

Usage of System Level Modelling with SystemC

Inputs from DATE'08

By senior consultant Kim Bjerger (kim.bjerger@teknologisk.dk)
Copyright © 2008 Danish Technological Institute

Preface.....	1
What is SystemC?	3
System-Level Specification using SystemC	5
Transaction Level Modelling	8
Analog Mixed Signal System modelling (AMS).....	10
A SystemC model-based codesign methodology.....	11
Functional View	12
Architectural view	18
Implementation view.....	23
References.....	25

Preface

This article will describe some of the areas and applications of SystemC for modelling of electronic systems consisting of hardware and embedded software. SystemC can be used in a HW/SW codesign flow where an executable model is created for both the digital and software part of the system being developed. Some of the new standards even target simulation of the analog part together with the digital part of the design where continuous and discrete simulation is mixed together in the same Electronic System Level design (ESL).

This article is based on inputs from the “Design, Automation and Test in Europe” (DATE'08) conference in Munich. Presentations were made by researchers, “Electronic Design Automation” (EDA) vendors and users covering progress of extensions and standards for the use of SystemC language in different areas of modelling, verification and test.

The rapid increase in design complexity and costs of failure raises creates a need in system development in many product domains for a top-down methodology based on a higher level of modelling than the Register Transfer Level (RTL). As design increases in size and simulation time, the RTL modelling level is also increased. The use of a Transaction-Level Modelling (TLM)

methodology makes it possible to create a design in which some communication details in terms of signals and timing can be eliminated. In later iterations of the design flow, signals and timing can be added by using adapters to create adaptation from TLM to RTL level. This approach enables faster simulations at TLM level.

The initial TLM design can be used as a “Programmers View” (PV) model whereby it is possible to start software development before the actual hardware is completed. This approach has in particular been used in the mobile technologies industry which has witnessed an increasing demand for time-to-market and requirements for parallel development of hardware and software.

In recent years a methodology based on TLM is evolving for large system designs, in particular for chip/ASIC development. It will have an effect on the FPGA area very soon. Xilinx announced last year that they have entered collaboration with some of the major EDA players (Cadence, Mentor Graphic and Synopsys) whereby some of the ideas concerning transaction-level modelling and formal verification will be available for FPGA designers. Many EDA vendors today support the use of SystemC in different ways, including modelling at different levels of abstractions, reuse of models for verification or even synthesis from a SystemC design to implementation in HDL.

A number of standards are being created such as the new TLM2 from OSCI (Open SystemC Innovative) in order to be able to have interoperability and fast simulation of IP models when used in designs by different vendors and users. This paper presents a brief overview of the new TLM2 standard.

Reasons for Using TLM

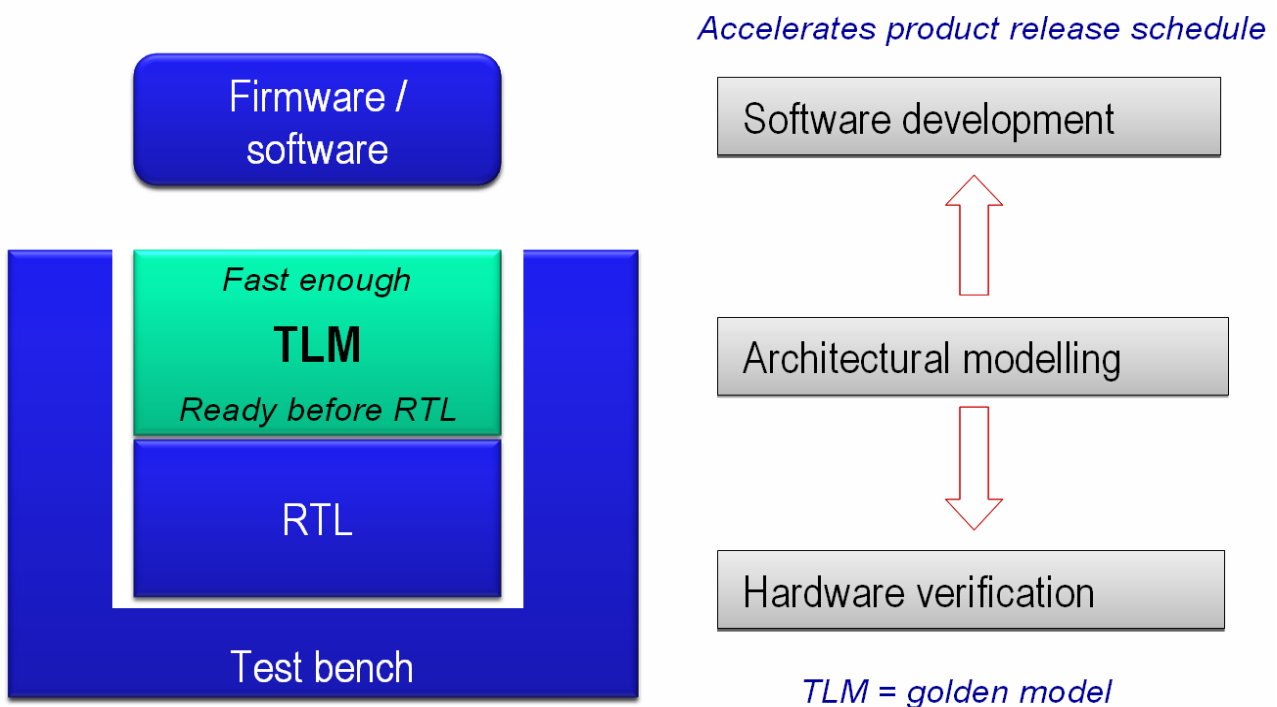


Figure 1 Reason for Using TLM

Usage of System Level Modelling with SystemC

Open SystemC Innovation (OSCI) is an independent, not-for-profit association composed of a range of organizations dedicated to define and promote SystemC as an open industry standard for system-level modelling, design and verification.

IP-XACT is another standard in this area which covers a set of specifications for IP meta-data and tool interfaces based on XML. This standard is driven by the SPIRIT consortium that has many of the same members as OSCI. The purpose is to enable automated design creation and configuration based on a tool independent standard.

The last chapter of this document contains a SystemC model-based codesign methodology where the Unified Modelling Language (UML) and SystemC are used in an example to describe the development method. The example targets an FPGA implementation including an embedded soft-core processor. The HW/SW codesign methodology is currently being developed and used by the Danish Technological Institute.

What is SystemC?

SystemC is an extended library based on C++ for creating models of electronic systems covering hardware and software elements. The following list shows the major components of SystemC.

- Simulation Kernel
- Data types covering logic, integers and fixed point
- Modules and hierarchy
- Channels and interfaces
- Events, sensitivity and notifications
- Threads and methods
- Predefined primitive channels as mutex's, FIFO's and signals

SystemC Language Overview

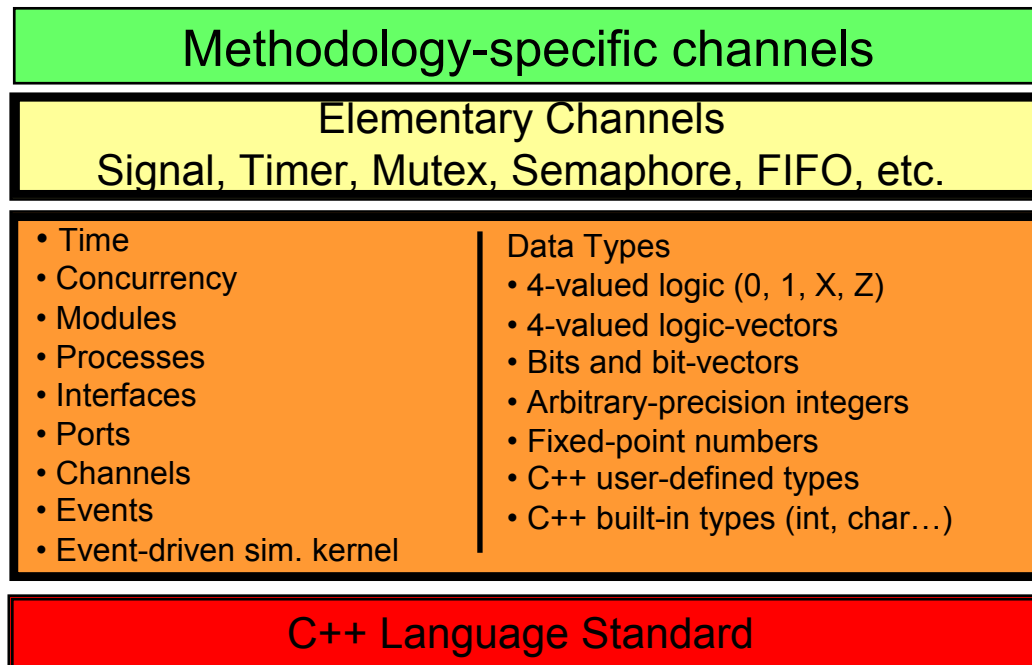


Figure 2 SystemC Language Architecture Overview

SystemC addresses the modelling of both software and hardware using C++. Since C++ already covers software, it should come as no surprise that SystemC focuses primarily on non-software areas. The primary application area of SystemC is design of electronic systems. The major hardware-oriented features implemented within SystemC include:

- Time model
- Hardware data types
- Module hierarchy to manage structure and connectivity
- Communications management between concurrent units of execution
- Concurrency model

SystemC contains an ultra light-weight cycle-based kernel for high-speed simulation. It makes it possible to have multiple levels of abstractions ranging from high-level functional models to detailed clock cycle accurate RTL models. It supports an iterative refinement of high-level models into lower levels of abstraction. The high-level part of a model can be reused for creation of test bench for verifying design and hardware implementation.

Electronic systems development using SystemC makes it possible to create an executable specification which is executed with the same behaviour as the final system.

The standard SystemC library can be downloaded free of charge from the OSCI website. It was approved by the IEEE Standard Association in 2005 and can be built and executed on Windows, Unix and Linux platforms.

System-Level Specification using SystemC

Today, all development of complex, heterogeneous embedded systems are model-based. Sound and well-understood models are necessary bases for all design, analysis and verification activities.

At DATE'08 some of the tutorials contained theoretical concepts of Models of Computation (MoC) which can be of practical use for industrial design flows by using proper libraries based on SystemC, methodologies and modelling rules in contrast to industry practices based on 'ad-hoc' solutions.

One of the tutorials was presented by the Royal Institute of Technology in Sweden. The ForSyDe "Formal System Design" is a theoretical framework for system specification which contains different models of computation and defines 4 different models:

- Untimed MoC:
 - No explicit time, ordering of events
 - Invocation based on data availability
- Synchronous Time MoC:
 - Slot-based time abstraction
 - Invocation in every slot
- Discrete Time MoC:
 - Physical, discrete time in seconds
 - Invocation based on data availability and progress of time
- Continuous Time MoC:
 - Physical, continuous time in seconds
 - Continuous invocation based on transfer functions

The refinement model steps contain 2 levels of abstraction:

- Ideal System Model
 - No resource limitation on
 - Processors, Communication bandwidth or memory
- Implementation Model
 - With finite resources
 - Processors, HW blocks
 - Reconfigurable resources
 - Buffers
 - Communication architecture
 - Schedulers, arbiters

The framework can be downloaded from: <http://www.imit.kth.se/info/FOFU/ForSyDe/>

HetSC a “Heterogeneous System Specification & Library” contains a specification methodology with an associated SystemC library for heterogeneous, system specification and design. This library is developed by the University of Cantabria and can be downloaded from:

www.teisa.unican.es/HetSC

HetSC supports untimed, synchronous and discrete time for different MoCs. HetSC makes it possible to create a heterogeneous system specification to integrate different computation models.

CoWare has created an extension to TLM which is named SCML. It can be downloaded from the www.coware.com website. It is used in CoWare products for modelling reuse. SCML combined with CoWare tools and IP's enables a Platform driven ESL Design solution.

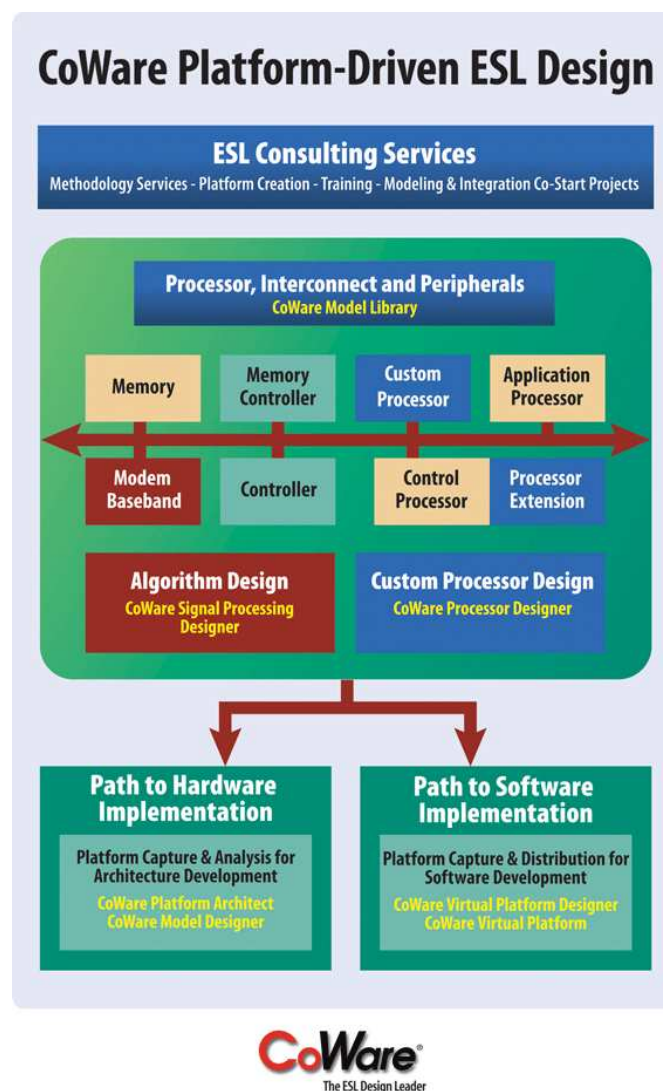


Figure 3 CoWare Platform-Driven ESL Design

A demonstration of the CoWare tools for making an architectural design in SystemC was shown at the conference. The tool allows you design an architectural timed model and subsequently to deliver a “Programmers View” (PV) model of the new ASCI design. It thereby becomes possible to start software development before the final hardware is ready. The demonstration also showed how to debug Java code on a virtual mobile phone consisting of an ARM processor, memory and peripherals.

Transaction Level Modelling

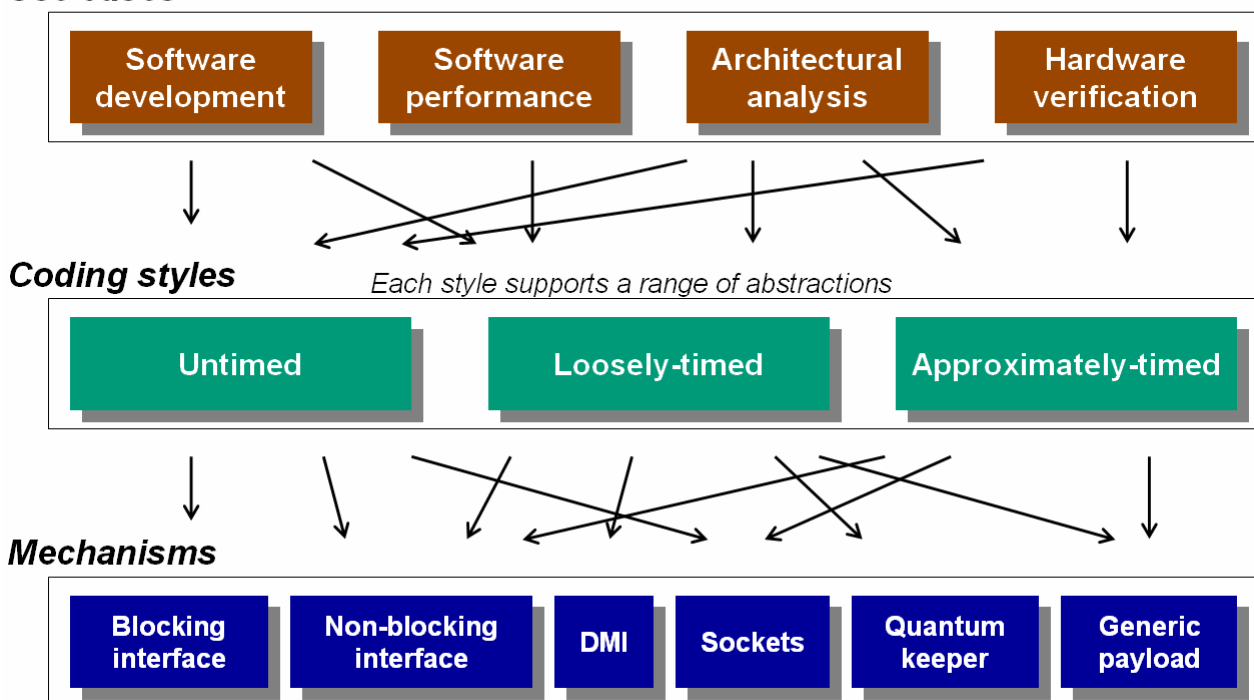
The importance of transaction level modelling (TLM) for architectural exploration, performance analysis, building of virtual platforms for software development and functional verification continues to grow. The December 2007 release of the TLM-2 Draft-2 standard from the OSCI TLM working group marks the culmination of several years of intensive work. It is a significant step towards making interoperable transaction-level modelling a reality.

At DATE'08, John Aynsley, Doulos presented the TLM-2 standard that explicitly addresses the interoperability of memory-mapped bus models at transaction level and provides a foundation and framework for the TLM of other protocols in the future.

Currently the plan is to have an official release of the TLM-2 standard at the Design Automation Conference in June 2008.

Use Cases, Coding Styles and Mechanisms

Use cases



Figur 4 TLM-2 Use Cases, Coding Styles and Mechanisms

TLM-2 defines a C++ library which contains coding styles based on simple mechanisms in different use cases of software development, architectural analyses and hardware verification.

The coding styles cover a range of abstractions using blocking and non-blocking interfaces.

Untimed

- Simulation time not required
- Each process runs until next explicit synchronization point
- Blocking interface methods or waits permit context switch

Loosely-timed

- Processes can be temporarily decoupled from simulation time
- Each process keeps a local tally of time and yields when quantum reached

Approximately-timed

- Processes run in lock-step with simulation time
- Delays annotated onto transactions cause waits or timed notifications

Some of the use cases for TLM-2 may be software development and performance modelling, architectural analysis and design exploration. Finally TLM-2 could be used as a golden model for hardware verification.

At the DATE conference, ARM showed a presentation of plans and expectations for the TLM-2 standard at the “European SystemC User Group” ESCUG meeting. ARM has planned to use a standard for IP cores, and in the future it will be based on the OSCI TLM 2.0 standard. ARM supplies a system generator tool where it is possible to create a TLM model of an electronic system. Currently, it supports ARM9 and ARM11 processors and peripherals like memory and other communication devices. Furthermore, the ARM’s tools are shown on the website www.arm.com (Search for the RealView SoC Designer and System Generator).

Infineon has used Transaction Level Modelling for many designs. The presentation by Thomas Wide, Infineon showed how the TLM approach has developed during the history of the company. As an example, Mr. Wide showed a very complex chip design for a Mobile Phone containing DSP, ARM9, ARM7, GPRS and UMTS accelerators and a large number of peripheral hardware blocks.

At present, there is a change in the paradigm of making SoC designs where Infineon see an increasing dependency on standard languages and interfaces. The interoperability of tools, IP and methodologies is therefore crucial. EDA solutions for the future must support standards to ensure this interoperability. Mr. Wide listed the SystemC, TLM2 and IP-XACT (XML) standards of great importance to this goal.

Analog Mixed Signal System modelling (AMS)

Some promising progress in the Analog Mixed Signal working group of OSCI was presented by Christoph Grimm, Vienna University of Technology.

SystemC-AMS focuses on making description, simulation and verification for:

- Functional complex integrated systems
- Analog Mixed Signal systems / Heterogeneous systems
- Specification / Concept Engineering
- System design, reference as the “golden” model development
- Embedded software development
- Next layer (driver) for software development
- Customer model

The presentation is of interest for modelling Analog Mixed-Signal systems. The progress of the AMS standard can be found at www.systemc-ams.org. It is also possible to download the SystemC-AMS library with examples from this website.

It is possible to simulate linear analogue circuits, including electronic components such as R, L, C, linear switches, current and voltage sources. A white paper on the SystemC-AMS will be available after DAC in June 2008.

With the AMS standard it will be possible to create mixed models containing software, digital and analog designs in the same language based on SystemC and C++.

A model example was presented for a Software Defined Radio (SDR) for exploration of the ideal partitioning for SW-FPGA-HW-Analog/RF parts. The system design is based on the SystemC-AMS.

The second example was a wireless sensor network containing Radio, Power, Sensor & Actuator, IP cores and Software.

A SystemC model-based codesign methodology

This chapter will present an example of a model-based codesign methodology using SystemC.

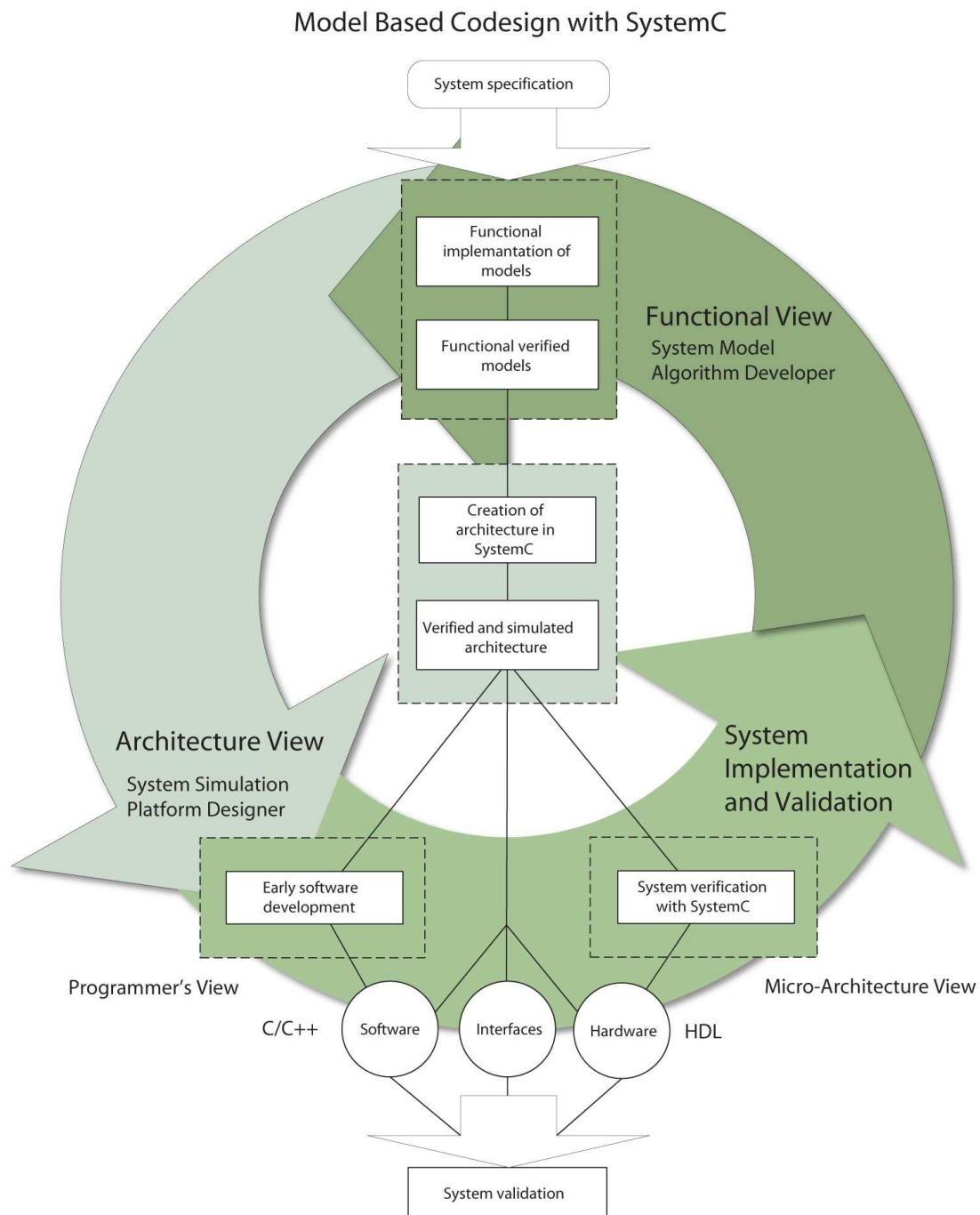


Figure 5 HW/SW codesign with SystemC

Usage of System Level Modelling with SystemC

A methodology can be viewed as a tool box containing a variety of tools – models, methods, solutions – to assist the designer in his work. All activities involved in bringing a product, a system of application from definition of requirements to finalized implementation. The method described below should be part of a complete methodology which is not yet complete.

This method should give the reader an idea of how to use SystemC from the functional level to implementation of an electronic system consisting of hardware and software components.

The example used addresses only a part of a larger system and is within the field of signal processing. SystemC could also be used for other areas of industrial data processing covering: Communication, Automation, Data Processing and Signal Processing.

The Unified Modelling Language (UML) is used to visualize the model before implementation in SystemC. UML is a widely used standard for software development, and in this methodology it will be used for a HW/SW codesign approach. UML will be used for system design to improve the graphical overview of the system covering classes, objects, components, ports, signals and their interrelationship. Based on the UML model, an executable specification is created by using the SystemC language.

The model is refined by adding timing in the architectural view. It finally reuses the SystemC models for system verification of hardware components and reuses the model for early software development.

It enables an iterative development process as the functional model is reused in the refinement. It would be easy to add functionality and extend the model that can easily be re-verified on more abstraction levels.

The tools used can all be downloaded from the internet free of charge - more information is available in the “Getting started with SystemC”. Please refer to the reference list on the last page of this document.

Functional View

This model view contains the highest abstraction in the design flow. It focuses on design without considering implementation details. The aim is to create an executable specification which is untimed and data driven with a point-to-point communication. It is hardware and software neutral and focuses on computation and algorithm functionality. The output is a functionally verified executable model which could be developed by the “Algorithm Developer”.

The example used to demonstrate the method consists of a FIR filter model which either implements a high or low pass filter. The functional view model is simulated with SystemC, and the result is verified with MATLAB by plotting the result files. The OSCI SystemC library, cygwin, GTKWave and the Eclipse platform have been used for developing the SystemC model. Please refer to “Getting started with SystemC” for guidelines of how to install and use these tools.

The model describes a FIR filter where the filter coefficient calculation is placed in software and the real time calculation in hardware. UML is used to describe the static view of this design.

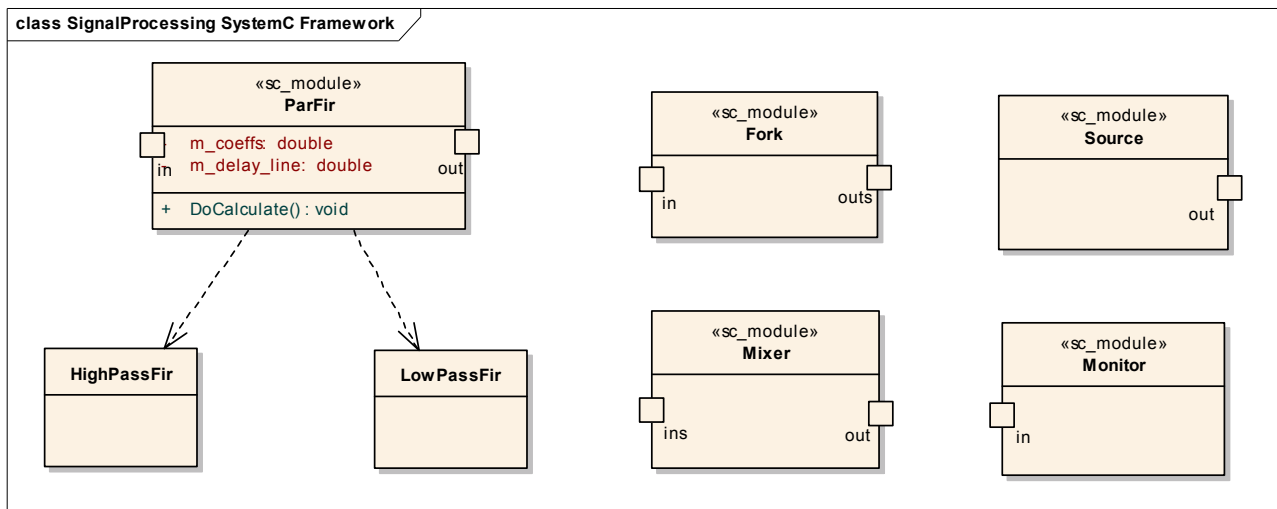
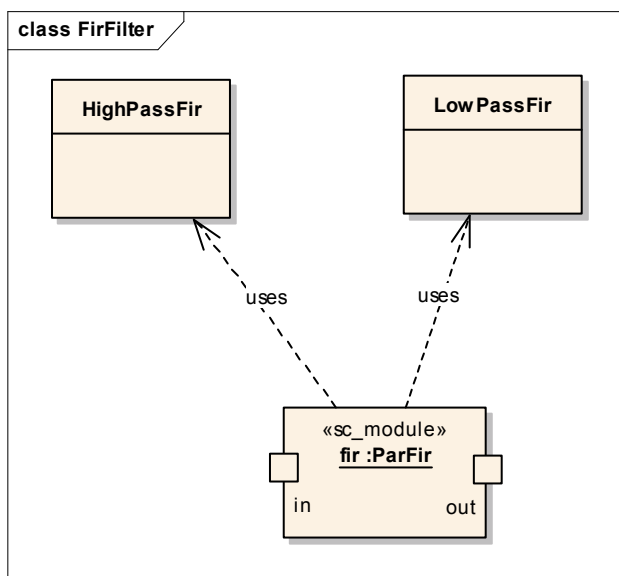


Figure 6 Functional Signal Processing modules

A minor set of SystemC modules has been defined which all connects together by ports, interfaces and the `sc_fifo` channel. The signal processing modules consist of a signal source, monitor, mixer, fork and the FIR filter. All modules are parameterized with the template functionality of C++. This approach makes it possible to decide on the data type the classes operate on when they are instantiated. It would also be easy to create other types of modules which can be added to the model. At the functional level, the type `double` is used for the data processing. Later we will reuse the same model for a fixed point simulation. In this case the signal source produces a sine wave signal. The monitor stores the digital monitored signal in a text file. The mixer takes more `sc_fifo` channels and mixes them based on a gain setting. The fork module is able to split a signal input to more `sc_fifo` channel outputs.



Figur 7 Parameterized FIR filter

Usage of System Level Modelling with SystemC

The FIR filter is created as an instance of the ParFir module. The high pass and low pass FIR coefficient calculation is implemented in 2 separate C++ classes. They are not using any SystemC functionality because they contain complex math operations which will only be partitioned to software. The FIR filter has an input port and an output port which connect to the sc_fifo_in and sc_fifo_out interfaces.

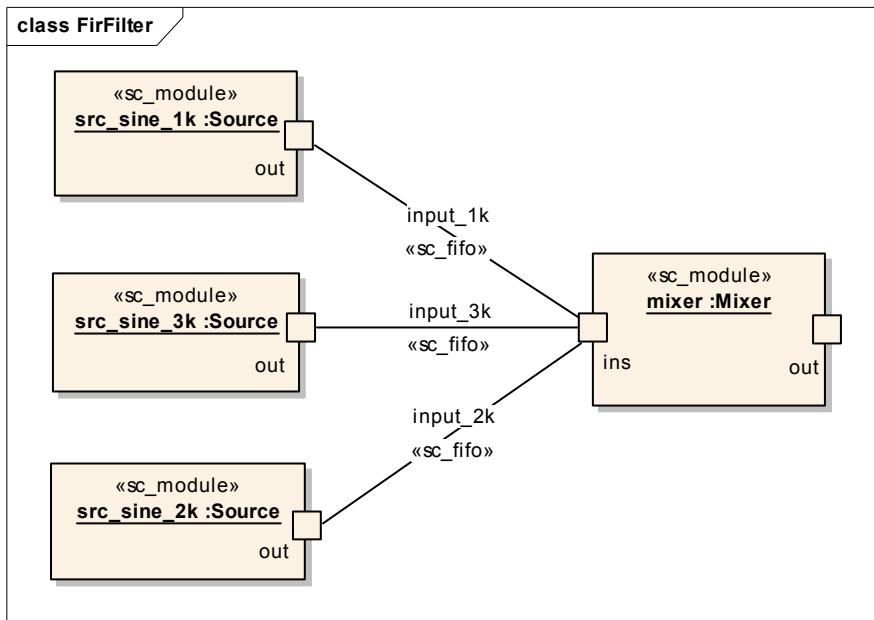
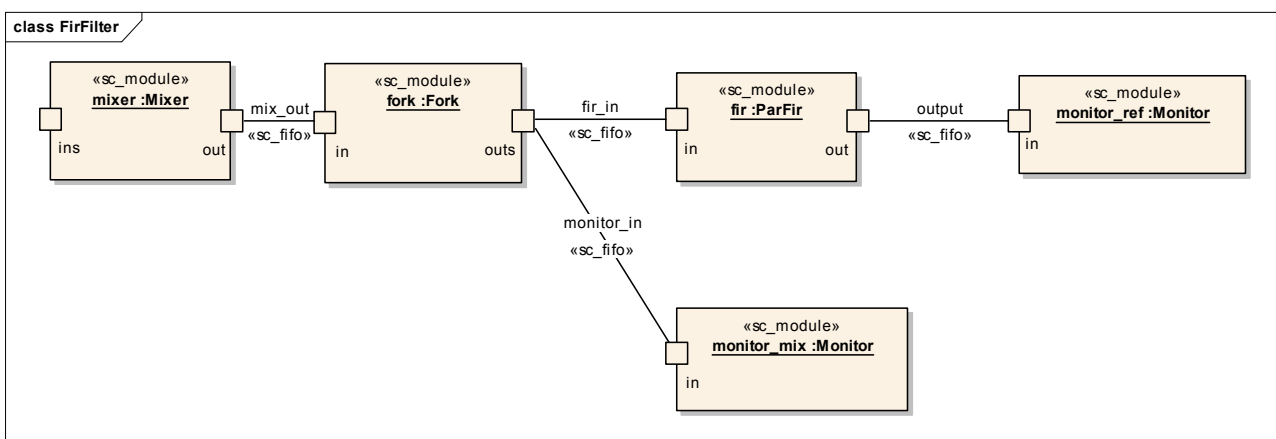


Figure 8 Signal sources and mixer

In this example, 3 instances of the sine signal source are instantiated and connected to the mixer module. The sources produce a 1, 2 and 3 kHz sine signal at a sample rate of 44.1 kHz. The design is set up to make it easy to change the parameters for the model concerning sample rate, number of filter taps and cut frequencies for the low or high pass filter.



Figur 9 Monitoring of FIR result

The Fork is splitting the output signal from the mixer to a monitor mix and the FIR filter. The filtered signal is monitored by the reference monitor.

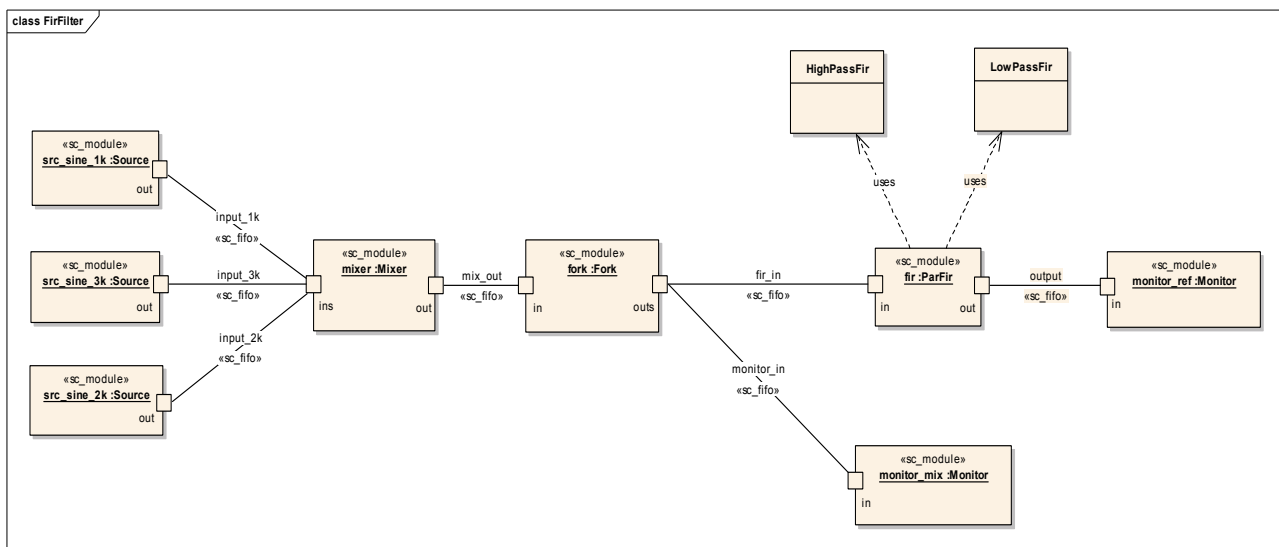


Figure 10 Completed functional model

The completed functional model is implemented in a top SystemC module which creates the instances of the modules, fifo channels and connections. On the next page, you can see the SystemC header file for the top module. All modules use the fir_T type definition that can be substituted later by the sc_fix in the architectural level of exploration.

```
typedef double fir_T;
const unsigned fs = 44100;           // Sample frequency
const unsigned fc = 2000;           // Cut frequency for filter
const int samples = 1000;          // Number of samples
const unsigned taps = TABS;         // Size of filter

class Top : public sc_module
{
public:

    // Create fifo channels
    sc_fifo<fir_T> input_1k;
    sc_fifo<fir_T> input_2k;
    sc_fifo<fir_T> input_3k;
    sc_fifo<fir_T> mix_out;
    sc_fifo<fir_T> fir_in;
    sc_fifo<fir_T> monitor_in;
    sc_fifo<fir_T> output;

    // Signal generator 1 kHz sine
    Source<fir_T> src_sine_1k;

    // Signal generator 2 kHz sine
    Source<fir_T> src_sine_2k;

    // Signal generator 3 kHz sine
    Source<fir_T> src_sine_3k;

    // Mixer to add the sine signals together
    Mixer<fir_T> mix;

    // Split the mixed signal in 2
    Fork<fir_T> fork;

    // Monitor of mixed input result
    Monitor<fir_T,1> monitor_mix;

    HighPassFir<double, taps> *pHighPass; // HighPass filter coefficients
    LowPassFir<double, taps> *pLowPass; // LowPass filter coefficients
    ParFir<fir_T, taps> *pFir; // Fir filter

    // Monitor of filtered output result
    Monitor<fir_T,6> monitor_ref;

    SC_HAS_PROCESS(Top);
    Top(sc_module_name name);
};
```

The monitors store the result of the simulated signal in text files. The contents of the text files are plotted in the following figures (11, 12) using MATLAB. The figures show the simulation of a 1 kHz low pass filter. This design allows the algorithm designer to experiment with different sample rates and filter types.

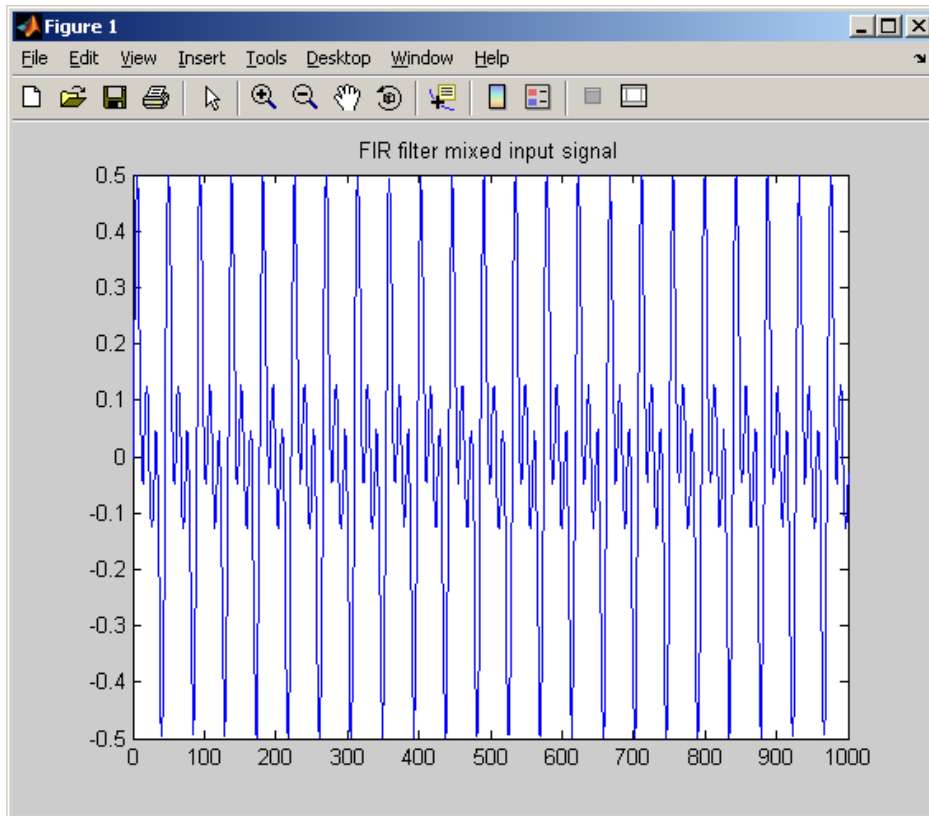


Figure 11 Monitor for mixed input signal

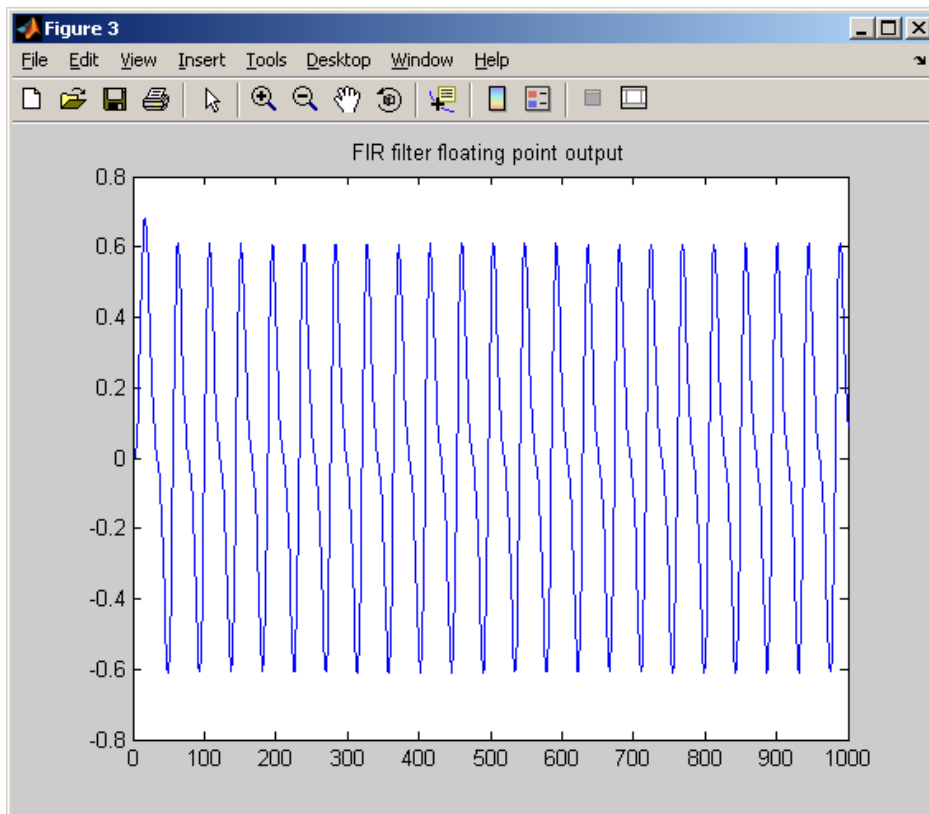


Figure 12 Monitor for FIR output signal

Architectural view

The architectural view is a refinement of the functional model. It is a system prototype with hardware and software components. At this level the partitioning of SystemC modules in hardware and software is defined manually. For the hardware components, cycle time is added and signals between the hardware modules are defined. The architectural platform is used for design space exploration concerning performance, communication and resource handling.

The architectural view can be used for early software validation and is a streamlined model used by software developers to execute software before the final hardware is available. Performance estimation of the complete system response can be evaluated in terms of cycle time approximation. This is performed by adding hardware signals to the model including clock, reset, handshake and bus signals.

The programmers view (PV) should normally simulate a bit accurate memory map. In the case of the FIR filter example, the coefficients calculated by high pass and low pass filter classes are mapped to a fixed point implementation.

In this example a RTL version of the FIR filter is implemented. Timing signals are added to the model in terms of a processor clock (100 mhz) and sample clock (44.1 kHz). The source for these signals is not shown in the UML model below.

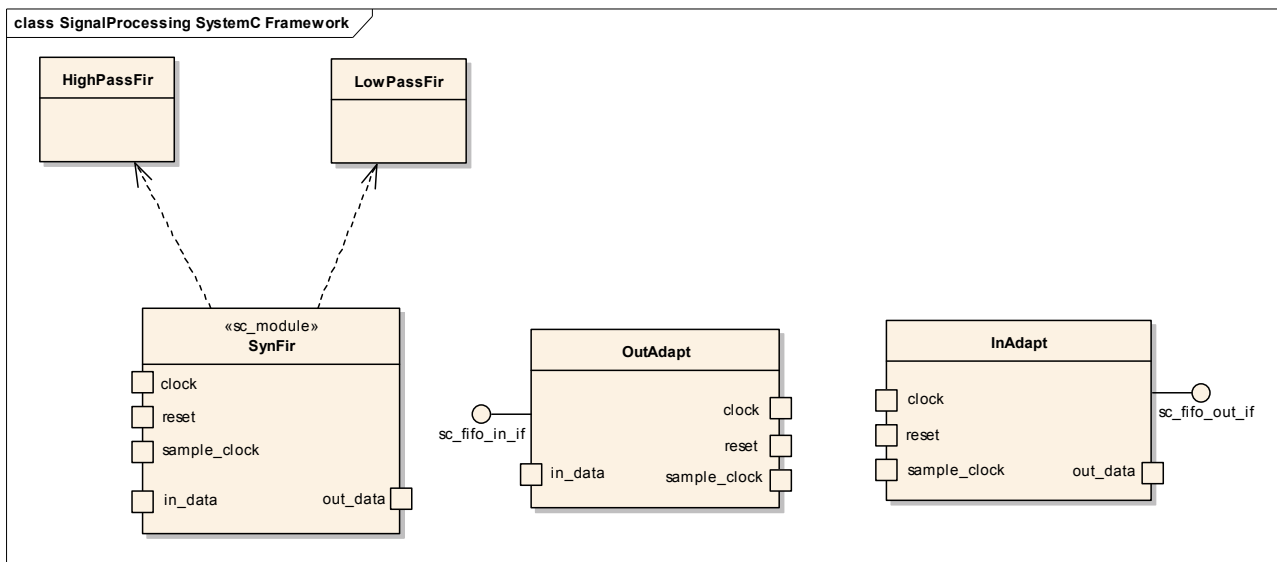


Figure 13 Architectural Signal processing modules

The output and input adapter implements the `sc_fifo_in_if` and `sc_fifo_out_if` interfaces and is used to connect and reuse all modules designed in the functional model. The `SynFir` module uses the same low and high pass filter calculation classes as created in the functional model. The data from the input adapter is clocked into the `SynFir` filter by the rising edge of the `sample_clock`. The output sample from the FIR filter is clocked on the falling edge of the `sample_clock`. The FIR calculation

is based on a one sample processing approach. It would be possible to experiment and use a buffered approach by implementing another type of adapters.

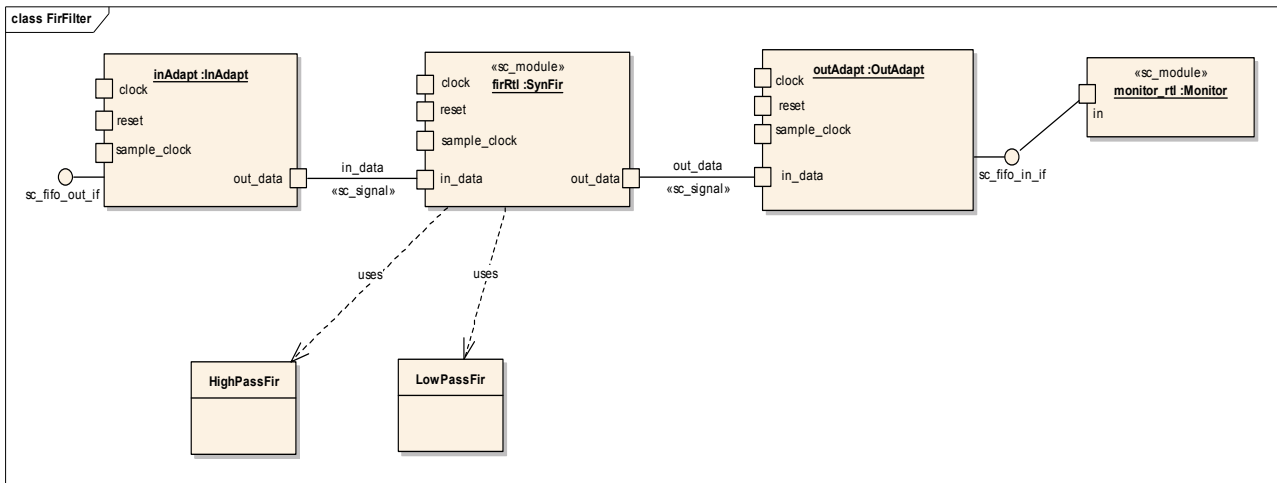


Figure 14 Completed architectural model

The list below shows the source code for the header file of the FIR filter's architectural model. In this case the fir_T type definition is replaced by a fixed point value of 18 bits wide with 2 decimal bits. The ArchFir module is added and created in the same top module as in the functional model.

```
typedef sc_fixed<18,2> fir_T;

class ArchFir : public sc_module
{
public:
    // Input and output fifo adapters
    InAdapter<fir_T, ALGO_BITS> inAdapt;
    OutAdapter<fir_T, ALGO_BITS> outAdapt;

    // Creation of clock, sample_clock and synthesized signals
    sc_clock clock;
    sc_clock sample_clock;
    sc_signal<bool> reset;

    sc_signal<sc_int<ALGO_BITS> > in_data;
    sc_signal<sc_int<ALGO_BITS> > out_data;

    // Coefficients
    sc_signal<sc_int<WBUS> > coeffs[TABS];

    // Timed synthesis version of FIR filter
    SynFir firRtl;

    // Monitor of filtered output result from RTL level
    Monitor<fir_T> monitor_rtl;

    sc_trace_file *tracefile;

    SC_HAS_PROCESS (ArchFir);
};
```

```

ArchFir(sc_module_name name,
        FirCoeff_if<double> *pCoeff,
        int samples);

~ArchFir();

private:
    FirCoeff_if<double> *pFirCoeff;
    int m_samples;

};

class Top : public sc_module
{
public:
    .....

    // Architecture model of Fir filter
    ArchFir *pArchFir;

    SC_HAS_PROCESS(Top);
    Top(sc_module_name name);
};

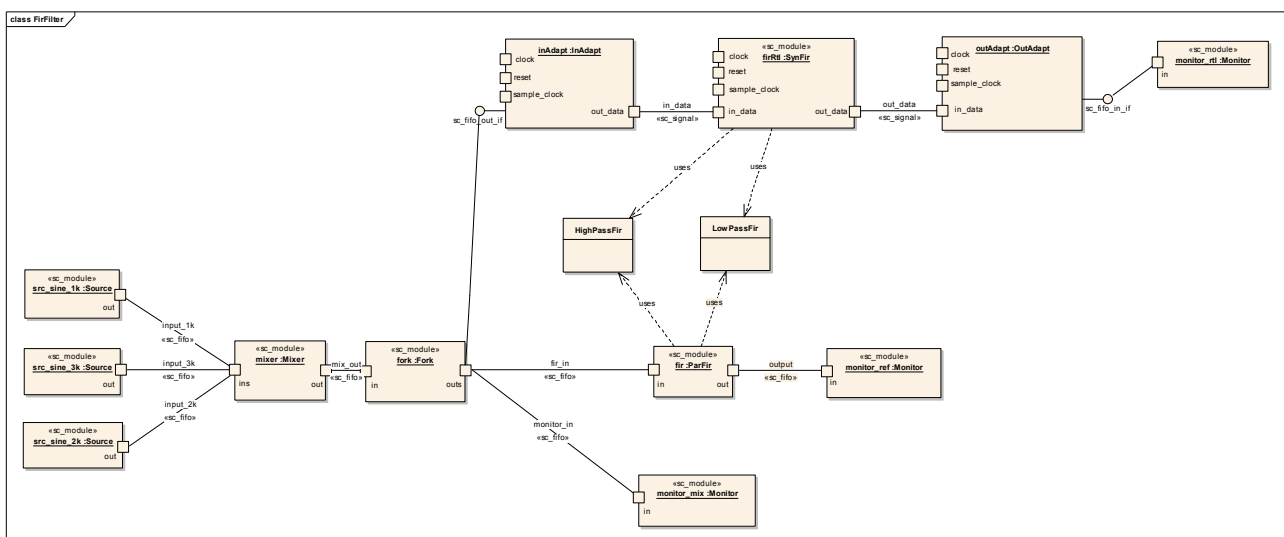
```

In the constructor of the top module, the adapters and RTL version of the FIR filter are created and connected to the output port of the Fork from the functional model.

```

#ifdef SC_ARCHITECTURE
    pArchFir = new ArchFir("ArchFir", pFirCoeff, samples);
    fork.out(pArchFir->inAdapt); // Connection to fork
#endif

```



Figur 15 Completed architectural model with functional verification

The completed architectural design is verified against the functional model used as the golden reference model. This example compares the difference between fixed point and floating point

results. Below is shown a screen dump for the simulated result of the architectural design compared to the functional model.

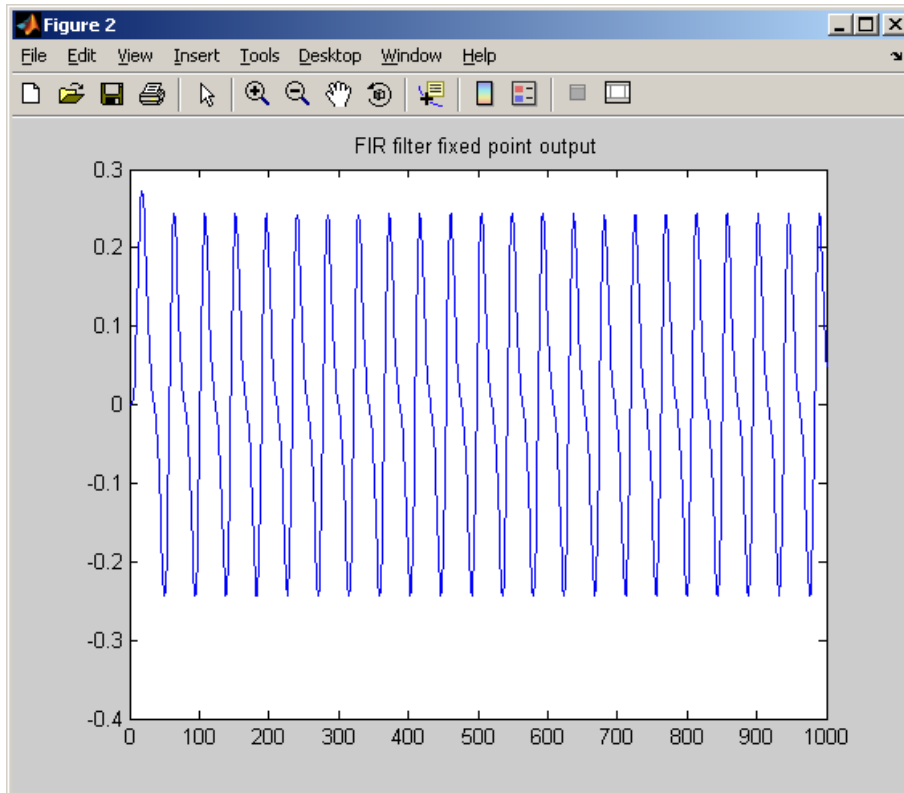


Figure 16 Monitor for architectural FIR output signal

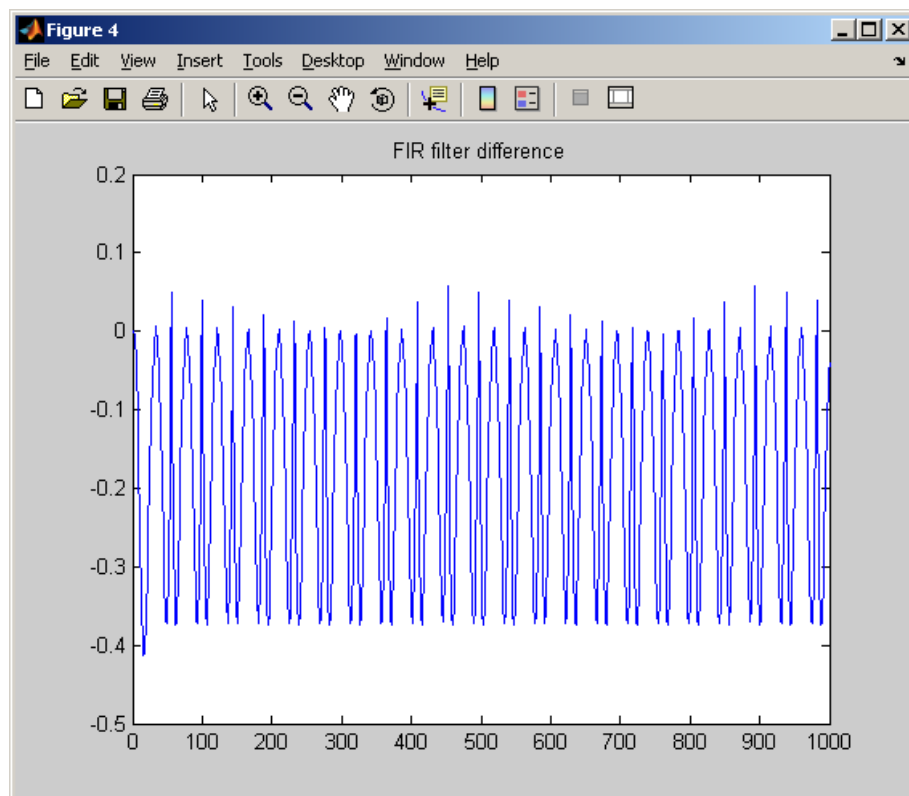


Figure 17 Monitor for difference between functional and architectural designs

The figure below shows the wave view of the signals added in the architectural design. It shows that the input data is ready on the falling edge of the sample clock, and that the output data is ready on the rising edge of the clock.

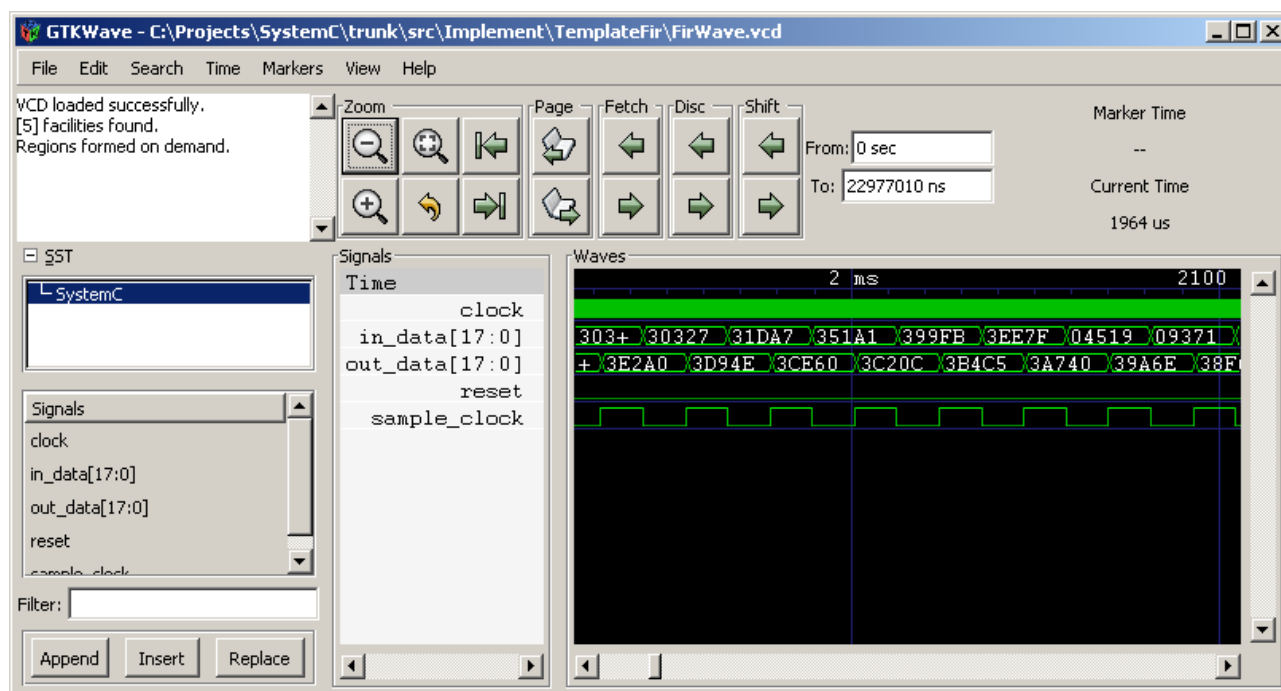


Figure 18 Wave view of parallel input/output data bus and sample clock in the architectural view

Implementation view

Most of the architectural model concerning the software part of the model can be used directly on an embedded platform supporting C++. In this case a Xilinx FPGA running a MicroBlaze softcore processor could be used. The low and high pass filter coefficient calculation classes should be reused running on this processor. A memory map should then be defined between the processor and the IP core of the FIR filter, and a mapping layer should be implemented for interfacing to the filter coefficient calculation code.

The hardware implementation of the FIR filter is carried out by translating the SystemC code. This is performed by using the SystemCrafterSC which is a tool capable of producing VHDL or Verilog code directly from a reduced set of SystemC. The auto generated VHDL code is simulated and verified at the gate level targeting the Xilinx FPGA. The other alternative would be to manually convert the SystemC architectural model of the RTL FIR filter to HDL code. ModelSim has been used to verify the VHDL gate level implementation against the architectural model with reuse of all SystemC modules. The same source code has been used as test bench for ModelSim where it is possible to mix between SystemC, VHDL and Verilog.

The final step would be to add the FIR VHDL code to an auto-generated IP core, include it in the processor design and test the final port on the Xilinx FPGA platform. In this case, the FIR IP core has to be connected to an external 18 bit AD/DA converter with a parallel interface.

The figure below shows the project setup in the Eclipse Platform with all source and test files. The model can be build for configuration of functional, architectural or gate-level model views. The same eclipse project has been used for the all views. Changing between views is carried out by selecting the build configuration within eclipse. The complete FIR code example and Eclipse project files can be downloaded from the homepage: <http://www.teknologisk.dk/23191>

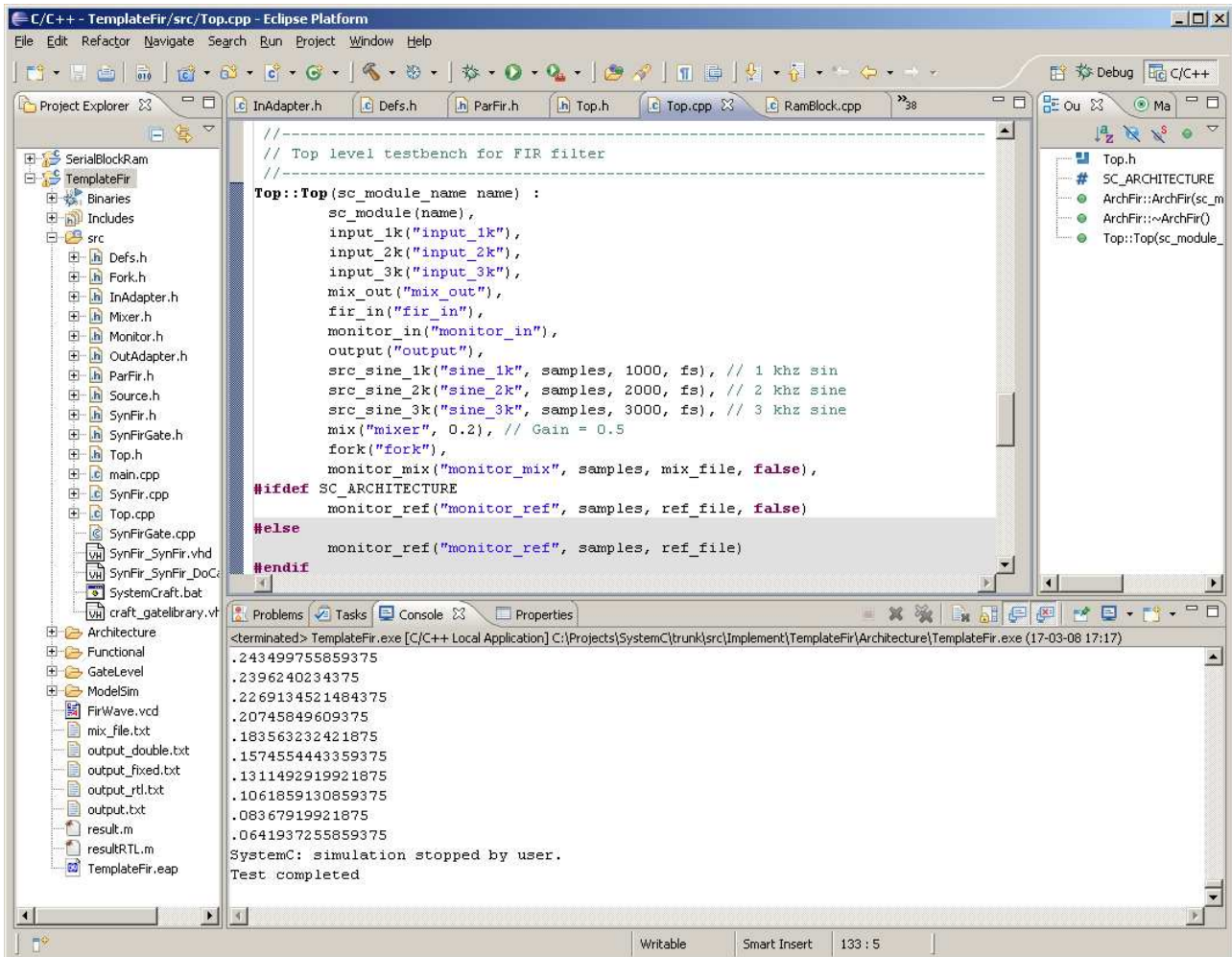


Figure 19 Eclipse environment used to execute the specification for functional and architectural views

References

The following books will serve as a good starting point for learning about SystemC and using TLM.

“SystemC: From the Ground Up” by David C. Black and Jack Donovan

This book is good for learning SystemC from a system-level design point. It gives an introduction to the Transaction-Level Modelling (TLM) and the SystemC language including the architecture and C++ classes' library. It covers the newer version of SystemC 2.1 and requires good knowledge of C++. There are many SystemC examples for every chapter which can be downloaded from <http://www.eklektically.com/>.

“A SystemC Primer, Second Edition” by J. Bhasker

This book is good for learning SystemC from a register transfer level (RTL) design point. The book is focusing on making simulations in SystemC of digital electronics and is a good starting point if you are used to make implementations in Verilog or VHDL code. It only covers the TLM modelling features of the SystemC languages beyond RTL to some extent.

“System Design with SystemC” by Thorsten Grötter, Stan Liao, Grant Martin, Stuart Swan

This book is a good place to start if you have a basic knowledge of SystemC and would like to have a better idea of the different usage of language from classical hardware, TLM and functional modelling. The FIR example in this document uses some ideas from the book concerning parameterized designs, interfaces and channel designs and communication refinement.

“Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded System” by Frank Ghenassia

This book gives a good overview of designing System-on-Chip (SoC) including complex FPGA designs. It covers modelling and verification theories. It is not a book for how to programme SystemC but gives a good overview of areas like Transaction Level Modelling, Techniques, Embedded Software Development and Functional Verification in the area of modelling and simulation for developing SoC and advanced FPGA systems.

“Getting started with TLM2”, by John Aynsley, Doulos

A very good TLM2 tutorial which contains a series of 8 complete examples for OSCI TLM 2.0 draft standard. <http://www.doulos.com/knowhow/systemc/tlm2/>

“Getting started with SystemC”, by Kim Bjerger, Danish Technological Institute

This is a guide for installing and setting up a SystemC platform for Eclipse or Visual Studio running on Windows or Linux. <http://www.teknologisk.dk/21871>