DTIC FILE COPY

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, gathering and maintaining the data needed, and completing and reviewing the collection of informati collection of information, including suggestions for reducing this burden, to Washington Headquarter Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, f

AD-A224 476

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1990	Thesis/~~Dissertation~~

4. TITLE AND SUBTITLE

A VHDL INTERFACE FOR ALTERA DESIGN FILES

5. FUNDING NUMBERS

6. AUTHOR(S)

JEROME PAUL NUTTER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

AFIT Student at: Wright State Univ

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/CI/CIA - 90-046

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFIT/CI
Wright-Ptatterson AFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release IAW AFR 190-1
Distribution Unlimited
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer, Civilian Institution Programs

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

DTIC
ELECTE
AUG 0 1 1990
S B D

14. SUBJECT TERMS

15. NUMBER OF PAGES

137

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED			

A VHDL INTERFACE FOR ALTERA DESIGN FILES

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

JEROME PAUL NUTTER
B.S., Troy State University, 1984

1990
Wright State University

90 071

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

July 6, 1990

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Jerome P. Nutter ENTITLED A VHDL Interface for Altera Design Files BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Thesis Director

Department Chair

Committee on
Final Examination

Dean of the School of Graduate Studies

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

ABSTRACT

Nutter, Jerome Paul. M.S., Department of Computer Science and Engineering, Wright State University, 1990. A VHDL Interface for Altera Design Files.

Altera Erasable Programmable Logic Devices (EPLDs) are chips that can be custom designed. These EPLDs are individually described by their Altera Design Files (ADFs). The language structure of ADFs is not directly supported by the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). VHDL is a software language that was selected as the IEEE standard for a hardware description language.

This thesis describes a program that is capable of transforming an ADF into a VHDL entity declaration and entity structural architecture. The scope of ADFs the program transforms is limited to ADFs that contained only Altera primitives and are not of the State Machine Format.

The transformation program was developed on a Personal Computer (PC) and the programing language used was Turbo C by Borland.

Further development of this program in an expanded way would be a useful direction for future research.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

LIST OF FIGURES

I. INTRODUCTION

This thesis describes a computer program that translates descriptions of a class of programmable logic devices from a proprietary format (Altera EPLDs) to an industry standard format (VHDL). The reader is expected to be familiar with VHDL, the Altera Programmable Logic User System (A+PLUS), and EPLDs. References in the bibliography (3, 5, 6, 7, 8) can be used for refreshment of a specific topic.

Overview

Figure 1 shows the overall structure of the Altera and VHDL interface. The figure is provided as a guide for the transformation process.

Through the A+PLUS system, an ADF is created. This ADF is what is used to program an EPLD. The ADF represents a digital device description.

The transformation program is the subject of this thesis. The program transforms an ADF to a VHDL entity description file. The entity description file contains an entity declaration and entity architecture. The entity file name is selected by the user and must have a ".vhd" extension.

The new file can then be processed by a VHDL analyzer. The analyzer accesses predefined Altera primitive component

descriptions during the analysis. If the analysis is successful, the entity declaration and entity architecture are stored in a VHDL library. The new entity can now be used in a larger entity description. The operation of the new entity can also be simulated using a VHDL simulator.

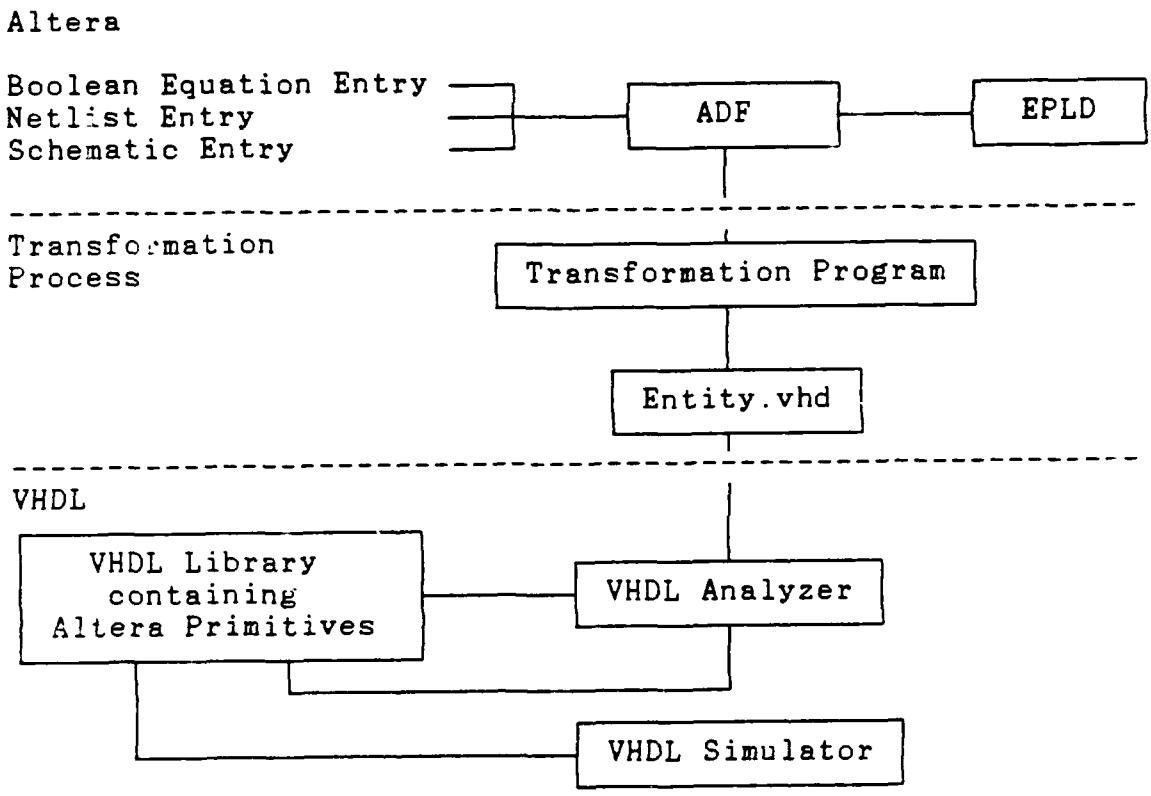

Figure 1. Altera to VHDL Interface Overview

Background

The Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) is a full bodied software language that supports hardware device design and simulation. VHDL was developed at the request of the Department of Defense, and is approved and accepted by the

IEEE as the standard for a hardware description language. VHDL provides the capability to declare, describe, and simulate hardware devices.

VHDL hardware devices are referred to as entities. An entity is made up of a declaration and an architecture. The declaration contains the name of the entity and associated ports. Ports are the input and output paths of the entity. The entity architecture is a description of how the device operates.

There are two types of architectures. The first type of entity architecture is behavioral. Behavioral architectures describe an entity in terms of signal declarations and signal assignment statements. Signals are the paths or connections between ports. Signal assignment statements are VHDL processes that contain timing information.

The second type of entity architecture is structural. Structural architectures describe an entity in terms of signal declarations and signal assignments but, also include component declarations and component instantiations. Component declarations contain the name of an entity to be used and also the ports associated with that component. Component instantiations are individual representations of a component and any signal to port associations that are necessary. Typically, a structural architecture is composed

of component instantiations with signal assignments that connect the instantiated components.

Developing an entity involves producing VHDL code for an entity declaration and entity architecture that can be analyzed by the VHDL analyzer. The VHDL analyzer checks VHDL code for syntactic and semantic correctness. Valid VHDL code is then stored in a VHDL library.

Once a declaration and architecture for an entity has been developed, the entity can be tested through the use of the VHDL software simulator. Input signals are established prior to simulation and output signals are recorded by the simulator. A report can be generated to print the results of a simulation run. The report contains the values, names, and event times of desired ports or signals described in the entity architecture. If the simulation reveals an unwanted result, the entity architecture can be modified and the simulation can be rerun.

The design and simulation are technology independent. If one desires, an entity can be tested using different architectures. Each architecture would represent a different behavior (technology).

VHDL is currently installed on the SUN computers at the Research Park.

Altera design files contain the information necessary to program EPLDs when using the Altera Programmable Logic User System (A+PLUS). ADFs represent hardware device descriptions. An ADF has seven major sections. Three of these sections OPTIONS, PART, and header are not germane to this thesis. The other five sections are discussed below and throughout this paper.

The header section is the first part of an ADF and contains textual information about the device being described. This information includes but is not limited to the name of the designer, date, revision number, EPLD used, and comments.

The second section of concern is the INPUTS section. This section lists the input pin names and possibly the actual pin numbers associated with the EPLD.

The third section is the OUTPUTS section. The OUTPUTS section lists the output pin names and optionally contains actual EPLD pin numbers.

The fourth section is the NETWORK section. This section lists the Altera primitives used in a particular design and the associated inputs and outputs for each primitive. An Altera primitive is one of many digital devices that are predefined by Altera and are used as building blocks for more complex circuit designs.

The last section of concern is the EQUATIONS section. The EQUATIONS section contains boolean descriptions of device nodes or possibly device outputs. A device node is a labeled connection in the ADF.

Problem

VHDL design libraries are composed of many basic and complex hardware components. Each component was created or described using a VHDL system. Outside of the VHDL environment, there exist many custom designed hardware devices which were developed using proprietary software packages. Unfortunately, the format used for the customized components is generally not compatible with VHDL. This makes it difficult (if not impossible) for a designer to incorporate components designed using proprietary software packages into VHDL simulations.

The Altera Erasable Programmable Logic Device family is a semi-custom chip design set. Altera chip designs are used in Wright State University's Computer Engineering Department's microprocessor laboratories. Although the structure of the Altera design files (ADF) is similar to a VHDL entity architecture, ADFs are not compatible with VHDL. A+PLUS is currently not capable of producing VHDL descriptions for Altera device designs. Therefore, there exists the problem of including existing Altera hardware devices in a VHDL design and simulation. A solution is to

translate the ADFs into VHDL entity declarations and entity architectures.

Scope

The translation process described in this thesis is limited to ADFs that contain only Altera primitives and ADFs that were not created using State Machine Entry. State Machine Entry is a method for creating ADFs that describe the operation of state machine designs using Boolean expressions, truth tables, state diagrams, and Algorithmic State Machine charts.

Each ADF is transformed into a VHDL entity declaration and associated entity architecture of the structural type. The translation software runs under MS-DOS on an IBM Personal Computer (PC) or compatible.

Assumptions

In order to define and restrict the scope of the translation program, several assumptions were made before starting it's development:

1. It was assumed that all ADFs to be converted by the transformation process would be valid ADFs;

2. The translation process developed would not be responsible for producing Altera primitive VHDL entity declarations or entity architectures. These

would be present in the current working VHDL library and would be created as a separate project;

3. The translation process would establish the format for the Altera primitive components. The component format would include the component name, port names, port mode and port order;

4. The Altera input file and the translator produced output would both be in ASCII form.

Approach

Program prototyping would be used to develop the program under design. First a parser would be developed that would break down the ADF into individual tokens. The tokens would be stored in a structure that could be searched by an index. A set of procedures would be developed that scanned the tokens to extract the data necessary to then produce an entity declaration. Another set of procedures would be developed that would build a structural architecture of the entity by once again using the tokens. All procedures and functions would be verified individually after being written.

The entity declaration and entity architecture produced would then be transferred to a VHDL system to be tested on a VHDL analyzer. Modifications would be made as necessary

to the design program in order to produce an entity description that would be valid VHDL code.

Sequence of Presentation

This thesis is organized into five chapters. The five chapters are Introduction, Detailed Analysis, Design, Program Testing, and Conclusions and Recommendations.

After this Introduction section, the Detailed Analysis chapter explores in detail the thesis problem and defines the research intended. Next, the Design chapter describes the design process for the development of the transformation program. The Program Testing chapter discusses the testing process and the associated test files. Finally, the Conclusion and Recommendations chapter gives a summary of the success of this thesis and recommendations for further study.

II.DETAILED ANALYSIS

This section is concerned with a thorough analysis and explanation of the research problem. Many factors had an impact on the definition of the research problem. External, internal, and self imposed restrictions limited the scope of the problem.

The external factors discussed in this section are Projected Use, Design Conditions, Conditions Under Use, and Reasons For Program Development.

The internal factors affecting the problem are Current Capabilities and Limitations, and Performance.

Finally, the self imposed restrictions are covered in the Exact Design Requirements, Imposed Constraints, and Established Design Criteria areas.

Projected Use

The program that was developed during the research of this thesis is intended to be useful to a wide range of people. Users would include students, university staff, and outside organizations.

Students will be able to use the transformation program on the ADF files they develop during design studies. The ADF circuits can be tested prior to implementation in a large scale design. In addition to learning how to design

with the A+PLUS system, the student will be exposed to the VHDL description and simulation environment. Testing can be performed by the students and if flaws are detected, the design can be modified and retested. All of these steps could be performed prior to actual programming of an EPLD. Modifying a design on an EPLD involves erasing the EPLD and reprogramming the EPLD. That previously mentioned process could take as long as a half hour per reprogramming. Since the design and testing of a circuit can involve many redesign stages, a great amount of time could be saved by the student when using the transformation program and ultimately the VHDL environment.

University staff would be able to use the transformation program for all of the same reasons as students as well as for the following additions. Previous ADF designs that had been developed could be transformed and simulated in VHDL. Staff could evaluate a previous design more completely using the VHDL simulator. This would ensure better quality control for designs to be given to students for their use. The designs once stored on a VHDL system could then be incorporated into a much larger design for a more complete simulation. The tedious task of hardware testing for each design could be simplified and VHDL software simulations could be substituted. The time saved for staff would be especially valuable due to the limited amount of time they have for course development.

The use of the transforming program by outside organizations would only be limited by the number of institutions that requested the program and any restrictions placed on the use of the program by Wright State University. The transformation program would be useful to any organization that has ADFs that they would like to incorporate into a VHDL environment. These institutions might include individuals, private firms, other universities, and research groups that use the A+PLUS system.

Current Capabilities and Limitations

Currently the design and simulation of electronic circuits described in ADFs is limited to the capabilities of the A+PLUS system. The use of a design in the VHDL arena is limited to designs that were completely done using VHDL code. There is no way known to the author of translating ADFs to VHDL legal code other than the transformation program developed during the research of this thesis.

The A+PLUS system is a stand alone design apparatus that utilizes the capabilities of a PC XT and an A+PLUS option board to create electronic circuit designs and encode these designs into EPLDs. The A+PLUS system also has a limited simulator for checking ADFs. Using the A+PLUS system a person can create an electronic design using schematic design entry, boolean entry, state machine entry, and

netlist entry. Schematic design entry, netlist entry and boolean entry are the only three methods that produce ADFs that can be processed by the transformation program. There are multiple schematic capture schemes offered with the A+PLUS system. The final product of all of the schematic capture methods is a valid ADF. Boolean entry is exactly what it states, entry using boolean equations. Netlist entry is the process of manually entering the description of a design using standard forms. The standard forms are limited to Altera design primitives described in the A+PLUS system and boolean logic operators. Altera design primitives are predefined circuit descriptions which are stored in a library. The use of design primitives should be considered a limitation from the point of view of versatility. This is because the design would be limited to incorporating only Altera primitives. After an ADF is developed, the user can program an EPLD using the Altera option board. Testing of the design would then occur using manual input on a real-time powered system such as a prototyping board. The process is involved and long. Any mistakes would require the erasure and reprogramming of the EPLD.

Simulation of the operation of an ADF can be accomplished using the A+PLUS system. Simulation would best be described as individual circuit input manipulation with resolved output states being recorded. Each simulation is

of one ADF only. No accounting for implementation of other components in the simulation is possible.

VHDL capabilities are vast. A full and robust software language allows a user to define and test a circuit design. Designs are created with a text editor and then checked and stored using a VHDL analyzer. Testing is accomplished using the VHDL simulator.

VHDL is a software language that is restricted only to the environment the designer describes. There are no primitives that restrict the user in his designing process. A VHDL simulation allows for completely different implementations of hardware technologies, variations of signal delay, concurrent processing of signals, model generation (simulation scripting), simulation event time variations, and report generation for each simulation timing frame. Testing to an expanded degree can be done using the VHDL simulator. All of the previous VHDL capabilities are not found on the A+PLUS system.

Currently, if a design which is described using the A+PLUS system is needed or required in a VHDL simulation, the design must be completely be redone using VHDL code. This would require the designer to recode the circuit from the ground up. Recoding is a possible source of new errors and is a time-consuming operation.

The use of the transformation program eliminates the need for recoding completed ADFs and expands the current capabilities to include automatic conversion of ADFs to VHDL code.

Performance

Even though VHDL and A+PLUS are similar, the focus of each system is different and the implementation methods are not compatible.

A+PLUS is a hardware oriented design system that is implemented on a PC XT. A+PLUS's main purpose for existence is to produce a hardware description from predefined primitives. The whole design concept is hardware oriented. The final product of A+PLUS is a file that is used to program an EPLD. Because the concept of A+PLUS is restricted to in-house building blocks, very little emphasis is placed on simulation. Simulation is geared towards finding out if a design that utilizes primitives will produce the desired output for a set of given inputs. The focus is not on the primitives themselves. The primitives cannot be modified and therefore represent the basis of all designs. In other words, the design process using A+PLUS is a constrained procedure.

VHDL on the other hand is an open, versatile, and complete hardware description language. The focus of VHDL is modeling of systems with as few restrictions as possible.

This makes the designs described in VHDL very useful. A designer is only limited to the environment or constructs made within VHDL. VHDL designs themselves become the primitives and are as malleable as any design construct created with VHDL. Therefore, the performance of VHDL is mainly limited to the ability of the designer. VHDL is currently implemented on mini and mainframe computer systems.

Simulation with VHDL is a system within a system. The environment around the entity is under the complete control of the user. Port levels, timing constraints, and input values are some of the things a user can manipulate. The design signals are processed in a concurrent manner. A report on each signal value during a timing frame is available. The input scripting provides a way to test a design through all possible transformations. Entity architectures can be substituted between simulations. This is a way of simulating the differences in design behaviors dictated by differing technologies. Simulations are on entities which can be composed of any number of components. An entity is not restricted to the size of any real or existing hardware device. All of these factors lead to a very versatile simulation environment.

Design Conditions

Numerous factors affected the way the transforming program was designed. The equipment used, programming languages, and existing code all had an impact on the design.

A PC XT was selected as the computer system for the transformation process because A+PLUS is installed on an IBM PC XT type computer at Wright State University. The requirements for the computer were that it had to use DOS 2.0 or higher and have at least 448K of RAM. These requirements were dictated by the programming language.[1]

The programming language selected was Borland Turbo C version 2.0. Turbo C supports the Draft-Proposed American National Standards Institute (ANSI) C standard, fully supports the Kernighan and Ritchie definition, and includes certain optional extensions for mixed-language and mixed-model programming.[1]

The Turbo C package has standard include files. Some of these include files were utilized in the transformation program. Other than the include files, all of the transforming program is original code developed during the research of this thesis. Ultimately, another support package had to be coded in VHDL. All of this code was original and developed during this thesis.

The VHDL environment utilized to test and develop code was the Intermetrics Standard VHDL 1076 Support Environment.

This support system contains the analyzer and simulation software used to validate the VHDL code generated by the transformation program. The code produced by the transformation program should be valid on any standard VHDL environment. No code specific to the Intermetrics VHDL toolset is produced by the transformation program.

Exact Design Requirements

Specific goals were established for this research. The intent was to develop a program that was able to accept an ADF and produce a valid VHDL description of the device described by the ADF. The complete process had to take place on a PC. The exact requirements were to produce a VHDL entity describing the ADF, produce a structural architecture describing the behavior of the ADF, and both entity and architecture had to be valid VHDL code.

The first step in a transformation process would be to produce an entity declaration for the ADF device. An entity declaration is a VHDL requirement and represents an external view of the device being described. The input and output pins of the ADF device would have to be converted to ports in an entity declaration. An entity name would have to be determined and assigned.

The entity created would also have to have a VHDL architecture. The architecture would describe the behavior of the device. A structural architecture format was chosen

because it would allow the use of components already stored in the VHDL library and reduce the amount of repetition involved in the transformation process. In other words, by assuming the Altera primitives to already be stored as components on the VHDL environment, repetitive code describing the primitives behaviors could be eliminated from the produced ADF device description. External development of the behaviors of the Altera primitives is also the most appropriate way of dealing with the primitives. The behavior of each Altera primitive is dependent on the EPLD it is implemented on. There are variations in speed depending on the type and recency of the EPLD. By making the behaviors of the primitives independent of the transformation program, the maintainability and versatility of the transformation process is facilitated. When a new or faster EPLD becomes available, a user would just have to create a new component architecture and never have to modify the transformation program.

The most important requirement for the transformation process was that it would have to produce valid VHDL code. This means that the entity declaration and entity architecture would both have to be capable of being successfully analyzed on a VHDL system. Basically, this requirement dictated that all prototype products developed during this research would have to mirror standard VHDL code. In fact, the final product would have to meet the

syntactic and semantic requirements of standard VHDL. The coding style used to create the transformation program was the only variable.

Conditions Under Use

This section describes the environment or setup to be employed in the use of the transformation program. The important aspects of the transformation environment would be the location of the ADF files, location of the transformation program, and the location of the VHDL system.

The ADF files would have to be accessible to the transforming program. This would require the ADFs to be on floppy disk or on the hard disk of the PC.

The transformation program would be located on a PC because the programming language is for a PC. The transformation program would access the appropriate ADF from either a floppy disk or hard drive.

The VHDL system would be on a mini to mainframe computer system. This would mean that the transformation file would have to be transferred from the PC to the VHDL computer system. This could be accomplished through physical media transportation or modem transfer. The transformed file could then be analyzed into a VHDL library.

Imposed Constraints

The constraints of the research were applied in an effort to limit the tasks associated with this thesis to a level that would promote success and the development of an end produce that would be useful. The areas where restrictions were imposed were equipment used, needed supplemental code, and what a valid ADF would be.

The transformation program was developed for use on a PC because, the Altera software is on a PC in the Wright State University engineering laboratory and for convenient access for computer science and engineering students. However, because the program is written in C, porting it to other environments should not be difficult.

The supplemental code consists of external component descriptions and a VHDL logic package that would support the transformed files. This research was an attempt to develop a program that would transform ADFs into VHDL entities with structural architectures. To that degree it was decided that the development of the entity declarations and entity architectures for the Altera primitives would be a given assumption and not part of this thesis. The behavior of the EPLDs, which is what dictates the behavior of the Altera primitives, is not germane to this thesis. A valid translation could be done with the understanding that the Altera primitive components would eventually be coded, analyzed, and stored in the VHDL library to be used. The transformed entity would simply reference or instantiate a

primitive component by name. The eventual name and port structure of the primitive components would have to match the structure used by the transformation program. The structure or port format used in the design of the transformation program will be discussed in the design chapter of this paper.

An external VHDL logic support package was also needed. This package was developed during the research of this thesis and was a necessity for the successful analysis of the transformed files by a VHDL analyzer. This subject will also be explained in detail in following chapters.

The scope of ADFs that the transformation program would be able to convert was limited for this research. Only ADFs developed through schematic entry, boolean entry, and netlist entry would be valid. State machine entry was not considered in the development of the transformation program. The ADFs would also have to have the INPUTS, OUTPUTS, NETWORK, and EQUATIONS sections to be valid. The transformation program would key off these section headings. A valid ADF was to only use Altera primitives and not have macros in it.

Established Design Criteria

The chosen method for program design was prototyping. This method was selected because of the many unknowns involved with coding the transformation program.

Prototyping consisted of trying program code to solve a small core transformation problem and then building on the prototype code to develop the larger program. Much experimentation was done on the development of data constructs and conversion algorithms. Prototyping facilitated this experimentation and was consistent with the idea of researching the transformation process.

A complete solution to the transformation of ADFs to valid VHDL entities was not the goal of this thesis. This is obvious from the constraints imposed on the design. The intent was to develop a program that could transform the majority of ADF types and stand as a good building block for future modification. It was also taken for granted that many new areas of improvement would be discovered during the development of the transformation program. The inadequacies and possible improvements of the transformation program will be covered in the Conclusions and Recommendations chapter.

Reasons for Program Development

VHDL is now the IEEE standard for hardware description languages. There are advantages to being compatible with this standard. Compatibility and versatility were the main reasons this thesis was undertaken.

With VHDL being a standard and with it's incorporation into the Wright State curriculum and all of VHDL's advantages to the hardware design environment, the need to

make existing hardware descriptions compatible with VHDL became a necessity. There are existing ADFs which are part of courses taught at Wright State and continuing research is being done with them. A more complete design can be realized with VHDL and therefore the need to convert these ADFs to VHDL valid entities exists. As expressed before, a great deal of development time can be saved with the use of the VHDL environment. A software development and simulation of a hardware device is advantageous over a burn and test hard-wire method. Since no known way exists to transform ADFs into VHDL code, the need for a transformation process is obvious. The goal of this thesis is to fill this need.

III.DESIGN

The design of the transformation program is discussed in this chapter. The structure and logic behind the development of the transformation program are explained in detail. The information covered is Main Program Structure, Parsing Code and Data Structure of Tokens, Name Modifying Code, Entity Declaration Generating Code, Entity Architecture Generating Code, and Driver Code.

Main Program Structure

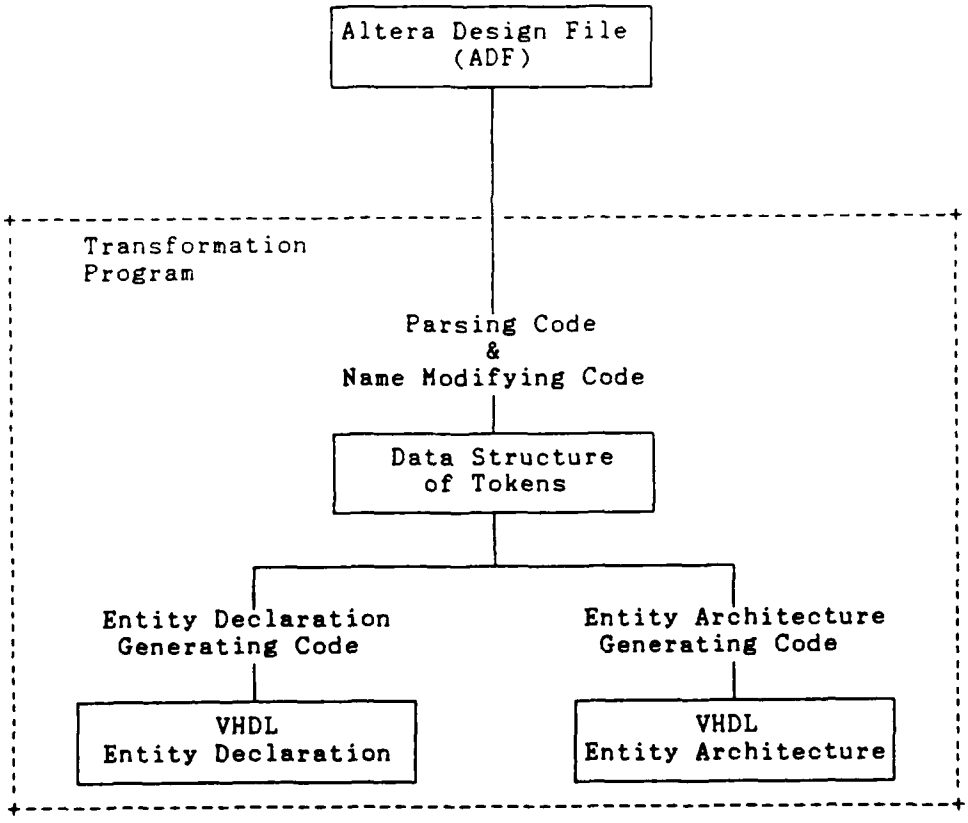

Fig 2. Transformation Process

The transformation process is shown in figure 2. There are four major code sections in the transformation process.

The four sections are Parsing, Name modifying, entity declaration generation, and entity architecture generation. The boxes in figure 2 show the three changes the ADF goes through. The three changes are represented by the Data Structure of Tokens, VHDL entity declaration, and VHDL entity architecture. All of the parts of figure 2 are described below in order of appearance.

Parsing Code and Data Structure of Tokens

The parsing of the ADF into tokens was the first part of the transforming program to be tackled. A parser was need to accept the input ADF and put the information gathered from the file into a structure that could be manipulated by the rest of the transformation program. The data structure selected to hold the tokens was an array and is discussed in detail after the parser.

The main idea behind the parser was that it should take characters from standard input and assemble them into tokens. The tokens would be distinguished by the delimiters. Since no appropriate parsers could be found that processed ADFs , an original design for ADFs was done. The development of the parser was also an exercise in design experience. The ADF is attached to standard input and processed one character at a time. While the current character is not a delimiter, the character is added onto any preceding characters to build a token. When a delimiter

is found the token being built is processed to determine type and then stored in the data structure of tokens. The type of the token is also stored in the data structure of tokens. The delimiters and definitions of type needed for the parser were found in the Altera users guide.[3] When the parsing process is complete, the data structure of tokens remains resident as a variable to be accessed.

The data structure of tokens is composed of four parts. There is an array holding the tokens and another array holding the types of the tokens. There is an index value which points to the current token and there is a value representing the total number of tokens stored. Since the tokens and types of the tokens are stored in their arrays in parallel, the index points to the current token and current type. Token types can be names, functions, and delimiters. The delimiter type also includes the specific delimiter in question. The index is an important value because it is a static variable and therefore maintains the current token position throughout the transformation process.

Name Modifying Code

Name modifying code was necessary because of the differences between valid names in A+PLUS and VHDL. There are certain Altera naming conventions that are not permissible in VHDL. The modifying code makes legal names out of all the names in the ADF.

Altera allows many more characters as valid in Altera names than does VHDL. Specifically, the input and output pins for an Altera EPLD can contain many types of characters other than the VHDL legal A-Z, a-z, and 0-9 characters. The previous restrictions do hold for ADF node names. Nodes are connection points within the device design. Since the Altera names could cause an error in the analysis of the transformation VHDL files, the inappropriate names had to be changed. The method chosen to fix the names was character substitution.

Prior to a token being inserted into the data structure of tokens, if the token is a name type then it is checked by the name modifying code. If any invalid characters are found in the name a substitute character is inserted in place of the invalid character. The current substitution character is a lower case "v" and could be changed if one modified the C source code and recompiled. If a name does not start with an alpha character, the prefix "alpha_mod" is added to the name. If a name contains a pin reference designated by the "@" symbol, the pin citation is removed.

Entity Declaration Generating Code

The entity declaration generating code produces the first of the two products of the transformation program. The first product is the entity declaration for the ADF device being transformed and the second product is the

architecture for the entity (see Appendix B). The entity declaration is required if the VHDL transformation is to analyze successfully.

The approach to designing the entity declaration generating code was to look at the requirements for a valid entity declaration and write code that would produce an entity declaration that fulfilled those requirements. The parts of a valid entity declaration that the transformation program generates are the entity declaration identifier, port interface list, and closing identifier.

The declaration identifier is created from the ADF input file name. Any prefix path and any extension of the ADF file name is stripped and the remaining portion of the file name is used as the entity identifier. This method was chosen for the sake of simplicity. The key word "entity" is written to standard output and then the declaration identifier.

The next item needed is the port interface list. The port interface list contains a list of the port names, port modes and port types. The port names and mode are determined from the tokens. First, the key word "port" is output and the structure of tokens is scanned for the INPUTS section. All of the device input pins represent "in" mode ports for the entity. Therefore, the input pin names can be output to the declaration as ports of that name and mode

"in". The type of the port is written as altera_logic type. This logic type is assigned because it can be described in an external package to the liking of the user. If the type had been declared as "bit", the port values would have been limited to two values. To eliminate this restriction a generic type that can be user defined is assigned to each port. The output or mode "out" ports are found in the OUTPUTS section of the tokens. An ADF tokens that represent comments that are encounter are written to standard output as VHDL comments and processing continues. This is the manner for handling all comment tokens. This method preserves the order and hopefully the usefulness of the comments.

The declaration closure is handled by closing the port interface list with a semi colon and printing the key word "end" followed by the declaration identifier with a semi colon. This is the last step in the entity declaration generating process.

Entity Architecture Generating Code

The entity architecture generating code produces a structural VHDL architecture for the entity already declared. The architecture contains the architecture signal and component declarations, signal assignment statements, component instantiation, and architecture closure.

The architecture body is started by outputting the key word "architecture", the entity identifier prefixed with "structured_", the key word "of", the entity identifier, and the key word "is". This represents the architecture body header. An example might be:

architecture structured_sample of sample is .

The signal and component declarations follow the header. A signal is a connection path within the design other than a port. Components are the predefined Altera primitives. Signals connect components together and are also any intermediate nodes within the device. To find the signals, the network and equation sections of tokens are scanned for any node names other than primitive names. The signal is then declared as an altera_logic type and sent to the standard output. This process continues until the end of the equation section. The signal name and type are separated with a colon and the signal declarations are closed with a semi colon. An example would be:

signal_name : altera_logic; .

Component declarations represent the primi ives used in the ADF design. The network section is scanned for primitive names and any names found are stored with no duplication in an array. The components are then output

with the header "component", primitive name, port interface list, and closing primitive name. The port interface list is retrieved from a function that holds specific information on all the Altera primitives. The port list is enclosed in parenthesis followed by a semi colon and closed with an end statement with the key word "component" and a semi colon. An example would be:

```
component And2
        port (In1 : altera_logic, In2 : altera_logic;
              Out1 : altera_logic);
end component; .
```

The body of the architecture is all that remains to be generated. The body begins with the key word "begin". This word is output and the component instantiation are created. The component instantiation represent all of the primitives used in the design with their associated node connections. The tokens are again scanned for the network section. Each primitive is located and the associated node connections for that primitive are collected from the tokens that follow and proceed the primitive token. The outputs of a primitive precede the primitive token name and are separated from the name by an equal sign. The inputs to a primitive follow the primitive token name and are enclosed with parenthesis. All of this information is collected and sent to standard output. The running label "U?" is printed with the question mark being replace by the current instantiation number. The

primitive name is output followed by the key word "port map". A parenthesis encloses the port names associated with the node names. The port names for the primitive are found from the same function as mentioned before. VHDL allows for a no connection to be labeled as "open". Therefore, if no specific name is associated with a port as determined from the tokens, the default value is used or open if no default value is listed. A no default condition is labeled "ndf" in the information passed from the primitive information function. An example of this might be a sample primitive where input 1 is VCC, input 2 is GND, input 3 is TEST_IN, and input 4 in a no connection. The Altera description would read "OUTPUT = SAMPLE (,,TEST_IN,);". The produced VHDL instantiation would read:

```
U0: SAMPLE
port map (in1 => VCC, in2 => GND, in3 => TEST_IN, in4 =>
          open, out1 => OUTPUT); .
```

The information passed by the function that holds the primitive's information is a list describing the parameters for that primitive. The list includes, for each parameter, the parameter name , the mode, and the default value. The example for SAMPLE would be:

in1 in VCC in2 in GND in3 in ndf in4 in ndf out1 out ndf .

The architecture body is closed with the end statement and architecture identifier with semi colon. Any comments encountered during the architecture processing are handled as stated before in an effort to maintain the designers intended placement of his comments.

Driver Code

The driver code is simply the code that calls the procedures and functions necessary to produce the transformation process. The driver code is arranged in the order that produces the entity declaration first and entity architecture second.

The driver first opens the ADF file or asks the user for valid name if it can't open the ADF with the name provided. The driver will except an ADF name issued at the command line or will prompt for the file name if none is given on the command line.

The driver next builds the data structure of tokens. This data structure is then used by the entity declaration and entity architecture generating code driver calls.

All output is directed to standard output and all input is read from standard input. Prompts are issued to and responses retrieved from the display.

IV.PROGRAM TESTING

Program testing involves validating program requirements and verification of program design. This section is a discussion of validation and verification of the transformation program. The testing methods used on the program and the process of VHDL validation will also be presented.

Testing Methods

Both top down and bottom up testing were employed to check the transformation program during it's development. Bottom up testing was used the majority of the time. Top down testing was used to test the overall progress and validity of the program.

Many lower modules were developed for the transformation program. These low level modules were tested for most of their possible permutations. With the C programming language many errors in low level modules are not detected until additions are made to high level code. This is usually caused by memory space not being properly allocated for variables. This became a problem because low level testing would not detect an error, but higher level testing would fail. The debugging process was extremely difficult during the testing of the parsing code.

The parsing code manipulated the data structure of tokens. The tokens and their types were stored in arrays of pointers pointing to character strings. Sometimes, program halts would occur after seemingly small changes were made to the parsing code. This was caused because the new code modification might cause an improperly allocated variable's memory storage area to be over-written and this would eventually halt the program. Once the arrays were sorted out, the rest of the low level modules became fairly manageable during the testing process.

Top down testing was employed to verify the overall development progress. Test ADF files were constantly tested in whole to verify the correct direction of implementation of the transformation process. The input files were ADFs that are currently in use in the Wright State University computer and engineering course curriculum. The test files were modified as necessary to make them exercise the full range of input possibilities. Two of the test files are included in Appendix A.

Unfortunately, the low level testing was unable to reveal the major errors in design requirements. Many syntactic and semantic errors were found only after actual VHDL analysis was performed. Use of the VHDL analyzer was not attempted prior to the completion of the initial version of the transformation program because even slight omissions of code in a VHDL file will stop the analysis process.

Major Modifications

The first major problem with the output of the transformation process was the problem of illegal names in the transformation file. VHDL has strict naming conventions and not all of the illegal Altera naming methods were taken into account in the original design. After the naming methods were solved, the problem of a support package became apparent.

The use of a generic type for all of the ports and signals caused a problem in that the type had to be defined and all operations on that type had to be defined. Instead of generating the support package each time an ADF is transformed, it was decided to develop a support package, store it in the working VHDL library, and make it visible to the analyzer. The support package contains type information and operator overload code to define the environment of the type *altera_logic*. The support package is listed in Appendix C.

The final group of errors was confined to the areas of syntax and semantics. These errors were caused by omissions of required verbiage or misunderstandings on the designer's part as to what was legal VHDL code. The VHDL analyzer once again was the source of code checking. Validation of the VHDL requirements was accomplished by extensive testing of the transformation program product with the VHDL analyzer.

V.CONCLUSIONS AND RECOMMENDATIONS

The research accomplished for this thesis was a successful endeavor. The transformation program was developed and met the goals established at the onset of this thesis research. The major solutions and recommendations will be expounded upon in this section.

Major Solutions

A transformation program was developed that processes ADFs into valid VHDL entity declarations and entity structural architectures. The transformation program will transform ADFs obtained the Boolean entry, Netlist entry, or Schematic entry format. All of the allowable Altera primitives used in an ADF are correctly transformed into VHDL component instantiations.

The produce of the transformation program is a file that contains an entity declaration and entity architecture. Both of these items will analyze into a VHDL library by passing syntactic and semantic checks by the VHDL analyzer.

This research showed that there is a way to transform ADF device descriptions into VHDL entity descriptions. The end product of this thesis research is a transformation program that accomplished the preceding goal.

Recommendations

Some improvements on the transformation program are possible. The recommended improvements are:

1. The program should be menu driven with more set up options possible;

2. The program should be able to handle macros in an ADF;

3. The State Machine entry method should be allowed for ADFs to be processed by the program;

4. A method for adding new Altera primitives to the allowable primitives list should be found with the removal of all internal code referencing of specific primitives.

5. Timing for signal assignment statements should be more generalized.

6. ADF pins which are both input and output should be converted to "inout" mode VHDL ports.

Currently, the program accepts an ADF name and processes the ADF automatically. It would be better if selection of input files were menu driven with the availability to modify program operation. An example of program modification might be the ability to change the invalid name substitution character.

The current version of the transformation program cannot handle macros. Changes could be made to the program to allow macros. This could be done by simply treating macros as primitives.

The State Machine entry form of an ADF is different from the currently allowed entry methods. Modifications could be made to the transformation program to allow this form of ADF.

Currently, the Altera primitives allowed and each primitive's parameter information is hard coded into the transformation program. This requires that, if a new primitive needs to be added to the allowable ones, redesign and recompilation would be necessary. A method should be found to remove all specific references to primitives from the transformation program and change the primitive information to an external information source that the program can access each time the program is invoked.

This version of the transformation program applies the timing constraint "after 5 ns" to each signal assignment statement. A more general way to do this would be to output the timing constraint "DELAY" for each signal assignment statement and assign a constant value to DELAY in the Altera support package. This would allow the delay for signal assignment statements to be varied by the user.

If an ADF has a pin which is both an input and output pin, the transformed file will not successfully analyze. The produced port modes will be incompatible and cause an error during analysis. This error could be corrected by parsing over the INPUTS and OUTPUTS section tokens to determine if the condition exists and then assigning the mode "inout" to the declared port.

Overall, I feel this thesis research was successful and produced a useful transformation program. The transformation program should be helpful to anyone needing to transform ADFs into VHDL entity descriptions.

APPENDIX A

Test Files

Decoder ADF

```
Tom G. Purnhagen
CEG 453
02/02/89
45301.500
5.00
5C090
DECODER/WAIT-STATE GENERATOR/VPA-VMA GENERATOR


OPTIONS:   TURBO = ON

PART:      5C090

INPUTS:    A19, A18, A17, A16, A15, A14, A7, A6,
           A0, AS*, DS*, RW*, RESET*, FC2, FC1, FC0, ROMWS1,
           ROMWS0, CLOCK, E

OUTPUTS:   RAMEN0*, RAMEN1*, ROMEN0*, ROMEN1*, ACIAEN*,
           PIAEN*, VPA*, DTACK*

NETWORK:   %      INPUTS      %
           A19 = INP (A19)
           A18 = INP (A18)
           A17 = INP (A17)
           A16 = INP (A16)
           A15 = INP (A15)
           A14 = INP (A14)
           A7 = INP (A7)
           A6 = INP (A6)
           A0 = INP (A0)
           ASB = INP (AS*)
           DSB = INP (DS*)
           RWB = INP (RW*)
           RESETB = INP (RESET*)
           FC2 = INP (FC2)
           FC1 = INP (FC1)
           FC0 = INP (FC0)
           ROMWS1 = INP (ROMWS1)
           ROMWS0 = INP (ROMWS0)
           CLOCK = INP (CLOCK)
           E = INP (E)

           %      ASYNCHRONOUS CLOCKS      %
```

```
ASBa = CLKB (ASB)
Ea = CLKB (En)

%     DEVICE SELECTS     %

RAMEN0* = CONF (RAMEN0c,)
RAMEN1* = CONF (RAMEN1c,)
ROMEN0*,ROMEN0f = COIF (ROMEN0c,)
ROMEN1*,ROMEN1f = COIF (ROMEN1c,)

ACIAEN* = CONF (ACIAENb,)
PIAEN* = CONF (PIAENb,)

%     BOOT CIRCUIT     %

QA = NORF (DA, ASBa, RESET, GND)
QB = NORF (DB, ASBa, RESET, GND)
QC = NORF (DC, ASBa, RESET, GND)
BOOTB = NORF (BOOTBd, ASBa, RESET, GND)

%     DTACK* GENERATOR     %

WS0 = NORF (VCC, CLOCK, ROMSELc, GND)
WS1 = NORF (WS1d, CLOCK, ROMSELc, GND)
WS2 = NORF (WS2d, CLOCK, ROMSELc, GND)
DTACK* = CONF (DTACKc,)

%     VPA*/VMA GENERATOR     %

VPA = NOJF (VPAj,Ea,GND,ASB,GND)
VMA = NORF (VMAd,CLOCK,ASB,GND)
VPA* = CONF (VPAn,)

EQUATIONS:

%     DEVICE SELECTS     %

RAMEN0c = /(BOOTB * /DSB * /(FC0 * FC1 * FC2) *
          /A19 * /A18 * /A17 * /A16 * /A15 * /A14
          * /A0);
RAMEN1c = /(BOOTB * /DSB * /(FC0 * FC1 * FC2) *
          /A19 * /A18 * /A17 * /A16 * /A15 * /A14
          * A0);
ROMEN0c = /((/BOOTB * /DSB * /A0) + (/ASB * /(FC0
          * FC1 * FC2) * RWB * /A19 * /A18 * /A17
          * /A16 * A15 * /A14 * /A0));
ROMEN1c = /((/BOOTB * /DSB * A0) + (/ASB * /(FC0 *
          FC1 * FC2) * RWB * /A19 * /A18 * /A17 *
          /A16 * A15 * /A14 * A0));
ACIAENc = /(/DSB * /(FC0 * FC1 * FC2) * /A19 *
          /A18 * /A17 * A16 * /A15 * /A14 * /A7 *
          A6 * /A0);
```

```
PIAENc = /(/DSB * /(FC0 * FC1 * FC2) * /A19 * /A18
          * /A17 * A16 * /A15 * /A14 * A7 * /A6 *
          A0);

%     BOOT CIRCUIT     %

DA = /QA;
DB = (QA * /QB) + (/QA * QB);
DC = (QA * QB) + QC;
BOOTBd = (QA * QB * QC) + BOOTB;
RESET = /RESETB;

%     DTACK* GENERATOR     %

DTACKRAMc = /(BOOTB * (/DSB + /ASB * /RWB) * /A19
             * /A18 * /A17 * /A16 * /A15 * /A14);
ROMSELc = ROMEN0f * ROMEN1f;

WS1d = WS0;
WS2d = WS1;
DTACKc = (DTACKRAMc * (/WS0 + ROMWS1 + ROMWS0)
         * (/WS1 + ROMWS1 + /ROMWS0)
         * (/WS2 + /ROMWS1 + ROMWS0))
         + (ROMWS0 * ROMWS1);

%     VPA*/VMA GENERATOR     %

VPAj = ((/A19 * /A18 * /A17 * A16 * /A15 * /A14) +
          (FC0 * FC1 * FC2)) * /ASB;
VPAn = /VPA;
VMAd = (/ACIAENc + /PIAENc) * VPA;
ACIAENb = ACIAENc + /VMA;
PIAENb = PIAENc + /VMA;
En = /E;

END$
```

Swim ADF

Tom G. Purnhagen
CEG 453
02/09/89
45302.100
1.00
5C090
SINGLE-STEP/WATCHDOG TIMER/INTERRUPT MODULE

OPTIONS: TURBO = ON

PART: 5C090

```
INPUTS:   RUNMODE*, STEPMODE*, ADVANCE*, HOLD*, ABORT*,
NOABORT*,
          AS*, CLOCK, E, IRQ2*, IRQ5*

OUTPUTS:  RUN*, BERR*, IPL20*, IPL1*

NETWORK:

          %     INPUTS     %

          RUNMODEb = INP (RUNMODE*)
          STEPMODEb = INP (STEPMODE*)
          ADVANCEb = INP (ADVANCE*)
          HOLDb = INP (HOLD*)
          ABORTb = INP (ABORT*)
          NOABORTb = INP (NOABORT*)
          ASb = INP (AS*)
          CLOCK = INP (CLOCK)
          E = INP (E)
          IRQ2b = INP (IRQ2*)
          IRQ5b = INP (IRQ5*)

          %     ASYNCHRONOUS CLOCKS     %

          ASc = CLKB (AS)
          STEPc = CLKB (QSTEPf)

          %     SINGLE STEP MODULE     %

          QSTEPSf = NOCF (QSTEPS)
          QNSTEPMf = NOCF (QNSTEPM)
          QSTEPf = NOCF (QSTEP)
          QRUN = NORF (VCC, STEPc, ASb, GND)
          QSTEPMODE = NORF (QSTEPM, ASc, QNSTEPMf, GND)
          RUN* = CONF (RUNb,)

          %     WATCHDOG TIMER MODULE     %

          QT1 = NORF (AS, E, WCLf, GND)
          QT2 = NORF (QT1, E, WCLf, GND)
          QT3 = NORF (QT2, E, WCLf, GND)
          BERR = NORF (QT3, E, WCLf, GND)
          BERR* = CONF (BERRb,)
          WCLf = NOCF (WCLR)

          %     INTERRUPT ENCODER MODULE     %

          QSWAf = NOCF (QSWA)
          Q2b = NORF (IRQ2b, CLOCK, GND, GND)
          Q5b = NORF (IRQ5b, CLOCK, GND, GND)
          QABT = NORF (QSWA, CLOCK, GND, GND)
          IPL20* = RONF (IPL20b, CLOCK, GND, GND,)
          IPL1* = RONF (IPL1b, CLOCK, GND, GND,)
```

```
EQUATIONS:

          %     SINGLE STEP MODULE     %

          AS = /ASb;
          QSTEPS = /(ADVANCEb *  QNSTEPS);
          QNSTEPS = /(QSTEPSf* HOLDb);
          QSTEP = QSTEPS * QSTEPM;
          QSTEPM = /(STEPMODEb * QNSTEPMf);
          QNSTEPM = /(QSTEPM * RUNMODEb);
          RUNb = /(/QSTEPMODE + QRUN);

          %     WATCHDOG TIMER MODULE     %

          BERRb = BERR';
          WCLR = ASb # !QNSTEPMf;

          %     INTERRUPT ENCODER MODULE     %

          QSWA = /(ABORTb & QNSWA);
          QNSWA = /(QSWAf * NOABORTb);
          IPL20b = Q5b * /QABT;
          IPL1b= /QABT * /(/Q2b * Q5b);

END$
```

APPENDIX B

Sample Transformed File

Transformed Decoder File

```
-- Tom G. Purnhagen
-- CEG 453
-- 02/02/89
-- 45301.500
-- 5.00
-- 5C090
-- DECODER/WAIT-STATE GENERATOR/VPA-VMA GENERATOR


-- OPTIONS:  TURBO = ON

-- PART:     5C090

library work;
use work.altera_package.all;
entity decoder is
  port (A19 : in altera_logic;
        A18 : in altera_logic;
        A17 : in altera_logic;
        A16 : in altera_logic;
        A15 : in altera_logic;
        A14 : in altera_logic;
        A7 : in altera_logic;
        A6 : in altera_logic;
        A0 : in altera_logic;
        ASv : in altera_logic;
        DSv : in altera_logic;
        RWv : in altera_logic;
        RESETv : in altera_logic;
        FC2 : in altera_logic;
        FC1 : in altera_logic;
        FC0 : in altera_logic;
        ROMWS1 : in altera_logic;
        ROMWS0 : in altera_logic;
        CLOCK : in altera_logic;
        E : in altera_logic;
        RAMEN0v : out altera_logic;
        RAMEN1v : out altera_logic;
        ROMEN0v : out altera_logic;
        ROMEN1v : out altera_logic;
        ACIAENv : out altera_logic;
        PIAENv : out altera_logic;
        VPAv : out altera_logic;
```

```
        DTACKv : out altera_logic);
end decoder;

architecture structured_decoder of decoder is

  signal A19mod : altera_logic;
  signal A18mod : altera_logic;
  signal A17mod : altera_logic;
  signal A16mod : altera_logic;
  signal A15mod : altera_logic;
  signal A14mod : altera_logic;
  signal A7mod : altera_logic;
  signal A6mod : altera_logic;
  signal A0mod : altera_logic;
  signal ASB : altera_logic;
  signal DSB : altera_logic;
  signal RWB : altera_logic;
  signal RESETB : altera_logic;
  signal FC2mod : altera_logic;
  signal FC1mod : altera_logic;
  signal FC0mod : altera_logic;
  signal ROMWS1mod : altera_logic;
  signal ROMWS0mod : altera_logic;
  signal CLOCKmod : altera_logic;
  signal Emod : altera_logic;
  signal ASBa : altera_logic;
  signal Ea : altera_logic;
  signal ROMEN0f : altera_logic;
  signal ROMEN1f : altera_logic;
  signal QA : altera_logic;
  signal QB : altera_logic;
  signal QC : altera_logic;
  signal BOOTB : altera_logic;
  signal WS0 : altera_logic;
  signal WS1 : altera_logic;
  signal WS2 : altera_logic;
  signal VPA : altera_logic;
  signal VMA : altera_logic;
  signal RAMEN0c : altera_logic;
  signal RAMEN1c : altera_logic;
  signal ROMEN0c : altera_logic;
  signal ROMEN1c : altera_logic;
  signal ACIAENc : altera_logic;
  signal PIAENc : altera_logic;
  signal DA : altera_logic;
  signal DB : altera_logic;
  signal DC : altera_logic;
  signal BOOTBd : altera_logic;
  signal RESET : altera_logic;
  signal DTACKRAMc : altera_logic;
  signal ROMSELc : altera_logic;
  signal WS1d : altera_logic;
  signal WS2d : altera_logic;
  signal DTACKc : altera_logic;
```

```
signal VPAj : altera_logic;
signal VPAn : altera_logic;
signal VMAd : altera_logic;
signal ACIAENb : altera_logic;
signal PIAENb : altera_logic;
signal En : altera_logic;


component INP
  port (In1 : in altera_logic; Out1 : out altera_logic);
end component;

component CLKB
  port (In1 : in altera_logic; Out1 : out altera_logic);
end component;

component CONF
  port (In1 : in altera_logic; Oe : in altera_logic;
        Out1 : out altera_logic);
end component;

component COIF
  port (In1 : in altera_logic; Oe : in altera_logic;
        Out1 : out altera_logic; Fbk : out altera_logic);
end component;

component NORF
  port (In1 : in altera_logic; Clk : in altera_logic; C :
        in altera_logic; P : in altera_logic; Fbk : out
        ltera_logic);
end component;

component NOJF
  port (Jn : in altera_logic; Clk : in altera_logic; Kin :
        in altera_logic; C : in altera_logic; P : in
        altera_logic; Fbk : out altera_logic);
end component;


begin

--      DEVICE SELECTS

RAMEN0c <= not (BOOTB and not DSB and not (FC0mod and FC1mod
               and FC2mod) and not A19mod and not A18mod and
               not A17mod and not A16mod and not A15mod and
               not A14mod and not A0mod)  after 5 ns;

RAMEN1c <= not (BOOTB and not DSB and not (FC0mod and FC1mod
               and FC2mod) and not A19mod and not A18mod and
               not A17mod and not A16mod and not A15mod and
               not A14mod and A0mod)  after 5 ns;
```

```
ROMEN0c <= not ((not BOOTB and not DSB and not A0mod) or
               (not ASB and not (FC0mod and FC1mod and
               FC2mod) and RWB and not A19mod and not A18mod
               and not A17mod and not A16mod and A15mod and
               not A14mod and not A0mod))  after 5 ns;

ROMEN1c <= not ((not BOOTB and not DSB and A0mod) or (not
               ASB and not (FC0mod and FC1mod and FC2mod)
               and RWB and not A19mod and not A18mod and not
               A17mod and not A16mod and A15mod and not
               A14mod and A0mod))  after 5 ns;

ACIAENc <= not (not DSB and not (FC0mod and FC1mod and
               FC2mod) and not A19mod and not A18mod and not
               A17mod and A16mod and not A15mod and not
               A14mod and not A7mod and A6mod and not A0mod)
               after 5 ns;

PIAENc <= not (not DSB and not (FC0mod and FC1mod and
          FC2mod) and not A19mod and not A18mod and not
          A17mod and A16mod and not A15mod and not A14mod
          and A7mod and not A6mod and A0mod)  after 5 ns;

--      BOOT CIRCUIT

DA <= not QA  after 5 ns;

DB <= (QA and not QB) or (not QA and QB)  after 5 ns;

DC <= (QA and QB) or QC  after 5 ns;

BOOTBd <= (QA and QB and QC) or BOOTB  after 5 ns;

RESET <= not RESETB  after 5 ns;

--      DTACK* GENERATOR

DTACKRAMc <= not (BOOTB and (not DSB or not ASB and not RWB)
             and not A19mod and not A18mod and not A17mod
             and not A16mod and not A15mod and not A14mod)
             after 5 ns;

ROMSELc <= ROMEN0f and ROMEN1f  after 5 ns;

WS1d <= WS0  after 5 ns;

WS2d <= WS1  after 5 ns;

DTACKc <= (DTACKRAMc and (not WS0 or ROMWS1mod or ROMWS0mod)
          and (not WS1 or ROMWS1mod or not ROMWS0mod) and
          (not WS2 or not ROMWS1mod or ROMWS0mod)) or
          (ROMWS0mod and ROMWS1mod)  after 5 ns;

--      VPA*/VMA GENERATOR
```

```
VPAj <= ((not A19mod and not A18mod and not A17mod and
          A16mod and not A15mod and not A14mod) or (FC0mod
          and FC1mod and FC2mod)) and not ASB after 5 ns;

VPAn <= not VPA  after 5 ns;

VMAd <= (not ACIAENc or not PIAENc) and VPA  after 5 ns;

ACIAENb <= ACIAENc or not VMA  after 5 ns;

PIAENb <= PIAENc or not VMA  after 5 ns;

En <= not Emod  after 5 ns;

--      INPUTS

  U0 : INP
    port map (In1 => A19, Out1 => A19mod);

  U1 : INP
    port map (In1 => A18, Out1 => A18mod);

  U2 : INP
    port map (In1 => A17, Out1 => A17mod);

  U3 : INP
    port map (In1 => A16, Out1 => A16mod);

  U4 : INP
    port map (In1 => A15, Out1 => A15mod);

  U5 : INP
    port map (In1 => A14, Out1 => A14mod);

  U6 : INP
    port map (In1 => A7, Out1 => A7mod);

  U7 : INP
    port map (In1 => A6, Out1 => A6mod);

  U8 : INP
    port map (In1 => A0, Out1 => A0mod);

  U9 : INP
    port map (In1 => ASv, Out1 => ASB);

  U10 : INP
    port map (In1 => DSv, Out1 => DSB);

  U11 : INP
    port map (In1 => RWv, Out1 => RWB);

  U12 : INP
```

```
    port map (In1 => RESETv, Out1 => RESETB);

  U13 : INP
    port map (In1 => FC2, Out1 => FC2mod);

  U14 : INP
    port map (In1 => FC1, Out1 => FC1mod);

  U15 : INP
    port map (In1 => FC0, Out1 => FC0mod);

  U16 : INP
    port map (In1 => ROMWS1, Out1 => ROMWS1mod);

  U17 : INP
    port map (In1 => ROMWS0, Out1 => ROMWS0mod);

  U18 : INP
    port map (In1 => CLOCK, Out1 => CLOCKmod);

  U19 : INP
    port map (In1 => E, Out1 => Emod);

--      ASYNCHRONOUS CLOCKS

  U20 : CLKB
    port map (In1 => ASB, Out1 => ASBa);

  U21 : CLKB
    port map (In1 => En, Out1 => Ea);

--      DEVICE SELECTS

  U22 : CONF
    port map (In1 => RAMEN0c, Oe => VCC, Out1 => RAMEN0v);

  U23 : CONF
    port map (In1 => RAMEN1c, Oe => VCC, Out1 => RAMEN1v);

  U24 : COIF
    port map (In1 => ROMEN0c, Oe => VCC, Out1 => ROMEN0v,
              Fbk => ROMEN0f);

  U25 : COIF
    port map (In1 => ROMEN1c, Oe => VCC, Out1 => ROMEN1v,
              Fbk => ROMEN1f);

  U26 : CONF
    port map (In1 => ACIAENb, Oe => VCC, Out1 => ACIAENv);

  U27 : CONF
    port map (In1 => PIAENb, Oe => VCC, Out1 => PIAENv);

--      BOOT CIRCUIT
```

```
  U28 : NORF
    port map (In1 => DA, Clk => ASBa, C => RESET, P => GND,
              Fbk => QA);

  U29 : NORF
    port map (In1 => DB, Clk => ASBa, C => RESET, P => GND,
              Fbk => QB);

  U30 : NORF
    port map (In1 => DC, Clk => ASBa, C => RESET, P => GND,
              Fbk => QC);

  U31 : NORF
    port map (In1 => BOOTBd, Clk => ASBa, C => RESET, P =>
              GND, Fbk => BOOTB);

--      DTACK* GENERATOR

  U32 : NORF
    port map (In1 => VCC, Clk => CLOCKmod, C => ROMSELc, P
              => GND, Fbk => WS0);

  U33 : NORF
    port map (In1 => WS1d, Clk => CLOCKmod, C => ROMSELc, P
              => GND, Fbk => WS1);

  U34 : NORF
    port map (In1 => WS2d, Clk => CLOCKmod, C => ROMSELc, P
              => GND, Fbk => WS2);

  U35 : CONF
    port map (In1 => DTACKc, Oe => VCC, Out1 => DTACKv);

--      VPA*/VMA GENERATOR

  U36 : NOJF
    port map (Jn => VPAj, Clk => Ea, Kin => GND, C => ASB, P
              => GND, Fbk => VPA);

  U37 : NORF
    port map (In1 => VMAd, Clk => CLOCKmod, C => ASB, P =>
              GND, Fbk => VMA);

  U38 : CONF
    port map (In1 => VPAn, Oe => VCC, Out1 => VPAv);

end structured_decoder;
```

APPENDIX C

Supplemental VHDL Package Source Code

Altpk.vhd

```
package altera_package is

type altera_logic is ('0', '1', 'Z');

signal VCC    : altera_logic := '1';
signal GND    : altera_logic := '0';

function "not" (L : altera_logic) return altera_logic;
function "and" (L,R : altera_logic) return altera_logic;
function "or" (L,R : altera_logic) return altera_logic;

end altera_package;


package body altera_package is

function "or" (L,R : altera_logic) return altera_logic is
begin
     if l = 'Z' or r = 'Z' then return '1';
     elsif l = 'Z' or r = '1' then return '1';
     elsif l = '1' or r = 'Z' then return '1';
     elsif l = '1' or r = '1' then return '1';
     else return '0';
     end if;

end;

function "not" (L : altera_logic) return altera_logic is
begin
     if l = 'Z' then return '0';
     elsif l = 'Z' then return '0';
     elsif l = '1' then return '0';
     elsif l = '0' then return '1';
     end if;

end;

function "and" (L,R : altera_logic) return altera_logic is
begin
     if l = 'Z' and r = 'Z' then return '1';
     elsif l = 'Z' and r = '1' then return '1';
     elsif l = '1' and r = '1' then return '1';
     else return '0';
     end if;
```

```
end;
end altera_package;
```

APPENDIX D

User Manual

Required Files

Alttovhd is the name of the transformation program and is required to perform the transformation process.

Altpk.vhd is the name of the VHDL support package. It must located in the VHDL library "work" and is required to successfully analyze a transformed ADF with a VHDL analyzer.

Command Line Entry

The command line entry to invoke the transformation program is of the form:

alttovhd [*d:*][*pathname*][*input_file_name*.adf] [> [*d:*][*pathname*][*output_file_name*.vhd]] .

d: is the drive specification if other than the current drive.

pathname is the path to input file if other than the current directory.

input_file_name is the input ADF and must have the extension ".adf".

output_file_name is the file name that standard output will be directed to and should have the extension ".vhd".

The transformation program "alttovhd" will prompt the user for the input file name if *input_file_name*.adf is not included in the command line entry. The output will default

to the screen if standard output is not redirected to *output_file_name*.vhd.

Alttovhd will prompt the user as major portions of the transformation process are accomplished.

APPENDIX E

Source Code for Transformation Program

adftovhd.c

```
/*************************************************************
*                                                            *
*                        adftovhd.c                          *
*                                                            *
*************************************************************
*
*  Module:     adftovhd.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedure
*          for driving the transformation
*          of an Altera Design File to a VHDL
*          entity description.
*
*/

#include <adftovhd.h>

/*************************************************************
*
*  Function:   main
*
*  Interface:  main (int argc, char *argv[])
*
*  Purpose:     This procedure drives the transformation
*          process by calling the procedures contained
*          in the external modules.
*
*************************************************************/

main(int argc, char *argv[])
{
     FILE *in_stream;
     char ch, *input_file;
     int len, conclude = 0, start_or_continue = 1;

     clrscr();
     cputs(HEADER1);
     gotoxy(1,2);
     cputs(HEADER2);
     gotoxy(1,3);
     cputs(HEADER3);
     gotoxy(1,5);
     if (argv[1] == NULL)
```

```
            {
            input_file = get_file_name();
            }
      else
            {
            if (is_good_file_name(argv[1]))
                  {
                  strcpy(input_file, argv[1]);
                  }
            else
                  {
                  screen_message("FILE <%s> NOT FOUND\r\n\n");
                  input_file = get_file_name();
                  }
            }
      in_stream = get_file_stream(input_file);


      build_tokens(in_stream);
      set_token_index(FIRST);
      while (! is_section_header(top_token()))
            {
            make_comment(top_token(), start_or_continue);
            advance_to_next_token();
            }
      generate_entity_declaration(input_file);
      if (set_to_network_section())
            {
            set_to_previous_token();
            }
      build_entity_architecture(input_file);
}
```

adftovhd.h

```
/*************************************************************
*                                                            *
*                        adftovhd.h                          *
*                                                            *
*************************************************************
*
*  Module:      adftovhd.h
*
*  Version:     1.0
*
*  Purpose:      This is the header file for adftovhd.c.
*
*
*
*
*/
```

```
#include <stdio.h>
#include <conio.h>
#include <altera_t.h>
#include <display.h>
```

alt_equa.h

```
/*************************************************************
*                                                            *
*                        alt_equa.h                          *
*                                                            *
*************************************************************
*
*  Module:     alt_equa.h
*
*  Version:    1.0
*
*  Purpose:     This is the header file for alt_equa.c.
*
*
*
*
*/

#include <stdio.h>
#include <altera_t.h>
#include <asciidef.h>


extern void generate_signal_assignment_statements();
extern int is_boolean_operator();
extern void output_identifier();
extern void output_signal_assignment_symbol();
extern void output_boolean_identitfier();
extern void output_string();
extern void terminate_signal_assignment();
extern int is_delimiter_semi();
```

asciidef.h

```
/*************************************************************
*                                                            *
*                        asciidef.h                          *
*                                                            *
*************************************************************
*
*  Module:     asciidef.h
*
*  Version:    1.0
*
```

```
*  Purpose:     This is a header file that contains the
*          ASCII definitions use in the adftovhd
*          program.
*
*
*
*/

#define TAB     9
#define LF     10
#define CR     13
#define SPACE  32
#define EXCLA  33
#define LBSYM  35
#define PCENT  37
#define LOGAN  38
#define SQUOT  39
#define LPARN  40
#define RPARN  41
#define ASTRC  42
#define PLUS   43
#define COMA   44
#define SLASH  47
#define SEMI   59
#define EQUAL  61
#define BSLASH 92
```

name_mod.h

```
/*************************************************************
*                                                            *
*                        name_mod.h                          *
*                                                            *
*************************************************************
*
*  Module:     name_mod.h
*
*  Version:    1.0
*
*  Purpose:     This is a header file for name_mod.c.
*
*
*
*
*
*/

#include <stdio.h>
#include <display.h>
#include <altera_t.h>
```

```
extern char *is_making_legal_vhdl_name(char *token_name);
extern void advance_past_comment();
extern void search_and_change(char *the_name, char
                          *new_name, char *input_pin_name);
extern void modify_name(char *node_name, char
                                  *input_pin_name);
extern void check_and_change_identifiers();
extern void remove_illegal_vhdl_name_characters();
extern char *concat_strings(char *prefix_string, char
                                        *suffix_string);
```

alt_inst.h

```
/*************************************************************
*                                                            *
*                        alt_inst.h                          *
*                                                            *
*************************************************************
*
*  Module:     alt_inst.h
*
*  Version:    1.0
*
*  Purpose:     This is the header file for alt_inst.c.
*
*
*
*
*/


#include <stdio.h>
#include <altera_t.h>
#include <calloc.h>

extern int is_delimiter_left_paren();
extern int is_delimiter_right_paren();
extern void output_comment_statement();
extern void output_instantiation_close();
extern void output_association(char *local, char *actual,
                              int component_mark);
extern void output_instantiation_header(char
                                  *component_mark);
extern void output_component_instantiations(char
                          **component_inputs_outputs, char
                          *component_mark);
extern void output_instantiation_close();
extern char **collect_inputs_and_append(char **outputs);
extern char *determine_component_mark();
extern void generate_instantiations();
extern int is_equation_section();
```

calloc.h

```
/*************************************************************
*                                                            *
*                          calloc.h                          *
*                                                            *
*************************************************************
*
*  Module:     calloc.h
*
*  Version:    1.0
*
*  Purpose:     This is the header file for calloc.c.
*
*
*
*
*/

#include <asciidef.h>
#include <altera_t.h>
#include <stdio.h>


#define MIN_FILE_SIZE 1000


extern char *get_new_ptr (int number_of_chars);
extern char *append_to_token (char *token_ptr, char
                                    *new_char_str);
extern int is_delimiter (int ascii_char, int
                              in_equation_section);
extern char **get_token_array(long filesize);
extern long get_file_length(FILE *in);
```

new_fncs.h

```
/*************************************************************
*                                                            *
*                        new_fncs.h                          *
*                                                            *
*************************************************************
*
*  Module:     new_fncs.h
*
*  Version:    1.0
*
*  Purpose:     This is the header file for new_fncs.c.
*
*
```

```
*
*
*/

#include <stdio.h>
#include <display.h>
#include <altera_t.h>
#include <conio.h>

extern int is_good_file_name(char *input_file);
extern FILE *get_file_stream();
extern char *get_file_name();
extern int has_leading_periods();
extern char *is_removing_leading_periods();
extern char *is_prefixing_xycord_to();
extern char *is_removing_pin_reference(char *port_name);
extern void generate_entity_declaration(char
                                        *the_file_name);
extern char *get_port_name();
extern void output_port(char *name, char *mode, char *type);
extern void begin_port_decl();
extern void end_port_decl();
extern void end_entity_decls(char *entity_name);
```

tokens.h

```
/*************************************************************
*                                                            *
*                        tokens.h                            *
*                                                            *
*************************************************************
*
*  Module:     tokens.h
*
*  Version:    1.0
*
*  Purpose:     This is the header file for altera_p.c.
*
*
*
*
*/

#include <stdio.h>
#include <altera_t.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#include <display.h>

#ifndef TOKENS_H
#define TOKENS_H
```

```
typedef struct {
     char **token_array;
     char **token_type_array;
     int  index;
     int  total_entries;
     } tokens_struct;

typedef char *tokon;


extern char *get_identifier_type();
extern void advance_to_next_token();
extern tokon *get_token();
extern tokon *next_token();
extern char *get_delimiter_type();
extern void build_tokens();
extern int is_identifier_variable();
extern int set_to_previous_token();
extern int is_delimiter_comment();
extern int end_of_tokens();
extern char *get_token_type();
extern tokon *top_token();
extern tokon *get_token();
extern int get_token_index();
extern void set_token_index(int index);
extern int beginning_of_network_section();


#endif
```

altera_t.h

```
/*************************************************************
*                                                            *
*                        altera_t.h                          *
*                                                            *
*************************************************************
*
*  Module:     altera_t.h
*
*  Version:    1.0
*
*  Purpose:     This header file contains the type
*               definitions for the Altera transformation
*               types.
*
*
*/
```

```
#ifndef ALTERA_TYPES
#define ALTERA_TYPE "altera_logic"
#define MAX_LINE_LENGTH_DECLS 77
#define FIRST 0
#define SECOND 1
#define MAX_NUM_FUNCTIONS 100
#define STANDARD_STR_LEN 40
#define MAX_NUM_COMPONENT_OUTPUTS 8
#define MAX_NAME_LEN 40
#define MAX_BUFFER_SIZE 40
#define DELAY "5 ns"
#define HEADER1 "                                    ALTERA to
VHDL"
#define HEADER2 "                                    File
Translater"
#define HEADER3 "                                        ver
1.0"
#define REPLACEMENT_CHARACTER 'v'

#ifndef TRUTH_LOGIC
#define TRUE 1
#define FALSE 0
#endif

#endif
```

display.h

```
/*************************************************************
*                                                            *
*                        display.h                           *
*                                                            *
*************************************************************
*
*  Module:     display.h
*
*  Version:    1.0
*
*  Purpose:     This header file contains a display macro.
*
*
*
*
*/

#include <conio.h>


#ifndef DISPLAY_FUNC
#define screen_message(s_message) cputs(s_message);
#endif
```

altransf.h

```
/*************************************************************
*                                                            *
*                        altransf.h                          *
*                                                            *
*************************************************************
*
*  Module:     altransf.h
*
*  Version:    1.0
*
*  Purpose:     This is the header file for altransf.c.
*
*
*
*
*/

#include <stdio.h>
#include <asciidef.h>
#include <tokens.h>


#ifndef ALTRANSF_H

#include <tokens.h>


extern void declare_entity();
extern void error_message();
extern void error_message2();
extern int is_section_header(tokon *current_token);
extern char is_code_for(tokon *current_token);
extern void make_comment(tokon *current_token, int
                                        start_continue);

#endif
```

ent_arch.hc

```
/*************************************************************
*                                                            *
*                        ent_arch.h                          *
*                                                            *
*************************************************************
*
*  Module:     ent_arch.h
*
```

```
*  Version:    1.0
*
*  Purpose:     This is the header file for ent_arch.c.
*
*
*
*
*/

#include <stdio.h>
#include <altera_t.h>
#include <display.h>


#ifndef ENT_ARCH_H

extern int set_to_inputs_section();
extern char *get_input_pin_name();
extern void end_the_signal_decl();
extern int is_io_pin_name(char *signal_name, char
                                **list_of_io_pin_names);
extern char **get_list_of_io_pins();
extern int set_to_input_section();
extern int is_delimiter_comma();
extern int at_network_section();
extern char *get_ports_for(char *component_name);
extern void output_decl(char *name, char *mode, char *type,
                                          int flag);
extern void output_component_decl_header(char
                                    *component_name);
extern void output_component_decl_close();
extern void output_component_ports(char *ports);
extern void get_port_values(char *ports, char *name, char
                                    *mode, char *type);
extern int set_to_left_paren();
extern int get_parameter_count();
extern void append_number_of_parameters_if_necessary(char
                                    *component_name);
extern int comma_count();
extern void build_entity_architecture(char *input_file);
extern void end_body(char *entity_name);
extern void begin_body();
extern void generate_signal_decls();
extern void generate_component_decls();
extern void generate_architecture_header(char *entity_name);
extern int set_to_equations_section();
extern void output_signal(char *signal_name, char *type);
extern char *get_signal_name();
extern void output_signal_header();
extern void end_signal_decls();
extern int is_delimiter_equal();
extern void read_past_equation();
extern int set_to_network_section();
```

```
extern void output_component_decl(char *component_name);

#endif
```

calloc.c

```
/*************************************************************
*                                                            *
*                          calloc.c                          *
*                                                            *
*************************************************************
*
*  Module:     calloc.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedures and
*         functions for manipulating the tokens
*         data structure of the adftovhd driver
*         program.  Specifically, this module is
*         concerned with the allocating memory and
*         evaluating the tokens to determine type.
*/

#include <calloc.h>

/*************************************************************
*
*  Function:   get_new_ptr
*
*  Interface:  char *get_new_ptr(int number_of_chars)
*
*  Purpose:     This function returns a new pointer to a
*    string memory space of size "number_of_chars"
*
*************************************************************/

char *get_new_ptr (int number_of_chars)
{
     return ( (char *)calloc(number_of_chars,sizeof(char)));
}

/*************************************************************
*
*  Function:   is_component
*
*  Interface:  int is_component(char *token)
*
*  Purpose:     This function returns true is "token" is an
*         Altera primitive.
*
*************************************************************/
```

```
int is_component(char *token)
{
     int length = strlen(token);

     return (
           strcmp(token, "INP")        == 0 ||
           strcmp(token, "LINP")       == 0 ||
           strncmp(token, "AND", 3)    == 0 ||
           strncmp(token, "BAND", 4)   == 0 ||
           strcmp(token, "BBUF")       == 0 ||
           strcmp(token, "CLKB")       == 0 ||
           strncmp(token, "NAND", 4)   == 0 ||
           strncmp(token, "BNAND", 5)  == 0 ||
           strncmp(token, "NOR", 3)    == 0 ||
           strncmp(token, "BNOR", 4)   == 0 ||
           strcmp(token, "NOT")        == 0 ||
           strncmp(token, "OR", 2)     == 0 ||
           strncmp(token, "BOR", 3)    == 0 ||
           strcmp(token, "XNOR")       == 0 ||
           strcmp(token, "XOR")        == 0 ||
           strcmp(token, "COCF")       == 0 ||
           strcmp(token, "COIF")       == 0 ||
           strcmp(token, "COLF")       == 0 ||
           strcmp(token, "CONF")       == 0 ||
           strcmp(token, "CORF")       == 0 ||
           strcmp(token, "JOJF")       == 0 ||
           strcmp(token, "JONF")       == 0 ||
           strcmp(token, "NOCF")       == 0 ||
           strcmp(token, "NOJF")       == 0 ||
           strcmp(token, "NORF")       == 0 ||
           strcmp(token, "NOSF")       == 0 ||
           strcmp(token, "NOTF")       == 0 ||
           strcmp(token, "ROCF")       == 0 ||
           strcmp(token, "ROIF")       == 0 ||
           strcmp(token, "ROLF")       == 0 ||
           strcmp(token, "RONF")       == 0 ||
           strcmp(token, "RORF")       == 0 ||
           strcmp(token, "SONF")       == 0 ||
           strcmp(token, "SOSF")       == 0 ||
           strcmp(token, "TOIF")       == 0 ||
           strcmp(token, "TONF")       == 0 ||
           strcmp(token, "TOTF")       == 0 ||
           strcmp(token, "BUSX")       == 0 ||
           strcmp(token, "LBUSI")      == 0 ||
           strcmp(token, "LBUSO")      == 0 ||
           strcmp(token, "LINP8")      == 0 ||
           strcmp(token, "RBUSI")      == 0 ||
           strcmp(token, "RINP8")      == 0 );
}

/*************************************************************
*
*  Function:   append_to_token
```

```
*
*  Interface:  char *append_to_token(char *token_ptr,
*                          char *new_char_str)
*
*  Purpose:     This function returns a pointer to a string
*         which is the result of appending
*         "new_char_str" to "token_ptr".
*
*************************************************************/

char *append_to_token (char *token_ptr, char *new_char_str)
{
     return(strcat( token_ptr, new_char_str));
}

/*************************************************************
*
*  Function:   get_file_length
*
*  Interface:  long get_file_length(FILE *in)
*
*  Purpose:     This function returns the length of the
*         file pointed to by "in".
*
*************************************************************/

long get_file_length(FILE *in)
{
     long filesize = 0;

     while (fgetc(in) != EOF)
          filesize++;
     rewind(in);
     if (filesize < MIN_FILE_SIZE)
          return(MIN_FILE_SIZE);
     else
          return(filesize);
}

/*************************************************************
*
*  Function:   get_token_array
*
*  Interface:  char **get_token_array(long filesize)
*
*  Purpose:     This function returns a pointer to an array
*         of pointers that number "filesize".
*
*************************************************************/

char **get_token_array(long filesize)
{
     int i;
     char **temp_array;
```

```
     temp_array = (char **)malloc(filesize * sizeof(char
                                                        *));
     for (i = 0; i < filesize; i++)
          temp_array[i] = NULL;
     return(temp_array);
}

/*********************************************************
*
*  Function:   is_delimiter
*
*  Interface:  int is_delimiter(int ascii_char,
*                    int in_equation_section)
*
*  P rpose:     This function returns true if "ascii_char"
*         is a delimiter.
*
*********************************************************/

int is_delimiter (int ascii_char, int in_equation_section)
{
     int next_character;

     if (ascii_char == TAB     ||
         ascii_char == LF      ||
         ascii_char == CR      ||
         ascii_char == SPACE   ||
         ascii_char == PCENT   ||
         ascii_char == LPARN   ||
         ascii_char == RPARN   ||
         ascii_char == EQUAL   ||
         ascii_char == COMA    )
          {
          return(TRUE);
          }
     else if (in_equation_section)
          {
          if (ascii_char == ASTRC   ||
              ascii_char == PLUS    ||
              ascii_char == SLASH   ||
              ascii_char == EXCLA   ||
              ascii_char == LBSYM   ||
              ascii_char == LOGAN   ||
              ascii_char == SQUOT   ||
              ascii_char == SEMI    )
              {
              return(TRUE);
              }
          }
     return(FALSE);
}
```

new_fncs.c

```
/*********************************************************
*                                                        *
*                        new_fncs.c                      *
*                                                        *
*********************************************************
*
*  Module:     new_fncs.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedures and
*          functions for manipulating the tokens
*          data structure of the adftovhd driver
*          program.  Specifically, this module is
*          concerned with miscellaneous functions.
*/

#include <new_fncs.h>


/*********************************************************
*
*  Function:   get_file_stream
*
*  Interface:  FILE *get_file_stream(char *file_name)
*
*  Purpose:     This function returns a FILE pointer to
*          a file stream for the file "file_name".
*
*********************************************************/

FILE *get_file_stream(char *file_name)
{
     FILE *input_stream;

     input_stream = (FILE *)malloc(sizeof(FILE));
     if ((input_stream = fopen(file_name, "rt")) == NULL)
          screen_message("File Not Found.\r\n");
     return(input_stream);
}

/*********************************************************
*
*  Function:   is_good_file_name
*
*  Interface:  int is_good_file_name(char *input_file)
*
*  Purpose:     This function returns true if "input_file"
*          exists.
*
*********************************************************/
```

```
int is_good_file_name(char *input_file)
{
     FILE *input_stream;

     if ((input_stream = fopen(input_file, "rt")) == NULL)
          {
          return(FALSE);
          }
     else
          {
          fclose(input_stream);
          return(TRUE);
          }
}

/*************************************************************
*
*  Function:   get_file_name
*
*  Interface:  char *get_file_name()
*
*  Purpose:     This function returns a pointer to a file
*          name retrieved from the user.
*
*************************************************************/

char *get_file_name()
{

     FILE *input_stream;
     char *temp_file_name, *buffer, *temp_str = " ";
     int file_not_found = 1, ch;

     input_stream = (FILE *)malloc(sizeof(FILE));
     temp_file_name = (char *)malloc(256 * sizeof(char));
     while (file_not_found)
          {
          screen_message("\r\nEnter the file name.\r\n");
          strcpy(temp_file_name, "\0");
          while ((ch = getche()) != 13)
               {
               temp_str[0] = ch;
               strcat(temp_file_name, temp_str);
               }
          screen_message("\r\n");
          if ((input_stream = fopen(temp_file_name, "rt"))
                                                  == NULL)
               {
               screen_message("File <");
               screen_message(temp_file_name);
               screen_message("> Not Found.\r\n");
               }
          else
```

```
                {
                file_not_found = 0;
                }
            }
        fclose(input_stream);
        return(temp_file_name);

}

/*************************************************************
*
*  Function:   get_entity_name
*
*  Interface:  char *get_entity_name(char *the_file_name)
*
*  Purpose:     This function returns a pointer to an entity
*          name created from "the_file_name".
*
*************************************************************/

char * get_entity_name(char *the_file_name)
{
        int  number_of_characters = 0;
        char *temp_ptr, *start_ptr, *end_ptr;

        if ((start_ptr = strrchr(the_file_name, '\\')) != NULL)
            {
            start_ptr = start_ptr + 1;
            }
        else if ((start_ptr = strrchr(the_file_name, ':')) !=
                                                          NULL)
            {
            start_ptr = start_ptr + 1;
            }
        else
            {
            start_ptr = the_file_name;
            }
        if ((end_ptr = strchr(the_file_name, '.')) != NULL)
            {
            number_of_characters = (end_ptr - start_ptr);
            }
        else
            {
            number_of_characters = strlen(start_ptr);
            }
        temp_ptr = (char *)malloc((number_of_characters + 1) *
                                              sizeof(char));
        *temp_ptr = NULL;
        return(strncat(temp_ptr, start_ptr,
                            number_of_characters));

}
```

```
/*********************************************************
*
*  Function:   has_leading_periods
*
*  Interface:  int has_leading_periods(char *str_name)
*
*  Purpose:     This function returns true if there are
*         leading periods on "str_name".
*
*********************************************************/

int has_leading_periods(char *str_name)
{
     return(*str_name == '.');
}

/*********************************************************
*
*  Function:   is_removing_leading_periods
*
*  Interface:  char *is_removing_leading_periods(
*                        char *str_name)
*
*  Purpose:     This function returns a pointer to
*         "str_name" after the periods have been
*         removed.
*
*********************************************************/

char * is_removing_leading_periods(char *str_name)
{
     int index = 0;
     char *temp_str;

     while(*(str_name + index) == '.')
          {
          index++;
          }
     temp_str = (char *)malloc(strlen(str_name + index) *
                                        sizeof(char));
     strcpy(temp_str, str_name + index);
     return(temp_str);
}

/*********************************************************
*
*  Function:   is_prefixing_xycord_to
*
*  Interface:  char *is_prefixing_xycord_to(
*                    char *input_string)
*
*  Purpose:     This function returns a pointer to
*         "input_string" after prefixing "xycord".
*
```

```
*************************************************************/

char * is_prefixing_xycord_to(char *input_string)
{
     char *resolved_name = "xycord\0";

     return(strcat(resolved_name, input_string));
}


/*************************************************************
*
*  Function:   is_removing_pin_reference
*
*  Interface:  char *is_removing_pin_reference(
*                                char *port_name)
*
*  Purpose:     This function returns a pointer to
*          "port_name" after removing a pin reference.
*
*************************************************************/

char *is_removing_pin_reference(char *port_name)
{
     char *temp_ptr;

     if((temp_ptr = strchr(port_name, '@')) != NULL)
          {
          *temp_ptr = '\0';
          }
     return(port_name);
}

/*************************************************************
*
*  Function:   output_port
*
*  Interface:  void output_port(char *name, char *mode,
*                                char *type)
*
*  Purpose:     This function outputs a string built from
*          "name", "mode", and "type".
*
*************************************************************/

void output_port(char *name, char *mode, char *type)
{
     printf("%s : %s %s", name, mode, type);
}

/*************************************************************
*
*  Function:   begin_port_decl
*
```

```
*  Interface:  void begin_port_decl()
*
*  Purpose:     This function outputs the string
*          "port (" for the beginning of a port decl.
*
*************************************************************/

void begin_port_decl()
{
     printf("  port (");
}

/*************************************************************
*
*  Function:   end_port_decl
*
*  Interface:  void end_port_decl()
*
*  Purpose:     This function outputs the string
*          ");" to close a port declaration.
*
*************************************************************/

void end_port_decl()
{
     printf(");\n");
}

/*************************************************************
*
*  Function:   end_entity_decl
*
*  Interface:  void end_entity_decl(char *entity_name)
*
*  Purpose:     This function outputs the string
*          that closes an entity declaration.
*
*************************************************************/

void end_entity_decl(char *entity_name)
{
     printf("end %s;\n\n", entity_name);
}

/*************************************************************
*
*  Function:   more_ports_present
*
*  Interface:  int more_ports_present()
*
*  Purpose:     This function returns true if more ports
*          need to be processed.
*
*************************************************************/
```

```
int more_ports_present()
{
     int found_port = 0, token_index, temp_return_value;
     char *dummy_token;

     token_index = get_token_index();
     while (! is_delimiter_comment())
          {
          advance_to_next_token();
          }
     advance_to_next_token();
     while (! beginning_of_network_section())
          {
          if ( is_delimiter_comment())
               {
               dummy_token = get_token();
               temp_return_value = more_ports_present();
               set_token_index(token_index);
               if (temp_return_value)
                    {
                    return(temp_return_value);
                    }
               else
                    {
                    return(found_port);
                    }
               }
          if (is_identifier_variable())
               {
               set_token_index(token_index);
               return(++found_port);
               }
          dummy_token = get_token();
          }
          set_token_index(token_index);
          return(found_port);
}


/*************************************************************
*
*  Function:   get_port_name
*
*  Interface:  char *get_port_name()
*
*  Purpose:     This function returns a pointer to a port
*         port identifier.
*
*************************************************************/

char *get_port_name()
{
```

```
int searching_for_token = 1, start_continue_comment =
                                                  1,
    stop_comment = 0;
char *temp_name = "", *print_type;
char *current_token;

while (searching_for_token)
     {
     if (end_of_tokens() || is_delimiter_comment() ||
         is_section_header(top_token()) ||
                   is_identifier_variable())
          {
          searching_for_token = 0;
          }
     else
          {
          current_token = get_token();
          }
     }
if (is_delimiter_comment())
     {
     current_token = get_token();
     if (more_ports_present())
          {
          printf(";\n\n");
          }
     else
          {
          end_port_decl();
          printf("\n");
          }
     while(! is_delimiter_comment())
          {
          current_token = get_token();
          make_comment(current_token,
                   start_continue_comment);
          }
     current_token = get_token();
     make_comment(current_token, stop_comment);
     printf("        ");
     return("comment_interupt");
     }
else if (end_of_tokens())
     {
     return(NULL);
     }
else if (is_section_header(top_token()))
     {
     current_token = get_token();
     return(NULL);
     }
else
     {
     current_token = get_token();
```

```
            temp_name =
                 is_removing_pin_reference(current_token);
            return(temp_name);
            }
}

/*********************************************************
*
*  Function:   generate_entity_declaration
*
*  Interface:  void generate_entity_declaration(
*                        char *the_file_name)
*
*  Purpose:     This procedure generates the entity
*         declaration from the tokens.
*
*********************************************************/

void generate_entity_declaration(char *the_file_name)
{
     int port_decl_started = 0, last_was_comment = 0,
                                   first_port = TRUE;
     char *current_port, *in_mode = "in", *out_mode = "out",
          *type = ALTERA_TYPE, *entity_name, *last_port;

     entity_name = get_entity_name(the_file_name);
     screen_message("\r\nmaking entity
                              declaration....\r\n");
     printf("library work;\n");
     printf("use work.altera_package.all;\n");
     printf("entity %s is\n", entity_name);
     if (set_to_inputs_section())
     advance_to_next_token();
     {
     while ((current_port = get_port_name()) != NULL)
          {
          if (! port_decl_started)
               {
               begin_port_decl();
               port_decl_started = 1;
               }
          if (strcmp(current_port, "comment_interupt") == 0)
               {
               last_was_comment = 1;
               }
          else
               {
               if (! last_was_comment)
                    {
                    if (first_port)
                         {
                         output_port(current_port, in_mode,
                                        type);
                         first_port = FALSE;
```

```
                    }
                else
                    {
                    printf(";\n        ");
                    output_port(current_port, in_mode,
                                                type);
                    }
                }
            else
                {
                output_port(current_port, in_mode,
                                                type);
                last_was_comment = 0;
                }
            }
        }
while ((current_port = get_port_name()) != NULL)
        {
        if (! port_decl_started)
            {
            begin_port_decl();
            port_decl_started = 1;
            }
        if (strcmp(current_port, "comment_interupt") == 0)
            {
            last_was_comment = 1;
            }
        else
            {
            if (! last_was_comment)
                {
                if (first_port)
                    {
                    output_port(current_port, out_mode,
                                                type);
                    first_port = FALSE;
                    }
                else
                    {
                    printf(";\n        ");
                    output_port(current_port, out_mode,
                                                type);
                    }
                }
            else
                {
                output_port(current_port, out_mode,
                                                type);
                last_was_comment = 0;
                }
            }
        }
}
if (port_decl_started && ! last_was_comment)
```

```
        {
        end_port_decl();
        }
    if (last_was_comment)
        {
        printf("\n");
        }
    end_entity_decl(entity_name);
}
```

alt_equa.c

```
/*************************************************************
*                                                            *
*                        alt_equa.c                          *
*                                                            *
*************************************************************
*
*  Module:     alt_equa.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedures and
*         functions for manipulating the tokens
*         data structure of the adftovhd driver
*         program.  Specifically, this module is
*         concerned with the EQUATIONS section.
*/


#include <alt_equa.h>

/*************************************************************
*
*  Function:   is_delimiter_semi
*
*  Interface:  int is_delimiter_semi()
*
*  Purpose:     This function returns true if the current
*         token in the token data structure is a
*         semi-colon delimiter.
*
*************************************************************/

int is_delimiter_semi()
{
    return(strcmp(top_token(), ";") == 0);
}

/*************************************************************
*
*  Function:   terminate_signal_assignment
```

```
*
*  Interface:  void terminate_signal_assignment()
*
*  Purpose:     This procedure outputs the termination
*         string for a signal_assignment statement.
*
*********************************************************/

void terminate_signal_assignment()
{
     char *string_buffer;

     string_buffer =(char *)malloc(MAX_BUFFER_SIZE *
                    sizeof(char));
     sprintf(string_buffer, " after %s", DELAY);
     output_string(string_buffer);
     output_string(";");

}


/*********************************************************
*
*  Function:   output_string
*
*  Interface:  void output_string(char *the_string)
*
*  Purpose:     This procedure outputs "the_string" which
*         is a portion of a signal assignment
*         statement.
*
*********************************************************/

void output_string(char *the_string)
{
            int i, no_space = FALSE;
     static int line_length = 0, indent = 0, new_signal =
                                                TRUE,
             last_was_left_paren = FALSE;


     if (strpbrk(the_string, "\t\f\n\r\v") == NULL &&
         strcmp(the_string, " ") != 0)
     {
     if (last_was_left_paren ||
         strcmp(the_string, ")") == 0)
          {
          no_space = TRUE;
          }
     if (strcmp(the_string, "(") == 0)
          {
          last_was_left_paren = TRUE;
          }
```

```
        else
             {
             last_was_left_paren = FALSE;
             }
        if (new_signal)
             {
             indent = strlen(the_string) + 4;
             }
        if (strcmp(the_string, ";") == 0)
             {
             printf(";\n\n");
             new_signal = TRUE;
             line_length = 0;
             no_space = FALSE;
             }
        else
             {
             if ((strlen(the_string) + line_length + 3) > 78)
                  {
                  printf("\n");
                  for(i = 0; i < indent; i++)
                       {
                       printf(" ");
                       }
                  printf("%s", the_string);
                  new_signal = FALSE;
                  line_length =  strlen(the_string) + indent;
                  }
             else
                  {
                  if (no_space || new_signal)
                       {
                       printf("%s", the_string);
                       new_signal = FALSE;
                       line_length = line_length +
                                 strlen(the_string);
                       }
                  else
                       {
                       printf(" %s", the_string);
                       new_signal = FALSE;
                       line_length = line_length +
                                 strlen(the_string) + 1;
                       }
                  }
             no_space = FALSE;
             }
        }
}

/*****************************************************************
*
*  Function:   output_signal_assignment_symbol
```

```
*
*  Interface:  void output_signal_assignment_symbol()
*
*  Purpose:     This procedure outputs "<=" which is a
*         signal assignment symbol.
*
*************************************************************/

void output_signal_assignment_symbol()
{
     output_string("<=");
}


/*************************************************************
*
*  Function:   output_identifier
*
*  Interface:  void output_identifier()
*
*  Purpose:     This procedure outputs the current token
*         preceded by "not" if the token is
*         followed by a "'".
*
*************************************************************/

void output_identifier()
{
     if (strcmp(next_token(), "'") == 0)
          {
          output_string("not");
          output_string(top_token());
          advance_to_next_token();
          }
     else
          {
          output_string(top_token());
          }
}


/*************************************************************
*
*  Function:   output_boolean_operator
*
*  Interface:  void output_boolean_operator()
*
*  Purpose:     This procedure outputs the VHDL boolean
*         operator for a given Altera boolean
*         operator.
*
*************************************************************/

void output_boolean_operator()
```

```
{
     if (strcmp(top_token(), "/")  == 0 ||
         strcmp(top_token(), "!")  == 0)
          {
          output_string("not");
          }
     else if (strcmp(top_token(), "*") == 0 ||
           strcmp(top_token(), "&") == 0)
          {
          output_string("and");
          }
     else if (strcmp(top_token(), "+") == 0 ||
           strcmp(top_token(), "#") == 0)
          {
          output_string("or");
          }
     else
          {
          printf("ERROR in output boolean operator\n");
          }
}


/*************************************************************
*
*  Function:   output_identifier_variable
*
*  Interface:  void output_identifier_variable()
*
*  Purpose:     This procedure outputs the variable
*         identifier names which need to be preceded
*         by "not".
*
*************************************************************/

void output_identifier_variable()
{
     int string_length = strlen(top_token());
     char *string_buffer, *str_ptr;

     string_buffer = (char *)malloc(MAX_BUFFER_SIZE *
                     sizeof(char));

     str_ptr = strdup(top_token());
     if (*str_ptr == '/')
          {
          sprintf(string_buffer, "not %s", top_token() + 1);
          output_string(string_buffer);
          }
     if (string_length > 1)
          {
          if (*(str_ptr + string_length - 1) == '\'')
               {
```

```
                *(str_ptr + string_length -1) = '\0';
                sprintf(string_buffer, "not %s",
                top_token());
                output_string(string_buffer);
                }
            }
}


/*********************************************************
*
*  Function:   is_boolean_operator
*
*  Interface:  int is_boolean_operator()
*
*  Purpose:     This function returns true if the top token
*               is a boolean operator.
*
*********************************************************/

int is_boolean_operator()
{
     if (strcmp(top_token(), "/") == 0 ||
         strcmp(top_token(), "!") == 0 ||
         strcmp(top_token(), "*") == 0 ||
         strcmp(top_token(), "&") == 0 ||
         strcmp(top_token(), "+") == 0 ||
         strcmp(top_token(), "#") == 0)
          {
          return(TRUE);
          }
     else
          {
          return(FALSE);
          }

}


/*********************************************************
*
*  Function:  generate_signal_assignment_statements
*
*  Interface: void generate_signal_assignment_statements()
*
*  Purpose:   This procedure generates signal assignment
*             statements by scanning the tokens for signals
*             and outputting the VHDL code to represent
*             the signal.
*
*********************************************************/


void generate_signal_assignment_statements()
```

```
{
     if (set_to_equation_section())
          {
          advance_to_next_token();
          while (strcmp(top_token(), "END$") != 0)
               {
               if (is_delimiter_comment())
                    {
                    output_comment_statement();
                    }
               else if (is_identifier_variable())
                    {
                    output_identifier();
                    }
               else if (is_boolean_operator())
                    {
                    output_boolean_operator();
                    }
               else if (is_delimiter_equal())
                    {
                    output_signal_assignment_symbol();
                    }
               else if (is_delimiter_semi())
                    {
                    terminate_signal_assignment();
                    }
               else
                    {
                    output_string(top_token());
                    }
               advance_to_next_token();
               }
          }
     else
          {
          printf("ERROR in generate signal assignment
                                        tatement\n");
          }

}
```

name_mod.c

```
/*************************************************************
*                                                            *
*                        name_mod.c                          *
*                                                            *
*************************************************************
*
*  Module:      name_mod.c
*
*  Version:     1.0
*
*  Purpose:      This module contains procedures and
```

```
*         functions for manipulating the tokens
*         data structure of the adftovhd driver
*         program.  Specifically, this module is
*         concerned with modifying the identifiers
*         of the Altera design files to legal VHDL
*         identifier names.
*/


#include <name_mod.h>

/*************************************************************
*
*  Function:   concat_strings
*
*  Interface:  char *concat_strings(char *prefix_string,
*                        char *suffix_string)
*
*  Purpose:     This function returns a pointer to a string
*         which is the result of concatenating
*         "suffix_string" to "prefix_string".
*
*************************************************************/

char *concat_strings(char *prefix_string, char
                                        *suffix_string)
{
          int i, first_index = strlen(prefix_string),
              second_index = strlen(suffix_string);
          char *temp_str;

          temp_str = (char *)malloc((strlen(prefix_string) +
                              strlen(suffix_string) + 1)
                                        * sizeof(char));
          for (i = 0; i < first_index; i++)
               {
               *(temp_str + i) = *(prefix_string + i);
               }
          for (i = 0; i < second_index; i++)
               {
               *(temp_str + first_index + i) =
                              *(suffix_string + i);
               }
          *(temp_str + first_index + second_index) = '\0';
          return(temp_str);
}


/*************************************************************
*
*  Function:   advance_past_comment
*
*  Interface:  void advance_past_comment()
*
```

```
*  Purpose:     This procedure moves the current token to
*         the first token past a comment token.
*
*************************************************************/

void advance_past_comment()
{
     advance_to_next_token();
     while (! is_delimiter_comment())
          {
          advance_to_next_token();
          }
     advance_to_next_token();
}


/*************************************************************
*
*  Function:   search_and_change
*
*  Interface:  void search_and_change(char *old_name,
*                          char *new_name,
*                          char *input_pin_name)
*
*  Purpose:     This procedure substitues "new_name" for
*         "old_name" while not disturbing the original
*         "input_pin_name".   All of the tokens in the
*         token data structure are checked.
*
*************************************************************/

void search_and_change(char *old_name, char *new_name, char
*input_pin_name)
{
     int old_token_index;

     old_token_index = get_token_index();
     advance_to_next_token();
     if (set_to_network_section())
          {
          while (! end_of_tokens())
               {
               if (strcmp(top_token(), old_name) == 0 &&
                   top_token() != input_pin_name)
                    {
                    strcpy(top_token(), new_name);
                    }
               advance_to_next_token();
               }
          set_token_index(old_token_index);
          }
}
```

```
/*********************************************************
*
*  Function:   modify_name
*
*  Interface:  void modify_name(char *node_name,
*                     char *input_pin_name)
*
*  Purpose:     This procedure adds "mod" to the "node_name"
*         and calls search_and_replace to modify all
*         tokens of the same name.
*
*********************************************************/

void modify_name(char *node_name, char *input_pin_name)
{
     char  *old_name;

     old_name = strdup(node_name);
     node_name = concat_strings(node_name, "mod");
     search_and_change(old_name, node_name, input_pin_name);
     free(old_name);
}


/*********************************************************
*
*  Function:   is_making_legal_vhdl_name
*
*  Interface:  char *is_making_legal_vhdl_name(
*                     char *token_name)
*
*  Purpose:     This procedure checks to make sure that
*         "token_name" starts with an alpha character
*         and contains no illegal characters.  The
*         character "v" is substituted for any
*         illegal characters.  A pointer to the token
*         name is returned.
*
*********************************************************/

char *is_making_legal_vhdl_name(char *token_name)
{
     char *current_position, *first_character = " ",
          *alpha_string = "alpha_";

     int i, first_index = strlen(alpha_string),
            second_index = strlen(token_name);

     while ((current_position =
          strpbrk(token_name,
             "!@&#*{}[]|\\/\?.,<>;\'\"+-_.~$:^")) != NULL)
          {
          *current_position = REPLACEMENT_CHARACTER;
          }
```

```
     *first_character = *token_name;
     if ((current_position = strpbrk(first_character,
                                     "0123456789") != NULL))
          {
          return(concat_strings(alpha_string, token_name));
          }
     return(token_name);
}


/*********************************************************
*
*  Function:   remove_illegal_vhdl_name
*
*  Interface:  void remove_illegal_vhdl_name_characters()
*
*  Purpose:     This procedure checks each token to make
*          sure it contains not illegal characters.
*
*********************************************************/

void remove_illegal_vhdl_name_characters()
{
     char *temp_ptr, *current_token;

     if (set_to_inputs_section())
          {
          advance_to_next_token();
          while (! end_of_tokens())
               {
          if (is_delimiter_comment())
                    {
                    advance_past_comment();
                    }
               if (is_identifier_variable())
                    {
                    current_token = top_token();
                    temp_ptr = strdup(current_token);
                    strcpy(current_token,
                      is_making_legal_vhdl_name(temp_ptr));
                    free(temp_ptr);
                    }
               advance_to_next_token();
               }
          }
     else
          {
          screen_message("ERROR in remove illegal vhdl name
                                             characters");
          }
}


/*********************************************************
```

```
*
*  Function:   check_and_change_identifiers
*
*  Interface:  void check_and_change_identifiers()
*
*  Purpose:     This procedure checks each token to make
*         sure it is a legal vhdle name.
*
*************************************************************/

void check_and_change_identifiers()
{
     char *last_identifier, *input_pin_name;

     if (set_to_network_section())
          {
          advance_to_next_token();
          while (! is_equation_section())
               {
               if (is_identifier_variable())
                    {
                    last_identifier = top_token();
                    }
               if (strcmp(top_token(), "INP") == 0)
                    {
                    input_pin_name = get_input_pin_name();
                    if (strcmp(input_pin_name,
                                        last_identifier)
                              == 0)
                         {
                         modify_name(last_identifier,
                              input_pin_name);
                         }
                    }
               advance_to_next_token();
               }
          }
}
```

alt_inst.c

```
/*************************************************************
*                                                            *
*                        alt_inst.c                          *
*                                                            *
*************************************************************
*
*  Module:     alt_inst.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedures and
```

```
*         functions for manipulating the tokens
*         data structure of the adftovhd driver
*         program.  Specifically, this module is
*         concerned with generating VHDL component
*         instantiations from Altera design files.
*/


#include <alt_inst.h>


/*********************************************************
*
*  Function:   is_equation_section
*
*  Interface:  int is_equation_section()
*
*  Purpose:     This function returns true if the current
*         token in the token data structure is
*         EQUATIONS.
*
*********************************************************/

int is_equation_section()
{
     return(strcmp(top_token(), "EQUATIONS:") == 0);
}


/*********************************************************
*
*  Function:   is_delimiter_left_paren
*
*  Interface:  int is_delimiter_left_paren()
*
*  Purpose:     This function returns true if the current
*         token in the token data structure is
*         a left parenthesis.
*
*********************************************************/

int is_delimiter_left_paren()
{
     return(strcmp(top_token(), "(") == 0);
}

/*********************************************************
*
*  Function:   is_delimiter_right_paren
*
*  Interface:  int is_delimiter_right_paren()
*
*  Purpose:     This function returns true if the current
*         token in the token data structure is
```

```
*         a right parenthesis.
*
*************************************************************/

int is_delimiter_right_paren()
{
     return(strcmp(top_token(), ")") == 0);
}


/*************************************************************
*
*  Function:   output_comment_statement
*
*  Interface:  void output_comment_statement()
*
*  Purpose:     This procedure outputs a comment string.
*
*************************************************************/

void output_comment_statement()
{
     advance_to_next_token();
     while (! is_delimiter_comment())
          {
          make_comment(top_token(), TRUE);
          advance_to_next_token();
          }
          make_comment(top_token(), FALSE);
}

/*************************************************************
*
*  Function:   output_instantiation_close
*
*  Interface:  void output_comment_statement()
*
*  Purpose:     This procedure outputs a comment string.
*
*************************************************************/

void output_instantation_close()
{
     printf(");\n\n");
}

/*************************************************************
*
*  Function:   output_association
*
*  Interface:  void output_association(char *local,
*                        char *actual,
*                        int first_association)
*
```

```
*  Purpose:     This procedure outputs an association
*         between a "local" and "actual" and sets
*         a flag "first_association" if it is the
*         first association.
*
*********************************************************/

void output_association(char *local, char *actual, int
first_association)
{
     static int line_length = 0;

     if (first_association)
          {
          line_length = 0;
          }
     if ( strlen(local) + strlen (actual) + line_length + 7
                                                        > 65)
          {
          printf(",\n              %s => %s", local,
                                                  actual);
          line_length =  strlen(local) + strlen(actual) + 5;
          }
     else
          {
          if (first_association)
               {
               printf("%s => %s", local, actual);
               first_association = FALSE;
               line_length = line_length + strlen(local) +
                         strlen(actual) + 4;
               }
          else
               {
               printf(", %s => %s", local, actual);
               line_length = line_length + strlen(local) +
                         strlen(actual) + 6;
               }
          }
}

/*********************************************************
*
*  Function:   output_instantiation_header
*
*  Interface:  void output_instantiation(
*                         char *component_mark)
*
*  Purpose:     This procedure outputs an instantiation
*         header that includes the "component_mark".
*
*********************************************************/

void output_instantiation_header(char *component_mark)
```

```
{
     static int label_identifier_number = FIRST;
     char *label_id;

     label_id = (char *)malloc(MAX_NAME_LEN * sizeof(char));
     if (sprintf(label_id, "%c%d", 'U',
                         label_identifier_number) <= 0)
          {
          printf("ERROR in output instantiation header\n");
          }
     printf("  %s : %s\n    port map (", label_id,
                                        component_mark);
     label_identifier_number++;
}

/*********************************************************
*
*  Function:   output_component_instantiations
*
*  Interface:  void output_component_instantiations(
*                   char *component_inputs_outputs,
*                   char *component_mark)
*
*  Purpose:     This procedure outputs generates the
*          component instantiations from the tokens
*          data structure.
*
*********************************************************/

void output_component_instantiations(char
**component_inputs_outputs,
                              char *component_mark)
{
     int port_index = FIRST, first_association = TRUE, index
                                                  = FIRST;
     char *local_value, *default_value, *actual_value,
          *dummy_value, *port_list;


     port_list = strdup(get_ports_for(component_mark));
     local_value = strtok(port_list, " ");
     dummy_value = strtok(NULL, " ");
     default_value = strtok(NULL, " ");
     output_instantiation_header(component_mark);

     while (local_value != NULL)
          {
          if (strcmp(component_inputs_outputs[port_index],
                                                  "open")
                                             == 0)
               {
               if (strcmp(default_value, "ndf") != 0)
                    {
                    actual_value = default_value;
```

```
                    }
                else
                    {
                    actual_value =
                        component_inputs_outputs[port_index];
                    }
                }
            else
                {
                actual_value =
                        component_inputs_outputs[port_index];
                }
            port_index++;
            output_association(local_value, actual_value,
                                    first_association);
            first_association = FALSE;
            local_value = strtok(NULL, " ");
            dummy_value = strtok(NULL, " ");
            default_value = strtok(NULL, " ");
            }
        free(port_list);
        output_instantiation_close();
}


/*************************************************************
*
*  Function:   collect_inputs_and_append
*
*  Interface:  char **collect_inputs_and_append(
*                  char **outputs)
*
*  Purpose:     This function returns a pointer to an array
*          of pointers to the inputs for a component
*          and also contains the appended outputs.
*
*************************************************************/


char **collect_inputs_and_append(char **outputs)
{
    int index = FIRST, comma_count = 0, input_found =
                                                FALSE,
        output_index = FIRST;
    char **component_inputs_and_outputs, *open_input =
                                                "open";

    advance_to_next_token();
    while (! is_delimiter_left_paren())
        {
        if (is_delimiter_comment())
            {
            output_comment_statement();
            }
```

```
            advance_to_next_token();
            }

        comma_count = count_commas(")");
        component_inputs_and_outputs =
            get_token_array(comma_count +
                            MAX_NUM_COMPONENT_OUTPUTS + 2);
        advance_to_next_token();
        while (! is_delimiter_right_paren())
            {
            if (is_delimiter_comment())
                {
                output_comment_statement();
                }
            else  is_identifier_variable())

                component_inputs_and_outputs[index] =
                                            top_token();
                input_found = TRUE;
                index++;
                }
            else if (is_delimiter_comma())
                {
                if (! input_found)
                    {
                    component_inputs_and_outputs[index] =
                                        open_input;
                    index++;
                    }
                else
                    {
                    input_found = FALSE;
                    }
                }
            advance_to_next_token();
            }
        if (! input_found)
            {
            component_inputs_and_outputs[index] = open_input;
            index++;
            }
        while (outputs[output_index] != NULL)
            {
            component_inputs_and_outputs[index] =
                                outputs[output_index];
            output_index++;
            index++;
            }
        index = FIRST;
        return(component_inputs_and_outputs);
    }

    /*****************************************************
    *
```

```
*  Function:   determine_component_mark
*
*  Interface:  char *determine_component_mark()
*
*  Purpose:     This function returns a pointer to a string
*         that a component mark.
*
*************************************************************/

char *determine_component_mark()
{
     advance_to_next_token();
     while (! is_component(top_token()) && !
                                        end_of_tokens())
          {
          if (is_delimiter_comment())
               {
               output_comment_statement();
               }
          advance_to_next_token();
          }
     if (end_of_tokens())
          {
          printf("ERROR in determine component mark\n");
          }
     else
          {
          return(top_token());
          }
}

/*************************************************************
*
*  Function:   collect_component_outputs
*
*  Interface:  char **collect_component_outputs()
*
*  Purpose:     This function returns a pointer to an array
*         of pointers to the outputs to a component.
*
*************************************************************/

char **collect_component_outputs()
{
     int index = FIRST, end_of_input = FALSE;
     char **outputs_array;

     outputs_array =
          get_token_array(MAX_NUM_COMPONENT_OUTPUTS + 2);
     advance_to_next_token();
     while (! is_delimiter_equal())
          {
          if (is_equation_section())
               {
```

```
                return(NULL);
                }
            if (is_delimiter_comment())
                {
                output_comment_statement();
                }
            else if (is_identifier_variable())
                {
                outputs_array[index] = top_token();
                index++;
                }
            advance_to_next_token();
            }
            return(outputs_array);
}

/*************************************************************
*
*  Function:   generate_instantiations
*
*  Interface:  void generate_instantiations()
*
*  Purpose:     This procedure generates component
*          instantiations from the tokens.
*
*************************************************************/

void generate_instantiations()
{
     int  done_with_instantiations = FALSE;
     char **component_outputs,
          **component_inputs_and_outputs,
          *component_mark;

     if (set_to_network_section())
          {
          while (! done_with_instantiations)
               {
               if ((component_outputs =
                    collect_component_outputs()) != NULL)
                    {
                    component_mark =
                              determine_component_mark();
                    component_inputs_and_outputs =
                         collect_inputs_and_append(
                                        component_outputs);
                    output_component_instantiations(
                              component_inputs_and_outputs,
                              component_mark);
                    }
               else
                    {
                    done_with_instantiations = TRUE;
                    }
```

```
                }
            }
        else
            {
            printf("ERROR network section not present\n");
            }
}
```

altera_p.c

```
/*************************************************************
*                                                            *
*                        altera_p.c                          *
*                                                            *
*************************************************************
*
*  Module:     altera_p.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedures and
*          functions for manipulating the tokens
*          data structure of the adftovhd driver
*          program.  Specifically, this module is
*          concerned building the tokens structure.
*/

#include <tokens.h>

static tokens_struct tokens;

/*************************************************************
*
*  Function:   get_identifier_type
*
*  Interface:  char *get_identifier_type(
*                        char *token_entry)
*
*  Purpose:     This function returns a pointer to the
*          type of "token_entry".
*
*************************************************************/

char *get_identifier_type(char *token_entry)
{
     if (strcmp(token_entry, "EQUATIONS:") == 0 ||
         strcmp(token_entry, "INPUTS:")    == 0 ||
         strcmp(token_entry, "OUTPUTS:")   == 0 ||
         strcmp(token_entry, "NETWORK:")   == 0 ||
         strcmp(token_entry, "END$")       == 0 )
          {
          return("identifier_reserved");
```

```
        }
    else
        {
        return("name_type");
        }
}


/*********************************************************
*
*  Function:   advance_to_next_token
*
*  Interface:  void advance_to_next_token()
*
*  Purpose:     This procedure advances to the next token.
*
*********************************************************/

void advance_to_next_token()
{
    tokens.index++;
}


/*********************************************************
*
*  Function:   end_of_tokens
*
*  Interface:  int end_of_tokens()
*
*  Purpose:     This function returns true if the current
*         token is the last.
*
*********************************************************/

int end_of_tokens()
{
    return(tokens.index  >= tokens.total_entries ||
        strcmp(tokens.token_array[tokens.index], "END$")
                                        == 0);
}


/*********************************************************
*
*  Function:   get_token
*
*  Interface:  token *get_token()
*
*  Purpose:     This function returns a pointer to the
*         current token and advances to the next.
*
*********************************************************/
```

```
tokon *get_token()
{
     if (end_of_tokens())
          {
          return(NULL);
          }
     else
          {
          tokens.index = tokens.index + 1;
          return(tokens.token_array[tokens.index - 1]);
          }
}


/*************************************************************
*
*  Function:   next_token
*
*  Interface:  token *next_token()
*
*  Purpose:     This function advances to the next token.
*
*************************************************************/

tokon *next_token()
{
     if (tokens.index + 1 >= tokens.total_entries)
          {
          return('\0');
          }
     else
          {
          return(tokens.token_array[tokens.index + 1]);
          }
}


/*************************************************************
*
*  Function:   is_identifier_variable
*
*  Interface:  int is_identifier_variable()
*
*  Purpose:     This function returns true if the current
*          token is of the type "name_type".
*
*************************************************************/

int is_identifier_variable()
{
     return(strcmp(tokens.token_type_array[tokens.index],
                                        "name_type") == 0);
}
```

```
/*************************************************************
*
*  Function:   set_to_previous_token
*
*  Interface:  int set_to_previous_token()
*
*  Purpose:     This function sets the current token to the
*         previous token.
*
*************************************************************/

int set_to_previous_token()
{
     if (tokens.index < 1)
          {
          return(0);
          }
     else
          {
          tokens.index = tokens.index - 1;
          return(1);
          }
}


/*************************************************************
*
*  Function:   is_delimiter_comment
*
*  Interface:  int is_delimiter_comment()
*
*  Purpose:     This function returns true if the current
*         token is a comment.
*
*************************************************************/

int is_delimiter_comment()
{
     return(strcmp(tokens.token_type_array[tokens.index],
                    "delimiter_type_comment") == 0);
}


/*************************************************************
*
*  Function:   get_token_type
*
*  Interface:  char *get_token_type()
*
*  Purpose:     This function returns the type of the
*         current token.
*
```

```
*************************************************************/

char *get_token_type()
{
     return(tokens.token_type_array[tokens.index]);
}


/*************************************************************
*
*  Function:   top_token
*
*  Interface:  token *top_token()
*
*  Purpose:     This function returns a pointer to the
*          current token.
*
*************************************************************/

tokon *top_token()
{
     return(tokens.token_array[tokens.index]);
}


/*************************************************************
*
*  Function:   get_token_index
*
*  Interface:  int get_token_index()
*
*  Purpose:     This function returns the index to the
*          current token.
*
*************************************************************/
int get_token_index()
{
     return(tokens.index);
}


/*************************************************************
*
*  Function:   set_token_index
*
*  Interface:  void set_token_index(int index)
*
*  Purpose:     This procedure sets the index to "index".
*
*************************************************************/

void set_token_index(int index)
{
     tokens.index = index;
```

```
}


/*************************************************************
*
*  Function:   beginning_of_network_section
*
*  Interface:  int beginning_of_network_section()
*
*  Purpose:     This procedure returns true if the current
*         token is "NETWORK:".
*
*************************************************************/

int beginning_of_network_section()
{
     return(strcmp(tokens.token_array[tokens.index],
"NETWORK:") == 0);
}


/*************************************************************
*
*  Function:   get_delimiter_type
*
*  Interface:  char *get_delimiter_type(
*                        char *delimiter_name)
*
*  Purpose:     This procedure returns the type of
*         "delimiter_name".
*
*************************************************************/

char *get_delimiter_type(char *delimiter_name)
{
     if (strcmp(delimiter_name, "%") == 0)
          {
          return("delimiter_type_comment");
          }
     else if (strcmp(delimiter_name, "/") == 0)
          {
          return("delimiter_type_slash");
          }
     else if (strcmp(delimiter_name, "\\") == 0)
          {
          return("delimiter_type_back_slash");
          }
     else if (strcmp(delimiter_name, ",") == 0)
          {
          return("delimiter_type_comma");
          }
     else if (strcmp(delimiter_name, ";") == 0)
          {
```

```
            return("delimiter_type_semi_colon");
            }
        else if (strcmp(delimiter_name, ":") == 0)
            {
            return("delimiter_type_colon");
            }
        else if (strcmp(delimiter_name, "*") == 0)
            {
            return("delimiter_type_astric");
            }
        else if (strcmp(delimiter_name, "+") == 0)
            {
            return("delimiter_type_plus");
            }
        else if (strcmp(delimiter_name, "(") == 0)
            {
            return("delimiter_type_left_paren");
            }
        else if (strcmp(delimiter_name, ")") == 0)
            {
            return("delimiter_type_right_paren");
            }
        else if (strcmp(delimiter_name, "=") == 0)
            {
            return("delimiter_type_equal");
            }
        else if (strcmp(delimiter_name, "\n") == 0)
            {
            return("delimiter_type_lf");
            }
        else if (strcmp(delimiter_name, " ") == 0)
            {
            return("delimiter_type_space");
            }
        else
            {
            return("delimiter_type_unknown");
            }
}


/*************************************************************
*
*  Function:   build_tokens
*
*  Interface:  void build_tokens(FILE *input_stream)
*
*  Purpose:     This procedure builds the array of token
*          pointers from the input file.
*
*************************************************************/

void build_tokens(FILE *input_stream)
```

```
{

int i, ch, at_equation_section = FALSE,
     at_inputs_section = FALSE,
    word_started = FALSE, index = FIRST;
long filesize;
char *token_entry, *new_ptr,
    *new_str, **token_arr, **token_type_arr;

screen_message("\r\nparsing input file....\r\n");
filesize = get_file_length(input_stream);
token_arr = get_token_array(filesize);
token_type_arr = get_token_array(filesize);
while ((ch = fgetc(input_stream)) != EOF)
     {
     if (is_delimiter(ch, at_equation_section))
          {
          if (word_started)
               {
               token_arr[index] =  token_entry;
               if (strcmp(token_entry, "INPUTS:") == 0)
                    {
                    at_inputs_section = TRUE;
                    }
               if (strcmp(token_entry, "EQUATIONS:") ==
                                                  0)
                    {
                    at_equation_section = TRUE;
                    }
               if (is_component(token_entry))
                    {
                    token_type_arr[index] =
                                   "function_type";
                    }
               else
                    {
                    token_type_arr[index] =
                    get_identifier_type(token_entry);
                    }
               index++;
               word_started = FALSE;
               }
          new_ptr = get_new_ptr(2);
          *new_ptr = ch;
          *(new_ptr +1) = '\0';
          token_arr[index] = new_ptr;
          token_type_arr[index] =
               get_delimiter_type(new_ptr);
          index++;
          }

     else
          {
           if (! word_started)
```

```
                    {
                    token_entry = get_new_ptr(1);
                    *token_entry = '\0';
                    word_started = 1;
                    }
               new_str = get_new_ptr(2);
               *new_str = ch;
               *(new_str + 1) = '\0';
               token_entry = append_to_token (token_entry,
                                                  new_str);
               }
     }
     if (word_started)
          {
          token_arr[index] = token_entry;
          token_type_arr[index] = "terminator_type";
          index++;
          word_started = 0;
          }

     fclose(input_stream);
     tokens.token_array = token_arr;
     tokens.token_type_array = token_type_arr;
     tokens.index = 0;
     tokens.total_entries = index + 1;
     remove_illegal_vhdl_name_characters();
     check_and_change_identifiers();


}
```

ent_arch.c

```
/*************************************************************
*                                                            *
*                        ent_arch.c                          *
*                                                            *
*************************************************************
*
*  Module:     ent_arch.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedures and
*         functions for manipulating the tokens
*         data structure of the adftovhd driver
*         program.  Specifically, this module is
*         concerned with building the entity
*         architecture.
*/

#include <ent_arch.h>
```

```
/*********************************************************
*
*  Function:   get_ports_for
*
*  Interface:  char *get_ports_for(char *component_name)
*
*  Purpose:     This function returns a pointer to a list
*         of the ports for "component_name".
*
*********************************************************/


char *get_ports_for(char *component_name)
{

     if (strcmp(component_name, "INP") == 0)
          {
          return("In1 in ndf Out1 out ndf ");
          }
     else if (strcmp(component_name, "CLKB") == 0)
          {
          return("In1 in ndf Out1 out ndf ");
          }
     else if (strcmp(component_name, "CONF") == 0)
          {
          return("In1 in ndf Oe in VCC Out1 out ndf ");
          }
     else if (strcmp(component_name, "COIF") == 0)
          {
          return("In1 in ndf Oe in VCC Out1 out ndf Fbk out
                                                  ndf ");
          }
     else if (strcmp(component_name, "NORF") == 0)
          {
          return("In1 in ndf Clk in ndf C in GND P in GND
                                          Fbk out ndf ");
          }
     else if (strcmp(component_name, "NOJF") == 0)
          {
          return("Jn in ndf Clk in ndf Kin in ndf C in GND P
                                 in GND Fbk out ndf ");
          }
     else if (strcmp(component_name, "NOCF") == 0)
          {
          return("In1 in ndf Fbk1 out ");
          }
     else if (strcmp(component_name, "RONF") == 0)
          {
          return("In1 in ndf Clk in ndf C in GND P in GND
                                       Out1 out ndf ");
          }

     else if (strcmp(component_name, "AND2")  == 0 ||
```

```
    strcmp(component_name, "NAND2")       == 0 ||
    strcmp(component_name, "OR2")         == 0 ||
    strcmp(component_name, "NOR2")        == 0)
     {
     return("In1 in ndf In2 in ndf Out1 out ndf ");
     }
else if (strcmp(component_name, "AND3")   == 0 ||
    strcmp(component_name, "NAND3")       == 0 ||
    strcmp(component_name, "OR3")         == 0 ||
    strcmp(component_name, "NOR3")        == 0)
     {
     return("In1 in ndf In2 in ndf In3 in ndf Out1 out
                                                ndf ");
     }
else if (strcmp(component_name, "AND4")   == 0 ||
    strcmp(component_name, "NAND4")       == 0 ||
    strcmp(component_name, "OR4")         == 0 ||
    strcmp(component_name, "NOR4")        == 0)
     {
     return("In1 in ndf In2 in ndf In3 in ndf In4 in
                              ndf Out1 out ndf ");
     }
else if (strcmp(component_name, "AND6")   == 0 ||
    strcmp(component_name, "NAND6")       == 0 ||
    strcmp(component_name, "OR6")         == 0 ||
    strcmp(component_name, "NOR6")        == 0)
     {
     return("In1 in ndf In2 in ndf In3 in ndf In4 in
          ndf In5 in ndf In6 in ndf Out1 out ndf ");
     }
else if (strcmp(component_name, "AND8")   == 0 ||
    strcmp(component_name, "NAND8")       == 0 ||
    strcmp(component_name, "OR8")         == 0 ||
    strcmp(component_name, "NOR8")        == 0)
     {
     return("In1 in ndf In2 in ndf In3 in ndf In4 in
          ndf In5 in ndf In6 in ndf In7 in ndf In8 in
               ndf Out1 out ndf ");
     }
else if (strcmp(component_name, "AND12")   == 0 ||
    strcmp(component_name, "NAND12")       == 0 ||
    strcmp(component_name, "OR12")         == 0 ||
    strcmp(component_name, "NOR12")        == 0)
     {
     return("In1 in ndf In2 in ndf In3 in ndf In4 in
          ndf In5 in ndf In6 in ndf In7 in ndf In8 in
               ndf In9 in ndf In10 in ndf In11 in ndf
                    In12 in ndf Out1 out ndf ");
     }
else
     {
     printf("ERROR get ports for\n");
     return(NULL);
     }
```

```
}

/*************************************************************
*
*  Function:   output_decl
*
*  Interface:  void output_decl(char *name, char *mode,
*                   char *type, int new_component)
*
*  Purpose:     This procedure outputs a component
*         declaration.
*
*************************************************************/


void output_decl(char *name, char *mode, char *type, int
new_component)
{
     static int line_length = 0, first_signal;

     first_signal = new_component;
     if (first_signal)
          {
          line_length = 0;
          }
     if ( strlen(name) + strlen (mode) + strlen(type) +
                              line_length + 6 > 65)
          {
          printf(";\n          %s : %s %s", name, mode,
                                                  type);
          line_length =  strlen(name) + strlen(mode) +
                                  strlen(type) + 5;
          }
     else
          {
          if (first_signal)
               {
               printf("%s : %s %s", name, mode, type);
               first_signal = FALSE;
               line_length = line_length + strlen(name) +
                         strlen(mode) + strlen(type) + 3;
               }
          else
               {
               printf("; %s : %s %s", name, mode, type);
               line_length = line_length + strlen(name) +
                         strlen(mode) + strlen(type) + 5;
               }
          }
}

/*************************************************************
*
*  Function:   output_component_decl_header
```

```
*
*  Interface:  void output_component_decl_header(
*                              char *component_name)
*
*  Purpose:     This procedure outputs a component
*         declaration header.
*
*************************************************************/

void output_component_decl_header(char *component_name)
{
     printf("  component %s\n    port (", component_name);
}

/*************************************************************
*
*  Function:   output_close_component_decl
*
*  Interface:  void output_close_component_decl()
*
*  Purpose:     This procedure outputs a component
*         declaration end.
*
*************************************************************/

void output_close_component_decl()
{
     printf(");\n  end component;\n\n");
}

/*************************************************************
*
*  Function:   output_component_ports
*
*  Interface:  void output_component_ports(char *ports)
*
*  Purpose:     This procedure outputs a component ports
*         from the list "ports".
*
*************************************************************/

void output_component_ports(char *ports)
{
     int new_component;
     char  *port_name, *port_mode, *dummy, *ports_list;

     new_component = TRUE;
     ports_list = strdup(ports);
     port_name = strtok(ports_list, " ");
     port_mode = strtok(NULL, " ");
     dummy     = strtok(NULL, " ");
     while (port_name != NULL)
          {
```

```
            output_decl(port_name, port_mode, ALTERA_TYPE,
                                        new_component);
            new_component = FALSE;
            port_name = strtok(NULL, " ");
            port_mode = strtok(NULL, " ");
            dummy       = strtok(NULL, " ");
            }
        free(ports_list);
}


/*********************************************************
*
*  Function:   output_component_decl
*
*  Interface:  void output_component_ports(char *ports)
*
*  Purpose:     This procedure generates the output of
*           a component declaration.
*
*********************************************************/


void output_component_decl(char *component_name)
{
        char *ports;

        ports = get_ports_for (component_name);
        output_component_decl_header(component_name);
        output_component_ports(ports);
        output_close_component_decl();

}
/*********************************************************
*
*  Function:   set_to_inputs_section
*
*  Interface:  int set_to_inputs_section()
*
*  Purpose:     This function returns true if the current
*           token can be set to the inputs section.
*
*********************************************************/


int set_to_inputs_section()
{
        int index = FIRST;

        set_token_index(FIRST);
        while (! end_of_tokens())
            {
            if (strcmp(top_token(), "INPUTS:") == 0)
                {
```

```
                return(TRUE);
                }
            index++;
            set_token_index(index);
            }
        return(FALSE);
}
/*************************************************************
*
*  Function:   set_to_network_section
*
*  Interface:  int set_to_network_section()
*
*  Purpose:     This function returns true if the current
*          token can be set to the network section.
*
*************************************************************/


int set_to_network_section()
{
        int index = FIRST;

        set_token_index(FIRST);
        while (! end_of_tokens())
            {
            if (strcmp(top_token(), "NETWORK:") == 0)
                {
                return(TRUE);
                }
            index++;
            set_token_index(index);
            }
        return(FALSE);
}

/*************************************************************
*
*  Function:   count_commas
*
*  Interface:  int count_commas(char *stop_point)
*
*  Purpose:     This function returns the number of commas
*          between the current token and "stop_point".
*
*************************************************************/

int count_commas(char *stop_point)
{
        int comma_count = 0, old_index;

        old_index = get_token_index();
        advance_to_next_token();
        while (strcmp(top_token(), stop_point) != 0)
```

```
            {
            if (is_delimiter_comma())
                 {
                 comma_count++;
                 }
            advance_to_next_token();
            }
       set_token_index(old_index);
       return((comma_count));
}

/*********************************************************
*
*  Function:   set_to_left_paren
*
*  Interface:  int set_to_left_paren()
*
*  Purpose:     This function returns true when the current
*          token can be set to a left parenthesis.
*
*********************************************************/


int set_to_left_paren()
{
       while (! end_of_tokens())
            {
            if (strcmp(top_token(), "(") == 0)
                 {
                 return(TRUE);
                 }
            else
                 {
                 advance_to_next_token();
                 }
            }
       return(FALSE);
}

/*********************************************************
*
*  Function:   get_parameter_count
*
*  Interface:  int get_parameter_count()
*
*  Purpose:     This function returns the number of
*          parameters found by counting commas.
*
*********************************************************/

int get_parameter_count()
{
       int comma_count;
```

```
     if (set_to_left_paren())
          {
          comma_count = count_commas(
          ")");
          }
     return(comma_count + 1);
}

/*********************************************************
*
*  Function:   append_parameter_number_if_necessary
*
*  Interface:  void append_parameter_number_if_necessary(
*                              char *component_name)
*
*  Purpose:     This procedure will append the number of
*         inputs to a component name.
*
*********************************************************/

void append_parameter_number_if_necessary(char
*component_name)
{
     int parameter_count;
     char *temp_name, *temp_str, *parameter_count_string;

     if (strcmp(component_name, "AND")   == 0 ||
         strcmp(component_name, "NAND")  == 0 ||
         strcmp(component_name, "OR")    == 0 ||
         strcmp(component_name, "NOR")   == 0 )
          {
          parameter_count = get_parameter_count();
          temp_str = (char *)malloc(STANDARD_STR_LEN *
                                        sizeof(char));
          parameter_count_string = itoa(parameter_count,
                                        temp_str, 10);
          temp_name = strcat(component_name,
                              parameter_count_string);
          component_name = temp_name;
          }
     return(component_name);
}


/*********************************************************
*
*  Function:   generate_component_decls
*
*  Interface:  void generate_component_decls()
*
*  Purpose:     This procedure generates all the component
*         declarations.
*
*********************************************************/
```

```
void generate_component_decls()
{
     int index = 0, adding_to_list = TRUE;
     char **component_list;
     char *temp_token;

     component_list = get_token_array(MAX_NUM_FUNCTIONS);
     if (set_to_network_section())
          {
          temp_token = get_token();
          while (! end_of_tokens())
               {
               if (is_component(temp_token))
                    {
                    append_parameter_number_if_necessary(
                                                  temp_token);
                    index = 0;
                    adding_to_list = TRUE;
                    while (adding_to_list)
                         {
                         if (component_list[index] == NULL)
                              {
                              component_list[index] =
                                                  temp_token;
                              adding_to_list = FALSE;
                              }
                         else if
                              (strcmp(component_list[index],
                                   temp_token) == 0)
                                   {
                                   adding_to_list = FALSE;
                                   }
                         else
                              {
                              if (index < MAX_NUM_FUNCTIONS)
                                   {
                                   index++;
                                   }
                              else
                                   {
                                   printf("ERROR in
                                        gen_comp_decls\n");
                                   }
                              }
                         }
                    }
               temp_token = get_token();
               }
          index = 0;
          while (component_list[index] != NULL)
               {
               output_component_decl(component_list[index]);
               index++;
```

```
            }
        printf("\n");
        }
}


/*********************************************************
*
*  Function:   output_signal
*
*  Interface:  void output_signal(char *signal_name,
*                          char *type)
*
*  Purpose:     This procedure outputs a signal string.
*
*********************************************************/

void output_signal(char *signal_name, char *type)
{
     printf("%s : %s", signal_name, type);
}

/*********************************************************
*
*  Function:   end_the_signal_decl
*
*  Interface:  void end_the_signal_decl()
*
*  Purpose:     This procedure outputs an end string for a
*         signal declaration.
*
*********************************************************/


void end_the_signal_decl()
{
     printf(";\n  ");
}

/*********************************************************
*
*  Function:   end_the_signal_decls
*
*  Interface:  void end_the_signal_decls()
*
*  Purpose:     This procedure outputs an end string for a
*         all signal declarations.
*
*********************************************************/

void end_signal_decls()
{
     printf("\n\n");
}
```

```
/*********************************************************
*
*  Function:   output_signal_header
*
*  Interface:  void output_signal_header()
*
*  Purpose:     This procedure outputs a header string for
*          a signal declaration.
*
*********************************************************/

void output_signal_header()
{
     printf("signal ");
}

/*********************************************************
*
*  Function:   is_delimiter_equal
*
*  Interface:  int is_delimiter_equal()
*
*  Purpose:     This function returns true is the delimiter
*          is an equal operator.
*
*********************************************************/

int is_delimiter_equal()
{
     return(strcmp(top_token(), "=") == 0);
}

/*********************************************************
*
*  Function:   read_past_equation
*
*  Interface:  void read_past_equation()
*
*  Purpose:     This procedure advances the current token
*          past the equation semi-colon.
*
*********************************************************/

void read_past_equation()
{
     while (strcmp(top_token(), ";") != 0)
          {
          advance_to_next_token();
          }
     advance_to_next_token();
}

/*********************************************************
```

```
*
*  Function:   at_network_section
*
*  Interface:  int at_network_section()
*
*  Purpose:     This function returns true if the current
*          token is "NETWORK:".
*
*************************************************************/

int at_network_section()
{
     return(strcmp(top_token(), "NETWORK:") == 0);
}

/*************************************************************
*
*  Function:   is_delimiter_comma
*
*  Interface:  int is_delimiter_comma()
*
*  Purpose:     This function returns true if the current
*          token is a comma.
*
**************************************************************/

int is_delimiter_comma()
{
     return(strcmp(top_token(), ",") == 0);
}

/*************************************************************
*
*  Function:   set_to_input_section
*
*  Interface:  int set_to_input_section()
*
*  Purpose:     This function returns true when the current
*          token can be set to the inputs section.
*
*************************************************************/

int set_to_input_section()
{

     set_token_index(FIRST);
     while (strcmp(top_token(), "INPUTS:") != 0)
          {
          advance_to_next_token();
          }
}

/*************************************************************
*
```

```
*  Function:   get_list_of_io_pins
*
*  Interface:  char **get_list_of_io_pins()
*
*  Purpose:     This function returns a pointer to an array
*         of pointers to component io pins.
*
*************************************************************/

char **get_list_of_io_pins()
{
     int number_of_io_pins, io_index = FIRST;
     char **list_of_io_pins;

     set_to_input_section();
     number_of_io_pins = count_commas("NETWORK:") + 5;
     list_of_io_pins = get_token_array(number_of_io_pins);

     advance_to_next_token();
     while (! at_network_section())
          {
          if (is_delimiter_comment())
               {
               advance_past_comment();
               }
          else if (is_identifier_variable() && !
                         is_section_header(top_token()))
               {
               list_of_io_pins[io_index] = top_token();
               io_index++;
               }
          advance_to_next_token();
          }
     return(list_of_io_pins);
}

/*************************************************************
*
*  Function:   get_signal_name
*
*  Interface:  char *get_signal_name()
*
*  Purpose:     This function returns a pointer to signal
*         name found in the tokens.
*
*************************************************************/

char *get_signal_name()
{
     char *last_identifier;

     while (! end_of_tokens())
          {
          if (is_delimiter_comment())
```

```
                {
                advance_past_comment();
                }
            else if (is_identifier_variable())
                {
                last_identifier = top_token();
                }
            else if (is_delimiter_equal())
                {
                advance_to_next_token();
                return(last_identifier);
                }
            advance_to_next_token();
            }
            return(NULL);
}

/*********************************************************
*
*  Function:   set_to_equation_section
*
*  Interface:  int set_to_equation_section()
*
*  Purpose:     This function returns true when the current
*          token can be set to the equations section.
*
*********************************************************/

int set_to_equation_section()
{
     int index = FIRST;

     set_token_index(FIRST);
     while (! end_of_tokens())
          {
          if (strcmp(top_token(), "EQUATIONS:") == 0)
               {
               return(TRUE);
               }
          index++;
          set_token_index(index);
          }
     return(FALSE);
}

/*********************************************************
*
*  Function:   is_io_pin_name
*
*  Interface:  int is_io_pin_name(char *signal_name,
*              char **list_of_io_pin_names)
*
*  Purpose:     This function returns true when
*          "signal_name" is in the list of io pins.
```

```
*
*************************************************************/

int is_io_pin_name(char *signal_name, char
**list_of_io_pin_names)
{
     int io_index = FIRST;

     while (list_of_io_pin_names[io_index] != NULL)
          {
          if (strcmp(list_of_io_pin_names[io_index],
                                   signal_name) == 0)
               {
               return(TRUE);
               }
          else
               {
               io_index++;
               }
          }
     return(FALSE);
}

/*************************************************************
*
*  Function:   generate_signal_decls
*
*  Interface:  void generate_signal_decls()
*
*  Purpose:     This procedure generates all the signal
*          declarations from the tokens.
*
*************************************************************/

void generate_signal_decls()
{
     int  signal_header_printed = 0, io_index;
     char *signal_name, **list_of_io_pin_names;

     list_of_io_pin_names = get_list_of_io_pins();
     if (set_to_network_section())
          {
          advance_to_next_token();
          while ((signal_name = get_signal_name()) != NULL)
               {
               if (! is_io_pin_name(signal_name,
                                   list_of_io_pin_names))
                    {
                    output_signal_header();
                    output_signal(signal_name, ALTERA_TYPE);
                    end_the_signal_decl();
                    signal_header_printed = 1;
                    }
               }
```

```
        if (signal_header_printed)
            {
            end_signal_decls();
            }
        }
}

/*************************************************************
*
*  Function:   end_body
*
*  Interface:  void end_body(char *entity_name)
*
*  Purpose:     This procedure outputs the end to the
*         entity architecture body.
*
*************************************************************/

void end_body(char *entity_name)
{
     char *architecture_prefix, *architecture_name;

     architecture_prefix = (char *)malloc((12 +
          strlen(entity_name)) * sizeof(char));
          strcpy(architecture_prefix, "structured_");
     if((architecture_name = strcat(architecture_prefix,
                                    entity_name)) == NULL)
          {
          printf("ERROR in end_body\n");
          }
     printf("end %s;\n\n", architecture_name);
}

/*************************************************************
*
*  Function:   begin_body
*
*  Interface:  void begin_body(char *entity_name)
*
*  Purpose:     This procedure outputs the beginning of the
*         entity architecture body.
*
*************************************************************/

void begin_body()
{
     printf("begin\n\n");
}

/*************************************************************
*
*  Function:   generate_architecture_header
*
*  Interface:  void generate_architecture_header(
```

```
*                          char *entity_name)
*
*  Purpose:     This procedure outputs the header for the
*         entity architecture.
*
*************************************************************/

void generate_architecture_header(char *entity_name)
{
     char *architecture_prefix = "structured_",
          *architecture_name;

     if ((architecture_name = strcat(architecture_prefix,
                                     entity_name)) == NULL)
          {
          printf("ERROR in gen_arch_header\n");
          }
     printf("architecture %s of %s is\n\n  ",
architecture_name,
                                    entity_name);
}


/*************************************************************
*
*  Function:   get_input_pin_name
*
*  Interface:  char *get_input_pin_name()
*
*  Purpose:     This function returns a pointer to the name
*         of an input.
*
*************************************************************/

char *get_input_pin_name()
{
     char *input_pin_name;

     advance_to_next_token();
     while (strcmp(top_token(), ")") != 0)
          {
          if (is_identifier_variable())
               {
               input_pin_name = top_token();
               }
          advance_to_next_token();
          }
     return(input_pin_name);
}

/*************************************************************
*
*  Function:   build_entity_architecture
*
```

```
*  Interface:  void build_entity_architecture(
*                          char *input_file)
*
*  Purpose:     This procedure generates the entity
*         architecture from the tokens.
*
*************************************************************/

void build_entity_architecture(char *input_file)
{
     char *entity_name, **list_of_io_pins;

     entity_name = get_entity_name(input_file);
     screen_message("\r\nstarting entity
                          architecture....\r\n");
     generate_architecture_header(entity_name);
     screen_message("\r\nmaking signal
                          declarations....\r\n");
     generate_signal_decls(list_of_io_pins);
     screen_message("\r\nmaking component
                          declarations....\r\n");
     generatc_component_decls();
     begin_body();
     screen_message("\r\nmaking signal assignment
                                  statements....\r\n");
     generate_signal_assignment_statements();
     screen_message("\r\nmaking component
                                nstantiations....\r\n");
     generate_instantiations();
     end_body(entity_name);
     screen_message("\r\nfinished....\r\n");
}
```

altransf.c

```
/*************************************************************
*                                                            *
*                        altransf.c                          *
*                                                            *
*************************************************************
*
*  Module:     altransf.c
*
*  Version:    1.0
*
*  Purpose:     This module contains procedures and
*         functions for manipulating the tokens
*         data structure of the adftovhd driver
*         program.  Specifically, this module is
*         concerned with the transformation of an
*         Altera file.
*/
```

```
#include <altransf.h>

/*********************************************************
*
*  Function:   declare_entity
*
*  Interface:  void declare_entity()
*
*  Purpose:     This procedure outputs a message for
*         the start of the entity declaration.
*
*********************************************************/

void declare_entity()
{
     printf("Here is the entity declaration\n");
}

/*********************************************************
*
*  Function:   error_message()
*
*  Interface:  void error_message()
*
*  Purpose:     This procedure outputs an error message.
*
*********************************************************/

void error_message()
{
     printf("ERROR ***OUTPUT: SHOULD HAVE ALREADY BEEN
                                        PROCESSED***\n");
}


/*********************************************************
*
*  Function:   error_message2()
*
*  Interface:  void error_message2()
*
*  Purpose:     This procedure outputs an error message.
*
*********************************************************/

void error_message2()
{
     printf("ERROR ***EQUATIONS: SHOULD HAVE ALREADY BEEN
                                        PROCESSED***\n");
}


/*********************************************************
*
```

```
*  Function:   is_section_header
*
*  Interface:  int is_section_header(tokon *current_token)
*
*  Purpose:     This function returns true is
*         "current_token" is a section header.
*
*************************************************************/

int is_section_header(tokon *current_token)
{
          return(! (strcmp(current_token, "INPUTS:"))     |
              ! (strcmp(current_token, "OUTPUTS:"))    |
              ! (strcmp(current_token, "NETWORK:"))    |
              ! (strcmp(current_token, "EQUATIONS:")) |
              ! (strcmp(current_token, "END$")));
}


/*************************************************************
*
*  Function:   is_code_for
*
*  Interface:  char is_code_for(token *current_token)
*
*  Purpose:     This function returns a pointer to the code
*         for "current_token".
*
*************************************************************/

char is_code_for(tokon *current_token)
{
     if (! strcmp(current_token, "INPUTS:"))
          {
          return('I');
          }
     else if (! strcmp(current_token, "OUTPUTS:"))
          {
          return('O');
          }
     else if (! strcmp(current_token, "NETWORK:"))
          {
          return('N');
          }
     else if (!strcmp(current_token, "EQUATIONS:"))
          {
          return('Q');
          }
     else if (!strcmp(current_token, "END$"))
          {
          return('E');
          }
     else
          {
```

```
            return('Z');
            }
}


/*************************************************************
*
*  Function:   make_comment
*
*  Interface:  void make_comment(token *current_token,
*                   int start_or_continue)
*
*  Purpose:     This procedure outputs a comment from the
*         "current_token" while "start_or_continue".
*
*************************************************************/

void make_comment(tokon *current_token, int
start_or_continue)
{
     char *comment = "-- ";
     static int comment_character_count = 0, comment_started
                                                        = 0;

     if (! start_or_continue)
          {
          if (comment_started)
               {
               printf("\n\n");
               comment_started = 0;
               comment_character_count = 0;
               }
          else
               {
               printf("\n");
               }
          }
     else
          {
          if (! comment_started && *current_token != LF)
               {
               printf("%s", comment);
               comment_started = 1;
               }
          if (*current_token == LF)
               {
               printf("%s", current_token);
               comment_character_count = 0;
               comment_started = 0;
               }
          else if ((comment_character_count +
               strlen(current_token)) < 77)
               {
               printf("%s", current_token);
```

```
                    comment_character_count =
                         comment_character_count +
                              strlen(current_token);
                    }
               else
                    {
                    printf("%s", comment);
                    comment_character_count = 0;
                    printf("%s", current_token);
                    comment_character_count =
                         comment_character_count +
                              strlen(current_token);
                    }
               }
          }
```

BIBLIOGRAPHY

1. Borland. Turbo C User's Guide. Scotts Valley: Borland International, 1988.

2. Borland. Turbo C Reference Guide. Scotts Valley: Borland International, 1988.

3. Altera. A+plus User Guide. Santa Clara: Altera Corporation, 1985, 1986, 1987.

4. Altera. A+plus Reference Guide. Santa Clara: Altera Corporation, 1985, 1986, 1987.

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

6. Coelho, David R. The VHDL Handbook. Norwell: Kluwer Academic Publishers, 1989.

7. Kernighan, Brian W. and Ritchie, Dennis M. The C Programming Language. Murray Hill: Prentice Hall, 1988.

8. Alford, Roger C., Programmable Logic Designer's Guide. Howard W. Sams & Company, 1989..