SystemAda: An Ada Based System-Level Hardware Description Language

Negin Mahani, Parnian Mokri, Mahshid Sedghi, and Zainalabedin Navabi

Nanoelectronic Center of Excellence
Electrical and Computer Engineering Department
Faculty of Engineering, Campus #2
University of Tehran,
14399 Tehran, IRAN
{negin, parnian, mahshid, navabi}@cad.ut.ac.ir

Abstract

Recent research on system level design has produced a new level of abstraction for description of hardware that is referred to as Transaction Level Modeling, or TLM. TLM separates hardware into computation and communication units and describes each at a very abstract level. . Another important property of electronic circuits, which is also modeled well in TLM, is concurrency of tasks and operations. Inherent concurrency of Ada, makes this language a good candidate for design and description of modern electronic systems. This paper[*] describes how to use Ada as a system description language like SystemC, and will use Ada compilers (such as Gnat) to evaluate systems described using Ada. We refer to the adaptation of Ada for system level description as SystemAda. This paper reviews Ada programming language requirements for modeling behavior of a digital system at transaction level, and considers possible approaches for extending Ada to meet these requirements.

1. Introduction

Increasing complexity of today's electronic systems has obliged system designers to come up with high level description languages for system modeling. Today, Transaction Level Modeling (TLM) is becoming a common way of simplifying system level design and architecture exploration, allowing designers to focus on functionality of the design and to be free from Register Transfer Level (RTL) details [1]. At this level of abstraction, a component either refers to a communication component, called channel, or a computation one.

For design at the transaction level, different Hardware Description Languages (HDLs) have been developed based on application focus, tool environment, and correspondence with hardware [2]. A TLM language must meet two major requirements to be appropriate for hardware/software co-design as the first step in TLM design process [3]. These requirements are: the ability to describe complex hardware, and software development. SystemC, which is a library of C++ classes, is currently adopted as a TLM language by design community.

A study of the characteristics of Ada language shows that it can also be used as a TLM language. An important characteristic of the Ada compilers is that they can catch a big part of the errors that C compilers would miss.Inherent concerency of the Ada is also one of its outstanding advantages over C++. Ada has a high-level concurrency model whereas C/C++ programmers must use external libraries to mimic concurrency [4, 5]. In addition, Ada extensively supports multithreading and multiprocessing, while C/C++ programmers have to rely on additional patches [4, 5].

Predefined services in Ada 2005 make it possible to trigger events at a specified time or when a specified amount of CPU time has been consumed by a thread. Another advantage of Ada involves invoking threads as well as the notion of synchronized interfaces similar to those in Java, which specify synchronization properties. These features help linking object-oriented programming to real-time activities [4, 5, 6].

In this paper, we introduce SystemAda as a system description language for describing hardware systems. This work examines two aspects of SystemAda: it has to be able to describe hardware at the transaction level (TLM), and it must provide a link to RTL and incorporate RTL descriptions. For TLM, we describe

[*] This work has partially been funded by a grant from Iran National Science Foundation (INSF).

TLM FIFO as the basic TLM 1.0 channel using Ada and show how other TLM channels can be described using this channel. For RTL, we explain a method for linking Ada to RTL.

Section 2 contains an overview of Ada as an HDL. In Section 3, major requirements for SystemAda as a TLM language are explained. This section starts with introduction of a method for linking Ada to RTL, and then presents an overview of TLM channels and implementation of their functionality in Ada. Section 4 describes TLM-FIFO application in a master-slave architecture in Ada using tasks of this language. Finally, Section 5 concludes the paper.

2. The history of Ada as an HDL

Most hardware developers use an HDL to design hardware components, and a programming language, usually C or C++, for simulation and testing. With Kernel Ada, design can be integrated with testing, and an iterative design process can be greatly simplified [7]. The idea of using Ada as an HDL dates back to 1980's. At that time, Ada compilers were time-consuming and design abstraction level was at transistor level, making Ada useless for hardware description. Later, by migration of design to higher abstraction levels, it was proved that Ada83 can be used as an HDL for design at gate level.

In 1995, a new version of Ada called Ada95 was defined with object oriented features which were suitable for high level design such as TLM [8]. Considering Ada as the basis of the VHDL language, which is one of the most popular HDLs especially for complex hardware description at RTL, it will be more favorable to use it as a TLM description language core.

By exploiting the inherent concurrency of Ada and making use of its functions and constructs, we can describe hardware components using Ada.

Figure 1 shows how a simple multiplexer is described in Ada. The operation of the multiplexer is described as an Ada procedure. The package body (the lower part of the code) implements the multiplexer Boolean operation using basic logical operations.

As mentioned before, software development is an important part of design process in TLM and using Ada in this design step has the following advantages:

- Reduction of debugging time [10]
- Ada compilers show many design and semantics errors early in the compilation process
- Problem (exception) handling mechanism [10]

```
package multiplexer is
  subtype input is Boolean;
  subtype output is Boolean;
  mux_in : input := true;
  data1 : input := true;
  data2 : input := false;
  mux_out: output;
  mux_in_invert : input := false;

  procedure mux (mux_in : in input;
                 data1 : in input;
                 data2 : in input;
                 mux_out : out output);
end multiplexer;

package body multiplexer is
  procedure mux (mux_in : in input;
                 data1 : in input;
                 data2 : in input;
                 mux_out : out output) is
  begin
    mux_out := mux_in and data1;
    invert(mux_in , mux_in_invert);
    ...
    ...
  end;
end multiplexer;
```

Figure 1. A multiplexer spec and body package

3. Major requirements for SystemAda

Having the link to RTL and the ability to describe TLM channels are the major requirements of a TLM language. In order to provide a link between Ada and RTL, an HDL package containing primary RTL components is developed in Section 3.1. Since communication is the main focus of TLM, and all of TLM channels have been defined based on TLM FIFO channel, a TLM-FIFO channel has been described and instantiated in Section 3.2.

3.1. Linking to RTL

Providing an HDL package containing primitive logic components described in Ada is an appropriate way of linking Ada to RTL. Based on SIGADA kernel Ada project, we have implemented an HDL package which contains a subtype Boolean input and output, one and two dimension buses of Boolean arrays and primary logical operations which will use arrays as a data storing structure. This way, other components can be developed more easily and in a flexible fashion. Figure 2 demonstrates the specification of this package.

```
package HDL is
  subtype input is Boolean;
  subtype output is Boolean;

  type bus is
    array (natural range <>) of Boolean;
  type dimesion2_bus is
    array (natural range <>,
    natural range <>) of Boolean;

  procedure invert (xin : in input;
                    xout : out output);
  procedure and_bit (x1 : in input;
                     x2 : in input;
                     xout: out output);
  ...
end HDL;
```

Figure 2. Outline of our HDL package

3.2. Describing TLM channels using Ada

In this section, we show the implementation of TLM channels using Ada.

3.2.1. TLM-FIFO channel

FIFO is the channel based on which other TLM-1.0 channels are defined. The FIFO channel is generic in size and type. By using this feature, we define a generic FIFO channel in Ada to be used later for defining other TLM channels. The *GENERIC* keyword in Ada has the functionality of *TEMPLATE* in C [9].

Since in TLM data transmitted between components does not have any limitation in size and can be of any type, we use generic types in Ada for FIFO nodes to simulate these capabilities. Figure 3 shows the code of TLM FIFO channel described in Ada.

```
Generic TYPE fifo_element IS PRIVATE;
PACKAGE fifo IS
  input : fifo_element;
  output : fifo_element;
  empty_flag : boolean;
  TYPE fifo_node;
  TYPE fifo_channel IS ACCESS fifo_node;
  TYPE fifo_node IS RECORD
    data : fifo_element;
    link : fifo_channel;
  END RECORD;
  Head : fifo_channel;
  PROCEDURE add_fifo;
  PROCEDURE rem_fifo;
   ...
END fifo;
```

Figure 3. Generic FIFO

The program in Figure 4 shows how an integer FIFO can be instantiated.

```
with Ada.Text_IO;
use Ada.Text_IO;
with fifo;
package int_fifo is new fifo(integer);
```

Figure 4. Generating an integer FIFO

3.2.2. Other TLM Channels in Ada

To implement other TLM channels, hierarchical packages should be used. This allows defining new channels that are based on the FIFO channel.

Figure 5 shows hierarchical package format in Ada 95. Here, the package named *outer* contains two inner child packages and their bodies.

```
package outer is
  package inner_1 is
  ...
  end inner_1;
end outer;

package outer.inner_2 is
...
end outer.inner_2;
```

Figure 5. Hierarchical package format in Ada 95

By defining a generic FIFO and using hierarchical packages, other channels based on FIFO can be described. Figure 6 shows the way we have developed this concept.

```
with Ada.Text_IO;
use Ada.Text_IO;

Generic package fifo.channel2 is
  --channel2 extra functions
end fifo.channel2;
```

Figure 6. Describing a channel based on FIFO

channel2 shown in this figure has its own body and implementation and is also generic. The FIFO channel can be used to implement other kinds of channels.

4. Master-slave architecture in TLM

We have modeled a master-slave architecture using Ada. A master module puts data into a TLM FIFO channel and the slave gets the data from the FIFO and processes it. We have used the concept of tasks in Ada which are introduced in the following subsection.

4.1. Task overview

Tasks are the basic elements for implementing concurrency in Ada. Each Task can communicate with other tasks and will proceed until a specified delay, and works as though it is running on a separate computer. This is achieved by the *entry* concept which defines what information should be sent when a task is required, and what should be done. Tasks can be sensitive to activation of one or more entries [2, 8, 11]. Some of the features of tasks are as follows [8]:

- Waiting for other tasks to complete.
- Sending messages between each other (by using entries) called rendezvous
- Setting global variables to communicate.

Like packages, tasks have a declaration and a body. A task body defines what the task will do when it starts up. SELECT and ACCEPT statements together mark alternative entry points for messages into a task [13]. The SELECT block can contain many ACCEPT statements, separated by the reserve word "OR". Messages sent to the receiving task are processed in the select block in the order they are received [13].

4.2. Master-slave TLM model

The block diagram of TLM master-slave architecture is shown in Figure 7. There are two modules in the system: a Master module which adds numbers to an integer FIFO and a Slave module which removes numbers from the integer FIFO and processes them.

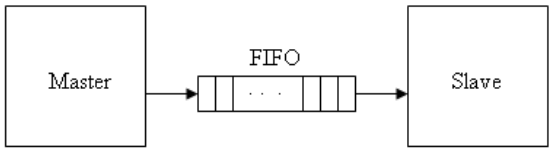

Figure 7. TLM master-slave architecture

The procedure of Figure 8 shows the TLM master-slave modeled in Ada using Ada tasks. Three tasks have been defined in the system: the first one, which is named *fifo_task*, performs the main operation of the FIFO through the entries *add* and *remove*, the names of which imply their functionalities. The other two tasks implement the functionality of master and slave modules. Since these modules start in non-conditional loops, this process will never end.

```
procedure TLM_Master_Slave is
  task type fifo_task is
    entry add;
    entry remove;
  end fifo_task;
  task body fifo_task is
  begin
    loop
      select
        accept add;
          add_fifo;
        accept remove;
          rem_fifo;
      end select;
    end loop;
  end fifo_task;
  fifo1:fifo_task;

  //task type slave definition
  //task type master definition

  slave1:slave;
  master1:master;
begin
  loop
    master1.start;
  end loop;
end TLM_Master_Slave;
```

Figure 8. Master-slave TLM modeled using task feature

Figure 9 shows the *Master* task. It has an entry named *start*. By accepting start in an infinite loop, it inserts a data into the FIFO and activates entry *start* for the *Slave* task.

```
  task type master is
    entry start;
  end master;
  task body master is
  begin
    loop
      accept start;
      ...
      fifo1.add;
      slave1.start;
    end loop;
  end master;
```

Figure 9. Master task

Salve is described in Figure 10. It also has an entry named *start*. By accepting start in a never-ending loop, it removes a data from the FIFO.

Although we have treated a simple example in Ada, but this clearly shows the power of this language for descriptions where communications happen through complex components, which is what characterizes the new system level design strategy.

```
task type slave is
  entry start;
end slave;
task body slave is
begin
  loop
    accept start;
      fifo1.remove;
      ...
  end loop;
end slave;
```

Figure 10. Slave Task

5. Conclusions

With designs getting more complex, the need for system description languages is more revealed. The first step in this level of abstraction is Hardware/Software partitioning. Ada with natural concurrency, early error detection, exclusive error description and extensive support for multithreading and multiprocessing would be a good choice for achieving this goal.

This paper introduces Ada as a TLM language by providing a link to RTL and also defining TLM-FIFO channel functionality in Ada. We also described how to develop other channels based on a single generic FIFO.

As a future work, we are going to complete the description of other necessary TLM features in Ada. We will also improve our HDL package in Ada to describe more basic RTL constructs, and finally we will use these features to implement some complex hardware components at the system level. This research parallels the work we are doing on defining languages and tools for designs at the system-level beyond RTL.

6. References

[1] T. J. Wheeler, "Embedded System Design with Ada as the System Design Language," 1984, Available: http://stinet.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA144232.

[2] L. Cai, and D. Gajski, "Transaction Level Modeling: An Overview," Proc. 1st IEEE/ACM/IFIP International Conference on Hardware/software codesign and system synthesis, 2003, pp. 19-24.

[3] Z. Navabi, *VHDL: Modular Design and Synthesis of Cores and Systems*, McGraw-Hill, 2007.

[4] R. Goering, "Ada 2005 speaks to real-time embedded application," 2007, EE Times, Available: http://www.embedded.com/news/embeddedindustry/198701828?_requestid=308128.

[5] J. Jackson, "The return of Ada," 2008, Government Computer News, Available: http://www.gcn.com/print/27_8/46116-1.html#;GCN Home.

[6] B. Brosgol, and R. Dewar, "Use Ada for Better Safety, Security, And Reliability," 2008, Electronic Design publishes, Available: http://electronicdesign.com/Articles/Index.cfm?AD=1&AD=1&ArticleID=18141.

[7] SIGAda Documents, 2007, Available: http://www.SIGAda.org/.

[8] Ada Reference Manual ISO/IEC 8652:1995(E), chapter 9, Available: www.adahome.com/rm95.

[9] J. G. P. Barnes, *Programming in Ada*, 3rd Edition, ???

[10] D. A. Wheeler, "Lovelace tutorial," Lesson 1-Brief Introduction to Ada, Available: www.dwheeler.com/lovelace.

[11] A. Burns, A. Wellings, and J. Barns, "Concurrency in Ada", 2ed edition, Cambridge University Press, 1998.

[12] S. Johnston, "Ada-95: A guide for C and C++ programmers," Available: http://www.adahome.com/Ammo/Cplpl2Ada.html.

[13] "Introductory Ada Concurrency Summary," 2007, Available: http://www.seas.gwu.edu/~csci51/fall99/ada_task.html