NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

DATA FLOW DESCRIPTION WITH VHDL

by

Lo, I-Lung

December 1990

Thesis Advisor: Chin-Hwa Lee

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol: 62	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (city, state, and ZIP code): Monterey, CA 93943-5000	7b Address (city, state, and ZIP code): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (If Applicable)	9 Procurement Instrument Identification Number
8c Address (city, state, and ZIP code)	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No
11 Title (Include Security Classification): DATA FLOW DESCRIPTION WITH VHDL	
12 Personal Author(s): Lo, I-Lung	
13a Type of Report: Master's Thesis; 13b Time Covered: From To	14 Date of Report (year, month,day): December 1990; 15 Page Count: 125
16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.	
17 Cosati Codes: Field; Group; Subgroup	18 Subject Terms (continue on reverse if necessary and identify by block number): W-4 COMPUTER, PC, TAR, RAM, ACC, ALU, B_REG, IR, CONTROLLER, TEST_BENCH, VHDL.

19 Abstract (continue on reverse if necessary and identify by block number

The purpose of this research is to apply the VHSIC Hardware Description Language (VHDL) to the Data Flow design of a simple W-4 computer. Two of the three description views of VHDL will be discussed in this research. One is the behavior description of the W-4 computer in VHDL, and the other one is the data flow description. Both models will be discussed in detail. The basic concepts and significant features of VHDL will also be shown here with the experimental results. The objective is to verify a data flow design of a computer in terms of its functionality and timing behavior. The data flow model studied here can be synthesized into a structural model in gates.

20 Distribution/Availability of Abstract: [X] unclassified/unlimited [ ] same as report [ ] DTIC users	21 Abstract Security Classification: Unclassified
22a Name of Responsible Individual: Chin-Hwa Lee	22b Telephone (Include Area code): (408) 655-0242; 22c Office Symbol: EC / Le

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted

All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

Data Flow Description with VHDL

by

Lo, I-Lung
Commander, Republic of China Navy
B.S., Chinese Naval Academy, 1976

Submitted in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1990

Author: _____
Lo, I-Lung

Approved by: _____
Chin-Hwa Lee, Thesis Advisor

_____
Mitchell L. Cotton, Second Reader

_____
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

The purpose of this research is to apply the VHSIC Hardware Description Language (VHDL) to the Data Flow design of a simple W-4 computer. Two of the three description views of VHDL will be discussed in this research. One is the behavior description of the W-4 computer in VHDL, and the other one is the data flow description. Both models will be discussed in detail. The basic concepts and significant features of VHDL will also be shown here with the experimental results. The objective is to verify a data flow design of a computer in terms of its functionality and timing behavior. The data flow model studied here can be synthesized into a structural model in gates.

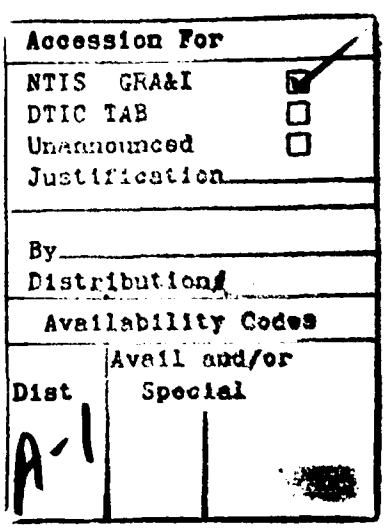


TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

First, I would like to give my great thanks to Professor Chin-Hwa Lee for his patience, support, and encouragement, especially for his guidance in the whole work. Also I want to thank Professor Mitchell L. Cotton who has provided valuable comments on my thesis. Finally, thanks go to Mr. Dan Zulaica for his assistance in the lab.

I am most grateful to my wife, Chun-Tieh, and my parents, whose love made all things seem possible throughout my studies at the Naval Postgraduate School.

I. INTRODUCTION

A. OBJECTIVE

The main purpose of this thesis is to apply the VHSIC Hardware Description Language (VHDL) to the Data Flow design of a simple computer, W-4, as well as to show why it is called a top-down design language, and how it works.

B. WHAT IS VHDL ?

VHDL is a new language to be used in the design and description of digital electronic chips, boards, and systems. VHDL opens the road for designs with the same description language used from the top system down to the gate level, and allows the designer to design the device independently by using the same consistent tools and standard descriptions, which was released in 1987 as the "IEEE Std 1076-1987" by IEEE.

C. WHY USE VHDL ?

The older Hardware Description Languages such as CDL, ISP, and AHPL have been used for the last 10 years. Because their timing modeling capability is not precise and can not handle complex hierarchical hardware structure, those language are out of date.

The newer language, VHDL, has more universal timing models, has a high degree of accuracy, implies no particular hardware structure, and is an executable and portable documentation language. It is likely to be the most important electronic design language in both the commercial and the military electronic design areas for the future.

D. WHAT IS A DATAFLOW DESCRIPTION ?

When viewing a design at a higher level of abstraction above the gate level, it is often convenient to discuss activity in a system in terms of directions of Data Flow. Data Flow describes the network of signals among functional units such as Arithmetic Logic Unit(ALU), Program Counter(PC), Temporary Address Register(TAR), Accumulator(ACC), Instruction Register(IR), control, and memory. Each part of the Data Flow description can be explicitly described in a VHDL text representation, just as in many RTL languages.

E. WHAT IS THE SCOPE OF THIS THESIS ?

W-4 is a tutorial and demonstration 4-bit machine with accumulators for arithmetic operations. The memory space of the machine has 16 locations. VHDL will be used in the description of this existing machine to check out the original design. This paper machine written in VHDL will also be simulated to find the limitation of its performance.

The objective is to study the VHDL design methodology in describing the W-4 computer, explore and investigate mixed level simulation involving models of different complexity levels, and show the advantage of the incremental development of various function models in the design.

This thesis addresses the following issues:

- First, to know about the operations of the W-4 computer, especially to know how the data transfer occurred in the system.
- Second, to become familiar with the VHDL language.
- Third, to describe the W-4 computer in VHDL and investigate the advantages and disadvantages in this modeling process.
- Fourth, to gain experience with the data flow modeling approach in VHDL.

F. AN OVERVIEW OF THE THESIS

The thesis subjects are divided into five chapters.

The second chapter first introduces the basic concepts of the VHDL language. These are classes of object and data types. Next, it introduces the basic structures of the VHDL. It includes entities, architectures, packages, and functions. Then, the processes, attributes, as well as control statements are discussed. Finally, the three views of description in VHDL, behavior, dataflow, and structure are introduced. All structures and terminology discussed here will be used throughout the thesis.

Chapter III relates to the data flow implementation of the W-4 computer constructed by Professor John R. Ward [Ref. 1]. These include PC, ACC, B_register, IR, TAR, ALU, RAM, controller as well as test_bench which integrates and simulates the W-4 computer of the design system.

Chapter IV discusses the experiences gained in the use of this system modeling approach, such as the basic concepts, the concurrence and sequence, the initial value setting, time modeling and accuracy, as well as inertial delay.

The last chapter is a conclusion, which discusses the advantages and disadvantages, as well as the further extension of the system to a lower level structure design.

II. BASIC FEATURES OF VHDL

It is necessary to introduce some ideas about VHDL in order to understand the thesis development. This chapter will present a brief overview of some important features of VHDL, which will be used throughout the thesis. The first section describes some basic concepts, classes of objects, and data types. The second section states some basic structures including entities, architectures, subprograms, and packages. The third section shows the process, attributes, and control statements. The last section introduces the three primary modeling views of the VHDL behavioral, data flow, and structural [Ref. 2]. The examples in this section are used to explain and demonstrate the language features to show how they are different from the usual languages, and how they are alike.

A. BASIC CONCEPTS OF VHDL

1. Classes of Object

There are three classes of object in the VHDL language: constants, variables and signals. An objects is created when it is declared. They are discussed as follows:

a. Constants

A constant is an object whose value can not be changed. Each constant declaration gives the name of the constant, its type, and its value. The following examples show the declarations of a constant. The bold characters are reserved word.

```
constant PCE : r_bit := '1';
constant DATA : r_word :="1111";
```

Where the "r_bit" and "r_word" are types. The numbers in quotes after the symbols ':=', are values assigned to constants PCE, and DATA as initial values.

b. Variables

Variables are objects whose value can be changed, and can only be declared in a "process" (The process will be discussed later) which they are considered to be static meaning that the value of each variable can be held until changed immediately by a variable assignment statement with no time delay. Two examples of variable declarations and one variable assignment are :

```
variable ADDR_B : r-word := "1111";
variable INVL :  r_bit;
ADDR_B := "0011";
```

The first two lines declare the names, types, and initialization value. The variable assignment statements use the symbol ":=" to assign the value. If no value is specified in the declaration, the default value will be the initialization value. For example, for the variable INVL, the initialization value is '0'. The last line is a variable assignment statement. It assigns the value "0011" to the variable ADDR_B.

c. Signals

Signal values can be changed by a signal assignment statement with symbol "<=", and followed by a reserved word "after" to suspend the execution time. For example:

```
DATA <= "1111" after 5 ns;
PC <= PC+1 ;
```

The signal DATA will be "1111" after 5 ns, and the new PC will be the old PC plus one immediately. Note, that the type of the right side must be the same type

as the left side, which means that the DATA must be a 4-bit vector type, or "r_word" defined in a package.

Since, the signals are used to transfer values through the ports of components or provide sensitivities among processes, components, or entities concurrently, the signals perhaps provide the most basic behavior in the VHDL.

There are two kinds of signals, simple signal and resolved signal. Each simple signal has a single source called driver, which is driving the signal to hold the result of the signal assignment. Only one source can provide a value to the signal which is similar to the situation in reality, where only one gate driver can be active at a time in a digital circuit.

A signal with more than one signal driver, is called resolved signal. In VHDL, there is a bus resolution function to deal with this situation. Such a situation is shown in Figure 2 :

```
type bit3 is ('0', '1', 'Z');
type bit_array is array (integer range <>) of bit3;
function resolved_fun (bitin : bit_array) return bit3;
subtype r_bit is resolved_fun bit3;
--------------------
function resolved_fun (bitin : bit_array) return bit3 is
        variable resolved_value:bit3:='Z';
begin
        for i in bitin'range loop
            if bitin(i)/='Z' then resolved_value := bitin(i);
            exit; end if; end loop;
        return resolved_value;
end  resolved_fun;
```

Figure 1. Bus resolution function

A resolution function is a function which takes a one-dimensional, unconstrained array of values of the resolved type, and returns a single value of the same resolved type. In this thesis, most of the signals are resolved signals, namely "r_bit", "r_word", or "r_type", whose types are defined in a package which will be discussed later.

In Figure 1, to define the bus resolution function, first, we must define the basic type which is "bit3". Next, use the defined "bit3" to declare the unconstrained array type "bit_array". Then declare the "resolved_fun" function, and build the function in the package body. Last, define the "r_bit" subtype, which is for one bit resolved signal. The details of the package and types will be talked about later.

2. Data Types

Type is a named set of values with a common characteristic, and subtype is a subset of the values of a type [Ref. 3]. As shown in Figure 2, there are numbers of types.

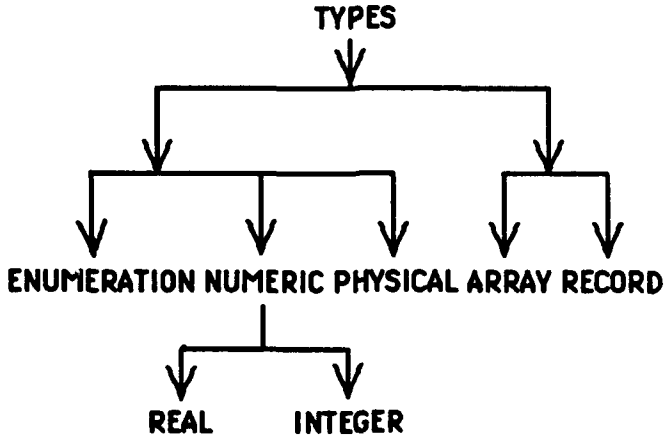


Figure 2. VHDL types

The most basic types in VHDL are "integer" type, "real" type, and "enumeration" type. For example:

```
type bit3 is ('0','1','Z');
```

is a "enumeration" type, which can be only three different values, '0', '1', and 'Z'. The other types, "boolean", "bit" and "character" are also "enumeration" type defined in the VHDL standard package. The other example:

```
type r_vector is array (range 0 to 15 of integer );
```

defines the ascending integer array type, "r_type". The range can be also descending, or not specified shown as unconstrainted symbol "<>". The third example:

```
type r_bit is resolved_fun bit3;
```

is a special type for the resolved value. The "resolved_fun" is a resolution function name, and "bit3" is a tristate type defined previously. The "r_bit" type is for one bit resolved signal. The resolved value will be used often in the thesis. Since this thesis only deals with "integer", "enumeration", and "array", the other types will not be discussed here.

B. BASIC STRUCTURES OF VHDL

1. Design Entity

The basic unit in VHDL is called a design "entity". Any one entity may be reused many times within an overall descriptions, and may have several architectures describing different ways of realizing the "entity". To easily understand the concept of an "entity", Figure 3 illustrates an example.

```
use work.pack3.all;
entity pcc is
        generic(DISDEL : TIME)
        port( PC : inout r_word;
              INC : in r_word;
              PCL : in r_bit
              PCE : in r_bit);
end pcc;
```

Figure 3. Example of a design entity

The unit starts with "entity pcc" , and end with "end pcc". Pcc is an identifier. Generic is a way for an instantiating architecture to pass environment parameter to an instantiated component. In this case, DISDEL means "delay", and TIME means a "type" defined in VHDL standard package. The port declares a number of signals, PC, INC, PCL, and PCE, which will be signals in the architecture body.

Each signal has an associated mode, which is "in", "out", or "inout". The types of "r_word" and "r_bit" are defined in a package, and that is why a "use" clause is present at the beginning of the entity declaration. The "use" clause specified that there is a "pack3" package in the working library which will be visible and referred to user defined types and functions.

2. Architecture Body

Architecture body basically defines how the inputs and outputs signals of the entity are obtained. It is a means of specifying the mechanics of the entity directly either in behavior description or in structural decomposition in terms of simpler components. Figure 4 shows the example of an architecture body. This is an example of an architecture body for a program counter (PC) register that

has input signals, enable control (PCE), increment control (INC), load control (PCL), and output PC signal.

```
use work.pack3.all;
architecture arcpc of pcc is
        signal PCI : r_word;
begin
        process(PCE, INC, PCL)

        begin
                if (PCE='1') then
                        PC<=PCI;
                else
                        PC<="ZZZZ" after DISDEL;
                end if;
                if  (inc='1') then
                        PCI<=inc_word(PCI);
                end if;
                if (PCL='1') then
                        PCI<=PC;
                end if;
        end process;
end arcpc;
```

Figure 4. Example of an architecture body

The VHDL program of this example is totally a behavior description of the pcc entity. As mentioned before, it is similar to high level programming languages. One internal signal PCI are declared as type "r_word". Three sensitivity signals PCE, PCL, INC are associated with the "process". It means that any change happens to any one of these signals will trigger the process.

The delay clause "after" has been used to support the sequential assignment statements to implement the correct behavior. The function call "inc_word" sends a value to and get a value back from the package. Any number

of concurrent statements may occur in an architecture body, but in Figure 4, the concurrency is restricted by the process statement, which means that the statements are executed in sequence in a process.

3. Packages

It may be tedious to repeat the user declarations whenever the designer wants to use them. So, VHDL uses a "package" mechanism for frequently used declarations to alleviate this problem.

Package is divided into two parts, "package declaration" and "package body". Data "types", "constants", and "subprograms" can be declared in the "package declaration" part, and the "package body" part contains the subprogram bodies. If no subprogram is contained, "package body" is not required. Figure 5 shows some of the system "package declaration" and "package body" used in this thesis.

```
package pack3 is
   type bit3 is ('0','1','Z');
   type bit_array is array (integer range <>) of bit3;
   function resolved_fun (bitin : bit_array) return bit3;
   subtype r_bit is resolved_fun bit3;
   type tsv is array (integer range <>) of r_bit;
   subtype r_word is tsv(3 downto 0);
   type r_type is array(integer range 0 to 15) of r_word;
   function inc_word(count : r_word) return r_word;
   ----------
   ----------
end pack3;
--------------------
package body pack3 is
   function resolved_fun (bitin : bit_array) return bit3 is
         constant floatvalue : bit3 :='0';
         variable resolved_value : bit3 :='Z';
```

Figure 5. Example of package declaration and package body

```
  begin
        if bitin'length=0 then return floatvalue;
        else for i in bitin'range loop
                    if bitin(i)/='Z' then resolved_value := bitin(i);
                    exit; end if; end loop;
              return resolved_value; end if;
  end  resolved_fun;
------------
  function inc_word(count:r_word) return r_word is
        variable A : r_word;
  begin
        A:=count;
        for i in count'low to count'high loop
              if A(i) = '0' then A(i) := '1'; exit;
              else A(i) := '0'; end if; end loop;
        return A;
  end inc_word;
end pack3;
```

Figure 5. Example of package declaration and package body (continued)

From Figure 4 and Figure 5, we can see that the package declaration contains the public or visible declarations, which can be used by other units with an "use" clause in front of the entities or architectures. On the other hand, the "package body" contains the private or invisible operations, which is only aroused by itself "package declaration".

4. Subprogram

A subprogram is a sequence of declarations and statements that can be invoked repeatedly from different locations in a VHDL program [Ref. 4]. There are two kinds of subprograms, function and procedure. The difference between them are:

- The forms of subprogram specification are different. The forms are shown as follows:

  procedure identifier interface-list

  function identifier interfact-list **return** type-mark

- The invocation of a procedure is a statement. While the function is an expression.
- Functions are used strictly for computing new values. While procedures are permitted to change the values.
- All parameters of functions must be of mode in. While procedures may be of mode in, out, or inout.
- All parameters of functions must be of class signal or constant. While procedures, if no class is specified, parameters of mode in are interpreted as being of constant, and parameters of mode out and inout are interpreted as being of class variable.

Only functions will be used in this thesis, so the explanation of the usage of procedure is skipped. Figure 4 and Figure 5 display the function call, function declaration, and function body. One of the function is named "inc_word". It is terminated by executing a "return" statement, which determines the value returned by the function call. It acts just like in a general programing language. The other function is called resolved_fun, is a very special function (resolution function) for multiple drivers or bus in VHDL.

C. PROCESSES, ATTRIBUTES, AND CONTROL STATEMENTS

1. Process

In VHDL there are two things we have to be concerned with. One is the "concurrency", and the other one is the "sequency". Within a process, all the statements including signal assignments are executed in sequence. The processes are executed in concurrency acting just like signal statements. In Figure 4, the statements within the process are executed in sequence. While in Figure 6 shown

in the following, the three processes, and one signal assignment, SYSCLK, RUN_P, FETCH_P, and STOP signal assignment, are executed concurrently.

```
entity control is
  generic(RDDEL, PER : TIME);
  port (   RUN  : in r_bit;
           FETCH : in boolean);
end control;
use work.pack3.all;
architecture arcc of control is
  signal CLK : boolean;
  signal STOP, STOPR :  r_bit;
begin
  SYSCLK: process(RUN, CLK, STOP)
  begin
        if (RUN='1') and (not RUN'stable) then
              CLK <= true;
        elsif (RUN='1') and (STOP= '0') then
              CLK <= transport not CLK after PER; end if;
  end process SYSCLK;
  ---------
  RUN_P: process(RUN)
  begin
    if RUN='1' then STOPR <= '0' ; else STOPR <= '1' ; end if;
  end process RUN_P;
  ---------
  FETCH_P: process
        begin
              wait on FETCH ;
              ----------          Inner executable statement
              ----------
              wait for RDDEL;
              ----------          Inner executable statement
              ----------
              end process FETCH_P;
  ----------
  STOP<=STOPR when not STOPR'quiet else
         STOP;
end arcc;
```

Figure 6. Example of a process

Each process defines a specific action, or behavior. The action is activated by any one of the changed sensitivity signals, and as shown in Figure 4, PCL, PCE, and INC are three sensitivity signals. If there is no changed sensitivity signal, the process is suspended. Therefore, a process is always in one of the two states, active or suspended.

Within a process, the VHDL provide three forms of "wait" to control the states. They are:

wait on signal_list;
wait until condition;
wait for time-expression;

Figure 6 demonstrates four processes in the same architecture body. They are SYSCLK, RUN_P, FETCH_P and one special process, STOP signal assignment. They are running concurrently. But in the process, the statements are running in sequence. Despite specifying the sensitivity signals within the parenthesis, the sensitivity signals can also be expressed just like in the FETCH_P process using the reserved word "wait on". The other suspension statement "wait for" hold the execution some delay time, which is the parameter defined in the generic declaration.

2. Attribute

Certain classes of items in VHDL may have attributes. These classes are:

types, subtypes
procedures, functions
signals, variables, constants

entities, architectures, configurations, packages
components
statement labels

An attribute is a named characteristic of items belonging to these classes. Some predefined attributes, which will be used in the thesis, are quite useful. These are [Ref. 4];

a. S'stable(T)

S'stable is of type BOOLEAN, signal-valued attribute. The T parameter is optional. The default value is 0 ns. This attribute defines that it is true if S has been stable for the length of T time units, otherwise it is false.

b. S'quiet(T)

S'quiet is of type BOOLEAN, signal-valued attribute. The T parameter is optional. The default value is 0 ns. This attribute defines that it is true if S has not had transaction (i.e. not active) for the length of time T, otherwise, it is false.

c. A'range(N)

A'range is of type integer, array attribute. The parameter is optional. The default value is 1. This attribute returns the range of the Nth index of the array object or constrained array subtype.

d. A'length(N)

A'length is of type integer, array attribute. The parameter is optional, and the default value is 1. This attribute returns the number of values in the Nth index of the array object or constrained array subtype.

3. Control Statements

The control statements in VHDL are IF, CASE, LOOP, NEXT, EXIT, RETURN, and WAIT. Most of them will be used in the thesis except the NEXT statement. Example 1 and 2 display some usages of them.

Example 1:

```
architecture arcalu of alu is
  signal ALUB : r_word;
begin
  process(CB, ALUEN, A_ALU, B_ALU)
        variable addb : r_word;
  begin
        case CB is
              when "0011" =>                    ---CALL ADD FUN
                    addb:=addv(A_ALU,B_ALU);
              when "1010" =>                    ---CALL SUB FUN
                    addb:=subv(A_ALU,B_ALU);
              when others =>                    --- DO NOTHING
        end case;
        if ALUEN='1' then ALUO<=addb;
        else ALUO<="ZZZZ" after DISDEL; end if;
  end process;
end arcalu;
--------------
```

Example 2:

```
function inc_word(count:r_word) return r_word is
        variable A:r_word;
begin
  A:=count;
  for i in count'low to count'high loop
        if A(i) = '0' then A(i) := '1'; exit;
        else A(i) := '0'; end if; end loop;
  return A;
end inc_word;
```

The WAIT statements , as shown in Figure 6, are used to suspend a process for a period of time or until an event occurs.

D. THREE MODELING VIEWS OF VHDL

VHDL capabilities are generally divided into three ways of modeling, behavioral, dataflow, and structural. The definitions are shown in Figure 7 [Ref. 2].

THREE "VIEW" OF VHDL		
Behavioral	**Structural**	**Dataflow**
Definition		
A purely algorithmic description of a component The language is similar to existing general-purpose languages.	A simple netlist description of the component. Subcomponents may be specified at lower levels of the hierarchy or drawn from a library.	Describes network of signals, where function of each transformer is explicit in a text representation. As in many RTL languages, time may be dealt with explicitly.

Figure 7. Comparision of three modeling views of VHDL

In all these three levels, the entity declaration identifies the primary interface signals of the system, and the architecture body relates to the contents of the entity.

Figure 8 is a full_adder scheme diagram of the following examples. It consists of 2 xor gate, 2 and gate, and 1 or gate. Which means it is composed of 2 half_adder and 1 or gate.

There are three input signals, X, Y, and Cin, as well as two output signals Sum and Cout, which are of type of bit. The a, b, and c are internal variables, which are not visible. This diagram can be described in different ways of description with VHDL, which are behavior, dataflow, and structure.

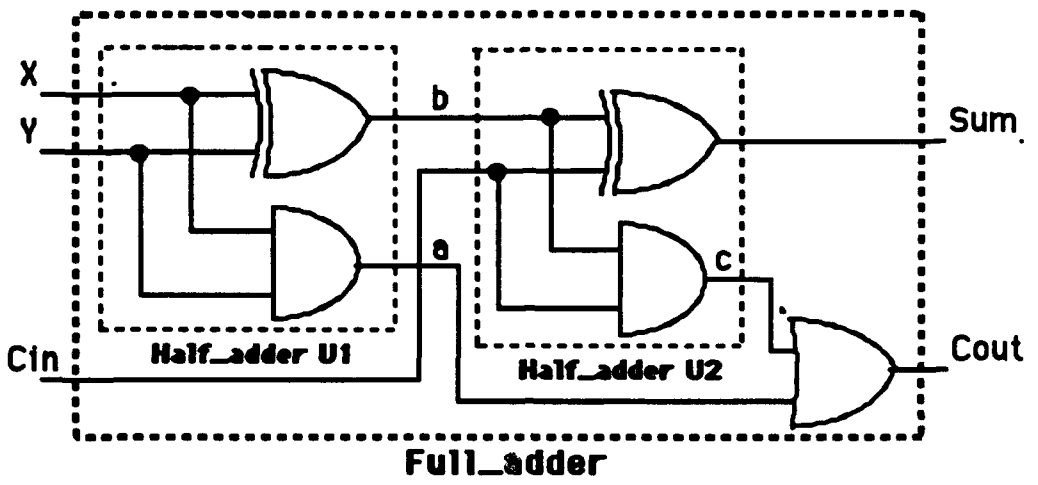


Figure 8. A Full_adder

Since all these three methodologies are using the same entity declaration, so the entity of full_adder will be shown first in Figure 9 before the discussion of the three modeling view of VHDL.

```
entity full_adder is
        port(  X, Y, Cin : in bit;
               Sum, Cout : out bit);
end full;
```

Figure 9. Entity of a Full_adder

The "full_adder" is the name of the entity. The input signals X, Y, and Cin, as well as the output signals Sum and Cout are to be generated in the architecture body in the following three ways.

1. Behavior

In the view of behavior, the VHDL may provide a purely algorithmic description of a component. The behavioral description is expressed in the same way as many high level languages such as PASCAL, FORTRAN, HDLs, or other programming languages. It allows some common commands such as If, Case, Loop, Next, Exit, Wait, and After statements [Ref. 2]. The language constructs can be seen in Figure 10.

```
architecture behavioral_view of full_adder is
begin
        process(X, Y, Cin)
            variable a, b, c : bit;
        begin
            if (X=Y) then b:='0'; else b:='1'; end if;
            if (X='0') or (Y='0') then a:='0';  else a:='1'; end if;
            if (b=Cin) then Sum<='0'; else Sum<='1'; end if;
            if (b='0') or (Cin='0') then c:='0'; else c:='1'; end if;
            if (a='1') or (c='1') then Cout<='1'; else Cout<='0'; end if;
        end process;
end behavioral_view;
```

Figure 10. Behavioral constructs

Figure 10 is a behavior model of a one bit full_adder declared before. The main feature used in behavior is the "process" of the VHDL. That allows declaration of local variables for calculation or to hold the value. The three sensitivity signals X, Y, and Cin are used to invoke the process for calculation in sequence. These signals are interfaced through the port of the entity declaration.

2. Data Flow

In the view of data flow, VHDL is similar to many RTL languages. The VHDL description may provide a description of an "entity" and "architecture" called "component" to describe a network of signals. Time may be explicitly dealt with. Register signals and bus signals can be described in the data flow. The data flow constructs for a bit full_adder is shown in Figure 11.

```
architecture dataflow_view of full_adder is
        signal b : bit;
begin
        b<=X xor Y after 10 ns;
        Sum<=b xor Cin after 10 ns;
        Cout<=(X and Y) or (b and Cin) after 20 ns;
end dataflow_view;
```

Figure 11. Data flow constructs

The main features of Figure 11 are the signal assignments. Three signal assignments act just like three different "processes" calculating in concurrence. The "after" clauses are involved for timing accuracy. The statements used here are also more close to the structural views than the behavioral view.

3. Structure

In the view of structure, A VHDL description may include the equivalent of a netlist description of components. The components in the netlist may be given in another descriptions at a lower level in the hierarchy, or in the existing library. Figure 12 displays the structure constructs.

You can see that half_adder and or_gate are the components which may exist in a lower architecture or in a library. The signals of the actual ports specification are in the same order as the local ports in the component. The

"use" clauses invoke the formal ports of components which are stored in a library.

```
use work.half_adder.all;
use work.or_gate.all;
architecture structure_view of full_adder is
component half_adder
        port(I1, I2 : in bit ; C, S : out bit);
end component;
component or_gate
        port(I1, I2 : in bit ; O : out bit);
end component;
signal a, b, c : bit;
begin
        U1 : half_adder port map(X, Y, a, b);
        U2 : half_adder port map(b, Cin, c, Sum);
        U3 : or_gate port map(a, c, Cout);
end structure_view;
```

Figure 12. Structural constructs

VHDL is a complex language. It is not possible to discussed all the features of this language here. Therefore, the emphasis in this chapter is concentrated on those features in the data flow simulation program of a later chapter.

III. DATA FLOW IMPLEMENTATION OF THE W-4 COMPUTER

This chapter describes an actual VHDL implementation of data flow configuration of the W-4 computer. Before details are discussed, a special package used in the VHDL implementation is described. It is also necessary to introduce the W-4 computer, therefore a VHDL behavior model will be presented first to illustrate the instructions.

This chapter consists of three sections. The first section is about the W-4 system "package declaration" and "package body", where the types and functions used in the data flow implementation are defined. The second section introduces the W-4 computer with the behavioral description in VHDL. The last section describes the data flow implementation of the W-4 system. Data flow components include Program Counter (PC), Accumulator (ACC), B Register (B_REG), Instruction Register (IR), Temporary Address Register(TAR), Arithmetic Logic Unit (ALU), Random Access Memory (RAM), controller and test_bench.

A. W-4 SYSTEM PACKAGE

The W-4 system package contains the types and functions frequently used in the data flow implementation. The package has two parts, the W-4 system package declaration and the W-4 system package body.

1. System Package Declaration

Figure 13 shows the complete W-4 system package declaration called "pack3". The general type "bit3" is first declared as a three_valued enumeration

type, '0', '1', and 'Z'. Second, the type "bit_array" is defined as an unconstrained array of "bit3". These types are only used by the bus resolution function, "resolved_fun" which is described in the package body. Following this, a subtype "r_bit" is defined. Whenever the "r_bit" is applied, it automatically invokes the bus resolution function "resolved_fun", and returns a value in type "bit3". The following type "tsv" is defined as an array type whose individual elements are of subtype "r_bit". The "tsv" is an unconstrained array. On the other hand, the type "r_word" is a resolved four bits vector. The last "r_type" is defined as an array type with 16 elements. Every element of the array is of type "r_word". This is intended to be used for the implementation of the RAM storage.

```
package pack3 is                          --- PACKAGE DECLARATION
    type bit3 is ('0', '1', 'Z');
    type bit_array is array (integer range <>) of bit3;
    function resolved_fun (bitin : bit_array) return bit3;
    subtype r_bit is resolved_fun bit3;
    type tsv is array (integer range <>) of r_bit;
    subtype r_word is tsv(3 downto 0);
    type r_type is array(integer range 0 to 15) of r_word;
    function bitarray_to_int (bits : r_word) return natural;
    function int_to_bitarray (int : natural) return r_word;
    function inc_word(count : r_word) return r_word;
    function inv_w(A : r_word) return r_word;
    function addv(A : r_word; B : r_word) return r_word;
    function subv(A : r_word; B:r_word) return r_word;
    function andv(A : r_word; B : r_word) return r_word;
    function orv(A : r_word; B : r_word) return r_word;
    function xorv(A : r_word; B : r_word) return r_word;
end pack3;
```

Figure 13. System package declaration

In addition to the "rsolved_fun" function, there are nine functions declared in the package, "bitarray_to_int", "int_to_bitarray", "inc_word", "inv_w", "addv", "subv", "andv", "orv", and "xorv". The parameter of a function includes the name and type declared in the package. The type "r_word", a resolved four bit array is used most of the time. The returned type is also shown in the package declaration. The interior of the functions are just like any other general programming language. The package body is shown in Figure 14.

```
package body pack3 is                          --- PACKAGE BODY
    function resolved_fun (bitin : bit_array) return bit3 is
      constant floatvalue : bit3 := '0';
      variable resolved_value : bit3 := 'Z';
    begin
      if bitin'length=0 then return floatvalue;
      else for i in bitin'range loop
              if bitin(i)/='Z' then resolved_value := bitin(i);
              exit; end if;
             end loop;  return resolved_value; end if;
    end  resolved_fun;
    function bitarray_to_int (bits : r_word) return natural is
      variable result : natural := 0;
    begin
      for i in bits'range loop
              result := result*2;
              if bits(i)='1' then result := result+1; end if;
      end loop; return result;
    end;
    function int_to_bitarray (int : natural) return r_word is
      variable digit : natural := 2**3;
      variable local : natural;
      variable result : r_word;
    begin
      local:=int;
```

Figure 14. System package body

```
   for i in result'range loop
         if local>=digit then result(i) := '1'; local := local-digit;
         else result(i) := '0'; end if;
         digit := digit/2;
   end loop;  return result;
end int_to_bitarray;
function inc_word(count : r_word) return r_word is
   variable A : r_word;
begin
   A:=count;
   for i in count'low to count'high loop
         if A(i) = '0' then A(i) := '1'; exit;
         else A(i) := '0'; end if;
   end loop; return A;
end inc_word;
function inv_w(A : r_word) return r_word is
   variable temp : r_word;
begin
   for i in A'range loop
         if A(i)='0' then temp(i) := '1';
         else temp(i) := '0'; end if;
   end loop; return temp;
end inv_w;
function addv(A : r_word; B : r_word) return r_word is
   variable c : integer;
   variable cv : r_word;
begin
   c:=bitarray_to_int(A)+bitarray_to_int(B);
   while c >= 16 loop c := c rem 16;
   end loop;
   cv := int_to_bitarray(c);  return cv;
end addv;
function subv(A : r_word; B : r_word) return r_word is
   variable cv, c, c0 : r_word;
begin
   c0 := "0001";  c:=addv(A, inv_w(B));  cv:=addv(c, c0);  return cv;
end subv;
function andv(A : r_word; B : r_word) return r_word is
   variable temp : r_word;
begin
   for i in A'range loop
```

Figure 14. System package body (continued)

```
            if A(i)=B(i) and A(i)='1' then temp(i) := '1';
            else temp(i) := '0'; end if;
        end loop;  return temp;
    end andv;
    function orv(A : r_word;  B : r_word) return r_word is
        variable temp : r_word;
    begin
        for i in A'range loop
            if A(i)='1' or B(i)= '1' then  temp(i) := '1';
            else temp(i) := '0'; end if;
        end loop; return temp;
    end orv;
    function xorv(A : r_word; B : r_word) return r_word is
        variable temp : r_word;
    begin
        for i in A'range loop
            if A(i)/=B(i) then temp(i) := '1';
            else temp(i) := '0'; end if;
        end loop; return temp;
    end xorv;
end pack3;
```

Figure 14. System package body (continued)

The function "bitarray_to_int" converts bit array, r_word, to a natural number. The function "int_to_bitarray" performs the opposite conversion. The function "inc_word" performs the increment of the value by one and returns a r_word type value. The next function "inv_w" generates the 1's complement of the bits of the argument. The functions "addv" and "subv" execute the operations of addition and subtraction. The last three functions "andv", "orv", and "xorv" perform the logical "and", "or", and "xor" operations .

B. A BEHAVIORAL DESCRIPTION

A functional block diagram of the W-4 simple computer is shown in Figure 15 [Ref. 1].

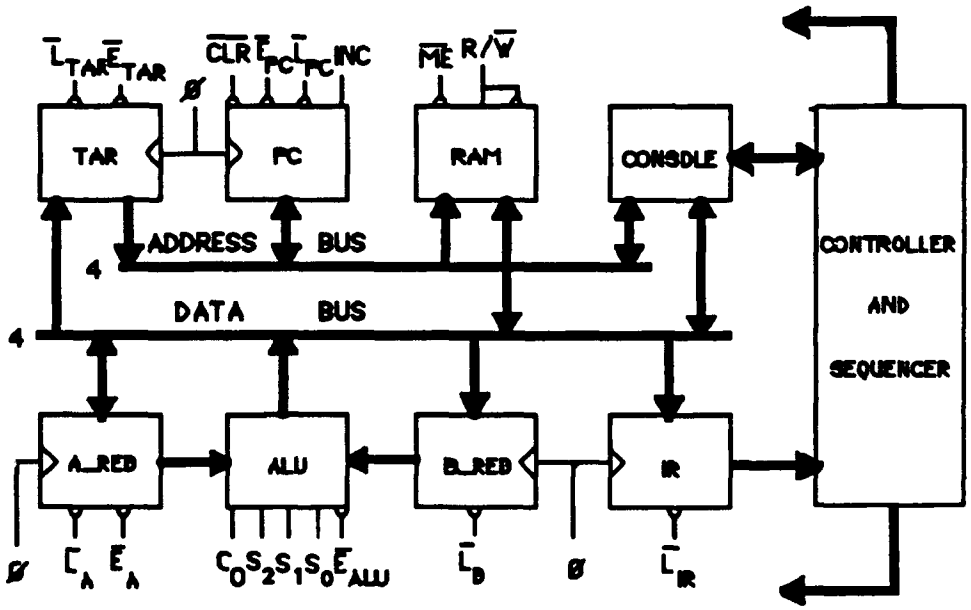


Figure 15. W-4 computer block diagram (adopted from [Ref. 1])

This 4-bit computer mainly consists of seven parts, temporary address register (TAR), program counter (PC), random access read/write memories (RAM), A register (A_REG), B register (B_REG), arithmetic and logic unit (ALU), and instruction register (IR). It is a 4-bit computer data flow model controlled by the controller and sequencer. There are 16 assembler instructions for the W-4 computer which are shown in Figure 16 [Ref. 1].

As you can see that the W-4 is a register oriented computer that has the basic move instructions, LDA, STA, and LDI. Arithmetic and logic instructions include ADD, SUB, AND, OR, and XOR. Branch control instructions are JUMP and JAN. The others, INCA, SHAL, CLA and CMA are accumulator manipulation instructions.

Mnemonic	Machine code	Operation	Remarks
LDA aaaa	0001 aaaa	[Raaaa] -> ACC	Load the Accumulator with [Raaaa](aaaa is a 4-bit address)
ADD aaaa	0010 aaaa	[ACC]+[Raaaa] -> ACC	C0=0; S1=S0=1; S2=0;
SUB aaaa	0011 aaaa	[ACC]-[Raaaa] -> ACC	C0=1; S1=1; S0=S2=0;
STA aaaa	0100 aaaa	[ACC] -> Raaaa	Store [ACC] in Raaaa
AND aaaa	0101 aaaa	[ACC]•[Raaaa] -> ACC	S2=S1=1; S0=C0=0;
OR aaaa	0110 aaaa	[ACC]+[Raaaa] -> ACC	S2=S0=1; S1=C0=0;
XOR aaaa	0111 aaaa	[ACC]⊕[Raaaa] -> ACC	S2=1; S0=S1=C0=0;
LDI dddd	1000 dddd	dddd -> ACC	Load Immediate data (dddd is a 4-bit data word)
JUMP aaaa	1001 aaaa	aaaa -> PC	Jump to next instruction at aaaa
JAN aaaa	1010 aaaa	If [ACC]<0, aaaa -> PC else, execute next instruction	Jump to aaaa if [ACC] is Negative
INCA	1011	[ACC]+1 -> ACC	Increment the Accumulator
SHAL	1100	$[A_2A_1A_0]0$ -> ACC	Shift [ACC] Left
CLA	1101	0000 -> ACC	Clear the Accumulator
CMA	1110	[ACC]' -> ACC	One's Complement [ACC]
HALT	1111	Halt the execution. Stop the clock	

Figure 16. Instructions and code operations (adopted from [Ref. 1])

The original register transfer language (RTL) specifications shown in Figure 17 describes the micro operations of the W-4 computer [Ref. 1].

Instruction	OP Code	Micro operations	Remarks
FETCH	0000	[Rpc]-----> IR [PC] + 1 ---> PC	
LDA	0001	[Rpc]-----> TAR	
		[R_{TAR}]----> ACC [PC] + 1 ---> PC	
ADD	0010	[Rpc]-----> TAR	
		[R_{TAR}]-----> B [PC] + 1 ---> PC	
		[ACC] + [B] + CO----> ACC	CO=0
SUB	0011	[Rpc]-----> TAR	
		[R_{TAR}]-----> B [PC] + 1 --> PC	
		[ACC] + [B]' + CO----> ACC	CO=1
STA	0100	[Rpc]-----> TAR	
		[ACC]-----> R_{TAR} [PC] + 1 ---> PC	
AND	0101	[Rpc]-----> TAR	
		[R_{TAR}]-----> B [PC] + 1 --> PC	
		[ACC] • [B] --> ACC	
OR	0110	[Rpc]-----> TAR	
		[R_{TAR}]-----> B [PC] + 1 --> PC	
		[ACC] OR [B] --> ACC	
XOR	0111	[Rpc]-----> TAR	
		[R_{TAR}]-----> B [PC] + 1 --> PC	
		[ACC] ⊕ [B] --> ACC	
LDI	1000	[Rpc] -----> ACC [PC] + 1 --> PC	
JUMP	1001	[Rpc]-----> TAR	
		[TAR]-----> PC	

Figure 17. Micro operations of W-4 computer (adopted from [Ref. 1])

Instruction	OP Code	Micro operations	Remarks
JAN	1010	[Rpc]-----> TAR [PC] + 1 ---> PC	
		[TAR] -----> PC*	
INCA	1011	0000 -----> B	
		[ACC] + [B] + CO----->ACC	CO=1
SHAL	1100	[ACC] -----> B	
		[ACC] + [B] + CO----->ACC	CO=0
CLA	1101	0000 -----> ACC	
CMA	1110	1111 -----> B	
		[ACC] ⊕ [B] -----> ACC	
HALT	1111	NOP	
		NOP	
		NOP	
		NOP	

Figure 17. Micro operations of W-4 computer (continued)(adopted from [Ref. 1])

For the instruction JAN, the symbol "*" there means that a conditional programmed jump will take place only if the MSB of [ACC] is '1', which means that the contents of ACC is smaller than zero. Otherwise, [TAR] will not be loaded into the PC.

A complete W-4 behavioral model in VHDL description is shown in appendix A. As a brief example for explanation, only two of the sixteen instructions, LDA and STA are shown in Figure 18. Since the purpose is to simulate the behavior of a W-4 computer, there is no need for signal port declaration in the entity statement of the first line in Figure 18.

```
entity behav3 is end behav3;
use work.pack3.all;
architecture arcbehav3 of behav3 is
    signal    P_C   : r_word := "0000";
    signal    A_CC, IN_STR   : r_word;
    signal    RAM   : r_type  := ("0001","1110","0100","1111","1111",
                                  "1111","1111","1110","0111","1110",
                                  "1110","1100","1101","1111","1100",
                                  "0001");
begin
    process(P_C)
      variable ir, b_reg, tar : r_word;
      variable int_pc, int_tar, int_acc : integer;
    begin
      ir := RAM(bitarray_to_int(P_C));
      int_pc := bitarray_to_int(P_C)+1;
      case ir is
            when "0001" =>                                    ---LOAD
                  IN_STR <= ir;
                  int_tar :=bitarray_to_int(RAM(int_pc));
                  A_CC <= RAM(int_tar);
                  int_pc := int_pc+1;
                  P_C <= int_to_bitarray(int_pc) ;
            when "0100" =>                                    ---STA
                  IN_STR <= ir;
                  int_tar := bitarray_to_int(RAM(int_pc));
                  RAM(int_tar) <=A_CC;
                  int_pc := int_pc+1;
                  P_C <= int_to_bitarray(int_pc) ;
            ------------------
            ------------------
            when others =>
                  IN_STR <= "1111" after 2 ns;
      end case;
    end process;
end arcbehav3;
```

Figure 18. A behavioral description

The total model is contained in one process. Therefore, the VHDL instructions are performed in sequence just like any other high level

programming language as mentioned in previous chapter. There are four signals, P_C, A_CC, IN_STR, and RAM, whose values can indicate the execution status of the W-4 computer. The initial values have been set in the P_C, and RAM signal declarations which is used to run a VHDL program.

The sixteen 4-bit pattern as the initialized value for the signal RAM is a simple demonstration program. The sensitivity signal P_C invokes the running process whenever the P_C value changes. The CASE statement acts like a general programming language construct to direct the control to different part of the process depending on the instruction decoding. Because it is a 4 bits computer, the results of arithmetic and logic operations more than 4 bits are truncated.

For comparison, the mnemonic instruction of this demo program is shown in Figure 19.

Address Location		Mnemonic Program	Coded Program	Remarks
0	0000	LDA E	0001	Load the A_register with the number
1	0001		1110	in memory register 14
2	0010	STA F	0100	Store the contents of the A-register
3	0011		1111	to the memory register 15
4	0100	HALT	1111	Stop
	------	Don't care	------	
14	1110	12	1100	Data in memory register 14
15	1111	1	0001	Data in memory register 15

Figure 19. Demonstration program in machine code

The first instruction is "LDA E" which loads the contents of memory register "E" into the A_REG. The instruction code of LDA, "0001" is stored at the address location "0" and the operand address "E", "1110", is stored at the address "1".

The second instruction is "STA F" which means to store the contents of the A_REG to the memory location "F". The instruction code of STA, "0100" is stored at the memory location "2", and the operand address "F", "1111", is stored in the address location "3". The third instruction "HALT" is stored in the next memory location "4". This instruction will stop the operations of the W-4 computer in this demo run.

The contents at the memory locations "E" and "F" are "1100" and "0001". These location are used to store the data. The rest of the instruction codes above address location "4" can be ignored because of the "HALT" instruction.

Figure 20 shows the VHDL simulation result. The signals were changed in the execution of the instructions LDA and STA. In both the LDA and the STA instruction, the operands are converted from bit array to integers so that the appropriate locations in VHDL can be accessed. This conversions are clearly shown in the VHDL program in Figure 18.

TIME(ns)	P_C	IN_STR	A_CC	RAM(14)	RAM(15)
0	"0000"	"0000"	"0000"	"1100"	"0001"
+1	"0010"	"0001"	"1100"		
+2	"0100"	"0100"			"1100"
2		"1111"			

Figure 20. Result of a behavioral model simulation

The first column of Figure 20 shows the time line progress from 0 ns to 2 ns, The "+1" and "+2" means one delta cycle and two delta cycle of the VHDL simulator of the simulation time zero nano second. Delta cycles are used in the simulator to update signals which are triggered without delays.

The second row of Figure 20 shows the contents of the signals, which are the initial value before the execution. The third row shows the contents of each signal at time "+1 delta cycle". At this point in time, we can see that the P_C is "0010", the IN_STR is "0001" which is the code of the LDA instruction, and the LDA have loaded the A_CC with the contents from the memory location 14 which is "1100". The fourth row shows the effects of STA operations. After the execution, the P_C is "0100", the IN_STR is "0100" which is the code of the "STA" instruction, and the RAM(15) has been stored with the value of A_CC which is "1100". The last rows shows the effect caused by "IN_STR <= "1111"" after 2 ns delay. The statement is shown in Figure 18, which terminate this behavior simulation.

C. A DATA FLOW DESCRIPTION

In general, data flow descriptions are much more complicated than behavioral descriptions. In this section a VHDL data flow implementation of the W-4 computer is introduced. Figure 21 shows the system block diagram of the data flow model. This model contains one W-4 system package which have been introduced in the first section, and 8 components, "PC", "ACC", "B_REG", "IR", "TAR", "ALU", "RAM", and "CONTROLLER" with five processes.

In Figure 21 the data flow description describes the W-4 model in terms of structures, which means that the W-4 computer is built from components.

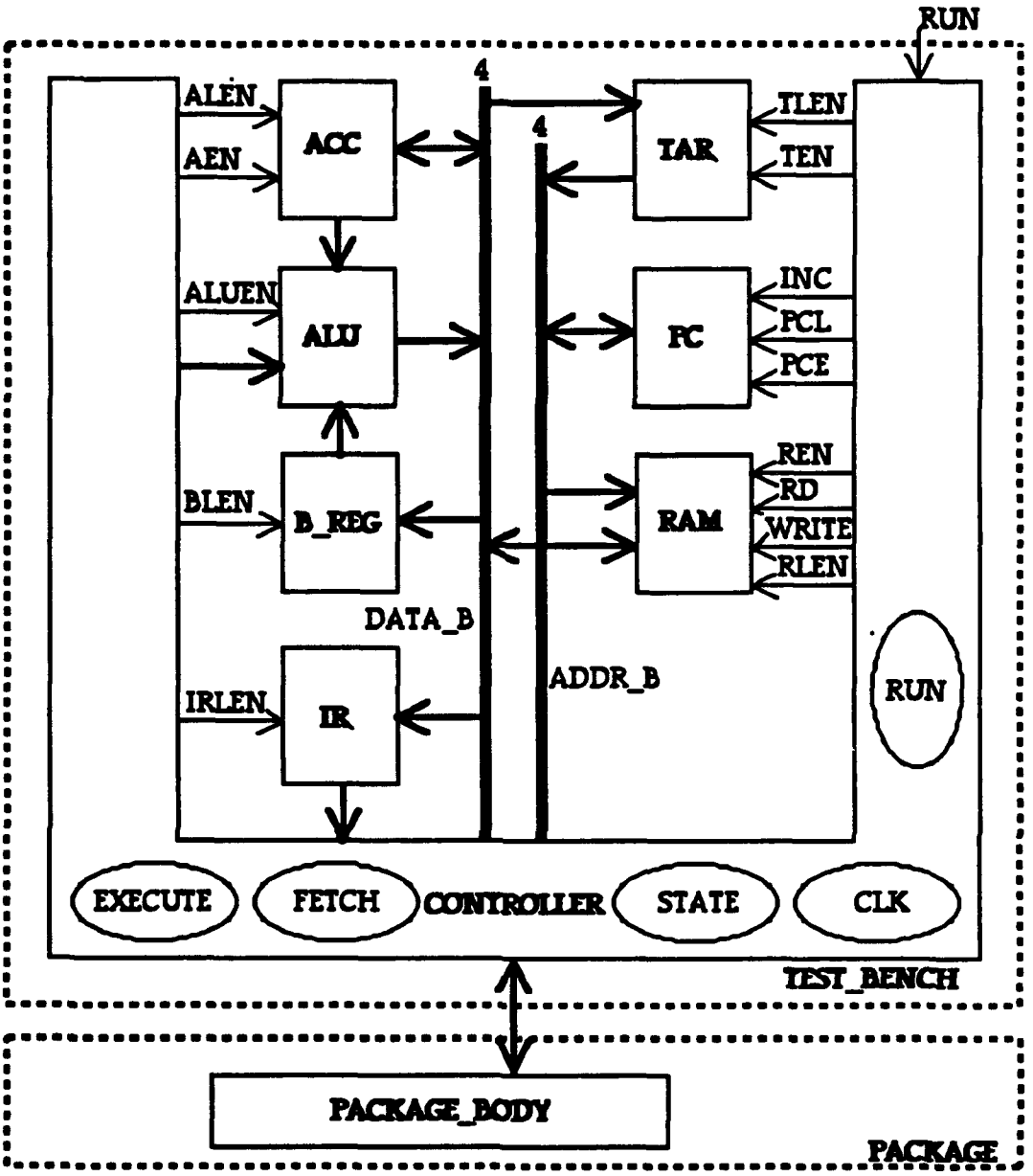


Figure 21. System block diagram of a data flow structure

Each component is activated by the individual enable signals which are generated from the controller component. It then transfers or loads the contents onto the DATA_B bus or ADDR_B bus at the proper time. The five processes in the controller component are invoked by the signal RUN to yield the enable

signals. The test_bench is the overall top entity that connects these components together.

1. Program Counter (PC) Model

Figure 22 is a program counter (PC) model. In this process, all statements are executed in sequence. The generic declaration specifies the delay DISDEL which will be a value given in the test_bench at a higher level of the hierarchy. The signal PC and three sensitivity signals INC, PCL, and PCE, are declared in the port. These interface signals activates three different functions, which enables, loads, or increment the program counter. The local signal PCI keeps the PC value in the process. The PC output is always "ZZZZ", unless the PCE is activated.

```
use work.pack3.all;
entity pcc is
    generic(DISDEL : TIME);
    port(     PC : inout r_word;
              INC : in r_bit;
              PCL : in r_bit;
              PCE : in r_bit);
end pcc;
----------
use work.pack3.all;
architecture arcpc of pcc is
    signal PCI : r_word;
begin
    process(PCE, INC, PCL)
    begin
      if (PCE = '1')  then PC <= PCI;
            else PC <= "ZZZZ" after DISDEL; end if;
      if (INC = '1') and (not INC'stable) then PCI <= inc_word(PCI) ; end if;
      if (PCL = '1')  then PCI <= PC; end if;
    end process;
end arcpc;
```

Figure 22. PC model source code in VHDL

2. Accumulator (ACC) Model

The accumulator (ACC) model is expressed in Figure 23. The ALEN signal triggers the process to load the value from the data bus which is the value of DATA, whenever its value is changed. The AEN enables the values of ACC signal forward onto the data bus. Otherwise, it sends the "ZZZZ" high impedance signal onto the bus. Since the signal A_ALU is connected to the ALU component without any tri-state devices, the values of the signal A_ALU is available to ALU all the times.

```
use work.pack3.all;
entity acc is
    generic(DISDEL : TIME);
    port(     DATA : inout r_word;
              A_ALU : out r_word;
              ALEN, AEN  : in r_bit);
end acc;
------------
use work.pack3.all;
architecture arcacc of acc is
    signal ACCB : r_word;
begin
    process(ALEN, AEN)
    begin
      if ALEN = '1' then ACCB <= DATA; end if;
      if AEN = '1' then DATA <= ACCB;
            else DATA <= "ZZZZ" after DISDEL; end if;
      A_ALU <= ACCB;
    end process;
end arcacc;
```

Figure 23. ACC model source model in VHDL

3. B Register (B_REG) Model

Figure 24 is a B_REGISTER model. It buffers the data from the port signal DATAI. If the signal BLEN is activated then the process loads the value from the data bus. In other words, DATAI will be loaded into the local buffer signal BB in the process. Since the B_ALU is connected to ALU component directly, the value of the BB signal is always available to the ALU component right away.

```
use work.pack3.all;
entity breg is
    port(     DATAI : in r_word;
              B_ALU : out r_word;
              BLEN  : in r_bit);
end breg;
------------
use work.pack3.all;
architecture arcb of breg is
    signal BB : r_word;
begin
    process(BLEN)
    begin
       if BLEN = '1' then BB <= DATAI; end if;
       B_ALU <= BB;
    end process;
end arcb;
```

Figure 24. B_REG model source code in VHDL

4. Instruction Register (IR) Model

The IR model is simpler than the others. There are three signals declared in the port. The sensitivity signal IRLEN triggers the process to transfer the values of DATA_I from the data bus to the output IRO. Figure 25 shows the IR model.

```
use work.pack3.all;
entity ir is
    port(     DATA_I  : in r_word;
              IRLEN   : in r_bit;
              IRO     : out r_word);
end ir;
-----------
use work.pack3.all;
architecture arcir of ir is
begin
    process(IRLEN)
    begin
      if IRLEN = '1' then IRO <= DATA_I; end if;
    end process;
end arcir;
```

Figure 25. IR model source code in VHDL

5. Temporary Address Register (TAR) Model

Figure 26 shows the TAR model. There are four signals declared in the port. The sensitivity signal TLEN enables the process to load the input values TARI from the data bus. On the other hand, the TEN signals transfers the buffer values, TARB, to the address bus. For the signal TARO, unless the signal TEN is enabled, the output is always in high impedance state, "ZZZZ".

```
use work.pack3.all;
entity tar is
    generic(DISDEL : TIME);
    port(     TARI : in r_word;
              TARO : out r_word;
              TLEN, TEN  : in r_bit);
end tar;
-----------
use work.pack3.all;
```

Figure 26. TAR model source code in VHDL

```
architecture arctar of tar is
    signal TARB : r_word;
begin
    process(TLEN, TEN)
    begin
      if TLEN = '1' then TARB <= TARI; end if;
      if TEN = '1' then TARO <= TARB;
            else TARO <= "ZZZZ" after DISDEL; end if;
    end process;
end arctar;
```

Figure 26. TAR model source code in VHDL (continued)

6. The Arithmetic Logic Unit (ALU) Model

The ALU model is shown in Figure 27. The signal CB are control bits C0, S2, S1 and S0, which is used for the "case" statement to invoke the proper function defined in the W-4 system package. In this way, what the instruction wants to do is implemented. The load variable "addb" can hold the returned value from the package. When the signal ALUEN is activated then the value of "addb" to the output, ALUO is forwarded. Otherwise, the output signal ALUO is always "ZZZZ".

```
use work.pack3.all;
entity alu is
    generic(DISDEL: TIME);
    port(     ALUO  : out r_word;
              A_ALU, B_ALU, CB : in r_word;
              ALUEN : in r_bit);
end alu;
-----------
architecture arcalu of alu is
    signal ALUB : r_word;
begin
    process(ALUEN)
```

Figure 27. ALU model source code in VHDL

```
    variable addb : r_word;
  begin
    case CB is
    when "0011" =>                          ---ADD,SHAL
         addb:=addv(A_ALU, B_ALU);
    when "1010" =>                          ---SUB
         addb:=subv(A_ALU, B_ALU);
    when "0110" =>                          ---AND
         addb:=andv(A_ALU, B_ALU);
    when "0101" =>                          ---OR
         addb:=orv(A_ALU, B_ALU);
    when "0100" =>                          ---XOR,CMA2
         addb:=xorv(A_ALU, B_ALU);
    when "1011" =>                          ---INCA
         addb:=inc_word(A_ALU);
    when "0111" =>                          ---CMA1
         addb:="1111";
    when others =>                          ---LDA,LDI,JUMP,JAN
                                            ---CLA,HALT
    end case;
    if ALUEN = '1' then ALUO <= addb;
    else ALUO <= "ZZZZ" after DISDEL; end if;
  end process;
end arcalu;
```

Figure 27. ALU model source code in VHDL (continued)

7. Random Access Memory (RAM) Model

The RAM model is a 16-location memory array. In Figure 28, the sensitivity signal REN together with signals RD or WRITE enables the RAM to read or write.

It is necessary to initiate the RAM at the beginning of the execution. The initial values are coming from RAM_I signal. To avoid unnecessary error caused by multiple reading of the incoming RAM_I values each time, the process uses a counter and a local variable, "ramb", to hold the first input values RAM_I. At any other time without enable signals the ram will output the "ZZZZ" to

represent the high impedance state of the memory, which no output values will appear on the data bus.

```
use work.pack3.all;
entity ram is
    generic( RDDEL, DISDEL : TIME);
    port(       RAM_I  : in r_type;
                DATA   : inout r_word;
                MA     : in r_word;
                RD, WRITE, REN  : in r_bit;
                RAM_O  : out r_type);
end ram;
------------
use work.pack3.all;
architecture arcram of ram is
begin
    process(REN)
       variable count : INTEGER := 0;
       variable ramb : r_type;
    begin
       if count=0 then
              ramb   :=   RAM_I;
              RAM_O <= ramb ;
              count :=   count+1;
       end if;
       if REN = '1' then
         if (RD = '1') then DATA <= ramb(bitarray_to_int(MA)) after RDDEL;
            else DATA <= "ZZZZ" after DISDEL; end if;
         if (WRITE = 1') then ramb(bitarray_to_int(MA)) := DATA; end if;
       else DATA <= "ZZZZ" after DISDEL; end if;
       RAM_O <= ramb after RDDEL ;
    end process;
end arcram;
```

Figure 28. RAM model source code in VHDL

8. Controller

The Controller is the center unit of the system. For convenience of explanation, only two of sixteen instructions, LDA and STA are shown in Figure 29. It provides enable signals for each component at the right time to stimulate the system and transfer the signals. Figure 29 shows only a part of the Appendix B, which is a complete controller program.

```
use work.pack3.all;
entity control is
    generic(MADEL, WDEL, ODEL, RDDEL, ENDEL, DISDEL, PER :
            TIME);
    port (  DATA : inout r_word;
            RUN  : in r_bit;
            A_ALU: in r_word;
            IRLEN, TLEN, TEN, INC, PCL, PCE   : out r_bit;
            ALEN, AEN, BLEN, ALUEN, REN, RD, WRITE : out r_bit;
            CB : out r_word);
end control;
use work.pack3.all;
architecture arcc of control is
    signal IRF : r_word;
    signal CLK, EXECUTE, FETCH : boolean;
    signal CSEN, CSENE, CSENF, STOP, STOPE, STOPR  : r_bit;
    signal IRLENF, IRLENB, ALENE, ALENB, AENE, AENB : r_bit;
    signal RDB, RDE, RDF, WRITEF, WRITEE, WRITEB  : r_bit;
    signal TLENE, TENE, TENB, TLENB, RENF, RENE, RENB : r_bit;
    signal BLENE, BLENB, ALUENE, ALUENB : r_bit;
    signal INCE, INCF, INCB : r_bit;
    signal PCLE, PCLF, PCLB, PCEE, PCEF, PCEB : r_bit;
begin
    ---------
    SYSCLK: process(RUN, CLK, STOP)
      begin
        if (RUN='1') and (not RUN'stable) then
```

Figure 29. Controller source code in VHDI.

```
            CLK <= true;
         elsif (RUN='1') and (STOP= '0')then
            CLK <= transport not CLK after PER;
       end if;
   end process SYSCLK;
---------
RUN_P: process(RUN)
   begin
      if RUN='1' then
          STOPR <= '0' ; else
          STOPR <= '1' ;
      end if;
   end process RUN_P;
---------
STATE: process(RUN, CLK, CSEN, STOP)
   begin
      if (not RUN'stable) and (RUN='1') then
          FETCH<=true;
          EXECUTE<=false;
      elsif (not CLK'stable) and (CLK)
          and (RUN='1') and (CSEN='1') then
          FETCH<=true;
          EXECUTE<=false;
      elsif  (not CLK'stable) and (CLK)
          and (RUN='1') and (CSEN='0')  then
          FETCH<=false;
          EXECUTE<=true;
      end if;
   end process STATE;
---------------
FETCH_P: process
   begin
      wait on FETCH until FETCH;
          PCEF<='1' after MADEL;              ---Memory address delay
          RENF<='1' after ODEL;
          RDF <= '1' after ODEL;
          WRITEF <= '0' after ODEL;
          IRLENF <='1' after RDDEL ;
          wait for RDDEL;
          IRF <= DATA;
```

Figure 29. Controller source code in VHDL (continued)

```
        PCEF<='0' after ODEL;
        IRLENF<='0' after ODEL;
        RDF <='0' after ODEL;
        INCF<='1' after ODEL,                    ---Inertial delay example
                '0' after ODEL+RDDEL ;
        RENF<='0' after ODEL+RDDEL;
        CSENF<='0';
end process FETCH_P;
-----------------
EXECUTE_P: process
  begin
        wait on EXECUTE until EXECUTE;
        CSENE<='0';
        case IRF is
        when "0001" =>                                  ---LDA
                PCEE<='1' after MADEL;
                RENE<='1'after ODEL;
                RDE<='1' after ODEL;
                WRITEE<='0' after ODEL;
                TLENE<='1' after ODEL;
                wait for RDDEL;
                PCEE<='0' after ODEL;
                TLENE<='0' after ODEL;
                RDE<='0' after ODEL;
                RENE<='0' after ODEL+RDDEL;
                wait for PER;
                wait on CLK;
                TENE<='1' after MADEL;
                RDE<='1' after ODEL;
                RENE<='1' after ODEL;
                wait for RDDEL;
                ALENE<='1'  after ENDEL,
                          '0' after DISDEL;
                TENE<='0' after MADEL;
                RDE<='0' after ODEL;
                RENE<='0' after ODEL;
                INCE<='1' after ODEL,
                        '0' after ODEL+RDDEL;
                RENE<='0' after ODEL+RDDEL;
```

Figure 29. Controller source code in VHDL (continued)

```
        when "0100" =>                                  ---STA
                PCEE<='1' after MADEL;
                RENE<='1' after ODEL;
                RDE<='1' after ODEL;
                WRITEE<='0' after ODEL;
                TLENE<='1' after ODEL;
                wait for RDDEL;
                TLENE<='0' after ODEL;
                PCEE<='0' after ODEL;
                RDE<='0' after ODEL;
                RENE<='0' after ODEL+RDDEL;
                wait for PER;
                TENE<='1' ;
                wait on CLK;
                AENE<='1' after MADEL;
                TENE<='1' after ODEL;
                RENE<='1' after ODEL;
                WRITEE<='1' after ODEL;
                wait for WDEL;
                AENE<='0' after ODEL;
                TENE<='0' after ODEL;
                WRITEE<='0' after ODEL;
                INCE<='1' after ODEL,
                        '0' after ODEL+RDDEL ;
                RENE<='0' after ODEL+RDDEL;
        ----------------------
        ----------------------
        when others =>                                  ---HALT
                STOPE<='1' after ODEL;
        end case;
        CSENE<='1';
  end process EXECUTE_P;
  ---------------                               ---OUTPUT SIGNAL
ALENB<=ALENE when not ALENE'quiet else          ---ACC
          ALENB;
ALEN<=ALENB;
AENB<=AENE when not AENE'quiet else
        AENB;
```

Figure 29. Controller source code in VHDL (continued)

```
AEN<=AENB;
BLENB<=BLENE when not BLENE'quiet else          ---B_REG
        BLENB;
BLEN<=BLENB;
ALUENB<=ALUENE when not ALUENE'quiet else      ---ALU
         ALUENB;
ALUEN<=ALUENB;
IRLENB<=IRLENF when not IRLENF'quiet else      ---IR
         IRLENB;
IRLEN<=IRLENB;
TLENB<=TLENE when not TLENE'quiet else         ---TAR
         TLENB;
TLEN<=TLENB;
TENB<= TENE when not TENE'quiet else
        TENB;
TEN<=TENB;
INCB<=INCF when not INCF'quiet else            ---PC
       INCE when not INCE'quiet else
       INCB;
INC<=INCB;
PCEB<=PCEE when not PCEE'quiet else
        PCEF when not PCEF'quiet else
        PCEB;
PCE<=PCEB;
PCLB<=PCLE when not PCLE'quiet else
        PCLB;
PCL<=PCLB;
RENB<=RENF when not RENF'quiet else            ---RAM
        RENE when not RENE'quiet else
        RENB;
REN<=RENB;
RDB<=RDF when not RDF'quiet else
      RDE when not RDE'quiet else
      RDB;
RD<=RDB;
WRITEB<=WRITEF when not WRITEF'quiet else
          WRITEE when not WRITEE'quiet else
          WRITEB;
WRITE<=WRITEB;
```

Figure 29. Controller source code in VHDL (continued)

```
CSEN<=CSENF when not CSENF'quiet else        ---CHANGE STATE
      CSENE when not CSENE'quiet else             ---ENABLE
      CSEN;
STOP<=STOPR when not STOPR'quiet else                ---STOP CLOCK
      STOPE when not STOPE'quiet else
      STOP;

end arcc;
```

Figure 29. Controller source code in VHDL (continued)

Compare the behavior model in Figure 18 to the data flow model in Figure 29, it is obvious that this controller is much more complicated. The model in Figure 18 has one deficiency which is that no timing behavior is modeled at all. The W-4 behavior model in Figure 18 has no time modeling accuracy. While in this controller, the data flow model involves the time modeling. Examples are the delays generics declared in the test_bench. The values of the delays will affect the signal's transfer timing in the system.

The controller consists of 5 processes, RUN_P, SYSCLK, STATE, FETCH_P, and EXECUTE_P. These 5 processes and the output signals are executed in concurrence. Concurrency is a significant feature in VHDL language. The signal network of the controller is shown in Figure 30.

The RUN_P process is first invoked by setting the signal RUN to '1', and it generates signal STOP to control the process SYSCLK, and STATE.

The SYSCLK process is aroused by changing the sensitivity signals, RUN, CLK, and STOP, and then creating the CLK signals to stimulate the STATE and EXECUTE processes. Meanwhile, the SYSCLK process sends a feedback signal to itself.

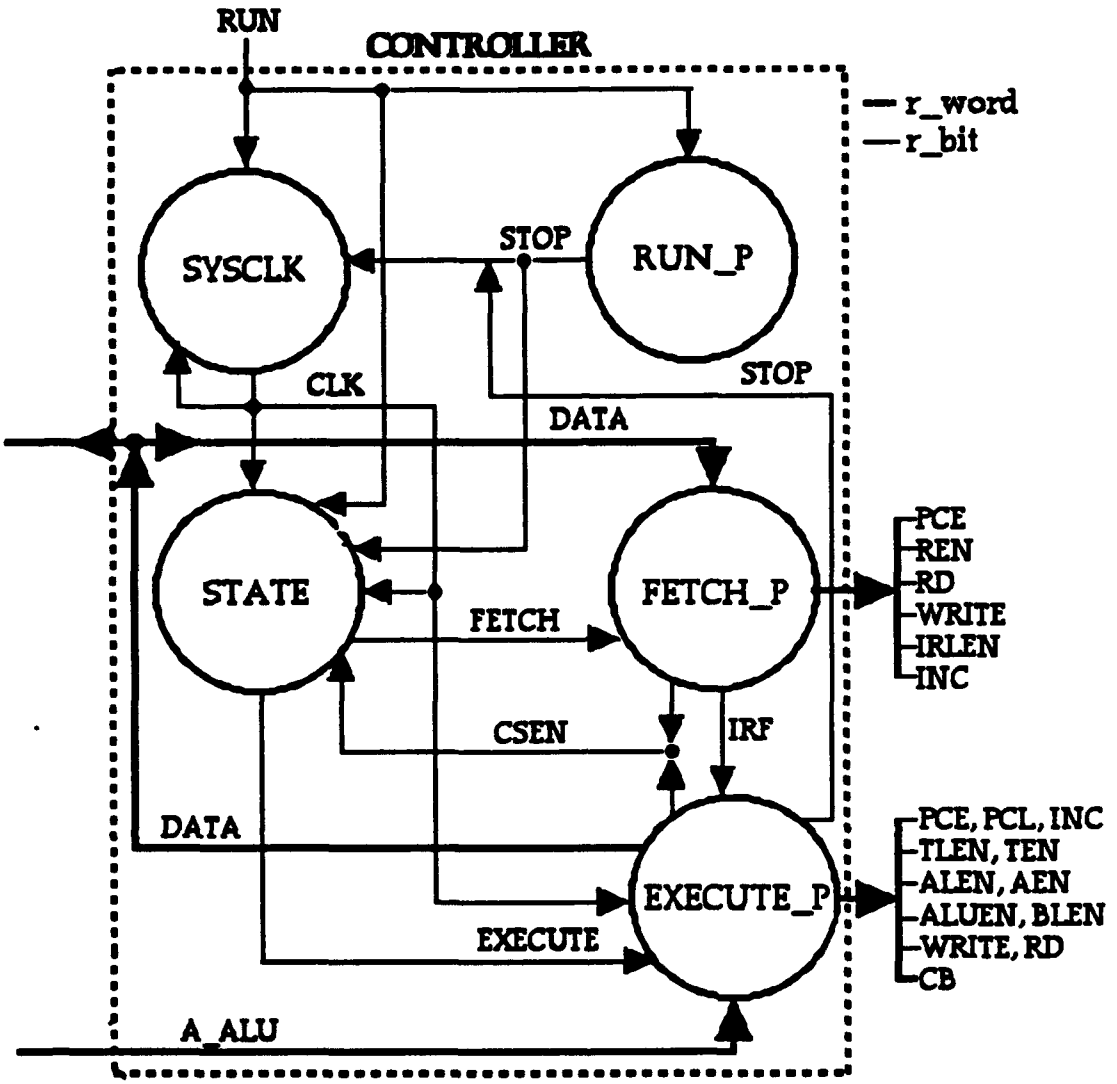


Figure 30. Controller signal network between processes

The STATE process is used to generate the FETCH and EXECUTE state signals for controlling and arousing the activities of "fetch", or "execute" processes. When the RUN signal is changed from '0' to '1', or the CLK signal is changed from '0' to '1', and both RUN and CSEN signals are '1', the STATE

process provides a FETCH signal. Otherwise, the STATE process yields an EXECUTE signal.

The first activity signal in the FETCH process yields the PCE signal, which stimulates the program counter (PC) to transfer the contents through the ADDR_B bus to the RAM component. Next, it provides REN, RD, and IRLEN signals to arouse the RAM component to read the specified contents output to the DATA_B bus. Then, the IR component loads in the data from the DATA_B bus. At the last step, it calls inc_word function to increase the PC value by one, changes the CSEN signal to terminate the FETCH process, and invokes the EXECUTE process.

The last EXECUTE process is aroused by the CSEN signal. It decodes the internal signal IRF, which are the 16 instructions shown in Figure 16. In Figure 29, the decoding is shown with only 2 of sixteen instructions as compared to Figure 18. The decoding function uses the delay generics, the "wait", the "after", the CLK, and the sequential assignments in process to produce the proper signals, and it also ceases the enable signal at the right time. To arouse the other 7 components which are ACC, ALU, B_REG, IR, TAR, PC, and RAM, there are 15 signals created in this process. Figure 31 exhibits the input output relationship of the signal network.

In order to transfer the correct signals at the right time, all the output signals through the port must use the local buffers to hold the values. For example, the "WRITE" signal uses the buffer signal "WRITEB" for transferring the correct signal.

	IN		OUT		INOUT
	r_word	r_bit	r_word	r_bit	r_word
CONTROLLER	A_ALU	RUN	CB	IRLEN, REN TLEN, TEN INC, PCL, PCE ALEN, AEN, ALUEN RD, WRITE, BLEN	DATA_B
RAM	ADDR_B RAM_I (r_type)	REN RD WRITE	RAM_O (r_type)		DATA_B
IR	DATA_B	IRLEN	IRO		
TAR	DATA_B	TLEN TEN	ADDR_B		
PC		INC, PCL PCE			ADDR_B
ACC		ALEN AEN	A_ALU		DATA_B
B_REG	DATA_B	BLEN	B_ALU		
ALU	A_ALU B_ALU CB	ALUEN	DATA_B		

Figure 31. Signals input / output relationship among the processes

9. Test Bench

The test bench is a top entity in the data flow model which includes other entities and architecture bodies. Since a test bench entity is not interfaced with any other units, there is no need to declare signals in the port.

The configuration specification is included in the architecture body, then, all the components are instantiated to implement the interconnect of the W-4 computer system. Figure 32 shows these statements. The comment symbol "-- " is used to indicate different portion of the architecture.

```
entity tb is end tb;
use work.pack3.all;
architecture arctb of tb is
    component controller                                    ---CONTROLLER
      generic(MADEL, WDEL, ODEL, RDDEL, ENDEL, DISDEL, PER :
              TIME);
      port(  DATA  : inout r_word;
             RUN   : in r_bit;
             A_ALU : in r_word;
             IRLEN, REN, TLEN, TEN, INC, PCL, PCE    : out r_bit;
             ALEN, AEN, BLEN, ALUEN, RD, WRITE : out r_bit;
             CB : out r_word);
      end component;
    component ram                                           ---RAM
      generic(RDDEL,DISDEL :TIME);
      port(  RAM_I : in r_type;
             DATA  : inout r_word;
             MA    : in    r_word;
             RD, WRITE,REN  : in r_bit;
             RAM_O : out r_type);
    end component;
    component ir                                            ---IR
      port(  DATA_I    : in r_word;
             IRLEN       : in r_bit;
             IRO         : out r_word);
    end component;
    component tar                                           ---TAR
      generic(DISDEL : TIME);
      port(  TARI : in r_word;
             TARO : out r_word;
             TLEN, TEN : in r_bit);
    end component;
    component pcc                                           ---PCC
      generic(DISDEL : TIME);
      port(  PC  : inout r_word;
             INC, PCL, PCE : in r_bit);
    end component;
    component acc                                           --ACC
```

Figure 32. System test bench source code in VHDL

```
    generic(DISDEL : TIME);
    port( DATA  : inout r_word;
          A_ALU  : out r_word;
          ALEN, AEN  : in r_bit);
  end component;
  component breg                                    ---B_REG
    port( DATAI : in r_word;
          B_ALU : out r_word;
          BLEN  : in r_bit);
  end component;
  component alu                                     ---ALU
    generic(DISDEL : TIME);
    port( ALUO  : out r_word;
          A_ALU, B_ALU, CB  : in r_word;
          ALUEN : in r_bit);
  end component;
  for all:control use entity work.controller(arcc);  ---CONFIGURATION
  for all:ram use entity work.ram(arcram);           ---SPECIFICATION
  for all:ir use entity work.ir(arcir);
  for all:tar use entity work.tar(arctar);
  for all:pcc use entity work.pcc(arcpc);
  for all:acc use entity work.acc(arcacc);
  for all:breg use entity work.breg(arcb);
  for all:alu use entity work.alu(arcalu);
  signal REN, IRLEN, RUN, RD, WRITE, TLEN, TEN : r_bit;
  signal INC, PCL, PCE, ALEN, AEN, BLEN, ALUEN : r_bit;
  signal ACC_O, IRO, A_ALU, B_ALU, CB : r_word;
  signal ADDR_B, DATA_B : r_word:="ZZZZ";       ---Initial value setting
  signal RAM_I : r_type :=("0001","1110","0100","1111","1111",
                           "1111","1111","1110","0111","1110",
                           "1110","1100","1101","1111","1100",
                           "0001");
  signal RAM_O : r_type;
begin                                    --- COMPONENT INSTANTIATION
  C1:controller
    generic map(40ns, 100ns, 50ns, 150ns, 5ns, 40ns, 500ns)
    port map (DATA_B, RUN, A_ALU, IRLEN, REN, TLEN, TEN, INC,
              PCL, PCE, ALEN, AEN, BLEN, ALUEN, RD, WRITE, CB);
  R1:ram
```

Figure 32. System test bench source code in VHDL (continued)

```
    generic map(100ns, 30ns)
    port map (RAM_I, DATA_B, ADDR_B, RD, WRITE, REN, RAM_O);
  I1:ir
    port map (DATA_B, IRLEN, IRO);
  T1:tar
    generic map(5ns)
    port map (DATA_B, ADDR_B, TLEN, TEN);
  P1:pcc
    generic map(5ns)
    port map (ADDR_B, NC, PCL, PCE);
  A1:acc
    generic map(5ns)
    port map (DATA_B, A_ALU, ALEN, AEN);
  B1:breg
    port map (DATA_B, B_ALU, BLEN);
  ALU1:alu
    generic map(5ns)
    port map (DATA_B, A_ALU, B_ALU, CB, ALUEN);
    RUN <= '1' ;
end arctb;
```

Figure 32. System test bench source code in VHDL (continued)

In order to hook up the components into a circuit, the component declarations make the components inside the design unit visible. The component ports are called local ports. The actual ports of a component is associated with a component instantiation statement, which states that there is a specific instance of the component.

For example, for b_reg the signals DATA_B, B_ALU, and BLEN of the actual port are connected to DATAI, B_ALU, and BLEN of the local ports. Thus, there are three lists used to hook up a component, the local ports in the component declaration, the actual ports in the instantiation, and the formal ports in the component entity.

The binding between a component instance and the design entity of a library is accomplished by a configuration specification, which uses the reserved key word "for all" and "use entity". Such configuration specification can be applied to all instantiations of the given component as shown in Figure 32.

Associating generic constants are similar to the association of ports in VHDL. An example is the generic in the component declaration which relates to the generic map in the component instantiation statement. The different setting of values of generic constants in the generic map will affect the simulation behavior using this constants.

The data flow model in this chapter has been used to run the same demonstration program set in the RAM of the test bench in Figure 32 as in the behavior model. It is doing the same operations, which is LDA and STA (read and write). The result is shown in Figure 33.

The ADDR_B shows the changing value of the address bus. The IRO shows the correct instructions. The A_ALU is the value of the "ACC", while the RAM_O(14) and RAM_O(15) shows the contents of the memory at location 14 and 15.

The ADDR_B is "ZZZZ" at all time unless the enable signal of a component has been activated for transfer. By checking the result of IRO which shows "0001", "0100", "1111" in sequence, and the result of A_ALU, the value "1100" has been transfered to RAM_O(15). All those results are the same as that of the behavior model except the fact that the timing in the data flow model was absent in the behavior simulation.

TIME(ns)	ADDR_B	IRO	A_ALU	RAM_O(14)	RAM_O(15)
0	"0000"	"0000"	"0000"	"0000"	"0000"
+1				"1100"	"0001"
5	"ZZZZ"				
40					
+3	"0000"				
150		"0001"			
205	"ZZZZ"				
1040					
+3	"0001"				
1205	"ZZZZ"				
2040					
+3	"1110"				
2190					
+3			"1100"		
2195	"ZZZZ"				
3040					
+3	"0010"				
3150					
+3		"0100"			
3205	"ZZZZ"				
4040					
+3	"0011"				
4205	"ZZZZ"				
4650					
+4	"1111"				
5150					"1100"
5155	"ZZZZ"				
6040					
+3	"0100				
6150					
+3		"1111"			
6205	"ZZZZ"				

Figure 33. Simulation result of the dataflow model

This is not the only way to represent the W-4 computer into a VHDL data flow model. On in experiment the activities of any signal can be traced in the simulation result for verification. Even the most difficult time activities can be

interpreted with accuracy here. Some modification to the data flow model can further improve the accuracy as desired. All these significant characteristics of the data flow model of the experience learned in this work will be discussed in the next chapter.

IV. EXPERIMENTS OF THE SYSTEM MODEL SIMULATION

There are five sections in this chapter to discuss some experiences obtained in the experiments. The first section talks about experiences of the basic VHDL concept, the data type, the mode, the methods of controlling the data flow, and the attributes. The second section discusses the concurrence and sequence to see when they happen. The third section discusses how the initial value affects the default value in the VHDL program. The fourth section discusses and demonstrates the usefulness accuracy and time simulation. Clock cycle, read write delay, and time modeling which shows how the time modeling can verify the signal flow design are also discussed. The last section states the inertial delay. How the inertial delay affects the signal transfer and how to compensate this problem.

A. EXPERIENCE OF THE BASIC CONCEPT

1. Data Type

As mentioned before, the VHDL language is a strongly "typed" language. The inadvertent mixing of different types in an operation will be notified as an error. For example,

```
TARO <= TARI;
```

The TARO is a "r_word" type. The TARI must be a "r_word" type too. If the TARI is defined as a 4 bits vector type, there will be an error, Because the TARO is a resolved 4 bits vector type. Another example,

```
PCE <= PCI;
```

If the PCE is a "r_bit" type, and the PCI is defined as a "bit" type which can have only 'Z', '0' and '1', there will also be an error. This is the same situation as the previous example. As a result, the type of an object is always the first thing to be concerned with. It means that the type of the right side object of the symbols "<=" and ":=" must have the same as the type as of the left side object.

2. Mode

There are three modes "in", "out", and "inout" associated with the signals. The mode of the object is specified in a port declarations. Consequently, it must put on the correct side of the statement. For example,

```
TEN <= TLEN;
```

If the TEN is of "in" mode or the TLEN is of "out" mode, then the error will occur. For this example, the TEN must be of "out" or "inout" mode, while the TLEN must be of "in" or "inout" mode.

3. Methods of Controlling the Dataflow

In this thesis, many ways are used to control the signals flow. The data transfer are controlled by statements with "if", "after delay", and "wait". Actually, the high impedance state "ZZZZ" plays a very important role to let the

correct signals flow through the bus to the other parts at the correct time. Note that there are two rules to follow:

- First, the "after" phrase can not be used after the variable assignment. In example 1, the statement "after 5 ns" can not be used after the variable assignment "ir := RAM(bitarray_to_int(P_C))".

```
example 1:
    entity behav3 is end behav3;
    use work.pack3.all;
    architecture arcbehav3 of behav3 is
            ----------
            ----------
            process(P_C)
                    variable ir, b_reg, tar : r_word;
                    variable int_pc, int_tar, int_acc : integer;
            begin
                    ir := RAM(bitarray_to_int(P_C)) after 5 ns;  ---error
                    int_pc := bitarray_to_int(P_C)+1;
                    case ir is
                      when "0001" =>                    ---LOAD
                            IN_STR <= ir;
                            int_tar :=bitarray_to_int(RAM(int_pc));
                            A_CC <= RAM(int_tar);
                            int_pc := int_pc+1;
                            P_C <= int_to_bitarray(int_pc) ;
                            ------------------
                            ------------------
                     end case;
            end process;
    end arcbehav3;
```

- Second, the "if" statement can only be used in the process for sequential execution. In example 2, it generates an error, because the "case" and "if" statements are not in the process for sequential execution.

example 2:

```
use work.pack3.all;
entity alu is
        generic(DISDEL: TIME);
        port(  ALUO  : out r_word;
               A_ALU, B_ALU, CB : in r_word;
               ALUEN : in r_bit);
end alu;
------------
architecture arcalu of alu is
        signal ALUB : r_word;
        variable addb : r_word;
begin
        case CB is                                    ---error
        when "0011" =>                          ---ADD,SHAL
               addb:=addv(A_ALU, B_ALU);
        -------------
        -------------
        end case;
        if ALUEN = '1' then ALUO <= addb;   ---error
               else ALUO <= "ZZZZ" after DISDEL; end if;
end arcalu;
```

4. Attributes

The attributes constructs are used often in this research work. It is a very useful feature to check the status of the signal in the VHDL language. For example, in hardware design a designer must often deal with positive edge or negative edge of a signal as shown in example 1.

example 1:

In the following, the example 2 is selected to explain the attribute feature. The bold statements "(not CLK'stable) and (CLK)" mean that the condition is "true" when the CLK changes state from '0' to '1'. The other bold statements (RUN='1') and (not RUN'stable) also mean that the condition is "true" whenever the RUN signal change value from '0' to '1'. These are used for decision test in the VHDL language.

Example 2:

```
--------------
--------------
SYSCLK: process(RUN, CLK, STOP)
begin
   if (RUN='1') and (not RUN'stable) then
       CLK <= true;
    elsif (RUN='1') and (STOP= '0') then
       CLK <= transport not CLK after PER;
   end if;
end process SYSCLK;
---------
STATE: process(RUN, CLK, CSEN, STOP)
begin
   if (not RUN'stable) and (RUN='1') then
       CSENS<='1';
       FETCH<=true;
       EXECUTE<=false;
   elsif (not CLK'stable) and (CLK) and (STOP='0')
       and (RUN='1') and (CSEN='1') then
       FETCH<=true;
       EXECUTE<=false;
   elsif (not CLK'stable) and (CLK) and (STOP='0')
       and (RUN='1') and (CSEN='0') then
       FETCH<=false;
       EXECUTE<=true;
   end if;
end process STATE;
---------------
---------------
```

B. CONCURRENCE AND SEQUENCE

In this section, the concurrence and the sequence related problem will be discussed. The ability to support concurrency is a significant feature in the VHDL. The concurrent statement or process executes whenever the sensitive signals change values. There are three ways to express concurrency. They are by concurrent signal assignment, component instantiation, and process. Figure 34 shows a part of the controller program. It shows many concurrent signal assignments.

```
----------------
------------------
RENB<=RENF when not RENF'quiet else          ---RAM
       RENE when not RENE'quiet else
       RENB;
REN<=RENB;
RDB<=RDF when not RDF'quiet else
       RDE when not RDE'quiet else
       RDB;
RD<=RDB;
WRITEB<=WRITEF when not WRITEF'quiet else
          WRITEE when not WRITEE'quiet else
          WRITEB;
WRITE<=WRITEB;
CSEN<=CSENF when not CSENF'quiet else        ---CHANGE
       CSENE when not CSENE'quiet else       ---STATE ENABLE
       CSENS when not CSENS'quiet else
       CSEN;
---------------
---------------
```

Figure 34. Concurrence via signal assignment

All the signal assignments execute at the same time whenever any signal on the right hand side of the assignment have changed its value. For example, REN and RD signals may execute at the same simulation time. Figure 35 displays another way to express the concurrency by component instantiation. This is a test_bench example.

```
-------------
-------------
begin                                   --- COMPONENT INSTANTIATION
    C1:controller
      generic map(40ns, 100ns, 50ns, 150ns, 5ns, 40ns, 500ns)
      port map (DATA_B, RUN, A_ALU, IRLEN, REN, TLEN, TEN, INC,
                PCL, PCE, ALEN, AEN, BLEN, ALUEN, RD, WRITE, CB);
    R1:ram
      generic map(100ns, 30ns)
      port map (RAM_I, DATA_B, ADDR_B, RD, WRITE, REN, RAM_O);
    I1:ir
      port map (DATA_B, IRLEN, IRO);
    T1:tar
      generic map(5ns)
      port map (DATA_B, ADDR_B, TLEN, TEN);
    P1:pcc
      generic map(5ns)
      port map (ADDR_B, NC, PCL, PCE);
    A1:acc
      generic map(5ns)
      port map (DATA_B, A_ALU, ALEN, AEN);
    B1:breg
      port map (DATA_B, B_ALU, BLEN);
    ALU1:alu
      generic map(5ns)
      port map (DATA_B, A_ALU, B_ALU, CB, ALUEN);
      RUN <= '1' ;
end arctb;
--------------
--------------
```

Figure 35. Concurrence via the component instantiation

The components such as "controller", "ram", "ir", "pcc", "tar", "acc", "breg", and "alu" are executed at the same time when the signal RUN invokes this test_bench. When the port signals of the components did not change its value, this component will be suspended until the next sensitive signals change. To express concurrence via processes is shown in Figure 36.

```
---------------
---------------
SYSCLK: process(RUN, CLK, STOP)
  begin
    if (RUN='1') and (not RUN'stable) then
        CLK <= true;
        elsif (RUN='1') and (STOP= '0')then
        CLK <= transport not CLK after PER;
    end if;
  end process SYSCLK;
  ---------
RUN_P: process(RUN)
  begin
    if RUN='1' then
        STOPR <= '0' ; else
        STOPR <= '1' ;
    end if;
end process RUN_P;
---------
STATE: process(RUN, CLK, CSEN, STOP)
  begin
    if (not RUN'stable) and (RUN='1') then
        CSENS<='1';
        FETCH<=true;
        EXECUTE<=false;
    elsif (not CLK'stable) and (CLK) and (STOP='0')
        and (RUN='1') and (CSEN='1') then
        FETCH<=true;
```

Figure 36. Concurrence via the process

```
        EXECUTE<=false;
    elsif (not CLK'stable) and (CLK) and (STOP='0')
        and (RUN='1') and (CSEN='0') then
        FETCH<=false;
        EXECUTE<=true;
    end if;
end process STATE;
---------------
---------------
```

Figure 36. Concurrence via the process (continued)

There are three processes in Figure 36, SYSCLK, RUN_P, and STATE. They are activated when the sensitivity signals change. The sensitivity signals appear in the parenthesis after the "process" keyward. For example, the RUN signal can invoke these three processes at the same time when RUN is changed.

The sequency in VHDL is supported just like any other general programming language. The statements "IF, THEN, ELSE", "CASE", "LOOP, NEXT, EXIT, RETURN" and "variable assignment" can occur in a package or in a process. Even the signal assignment statements can occur in a process where they are executed also in sequence.

C. INITIAL VALUE SETTING

The default value could be set in the port of an entity, component declaration, or a block statement. In VHDL, only ports of modes "out", "inout" can have drivers [Ref. 4]. The driver of every signal is defined to have an "implicit default value" shown in Figure 37. Both "implicit default value" of signals DATAI and B_ALU, are "0000" which is the left element of the defined type. And the signal BLEN is '0'.

```
entity breg is
        port(  DATAI : in r_word;
               B_ALU : out r_word;
               BLEN  : in r_bit);
end breg;
```

Figure 37. Implicit default value setting

It is also possible to explicitly specify a default value in the declaration of the signal or port. In this case, the "explicit default value" will override the implicit default value as shown in Figure 38. The explicit default value of the signal B_ALU is set as "ZZZZ" instead of the implicit default value "0000".

```
entity breg is
        port(  DATAI : in r_word;
               B_ALU : out r_word := "ZZZZ";
               BLEN  : in r_bit);
end breg;
```

Figure 38. Explicit default value setting

The simulation of the system test bench in Figure 32 is essentially. But, there is one flaw that the initial values of ADDR_B, IRO, and A_ALU are "0000" caused by the "implicit default values". They are not affected by the initial value setting shown in Figure 32, where DATA_B and ADDR_B are set to "ZZZZ". Anticipation of the ADDR_B, IRO, and A_ALU in high impedance state, "ZZZZ" are not seen at the beginning of the simulation.

In this case, The problem is that both signal DATA_B and signal ADDR_B are of mode "inout", hence the initial values of DATA_B and ADDR_B will be taken from the formal ports where they are connected to.

To solve this problem, a modified program is attached in appendix C. The explicit "ZZZZ" default values are set into eight places of formal ports of the components to override the "implicit default value", "0000". These formal port signals are the DATA in the "control entity", the IRO in the "ir entity", the B_ALU in the "breg entity", the ALU_O in the "alu entity", the DATA and A_ALU in the "acc entity", the DATA in the "ram entity", the P_C in the "pc entity", and the TARO in the "tar entity".

TIME(ns)	ADDR_B	IRO	A_ALU	RAM_O(14)	RAM_O(15)
0	"ZZZZ"	"ZZZZ"	"ZZZZ"	"0000"	"0000"
+1			"ZZZZ"	"1100"	"0001"
5					
40					
+3	"0000"				
150		"0001"			
205	"ZZZZ"				
1040					
+3	"0001"				
1205	"ZZZZ"				
2040					
+3	"1110"				
2190					
+3			"1100"		
2195	"ZZZZ"				
3040					
+3	"0010"				
3150					
+3		"0100"			
3205	"ZZZZ"				
4040					
+3	"0011"				
4205	"ZZZZ"				
4650					
+4	"1111"				
5150					"1100"
5155	"ZZZZ"				
6040					
+3	"0100				
6150					

```
+3              "1111"
6205 "ZZZZ"
```

Figure 39. Result of the modified program

The simulation results of the modified source program are shown in Figure 39. Where we can see that everything of Figure 33 is the same as that of Figure 39 except the initial value at the beginning of the simulation.

D. TIME MODELING AND ACCURACY

The advantage of the VHDL is able to handle time modeling with accuracy. Hence, in the thesis, the two important issues are that how time simulation results affects the signal flow design, and how accurately can timing model be achieved with VHDL. The following examples and explanation will demonstrate these issues.

1. Clock Cycle

The instructions of the W-4 computer system is executed according to clock cycles as shown in Figure 17. The LDA instruction takes one clock for "fetch" and two clocks for execution. The clock cycles for all instructions are from two to four. Hence, in the "controller" program the clock cycle is affected by the "PER" delay which will make the W-4 computer as accurate as the hardware implementation.

In the thesis, each clock cycle period is defined as 1000 ns. In Figure 39, a total of 6000 ns are used to complete 2 instructions, "LDA" and "STA", which means that 6 clock cycles have been used.

2. Read and Write Delay

There are two delay factors in the RAM memory, which are the "read" and "write" delay. The read and write timing are shown in Figure 40 and 41.

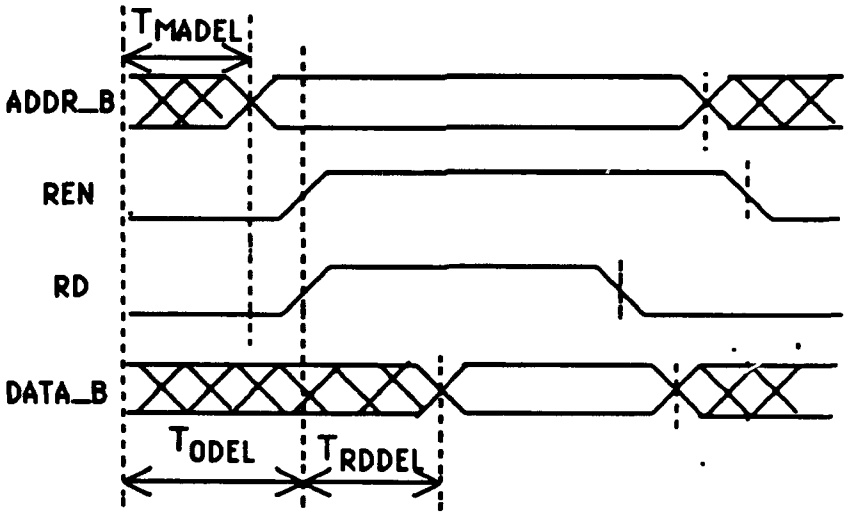


Figure 40. Read timing of RAM

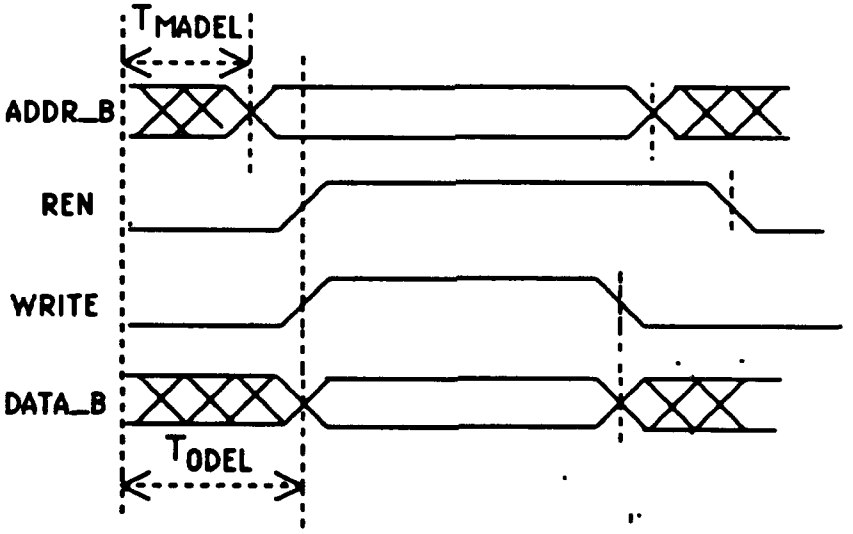


Figure 41. Write timing of RAM

In Figure 40 and 41, the delays are expressed as follows:

T_{MADEL} = 40 ns (memory address delay)
T_{ODEL} = 50 ns (output delay)
T_{RDDEL} = 100 ns (read delay)

It is known that the ADDR_B should occur earlier than REN and RD in Figure 40, or earlier than REN and WRITE in Figure 41. For proper transfer values, if the "memory address delay" TMADEL is extended to 50 ns in the FETCH_P process of the control architecture, that means the "delay MADEL" is equal or greater than the "delay ODEL", then there is an error occurred. Figure 42 shows the part of the flow of the correct simulation results, while Figure 43 shows the wrong simulation results.

	ADDR_B	INC	PCE	RD	DATA_B	IRO	IRLEN
3050ns	"ZZZZ"	"0"	"0"	"0"	"ZZZZ"	"0001"	"0"
3150ns	"0010"	"0"	"1"	"1"	"ZZZZ"	"0001"	"0"
3155ns	"0010"	"0"	"1"	"1"	"0100"	"0001"	"1"
3200ns	"0010"	"0"	"1"	"1"	"0100"	"0001"	"1"
3205ns	"0010"	"1"	"0"	"0"	"0100"	"0001"	"0"
3210ns	"ZZZZ"	"1"	"0"	"0"	"0100"	"0001"	"0"

Figure 42. Original signal transactions

	ADDR_B	INC	PCE	RD	DATA_B	IRO	IRLEN
3050ns	"ZZZZ"	"0"	"0"	"0"	"ZZZZ"	"0001"	"0"
3150ns	"0010"	"0"	"1"	"1"	"ZZZZ"	"0001"	"0"
3155ns	"0010"	"0"	"1"	"1"	"0001"	"0001"	"1"
3200ns	"0010"	"0"	"1"	"1"	"0001"	"0001"	"1"
3205ns	"0010"	"1"	"0"	"0"	"0001"	"0001"	"0"
3210ns	"ZZZZ"	"1"	"0"	"0"	"0001"	"0001"	"0"

Figure 43. Signal transactions for wrong MADEL

Compare Figure 43 to Figure 42, it is clear that the DATA_B is still "0001", where the correct value should be "0100". This shows that the "delay MADEL" must be shorter than any other enable signals in the RAM component. This demonstrate how the timing requirement can be checked in the VHDL system simulation.

E. INERTIAL DELAY

The "inertial delay" affects the assignment on the projected output waveform of the signal. It is used to represent signals which require the value on inputs to persist for a given time before the signal respond. It happens a lot in VHDL programming. For example, a form is shown as following:

```
signal-name <= value after time-expression;
```

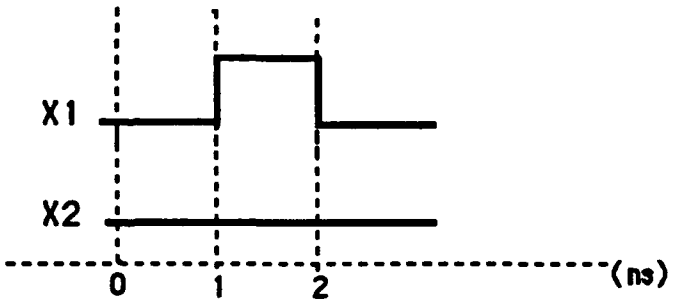
This is a quite simple assignment though. When there are two more assignments to express the same signal of the value transfer in the process, it will be a little tricky.

```
entity inert_del is
        port(  X :  inout  r_bit := '0'; Y :  out r_bit );
end inert_del;
architecture arc_inert_del of inert_del is
begin
        process(X)
        begin
                Y <= X;
                X <= '1' after 1 ns;
                X <= '0' after 2 ns;
        end process;
end arc_inert_del;
```

Figure 44. Example of an inertial delay

Because its effect on the driver of the signal is harder to see at the first glance. Figure 44 demonstrates this characteristic of the VHDL. After the execution of this process, most designers will think that the value of signal "X" must be '1' after 1 ns. The value of "X" will be changed to '0' after 2 ns shown as 'X1' in the following example. But, it does not act this way. There will be only a single transaction on the driver for X signal shown as 'X2' in the following example. Because the last assignment "X <= '0' after 2 ns;" overrides the first assignment "X <= '1' after 1 ns;", the value '1' can not be transfered to the X.

example :



In order to have these two assignments affecting the X, there are two solutions. First, the output X assignments can be adjusted as follows:

```
X <= '1' after 1 ns;
X <= transport '0' after 2 ns;
```

The "transport" is used to avoid the inertial effect built in the VHDL. The other method is to modify the output X assignments as follows:

```
X <= '1' after 1 ns,
     '0' after 2 ns;
```

The second method is used in this thesis often to get the correct answer.

For another example of this, the INCF signal in the FETCH_P process of Figure 29 has been changed to that in Figure 45:

```
          ----------
          ----------
FETCH_P: process
   begin
      wait on FETCH until FETCH;
          PCEF<='1' after MADEL;
          RENF<='1' after ODEL;
          RDF <= '1' after ODEL;
          WRITEF <= '0' after ODEL;
          IRLENF <='1' after RDDEL ;
          wait for RDDEL;
          IRF  <=  DATA;
          PCEF <= '0' after ODEL;
          IRLENF <= '0' after ODEL;
          RDF <= '0' after ODEL;
          INCF <= '1' after ODEL;
          INCF <= '0' after ODEL+RDDEL ;     ---modified for example
          RENF <= '0' after ODEL+RDDEL;      ---modified for example
          CSENF <= '0';
end process FETCH_P;
--------------
--------------
```

Figure 45. Modified source program for an inertial delay

Of course, the result of Figure 45 will be wrong. The value of '1' can not be transfered to the INCF. Because it is overridden by the second assignment value of '0'. Figure 46 is the simulation result of the correct program. Figure 47 is

the results after changing the assignment of the INCF for viewing the inertial delay problem.

	ADDR_B	INC	PCE	RD	RAM_O(14)	DATA_B	IRO	IRLEN
30ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"ZZZZ"	"ZZZZ"	"0"
40ns	"0000"	"0"	"1"	"1"	"1100"	"ZZZZ"	"ZZZZ"	"0"
50ns	"0000"	"0"	"1"	"1"	"1100"	"ZZZZ"	"ZZZZ"	"0"
100ns	"0000"	"0"	"1"	"1"	"1100"	"ZZZZ"	"ZZZZ"	"0"
150ns	"0000"	"0"	"1"	"1"	"1100"	"0001"	"0001"	"1"
155ns	"0000"	"0"	"1"	"1"	"1100"	"0001"	"0001"	"1"
200ns	"0000"	"1"	"0"	"0"	"1100"	"0001"	"0001"	"0"
205ns	"ZZZZ"	"1"	"0"	"0"	"1100"	"0001"	"0001"	"0"
210ns	"ZZZZ"	"1"	"0"	"0"	"1100"	"0001"	"0001"	"0"
350ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"0001"	"0001"	"0"
355ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"0001"	"0001"	"0"
380ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"ZZZZ"	"ZZZZ"	"0"

Figure 46. Signal transactions of the original program

	ADDR_B	INC	PCE	RD	RAM_O(14)	DATA_B	IRO	IRLEN
30ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"ZZZZ"	"ZZZZ"	"0"
40ns	"0000"	"0"	"1"	"1"	"1100"	"ZZZZ"	"ZZZZ"	"0"
50ns	"0000"	"0"	"1"	"1"	"1100"	"ZZZZ"	"ZZZZ"	"0"
100ns	"0000"	"0"	"1"	"1"	"1100"	"ZZZZ"	"ZZZZ"	"0"
150ns	"0000"	"0"	"1"	"1"	"1100"	"0001"	"0001"	"1"
155ns	"0000"	"0"	"1"	"1"	"1100"	"0001"	"0001"	"1"
200ns	"0000"	"0"	"0"	"0"	"1100"	"0001"	"0001"	"0"
205ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"0001"	"0001"	"0"
210ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"0001"	"0001"	"0"
350ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"0001"	"0001"	"0"
355ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"0001"	"0001"	"0"
380ns	"ZZZZ"	"0"	"0"	"0"	"1100"	"ZZZZ"	"ZZZZ"	"0"

Figure 47. Signal transactions for checking the "inertial delay"

Compare the signal INCF of Figure 47 with that of Figure 46, it's clear that the difference of signal INCF between time 200 ns to 210 ns is '0' due to inertial delay. Consequently, the program counter "PC" will not be incremented by one. This program will be suspended and yield no output.

In general, the contents of this chapter are concentrated on the experience gain in VHDL. They can not be ignored. Though there are many important features of the VHDL language, the issues discussed are enough to describe for the W-4 system data flow model implementation.

V. CONCLUSION

The primary objective of this research, to apply the VHSIC Hardware Description Language (VHDL) to the data flow model design of the W-4 computer, has been achieved. Not only is the source program of the data flow model completed, but also important experience using the VHDL has been obtained.

The data flow source program of the W-4 computer can be adopted by other designers by using a "use" clause before their programs. The main advantages of this program are:

- The machine model is expandable. The source program can be modified to extend to more than 4 bits.
- The simulation is traceable. The signal flow results can be traced for learning and verification.
- The timing is accurate. The timing can be modeled as close to the designed hardware characteristic as desired.
- The model is portable. The program can be used in any other VHDL environment.

In this research, all those benefits shown in the thesis match well with those published in the literature. There are still two flaws in the model source program. The "guard" and "block" constructs, which are also very important in the VHDL language, are not used. The model source program can be further developed to a structural description, where all components can be described at a lower level in gates.

Though, the VHDL language is complicated and takes time to become familiar with, the merits in time modeling and flexibility have been demonstrated in the previous chapters. The designers do not need to learn all language constructs. They can concentrate on a small portion of the large design.

Different parts of the design in VHDL can be connected easily just like the components used in the test bench.

Hence, from this experience the VHDL language is shown to be rich and powerful. It can handle large complex hardware structure in hierarchy.

APPENDIX A. PROGRAM OF BEHAVIOR DESCRIPTION

```
entity behav3 is end behav3;
use work.pack3.all;
architecture arcbehav3 of behav3 is
    signal    P_C   : r_word :="0000";
    signal    A_CC,IN_STR   :  r_word;
    signal    RAM   :  r_type:=("0001","1110","0100","1111","1111",
                               "1111","1111","1110","0111","1110",
                               "1110","1100","1101","1111","1100",
                               "0001");
begin
    process(P_C)
      variable ir,b_reg,tar : r_word;
      variable int_pc,int_tar,int_acc:integer;
    begin
      ir := RAM(bitarray_to_int(P_C));
      int_pc := bitarray_to_int(P_C)+1;
      case ir is
            when "0001" =>                                    ---LOAD
                  IN_STR <= ir;
                  int_tar :=bitarray_to_int(RAM(int_pc));
                  A_CC <= RAM(int_tar);
                  int_pc := int_pc+1;
                  P_C <= int_to_bitarray(int_pc) ;
            when "0010" =>                                    ---ADD
                  IN_STR <= ir;
                  tar := RAM(int_pc);
                  int_tar := bitarray_to_int(tar);
                  b_reg := RAM(int_tar);
                  int_acc := bitarray_to_int(A_CC)
                           +bitarray_to_int(b_reg);
                  while int_acc >= 16 loop
                        int_acc := int_acc rem 16;
                  end loop;
                  A_CC <= int_to_bitarray(int_acc);
                  int_pc := int_pc+1;
                  P_C <= int_to_bitarray(int_pc) ;
            when "0011" =>                                    ---SUB
                  IN_STR <= ir;
                  tar := RAM(int_pc);
```

```
        int_tar := bitarray_to_int(tar);
        b_reg := RAM(int_tar);
        int_acc        := bitarray_to_int(inv_w(b_reg))
                +bitarray_to_int(A_CC)+1;
        while int_acc >= 16 loop
                int_acc := int_acc rem 16;
        end loop;
        A_CC <= int_to_bitarray(int_acc);
        int_pc := int_pc+1;
        P_C <= int_to_bitarray(int_pc) ;
when "0100" =>                                  ---STA
        IN_STR <= ir;
        int_tar := bitarray_to_int(RAM(int_pc));
        RAM(int_tar) <=A_CC;
        int_pc := int_pc+1;
        P_C <=int_to_bitarray(int_pc) ;
when "0101" =>                                  ---AND
        IN_STR <= ir;
        tar := RAM(int_pc);
        int_tar := bitarray_to_int(tar);
        b_reg := RAM(int_tar);
        A_CC <= andv(A_CC,b_reg);
        int_pc := int_pc+1;
        P_C <= int_to_bitarray(int_pc) ;
when "0110" =>                                  ---OR
        IN_STR <= ir;
        tar := RAM(int_pc);
        int_tar := bitarray_to_int(tar);
        b_reg := RAM(int_tar);
        A_CC <= orv(A_CC,b_reg);
        int_pc := int_pc+1;
        P_C <= int_to_bitarray(int_pc) ;
when "0111" =>                                  ---XOR
        IN_STR <= ir;
        tar := RAM(int_pc);
        int_tar := bitarray_to_int(tar);
        b_reg := RAM(int_tar);
        A_CC <= xorv(A_CC,b_reg);
        int_pc := int_pc+1;
        P_C <= int_to_bitarray(int_pc) ;
when "1000" =>                                  ---LDI
        IN_STR <= ir;
        A_CC <= RAM(int_pc);
```

```
                int_pc := int_pc+1;
                P_C <= int_to_bitarray(int_pc) ;
            when "1001" =>                          ---JUMP
                IN_STR <= ir;
                tar := RAM(int_pc);
                P_C <= tar;
            when "1010" =>                                  ---JAN
                IN_STR <= ir;
                tar := RAM(int_pc);
                int_acc := bitarray_to_int(A_CC);
                if int_acc >=8 then
                        P_C <= tar;
                else
                        int_pc := int_pc+1;
                        P_C <= int_to_bitarray(int_pc);
                end if;
            when "1011" =>                                  ---INCA
                IN_STR <= ir;
                b_reg :="0000";
                int_acc := bitarray_to_int(A_CC)
                          +bitarray_to_int(b_reg)+1;
                A_CC <= int_to_bitarray(int_acc);
                P_C <= int_to_bitarray(int_pc) ;
            when "1100" =>                                  ---SHAL
                IN_STR <= ir;
                b_reg :=A_CC;
                A_CC <= addv(A_CC,b_reg);
                P_C <= int_to_bitarray(int_pc) ;
            when "1101" =>                                  ---CLA
                IN_STR <= ir;
                A_CC <= "0000";
                P_C <= int_to_bitarray(int_pc) ;
            when "1110" =>                                  ---CMA
                IN_STR <= ir;
                b_reg := "1111";
                A_CC <= xorv(A_CC,b_reg);
                P_C <= int_to_bitarray(int_pc) ;
            when others =>
                IN_STR<="1111" after 2 ns;
    end case;
  end process;
end arcbehav3;
```

APPENDIX B. PROGRAM OF CONTROLLER

```
use work.pack3.all;
entity control is
    generic(MADEL, WDEL, ODEL, RDDEL, ENDEL, DISDEL, PER :
            TIME);
    port (    DATA : inout r_word;
              RUN  : in r_bit;
              A_ALU: in r_word;
              IRLEN  : out r_bit;
              TLEN, TEN   : out r_bit;
              INC, PCL, PCE : out r_bit;
              ALEN, AEN, BLEN, ALUEN : out r_bit;
              CB : out r_word;
              REN, RD, WRITE : out r_bit);
end control;
------------
use work.pack3.all;
architecture arcc of control is
    signal IRF : r_word;
    signal CLK, EXECUTE, FETCH : boolean;
    signal CSEN, CSENE, CSENF, CSENS : r_bit;
    signal STOP, STOPE, STOPR : r_bit;
    signal RDB, RDE, RDF, IRLENF, IRLENB : r_bit;
    signal WRITEF, WRITEE, WRITEB : r_bit;
    signal RENF, RENE, RENB : r_bit;
    signal TLENE, TENE, TENB, TLENB : r_bit;
    signal ALENE, ALENB, AENE, AENB : r_bit;
    signal BLENE, BLENB : r_bit;
    signal ALUENE, ALUENB : r_bit;
    signal INCE, INCF, INCB : r_bit;
    signal PCLE, PCLF, PCLB, PCEE, PCEF, PCEB : r_bit;
begin
    ---------
    SYSCLK: process(RUN, CLK, STOP)
      begin
        if (RUN='1') and (not RUN'stable) then
            CLK <= true;
         elsif (RUN='1') and (STOP= '0')then
            CLK <= transport not CLK after PER;
        end if;
```

```
   end process SYSCLK;
---------
RUN_P: process(RUN)
   begin
      if RUN='1' then
          STOPR <= '0' ; else
          STOPR <= '1' ;
      end if;
   end process RUN_P;
---------
STATE: process(RUN, CLK, CSEN, STOP)
   begin
      if (not RUN'stable) and (RUN='1') then
          CSENS<='1';
          FETCH<=true;
          EXECUTE<=false;
      elsif (not CLK'stable) and (CLK) and (STOP='0')
          and (RUN='1') and (CSEN='1') then
          FETCH<=true;
          EXECUTE<=false;
      elsif (not CLK'stable) and (CLK) and (STOP='0')
          and (RUN='1') and (CSEN='0') then
          FETCH<=false;
          EXECUTE<=true;
      end if;
   end process STATE;
---------------
FETCH_P: process
   begin
      wait on FETCH until FETCH;
          PCEF<='1' after MADEL;
          RENF<='1' after ODEL;
          RDF <= '1' after ODEL;
          WRITEF <= '0' after ODEL;
          IRLENF <='1' after RDDEL ;
          wait for RDDEL;
          IRF <= DATA;
          PCEF<='0' after ODEL;
          IRLENF<='0' after ODEL;
          RDF <='0' after ODEL;
          INCF<='1' after ODEL,
                    '0' after ODEL+RDDEL ;
          RENF<='0' after ODEL+RDDEL;
```

```
        CSENF<='0';
end process FETCH_P;
----------------
EXECUTE_P: process
  begin
    wait on EXECUTE until EXECUTE;
      case IRF is
      when "0001" =>                                    ---LDA
            PCEE<='1' after MADEL;
            RENE<='1'after ODEL;
            RDE<='1' after ODEL;
            WRITEE<='0' after ODEL;
            TLENE<='1' after ODEL;
            wait for RDDEL;
            PCEE<='0' after ODEL;
            TLENE<='0' after ODEL;
            RDE<='0' after ODEL;
            RENE<='0' after ODEL+RDDEL;
            wait for PER;
            wait on CLK;
            TENE<='1' after MADEL;
            RDE<='1' after ODEL;
            RENE<='1' after ODEL;
            wait for RDDEL;
            ALENE<='1'  after ENDEL,
                        '0' after DISDEL;
            TENE<='0' after MADEL;
            RDE<='0' after ODEL;
            RENE<='0' after ODEL;
            INCE<='1' after ODEL,
                 '0' after ODEL+RDDEL;
            RENE<='0' after ODEL+RDDEL;
      when "0010" =>                                    ---ADD
            PCEE<='1' after MADEL;
            RENE<='1' after ODEL;
            RDE<='1' after ODEL;
            WRITEE<='0' after ODEL;
            TLENE<='1' after ODEL;
            wait for RDDEL;
            PCEE<='0' after ODEL;
            TLENE<='0' after ODEL;
            RDE<='0' after ODEL;
            RENE<='0' after ODEL+RDDEL;
```

```
        wait for PER;
        wait on CLK;
        TENE<='1'after MADEL ;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                    '0' after DISDEL;
        RDE<='0' after ODEL;
        TENE<='0' after ODEL;
        INCE<='1' after ODEL,
                   '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="0011" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                        '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                    '0' after 2*ODEL+RDDEL;
when "0011" =>                                    ---SUB
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                    '0' after DISDEL;
        TENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                  '0' after ODEL+RDDEL;
```

```
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="1010" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                    '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                   '0' after 2*ODEL+RDDEL;
when "0100" =>                                  ---STA
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        TLENE<='0' after ODEL;
        PCEE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        TENE<='1' ;
        wait on CLK;
        AENE<='1' after MADEL;
        TENE<='1' after ODEL;
        RENE<='1' after ODEL;
        WRITEE<='1' after ODEL;
        wait for WDEL;
        AENE<='0' after ODEL;
        TENE<='0' after ODEL;
        WRITEE<='0' after ODEL;
        INCE<='1' after ODEL,
                '0' after ODEL+RDDEL ;
        RENE<='0' after ODEL+RDDEL;
when "0101" =>                                  ---AND
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
```

```
                '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="0101" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                    '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                   '0' after 2*ODEL+RDDEL;
when "0111" =>                                  ---XOR
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                   '0' after DISDEL;
        TENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                  '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="0100" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                    '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                   '0' after 2*ODEL+RDDEL;
when "1000" =>                                  ---LDI
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
```

```
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        wait for RDDEL;
        ALENE<='1' after ENDEL,
                  '0' after DISDEL;
        PCEE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                 '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
when "1001" =>                                  ---JUMP
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        RDE<='0' after ODEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1'after MADEL,
                  '0'after ODEL+RDDEL;
        PCLE<='1' after ODEL,
                 '0' after ODEL+RDDEL;
when "1010" =>                                  ---JAN
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                 '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        if (A_ALU(3)='1') then
```

```
                TENE<='1' after MADEL,
                        '0' after ODEL+RDDEL;
                PCLE<='1' after ODEL,
                        '0' after ODEL+RDDEL;
        end if;
when "1011" =>                                  ---INCA
        DATA<="0000" after MADEL,
             "ZZZZ" after ODEL+RDDEL;
        BLENE<='1' after ODEL,
                '0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        wait for ODEL+RDDEL;
        CB<="1011" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                '0' after 2*ODEL+RDDEL;
when "1100" =>                                  ---SHAL
        wait for ODEL;
        AENE<='1' after MADEL,
                '0' after ODEL+RDDEL;
        BLENE<='1' after ODEL,
                '0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        wait for ODEL;
        CB<="0011" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                '0' after 2*ODEL+RDDEL;
when "1101" =>                                  ---CLA
        wait for ODEL;
        DATA<="0000" after MADEL,
                "ZZZZ" after ODEL+RDDEL;
        ALENE<='1' after ODEL,
                '0' after ODEL+RDDEL;
when "1110" =>                                  ---CMA
        DATA<="1111" after MADEL,
                "ZZZZ" after ODEL+RDDEL;
        BLENE<='1' after ODEL,
                '0' after ODEL+RDDEL;
```

```
                wait for PER;
                wait on CLK;
                wait for ODEL;
                CB<="0100" after ODEL;
                ALUENE<='1' after ODEL+MADEL,
                            '0' after 2*ODEL+RDDEL;
                ALENE<='1' after ODEL+ODEL,
                           '0' after 2*ODEL+RDDEL;
        when others =>                                  ---HALT
                STOPE<='1' after ODEL;
        end case;
        CSENE<='1';
  end process EXECUTE_P;
  ---------------
ALENB<=ALENE when not ALENE'quiet else          ---ACC
        ALENB;
ALEN<=ALENB;
AENB<=AENE when not AENE'quiet else
        AENB;
AEN<=AENB;
BLENB<=BLENE when not BLENE'quiet else          ---B_REG
        BLENB;
BLEN<=BLENB;
ALUENB<=ALUENE when not ALUENE'quiet else       ---ALU
          ALUENB;
ALUEN<=ALUENB;
IRLENB<=IRLENF when not IRLENF'quiet else       ---IR
         IRLENB;
IRLEN<=IRLENB;
TLENB<=TLENE when not TLENE'quiet else          ---TAR
        TLENB;
TLEN<=TLENB;
TENB<= TENE when not TENE'quiet else
        TENB;
TEN<=TENB;
INCB<=INCF when not INCF'quiet else             ---PC
      INCE when not INCE'quiet else
      INCB;
INC<=INCB;
PCEB<=PCEE when not PCEE'quiet else
       PCEF when not PCEF'quiet else
       PCEB;
PCE<=PCEB;
```

```
    PCLB<=PCLE when not PCLE'quiet else
          PCLB;
    PCL<=PCLB;
    RENB<=RENF when not RENF'quiet else          ---RAM
          RENE when not RENE'quiet else
          RENB;
    REN<=RENB;
    RDB<=RDF when not RDF'quiet else
         RDE when not RDE'quiet else
         RDB;
    RD<=RDB;
    WRITEB<=WRITEF when not WRITEF'quiet else
            WRITEE when not WRITEE'quiet else
            WRITEB;
    WRITE<=WRITEB;
    CSEN<=CSENF when not CSENF'quiet else            ---CHANGE
          CSENE when not CSENE'quiet else      ---STATE ENABLE
          CSENS when not CSENS'quiet else
          CSEN;
    STOP<=STOPR when not STOPR'quiet else            ---STOP CLOCK
          STOPE when not STOPE'quiet else
          STOP;
end arcc;
```

APPENDIX C. MODIFIED SOURCE PROGRAM

```
    ---------- TEST_BENCH COMPONENT----------
entity tb is
end tb;
-------------
use work.pack3.all;
architecture arctb of tb is
    component control
      generic(MADEL, WDEL, ODEL, RDDEL, ENDEL, DISDEL, PER :
              TIME);
      port(  DATA  : inout r_word;
             RUN   : in r_bit;
             A_ALU : in r_word;
             IRLEN  : out r_bit;
             TLEN, TEN  :  out r_bit;
             INC, PCL, PCE : out r_bit;
             ALEN, AEN, BLEN, ALUEN : out r_bit;
             CB : out r_word;
             REN, RD, WRITE : out r_bit);
    end component;
    -------------
    component ram
      generic(RDDEL, DISDEL : TIME);
      port(  RAM_I : in r_type;
             DATA  : inout r_word;
             MA    : in    r_word;
             RD, WRITE : in r_bit;
             REN   : in r_bit;
             RAM_O : out r_type);
    end component;
    -------------
    component ir
      port(  DATA_I      : in r_word;
             IRLEN       : in r_bit;
             IRO         : out r_word);
    end component;
    ------------
    component tar
      generic(DISDEL : TIME);
```

```
   port(  TARI : in r_word;
          TARO : out r_word;
          TLEN : in r_bit;
          TEN  : in r_bit);
end component;
------------
component pc
   generic(DISDEL : TIME);
   port(  P_C  : inout r_word;
          INC : in r_bit;
          PCL : in r_bit;
          PCE : in r_bit);
end component;
-------------
component acc
   generic(DISDEL : TIME);
   port(  DATA  : inout r_word;
          A_ALU : out r_word;
          ALEN  : in r_bit;
          AEN   : in r_bit);
end component;
-------------
component breg
   port(  DATAI : in r_word;
          B_ALU : out r_word;
          BLEN  : in r_bit);
end component;
component alu
   generic(DISDEL : TIME);
   port(  ALUO  : out r_word;
          A_ALU : in r_word;
          B_ALU : in r_word;
          ALUEN : in r_bit;
          CB    : in r_word);
end component;
  -------------
for all:control use entity work.control(arcc);
for all:ram use entity work.ram(arcram);
for all:ir use entity work.ir(arcir);
for all:tar use entity work.tar(arctar);
for all:pc use entity work.pc(arcpc);
for all:acc use entity work.acc(arcacc);
for all:breg use entity work.breg(arcb);
```

```
    for all:alu use entity work.alu(arcalu);
    ------------
    signal REN, IRLEN, RUN : r_bit;
    signal ACC_O, IRO:r_word;
    signal ADDR_B : r_word;
    signal DATA_B : r_word;
    signal INC, PCL, PCE : r_bit;
    signal RD, WRITE, TLEN, TEN : r_bit;
    signal A_ALU, B_ALU, CB : r_word;
    signal ALEN, AEN, BLEN, ALUEN : r_bit;
    signal RAM_I :r_type :=("0001","1110","0100","1111","1111",
                            "1111","1111","1110","0111","1110",
                            "1110","1100","1101","1111","1100",
                            "0001");
    signal RAM_O : r_type;
begin
    C1:control
      generic map(40ns, 100ns, 50ns, 150ns, 5ns, 40ns, 500ns)
      port map (DATA_B, RUN, A_ALU, IRLEN, TLEN, TEN, INC,
                PCL, PCE, ALEN, AEN, BLEN, ALUEN, CB, REN, RD,
                WRITE);
    r1:ram
      generic map(100ns, 30ns)
      port map (RAM_I, DATA_B, ADDR_B, RD, WRITE, REN, RAM_O);
    i1:ir
      port map (DATA_B, IRLEN, IRO);
    T1:TAR
      generic map(5ns)
      port map (DATA_B, ADDR_B, TLEN, TEN);
    P1:pc
      generic map(5ns)
      port map (ADDR_B, INC, PCL, PCE);
    A1:acc
      generic map(5ns)
      port map (DATA_B, A_ALU, ALEN, AEN);
    B1:breg
      port map (DATA_B, B_ALU, BLEN);
    ALU1:alu
      generic map(5ns)
      port map (DATA_B, A_ALU, B_ALU, ALUEN, CB);
      RUN <= '1' ;
end arctb;
```

```
    ------------------------- IR COMPONENT ------------------------------
use work.pack3.all;
entity ir is
    port(     DATA_I  : in r_word;
              IRLEN   : in r_bit;
              IRO     : out r_word := "ZZZZ");
end ir;
------------
use work.pack3.all;
architecture arcir of ir is
begin
    process(IRLEN)
    begin
      if IRLEN='1' then
            IRO<=DATA_I;
      end if;
    end process;
end arcir;

    ------------------------- B_REG COMPONENT --------------------------
use work.pack3.all;
entity breg is
    port(     DATAI : in r_word;
              B_ALU : out r_word := "ZZZZ";
              BLEN  : in r_bit);
end breg;
-------------
use work.pack3.all;
architecture arcb of breg is
    signal BB : r_word;
begin
    process(BLEN)
    begin
      if BLEN='1' then
            BB<=DATAI;
      end if;
      B_ALU<=BB;
    end process;
end arcb;

    ------------------------- ALU COMPONENT ----------------------------
use work.pack3.all;
entity alu is
```

```
    generic(DISDEL: TIME);
    port(     ALUO  : out r_word := "ZZZZ";
              A_ALU : in r_word;
              B_ALU : in r_word;
              ALUEN : in r_bit;
              CB    : in r_word);
end alu;
-----------
architecture arcalu of alu is
    signal ALUB : r_word;
begin
    process(ALUEN)
      variable addb : r_word;
    begin
      case CB is
      when "0011" =>                          ---ADD,SHAL
            addb:=addv(A_ALU,B_ALU);
      when "1010" =>                          ---SUB
            addb:=subv(A_ALU,B_ALU);
      when "0110" =>                          ---AND
            addb:=andv(A_ALU,B_ALU);
      when "0101" =>                          ---OR
            addb:=orv(A_ALU,B_ALU);
      when "0100" =>                          ---XOR,CMA2
            addb:=xorv(A_ALU,B_ALU);
      when "1011" =>                          ---INCA
            addb:=inc_word(A_ALU);
      when "0111" =>                          ---CMA1
            addb:="1111";
      when others =>                          ---LDA,LDI,JUMP,JAN
                                              ---CLA,HALT
      end case;
      if ALUEN='1' then
            ALUO<=addb;
      else
            ALUO<="ZZZZ" after DISDEL;
      end if;
    end process;

end arcalu;

    ------------------------- ACC COMPONENT ----------------------------
use work.pack3.all;
entity acc is
```

```
    generic(DISDEL : TIME);
    port(     DATA : inout r_word := "ZZZZ";
              A_ALU : out r_word := "ZZZZ";
              ALEN : in r_bit;
              AEN  : in r_bit);
end acc;
------------
use work.pack3.all;
architecture arcacc of acc is
    signal ACCB : r_word;
begin
    process(ALEN, AEN)
    begin
      if ALEN='1' then
              ACCB <= DATA;
              end if;
      if AEN='1' then
              DATA <= ACCB;
      else
              DATA <= "ZZZZ" after DISDEL;
      end if;
      A_ALU <= ACCB;
    end process;
end arcacc;

    ------------------------- RAM COMPONENT ------------------------------
use work.pack3.all;
entity ram is
    generic( RDDEL, DISDEL : TIME);
    port( RAM_I  : in r_type;
              DATA   : inout r_word := "ZZZZ";
              MA     : in r_word;
              RD, WRITE : in r_bit;
              REN    : in r_bit;
              RAM_O  : out r_type);
end ram;
------------
use work.pack3.all;
architecture arcram of ram is
begin
    process(REN)
              variable count:INTEGER:=0;
      variable ramb : r_type;
    begin
```

```
        if count=0 then
                ramb  :=  RAM_I;
                RAM_O<=ramb ;
                count :=  count+1;
        end if;
        if REN='1' then
                if (RD='1') then
                        DATA <= ramb(bitarray_to_int(MA)) after RDDEL;
                else
                        DATA <="ZZZZ" after DISDEL;
                end if;
                        if (WRITE='1') then
                        ramb(bitarray_to_int(MA)) := DATA;
                end if;
        else
        DATA <= "ZZZZ" after DISDEL;
        end if;
        RAM_O <= ramb after RDDEL ;
    end process;
end arcram;

    ------------------------------ PC COMPONENT -----------------------------
use work.pack3.all;
entity pc is
    generic(DISDEL : TIME);
    port(       P_C : inout r_word := "ZZZZ";
                INC: in r_bit;
                PCL: in r_bit;
                PCE: in r_bit);
end pc;
-----------
use work.pack3.all;
architecture arcpc of pc is
    signal PCI : r_word;
begin
    process(PCE, INC, PCL, P_C)
    begin
      if (PCE='1')  then
                P_C<=PCI;
       else
                P_C<="ZZZZ" after DISDEL;
      end if;
      if  (INC='1') and (not INC'stable) then
                PCI<=inc_word(PCI) ;
```

```
        end if;
        if (PCL='1')  then
                PCI<=P_C;
        end if;
    end process;
end arcpc;

    ------------------------- TAR COMPONENT ----------------------------
use work.pack3.all;
entity tar is
    generic(DISDEL : TIME);
    port(       TARI : in r_word;
                TARO : out r_word := "ZZZZ";
                TLEN : in r_bit;
                TEN  : in r_bit);
end tar;
----------
use work.pack3.all;
architecture arctar of tar is
    signal TARB : r_word;
begin
    process(TLEN, TEN, TARI)
    begin
        if TLEN='1' then
                TARB <= TARI;
        end if;
        if TEN='1' then
                TARO <= TARB;
        else
                TARO <= "ZZZZ" after DISDEL;
        end if;
    end process;
end arctar;

    ----------------------------      CONTROLLER      ----------------------------
use work.pack3.all;
entity control is
    generic(MADEL, WDEL, ODEL, RDDEL, ENDEL, DISDEL, PER :
            TIME);
    port (      DATA : inout r_word := "ZZZZ";
                RUN  : in r_bit;
                A_ALU: in r_word;
                IRLEN  : out r_bit;
                TLEN, TEN   : out r_bit;
```

```
            INC, PCL, PCE : out r_bit;
            ALEN, AEN, BLEN, ALUEN : out r_bit;
            CB : out r_word;
            REN, RD, WRITE : out r_bit);
end control;
------------
use work.pack3.all;
architecture arcc of control is
    signal IRF : r_word;
    signal CLK, EXECUTE, FETCH : boolean;
    signal CSEN, CSENE, CSENF, CSENS : r_bit;
    signal STOP, STOPE, STOPR : r_bit;
    signal RDB, RDE, RDF, IRLENF, IRLENB : r_bit;
    signal WRITEF, WRITEE, WRITEB : r_bit;
    signal RENF, RENE, RENB : r_bit;
    signal TLENE, TENE, TENB, TLENB : r_bit;
    signal ALENE, ALENB, AENE, AENB : r_bit;
    signal BLENE, BLENB : r_bit;
    signal ALUENE, ALUENB : r_bit;
    signal INCE, INCF, INCB : r_bit;
    signal PCLE, PCLF, PCLB, PCEE, PCEF, PCEB : r_bit;
begin
    ---------
    SYSCLK: process(RUN, CLK, STOP)
      begin
        if (RUN='1') and (not RUN'stable) then
            CLK <= true;
         elsif (RUN='1') and (STOP= '0')then
            CLK <= transport not CLK after PER;
        end if;
      end process SYSCLK;
    ---------
    RUN_P: process(RUN)
      begin
        if RUN='1' then
            STOPR <= '0' ; else
            STOPR <= '1' ;
        end if;
      end process RUN_P;
    ---------
    STATE: process(RUN, CLK, CSEN, STOP)
      begin
        if (not RUN'stable) and (RUN='1') then
```

```
            CSENS<='1';
            FETCH<=true;
            EXECUTE<=false;
        elsif (not CLK'stable) and (CLK) and (STOP='0')
            and (RUN='1') and (CSEN='1') then
            FETCH<=true;
            EXECUTE<=false;
        elsif (not CLK'stable) and (CLK) and (STOP='0')
            and (RUN='1') and (CSEN='0') then
            FETCH<=false;
            EXECUTE<=true;
        end if;
    end process STATE;
----------
FETCH_P: process
    begin
        wait on FETCH until FETCH;
            PCEF<='1' after MADEL;
            RENF<='1' after ODEL;
            RDF <= '1' after ODEL;
            WRITEF <= '0' after ODEL;
            IRLENF <='1' after RDDEL ;
            wait for RDDEL;
            IRF <= DATA;
            PCEF<='0' after ODEL;
            IRLENF<='0' after ODEL;
            RDF <='0' after ODEL;
            INCF<='1' after ODEL,
                    '0' after ODEL+RDDEL ;
            RENF<='0' after ODEL+RDDEL;
            CSENF<='0';
 end process FETCH_P;
----------
EXECUTE_P: process
    begin
        wait on EXECUTE until EXECUTE;
            case IRF is
            when "0001" =>                                  ---LDA
                    PCEE<='1' after MADEL;
                    RENE<='1'after ODEL;
                    RDE<='1' after ODEL;
                    WRITEE<='0' after ODEL;
                    TLENE<='1' after ODEL;
```

```
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1' after MADEL;
        RDE<='1' after ODEL;
        RENE<='1' after ODEL;
        wait for RDDEL;
        ALENE<='1'  after ENDEL,
                    '0' after DISDEL;
        TENE<='0' after MADEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL;
        INCE<='1' after ODEL,
             '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
when "0010" =>                                  ---ADD
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1'after MADEL ;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                   '0' after DISDEL;
        RDE<='0' after ODEL;
        TENE<='0' after ODEL;
        INCE<='1' after ODEL,
                  '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
```

```
        wait for PER;
        wait on CLK;
        CB<="0011" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                       '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                     '0' after 2*ODEL+RDDEL;
when "0011" =>                                          ---SUB
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                     '0' after DISDEL;
        TENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="1010" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                       '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                     '0' after 2*ODEL+RDDEL;
when "0100" =>                                          ---STA
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
```

```
        TLENE<='1' after ODEL;
        wait for RDDEL;
        TLENE<='0' after ODEL;
        PCEE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        TENE<='1' ;
        wait on CLK;
        AENE<='1' after MADEL;
        TENE<='1' after ODEL;
        RENE<='1' after ODEL;
        WRITEE<='1' after ODEL;
        wait for WDEL;
        AENE<='0' after ODEL;
        TENE<='0' after ODEL;
        WRITEE<='0' after ODEL;
        INCE<='1' after ODEL,
                '0' after ODEL+RDDEL ;
        RENE<='0' after ODEL+RDDEL;
when "0101" =>                                  ---AND
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1'after MADEL ;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                '0' after DISDEL;
        TENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                '0' after ODEL+RDDEL;
```

```
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="0110" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                    '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                   '0' after 2*ODEL+RDDEL;
when "0110" =>                                          ---OR
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1'after MADEL ;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                   '0' after DISDEL;
        TENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                  '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="0101" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                    '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                   '0' after 2*ODEL+RDDEL;
when "0111" =>                                          ---XOR
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
```

```
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        wait for RDDEL;
        BLENE<='1' after ENDEL,
                    '0' after DISDEL;
        TENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                  '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        CB<="0100" after ODEL;
        ALUENE<='1' after ODEL+MADEL,
                        '0' after 2*ODEL+RDDEL;
        ALENE<='1' after ODEL+ODEL,
                        '0' after 2*ODEL+RDDEL;
when "1000" =>                                      ---LDI
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        wait for RDDEL;
        ALENE<='1' after ENDEL,
                    '0' after DISDEL;
        PCEE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                  '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
when "1001" =>                                      ---JUMP
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
```

```
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        RDE<='0' after ODEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        TENE<='1'after MADEL,
                '0'after ODEL+RDDEL;
        PCLE<='1' after ODEL,
                '0' after ODEL+RDDEL;
when "1010" =>                                  ---JAN
        PCEE<='1' after MADEL;
        RENE<='1' after ODEL;
        RDE<='1' after ODEL;
        WRITEE<='0' after ODEL;
        TLENE<='1' after ODEL;
        wait for RDDEL;
        PCEE<='0' after ODEL;
        TLENE<='0' after ODEL;
        RDE<='0' after ODEL;
        INCE<='1' after ODEL,
                '0' after ODEL+RDDEL;
        RENE<='0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        if (A_ALU(3)='1') then
                TENE<='1' after MADEL,
                        '0' after ODEL+RDDEL;
                PCLE<='1' after ODEL,
                        '0' after ODEL+RDDEL;
        end if;
when "1011" =>                                  ---INCA
        DATA<="0000" after MADEL,
             "ZZZZ" after ODEL+RDDEL;
        BLENE<='1' after ODEL,
                '0' after ODEL+RDDEL;
        wait for PER;
        wait on CLK;
        wait for ODEL+RDDEL;
```

```
            CB<="1011" after ODEL;
            ALUENE<='1' after ODEL+MADEL,
                        '0' after 2*ODEL+RDDEL;
            ALENE<='1' after ODEL+ODEL,
                        '0' after 2*ODEL+RDDEL;
        when "1100" =>                                  ---SHAL
            wait for ODEL;
            AENE<='1' after MADEL,
                        '0' after ODEL+RDDEL;
            BLENE<='1' after ODEL,
                        '0' after ODEL+RDDEL;
            wait for PER;
            wait on CLK;
            wait for ODEL;
            CB<="0011" after ODEL;
            ALUENE<='1' after ODEL+MADEL,
                        '0' after 2*ODEL+RDDEL;
            ALENE<='1' after ODEL+ODEL,
                        '0' after 2*ODEL+RDDEL;
        when "1101" =>                                  ---CLA
            wait for ODEL;
            DATA<="0000" after MADEL,
                        "ZZZZ" after ODEL+RDDEL;
            ALENE<='1' after ODEL,
                        '0' after ODEL+RDDEL;
        when "1110" =>                                  ---CMA
            DATA<="1111" after MADEL,
                        "ZZZZ" after ODEL+RDDEL;
            BLENE<='1' after ODEL,
                        '0' after ODEL+RDDEL;
            wait for PER;
            wait on CLK;
            wait for ODEL;
            CB<="0100" after ODEL;
            ALUENE<='1' after ODEL+MADEL,
                        '0' after 2*ODEL+RDDEL;
            ALENE<='1' after ODEL+ODEL,
                        '0' after 2*ODEL+RDDEL;
        when others =>                                  ---HALT
            STOPE<='1' after ODEL;
        end case;
        CSENE<='1';
end process EXECUTE_P;
```

```
---------------
ALENB<=ALENE when not ALENE'quiet else          ---ACC
        ALENB;
ALEN<=ALENB;
AENB<=AENE when not AENE'quiet else
       AENB;
AEN<=AENB;
BLENB<=BLENE when not BLENE'quiet else          ---B_REG
        BLENB;
BLEN<=BLENB;
ALUENB<=ALUENE when not ALUENE'quiet else       ---ALU
          ALUENB;
ALUEN<=ALUENB;
IRLENB<=IRLENF when not IRLENF'quiet else       ---IR
         IRLENB;
IRLEN<=IRLENB;
TLENB<=TLENE when not TLENE'quiet else          ---TAR
         TLENB;
TLEN<=TLENB;
TENB<= TENE when not TENE'quiet else
        TENB;
TEN<=TENB;
INCB<=INCF when not INCF'quiet else             ---PC
       INCE when not INCE'quiet else
       INCB;
INC<=INCB;
PCEB<=PCEE when not PCEE'quiet else
        PCEF when not PCEF'quiet else
        PCEB;
PCE<=PCEB;
PCLB<=PCLE when not PCLE'quiet else
         PCLB;
PCL<=PCLB;
RENB<=RENF when not RENF'quiet else             ---RAM
        RENE when not RENE'quiet else
        RENB;
REN<=RENB;
RDB<=RDF when not RDF'quiet else
      RDE when not RDE'quiet else
      RDB;
RD<=RDB;
WRITEB<=WRITEF when not WRITEF'quiet else
           WRITEE when not WRITEE'quiet else
```

```
                WRITEB;
    WRITE<=WRITEB;
    CSEN<=CSENF when not CSENF'quiet else              ---CHANGE
          CSENE when not CSENE'quiet else      ---STATE ENABLE
          CSENS when not CSENS'quiet else
          CSEN;
    STOP<=STOPR when not STOPR'quiet else              ---STOP CLOCK
          STOPE when not STOPE'quiet else
          STOP;
end arcc;
```

LIST OF REFERENCES

1. John R. Ward, *The Anatomy of Computers*, May 1987.

2. Ernest Meyer, *VHDL Opens the Road to Top-down Design*, Computer Design, February 1989.

3. James R. Armstrong, *Chip-Level Modeling with VHDL*, Prentice Hall, 1989.

4. David L. Barton, *A First Course in VHDL*, Design Automation Guide, 1988.

5. Lipsett/Schaefer/Ussery, *VHDL : Hardware Description and Design*, Kluwer Academic Publishers, 1989.

6. P. Sanchez, E. Randon and E. Villar, *Some Experiences in the Use of VHDL in High-level Synthesis*, Dept. of Electronics, University of Cantabria, 1990.

7. *IEEE Standard VHDL Language Reference Manual Std 1076-1987*, Institute of Electronics Engineers, March 1988.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3. Department Chairman, Code EC/Mw Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4. Professor Chin-Hwa Lee, Code EC/Le Naval Postgraduate School Monterey, CA 93943-5000	5
5. Professor Mitchell L. Cotton, Code EC/Cc Naval Postgraduate School Monterey, CA 93943-5000	1
6. Lo, I-Lung 1F 5 ALY 1 LN 29 Chien-Min RD Peitou Taipei Taiwan, 11213 Republic of China	2
7. Library of Chinese Naval Academy P.O. Box 8494 Tso-Ying, Kaohsiung, Taiwan Republic of China	1
8. Library of Chung-Cheng Institute of Technology Tashih, Tao-Yuan, Taiwan Republic of China	1