RTFSM Architectural Overview

by Calin Culianu April 2008

Overview

The RTFSM core system consists of a realtime kernel module "RealtimeFSM.ko" and a userspace server program "FSMServer". The kernel module runs in kernelspace and communicates via shared memory (rt-shm) and FIFOS (RT-FIFOs) with the userspace program. The userspace program listens for TCP connections on port 3333 and accept protocol commands from a matlab client.

RealtimeFSM.ko Rough Breakdown

fsm.c

Main kernel module code. When module is loaded function `'int init(void) '` is called by the kernel in *nonrealtime* context. This function calls other functions that detect COMEDI hardware, set up shared memory (rt-shm) and RT-FIFOs, and start the realtime task thread, which is the function `'void *doFSM(void *) '`.

`doFSM()` runs at the task rate (usually 6kHz) in *realtime context* and does roughly the following each task cycle (default every 166 microseconds):

- wakes up
- grabs input channels from COMEDI (either DI or AI depending on configuration) (see `grabAI()` and `grabAllDIO()`)
- does threshold detection (optionally calling Embedded-C threshold detection functions)
- checks RT-FIFOs `(handleFifos())` to see if new commands came from userspace 'FSMServer' program
  - If new commands came, executes commands. Some commands that take a long time might be pended to nonrealtime via the 'softtask' mechanism (see **SoftTask.c** and **SoftTask.h**), or they might happen right away if they are simple commands.
- Loops through all the active and valid FSMs and for each valid FSM does
  - input detection `(detectInputEvents())`, scheduled wave processing `(processSchedWaves())`, state transitions `(gotoState())`, pends DOUT output (`doOutputs()`), etc for each FSM
- Commits DOUT outputs by writing them to hardware (`commitDataWrites()`)
- Do any DAQ (data acquisition) processing (this is a rarely used feature) (`doDAQ()`)
- Goes to sleep for 1 task cycle and starts from the top again.

FSM.h

This contains the shared memory struct declarations and the structs that go into the RT-FIFOs. Basically, all communication between the userspace "FSMServer" program and the "RealtimeFSM.ko" module happens using the data declarations present in this file. The structure `'struct ShmMsg'` is a good place to start analyzing the code since this is the data structure "FSMServer" and "RealtimeFSM.ko" use to communicate. How the structure is intepreted depends on the `'ShmMsgID'` in the `.id` field of the structure.

FSMServer Rough Breakdown

FSMServer.cpp

This is the server program that listens on port 3333 and accepts commands from matlab. If you wanted to add new commands or to modify existing commands, the main code of interest is in the `'ConnectionThread::threadFunc()'` member function. Here is where all protocol commands are parsed and, if need be, communication happens with the kernel *realtime task* via shared memory and fifos.

The mechanism for communication with the kernel is:

1. a `ShmMsg` struct is prepared (see **FSM.h**) which contains the correct `ShmMsgID` that corresponds to the command/query we are performing to the realtime task. Auxilliary data for he message is also prepared.
2. this prepared struct is written to the rt-shm that was attached at program startup
3. a small int is written to the rt-fifo to tell the kernel that the rt-shm as changed.

Have a look in `'ConnectionThread::sendToRT()'` to see this in the code.

Additional Thoughts and Observations

Aside from the client/server architecture between the matlab client(s) and the FSMServer, communication between the userspace FSMServer and the kernelspace RealtimeFSM.ko actually also follows a quasi-client/server architecture, because requests are sent synchronously from the FSMServer to the RealtimeFSM.ko module (via the aformentioned rt-shm and RT-FIFO mechanisms), where they are serviced, and results are sent back to the FSMServer when complete. Thus the flow of information would be:

MATLAB CALL -->(via TCP)--> FSMServer -->(via rt-shm and RT-FIFOS) --> RealtimeFSM.ko

and results of matlab calls would be returned in the reverse order:

RealtimeFSM.ko -->(via rt-shm and RT-FIFOS)--> FSMServer -->(via TCP) --> MATLAB RESULT

This is basically a 3-tiered client/server paradigm.