Fall 10-31-1994

CEBus demonstration system

Liang-Chuang Chen
New Jersey Institute of Technology

ABSTRACT

CEBUS DEMONSTRATION SYSTEM

by
Liang-Chuang Chen

A CEBus demonstration system is constructed and applied to aspects of home automation security. Two M68HC16Z1 evaluation boards are used as CEBus nodes. Two CEBus spread spectrum power line modems (SSC PLCEMS) interconnect the nodes. An SSC PLCEMS handles all spread spectrum signal generation and reception.

The application programs in the two boards are written in assembly language based upon CAL (Common Application Language), a CEBus standard.

The standard CEBus frame format and FCS (Frame Check Sequence) are implemented by software in the Data Link Layer. Encoding and decoding binary information into UST format are implemented in the Physical Layer.

The system responsiveness of 30 millisec was measured experimentally for a case with 3 information bytes. The noise sensitivity was measured experimentally with a straight 3-wire connection of the modems with and without AC power. The distorted AC line caused a packet error rate of 2%, a higher rate than obtained by injecting broadband random noise.

CEBUS DEMONSTRATION SYSTEM

by
Liang-Chuang Chen

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfilment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

October 1994

APPROVAL PAGE

CEBUS DEMONSTRATION SYSTEM

Liang-Chuang Chen

Dr. Anthony Robbi, Thesis Co-Advisor Associate Professor of Electrical and Computer Engineering, NJIT	Date
Dr. C. Manikopoulos, Thesis Co-Advisor Associate Professor of Electrical and Computer Engineering, NJIT	Date
Dr. John Carpinelli, Committee Member Associate Professor of Electrical and Computer Engineering, NJIT	Date

BIOGRAPHICAL SKETCH

Author:	Liang-Chuang Chen
Degree:	Master of Science in Electrical Engineering
Date:	October 1994

Undergraduate and Graduate Education

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology,
  Newark, New Jersey, 1994

- Bachelor of Science in ElectricalEngineering,
  Chung-Yung University,
  Taiwan, ROC, 1988

Major: Electrical Engineering

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to his advisors, Professor Anthony Robbi, and Professor Manikopoulos for his guidance, friendship, and moral support throught this research.

Special thanks to Professors Carpinelli for serving as members of the committee.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 CEBus: A New Standard in Home Automation

In 1984, EIA (Electronic Industries Association) formed the CEBus (Consumer Electronic Bus) committee to develop a standard to facilitate communications between various home automation devices and appliances. The committee was made up of such major corporations as Sony, Philips, Panasonic, General Instrument, AT&T, Texas Instruments, Mitsubishi, RCA, and Johnson Controls. It started out with the goal of unifying infrared hand-held controllers in an effort to reduce the jungle of remotes found in many entertainment rooms. The committee quickly discovered that it made sense to extend the standard to whole house communications over assorted media.

According to EIA documents, the committee had five primary goals for the CEBus: It would be retrofittable, use distributed intelligence (have no central computer in order to operate), be non-product specific, have an open architecture, and be expandable.

CEBus isn't actually a bus, but a network specification. It follows the ISO/OSI (International Organization for Standardization / Open System Interconnection) seven-layer network model which defines the physical, data link, network, transport, session, presentation, and application layers. Each layer is responsible for one aspect of network communication, with each layer only able to talk to the layers directly above and below it. For example, the physical layer is only concerned with getting bits from one node to another, without regard for what the bits mean or even whether they make it from one node to the next error free (error detection and correction are handled by the data link layer which is one level higher).

By breaking the network model into well defined pieces, implementation and support are greatly simplified. It is also possible for one company to implement specific layers, with another company implementing the rest. The two implementations communicate through a well-defined boundary between the layers.

1.2 OSI Reference Model Layers for CEBus

1.2.1 Physical Layer

At the lowest level is the physical layer. This is where CEBus's greatest strengths lie since several different media are defined in the specifications with the choice of which medium to use up to the appliance designer. All the layers above the physical layer are identical regardless of medium, so the network is medium independent[1].

Signaling is done on most of the media by switching between a "superior" state and an "inferior" state. Times between changes determine the information being conveyed.

"One" bits last one "Unit Symbol Time" (UST), "zero" bits last two USTs, end-of-field markers last three USTs, and end-of-packet markers last four USTs. Exactly what defines the superior and inferior states depends on the medium. Characterizing communication speed for a CEBus medium in bits per second is meaningless since one bits and zero bits are of different durations. Thus, CEBus data rates are defined in terms of "one bits per second." Statistically, the overall throughput in bits per second is around two-thirds the value of one bits per second.

1.2.2 Data Link Layer

The next highest OSI level is the data link layer. It is primarily responsible for providing a clean channel of communication for the higher levels. To do this, it must handle collision prevention, detection, and resolution; packet acknowledgment; and final packet construction.

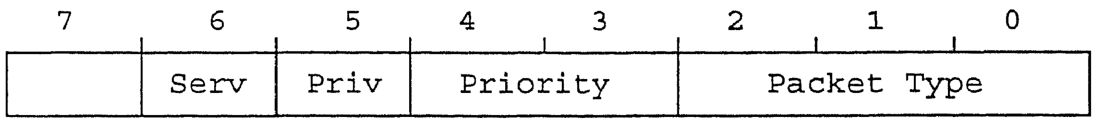
Collision prevention, detection, and resolution is handled using CSMA/CDCR (Carrier Sense, Multiple Access/with Collision Detection and Collision Resolution). Since all nodes are connected to a common medium with no master node dictating who may transmit when, there always exists the possibility that two nodes may try transmitting at the same time. Since it is always best to avoid such a situation, collision prevention is tried first[2].

Before transmitting, each node listens to the network to determine if another node is already transmitting. If so, it simply waits for the transmitting node to finish. When the network is free, the node waits a certain amount of time before trying to transmit. The time it waits is a fixed delay dependent on the packet's priority (high priority packets are tried sooner than low priority packets) plus a random channel access delay. Without the randomizing factor, if two nodes sense the network free at the same time and have equal priority packets to send, they would try transmitting at the same time, resulting in a collision.

When a node determines to start transmitting, it starts by sending out a preamble character. The preamble is a random number designed to be a "sacrifical lamb." The transmitting node listens as it sends out the preamble, and if the preamble survives intact, the rest of the packet is sent. If a collision is detected (another node sending a different preamble), transmission is aborted and the process starts again.

Depending on the packet type, an immediate acknowledgement may be requested by the sending node. After transmission is complete, but before the sending node gives up the communications channel, the receiving node will send an acknowledgment back to the sender. If the sender doesn't receive the acknowledgment before timing out, the packet is retransmitted once. If there is still a problem, a higher network layer decides what to do.

The byte sent after the preamble is the Logical Link Control Protocol Data Unit, or LPDU (Figure 1.1). It contains the packet type (which determines acknowledged or unacknowledged service and local or nonlocal medium), packet priority, privilege, and basic or extended service.



Service Class	
0	Basic Service
1	Extended Service

Privilege	
0	Nonprivileged LPDU
1	Privileged LPDU

Priority	
00	High
01	Standard
10	Deferred

Packet Type	
000	Acknowledged Response
001	Local Data Acknowledged
010	Data Unacknowledged
011	Nonlocal Data Acknowledged
100	Failure Response

Figure 1.1 LPDU structure

Following the LPDU are the destination address, destination house code, source address, and source house code. Each node is identified by both a unit number and a house code. It's possible to have several independent networks using the same media by assigning different house codes to different groups of devices. The house code is also used to avoid conflicts with neighbors.

Both destination and source addresses are sent so the receiving node knows the sender's address. For example, there may be several TVs and a single VCR connected to the same bus. One of the TVs may send a request to a VCR to start playing a tape, but the VCR must first send a command back to the TV to change to an appropriate channel. Knowing the address of the TV which sent the play command, it's possible for the VCR to tell the correct TV to change channel.

Packet data follow the addresses. The information comes from the higher network levels and will be discussed in more detail later.

The final field contains the Frame Check Sequence. It is simply an 8-bit checksum of all the bits in the packet excluding the preamble[3].

1.2.3 Network Layer

The network layer is responsible for determining which media are to receive the packet and for breaking apart packets which would exceed the 32-byte limit.

The Network Protocol Data Unit (NPDU) is added to the front of the information field passed down by the upper levels. See Figure 1.2. There are six bit fields which determine which media are to receive the packet. Setting a bit in the field results in the corresponding medium receiving the packet (assuming the proper bridge is present to transfer packets across media). The last two bits determine whether the packet is to be sent using flood routing, directory routine, or directory routing with a request for a return ID, and whether it is being segmented. The application layer breaks long messages that can not be contained in a single packet. If segmentation is requested, the NPDU header contains the segment number of the current packet.

7	6	5	4	3	2	1	0
Type		RF	SR	FO	CX	TP	PL

NPDU Type

11	Flood Routing, Unsegmented
10	Directory Routing, Unsegmented
01	Directory Routing, Return ID, Unsegmented
00	Segmented Variable Header

Figure 1.2 NPDU structure

1.2.4 Transport, Session, and Presentation Layers

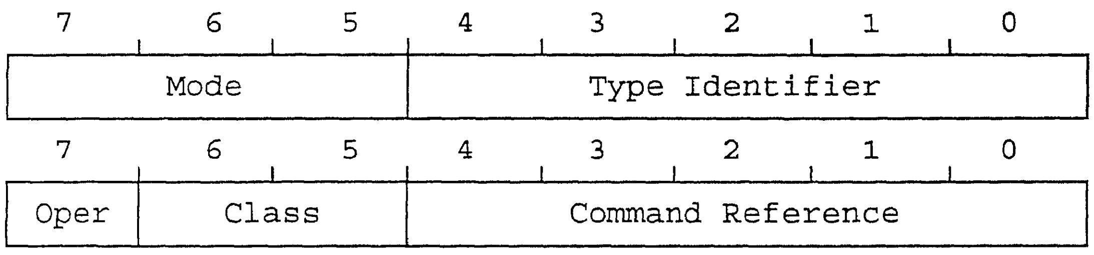
Since the OSI seven-layer network model was designed to be useful in just about any application, there is bound to be some fat that can be trimmed while implementing applications that don't require all the facilities or segmentation defined in the model. Such is the case in the definition of CEBus. The functions of the transport, session, and presentation layers as defined in the OSI model are handled by the application, network and data link layers in the CEBus definition. This doesn't mean that the OSI model isn't being followed or that corners are being cut. Certain facilities found in larger networks just don't exist in a simple control network.

1.2.5 Application Layer

The highest OSI level is the application layer and it is responsible for the end user functions. In the case of CEBus, the highest level defined isn't necessarily what the end user will see (because, in many cases, operation will be transparent or part of a device existing functionality), but what the programmer sees. EIA has defined CAL (Common Application Language) to allow CEBus devices to communicate intelligently with each other[3].

A header similar to those found in the lower layers is added to the front of the CAL command before being passed along. It is called the Application Protocol Data Unit (APDU). The APDU may be up to 3,810 bytes long, but only the first two bytes have been defined at present (Figure 1.3). The first byte contains the mode information and type identifier. The mode specifies the service class, header type and data field length for the command which follows. The service class may be either basic or privileged, though most commands in use at this time are basic. The header may be either fixed or variable in length, with fixed-length headers being the norm. Finally, the command length may be either short (up to 32 bytes), long (up to 3,808 bytes), or huge (up to 1,638,375 bytes), including data.

The type identifier determines whether a command is implicit or explicit, and defines the response code for an explicit command. An implicit command doesn't require a response, so is simpler to program and faster to send, but is subject to errors since the destination node does not respond. An explicit command requires a response from the destination node, with the response either a result, reject, or error code.



Mode

000	Basic service, Fixed header, Short data
001	Privileged service, Fixed header, Short data
010	Basic service, Variable header, Long data
011	Privileged service, Variable header, Long data
100	Basic service, Variable header, Huge data

Type ID

01011	Explicit Association Invoke
01100	Reject
01101	Error
01110	Result
01111	Implicit Association Invoke

Operation

0	Asynchronous Transmission
1	Synchronous Transmission

Class

00	No response requested
01	Error or Rejected requested
10	Result requested
11	Result, Error, or Reject requested

Command Ref
User-definable codes (except 00000)

Figure 1.3 APDU structure

The second byte in the APDU determines whether transmission will be synchronous or asynchronous, and what kind of response is desired in the case of an explicit command.

CAL is made up of three sections, with each section more precisely defining just what the final action should be. The first section is the context and defines general categories of devices. For example, there are categories already defined for an audio process (amplifiers, loudspeakers, equalizers, etc.), video monitor, communication control system (telephone, radio, etc.), time service element (real-time clock, timer start and stop times, etc.), environment management system, lighting system, etc.

The second section is the Specific Application Service Element, or SASE. The SASE defines the primary function of the command sequence. Each context has a list of SASEs defined, and are often similar or identical across different contexts. Examples of SASEs for, say, an audio process are primary mode switch (power), source switch (radio, CD, or tape), feature switch (noise reduction, surround sound), and level controls (volume, bass, treble, balance).

Finally the Common Application Service Element (CASE) defines just what the final action should be. CASEs are the same for all contexts and SASEs. Example CASEs include true, false, add, subtract, and load.

Using the context, SASE, and CASE, it's possible to create commands to do just about any function you can think of. EIA has tables of predefined contexts, SASEs, and CASEs, so most of the time it's just necessary to look up a command in the table and use it. Manufacturers who want to add commands can follow the rules for CAL and develop new commands. Escape codes have been put in place to allow unlimited extension of CAL commands should the main tables ever fill up. EIA presumably will have ultimate control over the tables and the commands that go in them. Possibilities exist for manufacturers to implement proprietary command sequences, but it's unlikely such a product would win any sort of official CEBus-compatible approval.

1.3 CEBus Specification

The CEBus specification defines six media which may be used to carry the signal: PLBus (Power Line Bus), SRBus (Single-Room Bus, or infrared), RFBus (Radio Frequency Bus), TPBus (Twisted-Pair Bus), CXBus (CoaX Bus), and FOBus (Fiber-Optic Bus), the last three of which are often collectively referred to as WIBus, or WIred Bus.

1.3.1 PLBus

PLBus is likely to be the medium of choice for most appliances meant for retrofit installations since almost every house and business in the world is wired for electricity. Since the power line is such a harsh environment, with noise and transients the norm, this is the slowest of all the media, but is still able to attain a rate of 1000 one bits per second with a UST of 1 ms.

Transmissions use a 120-kHz carrier to denote a superior state and the lack of a signal for an inferior state. Unlike the X-10 system which transmits only at the 60-Hz zero crossing, PLBus transmits regardless of the state of the AC power on the line. As a result, transmission can still take place even if power isn't present, something that can't be done with X-10. Even though both PLBus and X-10 use 120-kHz carriers, the two systems are completely incompatible and, indeed, interfere with one another[1].

1.4 Thesis Description

The purpose of this work is to build a CEBus demonstration system applied to home automation security. Two MC68HC16Z1 evaluation boards are used as CEBus nodes. See Figure 1.4 for the layout. One node is for a simulated living room, LR, and the other node is for a simulated bedroom, BR. It is supposed that there are 3 lights, one in the living room, another in the bedroom and the other outside the house, and 2 windows, one in the living room and the other in the bedroom.

There are two operating modes, normal and vacation. The microcontroller (LR node) always sends the status of all devices and the updated time by its SCI to a STATUS PC independent of the operating mode. The system state diagram is shown in Figure 1.5.

In the normal mode, all devices are controlled by switches and buttons as shown in Figure 1.4. When

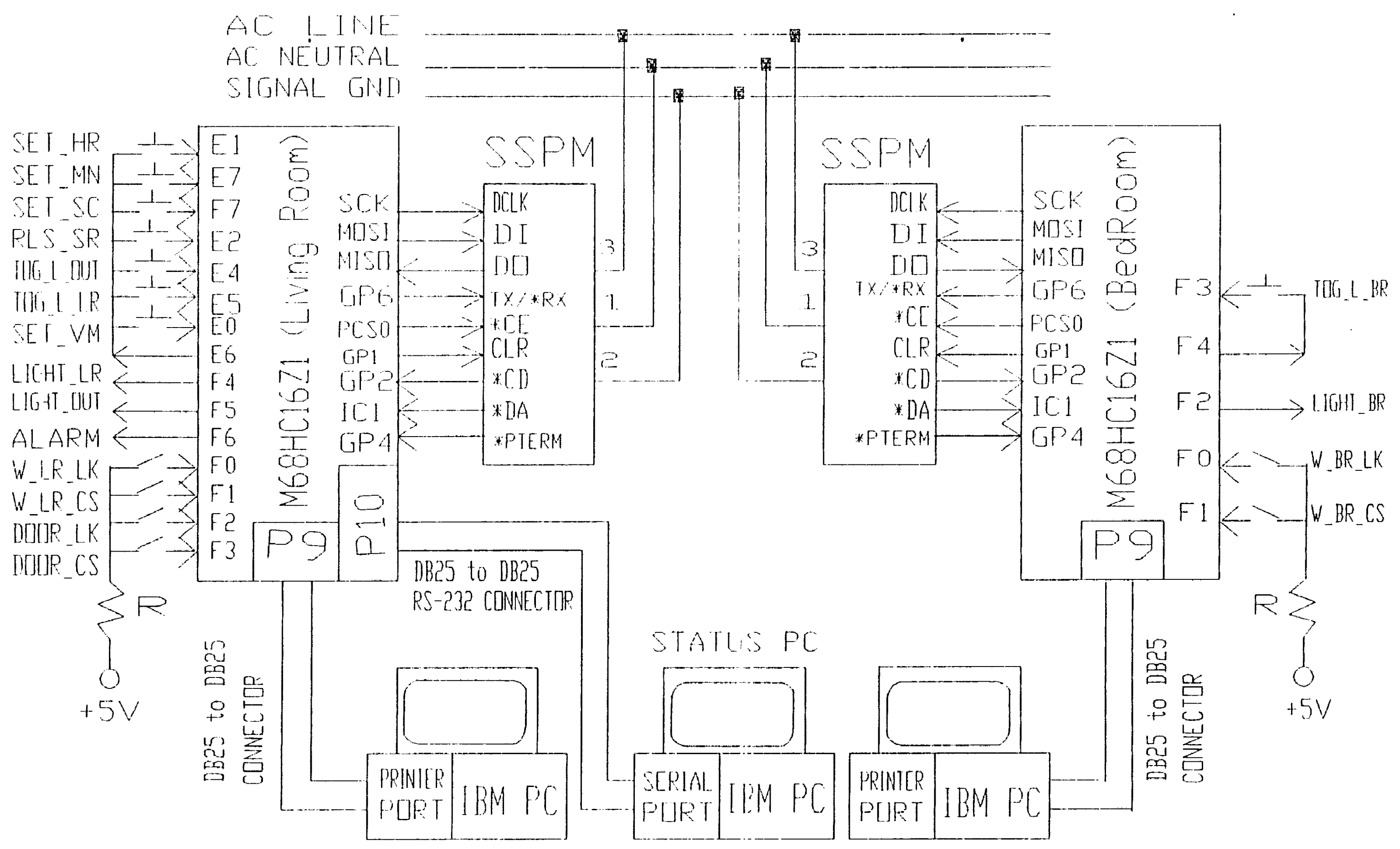


Figure 1.4 System configuration

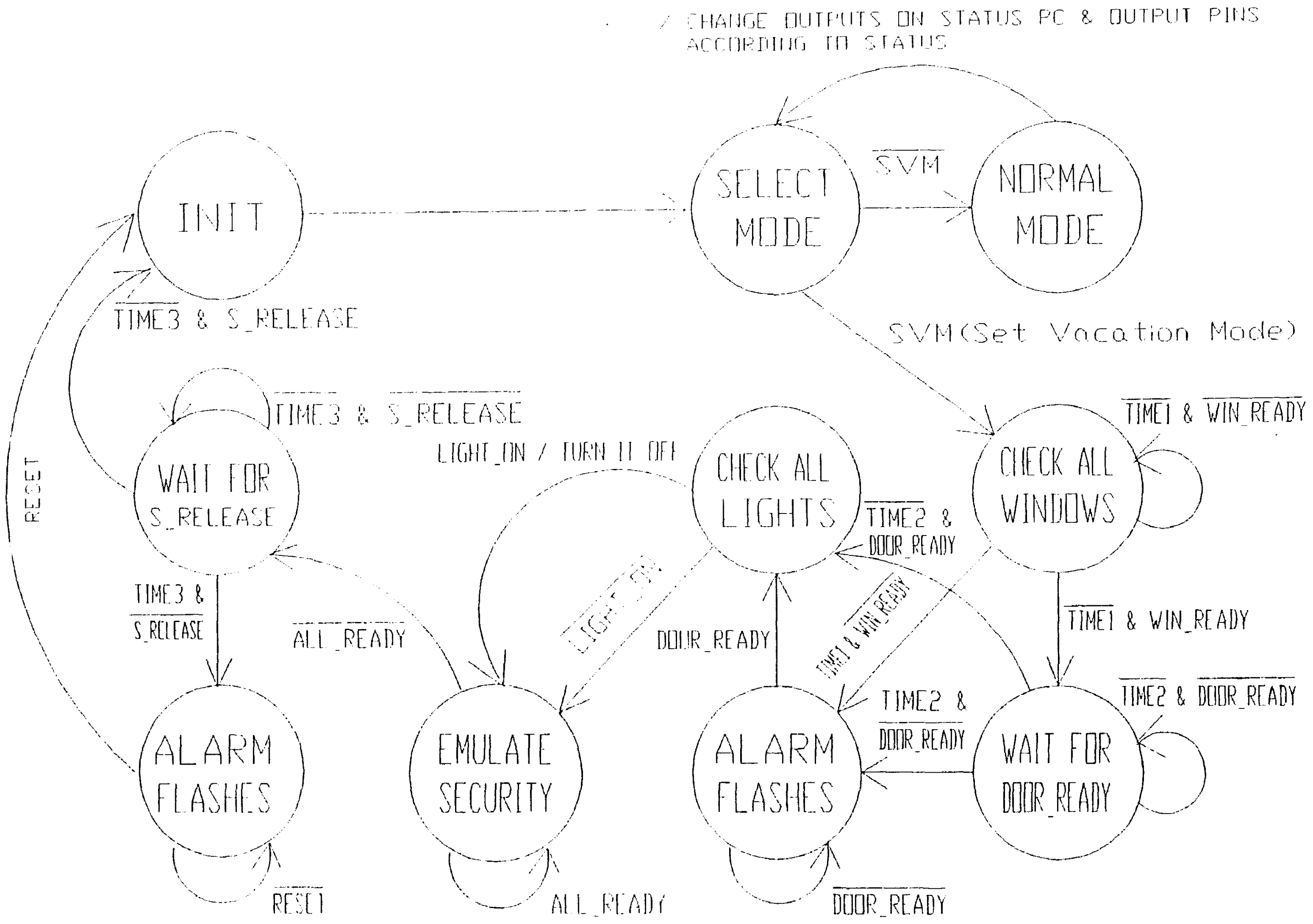

/ CHANGE OUTPUTS ON STATUS PC & OUTPUT PINS ACCORDING TO STATUS
INIT
SELECT MODE
$\overline{\text{SVM}}$
NORMAL MODE
$\overline{\text{TIME3}}$ & S_RELEASE
SVM(Set Vacation Mode)
$\overline{\text{TIME3}}$ & $\overline{\text{S\_RELEASE}}$
WAIT FOR S_RELEASE
LIGHT_ON / TURN IT OFF
CHECK ALL LIGHTS
CHECK ALL WINDOWS
$\overline{\text{TIME1}}$ & $\overline{\text{WIN\_READY}}$
RESET
$\overline{\text{TIME2}}$ & $\overline{\text{DOOR\_READY}}$
TIME3 & $\overline{\text{S\_RELEASE}}$
$\overline{\text{ALL\_READY}}$
LIGHT_ON
DOOR_READY
TIME1 & WIN_READY
$\overline{\text{TIME1}}$ & WIN_READY
TIME2 & $\overline{\text{DOOR\_READY}}$
$\overline{\text{TIME2}}$ & $\overline{\text{DOOR\_READY}}$
ALARM FLASHES
EMULATE SECURITY
ALARM FLASHES
WAIT FOR DOOR_READY
$\overline{\text{RESET}}$
ALL_READY
$\overline{\text{DOOR\_READY}}$

Figure 1.5 State diagram

setting the house in the vacation mode, the microcontroller (LR node) makes sure conditions are proper. For example, if the homeowner does not close and lock an open door in a short time, the ALARM will flash to indicate improper conditions. When the system is in vacation mode, the system first turns off all lights automatically. Later, the system takes suitable actions according to the time. The actions emulate that someone is in the house so as to enhance security.

1.5 Related Work

CyberLYNX Computer Products, working with Texas Instruments, and AISI Research Corp. are developing a single-chip CEBus interface that will handle all the details of CEBus communications. Though the chip is still in its early stages, CyberLYNX has a CEBus evaluation board that implements many of the CEBus functions in firmware. It also includes a power line interface so several boards can communicate[1].

AISI Research Corp. has implemented a CEBus demonstration. The system diagram is shown in Figure 1.6. The AISI SPIRIT chip has eight discrete inputs, eight outputs, microprocessor control lines, and the requisite CEBus input and output connections. In its simplest configuration, SPIRIT is capable of monitoring network communications and changing output bits based upon received commands, and monitoring input bits and sending out commands based on those inputs. The chip contains several hundred bytes of EEPROM and is trained before hand with which CAL commands to watch for, what outputs to change upon receipt of those commands, what inputs to monitor, and what commands to send out in response to those inputs. Once trained, the chip plus any interface circuitry will operate stand-alone.

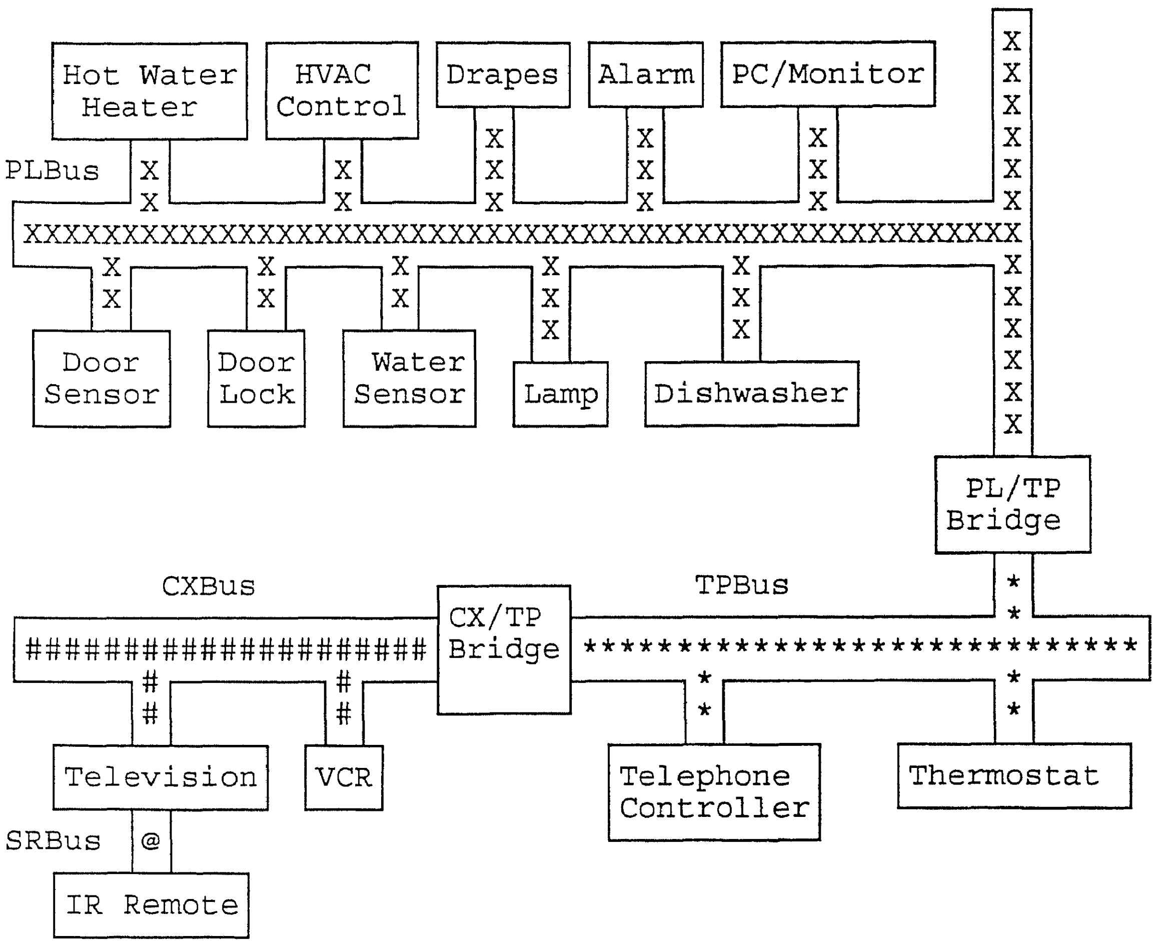


Figure 1.6 AISI demonstration system diagram

In cases where additional processor power is necessary, SPIRIT can also be connected serially, or to the microprocessor's data bus. The chip's hardware has been designed to allow numerous modes of operation depending on how it is hooked up.

The thesis work forms a more flexible foundation for a CEBus test bed - more I/O and possible use of a high level language at the application level. The microcontrollers use in the nodes have real time I/O capabilities which could be used in relatively sophisticated ways. Examples are light dimming and the emulation of infra-red remote controls for video and audio equipment.

CHAPTER 2

SYSTEM DESCRIPTION

2.1 System Configuration and Operation

As shown in Figure 1.4, two M68HC16Z1 evaluation boards are used as CEBus nodes. Two CEBus spread spectrum power line modems (SSC PLCEMS) interconnect the nodes. The SSC PLCEMS is used to handle all spread spectrum signal generation and reception. It also does preamble detection and stripping on reception, and it provides for all data synchronization and timing. The SSC PLCEMS handle portions of the CEBus Physical Layer Symbol Encoding (PLSE) sub-layer, including complete Cyclic Redundancy Codes (CRC) generation and detection, and portions of the Medium Access Control (MAC) sub-layer. SSC PLCEMS are ideal for adding power line communications to existing systems or for development of new products in which reliable power line communications are desired[4]. The evaluation boards connect to the modems using their SPI synchronous serial ports as shown in Figure 1.4.

The two M68HC16Z1 nodes are connected to host computers via user-supplied 25-conductor cable assemblies. One end of the cable assembly needs a female DB25 connector; this end of the cable connects to the EVB parallel port (connector P9) in Figure 1.4. The other end of this cable assembly needs a male DB25 connector; this end of the cable connects to the parallel printer port of the PC[5]. The two host computers are used to download the application program to each EVB from PC printer port to EVB PC printer port (connector P9). EVB16 software runs on the PC to load the application program[5].

The LR M68HC16Z1 node is connected to a RS-232C host computer port via a separate 25-conductor cable assembly. One end of the cable assembly needs a male DB25 connector; this end of the cable connects to the EVB user interface port (connector P10). The other end of the cable assembly needs the appropriate connector for the RS-232C compatible port of the PC, as shown in Figure 1.4. All of the messages are from the LR EVB user interface port (connector P10) to PC COM1 or COM2. The display program uses the RS-232 protocol to show all of the status messages. The parameters are set to 8 data bits, 1 start bit, 1 stop bit, no parity check, and 9600 bps.

The display program clears the STATUS PC screen, shows a framework for the system status, and waits for messages from the LR EVB. As part of its application level task, the LR node sends system status information to the STATUS PC, as detailed later. It does not matter which EVB application program is downloaded first.

2.2 I/O Hardware

LEDs are used to show the status of all lights: outside (LIGHT_OUT), living room (LIGHT_LR), bedroom (LIGHT_BR), and alarm (ALARM).

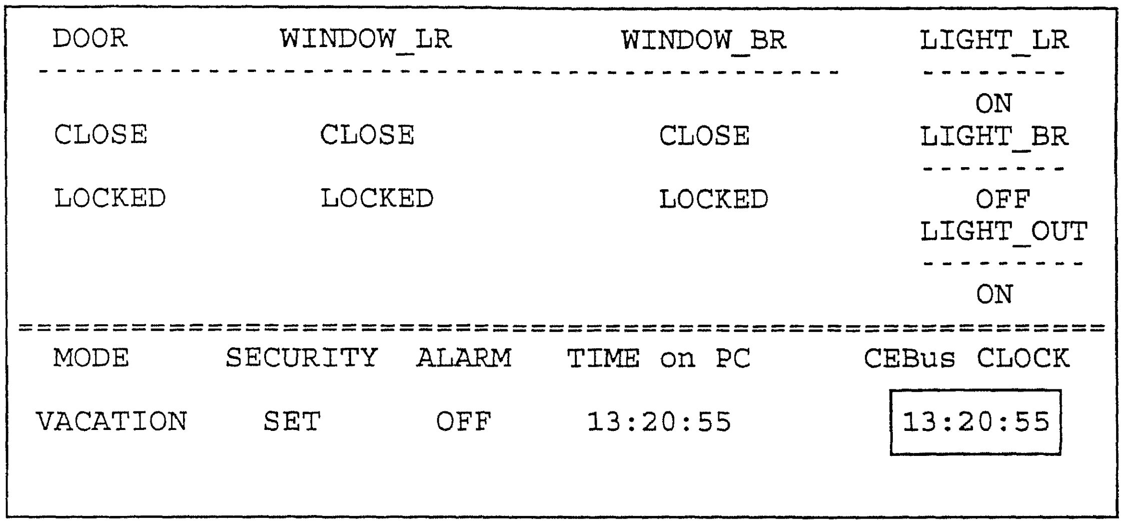
Switches are used to simulate condition detectors in a home as shown in Figure 2.1.

location	switch is ON	switch is OFF
W_LR_LK	locked	unlocked
W_LR_CS	closed	open
W_BR_LK	locked	unlocked
W_BR_CS	closed	open
DOOR_LK	locked	unlocked
DOOR_CS	closed	open

Figure 2.1 The relation between switches and devices

Buttons are used to change the status of all lights, security and mode. See Figures 1.4 and 1.5. For example, if the homeowner presses the TOG_L_OUT button once, the application program will complement the status of the outside light.

There are 4 LEDs on the boards standing for LIGHT_LR, LIGHT_BR, LIGHT_OUT and ALARM. Because the system is for demonstration, 7-segment LEDs and other LEDs which could be on the boards in the real situation are simulated on the STATUS PC screen. For example, if the system is in the vacation mode, all devices are in the proper conditions, LIGHT_LR and LIGHT_OUT are turned on automatically, and the time is 13:20:55, the screen will appear as in Figure 2.2.



```
 DOOR         WINDOW_LR          WINDOW_BR        LIGHT_LR
------------------------------------------        --------
                                                     ON
 CLOSE        CLOSE              CLOSE            LIGHT_BR
                                                  --------
 LOCKED       LOCKED             LOCKED              OFF
                                                  LIGHT_OUT
                                                  ---------
                                                     ON
==========================================================
 MODE      SECURITY  ALARM   TIME on PC       CEBus CLOCK

VACATION     SET      OFF     13:20:55         13:20:55
```

Figure 2.2 The structure of demonstration screen

2.3 Function Description

2.3.1 Mode

There are two operating modes, normal and vacation. The default MODE is normal. When SET_VM button is pressed, the system is going to enter the vacation mode. It will check if every device is in the proper condition before the vacation mode is set, as show in Figure 1.5.

2.3.2 CEBus Real-Time Clock

A periodic interrupt routine makes the CLOCK change every second. SET_HR adds an hour to the CLOCK when it is pressed once. HR number is from 0 to 23. 0 shows up after 23. SET_MN adds a minute to the CLOCK when it button is pressed once. MN number is from 0 to 59. 0 shows up after 59. SET_SC adds a second to the CLOCK when it button is pressed once. SC number is from 0 to 59. 0 shows up after 59. The CLOCK can be changed by pressing the appropriate buttons.

2.3.3 A Starting Time for The CEBus Real-Time Clock

The TIME is obtained from PC. It can be used as a basis when the CEBus CLOCK is set up.

2.3.4 Alarm

The ALARM flashes in 2 situations. One of the situations is when the SET_VM button is pressed, but some devices are still in improper conditions in few minutes. The other is when someone invades, but SECURITY is not released in few minutes. The flashing frequency is 1 Hz.

2.3.5 Security

SECURITY is set automatically when the system is in the vacation mode and every device is in the proper condition. When SECURITY is on, the system will turn on or turn off LIGHT_LR, LIGHT_OUT or LIGHT_BR in a random time. When SECURITY is released, the system is in the normal mode. All devices are controlled by switches and buttons.

2.3.6 Light Control

When the system is in normal mode, all lights are controlled by TOG_L_OUT, TOG_L_LR, and TOG_L_BR. Every time they are pressed once, the relative light will switch the status from ON to OFF, or from OFF to ON. When the system is in vacation mode, all lights are controlled by the system. The system turns on or off every light at a random time.

CHAPTER 3

SYSTEM DESIGN

As described in the previous chapter, the system comprises two M68HC16Z1 evaluation boards, two power line modems, and three computers which are Intel 8088-based system. Two of the computers are used to run monitor programs for the evaluation boards. A third computer is used to show messages from the two boards. A major work in this system is the implementation of the QSPI (Queue Serial Peripheral Interface) channel and SCI (Serial Communication Interface) for data and command transfer.

3.1 System Initialization

```
LDD     #$0003      ; at reset, the CSBOOT block size
                    ; is 512k. these instructions set
STD     CSBARBT     ; block size to 64k since that is
                    ; what physically comes with EVB16
```

A base address is the starting address for the block enabled by a given chip select. Block size determines the extent of the block above the base address. Each chip select has an associated base register so that an efficient address map can be constructed for each application. The BLKSZ (Block

Size) field in CSBARBT (Chip-Select Base Address Register Boot ROM) is set to $03 for block size 64K[6].

```
        LDAA    #$7F        ; w=0, x=1, y=111111
        STAA    SYNCR       ; set system clock to 16.78 MHz
```

When the on-chip clock synthesizer is used, system clock frequency is controlled by the W, X, and Y bits in the upper byte of SYNCR (Clock Synthesizer Control Register). W and X are 1 bit each, and Y is a 6 bit field. Bits in the lower byte show status of or control operation of internal and external clocks.

```
System Clock = (Reference Frequency)[4(Y+1)(2^(2*W+X))]
             = (32.768 KHz)[4(63+1)(2^(2*0+1))]
             = 16.78 MHz
```

```
        CLR     SYPCR       ; turn COP (software watchdog)
                            ; off, since COP is on after reset
```

SYPCR (System Protection Control Register) controls system monitor functions, software watchdog clock prescaling, and bus monitor timing. Clear SYPCR to disable software watchdog.

```
        LDD     #$0001
        STD     RAMBAH      ; store high ram array, bank 1
```

```
LDD     #$0000
STD     RAMBAL      ; store low ram array
CLR     RAMMCR      ; enable ram
LDAB    #$01        ; set SK to bank 1 for system
TBSK                ; stack. put SP at top of 1k
LDS     #$03FE      ; internal SRAM
```

The CPU16 in the MC68HC16Z1 operates in supervisory mode. Access to the SRAM array is controlled by the RASP (RAM Array Space) field in RAMMCR (RAM Module Configuration Register). SRAM responds to both program and data space accesses based on the value in the RASP field in RAMMCR. Internal SRAM is set to addresses $10000 - $103FF and stack is inside it[7].

```
ORG     $0070       ; put address of periodic
                    ; interrupt routine at 1st user
DC.W    VECRT       ; defined interrupt vector
LDD     #$0738      ; set the periodic interrupt at
STD     PICR        ; request level 7 & assign vector
                    ; #56 (address $00070) to it
LDD     #$0110      ; initialize PITR to interrupt
STD     PITR        ; every 1 sec
```

VECRT is the name of the interrupt routine. PICR (Periodic Interrupt Control Register) contains information concerning periodic interrupt priority and vectoring. Because the periodic interrupt request is assigned to level 7 & vector #56 (address $00070), PIRQ (Periodic Interrupt Request Level) is 7 and PIV (Periodic Interrupt Vector) is $38. PITR (Periodic Interrupt Timer Register) contains the count value for the periodic timer. The interrupt request for CEBus CLOCK happens every 1 second.

PIT Period = [(PITM)(Prescaler)(4)]/EXTAL

= [16*512*4]/32.768 KHz

= 1 sec

Where

PIT Period = Periodic interrupt timer period

PITM = Periodic interrupt timer register modulus

EXTAL = Crystal frequency

Prescaler = 512

```
ORG     $0080       ;Address for interrupt vector 64
DC.W    RECV        ;Input Capture 1
LDD     #$008E      ;Give the GPT an IARB of $E
STD     GPTMCR      ;generate interrupts, elevate
LDD     #$1640      ;interrupt priority of PAOV,
STD     ICR         ;set GPT IRQ level to 6,
                    ;& assign vector 64
LDAB    #$02        ;Input Captures
STAB    TCTL2       ;TIC1=FALL
```

IC1 (Input Capture 1) is used to detect *DA. When *DA goes low, it will produce an interrupt to receive 8 USTs from the modem[8].

3.2 QSPI and Data/Command Transfer

The QSPI is an intelligent, synchronous serial interface with 1 16-entry, full-duplex queue built into the M68HC16 microcontroller. It can continuously scan up to 16 independent peripherals and maintain a queue of the most recently acquired information with no central processor unit (CPU) intervention. The clock must be furnished as separate signal of the QSPI channel. Three QSPI signals are used: MISO, MOSI and SCK[9].

3.2.1 Master Mode

```
LDAA    #$08     ; set PSC0 high between serial
STAA    QPDR     ; transfers
LDAA    #$0B     ; assign Port D pins as PSC0,
STAA    QPAR     ; MOSI,MISO
LDAA    #$0E     ; set data direction as output on
STAA    QDDR     ; PCS0, MOSI, SCK pins
```

When operated in master mode, the QSPI may initiate serial transfers. MISO is used as the data input pin in master mode, and MOSI is used as the data output pin in master mode. SCK is the serial clock output in master mode[7]. The

QSPI does not respond to externally initiated serial transfers. QSM (Queued Serial Module) register QDDR should be written to direct the data flow on the QSPI pins used. The SCK pin should be configured as an output. Pins MOSI and PCS0/*SS should be configured as outputs as necessary. MISO should be configured as an input if necessary.

QSM register QPAR should be written to assign the necessary 8 bits to the QSPI. The pins necessary for master mode operation are MISO and MOSI, SCK, and one of the PCS pins.

```
        LDD     #$4000     ; set ENDQP to $0 for 1 serial
        STD     SPCR2      ; transfers, wraparound enable
```

The QSPI transmits the data found at the addresses $FFD21, and the QSPI stores received data at the addresses $FFD01. Data is transferred synchronously with the internally generated SCK.

```
        LDD     #$A320     ; set master mode, 8 bits per
        STD     SPCR0      ; transfer, clock polarity
                           ; inactive high, clock phase
                           ; change data on following edge,
                           ; baud 0.42 MHz
```

The number of bits transferred is determined by BITSE (Bits Per Transfer Enable) and BITS (Bits Per Transfer) fields. The system uses the default value of 8 bits because the power line modem allows only 8 bit transfers from the host to the SSC PLCEMS. The QSPI employs control bits, CPHA (Clock Phase) and CPOL (Clock Polarity), to determine which SCK edge the MISO pin uses to latch incoming data and which edge the MOSI pin uses to start driving the outgoing data. SPBR (Serial Clock Baud Rate) determines the baud rate of SCK.

```
SCK Baud Rate = System Clock / (2*SPBR)
              = 16.78 MHz / 2*20
              = 0.42 MHz
```

```
LDD     #$0202     ; set delay between PCS0 and SCK,
STD     SPCR1      ; set delay between transfers,
                   ; QSPI is disabled
```

DSCK (Delay before SCK) and DSCKL determine any peripheral chip-selects valid to SCK start delay. DT (Delay after Transfer) causes a delay to occur after the specified serial transfer is completed. The length of the delay is determined by DTL.

The QSPI cycles through the queue continuously. Each time the end of the queue is reached, the SPIF (QSPI Finished Flag) is set.

```
PCS to SCK Delay = DSCKL / System Clock
                 = 2 / 16.78 MHz
                 = 0.1 μs
Delay after Transfer = (32*DTL) / System Clock
                     = 32*2 / 16.78 MHz
                     = 4 μs
```

```
LDAB    #$B0      ; CONT=1, BITSE=0, DT=1, DSCK=1,
                  ; PCS0 active=0
STab    $FD40     ;initialize transfers 0
```

Command RAM consists of 16 bytes that are divided into two fields. The peripheral chip-select field enables peripherals for transfer. The command control field provides transfer options. Command RAM is used by the QSPI when in master mode. A maximum of 16 commands can be in the queue. Queue execution by the QSPI proceeds from the address in NEWQP (New Queue Pointer Value) through the address in ENDQP (End Queue Pointer Value).

```
TABLE:
        Fill data in the transmit data segment
        ORAA    SPCR1     ; read SPCR1
        STAA    SPCR1     ; enable the QSPI
*****  QSPI READY CHECK  *****
CHECK:  LDAB    #$00      ; halt not enabled
        STAB    SPCR3
        LDAA    SPSR
        ANDA    #$80      ; mask off everything but SPIF
        CMPA    #$80      ; see if SPIF is set
        BNE     CHECK     ; branch until set
        LDAA    #$7F
        ANDA    SPSR      ; don't disturb other flags
        STAA    SPSR      ; clear SPIF
        LDAB    #$01      ; halt enable
        STAB    SPCR3
        BRA     TABLE     ; next value
```

The system writes the data to the transmit data segment before enabling the QSPI. Shortly after SPE (QSPI Enable) is set, the QSPI commences operation at the address indicated by NEWQP[10]. Once the proper number of bits are transferred, the QSPI stores the received data in the receive data segment, stores the internal working queue pointer value in CPTQP (Completed Queue Pointer), increments the internal working queue pointer, and loads the next data required for transfer from the queue. The internal working queue pointer address is the next command executed unless the CPU writes a new value first. The SPIF (QSPI Finished Flag) bit must be checked because SPIF is set after execution of the command at the address in ENDQP.

3.3 SCI and Data/Command Transfer

The SCI (Serial Communication Interface) is used to communicate with an Intel 8088-based computer via an asynchronous serial bus (serial port on PC).

```
        LDD     #$0037    ; set the SCI baud rate to 9600
        STD     SCCR0     ; baud
```

SCCR0 (SCI Control Register 0) contains a baud rate selection parameter. Baud rate must be set before the SCI is enabled. SCI baud rate is programmed by writing a 13-bit value to SCBR (Baud Rate). The baud rate is derived from the MCU system clock by a modulus counter[9].

SCI Baud Rate = System Clock / (32*SCBR)

= 16.78MHz / 32*$37

= 9600

```
        LDD     #$000C    ; enable the SCI receiver and
        STD     SCCR1     ; transmitter, 1 start bit, 8 data
                          ; bits, 1 stop bit, parity disable
```

When initializing the SCI, the SCCR1 has two bits that should be written last: the transmitter enable (TE) and receiver enable (RE) bits, which enable the SCI. Registers SCCR0 and SCCR1 should both be initialized at the same time or before TE and RE are asserted.

```
SEND_CH:
        LDAA SCSR           ; read SCI status reg to
                            ; check/clear TDRE bit
        ANDA #$01           ; check only the TDRE flag bit
        CMPA #$00
        BEQ  SEND_CH        ; if TDR is not empty, go back to
                            ; check it again
        LDAA #$00           ; clear A to send a full word to
                            ; SCDR ($FFC0E)
        STD  SCDR           ; transmit one character to
                            ; the screen
TC_LOOP:
        LDAB SCSR+1
        ANDB #$80           ; test the TC bit (transfer
                            ; complete)
        CMPB #$00
        BEQ  TC_LOOP        ; continue to wait until TC is set
```

The CPU writes data to be transmitted to register TDR (Transmit Data Register), which automatically loads the data into the transmit serial shifter. Before writing to TDR, the system checks TDRE in SCSR. If TDRE = 0, then data is still waiting to be sent to the transmit serial shifter. Writing to TDR with TDRE clear overwrites previous data to be transferred. If TDRE = 1, the register TDR is empty, and new data may be written to TDR clearing TDRE. When the data is completely shifted out and no preamble or send break is requested, then TC (Transmit Complete Flag) is set to one.

3.4 PC Side

The Turbo C [11][12][13] function SetComConfig shown in Appendix B sets up the comport in the STATUS PC. The COM2 port is configured for 8-N-1, 9600 bps COM2.

The PC receives 5 status bytes from LR node in a loop.

```
while ( i < 5)
 {
   byte_read=inp(base+5);
   test=byte_read & 1;
   if (test==1)
  {
    buffer[i] = inp(base);
    i += 1;
   }
```

The bytes are stored to buffers waiting for retrieval.

```
if ((buffer[1] & '\x1') == '\x1')
  {
    lowvideo();
    cputs("LOCKED  ");
  }
   else
```

```
    {
      highvideo();
      cputs("UNLOCKED");
    }
```

The program gets the status of devices, mode and alarm from the first two bytes. If the bit of the byte is 1, it means something is locked, closed, on, or vacation mode. If the bit of the byte is 0, it means something is unlocked, open, off, or normal mode. Table 3.1 and Table 3.2 show the relation between bits in buffers and devices. The program knows the updated time on EVB from the last three bytes.

Table 3.1 Relation between bits in buffer[1] and devices

buffer[1]	status of which device (1/0)
b0	W_LR_LK (locked/unlocked)
b1	W_LR_CS (closed/open)
b2	DOOR_LK (locked/unlocked)
b3	DOOR_CS (closed/open)
b4	RLS_SR (set/release)
b5	W_BR_LK (locked/unlocked)
b6	W_BR_CS (closed/open)
b7	reserved

Table 3.2 Relation between bits in buffer[2] and devices

buffer[2]	status of which device (1/0)
b0	LIGHT_BR (on/off)
b1	LIGHT_LR (on/off)
b2	SET_VM (vacation/normal)
b3	ALARM (on/off)
b4	LIGHT_OUT (on/off)

The program also gets the updated time from PC itself.

```
time(&tnow);
tmnow=localtime(&tnow);              ; call time function
hour = (*tmnow).tm_hour;
min = (*tmnow).tm_min;
sec = (*tmnow).tm_sec;
gotoxy(38,16);
highvideo();
cprintf("%02d:%02d:%02d", hour, min, sec);
```

CHAPTER 4

PHYSICAL LAYER

The physical layer is made up of two sublayers as shown in Figure 4.1: the Medium Dependent Physical sublayer (MDP) and the Symbol Encoding sublayer (SE). The SE sublayer functions are as follows:

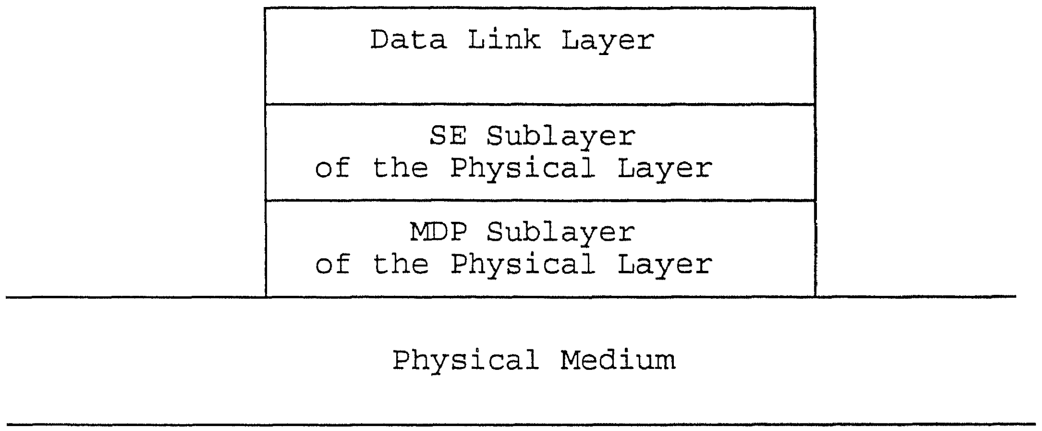

Data Link Layer
SE Sublayer of the Physical Layer
MDP Sublayer of the Physical Layer
Physical Medium

Figure 4.1 CEBus physical layer

- monitor the communication channel and reports channel failures and recoveries to the layer system management.
- provide the data link layer with a time base that allows its MAC sublayer to execute the CEBus medium access protocol.
- accept transmission requests from the MAC sublayer and reports back the success or failure of the transmission.

- accept state change indications from the MDP sublayer, translates the period between state changes to CEBus symbols, and relay the received symbols to the MAC sublayer.

The MDP sublayer includes electrical signaling requirements, transceiver hardware specifications (requirements, tolerances, delays, etc.), the physical attachment to the medium, and all other mechanical requirements. The MDP sublayer also includes the representation of SUPERIOR and INFERIOR states for the particular medium[14].

4.1 SSC PLCEMS

The SSC PLCEMS is a board level product for implementing power line communication networks utilizing Intellon's Spread Spectrum Carrier technology and the Electronic Industries Association Consumer Electronics Bus (EIA CEBus) protocol. The SSC PLCEMS handles portions of the CEBus Physical Layer Symbol Encoding (PLSE) sublayer, including complete CRC generation and detection, and portions of the Medium Access Control (MAC) sublayer. A host microprocessor is required to handle the remaining parts of the PLSE and MAC as well as the Data Link, Network, and Application layers of the CEBus protocol[4].

4.2 Spread Spectrum Carrier Technology

The SSC PLCE implements Spread Spectrum Carrier Technology with a chirp that is swept in the range of 100 to 400 KHz with a duration of 100 μsec, per the EIA CEBus Powerline Bus physical layer standard. The chirp is swept from approximately 200 KHz to 400 KHz and then from 100 KHz to 200 KHz. Figure 4.2 shows the CEBus PL chirp. This chirp represents the filtered version of the SSC PLCEMS output. Each chirp represents the shortest symbol time or Unit Symbol Time (UST), allowing for a data rate of 10,000 USTs per second[4].

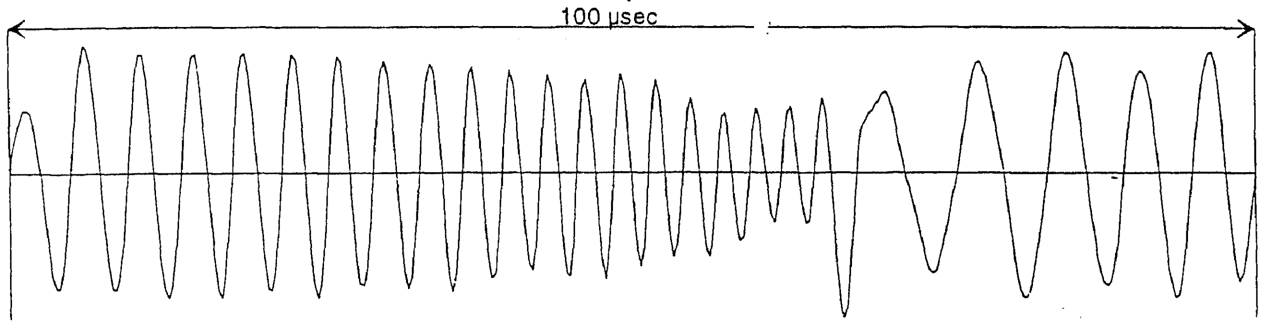


Figure 4.2 Spread spectrum carrier chirp

4.3 Signal Encoding

The SSC PLCEMS transmits and receives packets of data as a stream of symbols encoded following the EIA CEBus Power Line (PL) standard. The signal encoding is a Non Return to Zero (NRZ) Pulse Width Encoding format using the symbols "1", "0", "EOF", "EOP". Each of the available symbols is encoded as a number of USTs as shown in Table 4.1.

Each packet transmitted or received by the SSC PLCEMS consists of three sections: a preamble, packet body, and CRC. The SSC PLCEMS does not provide the symbol encoding for the preamble or packet body. For transmission the symbols for the preamble and packet body must be encoded by the host controller and transferred to the SSC PLCEMS via the synchronous serial port. Each bit transferred to or from the SSC PLCEMS represents one UST. A one bit requests chirp for one UST; a zero requests no chirp for one UST. The SSC PLCEMS will generate and encode all CRC information for transmission. Upon reception, the host controller must decode the symbols for the packet body and check the status of the CRC indicator. The SSC PLCEMS will decode and strip off all preamble information[4] and perform the CRC check.

Table 4.1 Power line symbol encoding

Symbol	Preamble		Packet Body	
	USTs	Timing	USTs	Timing
"1"	1.0	114μsec	1.0	100μsec
"0"	2.0	228μsec	2.0	200μsec
End of Frame (EOF)	8.0	800μsec	3.0	300μsec
End of Packet(EOP)	N/A	N/A	4.0	400μsec

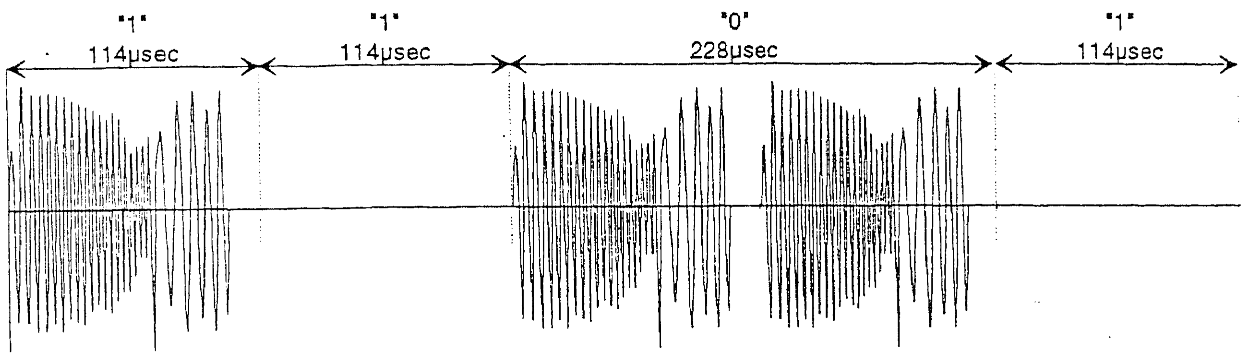


Figure 4.3 Amplitude shift keying (ASK) "1101"

4.4 Preamble Encoding

Two modulation schemes are used by the SSC PLCEMS. Amplitude Shift Keying (ASK) is used in the preamble of the data packet. ASK uses alternating SUPERIOR and INFERIOR states. A superior state is represented by the presence of a chirp, an inferior state by the absence of a chirp. Because the transmitter is quiet during inferior states, other devices contending for the channel can be detected during the preamble of the packet. An example of ASK is shown in Figure 4.3. This sequence of chirps represents the symbols "1101". Note that in the preamble the duration of a UST is slightly longer than in the body of the packet and the Preamble_EOF: 114 μsec for the preamble, 100 μsec for the body and Preamble_EOF. The SSC PLCEMS generates and recognizes the proper timing internally.

When the preamble data is encoded, the first symbol, whether it is a "1" or a "0" must be encoded as a superior state. Following the preamble data, a Preamble_EOF consisting of 8 superior states must be passed to the SSC PLCEMS to indicate the end of the preamble data. The Preamble_EOF must be byte-aligned with the serial 8 bit transfers from the host to the SSC PLCEMS. The encoded data must be padded with leading zeros to force this alignment. More detail on the SSC PLCEMS can be found in Appendix B.

4.5 Subroutine for Preamble Encoding

One byte of preamble must be translated to 2 bytes of UST. The MSB of preamble must be translated to a superior state, so the LSB of preamble must be translated to an inferior state. The preamble byte is shifted left and translated bit by bit. If the random preamble = $FF, let the first byte of UST = $AA to avoid the modem from misinterpreting it as PRE_EOF. The subroutine is shown in Appendix C.

4.6 Packet Body Encoding

A second type of modulation is used in the body of the data packet. Phase Reversal Keying (PRK) utilizes two phases of the superior state, SUPERIOR 01 and SUPERIOR 02, which are 180 degrees out of phase with one another, to modulate the encoded data. This modulation technique is more robust than the ASK technique because it allows the SSC PLCEMS to correlate and track each UST rather than just those encoded as superior states. Figure 4.4. shows an example of PRK.

The symbols are encoded by the host controller with the same scheme as the preamble data, but it is not necessary to align the EOF symbols or the EOP symbol in the data packet on byte boundaries. The first symbol following the Preamble_EOF must be encoded with the opposite state, superior. The translated data packet must end with an EOP symbol ("1111" or "0000") and not contain an EOP within the body of the packet in order for the SSC PLCEMS to correctly generate and transmit the CRC. The final byte to be transmitted to the SSC PLCEMS may contain don't care values after the final UST of the EOP symbol[4].

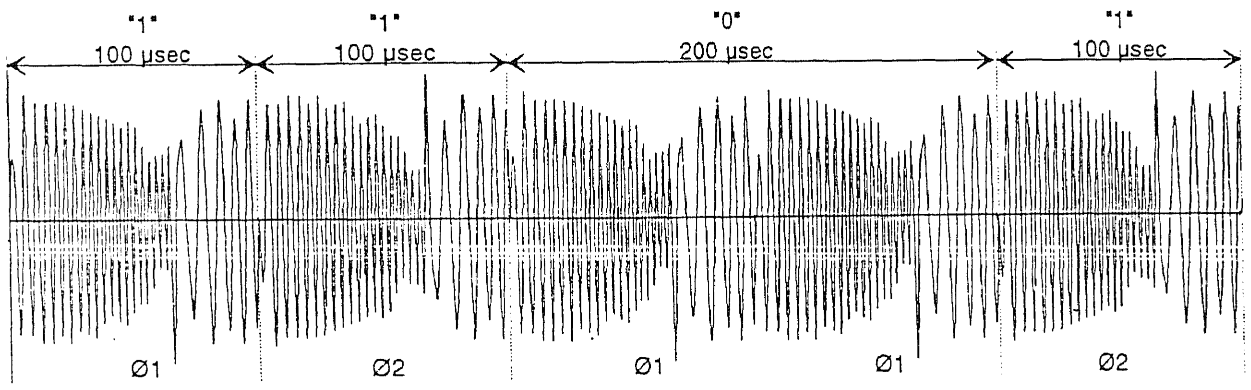


Figure 4.4 Phase reverse keying (PRK) "1101"

4.7 Subroutine for Packet Body Encoding

If PRE_EOF = "11111111", the MSB of packet body must be translated to an inferior state. Again we shift left once to get the next data bit and translate it. Because 16 USTs are created at a time, the program must take care of that there is no PRE_EOF before the end of packet. The subroutine is shown in Appendix D.

4.8 Subroutine for Packet Body Decoding

If PRE_EOF on the transmitting modem is "FF", the receiving modem will get "FF" or "00" because the phase will change according to the phase of power line. If USTs = "00" or "11", it will be translated to "0". If UST = "0" or "1", it will be translated to "1". The subroutine is shown in Appendix E.

CHAPTER 5

DATA LINK LAYER

The job of the data link layer is to make a transmission channel, which is subject to noise and interference from other data traffic, appear to the network layer as an open, error-free channel. This is done through standard frame construction, an appropriate channel access technique, error control, and a mechanism for acknowledgment and retransmission[3].

5.1 Normal Frame Format

Normal MAC frames using MAC level error detection must adhere to the following frame format:

PRE	PRE_EOF	Control	DA	DHC	SA	SHC	information	FCS	EOP

5.1.1 Preamble (PRE) Field

The Preamble is a fixed length, 8-bit field which contains a random value. It is a noninformation-bearing field, transmitted ahead of the information-carrying fields to vie for use of the channel. Another node may interfere during transmission of the Preamble, but no real information will be lost. PRE is a fixed value (69H) for this demonstration system[3].

5.1.2 Control Field

The Control field directs the handling of the frame at its destination. The field is variable in length up to a maximum of 8 bits. The contents of the Control field are generated within the Data Link Layer from parameters in the data request service primitive. Refer to Figure 1.1. The contents of the Control field are 0AH for unacknowledged data in this demonstration system. The Packet Priority is standard, it is nonprivileged, and the Service Class is basic.

5.1.3 Destination Address (DA) Field

The Destination Address field specifies which node within a network, or home system, is to receive the frame. The field is variable in length up to a maximum of 16 bits. There are two nodes in the demonstration system, so "0001" is assigned to the living room and "0002", the bedroom.

5.1.4 Destination House Code (DHC) Field

The Destination House field identifies the destination home system out of a group of systems which share common communications media. Together, the Destination Address and Destination House Code identify a unique node, or group of nodes. The destination House Code field is variable in length up to a maximum of 16 bits. There is one house in the demonstration system, so "0001" is assigned to the house.

5.1.5 Source Address (DA) Field

The Source Address field defines the node within a home system from which the frame originated. The Source Address may reference only one node. The field is variable in length up to a maximum of 16 bits.

5.1.6 Source House Code (SHC) Field

The Source House Code field identifies the source home system on a shared common communications media. Together, the Source Address and Source House Code identify a unique node.

5.1.7 Information Field

The information field contains an NPDU (see Figure 1.2) from the network Layer. The Data Link Layer performs no operations on this field. The field is variable in length up to a maximum of 32 bytes and may be Null.

5.1.8 Frame Check Sequence (FCS) Field

The Frame Check Sequence field is the last field in the frame and provides a means for determining the integrity of the contents of the frame. The field is variable in length up to a maximum of 8 bits and is generated within the Data Link Layer. Its value is a checksum which is calculated in the following manner:

The checksum value is initialized to zero.

Each 8-bit field from the frame (excluding the Preamble) is added with carries discarded. Fields longer than eight bits are subdivided into 8-bit subfields. The subfield containing the least-significant eight bits is added first. Other subfields are added in order up to the subfield containing the most-significant eight bits.

The two's complement operation is performed on the checksum and this final value is supplied to the FCS field.

Error detection is possible at the receiving Data Link Layer by summing each of the 8-bit fields and subfields as they arrive. If no errors are introduced during transmission, the data will sum in the same way it did for the checksum calculation. However, when the last arriving field (the checksum, which is the two's complement of all data summed so far) is added, the result will be zero. A non-zero value indicates the presence of errors.

CHAPTER 6

APPLICATION LAYER

The CEBus application layer consists of four main elements as shown in Figure 6.1. Layer System Management administers functions which have significance across layer boundaries or to the CEBus device as a whole. These functions may be of local importance, such as addressing, or of system-wide importance, such as routing information. The application process is the interface to the application layer. Services are provided by the Common Application Language (CAL) element to the user element of the application process. CAL is the language framework through which resource allocation and control functions are executed. Services are provided by the message transfer element to the CAL element. The message transfer element interfaces to the network layer either directly or through the association control element[4].

6.1 CAL Functions

CAL provides two main functions: resource allocation and control. Requests for these services are received from both the user element and LSM. Using CAL syntax, CAL translates these requests into Application Layer Service Data Units (ASDUs) and passes them to the message transfer element for delivery. The CAL element also receives and interprets incoming ASDUs[15].

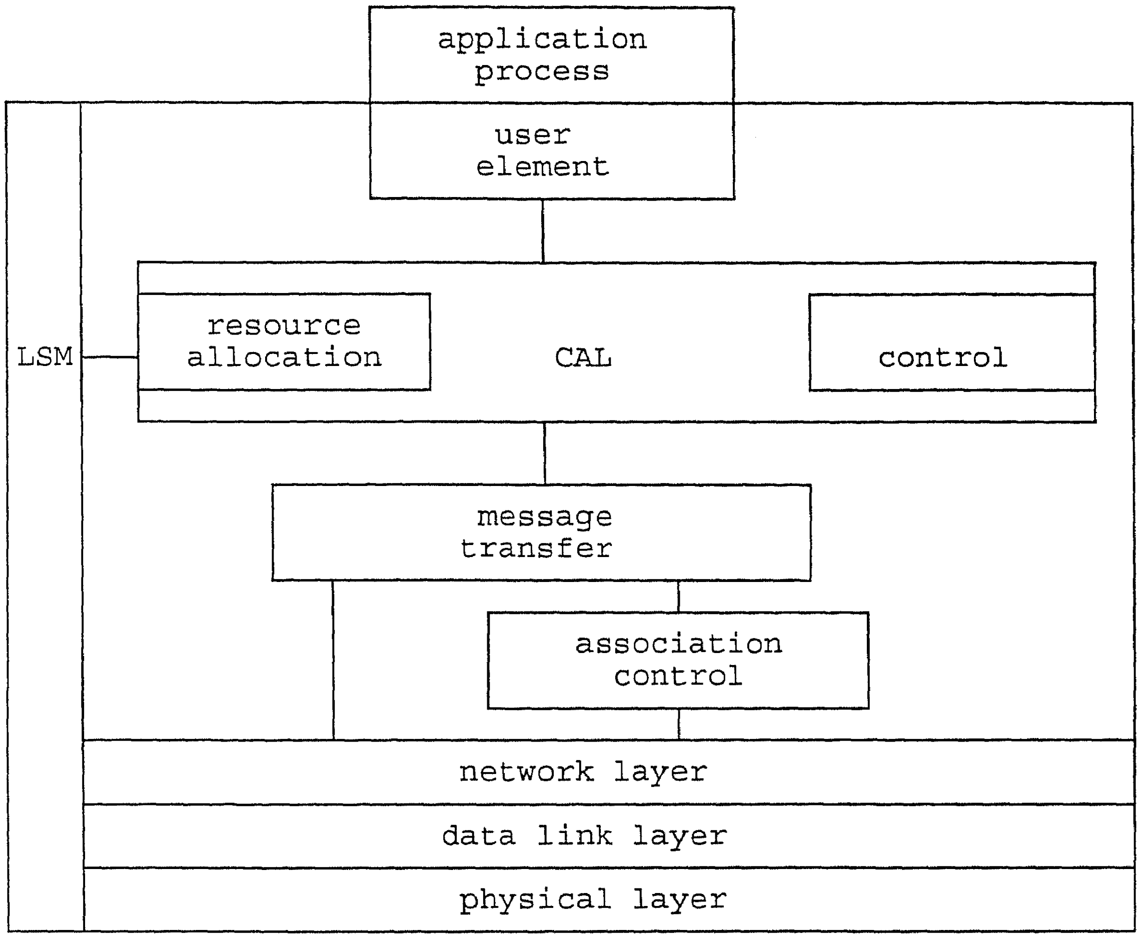


Figure 6.1 CEBus application layer interfaces

6.1.1 Resource Allocation Function

The resource allocation function within CAL is concerned with requesting, using, and releasing CEBus resources. These resources include, but are not limited to Medium Access Control (MAC) individual addresses, group addresses, system addresses (house codes), and data channels. The node control, data channel receiver, and data channel transmitter objects are used to perform the resource allocation function[15].

6.1.2 Control Function

The CAL control function provides to the user element the capability to send CAL commands to remote devices and to respond to incoming CAL commands. The CAL command syntax is designed to encode command packets using the smallest number of bytes possible, while still retaining the flexibility to control diverse current and future devices.

6.2 Contexts

Each context has a unique context identifier by which it is addressed. The first part of the context identifier is the context number. This number only needs to be used as part of the context identifier if there is more than one context of the same type in the product. The second part of the context identifier is the context class. The class identifies the context type (i.e., Audio, Tuner, etc.). The context class is always sent in CAL commands[15].

6.3 Objects

A CAL object is a model of a single functional entity used to perform a single control task within a product. Objects are designed to be generic in form such as a "switch" or a "button." Objects do not assume a specific application until they are placed in a specific context.

6.4 Methods

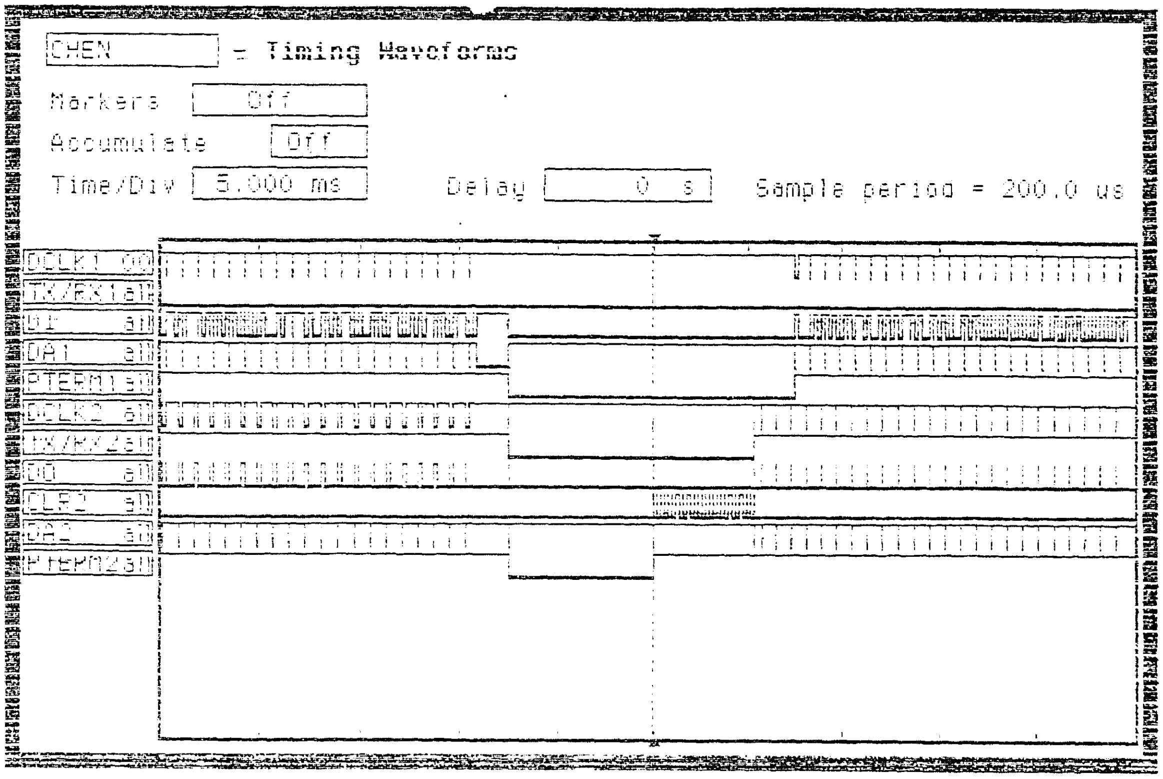
Methods represent the functions within an object which perform some action within the device. Methods are invoked when a message is received by an object. Methods are divided into two classes, simple and complex. A simple method uses only the arguments supplied in the message. An example of a simple method is increment, which takes as its arguments the variable to increment and the amount to add to the variable. A complex method is a method which may be composed of one or more simple and/or complex methods. An example of a complex method is the if method, which performs a similar function to the IF statement in programming languages such as Pascal or C.

6.5 Example CEBus Communication

If the window in the bedroom is closed, according to Table 6.1, a message ($81, $05, $42) is sent to the living room.

Table 6.1 Function code for CAL

context	object	method
LIGHTING[16] 21 Lighting CONVENIENCE 81 Window Control 83 Door/Gate Control	05 Binary Switch	41 Set Off 42 Set On


CHEN = Timing Waveforms

Markers Off

Accumulate Off

Time/Div 5.000 ms   Delay 0 s   Sample period = 200.0 us

Figure 6.2 Timing waveform

As shown in Figure 6.2, the house is in the normal mode. The bedroom (BR node) always sends the status of all devices to the living room (LR node). The LR node uses interruption to receive the packets. The first five fields are for the LR node. The others are for BR node. DCLK shows SPI clock. When a node is the receiver, TX/RX goes low. When a node is the transmitter, TX/RX goes high. DI means data input. DO means data output. When data is avaible, DA goes low. When it is at the end of one packet, PTERM goes low. CLR is used to delay a short time before the next packet is transmitted. See Figure 1.5 for the state diagram of the demonstration system.

When the 3-byte message above combined with the other packet bytes specified in the frame format above, a 16-byte packet results. When this is encoded in UST format, it becomes 27 bytes. The LR node receives packets with PREAMBLE and EOP stripped off. The physical layer software converts the rest of the UST formatted packet into information bytes (now 14).

6.6 C Version

A C version of an application program was written before the version described above. It sets all parameters the demonstration needs. One subroutine CHECK_DEVICE() is used to check all devices in the demonstration system in a routine. The RS-232 protocol is used to send messages continuously from the LR node to the STATUS PC. An internal interrupt routine is implemented every 1 second for the CEBus CLOCk. This version was not completed.

CHAPTER 7

SYSTEM PERFORMANCE

7.1 Responsiveness

The system responsiveness was measured experimentally for a case of the transmission of a packet with 3 information bytes. The entire packet has 16 bytes and becomes 27 bytes of UST after it is encoded. The BR node continuously sends the test packet to the LR node. When a packet is decided to be sent, a specific output pin on the BR node produces a pulse as a start indicator. When the packet is decoded completely, a specific output pin on the LR node produces a pulse as a complete indicator. We measure the delay between the two indicators to get the timing diagram of Figure 7.1. The delay between a decision to send and the time for the information bytes to become available to the receiving application is 30 ms. The time for transmitting the packet is 22 ms. The remaining 8 ms are spent with encoding, decoding, and the frame check, all accomplished in software.

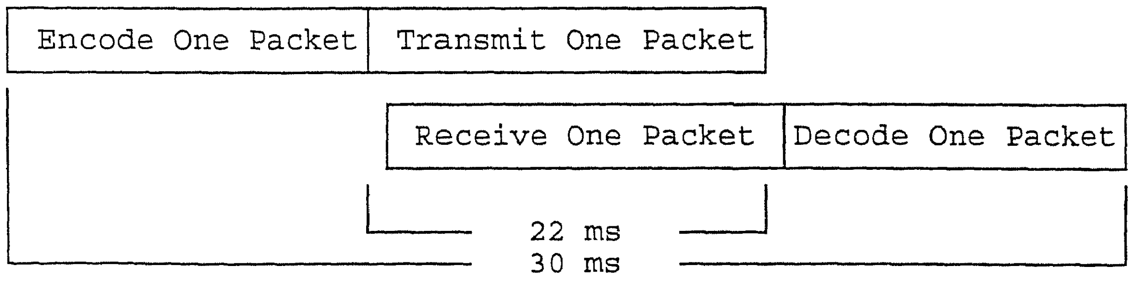


Figure 7.1 Timing diagram

7.2 Noise Immunity

The loaded AC line in the lab has an obvious distortion of a few volts amplitude at the positive peak. When power line modems are connected to it, a consistent CRC error rate of 2% (packet error rate) is produced. Capacitively coupling a broadband noise generator to the power line does not increase this error rate noticeably. The loading of all the devices plugged in the line under test and the need to block out 60 Hz combine to make the introduced noise amplitude not noticeable on the scope.

When a straight 3-wire connection of the modems, without AC power and with no introduced noise, is established, the result is error free operation for at least 30,000 packets.

When random noise up to 500 kHz (the modems cut off at 400 KHz) is introduced, using a coupling capacitor, both CRC and FCS errors occur. The random-noise generator is model 1390-B manufactured by General Radio Company. The noise rms value was measured from the voltmeter on the random-noise generator. The noise peak to peak voltage on the CEBus was also measured. For one case, 8 Vp-p was measured when the noise was 1.2 V-rms at the meter. The CRC error rate is approximately 1%, weakly dependent on noise amplitude above 400 mV-rms. The FCS error rate is about 1/3 of the CRC error rate. Figure 7.2 shows a curve of error packets vs. rms noise level.

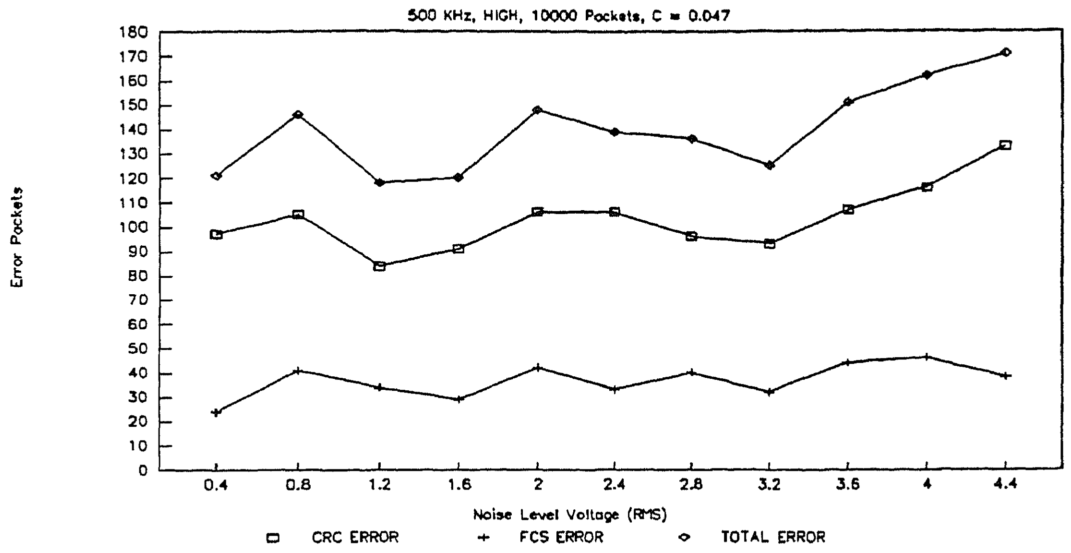


Figure 7.2 Errors vs. noise amplitude - 1

Figure 7.3 shows the curve of error packets vs. rms noise level from 0.04 V to 0.36 V. The CRC error rate of 0.2% with noise amplitude at 0.04 V-rms increases to 0.8% at a noise amplitude 0.36 V-rms. The FCS error rate 0.1% at a noise amplitude 0.04 V-rms increases to 0.3% at a noise amplitude 0.36 V-rms.

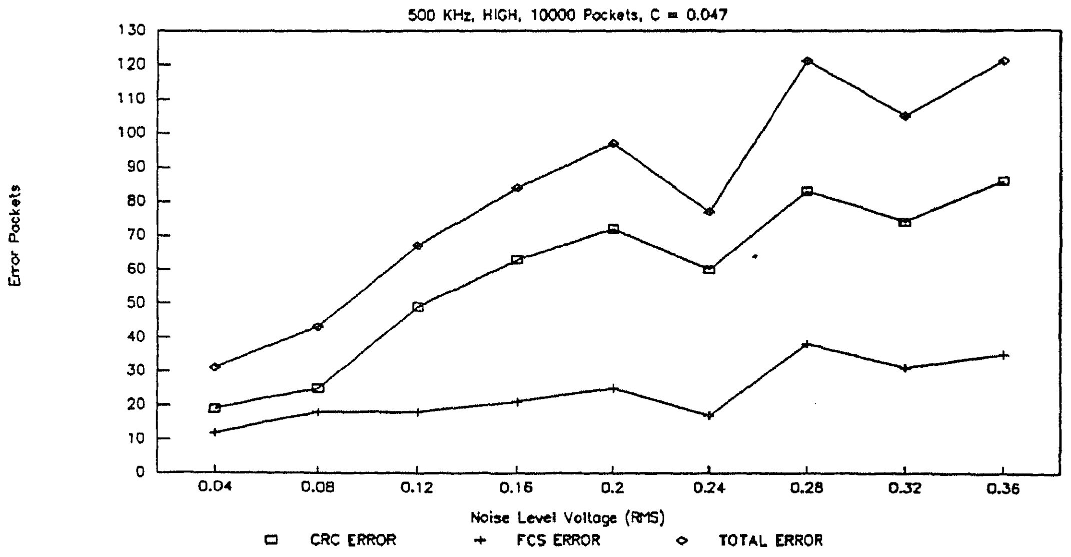


Figure 7.3 Errors vs. noise amplitude - 2

Figure 7.4 shows the curve of error packets vs. rms noise level from 0 V to 0.036 V. The CRC error rate of 0% at a noise amplitude 0 V-rms increases to 0.2% at a noise amplitude of 0.036 V-rms. The FCS error rate of 0% at a noise amplitude of 0 V-rms increases to 0.1% at a noise amplitude of 0.036 V-rms.

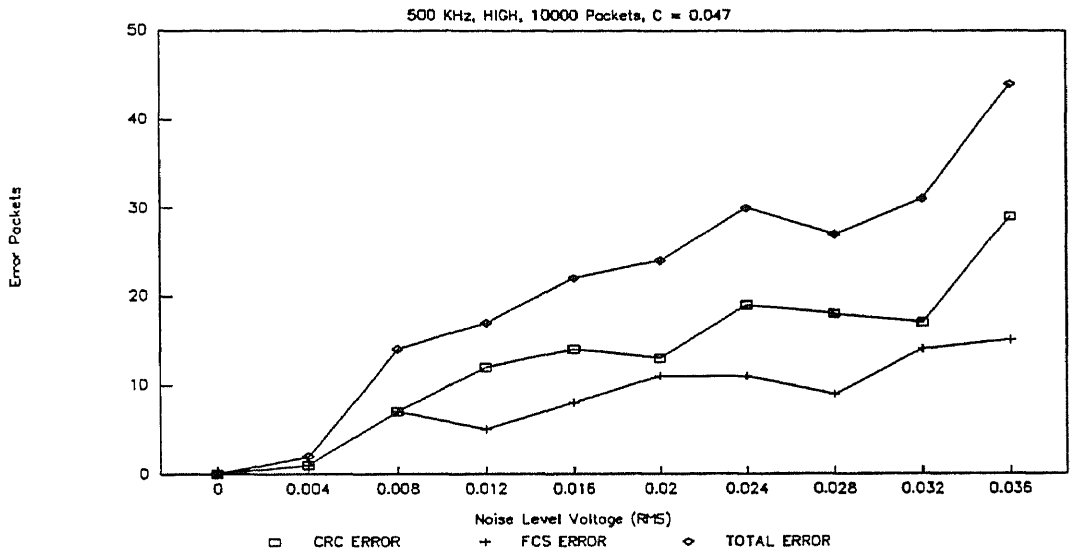


Figure 7.4 Errors vs. noise amplitude - 3

Evidently the designers of the CEBus PL standard have achieved their goal of low error rates. Line distortion causes more errors than injected high frequency noise. See Appendix F for the noise data.

CHAPTER 8

CONCLUSION AND POSSIBLE FUTURE WORK

8.1 Conclusion

In this thesis work, a flexible CEBus demonstration system applied to home automation security has been designed and implemented. When the system is in vacation mode, it emulates that someone is in a house so as to enhance security. The system demonstrates the application of a Motorola single-chip microcontroller and a power line modem to implement each of two system nodes. An interconnected personal computer PC-XT/AT acts as a special peripheral on one of the nodes to display system status.

The demonstration functions were implemented in CAL (Common Application Language), the standard language by which CEBus devices communicate. It provides a language for controlling CEbus devices, reporting status, and allocating resources.

To make microcontrollers communicate with power line modems through a serial synchronous link (QSPI), the QSPI (Queued Serial Peripheral Interface) parameters must be set up to match the modem parameters. Packet bytes must be encoded in UST before being sent to a power line modem. Then they must be decoded after reception by a modem. When this process is implemented in software, it adversely affects the system responsiveness. The delay between a decision to send and the time for 3 information bytes to become available to the receiving application is 30 ms, of which 8 ms is software delay.

When a straight 3-wire connection of the modems without AC line voltages is established, the result is error free operation with no introduced noise. When noise up to 500 kHz is introduced, CRC and FCS errors occur. The CRC error rate is approximately 1%, weakly dependent on noise amplitude above 0.4 V-rms. The FCS error rate is about 1/3 of the CRC error rate. A distorted AC line connection resulted in a packet error rate of about 2%.

8.2 Suggestions for Future Work

There are many approaches to advance the demonstration system functions.

The system can be expanded beyond 2 nodes to demonstrate more interesting applications.

Power line modems are used in the demonstration system. Other modems for different media can be connected and tried.

CAL is still under development, As it evolves, more complicated functions can be added to the demonstration system in the future.

APPENDIX A

CEBUS SPREAD SPECTRUM POWER LINE MODEM

A.1 SSC PLCE Integrated Circuit

A.1.1 Input Amplifier and A/D Converter

The input amplifier provides a gain of 20 to the filtered signal received from the powerline as shown in Figure A.1. The amplified input passed to a one-bit A/D converter consisting of a slope detector and single-bit quantizer. The characteristics of the slope detector are controlled by the external components connected to F1, F2, and F3. The quantizer samples the output of the slope detector at the XTAL frequency. The output of the quantizer is serial digital data indicating the slope of the input signal.

A.1.2 Matched Transversal Filter

The serial output of the A/D is passed through a digital low-pass filter and the supplied to a matched transversal filter which provides a fuzzy logic correlation of the received signal to the Spread Spectrum Carrier (SSC) chirp. The SSC chirp is designed to provide a very high self-correlation at a single point in the filter. The magnitude of the correlation is directly related to the quality of the received SSC chirp, and the sign (positive or negative) of the correlation is determined by the phase of the chirp.

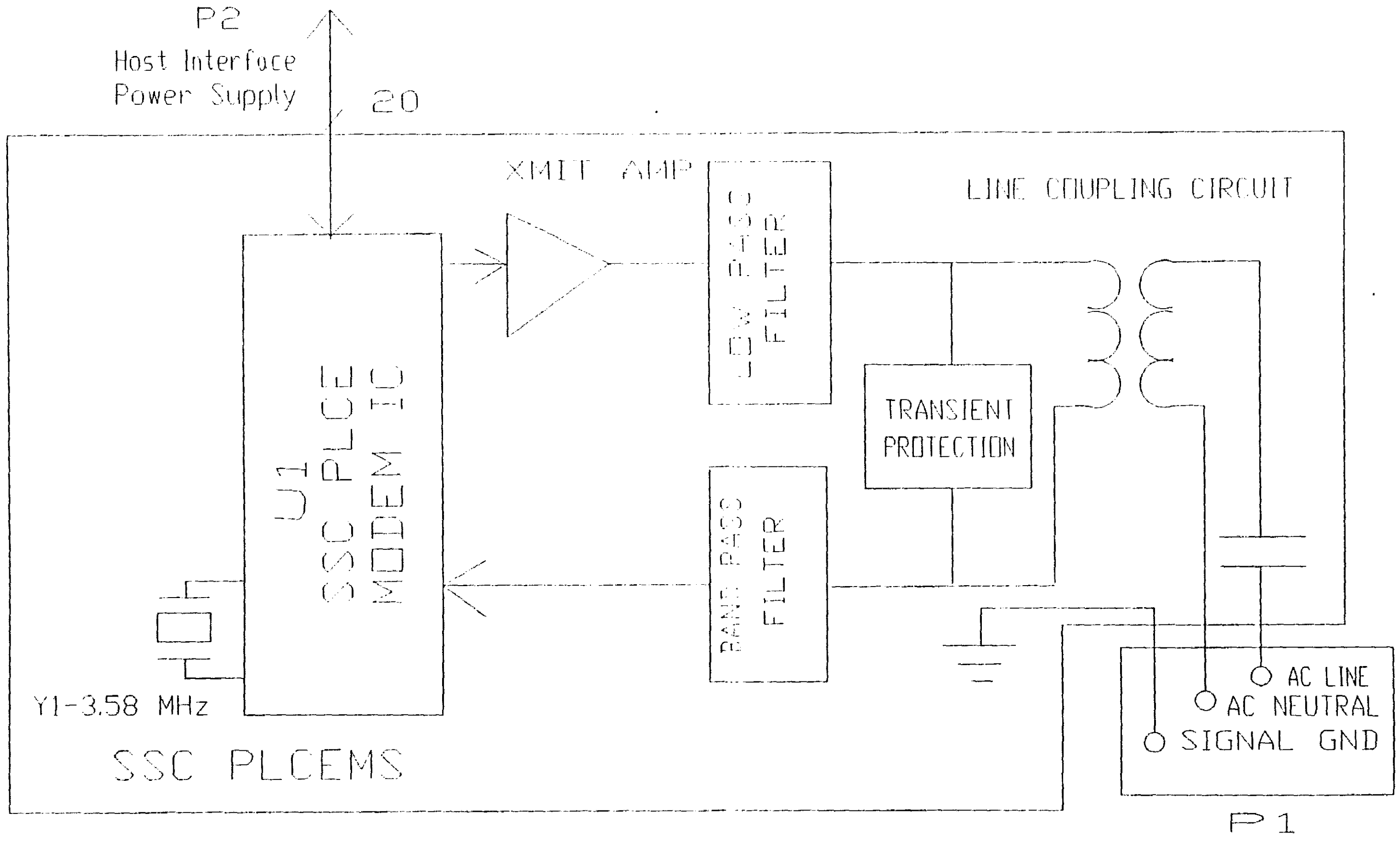


Figure A.1 SSC PLCEMS block diagram

Chirps which are corrupted by noise or other impairments will not correlate as strongly as clean signals.

SSC chirps provide the carrier function in the communications system. The presence of a carrier is determined by magnitude of the correlation output of the filter. When no chirps are present on the communications medium, the output of the filter will be at a nominal level. When chirps are present at the beginning of a reception, the filter will look for a correlation exceeding a detection threshold. When the threshold is initially broken, the filter indicates that a carrier is present on the medium and begins to track the incoming data. The detection threshold is adjustable with the T0 and T1 inputs.

When a carrier is present, correlations should occur at regular intervals as successive chirps are received by the filter. When the filter correlates a SSC chirp, it looks for the next correlation one chirp later. This allows marginal correlations to be interpreted correctly, thereby improving the performance of the system. Marginal correlations may occur if the characteristics of the communication medium change (i.e. impulse noise, impedance. modulation, etc.) while data are being received. Once the filter has begun to track the incoming data, it will maintain tracking with only marginal correlations for over 1 ms before indicating a loss of carrier.

The tracking system in the filter also adjusts for the asynchronicity between the transmitting and receiving nodes resulting from slightly different XTAL frequencies. By comparing when each correlation occurs to when it is expected, the filter can sense any drift in the frequency of the incoming data and adjust for it.

The final portion of the filter contains the steering logic necessary to extract the UST data from the correlator output and recognize the Preamble EOF and the EOP symbols. The data extraction logic recognizes positive correlations as 1's and negative correlations as 0's. This data is passed to the host interface for transfer to the host microprocessor.

A.1.3 Host Interface

The host interface provides all of the logic required to interface to the host microprocessor. This includes the synchronous serial port and handshake logic. The serial interface is used for transferring data to and from the SSC PLCEMS. The handshake logic provides status and control of the SSC PLCEMS.

The serial interface is a three wire synchronous serial interface compatible with industry standard synchronous serial interfaces and simple shift registers. Data is passed to, and received from, the SSC PLCEMS in 8-bit sections. The host microprocessor is the master and the SSC PLCEMS is the slave: the host provides the clock signal to the interface. In transmit mode, the SSC PLCEMS receives data MSB first on DI on the rising edge of DCLK. DO is high-impedance in transmit mode to allow DO and DI to be tied together if two wire synchronous serial communication is desired. In receive mode, the SSC PLCEMS presents new data MSB first to DO on the falling edge of DCLK. Data presented to DI during receive mode are not accepted by the SSC PLCEMS. Each data transfer requires exactly eight rising edges and eight falling edges of DCLK. The host must provide DCLK exactly eight clock pulses for each data transfer(see Figure A.1).

The handshake logic is used to control and receive status information from SSC PLCEMS. It provides the interrupt signal to the host based upon the status of the SSC PLCEMS. It also indicates when new data is available at the serial port or when the serial port is ready for more data. The handshake logic works in conjunction with the CRC logic to indicate the integrity of the received data.

The control functions of the handshake logic enable the SSC PLCEMS for operation and select the transmit or receive mode.

A.1.4 CRC Logic

The CRC logic provides the generation of 16-bit CRC on transmit and decoding of CRC on received packets. The CRC codes are generated based upon each UST received by the SSC PLCEMS from the host. Each 01 bit is counted as a 1 for CRC generation. On reception, the CRC logic must determine the polarity of the incoming data in order to properly decode the CRC appended to the packet. The polarity of the incoming packet is indicated by Preamble_EOF received and decoded by the data extraction logic. If the incoming data is determined to be of the opposite phase, the received data is complemented to calculate the CRC at the receiver. The CRC is transmitted and received in binary from as opposed to being encoded and decoded as symbols.

A.1.5 Clock Generation

The clock generation logic generates all of the clock frequencies required internally by the SSC PLCEMS. Each of the clock signals is derived from the XTAL frequency. The clock frequency also provides a buffered output at the XTAL frequency (CKOUT) suitable for driving the clock input of a microprocessor.

A.1.6 Waveform Generator

The waveform generator consists of three sections: a ROM-based wavetable, D/A converter, and an output amplifier. The wavetable is a 358x6-bit ROM which contains the binary image of the SSC chirp. An address generator is clocked at the XTAL frequency to sequence through the wavetable for generation of a chirp.

The output of the wavetable is latched at each clock cycle. Either the true or inverted version of the wavetable data is presented at the input to a 6-bit DAC. The output of the DAC is buffered by an output amplifier and sent out on SO.

A.1.7 Bias Generator

The bias generator provides the bias currents required by analog sections of the SSC PLCEMS. All bias currents are provided through a 5.1K(5%) resistor connected to ground from BIAS.

APPENDIX B

SUBROUTINE FOR RS-232 COMMUNICATION

```
 SetComConfig(comport, baudrate);
/*  baudrate        MSB (b+1)        LSB (b)
 *  110             04               17h
 *  300             01               80h
 *  1200            00               60h
 *  2400            00               30h
 *  4800            00               18h
 *  9600            00               0ch
 * 19200            00               07h
 *  wordlength        7 bits = 2
 *                    8 bits = 3
 *  stopbits          1 bit = 0
 *                    2 bit = 1
 *  parity            none = 0 or 2
 *                    odd  = 1
 *                    even = 3
 *  config = wb + sb + pb = (3+0+0)
 *  out (b+4), 0
 *  out (b+3), 128
 *  out (b),   LSB
 *  out (b+1), MSB
 *  out (b+4), 3
 *  out (b+3), config
 */
{
   int msb, lsb, base, config=0x03;
   switch (baudrate) {
     case 9600:
         msb=0x00;
         lsb=0x0c;
         break;
     case 4800:
         msb=0x00;
         lsb=0x18;
         break;
     case 2400:
         msb=0x00;
         lsb=0x30;
         break;
     case 1200:
         msb=0x00;
         lsb=0x60;
         break;
     case 300:
         msb=0x01;
         lsb=0x80;
         break;
     case 110:
         msb=0x04;
         lsb=0x17;
```

```
          break;
      default :
          printf("\n Baudrate error\n");
          exit(0);
    }
    switch (comport) {
      case 1:
          base=0x3f8;
          break;
      case 2:
          base=0x2f8;
          break;
      default:
          printf("\n Com port error:  Use com 1 or 2\n");
          exit(0);
    }
    outp(base+4, 0x00);
    outp(base+3, 0x80);
    outp(base, lsb);
    outp(base+1, msb);
    outp(base+4, 0x03);
    outp(base+3, config);
}
```

APPENDIX C

SUBROUTINE FOR PREAMBLE ENCODING

```
TRANS_PRE:
   BRSET PREAMBLE,X,#$80,BIT_FILL_1    ; check odd bit of pre
   JSR      FILL_11                    ; = "0", fill "11"
   BRA      TRANS_NEXT
BIT_FILL_1:
   JSR      FILL_1                     ; = "1", FILL "1"
TRANS_NEXT:
   INC      NO_BIT,X                   ; record no. of bit
   ROL      PREAMBLE,X                 ; shift left to get
                                       ; the next bit
   BRSET PREAMBLE,X,#$80,BIT_FILL_0   ; check even bit of pre
   JSR      FILL_00                    ; = "0", fill "00"
   BRA      CHECK_LAST                 ; if last bit
BIT_FILL_0:
   JSR      FILL_0                     ; = "1", fill "0"
CHECK_LAST:
   INC      NO_BIT,X
   LDAA     NO_BIT,X
   CMPA     #$08
   BEQ      GET_UST                    ; finish, get UST
   ROL      PREAMBLE,X                 ; or get next bit
   BRA      TRANS_PRE
GET_UST:
    LDAA    TEMP1,X                    ; get 2 bytes of USTs
    STAA    PRE1_UST,X
    LDAA    TEMP2,X
    STAA    PRE2_UST,X
    CMPA    #$00
    BNE     CLEAR_B
    LDAA    #$AA                  ; PRE1_UST = "00", fill "AA"
    STAA    PRE1_UST,X
CLEAR_B:
    CLRB
    CLRW    TEMP1,X
    CLR     NO_BIT,X
    RTS
```

APPENDIX D

SUBROUTINE FOR PACKET BODY ENCODING

```
TRANS_BIT:
   JSR     CHECK_UST16                 ; 16 USTs is completed
   BRSET   BYTE1,Y,#$80,D_BIT_FILL_0   ; odd bit of one byte
   JSR     FILL_00                     ; = "0", fill "00"
   BRA     TRANS_BIT_NEXT
D_BIT_FILL_0:
   JSR     FILL_0                      ; = "1", fill "0"
TRANS_BIT_NEXT:
   INC     NO_BIT,X                    ; record no. of bit
   JSR     CHECK_UST16                 ; 16 USTs is completed
   ROL     BYTE1,Y                     ; shift left one bit
   BRSET   BYTE1,Y,#$80,D_BIT_FILL_1   ; even bit of one byte
   JSR     FILL_11                     ; = "0", fill "11"
   BRA     CHECK_BIT_LAST
D_BIT_FILL_1:
   JSR     FILL_1                      ; = "1", fill "1"
CHECK_BIT_LAST:
   INC     NO_BIT,X                    ; record no. of bit
   LDAA    NO_BIT,X
   CMPA    #$08
   BEQ     CHECK_END
   ROL     BYTE1,Y                     ; shift left one bit
   BRA     TRANS_BIT
```

```
CHECK_END:
    CLR     NO_BIT,X
    AIY     #$01
    CPY     #$06
    BEQ     CHECK_UST                  ; last byte
    LBRA    TRANS_BIT                  ; not yet, next byte
CHECK_UST:
    CMPB    #$0F
    BEQ     PUSH_UST                   ; 15, 16 USTs, no need
    CMPB    #$10                       ; to shift data left
    BEQ     PUSH_UST                   ; until available UST
                                       ; is msb, or do it
    STAB    FILL_NO,X
SHIFT_UST:
    ROLW    TEMP1,X                    ; shift left once
    BCLR    TEMP2,X,#$01               ; clear bit 0 of UST
    INCB
    CMPB    #$10                       ; 16 USTs
    BNE     SHIFT_UST
PUSH_UST:
    PSHM    Y
    LDY     NO_Y,X
    LDAA    TEMP1,X                    ; GET 16 USTs
    STAA    UST1,Y
    AIY     #$01
    LDAA    TEMP2,X
    STAA    UST1,Y
    CLRW    TEMP1,X
    CLRB
    STY     NO_Y,X
    PULM    Y
    LDAA    FILL_NO,X
    CMPA    #$0C
    BGT     ADD_08                  ; 12<NO_UST<16, add
                                    ; one byte = "08"
    CMPA    #$08
    LBGT    END                     ; 8<NO_UST<12, return
    CMPA    #$04
    BGT     FILL_08                 ; 4<NO_UST<8, fill
                                    ; last byte = "08"
    DEC     NO_Y,X                  ; 0<NO_UST<4, delete
                                    ; last byte
END:
    RTS
```

APPENDIX E

SUBROUTINE FOR PACKET BODY DECODING

```
TRANS_UST_B7:
   LDAA    UST1,Y               ; check if PRE_EOF appears
   CMPA    #$00
   BEQ     ADD_Y
   CMPA    #$FF
   BEQ     ADD_Y
   BRA     TRANS_UST
ADD_Y:
   INC     PRE_EOF_NO,X         ; record no. of PRE_EOF
   LDAA    PRE_EOF_NO,X
   CMPA    #$05
   BEQ     TRANS_UST            ; not finish
   AIY     #$01
TRANS_UST:
   JSR     CHECK_DATA           ; if one byte appears
   BRSET   UST1,Y,#$80,UST_INC_1 ; check UST
   JSR     INC_0                ; = "0", record it
   BRA     TRANS_UST_NEXT       ; translate next UST
UST_INC_1:
   JSR     INC_1                ; = "1", record it
TRANS_UST_NEXT:
   INC     NO_UST,X             ; record no. of UST
   LDAA    NO_UST,X
```

```
    CMPA     #$08
    BEQ      CHECK_UST_END
    ROL      UST1,Y                   ; get next UST
    BRA      TRANS_UST
CHECK_UST_END:
    CLR      NO_UST,X
    AIY      #$01
    CPY      #$06
    LBEQ     END                      ; finish, return
    LBRA     TRANS_UST_B7
END:
    RTS
```

APPENDIX F

NOISE DATA IN TABLE

Noise Data for Figure 7.2			
Noise Amplitude (V-rms)	CRC Error	FCS Error	Total Error
0.4	97	24	121
0.8	105	41	146
1.2	84	34	118
1.6	91	29	120
2.0	106	42	148
2.4	106	33	139
2.8	96	40	136
3.2	93	32	125
3.6	107	44	151
4.0	116	46	162
4.4	133	38	171

Noise Data for Figure 7.3			
Noise Amplitude (V-rms)	CRC Error	FCS Error	Total Error
0.04	19	12	31
0.08	25	18	43
0.12	49	18	67
0.16	63	21	84
0.20	72	25	97
0.24	60	17	77
0.28	83	38	121
0.32	74	31	105
0.36	86	35	121

Noise Data for Figure 7.4			
Noise Amplitude (V-rms)	CRC Error	FCS Error	Total Error
0.0	0	0	0
0.004	1	1	2
0.008	7	7	14
0.012	12	5	17
0.016	14	8	22
0.020	13	11	24
0.024	19	11	30
0.028	18	9	27
0.032	17	14	31
0.036	29	15	44

REFERENCES

[1] Davidson, Ken, "CEBus: A New Standard in Home Automation, " *Circuit Cellar Ink*, Aug./Sep., 1989, pp. 40-46,

[2] Davies, D. W., *Computer Networks and Their Protocols*, Chichester, New York: Wiley, 1983.

[3] *EIA Home Automation System (CEBus)*, EIA Engineering Department, Oct. 1992.

[4] *CEBus Spread Spectrum Power Line Modem*, Intellon Corporation, 1992.

[5] *M68HC16Z1EVB User's Manual*, Motorola Inc., 1991.

[6] *M68HC16Z1 Technical Summary*, Motorola Inc., 1992.

[7] *M68HC16 Reference Manual*, Motorola Inc., 1991.

[8] *General Purpose Timer Reference Manual*, Motorola Inc., 1991.

[9] *Queued Serial Module Reference Manual*, Motorola Inc., 1991.

[10] *Toolware M68HC16 Macro Assembler User's Manual for MS-DOS*, Motorola Inc., 1991.

[11] *Turbo C Reference Guide*, Borland International Inc., 1987.

[12] *Turbo C User's Guide*, Borland International Inc., 1987.

[13] Barkakati, Nabajyti, *The Waite Group's Turbo C Bible*, Howard W. Sams & Company, Indianapolis, Indiana, 1989.

[14] Khawand, Jean; Douligeris, Christos; Khawand, Charbel, "A Physical Layer Implementation for a Twisted Pair Home Automation System ", *IEEE Transactions on Consumer Electronics*, Vol. 38, No. 3, August 1992, pp. 530-536.

[15] *IS-60.08 Common Application Language (CAL), Part1: CAL Specification*, EIA Engineering Department, May 1994.

[16] *IS-60.08 Common Application Language (CAL), Part2: CAL Context Description*, EIA Engineering Department, May 1994.