논문 2013-08-13

An Implementation of Automotive Parking Assistance System using Qplus-Auto OSEK Edition

Jeongho Son*, Jong Hyo Kim, Soo Young Ha, Kee-Koo Kwon

Abstract : Traditional implementation schemes for automotive electronic control units look simple, but the tradition schemes need so many coarse works to satisfy the user requirements regarding time constraints whenever their microprocessors are changed. Recently, a movement toward using middle-wares, such as OSEK operating system, has risen in automotive industry. In this paper, we describe how to use the features of operating systems to replace traditional firmware based softwares in points of views of services, such as multitask support, preemption, and realtime property. To show an example, we implemented a parking assistance system as a prototype.

Keywords : Preemptive Multitasking Operating System, Parking Assistance System, Ultra Sonic Sensor, Qplus-Auto OSEK Edition.

Ⅰ. INTRODUCTION

Many vehicles include a number of electronic control units (ECUs) which are used for safety and coziness of human. The ECUs use a variety of sensors to improve their performance and safety and then they can help and assist drivers with information about the status of the vehicle. A Parking assistance system (PAS) is a kind of these devices which can help drivers find an obstacle and park exactly in parking lines. Currently, it is widely spread in the world. The PAS was developed by Toyota in 2003. However, most of the parking assist systems were implemented using firmware development methodologies, which have simple control flows. In the systems, an interrupt service is only a way to change a control flow. The methodologies of firmware development mostly focused on functionalities in a low level viewpoint on a processor and peripherals without any considerations of reusability and extensions. Therefore, the size of code is growing up to be more complicated as the number of user requirements is increased. Recently, many automotive companies and organizations have striven to develop standard solutions to reduce the time and the cost for product development. OSEK (Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug - English: Open Systems and the Corresponding Interfaces for Automotive Electronics) operating system (OS) is an result from the hard works. In this paper, we show a prototype using multiple tasks, preemptive scheduler, events and alarms using an OSEK compliant OS and The prototype is used to show how to mitigate the drawbacks of the traditional programming method using features of the OSEK compliant OS.

The rest of this paper is organized as follows. Section II explains related works.

* Corresponding Author (phdson@etri.re.kr)
Received: 12 Feb. 2013, Accepted: 05 Mar. 2013.
J. Son, K.K. Kwon: ETRI
J.H. Kim, S.Y. Ha: Ajin Industrial Company

ISSN : 1975-5066

Section III describes an experience (PAS) and the ways to take advantages of an OSEK compliant OS. In Section IV, we discuss about our implementation results. Finally, we conclude in Section V.

II. RELATED WORKS

OSEK/VDX is a joint project to make an industry standard for an open-ended architecture for distributed control units in vehicles[1]. OSEK OS is a uniform environment which supports the automotive industry with standardized interfaces for task management, synchronization, interrupt management, alarms, intra processor message handling, and error treatment. The OSEK OS specification also defines four conformance classes[1, 4] to apply the OS from low end devices to high end devices widely; BCC1, BCC2, ECC1, and ECC2.

As an example in [2], we have to assure that the ECU is working well under all of conditions because safety is the most important in vehicles. To ensure that, Jan Krakora et al. and Wang Lei et al. studied about model checking tool for the distributed environments[5, 6]. In addition, the OSs have some drawbacks of the delay time for context switching, interrupt loss and big footprint [3]. Although the OSs have challenges, we use it for software reusability, application programming interfaces (APIs), OS objects (tasks, alarms, event, scheduler etc.) and to take advantages for fine granular modularity. In the next section, we present how to use the advantages of the OS when we implement an PAS.

III. IMPLEMENTATION: PAS

Most of the PASs use several ultrasonic sensors to estimate a distance from sensors to a nearest obstacle. As shown in Fig. 1, the PAS consists of four ultrasonic sensors and a micro-controller unit (MCU) that can control

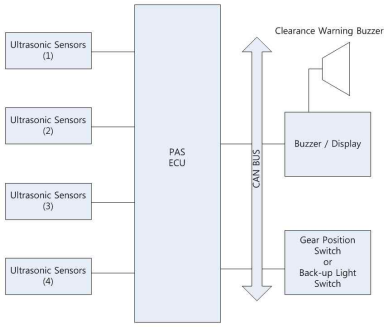
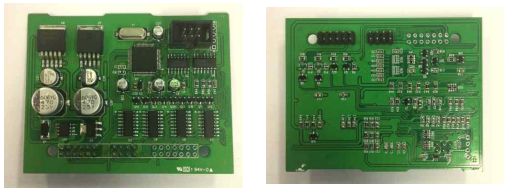


Fig. 1 Parking assistance system conceptual model

the ultrasonic sensor modules and supports communication for sending sensing data or warning messages to the other ECUs. Nowadays, a Controller Area Network (CAN) is widely used for communication among ECUs in vehicles. The PAS also use the CAN to get a status (gear position and speed) of the vehicle and send warning messages. Whenever the gear position of vehicle is in the reverse state, the PAS should gather data from the four ultrasonic sensors and calculate the distance from gathered data. If the distance indicates that an obstacle is within certain range, the PAS notifies the information to drivers with a beep sound using a built-in car audio system.

1. Basic User Requirements

First of all, we define user requirements and functionalities for the PAS. The information of vehicle status such as a gear position and speed is collected in every 100ms. Whenever a driver shifts into reverse gear and the speed is under 20 km/h, the ultrasonic sensors should work. The sensor activation interval for each ultrasonic sensor is limited to 25 msec to prevent interferences from reflections caused by other sensors. The accuracy of measured data should guarantee within 2 cm. In addition, the PAS should warn a driver with a beep sound when an obstacle exists within 2m. The frequency of beep is varied according to the distance,



(a) front side (b) back side

Fig. 2 Hardware prototype for our implementation

2. Hardware Implementation

We developed a hardware prototype that consists of a Freescale MC9S12XEG128 16 bit MCU working at 64MHz PLL clock. We also use CAN transceiver, Input Capture Unit (ICU), General Purpose Timer (GPT), (Serial Programming Interface) SPI, and Digital Input Output (DIO) in the MCU. The ECU has interfaces for 4 ultrasonic sensor modules for short range object detection. The hardware prototype can be connected with other ECUs through a CAN bus in the vehicle. Fig. 2 shows the front and the back side images of our prototype.

3. Firmware Based Implementation

As shown in Fig. 3. An while statement is used to control the ultrasonic modules. The four ultrasonic sensors work when a driver shifts into reverse gear under the speed of 20Km/h. Because the sensor module that we use needs to get a signal for specific duration to activate itself, we connect a DIO port to the sensor modules to control the voltage level. To keep high for a specific time, we should disallow any interrupt as presented in bold in Fig. 3. The duration time is related directly to a wave emission power. If we do not keep the specific duration time, it cause an error when we estimate the distance between the ultrasonic sensors to an obstacle. Upon reflections come back, they are captured by the input capture units (ICU) and then we can calculate the distance from a sensor to a nearest obstacle using round trip time.

```
InitMcu();
while (TRUE) {
    Sensor_Diagnostic();
    System_Check(Speed, Gear);
    if(Reverse Gear == TRUE){
      /* Get distance value from ultrasonic sensors */
      for ( i = 0 ; i < 4;i++){   /*< spend 25msec */
            Disable_All_Interrupt();
            Burst_Start(i); /*< Send burst signal */
            Enable_All_Interrupt();
            Delay(25ms); /*< To avoid interferences */
            Distance_data[i] =     Get_Distance(i);
      }
    }
    /* Deliver sensing data over CAN bus */
    SendDistanceData();
}
```

Fig. 3 A pseudo code for PAS without OS

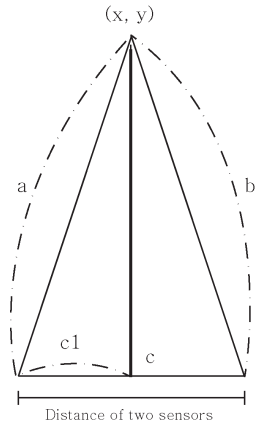


Fig. 4 Distance calculation

Where ultrasound is propagated at 331.5 + (0.60714 * Temperature) [m/s] and the distance is calculated by ((echo arrival time – echo departure time)/2)[us]. According to the Fig. 4, the coordinate (x, y) is calculated using area (S) and distance (c1) as follows.

$$s = \frac{(a+b+c)}{2} \tag{1}$$

$$y = \sqrt[3]{s(s-a)(s-b)(s-c)/c} \tag{2}$$

$$c1 = \sqrt{(the\ shortest\ distance)^2 - y^2} \tag{3}$$

$$x = (sensor\ position\ of\ the\ shortest\ distance) - c1 \tag{4}$$

In most cases, the firmware code structure is very intrinsic and it has small footprint, and simple control flows. However, only a interrupt is a way to change the control flow.

Moreover, large portion of the code should be changed when a micro controller processor is altered so that the source code might ruin consistency of the system architecture regarding the service interfaces for upper layer. In addition, the simple control flow in firmware based implementation makes delay of state acquisition because the state acquisition is done after a loop cycle of a while block if a state transition occurs by interrupts as shown Fig. 3. An unexpected hardware interrupt with a higher priority also causes delay. For example, the gear position cannot recognized immediately when the control entered into the loop statement block. In other words, we cannot predict precisely the system's activities because of the delay caused by loops.

Until now, we briefly listed up strengths and weaknesses of the firmware based development. Now we adopt an operating system in our product to migrate the weaknesses.

3. Operation System Based Implementation

As mentioned above, most difficulty is to satisfy time constraints. Qplus-Auto OSEK Edition is used as an OSEK compliant OS for the PAS application programming, which is compliant with OSEK OS V2.2.3 standard. The PAS consists of three tasks (OperationTask, SensingTask, and OutputTask) and Category 1 Interrupt Service Routines (Cat1ISRs) for ICU, GPT, and CAN Transceiver for the PAS.

A. Interrupt Service Routines (ISRs)
   - Category 1 Interrupt Service: Used for a system timer, CAN transceiver, DIO control timer and ICU channel interrupts.

B. Tasks
   - SensingTask: gets distance data from four ultrasonic sensors. This task is activated periodically in every 100ms.
   - OperationTask: used to check vehicle status, such as gear position, speed and sensor diagnosis.

```
//Task Name: OperationTask

 /* This task is activated by an Alarm periodically */
 if(ReadVehicleState() == 'R' …)
 {
     SensorStartStatus = START;
     ActivateTask(SensingTask);
 }else
 {
     SensorStartStatus = STOP;
 }
 TerminateTask();   // or WaitEvent is possible.
```

```
//Task Name: SensingTask

/* After Initialization */
While(SensorStartStatus == START)
{
    //ENTER_CRITICALSECTION()
    /* Burst signal using an Alarm */
    BurstStart(i);
    //EXIT_CRITICALSECTION();
    SetRelAlarm(alarm_25ms, 0, 25);
    WaitEvent(WAIT_25MS);
    ClearEvent(WAIT_25MS);
    DistanceData[i] = GetDistance(i);
    i++;
    i %=4;
}
TerminateTask();
```

Fig. 5 A pseudo code for the PAS using Qplus-Auto OSEK Edition

   - OutputTask: used to send distance information using CAN communication bus to another ECU that can make sound to notify a driver of the distance from the vehicle to the nearest object.

The priority assignment for the tasks is very important. In this case, we arrange that the priority of OperationTask is higher than that of 'SensingTask'. Therefore, a time constraint for getting the state of gear will be satisfied. We present pseudo codes of two of the tasks in Fig. 5.

A hardware timer is used for alarm services (as an Alarm object in the OS). The Alarms can supply tasks with a precision activation time, which will be a fundamental function to satisfy user requirements about time constraints. We also use 'WaitEvent', 'SetEvent' and 'ClearEvent' API functions to support with accurate waiting time (25msec) to prevent interference from the other ultrasonic sensors. These Event APIs of OSEK OS can be used instead of busy waiting of firmware based

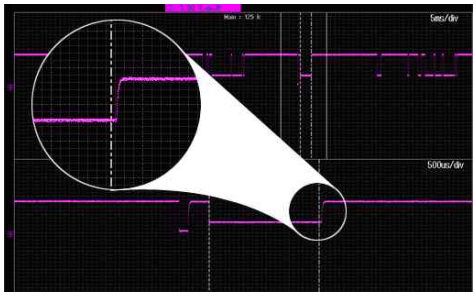
implementation. The Event mechanism is very useful for state transition management because It can reflect the state of machine immediately in urgent situations. For this reason, 'SensingTask' is configured as an extended task.

The procedure to get sensing data from ICU channels is as follows; Initially, SensingTask makes a signal using a DIO pin for an ultrasonic sensor module to generate a wave form. For this procedure, the DIO pin keeps high level for a specific time as mentioned above. And then, the ultrasonic sensor module emits internally a wave form by itself. ICU Cat1ISR is used to capture the reflections so that we can to reduce the processing time in the interrupt service routine.

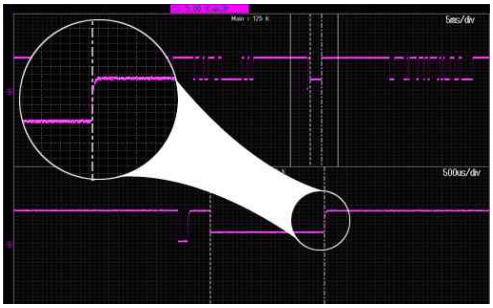
IV. DISCUSSIONS

As shown in Fig. 6(a), we show a difference between specified burst time and working time at approximate 1.2ms. Since our sensor module makes a difference about 1.17cm per 100us when a temperature is 20 celsius degree, it makes an error for distance measurements. Although we compensate the difference, we have to adjust for all other MCUs according to their characteristics. Fig. 6(b) shows that the Qplus-Auto OSEK Edition based PAS is more applicable for time constraint applications without any considerations about interrupt interferences. We show how difficult to fulfill user's requirements about time constraints in firmware based development. Our implementation architecture provides accuracy for time constrained of user requirements and utilize the advantages of preemptive multitasking operating system. In addition, we can archive an application with a variety of forms.

In our experience, OperatingTask and SensingTask are separated to reduce state checking delay which is caused by loop statement of ultrasonic sensor in the firmware example code.



(a) Burst signal form without OS



(b) Burst signal form with OS.

Fig. 6 Results of two implementations

V. CONCLUSIONS

In this paper, we described weaknesses of firmware based implementation such as losses of interrupts by disabling all interrupts and delay caused by loop statements in state transition systems. In addition, we show an example to change the tradition program into an OSEK OS based program. As mentioned earlier, OS based implementation can give us so many advantages to fulfill user requirements about time constraints as well as reusability of program code. Obviously, OS can help developers implement rapidly application program with its APIs and keep a program architecture constantly with standardized functions.

REFERENCES

[1] The OSEK/VDX Group. OSEK/VDX Operating System. Version 2.2.3. http://www.osek-vdx.org. 2012-06-29. [on-line].

[2] C. Xiao-xia, Y. Wen-ju, Z. Kong-xin, "Research and Application Based on OSEK/VDX OS," Proceedings on Conference of Computer, Mechatronics, Control and Electronic Engineering, Vol. 5, pp.179-181, 2010.

[3] Z. Wu, H. Li, Z. Gao, J. Sun, J. Li, "An Improved Method of Task Context Switching in OSEK Operating System," Advanced Information Networking and Applications, Vol. 1, pp.217-222, 2006.

[4] G. Yang, W. Zhaohui, Y. Long, "AlphaOS, An Automotive RTOS Based on OSEK/VDX: Design and Test," Proceedings on IEEE International Conference of Networking, Sensing and Control, pp.174-179, 2005.

[5] J. Krakora, L. Waszniowski, P. Pisa, Z. Hanzalek, "Timed Automata Approach to Real Time Distributed System Verification," Proceedings on IEEE International Conference of Factory Communication Systems, pp.407-410, 2004.

[6] W. Lei, W. Zhaohui, Z. Mingde, "Worst-Case Response Time Analysis for OSEK/VDX Compliant Real-Time Distributed Control System." Proceedings on Conference of Computer Software and Applications, Vol. 1, pp.148-153, 2004.

저 자 소 개

Jeongho Son

2002: M.E. in information and communication from Kyungpook National University.

2007: Ph.D. degree in computer engineering from Kyungpook National University.

2009~current: senior researcher in Electronics and Telecommunications Research Institute (ETRI), Korea.

Research interests: communication protocols.

Email: phdson@etri.re.kr

Jong Hyo Kim

2009: B.S. degree in control and measurement engineering from Kyungil University.

2009 ~ current: senior staff in Ajin industrial company, Korea.

Research Interests: vehicle electrical control.

Email: jhkim@wamc.co.kr

Soo Young Ha

1997: B.S. in electronic engineering from Kumoh National Institute of Technology.
1999: M.S. in electronic engineering from Kyungpook National University.
2004 ~ current: senior researcher in Ajin industrial company, Korea.
Research Interests: electrical control.
Email: hsy0623@wamc.co.kr

Kee-Koo Kwon

2000: M.S degree in electronic engineering from Kyungpook National University.
2004: Ph.D. degree in electronic engineering from Kyungpook National University.
2004 ~ current: senior researcher in Electronics and Telecommunications Research Institute (ETRI), Korea.
Research Interests: embedded systems, automotive electronic units and signal processing.
Email: kwonkk@etri.re.kr