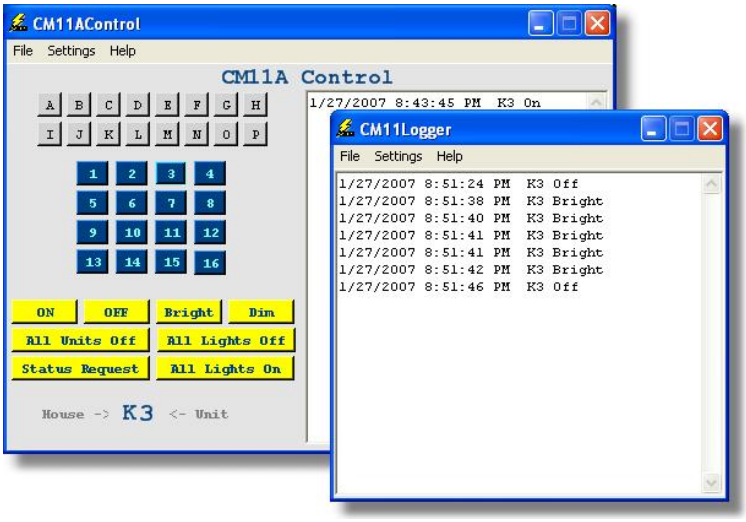
By Michael Simpson

X10 Home Automation
as seen in
April 2007 of Nuts & Volts Magazine

Pick up an issue at
http://www.nutsvolts.com



Last month we looked at interfacing your PC to various 1-Wire weather sensors. This month we are going to dive head first into home automation. I am going to show you how to use your PC to control lamps and appliances using the X10 protocol.

One of the most frustrating aspects for the PC to X10 interface is lack of quality documentation. For an industry that has so much demand, you would think that the manufacturers would create some quality manuals explaining in detail the protocol that their devices require. This is just not the case. The instructions for the various interface devices I tested went from nonexistent to poor. Most of the technical information I managed to get my hands on, was after many hours of searching the net. In many cases I had to use a lot of guess work to get the interfaces to work.

I will show you how to use three different RS232 interfaces. We will be looking at the FireCracker (CM17A), SmarHome PowerLinc (1132B), and the RCA ActiveHome (CM11A).

How does X10 Work?

It isn't necessary to know how the X10 protocol is transmitted over the power lines, but a brief overview may help you to troubleshoot problems. X10 uses a PLC technology, which stands for Power Line Carrier. A 1ms, 120Khz burst is transmitted near the zero crossing of the 60Hz AC signal. Two crossings are required to form a single bit. To generate "1" we need a burst at the first crossing and none at the second as shown in Figure 2. Figure 2 also shows the pattern is reversed for "0", we need a burst at the second crossing but not the first. Things would get out of sync very fast if we did not have some way of starting the whole data packet. A special start sequence is used. This sequence is three 120Khz bursts at consecutive crossings followed by no pulse as shown in Figure 3.

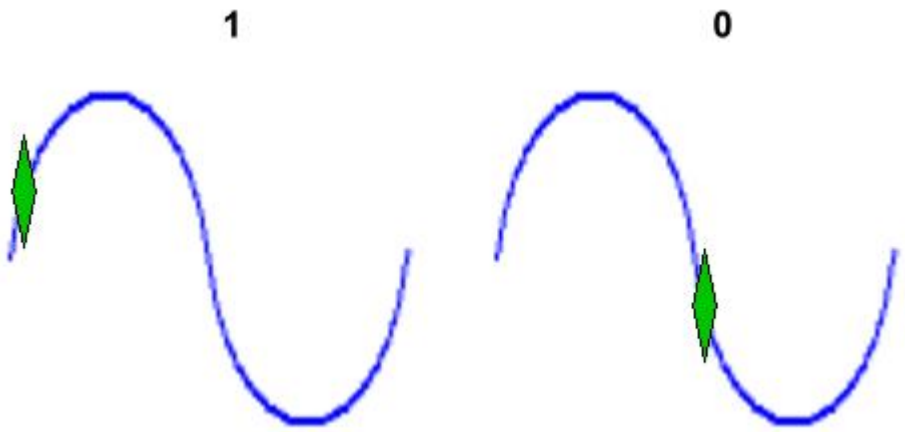


Figure 2

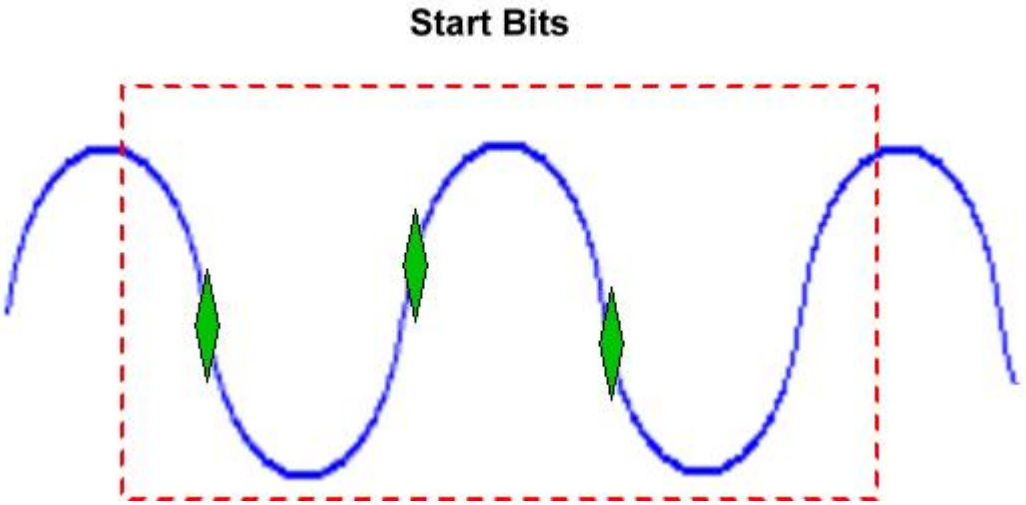


Figure 3

After the start sequence, the next 9 bits represent the actual data being transmitted. The first 4 bits are the house code. The house codes are represented by letters on the actual devices to make it easier on the consumer as shown in Table 1.

A	0110
B	1110
C	0010
D	1010
E	0001
F	1001
G	0101
H	1101
I	0111
J	1111
K	0011
L	1011
M	0000
N	1000
O	0100
P	1100

Table 1

After the house code, the next 5 bits represent the device code or function to perform as shown in Table 2. The last bit is actually used to indicate that the device is to perform a function if the bit is 1. The complete message from start to finish is sent twice for redundancy.

In reality, there are variations to the above protocol such as sending extended data, but I won't be getting into these in this article. Let's start to break down the various RS232 interfaces. Each has its own protocol and will translate your interface data into the X10 protocol I just talked about. It isn't necessary that you understand the actual details of the X10 protocol, just that you know how the House, Devices and Function codes are laid out.

All the previous examples show a single pulse that is synchronized with the zero crossing. In reality the X10 specification calls for three pulses to be transmitted to make the X10 compatible with a 3-Phase distribution system. The 2nd Pulse is sent 2.778ms after the zero crossing. The 3rd pulse is sent 5.556ms after the zero crossing.

1	01100
2	11100
3	00100
4	10100
5	00010
6	10010
7	01010
8	11010
9	01110
10	11110
11	00110
12	10110
13	00000
14	10000
15	01000
16	11000
All Units Off	00001
All Lights On	00011
On	00101
Off	00111
Dim	01001
Bright	01011
All Lights Off	01101
Extended Code	01111
Hail Request	10001
Hail Acknowledge	10011
Preset Dim	101-1
Extended Data	11001
Status On	11011
Status Off	11101
Status Request	11111

Table 2

Brief X10 History

Back in 1975 a company called Pico Electronics developed technology for what is now the X10 interface. It gets its name from the fact that it was the 10th project that Pico had created. The technology did not reach the market until 1978 when Radio Shack started to sell X10 products. Sears then started selling X10 products.

Pico formed a partnership with BSR and X10 Ltd was born. X10 Ltd bought out BSR's interest in 1987. The X10 Patent expired in 1997 and now X10 is an open standard.

FireCracker

Model CM17A

Figure 4

Let's start by looking at the FireCracker shown in Figure 4. Due to its price and simplicity, and the fact that it has been around for a long time, there are many of these in use today. The FireCracker is manufactured by X10 and can be purchased as an individual module or as part of the CM18A package. The cost for the CM17A will run you $12 to $40 depending on where you purchase.

The CM18A package shown in Figure 5 is priced you in the range of $22 to $39 and consists of the following modules:

- FireCracker Module (CM17A)
- Transceiver Module (TM751)
- Lamp Module (LM465)
- PalmPad Remote Control (HR12A)

Figure 5

I recommend the CM18A kit if you don't already have a wireless transceiver. You must have one for the FireCracker to operate. You can purchase the CM18A from X10.com but I found the website very difficult to navigate due to the overwhelming amount of popups and crazy flashing adds. I purchased a brand new CM18A package from one of the eBay online stores. The cost was only $22 and I received it in only 3 days.

The FireCracker is a small and compact module. Since it's wireless and gets its power from the RS232 signal, you can use it in portable interfaces such as a microcontroller or a Pocket PC. The only downside is that it is only a 1-way module. It does transmit only, so it cannot receive data from sensors or other controllers.

The FireCracker uses a standard DB9 connector with an RS232 interface. When I first started researching the FireCracker I had assumed that the TX and Ground lead on the DB9 connector were used at some predetermined baud rate. This is not the case. The FireCracker uses a 2-pin binary interface. The RTS and DTR leads are used to power and to clock-in the data.

To send a 1, RTS is high and DTR is low as shown in Figure 6. For a 0, DTR is high and RTS is low. Once signals are set the system should wait for 500-1000us then set both RTS and DTR high. The sequence is repeated 8 times for a complete byte.

To send a complete control signal to the FireCracker you need to send a sequence of 40 bits. This sequence consists of a header of 16 bits. Data consisting of 16 bits, and a footer of 8 bits.

The header is always 11010101 10101010 and the footer is always 10101101. As you might expect it's the 16 bits of data that tell the transceiver module what to send to the remote X10 devices.

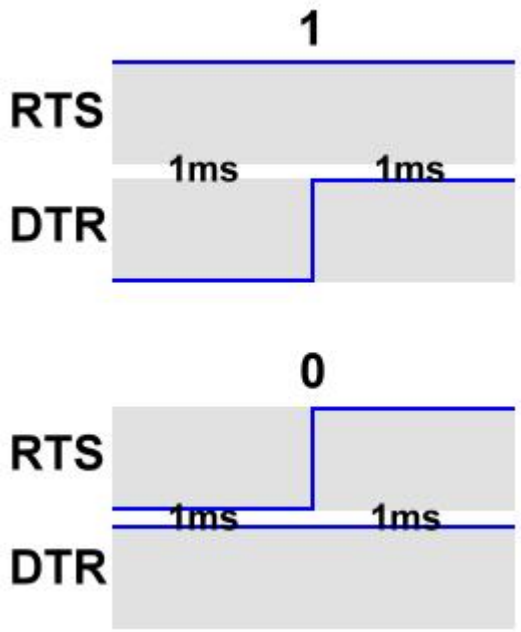


Figure 6

A	0110
B	0111
C	0100
D	0101
E	1000
F	1001
G	1010
H	1011
I	1110
J	1111
K	1100
L	1101
M	0000
N	0001
O	0010
P	0011

Table 3

Now you would expect the data portion of the FireCracker protocol to match that of the X10. Well it doesn't. Take a look at Table 3. The house codes bits are in reverse order.

If you look at Table 4 you can see that there is a pattern to the code/function sequence but it's not simple, so I just took the strong arm approach and listed all the combinations. In the downloads there is a Zeus file called FCm1.txt. This program has support functions called: SendX17Afooter, SendX17Aheader and SendX17Adata. The SendX17Adata makes a call to the header and footer functions so you need not worry about them. You simply pass the house code from Table 3 and the device/function code from Table 4.

1	0000 00000000
2	0000 00100000
3	0000 00010000
4	0000 00110000
5	0000 00001000
6	0000 00101000
7	0000 00011000
8	0000 00111000
9	0000 01000000
10	0000 01100000
11	0000 01010000
12	0000 01110000
13	0000 01001000
14	0000 01101000
15	0000 01011000
16	0000 01111000
9 On	0100 00000000
9 Off	0100 00100000
10 On	0100 00010000
10 Off	0100 00110000
11 On	0100 00001000
11 Off	0100 00101000
12 On	0100 00011000
12 Off	0100 00111000
13 On	0100 01000000
13 Off	0100 01100000
14 On	0100 01010000
14 Off	0100 01110000
15 On	0100 01001000
15 Off	0100 01101000
16 On	0100 01011000
16 Off	0100 01111000
Bright	0000 10001000
Dim	0000 10011000

Table 4

Here is an excerpt of the main loop in the FCm1.txt program. It will turn on Device K1 then wait 5 seconds and then turn it off again. The FCm1.txt program is very simple and will compile and run in the free ZeusLite compiler available on the KRMicros website.

Loop:

```
SendX17Adata(%1100, %000000000000)
pause(5000)
SendX17Adata(%1100, %000000100000)
pause(5000)

goto Loop
```

The **SendX17Adata** function will print the 1's and 0's as they are sent to the FireCracker so you can see what's happening.

Going Further with the FireCracker

I have given you the basics for interfacing to the FireCracker. The next logical step is to create a more intuitive program to allow you to pass actual strings like K1on or K1off. I have included a ZeusPro source file called **FireCracker1Pro.txt** as an example of how to get started.

I have also included a couple of programs called FireCracker2Pro and FireCracker3Pro that will allow you to use a small form to send codes to your FireCracker. Both the source and compiled exe's for windows as well as the Pocket PC exe are included in the download.

Last but not least is a program I call FireWorks shown in Figure 7. It was written and compiled with the ZeusPro compiler. This program will allow you to create script files that will turn lights and appliances on and off at various times. I have included the full ZeusPro source which includes documentation for the script editor. This is a work in progress and eventualy I want to create two-way versions for the other interfaces as well.

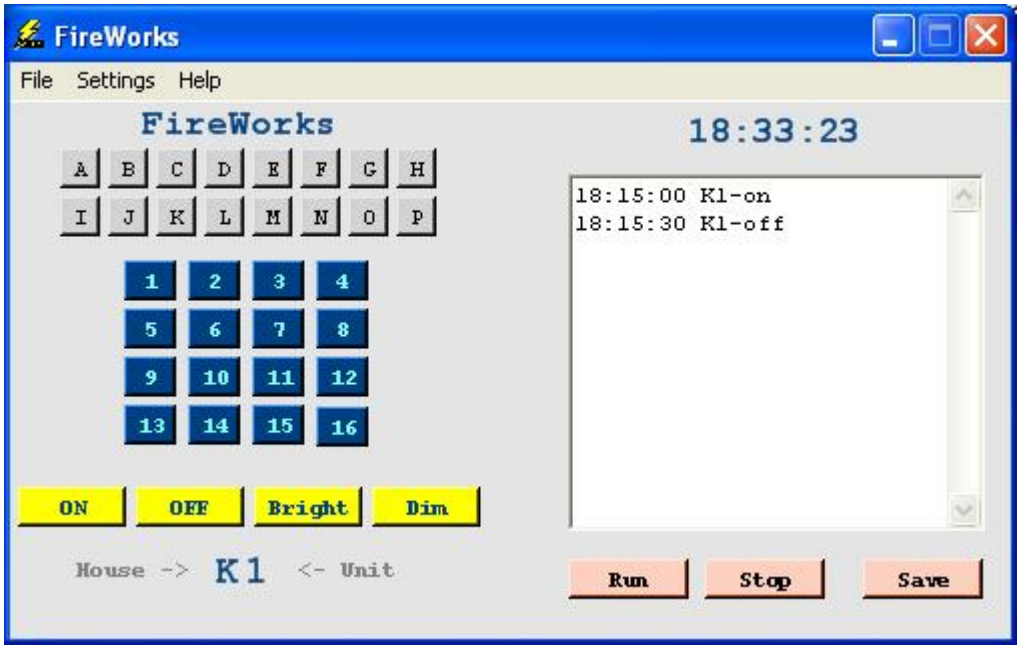


Figure 7

SmartHome PowerLinc

Model 1132B

Figure 8

The SmartHome 1132B shown in Figure 8 is a much more robust interface than the FireCracker. It offers full 2-way X10 communications as well as support for extended codes and data. You can purchase an 1132B for about $34 directly from SmartHome at www.smarthome.com.

The SmartHome 1132B has a more straight forward interface than that of the FireCracker. You communicate with the 1132B at 9600 baud, 8-bits and no parity using the standard RS232 RX and TX leads.

1132B Reception

Using a free copy of ZeusLite load up the following program:

```
'[PLm1.txt] PowerLinc 1132B interface
func main()
  dim x as integer
  dim curchar as integer

  x=ComOpen(1,baud=9600,port=3) '<------- Set Port here
  print "Open Status = ";x
  ComBGSuspend(1,1)


loop2:
  curchar = ComGetByte(1)
  if curchar = -1 then goto loop2

  print curchar
  goto loop2

endfunc
```

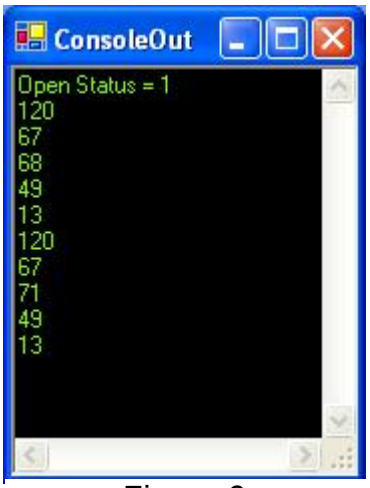


Figure 9

A	70
B	78
C	66
D	74
E	65
F	73
G	69
H	77
I	71
J	79
K	67
L	75
M	64
N	72
O	68
P	76

Table 5

In this example I issued the **K3-On** command using one of my X10 controllers. Each data packet reported on the 1132b begins with a 120 and ends with a 13 as shown in Figure 9. Excluding these bytes, the packet data actually consists of 3 bytes with the third always 49. The first data byte is the house code and returns the values shown in Table 5.

The second byte is the device/function code as shown in Table 6.

The 120 that starts the packet can also be replaced by other codes such as 88 which represent the response to an 1132B transmission. The 49 value at the end of the packet also represents the number of transmissions detected. From what I have found this is always 49.

	1	76
	2	92
	3	68
	4	84
	5	66
	6	82
	7	74
	8	90
	9	78
	10	94
	11	70
	12	86
	13	64
	14	80
	15	72
	16	88
Func 1	All Units Off	65
Func 2	All Lights On	67
Func 3	On	69
Func 4	Off	71
Func 5	Dim	73
Func 6	Bright	75
Func 7	All Lights Off	77
Func 8	Extended Code	79
Func 9	Hail Request	81
Func 10	PreSet Dim High	87
Func 11	PreSet Dim Low	85
Func 12	Extended Data	89
Func 13	Status On	91
Func 14	Status Off	93
Func 15	Status Request	95

Table 6

Looking at Figure 10, two packets were received. The first data packet sets the target device, in this case **K3**. The second packet sets the function **ON**.

Load up the included program PLm3.txt. I have created a couple of routines to make 1132B reception fool proof. The function called **check1132B** is called inside a loop and returns one of the following values:

0, No New Codes
1, New Device Set
2, Action Received

Figure 10

The following code snipit shows how you can access the three global variables HOUSE, DEVICE and ACTION each time a status of 2 is received from the check1132B function.

```
loop2:
  stat = check1132B()

  if stat = 2 then
    print HOUSE,DEVICE,"Action="+ACTION
  endif

  goto loop2
```

As you can see I have translated the raw data into usable values which could easily be translated into strings if need be. I have included a ZeusPro program called X10Logger.exe in the download shown in Figure 10. This program will allow you to monitor the traffic on your power line for X10 activity.

1132B Transmission

Transmitting to a device is just a little bit more complicated. The first thing you need to do is to send the code 02 to start the transmission. The 1132B will respond with an ACK consisting of 06 and 13 if it is available, and 21 if it's not.

X10 Command	99
House Code	See Table 5
Unit Code	See Table 6
Function Code	See Table 6
Repeat Code	65

Table 7

Once you receive the ACK you are free to transmit your device code data. Table 7 shows the sequence of bytes that need to be transmitted. The Repeat Code shown is 65, which tells the 1132B that we want to transmit the code one time. The 1132B supports up to 15 transmissions of the code, which gives this field a range of 65 - 79. In most of the code examples I have hard coded this field to 65.

Once you send the data packet the 1132B will respond with a data packet that we can capture and verify. The only difference is that the 120 start packet code is replaced with an 88 to let us know it's a command response.

Take a look at program **PLm4.txt** included with the downloads. In this program I added a function called **TX1132B()**. The command translates the house, unit, and function codes into the 1132B codes so that they can be transmitted. You simply pass the house code of 65-80 and the unit code 1-16 and the function you want to perform 1-16. The function codes start with the All Units On and proceed down the list as shown in Table 6.

After sending the code, the program calls the **check1132B()** function and displays the command results.

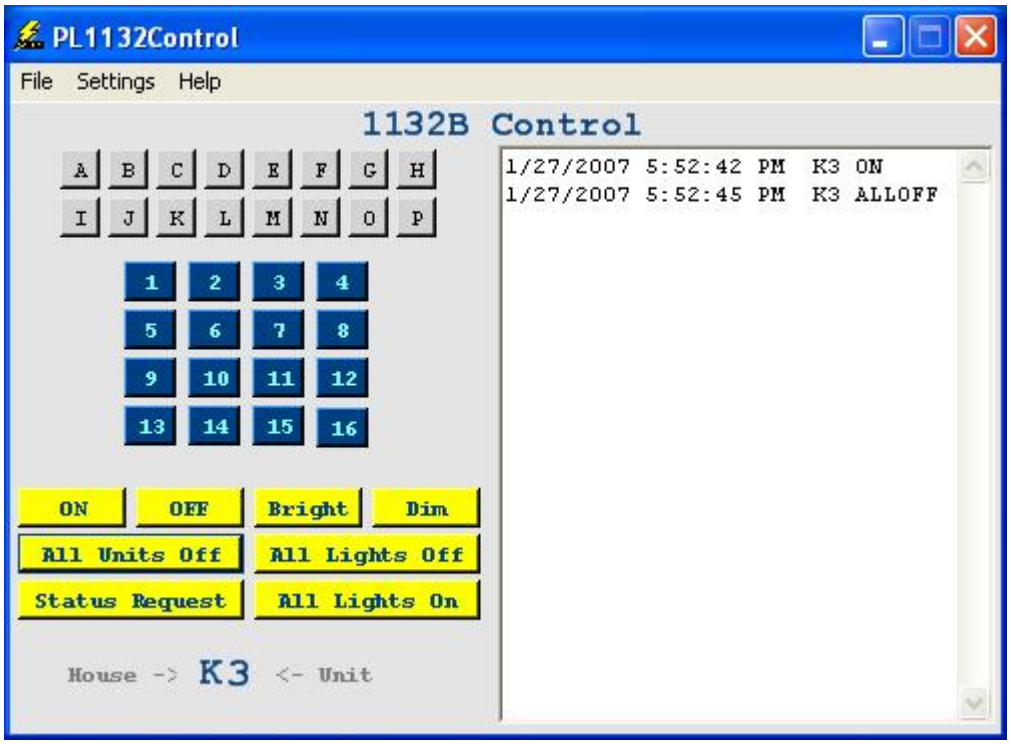


Figure 11

I have included yet another ZeusPro program called PL1132Control.exe shown in Figure 11. This program will let you control your X10 devices and will also log any activity.

RCA ActiveHome PC Interface

Model HC60CRX or CM11A

Figure 12

The RCA HC60CRX shown in Figure 12 is actually the very popular CM11A. You can still find both online. The one I purchased I got for $19 including shipping from an eBay online store. Of the three interfaces I am presenting in this article this is the most robust. It is also the most complicated.

Since the CM11A is much more complicated, I cannot detail every feature this little gem has to offer in the space of this article.

The interface is via a DB-9 cable and utilizes the standard RX and TX leads. The baud rate is 4800, 8-bits and no parity. The RI pin is also provided that is brought high when data is available to be read.

CM11A Reception

Load up the following program into ZeusLite and run it. Make sure you set the port that is connected to the DB-9 connector on the CM11A.

```
func main()
  dim x as integer
  x=ComOpen(1,baud=4800,port=3) '<----- Set Port here
  print "Open Status = ";x
  ComBGSuspend(1,1)

  dim curchar as integer

loop2:
  curchar = ComGetByte(1)
  if curchar = -1 then
    goto loop2
  endif

  print curchar

  goto loop2

endfunc
```

As soon as you plug the CM11A into a power outlet you will see the program start to print the code 165. This code is a request from the CM11A to set the internal timer. You won't be able to do anything else until you do.

In the program **CM11A2.txt** I have added a function called **setCM11CurTime()**. This function sets the 56 bit fields of the internal clock on the CM11A. I use the PC's internal clock so there is no need to pass any variables. You could place a call to the function at the start of your program after you open the com port. I however like to constantly look for the 165 code then make the call as needed. This way any time the CM11A loses power the PC will reset the timer. The CM11A has 2 batteries that I had assumed would keep the timer running. This does not seem to be the case. If power is removed then restored, the CM11A will still respond with the 165 code.

You may have noticed that once the timer is set and you send an X10 code with one of your remotes, you see the code 90 start to appear. This is the CM11A's way of telling you that data is ready to be pulled from the interface. In **CM11A3.txt** I added an additional **if** statement and when I see the code 90, I send back the code of 195 to tell the CM11 to transmit its information. The CM11A will transmit the 90 code once per second as long as data is in its buffer. In reality you need to pull data from the CM11A much faster than once a second.

Table 8 shows that up to 10 bytes may be received at one time. The very first byte tells you how many bytes follow. The second byte is a mask that tells you which of the data bytes are address bytes and which are function bytes. If bit 0 in the mask byte is a 1 then Data 0 represents a function byte. If it's 0 then Data 0 is an address byte. Bit 1 maps to Data 1 and so on for all 8 bits. There is one exception to this format: if a Dim or Bright function is received. The following byte represents the amount of the Dim or Bright. In my tests this value always seemed to be 25.

Byte 0	Number of Bytes in Packet
Byte 1	Function/Address Mask
Byte 2	Data 0
Byte 3	Data 1
Byte 4	Data 2
Byte 5	Data 3
Byte 6	Data 4
Byte 7	Data 5
Byte 8	Data 6
Byte 9	Data 7

Table 8

If the data is an address byte then the upper 4 bits are the house code and the lower 4 bits are the unit code. The house code maps directly to Table 1. The unit code maps to Table 2 minus the right-most bit as shown. If the data is a function byte the upper 4 bits maps to the house code and the lower 4 bits map to the function code (Tables 1 and 2).

In program **CM11A4.txt** I added a function called **checkCM11A()**. This function collects the data transmitted by the CM11A and then processes the packet once all the bytes in a single packet have been received. The function is called once every 10ms and if the 90 code is not found it sends the data request code of 165 regardless. This ensures that I can keep the CM11A buffer clear and process the commands in a timely manor.

CM11A Transmission

In order to transmit an X10 code you need to send two bytes. The first byte is the header byte and uses the format shown in Table 9.

Byte 1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Dim Amount					1	Func	Ext

Byte 2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
House Code				Unit/Function Code			

Table 9

The upper 5 bits are only used when doing a Dim or Bright function. It is supposed to direct how many times the CM11A transmits the command. In all the tests I have performed it looks like only bits 5-7 are used and will transmit from 1 to 7 times. In all my examples I leave all the Dim bits at 0. This will cause the Dim or Bright function to be issued once for each call. Bit-2 in the header is always 1. Bit-1 is set to 1 to flag a function (House/Function) code for the second byte in the transmission. If it's set to 0 the second byte is an address (House/Unit). Bit-0 is used to flag an extended format which I am not covering in this article, so it's set to 0.

After transmitting these two bytes you will receive an 8-bit checksum, and the CM11A expects you to send a 0 to indicate the checksum is a valid one. The CM11A will then respond with an 85, code which indicates that it is ready once again.

In program **CM11A5.txt** I added a function called **TXCM11A()**. Just like the **TX1132B()** function we created for the 1132B, you pass the house code 65-80, device code 1-16 and action code 1-16. The **TXCM11A()** function sends two command sequences to the CM11A. The first is an address sequence and the second is a function sequence.

One of the deficiencies of the CM11A interface is that if any data is in the interface buffer, the transmission of an X10 code will fail. At the very start of the **TXCM11A()** function I make a call to the **CollectCM11Adata()** function. This will empty the buffer and allow us to proceed with the transmission.

I have created two extra ZeusPro programs called **CM11Logger.exe** and **CM11AControl.exe** shown in Figure 1. Just like the 1132B, you can use them as a controller or logger for your X10 network.

X10 Performance Tips

A few notes on X10 Performance. Do not place X10 devices or interfaces on power strips. Many have suppression capacitors that will affect the performance of both. In some cases they may not work at all. The same goes for battery powered UPS systems. If you start to have problems with your X10 device or interface move it to a different location and try again. I don't recommend X10 systems for critical systems as it can be unreliable under certain circumstances. They do sell 2-way appliance and lamp modules to allow you to poll their status in order to verify they have the correct settings you have sent them.

Final Thoughts

All three of the interfaces presented here have their place. But for real automation you should look into the 1132B or the CM11A. It's a simple matter to take a very inexpensive X10 Motion Detector and automatically control lights or other devices upon entry or exit into a room.

In many cases we can't devote a PC to our home automation system or weather station, so starting next month I am going to start the process of migrating these projects to one or more microcontrollers. The microcontroller interface will also yield other advantages such as faster AtoD conversion and the ability to interface to more sensor types.

All the example programs as well as the source are available for download at:
http://www.kronosrobotics.com/Projects/x10.shtml

Links

Kronos Robotics
http://www.kronosrobotics.com/xcart/customer/home.php

KRMicros
http://www.krmicros.com/Development/ZeusPro/ZeusPro.htm

http://www.krmicros.com/Development/ZeusLite/ZeusLite.htm

SmartHome
http://www.smarthome.com/1132b.html

X10
http://www.x10.com