TELKOMNIKA, Vol. 11, No. 7, July 2013, pp. 3897 ~ 3904
e-ISSN: 2087-278X

Research on the Implementation of Porting uC/OS-II on the STM32F103 chip

Zhenghua Xin*, Yongliang Guo, Liangyi Hu
School of Information and Engineering, Suzhou University, Telp: +8618955711392
*Corresponding author, e-mail: begin0000@qq.com

Abstract

This work shows how to port µCOS-II kernel and to build the ARM cortex-m3 hardware platform. This program researches how to drive the basic external devices. This paper gives a detailed discussion that the driver programs of the underlying hardware is how to be expanded into the uC/OS_II to process the tasks from the platform. The applications can make the STM32 processor work normally by calling the functions of transplanted uC / OS_II. The system is an actual embedded real-time multi-task operating system. The speed of processing multi-tasks is faster than conventional micro-controller's. Because this system can run the highest priority task without return to the while loop. At last, this paper gives the structure of the main framework, the code of the tasks.

Key words: *embedded real-time, operating systems, real-time operating systems (RTOS), porting, uC/OS-*II

1. Introduction

The uC / OS-II is a program which is embedded in the object code and is equivalent to the user's main program. After the reset, the system executes the program first. The users' programs are built based on the RTOS [1] [2]. Moreover, the uC/OS-II uses the micro-kernel structure. It concludes the required functionality (such as process management, task communication, memory management, interrupt handling, process scheduling) on the kernel. The functionalities work when the system call the standard API which is in accordance with the micro-kernel structure [3] [4] 5] [6]. According to the priority of each task, the CPU can allocate time between them reasonably. The process of request for service establishes a customer to waiter relationship with the service delivery process through the inter-process communication mechanism.

2. The Hardware

The system uses the STM32F103 chip. It is the outstanding representative of the ARM Cortex-M3 processor which has the best new 32 bit ARMv7 Architecture supporting the Thumb-2 instruction set. Its frequency is up to 72MHZ. And it supports single-cycle multiplication and hardware division. According to the different type, the size of the memory is ranging from 16K to 32K bytes. There are three working modes sleeping, shutdown and standby. The DMA controller has 7 channels. The STM32F103 chip supports the peripheral device, such as the timer, ADC, SPI, I2C and USART and so on. It has two DAC and their conversion time is as long as 1us. Its I/O ports is up to 80. All I/O ports can be mapped to 16 external interrupts. And almost all ports can tolerate 5V signal. Its structure is as follows. The system will port the operating systems uC/OS-II on the STM32F103 chip to make many tasks run.

3. The Algorithm of the uC/OS-II

The task's scheduling and priority processing of the uC/OS-II of the operating system is on the basis of the ready list, an important data structure. The list saves the tasks' running states and the priority information. Because the uC/OS-II allows the 64 tasks. This table indicates each task's running status using 64 bits. Due to the higher priority task can only be scheduled, the space with the '1' means the task is ready. ' 0 ' means the task is not ready. As

Received February 2, 2013; Revised April 22, 2013; Accepted May 8, 2013

shown in the Figure 2, every position of 8 byte arrays OSRdyTbl[] represents the priority information of each task. The higher the position is, the higher priority of the task is. In addition, 64 bits are divided into 8 groups. As long as there is a '1' in the group, the value of the group variable OSRdyGrp is set '1'. So the value of the OSRdyGrp shows which eight bits of the OSRdyTbl[] exists the ready task. By it, the uC/OS-II cut down the search time. Figure 2 shows the relationship between the ready list and the group variable OSRdyGrp.

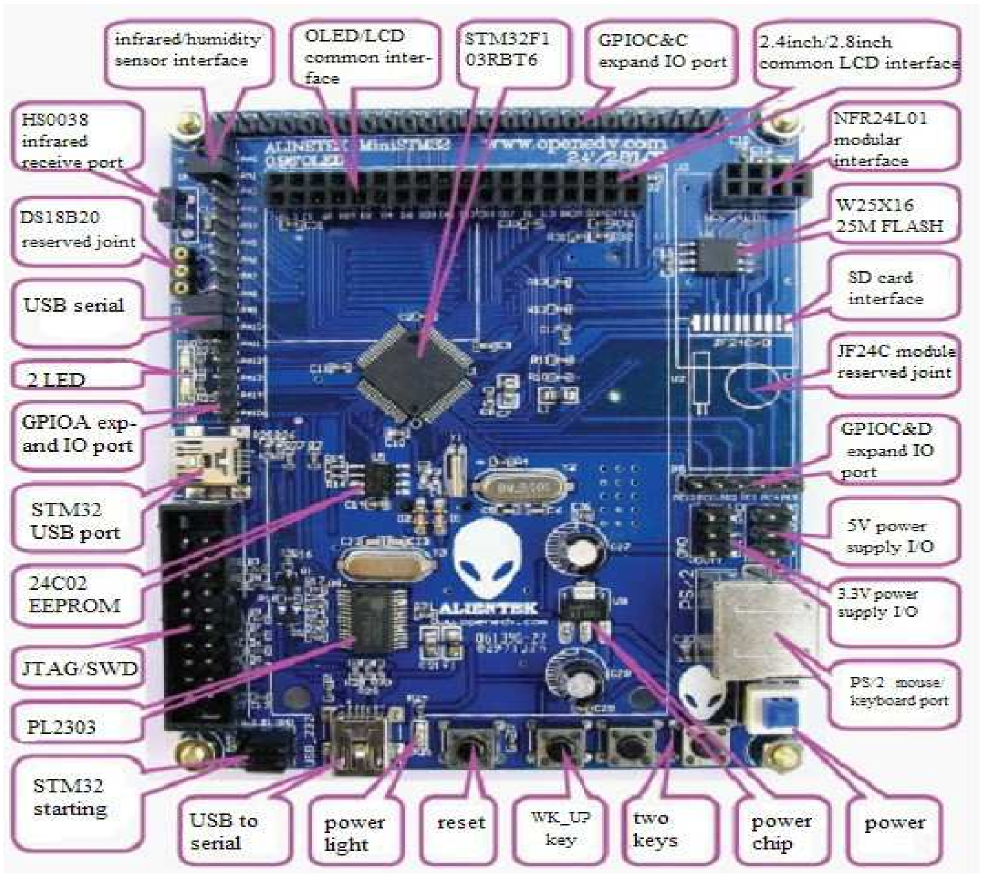

Figure 1. The structure of the STM32F103 chip

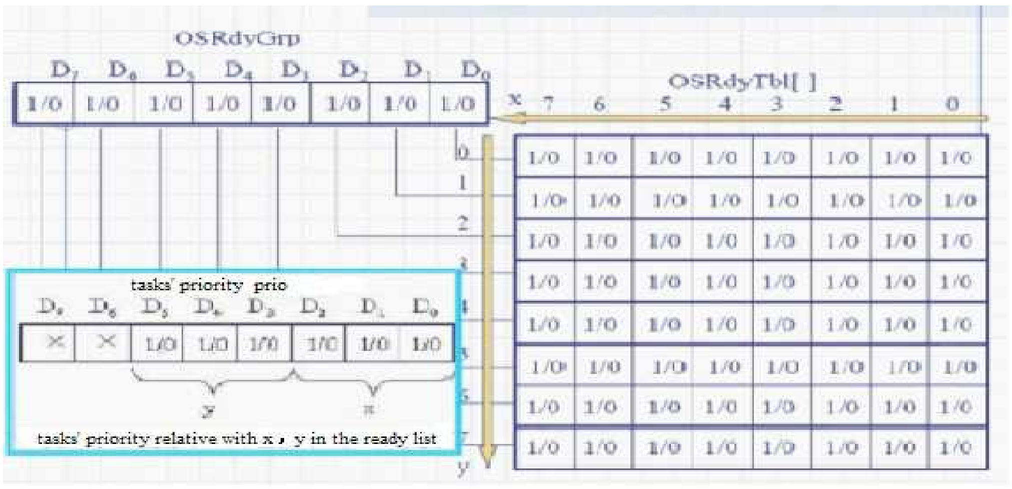

Figure 2. The uC/OS ready list

The follow sentences is to find the highest priority ready task algorithm.

```
Y=OSUnMapTbl[OSRdyGrp];
x=OSUnMapTbl[OSRdyTbl[y]];
OSPrioHighRdy=(INT8U)((y<<3)+x);
OSTCBHighRdy=OSTCBPrioTbl[OSPrioHighRdy];
```

4. The File Organization Of The UC / OS-II

The uC / OS-II includes about 3,500 lines of the core codes. The application tasks can be completed by them and the other auxiliary codes. The files are distributed as shown in Figure 3.

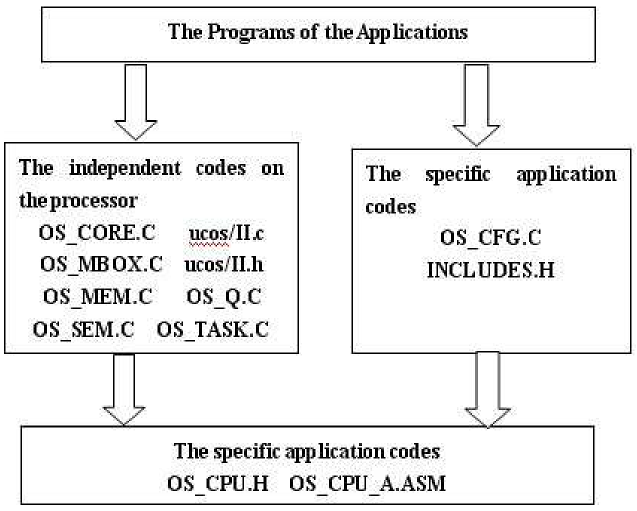
The Programs of the Applications
The independent codes on the processor
OS_CORE.C
ucos/II.c
OS_MBOX.C
ucos/II.h
OS_MEM.C
OS_Q.C
OS_SEM.C
OS_TASK.C
The specific application codes
OS_CFG.C
INCLUDES.H
The specific application codes
OS_CPU.H
OS_CPU_A.ASM

Figure 3. The architecture of the uC / OS-II file organization

5. The Porting And Application Of UC/OS-II

5.1. The Processor's Conditions for Porting the uC /OS-II

To make the uC / OS-II work on the ARM Cortex-M3 normally [1] [2], the processor must meet the following requirements. First, the C compiler of the processor can produce reentrant code. Reentrant code refers to a piece of the program, for example, a function. It can be called by many tasks at the same time without destroying data. In other words, the execution of reentrant functions can be interrupted at any time and can continue to run over a period of time without affecting the data because of the interruption called by other tasks. Second, the C program can open and close the interrupt service program [3] [4]. Third, the processor supports the interrupt and can produce a timer interrupt (usually between 10 to 100 Hz). The uC/OS-II generates the timer interrupt to achieve multi-tasks scheduling. Fourth, the processor can contain hardware stacks. When the uC/OS-II is scheduling some task, this stack will store its CPU registers. And then the CPU restores original registers of the another task from its stack. Therefore, the another task continues to run. Fifth, the processor supports the stack pointer and other instructions which reads information from the CPU registers and writes messages to the stack.

5.2. The Analysis and Work for the uC /OS-II Portion

The hardware layer and software layer construct the whole embedded system. The layer of software is divided into four parts: real-time operating system kernel, the part associated with the processor, the part relevant of the application and the user's application system [5] [6]. The code or programming associated with the processor. This is the most crucial part of the portion. The kernel combines the application system and the hardware into a real-time system. To make the uC /OS-II apply to different hardware systems, an intermediate layer between the kernel and the hardware is needed. The developers complete the code of the layer. Of course, this part of the code is different for the different processor. This research shows how to real the layer successfully. This layer is consist of OS_CPU.H, OS_CPU_A. ASM, OS_CPU_C. C. The paper shows how to complete three files.

5.2.1. OS_CPU. H

The head file Includes constants, macros and type definitions. Specifically, it contains the definition of data type, the direction of stack growth, disable interrupts and open interrupts and system software interrupts. Different compiler will use a different length of bytes to represent the same data type. Because the currently running task's register will be saved in the stack when the task switches, the OS_STK data types should be equal to the length of the processor registers. In this system, the codes of the data type about the compiler has been redefined. And the system has set the correlative codes for the running of the processor.

5.2.2. OS_CPU_A. ASM

Completing this file is to operate registers in the processor. In this system, the assembly language is necessary. Here the OSStartHighRdy(), OSCtxSw (), OSIntCtxSw () and OSTicklSR () must be completed.The OSStartHighRdy () function gets the stack pointer sp of the highest priority task in the multi-tasks under the control of the TCB after starting the OSStart(). The cpu scene is recovered by sp. Then the process created by user is executed from the system until the task is blocked or the cpu is seized by other higher priority tasks. This function is only executed once to start the first and the highest priority task execution. Then multi-task scheduling and switching is achieved by the following function.

When the task is blocked, task-level context switching is active to request cpu scheduling. The task switching is in the non-exception mode different from the interrupt service program. The OSCtxSw() function saves the cpu scene of the current task to the stack of the task. Then it continues to work after getting the stack pointer of the highest priority task and restoring the cpu scene. This completes a task switch.

After waiting for the clock arrival signal of the high priority task in the ISR, the interrupt level task switching need not return to the interrupted task after the interrupt exit. But it directly dispatches the execution of a ready and high priority task. The OSIntCtxSw () is in response to the high-priority task as soon as possible to ensure the real-time performance of the system. Its principle is basically as the same as the task-level switching. But it is no longer to save the cpu scene of the interrupted task and adjusts to the stack pointer because of the nesting of the function.

The OSTickISR() is a periodic system clock tick interrupt service function. The higher the frequency, the heavier the system load, the cycle determines the kernel provides the minimum time interval to the application system. It is generally confined to the ms level. Of course, the user creates interrupt to resolve the more demanding task. The specific content of the function is to save registers, to call OSIntEnter(), call 0StimeTick (), to call OSIntExit (), to restore registers and to return from the interrupt.

5.2.3. OS_CPU_C. C

The uC / OS-II defines several important functions. But the most important one is OSTaskStkInit(). The others are to expanse the system kernel. When the users create tasks, the system calls OSTaskStkInit () to initialize the stack of the user tasks. And the stack of the established and ready task is consistent with the structure of the stack which saves the variables of the execution environment when the system enter the interrupt. At least ten functions have been completed in this part of the system written by the C program language.

5.3. The Code Related Application

Developers complete their own application system by defining two head files to customize the core services. They are OS_CFG. H and the INCLUDES. H. The file named OS_CFG. H can configure the kernel. For example, it can define the maximum number of tasks, customize the mail service, the task suspended feature and dynamically change the tasks' priorities. The other one is the aggregate of header files needed by the entire real-time system programs, including the header files of the kernel and user.

5.4. Real-Time System Application

The uC / OS_II is only a real-time task scheduling and communication kernel [7] [8] [9] [10]. There is a lack of support for peripherals and interfaces. So the functions of file systems, network protocols, graphical interface are carried out by our necessary expansion in order to build a real-time multi-tasks operating system. This work drives the external peripherals by building the corresponding API functions and a practical interface function. The task modules are designed according to specific application requirements. And the communication interfaces among the modules are determined. The real-time multitask operating system of the expanded uC/OS_II kernel is as follows.

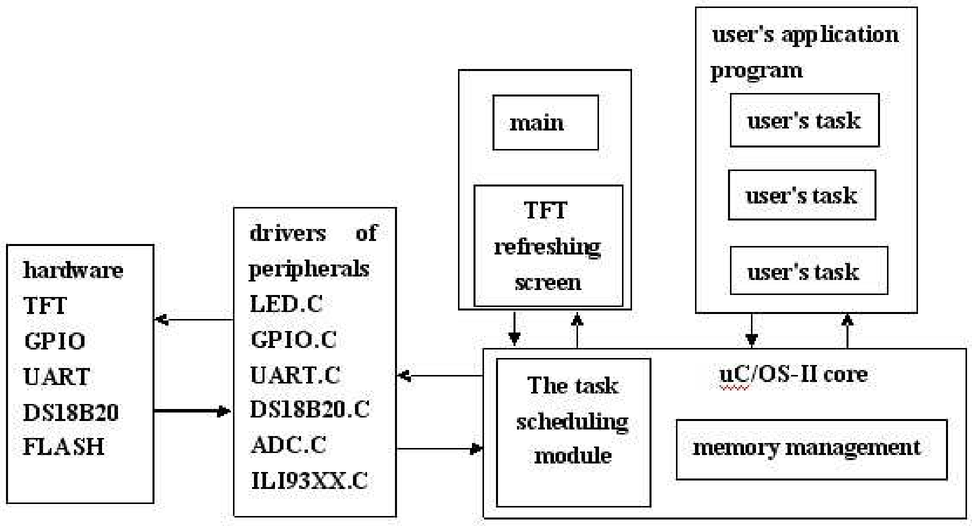

Figure 4. Software architecture block diagram

According to the tasks, the operating system is divided into the following modules. The peripheral devices is to ensure that the system can achieve the tasks assigned to the lowest level of components. The hardware includes LCD display, serial communication module, keyboard module, and general-purpose input and output interfaces GPIO. The LCD screen and the keyboard is expanded by the port. Driver programs separate API functions from the hardware. The RTOS management of UC / OS_II complete the multitask scheduling and synchronization. The user's system applications call the interface functions.

TFT LCD driver functions

void LCD_Init (void) // initialize lcd

void LCD_Clear (u16 Color) // clear screen function

void LCD_DrawLine (u16 x1, u16 y1, u16 x2, u16 y2) // draw a line

void LCD_DrawRectangle (u16 x1, u16 y1, u16 x2, u16 y2) // draw a rectangle

void LCD_ShowChar (u16 x, u16 y, u8 num, u8 size, u8 mode) // display a character in the specified location

void LCD_ShowNum (u16 x, u16 y, u32 num, u8 len, u8 size) // show figures

void LCD_ShowString (u16 x, u16 y, const u8 * p) // display string

void LCD_Chinese (u16 x, u16 y, u8 * china, u8 num, u16 color, u16 b_color) // Display Chinese in the specified location

The kernel and application software based on UC /OS_II Share the same address space. Therefore, after finishing booting, the system runs from the main() function developed by programmers. In the main() function, developers call the core function OSInit() to initialize the relative hardware and applications. Then calling a kernel function OSStart () to start the kernel process scheduling. The source code of UC / OS_II is open. There is a layer of HAL. Under this condition, the main () is a callback function. And the entrance of the whole system is set in the HAL.

In the conventional micro controller programming, the basic structure is an infinite loop. And all the applications tasks are in this super-cycle. In this system, the system has only one task to run. In this system, the ultra-loop structure mentioned is a basic structure in complex RTOS UC / OS_II. But It is no longer a system level, but only Is the task level. Creating and using the tasks are common . But their calling functions is different. Here is the basic structure of the UC/OS_II application. Each uC/OS_II application needs one task at least. It must be written in the form of an infinite loop. The task function can never return. So the task can only be an infinite loop or execute once then to be removed. The structure of this system is as followed.

The structure of the main function:

```
void main (void){ // Initialize the system hardware;
Stm32_Clock_Init (9); // set the system clock
LED_Init (); // initialize the LED
LCD_Init (); // initialize the LCD
Adc_Init (); // initialize the Adc
OSInit0;
// User tasks;
OSTaskCreate(mainTask,(Void *)0,
&stkMainTask[sizeof(stkMainTask)/4-1], 5);
OSTaskCreate(mainTask1,(Void *)0,
&stkMainTask1[sizeof(stkMainTask1)/4-1],2);
OSTaskCreate (mainTask2,(Void *)0,
&stkMainTask2[sizeof (stkMainTask2)/4-1], 4);
OSTaskCreate (mainTask3,(Void *)0, &stkMainTask3
[sizeof (stkMainTask3)/4 - 1], 3);
OSStart ();}
```

The structure of the Interrupt service program:

```
ISR{
Save the value of the processor registers;
Call OSIntEnter ();
Implemente the user's work;
Call OSIntExit0;
Restore the value of the processor registers;
RTI;}
```

This project increases OSIntEnter () and OSIntExit (). The two functions are on the basis of the original user's ISR. The example structure of a task is as follows.

```
void mainTask3 (void * pvData)
{Short the temp;
pvData = pvData;
while (DS18B20_Init ()) // initialize the DS18B20,
Chief detection 18B20
{LCD_ShowString (60,130, "DS18B20 Check Failed!");
OSTimeDly (OS_TICKS_PER_SEC / 2);
LCD_ShowString (60,130, "Please Check!");
OSTimeDly (OS_TICKS_PER_SEC / 2);}
LCD_ShowString (28,130, "DS18B20 Ready!");
POINT_COLOR = BLUE; // set the font to blue
LCD_ShowString (28,150, "ENV Temperate:. C");
while (1){temp = DS18B20_Get_Temp ();
if (temp <0){temp =-temp;
LCD_ShowChar(140,150, '-',
16,0); //show negative}
LCD_ShowNum(148,150, temp/10, 2,16);
// display the temperature value
LCD_ShowNum(172,150,
temp%10,1,16);// display the temperature value
OSTimeDly (OS_TICKS_PER_SEC / 2);}}
```

The above procedure is the basic framework and some of the procedures of application of this article. The program runs renderings shown in Figure 3.

Figure 5. The result of Porting Embedded RTOS uC/OS-II to ARM Cortex-M3

6. Conclusion

The most work of porting the uC / OS-II was mainly focused on transplanting the file named by the OS_CPU_A. ASM. The realization of the transplantation for this file is associated to the processor architecture and the principle of uC / OS-II transplantation.

In this file, the most difficult work is mainly epitomized on the realization of two functions. They are named by the OSIntCtxSw () and 0STickISR (). Because the transplantation of these two functions was related with the timer, the settings of the interrupt register according to the designer of the system. Of course, different designers have different ideas. In the actual procedures of the porting work, they are prone to produce errors. The most important role of the OSIntCtxSw () function is switching tasks in the interrupt ISR directly. Thus the real-time response speed is improved. The timing of switching tasks happens at the time of the execution to OSIntExit () in the the ISR. If a higher priority task is found to have implemented the conditions of waiting for the time tick arrival, then it can immediately be scheduled for execution without having to return to the interrupted task to switch tasks. In that case, it is not real-time.

To achieve the transplantation of the OSIntCtxSw functions is roughly categorized by two situations. First, it is through the adjustment for the stack pointer (sp). The compiler has its own way of processing the nested function. Depending on it, the accurate position of the sp needs to be saved through the calculation of the requisite adjustment. Then the scene saved by the chip before entering the interrupt can be reused. The benefit of this approach is to switch the task directly inside nested function. The priority of tasks are higher. They are much easier to be scheduled. The execution speed is faster. However, this approach needs more skills to set relevant compiler parameters according to specific compiler. This method is used to finish the system. The other one is setting switching flag bit. Then the tasks do not switch in OSIntCtxSw (). The nested functions' execution sequence is as follows.

OSIntExit()→OSENTERCRITICAL()→OSIntCtxSw()→OS_EXIT_CRITICAL()→OSIntExit(). After the exit, the system determines whether to switch the interrupt level task by taking out the flag. This approach does not need to set the parameters about the compiler and also does not have to do the calculation. But the real-time response of the system is much slower. To beginners, this method is much easier to understand. So it is also simple to realize some systems.

Acknowledgements

This paper are sponsored by the Education Department of Anhui Province Key Natural Science Research Foundation of China under Grant (KJ2010A352), Research Open Platform of

Suzhou University (Laboratory of Intelligent Information Processing, 2012YKF39; University Culture Research Center, 2012YKF24), University science research project of Anhui Province (KJ2011B180; KJ2012A263; KJ2012B185), Quality project of Anhui Province (20101077) and Professor (doctor) research starting foundation project (No.2012jb01).

References

[1] Wu, Huazhu, Wang, Zengcai. Research on laser system of airport perimeter based on ARM. *Journal of Convergence Information Technology*. 2012; 7(8): 140-148.
[2] Li, Gang, Shao, Chun-Hui. Research of automotive collision avoidance system based on ARM. *Journal of Convergence Information Technology*. 2012; 7(10): 255-264.
[3] Zhenghua Xin, Liangyi Hu, Hong Li. The application of infrared sensors integrating stepper motor based on C8051F120. Advanced Materials Research. 476-478: 2133-2136.
[4] Xin Zhenghua, Lu Hongmei, Hu Liangyi, Li Jianxin. Implementation of SPI and driver for CC2430 and C8051F120. 2nd International Conference on Consumer Electronics, Communications and Networks. CECNet 2012 – Proceedings. 2638-2641.
[5] Zhenghua Xin, Yi Zhao, XiaoXiao Shao, Hui Wang. The design for the intelligent agricultural system based on C/S structure and Internet of Things. *Journal of Suzhou University*. 2011; (11).
[6] Jiangping Wang, Zhenhong Fang, Cuiling Jiao. The building for the intelligent greenhouse based on Internet of Things. *Guangdong Agricultural Sciences*. 2011; (5).
[7] Xin Zhenghua, Lu Hongmei, Hu Liangyi, Zhou, Jing, The device design of driving cars astern based on the wireless sensor networks node. *Advanced Materials Research*. 2012; 542-543: 989-992.
[8] Yen-Lin Chen, Chuan-Yen Chiang, Wen-Yew Liang, Cheng-Hung Chuang. Embedded Vision-based Nighttime Driver Assistance System. *Journal of Convergence Information Technology*. 2011; 6(2): 283-292.
[9] Liu Ying, Zhu Yantao; Li Yurong; Ni Chao. The embedded information acquisition system of forest resource. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(7): 1843-1848.
[10] Yao Xiao Feng, Zhao Rui, Xu Hui Pu. City heating network dispatching and management information system. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(4): 851-857.