Description of RTOS Interface for Nohau Emulator Software

11/19/2002

For many years, Nohau has had real-time operating support. Different flavors have been done. The implementations based on dll's have been mostly successful but suffered from maintenance difficulties. Now, a new implementation method is available. Microsoft has defined the "com" interfaces to enable different applications talking to each other. A real-time operating system (RTOS) vendor can now write an ActiveX / com object that can present all the information on the screen for a particular RTOS. This ActiveX component will integrate easily with Nohau's Seehau user interface as long as it follows the Nohau RTOS interface specification below.

Nohau is here making available the interface specification to RTOS vendors that would like to implement an "RTOS Window" in Seehau. Several development platforms can be used by the RTOS vendor to take their current presentation screen and wrap it in an ActiveX control wrapper. For instance, with Microsoft, an MFC application can be used as the foundation for a presentation screen, and then turned it into an ActiveX. The ActiveX control usually carries the extension ".ocx".

The interface to the "OCX" is specified by methods and events. Methods are functions that are called by the emulator's user interface (Seehau, in Nohau's case). Events are functions that are called by the RTOS module. Nohau will respond to those events and, for instance, let the RTOS module perform a read of memory that resides in the emulator.

This document has the intention of describing the methods that Seehau will expect to find in the RTOS module. In addition, the section below also describes the event functions that the RTOS module can count on being available in Seehau.

Registry Information:

Nohau will read the registry key under **HKEY_CLASSES_ROOT\CLSID*** looking for the keys that have a subkey "**Embedded RTOS debug module**" these entries are for the **RTOS** application servers. Your install program will have to put the subkey in the registry for your activeX module.

Methods:

void ConnectEvents(INertosEvents * ine);
The ConnectEvents method will be the first method that is called to the RTOS module after its creation. This method provides a Event Interface pointer to the RTOS module. The RTOS module will use the interface pointer when firing event:
Example: FNertosEvents = ine;
Example: FNertosEvents.ReadMemory(0, AppAddr, SizeOf(App), int(&App));

void Initialize(BSTR * initstring);
The parameter carries the name of the emulator. It should be followed by the emulator family (EMUL51, EMUL68, EMUL16, EMUL300, EMUL196, EMUL296, EMUL251, EMUL51XA, EMUL166, EMUL-ST10, EMUL-M16C, EMUL12). The next parameter is optional, and will denote if the emulator is in demo mode (DEMO). The name is emulator manufacturer specific ("Nohau" in our case). The intention with this method call is to let the RTOS module perform any necessary initialization and notify it which emulator it is interfacing to.
Example: void initialize(Nohau EMUL196)
Example: void initialize(Nohau EMUL51 DEMO)

void Endian(short LEndian);
This method lets Seehau inform the RTOS module if the byte order that will be used should follow the Intel or Motorola standard. If a "1" is specified, the byte order used will follow Intel's standard, and the

---

Click Here to download Delphi example.
Click Here to download C++ example.

---

least significant byte in a word would reside in the least significant address. If the Endian parameter is specified as a "0", the byte order used will follow Motorola's standard, and the least significant byte will reside on the highest address of the information package transferred. Seehau will call this method based on which emulator family that is used. Normally, all bytes transferred between the RTOS and Seehau will follow Intel's standard. This method will be called after the initialization.

void ProgramLoaded(void);
The ProgramLoad edmethod is called by Seehau to inform the RTOS module that the emulator has completed a load of the customer's application. This method call to the RTOS module should be used to fire ReadMemory events to pick up information regarding the loaded application.
void ProgramReset(void);
The ProgramReset method is called when the emulator is reset from within Seehau. It is possible, though, that the emulator is reset by the target system under test and it cannot be guaranteed that the RTOS module would be informed about this. The RTOS module can use this method call to understand that the user application is likely to go through its initialization sequence.

void ReadMemoryTest(short memtype, long address, long size, long dest);
The ReadMemoryTest method call is used to allow easy testing of the RTOS module. It is intended that Seehau can call this method to request the RTOS module to fire a ReadMemory event. The parameters used in the read memory event should be the same as specified in this method call. For explanation of the parameters, please see under the events below.

void Started(void);
The Started method call is performed by Seehau after the emulator has started code execution of the user's application. This method call is done to let the RTOS module know that the emulator is busy executing user code. The emulator will therefore not be able to respond to ReadMemory or WriteMemory events, except for a ReadMemory event directed to ShadowRAM.

void Stopped(long pc);
The Stopped method call is performed by Seehau after the emulator has stopped code execution of the user's application. This method call is done to let the RTOS module know that the emulator is available. The emulator will therefore be able to respond to ReadMemory or WriteMemory events. The parameter that is fed in carries the program counter's address. This address can be used by the RTOS module to determine where code execution stopped. Code execution can have stopped due to a breakpoint, the user pushing the 'stop' button, a problem in the user application, or the trace stopping the user application execution.

void TraceStarted(void);
The TraceStarted method call is performed by Seehau after the trace has started recording. This method call is done to let the RTOS module know that the trace is busy recording. The trace will therefore not be able to respond to the TraceTrigger event.

void TraceStopped(void);
The TraceStopped method call is performed by Seehau after the trace has stopped recording. This method call is done to let the RTOS module know that the trace stopped because it triggered or emulation was stopped (see Stopped method). The emulator will therefore be able to respond to the TraceTrigger event.

void AboutBox(void);

Click Here to download Delphi example.
Click Here to download C++ example.

Description of RTOS Interface for Nohau Emulator Software

11/19/2002

This method call is performed in response to the user clicking on the AboutBox menu item in the RTOS window. The intention is to let the RTOS vendor present an "About" box containing version number and company information.

void Terminate(void);

The Terminate method call is the last method call to the RTOS module before it is expected to terminate. The intention is to allow the RTOS module to free allocated memory and unload possible dll's.

Events

The READ and WRITE memory events will provide bytes to the RTOS module. If the Endian method call described above was done specifying Motorola byte order, then it will be the RTOS module's responsibility to swap the bytes in the byte buffer as needed.

void Start(void);

This event can be fired by the RTOS module to start the emulator. The Start event will, of course, result in an immediate call to the Started method.

void Stop(void);

The Stop event lets the RTOS module stop the emulator. The Stop event will, of course, result in an immediate call to the Stopped method.

void ReadMemory(short memtype, long address, long size, long dest);

The ReadMemory event call lets the RTOS module read memories from the emulator / user target system. For instance, after the Stopped method call described above, the RTOS module is likely to want to read memory from the emulator. The first parameter specified in this function call is the memory type, which could be CODE (=0), DATA (=1) and SHADOW (=2). The second parameter specifies the address from which the read should be performed. The third parameter specifies the length in bytes of the object to be read. The last parameter specifies a buffer destination that should be allocated in memory by the RTOS module. It is the RTOS's responsibility to make sure that the allocated area is larger or equal to the amount of bytes requested.

void WriteMemory(short memtype, long address, long size, long source);

This event can be fired by the RTOS module to write to the emulator memory. The parameters are identical to the ReadMemory event as described above with the exception of the destination pointer that now is the source pointer that points to the memory unit containing the bytes that should be written to the emulator. The RTOS module can use this event call to change information in the RTOS's structures that reside in the emulator's memory.

void TraceStart(void);

This event can be fired by the RTOS module to start the trace. The trace is an option available for the emulator that allows recording of execution flow. The RTOS module can set trigger points in the trace (see TraceTrigger below). The trigger point can result in the TraceStopped method call that would let the RTOS module know that a certain address got executed in the user's application without intruding on the real-time execution. The TraceStart event will, of course, result in an immediate call to the TraceStarted method.

void TraceStop(void);

The TraceStop event lets the RTOS module stop the trace so it can be reconfigured. The TraceStop event will, of course, result in an immediate call to the TraceStopped method.

Click Here to download Delphi example.
Click Here to download C++ example.

Description of RTOS Interface for Nohau Emulator Software

11/19/2002

void Breakpoint(long address, BSTR * state);

This event allows the RTOS module to set a breakpoint in the user's application. The breakpoint will, if hit, stop execution of the user's application and therefore result in the Stopped method call. The breakpoint event should only be fired from the RTOS module if the emulator is not running. The first parameter specifies the address where the breakpoint should be inserted. The second parameter is a string that carries the value "On" or "Off" to allow the RTOS module to turn on versus turn off the breakpoint.

void TraceTrigger(short cycletype, long address, BSTR * state);

This event can be fired by the RTOS module to set a trigger point in the trace. A trigger point, when hit, will not stop user code execution unless this is specifically set up in the trace configuration screen. The RTOS module will get notified by the TraceStopped method call when a trigger point is hit. The intention with this event is to allow the RTOS module to set trigger points at addresses so it can detect execution flow. The first parameter specifies the cycle type that should be used for the trigger. It can be DATA READ, DATA WRITE, or CODE. The second parameter specifies the address for the trigger point. The last parameter specifies the state of the trigger point so the RTOS module can turn the trigger point "On" and "Off".

BSTR ExecCmd(BSTR str)

The RTOS can fire this event to execute a command in the emulator. Using this event is optional, but offers a powerful means to access the emulators macro capabilities (see Commands under Help in Seehau).

Click Here to download Delphi example.
Click Here to download C++ example.
