

CORBA over VMEbus Transport for Software Defined Radios

By Giovanni Middioni, Software Engineer
Selex Communications



To meet the needs of federal, defense and industrial communications customers, Selex Communications selected Motorola VMEbus hardware to develop a new transport protocol that optimizes data transfers within a CORBA architecture for a configurable software radio project.



The Italian firm Selex Communications, a Finmeccanica company, in cooperation with Kleisoft S.r.l., has developed CORBA over VMEbus Transport. This SCA-compliant data transport protocol optimizes VMEbus data transfers within a CORBA architecture based on ORBexpress middleware from OIS and the VxWorks real-time operating system from Wind River. The performance of this solution guarantees a high data-rate execution of software communications architecture waveforms on a VMEbus platform.

The US Department of Defense Joint Program Office, in cooperation with a consortium of military/commercial telecom companies, has defined the Software Communications Architecture Specification (SCAS) within the Joint Tactical Radio System (JTRS) international program.

The SCAS (release 3.0, dated August 2004) defines the basic requirements for designing configurable software radios without referring to a particular hardware implementation.

Standards-based performance

The Software Radio prototype from Selex Communications has been implemented on a hardware platform that consists of a VME64X chassis with a number of 6U commercial-off-the-shelf single-board computers (SBC) inserted into its backplane. Each SBC has a general purpose processor on which the SCA-defined operating environment and the software pertaining to a generic software communications architecture (SCA) application is executed. The operating environment is shown in Figure 1.

Selex Communications selected Motorola MVME5100 single-board computers for this project.

This decision was based on the excellent performance/price ratio and the performance and functionality of these boards, which has been demonstrated in a number of applications.

Also, the MVME5100 allows customers to implement a multiprocessing system distributed on the VMEbus very quickly – as confirmed by Selex Communications' experience.



SELEX
Communications

a Finmeccanica Company

Selex Communications has developed a proprietary transport protocol, called CORBA over VMEbus Transport, which is capable of optimizing the transmission of CORBA data using ORBexpress on a physical VMEbus platform. This new transport protocol has been developed with the goal of achieving the highest packet-transmission data rate possible on a VMEbus system.

Need for a new transport protocol

The software communications architecture demands that the overall operation of a software radio must be hierarchically controlled by a domain manager. This must be capable of controlling and managing the lifecycle of the applications in use and therefore capable of managing all of the waveforms that the user could implement.

The software relating to each application consists of independent logic units called resources. These resources are named devices if their specific function is to “enclose” hardware components, and thereby build a software proxy of a physical device.

From an implementation point of view these resources are objects, in the sense of the object oriented methodology, i.e., class instances whose interfaces must necessarily include the application programming interfaces (APIs) defined by the software communications architecture. It could be said that the software communications architecture defines a framework of correlated interfaces.

The dynamic operating mode of the system implies an exchange of messages between these objects: this is achieved by invoking the relative methods, i.e., the functionalities provided by each object, as shown in Figure 2.

The software objects, which are executed on the SBCs, communicate with one another using a distributed processing architecture (middleware) named CORBA, which acts as a logic bus connecting all units.

Common Object Request Broker Architecture (CORBA) provides a common framework for object oriented applications based on widely available interface specifications. The JTRS has selected CORBA as the architecture to implement the information-forwarding in an SCA-compliant software radio system.

In CORBA architecture, the “engine” of all interactions is an ORB (Object Request Broker). All ORBs communicate with each other to allow a client process to invoke the CORBA objects, called Servants, to be executed on a server.

The protocol used by the ORBs to exchange messages is named General Inter-Orb Protocol (GIOP); it has to rely on specific protocols depending on the transport method used by the virtual entities to communicate with each other.

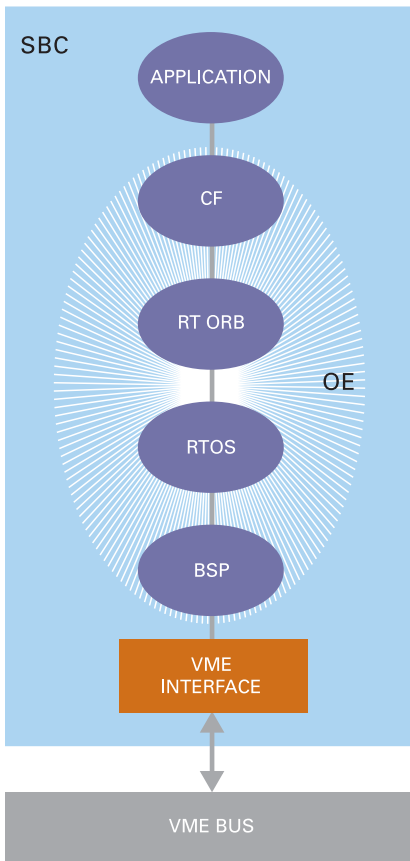
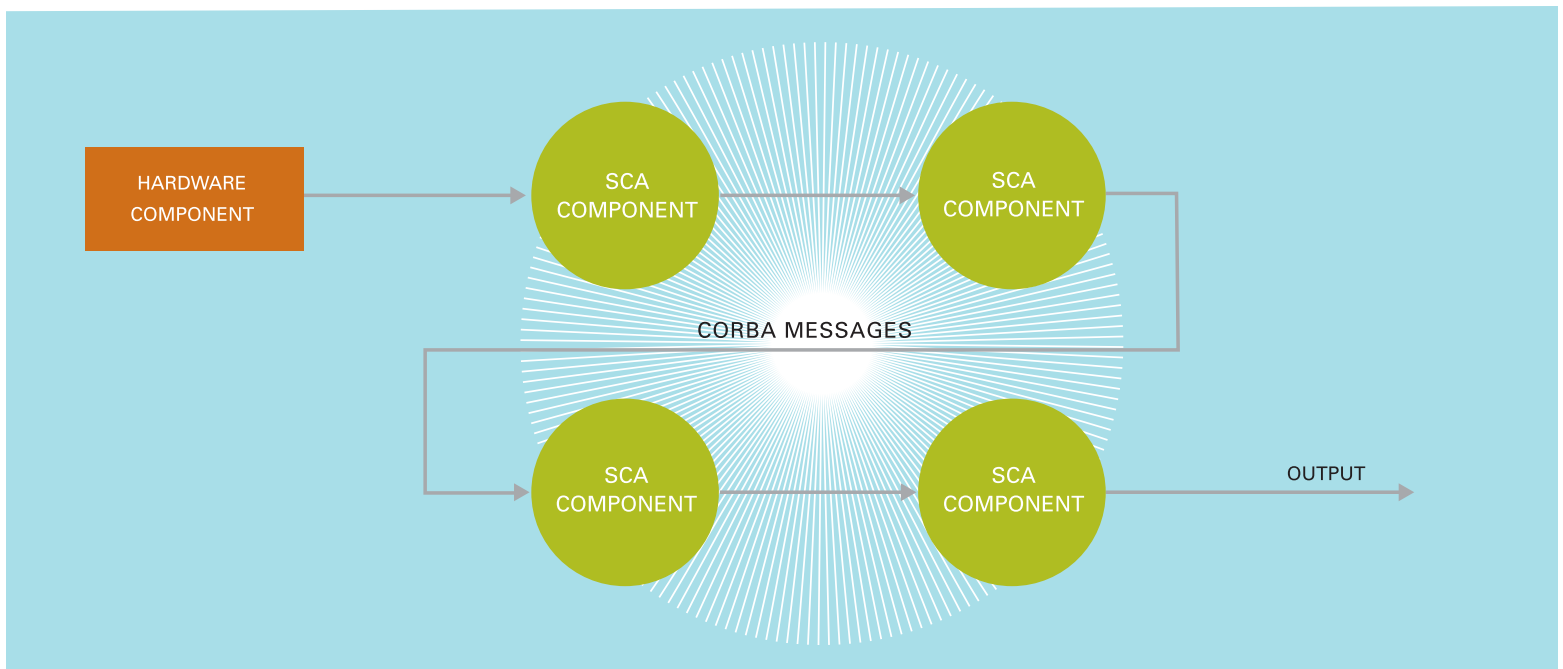


FIGURE 1: The operating environment consists of:

1. The core framework (a SC proprietary implementation)
2. CORBA middleware (ORBexpress from OIS)
3. Real-time operating system (VxWorks and board support package from Wind River)

The most widespread protocol is the Internet Inter-Orb Protocol (IIOP), based on the TCP/IP networking protocol used on the Internet. Benchmark tests have shown that the performance of existing transport methods does not fulfill the requirements for a real-time transport based on the VMEbus. CORBA allows for a new transport solution to be plugged-in if the latter provides the upper layer, represented by the ORB, with a given number of predefined APIs. Also, this new transport must interface with the layer below, which consists of drivers that can utilize the physical layer in order to implement the message transportation method.

FIGURE 2: Structure of a generic software communications architecture application



It is important to note that the CORBA over VMEbus Transport protocol can also be used outside the software radio environment, because it is a transport protocol that can be used by any generic software application using CORBA as a distributed processing architecture.

Technical characteristics of the CORBA over VMEbus Transport Protocol

The CORBA over VMEbus Transport has been developed on the following platform:

- A VME64X chassis with a number of Motorola MVME5100 boards featuring a general purpose PowerPC® processor
- VxWorks 5.5 RTOS
- ORBexpress RT 2.6.1
- The SNIFF+ PRO 4.1 application development environment
- C/C++ gnu compiler 2.9.6.

The addressing domain of the CORBA over VMEbus Transport has been deliberately kept limited to a VMEbus backplane environment which allows a maximum of 21 boards.

The format of the endpoint string has the form:

vme://<CPU number>:<Port number>

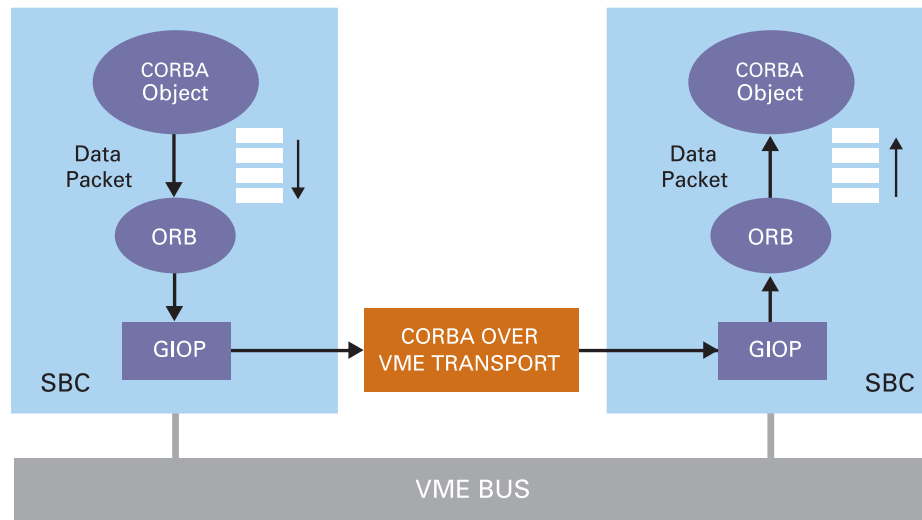
where <CPU number> is the number (any value between 0 and 20) assigned to a given board and <Port number> is a value between 1 and 65535 allowing the user to locate any given process being executed on the board.

The interfaces between an ORB and the available transports (one of which is the CORBA over VMEbus Transport) use the GIOP layer. The function which the GIOP expects to be executed by the transport layer below is a reliable data packet transmission between all CORBA entities.

The transport provides point-to-point connections between client/server processes with a confirmation signal for delivery of the messages for each connection, as shown in Figure 3.

The functional architecture of the transport is based on the protocol stratification required by the International Standards Organisation (ISO/OSI) model. Specifically, the transport is built up by the VME-IOP, VME-Transport, VME-Mac and VME-Physical layers which are highlighted in Figure 4.

FIGURE 3: Data-packet transfer by means of the CORBA over VMEbus Transport over the VMEbus



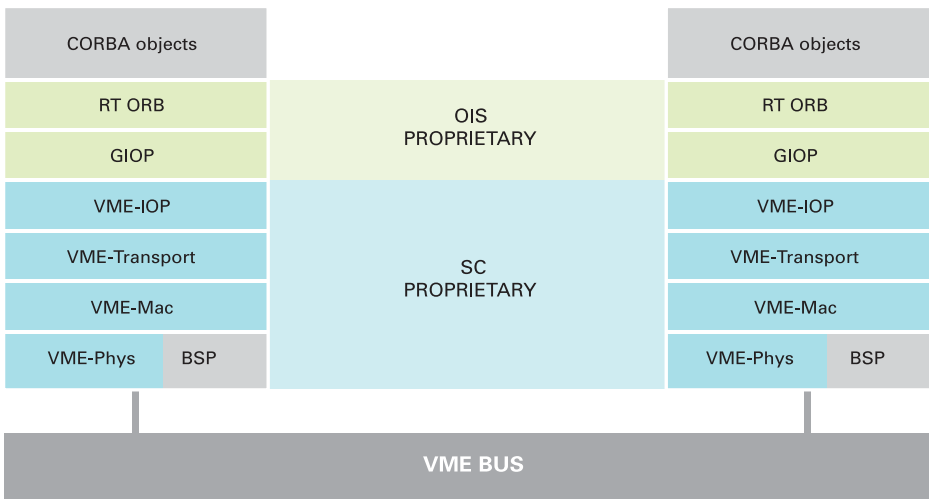


FIGURE 4: Protocol stack for the CORBA over VME Transport

The VME-Physical layer is a component of the operating system's board support package and uses direct memory access (DMA) transfers with multiplexed block transfer (MBLT) write-cycles to perform data transport on the VMEbus. The VME-Mac layer transfers packets of bytes from one board to another using the VMEbus and the <CPU number> information included in the endpoint string.

The VME-Transport layer uses the services offered by the underlying VME-Mac layer in order to implement connection-oriented communications between processes being executed on a single board or on different boards. The VME-Transport layer allows a generic pair of applications, each provided with its endpoint string, to exchange data packets after a point-to-point connection between them has been established.

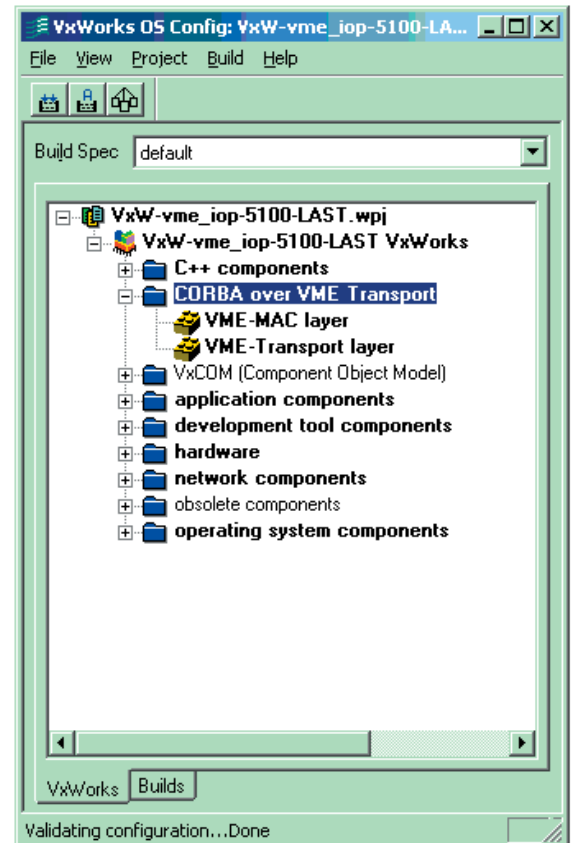
Both the VME-Mac and the VME-Transport layer have been designed as add-on components for the VxWorks operating system. It is therefore possible – using the VxWorks configuration tool that is integrated in the Sniff+ PRO project facility – to enable/disable and initialize all configurable parameters of both components, as shown in Figure 5.

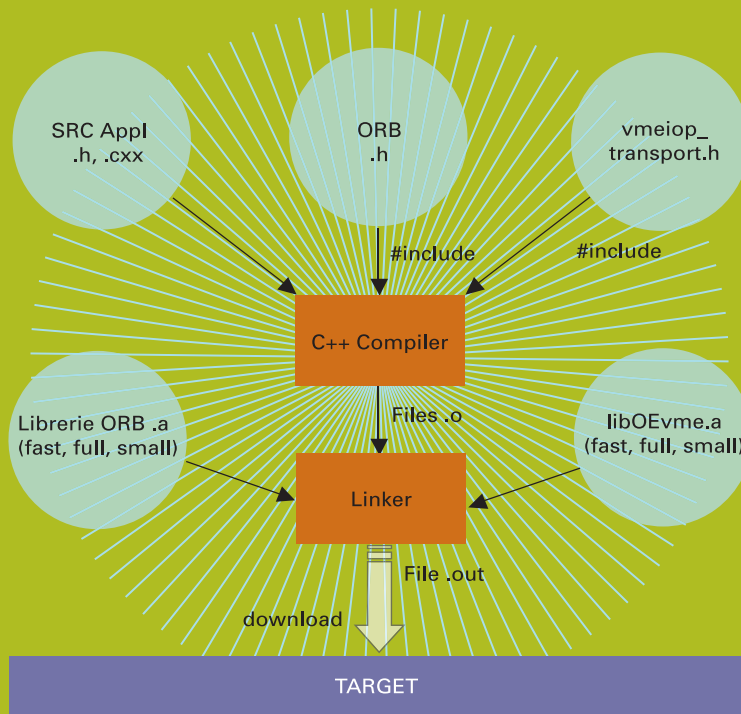
The VME-IOP layer implements all ORBexpress abstract classes pertaining to the interface between the GIOP and the transport itself, so that the transport can be used to communicate between ORBs. The VME-IOP layer can be downloaded as an application project (libOEvme.out) or as a library that can be linked up statically with all CORBA applications (libOEvme.a).

It is important to note that the CORBA over VMEbus Transport protocol can also be used outside the software radio environment, because it is a transport protocol that can be used by any generic software application using CORBA as a distributed processing architecture.

Thanks to its layered nature, this transport can also be used without CORBA because the three lowest layers (VME-Physical, VME-Mac and VME-Transport) enable communication between software entities after point-to-point connections on a single board or between different boards have been established. This is possible because it is the responsibility of the VME-IOP layer to allow the use of the underlying transportation infrastructure through CORBA.

FIGURE 5: A screenshot showing Wind River's VxWorks operating system configurator





Performance of CORBA over VMEbus Transport

The hardware used in this example is a VMEbus chassis with two Motorola MVME5100 SBCs inserted into its backplane. The Interface Definition Language (IDL) used by all CORBA client/server processes is:

```
/* This type is a CORBA unbounded sequence of octets. */
typedef sequence <octet> OctetSequence;
```

```
interface TransmitPacketPhys {
    oneway void pushPacket(in OctetSequence payload);
};
```

IDL contains a definition of a one-way method called `pushPacket()` which, when executed by the client, causes the transfer of an array of bytes to the server.

The `pushPacket()` method is defined by the `PacketBB` (API Supplement to the SCAS, Appendix C, Generic Packet Building Block Service Definition); it is used for real-time data transfers between two generic SCA components. Both the client and server processes are implemented by means of a static linking of the relevant ORB libraries (`libOErORB.a`), the TCP transports (`libOEtcp.a`) and VME (`libOEvme.a`), as shown in Figure 6.

FIGURE 6: Building-up transport solutions for client/server functions

Client/Server processes running on two different boards		
User payload (Byte)	TX side delay (µs)	End-to-End delay (µs)
10000	350	670
8192	300	550
4096	180	335
1024	105	190
100	80	150

TABLE 1: Measured delays (transmission times) of the CORBA over VMEbus Transport when transferring data between two MVME5100 boards

Client/Server processes running on the same board		
User payload (Byte)	TX side delay (µs)	End-to-End delay (µs)
10000	295	440
8192	265	380
4096	165	240
1024	90	160
100	75	135

TABLE 2: Measured delays of the CORBA over VMEbus Transport for data transfers on a single MVME5100 SBC

In order to evaluate the performance of the new transport, the time taken to execute the CORBA method has been measured when using the new transport for the physical data transmission over the VMEbus.

The measures have been obtained using a VME Bus Analyzer and the `oe_Timer` class, available in ORBexpress, which allows the determination of the average of several measured values. Table 1 shows the measured times when invoking the `pushPacket()` method with different payload sizes, and executing the client and the server on two different MVME5100 SBCs.

The values contained in the second column of Table 1 represent the average time required for executing the one-way `pushPacket()` method, i.e. the time it takes the packet to leave the transmitting entity. Also the total transmission time (end-to-end) has been measured under the same payload conditions. This is the time elapsed after invoking the `pushPacket()` method on the transmitting CPU and the execution of that method on the receiving CPU. The third column in Table 1 shows all the measured end-to-end delays.

The total transfer time of a packet has been determined by means of a VMETRO VME Bus Analyzer board. An instant before invoking the `pushPacket()` method, the client process executes a write operation into a register of its own VME-interface chip. This operation does not influence the measurement because it takes only a few microseconds and in addition it is visible through the Bus Analyzer.

A similar procedure takes place when executing the `pushPacket()` method on the server side. Using the Bus Analyzer it is possible to measure the time interval elapsed between the two write operations on the VMEbus, which gives the total data-packet transmission time. Figure 7 highlights the high linearity of all measured delays: the dots refer to measured values and the straight line represents the best fit (trend line).

In order to determine the transfer times when the client and server process are executed on a single board, the whole series of measurements has been repeated. Table 2 shows the measured time values.

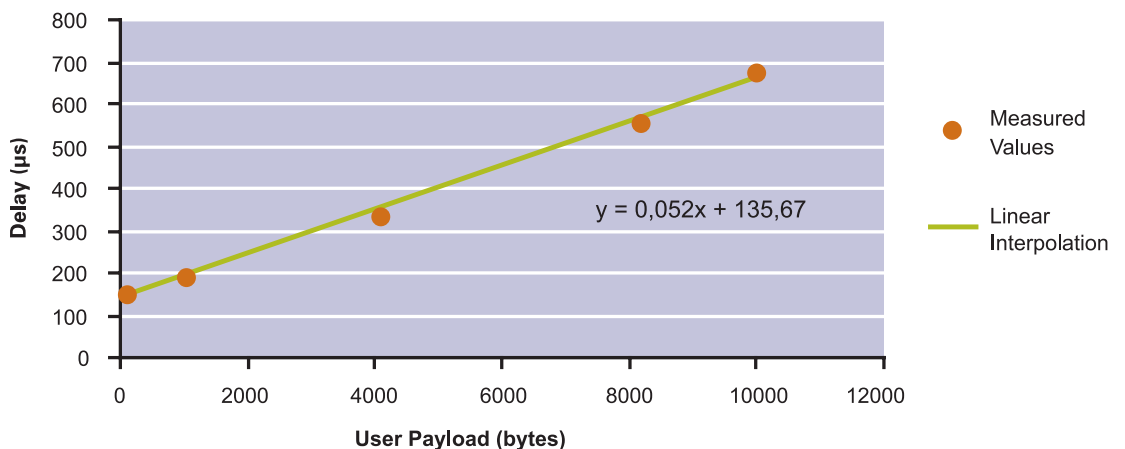


FIGURE 7: End-to-end delay with Client/Server processes running on two different MVME5100 boards

As both the client and the server process have their own ORB, any communication between them takes place through a loop-back made possible by the VMEbus transport, thus avoiding the necessity of a data transmission over the physical bus, as shown in Figure 8.

To sum up, Figure 9 shows a screenshot of the BusView program which displays all the data captured by the VMETRO Bus Analyzer. In the example used, the data refers to the transmission of a CORBA packet (lines 3 - 27) with a user payload of 100 bytes (with value 0xAA) and preceded by the headers of the (moving backwards) GIOP, VME-Transport and VME-Mac layers plus a certain amount of padding bytes.

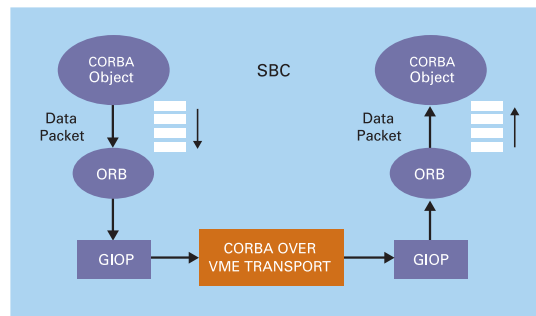


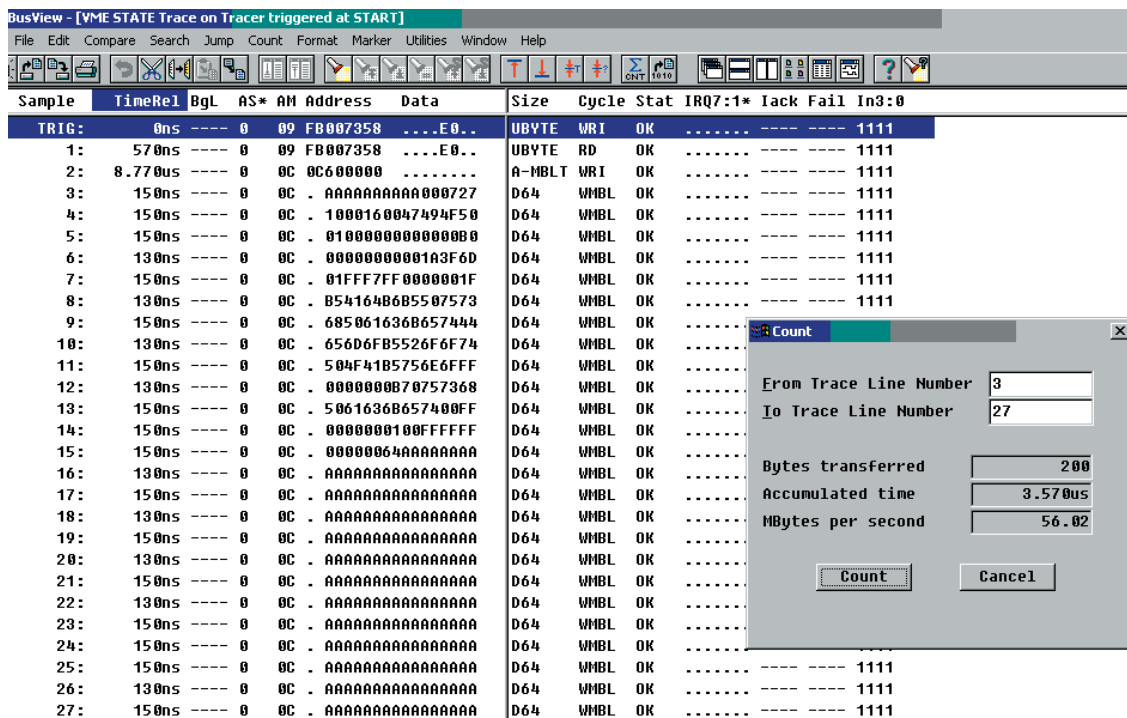
FIGURE 8: Data packet transfer on a single board when using the CORBA over VMEbus Transport

ABOUT THE AUTHOR

Giovanni Middioni has been a software engineer with Selex Communications since 1999. He has been involved in software radio development since 2003. Giovanni gained a Telecommunications Engineering degree from the University of Rome "La Sapienza" in 1998.

The total package of transferred data is 200 bytes with a transmission time of 3.57µs, therefore achieving a data rate of 56Mbytes/s. This demonstrates that the CORBA over VMEbus Transport protocol offers the high-data-rate execution of waveforms in an SCA environment through a fast data-packet transmission over the physical VMEbus and utilizing CORBA.

FIGURE 9: Screenshot of the BusView program which shows data captured by the VMETRO bus analyzer



Motorola, Inc., Embedded Communications Computing
2900 S. Diablo Way, Tempe, AZ 85282 U.S.A. www.motorola.com/computing

MOTOROLA and the Stylized M Logo are registered in the US Patent & Trademark Office and may be registered in certain other jurisdictions. PowerPC is a trademark of IBM Corp. and used under license. All other product or service names are the property of their respective owners. Published with the permission of Selex Communications. © Motorola, Inc. 2005.