Feature Creep for Efficiency

Malcolm Weir[1]
[1] Ampex Data Systems Corporation, Hayward, California USA,
mweir@ampex.com

Abstract:

For most project managers, additions to the scope of a project are problematic: not only must the project complete its original goals, but also new ones. And while cost is naturally important, perhaps the biggest challenge is adapting the design of a system to incorporate new functionality, which may lead to inefficiency in development and implementation. This paper describes the sequence of scope changes to a data collection system, and how collaboration between the customer, the end user and the suppliers increased the functionality of the solution as the capabilities expanded. The result is a system which started as a simple data collection device and has evolved into a traffic management system handling data format conversion, security oversight, instrumentation command and control, traffic routing, and, still, data collection. This paper provides insight into the process by which the new functionality was implemented, illustrating how economies of power, mass and volume can be achieved by the use of specialist software running on versatile hardware. That process is in contrast to the more traditional "one box, one function" approach with hardware modules, which is undoubtedly elegant but also less efficient than a software-centric design.”.

Key words: Data acquisition & networks, Networks & Architecture, Data Management,Applications, Data Management Standards, Security of data, links and networks.

Introduction

Over the past 20 years or so, the composition of telemetry and instrumentation systems has changed. For most of those years, there has been presentations, discussions, papers and exhibits on how the “new” technology can be used to deliver various advantages over the “old” approaches.

This is another such paper.

But unlike those earlier write-ups, this one is oriented from the problem-solving perspective: how can a device serve the required purposes.

It must be noted that this paper is “loosely based” on a real project, but it does not focus on the specifics of that program, for two main reasons: first, no two projects are identical, so whatever challenges and solutions one program team identifies, the chances are very likely that another effort would have, at best, similar-but-different tasks, and the second reason is that the a deep-dive into a single program is likely to be both irrelevant to even well-versed readers, and may also contain proprietary or even confidential information.

Accordingly, this paper describes a “lightly fictionalized” program, based on a real effort but with excessive detail omitted for the sake of readability.

Similarly, the specific detailed capabilities of the test equipment used in the program isn’t described unless that capability is relevant to the narrative. For example, efforts required to work around specific limitations of test equipment (ours or that of other vendors) are not particularly interesting to anyone who isn’t using that specific set of equipment, so are omitted regardless of the amount of effort involved in the work-around – unless, of course, the limitation was a product of a standard or architectural model that would apply to a whole class of devices.

Background

At first glance, the requirements of the program for a Mission Data Recorder were not particularly onerous: a few hundred megabits of data and a few terabytes of storage. The I/O interfaces were straightforward Ethernet, the recording format should be IRIG-106, *Inter-Range Instrumentation Group (IRIG) 106 Chapter 10 Digital Recording Standard*, with the only marginally unusual feature being a requirement

that the recorder be able to act as a Grand Master time source in accordance with IEEE/1588, *Institute of Electrical and Electronics Engineers Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,* version 2.

Certainly, there were also atypical file retrieval specifications: in accordance with IETF RFC 959, *File Transfer Protocol,* using Transport Layer Security (TLS) in accordance with IETF RFC 4217, *Securing FTP with TLS*. However, perhaps the atypicality of this is remarkable mostly by the fact that *all* FTP connections aren't routinely secured with TLS!

There were also requirements for video manipulation (changing frame rates and compression ratios), but these were fairly localized to a particular retrieval channel.

The environmental requirements were fairly conventional, with the exception of those for radiation exposure; since the vehicle was designed for extreme altitudes, the objective was for the solution to be "rad hard"; or to be exact, tolerant of higher-than-normal radiation levels; see Figure 1.

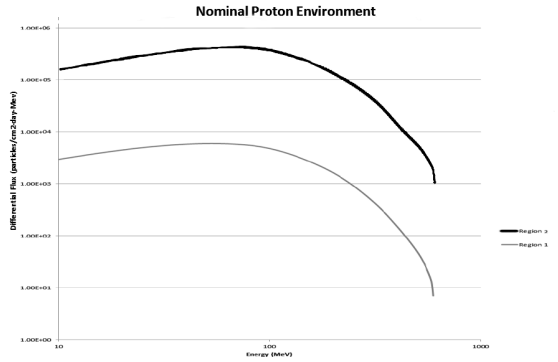

Figure 1 Radiation Levels

The relevance of the radiation tolerance may seem slight when viewed in the context of the topic of this paper, but as will be seen it is a critical factor in identifying solutions.

Trimming Chapter 10

As noted earlier, the requirements started by specifying an IRIG 106 Chapter 10 recorder.

However, Chapter 10 contains multiple disparate sections, some of which are organized into Chapters of their own: Chapter 6 for Command and Control, Chapter 9 for Metadata Description and Configuration, and Chapter 11 for Packet Formats.

And in addition to those sections in separate Chapters, there are significant functional requirements remaining in Chapter 10, which are relatively independent in isolation but combine to define a particularly specific and interoperable recorder. See Figure 2 for the visual guide to the standard from the 2019 version of the standard, which is a little out of date but not inherently incorrect (the Data Format & Packetization Definition that is referenced as being in section 10.6 is actually now in Chapter 11, but section 10.6 now effectively states "see Chapter 11").

In addition to requirements for an on-board recorder, Chapter 10 reaches beyond the device to mandate practices for naming files once downloaded, and even when modified. Obviously, this sort of detail is overreach for the standard's stated purpose, but relatively harmless and easily ignored without breaking much, if anything. (Things like the naming of files properly belong in a "best practices" handbook, not a formal standard).

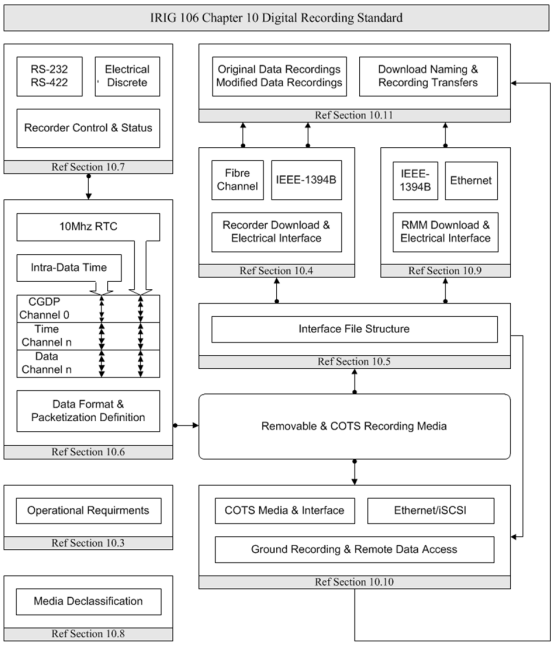

Figure 2 Diagram of Chapter 10 Standard (2019)

The Disk and Filesystem Problem

One of the challenges of using Chapter 10 for more than just instrumentation data collection (in this case, and for example, general file storage using FTP) is that Chapter 10 embraces a simplistic model of storage devices and requires a primitive filesystem to be used.

The first challenge is that the standard requires that the sector at Logical Block Address (LBA) 1 on the device be used for the "root" of the filesystem (i.e. the first directory block), and that LBA 0 is "reserved", which is interpreted as being set to all zeroes. These two requirements prevent the use of any commonly used volume/partitioning scheme (as LBA 0 technically must be zeroes), and prevent the use of one physical storage device to hold two or

more Chapter 10 filesystems (as the root of the filesystem has to be in LBA 1).

Of course, sophisticated storage controllers can easily get around those issues by combining and then dividing one or more physical devices into multiple targets, each of which appears to be an independent device. But this sort of work-around creates additional complexity as Chapter 10 mandates particular device addresses for the download interface and does not embrace the concept that there may be multiple devices on a single interface.

Given, then, that Chapter 10 only comfortably supports one filesystem, it is unfortunate that the filesystem it mandates is unsuited to general purpose applications.

This filesystem is adapted from a NATO standard, STANAG 4575. However, the intent of that STANAG was (and is) to define a "virtual" filesystem that manufacturers of Intelligence, Surveillance, and Reconnaissance (ISR) recorders could use as a common interface between their internal systems and a NATO download station. It was always envisaged (by the STANAG) that a processor could be in the data path during the download operation, so the filesystem was designed to be an easy "virtualization" of the underlying structures; the STANAG is explicitly designed to be usable without read/write access to the media.

But Chapter 10 adopted the STANAG filesystem in an architecture where there was no expectation of a processor in the download data path, instead expecting a bridge between the download interface and the storage interface (which was most often an industry-standard one like IDE or SATA and so to commodity storage devices like SSDs, although at least one recorder implemented the media interface directly to raw flash chips).

The biggest problem with the filesystem is that it requires all files to occupy (apparently) contiguous storage; that is, if the first sector of a 1,000-sector file is at LBA 1234, then the last sector will be at LBA (1234+1000-1) = 2233. This makes it challenging to support writing/appending to more than one file at a time. Of course, there are ways to mitigate this problem to a greater or lesser degree, such as by performing operations like the old Windows "Defragment" utility to move isolated chunks of one logical file into a single location, but these all impose costs in time, performance and capacity (or all three).

However, the Chapter 10 download interface, and the related filesystem / partitioning constraints, was designed for a use case where a single ground installation (e.g. a test range) would be retrieving test data from a variety of different recorders. But for this specific application, the logistics dictate that the media would only ever be directly accessed in one or two designated facilities, and perhaps not even then since the data can be extracted from the recorder *in situ* using the FTP interface, which might be quicker than waiting until the recorder can be accessed, the media extracted and transported to the facility where it would be accessed.

Therefore, it was agreed that the parts of Chapter 10 relating to the download interface were unnecessary and could be excluded from the requirements with which the recorder must comply that are derived from Chapter 10.

Solution Approach

The first design decision to be made is regarding the product baseline. Obviously, a solution that reuses an existing product is likely to be lower cost and lower risk than one which starts with a "clean sheet", with the tradeoff that a modified product may be less efficient by one or more metrics (size, weight, power, etc.).

The two viable "reuse" options would be to start with a classic "Chapter 10" recorder and add the file server features (support for secured FTP, etc.), or to start with a file server product and add Chapter 10 capabilities. As previously discussed, the filesystem issues inherent with a "pure" Chapter 10 system were moot, so the analysis of the two options boiled down to assessing how much of the workload was classic Chapter 10 data acquisition, and how much was related to post-acquisition processing.

With a view particularly to the video manipulation requirements, a decision was made that a file server platform adapted to handle Chapter 10 would be most appropriate.

A side benefit of using a server platform is that the hardware and software environments are more scalable: if more CPU power is needed, that's usually an easy upgrade.

Chapter 10 Data Type Fidelity

In the development of the Chapter 10 standard, a lot of attention was paid to the idea that the recorder should not manipulate the acquired signals, and therefore recordings made on different products from different vendors could be compared.

This lead to a slightly unexpected resistance within the standard-making body to certain "obvious" data types, initially centered around Time, Space, Position Information (TSPI) data: since that data was usually acquired through a

RS-232 serial port in accordance with the NMEA 0183 standard supported by many GPS receivers, it should be stored in the recording as RS-232 serial data, even though the data was TSPI data and it would be much more convenient if analysis/exploitation software could process it as expected.

The counterpoint to those resisting the sort of manipulation needed to pull TSPI data out of a data stream received over a serial port is simply that treating a serial port as a data type is similarly inappropriate: it should properly (using that logic) be captured as a stream of “1” / “0” pulses, or even a sequence of analog voltage levels. While obviously a preposterous approach, it illustrates the inflexibility of the “no manipulation” position.

This position extended especially to Ethernet: the interface type was Ethernet, therefore the recorder should (only) record data as a series of low-level Ethernet frames. While that might be useful for test instrumentation engineers building a test system (“Is the network working?”), in general that’s less useful than, for instance, pulling out a video streamed over Ethernet and storing it as a video, eliminating as irrelevant the fact that it was transported by Ethernet for at least part of the journey from camera to recorder.

Obviously, for this program, and indeed for all traditional programs, the decision as to how a particular data stream should be recorded belongs solely with the program, and standards should not interfere with the preferences and decisions of the people designing the solution.

This feature represents the first extension of the nominal set of requirements: instead of a pure “Chapter 10 Recorder”, the program needs a recorder that will repackage received data into the appropriate data types, regardless of how that data was delivered.

This feature extension also drives the need either for a more powerful processor and suitable software, or for a dedicated subsystem, likely based on an FPGA and customized firmware. While the latter is likely more efficient by many metrics, the former is more flexible and easier to implement in stages.

Hardening

Having identified that a fileserver system was the best choice on which to base the solution, the next step was to explore how to meet the radiation requirements.

This involved extensive discussion with the customer’s subject matter experts. Given that the recording system wasn’t critical for flight, it was determined that some core “data integrity” steps (such as Error Correcting Codes on the memory interfaces) and thickened chassis walls would likely address data corruption, while a watchdog circuit would be sufficient to protect against the system locking up or crashing.

Component selection for the system as a whole would include radiation considerations; for example, the CPU was selected to use a 22nm FinFET process, which has shown excellent radiation performance, and MOSFETs were derated to 75% of their operating voltages to reduce the probability of a “single event gate rupture”. In addition, lead-based solder was used in place of lead-free processes, and so on.

Obviously, the watchdog circuit would have to be manufactured out of radiation hardened components, as would the power supply, but beyond those two subsystems, everything else could be fairly standard. The watchdog was implemented in a FPGA using a Triple Module Redundancy, to provide an overall system monitor providing a series of counters, checks and responses to provide a fault detection, isolation and recovery (FDIR) methodology.

The only additional design decision was to switch the bulk data storage from being NVMe based on PCI Express back to SATA, since it was considered more likely that an radiation event upset on an NVMe interface could impact more of the system (as PCI Express is the main system bus), while a SATA controller can be reset without restarting the whole machine.

One curious aspect of this specific project is that the mission length is significantly longer than is typical for regular terrestrial flight tests. With a single mission estimated to last about 11 weeks, and the design life of the system required to be for 15 missions, this equates to approximately three years in the high-radiation environment. So while the dosage rates are quite low, the overall dose is not insignificant.

Chapter 10 Data Types, Redux

The implications of the radiation hardening extended beyond the recorder enclosure: since the data acquisition that arrived at the recorder did so over Ethernet, at some point in the overall instrumentation architecture there had to be Data Acquisition Units (DAUs).

The challenge for the recorder is that, because the DAUs *also* had to be radiation tolerant, the number of vendors / product options was extremely limited; in fact, only one vendor offered a range or “rad tolerant” DAUs. Unfortunately, these DAUs are manufactured by a European company, and so their adherence to the Chapter 10 standard was effectively nil; while they collected data and sent it out over Ethernet,

they used the IENA protocol, not the Chapter 10/11 format.

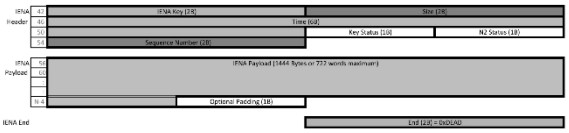

Figure 3 IENA Packet Format

This creates a new problem: since at least one of the customers for the data is a US Government agency, Chapter 11 data packets are a *de facto* requirement. So some degree of data conversion would be required. And the conversion would have to be much more sophisticated than was the case with video.

CHANNEL ID		PACKET SYNC PATTERN		Packet Header
PACKET LENGTH				
DATA LENGTH				
DATA TYPE	PACKET FLAGS	SEQUENCE NUMBER	DATA TYPE VERSION	
RELATIVE TIME COUNTER				
HEADER CHECKSUM		RELATIVE TIME COUNTER		
TIME (LEAST SIGNIFICANT LONG WORD [LSLW])				(Optional) Packet Secondary Header
TIME (MOST SIGNIFICANT LONG WORD [MSLW])				
SECONDARY HEADER CHECKSUM		RESERVED		
CHANNEL-SPECIFIC DATA				Packet Body
INTRA-PACKET TIME STAMP 1				
INTRA-PACKET TIME STAMP 1				
INTRA-PACKET DATA HEADER 1				
DATA 1 WORD 2		DATA 1 WORD 1		
DATA 1 WORD N [FILLER]		:		
DATA CHECKSUM				Packet Trailer

Figure 4 Chapter 11 Packet Structure

To expound on that point: video carried over Ethernet is (usually) "just" a series of MPEG Transport Stream (TS) packets placed in UDP/IP datagrams, while video in Chapter 10 is the same TS packets dropped into a Chapter 11 packet type, with some added header and trailer information. But other data types generally carry assumptions that may or may not align with a different standard.

The most prominent example is with time. A Chapter 11 packet has a "Relative Time Counter" in the header (48 bits), an optional time field in the Packet Secondary Header (64 bits), and an Intra-Packet Time Stamp in the payload (for some data types only, either 48 or 64 bits long). The IENA packet only has a single time field at 48 bits in length. Mapping the two packet formats requires understanding how the time is measured in each architecture.

And this is where the significance of the radiation requirements comes in: because the DAUs were COTS products, modifying them to output Chapter 11 packets would require significant software/firmware modifications, and possibly even hardware revisions. Therefore, the conversion had to happen at the recorder, not the DAUs.

Fortunately, there was a very limited number of data types, and despite the DAU vendor being unwilling to cooperate directly, we could establish a mapping between the IENA fields and values and Chapter 11 ones. This mapping is not universal and might not work too well for other applications, but it is sufficient for this specific project.

Parameter Extraction

Once the core data acquisition system, including the format conversion, was completed, the next added task was to implement a scheme to usefully reduce the data rate, so as to accommodate a very limited telemetry downlink channel. As may be expected, the greatest challenge was to define a very flexible control interface, so that the user could specify the parameters to extract and the rate at which to send the extracted samples; also considered, but ultimately rejected as unnecessary for this application, was whether provide averaging or min/max functions to the down-sampled parameters.

The control language was implemented by vendor extensions to IRIG 106 Chapter 9, the Telemetry Attributes Transfer Standard (TMATS). While pretty much universally disliked, the overwhelming advantage of TMATS is that it is widely understood, and tools to manage and manipulate TMATS records are relatively abundant.

The telemetry packets were assembled into a standard Chapter 10 UDP Data Transfer stream and dispatched to the address of the downlink transmitter. In this specific application, the downlink channel is an established radio network, but architecturally it could just as well be an IRIG 106 Chapter 7 Packet Telemetry Downlink implementation.

Vehicle Infrastructure Management

Once the radiation-hardened recorder system was designed, the program started to consider using it for additional tasks.

The first of which is the management of the networks and gateways within the vehicle. In addition to the on-board operational networks (such as the data acquisition networks), there are networks designed to be used by payloads carried by the vehicle. Naturally, the vehicle operators want to be able to isolate and connect those payload (cargo) networks to limit connectivity to various gateway points at various times during the mission, so the recorder (by now renamed the Multi-Function Data Recorder) was selected as the control node through which all other network functions were managed.

This obviously had relatively little to do with "recording", but it reflects the fact that the device hosting the recording software was central to the overall vehicle network, and thus represents a

good site for non-mission-critical *ad hoc* command and control.

As it turns out, on this specific program the implementation of the software to support these additional capabilities was via code written by Ampex. However, on other programs, the recorder was configured to support “containers” such as Docker and LXC.

In many ways, containers are similar to virtual machines, but instead of having a whole operating system and a virtual machine hypervisor, a container uses the same operating system kernel as the supporting system, much reducing the overhead involved: a container running on a particular operating system calls exactly the same library and kernel software as a traditional application would; the only significant difference is that containers use more storage space as they duplicate all the files and libraries needed so as to ensure integrity of the container (all the pieces are present).

It should be noted that supporting a full hypervisor/virtual machine architecture is also perfectly possible and allows additional capabilities such as the use of different operating environments (classically, Windows and Linux, with the former used for analysis and the latter collection and manipulation) and more rigorous cybersecurity boundaries.

This facility allows third parties to develop lightweight, self-contained software packages that will not interfere with with each other or with the main recording application.

Cargo Management

Following on from the concept of controlling the network access points that various payloads may use, it’s an obvious extension to provide health and status interfaces to the various payloads.

This includes both the generic capabilities provided by the vehicle, such as control of cameras and floodlighting, but also passing commands to the various payloads (in both the pressurized and unpressurized compartments) and returning the responses to the origin. In that way, the vehicle can carry “third party” payloads and provide access to them through the MDR interface.

Summary

Traditionally, an instrumentation recorder is perceived as an “endpoint” device: data flows to it, but the only data flows from it are downloading or streaming.

But as the requirements for data acquisition shift inexorably towards networks and DAUs, so does the role of the “recorder”: no longer is it just an endpoint, but now it becomes a central hub.

Naturally, it’s still possible to record network data using a traditional purpose-built recorder, but the very limitations of network recording -- primarily the non-deterministic delivery of packets – means that general-purpose systems with a software-defined recorder are not only viable, but more flexible.

And as the flexibility of the network hub recorder gets recognized, additional functionality inevitably migrates to the recorder. And the use of “packaging” technologies such as containers show one approach that will allow even greater versatility from these formerly “one trick” devices!

Acknowledgements

The author acknowledges the contributions and hard work of Paul Carrion, Peter Chan, Kevin Hudson, and Hal Steger without whom this solution would have remained a hypothetical though experiment, instead of an operational space-going Multi-Function Recorder.