A walk through the Mini-MOST DAQ code

Paul Hubbard
April 15, 2004

Outline of talk

- Walk through API calls and their use
- The goal is to see them in context, and discuss how they are used
- Two examples: Mini-MOST DAQ, and the Anco DAQ from Argonne.
- http://www.mcs.anl.gov/neesgrid/ has links to all of the code examples

Code URLs

- http://www.mcs.anl.gov/neesgrid/second-example/
  - Main program we'll be covering
- http://www.mcs.anl.gov/neesgrid/mmost-example/
  - Mini-MOST DAQ code, very similar
- http://www.mcs.anl.gov/neesgrid/anco-daq/
  - Custom DAQ subroutine
- http://www.mcs.anl.gov/neesgrid/mmost-daq/
  - Custom DAQ subroutine for Mini-MOST
- http://www.mcs.anl.gov/neesgrid/mmost-filter/
  - Load cell data filter for Mini-MOST

Misc. initialization, sequence 0

- Clear the front panel, set variables
- DAQ trigger occurrence - this is how the control program triggers acquisition at the end of a move
- DAQ hardware semaphore - this mediates access to the DAQ hardware between the control program and DAQ code. The underlying problems is that NI-DAQ cannot handle two programs reading the hardware at once. You get 'Error -10681' and other such informative messages.
- ADXL open - open init the serial ports; this example shows the use of 2 serial instruments.

Misc init, sequence frame 1

- String array to I/O:
  - DAQ channels are an array of strings; we need to convert them into a comma-delimited list that the other routines expect. At the same time we are adding the 4 ADXL channels to the list.
- Copy metadata:
  - If we are using SAMBA or NFS, copy metadata over and add .written semaphore file

Metadata save.vi

- Writes channel names (comma delimited), sample rate (0 for variable), data file name and channel units (from MAX) to metadata.ini file.
- Read by 'server daemon.vi' & datafile routines (saved into file header)
  - Server daemon relies upon this when listing channels
- Call this before any DAQ is started
- Note that the channel names include serial as well as NI-DAQ channels, so we have to merge them

Write file header.vi

- Writes metadata into a slightly different format, designed for parsing by the repository
- Call before acquiring any data; easiest to call right after metadata save and chain the errors
- Also writes the column headers
- Will overwrite any existing file with no warning!
- Example output:

Event ID: 19364841J12
Active channels: ACH1,Temp
Sample rate: 200.000000
Channel units: V,Deg C

Time ACH1 Temp

DAQ Status Write.vi

- Writes one of the enumerated statuses into the shared (global variable) area
  - Running, stopped, error, etc
- Read by ‘server daemon’ and sent to NSDS upon request.
- Should be called whenever status changes
- Can also add your own strings, see ‘Get status and time.vi’

Main loop - notes

- Anco uses two loops, running parallel. One does overhead (elapsed time calculations, updating front panel, check for ‘stop’ button, sleep), second loop does the heavy lifting.
- ‘Write data to sequenced file’ is a hack. It writes the current data to a standalone file, e.g. ‘minimost-224.txt’ and uploads it via SAMBA/NFS or ftp. This is grabbed by the NFMS scraper (a.k.a. ‘ingestion tool’) and uploaded into the repository. Lots of overhead, not super fast, may go away soon.

Anco DAQ routine

- Instead of just calling NI-DAQ routines, we have to:
  - Get the DAQ semaphore
  - Read the NI-DAQ channels
  - Release the semaphore
  - Read the serial channels
  - Merge the results, in the correct order
- All of this is to avoid colliding with the control program, which needs an analog input

Mini-MOST DAQ routine

- Similar to Anco DAQ, we have to lock/unlock the semaphore, read the normal channels, and then call the routine to read the load cell.
- The load cell has its own routine: The sensor's signal conditioning unit adds a 30Hz noise spike we need to remove. This routine reads at 1kHz for .5 seconds, and then runs the data through an 8-pole Butterworth lowpass.
  - This filtering routine is called by both DAQ and control, so it has its own lock/unlock calls.
- Since we have at least .5 seconds of read, this limits the max speed of the Mini-MOST.

Data array to NSDS stream.vi

- Input: Vector of real-valued measurements and a list of associated channels
  - Order must match!
- Output: ASCII data, with timestamp in ISO8601 format, with only subscribed channels included
- Reads from the global list of subscribed channels, as maintained by server daemon.
- Can set timestamp and precision

Example:

2003-01-24T15:42:02.73399 ACH1 -0.700531 Temp 27.603149

TCP Conditional data send.vi

- If no connection, returns immediately
  - Status and channel handle maintained by server daemon and this routine
  - No CPU load if no network connection
- If no data subscribed, returns immediately
- If error returned from TCP:
  - changes status of network to ‘disconnected’ - one time glitch only
  - The driver will initiate a reconnect, streaming resumed by the NSDS
- Buffering is done by the driver

Data array to datafile stream.vi

- Formats *all* channels to save to disk
  - Tab-delimited, channel IDs not repeated
  - We assume all data goes to disk
  - Therefore, only inputs data and not channel IDs
- Just builds the string; doesn't save
- Very similar to 'Data array to NSDS stream' - this version always saves all channels; NSDS only does subscribed channels.

E.g. 2003-01-24T15:42:02.73399 -0.700531 27.603149

FTP Three-phase write.vi

- Uploads finished data file to FTP server, sometimes called the local repository
- Requires NI 'Internet Toolkit' add-in package
- Writes datafile.txt, then datafile.txt.written
    - Filename just an example
    - .written is a signal to the repository that uploading is completed
- Sometimes first upload fails, reason unknown. Also need tcpfix.llb!

Things done with LV code

- Save to disk - use their code that opens and closes the file at each step.
- DAQ - this allows the easy drop-in of third-party hardware (e.g. UEI) or software
- INI files
- Internet toolkit - ftp portion only
- We call MAX routines to get channel units - this is in metadata save routine.

Server daemon interactions

- Sets TCP status, which governs the data sender
- Sets connection handles for control and data
- Repeats status, list of channels, channel units
- Handles subscriptions and associated list
  - No data sent unless server says so!

Other Notes

- Examples are low-speed code; not tuned for throughput
- One-time glitch on TCP error
  - Queueing code took more CPU, but is in CVS for exploration
- Server daemon is TCP-responsive; needs to poll the socket for better interactive response
- NSDS simulator essential for testing
- DNDTester for testing and validation