FSMEmulator Architecture Overview

by Calin Culianu January 2009

Overview

The FSM emulator (herafter referred to as “FSMEmulator”, “the FSMEmulator” or “the emulator”) is a set of userspace programs that work together to emulate the real RTFSM system. Where possible, the emulator uses the existing non-emulated RTFSM codebase to achieve as much overlap in logic and features as possible. Because of this, a useful companion guide to this overview is the document: “RTFSM Architectural Overview”.

Other Resources

Along with this guide, you should probably be aware of the following documentation:

“RTFSM Architectural Overview” - a document packaged with this document that describes the non-emulated real RTFSM system

rtfsm/emulator/doc/html – generated sourcecode documentation. Useful for perusing the sources which have, in places, even some level of function-level and class-level documentation.

Differences Between the FSMEmulator and the RTFSM System

Since the FSMEmulator has an analogous architecture to the real system, it is useful to summarize the differences between the two systems:

- The real FSM runs in kernel-space on the RTAI real-time platform, whereas the FSMEmulator runs as a userspace process on either regular Linux, Windows, or Mac OSX 10.4 or above.
- The real FSM is responsible for implementing **only** the logic of the FSM, whereas the FSMEmulator also implements (or rather, emulates) hardware and some portion of the COMEDI drivers. COMEDI calls to write to AI, AO, and DIO inputs and outputs are emulated and implemented using GUI controls (ex: a GUI button can trigger an analog input event, or a GUI LED can signify a DO line going high, etc).
- In the real FSM system, sound triggering is an add-on implemented as a set of separate realtime programs (LynxTrig-RT.ko and LynxTWO-RT.ko), whereas the FSMEmulator is intimately aware of sound triggering. The FSMEmulator process implements the entire sound triggering mechanism in the same process; sound is played using the operating system's sound services in a separate thread, and statistics on sound playing are updated in real-time on the screen.
- The real FSM's sound subsystem keeps all sound data files in memory to minimize latency and jitter. In the FSMEmulator, no such criterion exists, so sound files are stored on disk in an OS-specific temporary directory. This difference is important because the emulated FSM may have a higher capacity to store large amonts of sound data than the real FSM which is prone to run out of memory in situations with huge amounts of sound data.
- The real FSM is loaded separately into the kernel (then the FSMServer and SoundServer programs are typically started to listen for commands from Matlab). This is not the case for the the FSMEmulator: it is a single-click-to-launch type of affair -- it starts the SoundServer and FSMServer for you (they are still separate programs in both systems) and manages their life cycle.

- The real FSM implements “embedded-C” by having user-specified embedded-C-FSMs compiled as kernel modules (compilation is done by the FSMServer userspace program). When the modules get loaded, they get hooked into the existing RealtimeFSM.ko code. In the FSMEmulator, an analagous mechanism exists – that is, the FSMServer compiles code for the FSMEmulator process into a *dynamic library* (a .dll on Windows, .dylib on Mac, or .so on Linux), which is then loaded into the FSMEmulator process address-space at runtime.
- The real FSM uses GCC to compile its “embedded-C” fsm programs. The FSMEmulator uses GCC as well on both Linux and Mac OSX, but on Windows it uses the 'tcc' compiler, which is packaged with the system (see the ***tcc/*** subdirectory in the sources or in the installed package).
- The real FSM is obviously a hard real-time system with strict timing deadlines. Time in the FSMEmulator is 'emulated' abstractly and has little relation to real-world time.

FSMEmulator Executable

The main executable for the emulator is the FSMEmulator program (or FSMEmulator.exe on Windows), which is a Qt-based GUI application. Its responsibilities include:

- Implementing the GUI.
- Starting the FSMServer and SoundServer helper programs. (See “RTFSM Architectural Overview” for more detail on these programs).
- Implementing the FSM in a thread. This is analogous to the RealtimeFSM.ko kernel module in the real FSM. Code from fsm.c and its helper files is compiled into the FSMEmulator and executes in a separate FSM thread.
- Emulating the hardware layer of the RTFSM system (COMEDI-based DIO, AO, AI) using the GUI controls, so that a user can affect the inputs to the
- Emulating the sound subsystem of the FSM system.

FSMEmulator Sourcecode Breakdown

The sourcecode is almost entirely written in C++. Roughly speaking, the sourcecode can be broken down into GUI-related code and classes, and non-GUI code (which ends up being called by the GUI code, but implements nitty-gritty details of the system).

FSMEmulator GUI-related Files

<u>main.cpp</u>

A tiny file that starts the Qt application by creating an EmulApp object and executing it. See EmulApp below.

<u>EmulApp.cpp/.h</u>

The central class of the FSMEmulator program. Start here! It manages the lifecycle of the helper programs, starts the GUI windows, starts the FSM and sound threads, and in general is the mastermind of the whole system. Seriously – start here!

MainWindow.cpp/.h

A Qt widget that implements the application's main window. The main window of the application is actually just the little log window with its associated menu.

ControlWin.cpp/.h

A Qt widget that implements the application's control window. The control window is the second window the application creates on launch, and it contains the buttons and controls for interacting with the emulated hardware (AI, AO, DIO, etc) as well as real-time status displays for the FSM and sound player subsystems.

ComediView.cpp/.h

A set of Qt widgets for implementing the emulated COMEDI hardware controls, which live inside the Conrol Window (see above). Specific classes such as ComediDIOView and ComediAIView are implemented in this file. (See also ComediEmul.h/.cpp).

ModuleParmView.cpp/.h

A Qt window that implements editing and viewing of FSM module parameters. Background: In the real FSM, behavior of the RealtimeFSM.ko kernel module can be affected by modifying its module parameters at load-time. In the emulated FSM, these parameters still exist, and so, this editor allows you to modify the module parameters.

ProcFileViewer.cpp/.h

A Qt window that displays the 'proc file' of the FSM. Background: In the real RealtimeFSM.ko kernel process, a user can monitor the runtime statistics of the FSM module by reading a special (human-readable) text file in the Linux filesystem called /proc/RealtimeFSM. This file still exists in the emulator, but is now read by opening this window and refreshing its contents.

FSMEmulator non-GUI Files

ComediEmul.cpp/.h

A C++ class that emulates COMEDI hardware. While this class itself is unaware of the GUI (it is simply a shared data sink/source that emulates hardware), it is important to note that data inputs to this class come from the GUI and that the data outputs of this class eventually do end up being displayed in the GUI (see the ComediView class). In addition, the ***fsm.c*** file also can see the inputs and outputs of this shared resource – the inputs are inputs to the FSM and the outputs may be written to by the FSM. Several hardware profiles are implemented: NI-6025E, NI-6071E, NI-6229M. However, switching hardware profiles is unimplemented in the GUI, so only the default NI-6025E profile ends up being used in practice.

kernel_emul.c/.h

C functions that emulate the entire RTAI kernelspace API, along with the COMEDI kernelspace API. Parts of the Linux kernel API are also implemented. This is essentially 'glue' code that was needed to adapt the ***fsm.c*** file to run inside the FSMEmulator userspace process.

<u>Log.cpp/.h</u>

A generic stream-oriented logger. Parts of the FSMEmulator application log to this class, and logged data either gets drawn in the GUI's MainWindow (see ***MainWindow.cpp/.h*** above) or get written to the console if no GUI exists.

<u>SoundPlayer.cpp/.h</u>

Helper class that knows how to play audio from a disk file. Platform-specific sound-playing is either done by the ***OSXFilePlayer/**** classes for Max OSX, or using the Qt 'QSound' mechanism on Windows and Linux.

<u>windows_dlemul.cpp/.h</u>

A windows-only set of functions to implement the unix-like dlopen()/dlsym() API calls for working with dynamic libraries.

FSMEmulator Files that are shared with the real RTFSM

The following files that are part of the emulator package are also part of the real RTFSM system (meaning, their code is compiled into one of the FSMEmulator program, FSMServer program, or SoundServer proram):

- Code shared between FSMEmulator and RealtimeFSM.ko:

fsm.c	implements FSM logic
scanproc.c	Linux-specific functions for working with proc
rtos_utility.cpp	generic functions for using shared memory, fifos, etc
extra_mathfuncs.c	some math functions used by embedded-C
deflate_helper.c	functions for compressing/uncompressing data used by embedded-C mechanism

- Programs shared with the real system:

  FSMServer
  SoundServer

  Both of the above programs are compiled with different options set for the emulator (thus they only work with the emulator), but they share the exact same codebase with the real system.