ESSOR Architecture Description Documents Set

DSP AEP and IDL Profile Description Document

Ref.: CA-PRA-SAS-63996052-549-vAB-UC

Composed of: 25 pages.

Prepared by ESSOR Industries: a4ESSOR, BITTIUM WIRELESS Ltd., INDRA SISTEMAS S.A. LEONARDO S.p.A., RADMOR S.A., THALES SIX GTS FRANCE S.A.S.

Terms and conditions

The distribution of this specification is unlimited.

A free, permanent and irrevocable right to use the information contained in the specification is granted for whatever purpose by ESSOR Participating States and ESSOR Industries, providing usage of the specification as an input is explicitly credited in any work product using the terminology “ESSOR SDR Architecture”. Distribution to the public of any derivate specification that would have been established without having given a possibility for ESSOR Participating States and ESSOR Industries to get involved in elaboration of the said derivate specification is subordinate to prior written agreement from ESSOR Participating States and ESSOR Industries.

In case an extract of this document is distributed, the present Terms and Conditions have to be included into that extract.

No patent or patent claim is attached to the content of this specification.

Release notes

The ESSOR SDR Architecture is one of the strategic objectives of the ESSOR programme. The intention is to make it available as a SCA-based contribution to international SDR Standards. As part of Release 1 of the ESSOR Architecture, this specification reflects the revision of ESSOR Architecture that has been approved end of year 2012 to serve as basis for implementation on the national platforms of the ESSOR Participating States. It is therefore incorporating no implementation feedback from any later national implementation, and no usage feedback from the porting of the HDR WF on platforms.

ESSOR Main Subcontractors

The ESSOR programme has been established under the umbrella of the European Defence Agency (EDA) and the ESSOR OC1 phase is sponsored by the governments of Finland, France, Italy, Poland, Spain and awarded by the Organisation Conjointe de Coopération en matière d'ARmement (OCCAR) to the dedicated joint venture Alliance for ESSOR (a4ESSOR S.A.S.) in charge of managing the industrial consortium composed of the following respective National Champions:

Name of Main Subcontractors	*Short Name*	*Country*
BITTIUM WIRELESS Ltd.	BITTIUM	FI
INDRA SISTEMAS S.A.	INDRA	ES
LEONARDO S.p.A.	LEO	IT
RADMOR S.A.	RAD	PL
THALES SIX GTS FRANCE S.A.S.	THALES	FR

REVISIONS

VERSION HISTORY:

Version	Date	Lead Author	Notes	Status
AA	27/11/2018	LEONARDO A. Di Rocco	Reflects the version approved as outcome of ESSOR Architecture definition work end of year 2010.	First Release
AB	18/01/2019	LEONARDO A. Di Rocco	Update after review.	Second Release

TABLE OF CONTENTS

1 SCOPE 7
1.1 IDENTIFICATION 7
1.2 OVERVIEW OF ESSOR ARCHITECTURE 7
1.2.1 Origin 7
1.2.2 Structure of ESSOR Architecture 8
1.3 DOCUMENT OVERVIEW 11
2 REFERENCED DOCUMENTS 12
2.1 ESSOR ARCHITECTURE INTRODUCTORY DOCUMENTS SET 12
2.2 OTHERS DOCUMENTS 12
3 IDL PROFILE FOR DSP AND FPGA PE 13
3.1 DSP CORBA PROFILE 13
3.2 CORBA FOR FPGA ENVIRONMENTS 13
3.3 COMMON IDL PROFILE FOR DSP AND FPGA PROCESSING ELEMENTS 17
4 IDL PROFILE FOR DSP AND FPGA PE 20
4.1 DSP CORBA PROFILE 20
5 ACRONYMS 24

LIST OF FIGURES

Figure 1 - SDR composition and structure 9
Figure 2 - FPGA basic environment for CORBA support 14

LIST OF TABLES

Table 1 - ESSOR Architecture Introductory Documents 12
Table 2 – Other Documents 12
Table 3 – ESSOR Common IDL profile – Supported features 18
Table 4 – ESSOR Common IDL profile – Optional features 19
Table 5 - DSP AEP 23
Table 6 - Listing of acronyms 24

1 SCOPE

1.1 Identification

The present document specifies the following two items:

i) an Application Environment Profile (AEP) for DSP Processing Elements, applicable to both CORBA and Non-CORBA Execution Environments

ii) a common IDL profile for DSP and FPGA Processing Elements.

As part of Release 1 of the ESSOR Architecture [1], it reflects the revision of ESSOR Architecture that has been approved end of year 2012 to serve as basis for implementation on top of the national platforms of the ESSOR Participating States. It is therefore incorporating no implementation feedback from any national implementation, and no usage feedback from the porting of the HDR WF on top of the platforms.

1.2 Overview of ESSOR Architecture

1.2.1 Origin

The goals of developing a Software Defined Radio (SDR) architecture is to facilitate waveform portability between heterogeneous Software Defined Radio hardware platforms and to foster radio re-configurability of a large set of waveforms. In front of these goals, the JTRS has initiated the path, publishing the Software Communications Architecture (SCA) [4], recognized as a de-facto standard by the SDR community. The ESSOR Architecture is an SDR architecture which extends the SCA on the following areas:

- Definition of the Operating Environments (OE) for DSP & FPGA processors providing scalable architectural approaches between Modem Hardware Abstraction Layer (MHAL) and Common Object Request Broker Architecture (CORBA) based solutions [5].
- Definition of extensions and additions to the already published JTRS Radio Devices (RD) and Radio Services (RS) Application Programming Interfaces (API).

1.2.2 Structure of ESSOR Architecture

The ESSOR architecture is a SDR architectural framework, aiming to establish a “Reference Architecture” scalable to different SDR platform classes and suitable for different implementations.

The SDR Platform is defined as the aggregation of Software (SW) and Hardware (HW) Platform, where the SW Platform is a particular implementation of the ESSOR Architecture, scaled for specific needs, that rely on the HW platform.

HW Platform are typically characterised by different constraints on size, weight and power (SWAP), processing capacity, RF frontend capability (e.g. simplex/duplex), mainly determined by the operational usage of the Radio Set. Depending on these characteristics, SDR Platforms can be categorised in classes (typically: handheld, manpack, vehicular, naval/fixed and airborne).

Figure 1 shows an overview of the composition and the structure of an SDR, in which the SDR Platform (PTF) provides capabilities to the instantiated waveforms by means of APIs.

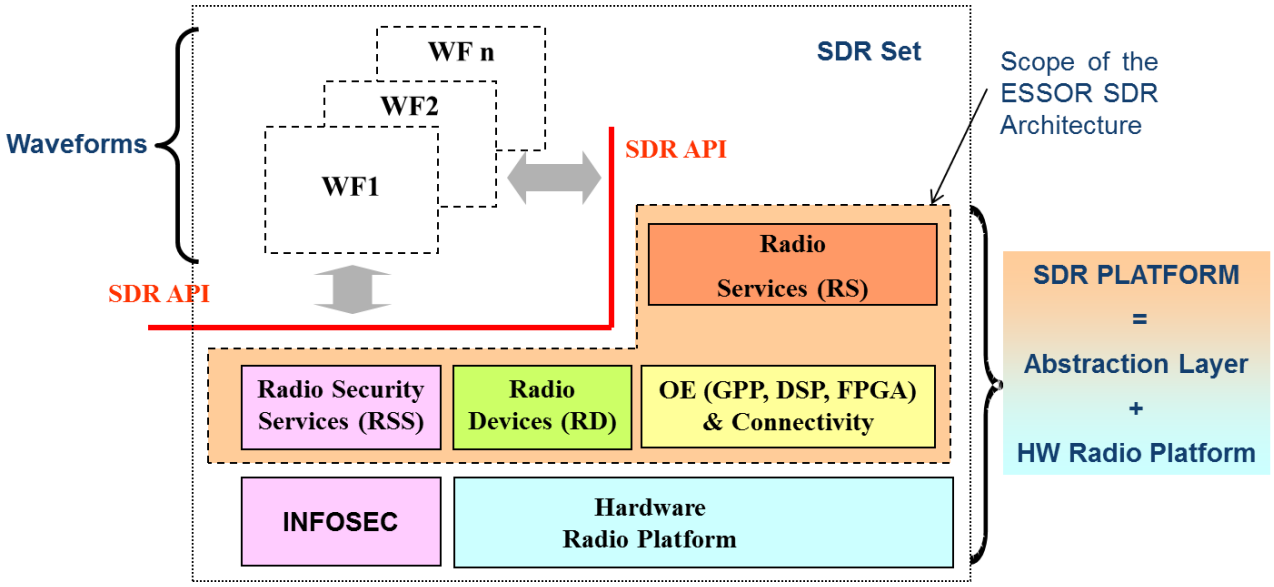

Figure 1 - SDR composition and structure

The definition of the ESSOR Architecture answers to the following motivations and objectives:

- The identification and definition of a full set of capabilities required for the ESSOR Architecture, that are accessible through the defined set of APIs,
- The facilitation of the porting of new waveforms (as the ESSOR HDR waveform) and identified legacy waveforms, on different classes of SDR platforms,
- To be compatible with the SCA 2.2.2 specification,
- To maintain compatibility with JTRS SDR architecture at the maximum level possible.

The ESSOR Architecture is composed of the following areas:

- **Core Framework (CF):** entities implementing JTRS SCA 2.2.2 CF interfaces, to which ESSOR Architecture is compliant, with the addition of some minor modifications, clarifications and extensions.

- **Operating Environment (OE) for GPP, DSP and FPGA:** entities related to Execution Environments (for code loading and execution) and Connectivity for GPP, DSP and FPGA, they are important foundations of the ESSOR Architecture. Two main categories of OE exist, depending on the connectivity solution adopted between the two specified by the ESSOR Architecture: CORBA and ESSOR MHAL. In some cases, the same component is applicable to both execution environments. For the identified OE components, ESSOR Architecture provides rationale, recommendations, specifications and API definition, when applicable.

- **Radio Devices (RD):** entities that provide an abstraction of the HW modules of an SDR. Radio Devices offer a high-level SW interface (API) to the other components of an SDR (e.g. WF application or Radio Services) that need to access HW modules. For RD, ESSOR Architecture provides detailed API definition, relying on published JTRS API specifications when possible, with some modifications/clarifications, and extending them with ESSOR additional functionalities, such as the ESSOR Transceiver API [2].

- **Radio Services (RS):** entities that provide software functionalities useful for the waveform application. RS are related to the operations of the waveform, its control and monitoring and the download of its files and the properties configuration. Similarly to Radio Devices, ESSOR Architecture provides detailed RS API definition, relying on published JTRS API specifications when possible and extending them with ESSOR additional functionalities. [3].

- **Radio Security Services (RSS):** entities that provide security functionalities in conformance with the security objectives of ESSOR. Presentation of this aspect of the ESSOR Architecture is not in the scope of this document.

The ESSOR Architecture is providing the following benefits:

- **Flexibility:** provides the capability to adapt the implementation of the ESSOR Architecture to different types of hardware architectures, since even for different platforms of the same class, underlying hardware can be very heterogeneous.
- **Scalability:** provides the capacity to select APIs and features adapted to the class of the platform which implements the ESSOR Architecture (tactical, naval, …).

- **WF portability:** provides the capability to minimize the effort required for porting an ESSOR-compliant WF application from an ESSOR-compliant PTF to another one.

Further detailed description of ESSOR architecture is [1].

1.3 Document overview

The document has the following structure:

- Chapter 1 “Scope”: describes the scope and provides an overview.
- Chapter 2 “Referenced documents”: reports referenced documents.
- Chapter 3 “IDL profile for DSP and FPGA PE”: contains a detailed descriptions of a common IDL profile for both DSP and FPGA Pes.
- Chapter 4 “EAP for DSP PE”: contains a detailed descriptions of the AEP for DSP Processing Elements.
- Chapter 5 ”Acronyms”: contains Acronyms.

2 REFERENCED DOCUMENTS

2.1 ESSOR Architecture Introductory Documents set

Nr.	Document Id	Version	Document Title
[1]	CA-PRA-SAS-63996053-549-VAB-UC	Issue AB	ESSOR Architecture Introductory Document
[2]	CA-PRA-SAS- 63996048-549-VAB-UC	Issue AB	Radio Devices API Description Document
[3]	CA-PRA-SAS- 63996050-549-VAB-UC	Issue AB	Radio Services API Description Document

Table 1 - ESSOR Architecture Introductory Documents

2.2 Others Documents

Nr.	Document Id	Version	Document Title
[4]	JTRS-5000SCA	V 2.2.2 15 May, 2006	Software Communications Architecture Specification
[5]	OMG (Object Management Group)	version 3.0.3 August 1997	The Common Object Request Broker: Architecture and Specification.
[6]	IEEE Std 1003.1 (IEEE Standard for Information Technology)	2004 Edition	IEEE Standard for Information Technology - Portable Operating System Interface (POSIX(R))

Table 2 – Other Documents

3 IDL PROFILE FOR DSP AND FPGA PE

3.1 DSP CORBA Profile

It is recommended, to adopt the “Common IDL profile for DSP and FPGA processing elements” for the specification of waveform-components interfaces into DSP Processing Elements. The adoption of a “Common IDL profile for DSP and FPGA processing elements” in DSP PEs improves the portability of a WF component from a DSP to an FPGA and vice versa.

A WF component developed for a DSP environment, and whose IDL interfaces are designed according to the Common IDL Profile, can be ported to an FPGA target. In the same way, starting from the same IDL interfaces and keeping the behaviour of the component, it can be easily ported from FPGA to a DSP.

3.2 CORBA for FPGA Environments

FPGA Execution Environments are described in this paragraph, together with the services needed by CORBA infrastructure on such FPGA Execution Environments. FPGAs are concurrent processing elements not characterized by the presence of an Operating System providing basic services to Waveform components, such as into GPPs and DSPs. FPGAs functionalities are performed by reconfigurable hardware structures, having “true parallel” operations execution: the intrinsic parallelism of such devices in fact allows functionalities to be performed simultaneously.

CORBA on FPGAs can be essentially achieved in two ways:

- by using a standard SCA Software-ORB approach, i.e. a GPP processor core embedded in the FPGA.
- by using a Hardware-ORB approach, i.e. an ad-hoc IP-core, implemented in the common resources present in FPGAs, that carries out the functionalities of an ORB. IP vendors marketing HW ORBs can be considered as other common FPGA IPs commercially available, such as FIFOs, Ethernet, MAC, etc.

In the following, only the second approach is considered because the Hardware-ORB approach has substantial advantages compared with the Software-ORB such as:

- performance: not being able to be clocked fast enough to deal with the ever-increasing performance requirements of SDR applications,
- size: the IP core taking up large amounts of gates,
- development times: as processors embedded in FPGAs represent a more complicated HW/SW technology than separated processors and FPGAs.

Figure 1 illustrates the basic CORBA environment on FPGA, related with the transport facility, that is the essential and one of the most critical part of the support facilities to be provided.

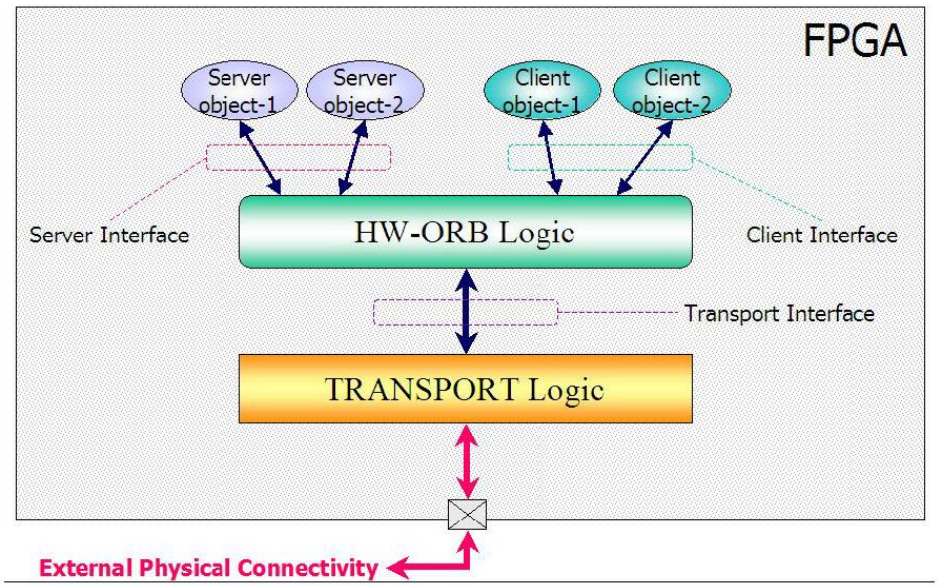

Figure 2 - FPGA basic environment for CORBA support

With the new technology improvement, constituted by native gate-level implementation of ORBs [5], the need to develop custom proxies on General Purpose Processors (GPPs) and Digital Signal Processors (DSPs) can be eliminated. In fact it would be solely used to establish communication to waveform objects residing within FPGAs. These proxies are meant to increase portability and re-use, but in practice, they tend to increase latency, reduce throughput, and lower re-use.

The hardware ORB engine can be delivered as an IP core, and is responsible for implementing the transfer syntax at GIOP (General Inter-ORB Protocol) level used in CORBA messages. The engine unmarshals an incoming GIOP stream and extracts header and data fields. Endianness conversion, when needed, is performed on incoming data, based on information in the GIOP message header. In the incoming direction, the engine performs operation named de-multiplexing to determine which object the data in the GIOP message is being transferred to. Message data is then extracted and transferred to the appropriate logic.

If the incoming GIOP message indicates that a response is expected, the ORB engine generates a reply message. The ORB-engine performs a read operation to the involved object in order to obtain data for the reply. When a reply message has been built, the ORB-engine transfers the data to the outside world via the transport interface.

Similar to software ORBs IDL compilers that maps IDL into software languages, the hardware ORB development environment includes an IDL-to-HDL compiler. This compiler is also responsible for generating configuration parameters needed by the ORB engine to perform the operation of demultiplexing and data routing.

IDL interfaces specification, such as the WF component interfaces (CF Resource and user-defined), can be compiled by the IDL compiler and supported in a native FPGA implementation of the SCA WF component.

The hardware developer treats the hardware ORB as any other IP interface core. The ORB-core can be instantiated (in the HDL capture of the FPGA design) between the native waveform logic and the transport side. The transport side of the ORB-core appears typically as a FIFO interface. The WF-logic side of the core has a simple and open interface to communicate with the waveform logic, depending on the supported IDL interfaces.

Software developers can treat hardware-implemented CORBA components as they would treat any other CORBA object. This design approach makes communication between the SW and HW objects seamless. Using a hardware ORB, waveform developers can host WF elements in an FPGA, which can still be addressed and called by an SCA-compliant software as though it was a standard SCA object, without any perception that it is residing in an FPGA.

It is recommended, for FPGA CORBA Operating Environments, to adopt a hardware-ORB (FPGA_ORB), i.e. an ad-hoc IP core implemented in gates within FPGAs, which carries out the typical functionalities of an ORB.

An HW ORB-on-FPGA provides 10 to 100 times performance improvement over a software based ORB, at a cost of a minimal silicon space on the FPGA total footprint.

It is recommended, for FPGA CORBA Processing Elements, to adopt the “Common IDL profile for DSP and FPGA processing elements”, for the specification of waveform-components interfaces.

The adoption of a “Common IDL profile for DSP and FPGA processing elements” in FPGA PEs improves portability of a CORBA-based WF component from an FPGA to a DSP and vice versa.

A WF component developed for an FPGA environment, and whose IDL interfaces are designed according to the Common IDL Profile, can be ported to a DSP, starting from the same IDL interfaces, and keeping the behaviour of the component transparent towards the rest of the system, independently from its location. The opposite is valid too, i.e. an FPGA component can be ported to a DSP.

3.3 Common IDL profile for DSP and FPGA processing elements

DSP and FPGA hosted waveform components are typically used for the fast processing capabilities that these kinds of processing elements present.

All the IDL features are neither typically needed in DSP or FPGA processing nor easily implementable on these kinds of processing elements. In order to ease waveform portability it is defined the following “common IDL profile”, for both CORBA and Non-CORBA approaches; it is composed by supported features and optional features.

The IDL features of the common profile have been selected considering the following points:

- typical DSP/FPGA algorithms implementations have to operate very fast, but on simple data types.
- it is possible to convey information (back and forth) by using simple data containers the IDL features of the common profile have to be supported by COTS ORBs current implementations (also in FPGAs that represent the more constrained processing elements in terms of ORB support).
- IDL interfaces, based on selected profile, shall be not too limited, but present an adequate flexibility to allow the required data communication.

1 Supported IDL features

Feature	Description
IDL basic data types	Short
	Long
	unsigned short
	unsigned long
	Boolean
	Octet
IDL complex data types	struct (restricted to supported basic data types)
	sequence (restricted to supported basic data types)
	Enum
IDL keywords	Module
	Interface
	In
	Out
	Inout
	Void
	Typedef
	oneway
Return value	Return values of a basic data type to be supported

2 **Table 3 – ESSOR Common IDL profile – Supported features**

3

1 Optional IDL features

Feature	Description
Usage of `struct` in `sequence`	A `sequence` can contain also `struct` complex type (of supported basic data types), in addition to supported basic data types, as already specified in the supported features.
Unbounded `sequence`	A `sequence` should be bounded whenever it is possible. In other words, the usage of unbounded sequences should be advised only if unavoidable.
`Const`, `FALSE`, `TRUE`	Constant data values can be specified by using supported basic data types. `FALSE` and `TRUE` are the values needed by the `boolean` basic data type.
`raises` (`exception`)	The usage of `exceptions` should be avoided whenever possible. In fact in most cases their function can be carried out and so replaced by using a '`out`' parameter or a return value.

Table 4 – ESSOR Common IDL profile – Optional features

4 IDL PROFILE FOR DSP AND FPGA PE

4.1 DSP CORBA Profile

The DSP AEP is a RTOS profile that consists in a subset of POSIX (IEEE Std 1003.1) source code-level interface and associated behaviour. For some API, only a restricted behaviour (compared to the behaviour described in [6]) is defined. The restricted behaviour is described when applicable. When conflict between the current RTOS profile and POSIX occurs, RTOS profile is the reference.

There are some implications on the different API, when error management or nominal behaviour is too heavy and too complex to be supported in DSP, a restricted behaviour (compared to the behaviour described in [6] is defined and described in the DSP AEP Table.

API	POSIX Profiling	RATIONALE
pthread :		
pthread_attr_init		Used to initialize the task attributes before creating the
pthread_attr_setschedparam		Used to modify the task priority attribute
pthread_attr_setstacksize		Used to modify the task Stack Size attribute
pthread_create		Used to create the task
pthread_cancel	At least allow the following behaviour: the cancellation processing is run in the calling thread	Used to delete the task. [POSIX] requires that the cancellation processing run asynchronously with respect to the calling thread returning from pthread_cancel(). This needs to have a dedicated pthread for the cancellation processing which creates useless overhead.
Mqueue :		
mq_open	At least the following mq_attr structure values are supported: mq_flags == O_RDWR lmq_msgsize = 4 bytes	Used to create a message queue. [POSIX] defines mq_open to 1) creates a message queue with access permissions (mode parameter) or 2) open a message queue referenced by a name with three access mode (Read,Write,RW). =>Only the first feature is needed. =>Access permissions are not handled by the OE
mq_close		Used to delete a message queue
mq_send	Msg_prio is ignored , all message are processed with the same priority within a given queue	Used to send a message on a message queue. [POSIX] requires that the message is inserted into the message queue at the position indicated by the msg_prio argument. This behavior creates useless overhead

mq_receive	At least msg_prio = NULL is supported	Used to receive the oldest message on a message queue. [POSIX] requires that ,If the argument msg_prio is not NULL, the priority of the selected message shall be stored in the location referenced by msg_prio. As no priority are passes in mq_send, this feature is not needed.
mq_timedreceive	At least msg_prio = NULL is supported	Used for Non-Blocking receive. identical to mq_receive except that if there is no message in the queue, the caller wait till timeout is reached.
Semaphore :		
sem_init		Used to initialize the semaphore
sem_timedwait		Used to wait for a semaphore with
sem_post		Used to signal a semaphore
API	POSIX Profiling	RATIONALE
sem_wait		Used to wait for a semaphore. sem_wait and sem_post are for use with counting semaphores, which keep track of the number of times the semaphore has been posted.
sem_destroy		Used to delete a semaphore
Mutex :		
pthread_mutex_init	At least attr = NULL (mutex type is PTHREAD_MUTEX_DEFAULT) is supported	Used to initialize a mutal exclusion – semaphore. The difference with semaphore is that mutex avoid Priority inversion
pthread_mutex_lock		Used to wait for a mutex
pthread_mutex_trylock		Used for Non-blocking wait for a
pthread_mutex_unlock		Used to signal a mutex
pthread_mutex_destroy		Used to delete the mutex
pthread_mutex_attr_init		Used to modify the mutex

Timer :		
timer_create	At least Clockid = CLOCK_REALTIME is supported At least the following sigevent structure values are supported : int Sigev_notify = SIGEV_THREAD; (pthread_attr_t*) sigev_notify_attributes = NULL;	Used to create a timer. [POSIX] defines Clockid parameter to specify timing base. Only one timing base is needed in the OE. [POSIX] defines sig_event structure to specify the notification when the timer expires. The need is to have a notification function called in the current task context when the timer expires.
timer_delete		Used to delete a timer
timer_settime	At least Old_setting =NULL is supported At least the following Timespec structure values are supported : At least seconds = 0 is supported Timer Accuracy is 1 us.	Used to arm a timer. [POSIX] defines an old_value parameter used only if timer when timer already armed.this feature create useless overhead
Memory management :		
malloc		Used to get a memory block
free		Used to release a memory block

Table 5 - DSP AEP

5 ACRONYMS

Acronym	Description
AEP	Application Environment Profile
API	Application Programming Interface
CF	Core Framework
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DSP	Digital Signal Processor
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GIOP	General Inter-ORB Protocol
GPP	General Purpose Processors
HDL	Hardware Description Language
HW	Hardware
IDL	Interface Description Language
JTRS	Joint Tactical Radio System
MHAL	Modem Hardware Abstraction Layer
OE	Operating Environment
ORB	Object Request Broker
OS	Operating System
PE	Processing Element
POSIX	Portable Operating System Interface for UniX
RD	Radio Device
RS	Radio Service
RTOS	Real Time Operating System
SCA	Software Communications Architecture
SW	Software
WF	Waveform

Table 6 - Listing of acronyms

END OF THE DOCUMENT