Application Note
C-BUS Coding Example using the CMX868A

AN/General/C-BUS/Coding/2 April 2007

1 Introduction

CML Microcircuits offers a wide variety of highly integrated products for wireless and wireline communications, and many of these products utilize a simple serial microcontroller interface named "C-BUS". The C-BUS protocol is easily implemented, but example C-BUS programs can be useful for designers who are not familiar with this type of interface.

The purpose of this document is to provide example programs that can be used to control a C-BUS compatible device. The CMX868A V.22 bis wireline modem IC utilizes a C-BUS interface and will be used as the platform for the programs listed in this document. While these programs specifically reference the CMX868A, the programs themselves are directly applicable to any C-BUS product manufactured by CML Microcircuits.

The CMX868A datasheet should be reviewed while reading this application note.

TABLE OF CONTENTS

2 The C-BUS Interface

The C-BUS interface is a simple serial interface that uses five wires to exchange configuration settings and data with the host processor. The C-BUS interface is very similar to the SPI interface developed by Motorola, and a comparison of the C-BUS and SPI pins is provided in the following table:

C-BUS Interface Line	Corresponding SPI Interface Line
Serial Clock (SCLK)	Serial Clock (SCLK)
Interrupt Request (IRQN)	None
Chip Select (CSN)	Slave Select (/SS)
Command Data (CDATA)	Master Out Slave In (MOSI)
Reply Data (RDATA)	Master In Slave Out (MISO)

Table 1: C-BUS and SPI Comparison

There is only one significant difference between the two schemes. C-BUS is a half-duplex protocol, but SPI allows full-duplex communications between the host processor and the peripheral.

A typical C-BUS information exchange ("transaction") consists of the following actions:

- CSN line is pulled low (asserted) and SCLK is started.
- Address of register to be written to/read from is written on the CDATA line.
- The next step depends on whether a write or read operation is to be performed:
    - For write operation, one or two bytes of configuration settings are written to CDATA line.
    - For read operation, one or two bytes of information are read out on the RDATA line.
- CSN line is pulled high (de-asserted) and SCLK can be stopped. The C-BUS transaction is now complete.

The C-BUS protocol transmits all information in "most significant bit" format. For more information on C-BUS, the reader is referred to the "C-BUS Microcontroller Interface" application note on the CML Microcircuits' website.

3 The CMX868A V.22 bis Wireline Modem

The CMX868A is a highly integrated V.22 bis modem IC that supports operation at speeds of 2400bps and below. The rich feature set of the CMX868A includes the following:

- 2400bps operation (V.22 bis)
- 1200bps operation (V.22/Bell 212A, V.23/Bell 202)
- 300bps operation (V.21/Bell 103)
- DTMF Tx/Rx with high-performance DTMF decoder
- User-Defined Tone Tx/Rx
- Call Progress Tone Detection
- Ring Detector
- Hook Switch Relay Driver

The CMX868A was chosen for this project because it uses a C-BUS interface that is representative of the C-BUS ports used on most CML products. The programs listed in this document can easily be modified to work with any CML C-BUS device. The primary modification needed to allow these programs to work with other C-BUS parts is to modify the register definition section, at the beginning of each program, to reflect the chosen C-BUS device.

Consideration should also be given to the size of the registers being written to/read from. All CMX868A registers, with the exception of TxData and RxData registers, are 16-bit registers. Functions for 8-bit register writes and reads are not used in these programs but are provided for completeness.

4 Sample Programs

There are many C-BUS functions that need to be performed in a typical design, and two programs are provided in this section that demonstrate these functions. These sample programs are also available as text file downloads from the CML Microcircuits' website.

Each of these programs were written in the C programming language for a simple 8051 host processor (Atmel AT89C2051). The 8051 architecture was chosen because it is widely available and understood. The programs were written in C to enhance the portability of the programs from one processor to another. The programs were compiled using a C compiler (V7.5) from Keil.

C-BUS communications can be performed with a dedicated SPI port or by "bit banging" with general purpose I/O pins. The bit banging approach is slightly more involved and is used in both of the programs in this document.

The AT89C2051 processor uses 12 clock cycles per instruction cycle, so the CMX868A's C-BUS interface timings were easily met with no special considerations. Faster processors, however, may require delays to be implemented at certain points of the C-BUS transaction. Some of these points

have been identified in the sample programs, but the user is encouraged to review the required C-BUS timings in their respective CML product's datasheet.

4.1 Program 1: Programmed Tone Tx

The first program, "Programmed Tone Tx", demonstrates many common C-BUS functions. In addition to properly initializing the CMX868A, this program configures the CMX868A to generate one of four different custom tones. The act of programming these custom tones can be summarized as follows:

- Check CMX868A Status Register to ensure "Programming Flag" bit is set to 1.
- Once Programming Flag bit is set to 1, write a value to the 16-bit Programming Register.
- Repeat these two steps until all necessary values have been programmed.

After the custom tone configuration is complete, the program places the CMX868A in programmed tone transmit mode, and an infinite loop is entered to complete the program.

The CMX868A registers are defined using "#define" statements at the beginning of the program. Not all of the CMX868A registers are accessed in this particular program.

```
//PROJECT:        Programmed Tone Tx with CMX868A
//
//PROCESSOR:      Atmel AT89C2051
//
// This code programs the CMX868A with four different "tone pair" values.
// This code uses bit-banging over CBUS.
//
// Microcontroller & CMX868A use separate 11.0592MHz xtals
//
// Code occupies 670 bytes of program memory.

#include <REG2051.H>

//CMX868A Register Definitions
#define RESET                 0x01
#define GENERALCONTROL        0xE0
#define TXMODE                      0xE1
#define RXMODE                0xE2
#define TXDATA1               0xE3
#define TXDATA2               0xE4 //for V.14 implementation
#define RXDATA                0xE5
#define STATUS                0xE6
#define PRGREG                0xE8

#define POWERUP               0x1580 //Pwrup and Reset bits to 1 after
power is first
                                     //applied

//Function Declarations
void generalreset(void);
void wr_byte(unsigned char byte);
void wr1(unsigned char address, unsigned char databyte);
void wr2(unsigned char address, unsigned int dataword);
unsigned char rd_byte (void);
unsigned char rd1(unsigned char address);
unsigned int rd2(unsigned char address);
void initialize_uC(void);
void initialize_868(void);
void programtonetx(void);

//C-BUS to Microcontroller Pin Mapping
sbit CSN    = P1^7;
sbit CDATA  = P1^6;
sbit SCLK   = P1^5;
sbit RDATA  = P1^4;
sbit IRQ    = P3^3;                //this uC pin selected because it is an
external IRQ pin

bit pwrupdelay=0, CMX868AIRQ;       //flags
```

```
//****************************************
// FUNCTION: void initialize_uC()
//
// PURPOSE: Configure the microcontroller.
//
// DESCRIPTION: This function neither takes in nor returns a variable to
// the calling routine.
//
// Timer1 interrupt is enabled so it can be used for the CMX868A powerup
// delay.  Timer1 is placed in
// 16-bit timer mode and loaded with 0xB800 to cause a 20ms delay.
//****************************************
void initialize_uC()
{
      IE=0x8C;          //interrupts enabled; global IRQ, EX1, T1
      IT1=1;                  //EX1 configured as falling-edge triggered
IRQ
      IRQ=1;                  //write 1 to EX1 to configure as input
      TMOD=0x10;        //Timer1 - 16bit timer mode
      TH1=0xB8;         //0xB800=18432counts=20ms delay
      TL1=0x00;
}

//****************************************
// FUNCTION: void ex1() interrupt 2
//
// PURPOSE: /INT1 interrupt service routine (ISR).
//
// DESCRIPTION: This function neither takes in nor returns a variable to
// the calling routine.
//
// When a falling-edge signal is sensed on /INT1, this routine sets the
// CMX868AIRQ flag to 1.
// Control then reverts to the previously running function.
//****************************************
void ex1() interrupt 2
{
 CMX868AIRQ=1;
}

//****************************************
// FUNCTION: void timer1() interrupt 3
//
// PURPOSE: Timer1 interrupt service routine (ISR).
//
// DESCRIPTION: This function neither takes in nor returns a variable to
// the calling routine.  When Timer1 overflows and interrupts, this routine
// sets the pwrupdelay flag to 1.
// Control then reverts to the previously running function.
//****************************************
void timer1() interrupt 3
{
 pwrupdelay=1;
}
```

```
//****************************************
// FUNCTION: void initialize_868()
//
// PURPOSE: Properly powerup the CMX868A.
//
// DESCRIPTION: This function neither takes in nor returns a variable to
// the calling routine.
//
// A General Reset is first written to the CMX868A, after which a 2-byte
// write is performed to the General Control register with Pwrup & Reset
// bits = 1.  A 20ms delay is then observed, and control is passed back to
// the calling routine once the delay is complete.
// At this point the CMX868A can be loaded as necessary for operation.
//****************************************
void initialize_868()
{
  generalreset();
  wr2(GENERALCONTROL, POWERUP);     //write 1s to Pwrup & Reset bits

  TR1=1;                            //turn on uC Timer1 for 20ms delay
  while(pwrupdelay!=1);                   //wait for xtal osc to stabilize
  TR1=0;                            //turn off uC Timer1

//CMX868A is now powered up and ready for configuration
}

//****************************************
// FUNCTION: void wr_byte(unsigned char byte)
//
// PURPOSE: Write a single byte to the CMX868A CDATA pin.
//
// DESCRIPTION: This function takes in a single-byte corresponding to the
// value to be written to the CDATA line.
//
// A 'for' loop is started and the SCLK line is pulled low.  The msb of the
// byte is obtained and written to CDATA.  The byte is left-shifted one
// place and SCLK is pulled high to latch the bit into the CMX868A.  The
// 'for' loop iterates 7 more times to complete the byte write operation.
//
// This function does not return a variable.
//****************************************
void wr_byte(unsigned char byte)
{
 unsigned char i;

 for(i=8; i!=0; i--)
  {
   SCLK=0;
   if(byte & 0x80)
     CDATA=1;
   else
     CDATA=0;
   byte <<= 1;
   SCLK=1;
   //fast processors may need a delay here
  }
}
```

```
//****************************************
// FUNCTION: void wr1(unsigned char address, unsigned char databyte)
//
// PURPOSE: Write one byte to a CMX868A write register.
//
// DESCRIPTION: This function takes in two single-byte variables
// corresponding to the register address and register contents to be
// written to the CDATA line.
//
// CSN is taken low to start the C-BUS transaction.  The address byte is
// written to the CDATA line,and then the register contents are written to
// CDATA.  CSN is taken high to complete the C-BUS transaction.
//
// This function does not return a variable.
//****************************************
void wr1(unsigned char address, unsigned char databyte)
{
 CSN=0;
    //fast processors may need a delay to observe Tcse
 wr_byte(address);
    //fast processors may need a delay to observe Tnxt
 wr_byte(databyte);
    //fast processors may need a delay to observe Tcsh
 CSN=1;
}

//****************************************
// FUNCTION: void wr2(unsigned char address, unsigned int dataword)
//
// PURPOSE: Write two bytes to a CMX868A write register.
//
// DESCRIPTION: This function takes in three variables; a single-byte
// variable corresponding to the register address, and a two-byte variable
// corresponding to the register contents, all of which will be written to
// the CDATA line.
//
// CSN is taken low to start the C-BUS transaction.  The address byte is
// written to the CDATA line. The register contents are split into two
// single-byte variables, both of which are written to CDATA. CSN is then
// taken high to complete the C-BUS transaction.
//
// This function does not return a variable.
//****************************************
void wr2(unsigned char address, unsigned int dataword)
{
 unsigned char temp;

 CSN=0;
   //fast processors may need a delay to observe Tcse
 wr_byte(address);
   //fast processors may need a delay to observe Tnxt
 temp=dataword;         //get LSB
 dataword >>= 8;  //shift most significant 8 bits down for MSB
 wr_byte(dataword);     //write MSB
   //fast processors may need a delay to observe Tnxt
 wr_byte(temp);         //write LSB
   //fast processors may need a delay to observe Tcsh
 CSN=1;
}
```

```
//****************************************
// FUNCTION: unsigned char rd_byte(void)
//
// PURPOSE: Read a single byte from the CMX868A RDATA pin.
//
// DESCRIPTION: This function does not take in a variable.
//
// The SCLK line is pulled low and a 'for' loop is started.  The SCLK line
// is pulled high to latch out the latest data bit, and this bit is
// appended to any previously received bits. The received bits are left-
// shifted one place and SCLK is pulled low. (The left-shift is at the top
// of the loop due to the decrementing counter.) The 'for' loop iterates 7
// more times to complete the read operation.
//
// This function returns the received byte to the calling routine.
//****************************************
unsigned char rd_byte(void)
{
 unsigned char byte=0x00, i;

 for(i=8; i != 0; i--)
  {
   byte <<= 1;
   SCLK=1;
   if(RDATA)            //append 1 to byte if RDATA=1
      byte |= 0x01;
   else                 //else, append 0 to byte
      byte &= 0xFE;
   SCLK=0;
   //fast processors may need a delay here
  }
 return(byte);
}

//****************************************
// FUNCTION: unsigned char rd1(unsigned char address)
//
// PURPOSE: Read a single byte from a CMX868A read register.
//
// DESCRIPTION: This function receives a variable corresponding to the
// register address to be read.
//
// The CSN line is pulled low to start the C-BUS transaction.
// The read register address is written to the CMX868A, and then the
// register contents are read out. The C-BUS transaction ends when the CSN
// line is pulled high.
//
// This function returns the received byte to the calling routine.
//****************************************
unsigned char rd1(unsigned char address)
{
 unsigned char rbyte;

 CSN=0;
  //fast processors may need a delay to observe Tcse
 wr_byte(address);
  //fast processors may need a delay to observe Tnxt
 rbyte=rd_byte();
  //fast processors may need a delay to observe Tcsh
 CSN=1;
 return(rbyte);
```

```
}

//****************************************
// FUNCTION: unsigned int rd2(unsigned char address)
//
// PURPOSE: Read two bytes from a CMX868A read register.
//
// DESCRIPTION: This function receives a variable corresponding to the
// register address to be read.
//
// The CSN line is pulled low to start the C-BUS transaction.
// The read register address is written to the CMX868A, and then the
// register contents are read out.
// The register contents are read out 8-bits at a time and are combined to
// form the 16-bit word.
// The C-BUS transaction ends when the CSN line is pulled high.
//
// This function returns the received word (16 bits) to the calling
routine.
//****************************************
unsigned int rd2(unsigned char address)
{
 unsigned int rword=0x0000;

 CSN=0;
 //fast processors may need a delay to observe Tcse

 wr_byte(address);
 //fast processors may need a delay to observe Tnxt
 //SCLK=1 at end of wr_byte

 SCLK=0;          //RDATA becomes active here

 rword = rd_byte();     //8 bits returned into 16-bit variable (only least
8 significant bits copied)

 rword <<= 8;           //left-shift bits into most significant position

 rword |= rd_byte();          //append next 8 bits onto existing 16-bit
variable

 CSN=1;
 return(rword);
}

//****************************************
// FUNCTION: void generalreset(void)
//
// PURPOSE: Issue General Reset to CMX868A.
//
// DESCRIPTION: This function neither receives nor takes in a variable.
//
// The CSN line is pulled low to start the C-BUS transaction.
// The General Reset byte (0x01) is written to the CMX868A.
// The C-BUS transaction ends when the CSN line is pulled high.
//****************************************
void generalreset(void)   //issues General Reset to chip
{
 CSN=0;
 //fast processors may need a delay to observe Tcse
 wr_byte(RESET);
```

```
 CSN=1;
}

//*************************************
// FUNCTION: void programtonetx(void)
//
// PURPOSE: Write to the Programming Register to configure user-defined Tx
tones.
//
// DESCRIPTION: The function does not take in a variable from the calling
// routine. Variables are declared, including an 17-member int array that
//  contains the values to be written to the Programming Register.
//
// A loop is entered that first waits until the Programming Flag in the
// Status Register is set.  Once this bit is set, a value is written to the
// Programming Register and the loop iterates.  This loop runs 17 times to
// allow 17 values to be written to the Programming Register.  If not all
// Tone Pairs are required to be programmed, this loop can be shortened as
// needed.
//
// No variable is returned to the calling function.
//*************************************
void programtonetx(void)
{
 unsigned char i;
 unsigned int pgmtonetxwords[17]={0x8000,             //Increment pointer
            0x12AC, 0x24A2, 0x0000, 0x0000,     //Tone pair A (one tone),
            1400Hz @ 0.5Vrms
            0x1556, 0x24A2, 0x0000, 0x0000,     //Tone pair B (one tone),
            1600Hz @ 0.5Vrms
            0x1EAC, 0x24A2, 0x0000, 0x0000,     //Tone pair C (one tone),
2300Hz @ 0.5Vrms
            0x2157, 0x24A2, 0x0000, 0x0000};    //Tone pair D (one tone),
      2500Hz @ 0.5Vrms

 CMX868AIRQ=0;
 rd2(STATUS);                              //read Status register to clear
any IRQs
 for(i=0; i<17; i++)                       //17 writes required to program
all Tx tones
  {
   wr2(PRGREG, pgmtonetxwords[i]);
   while(CMX868AIRQ!=1);            //wait for IRQ, Pgm Flag is only IRQ
that can happen here
   CMX868AIRQ=0;              //reset IRQ flag bit
   rd2(STATUS);                     //read Status register to enable next
IRQ
  }
}

//*************************************
// FUNCTION: void main()
//
// PURPOSE: Main routine of program.
//
// DESCRIPTION: After the variables are declared, the uC and CMX868A are
// initialized.
// The General Control register is initialized, and the Programming
// Register is loaded for programmed tone tx.  Next, the Tx Mode Register
// is configured for programmed tone
// Tx, and the program then enters an infinite loop.
```

```
//****************************************
void main()
{
   unsigned int status=0;

   initialize_uC();
   initialize_868();

   wr2(GENERALCONTROL, 0x1550);     //11.0592MHz xtal, normal pwr, IRQ
enabled, Pgm Flag IRQ enabled
   programtonetx();
   wr2(TXMODE, 0x1E0F);             //Tx tone pair TD, 0dB attenuation.

   while(1);
}
```

4.2 Program 2: Programmed Tone Rx

The “Programmed Tone Rx” program begins by properly initializing the CMX868A, and then the Programming Register is loaded with values for 1400Hz and 2300Hz tone detection. The act of programming the CMX868A with custom tones (for detection) can be summarized as follows:

- Check CMX868A Status Register to ensure “Programming Flag” bit is set to 1.
- Once Programming Flag bit is set to 1, write a value to the 16-bit Programming Register.
- Repeat these two steps until all necessary values have been programmed.

(Note: the programmed values corresponding to 1400Hz and 2300Hz tones were derived by the “Filter Coefficient Generator” spreadsheet, which is available in the “Application Notes” area of the CML Microcircuits’ website.)

After the custom tone configuration is complete, the program places the CMX868A in programmed tone detection mode, and programmed tone detection IRQs are enabled. An infinite loop is then entered but no functions are performed in this loop.

The CMX868A registers are defined using “#define” statements at the beginning of the program. Not all of the CMX868A registers are accessed in this particular program. The CMX868A register mapping should be replaced with the register mapping of the desired CML device.

```
//PROJECT:        Programmed Tone Rx with CMX868A
//
//PROCESSOR:      Atmel AT89C2051
//
// This code programs the CMX868A with four different "tone pair" values.
//
// This code uses bit-banging over CBUS.
//
// Microcontroller & CMX868A use separate 11.0592MHz xtals
//
// Code occupies 695 bytes of program memory.

#include <REG2051.H>

//CMX868A Register Definitions
#define RESET                 0x01
#define GENERALCONTROL              0xE0
#define TXMODE                      0xE1
#define RXMODE                0xE2
#define TXDATA1               0xE3
#define TXDATA2               0xE4 //for V.14 implementation
#define RXDATA                0xE5
#define STATUS                0xE6
#define PRGREG                0xE8

#define POWERUP               0x1580 //Pwrup and Reset bits to 1 after
power is first
                                    //applied

//Function Declarations
void generalreset(void);
void wr_byte(unsigned char byte);
void wr1(unsigned char address, unsigned char databyte);
void wr2(unsigned char address, unsigned int dataword);
unsigned char rd_byte (void);
unsigned char rd1(unsigned char address);
unsigned int rd2(unsigned char address);
void initialize_uC(void);
void initialize_868(void);
void programtonerx(void);

//C-BUS to Microcontroller Pin Mapping
sbit CSN    = P1^7;
sbit CDATA  = P1^6;
sbit SCLK   = P1^5;
sbit RDATA  = P1^4;
sbit IRQ    = P3^3;                 //this uC pin selected because it is an
external IRQ pin

bit pwrupdelay=0, CMX868AIRQ=0;     //flags
```

```
//****************************************
// FUNCTION: void initialize_uC()
//
// PURPOSE: Configure the microcontroller.
//
// DESCRIPTION: This function neither takes in nor returns a variable
// to the calling routine.
//
// Timer1 interrupt is enabled so it can be used for the CMX868A
// powerup delay.  Timer1 is placed in 16-bit timer mode and loaded
// with 0xB800 to cause a 20ms delay.
//****************************************
void initialize_uC()
{
      IE=0x8C;          //interrupts enabled; global IRQ, EX1, T1
      IT1=1;                  //EX1 configured as falling-edge triggered
IRQ
      IRQ=1;                  //write 1 to EX1 to configure as input
      TMOD=0x10;        //Timer1 - 16bit timer mode
      TH1=0xB8;         //0xB800=18432counts=20ms delay
      TL1=0x00;
}

//****************************************
// FUNCTION: void ex1() interrupt 2
//
// PURPOSE: /INT1 interrupt service routine (ISR).
//
// DESCRIPTION: This function neither takes in nor returns a variable
// to the calling routine.
//
// When a falling-edge signal is sensed on /INT1, this routine sets the
// CMX868AIRQ flag to 1.
// Control then reverts to the previously running function.
//****************************************
void ex1() interrupt 2
{
 CMX868AIRQ=1;
}

//****************************************
// FUNCTION: void timer1() interrupt 3
//
// PURPOSE: Timer1 interrupt service routine (ISR).
//
// DESCRIPTION: This function neither takes in nor returns a variable
// to the calling routine.
//
// When Timer1 overflows and interrupts, this routine sets the pwrupdelay
// flag to 1.
// Control then reverts to the previously running function.
//****************************************
void timer1() interrupt 3
{
 pwrupdelay=1;
}
```

```
//****************************************
// FUNCTION: void initialize_868()
//
// PURPOSE: Properly powerup the CMX868A.
//
// DESCRIPTION: This function neither takes in nor returns a variable
// to the calling routine.
//
// A General Reset is first written to the CMX868A, after which a 2-byte
// write is performed to the General Control register with Pwrup & Reset
// bits = 1.  A 20ms delay is then observed and control is then passed back
// to the calling routine.  At this point the CMX868A can be loaded as
// necessary for operation.
//****************************************
void initialize_868()
{
  generalreset();
  wr2(GENERALCONTROL, POWERUP);     //write 1s to Pwrup & Reset bits

  TR1=1;                            //turn on uC Timer1 for 20ms delay
  while(pwrupdelay!=1);                   //wait for xtal osc to stabilize
  TR1=0;                            //turn off uC Timer1

//CMX868A IS NOW POWERED UP AND READY FOR CONFIGURATION
}


//****************************************
// FUNCTION: void wr_byte(unsigned char byte)
//
// PURPOSE: Write a single byte to the CMX868A CDATA pin.
//
// DESCRIPTION: This function takes in a single-byte corresponding to the
// value to be written to the CDATA line.
//
// A 'for' loop is started and the SCLK line is pulled low.  The msb of the
// byte is obtained and written to CDATA.  The byte is left-shifted one
// place and SCLK is pulled high to latch the bit into the CMX868A.  The
// 'for' loop iterates 7 more times to complete the byte write operation.
//
// This function does not return a variable.
//****************************************
void wr_byte(unsigned char byte)
{
 unsigned char i;

 for(i=8; i!=0; i--)
  {
   SCLK=0;
   if(byte & 0x80)
     CDATA=1;
   else
     CDATA=0;
   byte <<= 1;
   SCLK=1;
   //fast processors may need a delay here
  }
}
```

```
//****************************************
// FUNCTION: void wr1(unsigned char address, unsigned char databyte)
//
// PURPOSE: Write one byte to a CMX868A write register.
//
// DESCRIPTION: This function takes in two single-byte variables
// corresponding to the register address and register contents to be
// written to the CDATA line.
//
// CSN is taken low to start the C-BUS transaction.  The address byte is
// written to the CDATA line, and then the register contents are written to
// CDATA.  CSN is taken high to complete the C-BUS transaction.
//
// This function does not return a variable.
//****************************************
void wr1(unsigned char address, unsigned char databyte)
{
 CSN=0;
    //fast processors may need a delay to observe Tcse
 wr_byte(address);
    //fast processors may need a delay to observe Tnxt
 wr_byte(databyte);
    //fast processors may need a delay to observe Tcsh
 CSN=1;
}


//****************************************
// FUNCTION: void wr2(unsigned char address, unsigned int dataword)
//
// PURPOSE: Write two bytes to a CMX868A write register.
//
// DESCRIPTION: This function takes in three variables; a single-byte
// variable corresponding to the register address, and a two-byte variable
// corresponding to the register contents, all of which will be written to
// the CDATA line.
//
// CSN is taken low to start the C-BUS transaction.  The address byte is
// written to the CDATA line. The register contents are split into two
// single-byte variables, both of which are written to CDATA.
// CSN is then taken high to complete the C-BUS transaction.
//
// This function does not return a variable.
//****************************************
void wr2(unsigned char address, unsigned int dataword)
{
 unsigned char temp;

 CSN=0;
   //fast processors may need a delay to observe Tcse
 wr_byte(address);
   //fast processors may need a delay to observe Tnxt
 temp=dataword;         //get LSB
 dataword >>= 8;  //shift most significant 8 bits down for MSB
 wr_byte(dataword);     //write MSB
   //fast processors may need a delay to observe Tnxt
 wr_byte(temp);         //write LSB
   //fast processors may need a delay to observe Tcsh
 CSN=1;
}
```

```
//****************************************
// FUNCTION: unsigned char rd_byte(void)
//
// PURPOSE: Read a single byte from the CMX868A RDATA pin.
//
// DESCRIPTION: This function does not take in a variable.
//
// The SCLK line is pulled low and a 'for' loop is started.  The SCLK line
// is pulled high to latch out the latest data bit, and this bit is
// appended to any previously received bits. The received bits are left-
// shifted one place and SCLK is pulled low. (The left-shift is at the top
// of the loop due to the decrementing counter.)
// The 'for' loop iterates 7 more times to complete the read operation.
//
// This function returns the received byte to the calling routine.
//****************************************
unsigned char rd_byte(void)
{
 unsigned char byte=0x00, i;

 for(i=8; i != 0; i--)
  {
   byte <<= 1;
   SCLK=1;
   if(RDATA)            //append 1 to byte if RDATA=1
      byte |= 0x01;
   else                 //else, append 0 to byte
      byte &= 0xFE;
   SCLK=0;
   //fast processors may need a delay here
  }
 return(byte);
}

//****************************************
// FUNCTION: unsigned char rd1(unsigned char address)
//
// PURPOSE: Read a single byte from a CMX868A read register.
//
// DESCRIPTION: This function receives a variable corresponding to the
// register address to be read. The CSN line is pulled low to start the C-
// BUS transaction.
// The read register address is written to the CMX868A, and then the
// register contents are read out. The C-BUS transaction ends when the CSN
// line is pulled high.
//
// This function returns the received byte to the calling routine.
//****************************************
unsigned char rd1(unsigned char address)
{
 unsigned char rbyte;

 CSN=0;
  //fast processors may need a delay to observe Tcse
 wr_byte(address);
  //fast processors may need a delay to observe Tnxt
 rbyte=rd_byte();
  //fast processors may need a delay to observe Tcsh
 CSN=1;
 return(rbyte);
}
```

```
//****************************************
// FUNCTION: unsigned int rd2(unsigned char address)
//
// PURPOSE: Read two bytes from a CMX868A read register.
//
// DESCRIPTION: This function receives a variable corresponding to the
// register address to be read.
//
// The CSN line is pulled low to start the C-BUS transaction. The read
// register address is written to the CMX868A, and then the register
// contents are read out. The register contents are read out 8-bits at a
// time and are combined to form the 16-bit word. The C-BUS transaction
// ends when the CSN line is pulled high.
//
// This function returns the received word (16 bits) to the calling
// routine.
//****************************************
unsigned int rd2(unsigned char address)
{
 unsigned int rword=0x0000;

 CSN=0;
 //fast processors may need a delay to observe Tcse

 wr_byte(address);
 //fast processors may need a delay to observe Tnxt
 //SCLK=1 at end of wr_byte

 SCLK=0;          //RDATA becomes active here

 rword = rd_byte();     //8 bits returned into 16-bit variable (only least
8 significant bits copied)

 rword <<= 8;           //left-shift bits into most significant position

 rword |= rd_byte();          //append next 8 bits onto existing 16-bit
variable

 CSN=1;
 return(rword);
}




//****************************************
// FUNCTION: void generalreset(void)
//
// PURPOSE: Issue General Reset to CMX868A.
//
// DESCRIPTION: This function neither receives nor takes in a variable.
//
// The CSN line is pulled low to start the C-BUS transaction.
// The General Reset byte (0x01) is written to the CMX868A.
// The C-BUS transaction ends when the CSN line is pulled high.
//****************************************
void generalreset(void)
{
 CSN=0;
 //fast processors may need a delay to observe Tcse
 wr_byte(RESET);
```

```
 CSN=1;
}

//*************************************
// FUNCTION: void programtonerx(void)
//
// PURPOSE: Load values to the CMX868A Programming Register to configure
// user-defined Rx tones.
//
// DESCRIPTION: The function neither takes in nor returns a variable.
// Variables are declared including an 27-member integer array that
// contains the values to be written to the Programming Register.  These
// values configure the CMX868A for 1400Hz and 2300Hz tone detection.
// The Status register is read to clear any outstanding IRQs.  A loop is
// entered and the first value is written to the Programming Register.
// After waiting for an IRQ corresponding to "Programming Flag bit set",
// the Status register is read to clear the IRQ and the loop iterates.
// This loop runs 27 times to allow 27 values to be written to the
// Programming Register.
//*************************************
void programtonerx(void)
{
 unsigned char i;
 unsigned int pgmtonerxwords[27]={0x8001,                  //Increment
pointer
            0x0000, 0x017A, 0x7E85, 0x1E59,           //1400Hz starts
      here...
            0x5A0B, 0x0000, 0x017A, 0x7E85,
            0x1E59, 0x5A0B, 0x001C, 0x00B6,
            0x00CA,
            0x0000, 0x0180, 0x7E7F, 0x1DF0,           //2300Hz starts
here...
            0x7BF3, 0x0000, 0x0180, 0x7E7F,
            0x1DF0, 0x7BF3, 0x002E, 0x00B6,
            0x00CA};

 rd2(STATUS);                        //read Status register to clear
outstanding IRQs
 CMX868AIRQ=0;                //clear flag
 for(i=0; i<27; i++)                 //27 writes required to program both Rx
tones
  {
   wr2(PRGREG, pgmtonerxwords[i]);  //write value to Programming Register
   while(CMX868AIRQ!=1);            //wait for IRQ, only IRQ possible is
Programming Flag
   CMX868AIRQ=0;              //clear flag
   rd2(STATUS);                      //read Status register to clear
outstanding IRQs
  }
}
//*************************************
// FUNCTION: void main()
//
// PURPOSE: Main routine of program.
//
// DESCRIPTION: The uC and CMX868A are initialized.  The General Control
// register is initialized, and the Programming Register is loaded for with
// values for 1400Hz & 2300Hz detection.
// Next, the Rx Mode Register is configured for programmed tone receive,
// and the program enters an infinite loop.
//*************************************
```

```
void main()
{
   initialize_uC();
   initialize_868();

   wr2(GENERALCONTROL, 0x1550);     //11.0592MHz xtal, normal pwr, IRQ
enabled, Pgm Flag IRQ
                              //enabled
   programtonerx();
   wr2(RXMODE, 0x1E04);       //Programmed tone Rx, 0dB atten.
   wr2(GENERALCONTROL, 0x1543);     //1st & 2nd programmed tone detection
IRQ enabled
   while(1);                        //infinite loop
}
```

5 Conclusion

C-BUS is a serial microcontroller interface that is widely used in CML products. This document presented two simple programs that demonstrate typical C-BUS interactions with an 8051-class microcontroller. It is hoped that the information in this document will speed up C-BUS driver development for CML designs.

CML Microcircuits (UK) Ltd COMMUNICATION SEMICONDUCTORS	CML Microcircuits (USA) Inc. COMMUNICATION SEMICONDUCTORS	CML Microcircuits (Singapore) Pte Ltd COMMUNICATION SEMICONDUCTORS
Tel: +44 (0)1621 875500 **Fax:** +44 (0)1621 875600 **Sales:** sales@cmlmicro.com **Tech Support:** techsupport@cmlmicro.com	**Tel:** +1 336 744 5050 800 638 5577 **Fax:** +1 336 744 5054 **Sales:** us.sales@cmlmicro.com **Tech Support:** us.techsupport@cmlmicro.com	**Tel:** +65 62 888129 **Fax:** +65 62 888230 **Sales:** sg.sales@cmlmicro.com **Tech Support:** sg.techsupport@cmlmicro.com
- www.cmlmicro.com -		