Porting Window CE Operating System to Arm based board device

Byungchan An*, and Woonchul Ham**

* Division of Electronics and Information Engineering, Chonbuk National University, Chonbuk, Korea
(Tel : +82-63-270-2399; E-mail: chanist@mail.chonbuk.ac.kr)
** Division of Electronics and Information Engineering, Chonbuk National University, Chonbuk, Korea
(Tel : +82-63-270-2400; E-mail: wcham@mail.chonbuk.ac.kr)

Abstract: Hand carried computing machinery and tools have been developed into an embedded system which the small footprint operating system is contained internally. Windows CE which is one of imbedded operating system is a lightweight, multithreaded operating system with an optional graphical user interface. Its strength lies in its small size, its Win32 subset API, and its multiplatform support. Therefore we choose to port this OS on Arm based board that is provided high performance, low cost, and low power consumption.
In this paper, we describe the architecture of ARM based board, the feature of Windows CE, techniques and steps involved in this porting process.

Keywords: Windows CE, Embedded, Device Driver

1. INTRODUCTION

An embedded system is accomplishing complex task to fast development of the hardware and demand by user. The software these computers are running is becoming increasingly sophisticated with a wide range of applications, including those which require connectivity such as e-mail and web browsing. The major operating systems for embedded, such as Microsoft Windows CE and embedded Linux, have been used widely. In this paper, we will describe one of our efforts of porting Windows CE to Arm based board device. This paper is organized as follows: we will describe the target platform. We described the system architecture and memory map. Secondly, we describe briefly Windows CE OS. Thirdly, we describe briefly about the porting process.

2. ARCHITECTURE OF THE HARDWARE PLATFORM

2.1 ARM based board Architecture

The ARM based board employs a small single board computer featuring the powerful, highly integrated and power efficient Intel StrongARM SA1110. It includes 16MB of Flash, 32MB of SDRAM, an Ethernet controller, USB(client) support, a Compact Flash interface, three serial ports, the StrongARM-1110's LCD controller interface, a CODEC interface, and 16 general-purpose I/O lines.

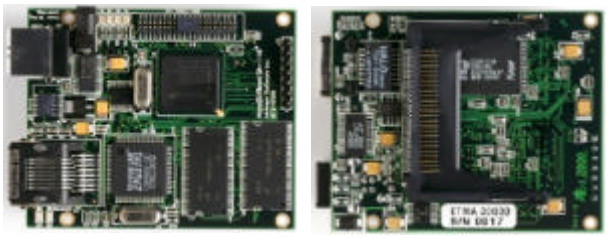

Fig. 1 target device

2.2 Memory Map

The SA1110 MMU logically expands the physical address space by mapping addresses in a large virtual address space.
The entire virtual address space is 4GB. The upper 2GB is used by the system for various purposes. The kernel mode allows access to the entire virtual address space, whereas the user mode is restricted to the lower 2GB of address space.

The MMU translates virtual addresses generated by the by the CPU into physical addresses. Page Table is a mapping from the physical address location of the RAM, or other types of memory, to a statically mapped virtual address that is used by applications and ISRs. Each entry in the Page Table specifies a physical location in memory, the size of the memory, and the static virtual memory address to which to map it. The static virtual address is specified in the cached memory range and the kernel can then create the uncached address that points to the same physical address.

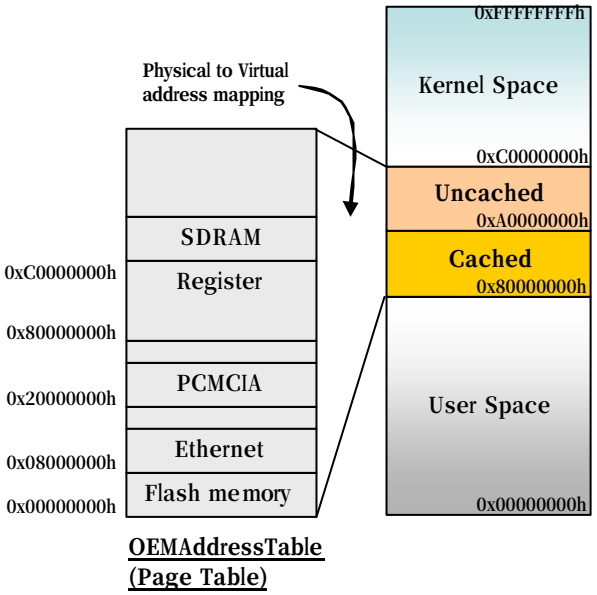

Fig. 2 Memory Map of SA1110

This work was supported in part by the Mechatronics Research Center of Chonbuk National University

3. INTRODUCTION OF WINDOWS CE

Windows CE is a portable, real-time, modular operating system that features popular Microsoft programming interfaces and is supported by tools that enable rapid development of embedded.

3.1 The OS Architecture

Windows CE was developed to support embedded applications with a broad range of different hardware platforms. Windows CE maintains most of the features of the Windows operating systems. Windows CE is componentized and ROMable. Windows CE isn't backward compatible with MS-DOS or Windows. Instead, Windows CE is a lightweight, multithread operating system with an optional graphical user interface. Its advantage is in its small size, its Win32 subset API, and its multiplatform support. Windows CE is designed to support a subset of the already familiar Microsoft API's, which proves to be much beneficial from the software development point of view.

Window CE architecture consists of three main parts: Kernel, OEM Adaptation Layer (OAL), and Boot Loader. The kernel is the core of the OS, while OAL accommodates different processors to allow Windows CE support of a rich set of processors: MIPS, ARM, SHx, PowerPC. Although Windows CE is designed to be portable across processors, it contains processor-specific code. Therefore, OAL layer development is always an interesting and crucial part for porting CE to a given hardware platform. The Boot Loader function is to allow the OS image to be booted from Rom or other devices. Now we describe each of the main CE components.

Kernel: the core functions of Windows CE, e.g., process handling, memory management, resource management and interrupt handling. It is designed to be small and fast. It is provided by Microsoft and is reconstructed by developer based on processor's type and requirement for the system.

OAL: code specific to a particular hardware platform that is built through the use of a given microprocessor, and it is responsible for abstracting and managing hardware resources of the processor. It is commonly supported by board support package (BSP).

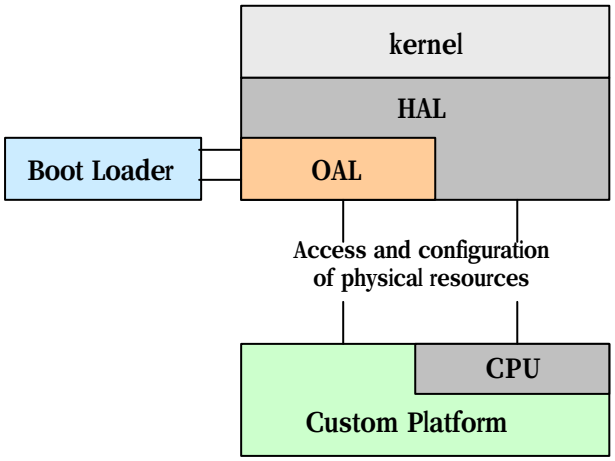

Fig. 3 Block diagram of the OAL architecture on a custom platform

Boot Loader: responsible for booting the system by correctly configuring the processor and peripheral chips. Its job is to set the stage for starting the heart of most devices: the system software. The development of Boot Loader is the first step for porting.

3.2 Device Driver

A device is a physical or logical entity that requires control, requires control, resource management, or both from the operating system. A device driver is a software module that manages the operation of a device, a protocol, or a service.
The architecture of most operating systems requires that device driver run in kernel mode, But Windows CE drivers run in user mode. Device drivers differ from applications in that they're DLLs.

Commonly device driver architecture consists of two main parts: Stream-interface driver and Native driver. The stream-interface device driver model is used most commonly to support installable device and is managed by the device manager. Native device drivers used for built-in devices and are usually managed by the Graphics, Windowing, and Events Subsystem (GWES) module. Note, however, that native drivers can also be managed by the device manager if their upper interface uses the mechanism of the stream-interface model. The stream interface is appropriate for any I/O device that can be thought of logically as a data source or a data sink. That is, any peripheral that produces or consumes streams of data as its primary function is a good candidate to expose the stream interface.

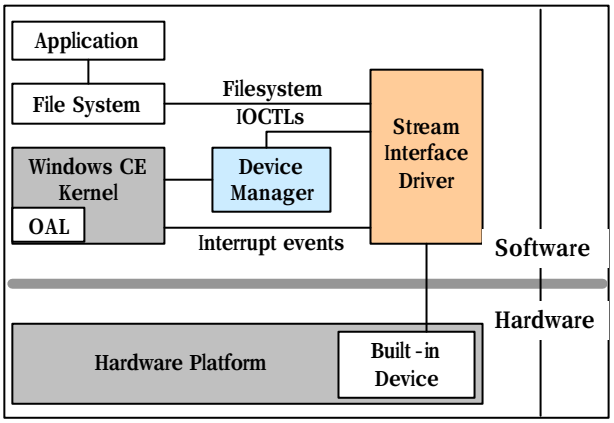

Fig. 4 Block diagram of the streams driver architecture

The stream interface functions themselves are designed to closely match the semantics of the usual file system APIs such as ReadFile, IOControl, and so on. As a side effect of this design, devices that are managed by the stream interface are exposed to applications through the file system; applications interact with the driver by opening special files in the file system.

4. DEVELOPMENT OF THE EMBEDDED SYSTEM

4.1 Boot loader

The boot loader is the beginning of the beginning. The boot loader manages the boot process of the device by initializing the device, downloading the OS image from the development workstation to the target device, and starting the image on the

device.

The boot loader is typically used during the development process to save time. Rather than transferring the development image to the target device through a manual process, such as a flash programmer or the IEEE 1149.1 standard for the test access port and boundary scan (JTAG).

The elements of boot loader must be implemented: OEM startup code, OEM platform initialization code, Image download code, General purpose I/O code, Debug serial port cord, Kernel startup code.

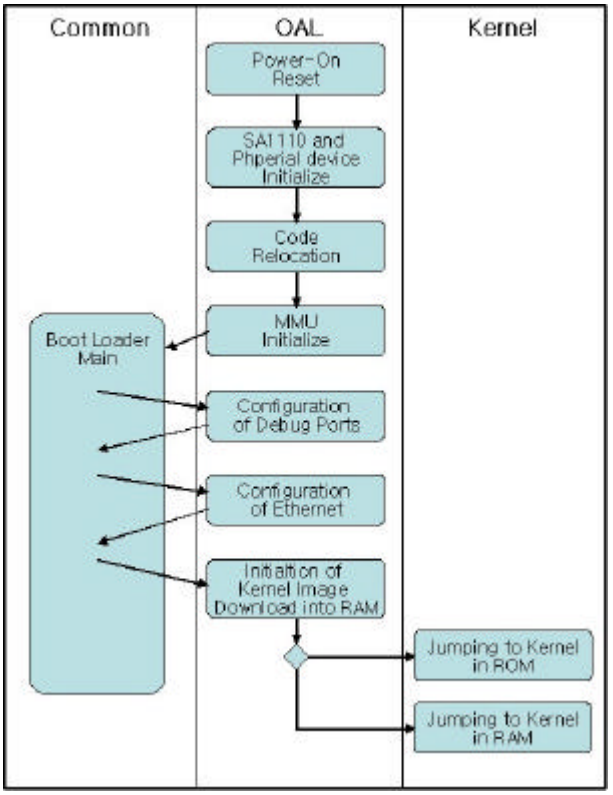

Fig. 5 Activity diagram of the CE boot process

Beginning with code relocation, the boot loader often copies itself from one location in memory to another that provides better access times, perhaps copying from flash memory to RAM. Next, we configure the Memory Management Unit (MMU) for virtual to physical address mapping. The next step requires that we initialize all ports used on this platform for debugging. We configure the Ethernet for downloading the kernel. Once the download is complete, the starting address is validated and the code jumps into the kernel. The precise address to which the code jumps depends on how the kernel was downloaded. The address of the entry point must correspond to the address used in the RAMIMAGE entry in the boot.bib file, after the ROMOFFSE -T value is added.
The kernel is loaded at the memory by the boot loader. Contents to be loaded at the memory expressed at the Fig. 6.

4.2 OAL

An OEM adaptation layer (OAL) is the layer between the Windows CE kernel and the hardware of your target device. OAL is a collection of functions that may be accessed by the

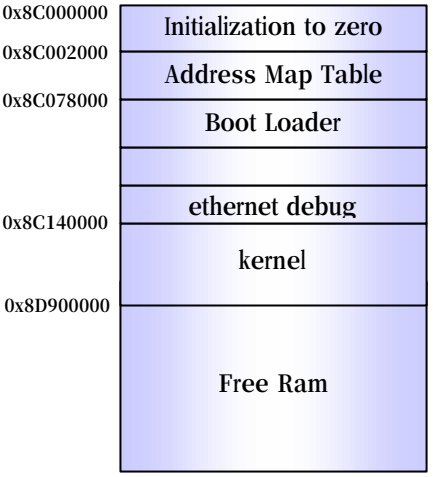

Fig. 6 Memory Map of the SDRAM is loaded by boot loader

CE operating system to gain access to platform-specific features. The CE kernel calls these functions to obtain access to the capabilities of the platform. OAL forms the connection between the Windows CE kernel and the hardware.

The boot loader is built as a stand-alone image and the OAL functions are included in the kernel image. OAL is the first code to run in the CE kernel. The Fig. 3 shows the OAL architecture.

The OAL development process is divided into three phases.
Phase 1. Developing a minimal set of OAL functions required to successfully start the CE kernel
Phase 2. Expanding on the functionality of the first phase by adding remote debugging support and interrupt service routines (ISRs)
Phase 3. Adding features to provide module certification, power management, and a persistent registry

OAL isolates the hardware with the kernel. It configures the hardware that is not initialized in the boot loader step. After the stage of OAL, hardware in the platform is configured.

5. CONCLUSION

In this paper, we have described the procedures of porting Windows CE to Intel StrongArm SA1110 platform, which includes building a boot loader, and implementing the OAL Layer. As we can see, the modular design of Window CE leads to a systematic approach to port it to an embedded system. Through this process, we demonstrated it's very efficient to port Windows CE to any customized target platform.

Inaddition, once the OS kernel is up and running, there are various existing application available, such as Internet browsing, word processing, multimedia and etc. When necessary, we can also develop specific application using WIN32 API, whose interfaces are quite familiar to most developers from its general OS counterpart.

REFERENCES

[1] James Y. Wilson and Aspi Havevala, "Building Powerful Platforms with Windows CE", Addison Wesley Publishers, s, 2001.

[2] Microsoft WinCE, "Microsoft Windows CE Platform Builder Library", Microsoft Inc.

[3] Abraham Silberschatz, Operating System Concept, 6th edition, John Siley & Sons, Inc., July 2001.

[4] William B. Giles, "Assembly Language Programming", Macmillan Publishing Inc., 1991

[5] Hua Harry Li, Ph.d., Yu Raine Wang, Vincent Chju, Qingling Ning, and Tong Zhang , " Porting Window CE Operating System to Broadband Enabled STB Devices"

[6] Douglas Boling, " Programming Windows CE 2nd Edition ", Microsoft Press.

[7] Bill Gallas, Vandana Verma, " Embedded Pentium Processor System Design for Windows CE", 1998 IEEE