PORTING OF µC/OS-II AND DEVELOPMENT OF USB MASS STORAGE DEVICE DRIVER FOR EMBEDDED DATA ACQUISITION

D.S.Vignesh prabhu[1], A.Ramya[2]

[1]*M.Tech, Embedded System Technology, SRM University, (India)*
[2]*Assistant professor, Department of Electronics And Communication, SRM University,(India)*

ABSTRACT

The purpose of the project is porting of µC/OS-II and development of USBmass storage device driver for embedded data acquisition. This driver is developed in Real Time Operating System environment so the data can be transferred between the mass storage device to and from the microcontroller. The core of the microcontroller is cortex M3, it has deterministic behaviour and also support co-processor so the any application need of multiprocessor operation is easy. The core also implement memory protection unit. This driver ensures the re-entrancy and thread safety. The advantage of this driver is, it is implemented in Real Time Operating environment which supports multiple tasksand processes so the data can be transferred to any output device like LCD and GSM module by the support of hardware.

Keywords—*ARM Cortex m3, Device Driver, File System, Real Time OS, µC/OS-II.*

II. INTRODUCTION

An operating system is generally a computer program that supports a computer's basic functions, and provides services to other programs (or application) that run on the computer [2]. The services provided by the operating system make writing the applications faster, simpler, and more maintainable. RTOS is the Real Time Operating System intended for real time application and multitasking. So the data from application can be read from and write to mass storage. RTOS keep monitoring the input consistently and in timely manner. Most operating systems appear to allow multiple programs to execute at the same time. This is called multi-tasking. In reality, each processor core can only be running a single thread of execution at any given point in time. A part of the operating system called the scheduler is responsible for deciding which program to run when, and provides the illusion of simultaneous execution by rapidly switching between each program. An RTOS that can usually or generally meet a deadline is a soft real-time OS, but if it can meet a deadline deterministically it is a hard real-time OS. There are list of RTOS available like Micrium µc/os-ii, Micrium µc/os-iii, Free RTOS and Mipos.µC/OS-II is ported in the microcontroller LPC1768.In the present system SD card is interfaced with microcontroller to store the data from developed application so the data can be transferred between the microcontroller and SD card.In SD card there is no need of File System to access the data. So the data protection

will be low and accessing rate of data also low. In proposed system USB mass storage is developed and data can be stored in this data accessing rate is high and large amount of data also stored.

II. PORTING OF µC/OS-II

µC/OS-II is the low cost priority based pre-emptive real time multitasking operating system for microprocessor mainly written in c programming language. It is highly portable, ROMable, scalable, pre-emptive real time and it support up to 250 task. File systems are generally used to organise the files in mass storage device. FAT file system is commonly used and it has FAT16 and FAT32 depends on mass storage.

Porting is generally downloading kernel to target board. Before porting any RTOS to the board, we need to find the source code of the kernel and make sure if it suits the memory requirement of the board. µC/OS-II is ported in LPC1768 for developing mass storage driver, data from developed application perform write operation through file system and the data can be transferred between the file system and USB firmware.

Microcontroller used here is LPC1768 and core is cortex M3 which supports pending of the interrupts and nesting of the interrupts by separate feature implemented in architecture called NVIC for the deferred processing[1].This feature is helpful for the deterministic behaviour which is the most important in the real time applications. Here the main concern is to develop mass storage device, the core architecture does not support virtual addressing so the access of memory for read and write operation will be fast. It has two operating modes thread mode and handler mode. Thread mode is entered on reset and can be entered as a result of exception return. Privileged and user code can run in thread mode. Handler mode is entered as a result of exception. All code is privileged in handler mode. It has 100 MHz ARM core with 64kb of SRAM and 512kb of Flash.The core supports the co-processor so that any application which may be in need of the multiprocessor operation is easy. The core also implements memory protection unit which is the safety feature of the kernel.

Fig.1. LPC1768

Adapting a real time kernel to microprocessor or microcontroller is called porting. Most of the µC/OS-II is written in c programming language for porting, however still necessary to write some processor specific code in

c and assembly language. Specifically µC/OS-II manipulates process register which can be done through assembly language. Porting µc/os to different processor is easy because µc/os was designed to be portable[3]. A processor can run the µC/OS-II if the processor satisfy the general requirements.

1. Processor has a c compiler that generates re-entrant code.
2. Processor supports interrupt and that provide an interrupt that occurs at regular interval.
3. Interrupt can be disabled and enabled from c.
4. The processor should support a hardware stack to store large amount of data.
5. The processor has a instruction to load and store the stack pointer and CPU register in memory or stack

Fig.2 shows the µC/OS-II architecture and relationship with hardware. When µC/OS-II issued in Application it is necessary to provide application software and µC/OS-II configuration sections. Depending on the processor the porting can consist of writing or changing 50 to 300 line of code.

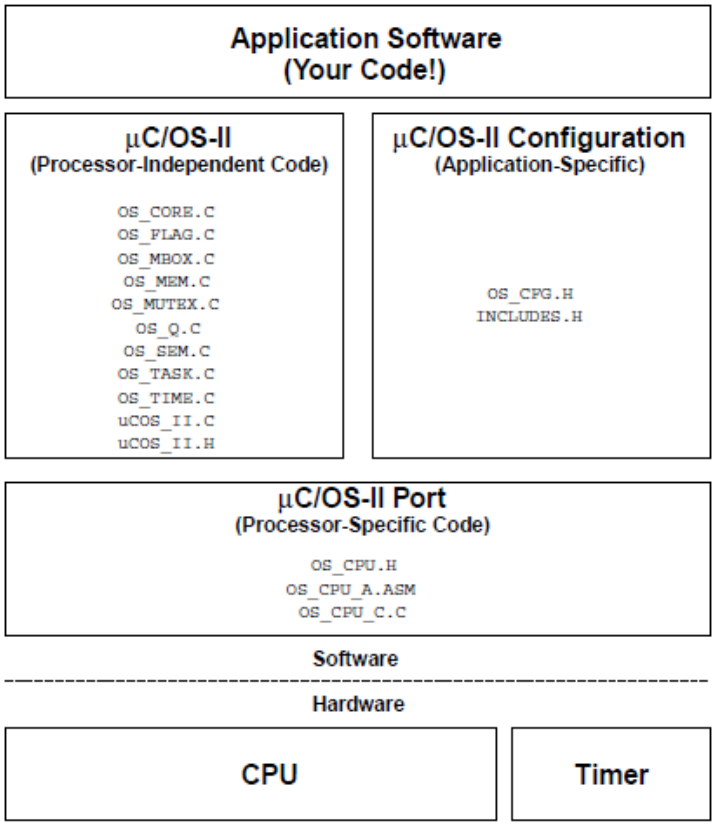

Fig.2. µC/OS-II Architecture And Relationship With Hardware

III. Device Driver

A device driver (commonly referred to as a driver) is a computer program that operates or controls a particular type of device that is attached to a computer. A driver provides a software interface to hardware devices, enabling operating systems and other computer programs to access hardware functions without needing to know precise details of the hardware being used. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

Writing a device driver requires an in-depth understanding of how the hardware and the software works for a given platform function. Because drivers require low-level access to hardware functions in order to operate, drivers typically operate in a highly privileged environment and can cause disaster if they get things wrong. In contrast, most user-level software on modern operating systems can be stopped without greatly affecting the rest of the system. Even drivers executing in user mode can crash a system if the device is erroneously programmed. These factors make it more difficult and dangerous to diagnose problems.The driver can be written by either software engineer or hardware engineer who works for hardware development companies. This is because they have better information than most outsiders about the design of their hardware. Moreover, it was traditionally considered in the hardware manufacturer's interest to guarantee that their clients can use their hardware in an optimum way. Typically, the *logical device driver* (LDD) is written by the operating system vendor, while the *physical device driver* (PDD) is implemented by the device vendor. But in recent years non-vendors have written numerous device drivers, mainly for use with free and open source operating systems. In such cases, it is important that the hardware manufacturer provides information on how the device communicates. Although this information can instead be learned by reverse engineering, this is much more difficult with hardware than it is with software.

3.1 Purpose

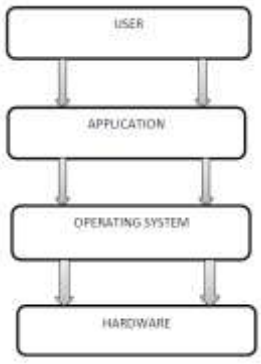

Fig.3.OPERATING SYSTEM

Device drivers simplify programming by acting as translator between a hardware device and the applications or operating systems that use it. Programmers can write the higher-level application code independently of whatever specific hardware the end-user is using. For example, a high-level application for interacting with a serial port may simply have two functions for "send data" and "receive data". At a lower level, a device driver implementing these functions would communicate to the particular serial port controller installed on a user's computer.

IV. USB HOST

USB device driver is the peripheral driver which transfers the data to and from the mass storage device. The data from the file system is transferred into the USB peripheral driver which creates the USB frame and transfers the user application's data to and from the external device.µC/USB Host is a real-time USB Host software stack designed for embedded systems equipped with a USB Host orOTG controller. It includes many class drivers (MSC, HID and CDC ACM). The stack requires a Kernel.

µC/USB Host uses a modular architecture with three software layers between the application and the hardware.

- The Class Driver layer provides class-specific services to the application. For example, the Mass Storage Class (MSC) Driver includes interface functions for reading and writing sectors from a storage device.
- The Host core layer enumerates the device, loads a matching class driver, and provides the mechanism for data transfers.
- The Host Interface Driver (HID) interfaces with the host controller hardware to enable data transfers and detect devices.

V. FILE SYSTEM

The data in any Mass Storage Device is accessed only through the File system following the peripheral driver. So the user application calls the file system which calls the peripheral driver for accessing data on the mass storage media. Here, in this project we are implementing FAT32 file system, which supports from 4 GB to 32 GB of memory space.

5.1 FAT 32

FAT32 file system allows for a default cluster size as small as 4 KB, and includes support for EIDE hard disk sizes larger than 2 gigabytes (GB).FAT32 uses space more efficiently. FAT32 uses smaller clusters (that is, 4-KB clusters for drives up to 8 GB in size), resulting in 10 to 15 percent more efficient use of disk space relative to large FAT or FAT16 drives. FAT32 can relocate the root folder and use the backup copy of the file allocation table instead of the default copy. In addition, the boot record on FAT32 drives is expanded to include a backup copy of critical data. FAT32 is more flexible and the root folder on a FAT32 drive is an ordinary cluster chain, so it can be located anywhere on the drive. The previous limitations on the number of root folder entries no

longer exist. In addition, file allocation table mirroring can be disabled, allowing a copy of the file allocation table other than the first one to be active. These features allow for dynamic resizing of FAT32 partitions.

VI. SOFTWARE

Keil is the windows operating system software that runs on a pc to develop application for ARM microcontroller. It is also called as single integrated environment or IDE because it provides a single integrated environment to develop code for embedded microcontroller. The version of the keil used here is the MDK version5 and it is the latest release of our complete software development environment for a wide range of ARM, Cortex-M, and Cortex-R based microcontroller devices. MDK includes the µvision IDE/Debugger; ARM C/C++ Compiler, and essential middleware components.

VII. METHODOLOGY

The proposed RTOS based mass storage device driver consists of microcontroller, keypad, LCD and mass storage. µC/OS-II is ported on the microcontroller to develop the mass storage device driver. OS is started by initializing the hardware and software. The hardware core is cortex m3 and OS is µC/OS-II. The resource is allocated for the task defined in application. The scheduler is started to process which task to run first in priority manner.

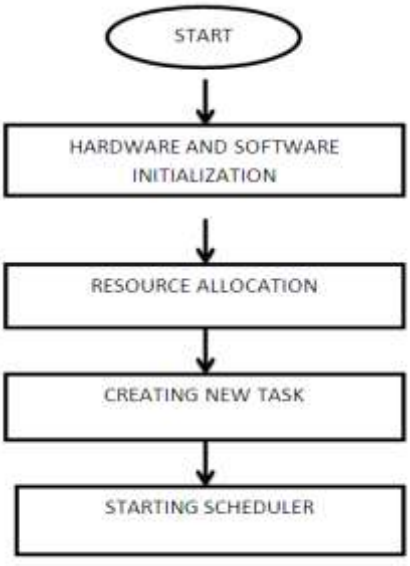

Fig.4. Operating system initialization

VIII. RESULT

Microcontroller based USB mass storage device driver has been developed. Input is taken from the developed application or sensor. Here 4x3 matrix keypad is interfaced with microcontroller to give the input and 16x2

LCD is used, it generally provides user with relevant information. So the input data given by the keypad has been stored in the USB mass storage device driver and the data will be displayed in LCD. The code is simulated by keil MDK version5. The code to run the each peripheral is done by separate module. Each of these modules are tested separately for proper functioning and integrated into single code file.

IX. CONCLUSION AND FUTURE WORK

In this paper porting of µC/OS-II to ARM cortex m3 is presented. It mainly focuses on the developing of USB mass storage device driver. The steps involved in porting of RTOS and final implementation details are provided. The OS used here is the RTOS so the data from developed application can be transferred to USB firm ware and vice versa. The core of the microcontroller provides co-processor which is very important in real time application and also provides memory protection unit here the important concern is for data acquisition which will be very useful for that and the future work is to develop the device driver to support different file system such as NTFS, FAT, ReFS.

REFERENCE

[1] Wright Nick, Judd Bob. Using *"USB as a data acquisition interface".* Journal of Evaluation Engineering, Vol. 43, No. 6, pp. 20-26, 2004.

[2] Richard Barry. Quality RTOS & Embedded Software [online].Available:www.freeRTOS.org.

[3] Jean J.Labrosse. Microc/os-II The Real Time Kernel, Second Edition. Beijing University Of Aeronautics And Astronautics.

[4] NXP, 11 February 2009, LPC1768 Objective Data sheet.

[5] Atmel Corporation. AT91 ARM Cortex-M3 based Microcontrollers: SAM3U Specifications, 2009.

[6] Liu Zhongyuan, Cui Lili, Ding Hong "Design of Monitors Based on ARM7 and Micro c/os-II", College of Computer and Information, Shanghai Second polytechnic University, Shanghai, China, IEEE 2010.