AFDX Training Emulator

User's Guide

v0.1

Jesús Fernández, Héctor Pérez, J. Javier Gutiérrez, and Michael González Harbour

Software Engineering and Real-Time Group

{fsainzj, perezh, gutierjj, mgh}@unican.es

http://www.istr.unican.es/

Introduction

AFDX (Avionics Full Duplex Switched Ethernet) is a standard communication network for avionics based on Ethernet links and special-purpose switches. This software provides an AFDX emulator based on standard Ethernet hardware (cards and switches) which aims at building a low cost AFDX network for training or basic research purposes. It is also possible to integrate the emulator within an ARINC-653 platform.

The AFDX emulator is being developed at "Software Engineering and Real-Time Group" of the University of Cantabria. It is not a finished product, but we share it hopping it can be useful for educational or research purposes. This software is available under the GNU General Public License.

This user guide assumes that you feel comfortable with basic ARINC-664 and ARINC-653 concepts. Complete information about both technologies can be found on the following references [1] [2] [3].

Target platform requirements

- Operating system: MaRTE OS v1.9
- Ethernet network cards with drivers available for your OS
- Network switch with support for traffic prioritization
- (Optional) XtratuM hypervisor if an ARINC-653 platform is required.

Installation

1. Installing Adacore GNAT compiler and MaRTE OS

   Read the INSTALL file in MaRTE OS to perform this step. Before starting the installation of MaRTE OS, you should make sure that your network card is supported by MaRTE OS and the corresponding driver is added to the devices table ("marte-kernel-devices_table.ads"). Furthermore, user should configure MaRTE OS to build the x86 or xtratum architecture depending on the available platform to run the emulator. Further information can be found in http://marte.unican.es/index.htm.

2. Switch configuration

   The AFDX emulator requires configuring the network switch to be as deterministic as possible. To this end, the internal traffic associated with third-party protocols must be disabled: for instance, both the Spanning Tree protocol and the Neighbor Discovery Protocol must be disabled, and the ARP protocol should be statically configured. Furthermore, support for network traffic prioritization is also required to differentiate virtual links with different priorities. Nowadays, most of the high-end switches support traffic prioritization features based on the Class of Service/Quality of Service (CoS/QoS) technologies.

Functionality

The AFDX emulator has basic functionality for transmission and reception of messages at the end systems through the ARINC ports (sampling and queuing), allowing the configuration of Virtual Links and also sub-Virtual Links, which are scheduled according to the traffic regulation rules specified by the standard (Lmax and BAG parameters for Virtual Links, and round-robin policy for sub- Virtual Links in a Virtual Link). Messages that cannot be sent in an Ethernet frame of length Lmax are split in packets.

AFDX Emulator Internals

The AFDX emulator has been implemented in Ada and presents two different APIs: one for the configuration of the network (end systems, Virtual Links, and communication ports); the other one to allow the application tasks to send and receive messages through the emulated AFDX network.

Configuration API

A configuration API is provided in the package **AFDX.Config** where a set of procedures are given in order to declare the information related to the configuration of the network for end systems, Virtual Links and AFDX ports. This information will be statically defined at compilation time in order to obtain a more efficient behavior of the emulator. The definition of the network configuration has to be done in a child package called **AFDX.Config.Definitions**. It is recommended to create a common definition package and share it among all the running end systems, since each node will identify itself with an end system identifier based on the MAC address, so that the appropriate set of Virtual Links and AFDX ports will be automatically created.

In a real AFDX network, end systems and switches contain static tables that we try to replicate with this API. Thus, we have to define each of the running end systems in the network with the procedure ***Add_ES***. Each of these elements is identified with a number and shall be constructed upon a unique MAC and IP addresses.

The properties of the Virtual Links are declared by means of the procedure ***Add_VL***. A number to identify the VL, the BAG and Lmax parameters, the identifier of the sending end system, and a list of receiving end systems identifiers (only one destination is supported for the moment) should be provided. We also have to provide the priority that will be used at the outgoing port of the switch; this method allows avoiding the configuration of priorities at the switch level. The configured priority will be set by the implementation in the Ethernet frame so that the switch can manage it properly. The last parameter to be specified is the size of each sub-VL queue (up to four) that a VL might have. Sub-Virtual Links configured with a size of zero are not used.

Finally, AFDX ports are defined through the procedures ***Add_Transmission_Port*** and ***Add_Reception_Port***, which respectively create a sending AFDX port at the sending end system and a receiving AFDX port at the receiving end system. We have to provide an identifier, the kind of port (sampling or queuing), the VL and sub-VL used for transmission, and the size of the Inbound Buffer at the receiving end system. In order to simplify the mapping of transmission and reception ports, we use the port number to make this mapping. Ports with the same number are linked together.

Communications API

Applications can use the AFDX emulator through the communications API defined in the package **AFDX.System**, which enables accessing the AFDX ports in order to send or receive data depending on whether the end system is a source or a destination. The class AFDX_Port is defined by inheritance from Ada's Root_Stream_Type and ***Read*** and ***Write*** procedures exchange messages with the AFDX ports declared in the configuration package **AFDX.Config.Definitions**. The ***Read*** operation can be configured as blocking or non-blocking by specifying the Mode parameter in the ***Bind*** procedure, which establishes the connection between AFDX_Port objects and the port identifiers defined in the configuration.

A set of additional functions are provided in this API to obtain information about the characteristics of AFDX_Port objects, in particular whether they support Read or Write operations, the kind of port (queuing/sampling), the Read operation mode (blocking/non-blocking), or the freshness of data in a sampling port. The value returned by the Freshness function corresponds to the last message obtained by Read. Time_First is returned for a queuing port, or if Read has never been called for the specific port.

Tests

A simple test is included to help in the development of new systems using the emulator. In this case, the example interconnects two nodes and evaluates the usage of sub-Virtual Links in the AFDX emulator. In particular, Client 1 and Client 2 are allocated in node 1 and each one executes a remote blocking operation in node 2. Two data flows are defined: one to call the remote operation by sending data from node 1 to node 2, and the other one to send the reply from node 2 to node 1. On the one hand, the *Write* data flow uses one Virtual Link composed of two sub-Virtual Links (see Figure 1). On the other hand, the *Read* data flows are implemented by using different Virtual Links. In this example, AFDX frames are restricted to an Lmax value of 98 bytes, and the BAG is configured to 1 ms for all the Virtual Links.

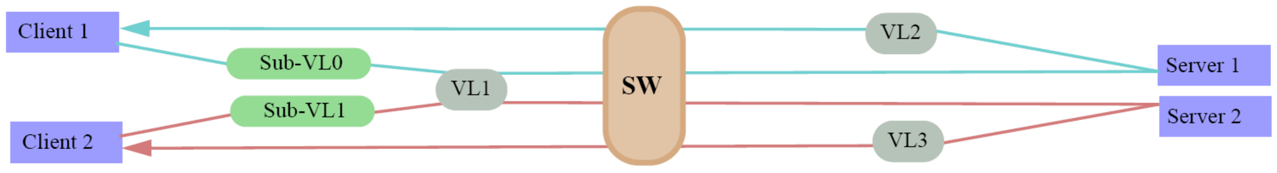

Figure 1. *Example of usage*

For this test, the "*afdx-config-definitions.adb*" files has the following configuration:

	ES 1 (Client)	ES 2 (Server)
MAC	00:02:44:3C:08:21	00:02:44:3B:6A:DE
IP	192.168.85.1	192.168.85.2

VL	BAG	Priority	Lmax	Source ES	Destination ES	Sub VL: Queue size (bytes)
1	1	High	98	1	2	0 : 8192
						1 : 8192
2	1	High	98	2	1	0 : 1024
3	1	Low	98	2	1	0 : 1024

Port	Tx		Rx	
	VL	Sub VL	Mode	Buffer (bytes)
1	1	0	Queueing	8192
2	1	1	Queueing	8192
3	2	0	Queueing	1024
4	3	0	Queueing	1024

Once MaRTE OS is installed, the attached test can be compiled by executing the following command:

```
>> mgnatmake -P proyecto.gpr -O2 -gnat2012 -Imarte_src_dirs
```

The attached test can be run using a bare machine or on top of XtratuM. Further information on this can be found in the website of MaRTE OS. You can even use QEMU to check the functionality of the software without relying on specific hardware. The provided script *run.sh* compiles and executes the test in QEMU.

Known limitations

- Message filtering is not implemented
- Redundancy management is not implemented
- Multicast is not supported
- The AFDX emulator does not comply with some timing requirements specified by the standard such as the typical technological latency in transmission, as this usually relies on hardware.

References

1. Airlines Electronic Engineering Committee, Aeronautical Radio INC. “Avionics Application Software Standard Interface”. ARINC Specification 653-1. March (2006).
2. Airlines Electronic Engineering Committee, Aeronautical Radio INC. “ARINC Specification 664 P7-1: Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet Network”. September 23 (2009).
3. Jesús Fernández, Héctor Pérez, J. Javier Gutiérrez, Michael González Harbour. “AFDX Emulator for an ARINC-Based Training Platform” Reliable Software Technologies – Ada-Europe 2015. Lecture Notes in Computer Science Volume 9111, 2015, pp 212-227.