1 Introduction

The purpose of this application note is to summarize and explain tasks required to use software device drivers for Xylon's logicBRICKS™ IP cores in standalone (no operating system) Xilinx® FPGA based system configurations. The described workflow works with the Xilinx Software Development Kit (SDK) embedded software development environment for Xilinx embedded processors.

logicBRICKS is Xylon's library of IP cores designed for Xilinx FPGA and EPP products. All logicBRICKS IP cores come with detailed documentation, software support and reference designs. The IP cores are fully supported by the Xilinx Platform Studio (XPS) and the EDK integrated software solution, which enables FPGA designers that are familiar with Xilinx tools to start working immediately. The logicBRICKS allow for configuration through a graphical user interface (GUI) and the implementation of targeted SoCs without hand coding.

logicBRICKS software device drivers fully support the functionality and configurability of related logicBRICKS IP cores.

2 Typical logicBRICKS IP Core Deliverables

logicBRICKS IP core deliverables are compatible with the Xilinx Platform Studio and the EDK integrated software solution. The deliverables include: encrypted VHDL IP core's sources, device driver and xyl_oslib library.

The folder structure of the logicBRICKS IP core:

- pcores
  - ip_core HW description
    - data (XPS related files)
    - doc (IP documentation)
    - hdl (encrypted IP HDL sources)
    - simmodel (simulation model)
- drivers
  - ip_core_generic_driver
    - src – (driver sources and makefile)
    - data – (the .mdd and .tcl driver files)
    - examples – (driver examples)
    - doc - (driver documentation (please see *doc/html/index.html*))
- sw_services
  - xyl_oslib (Xylon OS abstraction library for Xilinx Xilkernel embedded kernel and BSP; support for other OS on request)

3 What Are xyl_oslib and logicBRICKS Generic Device Drivers?

Xylon's generic device drivers for logicBRICKS IP cores are similar to software device drivers for Xilinx's IP cores delivered with the Xilinx ISE® Design Suite: Embedded Edition. The logicBRICKS device drivers are generic because they are fully independent from the operating system.

In addition to the logicBRICKS software device drivers, Xylon provides an operating system abstraction library called xyl_oslib.

Xyl_oslib provides the following interfaces to generic device drivers:

- Bus access to memory/registers
- Data types
- Memory allocation
- Timers
- DMA
- Debug
- SpinLock

A quick look at logicBRICKS device drivers' source code shows that they use the xyl_oslib system abstraction library extensively. It makes the maintenance easier, and reduces design efforts during IP core porting to different operating systems and hardware platforms.

Please check the Chapter 7 for more details on xyl_oslib library.

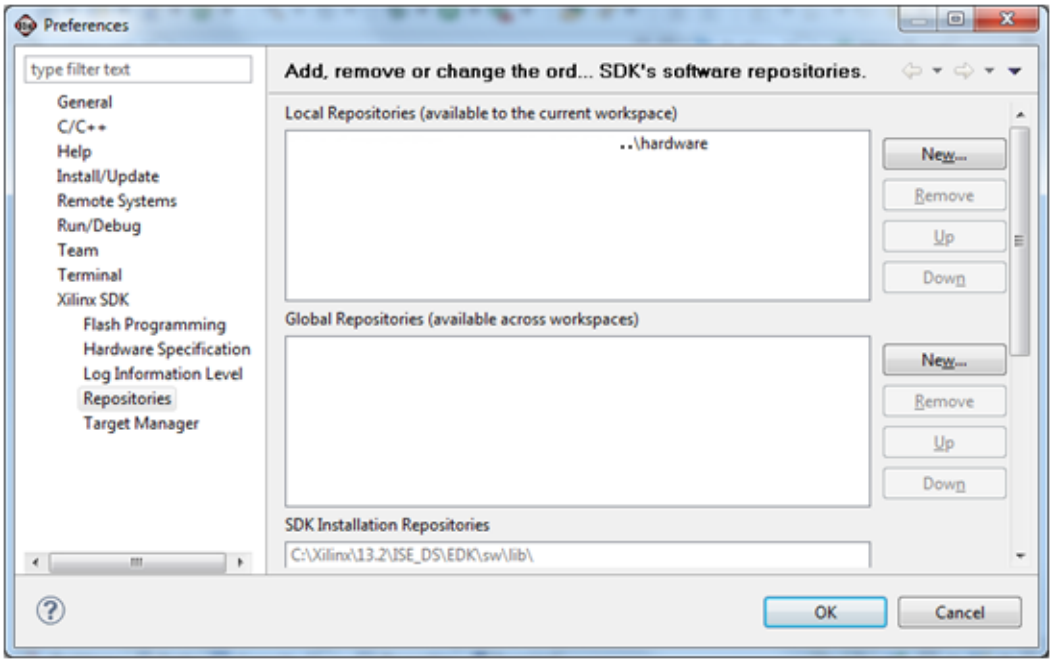
4 Compiling the logicBRICKS Device Driver in the Xilinx SDK

Xylon logicBRICKS device drivers are designed to work with the Xilinx SDK Software Development Kit, and are very similar to Xilinx device drivers. However, there are small differences when working with the Xylon and Xilinx device drivers.

4.1 Including logicBRICKS Drivers in the SDK Search Path

Xylon logicBRICKS drivers are not delivered with the Xilinx ISE/EDK installation. They have to be obtained from Xylon and added in the SDK search path:

- select **SDK**->**Xilinx Tools** -> **Repositories**, the Preferences dialog box opens (Fig. 4.1)
- select **New** and browse for directory containing the Xylon drivers and SW_services folders
- click on **Rescan Repositories**



4.1 Setting up the Search Path to Xylon Drivers/SW Services

4.2 Including logicBRICKS Drivers in the SDK "Xilinx BSP project"

Properly setup search path enables the SDK to locate device drivers for Xylon logicBRICKS IP cores. The following steps describe the procedure for a new BSP project creation:

1. **SDK-> New -> Xilinx Board Support Project** (select HW platform and standalone/Xilkernel) and click Finish (Fig. 4.2)

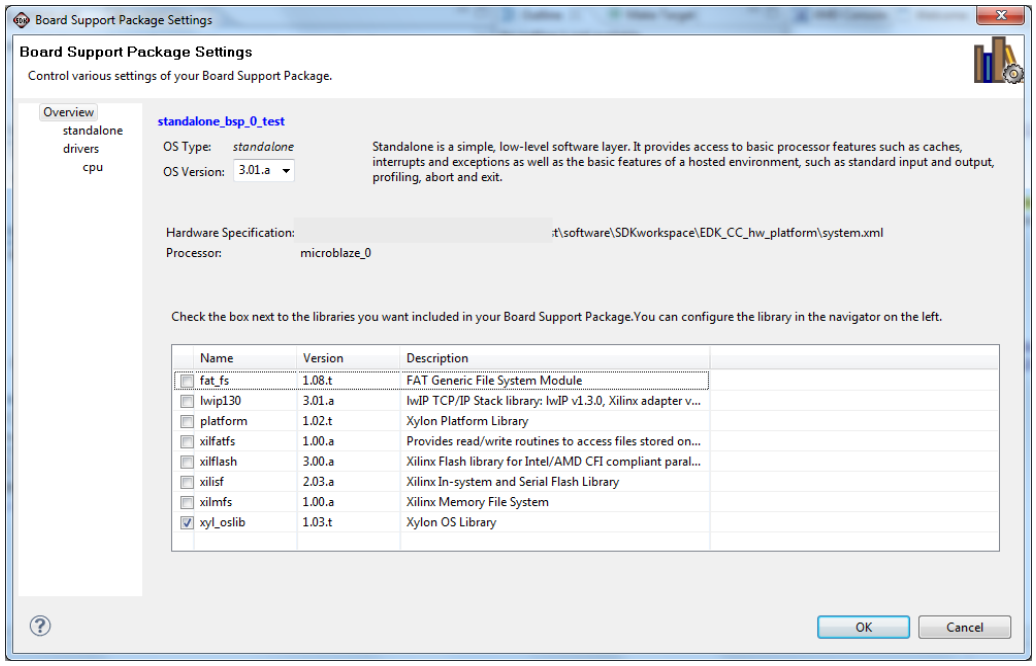
2. In the Board Support Package Settings window (Fig. 4.3):

    a. Select the **xyl_oslib checkbox** under **Overview -> Supported libraries**

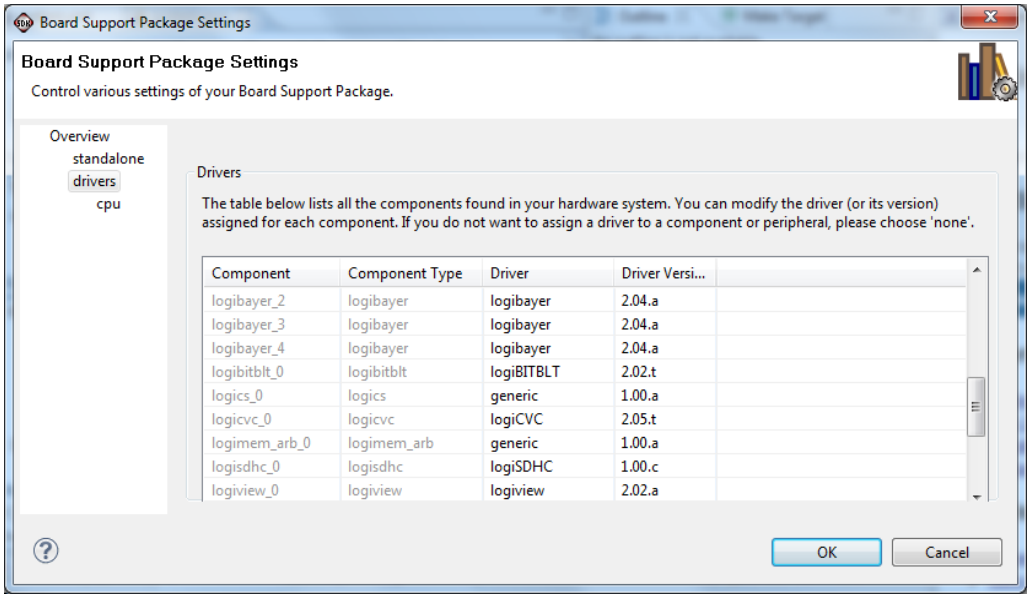
    b. Select the Xylon **drivers** for Xylon IP cores under Overview->drivers

    c. Add "**–DNDEBUG**" in the **Overview->Drivers->CPU->extra_compiler_flags** (Fig. 4.4). This recommended (not mandatory) compiler flag reduces the size of the xyl_oslib library when it is used by drivers.

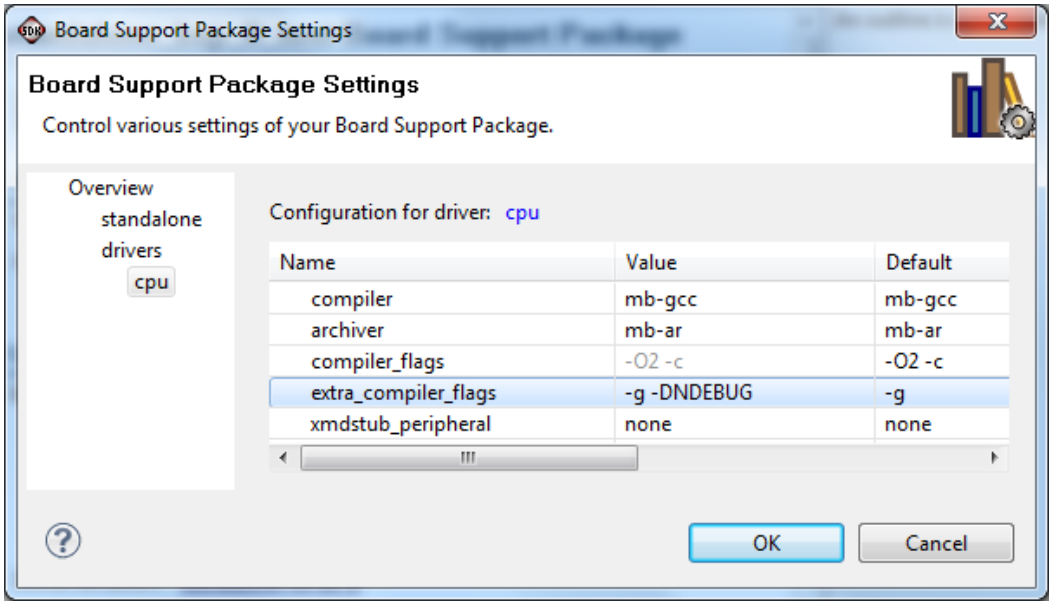
Xylon drivers and xyl_oslib library compile like Xilinx drivers, but the resulting static library is called "libdrivers.a".



4.2 Selecting xyl_oslib Library



4.3 Selecting Xylon Drivers



4.4 Compiler Flags (-DNDEBUG)

5 Using logicBRICKS Drivers from the SDK SW Applications

There are two main differences between the typical Xilinx SDK driver and Xylon logicBRICKS generic driver:

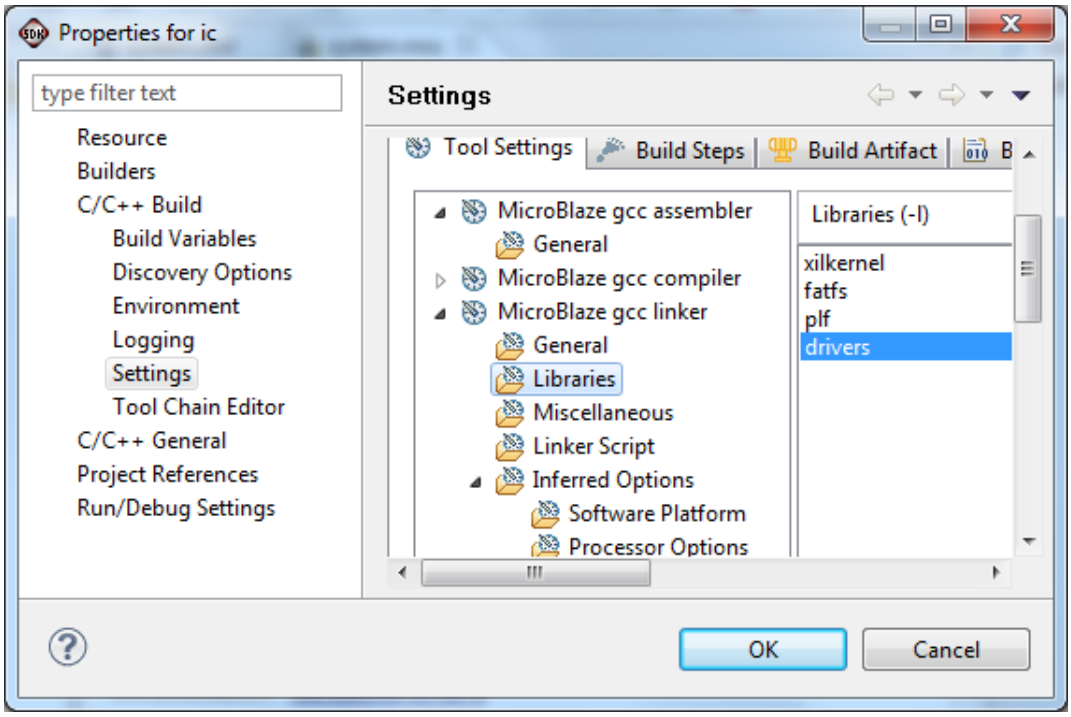
1. logicBRICKS drivers and xyl_oslib binaries are linked to a static library called “libdrivers.a”, which has to be setup in the application project linker settings

2. logicBRICKS drivers depend on the xyl_oslib library. Xyl_oslib for Xilinx Xilkernel or Xilinx BSP requires additional functions to be provided from the application. For this purpose Xylon provides **SwConfig_X.c** configuration templates, described in 5.2.

5.1 Linking Xylon logicBRICKS Drivers from the Application

The “drivers” entry should be added in SW application project’s property box **C/C++ Build settings->gcc linker->Libraries** (Fig. 5.1).

NOTE: application settings for Debug and Release build configurations have to be set separately.



5.1 Add Option to Link libdrivers.a in the Application

5.2 The SwConfig configuration files for the xyl_oslib library

SwConfig files were introduced in the xyl_oslib library because there is an often need for additional configurations on a per-application basis. Some software applications use the Xilinx Xilkernel and link with it in RAM memory, while others with a very small memory footprint, use Xilinx BSP without the Xilkernel and link in on-chip Block RAM (BRAM) memory. Instead of providing several specialized instances of the xyl_oslib library, Xylon provides this configuration at the software application level.

The SwConfig.c file has to implement the following functions
(see xyl_oslib_vX_y_a\src\possix\common\inc\SwConfig.h):

```
/*************************** TIMER MANAGEMENT ****************************/
void    swConfig_initTimerSubsystem(void);
OsSizeT swConfig_getTicksInMs(void);
void    swConfig_startTimer(void);
void    swConfig_stopTimer(void);
OsSizeT swConfig_getTickCount(void);
void    swConfig_sleep(OsSizeT ms);
OsSizeT swConfig_getElapsedMs(OsSizeT ms);
```

```
/********************** MEMORY ALLOCATION MANAGEMENT ***********************/
void *swConfig_malloc(OsSizeT size);
void  swConfig_free(void *p);

/************************* THREAD LOCK MANAGEMENT **************************/
OsSpinLockT *swConfig_spinLockConstruct(OsObjectHandleT hParent);
void        swConfig_spinLockDestruct(OsSpinLockT *pLock);
void        swConfig_spinLockAcquire(OsSpinLockT *pLock);
void        swConfig_spinLockRelease(OsSpinLockT *pLock);
```

Xylon provides several SwConfig.c template files with the library:

- xyl_oslib_vX_y_a\src\possix\common\src\swconfig_template**SwConfig_xilkernel.c**
  - to be used when the application uses the Xilinx Xilkernel
- xyl_oslib_vX_y_a\src\possix\common\src\swconfig_template**SwConfig_bsp.c**
  - to be used when the application uses the Xilinx BSP (no Xilkernel)
- xyl_oslib_vX_y_a\src\possix\common\src\swconfig_template**SwConfig_no_heap.c**
  - to be used when the application uses the Xilinx BSP (no Xilkernel) and the available memory is very small (redefined malloc/free to some simple implementation)

User can modify existing templates to fit system requirements.

5.3 Setting up the SwConfig Files Required by the xyl_oslib Library

Xylon provides template SwConfig.c files in the following folder:
xyl_oslib_vX_y_a\src\possix\common\src\swconfig_template

The SwConfig files setup:

1. Setup the application project build settings to compile one of the template SwConfig.c files located in the xyl_oslib
2. Copy one of the template SwConfig.c files from xyl_oslib folder to the application project folder. Setup application project build settings to build this local file
3. If the provided template SwConfig.c files miss needed functionality, copy the template file and make modifications. Setup the application project build settings to build the modified SwConfig.c file

6 Making Application Project with Xylon Device Driver Examples

All Xylon logicBRICKS device drivers are delivered with example source files, which are located in the ${DRIVER_NAME}/examples folder. The examples are typically tested on specific hardware platforms and include hardware platform dependant and independent code.

NOTE: Platform dependent parts of the driver example may not be directly applicable to other targeted platform and driver's source code changes may be necessary.

NOTE: Xylon provides a platform library that supports several hardware platforms manufactured by Xylon and Xilinx. The provided device driver examples can be used on already supported hardware platforms with no modifications.

6.1 Example Project Setting Up

The logiCVC-ML Compact Multilayered Video Controller IP core is an advanced display graphics controller for LCD and CRT displays, which enables an easy video and graphics integration into embedded systems with the Xilinx FPGAs. This IP core is a typical example of logicBRICKS IP cores that come with the software device driver and here presented information applies to other logicBRICKS IP cores as well.

The logiCVC-ML IP core's deliverables include the device driver, which allows for an easier work with the display graphics, with the following files provided in the examples folder:

1. logiCVC_main.c (hardware platform dependant)
2. logiCVC_demo.h (hardware platform independent)
3. logiCVC_demo.c (hardware platform independent)

logiCVC_main.c file initializes the hardware platform and calls functions from the logiCVC_demo.c.

To make an application project with the video controller IP core's device driver example:

1. Include the platform independent part (logiCVC_demo.c) of the example code in the project.

2. Modify the platform dependent part (logiCVC_main.c) of the example to match used hardware platforms, and include it to the project. Code modifications are not required if Xylon already supports the selected hardware platform.

3. Add the "–ldrivers" in the application project linker properties (see 5.1)

4. Add the SwConfig.c file in the application project build settings (see 5.3)

7 Xyl_oslib - In More Detail

7.1 General Idea Behind the xyl_oslib Library

Device driver for specific hardware is usually written multiple times for different operating systems and communication busses which are used to access the hardware.

The idea behind the Xyl_oslib library creation can be demonstrated by an example of two hardware blocks, the UART and the I2C IP cores, which should work with two different operating systems (Linux and Microsoft® Windows® CE) and two different communication busses (PCI and USB bus).

There are 8 HW/SW combinations that should be supported by separated device drivers:

1. UART, Linux OS, PCI bus
2. UART, Linux OS, USB bus
3. UART, Microsoft WCE OS, PCI bus
4. UART, Microsoft WCE OS, USB bus
5. I2C, Linux OS, PCI bus
6. I2C, Linux OS, USB bus
7. I2C, Microsoft WCE OS, PCI bus
8. I2C, Microsoft WCE OS, USB bus

Device driver writing is an expensive process, and in order to increase efficiency, implement optimal code and decrease work efforts and costs, Xylon programmers isolate common drivers' elements.

10 separated software modules can be isolated from the above example requiring 8 different HW/SW combinations:

1. Linux I2C driver
2. Linux UART driver
3. Microsoft WCE I2C driver
4. Microsoft WCE UART driver
5. I2C generic part of driver
6. UART generic part of driver
7. Microsoft WCE PCI bus interface
8. Microsoft WCE USB bus interface
9. Linux PCI bus interface
10. Linux USB bus interface

The number of isolated modules is greater than a number of the planned HW/SW configurations, but at the end, this way of device driver partitioning adds value through reusability:

1 - 4 are OS dependant and require upper device driver's layer to be written for each new OS. These driver's layers are not reusable. They are only wrappers and require relatively small working efforts from designers

5 and 6 are generic hardware drivers that describe the hardware (IP core) functionality. The IP core generic drivers can be reused during driver porting to new OS/BUS combination.

7 and 8 are bus access interface implementations for the Microsoft WCE OS. This part is common for all drivers that will be written on Microsoft WCE and can be reused.

9 and 10 are bus access interface implementations for the Linux OS. This part is common for all drivers that will be written on Linux and can be reused.

7.2 Layered Device Driver's Architecture

Xylon logicBRICKS device drivers implement the layered architecture to leverage reusable portions of the drivers' code. The driver layers are:

1. OS driver (OSDRV)

   OS dependant part of the driver, which is specifically made for the targeted OS and the targeted bus interface (i.e. Linux driver for UART working with the USB bus). The OSDRV is usually just a small code wrapper required by OS for a particular type of driver. Main driver functionality should reside in the next layer.

2. Generic driver (GENDRV)

   Generic part of the driver that describes the hardware functionality, i.e. the generic part of the UART driver defines UART registers and common functions for usual UART operation. The generic driver is completely independent from the OS and the bus interface.

3. OS library (OSLIB)

   A library written for the specific OS system that provides a common interface which generic drivers can use with any OS/bus combination. It provides the interface to the specific OS types, synchronization mechanisms, bus access mechanisms, and other.

   NOTE: Driver layers are separated only in the source code format. Compiled and linked driver layers make a single binary file.

The layered device driver's architecture minimizes the amount of required work for driver porting to new OS/bus combinations:

1. For each OS, the OSLIB has to be written only once
2. For specific HW platform, the generic GENDRV has to be written only once
3. OSDRV has to be written for each new OS/bus combination

7.3 Xyl_oslib Interface

Xylon OS library (xyl_oslib) provides interfaces to all OS specific functions required by the IP core's device driver and enables logicBRICKS generic device drivers (GENDRV) to remain completely OS and bus interface type independent.

The library provides the following interface groups:

- OS types - it provides typical types (pointers, unsigned 8, 16, 32, etc.). See Oslibtypes.h

- OS debug - provides debug macros for printing, asserting. See OslibDebug.h

- OS HW access - provides access functions to hardware register. Hardware access is independent from the bus interface type. Bus independency is obtained by pointer to struct _OsHwAccessObject, which is an argument to all access functions. See OslibHwAccess.h.

    - Functions OsRegRangeInit and OsRegRangeDeinit provide init/deinit interface for the bus access descriptor.

    - OsHwResourceT represents the hardware resource input structure

    - struct _OsHwAccessObject represents the OSLIB descriptor for accessing hardware registers

- OS spinlock -provides interface for spinlocks, see OslibSpinLock.h

- OS misc - provides interface for miscellaneous OS functions, see OslibMisc.h

- OS Time – provides interface to time functions, see OslibTime.h

Revision History

Version	Date	Note
1.00.a	9.1.2012.	Initial Xylon release

Xylon d.o.o.
Fallerovo setaliste 22,
10000 Zagreb, Croatia
www.logicbricks.com