NDAS Hardware Translation Layer Development

Ryan N. Nazaretian
USRP Intern, Long Beach, MS 39560

I. ABSTRACT

The NASA Data Acquisition System (NDAS) project is aimed to replace all DAS software for NASA's Rocket Testing Facilities. There must be a software-hardware translation layer so the software can properly talk to the hardware. Since the hardware from each test stand varies, drivers for each stand have to be made. These drivers will act more like plugins for the software. If the software is being used in E3, then the software should point to the E3 driver package. If the software is being used at B2, then the software should point to the B2 driver package. The driver packages should also be filled with hardware drivers that are universal to the DAS system. For example, since A1, A2, and B2 all use the Preston 8300AU signal conditioners, then the driver for those three stands should be the same and updated collectively.

II. INTRODUCTION

The NASA Data Acquisition Project NDAS is a new software suite designed to handle the low speed data acquisition processes of NASA's rocket propulsion test facilities. In Phase I, NDAS is to support A1, A2, and B2 at Stennis Space Center (SSC). Phase 2, to be completed at a later date, is to support the rocket propulsion test facilities at B2 and the E-Complex (building 4010) at SSC, as well as the rocket propulsion test facilities at White Sands Test Facility in Las Cruces, New Mexico, and rocket propulsion test facilities at the Marshall Space Flight Center in Huntsville, Alabama.

This project is designed to eliminate the complications and differences between test stands. An issue that has existed for many years is the lack of consistency between the DAS systems. At the E-Complex, all three test stands use a LabVIEW based DAS, but each is run differently. Even larger changes occur once you compare the E-Complex DAS system with the DAS system's running at the A-Complex and B-Complex.

With the changes in computer technology and software over the past few years, NASA decided to start a project to replace every DAS system for its rocket propulsion test facilities. NDAS will have the same interface for each test stand, and will communicate with all the different hardware present at each test stand, which means it has to be adaptable with all the hardware. The NDAS Hardware Translation Layer (NXLT) and the NDAS Calibration Hardware Translation Layer (NCXLT) will be responsible for communicating with the hardware on the test stands. Drivers must be made for each piece of hardware to allow the software to communicate properly and efficiently. There are two main parts of the DAS hardware system, the data acquisition equipment, and the calibration equipment.

III. NXLT & NCXLT INTRODUCTION

The hardware translation layer and calibration hardware translation layer are divided into two systems. The NXLT consists of only the measurement acquisition devices, in my case, the Preston Presys 1000 Data Conversion System. The NCXLT or the NDAS Calibration Hardware Translation Layer consists of all the other devices including signal conditioners/amplifiers, voltage standards, relay switch boxes, and function generators. The main difference between the two is that the NCXLT equipment is not written to, or read from during testing. It remains in a stable set state until calibration.

All of the drivers for the devices had to be made from scratch. I was responsible for making the Preston 8300AU Signal Conditioner drivers. I was also responsible for designing and testing some functions of the Presys. I developed a real time application to run on top of the LabVIEW Real Time OS. This program communicates with the Preston Presys 1000. It configures the Presys 1000 for an external burst rate, sets up the maximum channel scan list, and places the Presys 1000 into run mode. It will then begin reading all the channels and placing the data on the IEEE 488 GPIB bus to be read by the real time computer. The real time computer will then push the data to a shared variable, either locally hosted on the RT computer, or remotely on a shared variable server.

Figure 1, GPIB Connector

Shared Variables are a useful tool included with LabVIEW. They are based on TCP/IP, but are configured and maintained in the background by LabVIEW. Basically, they allow for data to be shared over the network with minimal configuration, usually just an IP address or computer name. They share the same data type and work quickly and reliable, perfect for distributing hundreds of measurements at 250 samples per second over the network.

All of the software talks to the hardware over the IEEE 488 GPIB standard. GPIB, or General Purpose Interface Bus, or often called the General Purpose Instrument Bus, was developed originally in the late 1960s by Hewlett Packard, and was originally called the HPIB. GPIB is an address based parallel communication architecture. It is commonly used in lab equipment, and can be found on many lab multimeters, such as the Agilent 34401A.

IV. NXLT DEVELOPMENT

The NXLT is responsible for communicating with the device responsible for taking the signals and sending them to the computer. We often call them digitizers, due to the fact that they convert the signal data into a digital signal for the computer. The two types of digitizers at Stennis are the Preston Presys 1000 and the Tustin 410 family of data acquisition equipment. The digitizer typically takes in hundreds of signals (analog and discrete), and adds them to the

Figure 2, Presys 1000 Data Acquisition System

GPIB buffer. The computer reads the buffer and translates the data.

For the Presys, all analog inputs are 16bit Signed Integers, meaning they have a range from -32768 to 32767. We call these numbers counts, as they do not represent voltage, pressure, temperature, etc… they are just a number. To convert counts to an engineering unit, it usually needs to be converted to voltage first. The Presys 1000 has a voltage range of -10.24V to 10.24V. This is the equation to convert the Presys 1000 counts to voltage.

$$Voltage = \frac{Counts}{32767} \times 10.24$$

Or simply…

$$Voltage = \ 0.000312509537034211 2 \times Counts$$

After getting the signal to a voltage, most conversions are a simple liner equation following the $Y = M \cdot X + B$ equation with slope and Y-intercept.

The discrete inputs are fairly basic. The Presys will send you the signed 16-bit integer, but then you have to convert it to binary to give you 16 digital channels. The time word from the Inter-range instrumentation group time code (IRIG Time Code) is given in binary coded decimal format (BCD) using discrete inputs. The conversion for this is quite complex, but is given in the following LabVIEW programming. Also shown is the table on how to decode BCD for IRIG B.

Word0															
Seconds (0-9) x 1				Milliseconds (0-9) x 100				(0-9) x 10				(0-9) x 1			
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Word1															
Hours						Minutes							Seconds		
(0-3) x 10		(0-9) x 1				(0-7) x 10			(0-9) x 1				(0-7) x 10		
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Word2															
Unused						Day of Year (0-3) x 100		(0-9) x 10				(0-9) x 1			
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Figure 3, NASA's IRIG BCD Format

What this program does is take three array elements from the data array and separates them into 3 individual elements signed 16-bit integers (I16s). The I16s are then converted to binary arrays. These arrays then stripped even further to make the individual parts of the BCD format. BCD works with binary (0, 1, 2, 4, 8, etc…) then uses a known multiplier to build each digit. So If I want the number 24, I would need the x10 multiplier to be 2, and the x1 multiplier to be 4, so the final output would be 0b0010, 0b0100. The only issue with this format of IRIG is that it does not include the year information. To convert this to a LabVIEW compatible time stamp, you must include the year. I developed a way to do this. LabVIEW time stamps are two 64-bit numbers based on the number of seconds since Friday, January 1st, 1904 and the fractional part of a second. January 1st, 1904 was picked as the epoch because it was the first leap year of the 20th century (2). I have to solve how many seconds there have been since January 1, 1904 and January 1, 2011. I then found a mean for how many days there are in a year (as it is not exactly 365.25), and made can now multiply by what year the computer says it is, and append that with the IRIG data to get the LabVIEW Time Code.

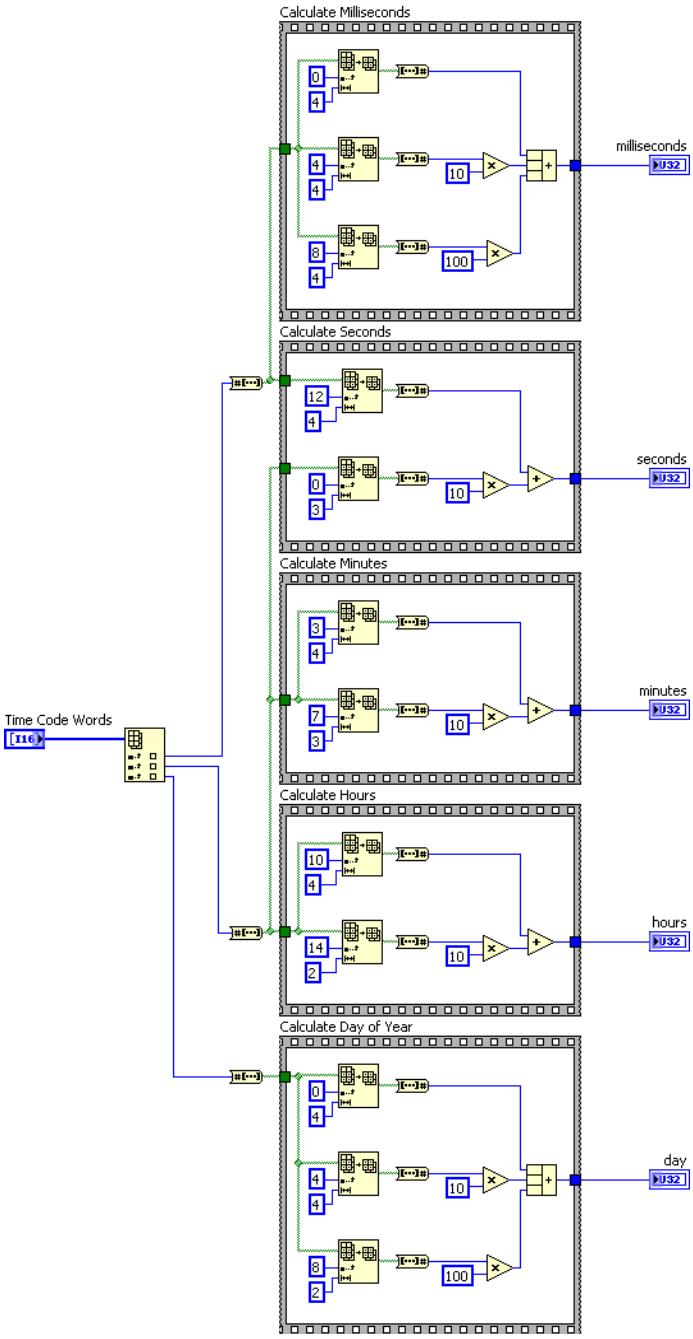

Figure 4, IRIG Decoding Program

Another design challenge was making the system reliable and self-debugging. It goes through a quick check process during boot up, checking Presys communication, among other things. When it starts the main process, it has five independent 'while' loops running. What this allows me to do is to keep most things running if one part of the program encounters an error. The five independent loops are Performance and Memory Monitor, System Events, Data Acquisition (from Presys), processing and pushing out the data to the network, and recording the

data locally. This system is designed to run at all times, recording a temporary file at 250 samples per second. It will record in a circular loop, of a defined length, as to never require user interaction, and to always record data in case unforeseen events. This is also a way to keep a redundant recording system. If the network link between the test control center (TCC) and the test stand is lost, then the computer in the signal conditioning room will continue recording all the data. On top of this redundancy, there is also a secondary DAS system running, so there is almost complete redundancy in the whole system.

Most of this redundancy was put in place before the NDAS project, but the NDAS project group went through a fault tree analysis (FTA) for the project, brainstorming all the possible failures that could occur with the NDAS system. After the FTA, we designed ways to prevent, or mitigate, the loss of mission for the NDAS project, which is data acquisition. We have to make sure that if one part of the system fails, such as the network stream between the actual data acquisition computer and the data logging machine, that no data is lost. We mitigate this by including a circular buffer at the data acquisition level that will record a pre-determined length of data at the full sample rate, for example, a week's worth of data at 250 samples per second. This functionality will run all the time, so if there was an unexpected event on the test stand, even on the weekend or during the middle of the night, then the data system will capture it at the full sample rate, plus the data system will be recording data permanently at a lower sample rate. This will allow engineers to solve what caused the event. The test stand itself has many redundant systems engineered in as well.

As mentioned before, B2 has Primary and Secondary DAS systems as well as Primary and Secondary Event systems. This will hopefully prevent data loss if one of the systems fails. Another engineered in redundancy are the IRIG time generators. There are, however, some systems in place that are not redundant. If one of the signal conditioners fails, then there will be complete data loss, or loss of data integrity on that particular channel. Another aspect to look at is how the power is run to all of the equipment. Each rack of equipment is fed by one power outlet, which is also fed from a dedicated breaker for that one outlet for that one rack. If the breaker trips, all the equipment in that rack will stop be powered off and cease to work. Most of the equipment has been installed in a way to minimize the impact of a power loss, such as the Primary and Secondary DAS's, but the Event's master units runs off of the same power line, so a loss of power for that rack will end up with complete data loss of the Event's Primary and Secondary system. Another note to make is that a power loss for a rack of the 8300AU signal conditioners in B2 will result in a significant amount of data loss (6 chassis at 16 channels per chassis would result in 96 channels of complete data loss).

An unusual setup for redundancy we found involves the previously mentioned IRIG time generators. One IRIG generator feeds the Primary DAS System, and the Secondary Event's System, while the second IRIG generator feeds the Secondary DAS System and the Primary Event's System. While this is still redundant, it does not make sense to take out one part of each DAS system. As a note though, the B2 DAS system was never verified after it was installed

because the program it was renovated for was canceled before it ever started. This also may have caused some of the problems we were having with the translation layer.

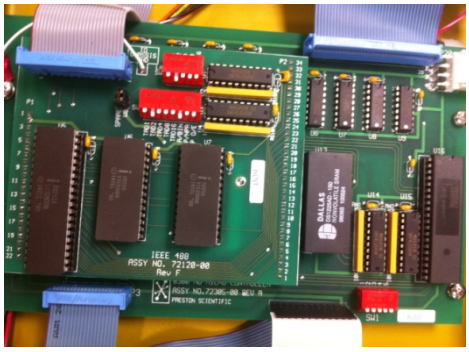

Figure 5, Motherboard of the Preston 8300AU Signal Conditioner

During development, we moved some hardware from the B2 test stand to the B2 test control center, where we are doing all of our development. Some of the hardware has been troublesome for us. The Preston Presys 1000 Data Acquisition System sometimes does not start up correctly. We also have been having issues with the Preston 8300AU Signal Conditioners locking up while communicating with them. On top of this, all of the electronics inside of the Preston equipment is even older. The Presys 1000s were purchased in 1996, 15 years ago. They are running on an Intel 8085 microprocessor, which was designed in 1977, so the actual hardware is over 30 years old. All of this hardware is expected to be able record over 600 channels of data at 250 samples per second with high accuracy and precision. At this point of development, we are able to get the required data rate and data integrity, so we are confident we can develop the new software to meet all the requirements.

The actual computers we're using for the translation layers are rack mountable commercial-off-the-shelf (COTS) computers with an ordinary Intel Core 2 Duo CPU running at 2.66GHz, 2GB of DDR2 RAM, and dual 160GB 7200RPM hard drives. These machines are capable of acquiring the 600+ channels of data at 250 samples per second, converting the data into voltage, converting the voltages into engineering units, converting the time stamp information, recording the data locally to the hard drive, reporting CPU and RAM usage, and pushing the data out onto the network at the 250 samples per second with around 10% CPU usage, 85MB of RAM, and using about 1% of the network utilization. This is the computer that runs the LabVIEW Real Time Operating System mentioned earlier. It is also using the Datalight Reliance file system. Both the Real Time OS and the Datalight Reliance file system will provide a robust data system

Figure 6, NDAS Computer

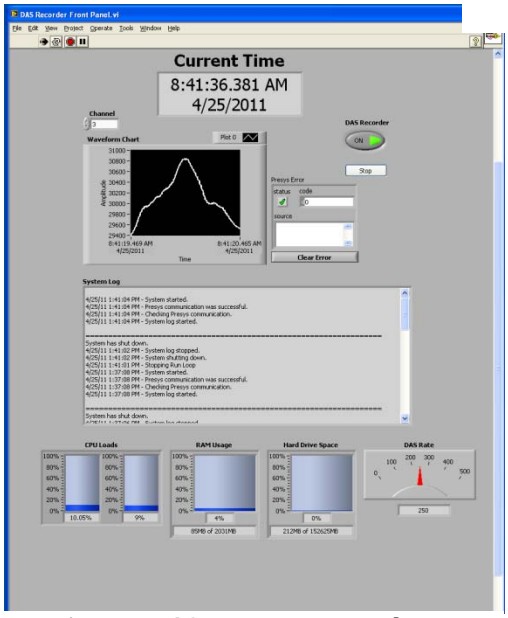

Figure 7, DAS NXLT Front Panel Test

that will need very little maintenance, and will prevent issues from occurring. The old DAS system ran on Windows XP, which was optimized for real-time use. Windows, by default, includes many 'behind-the-scenes' optimizations to keep itself running well. Those optimizations can impact the data integrity, causing lags in the data processing, and reduces overall reliability of the system as a whole. Windows is simply not needed, and the LabVIEW Real Time Operating System is a much better choice for NDAS. One big feature is that it is command line based, with no user input. This provides a denial of service security policy for the machine, where no user can alter settings or run additional software on the machine, providing a more reliable system onto which to run the DAS software.

Figure 8, DAS NXLT Command Line Interface

V. NCXLT DEVELOPMENT

Like the NXLT Development, the NASA Calibration Hardware Translation Layer Development involves much of the same hardware, including the same COTS rack mounted computer and same GPIB connections. The big difference between NCXLT and NXLT is that NCXLT does not take any data in. NCXLT sets parameters in order to perform calibration. Even the data taken in needed for calibration is read from the NXLT. NCXLT does have the task of communicating to multiple pieces of equipment though. The NXLT only has to communicate to the data acquisition hardware, in our test cases, the Presys 1000. The NCXLT has to communicate with the Preston 8300AU Signal Conditioners, Agilent Multimeter, Agilent Multifunction Switch,

Figure 9, Preston 8300AU Signal Conditioners

Krohn-Hite Voltage Source, and Stanford Research System's Function Generator with the GPIB interface for the B2 test stand (other test stands may have other equipment). At this time, I only have drivers made for the Preston 8300AU Signal Conditioners. The driver development took much longer than expected as the hardware I was trying to test them on was defective. Rather than blaming the hardware, I decided to blame my software until I was absolutely convinced that the hardware was faulty when the old calibration software, made by Dan Goad at SSC also would crash while communicating with the signal conditioners. Now that I'm finished with the Preston 8300AU drivers, I was given more documentation that could speed up programming the signal conditioners significantly and provide for a way to backup the signal conditioner setup by creating a memory map for the amplifiers and dumping that memory map to the signal conditioner's controller. Luckily, the multimeters used at the test stands are widely used, and there are already drivers pre-built in LabVIEW to use. All the other equipment must have drivers written.

VI. CONCLUSION

The NDAS Hardware Translation Layer development has been a mixed experience. The age of equipment, lack of documentation, or just the differences in documentation from different time periods made designing the drivers for the hardware difficult for Jon Morris and I. The Presys 1000 uses binary programming and the documentation has scattered information throughout the user manual, all of which is very important for the programming sequence. For the Preston 8300AU Signal Conditioners, there are about three different versions of the manual available, and then there were undocumented commands that were implemented for NASA's use back in 1997. A month after beginning driver development for the signal conditioners, I was given the official documentation for those additional commands.

The lack of boarding the equipment also caused quite a few problems. We were never sure if the equipment worked, and if it was not working at all, what to even do. We kept finding tidbits of information that, in the end, helped us write our software. Our goal was to get a fast and reliable data acquisition system working for NASA. So far, we're meeting our goal. Nothing ever works as expected the first time it is tried. Work is still needed on all parts of the NXLT and NCXLT, but with some time, all the problems will be ironed out.

REFERENCES

1. Wikipedia – IEEE-488, http://en.wikipedia.org/wiki/IEEE-488, Accessed April 20, 2011.
2. Wikipedia – Epoch (reference date), http://en.wikipedia.org/wiki/Epoch_%28reference_date%29, Accessed April 21, 2011.