Refinement of Mixed-Signal Systems with SystemC

Ch. Grimm, Ch. Meise, W. Heupke, K. Waldschmidt
University Frankfurt, Professur Technische Informatik
Robert-Mayer-Strasse 11-15; 60054 Frankfurt, Germany
grimm@ti.informatik.uni-frankfurt.de

Abstract

This paper gives an overview of a design methodology specific extension library for SystemC. The supported design methodology successively refines an executable specification to a concrete mixed-signal architecture. A first prototype, the ASC library, has been implemented and evaluated.

1. Introduction

Extensions for the modeling and simulation of analog and mixed-signal systems are discussed in the "SystemC-AMS Working Group" ([1]). These shall provide an open platform for behavioral modeling and executable specification of signal processing systems and the coupling of different analog simulators. In this paper, we describe the refinement of mixed-signal systems using a very first prototype for SystemC-AMS (ASC-library).

The classical top-down design flow, using for example VHDL-AMS starts with an executable specification in the form of block diagrams. Within the design process, idealized blocks are replaced by more realistic models, which usually introduce different interfaces: Abstract signals are replaced by analog signals or bit vectors, and ports are added that control the execution.

The object oriented modeling enabled by C++ allows us to add features to the executable specification, refining it sucessively by small steps to a detailed model. This refinement process preserves the compatibility of the interfaces used for modeling communication on different levels of abstraction. This compatibility directly supports mixed-level simulation and re-use of models.

2. Refinement of Mixed-Signal Systems

We distinguish three levels of refinement: The design starts with an *executable specification*. The executable specification is a block diagram with continuous-time semantics and directed signal flow between the blocks modeled using C++. The behavior of each block is specified by dynamic linear functions (transfer functions that specify filters or controllers) and algebraic nonlinear functions.

By the *refinement of computation*, the executable specification is augmented by methods and parameters, that specify *how* the intended behavior is computed efficiently and with sufficient accuracy. For an analog implementation, no further refinement is required – the realization is just a physical system whose behavior is "analog" to the equations. For a digital implementation (hardware or software), we determine appropriate sample rates and bit-widths. Furthermore, converters (a/d, d/a, sample-rate converters (src)) must be added. We call the resulting model a *computation-accurate model*.

In a second step, communication and synchronization are bound to concrete physical signals. The basic idea of this refinement is already known from SystemC design methodology: Adapter classes translate the abstract interface to a concrete, pin- and cycle accurate interface. We call the resulting model a *pin-accurate model*. This model is the starting point for circuit-level design (analog modules) or Hw/Sw Codesign (discrete modules).

Note, that the structuring in the above mentioned design steps is just for illustration – the refinement is done by small steps, that are validated separately and that can be done in arbitrary order or in parallel by different designers.

3. The ASC library

The ASC library provides an "analog" or signal processing process type. The execution of analog processes is not controlled by the discrete event kernel. The execution of such processes is controlled by a coordinator interface. Via this interface, a coordinator can call the signal processing C++ methods via remote method invocation. As a very simple example, we can model an analog integrator by the following C++ method in a module:

```
void compute_integ() {
```

1530-1591/03 $17.00

```
    state += x*dt.to_seconds();
    y = state;
  }
```

This method is declared as an analog process:

```
ASC_SIG_PROC(compute_integ);
```

More complex analog processes could be external analog simulators, such as Saber or Spice, for example. In order to be able to simulate the analog process, signal processing methods are bound to coordinators, that control the execution of clusters of analog processes. A coordinator is an object, which realizes the abstract coordinator interface (`asc_coordinator_if`). The binding of analog processes to coordinators can be made either "fixed" (for distribution, re-use), or can be left open.

"Analog" processes communicate via "analog" ports and signals. Analog signals realize the interface used in discrete-event simulation (`sc_signal_inout_if`) *and* an interface for the synchronization of analog processes (`asc_signal_if`). By this multiple inheritance, the signal class can as well be read or written by discrete event process, without the need for an explicit converter (as needed, for example in VHDL-AMS between Quantities and Signals). This allows us to connect as well digital modules to an abstract "analog" signal, and to refine the abstract signal to an analog signal, an A/D converter and a digital signal. Analog ports map the abstract interface functions to the functions of an instanciated signal. Doing that, the ports decide, which synchronization mechanism is applied: Coupling of analog processes among each other or coupling of analog and discrete event processes.

4. Design of a PWM controller

The ASC library and the refinement have been evaluated in a case study by designing a PWM controller. This controller controls a continuous voltage on a load by switching a transistor either on or off. The load in this example is a resistor in parallel with a capacitor. The system computes the difference between the actual voltage $U(C)$ and a programmed voltage $Uprog$. The difference is input for a PI controller. A pulse former creates pulses, whose width is proportional with the output of the PI controller.

An executable specification was created using the predefined modules `asc_s_pi_controller` specifying the PI controller and `asc_s_partial` modeling the CMOS switches with the load. The behavior of the pulse former is specified by a discrete SystemC thread:

```
void pulse_generator() {
  do {
    on_off = 1; wait(sc_time(ctrl, SC_US));
    on_off = 0; wait(sc_time(255-ctrl, SC_US));
  } while (true);
}
```

We started simulation with small step widths, which were reduced subsequently to 255 μs. This reduction was only possible with further refinements. First, we added an A/D converter model to the signal $U(C)$. Second, we refined the schedule of the computations: Instead of unnecessary high oversampling, we defined a precise cycle: First, the power switches are enabled, after a certain "on-time", the converter is requested to sample, after the conversion time, and then the PI controller computes one cycle. These refinements were done in small steps, that were validated separately. We started by reducing the step width of the coordinator `clk` to zero and by triggering the execution explicitely.

Later, we refined this method to a controller, which uses enable and clock signals as shown in figure 1 to control the execution of the PI controller, the converter, and to generate the pulses. Note, that the refined controller also includes a counter, which reduces simulation performance by magnitudes in exchange for modeling communication pin accurate.

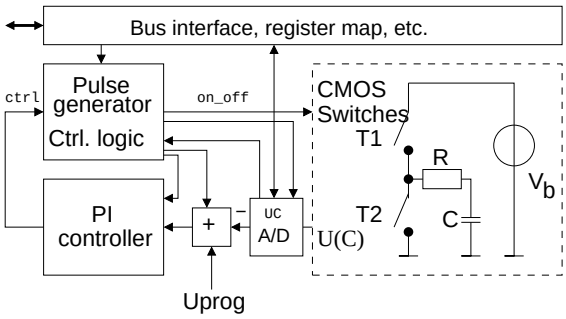

Figure 1. Refined PWM model.

Figure 1 gives an overview of the refined model, that was the starting point for the design of the single modules on RT-level (the PI controller using Coware synthesis tool). Note, that the system shown in figure 1 still provides all signals and abstract communication interfaces that were used in the executable specification – this allows us to configure less accurate but "faster" models: In order to get a C++ model, that can be used in software development, only cycle precise communication on the bus interface is required. Therefore, we can easily replace the clock cycle precise controller and internal models by a more abstract and faster "computation-precise" model, which uses the coordinator interface instead of clocks and enable signals.

References

[1] A. Vachoux, C. Grimm, and K. Einwich. SystemC-AMS Requirements, Design Objectives and Rationale. In *Design and Test in Europe 2003 (DATE '03)*, Paris, France, 2003.