

System Level Hardware Design and Simulation with SystemAda

Negin Mahani, Parnian Mokri, Zainalabedin Navabi

Electrical and Computer Engineering Department, Faculty of Engineering, Campus #2

University of Tehran, 14399 Tehran, IRAN

{negin, parnian, navabi}@cad.ece.ut.ac.ir

Abstract

Recent research in the system level design field has produced a number of techniques for structuring the understanding of systems. Many of these techniques produce design structures that are easily expressible in Ada language. Ada language has a structure which allows the design of systems to be expressed independently of its implementation and thus can be a good system design language to use with these techniques [1]. This paper describes how to use Ada as a system description language like SystemC, which will use Ada compilers (such as Gnat) to evaluate the written program as a system; we have named it, SystemAda. This paper reviews Ada programming language requirements for modeling system behavior and structures at Transaction Level Modeling (TLM). This paper considers possible approaches for extending Ada to meet these requirements [2].

1. Introduction

Along with complex electronic systems, system designers have developed high abstraction level description languages for system modeling. Today, Transaction Level Modeling is becoming a common way of simplifying system-level design and architecture exploration, allowing designers to focus on functionality of the design and get rid of RTL details. At this level of abstraction, a component refers to a communication component, called channels, or a computation one. To achieve these benefits, different hardware description languages have been developed with different application focus, acceptance, and strength [2].

These languages have two major capabilities for hardware/software co-design as the first step in TLM design process [3]:

- Description of complex hardware
- Software programming development

Ada has always been strong in security-critical applications. Security is becoming more important in commercial embedded applications [4]. Ada compilers can catch many errors that C compilers would miss. Ada has a high-level concurrency model however

C/C++ programmers must provide it with the vast use of external libraries [4, 5]. Ada is a natural choice for an emerging challenge in embedded-system programming: multicore applications. Ada has extensive support for 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 area of improvement, involves dispatching of threads. In addition to preemptive, priority based scheduling; Ada 2005 supports non-preemptive, round-robin and earliest-deadline-first policies. A third area of improvement is the Java-like notion of synchronized interfaces, which specify synchronization properties. These features help linking object-oriented programming to real-time activities [4, 5, 6].

Since VHDL is based on Ada's first prescription, after 1995 renewed specification of Ada, many projects use it as a Hardware Description Language such as Alpha system at Fairleigh Dickinson University [7].

In this paper we introduce SystemAda as a System description language for describing hardware systems. The work is partitioned based on two distinct concepts: TLM concept and RTL concept. In TLM concept, we describe TLM-FIFO as the most basic TLM channel in Ada and show how other TLM channels can be described using it. In RTL concept, we explain and expand a method for linking Ada to RTL.

Section 2 contains an overview of Ada as a Hardware Description Language (HDL). In Section 3, major requirements for SystemAda are explained. Section 3.1 introduces an existing method for linking Ada to RTL and develops it with some new features. Section 3.2 contains an overview on TLM channels and their functionality implementation in Ada. Section 4 describes TLM-FIFO application in master-slave architecture in Ada Using Tasks. Finally Section 5 is the conclusion.

2. The history of Ada as an HDL

Most development organizations use an HDL to design a component, 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 goes back to 1980's. Ada compilers were time consuming and design abstraction level was at transistor level, making Ada useless for means of 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. Figure 1 shows how to describe a multiplexer using Ada functions.

```

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);
    mux_out:=mux_in_invert and data2;
  end;
end multiplexer;

```

Figure 1. A multiplexer specification and body packages

In 1995, a new version of Ada called Ada95 was defined with Object Oriented features which could be used for high level abstraction design such as TLM [8]. Considering Ada as VHDL base language, 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.

As mentioned before, software development is an important part of design process at TLM level. Using Ada in this step has the following advantages:

- Reduction of debugging time [10]
- Ada compilers show as many errors as reasonably possible, as early as possible [10]
- Problem (exception) handling mechanism [10]

3. Major requirements for SystemAda

In addition to the TLM abstraction layer, system level languages like SystemC also provide a link to

RTL. Therefore, there are two major functionalities to imply TLM features:

1. Linking to RTL
2. Describing TLM channels

Since communication is one of the most important concepts in TLM, and all the channels have been defined based on FIFO channel, in this paper a TLM-FIFO channel has been described and we have explained how to instantiate a channel based on developed FIFO. Also, in order to make a link between Ada and RTL, we have extended an existing HDL package containing primary RTL design concepts.

3.1. Linking to RTL

3.1.1. Writing primary HDL package in Ada itself.

One of the options to write primary HDL package in Ada is describing the primary RTL functionality in Ada. Based on SIGADA “kernel Ada project”, we have extended an existing 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. In this way, developing other components is more flexible. 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 dimeision2_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. Our evolved HDL package

3.2. Describing TLM channels

3.2.1. TLM-FIFO channel in Ada. Every channel in TLM is built on FIFO channel and is Generic in Size and type. As a result, we define a generic FIFO channel in Ada to be used later to define other TLM channels. GENERIC keyword in Ada has the functionality of TEMPLATE in C [9].

Since in TLM data transmitted between components don't have any limitations in size and can be of any type, we decided to use generic types in Ada for FIFO nodes to simulate these capabilities.

Figure 3 shows the code of our FIFO specification that can be used as a TLM channel.

```

generic type fifo_element is (<>);
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

A generic FIFO gives users the opportunity to determine the type of its elements. The program in Figure 4 shows how we define an integer FIFO:

```

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. Figure 5 shows hierarchical package format in Ada 95. The outer package contains two inner child packages and their bodies [12].

```

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, another channel 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 funstions
end fifo.channel2;

```

Figure 6. Describing a channel based on FIFO

“Chanel2” has its own body and implementation and is also generic. This FIFO can be used to implement other kinds of channels.

4. FIFO application in master-slave architecture TLM modeled in Ada

4.1. Task overview

Tasks are the basic elements for implementing concurrency in Ada. Each Task can communicate with other tasks and will proceed for a specified time and, works as though it is running on a separate computer. This quality is achieved by ENTRY context which defines what information should be sent when the task is required, and what should be done. Tasks can be sensitive to activation of one or more ENTRY [2, 8, 10,11]. Some of task capabilities are as follows [8, 10]:

- Waiting for other tasks to complete [8, 10].
- Sending messages among each other; this is called a rendezvous (by using entries).
- Setting global variables to communicate [8, 10].

Like packages, tasks have a declaration part and a body. A task body defines what the task will do when it start up [10].

SELECT and ACCEPT statements together mark entry points for messages into a task which are alternatives (or optional) [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 procedure of Figure 8 shows the modeled TLM master-slave in Ada using task capabilities with two modules:

- **Master module:** adds numbers to an integer FIFO.
- **Slave module:** removes numbers from the integer FIFO.

The block diagram of TLM master-slave architecture is shown in Figure 7.

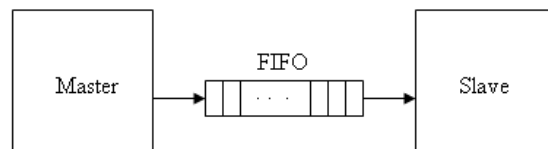


Figure 7. TLM master-slave architecture

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 is
  entry start;
end slave;
task body slave is
begin
  loop
    accept start;
    fifo1.remove;
    ...
  end loop;
end slave;
slave1: slave;
task type master is
  entry start;
end master;
task body master is
begin
  loop
    accept start;
    ...
    fifo1.add;
    slave1.start;
  end loop;
end master;
master1: master;
begin
  loop
    master1.start;
  end loop;
end TLM_Master_Slave;

```

Figure 8. Master-slave TLM modeled using task feature

As shown in Figure 8, our FIFO task has the ADD and REMOVE entries, whose names describe the functionality.

5. Conclusion

The need for system description languages which supports TLM is more revealed, since designs getting more complex. 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 to do this.

This paper introduced Ada as a TLM modeling language by defining TLM channels functionality in Ada and continues by implementing different channels in Ada. We also described how to develop other channels based on a single generic FIFO. One of the

options for linking Ada to VHDL is discussed. We have extended an existing HDL package to have more generalization in type and size of its elements.

As a future work, we are going to complete the description of other necessary TLM features in Ada. We'll also complete HDL package in Ada to describe more basic RTL concepts. We will use these to implement some of the most popular hardware designs.

6. References

- [1] Wheeler, T. J.; 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] Cai, L. and Gajski, D. "Transaction Level Modeling: An Overview", 2003. First IEEE/ACM/IFIP International Conference on Volume, Issue, 1-3 Oct. 2003; Available: ieeexplore.ieee.org/iel5/8991/28532/01275250.pdf; pp. 19- 24.
- [3] Zainalabedin Navabi; VHDL: Modular Design and Synthesis of Cores and Systems; McGraw-Hill Professional; 2007; 0071475451; pp. 22.
- [4] Richard Goering; Ada 2005 speaks to real-time embedded application ;(2007, 4, 2); EE Times; Available: <http://www.embedded.com/news/embeddedindustry/198701828?requestid=308128>
- [5] Joab Jackson; The return of Ada; (2008, 4, 14); Government Computer News; Available: http://www.gcn.com/print/27_8/46116-1.html#:GCNHome.
- [6] Ben Brosgol, Robert Dewar ; Use Ada for Better Safety, Security, And Reliability; (2008, 1, 7); Electronic Design publishes; Available: <http://electronicdesign.com/Articles/Index.cfm?AD=1&AD=1&ArticleID=18141>.
- [7] SIGAda Documents; (2007, 8, 6) Available: <http://www.SIGAda.org/>.
- [8] Ada Reference Manual ISO/IEC 8652:1995(E); ch 9. Available: www.adahome.com/rm95.
- [9] J.G.p. Barnes; Programming in Ada; third edition; ISBN: 0-201-17566-5.
- [10] David A. Wheeler; Lovelace tutorial; Lesson 1 - Brief Introduction to Ada; Available: www.dwheeler.com/lovelace.
- [11] Alan Burns, Andy Wellings, John Barns, "Concurrency in Ada"; 2 edition; 1998; Cambridge University Press; ISBN: 052162911X. ch 4.11.
- [12] Simon Johnston; Ada-95: A guide for C and C++ programmers; Available: <http://www.adahome.com/Ammo/Cplpl2Ada.html>
- [13] Introductory Ada Concurrency Summary; (2007, 7, 2) Available: http://www.seas.gwu.edu/~csci51/fall99/ada_task.html