(stderr, "Usage: %s number\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        /* check the bounds of the value being set */
        value = atoi(argv[1]);
        if (value < 0 || value > 7) {
                fprintf(stderr, "Invalid delay setting."
                                "Delay must be between 1 and 15, inclusive.\n");
                exit(EXIT_FAILURE);
        }

        /* open the memory device file */
        fd = open("/dev/mem", O_RDWR|O_SYNC);
        if (fd < 0) {
                perror("open");
                exit(EXIT_FAILURE);
        }

        /* map the LWHPS2FPGA bridge into process memory */
        bridge_map = mmap(NULL, PAGE_SIZE, PROT_WRITE, MAP_SHARED,
```

```
                                fd, blink_base);
        if (bridge_map == MAP_FAILED) {
                perror("mmap");
                goto cleanup;
        }

        /* get the delay_ctrl peripheral's base address */
        blink_mem = (unsigned char *) (bridge_map + BLINK_OFFSET);

        /* write the value */
        *blink_mem = value;

        if (munmap(bridge_map, PAGE_SIZE) < 0) {
                perror("munmap");
                goto cleanup;
        }

        ret = 0;

cleanup:
        close(fd);
        return ret;
}
```

5、测试结果

运行 ./blinker 0　LED 全亮

运行 ./blinker 7　LED 全灭

6、小结

1）通过上述过程，可以完成通过 lightweight HPS-to-FPGA bridge 控制 FPGA 侧 LED 的目的

2）这个应用比较简单，易于操作，还需要向更加深入的方向努力

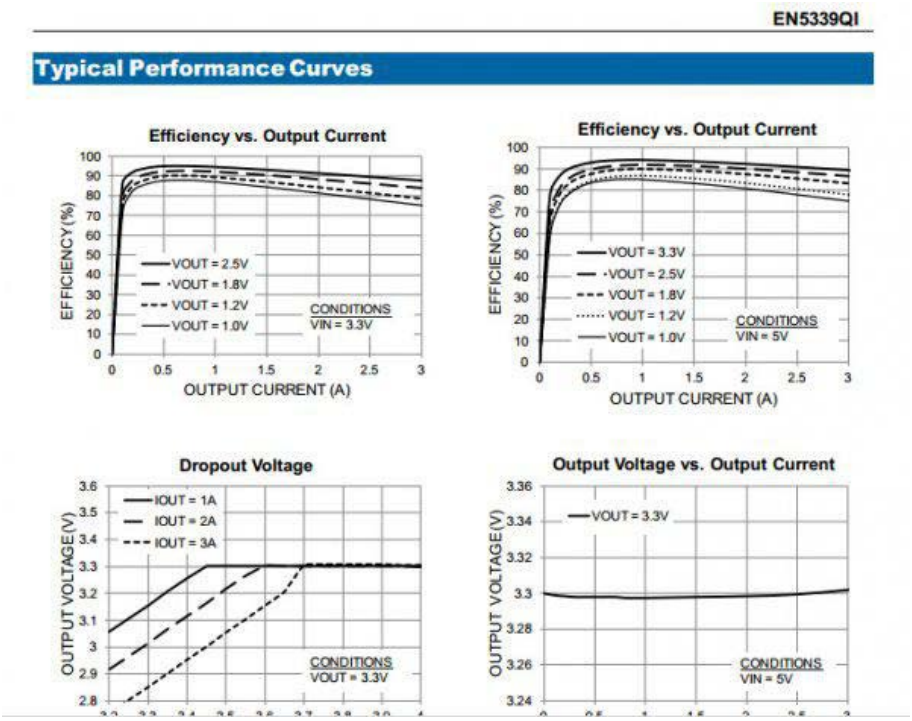
详情请查看：
http://bbs.eeworld.com.cn/thread-456187-1-1.html

2.2 Enpirion Altera SoC 电源系统

Altera SoC demo 板上的电源系统，采用的 Enpirion 整套方案；

从 19VDC 转换到 5V 节点电压后面吧 bulk 电路都是采用的 Altera 自己的解决方案；

首先是 Altera 的 ARM 和 FPGA 核外挂的 DDR3 系统，采用的 1.5VIO 供电和 0.75V 的参考供电，采用的 EN5339QI 和 EV1320QIEN5339QI 为 5V 转 1.5V 3A 的电源；而 EV1320QI 为 1.5V 转 0.75V 2A 的电源，均是为了保证很高的转换效率，也均是采用内置 mos，内置电感，保证设计者负担，好的 EMI 性能，良好的散热性能。



EV1320QI devices can be paralleled to source up to 8A of current. An ENABLE pin with output discharge is available for S3 (suspend to RAM) states.

EV1320QI is specifically designed to meet the precise voltage, fast transient requirements of present and future high-performance, DDR2, DDR3, and low power DDR4 JEDEC VTT requirements. Advanced circuit techniques and high switching frequency deliver high-quality, compact, non-isolated DC-DC conversion.

- Cost Effective Integrated Solution
- Thermal Overload, Over Current, Short Circuit, and Under-Voltage Protection
- RoHS Compliant, MSL level 3, 260C Reflow

Applications

- VTT Bus Termination for DDR2, DDR3, Low Power DDR4, and QDR Memories

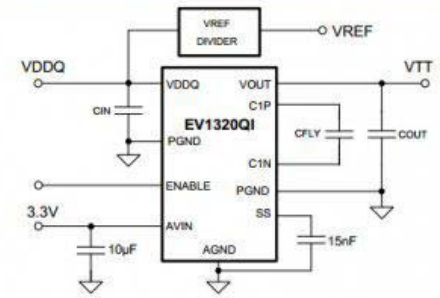


Figure 1. Simplified Applications Circuit

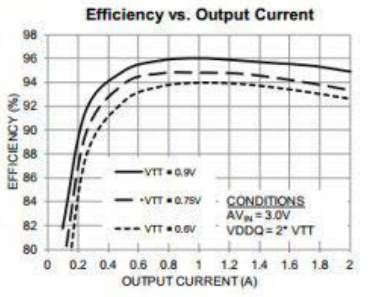


Figure 2. Highest Efficiency in Smallest Solution Size

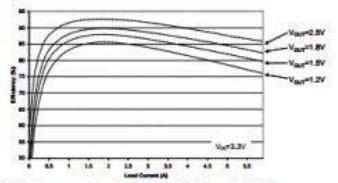
对于 Altera 的电源系统，也有一个很好的特性，具有很好的控制使能功能，便于控制电源的开启和关闭，同时每个电源都有故障告警功能，DEMO 设计的时候，如果哪路电源故障了，都会告警，并且该路电源的红灯亮起。

接下来谈谈 Altera SoC 核心 IO 供电，3.3V，采用 EN5366QI ，5V 输入 3.3V 输出，6A 的电源，同样是内置 mos，内置电感，具有同的优势

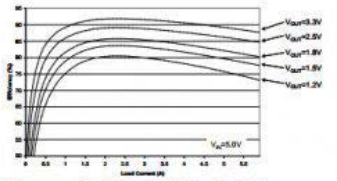
Output Rise Time						
V_{OUT} Rise Time Accuracy	ΔT_{RISE}	$T_{RISE} = C_{SS}$* 75KΩ; 10nF ≤ C_{SS} ≤30nF **(Note 3)**	-25		+25	%
Parallel Operation						
Current Balance	ΔI_{OUT}	With 2 – 4 converters in parallel, the difference between any 2 parts. ΔV_{IN} < 50mV; R_{TRACE} < 10mΩ.		+/-10		%

NOTES:
2. Maximum output current may need to be de-rated, based on operating condition, to meet T_J requirements.
3. Parameter not production tested but is guaranteed by design. Rise time begins when AVIN > V_{UVLO} and Enable=HIGH.

Typical Performance Characteristics



Efficiency vs. Load, V_{IN} = 3.3V.; Load = 0-6A.



Efficiency vs. Load, V_{IN} = 5.0V.; Load = 0-6A.

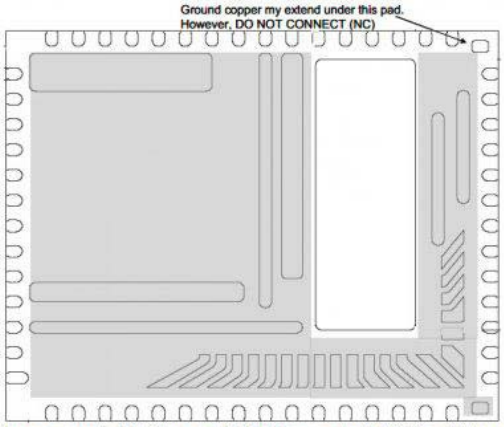


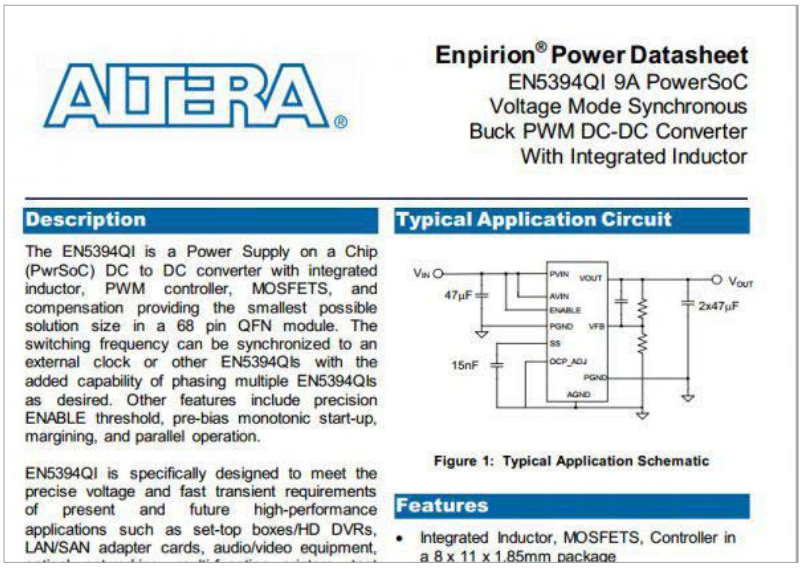
Figure 7. Lead-Frame exposed metal. Grey area highlights exposed metal that is not to be mechanically or electrically connected to the PCB.

从上面看，这个片子具有极高的转换效率，很好的散热管脚，对于散热考虑也是非常周期的，全部集成器件具有很好的 EMI 性能。

并且这些片子的开关频率也非常高，5M，是一般企业很难做到的。

接下来是 Altera SoC 的内核 1.1V 供电；

EN5394QI 为 5V 输入，1.1V 输出，9A 的电源，4MHz 的开关频率，在极小的面积上做到如此大流的电源，确实很不错，并且均匀 93% 的转换效率。



ALTERA

Enpirion® Power Datasheet
EN5394QI 9A PowerSoC
Voltage Mode Synchronous
Buck PWM DC-DC Converter
With Integrated Inductor

Description

The EN5394QI is a Power Supply on a Chip (PwrSoC) DC to DC converter with integrated inductor, PWM controller, MOSFETS, and compensation providing the smallest possible solution size in a 68 pin QFN module. The switching frequency can be synchronized to an external clock or other EN5394QIs with the added capability of phasing multiple EN5394QIs as desired. Other features include precision ENABLE threshold, pre-bias monotonic start-up, margining, and parallel operation.

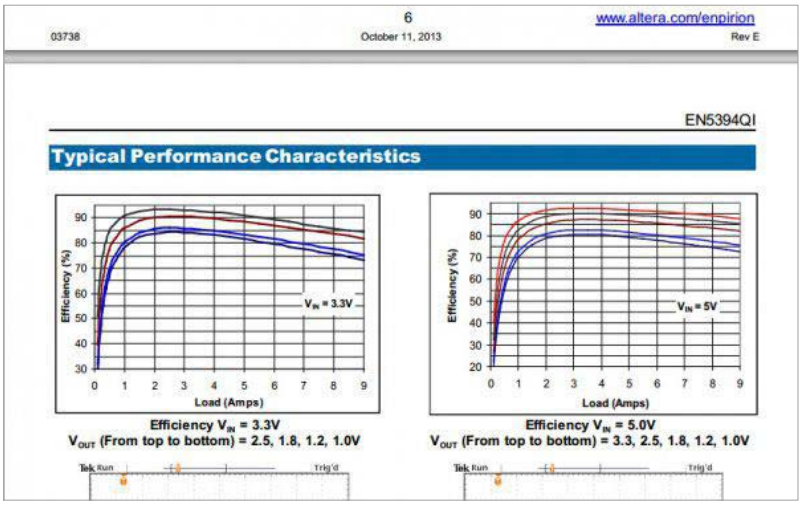
EN5394QI is specifically designed to meet the precise voltage and fast transient requirements of present and future high-performance applications such as set-top boxes/HD DVRs, LAN/SAN adapter cards, audio/video equipment,

Typical Application Circuit

Figure 1: Typical Application Schematic

Features

- Integrated Inductor, MOSFETS, Controller in a 8 x 11 x 1.85mm package



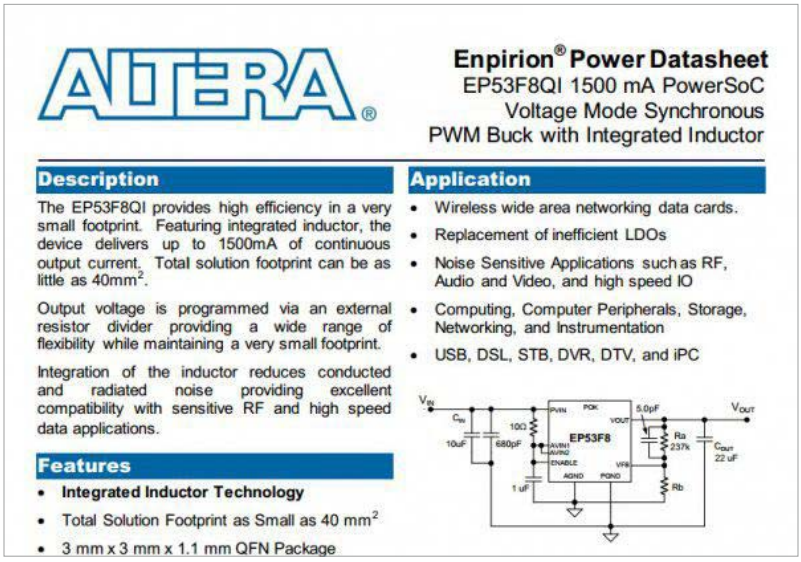
EN5394QI

Typical Performance Characteristics

Efficiency V_{IN} = 3.3V
V_{OUT} (From top to bottom) = 2.5, 1.8, 1.2, 1.0V

Efficiency V_{IN} = 5.0V
V_{OUT} (From top to bottom) = 3.3, 2.5, 1.8, 1.2, 1.0V

最后就是 IO 供电了，2.5V ,1.8V 供电系统；EP53F8QI 为 5V 输入，1.8V 输出，1.5A 的电源，具有 4Mhz 的开关频率，具有极高的转换效率，94%。



ALTERA

Enpirion® Power Datasheet
EP53F8QI 1500 mA PowerSoC
Voltage Mode Synchronous
PWM Buck with Integrated Inductor

Description

The EP53F8QI provides high efficiency in a very small footprint. Featuring integrated inductor, the device delivers up to 1500mA of continuous output current. Total solution footprint can be as little as 40mm^2.

Output voltage is programmed via an external resistor divider providing a wide range of flexibility while maintaining a very small footprint.

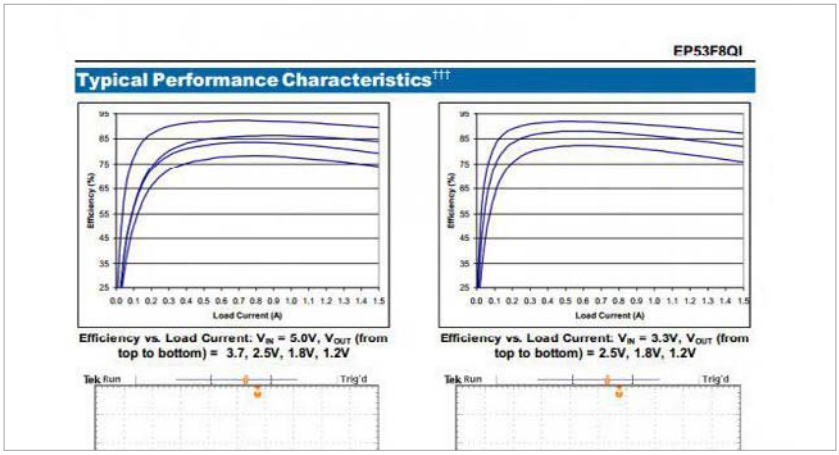
Integration of the inductor reduces conducted and radiated noise providing excellent compatibility with sensitive RF and high speed data applications.

Features

- **Integrated Inductor Technology**
- Total Solution Footprint as Small as 40 mm^2
- 3 mm x 3 mm x 1.1 mm QFN Package

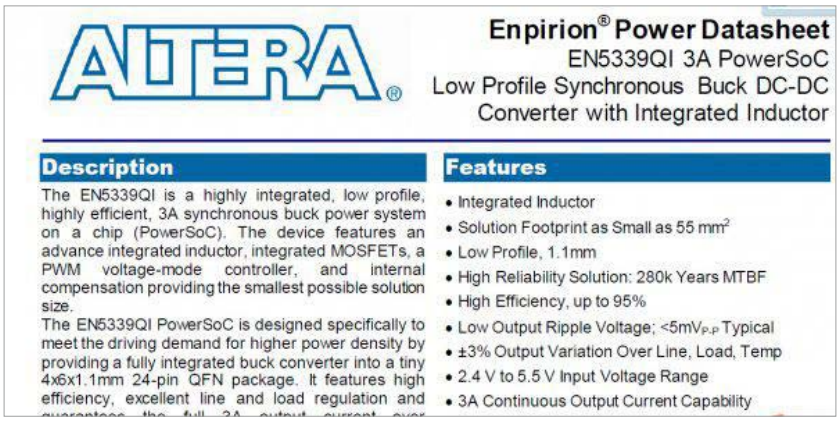
Application

- Wireless wide area networking data cards.
- Replacement of inefficient LDOs
- Noise Sensitive Applications such as RF, Audio and Video, and high speed IO
- Computing, Computer Peripherals, Storage, Networking, and Instrumentation
- USB, DSL, STB, DVR, DTV, and iPC



这颗小电流的电源可以代替 LDO，是非常好的选择。

EN5339QI 为 5V 转 2.5V 3A 的电源 ,3.2Mhz 的开关率，具有极高的转换效率，95%，具有极低的纹波，5mv。



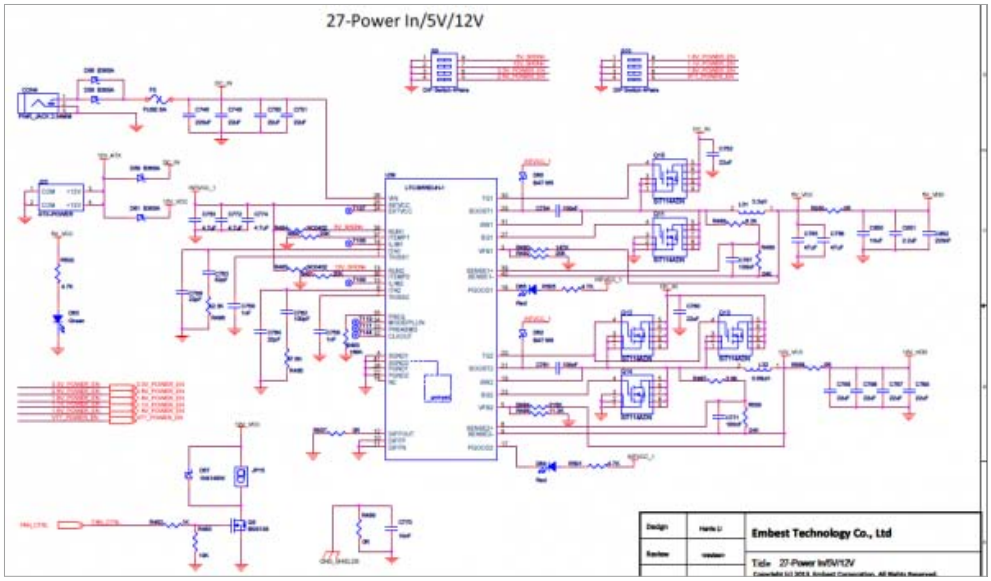
Altera 既是 FPGA 行业的领导者，同样是低电源转换的领导者，其电压系统均采用内部高度集成，良好的散热，极高的转换效率，很低的 EMI，很低的纹波噪声，极高的开关频率，这些都是很多厂家不能做到的，其电流也从几 A 到几十 A，产品种类也是非常企业，在电源做到一流的程度后，在性价比上也做的更好，将对 Alera 电源的应用将是极大的好处。

详情请见：

http://bbs.eeworld.com.cn/thread-460273-1-1.html

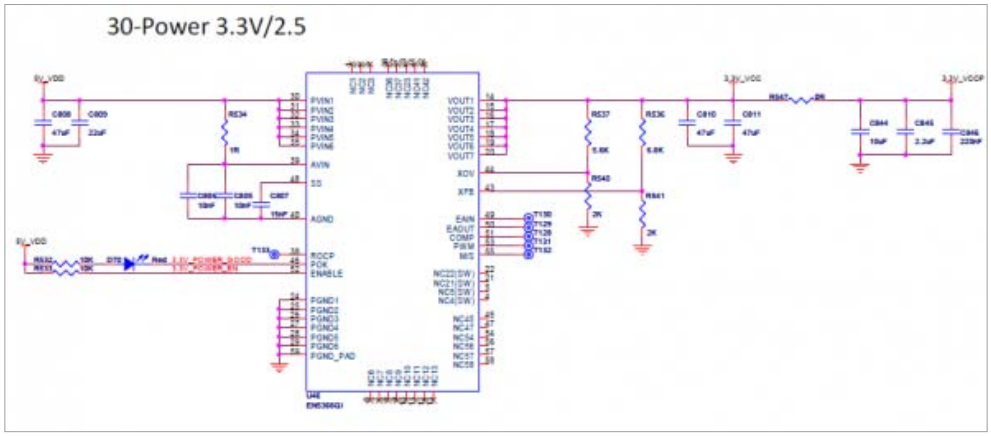
2.3 Enpirion Altera SoC 电源上电时序

Lark Board 看似简单的评估板的电源上电时序看似简单，里面实际隐藏了很深，似 EN 脚都接的 5V，却是上电时间却是不一样。电源输入为笔记本电源的 19V DC；第一级通过 LTC3855 变换为 12 和 5V；



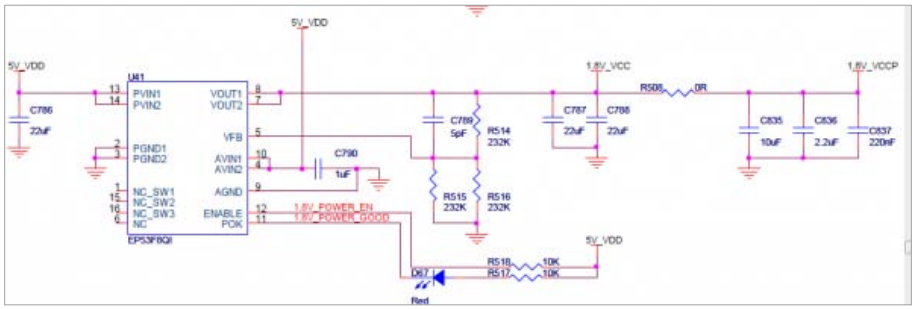
LTC3855 的上电时序 RUN1，RU2 都是内部上拉，所以一上电电压都会起来。ss 软启动设置的都是 1nF，所以 12V 和 5V 最开始是一起起来的。

然后就是 O 3.3V 启动了，3.3V 启动 EN 接 5V_VDD, 一上电就会起来，通过外部的 ss 处接的电容 15nF 确定软启动时间；



上电后软启动时间计算为，具体看芯片资料，EN5366QI 软启动时间计算为：TSS = CSS * 75KΩ (seconds)=1.125ms。

接下来就是 1.8V 了。



EN 接的 5V_VDD，因为 5 V已经起来了。所以 1.8V 该马上起来，外部没有 ss 电容，内部软启动。查看资料；

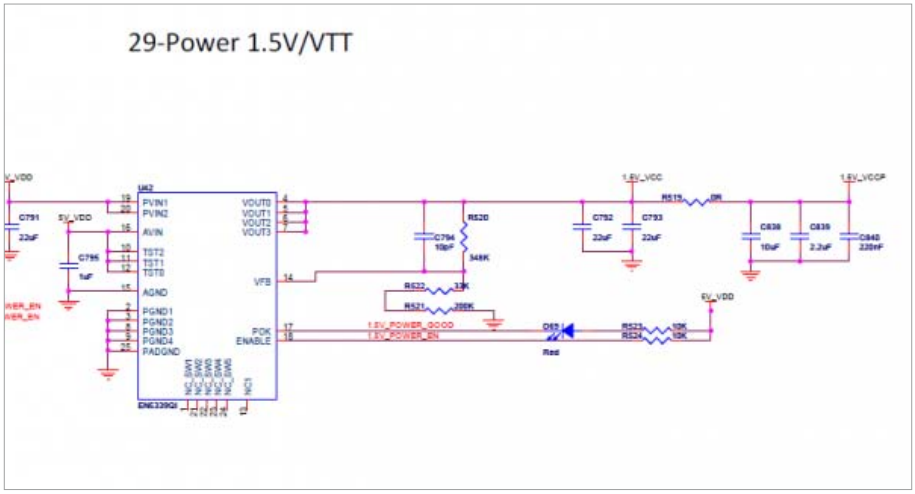
EP53F8QI

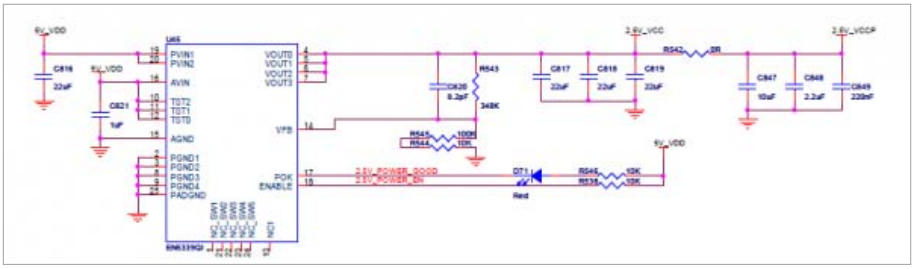
Electrical Characteristics

Typical values for V_{IN} = 5V and T_A =25°C, unless otherwise noted.

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Operating Input Voltage	V_{IN}		2.4		5.5	V
Under Voltage Lockout	V_{UVLO}	V_{IN} going low to high		2.2		V
Under Voltage Lockout	V_{UVLO}	V_{IN} going high to low		2.1		V
VFB Voltage Initial Accuracy	V_{FB}	T_A = 25 °C; V_{IN} = 5V I_{LOAD} = 100 mA	0.588	0.600	0.612	V
Line Regulation		2.4 V ≤ V_{IN} ≤ 5.5V		0.31		%/V
Load Regulation		I_{LOAD} = 0 to 1.5A		0.420		%/A
Temperature Variation		-40°C ≤ T_A ≤ +85°C		0.0012		%/°C
Typical Accuracy		-40°C ≤ T_A ≤ +85°C; 0.5A ≤ I_{LOAD} ≤ 1.0A; 4.5V ≤ V_{IN} ≤ 5.5V; 0.95V ≤ V_{OUT} ≤ 1.5V;		1.5		%
V_{OUT} Rise Time	T_{RISE}	From time ENABLE goes high	0.78	1.2	1.62	mS

必须认真看 EP53F8QI，From time NABLE goes high，然后 Vout rise time 是 1.2ms，典型值。如果不认真看内部的这些参数，却是不知道电源动正常的时间。然后就是 2.5V，1.5V，采用的都是同一颗芯，外部没有 ss 电容，内部软启动，EN 也是接的 5V_VDD; 一上电通过内部软启动就会起来，具体看芯片资料；





具体启动时间

EN5339QI 的软启动时间为 1.4ms，就是上电后 1.4ms 后，对应 2.5,1.5V 启动。

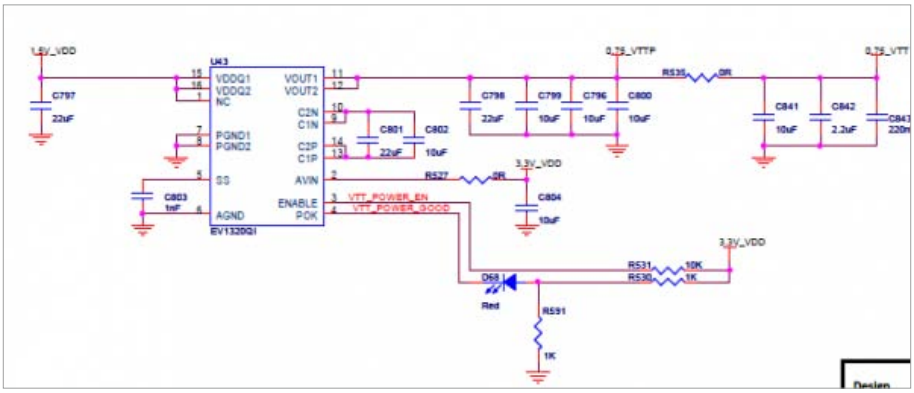
EN5339QI

Electrical Characteristics

NOTE: VIN = 5V, Minimum and Maximum values are over operating ambient temperature range unless otherwise noted. Typical values are at T_A = 25°C.

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Operating Input Voltage	V_{IN}		2.4		5.5	V
Feedback Node Initial Accuracy	V_{VFB}	T_A = 25°C; V_{IN} = 5V I_{LOAD} = 100 mA	0.588	0.600	0.612	V
Output Variation (Note 3) (Line, Load, Temperature)	V_{OUT}	2.4V ≤ V_{IN} ≤ 5.5V 0 ≤ I_{LOAD} ≤ 3A	-3		+3	%
VFB, ENABLE, TST0/1/2 Pin Input Current (Note 4)					+/-40	nA
Shutdown Current		ENABLE Low		20		µA
Under Voltage Lock-out – V_{IN} Rising	V_{UVLOR}	Voltage Above Which UVLO is Not Asserted		2.2		V
Under Voltage Lock-out – t_{VIN} Falling	V_{UVLOF}	Voltage Below Which UVLO is Asserted		2.1		V
Soft-start Time		Time from Enable High (Note 4)	0.91	1.40	1.89	ms

接下来就是 VTT，DDR3 的参考电压了，0.75 V，输入为 1．5 V，EN 为 3.3V_VDD，然后软启动 ss 电容为 1nF，



一上电后的软启动时间为，EV1320QI 对应软启动时间，

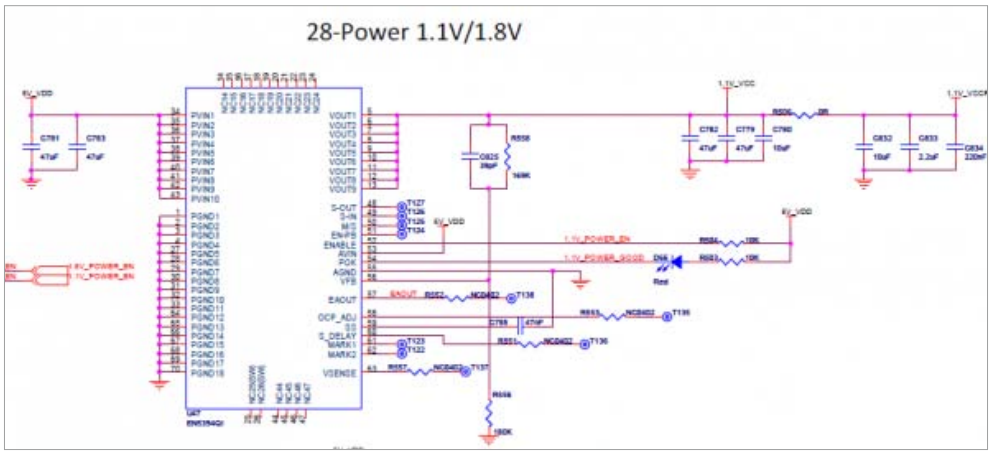
Table 1. Typical Soft-Start Capacitance Time Table

(No Load)

SS Capacitance (nF)	VTT Rise Time (µs)
27	450
15	265
6.8	140
2.7	70
1	40
0.47	30
0.27	25
0.1	20

由于 ss 电容为 1nF，所以启动时间 40us。

接下来是内核电压启动了，EN 接的是 5 V _VDD，软启动电容外接电容为 47nF，对应启动时间要看芯片资料。



EN5394QI 的软启动时间为，TSS = (CSS * 65KΩ) ± 25%=3.055+-25%ms；

综上锁住，19V 点接进来后

first 同时启动 12V，5V；

second 经过 1.125ms 后启动 3.3V；（EN 为 5V_VDD, 输入 5 V）

经过 1.2ms 后启动 1.8V；（EN 为 5V_VDD, 输入 5 V）

经过 1.4ms 启动 2.5 V和 1.5V;(EN 为 5V_VDD, 输入 5V)

经过 1.44ms 启动 0.75V；(EN 为 3.3V_VDD, 输入 1.5V)

经过 3.055+-25%ms 后启动 1.1V；(EN 为 5V_VDD, 输入 5V) 看似简单的启动，确实要认真看电路，看芯片资料，确定 SoC 启动时序是 IO 先启动，最后启动的内核 1.1V。上面就是对应计算的时间。

以上是我所阐述的，有不对的地方，欢迎回帖指正。

详情请见：
http://bbs.eeworld.com.cn/thread-462012-1-1.html

2.4 显示系统初探

1、 硬件环境

硬件平台： Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

概述

为什么需要研究一下显示系统呢，因为我想接个 LCD，那么我可以接个什么样的 LCD 呢，这个问题就来了，需要通过什么样的配置来实现这个目标呢？这就是需要研究和学习的目标。

注意 SoC 的 ARM 是不带 LCD 控制器的，如果要是用 LCD，还需要 FPGA 的配合，使用 VIP 的 Ipcore，这个过程是相当复杂。一时半会还是一知半解的

3、显示流程

1） ARM 端

当然是用的是 Frambuffer，负责把 LCD 的显示数据准备好，并传送给 FPGA

2）FPGA 端

使用了一个 Frame Reader IP core 负责 Framebuffer 数据的接收；一个 Clocked Video Output IP Core，这个显而易见就是把数据解析好输出给 LCD 了。

还需要认真的阅读下面这个文档哦，都是新东西，一下子还能完全的消化。


ug_vip.pdf（点击下载：http://bbs.eeworld.com.cn/thread-462178-1-1.html）

Linux 驱动是 Larkboardfb.c

那么 LCD 的 Frambuffer 数据是通过那个总线传输给 FPGA 的呢，lw_h2f 总线，这个可以从驱动的代码中可以看出来

```
#define CLOCK_CTR_BASE (0xff200000 + 0x00010040)
```

这又引申出另外一个问题，驱动和硬件绑定太死，如果需要调整，需要修改的地方就多了哦

驱动也只定义了几种 LCD 尺寸：

```
#define VGA_IMAGE_WIDTH 1024
#define VGA_IMAGE_HEIGHT 768
#define LCD7_IMAGE_WIDTH 800
#define LCD7_IMAGE_HEIGHT 480
#define LCD4_3_IMAGE_WIDTH 480 #define LCD4_3_IMAGE_HEIGHT 272
```

5、小结

1） 抛砖引玉的来个开场白，我是打算接我的 1024x600 的 HDMI 屏，不过没有成功，屏幕上啥也没有显示；

2） 通过这段时间的学习，真心的觉得 SoC 的显示很复杂，且驱动部分做的不够灵活，限制的东西太多了，一下子的提高了难度和复杂度，是不是就不建议用 LCD ？这就不得而知了。

详情请见：
http://bbs.eeworld.com.cn/thread-462178-1-1.html



3. 设计与应用

3.1 基于 Cyclone V SoC 的高速数据采集系统设计

3.1.1 概述

1、 硬件环境

硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、 系统主要功能设计

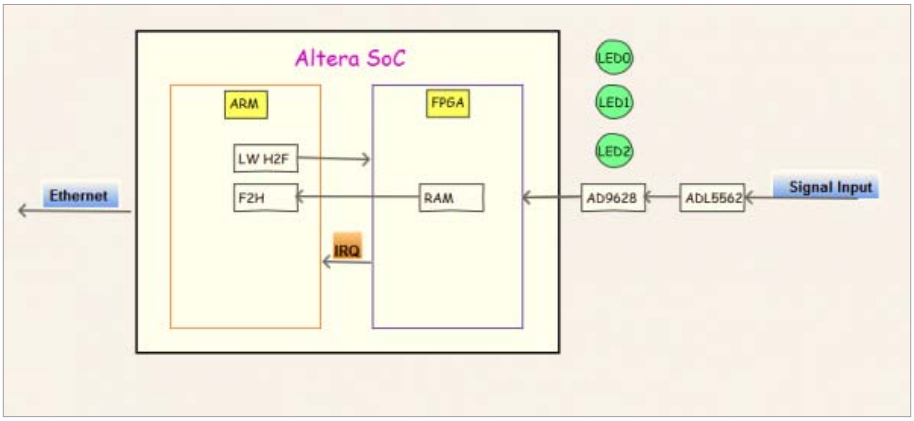
通过 FPGA 控制高速 ADC 采集数据；

系统采用的是AD9628 ,是一款单芯片、双通道、12位、125/105 MSPS模数转换器(ADC) ,采用1.8V电源供电，内置高性能采样保持电路和片内基准电压源。

通过内部总线完成 FPGA 和 ARM 之间的数据交互；

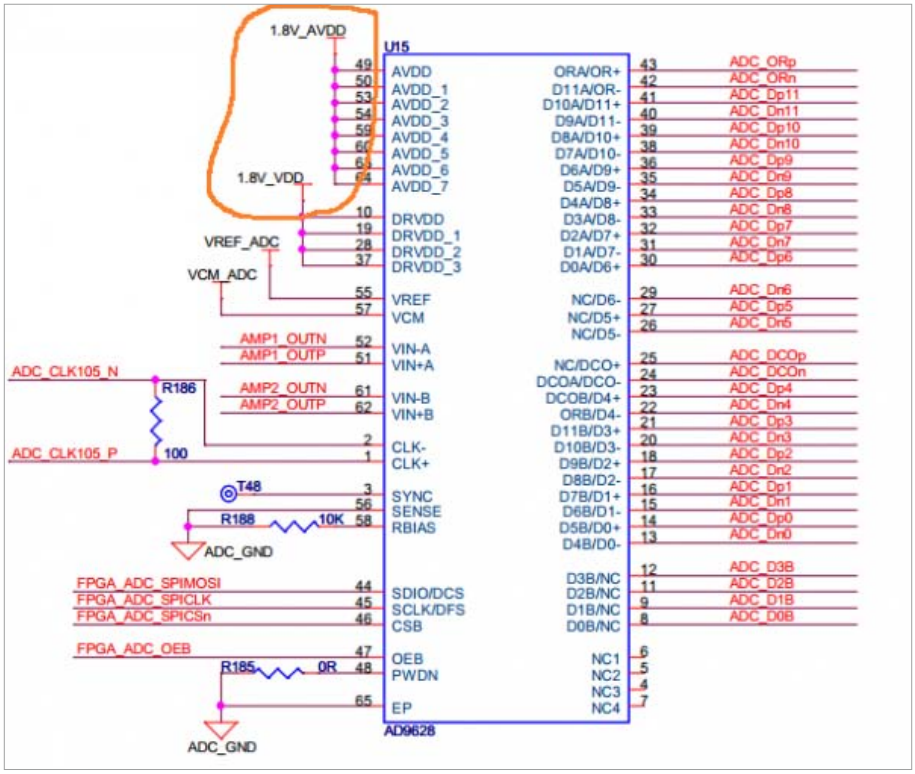
可以通过远程 web 访问或通过网口传递给服务器，便于数据期处理；

系统框图如下所示：

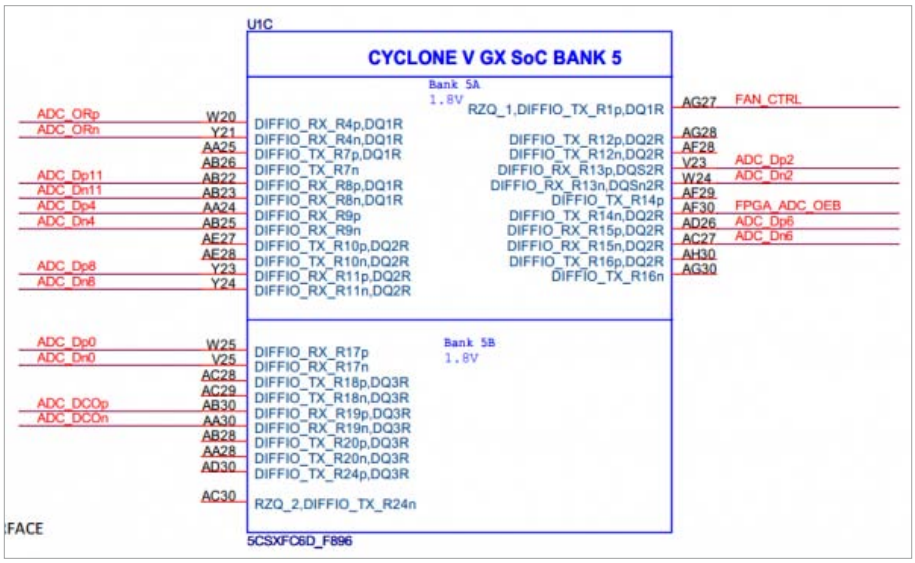


3、 ADC 电路设计和注意事项

最关键的部分就是 ADC 的电路设计，从 datasheet 上可以看出 AD9268 采用的是 18V 电源供电，电路如下所示：

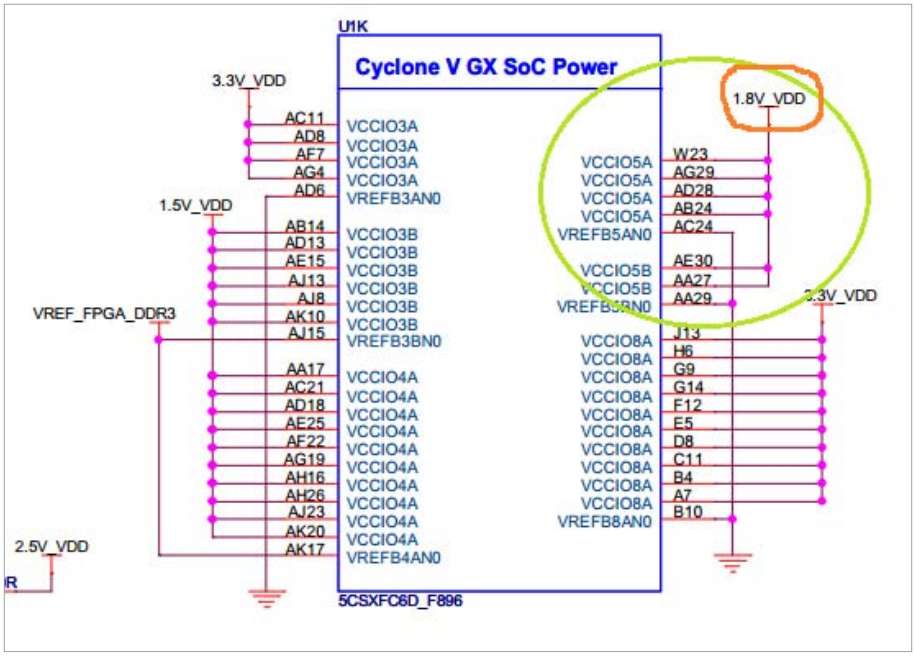


从图上可以很清晰的看出是双通道输入，1.8V 供电，我们接着看看控制 FPGA 的管脚都接哪里，接在 SoC 的 BANK5 上



继续看看 BANK5 的 O 口电压是多少，如下图所示：

可以看出接的是 1.8V 电压哦。



这地方需要引起注意：

需要注意 ADC 的电压等级，差分信号的要接 FPGA 的差分引脚

ADC 的 IO 电压等级要和 FPGA 的 IO 电压等级相匹配

这两点非常重要，但是在做系统的时候往往容易忽视，而造成不必要的损失！

4、 一些个人感悟

1） Altera SoC 对这种数据采集和传输的系统，简直就是个大杀器：

在系统设计方面，避免了传统 arm+FPGA 架构之间繁琐的走线，简化了硬件设计；不管是从硬件设计难度，还是从 PCB 布线来说，都有很大程度的改善和提升；

化解了 FPGA 在通信接口方面的弱点，比如 FPGA 实现网口就很难了，如果要加上 TCP/IP 协议栈就难上加难了，而 Altera SoC 恰恰弥补这样的不足，在一个芯片内让 ARM 和 FPGA 各自发挥自己的优势，从而可以顺利的完成复杂系统设计；

2）ADC 设计部分是重点，注意事项前面已经注重说了，一定要引起注意哦。

3） Altera SoC 给我们带来便利性的同时，同时也给我带来许多技术上的难题，还需要一个一个的解决，接着会聊聊系统详细设计。

详情请点击：

http://bbs.eeworld.com.cn/thread-458280-1-1.html

3.1.2 高速数据采集之 ADC（1）

1、 硬件环境

硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

AD9628 简介

Embest SoC --Lark Board 使用的 ADC 为 ADI 出品的 AD9628，AD9628 是一款单芯片、双通道、12 位、125/105 MSPS 模数转换器 (ADC)，采用 1.8V 电源供电，内置高性能采样保持路和片内基准电压源。系统功能框图如下所示：

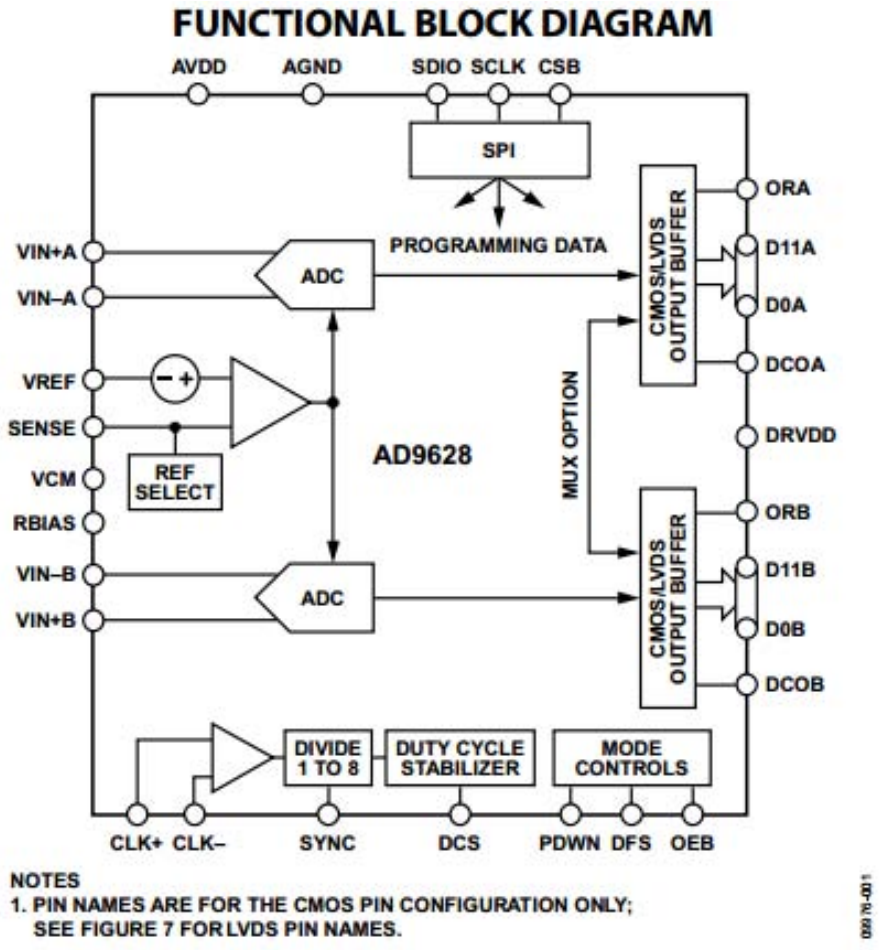
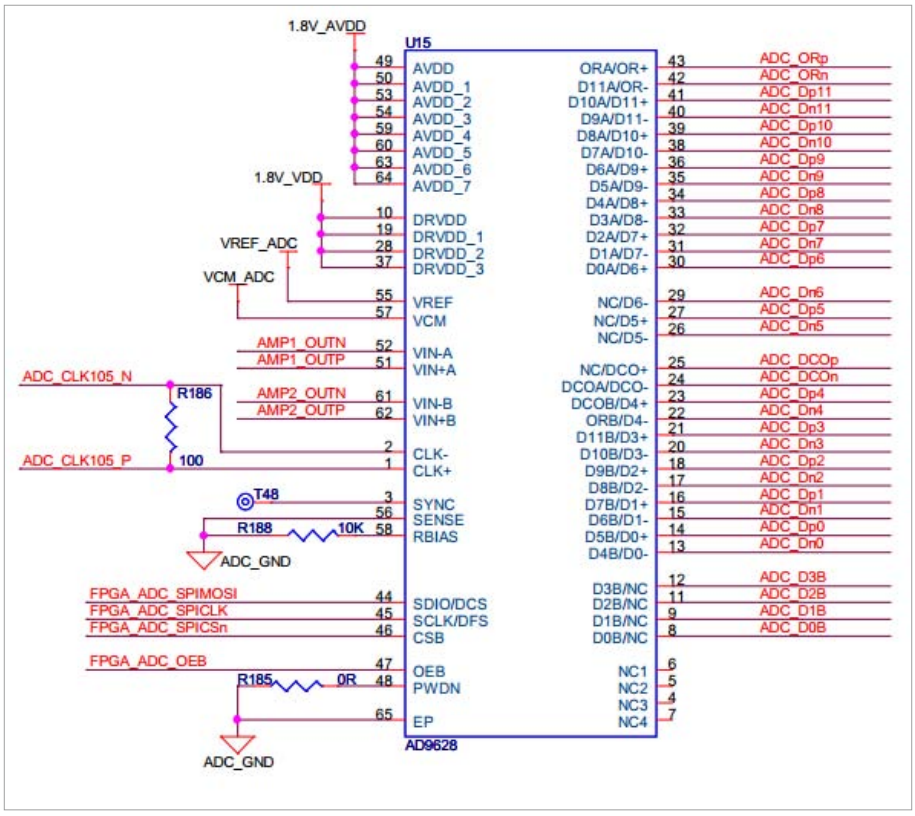


Figure 1.

Lark Board 设计的原理图如下所示：



管脚接线的几点说明：1、SENSE 管脚接地：参考电压的模式选择

Table 12. Reference Configuration Summary

Selected Mode	SENSE Voltage (V)	Resulting VREF (V)	Resulting Differential Span (V p-p)
Fixed Internal Reference	AGND to 0.2	1.0 internal	2.0
Fixed External Reference	AVDD	1.0 applied to external VREF pin	2.0

2、RBIAS 引脚经电阻接地，外部参考偏压，目前也就是 0V 了；

3、clk+，clk- 为 ADC 芯片的差分时钟输入，在 Lark Board 上这个号时通过 FPGA 管脚直接输出给 ADC 芯片的



4、DCOA,DCOB 为 ADC 输出数据的同步时钟，直接接入 FPGA;

5、VCM，VREF 悬空未接；

6、OE 需要设置成 0

3、ADC 时序图如下所示：

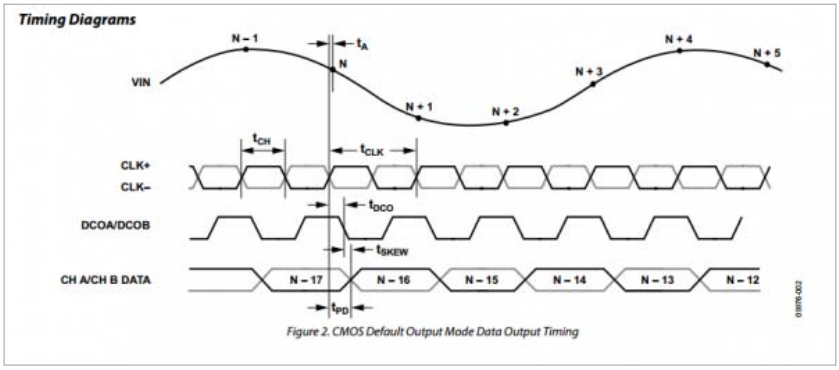


Figure 2. CMOS Default Output Mode Data Output Timing

ADC 数据的输出是与 DCOADCOB 相关的

4、ADC 数据格式如下：

Table 14. Output Data Format

Input (V)	Condition (V)	Offset Binary Output Mode	Twos Complement Mode	OR
VIN+ – VIN–	< –VREF – 0.5 LSB	0000 0000 0000	1000 0000 0000	1
VIN+ – VIN–	= –VREF	0000 0000 0000	1000 0000 0000	0
VIN+ – VIN–	= 0	1000 0000 0000	0000 0000 0000	0
VIN+ – VIN–	= +VREF – 1.0 LSB	1111 1111 1111	0111 1111 1111	0
VIN+ – VIN–	> +VREF – 0.5 LSB	1111 1111 1111	0111 1111 1111	1

可以根据此计算得到的数据，是否和预期的一致

硬件方面主要也就需要注意东西了，下面来看看软件方面需要注意什么呢？

ADC 的时钟是由 FPGA 输出的，么 FPGA 的输出管脚要设置成 LVDS 格式；

io_tsc_int_pin	Bidir	PIN_AF6	3A	B3A_N0	PIN_AF6	3.3-V LVCMOS
o_adc_clk_105m_p	Output	PIN_AE29	5B	B5B_N0	PIN_AE29	LVDS
o_adc_oeb	Output	PIN_AF30	5A	B5A_N0	PIN_AF30	2.5 V (default)

同时需要注意 lvds 信号的定义，只需定义 + 信号的端就行了，quartus 会自动分配 AE29

io_user_dip3	Unknown	PIN_C9	8A	B8A_N0		2.5 V (defau
o_adc_clk_105m_p(n)	Output	PIN_AD29	5B	B5B_N0	PIN_AD29	LVDS

5、一些个人感悟

1）有坛友说这都是别人设计好的，都没自己的东西，明显偷懒。我依然还在乐此不疲的干这个事情。咋一看确实如此，都是在分析别人的东西是怎么做的，怎么设计的。我想说的是这点很重要，很重要。能够分析一款成功的硬件案例都是一笔不小的财富，这样可以提高自己今后设计的正确性，同时也避免不必要的错误或失误。硬件和软件不同，做错一个连接，或画错一根线，那导致的就是整个板子报废。以上我提到的一些注意事项，都是实践中血淋淋的教训。

2）认真的分析硬件原理图是作为硬件工程师必备的技能之一，模仿，消化，再设计，这也是我们学习过程的一个基本套路，比如上图的 ADC 吧，如果不仔细的进行分析，你都不知道得到的数据是什么，为什么是这样的格式，得到的数据又是对应的是什么样的模拟量，貌似扯远了，哈哈

3）Altera SoC IO 能力还是挺强的居然可以轻松容易的从端口输出 105Mhz 的信号，不知道它的极限是多少

4）接下来就要写写代码做测试了，哈哈

详情请点击：
http://bbs.eeworld.com.cn/thread-458879-1-1.html

3.1.3 高速数据采集之 ADC（2）

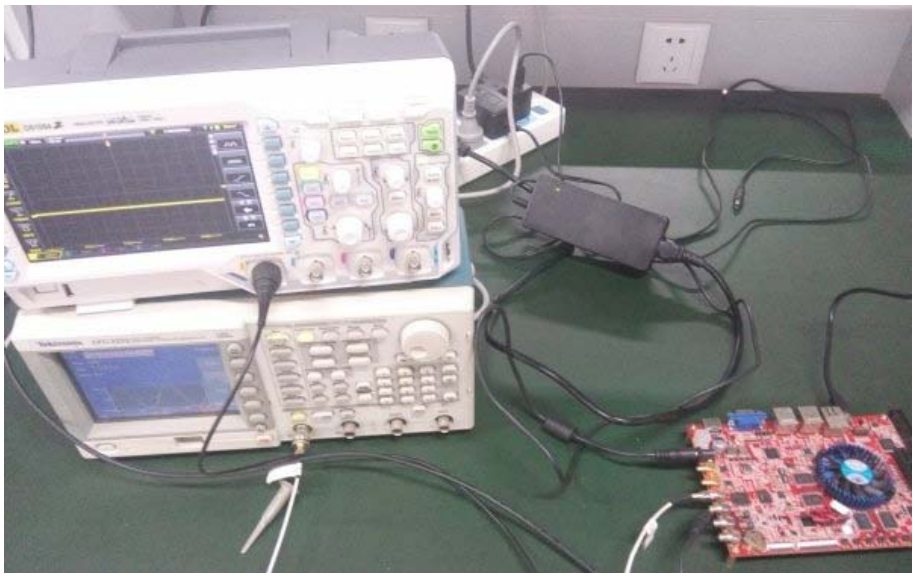
1、 硬件环境

硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、测试环境



示波器：DS1054，（论坛送的哦，一直没有写获奖感谢，羞愧啊）

信号源：泰克的 AFG3252

3、AD9628 的 FPGA 驱动

程序其实很简单，硬件 ok 的话，几行代码就搞定了，主要注意几点：

1）ADC 需要时钟输入，这里输入的是 105M 时钟，如果按奈奎斯特定律 2 倍的信号频率需要小于采样频率，那么就是大概能够正确还原 50Mhz 左右的模拟信号；

2）通道 1 接入的是泰克的 AFG3252，最大可输入 240Mhz 的正弦信号

3）OEB 需要设置成 0

4）ADC 数据输出的同步时钟为 DCOA 和 DCOB

AD9628 的 verilog 驱动代码如下：

```
assign  o_adc_clk_105m_p = s_video_clk_105m ;
assign  o_adc_oeb  = 1'b0;
```

AD9628 的输出同步时钟

```
always @(*)
    begin
```

```
        s_chnl_a_clk = i_adc_dco_p;
        s_chnl_b_clk = i_adc_dco_n;
    end
```

获取 AD9628 的数据

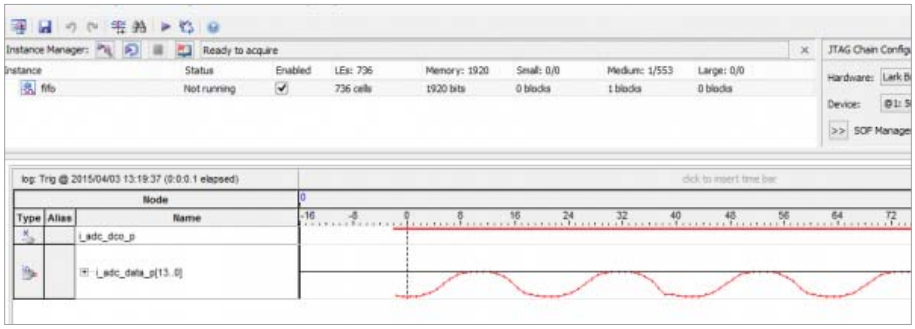
```
always @(posedge s_chnl_a_clk or negedge FPGA_rst_n)
    if (!FPGA_rst_n)
        s_a_data <= 12'd0;
    else// if (i_adc_dco_p)
        s_a_data <= i_adc_data_p[11:0];
```

4、测试结果

这个测试结果有些简陋了，说明是有数据上来的，而且也是符合预期的，我输入了 100Mhz，10Mhz，30Mz 的正弦信号做测试，都能得到大致的正弦波形，但是数据的正确性还没有进一步的检验。下图是信号发生器的输出：



下图是试利器 SignalTap 的截图（怎么使用 SignalTap 就不在这里细说了）可以很清晰的看出是个正弦信号



5、小结

1） 做到这，可以说已经可以通过 Altera SoC 控制 AD9628 采集到模拟信号了，但是依然还有许多问题，正如我上面说的那样，对于数据的格式，和数据的正确性还没有深究，只是这条路通了，这两个问题留着以后再去进一步完善，AD9628 的 datasheet 还需要再认真的读读；

2）AD9628 的 spi 控制端口还没有弄明白，目前都是默认模式；

3）接下来要找个方法把 FPGA 采集到的数据发送的 ARM 端去；

详情请点击：

http://bbs.eeworld.com.cn/thread-458944-1-1.html

3.1.4 高速数据采集之 ADC（3）

1、 硬件环境

硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、信号链深入剖析

原本只打算简单的介绍一下 AD9628，然后用 FPGA 控制，采集到数据就完事了，哪知水太深，还需进一步的深入分析，方能更加清晰的明白整个数据信号链的组成，理清楚数据的来龙去脉。

1）浅谈 AD9628 的默认配置

AD9628 这个控制器是可以通过 SPI 接口来配置操作模式或者一些特殊功能，我在测试的过程中并没有进行配置，使用的是芯片的默认配置，那么都是那些功能呢？刚开始的时候也疏忽了，还是需要静下心来认真阅读啊

The AD9628 serial port interface (SPI) allows the user to configure

the converter for specific functions or operations through a

structured register space provided inside the ADC

具体 spi 控制寄存器的详细描述可见 AD9628 数据手册的 36 页，以下是我摘录的一些主要参数：

采样率为 105MSPS, 芯片型号 AD9268-105

The AD9628 is a monolithic, dual-channel, 1.8 V supply, 12-bit,

125 MSPS/105 MSPS analog-to-digital converter (ADC). It

features a high performance sample-and-hold ciruit and onchip vltage reference.

模拟电压的输入范围，这点很重要，慢慢向下看

Table 3.

Parameter	Temp	AD9628-105/125 Min	Typ	Max	Unit
DIFFERENTIAL CLOCK INPUTS (CLK+, CLK−)					
Logic Compliance			CMOS/LVDS/LVPECL		
Internal Common-Mode Bias	Full		0.9		V
Differential Input Voltage	Full	0.3		3.6	V p-p
Input Voltage Range	Full	AGND − 0.3		AVDD + 0.2	V
Input Common-Mode Range	Full	0.9		1.4	V
High Level Input Current	Full	−10		+10	μA
Low Level Input Current	Full	−10		+10	μA
Input Capacitance	Full		4		pF
Input Resistance	Full	8	10	12	kΩ

时钟：默认时钟分频数为 1

0x09	Global clock (global)	Open	Open	Open	Open	Open	Open	Open	Duty cycle stabilizer 0 = disabled 1 = enabled	0x01	
0x0B	Clock divide (global)	Open	Open	Open	Open	Open	Clock divide ratio 000 = divide by 1 001 = divide by 2 010 = divide by 3 011 = divide by 4 100 = divide by 5 101 = divide by 6 110 = divide by 7 111 = divide by 8			0x00	The divide ratio is value plus 1

输出模式：默认输出接口为 MOS，而不是 LVDS，一开始我也理解错了；编码格式为 offset binar

	adjust (local)									
0x14	Output mode	Output port logic type (global) 00 = CMOS, 1.8 V 10 = LVDS, ANSI 11 = LVDS, reduced range	Output interleave enable (global)	Output port disable (local)	Open (global)	Output invert (local)	Output format 00 = offset binary 01 = twos complement 10 = Gray code		0x00	Configures the outputs and the format of the data

参考电压：固定参考电压为 1.0V

VOLTAGE REFERENCE

A stable and accurate 1.0 V voltage reference is built into the AD9628. The VREF can be configured using either the internal 1.0 V reference or an externally applied 1.0 V reference voltage. The various reference modes are summarized in the sections that follow. The Reference Decoupling section describes the best practices PCB layout of the reference.

这个地方是有疑问的，默认配置是 2.0Vp-p，实际用示波器测是 1.0V

0x18	VREF select (global)	Open	Open	Open	Open	Open	Internal V_{REF} digital adjustment 000 = 1.0 V p-p 001 = 1.14 V p-p 010 = 1.33 V p-p 011 = 1.6 V p-p 100 = 2.0 V p-p		0x04	Select and/or adjust V_{REF}

SENSE 接地，固定使用内部参考电压，也就是 1.0V

Table 12. Reference Configuration Summary

Selected Mode	SENSE Voltage (V)	Resulting VREF (V)	Resulting Differential Span (V p-p)
Fixed Internal Reference	AGND to 0.2	1.0 internal	2.0
Fixed External Reference	AVDD	1.0 applied to external VREF pin	2.0

数据的编码格式，offset binary

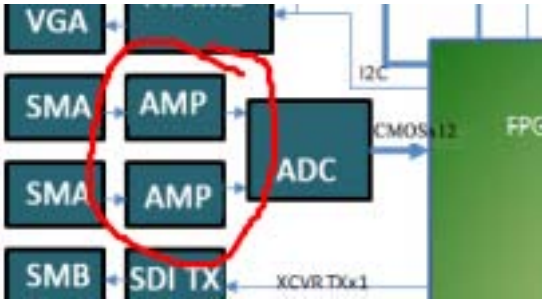
Table 14. Output Data Format

Input (V)	Condition (V)	Offset Binary Output Mode	Twos Complement Mode	OR
VIN+ – VIN–	< –VREF – 0.5 LSB	0000 0000 0000	1000 0000 0000	1
VIN+ – VIN–	= –VREF	0000 0000 0000	1000 0000 0000	0
VIN+ – VIN–	= 0	1000 0000 0000	0000 0000 0000	0
VIN+ – VIN–	= +VREF – 1.0 LSB	1111 1111 1111	0111 1111 1111	0
VIN+ – VIN–	> +VREF – 0.5 LSB	1111 1111 1111	0111 1111 1111	1

这样通过 VIN+ 和 VIN- 我们就可以计算出它的编码，并且可以和实际的测量值进行比较。这时候我们不得不研究一下前级放大电路。

2）前级放大干了什么，模拟信号的前世今生

从实际的电路中可以看出，入 AD9628 的信号是进过前级放大电路 ADL5562 的它在电路中起了什么样的作用，对后续的 ADC 到底产生了什么影响呢？



实际的 ADL5562 电路就是按照参考电路设计的：

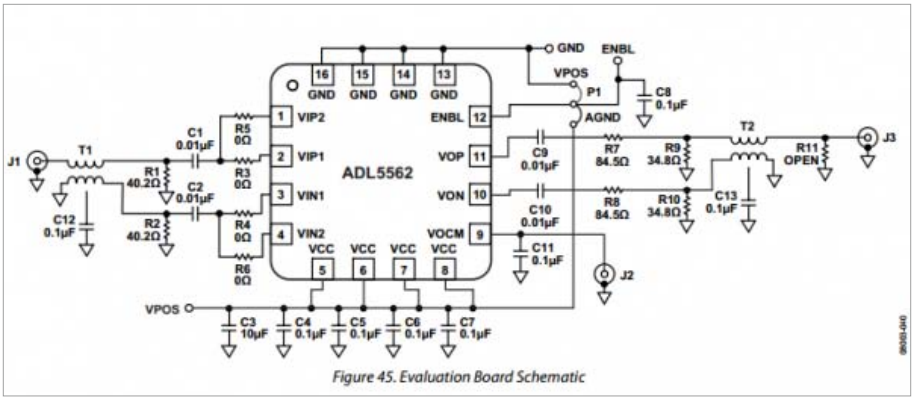


Figure 45. Evaluation Board Schematic

注意：这依然是差分输入，和单独输入还是不一样的，有兴趣的见数据手册第 15 页。

这个电路的增益是多少呢？选择 R1=R2=40.2 欧姆的时候增益为 15.5db，相当于信号放大 6 倍。

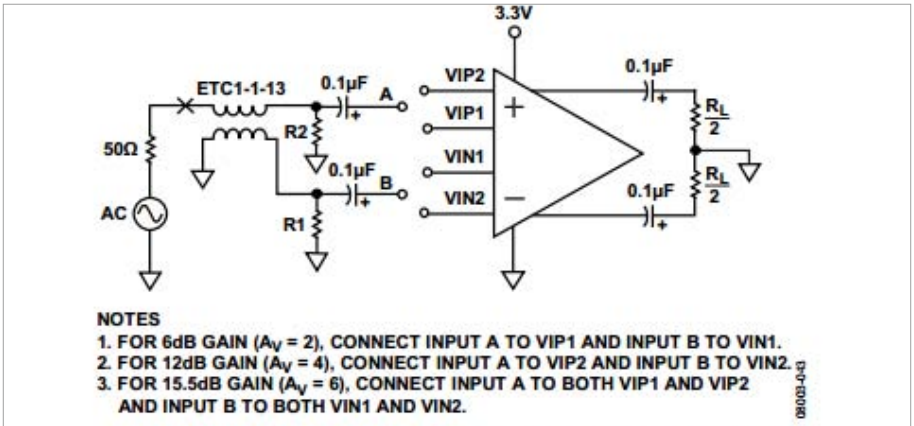


Figure 34. Differential-Input to Differential-Output Configuration

Table 4. Differential Termination Values for Figure 34

Gain (dB)	R1 (Ω)	R2 (Ω)
6	28.7	28.7
12	33.2	33.2
15.5	40.2	40.2

注意：这个地方可以通过修改实际电路中电阻，改变整个链路的增益，可以设置为 6db（2 倍），12db（4 倍）

从这个电路可以看出，原始信号进入 ADL5562 后被放大 6 倍，然后一差分信号的方式输出给下一级 AD9268

由于 VIN- 接地，那么 VIN 的原始电压输入范围为 -0.3/6V<VIN+<2.0/6V

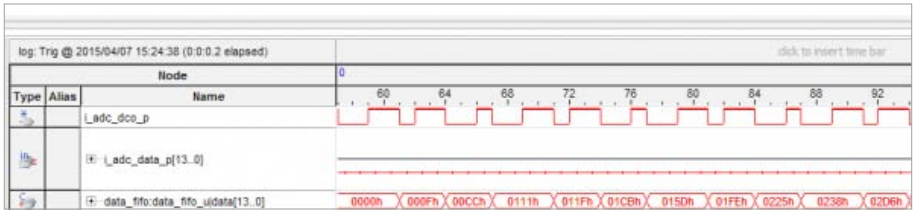
3、测试数据

初步的试验结果，配置 AFG322 如下：



高电平配置的有些不合适，要不可以计算一下实测量值是否正确，明天看看可有时间补测一下。

低电平的数据：



高电平的数据：电压设置太高，肯定超过了



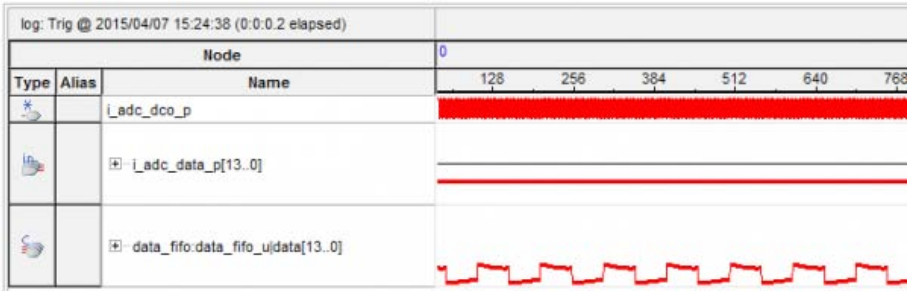
数据频率：



singtap 的采样率设置的是 100M，可以数一数高电平大概维持了 50 个采样点，100M/（50*2）=1Mhz

抓出的模拟波形

波形有畸变，原因不明，直观感觉和硬件电路有关，用示波器测量时候发现，ADC 前级输入的信号已经发生畸变，在这就不深究了。



4、总结

1） 看数据手册太累了，这部分最好由开发板厂商来说清楚，而实际上他们提供的入门指南确实有些简陋；

2）信号采集部分也就算整理到这个程度了，后面还有很多工作需要做，就不在这个地方过多的停留；

3） 欢迎大家批评指正，一起学习，共同进步。

（点击下载：http://bbs.eworld.com.cn/thread-459030-1-1.html）

U13_U14_ADL5562_ADI.pdf

（点击下载：http://bbs.eeworld.com.cn/thread-459030-1-1.html）

U15_AD9628_ADI.pdf

详情请点击：

http://bbs.eeworld.com.cn/thread-459030-1-1.html

3.1.5 高速数据采集之中断

1、 硬件环境

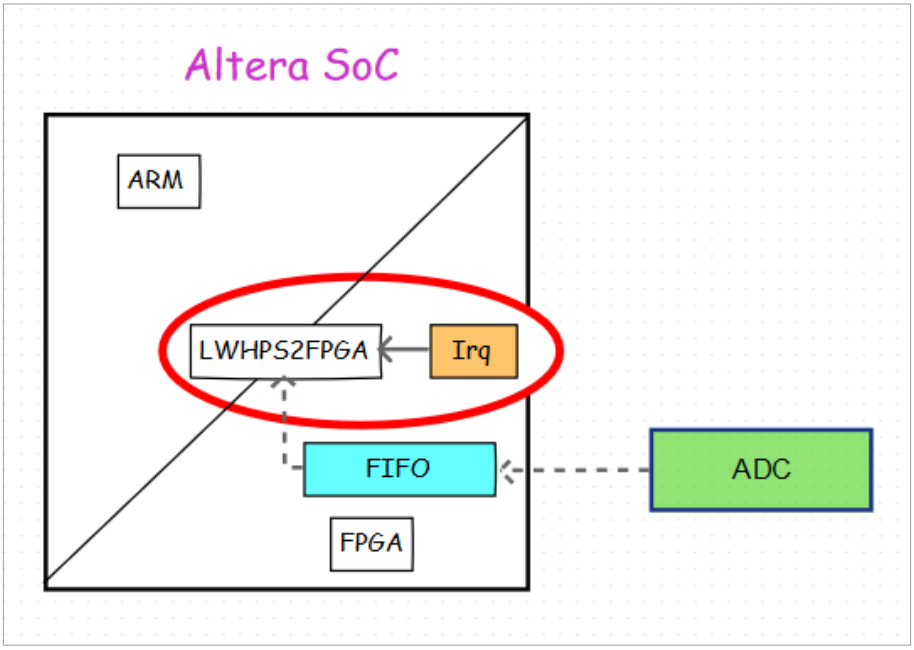
硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、系统概述

简单点说，就是 FPGA 准备好数据了，现在需要通知 ARM 过来取数据。在传统的设计里面可以把 FPGA 的一个 O 挂在 ARM 的外部中断上，准备好数据了就触发一次中断，大喊一声“HI，我来啦！准备接客”。那么在 SoC 中怎么设计呢，如下图所示：

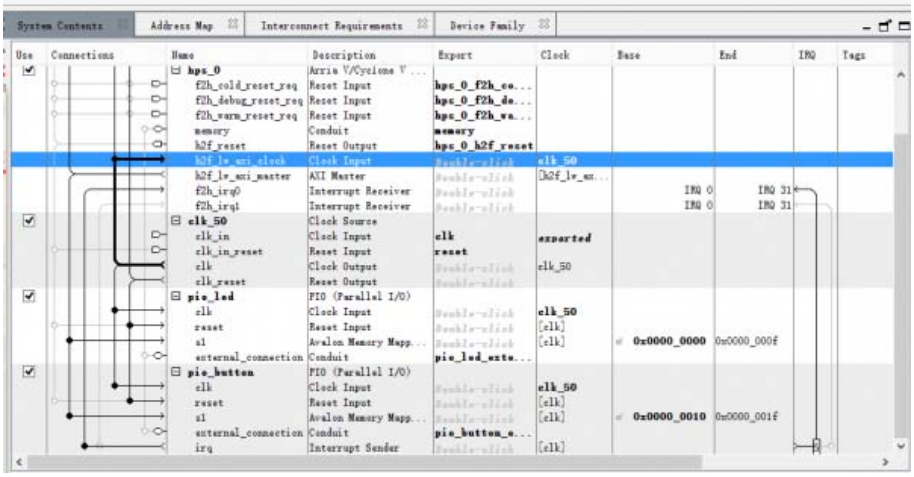


1） ADC 采集来的数据先进 FIFO 缓存

2） 在 FPGA 内部生成 IRQ 逻辑，并挂在 lwhps 上

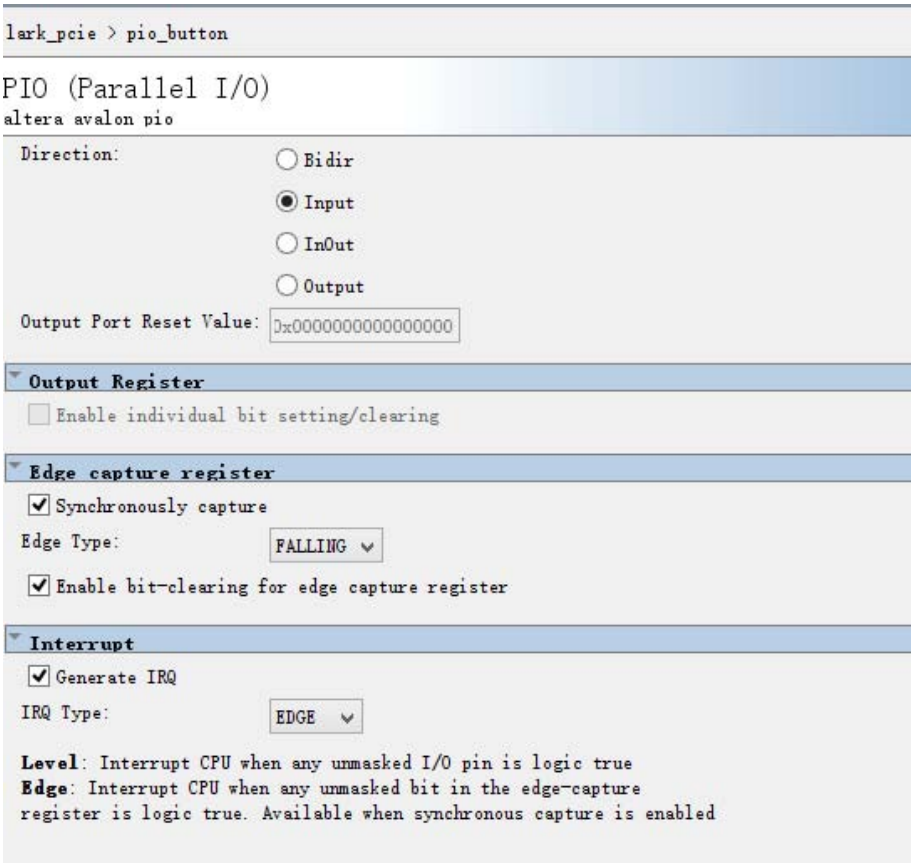
3、统生成

在 qsys 中我们生成如下的硬件系统，整体连接关系如下：

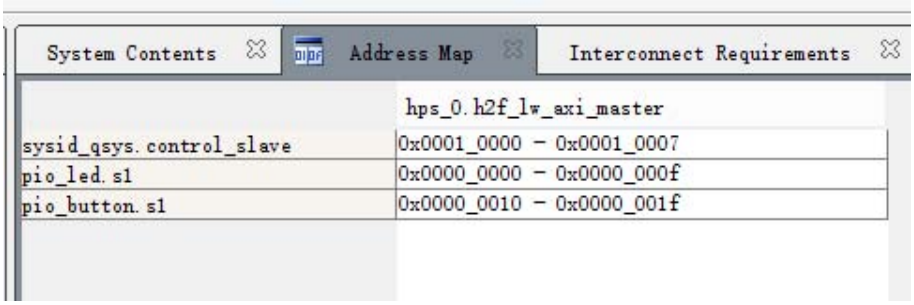


中断号应该是几呢？请看下面分解。

中断的参数设计如下：



地址映射关系如下：



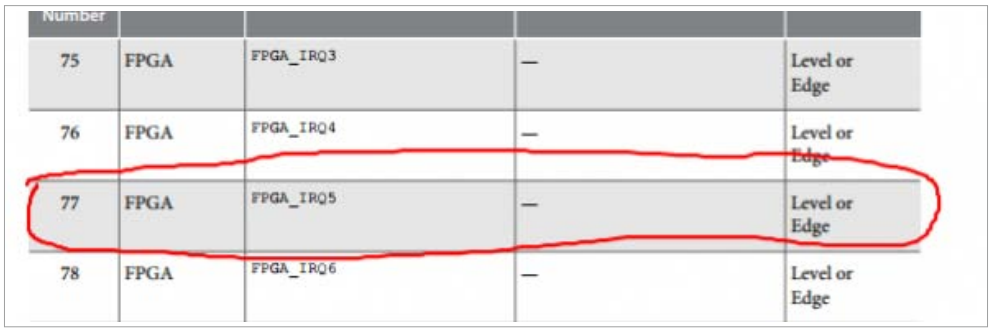
HDL Example 如下：

```
lark_pcie u0 (
        .memory_mem_a      (<connected-to-memory_mem_a>),          //          memory.mem_a
        .memory_mem_ba     (<connected-to-memory_mem_ba>),         //          .mem_ba
        .memory_mem_ck     (<connected-to-memory_mem_ck>),         //      .mem_ck
        .memory_mem_ck_n     (<connected-to-memory_mem_ck_n>),     //      .mem_ck_n
        .memory_mem_cke      (<connected-to-memory_mem_cke>),      //          .mem_cke
        .memory_mem_cs_n     (<connected-to-memory_mem_cs_n>),     //      .mem_cs_n
        .memory_mem_ras_n    (<connected-to-memory_mem_ras_n>),    //    .mem_ras_n
        .memory_mem_cas_n    (<connected-to-memory_mem_cas_n>),    //    .mem_cas_n
        .memory_mem_we_n     (<connected-to-memory_mem_we_n>),     //    .mem_we_n
        .memory_mem_reset_n    (<connected-to-memory_mem_reset_n>),  //  .mem_reset_n
        .memory_mem_dq       (<connected-to-memory_mem_dq>),       //    .mem_dq
        .memory_mem_dqs    (<connected-to-memory_mem_dqs>),       //        .mem_dqs
        .memory_mem_dqs_n    (<connected-to-memory_mem_dqs_n>),    //     .mem_dqs_n
        .memory_mem_odt    (<connected-to-memory_mem_odt>),       //        .mem_odt
        .memory_mem_dm    (<connected-to-memory_mem_dm>),        //        .mem_dm
        .memory_oct_rzqin  (<connected-to-memory_oct_rzqin>),     //       .oct_rzqin
        .clk_clk             (<connected-to-clk_clk>),           //    clk.clk
        .reset_reset_n      (<connected-to-reset_reset_n>),     //     reset.reset_n
        .hps_0_h2f_reset_reset_n      (<connected-to-hps_0_h2f_reset_reset_n>),     //
hps_0_h2f_reset.reset_n
        .hps_0_f2h_cold_reset_req_reset_n       (<connected-to-hps_0_f2h_col_reset_req_
reset_n>),       //        hps_0_f2h_cold_reset_req.reset_n
        .hps_0_f2h_debug_reset_req_reset_n      (<connected-to-hps_0_f2h_debug_reset_
req_reset_n>),      //       hps_0_f2h_debug_reset_req.reset_n
        .hps_0_f2h_warm_reset_req_reset_n       (<connected-to-hps_0_f2h_warm_reset_
req_reset_n>),       //        hps_0_f2h_warm_reset_req.reset_n
        .pio_led_external_connection_export     (<connected-to-pio_led_external_
connection_export>),     //     pio_led_external_connection.export
        .pio_button_external_connection_export (<connected-to-pio_button_external_
connection_export>)   // pio_button_external_connection.export
    );
```

做到这里硬件系统就设计完成了！

4、关于中断号

从上图看到，我们的中断是挂在 f2h_irq0 上，实际的物理中断是多少呢，需要认真看手册了



Number				
75	FPGA	FPGA_IRQ3	—	Level or Edge
76	FPGA	FPGA_IRQ4	—	Level or Edge
77	FPGA	FPGA_IRQ5	—	Level or Edge
78	FPGA	FPGA_IRQ6	—	Level or Edge

总共 FPGA 到 arm 的有 64 个中断源，分别分配在 f2h_irq0 和 f2h_irq1 上，所以上面的设计对应的物理中断号是 77，记住了哦，下面还有用！

5、产生中断源

简单的写了一个计数器，作为中断源，如下：

```
module count_int (
    input clk,
    input reset,

    output avl_irq
);

reg  [31:0] s_cnt;

always @(posedge clk or negedge reset)
    begin
        if (!reset)
            s_cnt <= 32'd0;
        else
            s_cnt <= s_cnt + 1'b1;
    end

assign avl_irq = s_cnt[24];

endmodule
```

把输出挂在 HPS 的模块上就可以

```
.pio_button_external_connection_export                    (avl_irq)
```

6、linux 驱动

1）首先要告知内核，要挂在这个中断啊，dts 中需要添加

```
Altera_pio: gpio@0xff200000{
```

```
                        compatible = “altr,pio-1.0”;
                        reg = <0xff200010 0x10>;
                        //reg = <0xff200000 0x10>;
                        interrupts = <0 45 4>;
                        //interrupts = <0 40 4>;
                        altr,gpio-bank-width = <32>;
                        altr,interrupt_type = <2>;                    //failing edge
                        #gpio-cells = <2>;
                        gpio-controller;
                        //inter_gpios = < &Altera_pio 0 1 >;
                        #interrupt-cells = <1>;
                        interrupt-controller;
                };
```

注意：

reg = <0xff200010 0x10>; 和上面的地址分配是对应的

interrupts = <0 45 4>;，这个 45 需要加上 32，那么和上面的中断号 77 也是对应的

2）挂载自己的驱动程序

还要告知内核，我用了它的哪个中断源，取个名字叫 key_int 吧

```
<blockquote>key_int {
```

自己写的简单驱动如下：

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/device.h>
#include <linux/platform_device.h>
#include <linux/leds.h>
#include <linux/io.h>
#include <linux/uaccess.h>
#include <linux/semaphore.h>
#include <linux/cdev.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/of.h>
#include <linux/of_gpio.h>
#include <linux/of_device.h>
#include <linux/gpio.h>
int key_gpio;
int key_irq;
/*****************************************/
// 中断处理函数：
static irqreturn_t key_interrupt (int irq, void *dev_id)
{
```

```
//        printk("=============key_interrupt============\n");
         return 0;
}

int button_release(struct platform_device *pdev)
{
    return 0;
}

int button_probe(struct platform_device *pdev)
{
         int result;
         struct device_node *node = pdev->dev.of_node;
         key_gpio = of_get_named_gpio(node,"inter_gpios",0);
         key_irq = __gpio_to_irq(key_gpio);
         printk("=============key_irq = %d============\n",key_irq);

         result = request_irq(key_irq, key_interrupt,IRQF_TRIGGER_FALLING, "key_int",
NULL);
  if(result)
        {
           printk(KERN_INFO"[FALLED: Cannot register Key  Interrupt!]\n");
        }

         return 0;
}

static struct of_device_id Altera_gpio_of_match[] = {
         { .compatible = "altr,key-int", },
         {},
};
MODULE_DEVICE_TABLE(of, Altera_gpio_of_match);

static struct platform_driver button_fops = {
         .driver = {
                  .name         = "key_int",
                  .owner         = THIS_MODULE,
                  .of_match_table = of_match_ptr(Altera_gpio_of_match),
         },
    .probe = button_probe,
    .remove = button_release,
};

// 模块加载函数
static int __init button_init(void)
{
    printk(KERN_ALERT "key modules is install\n");
    return    platform_driver_register(&button_fops);

}
```

```
// 模块卸载函数
static void  __exit button_cleanup(void)
{
    printk("Goodbye,chenzhufly!\n");
    platform_driver_unregister(&button_fops);
}

module_init(button_init);
module_exit(button_leanup);

MODULE_AUTHOR("chenzhufly");
MODULE_LICENSE("Dual BSD/GPL");
```

7、测试

```
root@arm:~# cat /proc/interrupts
           CPU0       CPU1
 29:       9775       9721       GIC  twd
160:     755920          0       GIC  ffb40000.usb, dwc2_hsotg:usb1
171:      56186          0       GIC  dw-mci
183:          0          0       GIC  ff705000.spi
186:          0          0       GIC  dw_spi0
190:       3488          0       GIC  ffc04000.i2c
194:        311          0       GIC  serial
201:          9          0       GIC  timer2
207:          0          0       GIC  fpga-mgr
256:        144          0  altera-gpio  key_int
IPI0:         0          0  CPU wakeup interrupts
IPI1:         0          0  Timer broadcast interrupts
IPI2:      3440       3817  Rescheduling interrupts
IPI3:         0          0  Function call interrupts
IPI4:         6          3  Single function call interrupts
IPI5:         0          0  CPU stop interrupts
Err:          0
root@arm:~#
```

看到 key_int 了吗？

8、小结

1）学习的过程总是苦逼的，但结果总是异常的简单明了，纵观下来，其实所涉及的东西并不多，但如何能在一开始就融汇贯通，难度相当的大啊

2）SoC 中断设计比传统的 arm+FPGA 架构要好使很多，避免很多麻烦的事情，自从调通以后，我就喜欢上了它

3）提醒一下 3.10 内核的 linux 版本，不支持 arm 端的 IO 中断，有需要的需要自己移植了，不要再在这个上面浪费时间了

详情请查看：

http://bbs.eeworld.com.cn/thread-459615-1-1.html

3.1.6 高速数据采集之数据传输（1）

1、 硬件环境

硬件平台： Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、系统设计

上回写到，中断是干什么的？中断就是通知 ARM 去接客的，那么 ARM 如何去接待尊贵的客人呢？，又如何能够获取到正确的数据呢？天空一声霹雳，HPS-FPGA Interfaces 闪亮登场！主要包含：

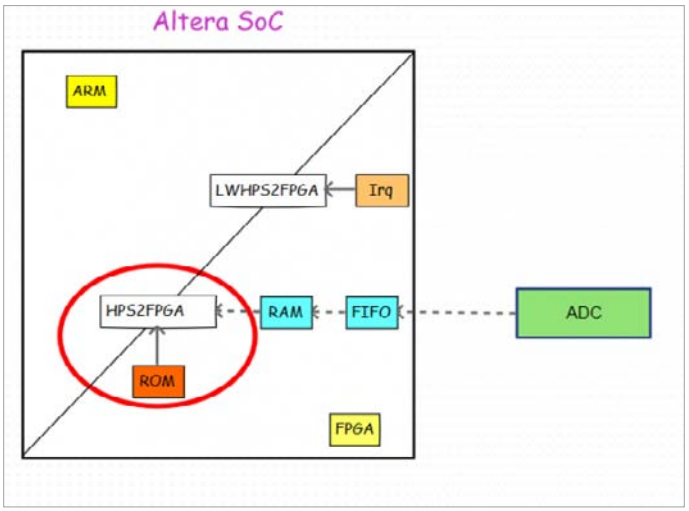
FPGA-to-HPS bridge

HPS-to-FPGA bridge

Lightweight HPS-to-FPGA bride

FPGA-to-HPS SDRAM interface

通过对 HPS-FPGA Interfaces 的深入学习和了解，我修改了高速数据采集的框架设计，系统框图如下：



1）中断挂在 Lightweight HPS-to-FPGA bridge

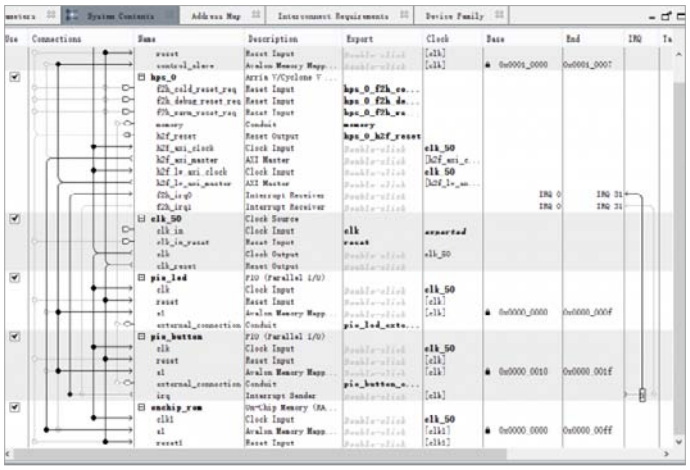
2)数据缓存挂在 HPS-to-FPGA bridge；Lightweight HPS-to-FPGA bridge 只有 2M 的寻址空间，感觉不太适合挂数据 buffer，HPS-to-FPGA bridge 有 640M 的寻址空间，够用了。

3）每中断一次，ARM 从 FPGA 的 ROM 中取一次数据，这是本次实验的点！

这次主要是测试一下红圈圈里面的东西，为进一步的数据传输做准备，先传个 ROM 的数据，这个方便测试！

3、qsys 的设计

下图是 qsys 的设计和连接关系如下图所示：



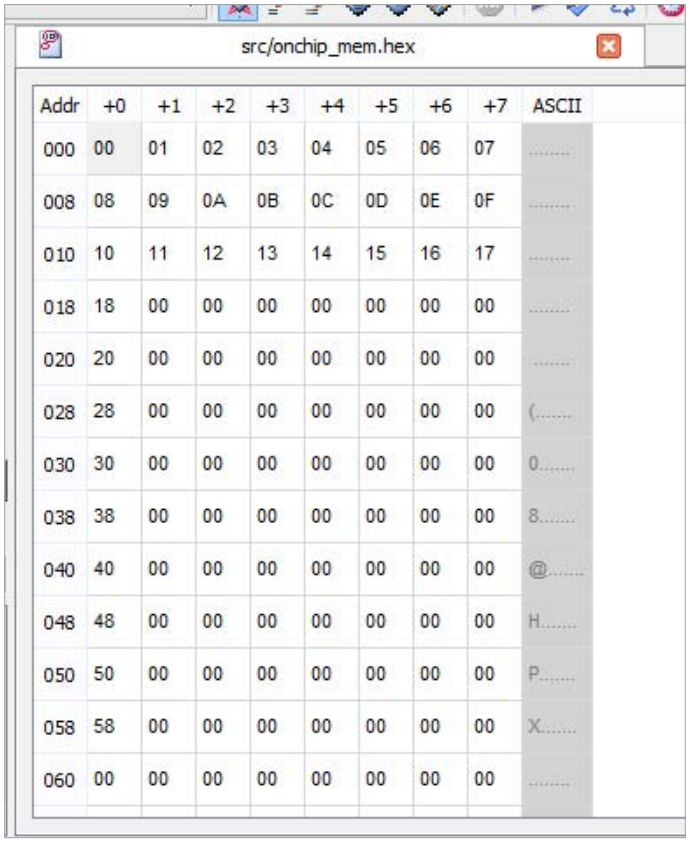
从图上可以看出 rom 挂载在 HPS-to-FPGA bridge 上，偏移地址为 0

rom 加载的文件方法可参照：

用 MATLAB 生成 mif 文件

http://bbs.eeworld.com.cn/thred-459695-1-1.html

当然也可以在 quartus 中新建一个 Memory files，然后自己手添测试数据，我就是这么干的，哈哈。下图是 ROM 的初始化数据



Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
000	00	01	02	03	04	05	06	07	.....
008	08	09	0A	0B	0C	0D	0E	0F	.....
010	10	11	12	13	14	15	16	17	.....
018	18	00	00	00	00	00	00	00	.....
020	20	00	00	00	00	00	00	00	.....
028	28	00	00	00	00	00	00	00	(.....
030	30	00	00	00	00	00	00	00	0.....
038	38	00	00	00	00	00	00	00	8.....
040	40	00	00	00	00	00	00	00	@.....
048	48	00	00	00	00	00	00	00	H.....
050	50	00	00	00	00	00	00	00	P.....
058	58	00	00	00	00	00	00	00	X.....
060	00	00	00	00	00	00	00	00	.....

4、驱动更新

在上回的驱动上做一些小小的修改，支持每次中断的时候把 ROM 中的数据取出来并打印到中断上；

1）定义一些宏

```
#define ROM_BASE 0xc0000000
#define ROM_SIZE 64
void *FPGA_rom_mem;
```

2）初始化

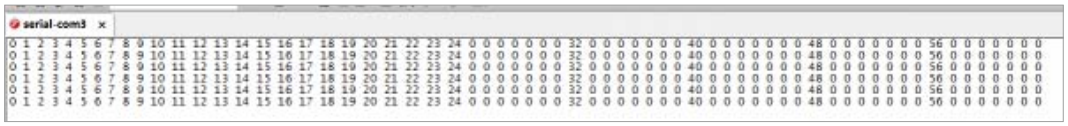
```
if (!request_mem_region(ROM_BASE,ROM_SIZE,"rom_test")) {
                printk( KERN_INFO"rom_test Memory region busy\n");
                return  -EBUSY;

        }
        FPGA_rom_mem = ioremap(ROM_BASE,ROM_SIZE);
        if(!FPGA_rom_mem) {
            printk( KERN_INFO" rom_test ioremap failed\n");
                return -EIO;
        }
```

3）中断处理数

```
    for (i=0;i < ROM_SIZE;i++){
                tmp = ioread8(FPGA_rom_mem + i);
    printk("%d ",tmp);
    }
    printk("\n");
```

5、测试结果



这是终端打印的结果，每中断一次打印一次，对比一下是不是和上面 ROM 的初始化文件一样一样的，说明整个数据链路已经通了！

6、小结

1）这次主要是测试一下 HPS-to-FPGA bridge，通过读取 ROM 中的数据来确认正确性，目前已经测试成功；

2）修改了一下系统设计，随着对 SoC 的不断认识，可能后续还会做一些调整，SoC 的内容太多了，一时半会学不透啊；

3）下一步准备把 ADC 采集的数据挂上去，做进一步的测试，欢迎大伙一起交流

详情请见：

http://bbs.eeworld.com.cn/thread-459742-1-1.html

3.1.7 高速数据采集之数据传输（2）

1. 硬件平台

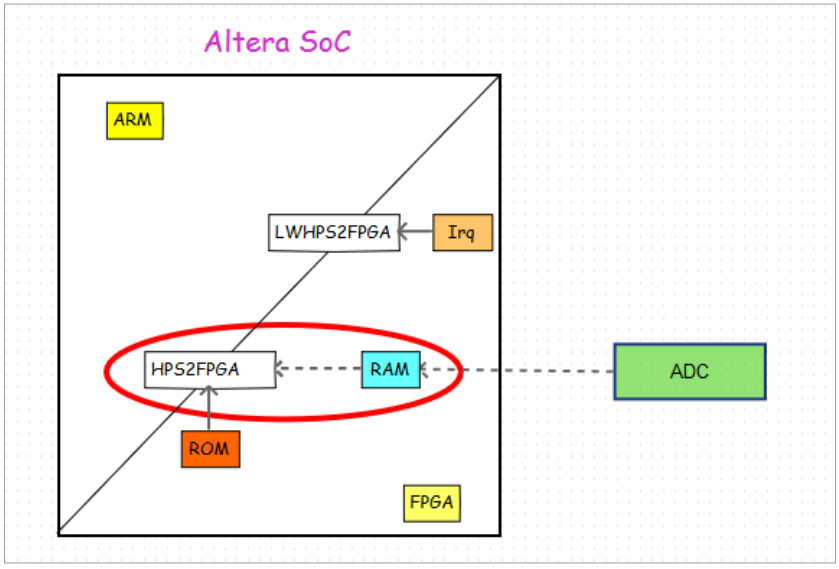
硬件平台： Embest SoC --LarkBoard

软件平台：发板 -linux-3.10.31

Quartus 14.0

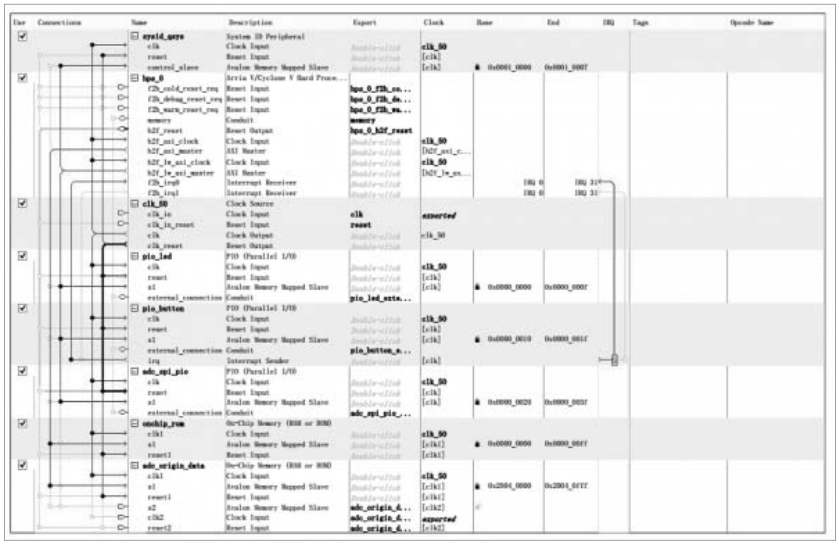
2. 系统概述

在上一篇文章已经成功进行了这样的测试，每中断一次，ARM 从 FPGA 的 ROM 中取一次数据，这边文章主要是测试一下，ADC 的数据经过双口 RAM 传递给 ARM 的过程，设计框图如下所示：



3、Qsys 设计

下图是 qsys 的设计和连接关系如下图所示：



1) 图所示 adc_origin_data 挂载在 HPS-to-FPGA bridge 上，偏移地址为 20040000(这个地址是可以任意改动的，注意和驱动对应就行了)

2) 增加了 adc_spi_pio 的控制接口，实现对 ADC 的配置和初始化，这个前面的测试中是没有注意的

quarus 工程文件见附件：  test-irq.zip (点击下载：http://bbs.eeworld.com.cn/thread-460022-1-1.html)

4、ADC 驱动设计

这里测试驱动主要使用了 Embest 自带的驱动，位于 drivers\char\adc9628.c，具体代码可见附件：  adc9628.zip (点击下载：http://bbs.eeworld.com.cn/thread-460022-1-1.html)

这是一个很好的参考用例，实际使用的时候还需要进一步的修改。上面 qsys 中偏移地址的设置就是为了和驱动相对应的。

主要完成以下功能：

1) 通过模拟 SPI 接口完成 ADC9628 的初始化

```
/*
*init the registers of ADC9628,
*write to Altera's PIO IP core to operate the gpio as spi pins
*/
static void bsp_adc_init(struct adc_priv_data *adc_data)
{
    int iadc_val  ;


    writew( 0x7,adc_data->regbase + (0x1*4));
/* pio_dir = 0x7 */
        writew( 0x6,adc_data->regbase + (0x4*4));
/* cs = 1  */
        writew( 0x2,adc_data->regbase + (0x4*4));
/* sclk=0 */
        mdelay(2);

    iadc_val = ad9628_read(0x1
);                                                              /*  chip
ID, 0x89 */

    printk(KERN_INFO"current id = %x \n", iadc_val);

    iadc_val = ad9628_read(0x2
);                                                              /*  chip
grade */
```

```
        iadc_val = ad9628_read(0x0b );
/* clock divide */
        iadc_val = 0x0;
        ad9628_write(0x0b, iadc_val);
        iadc_val = ad9628_read(0x0b );
    iadc_val = ad9628_read(0x09
);                                                          /* DUTY stabilty
*/
        iadc_val = ad9628_read(0x0b
);                                                          /* clock ratio */
        ad9628_write(0x05, 0x03)
;                                                                   /*  Ch
annel a b */
    iadc_val = ad9628_read(0x5 );
        ad9628_write(0x0d, 0x0);
/*  normal mode */

    iadc_val = ad9628_read(0xd );
        iadc_val = ad9628_read(0x101 );

        iadc_val |= (0x1<<7);
        ad9628_write(0x101, iadc_val);
        iadc_val = ad9628_read(0x101 );

        bsp_adc_cmos();
        iadc_val = ad9628_read(0x14 );

}
```

2）完成读双口 RAM 的操作

```
static const struct file_operations adc_fops = {
        .read                = adc_read,
        .open                = adc_open,
        .release                = adc_release,
        .owner                = THIS_MODULE,
};
```

3）把 ADC 注册成一个 char 型设备

```
static int __init adc9628_init(void)
{
        dev_t dev = MKDEV(LARK_ADC_MAJOR, 0);
        int ret;

        ret = register_chrdev_region(dev, max_adc_minors, "adc");
        if (ret)
                goto error;
```

```
        cdev_init(&adc_cdev, &adc_fops);
        ret = cdev_add(&adc_cdev, dev, max_adc_minors);
        if (ret) {
                goto error_region;
        }

        adc_class = class_create(THIS_MODULE, "adc");
        if (IS_ERR(adc_class)) {
                printk(KERN_ERR "Error creating adc class.\n");
                cdev_del(&adc_cdev);
                ret = PTR_ERR(adc_class);
                goto error_region;
        }
        adc_class->devnode = adc_devnode;
        device_create(adc_class, NULL, MKDEV(LARK_ADC_MAJOR, 0), NULL, "adc");

        if (!request_mem_region(BSP_ADC_ADDR,sizeof(u32),"adc9628")) {
                printk( "Memory region busy\n");
                return  -EBUSY;

        }
        gRegbase = ioremap_noache(BSP_ADC_ADDR, sizeof(u32));
        if(!gRegbase) {
            printk( KERN_INFO" ioremap failed\n");
                return -EIO;
        }

        if (!request_mem_region(BSPFIFO_BASE,siof(u32)*1024,"adc9628_fifo")) {
                printk( KERN_INFO"Memory region busy\n");
                return  -EBUSY;

        }
        gFifobase = ioremap_nocache(BSP_FIFO_BASE, sizeof(u32)*1024);
        if(!gFifobase) {
            printk( KERN_INFO" ioremap failed\n");
                return -EIO;
        }

        return 0;

error_region:
        unregister_chrdev_region(dev, max_adc_minors);
error:
        return ret;
}
```

至此 ADC 驱动介绍完毕，记得在编译系统的时候把驱动编译进去就行了

5、应用程序设计

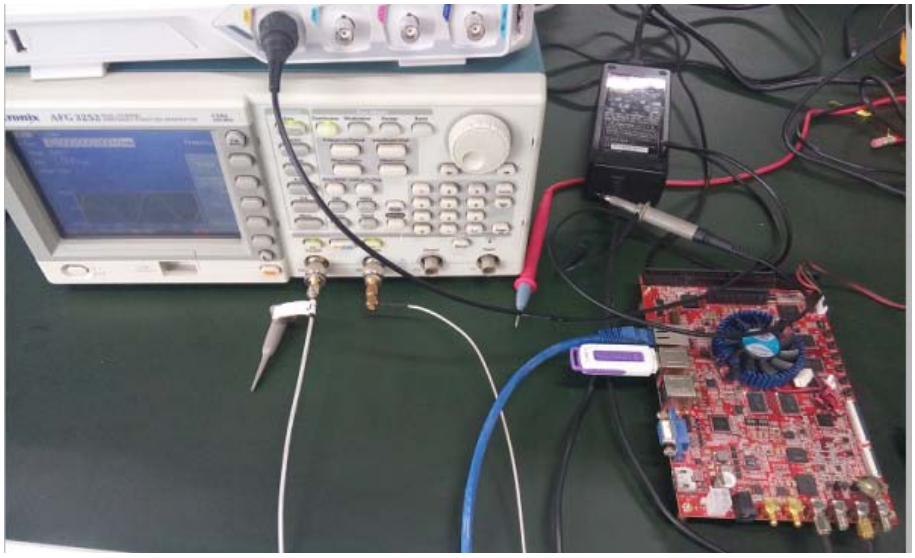
这里使用的也是 Embest 提供的测试程序，详细程序可见附件：adc_test.zip（点击下载：http://bbs.eeworld.com.cn/thread-460022-1-1.html)

测试程序主要完成了 FFT 计算，我也没有深入的研究，因为并不是我想要的东西，只是用它做个测试吧，有兴趣的自行研究吧，这里就不分析代码了；

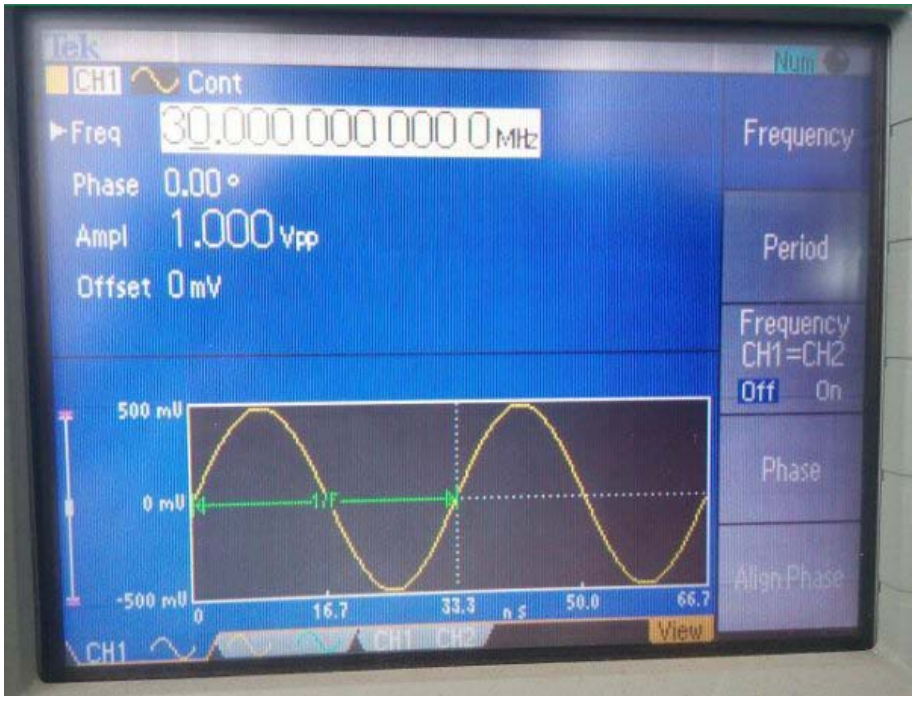
6、测试结果

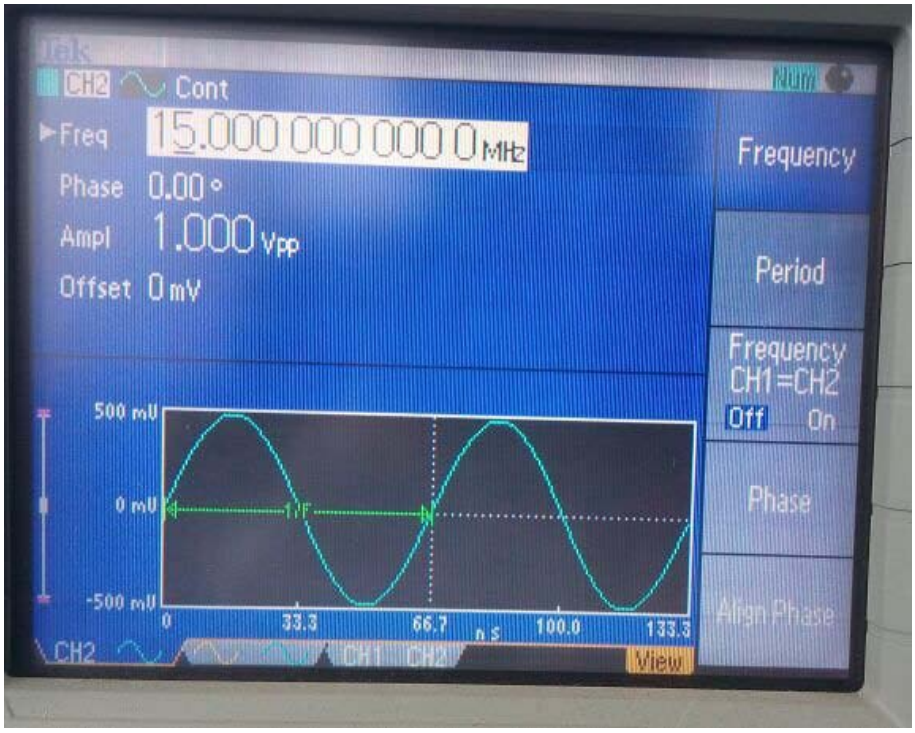
1）环境搭建

把信号发生器的两个输出口，分别接 ADC9628 的两个输入端，如下图所示：



2）设置信号发生器





3）运行测试程序

```
root@arm:~/adc# ./adc_test
current id = 89
dong fft
caculate real valule

146:chanel0 frequency = 14.970703, amplitude=2185

293:chanel1 frequency = 30.043945, amplitude=1806
```

从测试结果看 chanel0 的频率测量是 14.970703Mhz，chanel1 的频率测量是 30.043945Mhz，和实际设置值相似，且可以按照设定频率进行准确的跟踪，说明数据传输应该没有问题

7、小结

1） 本次主要测试 ADC 的采集数据通过双口 RAM 传递给 ARM 的过程，测试结果一切正常，为下一步的工作打下了基础；

2）HPS-to-FPGA bridge 接口传输的性能还有待进一步的测试，目前优先完成系统设计；

3）下一步准备把前面设计的中断，加入到驱动中去，这样驱动就更加实用了；

4）同时准备移植 web 服务器，把采集的数据能够在 web 上进行展示，这方面不擅长，有大侠可帮忙吗？非常感谢！

详情请点击：

http://bbs.eeworld.com.cn/thread-460022-1-1.html

3.1.8 高速数据采集之数据显示（1）

前面已经完成了数据的传输，下面主要谈谈数据的显示，本来打算是用 web 来显示的，但目前对于我来说还是有不小的难度的，暂且用 matlab 来辅助显示，并分析一下数据的正确性吧

1、 硬件环境

硬件平台： Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、数据获取

前面已经完成了 ADC 的驱动了，并且能够正确的获取到数据，这次的任务是把得到数据存入到文件中，便于后期的数据处理和分析，很自然的需要编写一个应用程序，把 ADC 通道 0 的数据存入 adc0_source，把 ADC 通道 1 的数据存入 adc1_source，程序如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
#define  POINT   (1024)
static unsigned int gSource_buff[POINT];
int main(int argc, char* argv[])
{
  int i = 0;
  int val;
  FILE * adc0Fp;
  FILE * adc1Fp;
        unsigned int tmp = 0;
         int fd;
        adc0Fp = fopen("/root/adc/adc0_source", "wb");
        if(!adc0Fp){
                perror("adc0_source open failed\n");
                return 1;
        }
        adc1Fp = fopen("/root/adc/adc1_source", "wb");
        if(!adc1Fp){
                perror("adc0_result open failed\n");
                return 1;
        }

    fd = open("/dev/adc/adc", O_RDONLY);
    if (fd ==  -1) {
                perror("device open error\n");
                return 1;
    }
```

```
        read(fd,gSource_buff,POINT*4);

        for(i=0;i<10;i++)
                printf("0x%x ",gSource_buff[];
               print(\n");

        for(i = 0;i < POINT; i++ ){
                tmp = gSource_buff[i]&0x0000ffff;                          //chanel 0
                if( (tmp&0x800) != 0){
//negative
                        tmp = (tmp | 0xfffff000);
                }
                fprintf(adc0Fp,"%d\n",tmp);

                tmp = (gSource_buff[i]&0xffff0000)>>16;                  //chanel 1
                if( (tmp&0x800) != 0){
//negative
                        tmp = (tmp | 0xfffff000);
                }
                fprintf(adc1Fp,"%d\n",tmp);
        }

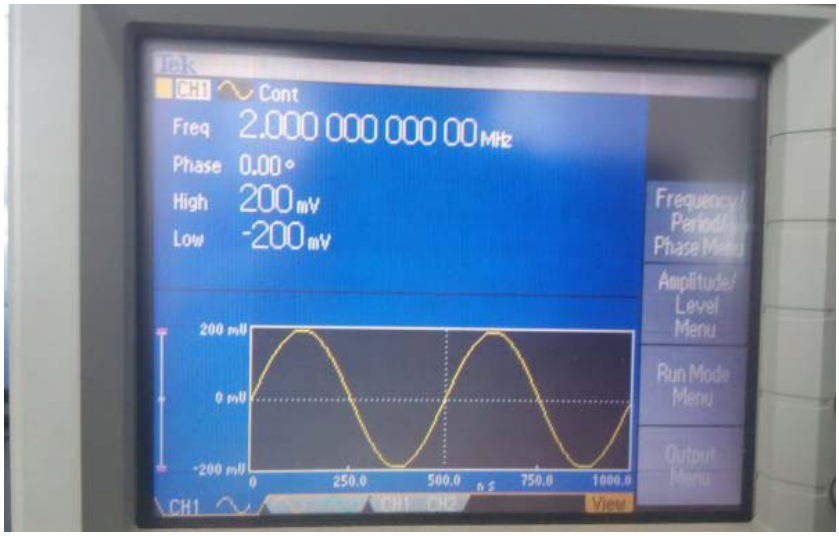
        return 0;
}
```

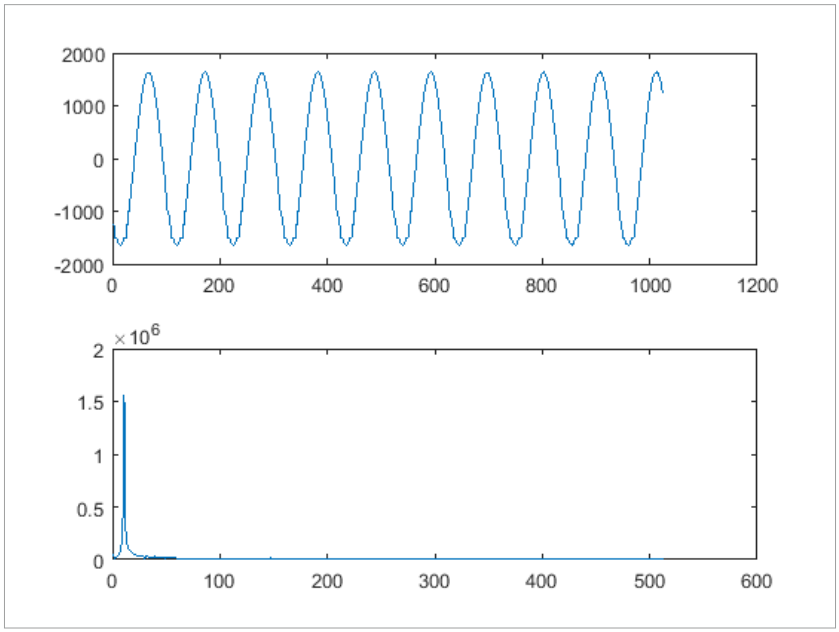
3、编写一个 matlab 程序，读取文件，并做 FFT 分析

```
clear all;
close all;
N=1024;
fid_adc1 = fopen('adc1_source','r');
adc1_data = fscanf(fid_adc1,'%d);
fclose(fidadc1);
figure;
subplot(2,1,1);
plot(adc1_data);
x = adc1_data';
y = fft(x,N);
subplot(2,1,2);
plot(2*abs(y(1:N/2+1)));
```

4、测试结果

1）正弦信号

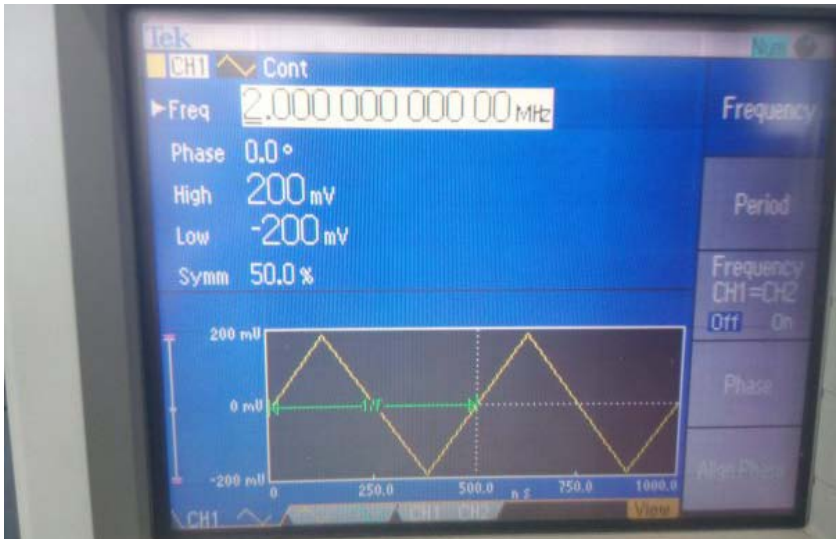


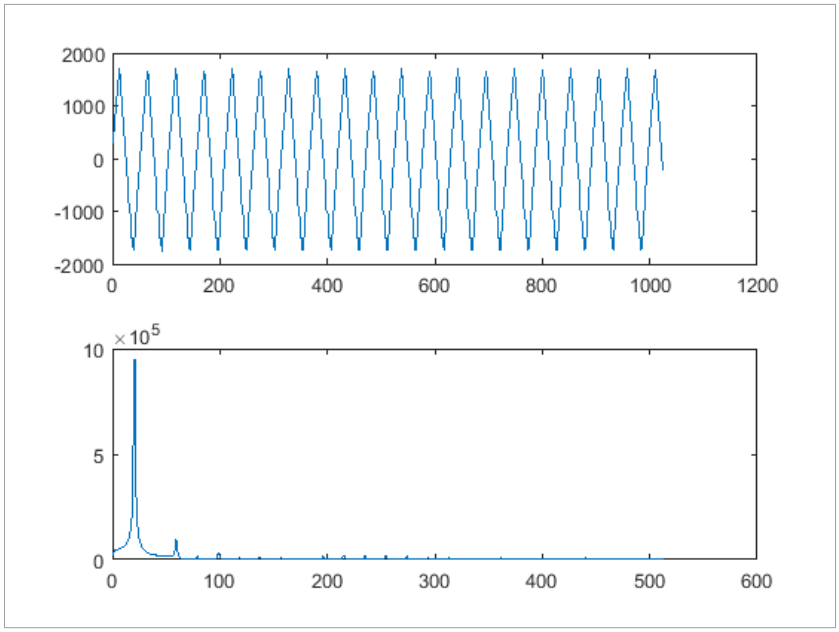


2）方波信号



3）三角波信号





5、小结

1）方波信号感觉怪怪的，畸变的比较严重，看的很明显；三角波信号在下面的时候也有一点畸变，原因不是很明确，我估计是硬件电路设计的问题；

2）从 matlab 分析的结果来看，信号获取应该是没什么问题了；

3）怎么用 web 显示这些结果呢？对于我来说是个头疼的问题，进一步努力吧

详情请查看：

http://bbs.eeworld.com.cn/thread-461339-1-1.html

3.1.9 高速数据采集之数据显示（2）

1、 硬件环境

硬件平台： Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、Web 服务器

我这次使用的是 Lighttpd

Lighttpd 是一个德国人领导的开源 Web 服务器软件，其根本的目的是提供一个专门针对高性能网站，安全、快速、兼容性好并且灵活的 web server 环境。具有非常低的内存开销、cpu 占用率低、效能好以及丰富的模块等特点。

Lighttpd 是众多 OpenSource 轻量级的 web server 中较为优秀的一个。支持 FastCGI，CGI，Auth，输出压缩 (output compress)，URL 重写，Alias 等重要功能；而 Apache 之所以流行，很大程度也是因为功能丰富，在 lighttpd 上很多功能都有相应的实现了，这点对于 apache 的用户是非常重要的，因为迁移到 lighttpd 就必须面对这些问题。

debian 系统就是这点好，直接安装就可以了

apt-get install lighttpd

配置文件在 /etc/lighttpd/lighttpd.conf，可自行更具需要修改

3、曲线显示控件

我使用的是 Highcharts

链接地址：http://www.hcharts.cn/，资源很丰富，可以下载例程，自己学习学习



演示曲线非常漂亮：



4、数据的获取

Highchart 支持多种方式的数据获取，由于能力有限，就没继续研究了，我使用了比较笨的办法，把数据存在一个文件中，然后从文件中数据读出并展示出来；数据都存入到 adcdata.js 这个文件，分别是原始数据 adcdata、FFT 后的数据 fftdata，信号频率 freq

数据获取的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
#define         PI                                              (3.14159f)
#define         TWOPI                                         (6.2831853f)
#define         POINT                                         (1024)
struct complex{
        double real;
        double imag;
}complex ;
static float  gFft_res_ch0[POINT];
static float  gFft_res_ch1[POINT];
static struct  complex gFft_sr_ch0[POINT];
static struct  complex gFft_sr_ch1[POINT];
static unsigned int gSource_buff[POINT];
static int gChanel0_buff[POINT];
static int gChanel1_buff[POINT];
struct complex EE(struct complex c1,struct complex c2)
{
        struct complex c3;
        c3.real        = c1.real*c2.real - c1.imag*c2.imag;
        c3.imag = c1.real*c2.imag + c1.imag*c2.real;
        return (c3);
}
void fft(struct complex *xin,int N)
```

```
{
        int f, m, LH, nm, i, k, j, L;
        int le, B, ip;
        double p , ps;
        struct complex w, t;
        LH = N/2;
        f  = N;
        for(m = 1;(f = f/2) != 1;m++);
        for(L = m;L >= 1;L--)
        {
                le = pow((double)2,(int)L);
                B  = le/2;
                for(j = 0;j <= B - 1;j++)
                {
                        p  = pow((double)2,(int)(m-L))*j;
                        ps = TWOPI/N*p;
                        w.real = cos(ps);
                        w.imag = -sin(ps);
                        for(i = j;i <= N - 1;i = i + le)
                        {
                                ip = i + B;
                                t  = xin[i];
                                xin[i].real    = xin[i].real  + xin[ip].real;
                                xin[i].imag    = xin[i].imag  + xin[ip].imag;
                                xin[ip].real   = xin[ip].real - t.real;
                                xin[ip].imag   = xin[ip].imag - t.imag;
                                xin[ip]        = EE(xin[ip],w);
                        }
                }
        }

        nm  = N - 2;
        j   = N / 2;
        for(i = 1;i <= nm;i++)
        {
                if(i < j)
                {
                        t  = xin[j];
                        xin[j] = xin[i];
                        xin[i] = t;
                }

                k = LH;
                while(j >= k)
                {
                        j = j - k;
                        k = k/2;
                }
                j = j + k;
        }
}
```

```
int main(int argc, char* argv[])
{
        int i = 0;
        int val;
        size_t write_len = 0;
        FILE * srFp;
        FILE * adcFp;
        double freq_ch0 = 0;
        double freq_ch1 = 0;

        int max_ch0 = 0;
        int max_num_ch0 = 0;
        int max_ch1 = 0;
        int max_num_ch1 = 0;
        unsigned int tmp = 0;
        int fd;
        adcFp = fopen("/www/pages/adcdata.js", "wb");
        if(!adcFp){
                perror("adcdata.js open failed\n");
                return 1;
        }

    fd = open("/dev/adc/adc", O_RDONLY);
    if (fd ==  -1) {
                perror("device open error\n");
                return 1;
    }

        read(fd,gSource_buff,POINT*4);
        fprintf(adcFp,"%s","var adcdata = [");
        for(i = 0;i < POINT; i++ ){
                tmp = (gSource_buff[i]&0xffff0000)>>16;                  //chanel 1
                if( (tmp&0x800) != 0){
//negative
                        tmp = (tmp | 0xfffff000);
                }
                gChanel1_buff[i] = (int)tmp;
                fprintf(adcFp,"%d, ",tmp);
                gFft_sr_ch1[i].real = gChanel1_buff[i];
        }
        fprintf(adcFp,"%s\n"," ];");
        fft(gFft_sr_ch1,POINT);
        fprintf(adcFp,"%s","var fftdata = [");
        for(i = 0;i < POINT/2 ;i++){
                gFft_res_ch1[i] = sqrt( gFft_sr_ch1[i].real*gFft_sr_ch1[i].real + gFft_sr_
ch1[i].imag*gFft_sr_ch1[i].imag ) * 2 / POINT; /* caculate the  amplitude ch1*/
                fprintf(adcFp,"%f, ",gFft_res_ch1[i]);
                if((int)gFft_res_ch1[i] > max_ch1){
                        max_ch1 = (int)gFft_res_ch1[i];
```

```
                        max_num_ch1 = i;
                }
        }
                fprintf(adcFp,"%s\n"," ];");
        close(fd);
        fprintf(adcFp,"%s","freq = ");
        freq_ch1 = 105.0 / POINT*max_num_ch1;
        printf("\n%d:chanel1 frequency = %f Mhz, amplitude=%d\n", max_num_ch1, freq_ch1,
max_ch1);
         fprintf(adcFp,"%d;",freq_ch1);
        return 0;
}
```

5、网页代码

利用 highcharts 的例程，自己写了一个 index.html，放在 /var/www 文件夹下面。

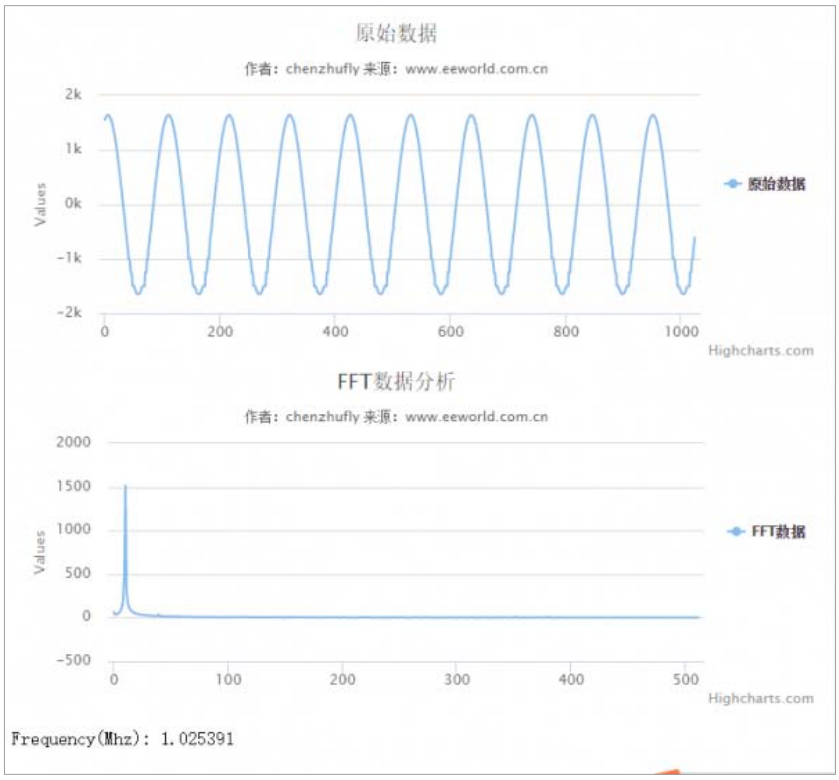
```
<!DOCTYPE HTML>
<html>
<meta http-equiv="refresh" content="5">
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
                <title>Data Acquisition System</title>
               <script type="text/javascript" src="http://cdn.hcharts.cn/jquery/jquery-
1.8.3.min.js"></script>
                <style type="text/css">
${demo.css}
                </style>
                <script src="adcdata.js"></script>
                <script type="text/javascript">
$(function () {
    $('#container').highcharts({
        title: {
            text: '原始数据',
            x: -20 //center
        },
        subtitle: {
            text: '作者: chenzhufly        来源: www.eeworld.com.cn',
            x: -20
        },
        tooltip: {
            valueSuffix: ''
        },
        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle',
            borderWidth: 0
        },
        series: [{
            name: '原始数据',
```

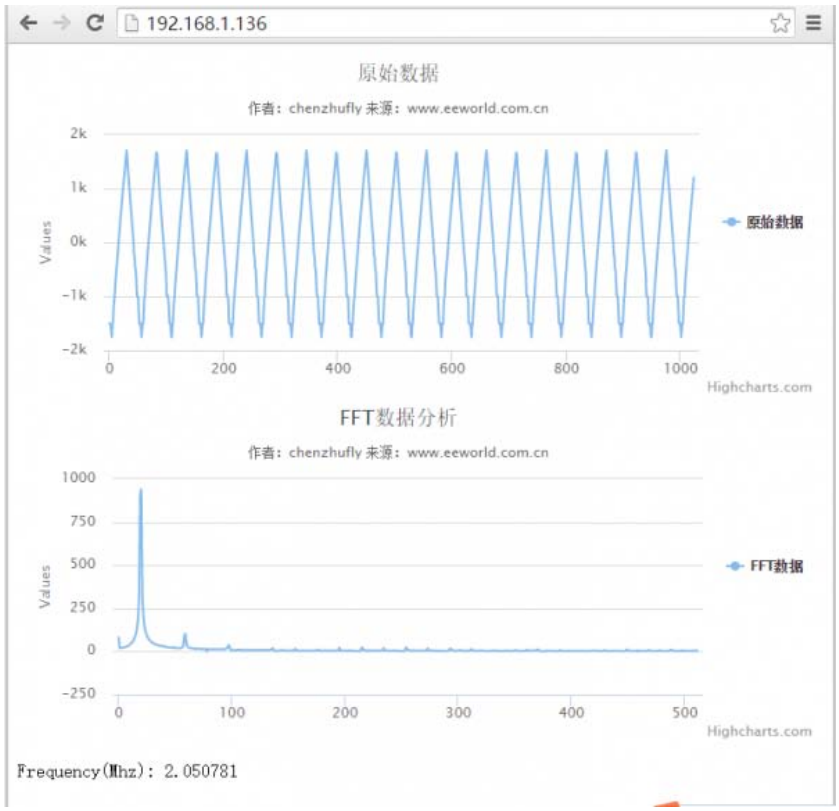
```
            data: adcdata
//            data : [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1, 14.1, 8.6, 2.5]
        }
        ]
    });
});
$(function () {
    $('#container1').highcharts({
        title: {
            text: 'FFT 数据分析 ',
            x: -20 //center
        },
        subtitle: {
            text: ' 作者: chenzhufly        来源: www.eeworld.com.cn',
            x: -20
        },
        tooltip: {
            valueSuffix: ''
        },
        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle',
            borderWidth: 0
        },
        series: [ {
            name: 'FFT 数据 ',
            data: fftdata
//            data : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
        }
        ]
    });
});
                </script>
        </head>
        <body>
<script src="http://cdn.hcharts.cn/highcharts/highcharts.js"° ></script>
<script src="http://cdn.hcharts.cn/highcharts/exporting.js"></script>
<div id="container" style="min-width: 250px; height: 300px; magi:  at"></div>
<div id="container1" style="min-width: 250px; height: 300px; margin: 0 auto"></div>
<FORM >
<p>Frequency(Mhz):
<script>document.write(freq);</script>
</FORM>
        </body>
</html>
```

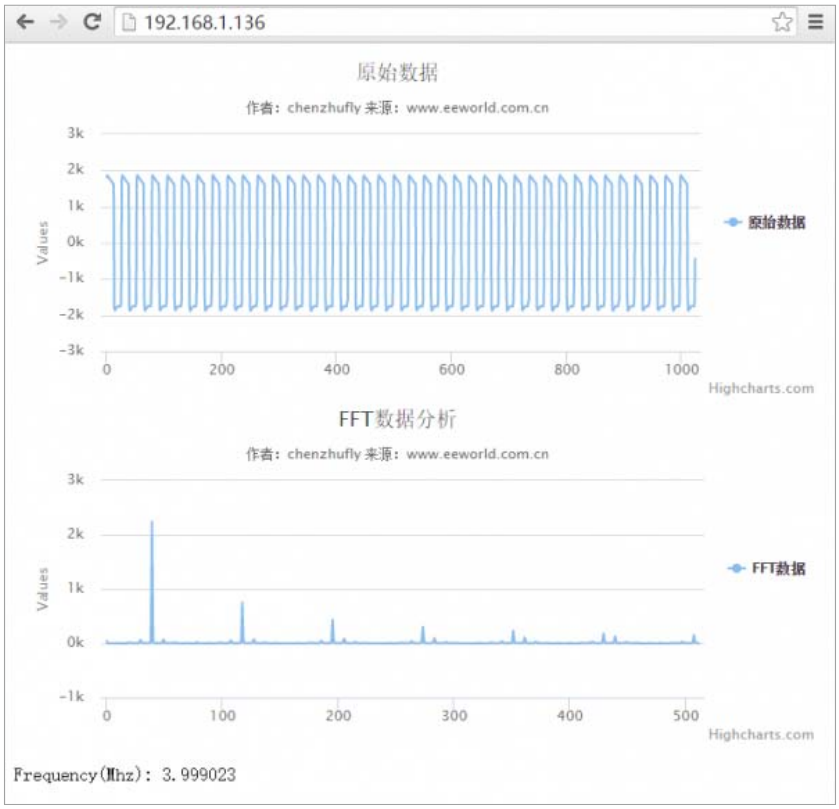
网页 5 秒钟自动刷新一次，使用如下的代码

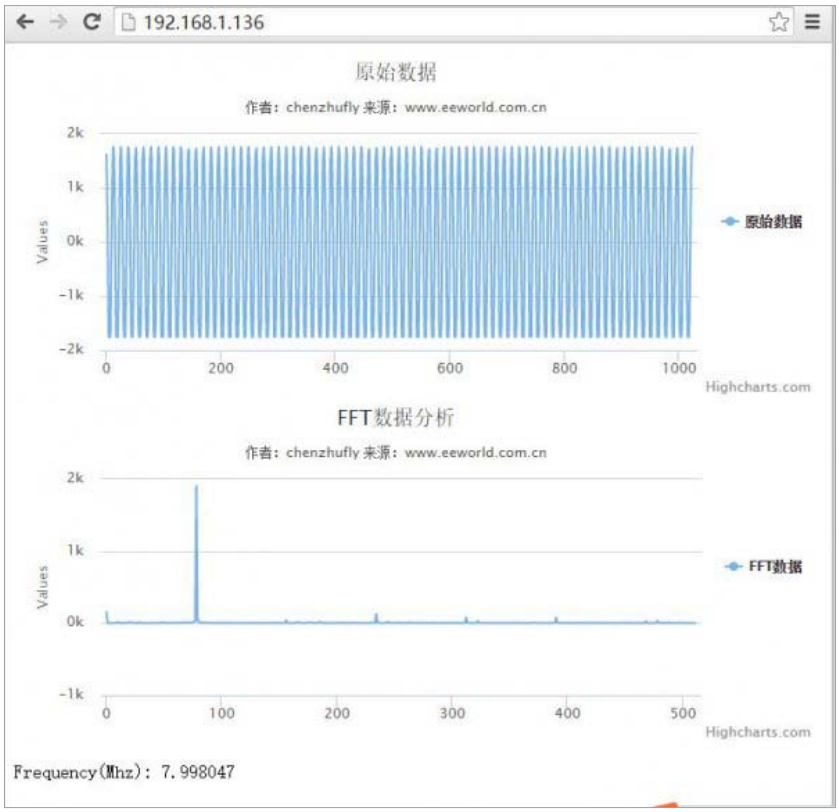
```
<meta http-equiv="refresh"  content="5">
```

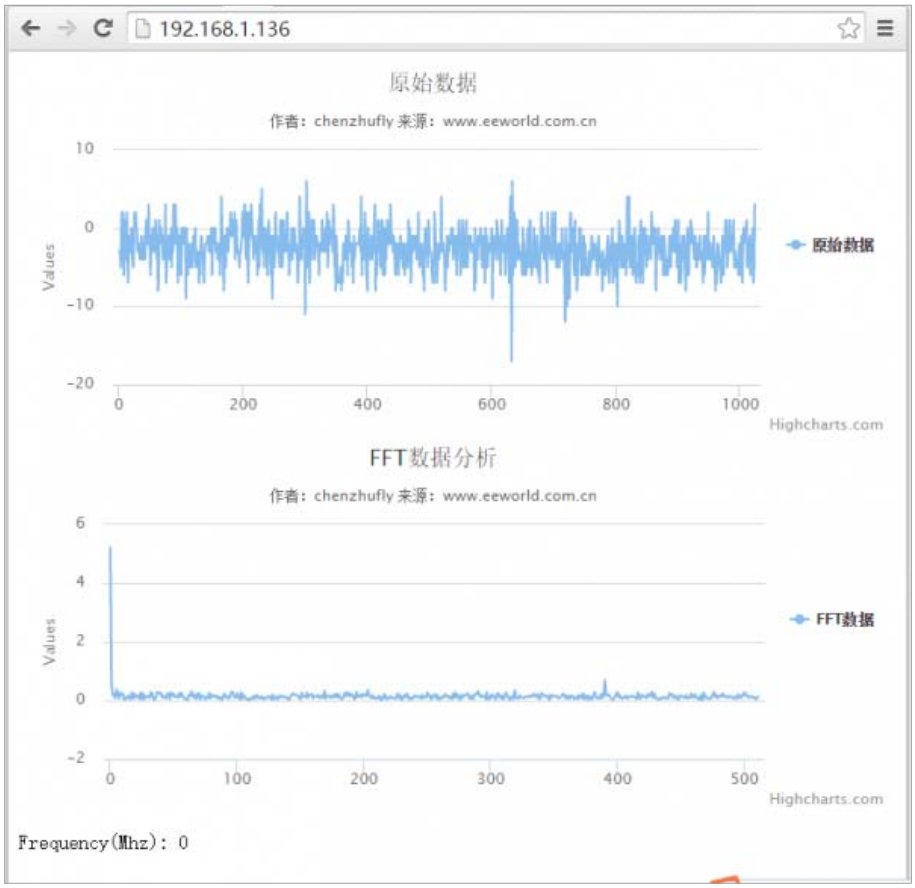
6、测试结果


原始数据
作者：chenzhufly 来源：www.eeworld.com.cn
2k
1k
0k
-1k
-2k
Values
0
200
400
600
800
1000
原始数据
Highcharts.com
FFT数据分析
作者：chenzhufly 来源：www.eeworld.com.cn
2000
1500
1000
500
0
-500
Values
0
100
200
300
400
500
FFT数据
Highcharts.com
Frequency(Mhz): 1.025391


192.168.1.136
原始数据
作者：chenzhufly 来源：www.eeworld.com.cn
2k
1k
0k
-1k
-2k
Values
0
200
400
600
800
1000
原始数据
Highcharts.com
FFT数据分析
作者：chenzhufly 来源：www.eeworld.com.cn
1000
750
500
250
0
-250
Values
0
100
200
300
400
500
FFT数据
Highcharts.com
Frequency(Mhz): 2.050781


192.168.1.136
原始数据
作者：chenzhufly 来源：www.eeworld.com.cn
3k
2k
1k
0k
-1k
-2k
-3k
Values
0
200
400
600
800
1000
原始数据
Highcharts.com
FFT数据分析
作者：chenzhufly 来源：www.eeworld.com.cn
3k
2k
1k
0k
-1k
Values
0
100
200
300
400
500
FFT数据
Highcharts.com
Frequency(Mhz): 3.999023


192.168.1.136
原始数据
作者：chenzhufly 来源：www.eeworld.com.cn
2k
1k
0k
-1k
-2k
Values
0
200
400
600
800
1000
原始数据
Highcharts.com
FFT数据分析
作者：chenzhufly 来源：www.eeworld.com.cn
2k
1k
0k
-1k
Values
0
100
200
300
400
500
FFT数据
Highcharts.com
Frequency(Mhz): 7.998047



详情请见：

http://bbs.eeworld.co.cn/thread-461500-1-1.hm

3.2 多串口服务器设计

32.1 概述

1、 硬件环境

硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、概述

为什么想要做个串口服务器呢？串口服务器的应用领域很广，主要应用门禁系统、考勤系统、售贩系统、POS 系统、楼宇自控系统、自助银行系统、电信机房监控、电力监控等；所以有一定研究价值和使用价值，况且 SoC 的架构很适用与这样的设计，能够完成相关的功能设计，同时又不缺乏灵活性和可扩展性。



一些现有产品的照片，这个可以从网上搜的

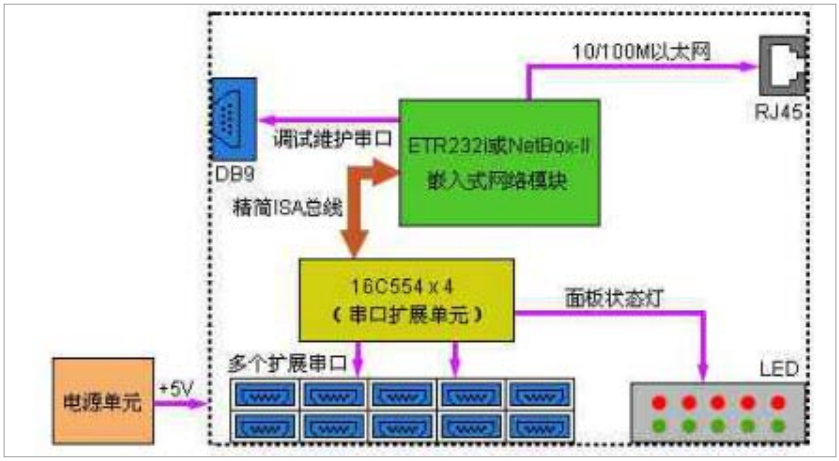


一个产品的介绍和组建方案：



3、一些方案介

1）见得最多的就是用 16C554 做扩展了，当然这必然还需要有个 CPU 来做管理，CPU 的性能也同时影响串口的数量和速率；



2）通过 SPI 或 I2C 接口扩展串口，比如 SC16IS752/762，OD2101/ OD2101A。

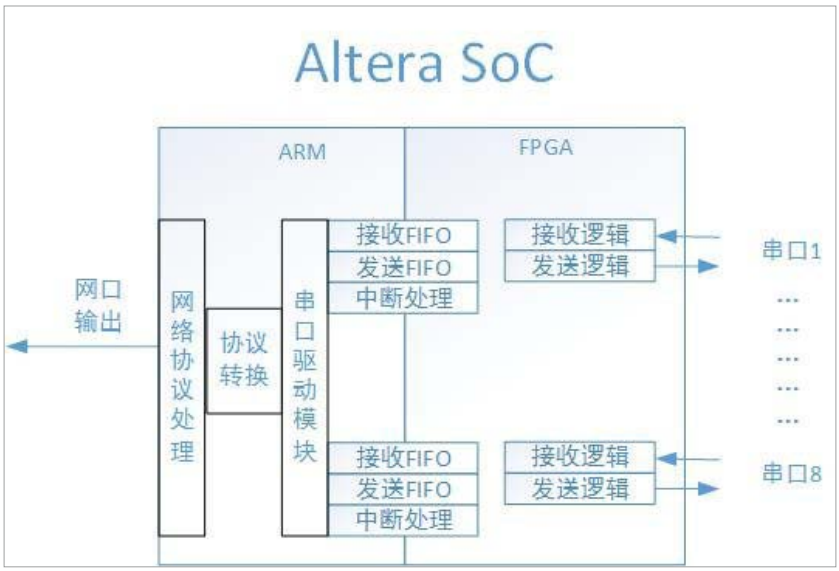
SC16IS75.pdf（点击下载：http://bbs.eeworld.com.cn/thread-463151-1-1.html)

3）FPGA 做扩展，这里我们准备用 SoC 来做，既有 ARM 来做管理，又有 FPGA 来做接口转换，一举两得。

4）其它的欢迎大家补充。

4、设计思路

准备按照下面框图的设计思路来进行设计：



设计到的技术要点：

1）带 FIFO 的串口逻辑设计

2）ARM 和 FPGA 之间的总线设计

3）ARM 和 FPGA 之间的中断管理设计

4）ARM 端的串口驱动设计

5）ARM 端的协议转换和数据处理模块设计

6）ARM 端的网络相关服务设计

5、小结

1）多串口服务器是一个比较常用的一款产品，应用的范围很广，希望在这里能够使用 Altera SoC 芯片来做一次全新的设计，增加灵活性和扩展性；

2）此次设计的内容很多，活动虽然快结束了，但如果哪位感兴趣的话，我们一起弄弄，期待早日能够出成果。

情请点：

http://bbs.eeworld.com.cn/thread-463151-1-1.html

3.2.2 多串口服务器之串口模块设计

1、 硬件环境

硬件平台：Embest SoC --LarkBoard

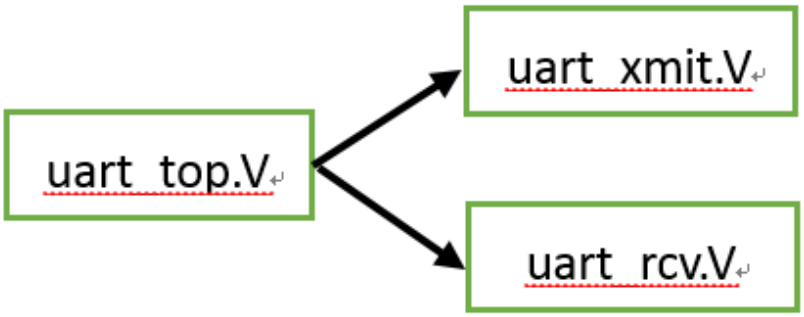
软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、代码结构

我不生产代码，我只是代码的搬运工，代码来源于网络，分享与网络。但还是需要做一些修改才能使用，慢慢来吧，结构还是不错的，我们一起学习学习。

主要有三个文件 uart_top.v，uart_xmit. v，uart_rcv. v，结构如下：



1）uart_top.v 是串口逻辑模块的顶层文件，负责管理串口的发送和接收

2）uart_xmit. v 为串口的发送模块

3）uart_rcv. v 位串口的接收模块

3、代码分析

1）uart_top.v

可以清楚的看到在，在接收模块和发送模块的前端都加了 fifo 模块，深度为 256 个字节。

```
`timescale 1 ns / 10 ps
`define BAUD_DIV 6875
module uart_top
(
        // Global signal definition
        inpt                            rst   ,
        input                           sys_clk    ,
        input                           pci_clk    ,
        input                           pci_wren   ,
        input        [7:0]                                  pci_wrdat               ,
        input                                               pci_rden                ,
        output        [7:0]                                 pci_rddat               ,
```

```
          // Local bus register
         input                                    SEND_CMD                ,
         input                                                RCV_EN     ,
         output                                               RCV_IRQ      ,
         input                                                PARITY_BIT_EN        ,
         input                                                PARITY_SEL           ,
         output                                               PARITY_ERROR         ,
         //output                        uart_ok        ,
         output                        xmit_busy        ,
         output                        xmit_ok          ,
         output                        xmit_fifo_empty  ,
         output                                                rcv_fifo_empty        ,
         // Uart interface
         inpt                                               rxd   ,
         output                                               txd
);
         wire                                               xmit_fifo_rden;
         wire                 [7:0]                      xmit_fifo_rddat;
         wire                                               rcv_fifo_wren;
         wire                 [7:0]                      rcv_fifo_wrdat;
         //wire                                                rcv_fifo_empty;
         /*-----------------------------------------------------------------------
         --  transmit module instantiation
         -----------------------------------------------------------------------*/
         T_FIFO_256x8         xmit_fifo_256x8
         (
                  .din                        (pci_wrdat                ),
                  .rd_clk             (sys_clk                ),
                  .rd_en              (xmit_fifo_rden        ),
                  .rst                (1'b0                        ),
                  .wr_clk                 (pci_clk                ),
                  .wr_en              (pci_wren                ),
                  .almost_empty       (                                ),
                  .almost_full        (                                ),
                  .dout               (xmit_fifo_rddat),
                  .empty              (xmit_fifo_empty),
                  .full               (                                )
         );
         uart_xmit uart_xmit_inst
         (
                  .sys_clk                (sys_clk                ),
                  .rst                        (rst                        ),
                   // PCI command interface
                  .SEND_CMD                (SEND_CMD                ),
                  .PARITY_BIT_EN        (PARITY_BIT_EN        ),        // parity bit
enable or not, 1 = enable, 0 = disable;
                  .PARITY_SEL                (PARITY_SEL                ),         //
determine ODD/EVEN parity, 1 = ODD,        1 = EVEN;
                 // UART command data fifofer interface
                  .fifo_rden                (xmit_fifo_rden        ),
```

```
                .fifo_rddat                 (xmit_fifo_rddat),
                .fifo_empty                 (xmit_fifo_empty),
                 // UART status interface
                .uart_busy                  (xmit_busy                ),
                .uart_ok                 (xmit_ok                ),
                // UART interface
                .txd                        (txd                      )
        );
        /*-------------------------------------------------------------------
        --  receive module instantiation
        -------------------------------------------------------------------*/
        T_FIFO_256x8        rcv_fifo_256x8
        (
                .din                        (rcv_fifo_wrdat        ),
                .rd_clk             (sys_clk                ),
                .rd_en              (pci_rden                ),
                .rst                (1'b0                        ),
                .wr_clk                 (sys_clk                ),
                .wr_en              (rcv_fifo_wren        ),
                .almost_empty       (                                ),
                .almost_full        (                                ),
                .dout               (pci_rddat                ),
                .empty              (rcv_fifo_empty        ),
                .full               (                                )
        );
        uart_rcv uart_rcv_inst
        (
                .sys_clk                (sys_clk                ),
                .rst                        (rst                        ),

                // PCI command interface
                .RCV_EN                         (RCV_EN                        ),
                .RCV_IRQ                (RCV_IRQ                ),
                .PARITY_BIT_EN        (PARITY_BIT_EN        ),        // parity bit
enable or not, 1 = enable, 0 = disable;
                .PARITY_SEL                (PARITY_SEL                ),         //
determine ODD/EVEN parity, 1 = ODD,        1 = EVEN;
                .PARITY_ERROR        (PARITY_ERROR        ),

                // UART command data fifofer interface
                .fifo_wren                (rcv_fifo_wren        ),
                .fifo_wrdat                (rcv_fifo_wrdat        ),
                .fifo_empty                (rcv_fifo_empty        ),

                // UART interface, which should be registered before internal access
                .rxd                        (rxd                        )
                 );
         endmodule
```

2)

```
uart_xmit. v
串口发送协议的解析和处理
`timescale 1 ns / 10 ps
`define BAUD_DIV 6875
module uart_xmit
(
        input                                                   sys_clk      ,
        inpt                                                    rst     ,

        // PCI command interface
        input                                                   SEND_CMD

,
        input                                                   PARITY_BIT_

EN        ,        // parity bit enable or not, 1 = enable, 0 = disable;
        input                                                   PARITY_SEL

,         // determine ODD/EVEN parity, 1 = ODD,        1 = EVEN;
         // UART command data fifofer interface
        output        reg                                           fifo_rden          ,
        input                          [7:0]                fifo_rddat                ,
        input                                               fifo_empty                ,
         // UART status interface
        output        reg                                   uart_busy                 ,
        output        reg                                   uart_ok       ,
         // UART interface
        output        reg                                   txd
);
parameter frame_idle = 3'b000, frame_putchar = 3'b001, frame_bit_call = 3'b010, frame_

done = 3'b011;
parameter bit_idle = 3'b000, bit_start = 3'b001, bit_xmit = 3'b010, bit_parity = 3'b011,

bit_stop = 3'b100;
        reg                 [2:0]        cur_frame;
        reg                 [2:0]        next_frame;
        reg                 [2:0]        cur_bit;
        reg                 [2:0]        next_bit;
        reg                 [7:0]        xmit_char;
        reg                                  xmit_start;
        reg                                  xmit_done;
        reg                 [3:0]        bit_cnt;
        reg                                  bit_over;
        reg                 [15:0]        baud_cnt;
        reg                                  shift_en;
        /*-------------------------------------------------------------------
        --  uart byte frame state control state machine
        -------------------------------------------------------------------*/
        always [url=home.php?mod=space&uid=496176]@[/url] ( * )
        begin
                case ( cur_frame )
                        frame_idle:
```

```
                                if ( SEND_CMD == 1'b1 )         // put frame char when
pci send command event detected
                                //if ( fifo_empty == 1'b0 )
                                        next_frame <= frame_putchar;
                                else
                                        next_frame <= frame_idle;
                        frame_putchar:
                                //if ( char_over == 1'b1 )
                                if ( fifo_empty == 1'b1 )
                                        next_frame <= frame_done;
                                else
                                        next_frame <= frame_bit_call;
                        frame_bit_call:
                            if ( xmit_done == 1'b0 )
                                next_frame <= frame_bit_call;
                            else if ( fifo_empty == 1'b1 )
                                next_frame <= frame_idle;
                            else
                                next_frame <= frame_putchar;
                                //if ( xmit_done == 1'b1 )
                                //        next_frame <= frame_putchar;
                                //else
                                //        next_frame <= frame_bit_call;
                        frame_done:
                                        next_frame <= frame_idle;
                        default:
                                        next_frame <= frame_idle;
                endcase
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        cur_frame <= frame_idle;
                else
                        cur_frame <= next_frame;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        xmit_start <= 1'b0;
                else
                        if ( cur_frame == frame_putchar )
                                xmit_start <= 1'b1;
                        else
                                xmit_start <= 1'b0;
        end

        /*-------------------------------------------------------------------
```

```
--  uart byte frame state control state machine
-------------------------------------------------------------------------*/
always @ ( * )
begin
        case ( cur_bit )
                bit_idle:
                        if ( xmit_start == 1'b1 )
                                next_bit <= bit_start;
                        else
                                next_bit <= bit_idle;
                bit_start:
                        if ( shift_en == 1'b1 )
                                next_bit <= bit_xmit;
                        else
                                next_bit <= bit_start;
                bit_xmit:
                        if ( bit_over == 1'b0 )
                                next_bit <= bit_xmit;
                        else if ( PARITY_BIT_EN == 1'b1 )
                                next_bit <= bit_parity;
                        else
                                next_bit <= bit_stop;
                bit_parity:
                        if ( shift_en == 1'b0 )
                                next_bit <= bit_parity;
                        else
                                next_bit <= bit_stop;
                bit_stop:
                        if ( shift_en == 1'b1 )
                                next_bit <= bit_idle;
                        else
                                next_bit <= bit_stop;
                default:
                        next_bit <= bit_idle;
        endcase
end

always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                cur_bit <= bit_idle;
        else
                cur_bit <= next_bit;
end

always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                xmit_done <= 1'b0;
        else
                if ( cur_bit == bit_stop && shift_en == 1'b1 )
```

```
                        xmit_done <= 1'b1;
                else
                        xmit_done <= 1'b0;
end

/*-------------------------------------------------------------------
--  bit counter control
-------------------------------------------------------------------*/
always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                bit_cnt <= 4'h0;
        else
                if ( cur_bit == bit_idle )
                        bit_cnt <= 4'h0;
                else if ( shift_en == 1'b1 )
                        bit_cnt <= bit_cnt + 1'b1;
end

always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                bit_over <= 1'b0;
        else
                if ( cur_bit == bit_idle )
                        bit_over <= 1'b0;
                else if ( bit_cnt == 4'h9 )
                        bit_over <= 1'b1;
end

/*-------------------------------------------------------------------
--  baud rate control
-------------------------------------------------------------------*/
always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                baud_cnt <= 16'b0_0000_0000;
        else
                if ( cur_bit == bit_idle || baud_cnt == `BAUD_DIV )
                        baud_cnt <= 16'b0_0000_0000;
                else
                        baud_cnt <= baud_cnt + 1'b1;
end

always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                shift_en <= 1'b0;
        else
                if ( baud_cnt == `BAUD_DIV )
                        shift_en <= 1'b1;
```

```
                else
                        shift_en <= 1'b0;
end

/*-------------------------------------------------------------------------
--  UART fifo interface
-------------------------------------------------------------------------*/
always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                fifo_rden <= 1'b0;
        else
                fifo_rden <= xmit_start;
end

/*-------------------------------------------------------------------------
--  TXD controll logic
-------------------------------------------------------------------------*/
always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                xmit_char <= 8'h00;
        else
                if ( cur_bit == bit_start )
                        xmit_char <= fifo_rddat;
                else if ( cur_bit == bit_xmit && shift_en == 1'b1 )
        //                xmit_char <= { xmit_char[6:0], xmit_char[7] };
                        xmit_char <= { xmit_char[0], xmit_char[7:1] };
end

always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                txd <= 1'b1;
        else if ( shift_en == 1'b1 )
                case ( cur_bit )
                        bit_start :
                                txd <= 1'b0;
                        bit_xmit:
        //                        txd <= xmit_char[7];
                                txd <= xmit_char[0];
                        bit_parity:
 txd <= ( xmit_char[7] ^ xmit_char[6] ^ xmit_char[5] ^ xmit_char[4] ^
  xmit_char[3] ^ xmit_char[2] ^ xmit_char[1] ^ xmit_char[0] ) ^ (~PARITY_SEL);
                        default:
                                txd <= 1'b1;
                endcase
end

always @ ( negedge rst or posedge sys_clk )
begin
```

```
                if ( rst == 1'b0 )
                        uart_ok <= 1'b0;
                else
                        if ( SEND_CMD == 1'b1 )
                                uart_ok <= 1'b0;
                        else if ( fifo_empty == 1'b1 && xmit_done == 1'b1 )
                                uart_ok <= 1'b1;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        uart_busy <= 1'b0;
                else
                        if ( cur_frame == frame_idle )
                                uart_busy <= 1'b0;
                        else if ( fifo_empty == 1'b0 )
                                uart_busy <= 1'b1;
        end
endmodule
```

3)

uart_rcv. v

串口接收协议的解析和处理

```
`timescale 1 ns / 10 ps
`define BAUD_DIV 6875
module uart_rcv
(
        input                           sys_clk       ,
        inpt                            rst           ,
         // PCI command interface
        input                                                   RCV_EN    ,
        output                                                  RCV_IRQ    ,
        input                                                   PARITY_BIT_
EN        ,        // parity bit enable or not, 1 = enable, 0 = disable;
        input                                                   PARITY_SEL   ,         //
determine ODD/EVEN parity, 1 = ODD,        1 = EVEN;
        output        reg                               PARITY_ERROR        ,
        // UART command data fifofer interface
        output        reg                                fifo_wren                ,
        output        reg                [7:0]                fifo_wrdat        ,
        input                                                   fifo_empty      ,
        // UART interface, which should be registered before internal access
        input                                                   rxd
);
        reg                                rxd_reg;
        reg                                rxd_reg_sft;
        reg                [7:0]        rxd_s0_cnt;
```

```
        reg                 [7:0]       rxd_s1_cnt;
        reg                             rxd_anti_glitch;
        reg                 [2:0]       cur_bit;
        reg                 [2:0]       next_bit;
        reg                 [7:0]       rcv_char;
        reg                             start_event;
        reg                             start_event_dly;
        reg                 [3:0]       bit_cnt;
        reg                             bit_over;
        reg                 [15:0]       baud_cnt;
        reg                             shift_en;
        reg                             sample_en;

  parameter bit_idle = 3'b000, bit_start = 3'b001, bit_rcv = 3'b010, bit_parity = 3'b011,
bit_stop = 3'b100;

        /*-------------------------------------------------------------------
        --  rxd anti-glitch control
        -------------------------------------------------------------------*/
        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        rxd_s0_cnt <= 0;
                else
                        if ( rxd == 1'b1 )
                                rxd_s0_cnt <= 0;
                        else if ( rxd_s0_cnt[4] == 1'b0 )
                                rxd_s0_cnt <= rxd_s0_cnt + 1'b1;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        rxd_s1_cnt <= 0;
                else
                        if ( rxd == 1'b0 )
                                rxd_s1_cnt <= 0;
                        else if ( rxd_s1_cnt[4] == 1'b0 )
                                rxd_s1_cnt <= rxd_s1_cnt + 1'b1;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        rxd_anti_glitch <= 1'b1;
                else
                        if ( rxd_s0_cnt[4] == 1'b1 )
                                rxd_anti_glitch <= 1'b0;
                        else if ( rxd_s1_cnt[4] == 1'b1 )
                                rxd_anti_glitch <= 1'b1;
```

```
end

/*-------------------------------------------------------------------------
--  start bit detection
-------------------------------------------------------------------------*/
always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
        begin
                rxd_reg <= 1'b1;
                rxd_reg_sft <= 1'b1;
        end
        else
        begin
                rxd_reg <= rxd_anti_glitch;
                rxd_reg_sft <= rxd_reg;
        end
end

always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                start_event <= 1'b0;
        else
                if ( rxd_reg == 1'b0 && rxd_reg_sft == 1'b1 &&
                     ( cur_bit == bit_idle || cur_bit == bit_stop ) )
                        start_event <= 1'b1;
                else
                        start_event <= 1'b0;
end

always @ ( negedge rst or posedge sys_clk )
begin
        if ( rst == 1'b0 )
                start_event_dly <= 1'b0;
        else
                start_event_dly <= start_event;
end

/*-------------------------------------------------------------------------
--  uart byte frame state control state machine
-------------------------------------------------------------------------*/
always @ ( * )
begin
        case ( cur_bit )
                bit_idle:
                        if ( start_event == 1'b1 )
                                next_bit <= bit_start;
                        else
                                next_bit <= bit_idle;
                bit_start:
```

```
                                if ( shift_en == 1'b1 )
                                        next_bit <= bit_rcv;
                                else
                                        next_bit <= bit_start;
                        bit_rcv:
                                if ( bit_over == 1'b0 )
                                        next_bit <= bit_rcv;
                                else if ( PARITY_BIT_EN == 1'b1 )
                                        next_bit <= bit_parity;
                                else
                                        next_bit <= bit_stop;
                        bit_parity:
                                if ( shift_en == 1'b0 )
                                        next_bit <= bit_parity;
                                else
                                        next_bit <= bit_stop;
                        bit_stop:
                                //if ( shift_en == 1'b1 )
                                //        next_bit <= bit_idle;
                                //else
                                //        next_bit <= bit_stop;
                                if ( start_event == 1'b1 ) // Caution!!! start_event
will occur in bit_stop phase
                                    next_bit <= bit_start;
                                else if ( shift_en == 1'b1 )
                                    next_bit <= bit_idle;
                                else
                                    next_bit <= bit_stop;
                        default:
                                next_bit <= bit_idle;
                endcase
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        cur_bit <= bit_idle;
                else
                        cur_bit <= next_bit;
        end


        /*---------------------------------------------------------------------
        --  bit counter control
        ---------------------------------------------------------------------*/
        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        bit_cnt <= 4'h0;
                else
```

```
                        if ( cur_bit == bit_idle || ( cur_bit == bit_stop && start_
event == 1'b1 ) )
                                bit_cnt <= 4'h0;
                        //else if ( shift_en == 1'b1 )
                        else if ( shift_en == 1'b1 && start_event_dly == 1'b0 ) //
modified by andrew, 2007-5-24 11:46
                                bit_cnt <= bit_cnt + 1'b1;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        bit_over <= 1'b0;
                else
                    if ( cur_bit == bit_idle || ( cur_bit == bit_stop && start_event ==
1'b1 ) )
                                bit_over <= 1'b0;
                        else if ( bit_cnt == 4'h9 )
                                bit_over <= 1'b1;
        end
        /*-------------------------------------------------------------------
        --  baud rate control
        -------------------------------------------------------------------*/
        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        baud_cnt <= 9'b0_0000_0000;
                else
                        //if ( cur_bit == bit_idle || baud_cnt == `BAUD_DIV )
                        if ( cur_bit == bit_idle ||
                           ( cur_bit == bit_stop && start_event == 1'b1 ) ||
 baud_cnt == `BAUD_DIV ) // to avoid error to be accumulated!!!!!
                                baud_cnt <= 9'b0_0000_0000;
                        else
                                baud_cnt <= baud_cnt + 1'b1;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        shift_en <= 1'b0;
                else
                        if ( baud_cnt == `BAUD_DIV )
                                shift_en <= 1'b1;
                        else
                                shift_en <= 1'b0;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
```

```
                if ( rst == 1'b0 )
                        sample_en <= 1'b0;
                else
                        if ( baud_cnt == (`BAUD_DIV >> 1) )
                                sample_en <= 1'b1;
                        else
                                sample_en <= 1'b0;
        end

        /*-------------------------------------------------------------------
        --  Character received from uart interface
        -------------------------------------------------------------------*/
        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        rcv_char <= 8'h00;
                else
                        if ( cur_bit == bit_rcv && sample_en == 1'b1 )
                                //rcv_char <= { rcv_char[6:0], rxd_reg };
                                rcv_char <= { rxd_reg, rcv_char[7:1] };
        end

        /*
        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        PARITY_ERROR <= 1'b0;
                else
                        if ( cur_bit == bit_parity && shift_en == 1'b1 )
PARITY_ERROR <= ( rxd_reg == ( getchar[7] ^ getchar[6] ^ getchar[5] ^ getchar[4] ^
 getchar[3] ^ getchar[2] ^ getchar[1] ^ getchar[0] ) ^ (~PARITY_SEL)
                                                        ) ? 1'b0 : 1'b1;
        end
        */

        /*-------------------------------------------------------------------
        --  FIFO control logic
        -------------------------------------------------------------------*/
        always @ ( negedge rst or posedge sys_clk )
        begin
                if ( rst == 1'b0 )
                        fifo_wren <= 1'b0;
                else
  if ( cur_bit == bit_stop && ( shift_en == 1'b1 || start_event == 1'b1 ) )
                                fifo_wren <= 1'b1;
                        else
                                fifo_wren <= 1'b0;
        end

        always @ ( negedge rst or posedge sys_clk )
        begin
```

```
                if ( rst == 1'b0 )
                        fifo_wrdat <= 0;
                else
                        fifo_wrdat <= rcv_char;
        end

        assign RCV_IRQ = ~fifo_empty;

endmodule
```

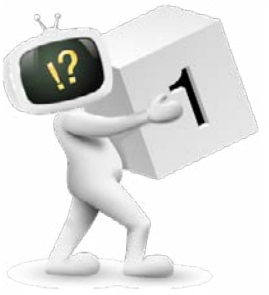
4、小结

1）这里主要展示了一下串口模块的逻辑和结构，不过还需要进一步的仿真和测试；

2）中断处理部分还没加进来。

详情请点击：

http://bbs.eeworld.com.cn/thread-463153-1-1.html





4. 用户体验与感想

4.1 关于 SoC 系统应用

曾经接触一个项目，涉及到一个非 ARM 的 CPU，DSP 和 FPGA，当时一看到板子，一下就觉得太复杂太麻烦，因为涉及的太广。从成本上说，每个芯片都是好几千上万，而且每个都需要仿真器，成本很高。从软件上说，每个芯片的公司不一样，开发环境的建立都需要太多的资源。艰苦的摸索下，建立起 CPU，DSP 和 FPGA 的联合 ... 辛酸泪啊。当时也考虑过 Altera 的 nois，觉得 nois 的 SOPC 系统，带宽，接口和软件使用上都不是很满意。

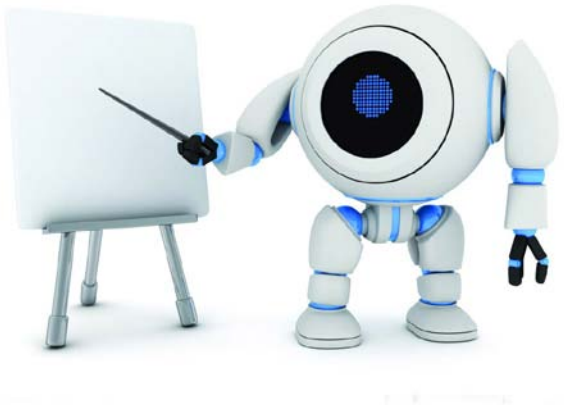
这次的 SoC 活动，当拿到 LARK board，看了 UG，觉得这个系统的还很能满足我接触的系统要求，很有优势。就跟着 lark board 的 UG，建软件环境，跟着 demo，启动操作系统 ...embest-tech 的服务还做的真不错，给点一个赞。

跟着已经提供的指令，没有什么好 show 的。

下一步就是想在这个基础上想建立我曾经遇到过的系统，通过 ARM 的操作系统去控制 FPGA 的应用，再把音频信号的 DSP 处理加在 FPGA 里面。通过以太网和 ARM 建立远程的应用操作 ... 不知道前面的路有多少坎坎 ... 如果真能够把这一套调通，就可以给老板建议用 SoC 了 ... 省钱省力啊

详情请点击：

http://bbs.eeworld.com.cn/thread-456102-1-1.html



4.2 基于 iperf 的 Altera SoC 千兆网测试

本文的目的是测试一下 Embest SoC 的千兆网功能和性能，使用常用网络测试工具的 Iperf。

Iperf 是一个网络性能测试工具。可以测试 TCP 和 UDP 带宽质量，可以测量最大 TCP 带宽，具有多种参数和 UDP 特性，可以报告带宽，延迟抖动和数据包丢失。Iperf 在 linux 和 windows 平台均有二进制版本供自由使用。

Iperf was developed by NLANRDAST as a modern alternative for measuring maximum TCP and UDP bandwidth performance. Iperf allows the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss.

1、环境搭建

硬件平台：Embest SoC --LarkBoard

软件平台：开发板 - 基于 linux-3.10.31 的 debian 系统，PC 机 WIN8 系统

开发板和 PC 机直连

开发板 IP 信息：192.168.1.11

PC 机 IP 信息：192.168.1.7

2、Iperf 下载与安装

1) iperf 下载地址

https://iperf.fr/download/iperf_2.0.5/iperf-2.0.5-3-win32.zip

Iperf 2.0.5-3 (1421 Kio) - The latest version of Iperf 2(2014). New: Fixed some performance issueswith iperf on windows byIuliu Rus (Google)

2) Embest SoC 下的 Iperf 安装

由于自带的的系统是 debian 系统，所以安装起来就方便多了，不需要进行源码的移植和编译，减轻了很多的工作量。

root@arm:~# apt-get install iperf

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following NEW packages will be installed:

Iperf

0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.

Need to get 51.1 kB of archives.

After this operation, 148 kB of additional disk space will be used.

Get:1 http://ftp.us.debian.org/debian/ wheezy/main iperf armhf 2.0.5-3 [51.1 kB]

Fetched 51.1 kB in 1s (33.7 kB/s)

Selecting previously unselected package iperf.

(Reading database ... 56671 files and directories currently installed.)

Unpacking iperf (from .../iperf_2.0.5-3_armhf.deb) ...

Processing triggers for man-db ...

Setting up iperf (2.0.5-3) ...

root@arm:~# iperf

Usage: iperf [-s|-c host] [options]

Try `iperf --help' for more information.

3、相关测试结果：

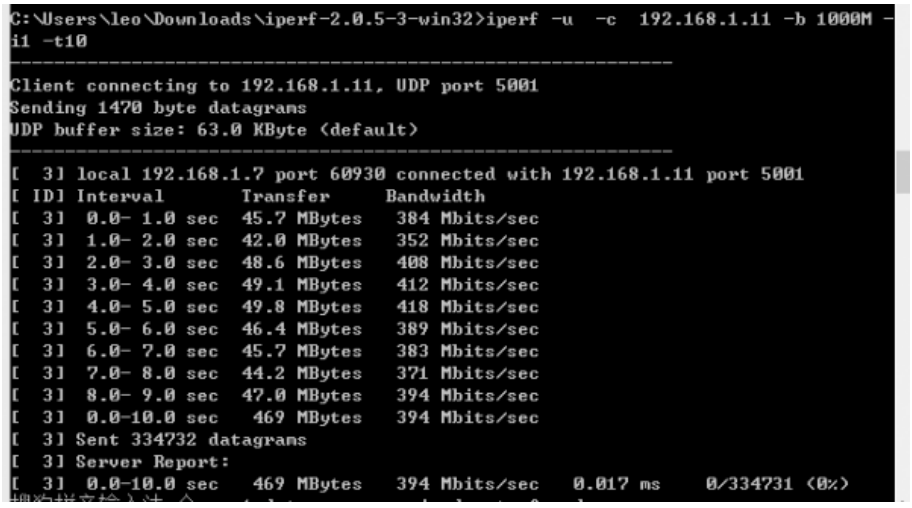
1）UDP 测试，开发板作为服务器，PC 机作为客户端

开发板作为服务器，输入：

```
root@arm:~#iperf -u -s
```

PC 机作为客户端：

iperf -u -c 192.168.1.11 -b 1000M -i1 -t0



2）UDP 测试，开发板作为客户端，PC 机作为服务器

开发板作为客户端，输入：

```
root@arm:~#iperf -u  -c  192.168.1.7 -b 1000M -i1 -t10
```

PC 机作为服务器：

```
iperf - -s
```

```
^Croot@arm:~# iperf -u  -c  192.168.1.7 -b 1000M -i1 -t10
------------------------------------------------------------
Client connecting to 192.168.1.7, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.11 port 34157 connected with 192.168.1.7 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  64.6 MBytes   542 Mbits/sec
[  3]  1.0- 2.0 sec  64.6 MBytes   542 Mbits/sec
[  3]  2.0- 3.0 sec  64.6 MBytes   542 Mbits/sec
[  3]  3.0- 4.0 sec  64.7 MBytes   543 Mbits/sec
[  3]  4.0- 5.0 sec  64.6 MBytes   542 Mbits/sec
[  3]  5.0- 6.0 sec  64.6 MBytes   542 Mbits/sec
[  3]  6.0- 7.0 sec  64.6 MBytes   542 Mbits/sec
[  3]  7.0- 8.0 sec  65.7 MBytes   551 Mbits/sec
[  3]  8.0- 9.0 sec  65.8 MBytes   552 Mbits/sec
[  3]  0.0-10.0 sec   650 MBytes   545 Mbits/sec
[  3] Sent 463378 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec   587 MBytes   493 Mbits/sec   0.018 ms 44315/463377 (9.6%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
root@arm:~#
```

3）TCP 测试，开发板作为服务器，PC 机作为客户端

开发板作为服务器，输入：

```
root@arm:~# iperf -s
```

PC 机作为客户端：

```
iperf -c  192.168.1.11 -b -i1-t10
```

```
C:\Users\leo\Downloads\iperf-2.0.5-3-win32>iperf -c 192.168.1.11 -i1 -t10
------------------------------------------------------------
Client connecting to 192.168.1.11, TCP port 5001
TCP window size: 63.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.7 port 63307 connected with 192.168.1.11 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  51.6 MBytes   433 Mbits/sec
[  3]  1.0- 2.0 sec  52.5 MBytes   440 Mbits/sec
[  3]  2.0- 3.0 sec  52.9 MBytes   444 Mbits/sec
[  3]  3.0- 4.0 sec  53.6 MBytes   450 Mbits/sec
[  3]  4.0- 5.0 sec  53.1 MBytes   446 Mbits/sec
[  3]  5.0- 6.0 sec  53.6 MBytes   450 Mbits/sec
[  3]  6.0- 7.0 sec  53.2 MBytes   447 Mbits/sec
[  3]  7.0- 8.0 sec  52.1 MBytes   437 Mbits/sec
[  3]  8.0- 9.0 sec  44.1 MBytes   370 Mbits/sec
[  3]  9.0-10.0 sec  48.0 MBytes   403 Mbits/sec
[  3]  0.0-10.0 sec   515 MBytes   432 Mbits/sec
```

4）TCP 测试，开发板作为客户端，PC 机作为服务器

开发板作为客户端，输入：

```
root@arm:~#iperf -c  192.168.1.7  -i1 -t10
```

PC 机作为服务器：

```
ipef -s
```

```
root@arm:~# iperf  -c  192.168.1.7  -i1 -t10
------------------------------------------------------------
Client connecting to 192.168.1.7, TCP port 5001
TCP window size: 20.7 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.11 port 48138 connected with 192.168.1.7 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  31.5 MBytes   264 Mbits/sec
[  3]  1.0- 2.0 sec  31.4 MBytes   263 Mbits/sec
[  3]  2.0- 3.0 sec  39.5 MBytes   331 Mbits/sec
[  3]  3.0- 4.0 sec  38.5 MBytes   323 Mbits/sec
[  3]  4.0- 5.0 sec  32.2 MBytes   271 Mbits/sec
[  3]  5.0- 6.0 sec  36.4 MBytes   305 Mbits/sec
[  3]  6.0- 7.0 sec  38.8 MBytes   325 Mbits/sec
[  3]  7.0- 8.0 sec  37.8 MBytes   317 Mbits/sec
[  3]  8.0- 9.0 sec  38.2 MBytes   321 Mbits/sec
[  3]  9.0-10.0 sec  38.4 MBytes   322 Mbits/sec
[  3]  0.0-10.0 sec   363 MBytes   304 Mbits/sec
```

测试数据汇总：

	服务器	客户端	速率
UDP 测试	开发板	PC 机	394Mbps
	PC 机	开发板	493 Mbps
TCP 测试	开发板	PC 机	409 Mbps
	PC 机	开发板	310 Mbps

4、测试总结

1） Embest SoC 自带的 debian 系统开发环境不完善，头文件之类的都无法安装；值得一提的是这个文件系统应该是从 Beaglebone Black 的 debian 系统修改过来的，整个文件系统中还留有 BBB 的痕迹

2）网络配置不完善，需要修改 /etc/network/interfaces

把 iface eth0 inet static 改为 iface eth0 inet dhcp，这样系统启动的时候才可以自动获取 IP

经过以上测试，可以看出 Altera SoC 的千兆网在 UDP 和 TCP 下的测试结果还是比较不错的，但测试的环境不够完善，测试结果仅供参考，有兴趣的可以深入的开展相关工作

详情请点击

http://bbs.eeworld.com.cn/thread-456177-1-1.html

4.3 使用 nbench 测试 Altera SoC 性能

1、硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

2、nbench 是什么？

nbench 是一个简单的用于测试处理器，存储器性能的基准测试程序。即著名的 BYTE Magazine 杂志的 BYTEmark benchmark program。nbench 在系统中运行并将结果和一台运行 Linux 的 AMD K6-233 电脑比较，得到的比值作为性能指数。由于是完全开源的，爱好者可以在各种平台和操作系统上运行 nbench，并进行优化和测试，是一个简单有效的性能测试工具。

主要测试三项内容：

memory 内存性能，主要体现处理器总线、CACHE 和存储器性能，INT 当然是整数处理性能；

interger 整数性能，主要体现整数处理性能；

float 浮点性能，主要体现双精度浮点性能；

3、nbench 的下载和安装：

下载地址：http://www.tux.org/~mayer/linux/bmark.html

在 Linux 中下载文件：

root@arm:~# wget http://www.tux.org/~mayer/linux/nbench-byte-2.2.3.tar.gz

--2015-03-23 10:51:37--

http://www.tux.org/~mayer/linux/nbench-byte-2.2.3.tar.gz

Resolving www.tux.org (www.tux.org)... 207.172.156.132

Connecting to www.tux.org (www.tux.org)|207.172.156.132|:80... connected.

HTTP request sent, awaiting response... 200 OK

Length: 111791 (109K) [application/octet-stream]

Saving to: `nbench-byte-2.2.3.tar.gz'

100%[======================================>] 111,791 7.87K/s in 12s

2015-03-23 10:51:50 (8.99 KB/s) - `nbench-byte-2.2.3.tar.gz' saved [111791/111791]

解压编译，直接在目录下执行 make

root@arm:~/nbench-byte-2.2.3# make

-bash: make: command not found

如果没有 make 命令，需要先安装

root@arm:~/nbench-byte-2.2.3# apt-get install make

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following packages were automatically installed and are no longer required: libmozjs24d xulrunner-24.0

Use ‘apt-get autoremove’ to remove them.

Suggested packages:

make-doc

4、测试结果：

编译后在目录下有个 nbench 可执行文件，运行结果如下：

```
root@arm:~/nbench-byte-2.2.3# ./nbench
BYTEmark* Native Mode Benchmark ver. 2 (10/95)
Index-split by Andrew D. Balsa (11/97)
Linux/Unix* port by Uwe F. Mayer (12/96,11/97)

·  TEST                 : Iterations/sec.  : Old Index    : New Index
                       :                  : Pentium 90* : AMD K6/233*
--------------------:------------------:-------------:------------
NUMERIC SORT         :           448.56  :       11.50  :        3.78
vgaarb: this pci device is not a vga device
STRING SORT          :           55.196  :       24.66  :        3.82
BITFIELD             :       1.6402e+08  :       28.13  :        5.88
FP EMULATION         :            62.63  :       30.05  :        6.93
FOURIER              :           5651.6  :        6.43  :        3.61
ASSIGNMENT           :            7.095  :       27.00  :        7.00
IDEA                 :           1408.2  :       21.54  :        6.39
HUFFMAN              :           764.71  :       21.21  :        6.77
NEURAL NET           :           7.5337  :       12.10  :        5.09
LU DECOMPOSITION     :           284.12  :       14.72  :       10.63
==========================ORIGINAL BYTEMARK RESULTS==========================
INTEGER INDEX        : 22.542
FLOATING-POINT INDEX: 10.461
Baseline (MSDOS*)    : Pentium* 90, 256 KB L2-cache, Watcom* compiler 10.0
============================LINUX DATA BELOW============================
CPU                  : Dual ARMv7 Processor rev 0 (v7l)
L2 Cache             :
OS                   : Linux 3.10.31-ltsi-05017-g4ce5333-dirty
C compiler           : gcc version 4.6.3 (Debian 4.6.3-14)
libc                 : libc-2.13.so
```

```
MEMORY INDEX         : 5.396
INTEGER INDEX        : 5.804
FLOATING-POINT INDEX: 5.802
Baseline (LINUX)     : AMD K6/233*, 512 KB L2-cache, gcc 2.7.2.3, libc-5.4.38
* Trademarks are property of their respective holder.
```

Altera SoC 使用 nbench 测试的性能指数如下：

MEM 性能指数：5.396

INT 性能指数：5.804

FP 性能指数：5.802

详情请点击：

http://bbs.eeworld.com.cn/thread-458125-1-1.html



4.4 主流平台的 nbench 性能测试大 PK

硬件平台：Embest SoC --LarkBoard

软件平台：开发板 -linux-3.10.31

Quartus 14.0

接前一篇文章《【Altera SoC 体验之旅】使用 nbench 测试 Altera SoC 性能》http://bbs.eeworld.com.cn/thread-458125-1-1.html

Altera SoC 使用 nbench 测试的性能指数如下：

MEM 性能指数：5.396

INT 性能指数：5.804

FP 性能指数：5.802

以下针对主流平台的 nbench 测试结果进行一些横向的比较，看看 SoC 处于什么样的位置，数据来源于网络，仅供参考，有不足之处欢迎指正

1、与常用 cpu 的横向比较：

数据来源：http://bbs.eetop.cn/thread-262963-1-1.html

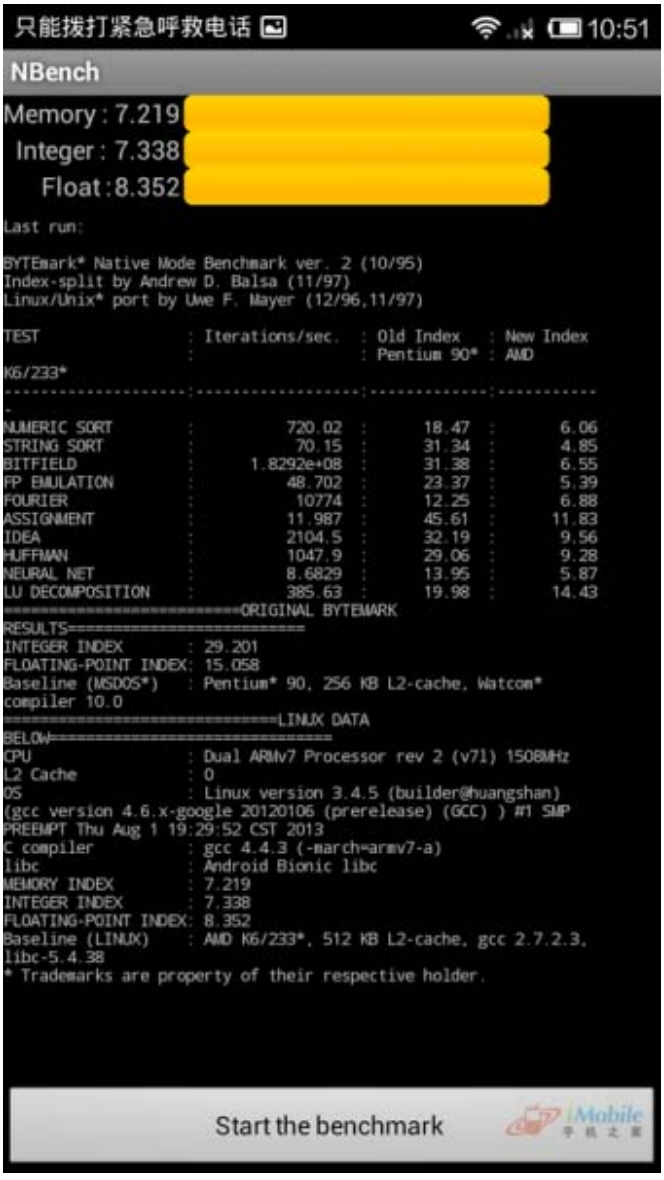
处理器	MEM 性能指数	INT 性能指数	FP 性能指数
Pentium MMX 166MHZ	0.51	0.53	0.85
S3C2410 200MHz	0.67	0.76	0.01
PXA255 400MHz	0.94	1.26	0.59
PXA270 520MHz	1.16	1.82	0.99
OMAP2420 400MHz	1.17	1.86	1.43
PPC440 400MHz	1.46	1.67	0.04
PentiumIII 800MHz	3.74	3.39	6.62
OMAP3530 600MHz	3.04	3.57	0.98
Altera SoC	5.396	5.804	5.802

Altera SoC 在以上的芯片测试对比中表现不俗，仅仅在 FP 性能指数上略逊色与 PentiumIII 800MHz。不过上面的芯片从目前看来有些过时了

2、红米手机 nbench 测试结果

数据来源：http://www.5577.com/d/14885_all.html
红米手机硬件参数：

CPU：MTK MT6589 1.5GHz 四核心
GPU：PowerVR SGX544MP、OpenGL ES 2.0
Display：4.7 英寸、分辨率 1280×720 、320dpi、312ppi
RAM：1GB
ROM：4GB
OS：MIUI V5（Android 4.2.1）
Network mode：GSM/TD-SCDMA 双卡双待
Camera：后置 800 万像素、前置 130 万像素
Sensors：陀螺仪、重力感应器、电子罗盘、光线感应器、距离感应器
Battery：2000/2050 mAh（min/typ）、可拆卸



从测试结果来看 MTK MT6589 1.5GHz 四核心的平台各项指标还是要强于 Altera SoC 的，不过

MT6589 是四核心且主频也比 Altera SoC 要高，性能强一些也是可以理解的；

3、RK3188 平板 nbench 测试结果

数据来源：

http://mp3.zol.com.cn/352/3529308_all.html#p3529308

RK3188 是中国首款采用 28 纳米制程工艺的芯片，是 Rockchip 公司 2013 年针对移动市场推出最新处理器方案！ RK3188 拥有成熟稳定的架构，集成了四核 ARM Cortex-A9 CPU。 GPU 采用强大的四核 ARM Mali-400，CPU 最高频率可达 1.8GHz，而 GPU 最高频率可达 533MHz。并内嵌高性能 2D 加速硬件，配备 High-Speed Inter Chip 高速通信接口，完美支持超高清分辨率。是具备高性能，低功耗特性的新一代四核 A9 处理器！



与 Altera SoC 相比，MEM 和 INT 性能参数旗鼓相当，但 FP 性能参数逊色与 Altera SoC，内核比较挫？

4、P1000(三星 galaxy tab) 测试结果

数据来源：http://bbs.imp3.net/thread-10297898-1-1.html

MEM 性能指数：2.947

INT 性能指数：4.017

FP 性能指数：0.771

浮点能力正常，总体差远了

5、飞思卡尔 A8 800MHZ 的测试结果

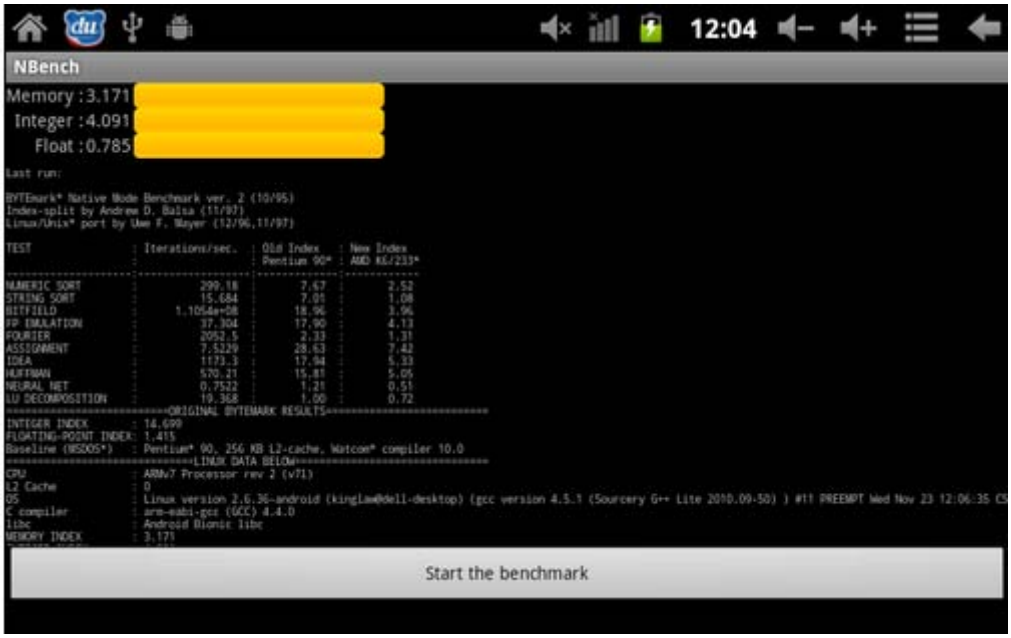
数据来源：http://bbs.imp3.net/thread-10297898-1-1.html



同样的和 Altera SoC 也不是一个量级上的！！

6、全志 A13 处理器测试结果

数据来源：http://detail.1688.com/offer/124 ... gsda_dongtai_liulan



全志 A13 经过测试在内存、整数运算、浮点运算的得分分别为 3.171、4.091 与 0.785，都不及 Altera SoC，浮点性能更差

PK 结论：

1、从测试数据上可以看出，Altera SoC 在内存、整数运算、浮点运算性能指数上比较出众。

2、很遗憾没有 xilinx zynq 的测试数据，如果谁有欢迎补充；

3、同样欢迎补充其他平台的 nbench 测试结果。

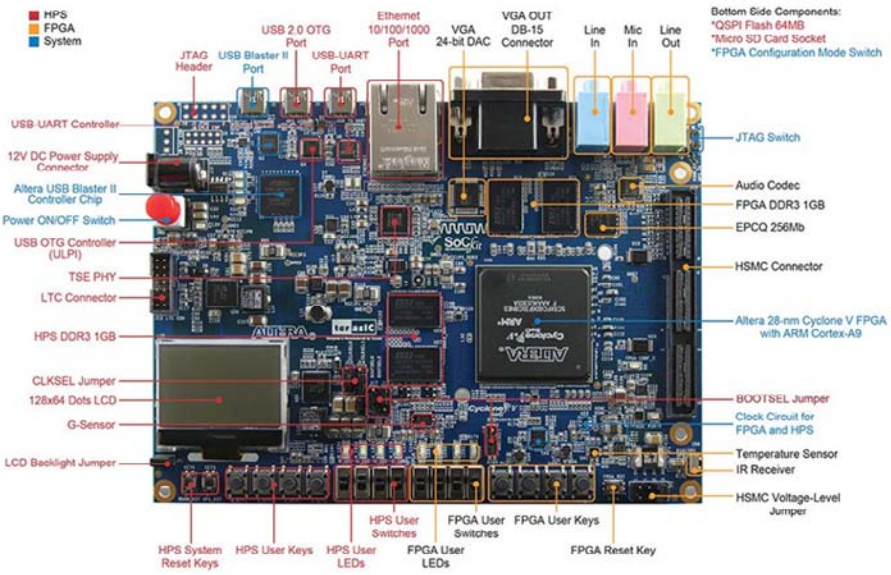
详情请查看：

http://bbs.eeworld.com.cn/thread-458132-1-1.html

三、本章应用平台为 ARROW SoCkit

1. 初见 ARROW SoCkit

1.1 官方资料、图片、参考设计



每套均包含：

Altera cyclone V SoC 包含：内置双核 ARM Cortex-A9;110K LE 可编程的逻辑资源

高速扩展接口（HSMC），并支持 3.125G 高速串行收发器

两排低功率 DDR3 内存

MicroSD 卡和 10/100/1000M 自适应以太接口

采用 Silicon Labs 公司的可编程时钟输出芯片

彩色 128*64LCD 显示屏（SPI 接口）

VGA 和音频接口

USB2.0 OTG（全速）和 USB 转 UART 接口

3 轴数字加速度计和温度传感器

用户手册 USB 接线及电源线

参考资料（视频）：

基于 Cyclone V SoC 可重构的网络控制系统

2014 Altera SoC 合作伙伴研讨会 昇伟电子 IEEE61850 方案

2014 Altera SoC 合作伙伴研讨会北京文通图像处理与识别方案

Arrow SoCkit JESD204B 演示

2014 Altera SoC 合作伙伴研讨会 Arrow Electronics Demo 方案演示

2014 Altera SoC 合作伙伴研讨会 Arrow SoC Kit 方案

2014 Altera SoC 合作伙伴研讨会 昇伟电子 IEEE61850 方案演示

详情请查看：

http://www.eeworld.com.cn/huodong/201411Altera/prize.html



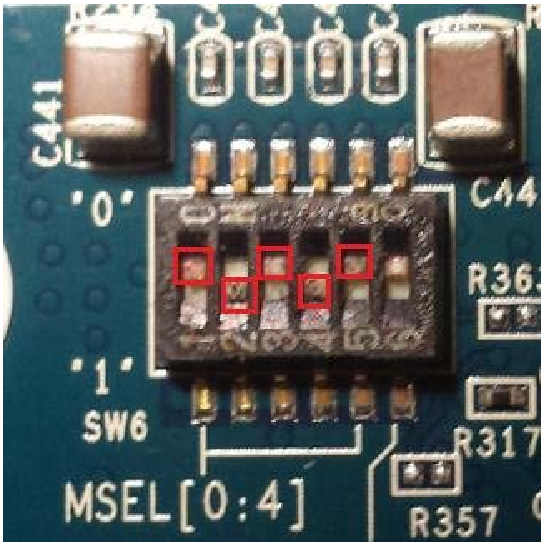
1.2 网友眼中的 SoCkit

1.2.1 烧写和运行 linux 系统

板子默认上电后啥也没有，只能自己烧写个系统看看情况了，造福大伙，做了个百度网盘链接。映像文件链接地址：http://pan.baidu.com/s/1i3GIIWH 映像名称 :sd_image_SoCkit_20140902.img

烧写工具用 win32diskimager-v0.7-binary，百度网盘链接：http://pan.baidu.com/s/1ntJszyT 把 SD 卡插入板子，连接好串口，有几个地方还需要注意一下。

MSEL 的配置如下：MSEL[0:4] 01010



然后上电启动，你可看到以下信息



```
Freeing unused kernel memory: 360K (806b7000 - 80711000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Configuring network interfaces... eth0: device MAC address 02:37:49:3
udhcpc (v1.20.2) started
Sending discover...
libphy: stmmac-0:01 - Link is Up - 100/Full
Sending discover...
Sending discover...
Sending select for 192.168.1.11...
Lease of 192.168.1.11 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
done.
Starting portmap daemon...
Sat Aug 16 15:00:00 UTC 2014
INIT: Entering runlevel: 5
Starting OpenBSD Secure Shell server: sshd
done.
Starting syslogd/klogd: done
Starting Lighttpd Web Server: lighttpd.
Starting blinking LED server
Stopping Bootlog daemon: bootlogd.

arrow_sockit login:
```

uboot 信息：

```
U-Boot 2013.01.01 (Jul 11 2014 - 08:17:20)
CPU   : Altera SoCFPGA Platform
BOARD : Altera SoCFPGA Cyclone V Board
I2C:   ready
DRAM:  1 GiB
MMC:   Altera DWMMC: 0
*** Warning - bad CRC, using default environment
In:    serial
Out:   serial
Err:   serial
Skipped ethaddr assignment due to invalid EMAC address in EEPROM
Net:   mii0
```

内核启动信息：

```
Booting Linux on physical CPU 0x0
Initializing cgroup subsys cpuset
Linux version 3.13.0-00298-g3c7cbb9 (SoCeds@localhost.localdomain) (gcc version 4.7.3
20130226 (prerelease) (crosstool-NG linaro-1.13.1-4.7-2013.03-20130313 - Linaro GCC 2013.03)
) #1 SMP Sun Aug 3 01:29:45 EDT 2014
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=10c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: Altera SoCFPGA Cyclone V
Memory policy: Data cache writealloc
```

3.13 的内核还是挺新的嘛

板子上的有一排小灯，也开始闪起来了。稍候继续

详情请查看：

http://bbs.eeworld.com.cn/thread-454522-1-1.html

1.2.2 Linux 系统初体验

既然系统已经装上了，我们来看看系统中有些什么吧

1、看一下 CPU 信息，是双核的耶，看到没有

```
root@arrow_SoCkit:~# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)
Features        : swp half thumb fastmult vfp edsp thumbee neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0
```

```
processor       : 1
model name      : ARMv7 Processor rev 0 (v7l)
Features        : swp half thumb fastmult vfp edsp thumbee neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

Hardware        : Altera SoCFPGA
Revision        : 0000
Serial          : 0000000000000000
```

2、用 top 命令看一下当前系统运行情况，有意外惊喜，居然还有个 webserver 哦

```
root@arrow_SoCkit:~# top
Mem: 15528K used, 1016500K free, 0K shrd, 904K buff, 5236K cached
CPU:   0% usr   4% sys   0% nic  95% idle   0% io   0% irq   0% sirq
Load average: 0.04 0.03 0.02 1/47 191
  PID  PPID USER     STAT   VSZ %VSZ %CPU COMMAND
  173     1 root     S    10628   1%   0% /www/pages/cgi-bin/scroll_server
  157     1 root     S     3592   0%   0% /usr/sbin/sshd
  188     1 root     S     2496   0%   0% -sh
  168     1 root     S     1960   0%   0% /usr/sbin/lighttpd -f /etc/lighttpd.co
  134     1 root     S     1660   0%   0% udhcpc -R -n -p /var/run/udhcpc.eth0.p
  161     1 root     S     1660   0%   0% /sbin/syslogd -n -O /var/log/messages
  164     1 root     S     1660   0%   0% /sbin/klogd -n
  191   188 root     R     1660   0%   0% top
  189     1 root     S     1564   0%   0% /sbin/getty 38400 tty1
  138     1 daemon   S     1460   0%   0% /sbin/portmap
    1     0 root     S     1316   0%   0% init [5]
    9     2 root     SW       0   0%   0% [migration/0]
   22     2 root     SW       0   0%   0% [kworker/0:1]
   46     2 root     SW       0   0%   0% [mmcqd/0]
    3     2 root     SW       0   0%   0% [ksoftirqd/0]
    6     2 root     SW       0   0%   0% [kworker/u4:0]
    7     2 root     SW       0   0%   0% [rcu_sched]
    4     2 root     SW       0   0%   0% [kworker/0:0]
   27     2 root     SW       0   0%   0% [kworker/u4:1]
```

3、 那么查看一下 IP 吧

```
root@arrow_SoCkit:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:37:49:34:7f:fe
          inet addr:192.168.1.11  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1236 (1.2 KiB)  TX bytes:1648 (1.6 KiB)
```

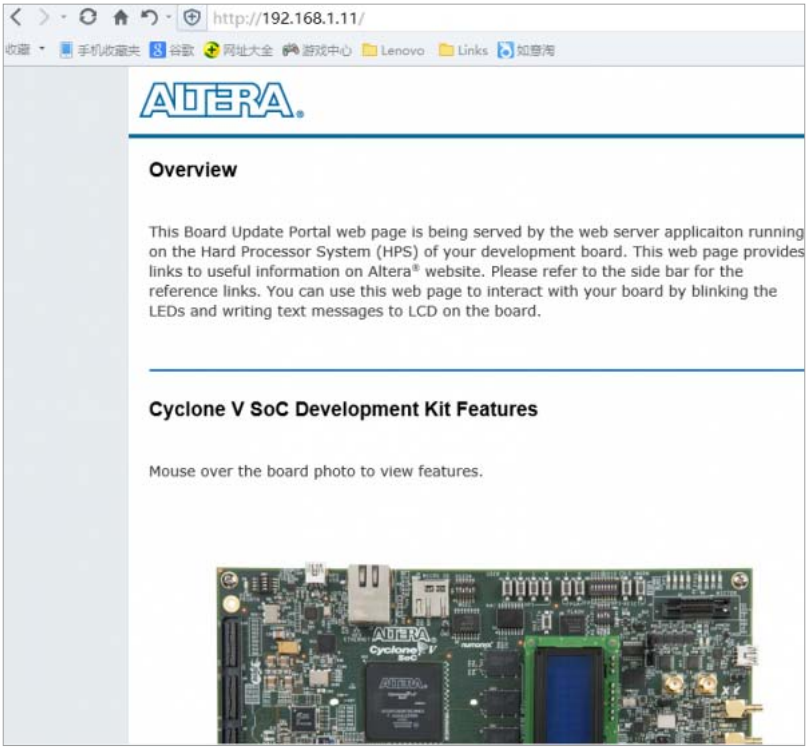
```
          Interrupt:152
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

ping 一下看看网络可通

```
root@arrow_SoCkit:~# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=2.10 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=0.958 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=1.2 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=0.948 ms
64 bytes from 192.168.1.1: icmp_req=5 ttl=64 time=0.941 ms
64 bytes from 192.168.1.1: icmp_req=6 ttl=64 time=0.874 ms
64 bytes from 192.168.1.1: icmp_req=7 ttl=64 time=0.945 ms
--- 192.168.1.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6003ms
rtt min/avg/max/mdev = 0.874/1.11/2.109/0.409 ms
```

一切正常

4、登陆一下 web 看看是啥东西，哇，居然是个控制界面，这个地方可以控制 LED 的



5、看看 home 目录有啥

```
root@arrow_SoCkit:~# ls
README                 clear_leds.sh~        led_blink_devices.sh
Altera                 hello
clear_leds.sh          led_blink
```

哇，又有新发现，还有两个可执行程序，看是啥情况呗

```
root@arrow_SoCkit:~# ./hello
Hello SoC FPGA!
```

```
root@arrow_SoCkit:~# ./led_blink
```

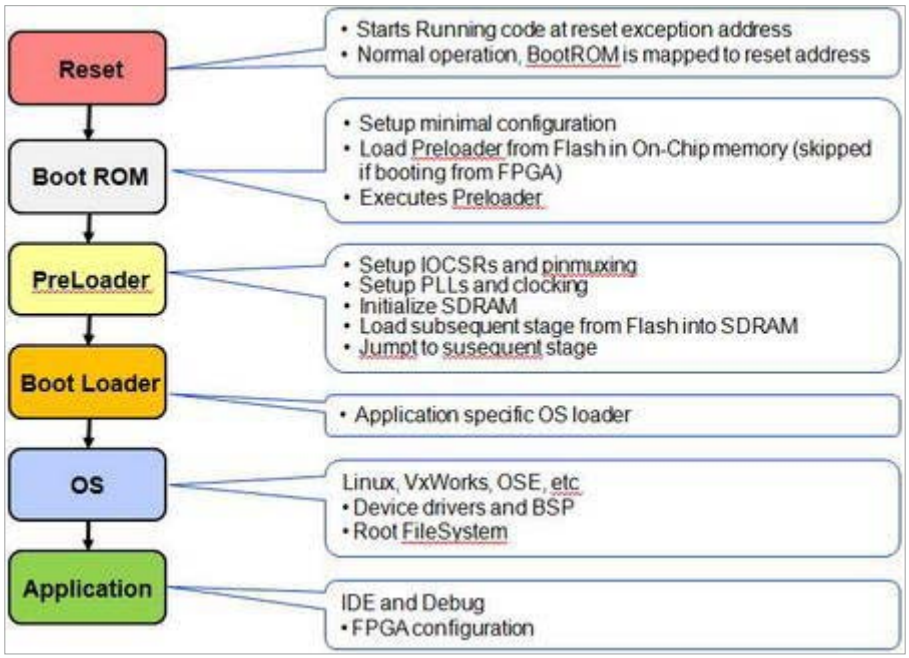
流水灯欢快的跑起来啦，你也可以试试哦，稍后继续。

详情请见：

http://bbs.eeworld.com.cn/thread-454528-1-1.html

1.2.3 So FPGA 配置与启动

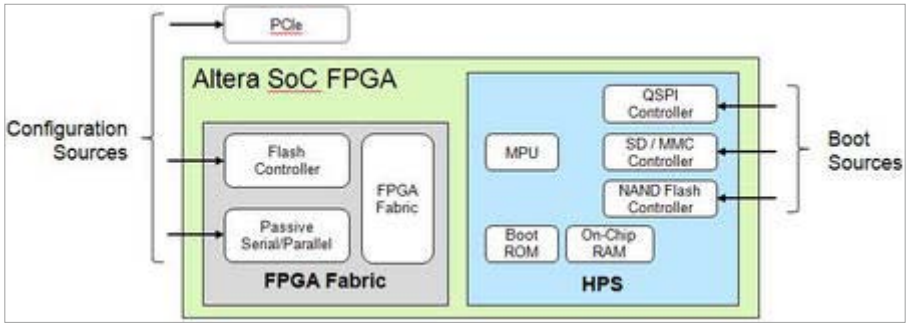
下图是 HPS 启动的详细步骤：



Altera 的 SoC FPGA 启动过程总体可以使用上图来概括，FPGA 和 arm 的配置和启动又可以分为 3 种不同情况（Xilinx 似乎只有一种情况）。分别是 FPGA 和 arm 彼此独立配置和启动、FPG 先配置然后通过 FPGA 启动 arm，第三种情况是 HPS 先启动然后通过 arm 来配置 FPGA 逻辑。下面分别对此进行简单介绍。

1、FPGA 和 hps 单独配置和启动

这种情况 FPGA 从非 HPS 源配置，属于标准的 FPGA 配置模式。同样，HPS 获取 Preloade 也与 FPGA 无关。如下图所示：



2、FPGA 首先配置

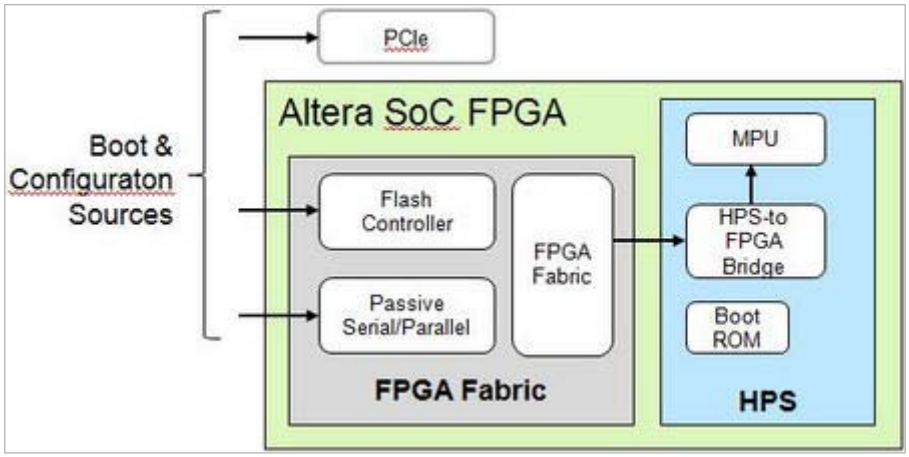
FPGA 通过传统的方式进行配置，然后 HPS 通过 FPGA 逻辑进行启动。

1）、首先需要等待 init_done 信号

2）、HPS 的 BootROM 代码透过 HPStoFPGA 桥执行 Preloader

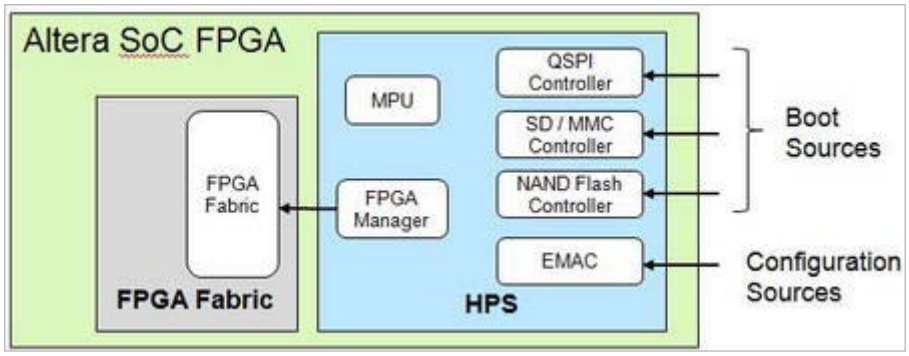
3）、Preloader 可以存放在 FPGA 内部 RAM 或者存放在外部存储器

下图展示了此过程：



3、HPS 先启动，然后通过 HPS 配置 FPGA 逻辑

HPS 从与 FPGA 无关的启动源启动，然后 HPS 的软件通过 FPGA 管理器来配置 FPGA 逻辑。注意，FPGA 的配置信息存储在 Flash 存储器或者其他可访问的通信接口处。下图展示了这种配置启动模式：

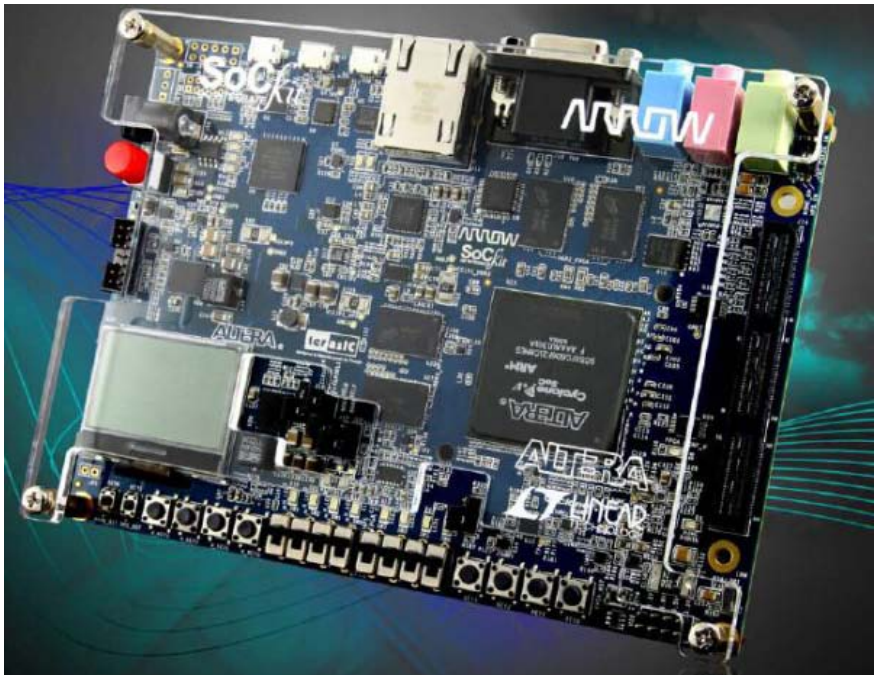


详情请见：

http://bbs.eeworld.com.cn/thread-459641-1-1.html

1.2.4 Altera SoC 特点总结

出差三天，昨天刚回来，从物业那拿到板子，板子非常漂亮，上面很多模块非常实用，而且有很好的例程。



SoC FPGA 系列的亮点

一是，集成 双核 ARM Cortex-A9 MPCore 处理器，Altera 的 Cyclone V 和 Arria V SoC FPGA 的处理器系统采用了双核 800 MHz ARM Cortex-A9 MPCore 处理器，同时具有 NEON 媒体处理引擎、单精度 / 双精度浮点单元、L1 和 L2 高速缓存、ECC 保护存储器控制器、ECC 保护高速暂存存储器，以及多种常用外设。处理器系统的峰值性能达到 4,000 DMIPS，而功耗不到 1.8W。处理器系统和 FPGA 架构独立供电 ,能够以任意顺序配置和启动。工作起来后 ,可以根据需要关断 FPGA 部分 ,以降低系统功耗。

通过大吞吐量数据通路实现 ARM Cortex-A9 MPCore 处理器系统和 FPGA 的互联，峰值带宽超过 125-Gbps，数据的连续性也很好。这种性能水平是两芯片解决方案无法实现的。集成单芯片 SoC FPGA 支持电路板设计人员在处理器和 FPGA 之间不采用外部 IO 通路，大幅度降低了系统功耗。

二，采用先进的 28 FPGA 技术，Altera 的 SoC FPGA 系列利用了其 28nm 系列产品，在多个方面进行创新，通过定制满足了用户的功耗、性能和成本要求，这些创新包括工艺技术、收发器技术、IO 资源和硬核 IP。Cyclone V 和 Arria V SoC FPGA 的推出将这一系列产品进一步拓展至嵌入式处理市场。

Cyclone V 和 Arria V SoC FPGA 基于低功耗 28nm 工艺 (28LP)。这些系列具有分别工作在 5Gbps 和 10Gbps 的嵌入式收发器。FPGA 架构包括精度可调 DSP 模块，以及三个 ECC 保护存储器控制器。Altera 的 Cyclone V SoC FPGA 具有 110K 逻辑单元 (LE)，系统功耗和成本是业界最低的器件性能水平非常适合大批量应用 ,包括下一代芯片工业驱动器、高级辅助驾驶以及视频监控等。对于中端应用，Arria V SoC FPGA 在成本和性能上达到均衡，总功耗也是最低的。器件具有 460K LE，适合满足对性能要求较高的应用，包括，远程射频前端、LTE 基站和多功能打印机等。



SoC FPGA 的系统级优点

提高了系统性能，4,000 DMIPS 时，功耗不到 1.8W，高达 1,600 GMACS、300 GFLOPS 的 DSP。>125 Gbps 处理器至 FPGA 之间的宽带互联，具有高速缓存连续硬件加速器。

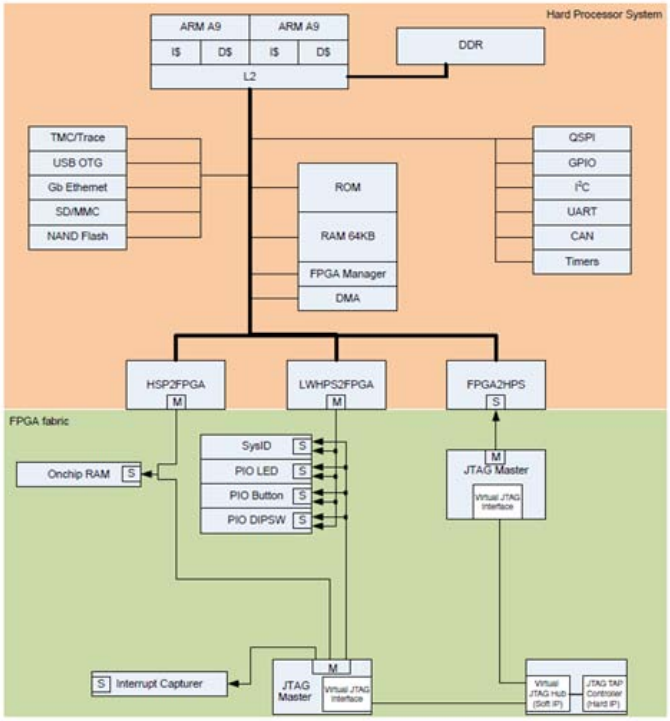
降低了功耗，相对于 2 芯片解决方案，功耗降低了 30%。

减小了电路板面积，外形封装减小了 55%，只有两种电源。

降低了系统成本，降低了元件成本、降低了 PCB 复杂度和成本和更少的布线，更少的板层。

Altera Cyclone SoC FPGA 的开发工具主要是 Quartus13. 或以上版本支持，以及 ARM 开发工具 ARM Development Studio 5 (DS-5)

芯片内部功能结构图如下：



详情请见：

http:/bbs.eeworld.com.cn/thread-454715-1-1.html

2. 让 SoC 飞一会儿吧

2.1 Arrow SoC Kit 学习之拨码点灯

拿到 Arrow SoC Kit 开发板第一眼感觉就是高大上，看着这上面的外设再看看自己的开发板简直是仙女与村姑的感觉，仙女自有仙女的绝美但村姑也有村姑的朴实，自己买的开发板用了有近一年时间说不上精通但这上面的外设基本上都能单独及联合起来实现一定的功能，换句话说就是能玩转这块板子。Arrow 这块板子主要是结合 FPGA 与 ARM 来实现相关的功能，由于缺乏 Linux 的基础所以感觉要想玩转这块板子并非易事，仙女毕竟是仙女嘛。

为了能和仙女套上近乎花了点时间阅读了 Arrow 的使用手册及部分器件的 datasheet, 在 FPGA 部分稍微有点基础就按自己以前学习方学习，但在下载时就遇到了图 1 所示的问题：



图 1 下载时遇到的问题

经过一番捣腾后解决了该问题，板子上能通过拨码开关控制相应 LED 的熄灭与点亮如图 2、图 3 所示：



图 2 点亮中间两灯



图 3 点左右侧灯

编写 Verilog 程序及 bdf 文件均实现该功能，如图 4、图 5 所示：

```
1  module switch(
2                 clk,
3                 rst_n,
4                 sw,
5                 led
6                 );
7    input clk;
8    input rst_n;
9    input [3:0] sw;
10   output [3:0] led;
11   reg [3:0] led;
12
13   always @(posedge clk,negedge rst_n)
14 begin
15   led<=~sw;
16 end
17   endmodule
18
```

图 4Veriog 程序

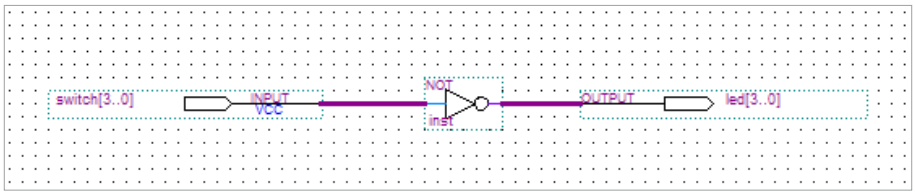


图 5bdf 文件

用 FPGA 实现了拨点灯后尝试着在 Linux 下点灯。Linux 内核的移植本身是一项复杂而繁重的工作但开发板的开发人员已经为我们移植好操作系统并已经制作成 zImage 文件放置于 SD 卡中，通过设置 BOOTSEL 拨码开关就可以从 SD 卡中启动 Linux 内核。

在进行试验之前首先阅读了开发板相关的资料。然后按照《SoC Kit_User_Manual》中第六章介绍的步骤进行了前期简单的实验。为了尽快看到实验结果按照要求迅速设置好各个拨码开关并连接 PC 与开发板。打开 putty 并设置好，按下开发板电源按钮，运气比较好没有出现任何异常，运行 hello world ！程序，结果如图 6 所示：

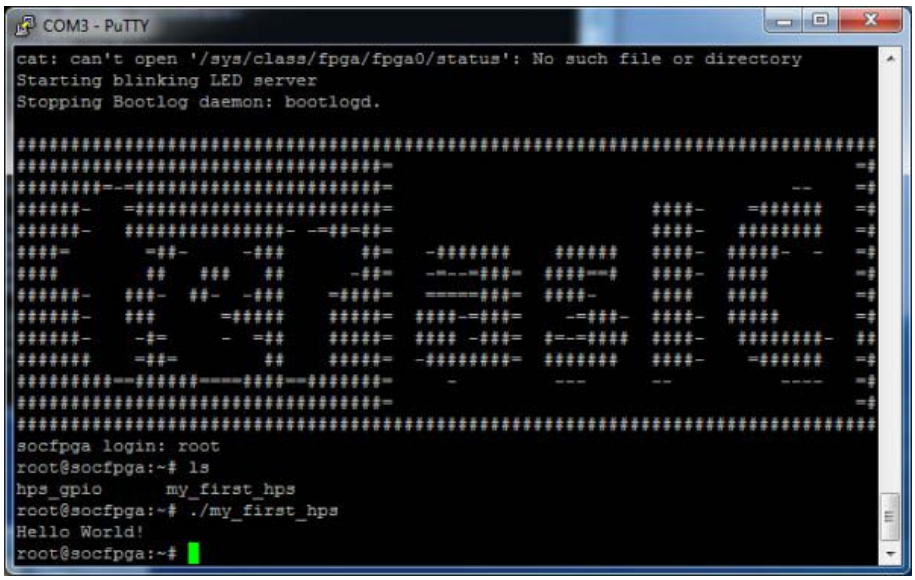


图 6 hello world 程序结果

迫不及待地运行下一个程序即手册中的按键程序，结果如下：

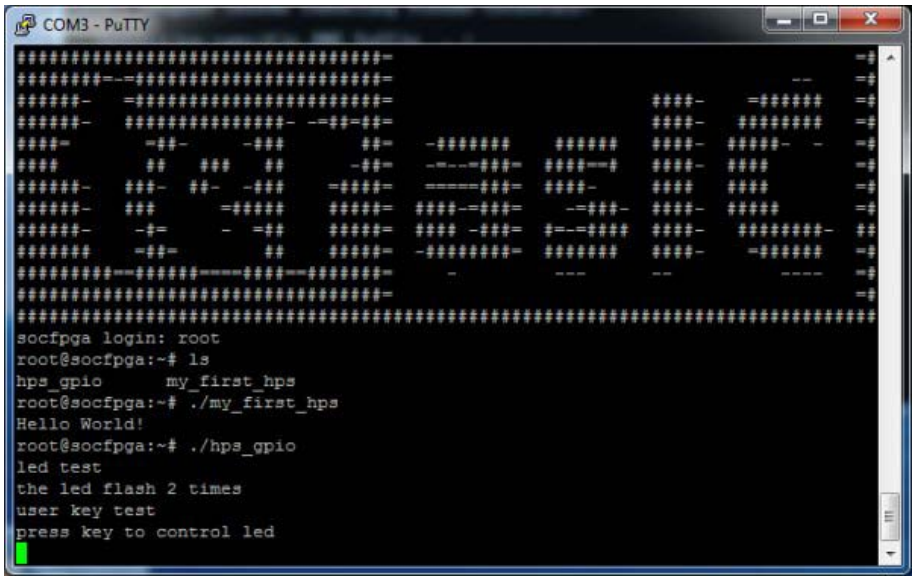


图 7 运行按键程序后 putty 上显示的结果



图 8 运行按键程序后开发板上的 hps_led 都处于关闭状态



图 9 运行按键程序后开发板上按下第一个按键

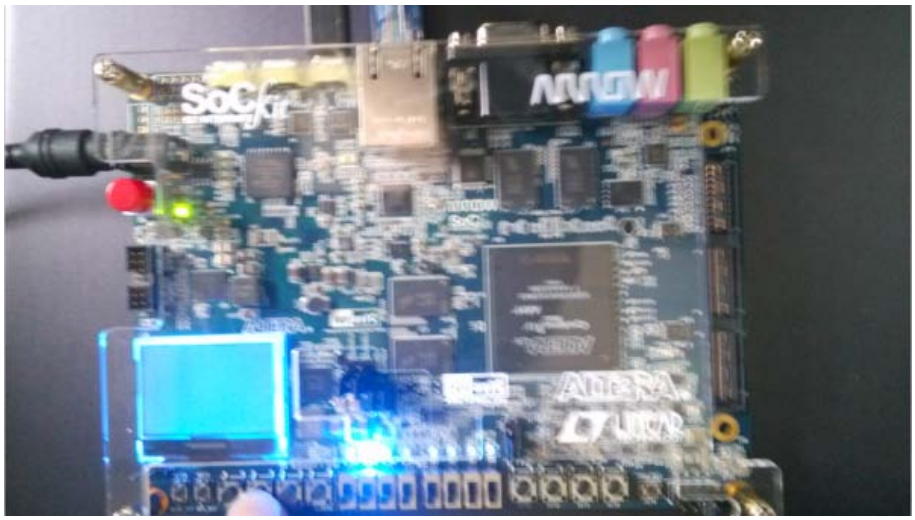


图 10 运行按键程序后开发板上按下第二个按键



图 11 运行按键程序后开发板上按下第三个按键



图 12 运行按键程序后开发板上按下第四个按键

至此，开发板上关于 hps 的前两个实验就成功完成了。由于这两个实验相对比较简单因此实验过程中没有出现什么大麻烦，倒是在安装 64 位操作系统时耗费了一些时间。

与仙女的首次接触后对她有了一定的认识：仙女除了冷艳也还有平易近人的一面，主要得看我们是否有诚心有毅力去接近她了解她，期待与仙女的下一次邂逅。

详情请见：

http://bbs.eeworld.com.cn/thread-456995-1-1.html

2.2 Arrow SoC Kit 学习之 LCD 显示

继上一次按键开关学习之后，中途断断续续看了看用户手册，阅读了下程序代码，但程序都没有跑过。今天终于下定决心把这个 LCD 程序跑一下。好久没使用 Embedded_Command_Shell 了……

下面正式开始学习。

首先，阅读 LCD 代码。由于之前断断续续看过 LCD 的代码，所以这次看起来不是很费劲。代码阅读完成后，为了满足自己迫不及待的心情，打算将例程直接 copy 进 SD 卡。这时候问题出现了，不知道怎么把程序拷进去……（之前没去仔细看《My_First_HPS》，后悔啊）。想了一下，打算先去网上看看有没有解决方法，百度了一下，还真找到了。先使用 udhcpc 获取 IP 地址，再使用 ifconfig 查看 IP 地址，最后使用 scp 命令将文件 copy 到 SD 卡中。看了别人的做法之后，立马按部就班，前两步还真的成功了，不过第三部问题出来了，scp 命令需要用户密码！这下我就郁闷，我哪知道密码……简单试了几个密码失败之后，我就算放弃这条路了。既然此路不通，那我只想其他的办法了。突灵光一闪。既然板子上的 Linux 系统可以识别 SD 卡文件系统，那我的 Ubuntu 肯定可以。于是我找了个读卡器，使用 Ubuntu 系统去读取，果然可以识别！看来办法总比困难多！在 copy 过程中，发现还需要修改 SD 卡中 /home/root 文件夹的权限。不过好歹学过一点 Linux，这都不是事儿！

怀着忐忑与激动的心情，将 SD 卡插回去，连上串口，打开电源，等待系统启动完成。系统启动后，直接进入输入命令 ls，果然看到了新 copy 进去的文件，于是立即运行程序，成功了！长舒一口气，觉得生活真美好！图一就是原程序的运行效果。

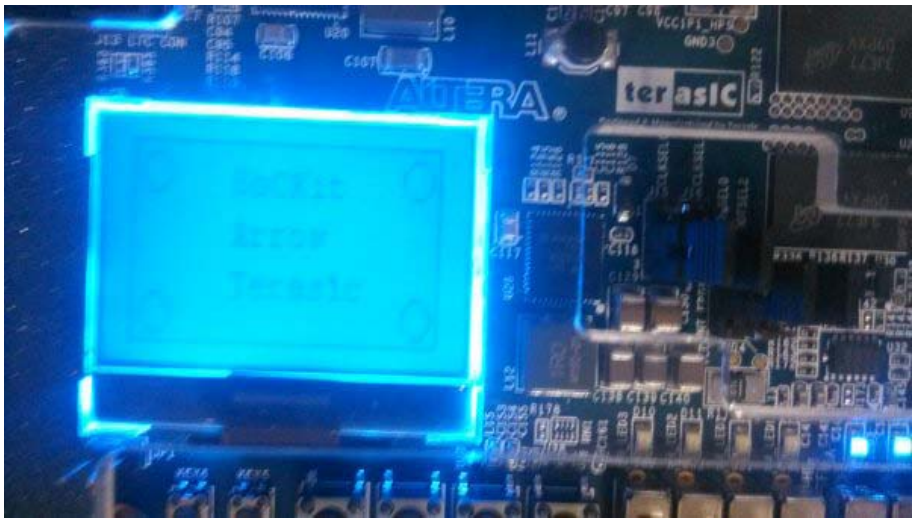


图 1

完成之后觉得这样将程序 copy 进去实在麻烦，于是我用将资料拿出来找了下，看看有没有告诉我系统的密码是什么。终于在一个文件中被我找到了密码。这下终于可以轻松的 copy 文件了。

接下来就简单了，我将源文件修改了下，重新编译，重新 copy。下面就看看我的成果吧。虽然很简单，但也是一种学习。

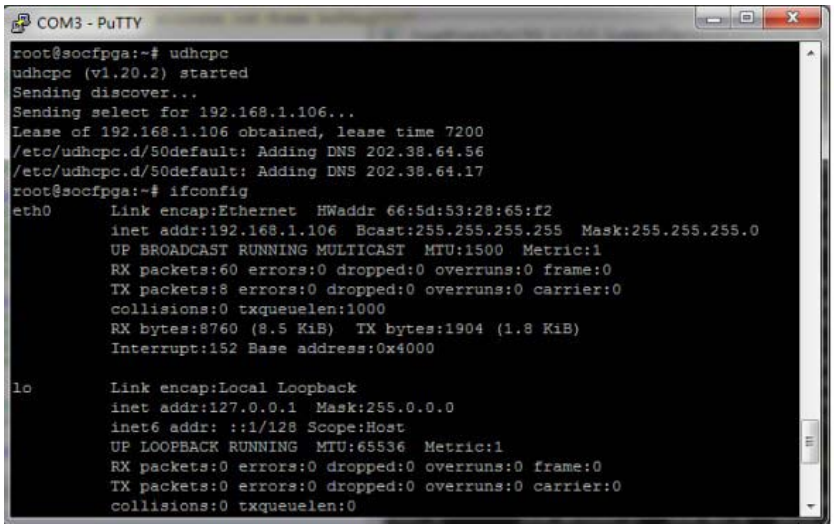


图 2 获取 IP 地址

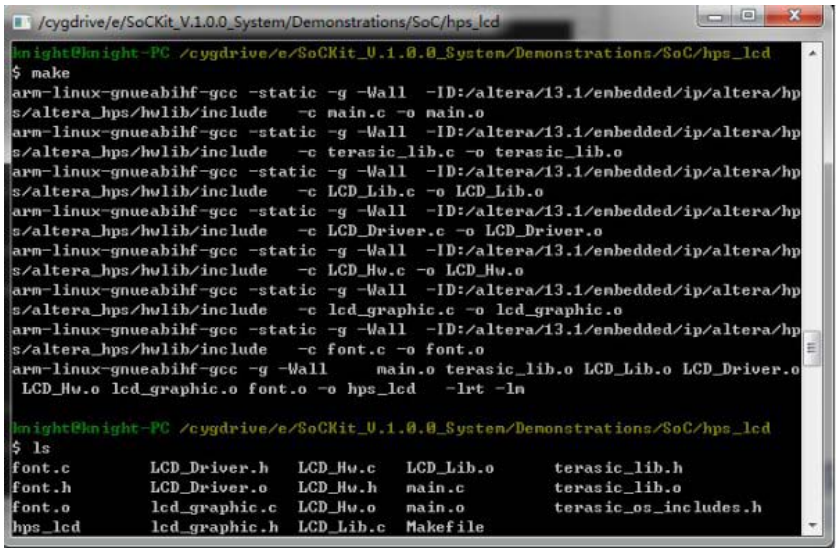


图 3 编译修改后的文件

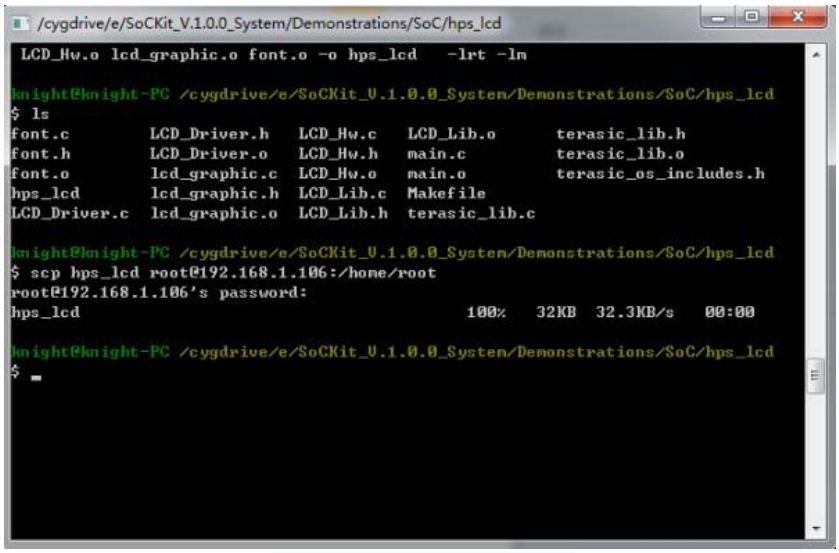


图 4 将程序 copy 到 SD 卡

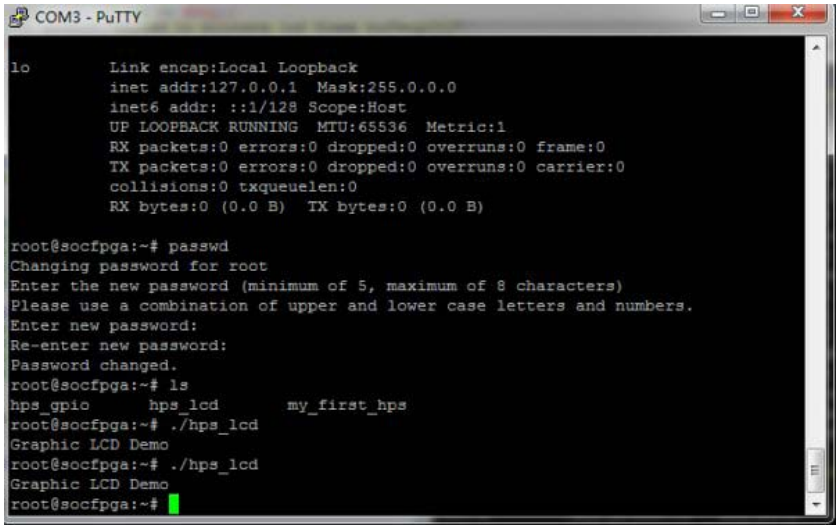


图 5 运行程序



图 6 运行结果

这是关于 SoC 学习之旅的第二次发帖，计划始终赶不上变化啊，相对于计划来说有点 out 了，得好好反省反省。

详情请见：

http://bbs.eeworld.com.cn/thread-462247-1-1.html

2.3 使用 System Console 验证硬件系统

如果使用的是 Arrow SoC 的板卡，并且烧写了 linux 内核，那么 SD 卡不要插哦，要不然测试 FPGA 侧灯一直在被 ARM 控制，会导致你误认为硬件系统无法控制。

1、编译 SoCkit_ghrd_14_0.tar.gz，并下载；

2、打开 qsys，并选择 Tools -> System Console

3、运行脚本文件， 选择 File -> Execute Script，若有弹出窗口 ， 选择 Don't Create 选项

4、选择 ghrd_sc_script.tcl，执行效果如下

```
scripts by issuing Tcl source commands.
------------------------------------------------------------------------------
llength of all_sp: 3
non-secure jtag master: /devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit/(link)/JTAG/
FPGA_only_master.jtag/phy_1/FPGA_only_master.master
secure jtag master    : /devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit/(link)/JTAG/
hps_only_master.jtag/phy_0/hps_only_master.master
```

5、开始测试

1）控制 FPGA 侧 LED，在 Tcl console 窗口输入

```
<blockquote>% led_on 5
```

有两个灯亮啦，后面跟的是 4 个 led 的编码

如果要关闭，输入

```
% led_off
written to address: 0x10040
```

2）控制 ARM 侧的 LED, 使用命令 hps_led_on 打开，使用命令 hps_led_off 关闭

详情请点击：

http://bbs.eeworld.com.cn/thread-455149-1-1.html

2.4 利用 System Console 工具对 SoC 设计进行调试

概述

传统 FPGA 逻辑工程师要开始进行基于 SoC 的 FPGA 设计，确实有一些难题。这一系列的文档大致记录下本人学习 SoC 开发的一些历程，根据设计文档跑了一些例程，记录这些参考设计学习的过程，以便随时“温故知新”。本文简单讲述如何利用 System Console 工具来对 SoC 设计硬件进行调试，这里假设 FPGA 硬件已经搭建完成（Altera 合作伙伴的各家开发板，也包括 Altera 自己的开发板都有典型的 Golden 参考设计，即所谓的 GHRD 工程）。

学习简述

位于 FPGA 内部的 ARM 处理器的行为，本质上来说和位于 FPGA 外部分立的 ARM 芯片并无差别。工程师拿到一片 SoC FPGA 可以将其看做一片普通的 FPGA，也可以将其看做一片普通的 ARM 芯片，当然系统工程师是将二者集成一起看待。所以对于传统的 FPGA 工程师来说，可以认为 arm 不存在一样去进行 FPGA 设计。直到软件工程师完成软件设计并将软件下载到处理器，处理器复位之后跳转到位于 Flash 之中的复位入口。

FPGA 工程师利用 Qsys 工具完成逻辑设计，在 HPS 周边添加了不同的 IP 单元，比如 LED，拨码开关等等。在处理器运行程序之前，硬件工程师可以利用 System Console 工具来对这些元件进行验证。如图 1 所示，就是一般的 SoC 系统硬件架构，我们可以利用 JTAG 到 Avalon 桥（master，图中黄色设备）来访问 Avalon 总线上的所有元件，这时候我们可以作为一个主设备来访问总线上的所有从设备。

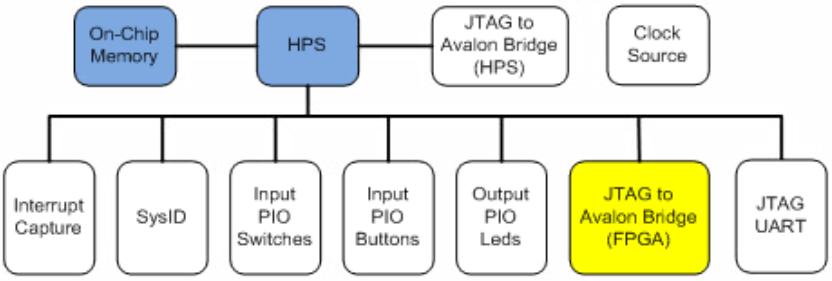


图 1：SoC GHRD 工程结构框图

如图 1 所示，系统中此时已经例化了 HPS 处理器核，只是此时我们还无法对其进行访问，因为这是还没有任何代码在处理器中运行。笔者在介绍基于 Qsys 的 PCIe 设计中就提到过，在利用 Qsys 进行 FPGA 设计的时候不再需要处理器的介入，这里其实我可以将 HPS 看做不存在一样，只是因为我们后面还要进行 HPS 软件设计的学习，所以 FPGA 硬件工程中事先例化后了 HPS 核。

启动 System Console 并对 FPGA 进行编程

我们打开 GHRD 工程，并打开 Qsys 软件，在 Qsys 软件中打开之前已经例化好的系统，即 SoC_system.qsys。这里不再详述。

我们可以从 Qsys 软件的 Tool 菜单里启动 System Console 工具，如果开发板（笔者使用的是 Arrow 的 SoCkit 开发板）连接状态正常，且已上电，那么我们可以在 System Console 工具中看到 JTAG 链上的器件连接情况，如图 2 所示，如果看不到，可以从 System Console 的 Tool 菜单中执行“Refresh Connection”命令刷新（该命令也可以直接鼠标右击图 2 中的 connection 文件夹找到）。

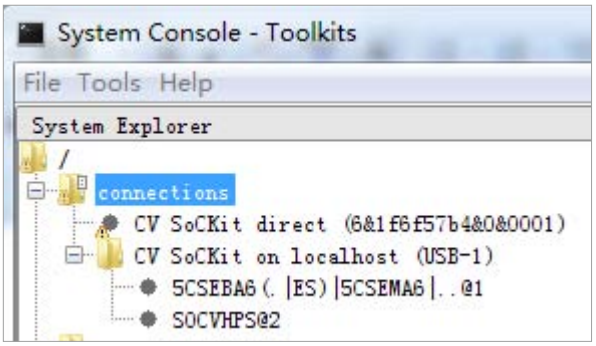


图 2：在 System Console 中查看板子连接状态

System Console 是基于 TCL 脚本设计及操作，所以后面的很多操作、命令都将使用类似虚拟 JTAG 中我们使用到的命令。首先我们来看器件服务，在脚本命令输入行敲入“get_service_paths device”命令，那么随后命令行窗口会列出链路上所有的器件，这里我们只有一块板子连接在 JTAG 链，所以如图 3 所示会给出一个器件。

```
% get_service_paths device
{/devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit}
```

图 3：列出 JTAG 链路上所有器件

下面对上述链路上的器件进行编程，在此之前我们需要定义一个变量来存放访问目标。使用的命令为：

set d_path [lindex [get_service_paths device] 0]

注意 lindex 会索引路径上所有的器件，这里因为只有一个器件，所以我们将索引号为“0”（即第一个器件）的器件作为目标赋值为一个变量。那么接下来我们可以直接使用程序下载命令对该变量指向的目标器件进行编程了，具体如图 4 所示。

```
% set d_path [lindex [get_service_paths device] 0]
/devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit
% device_download_sof $d_path output_files/soc_system.sof
```

图 4：定义目标变量并对目标进行编程

当然，我们也可以直接使用鼠标操作来完成器件编程，如图 5 所示，鼠标右击 device 文件夹，在弹出的菜单中选择编程器件即可。

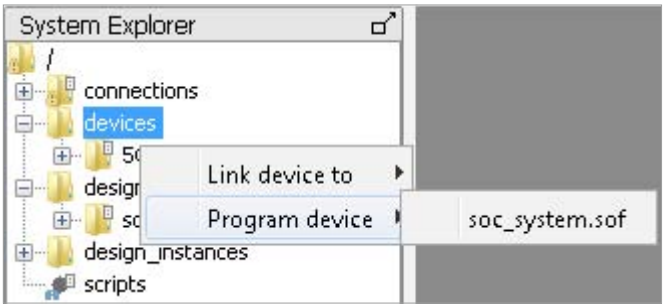


图 5：直接通过鼠标操作来完成器件编程

器件编程完成后，除了我们在设计中加入的测试程序来对编程操作进行验证外，在 System Console 的消息窗口中会有消息提示，编程完成后会在消息窗口提示：

Auto linking 5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit to SoC_system.sof

器件下载程序之后，此时我们如果展开 device 文件夹，那么就会看到我在 Qsys 中添加的各个元件，如图 6 所示。

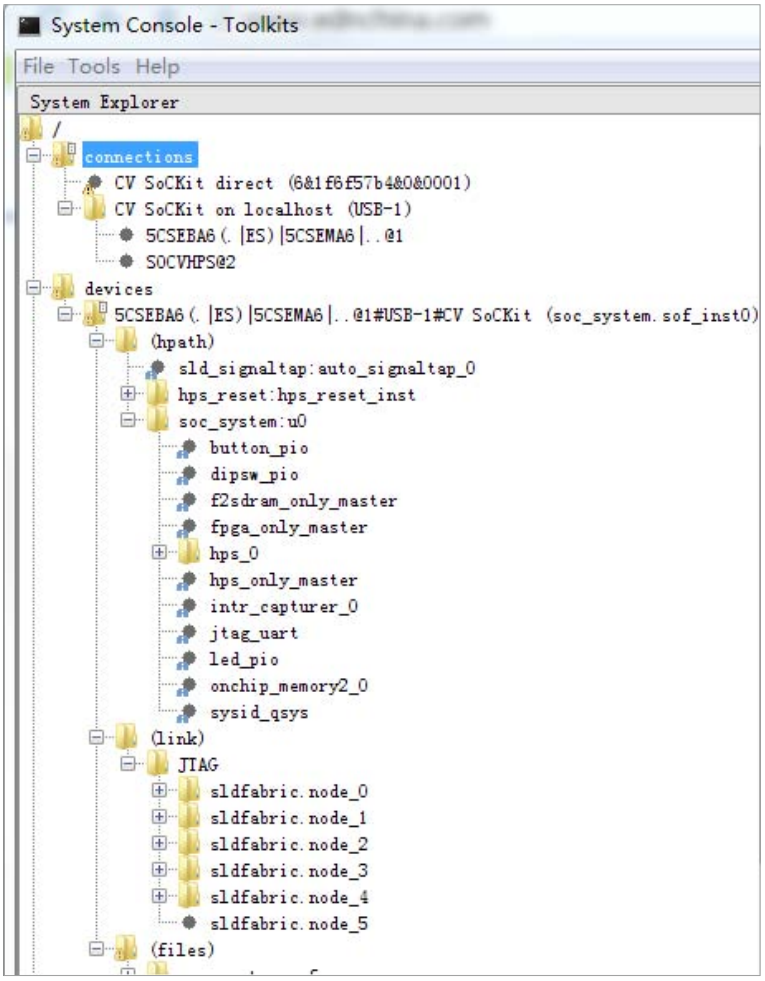


图 6：板子下载程序后

利用 System　Console 工具检验时钟和复位

我们再次使用“get_service_paths”命令，不过这次是获取 JTAG 调试服务。同样该命令会反馈并列出链路上所有的 JTAG 调试任务，其实图 6 中已经在“link”文件夹中为我们显示了 JTAG 调试任务了。如图 7 所示，执行该命令后结果。

```
% get_service_paths jtag_debug
{/devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit/(link)/JTAG/sldfabric.node_1/phy_0}
{/devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit/(link)/JTAG/sldfabric.node_2/phy_1}
{/devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit/(link)/JTAG/sldfabric.node_3/phy_2}
```

图 7：获取 JTAG 调试任务

图 7 以及图 6 中并未完全为我们展示这些任务，其实我们可以展开发现这三个任务分别是 FPGA

only master、hps only mster 以及 f2sdram only master。为了检验系统的时钟和复位，这里我们需要通过 FPGA only master 来对时钟和复位元件进行访问，这里定义一个变量来指向 FPGA only master。和之前的命令一样，这里需要用到 lindex 索引命令，索引号为 0，这里我们指向的并不是 FPGA only master（不过没有关系，只要打开 JTAG 调试即可，这里为了测试，打开了另外一个调试任务）。具体命令及其执行情况如图 8 所示。

```
% set jd_path [lindex [get_service_paths jtag_debug] 0]
/devices/5CSEBA6(.|ES)|5CSEMA6|..@1#USB-1#CV SoCKit/(link)/JTAG/sldfabric.node_1/phy_0
```

图 8：定义变量并指向 FPGA only master

为了验证 JTAG 链路的信号完整性，可以使用 jtag_debug_loop 命令向链路串入一些特定码来进行验证。如图 9 所示：

```
% jtag_debug_loop $jd_path [list 1 2 3 4 5 6 7 8 9 10]
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a
```

图 9：串入特定码进入 jtag 链来验证其信号完整性

下面来验证系统的时钟是否正常工作，我们可以使用的命令有两个，第一个命令是 jtag_debug_sense_clock $jd_path。该命令就是简单检查时钟是否还在正常的翻转，如果是，那么返回值为“1”。

第二个命令是 jtag_debug_sample_clock $jd_path。这个命令会采集当前时钟信号的值，由于时钟翻转速度很快，所以我们需要执行很多次该命令才能采集到不同的值（即 0 和 1），只要时钟还在正常翻转，那么使用这个命令我们就应该能分别采集到 0 和 1，如图 10 所示。

```
% jtag_debug_sense_clock $jd_path
1
% jtag_debug_sample_clock $jd_path
1
% jtag_debug_sample_clock $jd_path
0
```

图 10：验证系统时钟

本节最后来看看如何在 System Console 中验证系统的复位信号。使用的命令是 jtag_debug_smaple_reset $jd_path。和时钟验证的第二个命令一样，该命令采集的是复位信号的当前值，开发板的复位信号是低有效，所以在未按下复位按钮的时候，采集回来的值应该是 1，如图 11 所示。同时，通过命令行设计者还可以给整个系统发送一个系统复位，具体格式如图 11 所示。

```
% jtag_debug_sample_reset $jd_path
1
% jtag_debug_reset_system $jd_path
```

图 11：验证系统的复位以及给系统发送一个复位

利用 System Console 工具访问 FPGA 的外设

访问 FPGA 的外设是通过执行 master read 和 master write 来完成的。首先，我们需要获取可以利用的 master 服务，如图 12 所示执行 get_service_paths master 来获取。



图 12：获取 FPGA 已实现的 master 服务

大家注意图 12 与图 7 的差别，这里还是先定义一个变量来指向 FPGA only master 服务路径，注意这里的 lindex 索引号应该为 1，因为我们要指向 FPGA only master，如图 13 所示。



图 13：定义变量指向主设备服务

如果使用索引号 0 打开的就是如图 14 所示为 f2sdram only master。



图 14：索引 0 指向的主设备服务

和虚拟 jtag 一样，访问设备（或服务）之前都要先打开设备，这里打开主设备服务，如图 15 所示。

```
% open_service master $m_path
```

图 15：打开主设备服务

设备（服务）打开后，设计者就可以对其进行操作了。这里我们只对板子上的 LED 以及拨码开关进行读写操作。首先操作 LED，打开 Qsys 系统，找到 LED 元件的基地址，然后执行如下命令：

master_write_memory $m_path 0x10040 0xf，该命令熄灭所有 LED；

master_write_memory $m_path 0x10040 0x0，该命令点亮所有 LED；

master_write_memory $m_path 0x10040 0xc，该命令点亮 LED 0 和 1，熄灭 LED 2 和 3；

master_write_memory $m_path 0x10040 0xa，该命令点亮 LED 0 和 2，熄灭 LED 1 和 3。

如果不执行图 15 所示打开命令，那么在执行上述命令的时候会报错，如图 16 所示：

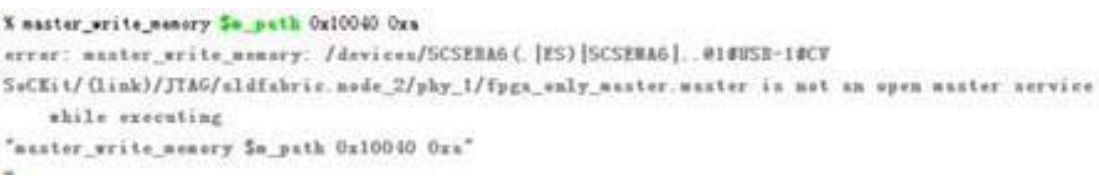


图 16：未打开设备服务的时候不能执行读写操作

由于 led_pio 元件宽度是 4 个比特，所以我们还可以使用 master_write_8、master_wirte_16 或 master_write_32 命令来获得相同结果。

以上介绍了 master write 命令，我们还可以使用 master read 命令来读取拨码开关上的当前值。从 Qsys 中获取 dip_sw_pio 的基地址后就可以执行读取操作了，如图 17 所示。

```
% master_read_memory $m_path  0x10080 1
0x0f
% master_read_32 $m_path  0x10080 1
0x0000000e
% master_read_16 $m_path  0x10080 1
0x000c
% master_read_8 $m_path  0x10080 1
0x08
```

图 17：读取操作

注意，返回数据格式与读取命令位宽一致。

小结

最后小结一下，以上只是在使用 Qsys 完成逻辑设计的基础上，使用 System Console 工具对于 ARM 核外设进行初步验证。比如使用 JTAG 链执行了底层的验证及对 FPGA 进行编程（下载 sof 文件）；对于系统的时钟和复位信号进行底层的验证；使用 System Console 工具执行了 master 读写命令。这些验证或操作都“旁路”了 ARM 核，但是被操作的外设又都挂在了 ARM 上，也即我们可以在 ARM 核工作起来之前利用模拟的方式对其部分外设进行验证；也就是说利用这个工具，逻辑工程师可以在软件工程师编写完软件之前就可以对处理器的外设进行验证。

详情请点击：

http://bbs.eeworld.com.cn/thread-461546-1-1.html

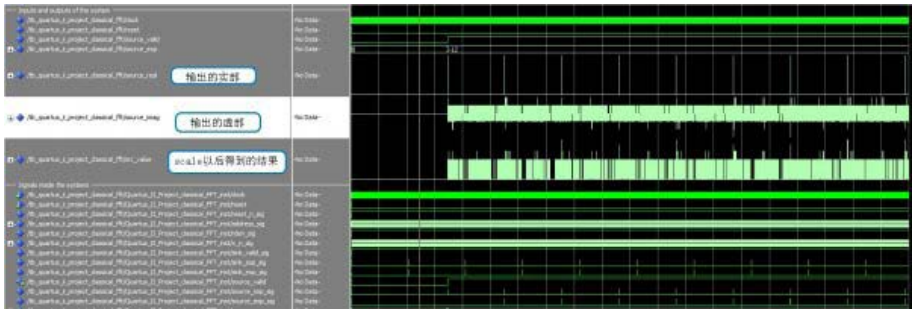


2.5 对 ADC 采样数据进行 FFT 处理

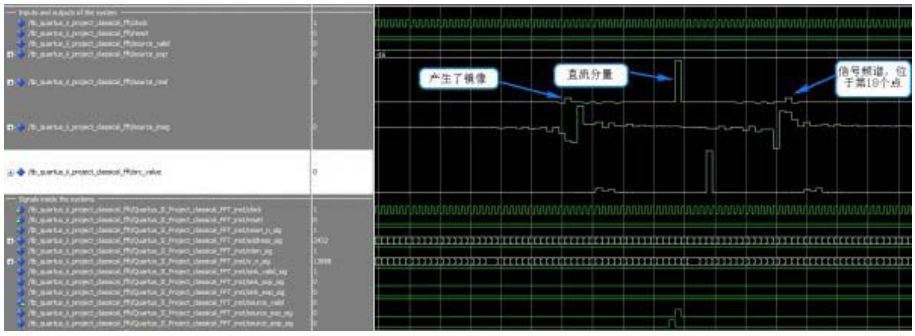
前面看了柱斑的 3 篇有关 ADC 采样的帖子，我的板子没有 ADC，所以我也只能从同事的 DAQ 采集得到的数据进行 FFT 处理，当然只是进行仿真。学校学习的 FFT 的原理老早就丢给了老师，最近一段时间花了点时间复习了 FFT 的物理意义，这些信息大家都可以通过度娘得到 FFT 的作用有很多种，那么我们做 FFT 的意义非常简单，就是分析信号源中的频率分量，并精确得到每个频率分量的幅度，相位信息我们不关注。对了，fft 我们没有能力自行设计代码，所以啦，只能使用 Altera 的 IP 核。

在例化 FFT 使用之前，先简单描述下我们的系统。信号源来自光电探测器，为了提取出微弱的信号，信号在光域进行了声光调制，所以最终电信号在频率分析可以获得最大的动态范围。信号调制的频率应该 KHz 级别，所以我们的 ADC 采样速率不是很高，20MHz 左右即可。只是信号调制时间长度一般是 1ms，要求采样时间长度达到秒级，所以虽然采样率不高，但是每次处理的数据量还是比较大，比如一帧（1ms）数据采样后得到 20000 个样本。

根据上述分析，那么在例化 FFT 的时候初步确定选用 Streaming 模式（个人感觉应该可以使用缓冲突发模式），fft 转换长度选为 2048. 这样最小分辨出的频率是 20000KHz/2048 大致等于 9KHz。由于事先不知道信号调制的频率，所以选择了上述 FFT 的参数，结果悲剧发生了，FFT 转换的结果始终是只有 DC 分量，询问同事得知信号调制在 10KHz 左右



上图没有放大，我仔细放大查看过，确实没有其他频率分量（我查的是输出实部，虚部是底噪的响应）；后来发现问题所在之后，重新例化了 FFT，使用了 32768 点转换，这样最小可以分辨到 610Hz。仿真结果如下图所示：



从上图我们可以看到信号频谱出现在了第 18 个点上，该点大致频率等于 610*18=10.98KHz。

大致仿真过程就是这样了，最后也学柱斑来几个总结：

1、是 FFT 给 ADC 采样数据进行转换的时候，ADC 的数据从 FFT 的实部输入，虚部输入端口接 0。

这个为何这么做我也是不太清楚，看到有人这么做就这么做，给 Altera 提了 SR，人家回复说可以这么做，至于具体虚部要不要接数据，人家说要根据自己的算法来决定。

2、FFT 的结果从上面仿真的结果来看，是不是我采用实部输出就可以呢？这个问题我也在 SR 里咨询了，答复也是可以的。但是 FFT 的用户手册说了，由于 FFT 核采用但是块浮点实现的，所以为了不丢失精度，最终结果是需要与 EXP（指数）输出进行缩放处理，上图的 src_value 就是我缩放处理后实部的平方加上虚部的平方再开发得到的，这样得到的结果应该是 FFT 转换后的幅度吧。由于项目需要获取信号的精确幅度，所以我的疑问是直接从实部得到幅度值还是需要进行上述转换得到？

3、最后一个问题是图中明显看到出现了频率分量的镜像，那么 32768 点 fft 是不是只能分辨出一半的频率分量呢？比如 20MHz 的话只能分辨出 610Hz 到 10MHz 而不是 610Hz 到 20MHz 呢？因为后半部分都是前半部分的镜像啊。

详情请见：

http://bbs.eeworld.com.cn/thread-459630-1-1.html



3. 应用与设计

3.1 电源设计

1. 硬件环境

Arrow SoC Kit　Quartus14.0

2. 设计概述

拿到了开发板之后，软件的安装和学习确实折腾了一番，能够换电脑，笔记本不行，买了个新的台式机再来，接着又更换操作系统。电脑准备就绪之后，我们三人兴趣小组中，刚开始我负责 FPGA 中 Verilog HDL 逻辑的设计与调试。后由于进度的安排，需要重新设计自己的一块 SoC 板子。我开始重操旧业，进行电路原理图的设计和 PCB 的绘制。

电路板的设计，电源部分的设计是首先要考虑的问题。需要考虑使用场合中输入电压，输出所需电流的问题。电源芯片的选取和电源电路的设计，将会对板子的可靠性和稳定性产生重要的影响，特别是在后期的调试过程中，一旦发现电源设计出现了问题，如输出电流不够等问题，将会对项目的进度产生很大的影响，很可能电路板需要重新制板设计。

当然，电源芯片自身消耗的功耗，即电源转换效率也是需要考虑的问题。在节能环保的今天，我们需要的是低功耗的产品。

TI 公司的网上关于 Cyclone V 的推荐的电源芯片很多，如 LMZ31506RUQ 等芯片，可支持 12V 电源电压直接输入，是一款不错的芯片。其它公司提供的电源芯片也比较多，可以供我们选择，包含 Altera 公司提供的 Enpirion 系列电源芯片。经过我的查询和早起部分电源芯片的使用，发现给 FPGA 供电的低电压输出的模块主要是以 LDO 为主，这些 LDO 芯片具有输出噪声低，纹波小的特点。但是对于≥ 5V 以上的外界电源输入场合，它们的自身消耗的功耗可就不太尽为人意，特别是当板子上所需电流增加的情况下。一般的 LDO 自身消耗的功耗随着输出电流的增加，自身损耗掉的功耗也是在迅猛增加的。

我后来试用过电源芯片 TPS75333 ,TPS75325 等小型封装的电源芯片 ,都会发现同样困扰我的问题。电源自身的损耗是个比较大的问题。

有些电源芯片在实际使用起来，转换效率觉得会比手册上面的低很多。

在某些场合下，每个芯片输出一路电源，采用多个电源芯片给 FPGA 供电，无疑也会增加电路板的面积，这不符合电路板设计的小型化的发展趋势。

某些情况下，板子上多个电源的上电顺序也是设计电路板时需要考虑的问题。

3. 设计结果

经过不断地选型，电源芯片系统的搭建和仿真，面包板上小型电源电路的验证。从宽输入电压范围、体积小、外围电路设计简单，输出电源电流大小，电源转换效率等方面进行考虑，最后选用了 LINEAR 公司的 LTM4644 芯片。

它一个芯片，就可以可配置 4A 输出阵列的四通道输出，便于减小电路板的面积；

宽输入电压范围：4V 至 14V；

每路输出电压范围可调，范围：0.6V 至 5.5V 输出电压

每通道可提供 4A DC、5A 峰值输出电流

可通过并联提供更高的输出电流

提供过压、过流和过热保护

每路电源输出时间，也就是上电顺序，可以通过调节外接的电容大小来简单方便的实现

关键一点：该电源模块的转换效率高

4. 电源部分电路已经设计了好几版电路，LTM4644 是感觉运行最为良好的一块。

现在我设计的板子已经焊接好了，并且初步的 FPGA 逻辑电路已经配置运行起来。

给我最大的一个感觉就是，这个芯片实际中使用起来，转换效率高。

5. 如果大家有什么更好的电源部分电路的设计，可以一起讨论学习；

当然，有什么更好的建议或者不同的见解也欢迎提出，我需要不断改进。

谢谢 EEWORLD 提供这么个机会，让我从硬件开始，一步步来学习 Altera SoC。

详情请见：

http://bbs.eeworld.com.cn/thread-459779-1-1.html

3.2 Altera SoC 之移植 LINUX 系统 + 示波器界面

其实我本来是打算学习板子 HPS 那部分的，然后计划在该硬核上移植个 Linux 操作系统，然后在该系统上用 QT 做数字个示波器显示界面，但是临近毕业了，事情比较多，所以做的很不尽如意，只是实现了显控功能，采集部分还在进行当中，但是活动已经结束了，很遗憾！！！！

接下来我就把我做到部分分享一下吧，部分也都是看各个论坛然后学习使用的，并非本人原创，但是也是学习的过程，希望大家共同学习进步！！！

我把我做的分两部分吧，第一部分是操作系统的移植，第二部分是 QT 示波器的制作。

首先我们来说一下这个 LINUX 操作系统的移植，其实这一块，在我移植的过程中还算比较顺利，但是在偶尔的机会我去了一个国外的论坛，上面有完好的例程，我觉得在大家以后使用的过程中直接可以拿来使用，非常方便，算是封装好的吧。所以，我把这个例程拿来说一下把！！

要求：

1、开发板一块
2、TF 卡卡托
3、16GB 的 TF 卡
4、OTG 转接线
5、USB Hub

具体步骤：

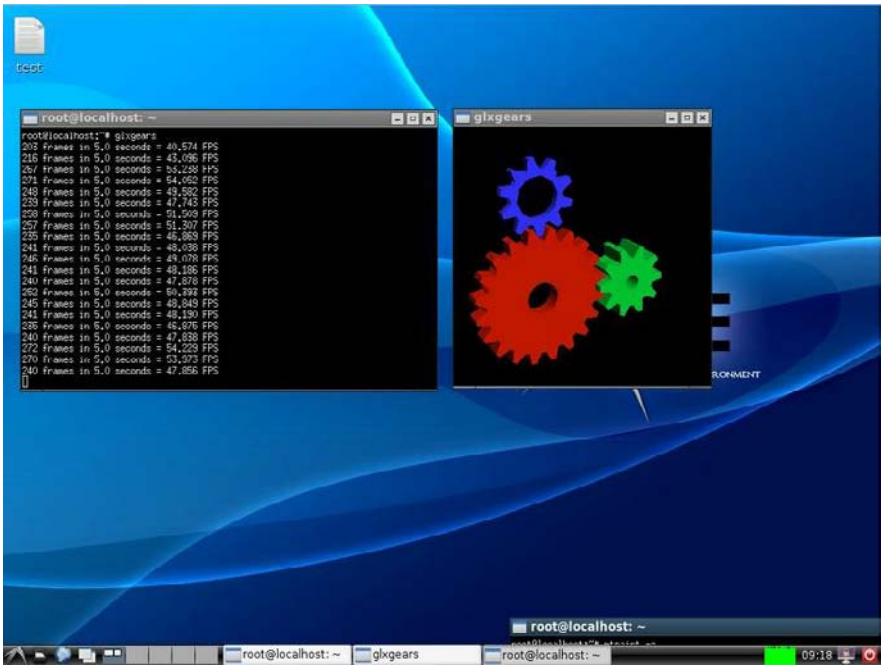
1、去下载系统镜像文件，这个文件我们可以直接用现成的，不用我们去移植了，如果已经会移植的小伙伴就可以直接拿来用了，如果不会的画还是去试着移植下，毕竟我们是来学习的，过程很重要。地址是：https://s3.amazonaws.com/rocketboards/SoCkit_lxde_20130907.tar.gz

2、在 windows 下用 Win32DiskImager 来把镜像文件写入 SD 卡中。

3、然后把 TF 卡插入到开发板中，确保 bootsel jumpers 设置的是 101，MSEL jumpers 设置为 000000，这么设置的原因我们可以参考 SoCKit User Manual 手册。

4、然后我们用 OTG 线插在板子三个 USB 空的中间那个，然后把 USB HUB 连接在 OTG 上，这样一来我们就可以把鼠标和键盘连接上了。然后把板子的 VGA 口连接到电脑显示器，给板子上电 ，板子中的 TF 卡中的 LINUX 系统就启动了。如图：





这样，LINUX 系统就启动成功了，但是有好多包需要安装，这些是在我使用的过程中需要的，所以我都装上了，我相信您用的时候也大概需要。比如：

apt-get install ssh

apt-get install vim

apt-get install less

apt-get installpsmisc

apt-get install lynx

apt-get install gcc g++

这些算基本的把，其他的等我们需要的时候在安装把，还有我个人在使用的时候发现这个系统的源也不太好使，希望你们在使用的时候最好换个源，再此就不用说这个了，因为怎么换源网上有介绍的。

我也把在使用过程中需要的 .sof 文件，.rbf 文件，.dtb 文件，uImage，preloader_with_header.img，u-boot.img 提供给大家，主要是用来大家一起学习使用。

附件中我会上传的。

这样 ，系统部分我们基本完成了。

接下来，我们谈谈我做的示波器这部分，说示波器还有点过，因为还没有全部完成，只是把软件部分完成了 ,也就是显控部分。应用的软件是 QT ,还有就是 QWT 库也要安装啊 ,因为有我们需要的绘图库。因为采集部分硬件还没有做好，所以用软件模拟给的信号，这样就可以给大家看波形的显示了。虽然很简单，但是也算小成就吧。到这个时间点也只能做到这个程度了。最后实现的示波器包括以下几点：

1、波形测量。包括周期、频率等；

2、触发模式。自动触发；

3、波形数学运算。CH1 和 CH2 的加、减、乘运算，以及 FFT 功能；

4、存储功能。可完成波形或设置参数的存储及调出；

5、光标测量功能。能用光标测量时间和幅度参数；

以下是数字示波器软件系统平台照片，如图 5.7 示。

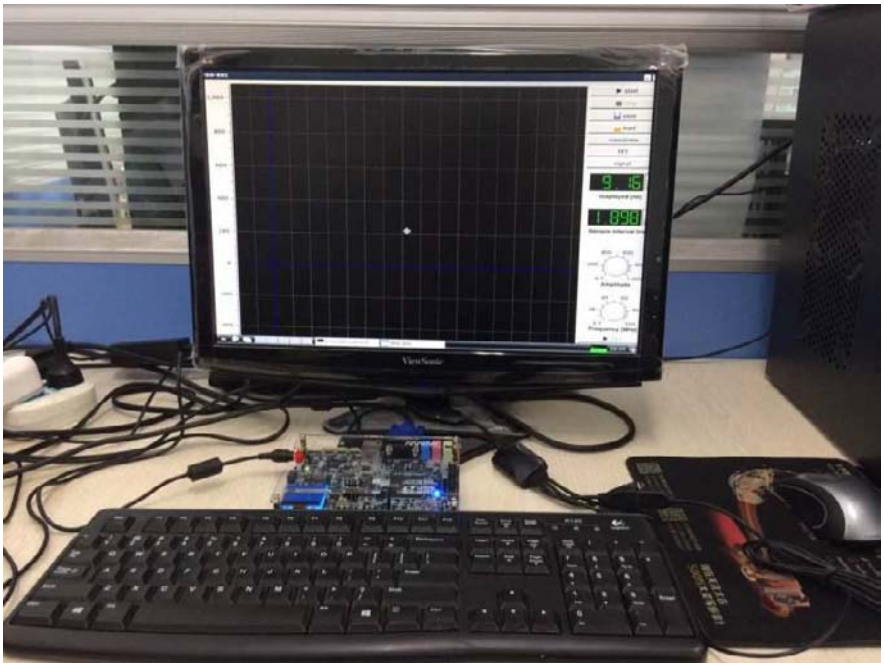


图 5.7 数字示波器软件系统平台

我们给通道 1(CH1) 输入幅度是 150mV，频率为 50MHz 正弦波时，如果采样率是在 67MHz 时，波形在 VGA 显示如图 5.8 所示。当采样率是在 200MHz 时，波形在 VGA 显示如图 5.9 示。

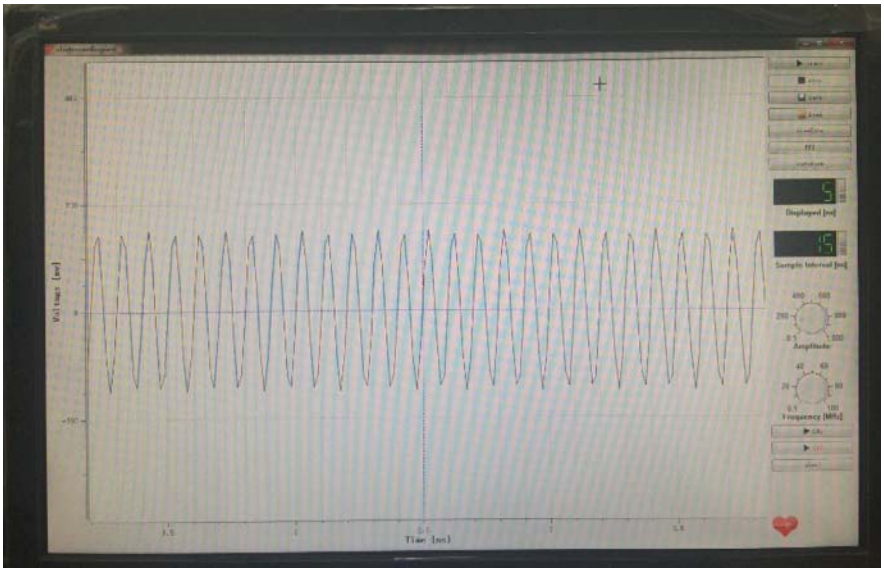


图 5.8 采样率为 67MHz 时的正弦波图



图 5.9 采样率为 200MHz 时的正弦波形图

通过观察波形显示，当采样率为 67MHz 时，显示的正弦波波形不正常，当采样率为 200MHz 时，能够显示正常的正弦波波形。说明 200MHz 采样率是我们波形正常采样的一个频率。

图 5.10 为通道 1(CH2) 给入正弦信号的波形显示图。当我们选择通道 1(CH1) 的时候，点击 CH1 这时候我们会发现按钮上的字体是红色，当我们选择通道 2(CH2) 的时候，同理，CH2 按钮上的文字也是红色，当两个通道都使用的时候就字体就都是红色的。但是，一般在使用的时候系统默认选择通道 1(CH1)。由图 5.10 可知，波形显示与通道选择均常。



图 5.10 通道 2 输入正弦波的波形显示

图 5.11 为通道 1(CH1) 给入方波信号的波形显图。

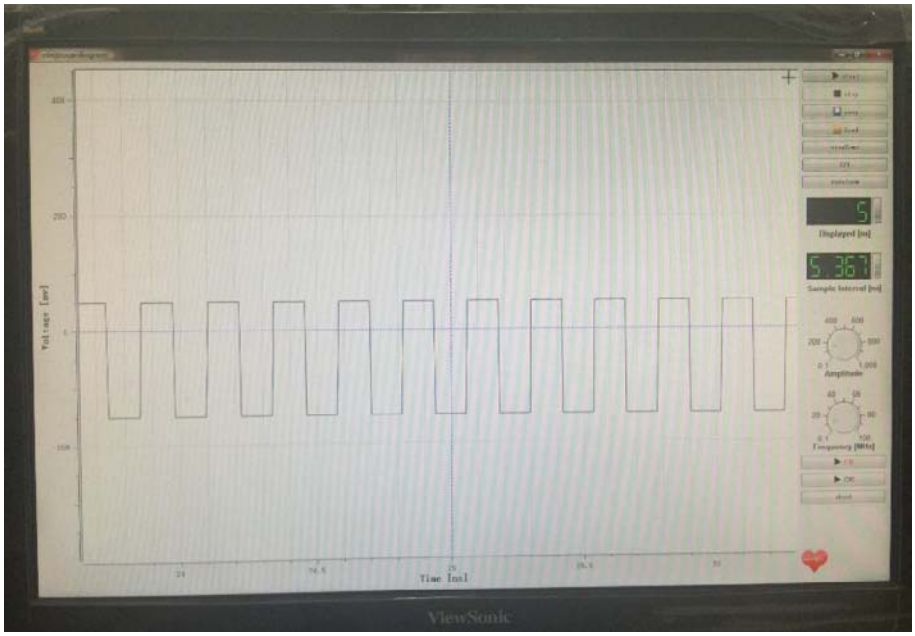


图 5.11 输入方波信号的波形显示

图 5.12 为通道 1(CH1) 给入三角波信号的波形显图。



图 5.12 输入三角波信号的波形显示

图 5.13 是通道 1(CH1) 输入方波信号 FFT 波形显示图。图 5.14 是通道 1(CH1) 输入的正弦信号的 FFT 波形显示图。从图中我们可以知道，FFT 结果都是正的。

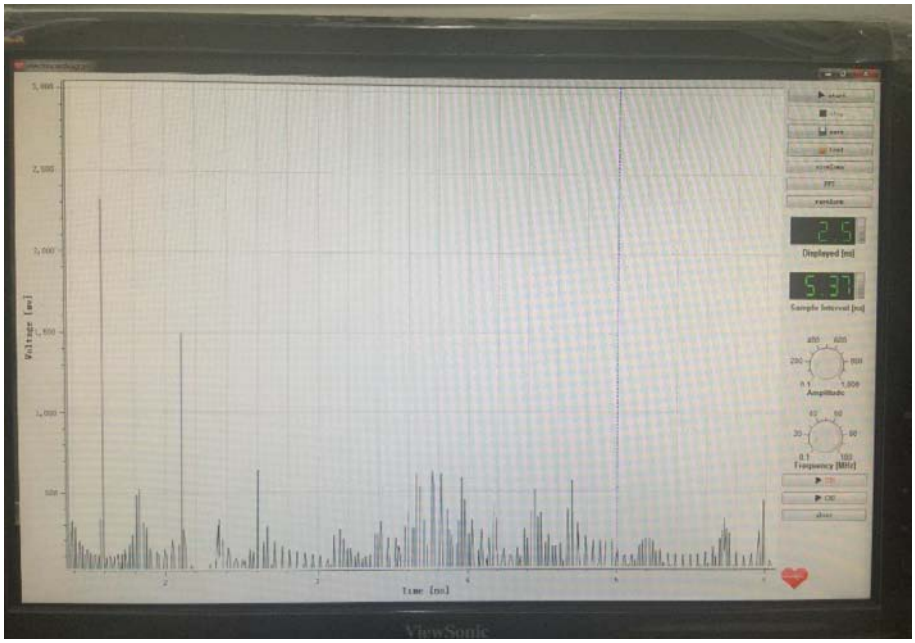


图 5.13 通道 1 输入方波经 FFT 运算后的波图

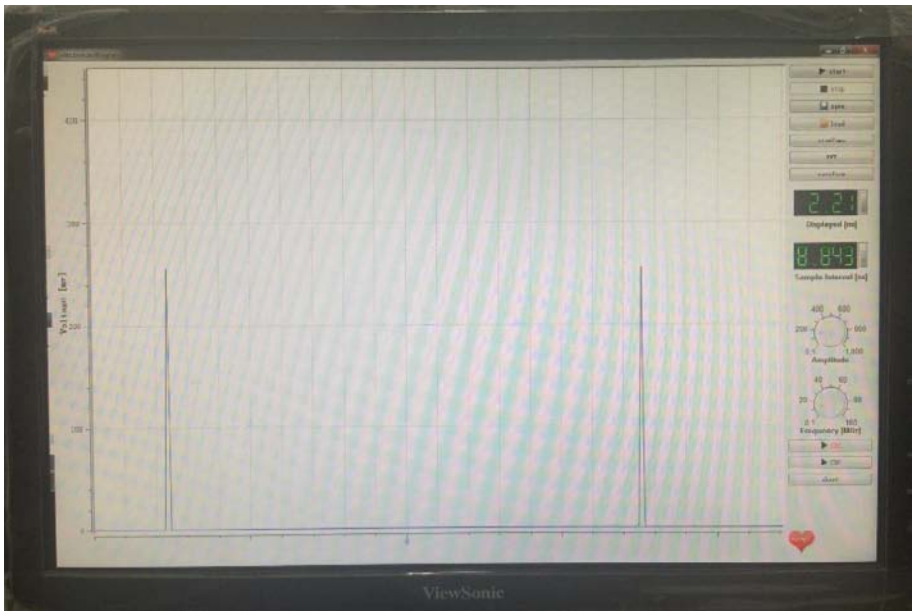


图 5.14 通道 1 输入正弦波经 FFT 运算后的波形

当我们需要把波形数据存储起来的时候我们可以通过界面的右上角的 save 按钮来存储，也可以通过 load 按钮载入我们的波形数据来进行分析处理。综上，以上就是我这段时间的小成就吧，写的不是很详细，但是大体思路都提到了，革命尚未成功，后续还需要努力。同时，希望在此一起学习的小伙伴不要放弃，一起共同进步。如果有什么问题可以联系以下 QQ。加油加油！！！！

详情请点击：

http://bbs.eeworld.com.cn/thread-463357-1-1.html

4. 用户体验与感想

4.1 对于 Altera SoC 用于高清相机控制分系统设计的感触

1、设计平台：

软件平台：Quartus 14.0

硬件平台：Cyclone V SoC 采用自己设计的电路板硬件

2、高清航拍相机系统主要功能设计：

1）通过 FPGA 控制实现高速 ADC 数据采集；

2）通过 FPGA 实现串口指令的接收和响应；

3）通过内部总线实现 FPGA 与其内部 ARM 之间的数据交互；

4）通过 FPGA 实现对相机光阑运动部件的精确位置控制，快门运动部件的精确时间控制，调焦组件的精准位置控制，像移补偿机构的精准速度和位置控制。

5）采用 FPGA 实现外部温度采集；

6）采用 FPGA 实现千兆以太网数据通信，发送相机控制分系统内部工作状态。

系统框图，如下图所示：

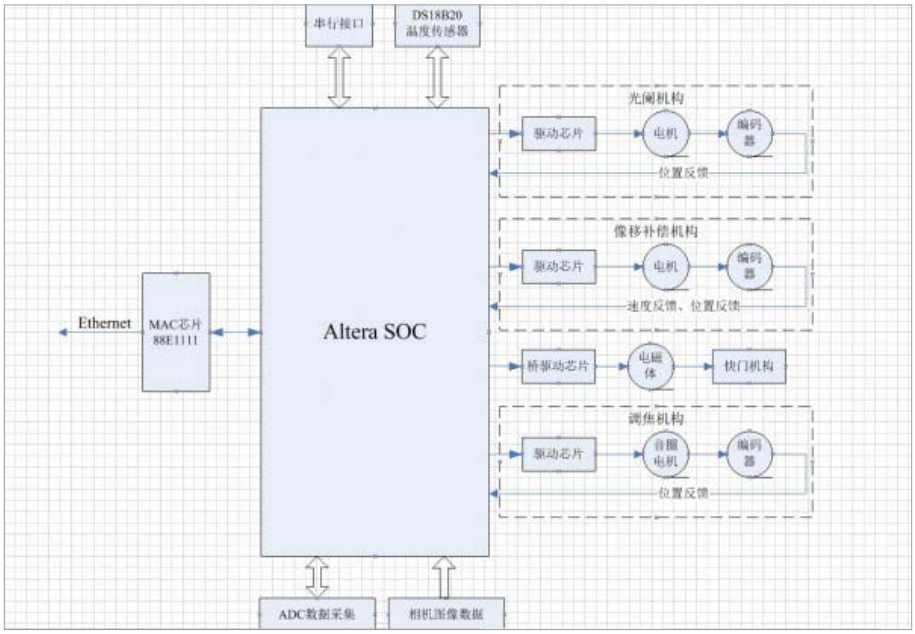


图 1 系统框图

3、设计和调试过程中的感触

1）ADC 电路设计需要注意 FPGA 的管脚电压输入范围，对外围输入电压要进行分压；需要充分考虑到 ADC 管教的 BANK 的 IO 电压输入范围。

2）FPGA 内部通过爬坡算法实现相机的自动调焦，在这部分原理实现部分，充分利用到了 SoC 的内部架构优势，FPGA 实现图数据监测处理，ARM 内部实现算法计算，将计算结果送给 FPGA，简化了硬件设计，这一点对我感触尤为深刻；减小了 PCB 的布线，对系统的整体性能有了很大的提高和改善；但是对软件的要求增加了。

3）在航拍过程中，飞机高速飞行拍摄出来的图像会造成拖影，为了解决这一问题，设计了像移补偿机构，相机在获得飞机的速高比之后，通过 ARM 进行浮点运算，很好的解决了 FPGA 逻辑在除法算法上的缺陷，提高了像移补偿的精度。

4）温度测量，用来获取相机本体的温度，以便在高空中低温情况下，对相机进行温度补偿，解决高空中低温低气压的情况下，对相机焦距造成的影响，保证相机焦距稳定，成像清晰。

总之，Altera SoC 减小了 PCB 的设计，提高了运算速率和计算精度，PCB 硬件设计的可靠性更加有保障了，这同时也对设计人员的软件要求越来越高。

详情请点击：

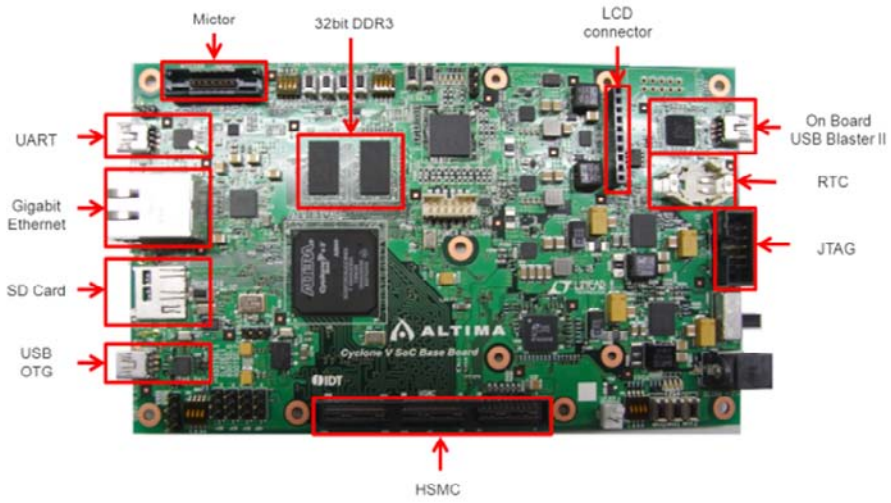
http://bbs.eeworld.com.cn/thread-463135-1-1.html



四 本章硬件平台为 Macnica Helio Kit

1. 初见 Macnica Helio Kit

1.1 官方资料、图片、考设计



Helio 主板功能：

主芯片：Cyclone V 5CSXFC6C6U23ES SoC with Dual-core ARM® Cortex ™ A9 processor　电源器件：LTC2978, LTC3605, LTC3613, LTC3026, LT3509

内存：Micron DDR3 2Gb @ 32bit @ 533MHz ECC

配置方式：On board USB Blaster II, Configuration ROM, JTAG

软件支持：Yocto Linux BSP support

外围接口：HSMC, Gigabit Ethernet, USB 2.0 OTG, UART, SD card, GPIO, LCD

其他功能：Power Supply Monitor by Linear Technology, Real Time clock, Buttons, LEDs, Switches

参考资料（视频）：

2014 Altera SoC 合作伙伴研讨会 骏龙科技 Demo 方案演示

many Cytech training on below website

详情请查看：

http://www.eeworld.com.cn/huodong/201411Altera/prize.html

1.2 网友眼中的 Helio Kit

1.2.1 Helio Cyclone V SoC 开发板应用之 FPGA 入门体验

Helio Cyclone V SoC 开发板，做工精美自不用多说，功能更是强大而复杂。可是到手后，虽然有些用户手册和基本的操作说明，还是感觉入手比较难。怎么办呢？既然是 FPGA+ARM 架构的 SoC, 那么它一定具有 FPGA 的功能，虽然 SoC 咱不熟悉，但是对 FPGA 而言，还是比较亲切熟悉的。那么体验的入门，就先从 FPGA 开始吧。

对于 Cyclone 系列的 FPGA 而言，一般手册上都会说明芯片的工作频率会在 200MHz~520MHz 左右，但是具体到每个可编程的 IO 管脚上的具体时钟，一般很少会注意到，但是这却是最关键的，尤其是在做并口或可编程时钟之类的处理时，误把总线频率当做 IO 也具有的频率。Altera 建议，3.3V CMOS 电平配置管脚的工作频率不要超过 100MHz。所以到手的芯片或板子可以在所有工作开展之间，做这样一个小小的验证工作，有图有真相，更会加深理解和实际应用。

验证原理如图 1 所示。启动 Quartus II 13.1 版本软件，新建工程文件，新建图形框图表文件，双击窗口空白区，弹出“Symbol”对话框，点击“MegaWizard Plug-In Manager”按钮，新建插件中选择 PLL，修改对应的参数，如图 2 所示。例如 outclk0 输出设置为 200MHz，outclk1 输出设置为 100MHz，指定 50MHz 时钟晶振管脚到 FPGA 的时钟输入管脚，指定管脚之前，先进行系统编译，然后再指定管脚，次编译。

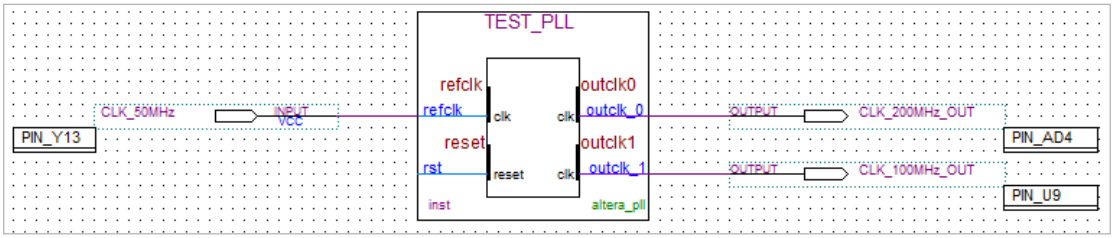


图 1 测试 IO 性原理框图

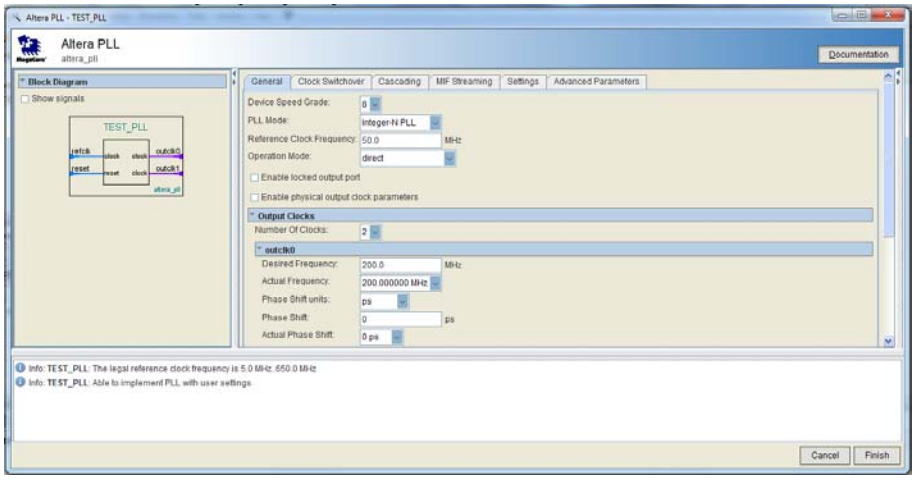


图 2 Altera PLL 模块参数配置界面

在进行图 3 所示的下载程序之前，需要将 Demo 板中的相关接口配置做个调整，具体调整内容参见图 4 示说明。

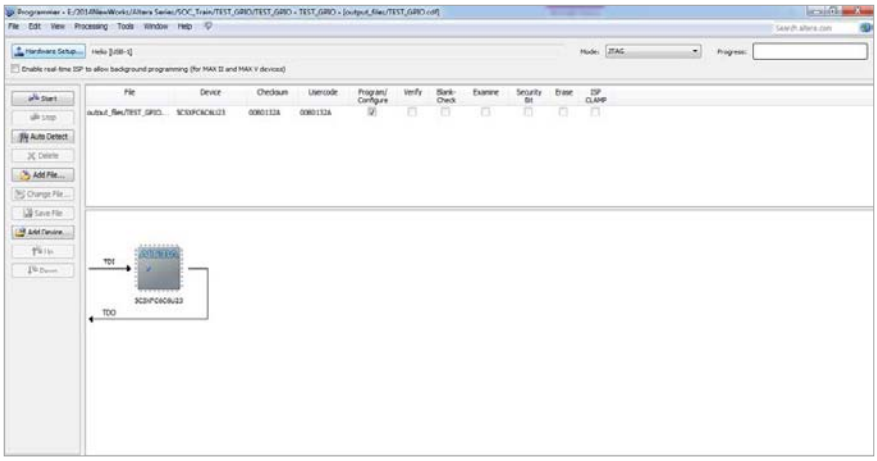


图 3 下载界面

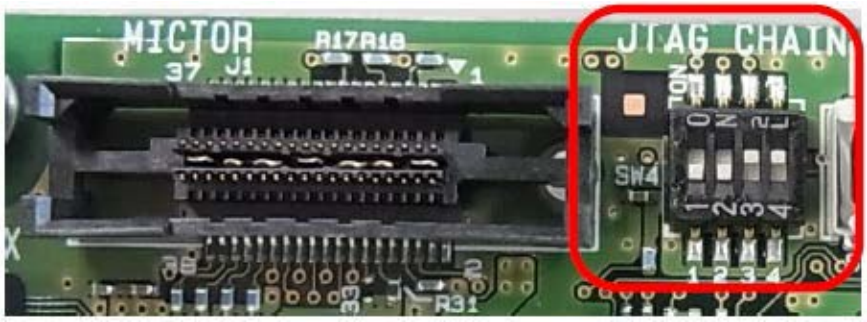


图 4 FPGA JTGA 下载配置 SW4 选项（SW4-1 ON，默认是 SW4-1 OFF）

分别配置输出频率为 60MHz、80MHz、100MHz、150MHz 以及 400MHz，使用 Agilent MSO7032A 型号示波器捕捉波形如图 5~9 所示。

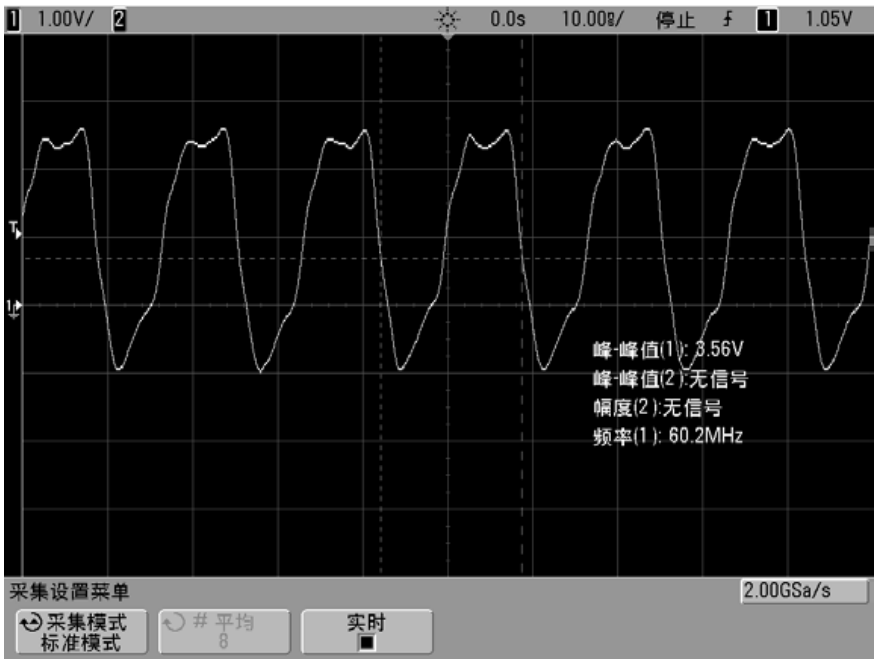


图 5 60MHz 时 GPIO 管脚出波形图

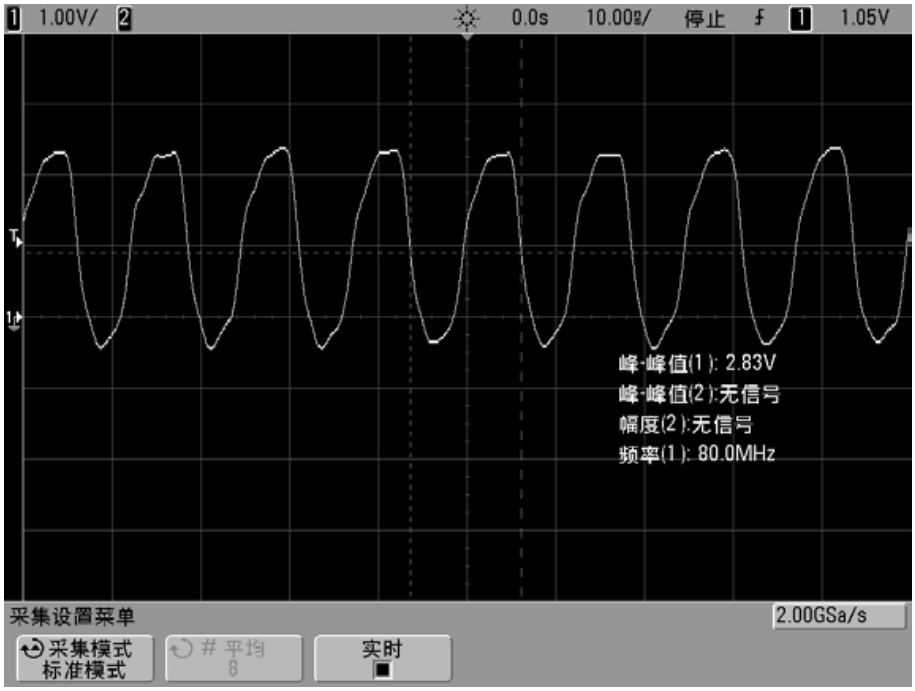


图 6 80MHz 时 GPIO 管脚输出波图

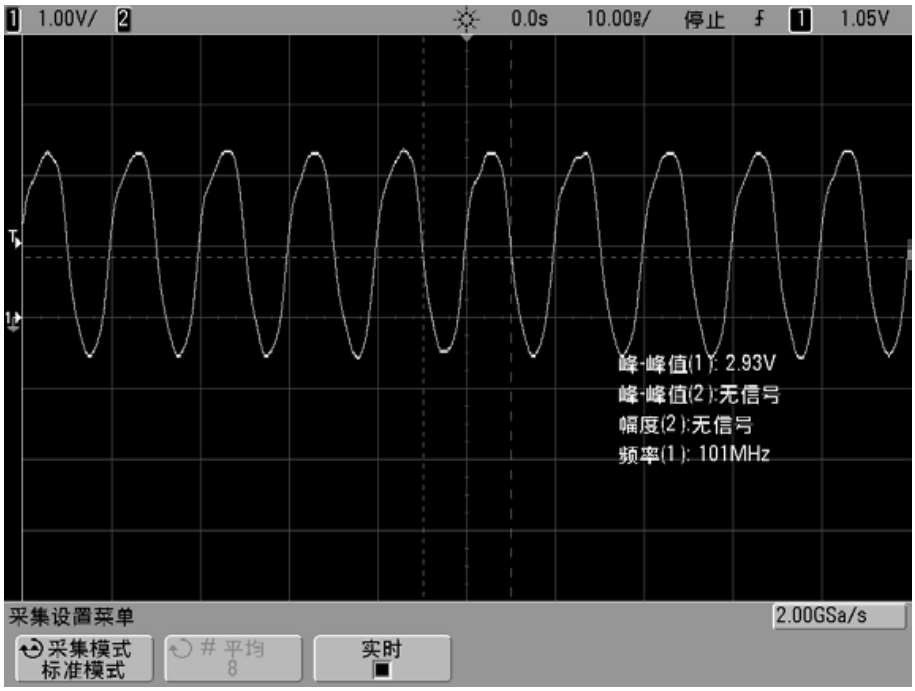


图 7 100MHz 时 GPIO 管脚出波形图

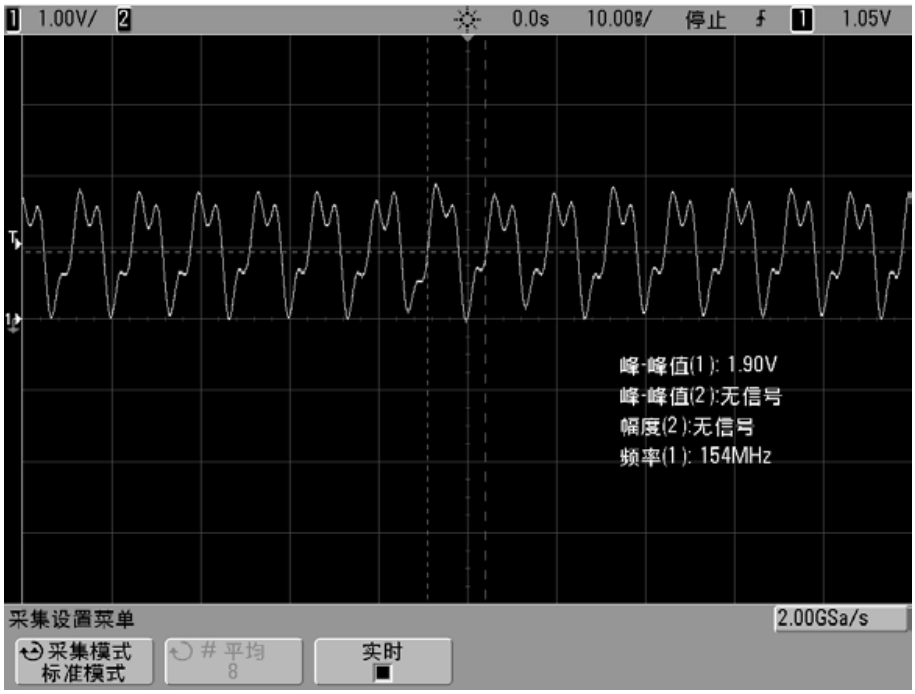


图 8 150MHz 时 GPIO 管脚输波形图

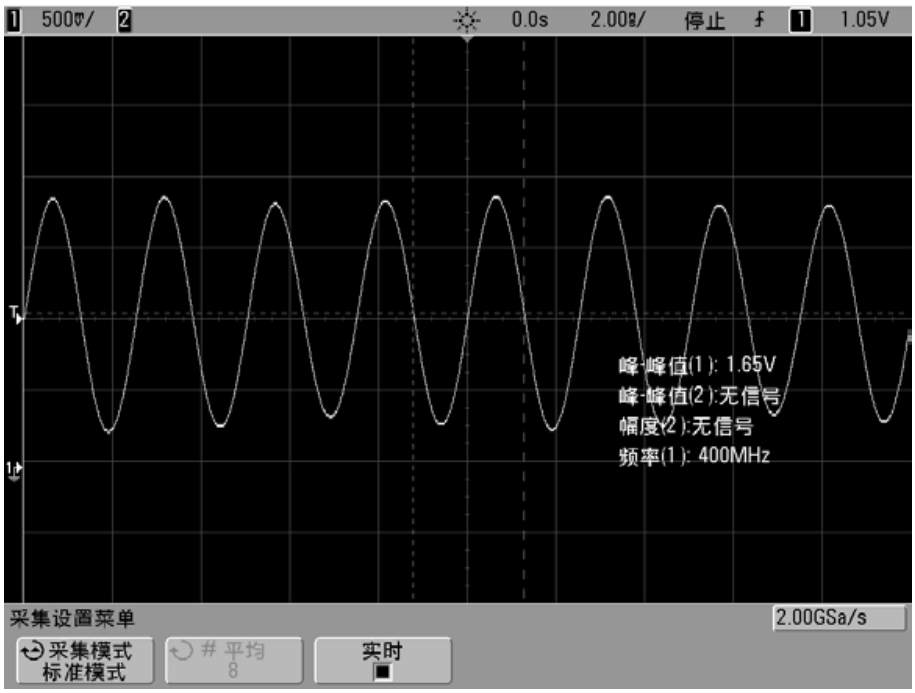


图 9 400MHz 时 GPIO 管脚输出波形图

上述时钟输出管脚，电平配置为 3.3V LVCMOS 电平。通过对比上述图 5~ 图 9 的波形，参照表 1 中 CMOS 的电平表，说明 GPIO 管脚的输出时钟在 100MHz 以上时，其峰峰值已经处于 CMOS 电平的未定义区间，容易产生不定逻辑状态。不建议使用。

表 1 TTL 和 CMO 逻辑电平

Device Type	Logic 0	Logic 1
TTL	0 to 0.8v	2.0 to 5v (V_{CC})
CMOS	0 to 1.5v	3.0 to 18v (V_{DD})

总结：Cyclone 系列的 GPIO 管脚，配置在 3.3VLVCMOS 电平时，其工作频率建议不要超过 100MHz，利用 Helio Demo 板实测结果验证这一结论。所以建议大家在设计的时候，可不要盲目追求系统芯片主频性能，一定要关注细节性能。细节决定成败呀！

另外，在使用 Helio Demo 板的 FPGA 应用时，指定管脚的时候需要注意，有些管脚是指定到 HPS 一侧的，无法进行 FPGA 可编程配置。

详情请查看：

http://bbs.eeworld.com.cn/thread-463010-1-1.html

1.2.2 Helio Cyclone V SoC 开发板应用之 linux 准备工作

通过阅读了用户参考手册以及上一篇 FPGA 的入门体验之后，对 Hleio Demo 板有全新的认识。现在趁热打铁，重点介绍一下 Helio Demo 板运行 linux 系统之前的一些准备工作体验吧。

首先，将 SW4 中的 SW4-1 管脚再回复到 OFF 的默认状态。如图 1 所示。表示 Demo 板可以继续工作在 HPS 关功能。



图 1 JTAG 配置按键位置示意图

按照用户手册（最新版本是 1.3 版本）中的介绍，将带有 linux 系统文件的 SD 卡安装到 Demo 板中，下载 silabs 的 UART 转 USB 驱动程序，安装完成后，连接 USB 线从 J9 接口到电脑 USB 接口。这样，运行 linux 系统的基本工作就已经完成了。接下来的一个重要工作，就是运行终端软件 Tera Term 了。

双击 Tera Term 软件图标，弹出图 2 所示界面，点选“Serial”口模式。

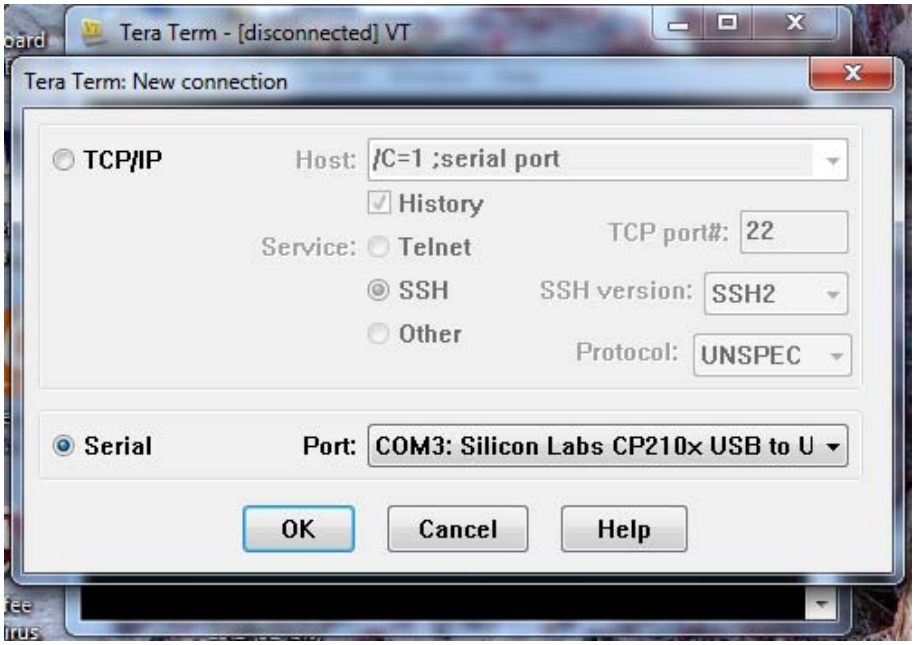


图 2 Tera Term 新连接界面

点击“OK”按钮后，弹出图 3 所示的界面。选择“Setup”→“Serial port”，弹出图 4 所示串口置界面。

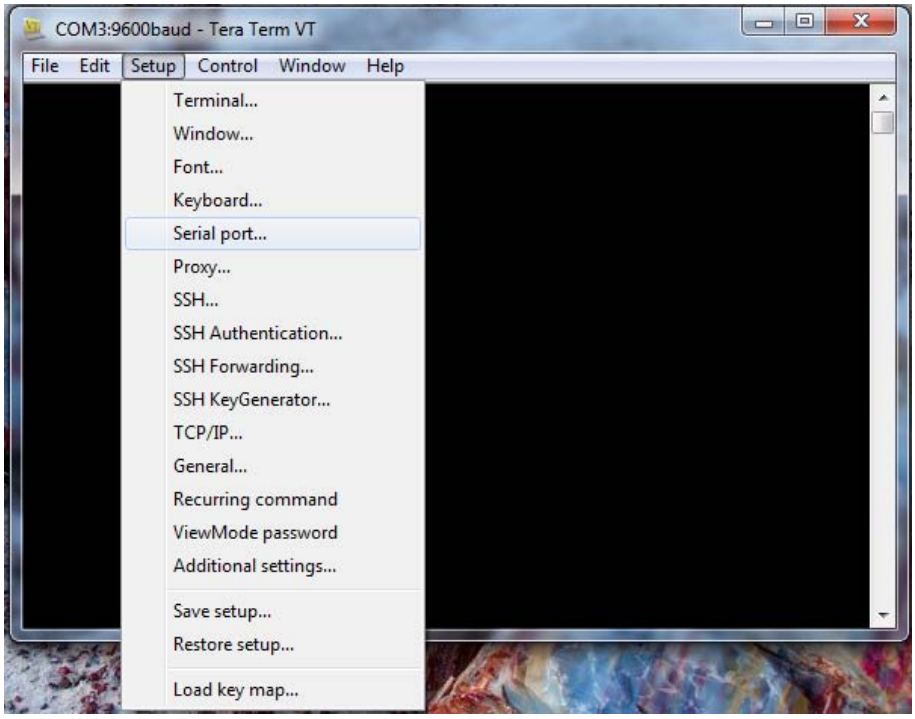


图 3 Tera Tem 主界面

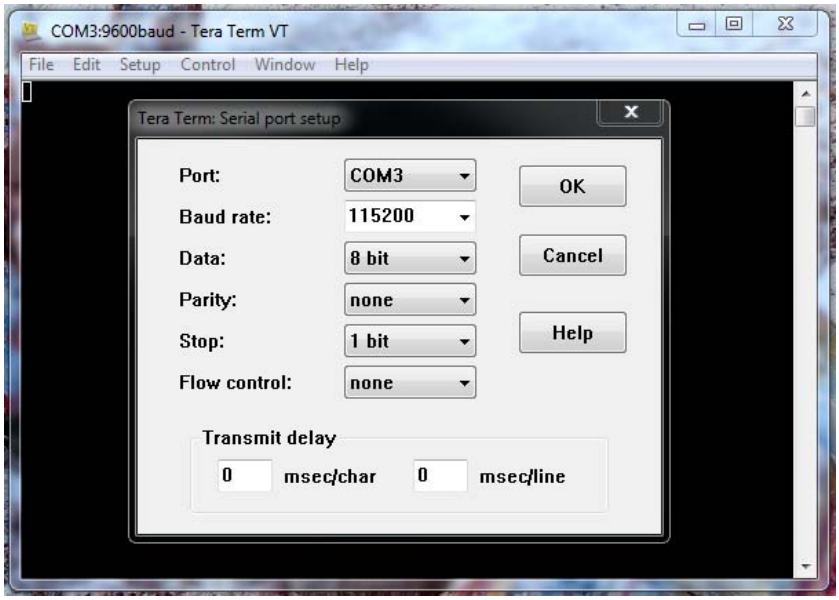


图 4 串口端口配置

在图 4 中的串口端口配置中，需要注意波特率参数的设置，在 1.3 版本以前的用户手册中，提到的波特率设置一般都是 57600，在更新的 linux 系统文件中，需要配置 115200，才能正常显示。

串口参数配置完成后，点击“OK”，关闭 Demo 板上的 PWR 开关，然后重新打开。Tera Term 主界面上弹出相关 linux 系统的信息。至此，demo 板已经配置好，进入到 linu 系统了。

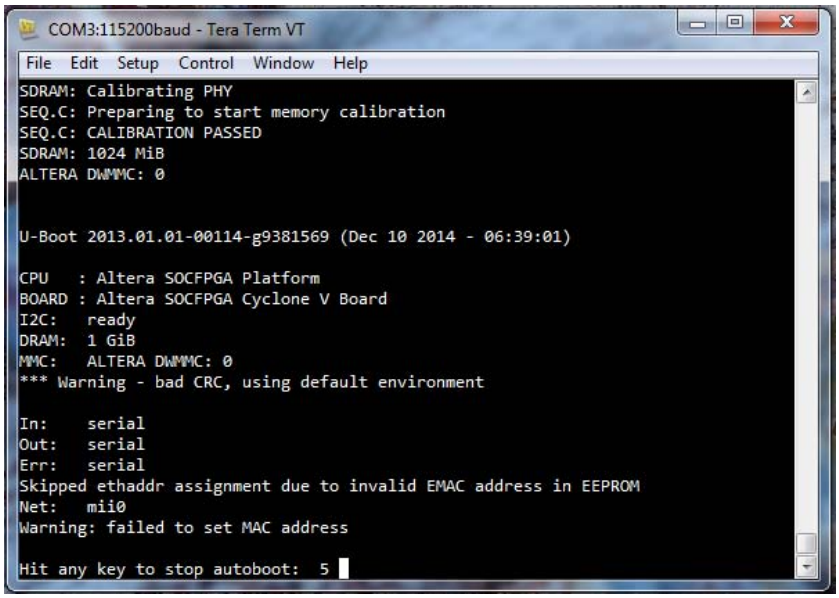


图 5 进入到 linux 界面

上述简要介绍了 Helio 开发板进入 linux 系统的基本流程和相关细节工作。后续文档将启动 linux 下 demo 板的开发学习之旅。

详情点击：

http://bbs.eeworld.com.cn/thread-463075-1-1.html

2. 让 SoC 飞一会儿吧

2.1 Helio Cyclone V SoC 开发板应用之高速 DAC 波形发生一

很多场合下，需要波形发生器产生任意波形以满足特定应用需求。比如一些光学压电陶瓷器件，通过施加在器件两端的电压有规律线型地调节，就可以改变光学的通过路径，最终产生需要的调谐光路信号。本文先重点介绍高速 DAC 波形发生的器件和 demo 板功能，以及针对 Helio SoC 开发板中的 FPGA SoC 器件进行必要的时序和功能分析。

LTC1668I 是 Linear 公司的一款高性能 16bit,50MSPS 的 DAC 器件，带有并行同步时钟接口。具体管脚示意图如 1 所示。

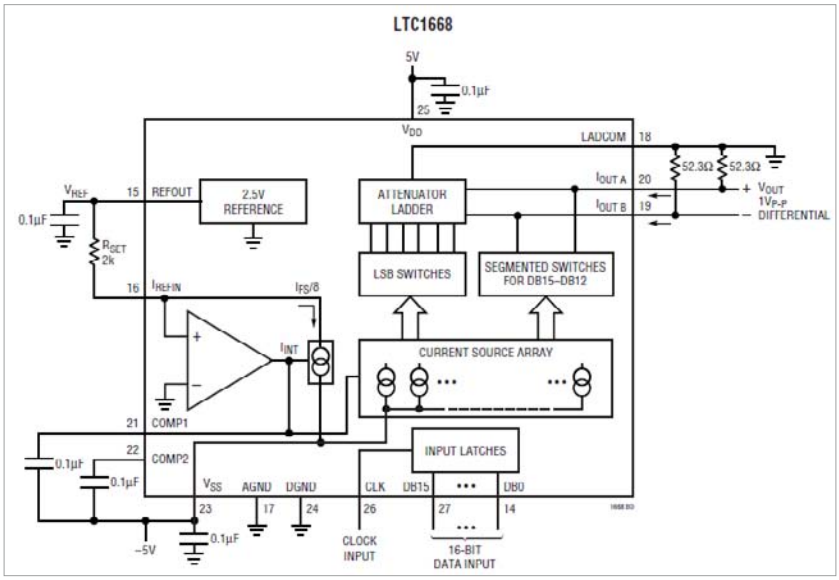


图 1 LTC1668 器件理框图

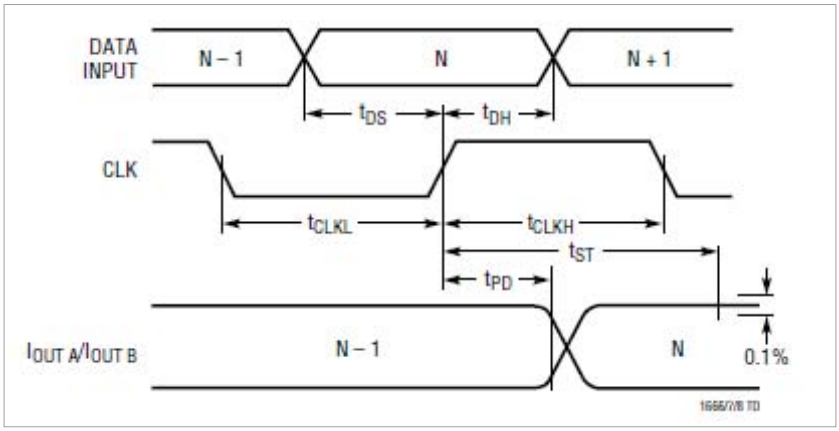


图 2 LTC1668 时序图

设计需求是需要提供足够的波形完整性，时钟频率在 1KHz，每个周期的采样点数要达到 20000 个点。只有满足这个要求的三角波形才能够满足光学压电器件的线性精度。

针对上述需求，分析如下：

三角波形发生，可以考虑使用地址发生器产生地址，使能片内 1 端口 ROM, 写入 65536 个地址内容，即数据。

按照 50MSPS 采样速率分析，1KHz 20000 个点的三角波形，需要每半个周期内，即 0.5ms 内输出 10000 个步阶为 1 的上升或下降的点，对应的 DAC 的输入时钟即为 1/(0.5/10000/1000)Hz, 即 20MHz。

针对 1 中的 65536 个地址的 ROM, 只需要使用其中的 20000 个地址，就可以完成上述的功能需求。

根据上述分析，FPGA 实现上述功能的原理框图如 3 所示。

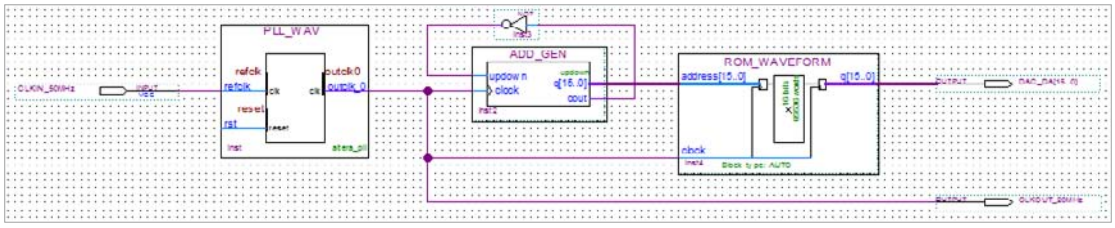


图 3 三角波形发生 FPGA 原理框图实现

Linear 公司提供了 LTC1668 的 demo 功能板做功能验证，功能板的原理图如 4

所示。

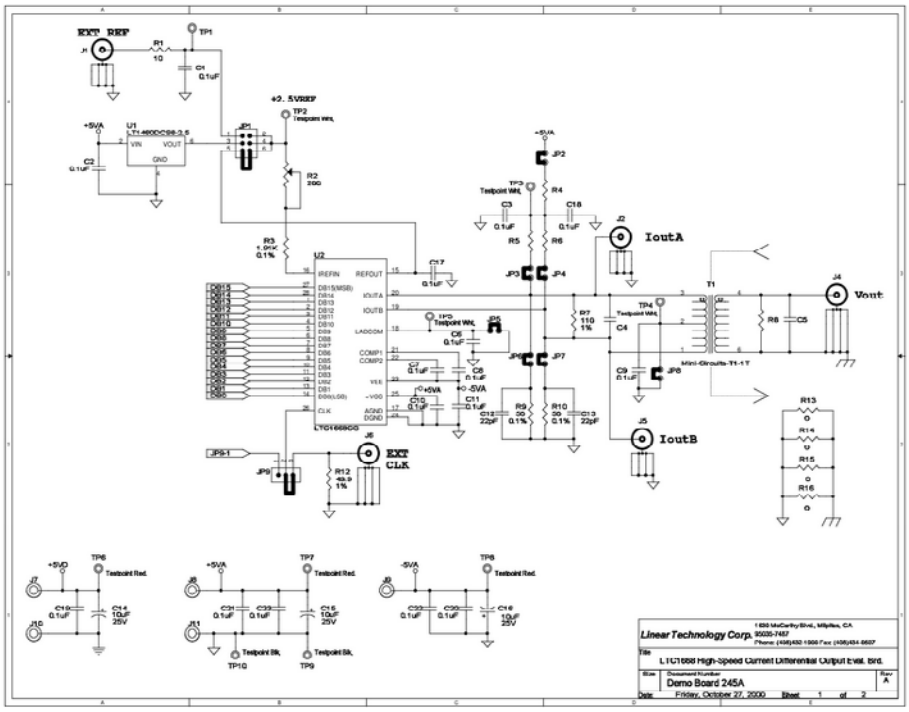


图 4 LTC1668 高速 DAC 的 DEMO 板原理图

详情请查看：

http://bbs.eeworld.com.cn/thread-463108-1-1.html

附录

【视频】2014 SoC 合作伙伴研讨会图像处理与识别

【视频】Altera 发售其第一款 SoC 器件

【视频】2014 SoC 合作伙伴研讨会 Terasic Technologies

【视频】Altera SoC Live Demo -- PCIe on Linux from Embest

【视频】深入了解：SoC FPGA 之系统性能 4

骏龙 SoC 培训课程

【Altera SoC 体验之旅】SoC 初体验之 HPSFPGA

ARM DS-5 使用文档



版权声明

1、《Altera Cyclone V SoC 深度体验之旅》内容来于 EEWORLD 论坛（http://bbs.eeworld.com.cn），系坛友们参与 Altera SoC 深度体验之旅活动产生的内容。由于本书内容乃坛友们自主提供，Altera 尊重坛友意见的相互分享，对于本书内容的真实、正确性不承担任何责任。

2、《Altera Cyclone V SoC 深度体验之旅》著作权属 Altera 和 EEWORLD 共同拥有。

3、本着开源思想，我们授权任何对 Altera SoC 以及开发板有兴趣的工程师免费下载、复制、传播该书。

4、用于商业用途须经 Altera 和 EEWORLD 书面同意。

Altera&EEWORLD 社区

