Release Notes 2019.4-sp1

SILEXICA

Destination Control Statement
All technical data contained in this publication is subject to the export control laws of the German Federal Republic. Disclosure to nationals of other countries contrary to Germany law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer
SILEXICA GMBH, MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks
Silexica and certain Silexica product names are trademarks of Silexica GmbH. All other product or company names may be trademarks of their respective owners.

Third-Party Links
Any links to third-party websites included in this document are for your convenience only. Silexica does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Silexica GmbH
Lichtstr. 25
50825 Cologne, Germany
www.silexica.com

Contents

Preface

Revision History

Date	Version	Revision
29/06/2015	2015.6	Official Release
22/01/2016	2016.1	Official Release
14/10/2016	2016.10	Official Release
23/12/2016	2016.10-sp1	Service Pack
24/03/2017	2016.10-sp2	Service Pack
29/04/2017	2017.4	Official Release
16/06/2017	2017.4-sp1	Service Pack
20/10/2017	2017.10	Official Release
29/03/2018	2018.3	Official Release
08/06/2018	2018.6	Official Release
17/10/2018	2018.10	Official Release
21/12/2018	2018.10-sp1	Service Pack
12/03/2019	2019.1	Official Release
18/07/2019	2019.2	Official Release
08/08/2019	2019.2-sp1	Service Pack
27/09/2019	2019.2-sp2	Service Pack
10/12/2019	2019.4	Official Release
05/02/2020	2019.4-sp1	Service Pack

Typographic Conventions

This manual uses specific typographic conventions. The following table summarizes how font styles are used to emphasize important elements throughout the text.

Style	Usage
Bold	Names of Silexica products, such as **SLX for FPGA**
`Typewriter`	Literal input, e.g., user input from the command line and actual code listings
Slanted	To introduce terminology used in the Silexica (SLX) series of products

Special symbols are used to denote useful information, a remark, a warning, an expected failure or a requirement for user interaction. They are used within a shaded box with rounded corners.

Symbol	Meaning
	Useful information to the user, specific to this context
	A noteworthy point or remark
	A warning on tool-specific behavior
	An expected error or failure that the user should be aware of
	The user is required to provide some form of input
	The user should select from one of a number of choices

1

Introduction

This document gives an overview of all **SLX** solutions that are deployed as a stand-alone desktop environment. Moreover, it describes what is new in **SLX** 2019.4 and how to migrate from previous releases.

1.1 A New Era in Computing

The shift from single-core to multicore hardware platforms affects virtually all computing domains today. Multicore CPUs and heterogeneous system-on-chip platforms promise to meet skyrocketing application performance demands at moderate power and energy consumption. Contemporary user applications ranging from wireless communication, artificial intelligence, robotics to embedded vision often require performance enhancements that cannot be reached solely by deploying commercial off-the-shelf (COTS) processors. In such cases, dedicated or specialized hardware implementations of key system parts on an FPGA (Field-Programmable Gate Array) are essential. The task of hand-designing optimized Intellectual Property blocks for implementing application hot-spots or critical algorithms is expensive, time consuming and dependent on highly specialized hardware design expertise. Transforming arbitrary software code into an optimized and synthesizable HLS FPGA-ready representation is a big challenge that requires major application rewrite investments and a wealth of software and FPGA expertise.

1.2 The Multicore Challenge

Manual optimization of legacy software for complex multicores is extremely costly, risky, tedious, and error-prone. With exponentially growing complexity in both application software and hardware platforms, the challenges are here to stay.

To successfully utilize HLS technology, the input specification, given as a high level language code (C/C++) must meet certain coding guidelines. Additionally, deriving an

optimal solution (in terms of area and performance) is neither an automated process nor an easy one. The causes for these inefficiencies are inherent to shortcomings of the HLS engine, such as:

- No early estimations of performance as well as of any violations of design and platform constraints
- Not taking advantage of all possible ways of data communication
- The absence of optimizing transformations on the user's code
- Unsupported code and inefficient code styles being reported too late

Overcoming any of these limitations requires a lot of time and therefore increases the non-recurring costs for projects based on HLS.

1.3 The SLX Solution

SLX hides away the target hardware complexity from the software programmer. The advanced code analysis and optimization technologies facilitate migration of existing software to heterogeneous multicore systems as well as an efficient implementation of new parallel applications. Using the **Silexica** tools, the following benefits can be reaped:

Quality of results:

- Full utilization of the available hardware resources, leading to optimal system performance that meets latency and throughput constraints.
- Hardware cost reduction by avoiding oversized hardware platforms.
- High software robustness due to "correct-by-construction" multicore software design.

NRE cost reduction:

- Dramatic cuts in software engineering costs due to automation and optimization in sequential software parallelization, task mapping, hardware/software partitioning and low-level code generation.
- Significant productivity gains by hiding multicore hardware complexity.
- Preserving legacy code investments when porting complex software to new hardware platforms.

Flexibility:

- Modular integration into the customer's programming flows and software tool environments.
- Straightforward scalability for new software workloads or future hardware platform generations.

End user benefits:

- Higher performance, longer battery lifetime, higher software reliability, maximum system availability, enhanced functionality and better quality-of-service and user experience.

2

Product Overview SLX FPGA

SLX FPGA helps to convert C/C++ code into an FPGA bitstream easier, faster, and with higher performance. Leveraging standard HLS (High Level Synthesis) tools from FPGA vendors, **SLX FPGA** tackles the challenges associated with the HLS design flow, including non-synthesizable C/C++ code, non-hardware aware C/C++ code, detecting application parallelism, where to insert pragmas, and how to determine optimal SW/HW partitioning. Using **SLX FPGA** enables you to get to market faster by leveraging the benefits of HLS for FPGA design entry. These benefits include improved productivity through designing at a higher level of abstraction, orders of magnitude faster simulation than traditional RTL simulation, and higher QoR through high-level optimizations and design space exploration.

SLX FPGA addresses the challenges of using a HLS design flow by first performing a static code analysis, and then a dynamic analysis that provide deep insights into the C/C++ code. **SLX FPGA** identifies non-synthesizable C/C++ code, detects non-hardware aware data types, and pinpoints parallelism within the SW that can be implemented in HW for acceleration. **SLX FPGA** provides guided and automatic code refactoring for HLS synthesizability helping to solve the biggest barrier and most time-consuming aspects of using HLS design flows. **SLX FPGA** then uses the detected parallelism to automatically generate and insert HLS pragmas which optimize the design for performance and area utilization.

2.1 Features and Capabilities

SLX FPGA provides a step by step flow to optimize C/C++ code for High Level Synthesis, significantly reducing development time and ensuring an optimized hardware implementation of C/C++ code.

2.1.1 Convert Non-Synthesizable C/C++ Code

C/C++ coding guidelines for HLS compilers are extensive and can be over 1000+ pages of documentation that needs to be comprehended when writing or refactoring C code for HLS synthesis. **SLX FPGA** eliminates the need to be an expert in coding for HLS by:

- Allowing the user to choose portions or all of the C/C++ code to test for synthesizability
- Performing automatic code refactoring for many common C libraries
- Providing guided code refactoring by supplying code examples to help re-write the code to make it synthesizable

2.1.2 HW Optimization and HW/SW Partitioning

After identifying functions that can be implemented to execute in parallel, **SLX FPGA** performs analysis of the functions to determine the theoretical maximum number of parallel executions. Using Silexica's proprietary algorithms, **SLX FPGA** then determines the ideal implementation of the parallel function based on user supplied constraints, ensuring an optimized implementation.

2.1.3 Pragma Insertion

Once the optimized hardware implementation is determined, **SLX FPGA** inserts HLS Pragmas to direct the HLS complier on how to implement the function in hardware. **SLX FPGA** is fully integrated with Xilinx Vivado HLS and the SDSoC Development Environment to create a complete path from C/C++ to FPGA synthesis. It can be used on the desktop from a powerful GUI, from the command-line or integrated into a continuous workflow.

3

Product Overview SLX C/C++

Optimizing C/C++ applications on complex multicore SoCs requires a complete understanding of the software architecture and its behaviour on the hardware. Developers are challenged with the task of optimizing sequential and parallel code for multicore SoCs that comprise a variety of compute engines. For the most efficient utilization of CPUs, DSPs and FPGAs, a full understanding of the code structure and interdependencies between applications, threads and variables is required for streamlined software development, guided refactoring and software design optimization.

SLX C/C++ enables the development and optimization of concurrent C/C++ applications for heterogeneous multicore platforms with unparalleled actionable insights. This allows for software professionals to achieve an optimized software architecture and improved performance for a specific code on a particular multicore SoC. SLX blends hardware understanding with code analysis to give unrivalled insights into the application's execution while providing a feedback loop to the original source code to provide full traceability. **SLX C/C++** can be used on the desktop from a powerful GUI, from command-line or integrated into your continuous workflow.

3.1 Features and Capabilities

3.1.1 Joint static and dynamic code analysis for multiple applications

State-of-the-art code analysis methods provide only a limited view of the software architecture. Static code analysis can find simple bugs or disregarded coding guidelines. Dynamic analysis provides a single profiling run without an understanding of the variance of its behaviour or any root-cause connection back to the source code.

The **SLX C/C++** analysis allows for understanding of the static software architecture down to the dynamic behavior of concurrency, synchronization and data flow. This provides a live overview from the source code to prevent architecture erosion and provide actionable insights to optimize the software implementation.

3.1.2 Software Architecture Analysis with SLX C/C++

SLX gives unprecedented insights into execution behaviour. It weighs the costs and benefits of software optimization to further exploit the target's resources. Key features include:

Provide deep application insights to multi-binary and multi-threaded applications. Through the combination of static, dynamic, and semantic analysis, SLX visualizes thread genealogy, communication, synchronization and data dependencies, providing a live architectural overview from the source code which can be checked for consistency with the envisioned architecture. SLX is the only development tool available today that provides this level of actionable insights.

Identify communication and memory bottlenecks by performing analysis at the function, thread and application level. SLX shared memory (POSIX shared memory variables) analysis makes you aware of how the application communicates among threads and with other applications. SLX shows all accesses to variables including sub-objects of arrays and structs even when accessed through pointers. This provides an up to date architectural overview based on the actual source code to assist with functional debugging, code refactoring and generation of documentation.

Identify missing inter-thread and inter-process shared memory protection such as semaphores or mutexes that may lead to data corruption by performing protection analysis. SLX understands protection mechanisms and can point directly to the problematic source lines so the code can be fixed. This enables not only the detection of data races in between threads, but also in between separate processes and applications.

Optimize software distribution for the hardware compute blocks on a heterogeneous multicore system by performing fast "what-if" analysis to visualize code execution. Optimizations are driven by performance, power and memory requirements for a specific code base given a combination of CPUs, DSPs, FPGAs described in a standardized hardware description format (SHIM2 by the Multicore Association).

Save development time by automatically identifying optimization opportunities in the code. SLX provides guidance to assist code refactoring for improved performance and parallelism detection. Different levels of parallelism are supported including task, pipeline, and data level parallelism.

4

Product Overview SLX Scheduling Design

SLX Scheduling Design receives C/C++ code, AUTOSAR specifications or AMALTHEA models as input and automatically identifies all different types of dependencies among application tasks and schedulable functions (or runnables) that have to be considered during multicore migration. Internally, it uses static and dynamic code analysis techniques, and performs powerful data-flow analyses. Supported functionality:

- Static and dynamic analysis, instrumentation and tracing of sequential C/C++ code.
- Visualization of task and runnable dependence graphs, call graphs, code profile information, code coverage, variable access information and cache analysis statistics.
- User-defined modeling of dependencies to perform what-if analysis or direct tool optimizations.
- Identification of inter-runnable parallelism and computation of multicore runnable schedules.
- Automatic identification of intra-runnable parallelism in the source code and production of easy-to-understand feedback to the user.
- Generation of multicore AUTOSAR configurations and inter-runnable synchronization primitives.
- Support for AUTOSAR 4.3.0 applications and Amalthea 0.8.2 models
- Integration with 3rd party AUTOSAR basic software generation tools like Elektrobit Tresos.

5

Product Documentation

Full documentation as well as an easy-to-follow getting started guide for all **SLX** products is provided with the tools installation. Please refer to the following end user manuals, which can be found inside the `doc` folder at the root of your installation folder:

- **Installation Guide**
- **SLX C/C++ User Guide**
- **SLX FPGA User Guide**
- **SLX Scheduling Design User Guide**
- **Platform and Core Modeling Guide**

6

2019.4 Release

6.1 Overview of New Features

The following sections give an overview of the new and exciting major features provided with **SLX FPGA** 2019.4.

Please refer to the corresponding sections in the SLX FPGA User Guide for more details.

- **Support for arbitrary precision integer data types**: **SLX FPGA** parallelism detection and analysis tools are extended to support applications that utilize arbitrary precision integers ("ap_int" and "ap_uint"). This enables **SLX FPGA** to perform parallelism detection in loops that use variables of these types for processing, allowing for the generation of pragmas on these loops.
- **Improved Synthesizability Checks and Guidance: SLX FPGA** has extended its synthesizability checker to support C++ specific constructs, and warns the user when non-synthesizable code is used. When non-synthesizable constructs are found, **SLX FPGA** provides guidance for re-writing the code to be synthesizable.
- **Function Mapping Editor:** The User Interface was fully reworked to provide earlier and faster feedback on Synthesizability of functions. This allows for faster turn-around times, design updates and design iterations. In addition, it gives hints on what to inspect further and provides guidance through the optimization flow. The Function Mapping Editor provides a centralized location that displays the project's functions and their dependencies in a Function Mapping Graph, as well as each function's properties. Users can choose which functions to test for Synthesizability from the Function Mapping Graph. Parallel Regions can be configured in the Properties section of each FPGA targeted function.
- **Modeling of Platform Interfaces:** The Properties section of the new Function Mapping Editor also allows the configuration of Interfaces and Bandwidth available to synthesizable (top-level hardware) functions.

- **Analysis time improvements:** With **SLX** 2019.4, the code analysis time has been improved. In addition, with the rework of the flow, the user can decide which functions to investigate further and where to spend time in optimizing the application. SLX hotspot analysis can be used to help with optimization decisions.

6.2 Migrating from Release 2019.2

6.2.1 New License Features Required

No new license features added since **SLX** 2019.2. No need to update your SLX license.

If you have a license for an older version of **SLX** you need to update to a new license. You can download it from FlexNet Operations also. Find a detailed guide in **Installation Guide** under License Setup. Or feel free to reach out to a sales distributor to get access to the new features.

6.2.2 SLX FPGA

6.2.2.1 Compatibility with Previous Results

This release has file formats containing profiling and analysis results that are incompatible with the 2019.2 release. In addition, the internal structure of these files in the **SLX** internal analysis directories shifted and cause incompatibilities with older workspaces. To experience all new features hassle-free, please re-run the tools. If you are experiencing difficulties migrating an existing project, please contact **Silexica's support portal**.

6.2.3 SLX C/C++

6.2.3.1 Compatibility with Previous Results

This release has file formats containing profiling and analysis results that are incompatible with the 2019.2 release. In addition, the internal structure of these files in the **SLX** internal analysis directories shifted and cause incompatibilities with older workspaces. To experience all new features hassle-free, please re-run the tools. If you are experiencing difficulties migrating an existing project, please contact **Silexica's support portal**.

6.3 Known Issues

- **SLX FPGA and C++14**: C++14 applications are not supported because of limitations in the Xilinx compiler. C++ applications are supported in **SLX FPGA** for

application profiling and analysis, but the automatic translation from general C++ into high-level synthesis compatible C++ is not yet supported.

- **IDE Rendering**: If there is a huge amount of data to be visualized, there may be BIRT[1] or GTK[2]-related rendering issues in **SLX**. We are very sorry for the inconvenience and working hard with the BIRT/GTK teams to solve these problems for the specific library for your platform.
- **GUI Font Sizes**: If GUI font sizes appear small in Linux e.g., in a ultra high-definition monitor, use the following command for starting the tools from a console:

```
GDK_DPI_SCALE=1.5 ./SLX
```

6.4 Third-party Open-source Software

SLX contains third-party open-source software components. The full list including the licenses can be found in the `data/third-party-licenses` sub-directory of the **SLX** installation and online at https://www.silexica.com/tps/.

[1] http://www.eclipse.org/birt/
[2] http://www.gtk.org/