RealBOX,
A Novel Approach to MiniRTL Implementation for Instrumentation And Control Applications

N.Akdeniz, T. Aydın
Havelsan A.Ş.
06520, Ankara, Turkey
{ nurpinar, taydin }@ havelsan.com.tr

Abstract

realBOX is a PC104 form factor computer, based on a 486DX2 processor with 8MB RAM and 8MB Disk-On-Chip Millennium. MiniRTL operating system boots from Disk-On-Chip and expands itself to RAM when the power is applied to the box. Our initial aim and the motivation behind our work were to have an embedded system with the following critical requirements which were fulfilled: -real-time scheduler, -operation without Hard Disk, -small footprint (so a solid state memory preferably the Disk-on-Chip), -no file system checks when power goes off and comes again.

Keywords: Real-time Operating Systems, Embedded Systems, Disk-On-Chip, Instrumentation

1 Introduction

Presented in this paper is a rugged computing and control box with an embedded hard real-time operating system. PC104 based CPU and Data Acquisition boards constitute the realBOX hardware while MiniRTL operating system along with the application modules reside in Disk-On-Chip. This hardware and software combination stands as a reasonable substitute to expensive embedded control hardware running with high-royalty real-time operating systems. We have applied this approach to an instrumentation project, which can be qualified as a hard real-time application and got satisfactory results. In fact what makes this approach original is embedding the MiniRTL (the minimized version of the spreading Real-time Linux) into a solid-state memory like Disk-On-Chip.

Trade-off factors considered while choosing the MiniRTL and the PC104 type computer hardware, a brief description of how the operating system was embedded into the Disk-On-Chip and a simple evaluation application for the proposed system are described within the following sections.

2 RTLinux and PC104 Combination

Embedded computer systems differ from the desktop computing platforms in that they are more likely to be subjected to harsh environmental conditions and they are usually supposed to do their job without any operator intervention for extremely long periods of time. Apparently, those additional performance requirements are what make them expensive. It should be noted that the price to pay will increase when real-time responsiveness is expected from the embedded system.

In a quest for a cheap and optimal embedded software and hardware combination, Linux and the PC104 are the arrival points for us. Linux became an attractive alternative to commercial real-time operating systems by pragmatic efforts to add real-time behavior to it. Advantages of using a real-time operating system based on Linux are clearly explained in further by N.Mc.Guire.[1] Among Linux based real-time operating systems available, we have selected MiniRTL, a minimized version of the Real-time Linux, which is a hard real-time kernel running the Linux as its lowest priority thread. The distinctive features of the MiniRTL are its minimal size (it can

fit into a 1.44 MB floppy) and that it completely runs from RAM memory by expanding itself from a static record on boot time. Especially the later feature is vital for an embedded system having a power on/off button as the only user interface. No *fsck* operations are initiated, so no need to interact with the box during boot process after when power goes off and comes again. Startup scheme of such a system is depicted in FIGURE 1.

For the hardware part, PC104 form factor computer and peripheral boards are used for their compactness, stack through expandability, modularity and their price performance ratio over VME hardware that we have used for our applications so far. In addition, most PC104 CPU board manufacturers reserve an expansion socket for solid-state memory devices like Disk-On-Chip into which we have planned to embed the operating system along with the application code. By using a rugged modular housing like the IDAN system from the Real-time Devices Inc. now we have a real real-time computing and control box; realBOX as we called it.

As a result, combination of the x86 based PC104 hardware and MiniRTL formed a cheap and applicable solution for most real-time embedded control applications.

3 MiniRTL on Disk-On-Chip

MiniRTL was originally developed to boot from floppy or from flash-disk with some minor modifications.[2] Using a solid-state memory like Disk-On-Chip instead of the floppy to keep the operating system with the eventual application code, would be a more applicable approach. Because, expecting the floppy to function without giving a failure after an unlimited number of on/off cycles during the lifetime of the system is impractical and what we need is an install-and-forget type embedded computer. Using flash-disk on the other hand seems more reasonable but has some cost and reliability related concerns as well.

Disk-On-Chip is a solid-state memory with full hard-disk emulation. The one that we used in our configuration is an 8 MB Disk-On-Chip Millennium. [6] It has a 1.4Mbyte/sec read rate and it is observed that the start-up time for the Disk-On-Chip version of the MiniRTL is 3 times shorter than booting from floppy. That is another advantage of booting MiniRTL from Disk-On-Chip.

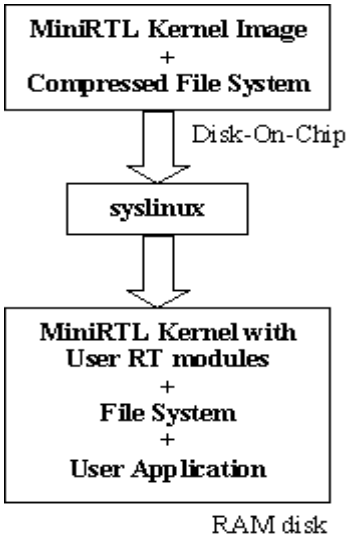

FIGURE 1: System startup scheme

3.1 Building the New Kernel Image

Kernel 2.2.20 was preferred because all the required patches are applicable for only this version. In order to extract the compressed file system into RAM disk during boot time, initrd and linuxrc patches were applied to Kernel 2.2.20. Inter-module and mtd patches were also applied, so that we could select the memory technology device as Disk-On-Chip Millenium with other complementary options.[5] Certainly, the last patch applied was RTLinux3.1.

To reduce the size of modules, debug option was unchecked when making menuconfig to RTLinux. RTLinux modules were replaced by those exist in MiniRTL to make the kernel versions be compatible. All the kernel modules in original MiniRTL were deleted since they are linked statically in Kernel 2.2.20. By doing so, the new image takes only 490 KB.

3.2 Burning the Disk-On-Chip

In order to inform the system about the boot record location, following devices were created by means of the root.dev.mk script:

/dev/nftla
/dev/nftla1

where the device name of the Disk-On-Chip is nftla1.

In syslinux.cfg file “boot=/dev/fd0” line is modified as “boot=/dev/nftla1”, since we want to boot from Disk-On-Chip instead of floppy and Disk-On-Chip was made bootable by syslinux.

Finally following files in compressed form were transferred to Disk-On-Chip:
root.rtl, etc.rtl, modules.rtl, log.rtl, local.rtl, syslinux.cfg, bzimage.

It's worth to mention that the Disk-On-Chip must be formatted with the doc42.exb to properly boot from.

After resetting the box, following message related to the Disk-On-Chip was displayed;

```
-------------------------------------------
M-Systems DiskOnChip driver. (C) 1999 Machine
Vision Holdings, Inc.
Using configured probe address 0xe8000
DiskOnChip Millennium found at address
0xE8000
Flash chip found: Manifacturer ID: 98, Chip
ID: E6 (Toshiba TC58V64AFT/DC)
1 flash chips found. Total DiskOnChip size:
8Mbytes
M-Systems NAND Flash Translation Layer
Driver. (C) 1999 MVHI
$Id: nftl.c,v 1.57 2000/12/01 17:51:54 dwmw2
Exp $
Partition check:
nftla: nftla1
-------------------------------------------
```

3.3 Running The User Modules and Application

User modules and the application were compiled on a development computer having Kernel 2.2.20 and glibc2.0.7. Compiled module shall be included in modules.rtl. As for the application executable to run at start-up, inittab file was modified.

4 Sample Application

Developed control application consists of a real-time thread module and a user space application communicating with each other via a real-time FIFO. Real-time module reads the sensor outputs periodically from PC104 compliant interface boards and sends these readings to the user space application through a real-time FIFO. Upon receiving the sensor data, user space application prepares a UDP packet consisting of the received sensor data and posts it to a central host where the sensor data is replayed with giving an almost real-time impression. An architectural diagram of the sample application is given in FIGURE 2.

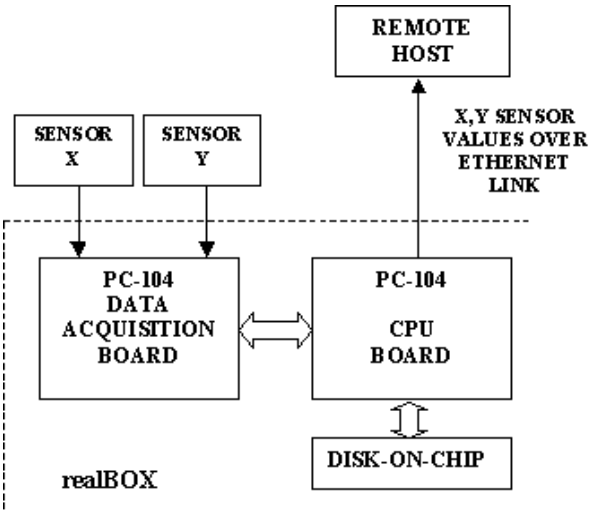

FIGURE 2: Sample application architecture

Simplified source code for RTLinux module is given below ;

```
#include <asm/io.h>
#include <rtl.h>
#include <time.h>
#include <pthread.h>
#include <rtl_fifo.h>

#define  DAQ_X_BASE    0x320
#define  DAQ_Y_BASE    0x340
#define  PERIOD_MS     50
pthread_t thread;

void * start_routine(void *arg)
{
   struct sched_param p;
   p . sched_priority = 1;
   pthread_setschedparam (pthread_self(),
   SCHED_FIFO, &p);
   pthread_make_periodic_np (pthread_self(),
   gethrtime(), PERIOD_MS*1000000);
   while (1)
   {
    unsigned x,y;
    char buff[20];
    pthread_wait_np();

    x=inw(DAQ_X_BASE);
    y=inw(DAQ_Y_BASE);

    sprintf(buff,"%5d %5d \n",x,y);
    rtf_put(1, buff, sizeof(buff));
   }
   return 0;
}

int init_module(void)
{
 rtf_destroy(1);
 rtf_create(1, 4000);
 return pthread_create (&thread,
        NULL, start_routine, 0);
}
```

```
void cleanup_module(void)
{
 rtf_destroy(1);
 pthread_cancel (thread);
 pthread_join (thread, NULL);
}
```

At the other end user space application opens the real-time fifo (/dev/rtf1) with a simple open system call;

```
open("/dev/rtf1", O_RDONLY);
```

then polls the output of the real-time module (so the FIFO) and sends the received sensor data as UDP packets to server host infinitely .

5 Conclusion

A particular and complete MiniRTL configuration, which works on PC104 x86 hardware platform without any magnetic disks, have been developed to act as an embedded and real-time system for control applications.

Main advantages of the system are the off-the-shelf availability of the low price hardware platform and the royalty free real-time operating system with the complementary Linux reliability, open source, applicability of standard APIs and highly preferable price/performance ratio.

Described system is now completely in use for an instrumentation application and it performs well enough.

For code development, a normal RTLinux terminal can be used as the host and/or development station. Debugging and development can be done on this host and the finalized modules and user space applications can be transferred to the Disk-On-Chip with accompanying file system items and boot image for a stand-alone operation. In a modular structure like PC104, it is even possible to use a cooking hard disk layer for development phase. This layer can be removed when everything is done and the file system with the kernel image has transferred to Disk-On-Chip, which will be ready to boot from when the power is applied to the box.

With the availability of the mini-http server in original MiniRTL configuration it will be possible to monitor the detailed status of the box and the controlled process by simple cgi-scripts. Using multiple partitions on Disk-On-Chip will allow developers to have a read/write solid state memory space to log process data and store system parameters.

6 References

[1] Nicholas Mc. Guire, MiniRTL Hard real-time Linux for embedded systems
[2] Nicholas Mc. Guire, miniRTL – A Minimum Real-time Linux System
[3] Real Time Devices Inc., 2000, CPU Module User's Manual
[4] FSM Labs Inc., Getting Started with RTLinux,
[5] David Woodhouse, 2000, MTD internal API documentation
[6] www.m-sys.com/