A REMOTE ELECTRONIC OBJECT EMULATION SYSTEM FOR HOME BUS APPLICATIONS

Peter M. Corcoran and Karl Lusted
Dept. of Electronic Engineering,
University College,
Galway

Abstract

The design and implementation of a hardware emulation system for CEBus Objects is described. The emulator is accessed remotely from a PC-based GUI. It may be easily programmed to combine several CAL Objects to emulate common CEBus devices.

1. Introduction

The Consumer Electronics Bus (CEBus), sponsored by the Electronic Industries Association (EIA), is a multi-media LAN Standard developed for home automation systems. CEBus was developed to facilitate communication between appliances in the home for automation and control purposes. The reader is referred to [1] for further details. The CEBus standard defines an object-oriented Common Application Language (CAL) which allows the behaviour of devices on the network to be modelled as a collection of Objects. A hardware emulation system for CAL Objects has been developed. The purpose of the emulator is to provide easy access to the higher level functionality of the CEBus standard (i.e. CAL), while shielding the User from the complexities of the lower layers (i.e. the communications protocols).

2. Background

The CAL provides a framework through which *resource allocation* and *control functions* are executed [2]. Resource allocation is concerned with CEBus resources such as addresses, spectrum space, etc. The control functions of CAL are geared towards controlling devices on the network. The emulator was developed with these control functions in mind.

Generic Objects (e.g. Analog Sensors, Binary Switches, etc.) can be grouped together to simulate various CEBus devices (e.g. Television, VCR, stereo receiver, etc.). There are currently 24 different Objects defined for CAL. Further Objects may be added as required. An Object, as defined in the CEBus standard, is a 'model of a single functional entity used to perform a single control task' [2]. An Object model is implemented by a collection of Instance Variables (IVs), operated upon by Methods. Objects can also be configured to report changes in the state of IVs. Objects are generic, that is they don't assume a specific application until placed in a specific Context.

Contexts represent major device subsystems, and several instances of a particular Context may exist within a device. The inter-relationship between Contexts, Objects, and their IVs is known as device hierarchy, as indicated in Figure 1.

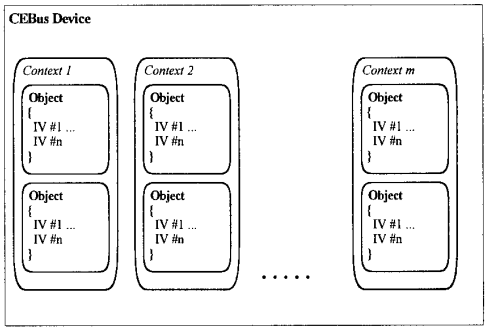


Figure 1: CEBus device hierarchy

3. System Overview

The complete system consists of a PC-based CEBus node and a remote 'generic' CEBus device (the emulator) containing a variety of CAL Objects. The PC provides access to the CEBus and the hardware emulator through a User-friendly Graphical User Interface (GUI). The CAL Objects may be linked to represent a variety of CEBus devices. Each Object consists of a virtual software representation on the PC and an actual hardware implementation on the emulator. This is achieved through a bi-directional mapping of CAL objects, possible by virtue of the object-oriented methodology inherent in the CAL specification. The emulation system overview is shown in Figure 2. The emulator has been verified using the CEBus standard over AC power line (PL) medium.

Manuscript received June 27, 1994

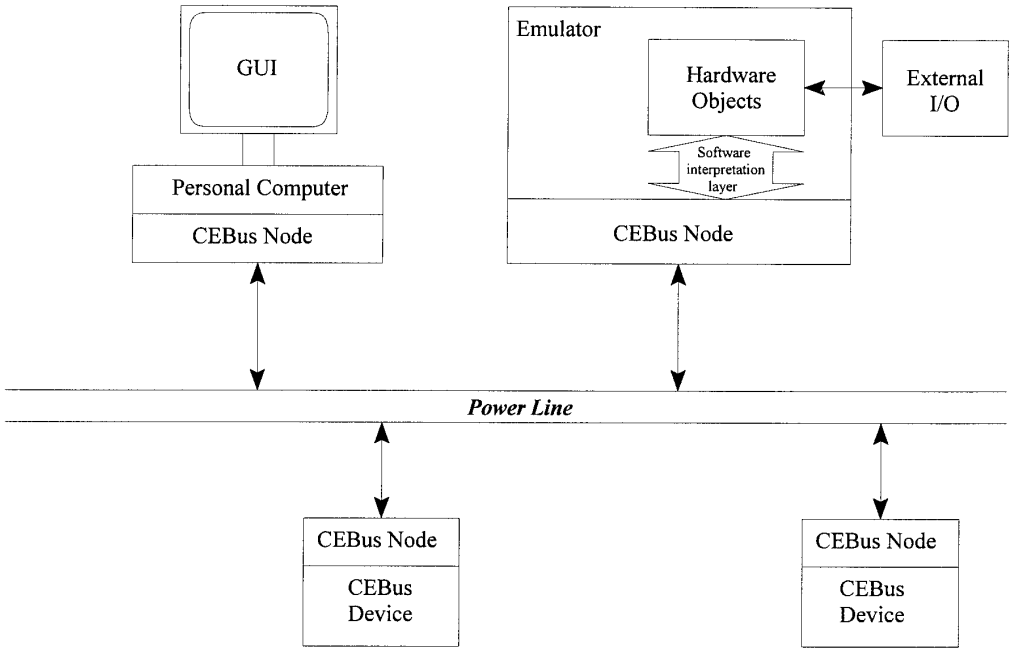


Figure 2: Emulator System Overview

4. CEBus Network

Fully functional CEBus nodes were constructed according to CEBus protocols [3],[4]. However, the data transmission rate over the power line medium has been reduced to 2 kBit/sec in order to comply with EU legislation concerning electromagnetic compatibility [5]. Each node consists of an LC-filter mains interface, a spread-spectrum power line modem and an 8-bit RISC microcontroller. The line interface and modem IC handle the CEBus physical layer and some elements of the Data Link Layer (DLL). The remaining CEBus protocol layers, up to and including the Network Layer (NL), are implemented by the RISC microcontroller.

5. Graphical User Interface (GUI)

The GUI is designed to provide User-friendly access to the Application Layer functionality of CEBus through the use of Windows-based software. The GUI provides full access to the Instance Variables (IVs) of each Object on the emulator, including the ability to report changes in state of IVs on the remote emulator. A library of generic Objects has been generated on the PC, and such Objects can be grouped into Contexts using the GUI. These Contexts are then physically implemented on the remote hardware emulator.

In order to emulate a particular device, the User simply 'creates' the various Objects that represent the device, and places them in the appropriate Context. A graphical representation of each Object created appears on the GUI.

Each Object has an associated Object model stored in memory. This Object model contains the Object IVs, and any other Object-related data (e.g. unique Object ID, Object Class, Context, etc.). Object models are updated by input from the GUI and from the remote emulator. These models can be considered to be operated upon by CAL Methods. Similar models reside on the emulator, allowing a *virtual* software representation of each actual Object on the emulator. Both sets of models are continually updated over the CEBus network. The structure of the GUI software is shown in Figure 3.

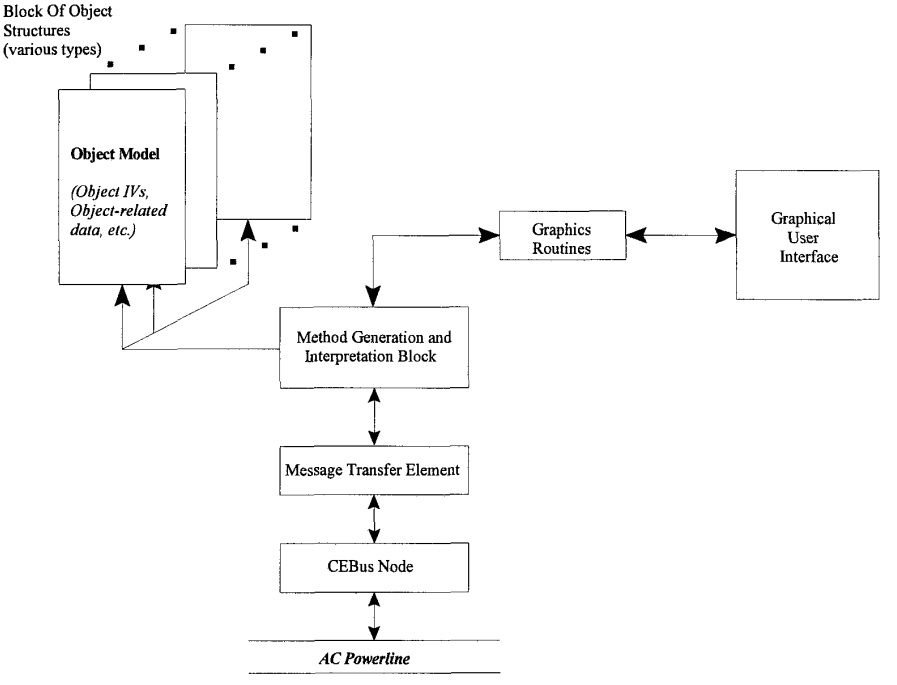


Figure 3: Software structure for PC-based CEBus GUI implementation

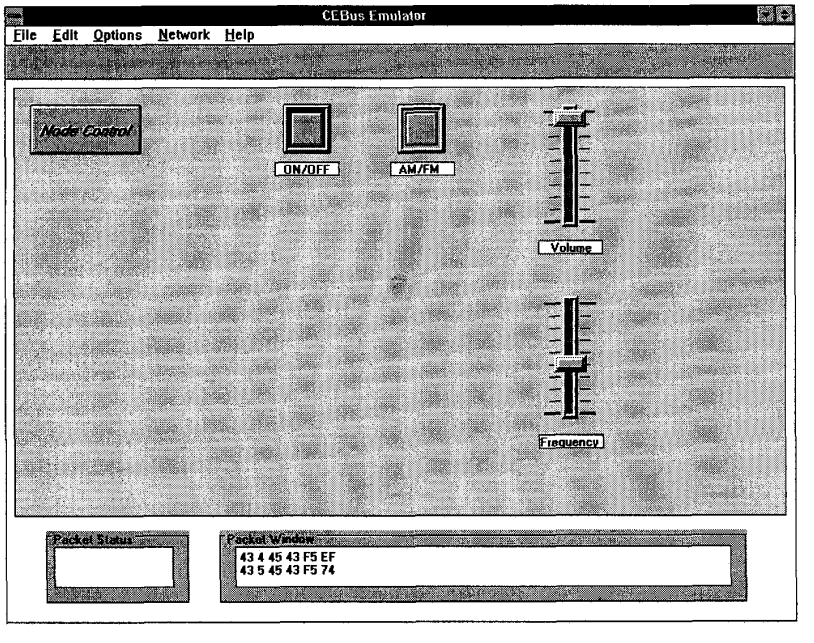


Figure 4: Graphical User Interface, configured as a simple radio receiver

Objects can also be removed from the GUI, deleting the Object model in memory, and releasing the corresponding hardware Object on the emulator. This Object is now free to be associated with another virtual Object on the GUI.

The GUI, configured as a simple CEBus device, is shown in Figure 4. In this case, a radio receiver is being emulated. This is achieved by creating the Objects associated with a simple receiver (i.e. 2 Analog Controls and 2 Binary Switches). The Objects are then placed in the Audio Context*. Actual hardware Objects on the emulator are configured to represent the radio receiver, as described above.

Multiple instances of an Object may exist within a Context, and across Contexts. Once an Object has been created, the User has access to the IVs of that Object by "clicking" on the graphical representation of the Object on the GUI. A property window appears, enabling the User to change certain IVs (Read/Write) or see the values of other IVs (Read only). A typical property window associated with an Object (in this case an Analog Control Object), is shown in Figure 5.

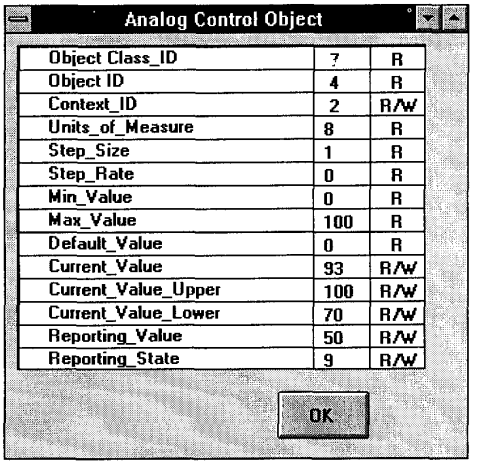


Figure 5: Property window of typical CEBus Object, displayed on GUI.

* GUI allows User to place Objects in any of the currently defined Contexts. The emulator does not differentiate between Contexts as far as the remote Objects are concerned.

The GUI may be used to provide remote access to any CEBus device on the Network. This is achieved by simply creating a software representation of the device on the GUI, and changing the Destination Address [4] of the CEBus packets to that of the desired CEBus device. The GUI can then be used to address individual Objects on the device.

6. Emulator Design

6.1 Emulator Hardware

The emulator consists of two primary subsystems: (i) a CEBus node handles the lower protocol layers; (ii) the actual emulator functionality is implemented using a separate embedded microcontroller subsystem. The Application Layer including the CAL implementation resides in the second subsystem which is based around a conventional 16-bit microcontroller module and an EPROM-based operating system. This is interfaced with a variety of external hardware including A/D channels, D/A channels, a range of digital I/O, switches, panel meters and more specialised components including a keypad and an LCD display. Object IVs and other Object-related data are stored in RAM on the microcontroller.

6.2 Emulator Software

The emulator software is responsible for maintaining accurate models of the CEBus Objects available on the Emulator. The structure of the emulator software is given in Figure 6. Object models are created in memory, and these models are updated by input from the CEBus network and from changes to the actual Objects themselves. The software monitors all input/output channels for changes in state of any Object and updates the corresponding Object model accordingly. The emulator software informs the GUI of any changes to the Objects by originating messages on the CEBus network. This allows the PC software to update its Object models and also update the GUI, if necessary. Incoming messages from The PC-based GUI are also interpreted, allowing the User to alter the Read/Write IVs of any Object on the Emulator.

Note that certain Read/Write IVs are not fully bi-directional. Consider a Binary Switch on the remote emulator. The Switch may have its physical state changed and the GUI will be updated to match, but it is not possible to change the physical state of the Switch directly from the GUI. These Uni-directional Read/Write IVs can be considered to behave like Read-Only IVs.

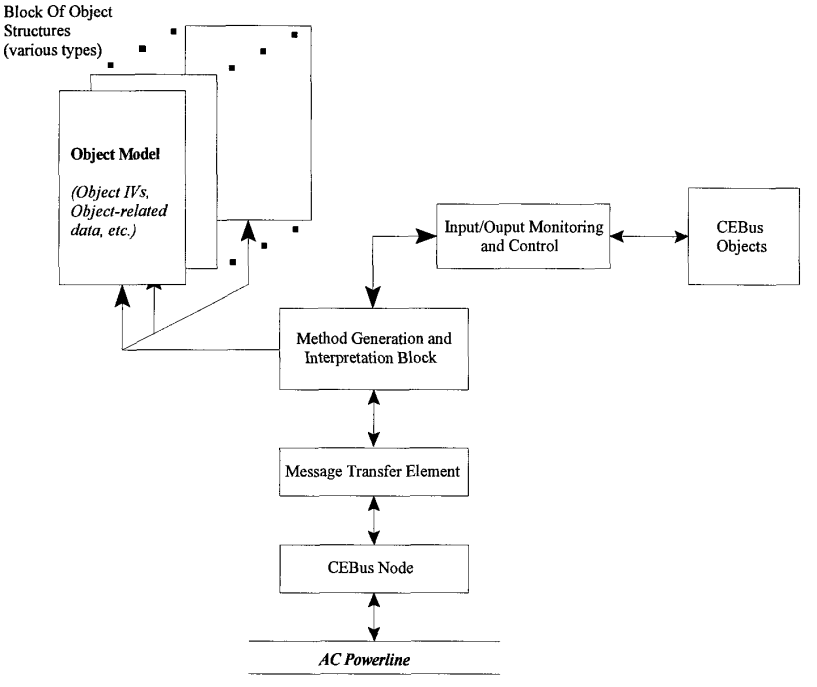


Figure 6: Software structure on remote emulator (Note similarities to PC-based structure)

The Node Control Object does not model any particular control on a CEBus device. Instead, it contains information about the device such as device address, product information, etc. Thus, the Node Control Object on the emulator is implemented entirely in RAM. One instance of the Node Control Object always exists on the emulator.

7. Conclusions

A remote Object emulator for CEBus applications has been developed. The emulator may be configured to represent a variety of common household devices. Access to the emulator is through a User-friendly GUI, giving access to individual Objects on the emulator. The entire system resides on a CEBus network over AC power line.

The system in its present form has many applications in CEBus system development. These include application for training and educational purposes. Much of the motivation behind the development of this system was to provide a demonstration of the functionality and application potential of CEBus communications. The authors feel that there has been a great deal of simulation work in this area [e.g. 6,7,8], but practical tools to demonstrate and develop working CEBus applications are still in short supply.

Further planned enhancements for the PC-based GUI include support for multiple devices on a CEBus network, and the development of a CAL 'Compiler' to allow the User to create and run programs for a CEBus network using CAL.

References

[1] Hofmann, J. " The Consumer Electronic Bus : An Integrated Multi-Media LAN for the Home", International Journal of Digital and Analog Communication Systems, Vol. 4, Issue 2, Apr. 1991, pp. 77-86.

[2] EIA Home Automation System (CEBus) Interim Standard IS-60, Vol. 8, EIA, Jun. 29, 1992.

[3] EIA Home Automation System (CEBus) Interim Standard IS-60, Vol. 1, Parts 1,7,8, EIA, Jun 29, 1992.

[4] EIA Home Automation System (CEBus) Interim Standard IS-60, Vol. 4, EIA, Jun 29, 1992.

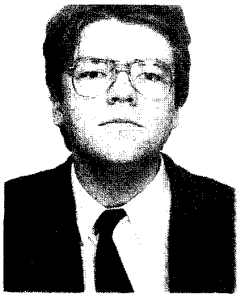
[5] Signalling on low-voltage electrical installations in the frequency range 3 kHz to 148.5 kHz, EN 50065-1, Part 1, CENELEC, 1991.

[6] Pakkam, S.R. and Manikopoulos, C. N. "Performance Evaluation of the Consumer Electronic Bus", IEEE Transactions on Consumer Electronics, Vol. 36, No. 4, Nov. 1990, pp 949-953.

[7] Yang, J. and Manikopoulos, C. N. "Theoretical analysis of the CEBus with Three priorities", IEEE Transactions on Consumer Electronics, Vol. 39, No. 4, Nov. 1993, pp 816-823.

[8] Yang, J. and Manikopoulos, C. N. "Performance Comparisons of the CEBus with other Protocols", IEEE Transactions on Consumer Electronics, Vol. 39, No. 4, Nov. 1993, pp 824-831.

Biographies



Peter Corcoran received the BAI (Electronic Engineering) and BA (Maths) degrees from Trinity College Dublin in 1984. He continued his studies at TCD and was awarded a Ph.D. in Electronic Engineering in 1987 for research work in the field of Dielectric Liquids. In 1986 he was appointed to a lectureship in Electronic Engineering at UCG. His research interests include microprocessor applications, environmental monitoring technologies, and automated testing of electronic components and equipment. He is a member of I.E.E.E.

Karl Lusted received his BE degree in Electronics from University College Galway in 1992. He is presently studying for the degree of M.Eng.Sc at UCG. His research interests include microcontroller applications and communications network protocols. He is a member of I.E.I. and an associate member of I.E.E.