International Journal on Recent and Innovation Trends in Computing and Communication SSN: 2321-8169
Volume: 3 Issue: 4 2343 – 2348

Prototype Implementation of Real Time CAN Driver for Distributed Embedded Applications

Kuldipsing Rajput[1]
M.Tech in Embedded System Technologies
Vel Tech University, Chennai, India
& CDAC ACTS, Pune, India
rajputkuldipsing@gmail.com

Babu Krishnamurthy[2]
Senior Technical officer
R & D Dept. CDAC ACTS,
Pune, India
babu_krishnamurthy@yahoo.com

Sasikala G.[3]
Associate Professor
Vel Tech Technical University,
Chennai, India
gsasikala@veltechuniv.edu.in

Abstract- The purpose of this proposed prototype is to make it useful in various distributed embedded applications. This system is implemented by using FreeRTOS, LPC1769, CAN (Controller Area Network), LwIP (Light Weight Internet Protocol) Protocol and Sensors. Now a day's real time communication is one of the key features in distributed embedded systems. Using FreeRTOS, distributed embedded application performance will be improved as compared to general operating system. FreeRTOS comprises CAN & LwIP protocol stack which transfer real time data from one microcontroller node to another. The Sensor acts as input to this prototype and microcontroller node in compliance of FreeRTOS that comprises the Real Time CAN driver which is used for secured data transfer. Real Time LwIP protocol stack is implemented on top of FreeRTOS to feed the data to PC Host application. It consumes less RAM as compared to other communication protocols such as TCP, UDP, etc. for data transmission so that the memory bandwidth is reduced. In this system, based on LwIP stack the data will be transferred from microcontroller Node to the Host application. PC Host application is used for data monitoring and control. Data acquisition and control of distributed embedded applications will be improved with help of this designed prototype.

Keywords- *Distributed Embedded System, FreeRTOS, CAN, LwIP.*

*****

I. INTRODUCTION

Distributed embedded system defines that multiple systems are interconnected by a network to achieve some specific goal. It is composed of firmware, microcontroller nodes, devices, and networks. Various nodes are connected to each other through interconnects. Distributed embedded system deals with the parameters such as performance, throughput, minimal latencies, compatibility, etc. When compared with the centralized system the distributed system gives more reliable throughput and performance. However the FreeRTOS is being ported on several multicore architectures so that it provides task execution simultaneously on the available cores on basis of its own task scheduling policies [11].

In this prototype we describe a usage of FreeRTOS for the development of multiple microcontroller node communication. It is hard real time operating system. FreeRTOS is mostly used for embedded application development. It is modularly designed, with a portable layer written in C, and a port specific layer for each compiler-processor pair written mostly in assembly language.

In proposed prototype the communication protocols CAN & LwIP plays vital role for data transmission from distributed nodes. CAN protocol is widely used in distributed embedded application development. It provides a reliable, high speed, event-triggered communication, simple and low cost utilization. It is used in mostly Automotive Industries to interconnect many Electronic Control Units (ECU) that exchange information through the CAN bus. Real time CAN protocol will improve the responsiveness and throughput [1].

LwIP protocol is free lightweight TCP/IP stack which retains main function of TCP/IP with reduction of RAM size. It supports IPV4, IPV6, TCP, UDP, ICMP, IGMP, SNMP, ARP and PPP protocols thus, it provide wide range of networking application support [14]. At Network application layer it supports DHCP and HTTP protocols for PC Host application development.

The paper presents prototype implementation for various distributed embedded applications such as Parking Guidance System, Elevator Control System, and Industrial Control System. In this implementation sensors used as input to Real Time CAN microcontroller Node. The sensors would be the replaced as per the requirement of the application. The Microcontroller Node comprises of CORTEX M3 LPC1769 microcontroller in compliance of FreeRTOS. The data from the sensor is stored in the CAN protocol data frame. Real Time data is then transmitted from one Node to other through CAN bus. Real Time LwIP stack is implemented on the FreeRTOS environment to give reliable speed and higher throughput. This communication protocol used to transfer data from the Microcontroller Node to the PC Host application. It provides the data to PC Host application with real time constraints. PC Host application will then able to provide real time information on the display. Data monitoring and control of this application will be done at PC Host application level with the help of physical interface.

The paper is organized as follow: in section II, prototype design components which includes RTOS framework, CAN and LwIP protocol implementation and their work is discussed; in section III, real time distributed embedded applications and their implementation possibilities is discussed; in section IV, real time distributed parking guidance system application is described; section V, concluding remarks are given.

II. PROTOTYPE DESIGN COMPONENTS

A. FreeRTOS + IO Framework-

Most of the embedded application uses FreeRTOS which provides the core real time scheduling polices (Preemptive, Co-operative and Dynamic), inter-task communication, timing

and synchronization primitives. Additional functionality such as a command console interface, or networking stacks, can be then be included with add-on components.[11].

FreeRTOS Task xTaskCreate API is used to create Real Time Task at application layer. It has various parameters such as its own task function, name, task stack size which is generally allocated from heap, task priority, and the task parameter. In FreeRTOS task execution is done with the control of Scheduler thus, the task which is of higher priority will execute first and then lower priority task executes [9].

FreeRTOS+IO framework provides a Linux/POSIX like open(), read(), write(), ioctl() type interface peripheral driver libraries. This framework resides in between the Peripheral driver library and user application layer. It provides a single, common, interface to all supported peripherals. It provides interface to various peripherals such as UART, I2C, SPI, CAN, etc. Also various different data transfer modes are created for read and write the data. It is implemented as a set of Application Program Interface (API) functions written in C [11]. FreeRTOS+IO framework provides the interface between the CMSIS (Cortex Microcontroller Software Interface Standard), FreeRTOS Libraries and certain application specific libraries. It includes the board specific files to develop the board specific firmware code. Board Support Package (BSP) consisting available peripherals on the hardware and among them the required peripheral is invoked in the driver layer with the help of FreeRTOS_ioctl() function call. Configuration file provides the facility to add or remove the certain parameters. It provides various data transfer modes such as polled transfer mode, interrupt driven zero copy transfer mode, circular buffer transfer mode, etc. so that according to the application requirement the mode has been selected. This framework also takes care of the peripheral specific parameters such as data frame type, message id, data frame length, etc with FreeRTOS_ioctl() function call.

B. CAN Driver Implementation-

FreeRTOS+IO framework is used to implement Real Time CAN driver on the FreeRTOS environment. The framework provides the API where controller supported peripherals is defined. The proposed prototype is implemented on the LPC1769 CORTEX M3 microcontroller which has two CAN peripherals as CAN1 & CAN2. Both are registered in the BSP file. The framework will then able to register the Real Time CAN driver. Application layer includes the user level program which calls the CAN driver APIs to perform various data transfer operations through the Real Time CAN driver [1].

Before execution of Real Time CAN driver activities like open, read, write, ioctl framework calls the function taskENTER_CRITICAL() which is used to disable the external interrupts with the help of “basepri” register of CORTEX M3 microcontroller. After completion of driver activities the interrupts are enabled again with the function call taskEXIT_CRITICAL().

Fig. 1 shows the FreeRTOS+IO framework for Real Time CAN driver. Figure consists of various library file and the layer are interconnected to each other with the help of FreeRTOS generic APIs.

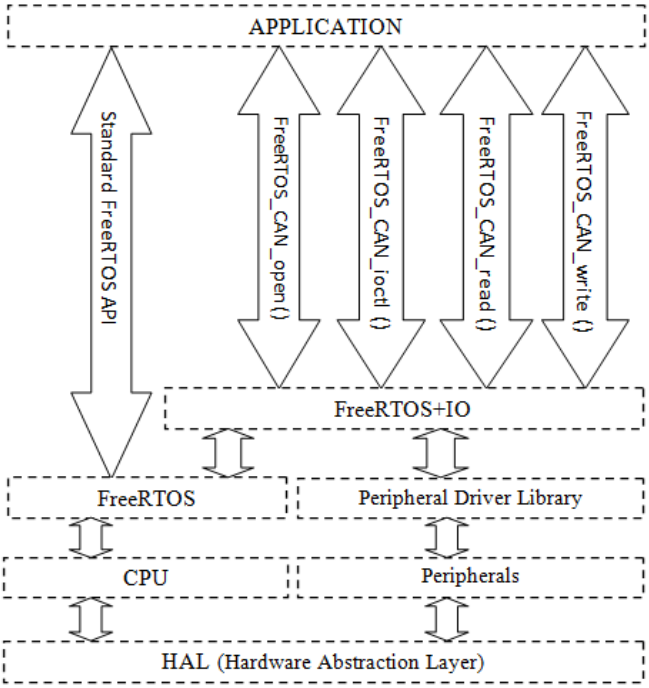

Fig.1 FreeRTOS+IO CAN driver Framework [11]

The functionality of generic system calls is as explained below-

1) *FreeRTOS_CAN_open()-*
   This API will take control over HAL (Hardware Abstraction Layer) where the CAN initialization and pin configuration is done at the given base address as per the requirement. Baud rate settings are done in the same layer. This protocol is specified in the BSP (Board Support Package). addition, peripheral-specific read(), write() and ioctl() functions are assigned to the peripheral control structure.
2) *FreeRTOS_CAN_ioctl()-*
   CAN is serial communication protocol provides the standard and extended data frame, the data in this protocol is transmitted in the form of message which is composed of following fields: Start-of-frame (SOF), arbitration, control, data, cyclic redundancy check (CRC), Acknowledgement (ACK) and end-of-frame (EOF). Thus, these fields are specified during the message transmission from one node to other. This API is designed in such a way that it provides user to select the parameters of data frame as per the requirement of the application.
3) *FreeRTOS_CAN_write()-*
   This API provide data buffer to send data from user layer to the Hardware layer. Here user defines the data at application layer and pointer to the data is then passed to this API as one of its parameter as pointer to the buffer. Then the as per the selection of CAN peripheral base address a specific CAN peripheral is selected among the present ones. Then after Hardware specific registers which are defined for the data transmission are accessed. The data from application layer is loaded in TFI (Transmit Frame Information Register), TID (Transmit Identifier), TDA (Transmit Data from byte 1-4); TDB (Transmit Data from byte 5-8) registers.

4) *FreeRTOS_CAN_read()*-
Priority based messages are transmitted through the CAN bus. The data from the bus is received with this function call. In this function call semaphore mechanism is called for synchronization. As taking and giving semaphores are atomic operations since interrupts are disabled and the scheduler is suspended in order to obtain a lock on the data buffer. The specified buffer is able to read data from the CAN register which is defined at the HAL. The data received from RFS (Receive Frame Status Register), RID (Receive Identifier), RDA (Receive Data Register-A), RDB (Receive Data Register-B).

C. LwIP Stack Implementation-

LwIP protocol is free lightweight TCP/IP stack developed by Adam Dunkels at the Swedish institute of computer science (SISC) [14]. It retains main function of TCP/IP with reduction of RAM size. It only needs several hundred bytes of RAM and about 40Kbyytes of ROM to run efficiently, which makes it very suitable protocol for distributed embedded applications [3]. LwIP uses a process model in which all protocols reside in a single process and are thus separated from the operating system kernel. LwIP does not provide any port to any microcontroller so we are implementing it on the CORTEX M3 LPC1769 controller.

LwIP consists following modules-

1) *Operating system emulation layer*- To make LwIP portable an operating system emulation layer provides a uniform interface to operating system services such as timers, process synchronization, and message passing mechanisms. Also it provides the timer functionality that is used by TCP [14].
2) *Buffer and memory management subsystems*- Buffer and memory management system is same as TCP/IP segment. In this stack implantation the "pbuf" is an internal representation of a packet. The "pbuf" has support of both dynamic and static memory allocation. It consists of three types as PBUF_RAM, PBUF_ROM and PBUF_POOL. These buffers are managed by "pbuf" subsystems [14].
3) *Network Interface*- Device drivers for physical network hardware are represented by a network interface structure. This structure consisting of various fields such as pointer to next node, Name, IP Address, Netmask, Gateway IP Address, etc [14].
4) *Application Program Interface*- For LwIP stack implementation LwIP API was designed. These API are similar to BSD (Berkeley Software Distribution) Socket API. LwIP API does not require any additional copy operation from the application program to TCP/IP stack. This would waste both processing time and memory. Thus the LwIP API allows the application program to manipulate data directly in the partitioned buffers in order to avoid the extra copy.

LwIP divides the task into two different processes as shown in the fig. 2. TCP/IP stack related API is communicating with Application process by Inter Process Communication mechanism such as Shared memory, Message Passing and Semaphore [14].

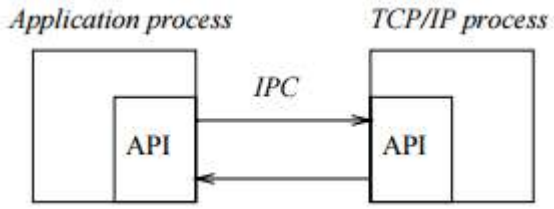

Fig.2 Division of API Implementation [14]

Fig No-02 shows the LwIP stack implementation is done on the LPC1769 microcontroller with compliance of FreeRTOS.

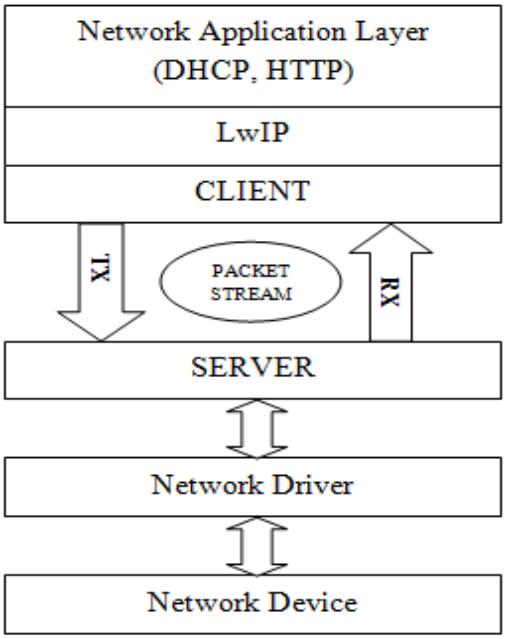

Fig. 3 LwIP Protocol stack layer

In the networking layer of LwIP protocol network devices and the respective drivers are interfaced through the system calls. Data is transmitted in the form of packets streams like TCP/IP packets. Network devices such as Ethernet hub, Switch, Routers are interfaced with various development boards with the help of network device drivers. The data from the network devices is fed to the Server. Server is the used to store data from various distributed nodes in the distributed embedded applications. The server is centralized for the specific distributed embedded application so that any node in the network can access it. The data packets are given to client as per its request. Network application layer contains the web based user application designed based on DHCP (Dynamic Host Configuration Protocol), HTTP (Hyper Text Transfer Protocol).

III. DISTRIBUTED EMBEDDED APPLICATIONS

Proposed prototype is designed for the development of distributed embedded applications. Some of these are as below-

A. Industrial Control System-

In the industries temperature control, air pressure control, smoke detection is most critical tasks to be controlled in the industrial environment. So the respective sensors are interfaced with the proposed designed prototype. Thus these sensors will give the parameter values to be controlled and the microcontroller node will then transmit information from node to PC Host application through LwIP stack for real time data monitoring and control [4].

B. Elevator Control System-

Elevator control system is distributed control system consists various sensors, switches, and peripherals like FAN, LED Display, Alarm, etc. Thus to get real time control over these peripherals and the most critical task that is door open and close operations can be easily handled by this real time distributed prototype. Only we need to change interconnects to the designed prototype and we will get the control over the elevator system. Also according to the application the respective changes made in the PC Host application [10].

C. Distributed system for Airport terminals-

At Airport Terminal the Flight information display board shows the real time information with the time factor. Also various lightening systems and LED indicators can be interfaced with the proposed prototype microcontroller node. Sensors at the airport terminal will gives the input in the form of the surrounding temperature, smoke, humidity and many other parameters. The sensor information is displayed on the PC host application which is fixed at control room. Also to display the flight status at display these Real Time communication protocol will give higher responsiveness and throughput.

D. Parking Guidance System-

Parking guidance system is the best real time distributed embedded application which can be developed by using this proposed prototype [10]. This proposed system is further described in next section.

IV. SYSTEM DESIGN IMPLEMENTATION

The proposed system will consists of two nodes Master and Slave respectively. Fig. 4 describes the prototype implementation of the parking guidance system. In this system design an Ultrasonic Sensor (HCSR-04) will be interfaced with the Slave node through the GPIO (General Purpose Input Output) inbuilt protocol of the microcontroller. Here the sensor is used to sense the presence of the vehicle in the parking lot. In this prototype CAR A, B, C, D are the parking lot. This prototype also provides the facility to allocate the parking lot for VIP (Very Important) person which is indicated in the red color at the CAR A place as shown in Fig 4.

The allocation of this parking lot is done by the parking management and control system. Slave Node A & B are connected to the Master Node via CAN Bus. In this prototype each Slave Node has three sensors to sense the car object. Sensor data will be send to the Master Node via CAN Bus through the Slave Node. Each Node in the system is in compliance of FreeRTOS environment.

Master node will then collect the data from the slave through the CAN bus. At the Master Node the CAN-LwIP Gateway is established. Then the data from CAN node to the Host PC application is transmitted through the LwIP stack. LwIP stack implementation is done at the Master Node. The Master Node will be then connected to the Host PC though the RJ45 Ethernet cable. As LwIP has support of TCP/IP standard protocol so it can transmit data from the Master to the Host PC [2].

At the Host side a Real time monitoring application will be designed with the HTML web development scripting language on the basis of HTTP or DHCP protocol [2, 4]. This real time distributed application will give information details of the parking area such as the allocated parking lot, free parking lot. This application also provides the control over the whole parking area with the help of control panel. Master Node is also interfaced with the Display unit for an indication of the available parking lot. The display unit will be situated at the parking entrance to guide the car driver. The display unit will show the real time information on it. Along with the sensor Led will be interfaced for an indication of allocated and free parking space.

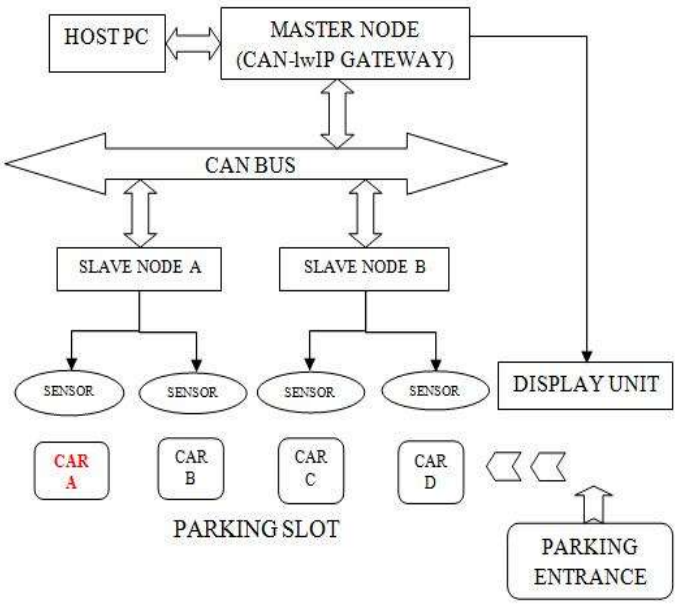

Fig. 4 System Implementation Prototype

Fig. 5 shows the programming flow chart of the proposed system design. The task implementation and its execution will be done based on the designed flow chart. The prototype firmware implemented by using the Real Time task for each protocol specific operations.

 SSN: 2321-8169
2343 – 2348

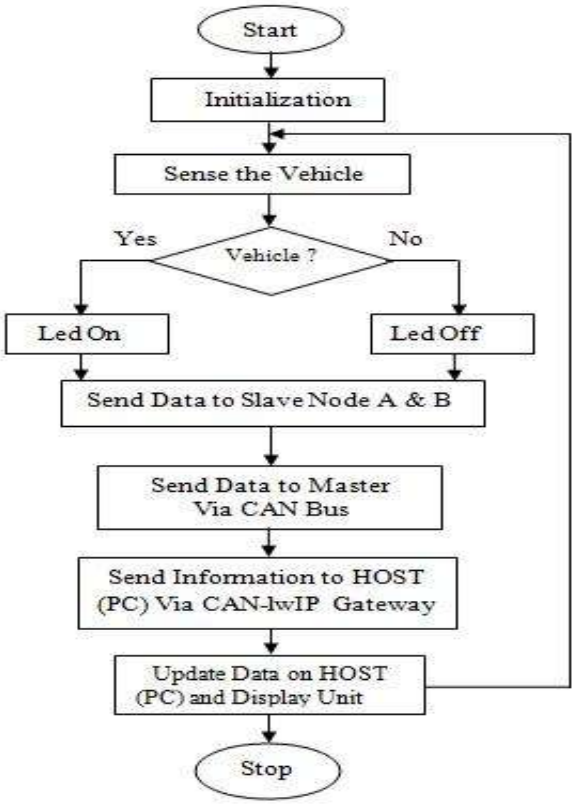

Fig. 5 Program Flow Chart

Three real time tasks are created to complete the prototype objective. These tasks are as below-

A. *GPIOSensorTask*-
This task configures the GPIO pins as input for sensor module interfacing and output for the sensor status indication. FIOPIN register is used to get the current status of the sensor interfaced port pin. The value of this register is then stored in the pointer variable and then passed to the FreeRTOS_Queue buffer. Then through the queue mechanism the GPIO port pin status send to CANWriteTask.

B. *CANWriteTask*-
In this Task the GPIO port pin status is received through the Queue receive buffer. This task specifies the CAN data Frame details such as Frame ID, Frame Length, and Data Frame Payload. Then with the help of FreeRTOS_CAN_ioctl() system call the respective registers are configured at HAL(Hardware Abstraction Layer) of the Framework. The FreeRTOS_CAN_wrtie() API is then used to send the data over CAN bus. At the Master node FreeRTOS_CAN_read() API is used to read the data from the CAN bus.

C. *LwIPServerTask*-
This Task is used to implement Real Time LwIP stack for data transmission from Master Node to the PC Host application. The data packets are same as the TCP/IP packets. IP address is assigned to the Base Board which acts as server or client and the PC Host acts as server. The LwIP API is responsible for data packet transmission. Thus, the Real Time CAN data is posted over Host application.

V. IMPLEMENTATION RESULTS

Proposed prototype is implemented on the Hardware base board LPCXpresso Base Board provided by *"Embedded Artists"* [13]. Also the main target stamp of ARM CORTEX M3 LPC1769 is provided along with the base board [13]. FreeRTOS porting become an easy task because of the extraordinary efforts of the LPCXpresso team [12]. The Proposed system implementation is based on the "*FreeRTOS+IO"* framework [11]. Hardware details are shown in below figure.

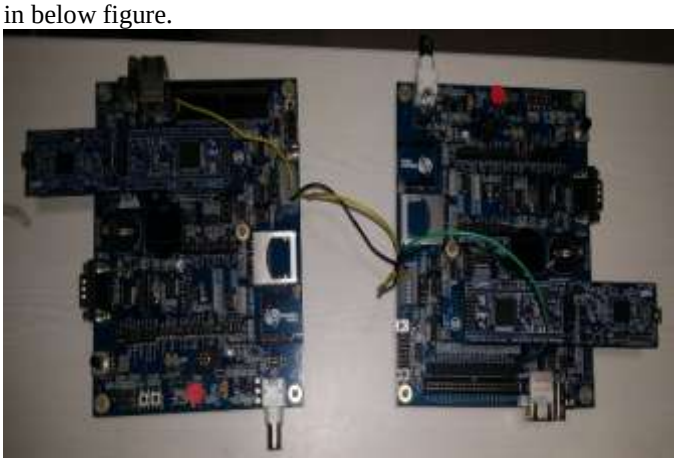

Fig.6 Master-Slave Communication

Figure 6 shows the communication between two nodes through CAN Bus. Slave node send the data through the TDA and TDB register of CAN protocol, at the Master side the data is received at RDA and RDB register. The console window is very useful feature to see the simulation results of the experiments on the LPCXpresso Kit. Thus the data in master and slave node can be seen through the console window as shown in below Fig. 7

```
the msg_length is=6
 Tx_buf1 used for TX
Data_frame
std_id_format
In hardware layer the CAN_msg Id is=0
 the data stored in TDA1 is 11111111
 the data stored in TDB1 is 1111
the message in write task is of length=6
The data written by write task is=10000298
In CAN_ReceiveMsg function in Hw_layer
one complete Message received by double rcv_buf
the received frame is data frame
 the data in rcv_buf data_reg_A is=11111111
 the data in rcv_buf data_reg_A is=11111111
 the data in rcv_buf data_reg_B is=1111
 the data in rcv_buf data_reg_B is=1111
the rcv_buf is released
```

Fig. 7 Output Console view

Here we have assigned the Frame length of 6 byte out of 8 byte so the TDA register has first 4 bytes and TDB register contains the remaining 2 bytes. At the Receiver side which is Master Node gets the same data in TDA and TDB registers. Thus the CAN protocol can send the real time data on FreeRTOS environment without any data frame loss.

Also the LwIP stack implementation and the communication between the Master Node and the Host PC Application implemented on the same hardware. The sample data packets are sent from the Master Node to the Host PC through the RJ45 Ethernet Cable. Below figure shows the packet transmission [2]. Here the LwIP protocol creates its own IP address and then it provide the IP address to the Host PC for communication [2, 3]. The "*pinging operation"* results are as shown in below figure.

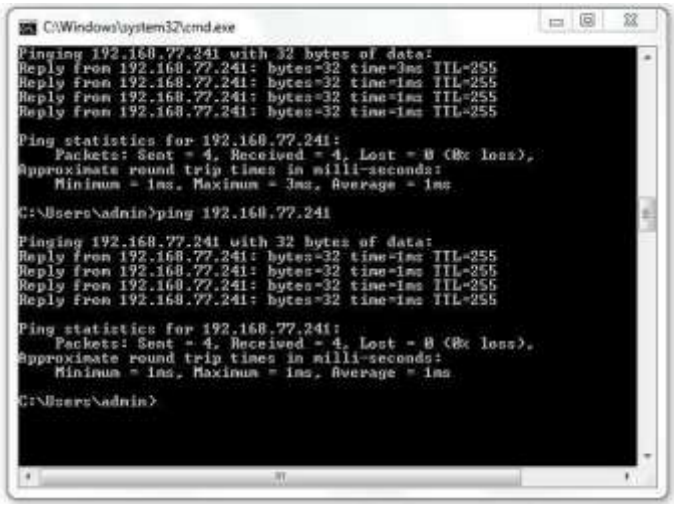

Fig. 8 Pinging Operation

VI. CONCLUSION & FUTURE WORK

This prototype has been designed and implemented to improve the performance of CAN and LwIP protocol based Distributed Embedded Applications. In comparison with other distributed embedded prototype, the proposed prototype is efficient and has higher throughput. The distributed embedded application involves parking guidance system which is designed on this prototype. This Prototype can also be mounted at various places such as Shopping Mall, Multiplex, and Function Hall. Due to lack of management in parking space driver finds difficulty to park the car and waste lot of time for car parking which results in the Traffic congestion, fuel Consumption, Air pollution etc. So to avoid these problems, this prototype is the best solution. This prototype provides flexibility in addition of sensor nodes in the existing prototype. The multitasking functionality of the system will be handled by the FreeRTOS API's which gives dedicated responsiveness.

Future work would provide security and parking billing system development. In this implementation, camera would be interfaced with the existing system which can capture the number plate of the car entering the parking lot and parking bill will be generated at the entrance. The security system will be monitored the whole parking lot with the help of interfaced cameras. The PC Host application can be designed more dynamically by using .Net and PHP framework.

ACKNOWLEDGEMENT

The author wishes to thanks to Prof. Rajesh Sola and Mrs. Vaishali Maheshkar for giving valuable suggestions and mentoring me in doing this work.

REFERENCES

[1] Florin Catalin Braescu, Lavinia Ferariu and Andrei Franciuc,"Monitoring CAN Performances in Distributed Embedded Systems", *IEEE System Theory, Control, and Computing (ICSTCC)*, 15th International Conference, 2011.

[2] Wei Chen, Shu-Bo-Qiu, Ying-Chun Zhang "The Porting and Implementation of Light-Weight TCP/IP for Embedded Web Sever", *IEEE Wireless Communication, Networking and Mobile Computing (WiCOM)*, 4TH International Conference, 2008.

[3] Zoican, S. Telecomm. Dept., Politeh. Univ. of Bucharest, Bucharest, Romania, "LwIP Stack Protocol for Embedded Sensor Network" *IEEE 9TH International Conference on Communications (COMM)*, 2012.

[4] BoQu & Daowei Fan, "Design of remote data monitoring and recording system Based on ARM*", IEEE 2nd International Conference on Industrial and Information Systems*, 2010.

[5] Yanfeng Geng, Christos G. Cassandras, "New Smart Parking System Based on Resource Allocation and Reservation" *IEEE transaction on Intelligent Transportation System* Vol no.3, Sept. 2013.

[6] Ran XueJun1,Wang Jianqun1,Li Zhenshan1,Yao Guozhong1 "Design of Parking Guidance System based on Embedded Internet Access Technology", *IEEE, Control and Decision Conference*, 2010.

[7] Sanaa Alfatihi, Soukaina Chihab and Yassine Salih Alj "Intelligent Parking System for Car Parking Guidance and Damage Notification", *IEEE 4TH International Conference on Intelligent Systems, Modeling and Simulation*, 2013.

[8] R. Bosch, GmbH, "CAN Specification 2.0," *Robert Bosch GmbH*, 1991.

[9] R. Barry, Using the FreeRTOS Real Time Kernel, 1st ed., 2010.

[10] "CAN In Automation (CiA)" CAN Newsletter [Online]. Available: http://www.can-newsletter.org/engineering/

[11] "FreeRTOS homepage." [Online]. Available: http://www.freertos.org/

[12] "LPCXPRESSO IDE (Integrated Development Tool)" [Online]. Available: http://www.lpcware.com/lpcxpresso.

[13] Embeddedartists.com, 'LPC1769 LPCXpresso Board | Embedded Artists AB', 2015. [Online]. Available: http://www.embeddedartists.com/products/lpcxpresso/lpc1769_xpr.php.

[14] Adam Dunkels "Design and Implementation of the LwIP TCP/IP Stack" Ph.D. Thesis, Swedish Institute of Computer Science (SICS), Feb-2000-2001.