USING EXTRA INFRARED SIGNALS
(BY LOADING SIGNALS INTO TEST AREA)

Introduction

It is possible for a PC software program to load an infrared signal into a special "test area" in a HomeVision (or HomeVision-PC or HomeVision-Pro) controller. The program could then command the controller to transmit it. This provides a means for such programs to overcome the limit of 255 learned IR signals. This capability is beneficial mainly to those using custom software to control HomeVision, where the PC software decides what signals are to be transmitted. The information contained here is provided for those writing such software. A sample Visual Basic application is also available.

It's also possible for the HomeVision controller to send a serial command to the PC software telling it what signal it wants to transmit. The PC software could then load it into the controller and command the transmission. This would benefit more users, as the HomeVision controller would be in charge, not the PC software. This is not detailed here, but developers able to understand the other information should be able to figure it out rather easily.

Overview

The controller has a special test area that can hold one IR signal. It is normally used during the IR signal learning process, allowing the user to load the learned signal into the test area and then transmit it to verify it works. However, this test area can also be used during normal operation as a means for PC software to load and transmit a signal not already loaded in the controller. Here's an overview of how this would be used:

1. The user learns the desired IR signals with the HomeVision, HomeVision-PC, or HomeVision-Pro software.

2. The uses exports the IR signals to an ".IRL" file by using the "Export" option on the IR Signals Screen.

3. The user runs the custom software program. The program must first read all the IR signals from the ".IRL" file. When it needs to transmit an IR signal, it then loads the signal into the controller's test area, and then commands it to be transmitted. More details on the software operation are provided in the following section and Appendix A.

Software Operation

This section describes the basic steps the software must perform. Appendix A provides sample Visual Basic code for the main steps of reading, loading, and transmitting IR signals. The serial communications routines are only described at a top-level because they depend upon how the user chooses to implement the serial communications, and there are many ways to do this. The sample Visual Basic application shows one simple way to do it. Also, the VB application expands on the Appendix A code by including some error checking and user-interface functions.

HomeVision, HomeVision-PC, and HomeVision-Pro all store IR signals in their schedule files in the same format. However, when loading the signals into the controller, or when commanding the signal transmission, HomeVision-Pro uses a slightly different format than the others. The sample code shows both methods.

Following are the steps the software must perform:

1. Read all the IR signals from the ".IRL" file.

2. Load the signal into the controller's special test area.

3. Transmit the signal. This is done with the following command over the serial interface:

   ,_NNAABB

   where: ,_ is the command to transmit the IR test signal
   NN is the signal length (number of pulses in the signal)
   AA is 256 - carrier base on-time
   BB is 256 - carrier base off-time

   The TransmitSignal() function in Appendix A shows how to determine these values.

The above approach requires you to send a serial command each time you want to transmit the IR signal. With HomeVision-Pro version 3.4 (but not HomeVision), it is also possible to transmit the IR signal repeatedly. This may be useful for IR signals such as "Volume up". You can do this as follows:

1. Read all the IR signals from the ".IRL" file.

2. Load the signal into the controller's test area.

3. Write three additional bytes to the controller RAM. The values to write are unique to the signal loaded in the test area. The variables are set using a special serial command that can write to the controller's internal RAM. The command format is as follows:

   ,6AAAADD

   where: ,6 is the command to write a value to RAM
   AAAA is the RAM address to write to
   DD is the data value, in hex format

   The first byte is written to address EF55, the second to address EF56, and the third to address EF57. For example, assume the three bytes should be 20 hex (32 decimal), 37 hex (55 decimal), and 0 hex (0 decimal). The following three serial commands would be used, and show the meaning of the three data values:

,6EF5520	Signal length (number of pulses in the signal)
,6EF5637	256 - carrier base on-time
,6EF5700	256 - carrier base off-time

   The three data values are the same as shown in the TransmitSignal() function when generating the serial command string (Step 3 above). The first byte is NumberOfFullPulses, the second is (256-CarrierOnTime), and the third is (256-CarrierOffTime).

4. Start transmitting the test signal repeatedly. This is done with a new IR command over the serial interface:

   ,;000B

5. Stop the signal transmissions whenever you're ready. This is done with a new IR command over the serial interface:

,;000A

APPENDIX – LOADING IR SIGNAL DATA INTO TEST AREA

In order to load an IR signal into the controller, a program must first read the IR signal data from a file. Although it's possible to read the data out of a HomeVision schedule file, it's not recommended. The schedule file format is quite complex and subject to change in future versions. Instead, users should export the IR signals to an "IRL" file. This is done by selecting the IR signals on the IR Signal Summary Screen, then clicking the "Export" button. The IR signal data will be saved to a file with a ".IRL" extension. This file has a much simpler format than the schedule file, and will be much easier to read with a software program.

The first step in writing the software is to declare the variables that will be used. Listing 1 shows the variables used by this sample code. These are all shown as global variables for simplicity, but you may want to treat them differently.

The next step is to read in the ".IRL" file into the variables. The function ReadIRLFile() in Listing 2 shows how to do this.

You're now reading to load any of the signals into the controller's test area. The function LoadIntoTestArea() in Listing 3 shows how to do this. Finally, you can command the controller to transmit the signal. The function TransmitSignal() in Listing 4 shows this. Listing 5 contains several functions used by LoadIntoTestArea() and TransmitSignal() to format and send the necessary serial commands to the controller. You will need to modify at least one of these (the SendCommand() function) based on how your software accesses the PC serial port.

LISTING 1 - DECLARATION OF GLOBAL VARIABLES

```
'Variables that tell us about the IR signals:
Global gIRCount As Integer      'The number of signals
Type IRSignalType               'General data for each signal
    Name As String * 30
    Type As Byte                '0 = Preset, 1 = Raw Pulse, 255 = none
    byte1 As Byte               '0 to 255
    byte2 As Byte               '0 to 255
    PulseLength As Integer      '# of bytes used in gPulseArray()
    PulseStart As Long          'Starting byte in gPulseArray()
    description As String * 40
End Type
Global gIRSignals() As IRSignalType 'Array to hold general data for each signal

'Global variables used for storing IR pulse data of learned signals:
Global gPulseArray() As Byte        'Stores all pulse lengths for all signals
Global gPulseArrayLength As Long    'Total # of bytes in gPulseArray

'Array used to temporarily store IR pulse data before loading it:
Global gLoadPulseData(400) As Byte

Global gControllerType As Byte      '0=HomeVision, 1=HomeVision-Pro

'Constants:
Global Const COUNTTOTIMERATIO = 0.36169
Global Const LOWRAM = 0
Global Const HIGHRAM = 1
Global Const HV = 0
Global Const HVPRO = 1
```

LISTING 2 - ReadIRLFile()

```
Public Function ReadIRLFile(IRFileName As String) As Integer
'Opens ".IRL" file and reads it in.
Dim i As Long, temp As Byte, FileNumber As Integer

On Error GoTo ExitError
FileNumber = FreeFile
Open IRFileName For Binary As #FileNumber 'Open the file
Get #1, 1, gIRCount                       'Read in the # of signals in the file
Get #1, , gPulseArrayLength               'Read in the total # of pulses

If gIRCount > 0 Then
    ReDim gIRSignals(gIRCount - 1)        'Resize gIRSignals array
    For i = 0 To gIRCount - 1             'Read in the general data on each signal
        Get #1, , gIRSignals(i).Name
        Get #1, , gIRSignals(i).Type
        Get #1, , gIRSignals(i).byte1     '256 - carrier on-time
        Get #1, , gIRSignals(i).byte2     '256 - carrier off-time
        Get #1, , temp                    'Number of FULL pulses
        gIRSignals(i).PulseLength = temp * 4    '4 bytes per pulse
        Get #1, , gIRSignals(i).PulseStart
        Get #1, , gIRSignals(i).description
    Next i
End If

If gPulseArrayLength > 0 Then           'Now read in all the pulse data
    ReDim gPulseArray(gPulseArrayLength - 1)   'Resize array
    For i = 0 To gPulseArrayLength - 1
        Get #1, , gPulseArray(i)
    Next i
End If

ReadIRLFile = 1  'OK
GoTo exitsub

ExitError:
    MsgBox ("Error reading file")
```

```
    ReadIRLFile = -1  'error

exitsub:
    Close #FileNumber
End Function
```

LISTING 3 – LOADING SIGNAL INTO CONTROLLER TEST AREA

```
Public Function LoadIntoTestArea(SignalNum As Integer) As Integer
Dim Address As Long, j As Integer, NumberOfFullPulses As Integer
Dim CarrierPeriod As Integer, CarrierPerioduS As Single, OnTimeuS As Single
Dim BytesToSend As Integer

On Error Resume Next

BytesToSend = gIRSignals(SignalNum).PulseLength
NumberOfFullPulses = BytesToSend \ 4

'Move the pulse length data from gPulseArray() into gLoadPulseData():
For j = 0 To BytesToSend - 1
    gLoadPulseData(j) = gPulseArray(j + gIRSignals(SignalNum).PulseStart)
Next j

CarrierPeriod = (256 - gIRSignals(SignalNum).byte1) + (256 - gIRSignals(SignalNum).byte2)
If gControllerType = HVPRO Then
     'For HomeVision-Pro, signals learned with a carrier frequency must be
     'converted into the HomeVision-Pro format.
     'The pulse on-times must be converted into the number of 10.127us periods.
     'The on-time is stored as the number of cycles of the carrier, using two
     'bytes in gLoadPulseData.
     'First, calculate the carrier period used for this signal:
     CarrierPerioduS = CarrierPeriod * COUNTTOTIMERATIO
     'Next, loop thru all pulses in groups of 4 bytes and convert the
     'on-time to the HV-Pro format
     For j = 0 To NumberOfFullPulses - 1
         'First, get the on-time from 2 bytes and convert to us:
         OnTimeuS = (gLoadPulseData(j * 4) * 256 + gLoadPulseData(j * 4 + 1)) * CarrierPerioduS
         'Now calculate number of 10.127uS periods:
         OnTimeuS = OnTimeuS / 10.127
         gLoadPulseData(j * 4) = CByte(OnTimeuS \ 256)
         gLoadPulseData(j * 4 + 1) = CByte(OnTimeuS Mod 256)
     Next j
End If
```

```
'Now load the IR signal into special test area in controller's high RAM:
    Address = 9216  'Address in controller RAM to load to (2400 hex)
    LoadIntoTestArea = LoadDataToRAM(HIGHRAM, Address, gLoadPulseData, BytesToSend)
End Function
```

LISTING 4 – COMMAND CONTROLLER TO TRANSMIT SIGNAL

```
Public Function TransmitSignal(IDNum As Integer) As Integer
'Returns 1 if OK, -1 or -2 if failed
Dim Out_String As String, NumberOfFullPulses As Integer, FreqInkHz As Single
Dim CarrierOnTime As Integer, CarrierOffTime As Integer, CarrierPeriod As Integer
Dim CarrierPerioduS As Single

'Generate the serial string needed to transmit the signal:
CarrierOnTime = 256 - CInt(gIRSignals(IDNum).byte1)
CarrierOffTime = 256 - CInt(gIRSignals(IDNum).byte2)
CarrierPeriod = CarrierOnTime + CarrierOffTime
NumberOfFullPulses = gIRSignals(IDNum).PulseLength \ 4  '4 bytes used for each pulse
If gControllerType = HVPRO Then
    'For HomeVision-Pro, signals learned with a carrier frequency must be
    'converted into the HomeVision-Pro format.
    'Convert CarrierOnTime and CarrierOffTime into the carrier frequency:
    CarrierPerioduS = COUNTTOTIMERATIO * CarrierPeriod
    FreqInkHz = 1000 / CarrierPerioduS
    'Calculate timer reload value used by uC, but with MSB clear (to force 50% duty cycle).
    'The carrier on and off times will be the same (i.e., 50% duty cycle).
    'Carrier on and off times = (128 - reload value) * 0.5425uS.
    'Final equation: Reload value = 128 - int(921.66 / Freq(in kHz)).
    FreqInkHz = 128 - (921.66 / FreqInkHz)
    Out_String = TwoD_Hex(NumberOfFullPulses) & TwoD_Hex(CInt(FreqInkHz)) & "00"
Else
    Out_String = TwoD_Hex(NumberOfFullPulses) & TwoD_Hex(256-CarrierOnTime) & TwoD_Hex(256-CarrierOffTime)
End If
Out_String = ",_" & Out_String & Chr$(13)    'Put ",_" on front and CR on end

'Finally, command the signal to be transmitted:
TransmitSignal = SendCommand(Out_String, "2F")
End Function
```

LISTING 5 – FUNCTIONS TO LOAD ARRAY OF DATA INTO CONTROLLER RAM

```
Public Function LoadDataToRAM(HighOrLow As Byte, ByVal Address As Long, Array_Name() As Byte, BytesToSend
As Integer) As Integer
'HighOrLow indicates whether data should be loaded into low or high RAM (0=low, 1=high).
'Address is the address in controller RAM to start loading at.
'Array_Name() is the array containing the data to load.
'BytesToSend is the number of bytes to load.

Dim i As Integer, status As Integer, Addr_LSB As Long, Addr_MSB As Long
Dim Count As Long, Blocks As Integer, Extra As Integer, Addr As String

Blocks = BytesToSend \ 13   'Determine # of 13-byte blocks we can send
Extra = BytesToSend Mod 13  'Determine # of left-over bytes to send
Count = 0                   'Indicates how many bytes have already been sent
For i = 0 To Blocks - 1     'Send 13-byte blocks of Data
    Addr_LSB = Address And &HFF
    Addr_MSB = Address And &HFF00
    Addr_MSB = Addr_MSB \ 256
    If HighOrLow = LOWRAM Then      'send to low RAM
        status = SendBlock(Count, Addr_LSB, Addr_MSB, Array_Name)
    Else                            'send to high RAM
        status = SendBlockHigh(Count, Addr_LSB, Addr_MSB, Array_Name)
    End If
    If status <> 1 Then GoTo ExitError
    Address = Address + 13
    Count = Count + 13
Next i

For i = 0 To Extra - 1  'Send any remaining data that didn't fit into 13-byte blocks
    If Address <= &HF Then
        Addr = "000" + Hex$(Address)
    ElseIf Address <= &HFF Then
        Addr = "00" + Hex$(Address)
    ElseIf Address <= &HFFF Then
        Addr = "0" + Hex$(Address)
    Else
        Addr = Hex$(Address)
```

```
    End If
    If HighOrLow = LOWRAM Then      'send to low RAM
        status = SendByte(TwoD_Hex(CInt(Array_Name(Count))), Addr)
    Else                            'send to high RAM
        status = SendByteHigh(TwoD_Hex(CInt(Array_Name(Count))), Addr)
    End If
    If status <> 1 Then GoTo ExitError
    Address = Address + 1
    Count = Count + 1
Next i

LoadDataToRAM = 1       'OK
Exit Function

ExitError:
    LoadDataToRAM = -1  'error
End Function

'-------------------------------------------------
Public Function SendBlock(Count As Long, Addr_LSB As Long, Addr_MSB As Long, Array_Name() As Byte) As
Integer
'Loads 13 bytes from the array Array_Name(), starting at location Count, into the
'controller's low RAM starting at the address defined by Addr_LSB and Addr_MSB.
'Returns 1 if OK, -1 if error
Dim Out_String As String, i As Long
Out_String = ",7" + Chr$(Addr_MSB) + Chr$(Addr_LSB) 'Create start of string
For i = Count To Count + 12
    Out_String = Out_String + Chr$(Array_Name(i))   'Add chars to string
Next i
Out_String = Out_String + Chr$(13)                  'Put CR on end
SendBlock = SendCommand(Out_String, "07")           'Now transmit the string
End Function

'-------------------------------------------------
Public Function SendBlockHigh(Count As Long, Addr_LSB As Long, Addr_MSB As Long, Array_Name() As Byte) As
Integer
'Loads 13 bytes from the array Array_Name(), starting at location Count, into the
'controller's high RAM starting at the address defined by Addr_LSB and Addr_MSB.
```

```
'Returns 1 if OK, -1 if error
Dim Out_String As String, i As Long
Out_String = ",b" + Chr$(Addr_MSB) + Chr$(Addr_LSB) 'Create start of string
For i = Count To Count + 12
    Out_String = Out_String + Chr$(Array_Name(i))   'Add chars to string
Next i
Out_String = Out_String + Chr$(13)                  'Put CR on end
SendBlockHigh = SendCommand(Out_String, "32")       'Now transmit the string
End Function

'-----------------------------------------------------
Public Function SendByte(Dat As String, Address As String) As Integer
'Returns 1 if OK, -1 if error
SendByte = SendCommand(",6" & Address & Dat & Chr$(13), "06")
End Function

'-----------------------------------------------------
Public Function SendByteHigh(Dat As String, Address As String) As Integer
'Returns 1 if OK, -1 if error
SendByteHigh = SendCommand(",]" & Address & Dat & Chr$(13), "2D")
End Function

'-----------------------------------------------------
Public Function SendCommand(Out_String As String, CCode As String) As Integer
'Sends and verifies it was done.  Doesn't return the response itself.
'Returns 1 if OK, -1 if error

'WE DON'T SHOW ANY CODE HERE BECAUSE IT WILL BE UNIQUE TO YOUR APPLICATION.
'INSERT YOUR CODE TO TRANSMIT THE SERIAL STRING "Out_String".
'YOUR CODE SHOULD ALSO WAIT FOR THE CONTROLLER RESPONSE TO ENSURE THE COMMAND WORKED.
'"CCode" CONTAINS THE FIRST TWO CHARACTERS OF THE EXPECTED RESPONSE.
'THE RESPONSE SHOULD LOOK LIKE THIS:
'      XX Cmd: Done
'Where XX would be replaced by the two characters of "CCode".
End Function

'-----------------------------------------------------
Public Function TwoD_Hex(Dat As Integer) As String
```

```
'Convert a number between 00 adn 255 into 2 hex characters
    If Dat <= &HF Then
        TwoD_Hex = "0" + Hex$(Dat)
    ElseIf Dat >= 256 Then
        TwoD_Hex = "00"
    Else
        TwoD_Hex = Hex$(Dat)
    End If
End Function
```