Sitar

Simulation Tool for Architectural Research

User's Manual

Neha V. Karanjkar Madhav P. Desai

Department of Electrical Engineering

Indian Institute of Technology Bombay

Abstract

Sitar is a tool for modeling and simulation of cycle-based systems. It consists of a system description language and a simulation kernel. A system in sitar can be described as a set of interconnected modules running concurrently on a single clock. The language supports hierarchical descriptions: modules can contain instances of other modules. The behavior of each module can be described in an imperative manner as a sequence of statements. The statements include constructs such as time delays, parallel blocks, branch/loop constructs, procedures and instantaneous code blocks. C++ code can be embedded in a module description at several places in a straightforward and well defined manner. Each module description gets translated to a C++ class. The generated code can be compiled and linked together with the simulation kernel to get a single simulation executable. The simulation kernel is lightweight, consisting of a small set of C++ classes, and supports parallel execution using OpenMP. The simulation algorithm uses two-phase execution : each clock cycle is divided into two phases, and input/output events are restricted to separate phases, leading to a race-free execution.

Contents

1 Basic Components

A system is composed of *modules* and *nets*. Modules represent behavioral entities in the system, and nets are channels of communication between them. All modules run concurrently on a single clock. A module can communicate with another module by transfer of *tokens* (which are packets of information) over nets. Nets provide finite-capacity FIFO buffering for tokens. A module's interface to a net is called a *port*. A port can either be an *inport* or an *outport*. Each net is connected to exactly one inport and one outport. Figure 1 shows an example of a system with three modules.

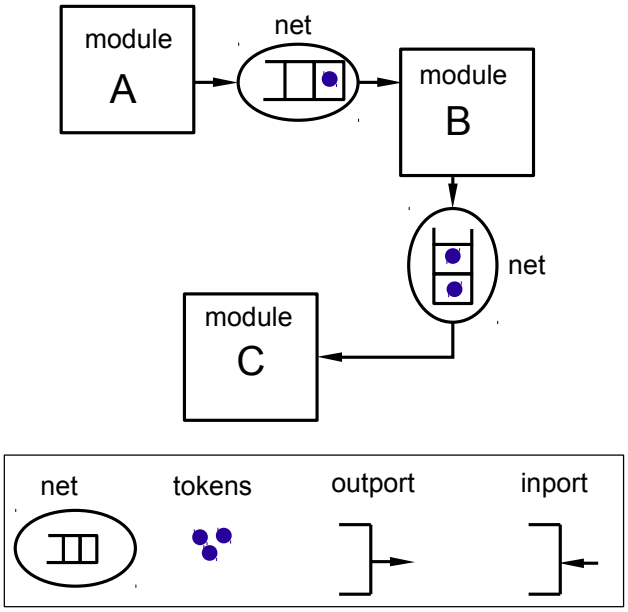

Figure 1: Example of a system in sitar

- Tokens have a `width` parameter, which represents the number of bytes of data that are carried by the token.
- A net has two parameters: `width` and `capacity`. A net with width W can only contain tokens of width W. The capacity of a net is the maximum number of tokens it can contain at a time.
- Ports have a single parameter `width`. A port with width W can only be connected to a net of width W.

2 The Execution Model

Sitar uses a two-phase execution model. Each clock cycle is divided into two phases : *phase-0* and *phase-1*. A module is allowed to perform input in phase-0 only, and output in phase-1 only. Thus, reads and writes to a net can never occur simultaneously and the execution is deterministic. As a consequence, the propagation of information from one module to another incurs a delay of at-least one clock cycle[1]. This restriction leads to a very simple simulation algorithm that is easily parallelized. During each phase, the behavior of each module in the system is executed exactly once. The order of execution among modules does not matter. The simulation algorithm is as follows:

```
cycle = 0
while (cycle < total_cycles)
{
   phase=0
   for each module m :
        run m
   phase=1
   for each module m :
        run m
   cycle=cycle+1
}
```

Within a phase, modules can be executed independently of each other. Thus the execution of individual modules can be mapped to separate threads running in parallel and synchronizing at the end of each phase.

3 System Description Language

3.1 Basic Design Units

The basic design units in a sitar description are *modules* and *procedures*. Modules represent structural entities in a system. They have a structure as well as a behavior. Procedures are akin to functions in C. They represent a set of actions that can be invoked by modules. A module or a procedure can be defined once and instantiated multiple times. Each definition must be contained within a single file. However a single file can contain multiple module/procedure definitions.

The system can be described in a hierarchical manner, with modules containing instances of other modules. The description must always contain a single

[1] The system can thus be viewed as a set of interconnected Moore machines

module named “Top” which represents the top-level module in the structural hierarchy, and contains all other modules in the system. This top-level module is instantiated during simulation. The following is an example of a system where the Top module contains two instances of a module ‘Foo’.

```
module Foo
end module

module Top
    submodule f1,f2 : Foo
end module
```