Project acronym:	**SACRA**
Project title:	Spectrum and Energy Efficiency through multi-band Cognitive Radio
Project number:	European Commission – 249060
Call identifier:	FP7-ICT-2007-1.1
Start date of project:	01/01/2010 **Duration:** 36 months

Document reference number:	**D5.3**
Document title:	Adapted BB processor, embedded software generators, integration in the TTool design framework
Version:	1.0
Due date of document:	30th of June 2012
Actual submission date:	2nd of August 2012
Lead beneficiary:	IT
Participants:	Jair GONZALEZ (IT), Renaud PACALET (IT)

Project co-funded by the European Commission within the 7th Framework Programme		
DISSEMINATION LEVEL		
PU	Public	**X**
PCA	Public with confidential annex	
CO	Confidential, only for members of the consortium (including Commission Services)	

EXECUTIVE SUMMARY

This document is a quick start helper for the SACRA baseband processor and for the SACRA UML-based software design framework.

CONTENTS

1 INTRODUCTION

This deliverable is composed of two technical contributions: the SACRA baseband processor, EMBB, and a software design framework, based on the TTool[1] open source project.

1. The SACRA baseband processor, EMBB, has been adapted to the specific SACRA needs. It is now up and running and the selected applications for the SACRA demonstration activities have been ported on EMBB. EMBB is delivered in three forms:

   - A user manual detailing all the supported functionalities that a programmer can access
   - `libembb`, an emulation software library, 100% bit accurate, written in C++ but with a C API, its user documentation, its technical documentation and several examples of use
   - A SystemC virtual prototype and a dedicated Real Time Operating System that altogether form a bit-accurate, cycle-approximate simulator of EMBB. This last view constitutes the SACRA milestone M5.5: *Complete software design suite available and ready for integration in WP6. It is not replicated.*

   The low level Hardware Description Language (HDL) source code is not delivered yet because it requires a deep clean-up phase before becoming usable by others than the designers. This last view, plus the programming bit-streams for the ExpressMIMO FPGA-based prototyping board, will be released at the end of the SACRA project.

2. The software design framework is a prototype of an advanced UML-based tool. It is intended to ease the software design by taking in charge most of the low-level and error prone design tasks. It relies on the TTool open source project to which it adds a dedicated UML profile (Diplodocus Data Flow), software generators that translate UML descriptions in plain C source code and examples.

1.1 Document versions sheet

Version	Date	Description, modifications, authors
0.1	29.06.2012	Initial version, Jair Gonzalez (IT), Renaud PACALET (IT)
1.0	02.08.2012	Final version, Jair Gonzalez (IT), Renaud PACALET (IT)

2 INSTALLATION INSTRUCTIONS

2.1 Installing, compiling and running `libembb`

Download the `embb-1.0.tar.gz` tarball, unpack it and install `libembb` in a directory of your choice (`<libembb-root>`) and define the `EMBB_INSTALL` environment variable such that it points to the `libembb` installation directory:

```
tar xvzf embb-1.0.tar.gz
cd embb-1.0
./configure --prefix=<libembb-root>
make
make check
make install
export EMBB_INSTALL=<libembb-root>
```

The installation process is a very standard one and is detailed in the README and INSTALL files of the unpacked distribution.

2.2 Installing, compiling and running the DiplodocusDF UML design framework

2.2.1 Pre-requisites

The DiplodocusDF tool chain has been developed and tested on Linux. It could be that it works unmodified on other UNIX-like systems but it has not been tested yet. Windows ports are not yet available. DiplodocusDF relies on several third party, open source tools. Please check that they are properly installed and configured on your system before installing DiplodocusDF:

- GNU Make
- gcc, the GNU C/C++ compiler
- flex, the Fast Lexical Analyser
- byacc, the Berkeley Yacc
- libembb, the emulation software library
- TTool, which is a part of DiplodocusDF distribution. Note: TTool relies on JDK 1.6 or later. For more information, please refer to `http://ttool.telecom-paristech.fr/installation.html`

DiplodocusDF is a toolchain meant for the development of wireless communications applications. It is based on abstract modelling and automatic code generation. It extends TTool, which is a tool for abstract modelling and design space exploration. DiplodocusDF mainly adds a UML profile and `ddf2c`, a converter capable

of translating UML diagrams into C-code. The UML diagrams model the application, the target hardware/software execution platform and the mapping of the former to the latter. Once an application has been modelled it is first translated into TTool's internal representation. The internal representation is in turn translated into a set of C-code source files and Makefiles that can be compiled and linked against the emulation software library.

2.2.2 Installation

First download the `diplodf.tar.gz` tarball, unpack it in the installation directory of your choice (`<diplodf-root>`), define the `DIPLODF_INSTALL` environment variable and adapt your `PATH`:

```
cd <diplodf-root>
tar xvzf diplodf.tar.gz
export DIPLODF_INSTALL=<diplodf-root>/diplodf
export PATH=$PATH:$DIPLODF_INSTALL/ddf2c:$DIPLODF_INSTALL/ttool
```

This creates a `diplodf` directory containing three sub-directories, `ttool/`, `ddf2c/` and `examples/`:

- `ttool/`
  - `ttool.jar`: TTool's Java archive
  - `ttool_conf.xml`: XML TTool's configuration file
  - `ttool.exe`: TTool's launcher
- `ddf2c/`
  - `Cgen.l` and `Cgen.y`: `ddf2c`'s flex/byacc parser
  - `Cgen.h`: header file of the code generator
  - `eMIMO_settings.h`: Model Extension Constructs (MEC) of the target ExpressMIMO platform. The MECs are platform specific language constructs defining the characteristics of the target platform.
  - `rdy_code/`: run-time library on which the generated software relies
  - `Makefile`: to build `ddf2c`
- `examples/`
  - `wpd.xml`: the DiplodocusDF model of an example application, the Welch periodogram detector

Build `ddf2c`, the TML to C translator:

```
cd $DIPLODF_INSTALL/ddf2c
make
```

2.2.3 Running the example

The provided example is a Welch periodograms-based signal detector. First create an empty directory for your first experiments with DiplodocusDF:

```
mkdir <some-other-path>/myFirstDF
cd <some-other-path>/myFirstDF
```

The next step involves TTool. As the provided example is already designed, there is no need to modify it. Note: this is not a TTool tutorial; to learn more about TTool and see how to design your own UML diagrams please visit `http://ttool.telecom-paristech.fr/`. Launch TTool:

```
ttool.exe
```

Open the DiplodocusDF UML model of the Welch periodogram detector example:

```
File -> open  -> $DIPLODF_INSTALL/examples/wpd.xml
```

Select the "DIPLODOCUS Architecture" tab and run the syntax analyser:

```
V&V -> Syntax analysis -> Start Syntax Analysis -> OK
```

Generate the TML / TMAP intermediate representation of the design in the current directory (files `spec.tml, spec.tarch, spec.tmap`):

```
Code Generation -> Generate TML / TMAP in text format
```

Generates the C code from intermediate representation, also in the current directory:

```
ddf2c -a $DIPLODF_INSTALL/ddf2c/rdy_code/
```

The generated C code is an almost complete software implementation of the application modelled in the Diplodocus Data Flow UML profile. The generated source files are:

- `wpd_init.c`: initializes the data structures defining the behaviour of all involved operations. In the current version, the memory allocation and the detailed definition of the instructions is left to the designer. In future versions, more of these settings will be synthesized. The manual personalization is achieved by editing this file and replacing the `/*USER TODO*/` fields by actual code.
- `wpd.c`: the actual code of the application.
- `wpd.h`: header file defining functions and variables used by the application. Most declarations are internal ones but two functions that are called by the designer from her top level including the input / output interface functions that the designer will call from her top level.
  - `void wpd_init( <parameters> )`: used to set the different parameters of the application
  - `int wpd(void)`: starts the execution
- `eMIMO.h, eMIMO.c`: declare and define functions and structures related to the target execution platform (the libembb emulation library in this example).
- `Makefile`: used to build the whole application.

First edit the `wpd_init.c` source file and add the missing personalization. Then, create a top level in a source file named `main.c`. The top level is a very simple piece of code containing a few declarations and calls to the `wpd_init` and `wpd` functions. Alternately, copy the already designed source files from the `$DIPLODF_INSTALL/examples/wpd` directory to the working directory (`<some-other-path>/myFirstDF`):

```
cp $DIPLODF_INSTALL/examples/wpd/main.c <some-other-path>/myFirstDF
cp $DIPLODF_INSTALL/examples/wpd/wpd_init.c <some-other-path>/myFirstDF
```

Note: subsequent runs of `ddf2c` would generate again the `wpd_init.c` template file, overwriting the carefully hand-crafted one. In order to avoid this, `ddf2c` will refuse to overwrite existing files without the user's explicit consent.

Once all the manual coding is done the `run.exe` executable is built by:

```
make
```

The provided example reads the input samples to process in a binary data file which can be found in the `examples` directory of the distribution. The `run.exe` built executable takes the data file as its single argument:

```
./run.exe $DIPLODF_INSTALL/examples/wpd/data.dat
```

The printed output is the final energy score calculated by Welch periodogram detector.:

```
Detection score: 1253587
```

BIBLIOGRAPHY

[1] TTool, a toolkit dedicated to the edition, formal verification and simulation of UML and SysML diagrams: `http://ttool.telecom-paristech.fr/`.