GLOBAL JOURNAL OF ENGINEERING, DESIGN & TECHNOLOGY
(Published By: Global Institute for Research & Education)
www.gifre.org

GPS Real-Time Location Based Control Drink- Driver & Patient Health Monitoring Analysis

V.R.Pra kash, P.Kumaraguru & P.Pandiraj
Assistant Professor, Hindustan University, Chennai, India

Abstract

Adaptive Cruise Control (ACC) is an electronic system that allows the vehicle to slow while approaching another vehicle and accelerate again to the preset speed when traffic is cleared. It also warns the driver and/or applies brake support if there is a high risk of a collision.

The project aim is to design a **GPS** equipped ACC system that (apart from performing normal ACC functions) slows down the vehicle intelligently when it enters speed restricted zones such as schools and colleges. It is also capable of detecting the speed breakers ahead and controls the vehicle dynamically according to the speed limit set for that part of the road. The system also continuously monitors driver distraction and driver health condition and brings the vehicle under ACC control if the need arises.

There are a variety of ways in which drivers can get distracted while driving, for example looking sideways, talking over a mobile phone etc. Driver head movement indicates if he is distracted or not. Our system is capable of sensing this. Another major issue is drivers in city buses or cars who are aged above 40 are at a higher risk of heart attack or similar heart related problems. A heart attack for a city bus driver while driving is fatal not only to him but also for the passengers. Heart rate is a vital symptom for identifying this condition. Our system senses the heart rate of the driver.

In real-world scenario this system should need to perform the operation within some timing deadline and must be extremely responsive or the result is fatal. Hence the system utilizes the services of a **RTOS (Real-Time Operating System).**

GPS aided ACC with Driver Status Monitoring can be implemented in all types of vehicles where safety will be given first priority and has the potential to become a standard part of any future vehicle.

Keywords - Real time monitoring; GPS& MCU.

I. Introduction

In general, an operating system (OS) is responsible for managing the hardware resources of a computer and hosting applications that execute on the computer. A RTOS is a specialized type of operating system designed to execute applications with very precise timing and a high degree of reliability. They are intended for use with real time applications. Such applications include embedded systems (such as programmable thermostats and household appliance controllers), industrial robots, spacecrafts, industrial controllers, scientific research equipments, etc. RTOS can be divided into two categories, hard real-time and soft real-time systems. In a hard real-time or immediate real-time system, the completion of an operation after its deadline is considered useless, and this may cause a critical failure of the complete system and can lead to an accident (e.g. Engine Control Unit of a car, Computer Numeric Control machines). A soft real-time system on the other hand can tolerate such lateness, and may respond with decreased.

II Overview of the System

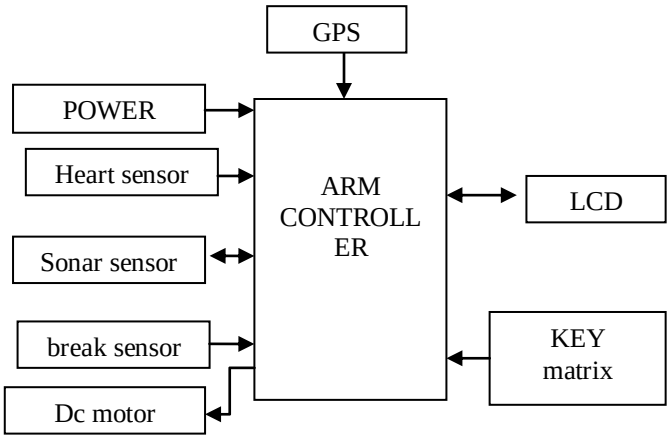
Service quality (e.g., omitting frames while displaying video, mobile phone applications).
A RTOS always contain multitasking, also known as multithreading. Multitasking is a technique used for processor time allocation [1]. Applications are divided into logical pieces commonly called threads and a kernel (core of the operating system) that coordinates their execution. Some threads of an application have greater importance or priority than others.

The high priority threads must meet their deadlines otherwise the system may lead to a complete failure or a deadly accident. A thread is an executing instance of an application and its context is the contents of the processor registers and program counter at any point of time. A scheduler, a part of the Real Time Operating System's kernel, schedules threads execution based upon their priority. There are a variety of efficient and fair scheduler algorithms available. Context switching function can be described in slightly more detail as the Kernel performs different activities with regard to threads on the CPU as follows: So, the context switch is needed every time the kernel suspends execution of a current running thread and resumes execution of some other high or same priority thread. These context switches occur as a result of threads voluntarily relinquishing their allocated execution time or as a result of the scheduler making the context switch when a process has used up its allocated time slice.

III. Hardware Implementation

To prove the concept and measure the performance of our suggested approach, MIPS processor architecture was selected [6] and the suggested approach was implemented on top of it. To be able to save the context, the register bank module of the processor was modified by adding register files in the register bank to save context. The size of each register file is equal to context size. The number of register files depends upon the number of threads in the system or number of threads that need real fast context switch. For the proof of concept, we have implemented 4 register files. Figure 1 shows the block diagram of the modified MIPS processor architecture.

IV. System Block Diagram



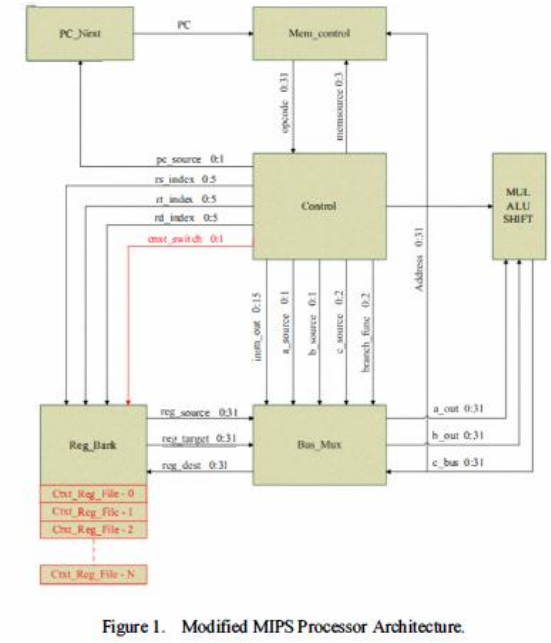


Figure 1. Modified MIPS Processor Architecture.

Figure 2: System Block diagram

V Software Implementation

A. Software Platform

Traditional embedded software systems are designed using single task with foreground / background style. Each task in the system has the same priority. Manual scheduling is the main way to occupy CPU and other resources. This strategy is clearly powerless to a complex system [6]. So we adopt µC/OS-ll here, which is a kind of free and open source embedded operating system. It has high efficiency, takes small space, and has excellent real-time performance and scalability. The kernel of µC/OS-Ⅱ has functions of task scheduling and management, time management, synchronization and communication between tasks, memory management and interruption service. So far, µC/OS-ll has run on more than 40 microprocessors with different architecture [7]. So it is suitable for the software platform of this control system.

B. porting of µC/OS-ll

Portability is a kind of performance representing that a real-time kernel can run on other microprocessors or microcontrollers. A processor can run µC/OS-ll if it meets the following general requirements:

1) There must be a C compiler for the processor and the C compiler must be able to generate reentrant code

.2) It must be able to disable and enable interrupts from C.

3) It must support interrupts and need to provide an interrupt that occurs at regular intervals.

4) It must support a hardware stack, and the processor must be able to store a fair amount of data on the stack.

5) It must have instructions to load and store the stack pointer and other CPU registers either on the stack or in memory[8].

Microcontroller LPC2214 meets all above–mentioned requirements, so µC/OS-Ⅱ can run on LPC2214. The main work of porting µC/OS-Ⅱ is programming. We choose ADS1.2 integrated development environment provided by ARM Company as software development kit. The code

contains three files as follows:
1) OS_CPU.H: This file mainly contains some definitions of date types, constants and macros related to processor. Because some data types in C such as "short", "int" and "long" are related to the word length of processor and complier, so they are non-portable. In order to ensure the portability, we should define these data types in this file.
2) OS_CPU_A.ASM: This file contains the operations on register, and must be programmed using assembly language. There are four functions related to processor need to be modified here, such as "OSStartHighRdy" function to start the highest priority task ready-to-run, "OSCtxSw" function for a task level context switch, "OSIntCtxSw" function to perform a context switch from an interrupt service routine and "OSTickISR" function to provide a periodic time source to keep track of time delays and timeouts.
3) OS_CPU_C.C: This file contains ten C functions to implement some basic operations. These functions are"OSTaskStkInit","OSTaskDelHook","OSTaskStatHook","OSTCBInitHook","OSTaskSwHook","OSTaskIdleHook", "OSTaskCreateHook","OSInitHookBegin","OSInitHook ,End" and "OSTimeTickHook".

C. System Tasks

Having ported μC/OS-Ⅱ to ARM microprocessor LPC2214, the next major work is task assignment according to the functional requirements of the system. The function of the system is collaboratively accomplished by multiple subtasks, each subtask taking charge of some part of the work. The scheme of task assignment will directly affect the efficiency and performance of the software [9]. The whole task of this system is mainly divided as follows:

1) Temp_Test_Task: Medium oil temperature test task has the priority of 19. In this task, the temperature of coolant is measured at first; then the sensor resistance collection task is woken up by the transmitted semaphore. The task is executed iteratively through time delay.

2) RS232_Com_Task: Serial port communication task has the priority of 21. This task is woken up by the semaphore transmitted by keyboard task. When it needs to communicate with upper computer, the corresponding key is pressed; then the communication task will be woken up by the semaphore transmitted by keyboard task.

D. Program Flow

The application program of the system begins at "main" function. The central program segmentation of "main" function is as follows:

```
 int main ( void )

  {
TargetInit ( );

OSInit ( );

OSTaskCreate ( R_Test, ( void * ) 0, (OS_STK *) &
                R_Test [ R_Test_STK_SIZE-1], 20 );

OSStart ( );

Return 0;
}
```

"TargetInit" function is used to initialize the system. Its work mainly contains defining relevant parameters and variables, setting all kinds of interrupts, and initializing devices. "OSInit" function is used to initialize μC/OS-Ⅱ. "OSTaskCreate" function is used to create different functional tasks, then to create the semaphore for communication between tasks. "OSStart" function is used to activate multi-task scheduling [10].

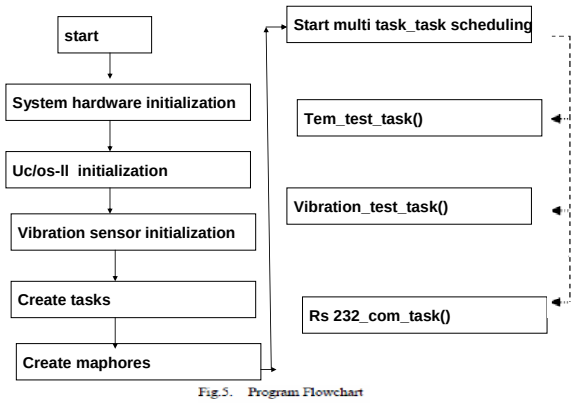


Fig.5. Program Flowchart

VI. Conclusion

This paper presents a new approach as an attempt to improve the hard RTOS system performance which helps in meeting the deadlines in an efficient way. It boosts the system performance for such systems where lot of context switching is required in a small period of time.

This approach can be easily extended to soft R TOS and regular operating systems to improve the overall system Performance. In these systems, threads are created at run time and it is difficult to know the number of threads at the design time. Therefore, threads that are part of frequent context switching could be set for fast context switching using internal register files using the specially designed scheduler algorithm. This paper also takes another step forward.

VII. References

[I] http://www.rtos.com/PDFs/ AnalyzingReal-TimeSystemBehavior.pdf
[2] Francis M. David, Jeffery C. Carlyle, Roy H. Campbell "Context Switch Overheads for Linux on ARM Platforms" San Diego, California Article No. : 3 Year of Publication: 2007 ISBN:978-I-59593-751-3
[3] Zhaohui Wu, Hong Li, Zhigang Gao, Jie Sun, Jiang Li " An Improved Method of Task Context Switching in OSEK Operating System" Advanced Information Networking and Applications, 2006. AlNA 2006. Publication date: 18-20 April 2006 ISSN : 1550-445X
[4] Jeffrey S. Snyder, David B. Whalley, Theodore P. Baker "Fast Context Switches: Compiler and Architectural support for Preemptive Scheduling" Microprocessors and Microsystems, pp. 35-42, 1995. Available:citesser.ist.psu.edul33707.html
[5] Xiangrong Zhou, Peter Petrov "Rapid and low-cost context-switch through embedded processor customization for real-time and control applications" DAC 2006, July 24- 28 San Francisco, CA
[6] http://www.opencores.org/projecLplasma
[7] MIPS Assembly Language Programmer's Guide, ASM - 0 I-DOC, Part Number 02-0036-005 October, 1992
[8] http://ftp. gnu.org/gnuibinutilsl
[9] Xilinx Corp, "Spartan 3E Starter Kit board user Guide" March 9, 2012
[10] http://model.com/contentlmodelsim-pe-simulation-and-debug