Domino: Mark5B control software

Bob Eldering, Software Engineer JIVE

July 29th, 2009

Contents

1 Introduction

In the current correlator system, Mark5As play data to their Station Units (SUs). The SUs apply the model to the data stream and forward the data in Correlator Frames to the correlator. The Mark5B will replace these two machine.

Domino is the program responsible for driving the Mark5B in output mode. The output mode has two sub modes, "pure" VSI and SU emulation mode. As we want to replace the SU, we will (for now) only use Domino in SU emulation mode.

See the Mark5B user manual at `http://www.haystack.mit.edu/tech/vlbi/mark5/docs/Mark%205B%20users%20manual.pdf` for details on the operation and capabilities of the Mark5B. In this document we will focus on the code of Domino and the changes that have been made to make it work with the JIVE correlator system.

2 Modules

Domino is composed of a number of modules:

- domino: the main module, it will call several initialize modules to initialize both the StreamStor and the IO board. After initializing, it will go in an infinite loop calling handlers for several events. The header file defines a few global utility functions and a big struct which contains the global state.
- configurer: the code in this module is mainly responsible for preparing the hardware components for a job (or scan in Domino terminology). It will configure the StreamStor, the IO board and setup sockets when realtime play back is used.
- iserver: handles ROT clock messages and hardware interrupts. These hardware interrupt include:
  - ROT clock ticks to start the internal ROT clock
  - Correlator Frame interrupts, iserver will prepare a Correlator Frame and update the internal ROT clock when these interrupts arrive.
  - Phase Calibration interrupts will make iserver update the global Phase Calibrator accumulator tables.
- package: creates the Correlator Frame headers. See the next section for more information.
- periodic_services: keeps track of state of the Mark5B and reports to whichever program has signed itself up for periodic state messages.

- sequencer: the control architecture of domino is focused around a state machine for each command. Sequencer is the module that runs these state machines.

  For the realtime command it writes data from the software buffer to the StreamStor.

- spooler: this module's task is to collect data from the socket and write it in the appropriate place in the software buffer. As mentioned, sequencer will forward the data from this buffer to the StreamStor.

- message_handler: this module will receive message from the correlator control system and set the state machine such that sequencer will handle it from there. See `http://www.jive.nl/~jive_cc/sin/sin16.pdf` for details on how messages are sent between the correlator control system and domino.

These modules are running in three separate threads. The main thread runs from the domino module, which simply calls sequencer, periodic_services and message_handler. The two helper threads that are spawned run iserver and spooler.

3 Changes

Domino has been written at Haystack, to make it work with the JIVE correlator control system in general and eVLBI in particular, we had to make some changes.

- Domino assumed a UDP packet size equal to a Mark5B data frame. Transferring data with fixed packet sizes is detrimental for the data stream. As what actually will happen is that the packets sent to the network layer will be divided into smaller packets which fit in the transfer used for that particular connection. This means more header overhead than strictly required. An even more important disadvantage is when those smaller packets are reconstructed to the original packets. As the transfer protocol used is UDP, if any of the smaller packets is lost, the whole original packet is lost. So splitting up packets will cause more data to be lost.

  This is one of the reasons most data sources we use do not comply to the assumption that packets have the size of a data frame.

  So we have changed the software such that it does not require data packets to be sized equal to data frames. Having this equality did have the advantage that you can enforce the data header to be at a fixed point in the data packet. The header is needed to determine the mapping from the packet sequence number to the time represented by the data samples. Therefore we now have to search the data for this header. The burden on the processor to do

this is limited since we only have to find one header to establish the mapping. The process of finding this header is executed before any data is actually streamed to the IO board.

- Haystack and JIVE use the same correlator and SUs. However they are controlled differently and the software on these machines has grown apart over the years.

  The Correlator Frames that are made by the SUs contain a quadratic model that should be applied by correlator. An error in the handling of this model was found at Haystack. The fix required changes in both the Correlator Frame header and the correlator software. The details are described in `http://www.haystack.mit.edu/geo/mark4/memos/291.pdf`.

  That document also briefly mentions that the Haystack correlator runs at two Correlator Frames per second. Since the JIVE correlator runs at a much higher Correlator Frame rate, the problems described are negligible for the JIVE correlator system. Therefore these changes were not made to the JIVE system.

  However, Domino did have these changes as it was developed as a replacement for the SUs at the Haystack correlator. This means that the whole package module has been replaced by the corresponding code from the SU control software.

- As said the Mark5B is designed to replace the Mark5A disk play back unit and the SU. Before the Mark5A was put into place, the SU used to get its data from a tape play back unit. The system of recording and playing back from tapes caused the data arriving at the SU to be not quite 100% reliable. Therefore the SU has been designed to be fault tolerant when it comes to its input data stream.

  As disk play back is a lot more reliable, the Mark5B IO board has been designed with a 100% reliable data stream in mind. Any header that is not as expected will cause the job to be aborted.

  Unfortunately, an UDP data stream is not 100% reliable, any packet may be lost. The Mark5B does support headers to be replaced by a byte stream of pre-determined invalid byte codes, so we can replace packets lost with invalid marked bytes. According to specification, if any byte in the header is marked invalid, the whole of the data frame will be considered invalid. However, testing showed that when only half of the header is marked invalid, this will cause the IO board to throw a header error and the job to be aborted. And even if none of the bytes are marked invalid, UDP does not guarantee that the value of bytes are as the data source has sent them.

  Since some data loss or corruption is bound to happen during an eVLBI experiment, we have to check and repair those headers. This means that we have to keep track of what the headers should look like and check all headers against these expected values. Any

time the check fails, we replace the complete header with bytes marked as invalid.

Another related problem is that the first header of the configured job has to be a header with valid data. With disk based play back, you can just search for the first valid header in your data and then configure the Mark5B to start play back from there. With eVLBI this is not possible of course. Therefore we will compute this header ourselves and insert it into the data stream. This does mean that one data frame that might contain invalid data will be considered valid unconditionally.

- Data rates collected by the Mark5B still have to be powers of two Mbps. Quite a few stations have a 1Gbps connections to JIVE. The power of two restriction would force them to send at a maximum data rate of 512Mbps, wasting almost half of the connection's capacity.

  To fully utilize all connections we have developed an algorithm that will compress a data packet by throwing away certain bitstreams. Bits from the bitstreams we want to keep, that occupy the bytes at the end of a packet will be moved to the bit positions in the beginning of the packet corresponding to bitstreams we want to throw away. At the receiving side these packets need to be decompressed, which means restoring the bits to their original positions in the packet. This feature has been added to Domino.

4 Future work

As said in Section 2, Domino has three threads. However none of these threads are in any way guaranteed to have time scheduled on the processor. For the main thread and spooler this usually is not a big problem, small hick-ups can be handled. On the other hand, iserver does need to run every $\frac{1}{64}^{th}$ of a second at least once, as it needs to catch every Begin Of Correlator Frame interrupt.

As said there is no such guarantee, certainly when the data stream needs to be decompressed as discussed in the previous section this may become a problems. Yet during testing this has not been a problem.

Possible solutions would be to run Domino on a realtime kernel and prioritize the iserver thread or to use a buffer to be able to handle small hick-ups.

As hinted in Section 2, a scan for Domino is what at JIVE is considered a (sub)job. This means that every scan needs a new configuration of Domino. Luckily scans that were recorded continuously on the disk can be joined to be presented as one scan to Domino. However when there is a jump in time in the data recorded on disk, a reconfiguration is required. Such a reconfiguration would require about five seconds.

This means that when a gap is as small as, say, ten seconds, we might not be able to reconfigure quick enough for Domino to be ready at the next scan's start time.

A possible solution would be to make a new subjob when such a situation is about to happen. This would take quite some time, usually more than two minutes, but then again, these situations do not occur often. If this solution takes too much operational time, it might be possible to change Domino such that it can actually handle gaps in one configuration.