International Journal of Engineering Research and General Science Volume 3, Issue 3, May-June, 2015
ISSN 2091-2730

Implementation of CANopen protocol for Industrial and Automotive Applications

Mayank M. Prajapati

VLSI & Embedded System Design

Gujarat Technological University Ahmedabad Gujarat, India

mayankprajapati74@gmail.com

Contact No. 09409163808

Abstract— High level protocol CANopen was developed by CiA (Can In Automation). For integration in embedded Systems there exist several CANopen protocol stacks. This paper proposes a setup of multiple CANopen nodes making use of the Canfestival framework in Linux environment.Implement the CANopen communication between 2 nodes using the CANopen Stacks and the application for at least one specific device profile. Implementing the CANopen communication on beagle board xM .Interfacing the application on the webpage.

Keywords— CANopen, Canfestival, CAN(Controlled area network), SDO(service data object), PDO(process data object), NMT(network management), Synchronization object (SYNC) , Emergency object (EMCY)

INTRODUCTION

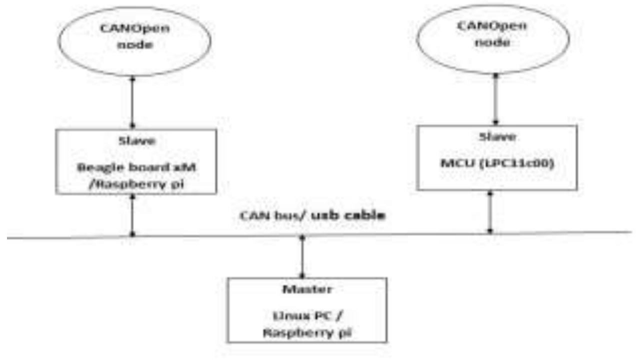
CANopen is the internationally standardized CAN-based higher-layer protocol for embedded control system. The set of CANopen specification comprises the application layer and communication profile as well as application, device, and interface profiles. CANopen provides very flexible configuration capabilities. These specifications are developed and maintained by CiA members.

CANopen networks are used in a very broad range of application fields such as machine control, medical devices, off-road and rail vehicles, maritime electronics, building automation as well as power generation

Objectives

The purpose of this project is to develop a method of communication between CANopen nodes and master to user with serial communication bus/USB through the use of CANopen protocol.

In order to establish communication using Canfestival Framework needed on Linux platform as Shown in Figure 1.



ure 1: Proposed System

Communication Objects:

Object Dictionary:

The central part of a CANopen device is the object dictionary. It is essentially a grouping of objects stored in a lookup table. It can be accessed from the network through a 16-bit index and an 8-bit sub index for individual data structure elements.

•**Index** – The object dictionary index.
•**Object** – The object type (Variable, Array, Record etc.).
•**Name** – The name of the entry.
•**Type** – The data type (Integer16, Boolean, Unsigned32 etc.).
•**Access Attributes** – Read and write attributes.

Service Data Objects (SDO)

Service data object provides remote access to the object dictionary. It uses a Client/Server communication scheme where the owner of the dictionary poses as a server and the device with upload/download request poses as a client which is shown in figure 2

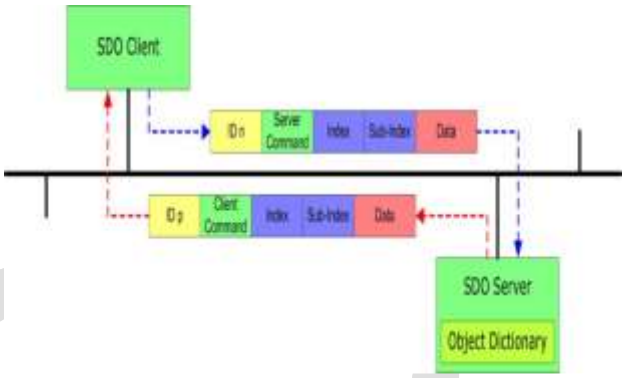


Figure: 2 Service data Object

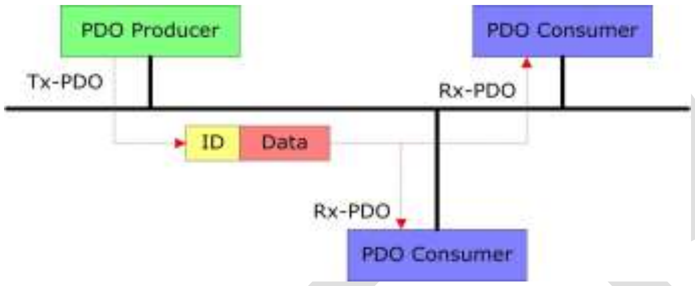


Figure 3: Process data Object

Process data object (PDO):

The PDOs are used to send and receive data used during the device operation, which must often be transmitted in a fast and efficient manner. Therefore, they have a higher priority than the SDOs

Synchronization object (SYNC)

Synchronization object provides a basic network clock. The SYNC message is transmitted periodically by SYNC producer and received by devices with support for synchronous TPDOs. TPDO may be configured to trigger transmission each n occurrences of SYNC message.

Emergency object (EMCY) :

The emergency object (EMCY) is used to signalize the occurrence of an error in the device. Every time that an error occurs (short-circuit, overvoltage, communication failure, etc.), this object will send an emergency message to the network. This message can be interpreted by an EMCY consumer (usually the network master), which will be able to take an action according to the programmed for the application, such as performing an error reset or disabling the other devices in the network.

Network Management (NMT):

The network management object is responsible for a series of services that control the communication of the device in a CANopen network , Every CANopen device has to implement a state machine. Network management services allow master device (or configuration tool) to remotely change the state. The device is set into the "initializing" state after a hardware reset and when initialization process is done the state is automatically changed to "pre- operational". Special BOOTUP message is generated on this transition so master is informed.

Canfestival Framework

Canfestival is an Open source CANopen framework.

Canfestival focuses on providing an ANSI-C platform independent CANopen stack that can be built as master or slave nodes on PCs, Real-time IPCs, and Microcontrollers.

It acts as a tool for CANopen.

It supports both Linux and WIN32.

Proposed System Architecture

System Works in three portions:

1. CANopen nodes for Monitoring or collecting data.

2. Host Device/Beagle Board gather data from CANopen nodes and send it to the Web server

3. User at any place can access that data from the Web server, also respond to the nodes

The main hardware which I will use Linux base board like beagle board xm as a Slave and another side pc/beagle bone as a Master which is connected to each other via CANopen over Serial Communication bus/USB and with the Slave node there is such a peripheral devices like sensor, actuator.

With the using of Framework Canfestival otherwise using CANopen Node, We can Implement CANopen in Linux based Environment with the serial communication bus,

But One Possible Idea is with the USB or UDP which is not implemented.

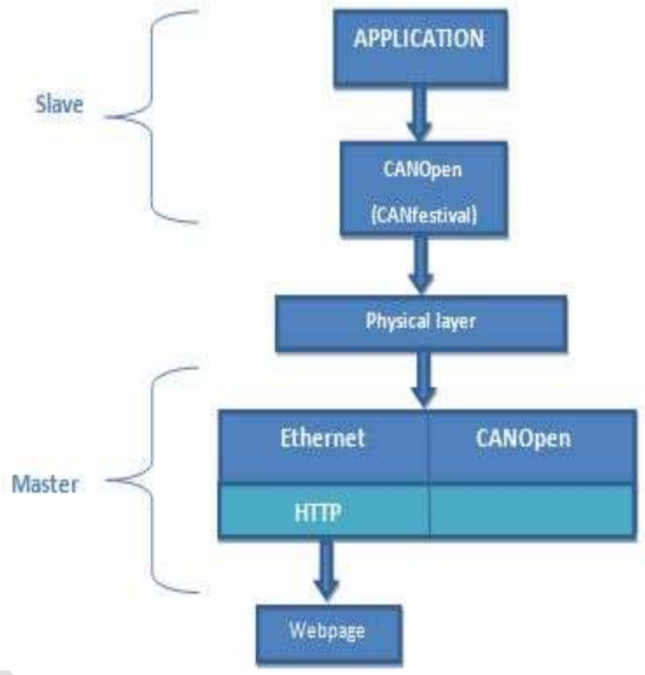
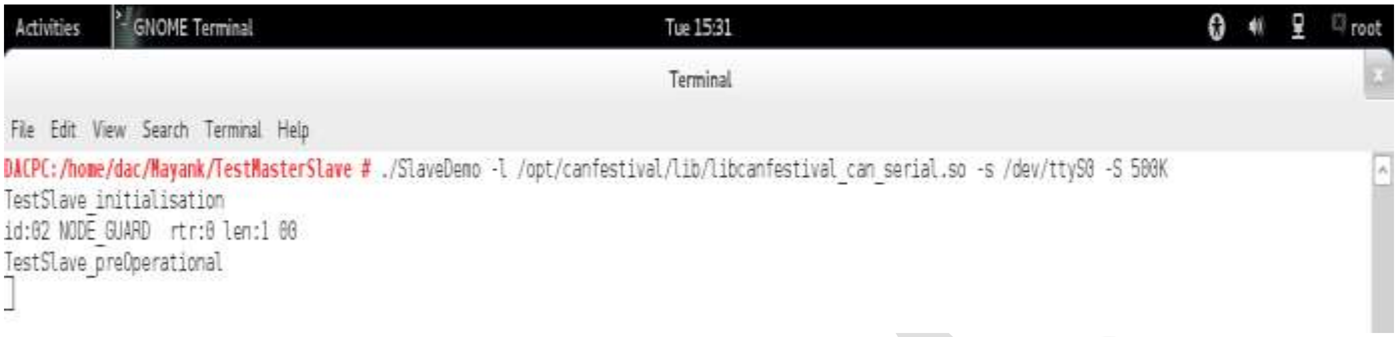


Figure: 4 Proposed system Architecture

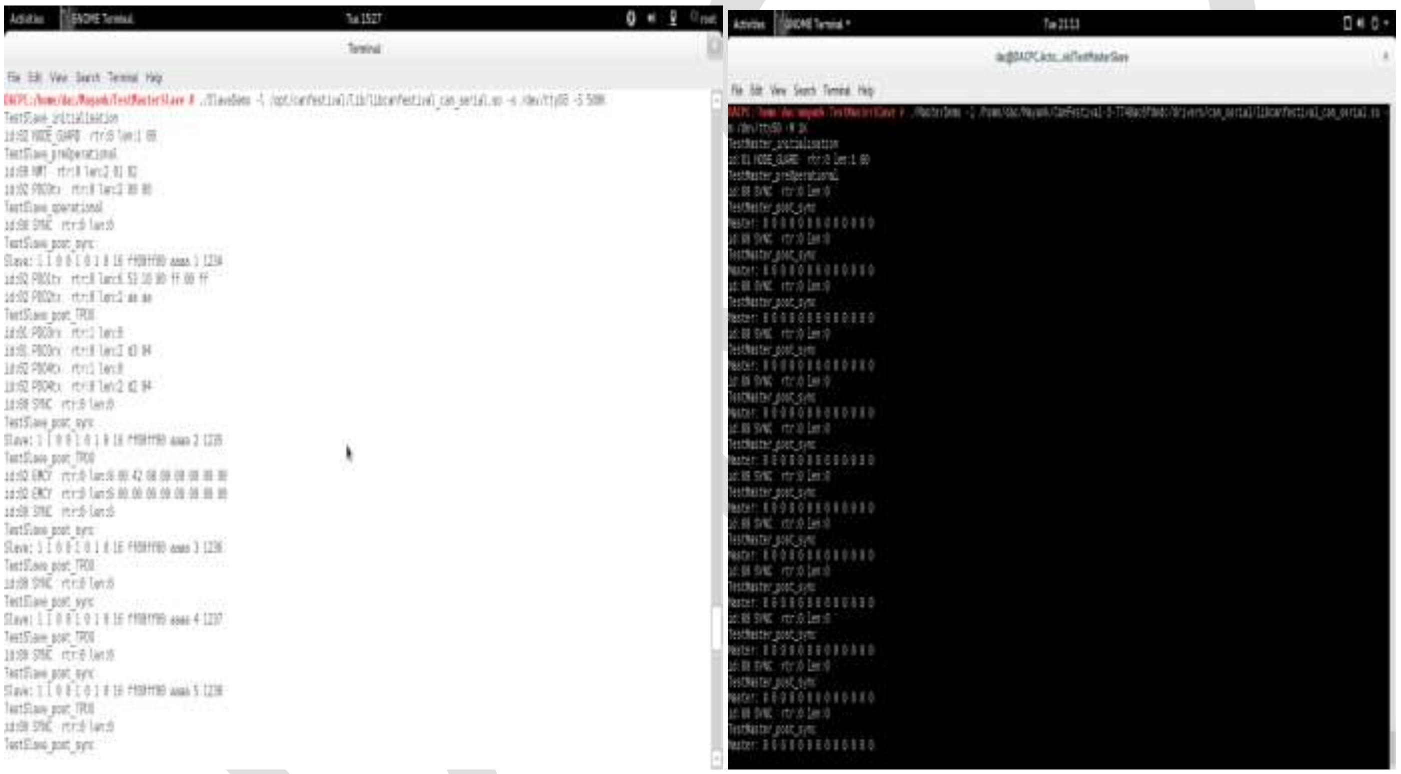
Result and Discussion

The testing for the system was conducted in different parts as this involves Testmasterslave testing for different methods (part by part implementation). With the single process with physical medium as a serial cable. The implementation involved separate codes written for testing method.



```
DACPC:/home/dac/Mayank/TestMasterSlave # ./SlaveDemo -l /opt/canfestival/lib/libcanfestival_can_serial.so -s /dev/ttyS0 -S 500K
TestSlave_initialisation
id:02 NODE_GUARD  rtr:0 len:1 00
TestSlave_preOperational
```

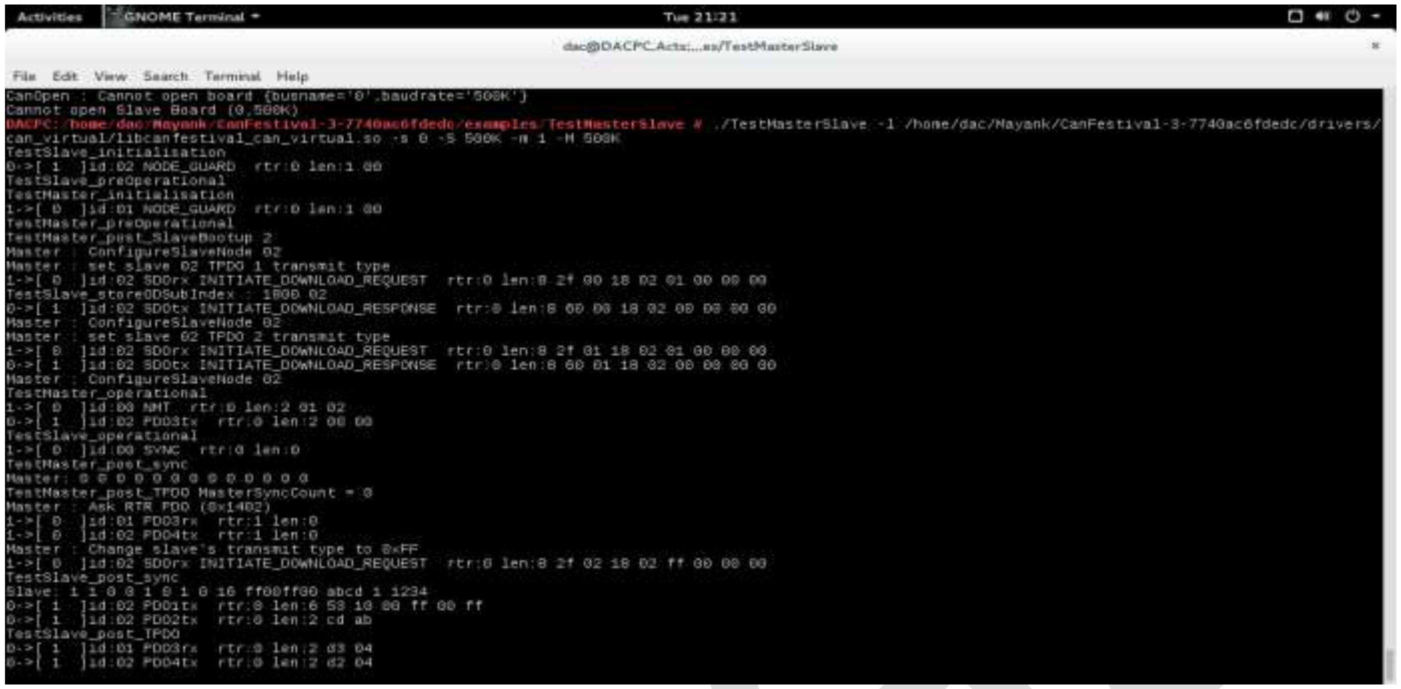
Figure: 5 Node in Pre-Operational mode



```
DACPC:/home/dac/Mayank/TestMasterSlave # ./SlaveDemo -l /opt/canfestival/lib/libcanfestival_can_serial.so -s /dev/ttyS0 -S 500K
TestSlave_initialisation
id:02 NODE_GUARD  rtr:0 len:1 00
TestSlave_preOperational
id:00 NMT  rtr:0 len:2 01 02
id:02 PDO0tx  rtr:0 len:2 00 00
TestSlave_operational
id:00 SYNC  rtr:0 len:0
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 aaaa 1 1234
id:02 PDO1tx  rtr:0 len:6 53 10 00 ff 00 ff
id:02 PDO2tx  rtr:0 len:2 aa aa
TestSlave_post_TPDO
id:01 PDO3rx  rtr:1 len:0
id:01 PDO3rx  rtr:0 len:2 03 04
id:02 PDO4rx  rtr:1 len:0
id:02 PDO4rx  rtr:0 len:2 02 04
id:00 SYNC  rtr:0 len:0
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 aaaa 2 1235
TestSlave_post_TPDO
id:02 EMCY  rtr:0 len:8 00 42 08 00 00 00 00 00
id:02 EMCY  rtr:0 len:8 00 00 00 00 00 00 00 00
id:00 SYNC  rtr:0 len:0
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 aaaa 3 1236
TestSlave_post_TPDO
id:00 SYNC  rtr:0 len:0
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 aaaa 4 1237
TestSlave_post_TPDO
id:00 SYNC  rtr:0 len:0
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 aaaa 5 1238
TestSlave_post_TPDO
id:00 SYNC  rtr:0 len:0
TestSlave_post_sync
```

```
DACPC:/home/dac/mayank/TestMasterSlave # ./MasterDemo -l /home/dac/Mayank/CanFestival-3-[illegible]/drivers/can_serial/libcanfestival_can_serial.so -s /dev/ttyS0 -M 1M
TestMaster_initialisation
id:01 NODE_GUARD  rtr:0 len:1 00
TestMaster_preOperational
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure: 6 Output with two different process


Activities
GNOME Terminal
Tue 21:21
dac@DACPC.Acts:...es/TestMasterSlave
File Edit View Search Terminal Help

```
CanOpen : Cannot open board {busname='0',baudrate='500K'}
Cannot open Slave Board (0,500K)
DACPC:/home/dac/Mayank/CanFestival-3-7740ac6fdedc/examples/TestMasterSlave # ./TestMasterSlave -l /home/dac/Mayank/CanFestival-3-7740ac6fdedc/drivers/
can_virtual/libcanfestival_can_virtual.so -s 0 -S 500K -m 1 -M 500K
TestSlave_initialisation
0->[ 1  ]id:02 NODE_GUARD  rtr:0 len:1 00
TestSlave_preOperational
TestMaster_initialisation
1->[ 0  ]id:01 NODE_GUARD  rtr:0 len:1 00
TestMaster_preOperational
TestMaster_post_SlaveBootup 2
Master : ConfigureSlaveNode 02
Master : set slave 02 TPDO 1 transmit type
1->[ 0  ]id:02 SDOrx INITIATE_DOWNLOAD_REQUEST  rtr:0 len:8 2f 00 18 02 01 00 00 00
TestSlave_storeODSubIndex : 1800 02
0->[ 1  ]id:02 SDOtx INITIATE_DOWNLOAD_RESPONSE  rtr:0 len:8 60 00 18 02 00 00 00 00
Master : ConfigureSlaveNode 02
Master : set slave 02 TPDO 2 transmit type
1->[ 0  ]id:02 SDOrx INITIATE_DOWNLOAD_REQUEST  rtr:0 len:8 2f 01 18 02 01 00 00 00
0->[ 1  ]id:02 SDOtx INITIATE_DOWNLOAD_RESPONSE  rtr:0 len:8 60 01 18 02 00 00 00 00
Master : ConfigureSlaveNode 02
TestMaster_operational
1->[ 0  ]id:00 NMT  rtr:0 len:2 01 02
0->[ 1  ]id:02 PDO3tx  rtr:0 len:2 00 00
TestSlave_operational
1->[ 0  ]id:00 SYNC  rtr:0 len:0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_TPDO MasterSyncCount = 0
Master : Ask RTR PDO (0x1402)
1->[ 0  ]id:01 PDO3rx  rtr:1 len:0
1->[ 0  ]id:02 PDO4tx  rtr:1 len:0
Master : Change slave's transmit type to 0xFF
1->[ 0  ]id:02 SDOrx INITIATE_DOWNLOAD_REQUEST  rtr:0 len:8 2f 02 18 02 ff 00 00 00
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1 1234
0->[ 1  ]id:02 PDO1tx  rtr:0 len:6 53 10 00 ff 00 ff
0->[ 1  ]id:02 PDO2tx  rtr:0 len:2 cd ab
TestSlave_post_TPDO
0->[ 1  ]id:01 PDO3rx  rtr:0 len:2 d3 04
0->[ 1  ]id:02 PDO4tx  rtr:0 len:2 d2 04
```

Figure: 5 Virtual Output with Single process

Acknowledgements

Any accomplishment requires the effort of many people and this work is not different. And it is my prime duty to acknowledge the person who directly or indirectly helped me during completion of this dissertation report. So I take opportunity to heartily thank our project guide respected **Mr.Aditya Sinha** and our co-guide respected **Mr. Rajesh sola** for his valuable guidance and touch of inspiration and motivation throughout the literature work without whose help the work would not have been in the shape what it is.

Conclusion

The concept introduces Implementation of CANopen protocol for Industrial and Automotive Applications making use of the Canfestival, Data transmission used Serial communication and CANopen protocol with the Canfestival framework. After testing and application, the system seems high precision and well compatibility, and the data seems exact and in real time. In the in-depth future study, I should add USB driver which is not yet available in Canfestival,

To minimize the power consumption and cost, User can access the data packet with own android gadgets or laptops through web server from any place.

REFERENCES:

[1] G. Cena and A. Valenzano, “A protocol for automatic node discovery in CANopen network,” IEEE Transactions on Industrial Electronics, vol. 50, no. 3, pp. 419-430, June 2003.

[2] M. Farsi, K. Ratcliff and Manuel Barbosa, “An introduction to CANopen,” Computing & Control Engineering Journal, vol. 10, pp. 161- 138, Aug. 1999.

[3] M. Farsi and M. Barbosa, CANopen implementation applications to industrial networks. England: Research Studies Press Ltd, 2003.

[4] Alexios Lekidis, Marius Bozga, Saddek Bensalem Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France CNRS, VERIMAG, F-38000 Grenoble, France : Model-based validation of CANopen systems.

[5] M. Barbosa, M. Farsi, C. Allen, and A. Carvalho, “Formal validation of the CANopen communication protocol”, in Fieldbus Systems and Their Applications 2003:(FET 2003): a Proceeedings Volume from the 5th IFAC International Conference, Aveiro, Portugal, 7-9 July 2003, volume 5, July 2003, pp. 226–238. Elsevier Science Limited, IFAC.

[6] O. Pfeiffer, A. Ayre, and C. Keydel, *Embedded networking with CAN and CANopen*, Copperhill Media, 2008.

[7] CAN in Automation, Application Note 802, August 2005.

[8] CAN in Automation, "CANopen Device Profile for Generic I/O Modules, Draft Standard 401", June 2008.

[9] Pfeiffer, O.; Ayre, A. & Keydel, C. Embedded Networking with CAN and CANopen, Copperhill Technologies Corporation, 2008

[10] CAN in Automation, *CANopen Electronic data sheet specification for CANopen*, CiA Draft Standard 306, Version 1.3, and January 2005.

[11] CAN in Automation, *CANopen Application Layer and Communication Profile*, CiA Draft Standard 301, CAN in Automation, Version 4.02, 13 February 2002.

[12] Li Pengfei School of Electronics and Information, Xi ' an Polytechnic University Xi ' an, 710048, China,li6208@163.com "Application of CANopen and Modbus Protocol in Rotary Screen Printing Machine Control System", 2010