The study of linux's porting to S3C2410

Dian-Tao li, Yin-Jing GUO

*(College*of Information and electronicEngineering, Shandong University of science and technology, Qingdao 266510, China*)*

Abstract **—This article we mainly analyse the source codes of the linux, we can modify the proper file that related to the architect,wealso use the cross-compile tools arm-linux-gcc and we eventually port to the processor. S3C2410,this article's method is easy and it is suitable for practical applications.**

Keywords **—** ***linux port S3C2410***

Manuscript Number: 1674-8042(2010)supp.-0126-03

dio: 10.3969/j.issn1674-8042.2010.supp..35

1 Introduction

.Recently, with the development of technology, the processor's speed is faster and faster, while the while the power becomes slower, in one word ,the performance becomes better and better, so the tasks that the processor can deal is more and more complex, so wheather we can port the OS to the processor come into our eyes. For the reason that the linux has a open software also have a good function , so we select it as our OS.

2 The hardware Platform

This article that we discuss the platform called S3C2410 base on the arm9 that the Sumsang company develop ,it apply in the electronic construction region it also integrate some device, such as the store device for example SDRAM,NAND flash .and network card DM9000 also have some USB interface and Audio Frequency interface .These devices can give us much convenient

3 The source code of linux

3.1 The architecture of linux

Linux's source codes organized well. Different files that have different functions locate in the different directories, this architecture can make the linux easy to debug .Also this source codes can give us some convenient when we port to the different processor.

Now let me show some important directories.
/arch that related to the architecture[1]
/driver this directory includes many driver program.
/fs file system directory.

Received: 2010-5-25
Corresponding author: Dian-tao Li (lidiantao2007@163.com)

3.2 This article we modify the files

When we port the OS into the target system, we mainly change the files that locates in the /arch .For the reason that we adopt the high edition,

/driver have our needed drivers and we don't modify them.[2]

4 The core of porting

4.1 The concept of porting

Porting refers to the codes of OS can run the different processor ,so we can make the processor have the function of the OS[3].

4.2 The problem of porting

Although linux is an open source codes's OS, also its source trees are organized well, but the embedded system's porting have many problems such as the storage devices and these problem need we to consider.

4.3 The tools and kernel's requirement

4.3.1 The choice of compiler

For the reason that the kernel that we compile the source codes of linux can run on our platform, so we can't use gcc which can compile the program and the binary file can run on the PC, what we need is called cross-compile that can compile in one processor while run in different processor. The cross-compile that we use is called arm-linux-gcc, and it can generate the program that can run in arm processor[4]

4.3.2 The edition of the linux kernel

The edition of linux kernel is higher and it have great functions, but this will also have many problems such as the high edition will generates more bugs, so we will compromise it .Final, we choose 2.6.24 and it can meet our request also it is stable.

4.3.3 The edition of the compiler

Because our kernel must be compiled by our cross-compiler ,so the edition of the r cross-compiler is also important, and we choose arm-linux-gcc 4.2.2[5].

5 The process of porting

5.1 Modify the top level of makefile

Firstly we should modify some top level's Makefile, we should modify like that:
ARCH=arm
CROSS_COMPILE=/usr/local/bin/arm-linux-
Especially for the second parameter, we should use by the absolute path, so that the system can find our cross-compiler[6].

5.2 Set the basic nand flash partition

For the reason that our platform use the NAND Flash of 64M as our storage device, so we should create a partition table to define the storage device. The files that we modify locates in
arch/arm/mach-s3c2410/devs.c
We can modify like this:

```
include<linux/mtd/partitions.h>
 include<linux/mtd/nand.h>
 include <asm/arch/nand.h>
/*the64M   NAND Flash partitions table*/
Static struct mtd_partition parttion_info[]={
   {     name: "bootloader",
         Size:0x00100000,
         Offset:0x0,
    },
   {     name: "kernel",
         Size:0x00300000,
         Offset:0x01000000,
    },
{        name: "rootfs",
         Size:0x02800000,
         Offset:0x0040000,
   },
{    name: "user",
        Size:0x01400000,
        Offset:0x02c00000,
    },
```

These codes can generates the four parts of partition table. The four parts are bootloader, kernel, file system and user applications.

Next, we should make the kernel support the NAND FLASH, so we should increase some driver program, and the programs are like that:

```
Struct s3c2410_platform_nand superlpplationform={
Tacls:0,
Twrph0:1,
Sets:&nandset,
Nr_sets: 1,
};
Struct platform_device s3c_device_nand={
.name="s3c2410-nand",
.id=-1,
.num_resources=ARRAY_SIZE(s3c_nand_resource),
.resource=s3c_nand_resource,
  //these dodes can support the nand flash
.dev={
          .platform_data=&superlpplatform
}
```

5.3 The configuration of the kernel

If we want the system to run [6]stable and faster ,the configuration is very important, and we choose the configuration that related to the system hardware, and the picture as follow:

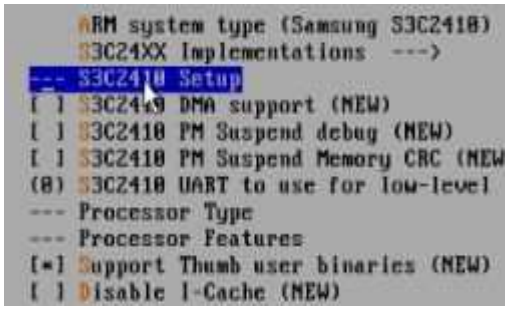

Figure 1 the kernel's configuration

5.4 The kernle's compiling and installing

When we configure the kernel the next step is compiling thg kernel, while the kernel also should be compressed, and the format is bzImage, and the command is like that: make zImage

This picture is the compiling's picture

Figure 2 linux's compiling

After that we should install the modules because the drivers are compiled as modules ,and we should install the modules .the command is :
make module_installed

5.5 linux's booting and running

We can use u-boot as our bootloader to boot the linux, and u-boot can load the kernel in the NAND flash to the SDRAM. because the address space of

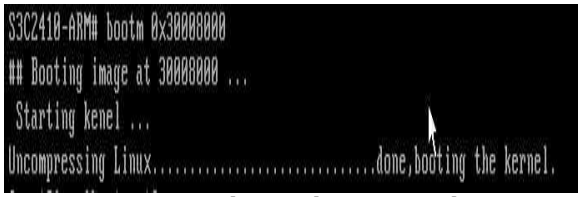

Figure 3 linux's decompressed

SDRAM is 0x30008000-3fffffffff,and this parameter is supplied by the manual of Samsung. So we can see the pictures as follow

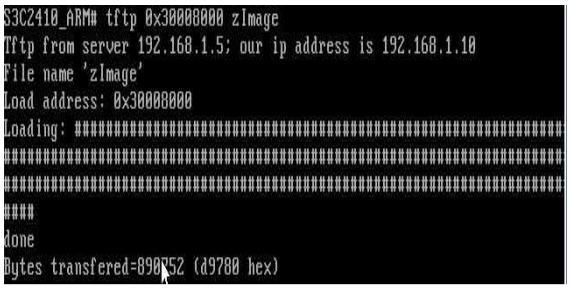

Figure 4 linux's running

6 Conclusion

After that steps we can see the kernel can run stablly on the S3C2410. We adopt an easy way to port linux to S3C2410.For the reason that the method we adopt has both less complexity and high effciency so it will give help to our embedded development.

References

[1] Fan Lei linux source codes BeiJing:people's public 2002. PP68-70
[2] Ying Yu Yao Feng The Foundation Course of linux & UNIX development BEIJING :BEIJING PUBLIC,2004 PP30-40
[3] Ayelet Israeli, Dror G. Feitelson The Linux kernel as a case study in software evolution Journal of Systems and Software, Volume 83, Issue 3, March 2010, PP 485-501
[4] David Egan, Paul Zikopoulos, Chris Rogers The Linux Operating System DBAs Guide to Databases Under Linux, 2000, PP 1-23
[5] The Linux kernel as a case study in software evolution ComputerVolume 51, Issue 14, 10 October 2007, PP 4050-4069
[6] High-speed data acquisition with the Solaris and Linux operating systems Fusion Engineering and Design, Volume 48, Issues 1-2, 1 August 2000, PP 193-197

(From P.120)

range of divide-by-3, it's from 25.5 to 30.3 GHz. The experimental data for divide-by-3 ILFD is not available at the present time. Table 1 shows the comparison between our presented injection-locked dividers and other ILFDs.

4 Conclusions

CMOS direct-injection divide-by-3 frequency dividers have been proposed and implemented in the 0.18um 1P6M CMOS and 90nm 1P9M CMOS technologies. The operation principle of the ILFDs has been described. And measurement results have been presented. Measurement data shows that the direct-injection divide-by-3 ILFD is potential for RF system applications.

.

5 Acknowledgments

The authors would like to thank the Staff of the CIC for the chip fabrication and technical supports.

References

[1] S.-L. Jang, C.-W. Chang, C.-F. Lee, and J.-F. Huang, " Divide-by-3 LC injection locked frequency divider implemented with 3D inductors," *IEICE Trans. Electronics.*, Vol. E91-C, No. 6, pp. 956-962, June, 2008.
[2] H. Wu and L. Zhang, "A 16-to-18GHz 0.18μm epi-CMOS divide-by-3 injection-locked frequency divider," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2006, pp. 2482 – 2491.
[3] S.-L. Jang, C.-Y. Lin, and C.-F. Lee, " A low voltage 0.35 um CMOS frequency divider with the body injection technique," *IEEE Microw. Wireless Compon. Lett.*, vol. 18, no. 7, pp. 470-472, July, 2008.
[4] S.-L. Jang, C.-F. Lee and J.-C. Luo,"A CMOS LC injection-locked frequency divider with the division ratio of 2 and 3," *Microwave and Optical Technology Lett.*, pp. 1263-1267, May, 2009