Using the ADI Ocelot CMAX programming capabilities in HAL 2000 Home Automation Applications

Abstract: This document discusses the ADI Ocelot capabilities and how they might augment the HAL Home Automation Application It uses real world examples to illustrate the power and process of using this powerful adjunct to HAL.

[Draft Version]

The Goal

Many HAL installations use the ADI Ocelot as an interface to the external world, via X10, IR and add-on interfaces. Ocelot can be much more than an I/O interface. In this document we illustrate how to employ the powers of the Ocelot far beyond the mundane I/O interface .

Overview

Ocelot is a small but powerful computer in it's own right. It is a class of computer known as an embedded system. The basic Ocelot is an IR interface for HAL.

With the addition of a TW523 X10 transceiver, Ocelot also becomes the X10 interface for HAL. Other modules can be connected to Ocelot to support relay closures and contact closures, measure temperature and other tasks.

Normally, HAL only knows how to use the various interfaces on the Ocelot, but does not support any programming in the Ocelot itself.

However, it is quite practical to use the Ocelot to offload many tasks from HAL. I like to think of this model as loosely comparable to the left-brain, right-brain physiological model of human intellect.

Benefits

There are two significant benefits to incorporating Ocelot programming in your Home Automation Project. They are **speed** and **reliability.**

Logic executes on the Ocelot much faster than on HAL. While it is possible to put HAL on ever faster computers, nonetheless complex rules and macros on HAL can slow down the system, and more importantly, cause HAL's speech recognition to be slow or even non-responsive. Offloading tasks to Ocelot allows HAL to concentrate on speech recognition and high level decision making processes.

HAL runs on a full function, general purpose computer, which must be on at all times. Power failures, and plain old computer crashes will incapacitate the system. Ocelot is an embedded system that can easily run on any 12V power source. Crashes are far less common than Windows crashes, and a simple battery system can weather power outages. Even without battery backup, Ocelot will recover from a power failure and resume it's activity reliably. Configuring HAL to do so reliably is difficult, especially with an ATX style PC.

Drawbacks

There are some difficulties with this however. First, there is no mechanism within HAL to define logic to be incorporated into the Ocelot. This forces the programmer to work in the ADI CMAX programming environment. Second, there is no mechanism for HAL and Ocelot to communicate, so we must use creative means to allow each processor to be aware of what the other is doing.

Task Types

There are four categories of tasks that can be incorporated into the Ocelot.

Ocelot ! HAL – Ocelot programs that have no interaction whatever in HAL

HAL = Ocelot – HAL and Ocelot independently execute similar logic without any dependencies between them.

HAL > Ocelot – HAL makes a decision that triggers action by the Ocelot.

Ocelot > HAL – Ocelot makes a decision that triggers action by HAL

H>O tasks require a mechanism for passing a trigger, or flag, to the Ocelot from HAL. Unfortunately, there is no convenient mechanism to pass information to Ocelot from HAL.

However, Ocelot monitors X10 device status and can react to an X10 device or message. This can be used to trigger events and states.

H=O tasks execute in parallel and unaware of each other. However the logic of the two tasks need not be identical, and HAL may leave actual I/O operations to Ocelot.

For example, pressing a button on a bedside console might generate a command to turn on a light. Ocelot might send the actual X10 command to turn on the light, while HAL would react to the same event by setting a flag to note that the light is on and starting a timer to turn the light off at some future time. HAL need not send a redundant X10 command to turn on the light.

O>H tasks occur when the Ocelot makes a decision and needs to trigger an action on the part of HAL. Like the H>O task, there are difficulties with communications, but here there is a convenient mechanism that can be employed to send messages without resorting to using X10 devices.

O ! H Tasks are Ocelot programs that run only in the Ocelot, and which have no parts whatever in HAL. Of course there are HAL rules and macros

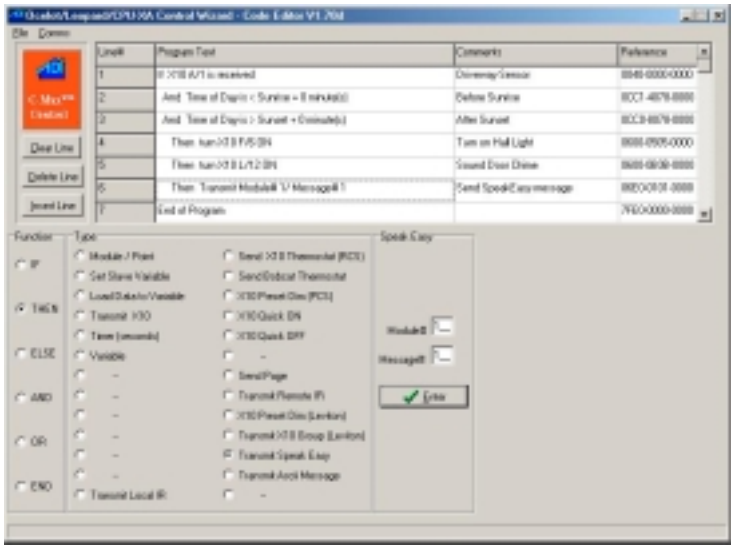
that have no involvement with CMAX code, but that is the norm and is unremarkable.

Getting Started with CMAX

The programming environment for the Ocelot is called CMAX. ADI ships CMAX free on CD with the purchase of the Ocelot, and it's available for download on their web site, as http://www.appdig.com/cmax.html. The current version is 2.00d, but V 1.70j. is probably more common at this time. The information in this paper is generic and applies to all versions.

The opening screen of CMAX is called the control wizard, a point-and-click line editor that allows you to create programs purely with the mouse.

Once the program is created, you then select the COMMS menu and Attach to the CPUXA (Historical note: CPUXA is an early product name for the Ocelot). This transforms the programming environment to the CPUXA Access environment where from the Program File menu you download your program to the CPUXA.



You will notice that the CPUXA Access environment shows blinking TX/RX lights. This is an indication of working communications between CMAX and the Ocelot. This communications is a continuous process.

When HAL is running, HAL maintains a similar communications activity with Ocelot, communicating X10 and other messages both ways. This mechanism allows HAL to treat interfaces on the Ocelot as if they were attached directly to HAL, but does not interfere with Ocelot using the interfaces itself.

CMAX programs run independently of this communications process, allowing CMAX programs to run without interfering with the I/O process. Thus CMAX programs run on Ocelot without impacting HAL.

Programming in CMAX is not greatly unlike programming in HAL. Similar data structures, timers, flags (variables) and IF/THEN statements exist in both environments. However there are differences. The variables are not simple Boolean flags like in HAL, but are integers. This does not inhibit setting them to 1 and 0 to use as flags.

Perhaps the biggest differences are in the timers, which count up instead of down, and may be read and trigger events based on their value.

Similar programs can be easily created for both environments, as we will see in the following examples.

The Bedside Button, an H=O example

Let's consider an example of automating bedside lights using a single X10 button conveniently located next to the bed.

The desired action is as follows. Press **on** once, the right side light turns on. Press **on** again, and the left side light turns on. Press **Off** and the right light turns off, press **off** again and the left light turns off.

The example shows the logic for the on process. The off process is similar, a mirror image of the on process.

The timer started for each light also has logic to turn off the lights when it expires, in case the user falls asleep and forgets the lights.

RULE Master Bedroom Right Light On
IF:
Bedside Console 4 Sensor ON
AND Master Bedroom Right Light Flag is FALSE
AND Master Bedroom Left Light Flag is FALSE
THEN:
Turn on Master Bed Room Right Light
Set Timer Master Bedroom Right Light to 3600 seconds
Set Flag Master Bedroom Right Light to TRUE

RULE Master Bedroom Left Light On
IF:
Bedside Console 4 Sensor ON
AND Master Bedroom Right Light Flag is TRUE
AND Master Bedroom Left Light Flag is FALSE
THEN:
Turn on Master Bedroom Left Light
Set Timer Master Bed Room Left Light to 3600 seconds
Set Flag Master Bedroom Left Light to TRUE

The HAL only version works quite well. However, there is room to optimize this process by offloading the actual on and off steps (flagged in red) to Ocelot. This increases reliability because the on and off steps still work even if HAL crashes, letting HAL skip the laborious send X10 command.

CMAX Version

IF X10 J/4 On Command	Bedside Console #4
AND X10 B/9 is Off	BR Lite #1 is off
AND X10 B/16 is Off	BR Lite #2 is off
THEN Turn X10 B/9 ON	Turn on BR Right
IF X10 J/4 On Command	Bedside Console #4
AND X10 B/9 is On	BR Lite #1 is ON
AND X10 B/16 is Off	BR Lite #2 is OFF
THEN Turn X10 B/16 ON	Turn on BR Left

First step is to create the CMAX program as shown and download it to the Ocelot. Once it is running in the Ocelot, you then restart HAL and remove the lines in red, and you will see that the function still works correctly.

There are a few points of difference between the HAL versions and the CMAX version. First, we do not need to test and set flags to remember the status of the light. CMAX certainly has variables that can be used for this purpose, but it is not necessary because Ocelot directly tracks the status of the light because it knows which X10 commands have been sent.

This feature is very important in the H>O type of program because this is the simplest way, indeed possibly the only way to pass status and commands from HAL to the Ocelot.

The second major difference between the two versions of the program is that in the Ocelot we do not use a timer to turn the lights off after a period of time.

We omit this because we want HAL to be able to reset the timer based on motion detectors, or other signs of activity, keeping the lights on as long as the occupant is active. Since there is no practical way to tell Ocelot to reset the timer, we don't have the Ocelot perform this part of the program.

This example is an H=O program, because both programs operate completely autonomously, and do not actually communicate a status in either direction. One event, the X10 Console ON command, triggers two programs on two different computers without any co-ordination between the two.

Simply removing a few X10 commands from HAL and moving them to the Ocelot may not seem like much. But, there are some key points that I'd like to stress.

First, sending X10 is a very laborious process for HAL, and the speech recognition process is often not available while it is doing so. At about a half second per event, this is worth removing. Offloading X10 transmit events to the Ocelot can have a profound impact on HAL's overall responsiveness.

Secondly, and more importantly, I do not like the prospect that a crash of the HAL computer during the night might leave me unable to turn on the lights by my bed. I want that button to WORK.

By implementing it this way, I pretty much guarantee that if power is on, the button will work. Perhaps not 100%, as Ocelot presumably could still crash (although I have never seen this happen), but reliability is tremendously improved over the original version.

ALL OFF, an H>O Example

Here is an example of a function that causes HAL to send a lot of X10 signals. As a result,

HAL Only Version
Macro ALL OFF {Turn Everything Off}
Send X10 Command All Units OFF to House Code A
+
+
Send X10 Command All Units OFF to House Code P

HAL is "Busy" for several seconds, and unresponsive to other functions.

Although the simple example shown merely sends X10 commands to each house code, in real life it is more complex than that. A number of house codes contain devices that we don't want to turn off even though the command is named ALL OFF. The macro actually contains a large number of discrete X10 Off Commands as well.

Since discrete commands are actually two commands (address command pair) these take twice as long to send. Then end result is a simple macro that ties up HAL for several seconds to execute.

As mentioned earlier, there is no mechanism for HAL to send a message to the Ocelot. However, the Ocelot does monitor the X10 traffic. We can create a bogus device in HAL and have HAL send commands to that device. Using the Ocelot IF/THEN logic, we can initiate action based on receipt of commands to that bogus device.

We first create in HAL a device named ALL OFF Light, at any convenient address, M16 in this example. Now our macro of many X10 commands is reduced to a single command, turn off the All Off Light.

In CMAX we then create an IF/THEN statement which looks for a M/16 OFF Command pair, and transmits the long list of X10 commands upon

H>O Version, HAL portion
Macro ALL OFF {Turn Everything Off}
Turn off All Off Light

seeing this command.

This is simple, clean, easy to implement, and completely frees the HAL environment from the laborious X10 transmission, except for that first, bogus, X10 command.

A couple of points about this logic. First, it burns an X10 address for no improvement in either functionality or reliability, albeit it does improve HAL's performance.

CMAX Code Portion

IF X10 M/16 Off Command	Bogus All Off Device
THEN Transmit X10 A/17	Send X10 Command
+	
+	
THEN Transmit X10 P/17	Send X10 Command

Second, other devices can invoke the process by simply sending the same M/16 OFF command pair. This exposes opportunities for convenience as well as a security risk.

For convenience, you can place an X10 transmitter conveniently by a door, etc. to easily turn everything off on leaving.

A Word about Security

The security risk with X10 is simply that anyone could walk up to the house, plug an X10 transmitter into an outside outlet (or even perhaps a neighbors) and execute the command.

The more power given to a single command, the greater the risk.

While risks are minor in the example given, it pays to keep these risks in mind as you develop your automation macros. Attention to a few rules can reduce risks.

(1) Never use a house code for which there is a wireless receiver active, as this exposes the macro to wireless activation from someone with a wireless key-fob. Also, since wireless devices are susceptible to RF collisions and resulting bogus commands, they should be avoided.

(2) Make sure you have a good blocking coupler in your breaker panel to prevent potential triggering from neighbors devices.

(3) Be careful what functions you expose in this manner. For example, it might not be a good idea to allow HAL to raise the garage door from such a macro.

IR Control, O>H Examples

Fellow Home Automation enthusiast and HAL user James Lipsit has written an excellent article on his web site (http://www.james.lipsit.com/ocelot_to_hal.htm) describing how to implement a communications mechanism for Ocelot to pass a message to HAL. I won't duplicate his article here, but I will use his information and describe an example of how I am using it.

The mechanism involves creating a fake sensor for the Ocelot, and then having this nonexistent sensor send a status message to HAL. This can ONLY be done if you have at least one set of real sensors, otherwise HAL won't recognize the existence of your bogus sensor. Jim gives the gory details in his article.

IR vs. Voice Control

Ocelot provides an IR interface for HAL. In a home theater installation HAL can control the system using Voice Recognition. However, VR is not practical for all functions.

Generally, once the sound cranks up, HAL must turn off the attention word to prevent false activation from the movie dialog. Once the movie is underway, the IR remote becomes the tool of choice for controlling the system.

It is highly desirable that HAL should recognize the IR commands issued from the remote, so as to be able to track the status of the system. Unfortunately, HAL has no ability to receive and respond to IR commands. HAL treats IR as output only, except in the limited case of training new commands.

The Ocelot however, is fully capable of recognizing and responding to IR commands. By using the mechanism described by Jim, it is a simple matter to program the Ocelot to recognize that a given IR command has been received, and having Ocelot then send a sensor message to HAL reflecting that message. HAL can then act upon the IR message, whether it is something that requires a specific response, or whether it merely requires tracking the state.

On beginning an evening's entertainment, I tell HAL, "Let's watch the Satellite System" and HAL will then turn on the TV, set the AV Receiver to the satellite system, and turn on the Satellite Receiver.

When I wish to change to a DVD or watch something on my Tivo, I can tell HAL to make the change, but that requires muting the audio, waking up HAL, getting his attention and then giving the appropriate command. Or I can hit the buttons on the IR remote.

The control example I am currently implementing involves having HAL recognize which entertainment source is in use, following any IR commands used to change the functional state. At the end of the evening, a single voice command can shut down the entire system, and HAL will know exactly which IR commands to send to effect the proper shutdown.

I do not have complete example code to display at this writing. However, by looking at Jim's article, the process of actual implementation should be self-evident. Hopefully at some future point I will be able to revise this article to include my functional code.