MODELLING AND REAL-TIME SIMULATION OF A NETWORKED-CONTROL SYSTEM WITH DISTRIBUTED LOADS

by
Isizoh A. N.[1], Ogu C.D.[2], Nwokoye A.O. C[3], Okide S.O.[4]

1,2: Department of Electronic and Computer Engineering, Nnamdi Azikiwe University, Awka, Nigeria.
3:Department of Physics and Industrial Physics, Nnamdi Azikiwe University, Awka, Nigeria.
4: Department of Computer Science, Nnamdi Azikiwe University, Awka, Nigeria.

Abstract
This paper discusses a control system model which has four access layer servers connected to several loads via four Controller Area Networks (CANs), and these loads are controlled from those servers or virtual interface machines. The system network involves interconnection of Access Layer Servers (e.g. laptops), Ethernet Controller (which establishes a Local Area Network, LAN), and CAN Controllers. This study can be implemented in the control of industrial processes in a real-time. In this work, the Programmable Logic Device (PLD) of the Ethernet controller which is its intelligent Application Specific Integrated Circuit (ASIC) was replaced with a programmed AT89C55 microcontroller to achieve the same switching logic. This microcontroller was programmed in Assembly Language, and the feedback mechanism was achieved using program. Simulation of the system was carried out in a real-time simulation environment using Proteus software (Proteus version 7.6 Professional), and this confirmed the workability of the model in a real-life situation.

Keywords and phrases: CAN controller, LAN network, Ethernet controller, AT89C55 microcontroller.

Introduction

There are many ways of controlling loads that are remotely distributed in a large industry or factory, such as using GSM handset, Internet-based method, etc. One of such ways that even appears to be one of the most effective methods is by the use of an Ethernet controller to form a LAN network which connects many computers together. And these computers which form the nodes in that network are used to control several loads via different CAN controllers [1]. One good thing about this method is that each load in the network is linked to a particular load, and this enhances excellent control performance.

How this is achieved is that, each CAN controller is given an address which uniquely identifies that CAN, and makes it accessible at the Access layer terminal. And for every operation within the network, there is a feedback from the network, which explains the nature of the operation taken and its status.

In other to handle this research work, the following areas were looked into:

a) The technologies and implementation of CAN controller and Ethernet controller in control systems.
b) Access terminals of four nodes
c) The appropriate feedback mechanism
d) Process control in the network

Block diagram of the developed Control System

The block diagram of the system comprises of five distinct units/parts as shown in figure 1. The units are: the Access Terminal Unit, the Switching Unit, the Signal Indicator Panel Unit, the Controller Unit and Loads.

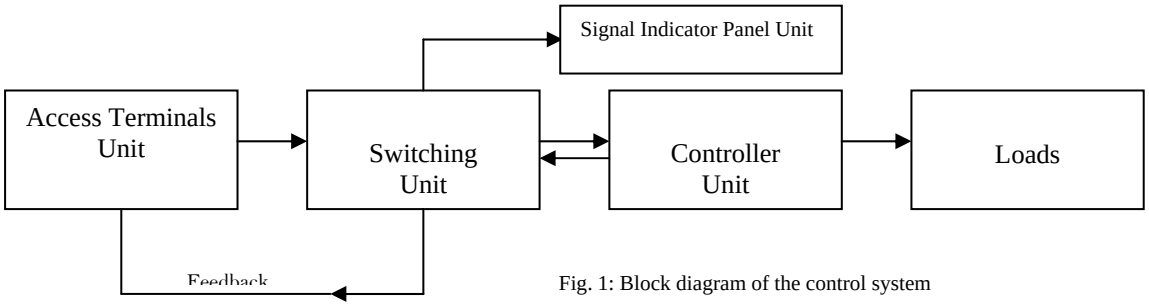

Fig. 1: Block diagram of the control system

Access Terminal Unit: This comprises of four (4) access layer servers. It can be of any number, depending on the user's request. Here, only four were used.
The primary function of this access terminal unit is to supply the needed address of each of the CAN controllers, based on their load priorities. Also, the terminal codes or signal controls are generated by these access layer servers for driving loads. This is possible because,

1. Each access terminal supplies the addresses of the CAN controllers, since every CAN has an address of its own which uniquely identifies it.
2. It contains port numbers of each CAN.
3. It sends the corresponding signal to the CAN which controls the loads.

In a real life situation, these access terminals are computers. Thus in this work, the access terminals are logically connected to the virtual ports of the Ethernet in a Proteus interface.

Switching Unit: This unit principally consists of an Ethernet. It is distributive in the sense that it ensures that any address which emanates from any of the access terminal servers uniquely identifies a particular CAN controller. It has the intelligence to know which address is for a CAN controller.
The Ethernet has an inbuilt MAC address table for tracking each of these CAN controllers.

Controller Unit: The main device in this unit is the Controller Area Network (CAN). It is a vehicle bus standard designed to allow microcontrollers and devices communicate with each other within a vehicle without a host computer. Each CAN bus has an address which uniquely identifies it. When an operator or user wants to control a device connected to any CAN, the operator has to enter the address of that CAN on any of the access terminals together with some appropriate codes.

Loads: These are the devices to be controlled. But in this dissertation, Light Emitting Diodes (LEDs) were used as loads to provide the necessary animations.

Signal Indicator Panel: All the control indicators like: error A, error B, error C, error D, receive and transmit messages are displayed on this panel. It is left for the programmer to determine the duration of each light.

Layout model
The layout model of the entire system, known as Isizoinyiama layout model, is shown in figure 2. The layout involves four access terminals connected to an Ethernet. From this Ethernet, connections are made to different Controller Area Network (CAN) devices. With this arrangement, several loads can then be connected to the CAN devices.

Much work was done on the switching system of the Ethernet in order to achieve the required switching logic function for the CANs, so as to be able to control loads.
Thus the control operation is done by the use of Controller Area Networks (CANs), which are linked to the loads through their various CAN buses. The Ethernet also provides the network part of the study. Since the typical industrial scenario used in this dissertation is a chemical industry, the network to be provided is a Local Area Network (LAN) because the industry is assumed to be within 100 meters square.

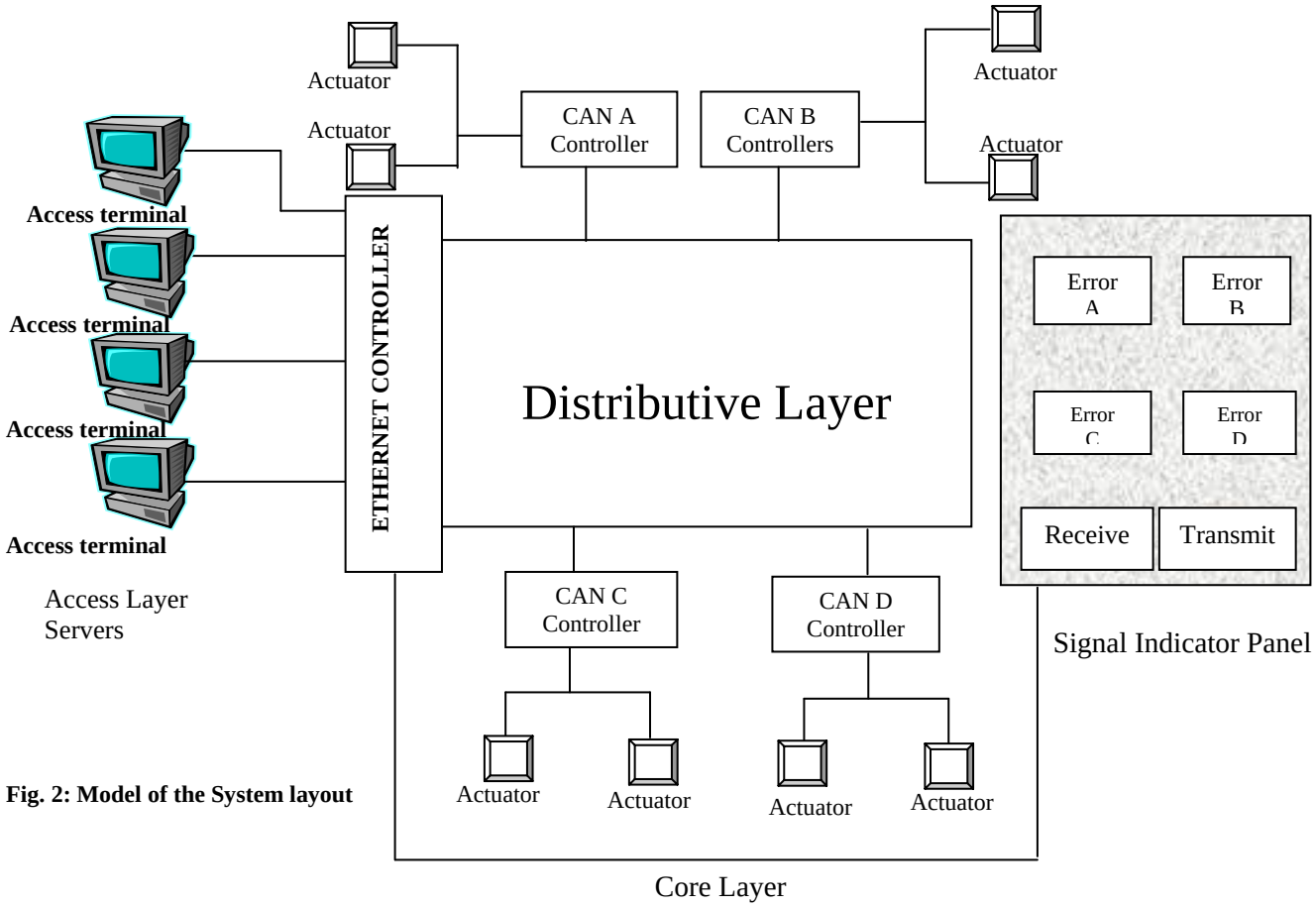

Fig. 2: Model of the System layout

A simplified model of this control system is shown in figure 3 below. There are two nodes. Node 1 has the Ethernet and the four Access terminals, while node 2 has the four CANs and loads. The model uses tree topology.

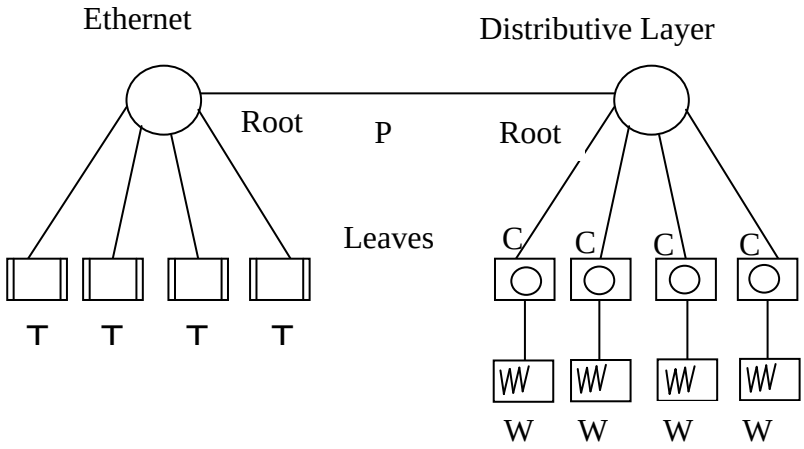

Fig. 3.4: Simplified model of the system

Where:
T = Access terminal
P = Packet
C = CAN Controller
W = Load

Development of the Switching Function

Every Ethernet controller is usually developed to provide a good switching function for its network. It is the part of the switch that stores or carries the program for the switching function of the Ethernet.

Thus for every Ethernet controller, there must be an accompanied program which the Ethernet uses for its function. Since this program is peculiar to it, the controller is usually referred to as Application Specific Integrated Circuit (ASIC); meaning that the manufacturer of the controller has already put in a program permanently and that the controller can only function based on that program and nothing else.

In this paper, this controller was replaced with an AT89C55 microcontroller. Thus, since the microcontroller is a computer of itself, the design of the switching function must be a software-base.
The developed switching system of the Ethernet is seen by placing cursor on the Ethernet controller in the diagram of figure 4, and pressing Ctrl C. When this is done, it will be seen that there are four AT89C55 microcontrollers labeled U1, U2, U3 and U4.

Each microcontroller pairs with one particular virtual terminal in order to control the distributed loads via CAN controllers. The analysis of each microcontroller arrangement here is the same with other three microcontrollers.

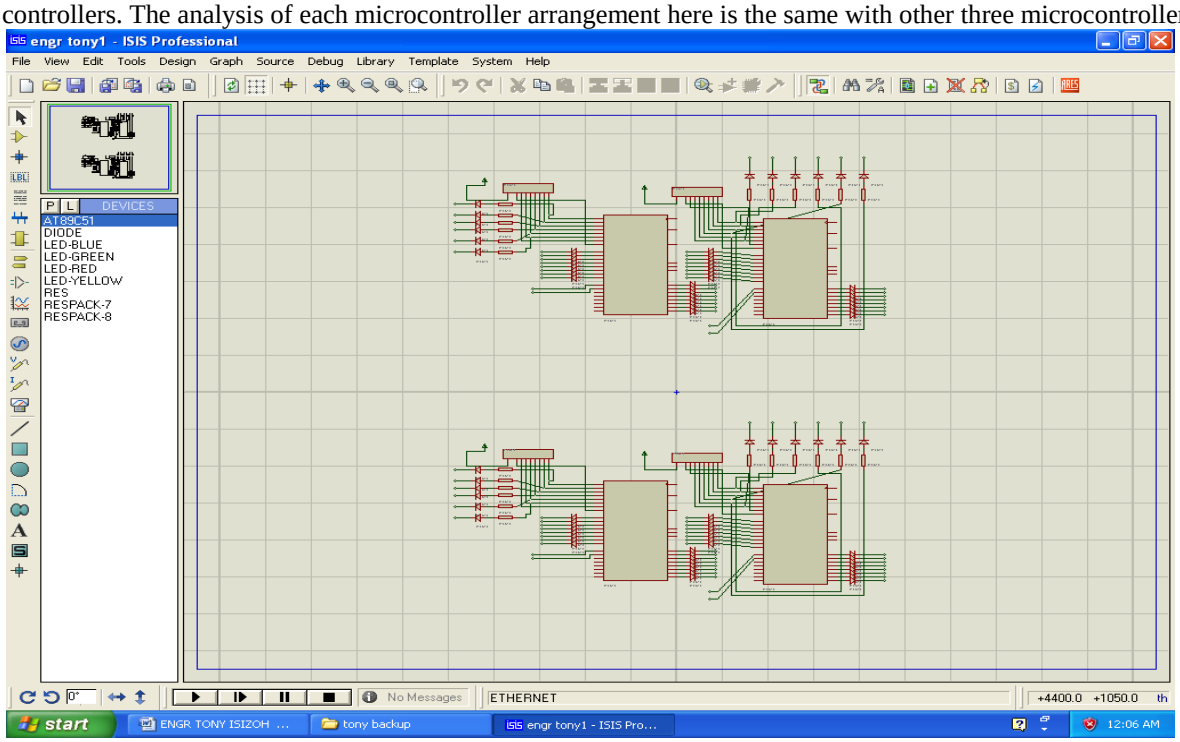

Fig.4: Arrangement of the four microcontrollers in the Ethernet

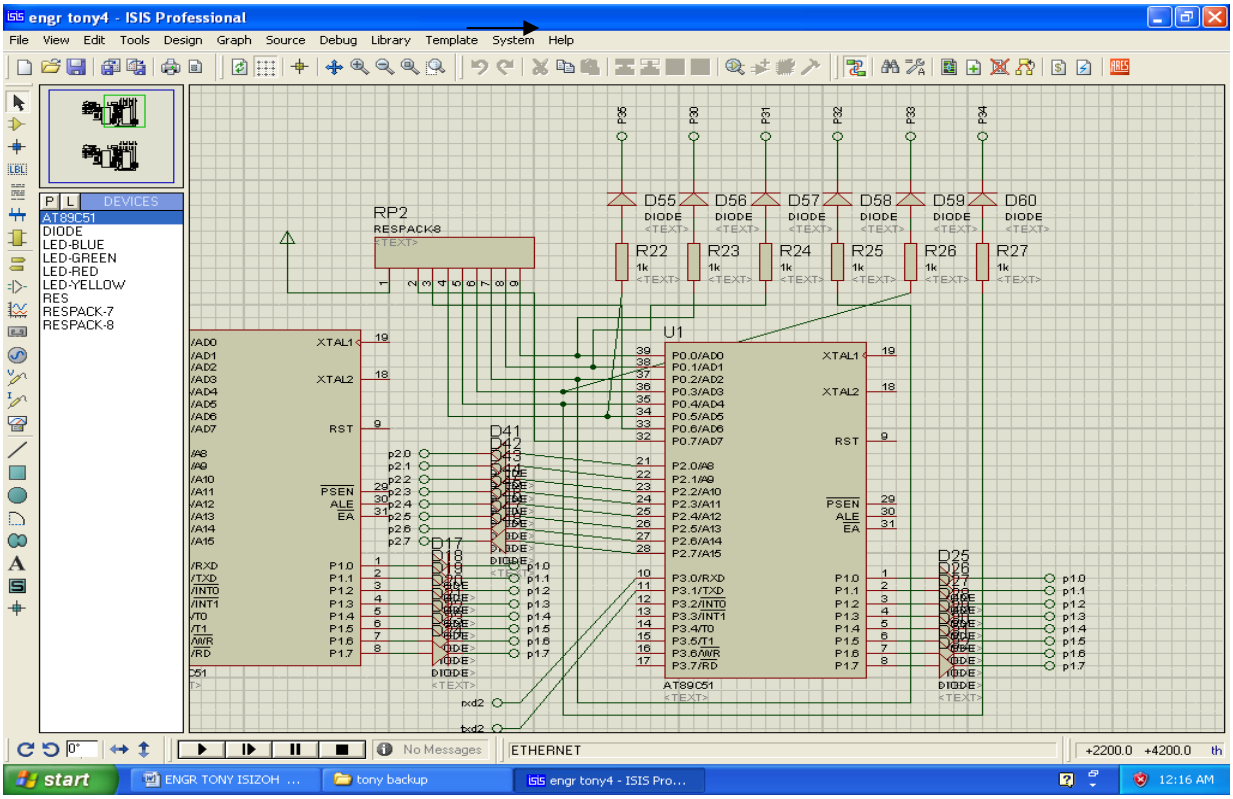

Fig. 5: Detailed connection of one microcontroller

By default, Port 0 (i.e. pin 32 to pin 39) is zero. Therefore to make it active (i.e. to have ones on pin 32 to pin 39), resistor pack-8 (labeled RP) called pull-up resistors are connected to the port (i.e. pins 32, 33, 34, 35, 36, 37, 38 and 39). Also in this Port 0, pin 34 to pin 39 are connected to the Signal Indicator Panel of the layout as listed below.

Pin 39 ⟶ Error A (LED)
Pin 38 ⟶ Error B (LED)
Pin 37 ⟶ Error C (LED)
Pin 36 ⟶ Error D (LED)
Pin 35 ⟶ Received (LED)
Pin 34 ⟶ Transmit (LED)

To enable the above connection to be achieved, these pins are connected to resistors (of 10kΩ each). These resistors are to limit the currents entering into the Light Emitting Diodes (LEDs) in the Signal Indicator Panel through their individual diodes (D61-D66) which serve a special purpose of making sure that there is no reverse current.

The Port 1 and Port 2 of the microcontroller are connected to the distributed loads in the network.
The connections between the virtual ports of the microcontroller and the CAN controllers are as written below:

For Port 1,

P1.0 ⟶ CAN A, P_0 ⟶ D_1
P1.1 ⟶ CAN A, P_1 ⟶ D_2
P1.2 ⟶ CAN A, P_2 ⟶ D_3
P1.3 ⟶ CAN A, P_3 ⟶ D_4
P1.4 ⟶ CAN B, P_4 ⟶ D_5
P1.5 ⟶ CAN B, P_5 ⟶ D_6
P1.6 ⟶ CAN B, P_6 ⟶ D_7
P1.7 ⟶ CAN B, P_7 ⟶ D_8

For Port 2

P2.0 ⟶ CAN C, P_8 ⟶ D_9
P2.1 ⟶ CAN C, P_9 ⟶ D_{10}
P2.2 ⟶ CAN C, P_{10} ⟶ D_{11}
P2.3 ⟶ CAN C, P_{11} ⟶ D_{12}
P2.4 ⟶ CAN D, P_{12} ⟶ D_{13}

P2.5 ⟶ CAN D, P_{13} ⟶ D_{14}
P2.6 ⟶ CAN D, P_{14} ⟶ D_{15}
P2.7 ⟶ CAN D, P_{15} ⟶ D_{16}

The Port 3 of the microcontroller is not fully connected to any input/output (I/O) device except pin 10 and pin 11 (i.e. P3.0 and P3.1) which are connected to a virtual terminal (or access terminal). It should be noted that each access terminal is connected to a particular microcontroller.
One can use Ctrl X to get back to the normal interface or 'Home' environment.

Considerations
Each microcontroller is assumed to be a terminal of its own and must be paired with one access terminal. All the diodes used in this work are IN4001. They are all signal diodes used for directing signals.

Here, each resistor that was used in the biasing of each of these diodes is 1KΩ. This is because the AT89C55 microcontroller manufacturer's standard stipulates that the maximum allowable pin current for its 'sinking' is 5mA.
So since V = IR …….. (1),
then R = V / I …….. (2)
But V = 5V, and I = 5mA
Therefore, R = 5 / 0.005, which is 1000Ω or 1K Ω.

Each RP (Resistor Pack) used in this work has 8 resistors inside one pack. It is a packed pull-up resistors connected to port 0 of each microcontroller. The reason for this connection is that, by default, the Port 0 of this microcontroller is actively low, and to be able to use it, this port has to be made actively high by connecting pull-up resistors to the port [2].
Based on the AT89C55 manufacturer's standard, the value of each of these resistors in a pack is between 1KΩ to 10KΩ. But in this paper, 10KΩ was used.

It takes each microcontroller a default maximum time of 2mS to set up for operation from power-up. This is a reset time to enable it get set for its internal operation.

Thus it is a standard thing that an RC circuit is connected to pin 9 of each microcontroller, and this RC circuit is also called a reset circuit. This circuit generates the reset time, T for the microcontroller. Figure 6 below shows this reset circuit.

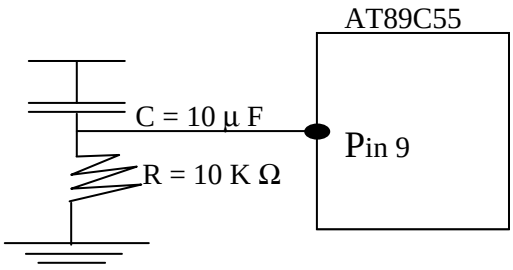

Fig. 6: Reset circuit connection to Pin 9

So the reset pin 9 of each microcontroller has a capacitor, C and a resistor, R. Based on AT89C55 manufacturer's standard, the choice of C and R are:
C = 10μF
R = 10KΩ.
The reason is because the period, T = 1.1 RC ……. (3).
Since C = 0.00001F, and R = 10000Ω, then,
T = 1.1 x 0.00001 x 10000 = 1.1mS. So these values of C and R are okay, since the value of T is not up to 2mS.
The AT89C55 has an on-chip oscillator but requires an external clock to run it. A quartz crystal oscillator is connected to inputs XTAL 1 (pin 19) and XTAL 2 (pin 18).

The essence of using crystal oscillator is to control the speed of the microcontroller. Here, the speed of the microcontroller refers to as the maximum oscillator frequency connected in between XTAL 1 and XTAL 2. In this paper, a crystal oscillator with 12MHz frequency was used. The frequency can be observed on the XTAL 2 (pin 18) using an oscilloscope.

For a CPU to execute an instruction, it takes a certain number of clock cycles, popularly referred to as machine cycles [3].
To calculate this, one machine cycle lasts for 12 oscillator periods. Therefore, the machine cycle is calculated to be the inverse of 12 of the crystal frequency. In this work, 12MHz crystal oscillator was used.
Based on this value of the crystal oscillator, 12 was however used as the Input clock value in MHz when the Delay program was being developed. This is based on the manufacturer's standard because the value of the crystal oscillator will determine the number to be used as the Input clock value in MHz for a Delay Program [4].

The Pseudocode

```
Begin
    Press button
    If button pressed is A
   Then load default IP for CAN A
    Else if button pressed is B
    Then load default IP for CAN B
    Else if button pressed is C
    Then load default IP for CAN C
    Else if button pressed is D
    Then load default IP for CAN D
    Else go to Begin
Press button
    If button pressed is 1
    Then ON Load_1 in CAN A
    Else if button pressed is 2
    Then ON Load_2 in CAN A
    Else if button pressed is 3
    Then ON Load_3 in CAN A
    Else if button pressed is 4
    Then ON Load_4 in CAN A
    Else if button pressed is 5
    Then OFF Load_1 in CAN A
    Else if button pressed is 6
    Then OFF Load_2 in CAN A
    Else if button pressed is 7
    Then OFF Load_3 in CAN A
    Else if button pressed is 8
    Then OFF Load_4 in CAN A
Stop.
```

The above pseudocode is the control sequence for the Controller Area Network A (CAN A). It has to be repeated for CAN B, CAN C, and CAN D.

The Flowchart

The flowchart for the system operation is represented as below:

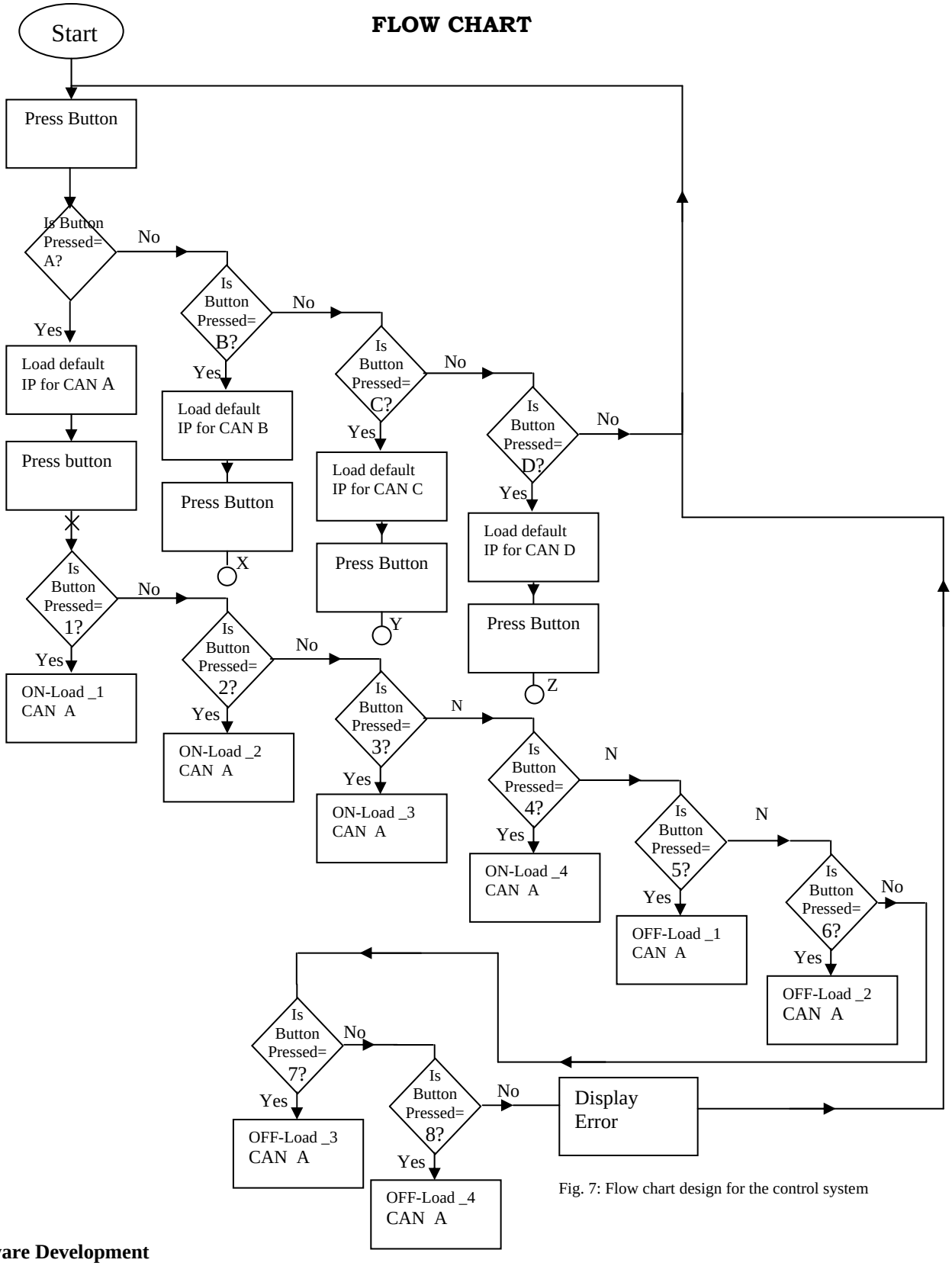

Fig. 7: Flow chart design for the control system

Software Development

Every microcontroller operation must be based on its control program. Without programming a microcontroller, it can never work. The pseudocode and flow chart above will enable software development be achieved.

A program for the control operation was developed in Assembly language using MIDE software. However, Assembly language provides one good thing: it lets you write programs at a low-level understanding, but does

not force you to memorize a set of numeric codes. Instead, you write instructions as English-like abbreviations, and then run an assembler program to convert those abbreviations to their numeric equivalents [5].
This original program is called the source program, while the numeric, microcontroller-compatible form of it is the object program. The assembler's job here is to convert the source program you can understand into an object program the microcontroller can understand.

System Implementation
The entire system can be set up for implementation as shown in figure 8. The Ethernet switch establishes the LAN network needed for this system implementation. The default setting of each virtual terminal is such that the CTS and RTS are connected together. With this provision, the four access terminals or nodes are connected to the Ethernet controller via the virtual ports of txd1, txd2, txd3, and txd4 for the four nodes respectively.

The output ports (P1 and P2) are connected to the CAN controllers, with each CAN port uniquely identified by its IP address. Each CAN has four loads connected to it. The loads are arranged as L1, L2, L3, and L4. In general, there are 16 loads connected to the network. For the purpose of this study, the loads are represented using LEDs.
Based on the above arrangement, the distributed loads can be controlled at any access terminal by typing the CAN controller address in capital letter, for example, A, B, C, or D. If what was entered was not any of these letters A to D, there would be a feedback error message, titled, "Error". Also if the letters A, B, C, or D you entered was not in capital letter (example, a, b, c, or d), error message would be displayed. Thus, if an error message is displayed, no operation can take place.
When the correct CAN address has been entered, the default ID address is automatically loaded on the access terminal, and then one is expected to enter the load control address or number. The operation at this stage will determine whether one wants to put on a load or put off a load.

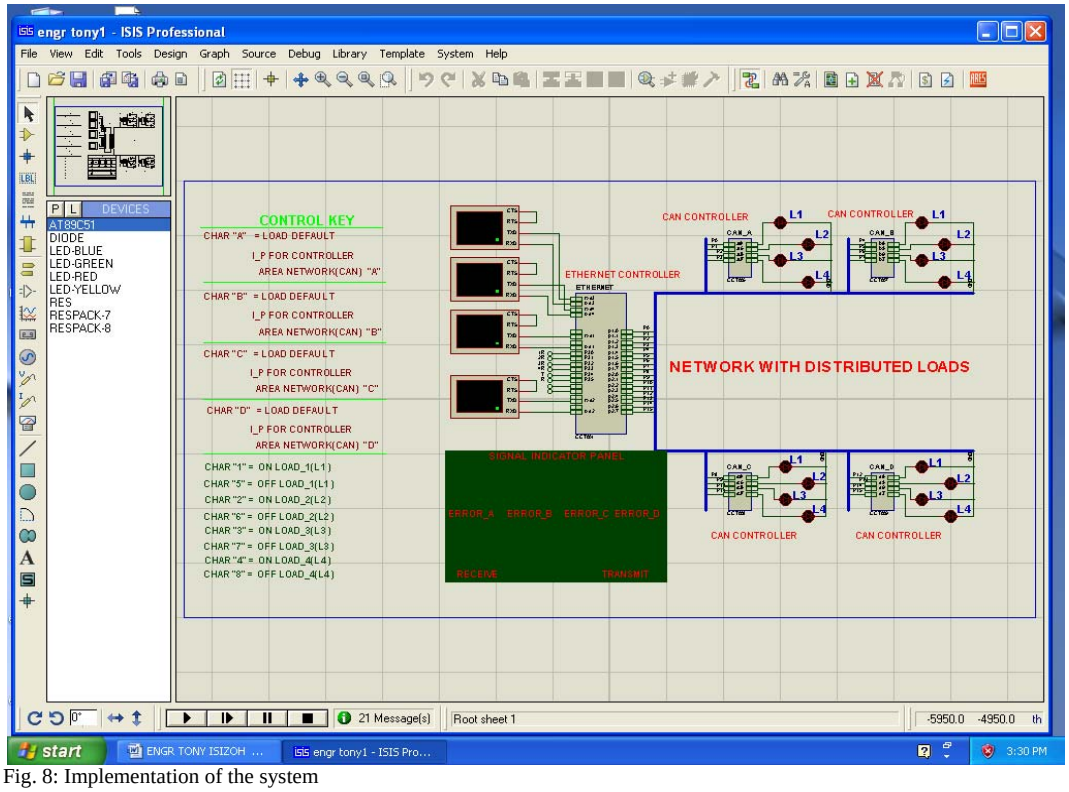

Fig. 8: Implementation of the system

Real-Time Simulation Results and Analysis
It is always necessary to perform a real-time simulation of any microcontroller-based system you have designed. This is because a program for an embedded system may be error-free, but when a microcontroller implements it, the desired goal is not achieved. It then becomes a matter of necessity to have a real-time simulation of the design first before the circuit is constructed so as to be sure of the workability of the program [6].
This kind of simulation helps one to see the operation of a system in a computer (virtual operation). Normally if a real-time simulation of any system is running then it means that the system is good and it will work in a real-life situation.

In general, real-time simulation confirms the workability of a program/design if implemented in a real life. To be able to carry out this, proteus software was used.

The system was developed in a run time so that when you click “play”, the four virtual terminals will come out. That is to say that when the real-time simulation of this control system is run (by clicking the play button), the four virtual terminals will be displayed. These represent the four nodes or workstations in the LAN network. In this case, the control of any load (i.e. L_1, L_2, L_3 or L_4) can be done by clicking on any of the virtual terminals or nodes, and then holding the “shift” key and type A or B or C or D depending on the CAN in which the load to be controlled is connected to. When this is done, the default IP is loaded on that particular network, expecting the user to start the control process. Figure 9 shows the real-time simulation interface of the four virtual terminals or nodes.

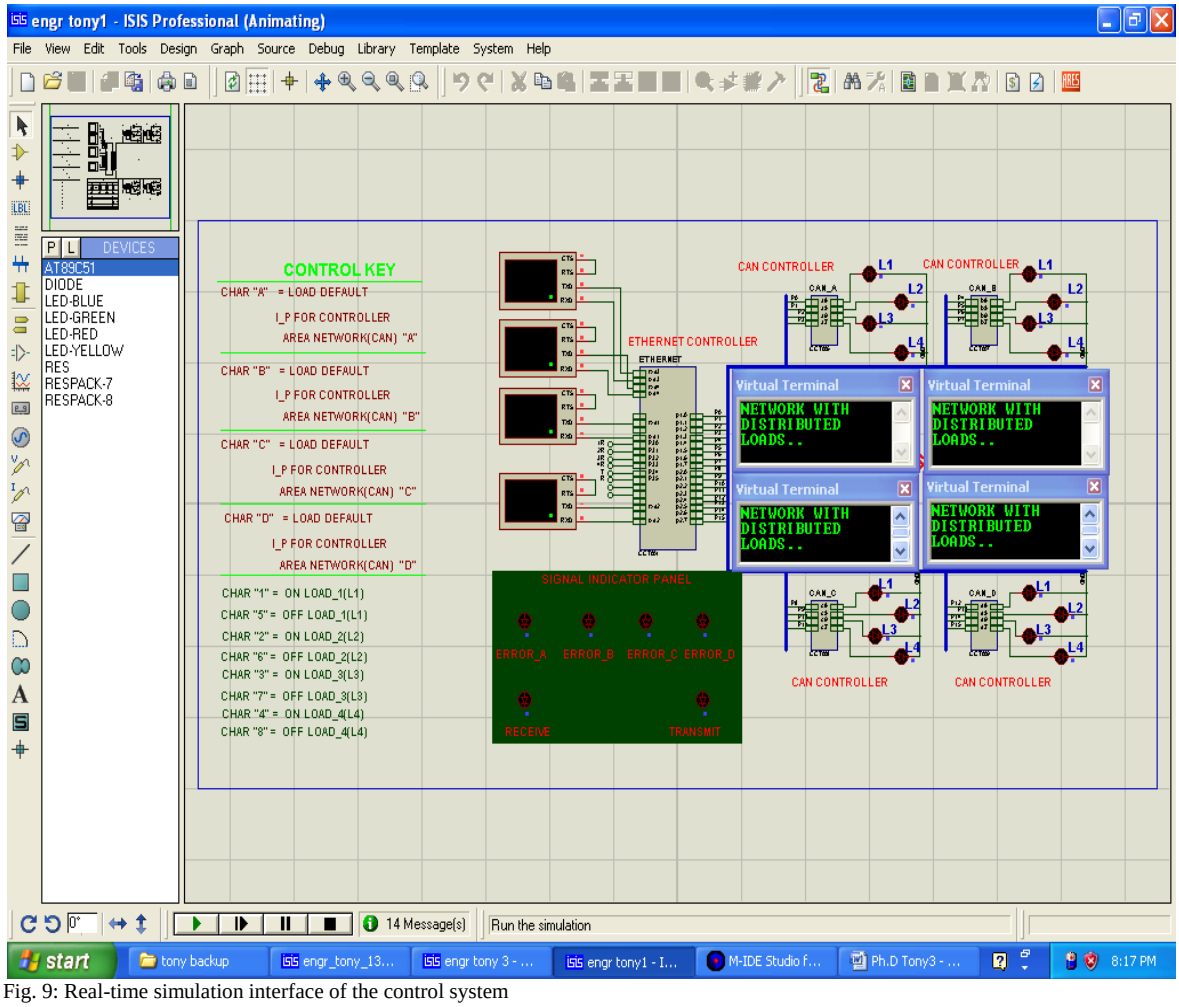

Fig. 9: Real-time simulation interface of the control system

The control keys for the loading of the IPs are as follows:
CHAR “A” - Load default IP for Controller Area Network (CAN) “A”
CHAR “B” - Load default IP for Controller Area Network (CAN) “B”
CHAR ‘C” - Load default IP for Controller Area Network (CAN) “C”
CHAR “D” - Load default IP for controller Area Network (CAN) “D”

When the default IP is loaded, it will be displayed on that virtual terminal or node. The virtual terminal will also display, “ENTER: 1 to 7 FOR LOADS CONTROL”. This message means that you can control the load (either ON or OFF) using numbers 1 to 7. The control keys for the load controls are stated hereunder:

CHAR “1” – On Load 1 (L_1)
CHAR “5”- Off Load 1 (L_1)
CHAR “2” – On Load 2(L_2)
CHAR “6”- Off Load 2 (L_2)
CHAR “3” – On Load 3 (L_3)
CHAR “7”- Off Load 3 (L_3)
CHAR “4” – On Load 4 (L_4)
CHAR “8”- Off Load 4 (L_4)

Any character you press which is not in line with the above will result in the display of an error message.

Once an error occurs, it will show on the signal indicator panel by blinking, while an "error" message will be displayed on that particular virtual terminal used. And if an error occurs then it means that control is no more possible because no control key can serve as input for any virtual terminal, although both the receive and transmit lights will still blink.

Conclusion

A very good aspect of a control system with distributed loads has been obtained. This was done by replacing the intelligent switch of the Ethernet controller with AT89C55 microcontroller which gives the same switching logic. In order to achieve this logic, a workable control programs, written in Assembly Language was designed using MIDE Assembler and TIME 8051 software. The program was later tested okay.

Thus by performing the real-time simulation of this networked control system with distributed loads, one was able to observe the reality of the work by viewing to operation of the system in a real-time simulation using Proteus VSM.

References

[1] Ogata Katsuhiko, "Modern Control Engineering", Fourth Edition, Prentice Hall of India, Private Limited, New Delhi, 2007, Pp 1-2.
[2] Predko Myke, "Handbook of Microcontrollers", McGraw Hill, New York, USA, 2003.
[3] Schuster A., "Microcontroller Principles and Applications", Maxon Press Ltd, Rochester, 2008.
[4] Douglas V.H., "Microcontrollers and Interfacing: "Programming Hardware" McGraw Hill Inc, New York, 2008.
[5] Holliday D. and Resmick Robert, "Fundamentals of Microcontrollers", Don Peters Press Ltd, Fulmar, 2008.
[6] Kopetz H., "Real-Time Systems: Design Principles for Distributed Embedded Applications", Kluwer Academic Publishers, Dordrecht, 2009.
[7] Wakerly John F., "Digital Designs: Principles and Practices", 4th Edition, PHI Learning Private Limited, New Delhi, 2008, Pp 5-6.
[8] Floyd T.F., "Digital Fundamental", 6th Edition, Prentice-Hall International Inc, New Delhi, 2008, Pp 4-7.
[9] Torngreen M., "A Perspective to the Design of Distributed Real-time Control Applications Based on CAN", 2nd International CiA CAN Conference, London, U.K., 1995, Pp 6.