NEXT GENRATION END TO END AVIONICS BUS MONITORING

Kathy Rodittis

Symvionics Inc.

Alan Cooke

Curtiss-Wright Controls Avionics & Electronics

Abstract: With the advent of networked based data acquisition systems comes the opportunity to acquire, transmit and store potentially very large volumes of data. Despite this, and the increased size of the data acquisition networks, the use of tightly integrated hardware, and setup and analysis software enable the FTI engineer to save time and increase productivity.

This paper outlines how the use of innovative bus packetizer technology and the close integration of FTI software can simplify this process. The paper describes how packetizer technology is used to acquire data from avionics buses, and how it packages this data in a format that is optimized for network based systems. The paper further describes how software can simplify the process of configuring avionics bus monitors in addition to automating and optimizing the transport of data from various nodes in the acquisition network for transmission to either network recorders or via a telemetry link.

Keywords: Avionics Bus Monitoring, Bus Packetisers, IADS, GS Works, DAS Studio, Fight Test Instrumentation, Real time data analysis, iNET, XML, XidML, XdefML,

INTRODUCTION

1.1 BUS PACKETIZERS

In this paper, a bus Packetizer is defined as a module that quickly and efficiently processes and repackages messages on an avionics or other asynchronous bus for retransmission over an Ethernet network. The transmitted packets are usually iNET-X packets, an iNET compliant packet format used in the transmission of data by Curtiss-Wright Controls Avionics & Electronics (CWC-AE) equipment.

1.1.1 Key Features of Bus Packetizers

Bus Packetizers are designed to operate with asynchronous avionics buses. Specifically, they are designed to simplify setup and optimise the use of network bandwidth while maintaining low latency. Some important features include

No transmit when empty: The nature of data from an avionics bus means that there may be times when there is no data in a given time interval to be transmitted. The no transmit when empty feature allows for the optimal use of network bandwidth by only transmitting packets over the network when there is data to be transmitted.

Timeout: In order to maximize the use of bandwidth packet sizes must be as close to the maximum transmission unit size of 1514 bytes as possible. However, a strict adherence to this policy may result in unacceptable delays before data is transmitted. To address this issue, bus Packetizers allow the user to set a timeout which specifies the maximum amount of time a packet will be delayed before transmission.

FPGA based architecture: Bus Packetizers feature dedicated FPGA based architectures which offer several advantages versus CPU based systems. These include higher throughput, robust and deterministic operation in addition to greater reliability.

Simplified Setup: A side-effect of the nature of bus Packetizers is that they require very little setup. This allows for greatly simplified software and the ability of users to quickly and easily configure bus monitors and get up and running almost immediately.

Supported buses include ARINC-429, CAN Bus, MIL-STD-1553, TTP, AFDX, RS-232/422/425, IRIG-106 Chapter 4 PCM and raw Ethernet traffic.

1.2 DAS STUDIO

DAS Studio is the latest generation of setup software from CWC-AE and is used to configure and program Data Acquisition Systems, Recorders and Ground Station equipment. The DAS Studio software is fully XidML and XdefML native and has been designed from the ground up to take advantage of modern multi-core processors.

1.2.1 Native Support for XidML and XdefML

XidML is an open, vendor neutral, XML[1] based meta-data standard designed for the Flight Test community [2][3][4][5]. Using XidML, the entire setup of a complete FTI system can be defined including sensors, data acquisition units, signal conditioning cards, avionics buses and entire acquisition networks.

XidML is based on a generic data model of an FTI system. This model is based five key components.

Instruments: An instrument represents the physical hardware that make up a data acquisition system. They include DAUs, sensors, signal conditioning cards, bus monitors and so on. The configuration of an Instrument is described using Settings. Changing the values of Settings changes the behavior of a device, and by extension, the entire acquisition system.

Parameters: Parameters represent what the FTI user is interested in measuring. They encapsulate everything the user needs to know about a

measurement such as the number of bits used to encode the parameter, its format (e.g. Offset Binary) and the engineering units used.

Packages: Packages describe how data is either transmitted or stored to a given medium. Examples of transmission packages include IRIG-106 Chapter 4 PCM, MIL-STD-1553 bus messages and Ethernet packets. Examples of storage packets include data stored on a flashcard or chapter 10 files.

Links: Links are connections between two points in an FTI acquisition network. They can include data links such as connections to avionics buses or sensors, connections to transmitters, programming links or connections to power supplies.

Algorithms: Algorithms describe operations that can be performed on data. Examples include lookup tables, polynomials and Fast Fourier Transforms.

An optional part of the XidML Schema is XdefML. While XidML is be used to define the flight test configuration, XdefML allows vendors and FTI users to define constraints on the data used to configure individual instruments that constitute the data acquisition system. This is achieved by writing an XdefML file for each of the devices that are part of the acquisition system.

Some of the constraints that can be defined using XdefML include:

The data that can be read from a device: Vendors can use XdefML to define what data (i.e. Parameters) can be read from the card in addition to the characteristics of this data such as number of bits and engineering units.

The allowed settings for a device: XdefML can be used to define what options (i.e. Settings) can be used to configure a device. It can also specify what the values for each Setting are in addition to any dependencies between Settings

The number of Channels on a device: The number of input and output channels can be defined in XdefML.

The directionality of a Channel: XdefML is used to define whether a Channel is a producer (e.g. transmitter) or consumer (e.g. sensor input, bus monitor) of data.

The type of data either consumed or produced by a device: The XdefML for a device determines what type of data a Channel on a particular device consumes/produces. For example, a Channel on a device could transmit IRIG-106 Chapter 4 PCM or consume MIL-STD-1553 message.

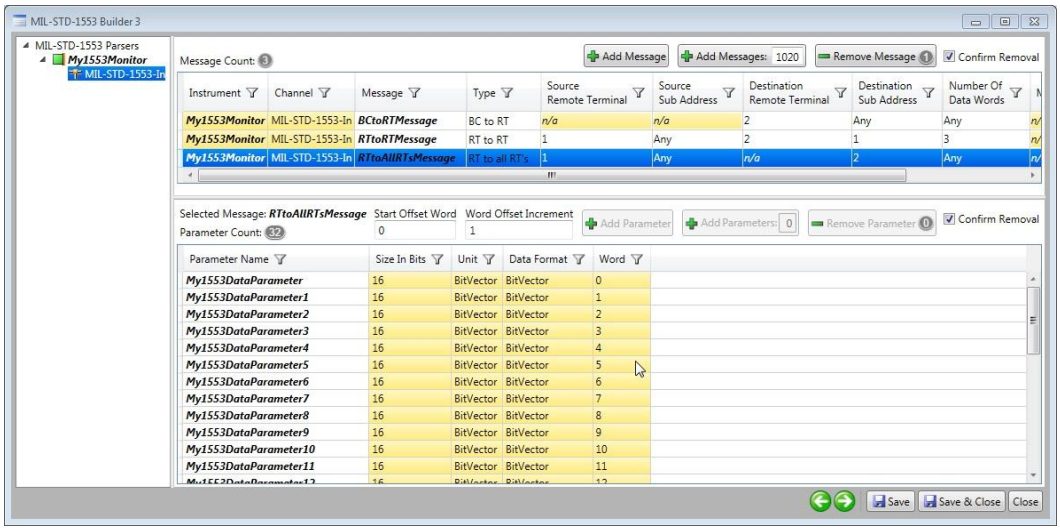
DAS Studio leverages the power of XidML and XdefML to auto-generate its user interface and to validate user data [6]. Users only need to provide an XdefML file for a device for it to be supported in DAS Studio and can include third party devices such as sensors, modules and even entire data acquisition systems.

1.2.2 Extensible Architecture

DAS Studio features a highly extensible architecture which facilitates the rapid integration of value added functionality and tools.

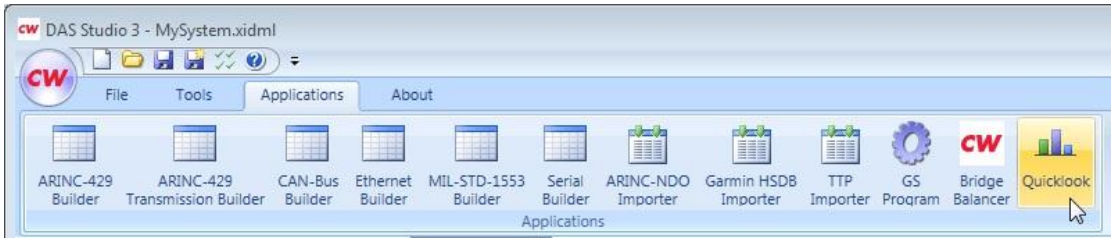
Value added applications: A dedicated plugin framework allows the functionality of DAS Studio to be extended. This mechanism has been used to write importers for proprietary bus formats and wizards to rapidly configure bus monitor modules (see Figure 1).

Figure 1: The MIL-STD-1553 message builder in DAS Studio



Users just need to write a plugin module and drop it into a predefined directory. These plugins can then be called from the Applications section of the main ribbon bar (See Figure 53) of the software or, like GS Works, can be called from context sensitive popup menus and buttons within the application itself.

Figure 2: The Applications menu in DAS Studio



Rapid integration with third party devices: In addition to allowing third party configurations to be defined and stored in XidML, DAS Studio's plugin framework allows the hardware to be programmed as well. This is achieved by writing a plugin module and dropping it into a pre-defined directory. Once the device configuration has been defined in DAS Studio it can then be programmed using the Program option in the Tools menu (See figure

Figure 3: Program option in DAS Studio

3).

1.3 GS WORKS

GS Works is a version of SYMVIONICS' well-known flight test software suite, IADS. Here are some of its features:

- Complete data scrollback for replay and analysis
- Critical analysis that is accomplished in realtime
- Event and test point/manoeuvre marking
- Derived equations added or modified on the fly
- Wide range of display choices
- Automatic configuration from XidML file
- Comprehensive list of data sources including:
    - Acra CF card
    - Acra GTSDEC Decoms
    - Acra Ethernet (IENA and iNetX)
    - Lumistar Decom
    - Chapter 10 Playback
    - Chapter 10 UDP Realtime

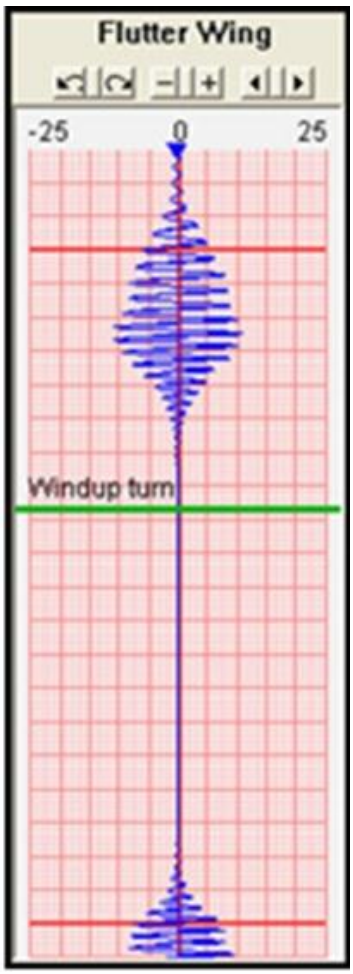


Figure 3: Example display from GS Works

2 PUTTING IT ALL TOGETHER

The combination of bus Packetizer technology, XidML, DAS Studio and GS Works allows FTI users to rapidly acquire and analyze data from avionics buses. This process involves three simple steps

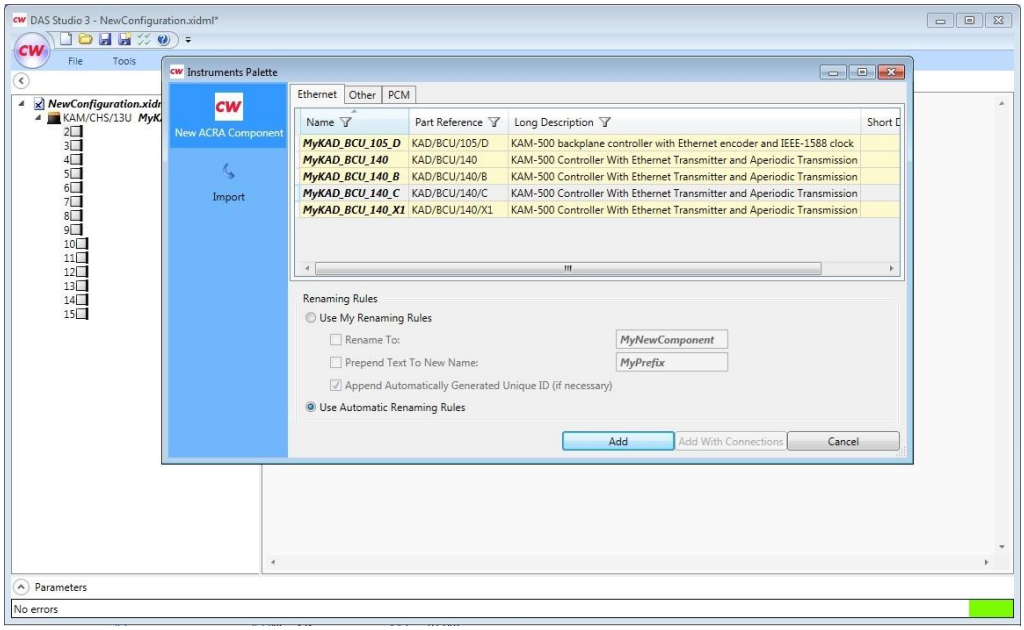
1. System definition
2. Program the system
3. Visualize and analyze data using GS Works

The following outlines in detail how to define a simple system with one ARINC 429 bus Packetizer.

2.1 System Definition

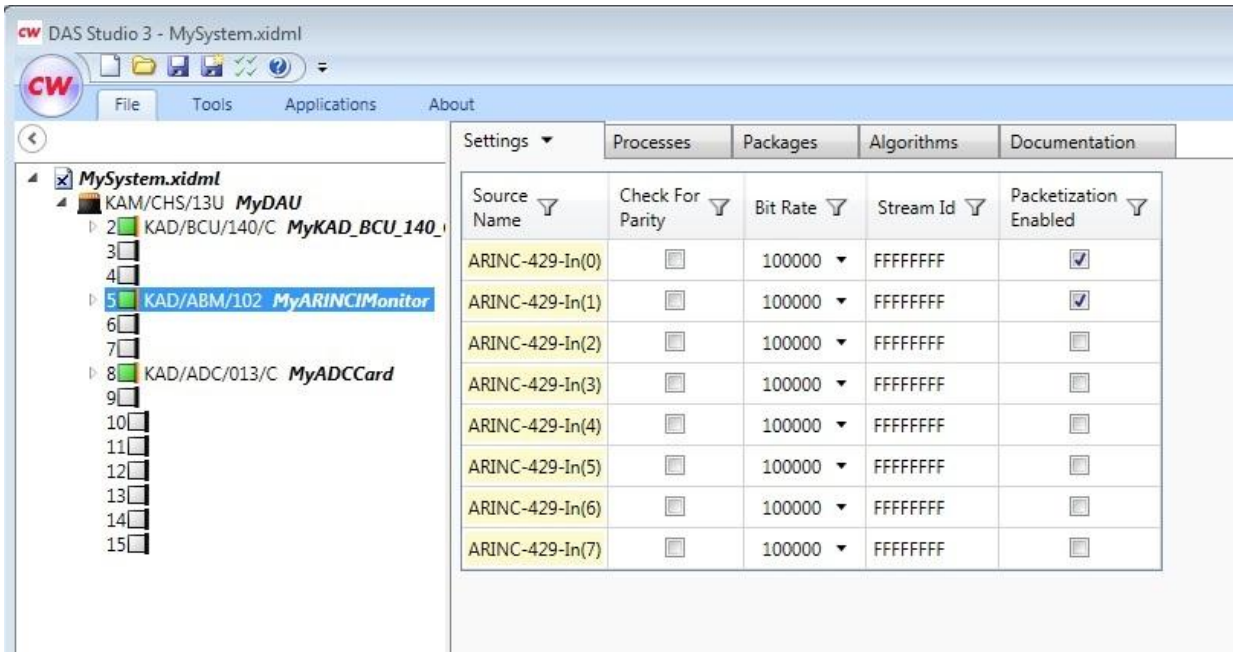
Step 1: This involves defining a system in DAS Studio. The first thing the FTI user needs to do is to use the Instrument Palette to add an ARINC 429 Bus Packetizer module, a simple analog card and an iNET-X transmitter (see Figure 5) to the configuration.

Figure 53: Adding a module in DAS Studio



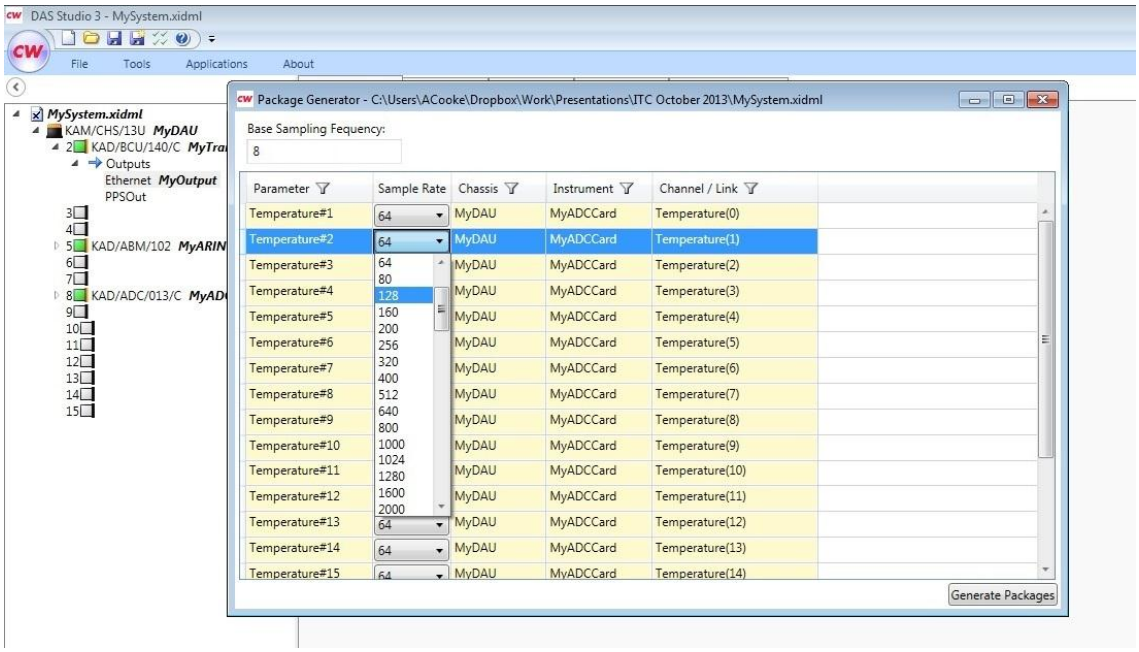
Step 2: Configure the Bus Packetizer module by selecting the Bus Packetizer module in tree view on the left (see Figure 6). The user then chooses which bus to monitor by selecting the Packetization Enabled option for that bus.

Figure 6: Configuring a Bus Packetizer module



Step 3: Configure the analog module by specifying what parameters to transmit in addition to their sample rates (see Figure 7).

Figure 7: Configuring a Bus Packetizer module



The above three steps should take no longer than five minutes.

2.2 Program System

Once the user has defined the system the results are then saved to a XidML file the FTI user selects the Program option in the Tools menu of DAS Studio. The software will automatically figure out which parameters need to be transmitted in addition to how the transmitted Ethernet packets should be constructed. The packetized data will be transmitted in iNET-X format.

2.3 Display and Analyze Data Using GS Works

2.3.1 Launching GS Works: Once the system has been defined and the hardware programmed the FTI user can launch GS Works to display and analyze data coming from the FTI system. This can be done via DAS Studio. The GS Works software will present the user context sensitive options that guide the user on how to use GS Works (see Figure 8).

Figure 8: Launching GS Works from DAS Studio

GS Works will read the associated XidML file to determine what is being transmitted by the data acquisition system in addition to how the data should be interpreted. GS Works has full support for iNET-X packets.

2.3.2 Detecting Bus Packets

GS Works uses a plugin methodology to customize the bus packet parsing. The plugin is invoked automatically based on a packet type id that is found in the XidML

file. The plugin is a Microsoft COM DLL that also registers itself in the Windows registry using the packet type id. In that way a specific piece of code can be invoke for each type of bus packet generated. Although several "stock" plugins are provided with GS Works for bus types such as ARINC 429 and MIL-STD-1553, users can customize the processing by developing their own plugin. This is particularly useful since it does not require a new release of GS Works to support a new or expanded bus type.

2.3.3 Setting up processing in GS Works

Once the correct plugin has been identified, GS Works will instantiate the plugin and give it an opportunity to define all measurements that will appear in GS Works. The plugin has the option of either invoking CreateBasicMeasurement() from the API or CreateDerivedMeasurement() to define the measurements that will later appear in the GS Works configuration file. The plugin can define both "primitive" measurements that can be represented by 32 or 64 bit values or an array of bytes known as a "blob". GS Works will then setup all the sockets and queues necessary for smooth UDP data processing.

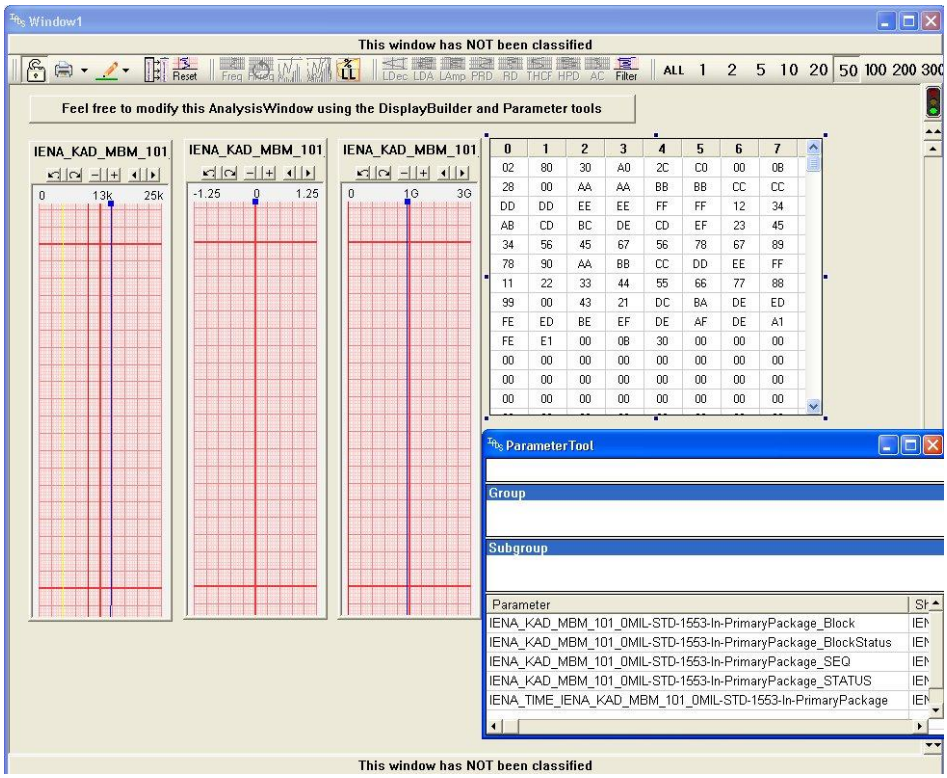
2.3.4 Processing bus packets in GS Works

GS Works will route the payload from the bus packets back to the plugin and pass it in as an array. The plugin is responsible for decoding the payload into the appropriate pre-defined measurement and passing that information along with the timestamp back to GS Works.

2.3.5 Additional "blob" processing in GS Works

Frequently the complete definition for a bus measurement is not known at the time the data is processed either because the information is encapsulated in an non-standard Interface Control Document (ICD) or because it's simply not known. This is when it is best for the plugin to return the entire bus message as an array of bytes called a "blob" in GS Works (see Figure 9).

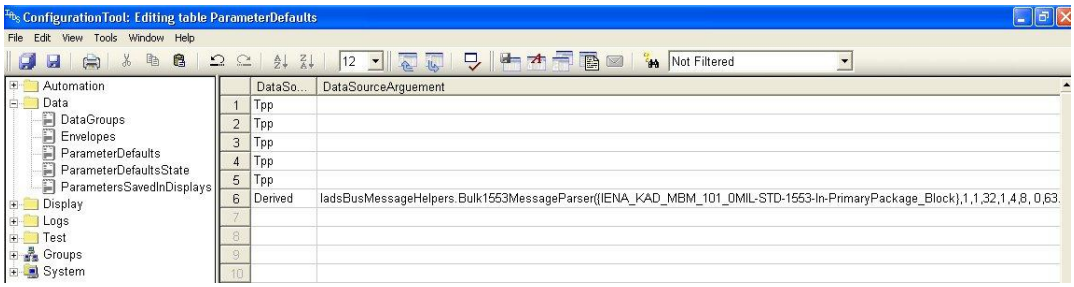
Figure 9: A screenshot of the Packetizer blob viewer in GS Works



From there the end user can invoke the GS Works DECOM() function to decode the piece of information he's interested into a GS Works derived parameter or he can use a special-built GS Works Custom Function to provide additional handling that is specific to the bus type.

A good example of this is the MIL-STD-1553 bus packets. The 1553 plugin will break the incoming packets into separate bus messages, arranged by bus. In this way all the data is processed without any definitions required ahead of time. Then the end user can create a derived measurement and use the GS Works Custom Function called IadsBusMessageHelpers.Bulk1553MessageParser() (see Figure 10) that will identify the measurement by remote terminal, subterminal, transmission type, etc.

Figure 104 A Custom 1553 function in GS Works



3 CONCLUSION

This paper discussed the latest techniques in avionics bus monitoring. Specifically, the paper discussed some of the advantages of Bus Packetizers and how they are designed for the optimal use of network bandwidth and to minimize latency. Additionally, the paper discussed how FTI users can use a combination of Bus Packetizers, XidML, DAS Studio and GS Works to rapidly configure, program and analyze data from one or more avionics buses.

5 REFERENCES

[1] World Wide Web Consortium, http://www.w3.org/

[2] XidML website, http://www.xidml.org/

[3] Alan Cooke, Diarmuid Corry: “XidML: A Global Standard for the Flight Test Community”, ETTC Proceedings, 2004

[4] Diarmuid Corry: “Unleashing the Power of XidML”, ITC Proceedings, 2007

[5] Alan Cooke: “Using XML in an FTI Environment”, ETC Proceedings, 2008

[6] Christian Herbepin, Alan Cooke: “Introduction To XidML-3.0 – An Open XML Standard for Flight Test Instrumentation Description”, ITC Proceedings, 2010