Behavioral Design Assistant (BdA) User's Manual Version 1.0

Loganath Ramachandran
Daniel D. Gajski

Technical Report #94-36
Sep 1, 1994

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-8059

ramachan@ics.uci.edu

Abstract

*This report provides instructions for installing and using the Behavioral Design Assistant (***BdA***). BdA is a behavioral synthesis system that synthesizes structures from an abstract description written with VHDL behavioral constructs. The system uses components from a generic component library (GENUS), which can be mapped into a user defined technology library. The output of BdA is in structural VHDL and could be verified using a commercial VHDL simulator. The designer can control the synthesis process by providing different resource constraints to the system. BdA is also capable of producing different architectures which can be selected by the designer.*

Contents

List of Figures

1 Overview

With wide acceptance of logic synthesis tools, the design process is gradually moving towards a **describe-and-synthesize** paradigm, where the designer specifies the intent of the design and the tool synthesizes a design that performs according to the given behavior. Also, with the growing complexity of the designs itself, the focus in the describe-and-synthesize paradigm is on higher levels of abstraction. By specifying the design at higher levels such as the *behavioral* or the *algorithmic* level, the designer can minimize the amount of specification details thereby increasing the productivity.

The Behavioral Design Assistant (BdA) is a tool to aid designers at the architectural level. It follows the describe-and-synthesize paradigm since it allows designers to specify the design behavior using VHDL, and it synthesizes a netlist consisting of a controller and a datapath. In Figure 1 we show the input and the output interfaces to BdA.

The input to BdA is a VHDL description of the design behavior. VHDL, which is an acronym for VHSIC(Very High Speed Integrated Circuit) Hardware Description Language, is being widely accepted as one of the standards for describing design behavior.

BdA supports a large subset of the VHDL language. It allows several data types such as bits, bit-vectors and integers with subranges. It also supports one-dimensional arrays of these data types. BdA supports standard sequential statements, such as assignment statements, conditionals statements, while loops and wait statements. The appendix provides a list of constructs that is not supported in this version of BdA.

The output of BdA is a structure netlist, which consists of two parts (a) a controller and (b) a datapath. The datapath consists of RTL components such as register files, ALUs and Memories as shown in Figure 1. The controller which is an implementation of a finite state machine, provides control signals to the datapath components. Since VHDL can also be used to represent a netlist structure, a VHDL structural description of the structural netlist is generated by BdA.

BdA follows a knobs-and-guages approach towards synthesis. The designer can control the synthesis trajectory, by setting the appropriate knobs. For example it is possible to control the amount of resources or the area of the design by adjusting the amount of *Resources*. After synthesis the characteristics of the design are reflected in the various guages. For example the *clock steps* gauge gives an indication of the size of the controller and the performance of the design.

1.1 BdA Advantages

There are many advantages in using the BdA tool for high level architectural design. We list some of the important ones below:

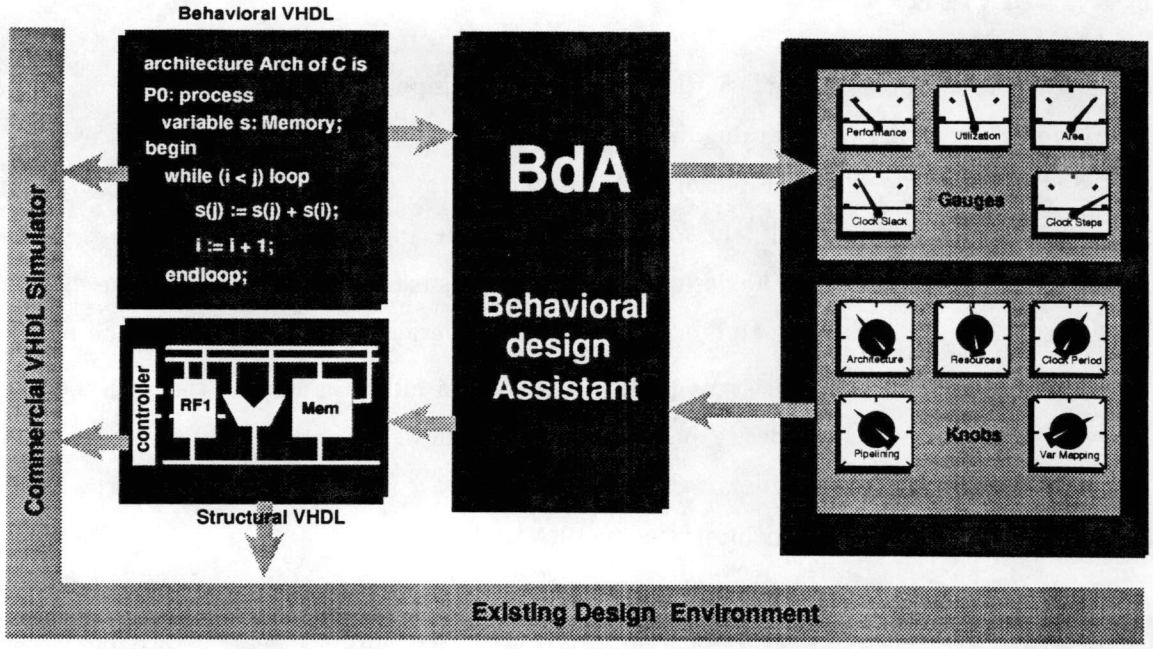


Figure 1: Behavioral Design Assistant

1. **Orders of magnitude in Productivity Gain:** BdA takes less than a couple of minutes to produce RT netlists comprising of thousands of gates. These rapid turnaround times decrease the design cycle tremendously. Moreover, BdA can automate the laborious controller-specification portion of the design process. With BdA it is possible to achieve orders of magnitude in productivity gain.

2. **Efficient Design Space Exploration:** BdA allows designers to explore different resource allocation, choose different memory architectures, select different architectural styles and pick any mapping of variables to storage modules. This large set of choices facilitates rapid and efficient exploration of the design space.

3. **Ease of modeling and use:** Since VHDL is one of the standard languages, no additional training is required to use the BdA system. Since the design is specified at the behavioral level, the amount of details are minimal. This results in smaller modules with very few errors.

4. **Integration into existing CAD environments:** BdA uses a generic library called GENUS [1] for synthesis. The library generates simulation models and logic equations for all the components used in the final design. The simulation models allow for easy verification of the synthesized design. The logic equations facilitate easy integration into the existing CAD environments, since the components can be further synthesized with existing logic synthesis tools.

This report provides instructions for installing and using the BdA software in your site and is organized as follows: the next section provides detailed instructions for installing the software. Section 3 gives a brief idea of the human interface and how to use the system. Section 4, we show the different commands that can be invoked with an Xwindows based environment or with a an UNIX shell. Section 5 provides a hands-on tutorial.

2 Installation Instructions

2.1 Requirements

This version of BdA has been tested on SUN-4 platforms with SUN OS 4.1.1. The BdA software requires around 20 Megabytes of disk space. The interactive displays, require XWindows (Release X11R5) and Motif (Release 1.1). The control synthesis requires misII [2] distributed as part of the Octtools release from by UC Berkeley.

2.2 Obtaining the Release

The BdA software can be obtained from UC Irvine through anonymous ftp. However, prior to obtaining the release it is necessary to sign a non-disclosure agreement. Copies of the agreement can be obtained at the contact address given in the title page of this document.

The release can be obtained via FTP in the following manner:

1. create a release dir: **mkdir /home/BdA**
2. ftp to the address: **ftp ftp.ics.uci.edu**
3. login: **anonymous**
4. password: **your email address**
5. change group name: **quote site group <grp-name>**
6. enter group passwd: **quote site gpass <grp-passwd>**

<NOTE: The <grp-name> and the <grp-passwd> for commands 5 and 6, will be provided after you sign the non-disclosure agreement forms.>

7. change directory: **cd pub/cadlab-releases/BdA**

8. get the file : **get BdA_1.0.tar.Z**

2.3 Installing the release

Let us assume that the software has been ftped into the /home/BdA directory. The main installation process consists of uncompressing and untarring the software. The steps for the installation are given below:

1. Uncompress the software with the following command

   % uncompress BdA_1.0.tar.Z

2. Extract the software from the tarred directory

   % tar -xvf BdA_1.0.tar

   This will create a directory BdA_1.0.

3. Edit the script file in /home/BdA/BdA_1.0/scripts/BdA_script, to set the environmental variables. The variable *RELEASE_DIR* points to the directory where the software was installed, which is /home/BdA/BdA_1.0 in this example. The *OCTTOOLS_DIR* points to the location where OCTTOOLS is installed on your network.

3 BdA System

The BdA system is a synthesis framework that contains a collection of tools. The overall picture of the BdA system is shown in Figure 2. In addition to a VHDL compiler, the system consists of three sets of tools. The VHDL compiler compiles the input behavioral description into an internal Control Data flowgraph (CDFG) format, which is used by the other tool sets in the BdA system. These sets of tools are: (i) *Synthesis Manager* tools, (ii) *Constraint Manager* tools and (iii) *Display Manager* tools.

3.1 Synthesis Manager

BdA uses the FSMD model for synthesis[3, 4]. The implementation of the FSMD model is shown in Figure 3. It contains two parts (a) **A datapath** that has storage units for storing data, functional units for performing arithmetic and logic operations on the data, and interconnect units for moving the data between the storage and the functional units. (b) **A control unit** represented by a finite-state

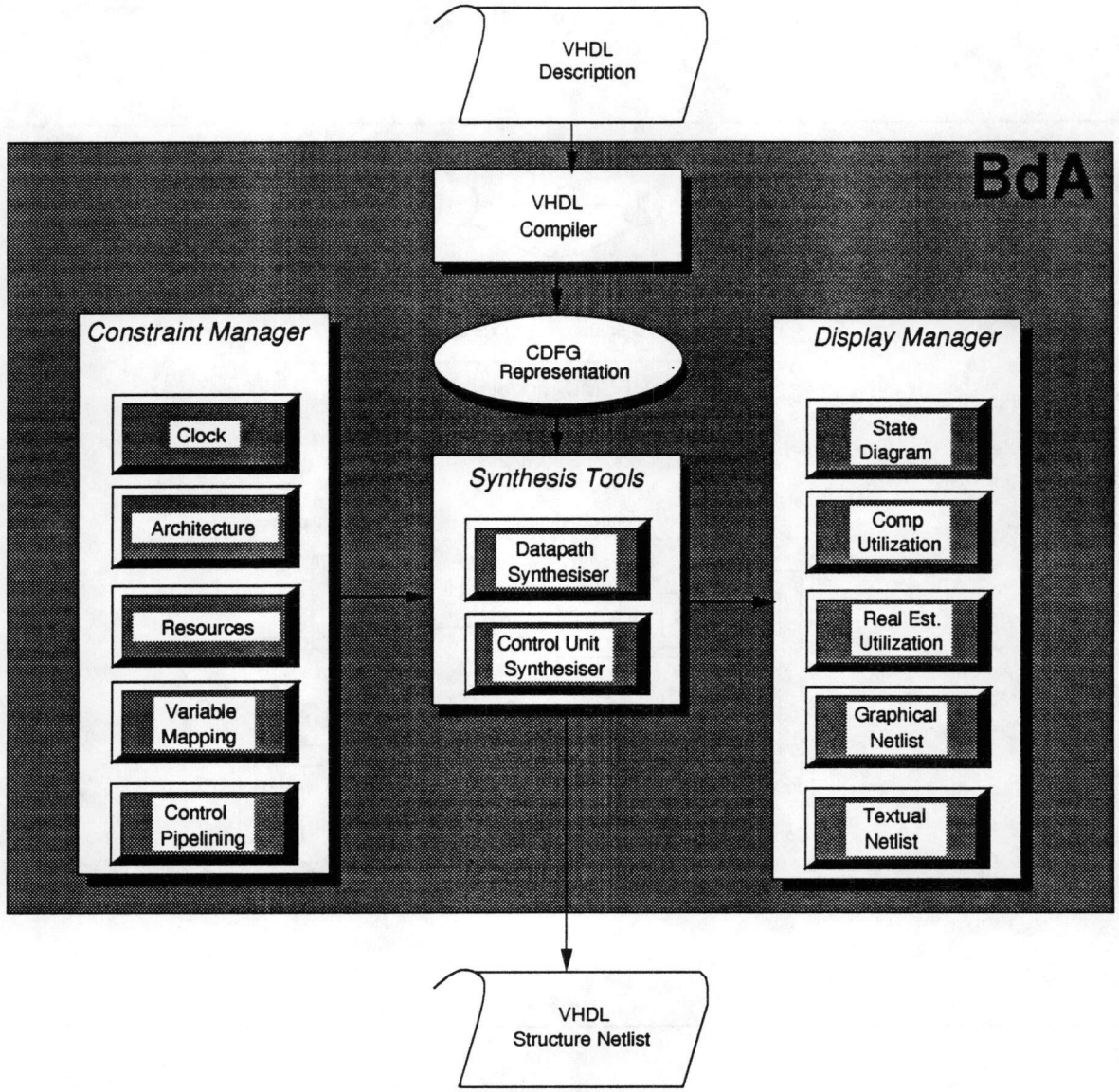
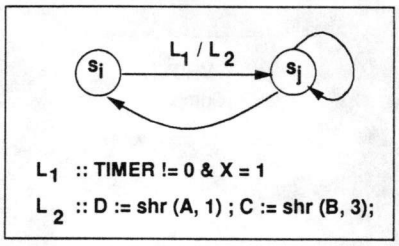
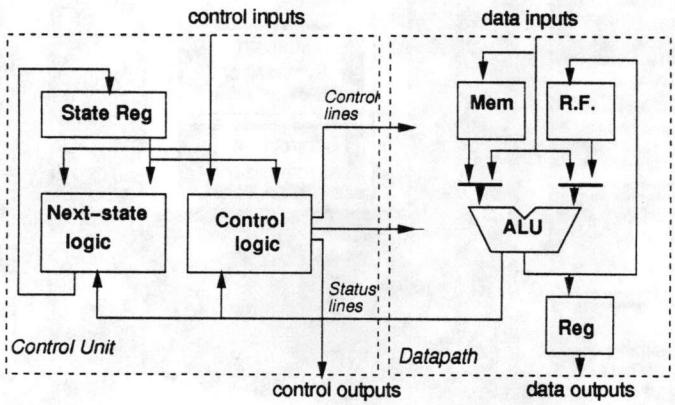


Figure 2: The BdA System



(a) FSMD model



(b) FSMD implementation

Figure 3: FSMD model

machine (FSM), that provides the signals to sequence the operations and the movement of data. The control unit is implemented with a state register and two combinational blocks that compute the next state values and the output values for each state.

The Datapath and the control communicate through the status lines and the control lines. The control lines carry the control signals from the control unit to the datapath, while the status lines carry the status information of certain computations from the data path to the control unit. The FSMD communicates to the external world by means of I/O points on the boundary. The datainputs carry the input data from the external world to the FSMD, while the external control inputs control the sequencing of the states.

The Synthesis Manager consists of two different tools, one for datapath synthesis and the other for control synthesis.

3.1.1 Datapath Synthesizer

The Datapath Synthesizer maps the operations, the variables and the data transfers to individual resources on the datapath. Since the resources available on the chip are limited, the datapath synthesizer has to share them efficiently. This is done in two steps. The operations and the variable accesses in the behavioral description are partitioned into states (or control steps), in such a way that they can be executed by the allocated resources. This task of partitioning the behavior into time intervals is called **scheduling.**

Although the scheduling task assigns each operation to a particular state, it does not assign it to a particular component. In order to obtain the proper implementation we have to assign each variable to a storage unit, each operation to a functional unit and each transfer from I/O ports to units and among units to an interconnect unit. This task is called **binding** or **resource sharing.**

After the scheduling and the binding tasks, the datapath is completely defined. More details about scheduling and binding can be got from the vast literature in behavioral synthesis (e.g., [3, 4]).

3.1.2 Control Unit Synthesizer

After datapath synthesis the control specifications are defined. These control specifications indicate the value on the control lines for each state of the design. The control unit synthesizer synthesizes the structure of the control unit from these control specifications.

The control unit synthesizer performs the tasks of state encoding and logic minimization. During state encoding a binary string is assigned to each state. This binary string is the encoded value for the state. Logic minimization ensures that the size of the combinational blocks, which generate the next state and the output values, are minimized. BdA uses misII [2] for logic minimization of the control

unit.

3.2 Constraints Manager

Given a behavioral description, it is possible to generate thousands of design implementations. It is necessary to constrain the synthesis tasks in order to generate designs that are useful. for a desired application. The Constraint Manager provides a mechanism for capturing different types of constraints for synthesis.

The Constraint Manager consists of five different constraint capture mechanisms, as shown in Figure 2. These include mechanisms for capturing clock constraints, architectural styles, quantity and type of resources, variable mapping schema and control pipelining styles. We will now examine each of these in detail.

3.2.1 Resources

BdA is a resource-constrained synthesis system. This implies that the designer can control the resources allocated for the synthesis and BdA produces an efficient schedule that maximizes the utilization of the allocated resources. Generally, if more resources are allocated then more operations can be executed simultaneously in a given clock period. This results in high-performance designs, where performance is defined as the product of the average number of states over all execution paths and the clock period. On the other hand, allocating fewer resources results in a larger number of states, thereby degrading the performance.

3.2.2 Clock

The clock value indicates the period of the clock that will be used to drive the design. If the clock period is known before synthesis (due to other system constraints), then this can be specified to the BdA, as a synthesis constraint. If the clock period is not a constraint, then it is possible to experiment with different clock periods and determine the clock period that results in the best performance.

Clock period affects the performance of the design. If a higher clock period is given, then many operations can be chained in a clock period. This *chaining* reduces the number of states in the design. On the other hand if a lower clock period is specified by the designer, then an operation may take multiple clock cycles to complete. This process of scheduling an operation over multiple clock cycles is called *multiclocking.*

3.2.3 Architecture

With BdA it is possible to synthesize two different types of architectures, which are: (i) Mux-based architecture and (ii) Bus-based architecture.

In the Mux-based architecture, BdA uses multiplexers to interconnect the functional units and the storage units. However, BdA attempts to optimize the number of mux inputs that are used in the design.

In Bus-based architecture, BdA uses busses to route data between the functional units and the storage units. However, it is possible to constrain the number of busses used in the design. by specifying a fixed number of busses as a constraint. As a general analysis, increasing the number of busses would improve the performance of the design, since the bottlenecks on the busses can be avoided.

3.2.4 Variable Mapping

By default BdA assigns a storage unit to store each scalar variable in the description. Therefore the number of registers in the design would be equal to the number of scalar variables in the description.

However, the designer can specify the desired variable mapping for the array variables in the description. These array variables are mapped onto register files and memory units. As an example, the designer can allocate one memory for the design and constrain all variables in the description to be stored in this memory module. The advantages of such variable grouping are discussed in [5].

3.2.5 Control Pipelining

The FSMD design style can further be divided into three different architectures, each with a different level of control pipelining. More details of the control pipelined architectures produced by BdA is available in [6]. These architectures include:

- non_pipelined architecture where the control and the status signals are not pipelined.
- status_pipelined architecture where the status signals are pipelined
- control_status_pipelined architecture where the control and status signals are pipelined.

Figure 4 shows the architecture for the three pipelining methods. The designer can select any one of these architectures.

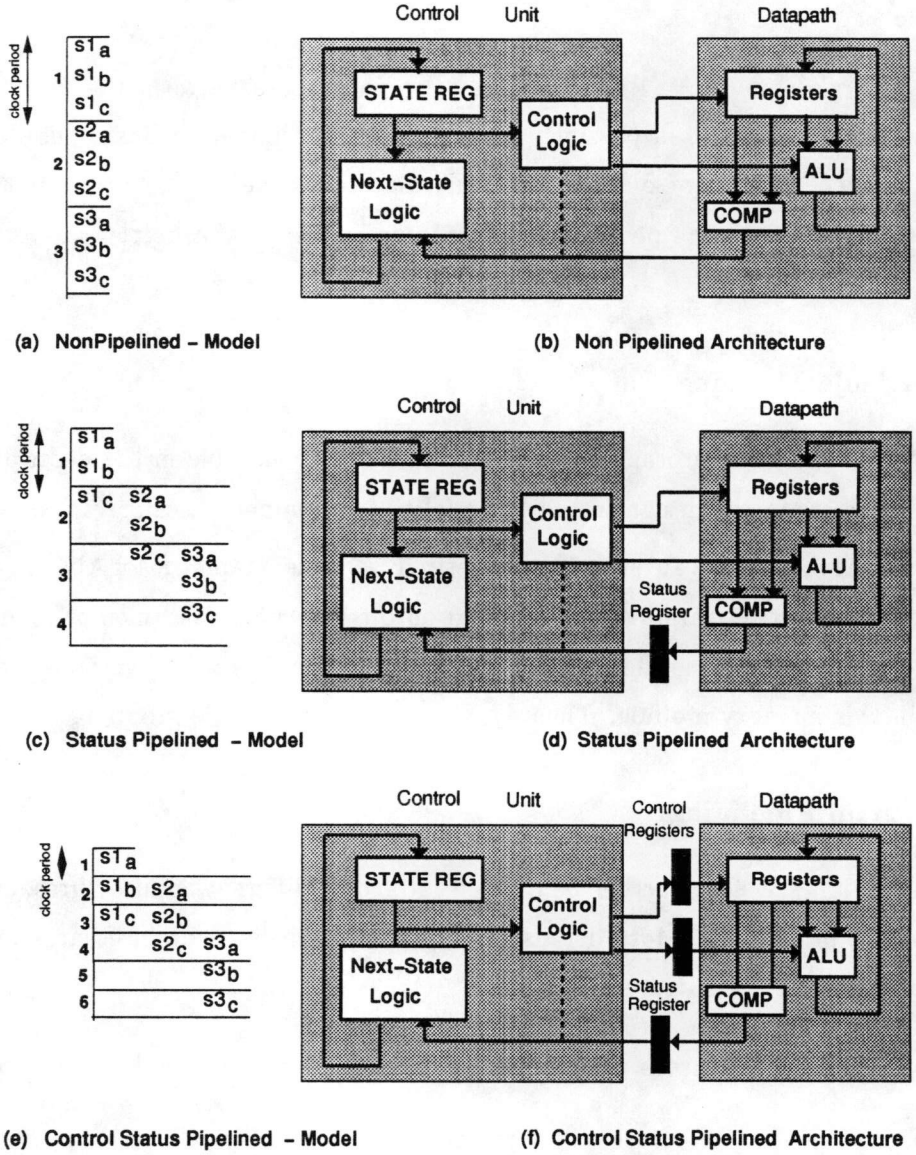


Figure 4: Control Pipelining Schemes

3.3 Display Manager Tools

After the synthesis process completes, different types of results and quality metrics are available from the BdA. The Display Manager Tools provides these quality metrics to the designer, Figure 2 shows some of the results available from BdA after synthesis.

3.3.1 State Diagram

The State Diagram displays the states and the state transitions. The conditions for each state transition and the actions that are performed during each state transition are also available from the state diagram.

3.3.2 Component Utilization

The Component Utilization table indicates how the components are utilized during each state or control step. The general goal of synthesis is to reuse the components as much as possible, and generate designs with a very high utilization.

3.3.3 Real Estate Utilization

This quality measure indicates the amount of the chip's real estate (area) that is used by each component. If a component is large then the percentage real estate used by that component is high.

3.4 Netlist

The Netlist displays show the synthesized RTL netlist. At the top level the synthesized netlist consists of a controller and datapath. The datapath consists of an interconnection of RT level components. After Control Synthesis, the controller implements a Finite State Machine, comprising of a state register and a combinational block,

4 Using BdA

BdA can be used in an **interactive mode**, where the designer can control the various features of the tool using the Motif based flexible user-interface. However, experienced designers, may also work with the **shell mode** where the important features of the tool can be invoked directly from the UNIX shell.

Before invoking BdA in either of the two modes the BdA_script must be sourced from the shell. This can be done by using the command

% source /home/BdA/BdA_1.0/scripts/BdA_script

We will first discuss the various commands available in the interactive mode. We will then examine the methods of invoking some of these same commands from the shell mode.

4.1 Interactive Mode

In order to use the interactive mode of BdA, enter the following command

% XBdA

The interface (shown in Figure 5) consists of two important regions. The *Command Area* consists of a row of buttons on the top, which can invoke the synthesis manager and the display manager tools. The *Constraint Area* which consists of a column of buttons on the left side can be used to setup the constraints for the synthesis. A typical sequence of commands in a BdA session would involve, setting up the constraints, invoking the synthesis tools, and then viewing the results of synthesis.

4.2 Constraint Area

As shown in Figure 5, the Constraint Area can be used to set 5 different types of constraints. These include **Clock, Architecture, Control Pipelining, Resources, and StorageMap**

We now provide the details of all the constraint specification mechanisms.

- Clock

  The default clock period assumed by BdA is 100.0ns. In order to change the clock constraint edit the information in the box marked *Clock*.

- Architecture

  BdA supports two different architectures. Select between the Mux-based and the Bus-based architecture, by placing the cursor on one of these choices and clicking the first mouse button <m1>.

- ControlPipelining

  BdA supports three different control pipelining styles. Select the desired style, by placing the cursor on one of the available choices and clicking the first mouse button <m1>.

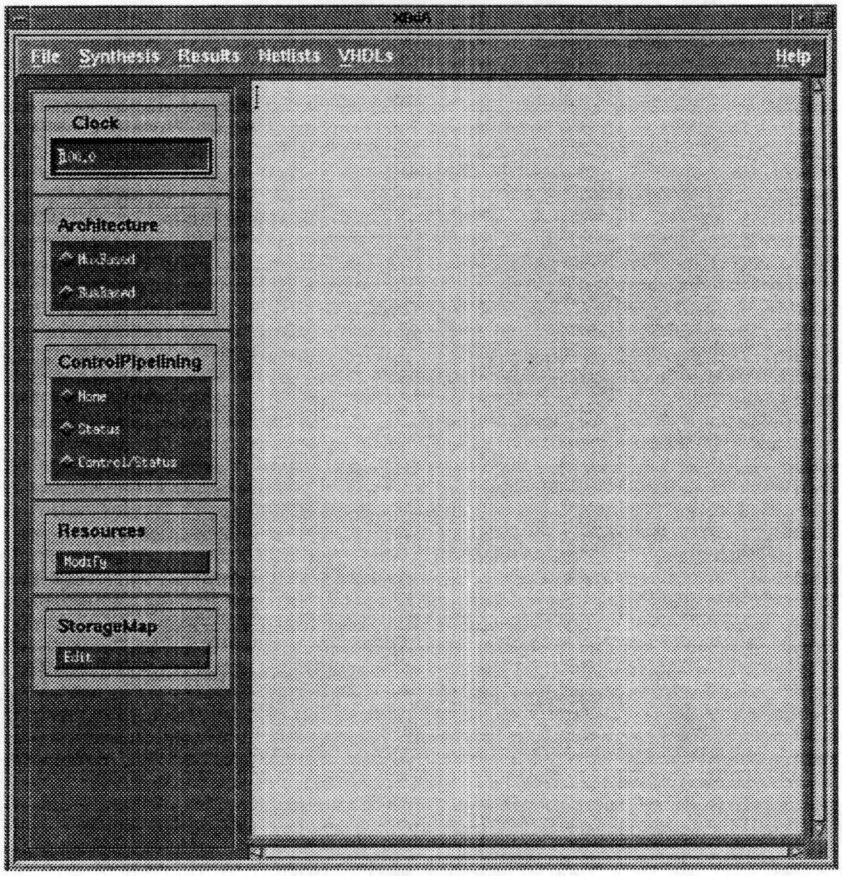


Figure 5: BdA - User Interface

- Resources

  Place the cursor on *Modify* and click the first mouse button <m1>. This brings up a Resource Editor shown in Figure 6. The Resource Editor window allows you to add, delete and modify resources for synthesis.

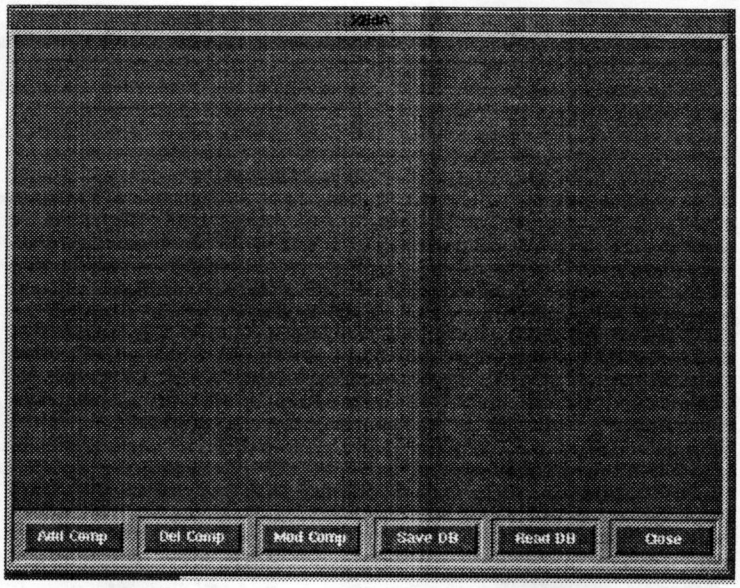


Figure 6: Resource Editor

  - **Add Comp** - allows you to add different types components to the allocation. Invoking this command, brings up a list of component types that are supported by this version of BdA. The command prompts you for the different parameters. For example, the command will prompt for (i) the number of words, (ii) number of read and write ports (iii) the bitwidth, and (iv) instance name, when allocating a register file.
  - **Del Comp** - allows you to delete one of the allocated components.
  - **Mod Comp** - allows you to modify the parameters for one of the allocated components.
  - **Save DB** - saves the allocated components onto a database which is used by the Synthesis Manager.
  - **Read DB** - reads the database containing a list of allocated components.
  - **Close** - allows you to dismiss the Resource Editor window.

- StorageMap

  Place the cursor on *Edit* and click the first mouse button <m1>. BdA displays a Storage Map Editor, which contains a list of the allocated register files. For each register file, you can create a list of variables that will be stored in that file.

4.3 Command Area

As shown in Figure 5, the Command Area consists of 6 different buttons. These buttons include **File, Synthesis, Results, Netlists, and ShowVHDL**

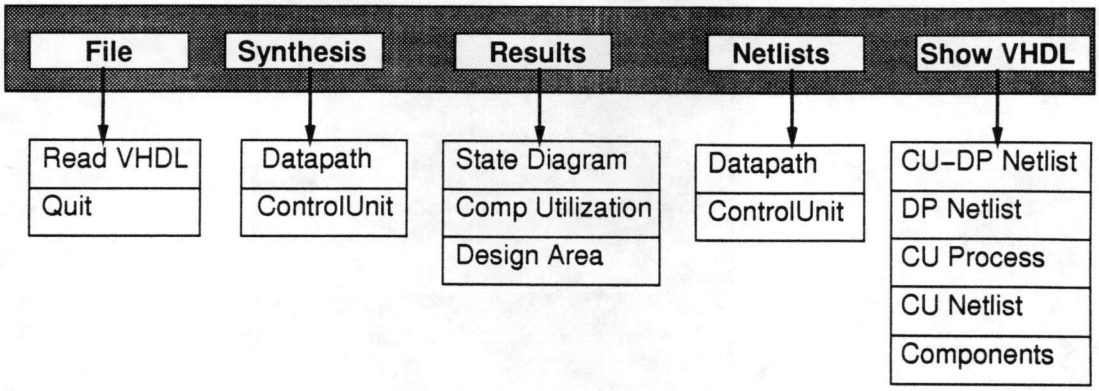


Figure 7: Command Area - List of Commands

You can place the cursor on any of these commands and click on the first mouse button to activate any of these buttons. This action brings up a submenu which contains a further set of commands. The list of commands in the submenu is shown in Figure 7. Without releasing the mouse button, move the cursor to any of these subcommands and then release the button in order to execute any of the subcommands.

We now provide a list of all the submenu commands and give an overview of the actions that would be invoked. We use the notation **XX** $\Rightarrow$ **YY** to indicate a subcommand **YY** which is under the main command **XX**. We now provide a brief description of each of the commands, that can be invoked from the Command Area.

- **File $\Rightarrow$ Read VHDL** This command can be used to specify the location of the VHDL behavioral description. As soon as this command is invoked, BdA pops up a file selection window (as shown in Figure 8), which contains a list of all the possible files. Select the desired behavioral VHDL file for synthesis.

- **File $\Rightarrow$ Quit**
  Use this command to quit the BdA tool.

- **Synthesis $\Rightarrow$ Datapath**
  This command invokes the *Datapath Synthesizer.*

- **Synthesis $\Rightarrow$ ControlUnit**
  This command invokes the *ControlUnit Synthesizer.*

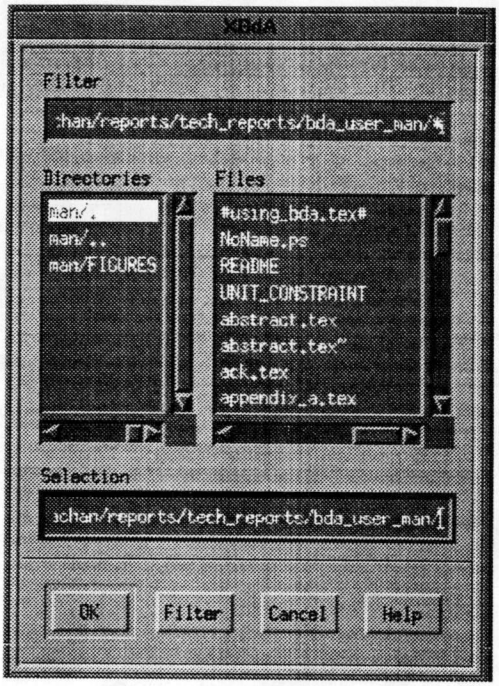


Figure 8: File selection window

- **Results ⇒ State Diagram**

  This command can be used after datapath synthesis, to view the State Diagram of the synthesized design. A sample state diagram display is shown in Figure 9. In this figure, each state is represented by a bubble and the state transitions are represented by arcs connecting the bubbles. The scrollbars on the right and the bottom can be used to scroll the display.

  The state diagram display tool can be used to analyze the actions that occur in each state, or on each transition. Each transition has a black square on the middle of the arc. In order to analyze the actions on a particular transition, move the cursor on the black square of the transition and press the <m1> key. In order to display all the actions that occur in a given state, press the <m1> key after placing the cursor on any of the states.

  The display tool also has 3 buttons at the bottom. They can be used as follows:

  - Delete Text

    Use this button to delete all the text of the actions that are displayed in the main window.

  - Total States

    Clicking on this command will popup an information window that displays the total number of states in the design.

  - Close

    This button can be used to exit the state display window.

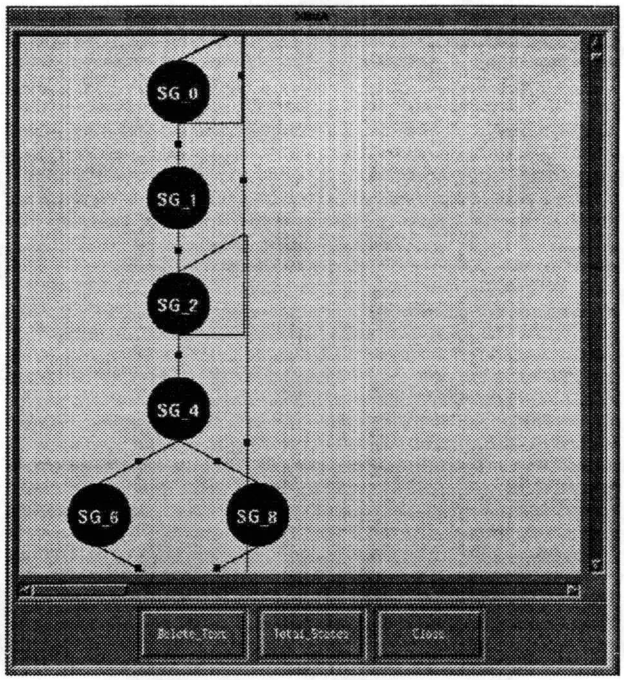


Figure 9: State Diagram display

- **Results ⇒ Comp. Utilization**

  This command can be used after datapath synthesis, to view the utilization of the components in the synthesized design. A sample component utilization diagram is shown in Figure 10. Each row represents a state, and each column represents a component. Each entry in the figure represents the utilization of a component in a given state. A value of 1.0 in entry (x, y) implies that the component is used in that state, while a value of 0.0 implies that the component is idle in that state. For multiport memories, the value of the entry will represent the percent utilization of the ports in each state.

  The scrollbars on the right and the bottom can be used to scroll the display. At the bottom of the figure, a histogram shows the overall utilization of each component in the entire design. This overall utilization is computed as a percentage of the total number of states, in which the component is used. Use the *Close* button to dismiss the component utilization display.

- **Results ⇒ Design Area**

  This command can be used after datapath synthesis, to view the real estate utilization. The real estate utilization is displayed as a histogram in which the 'y' axis is the list of components and the 'x' axis is the percentage of the area that is occupied by the component. The total estimated area of the design is displayed on the top of the figure. A sample histogram is shown in Figure 11.

  The scrollbars on the right and the bottom can be used to scroll the display. Use the *Close*

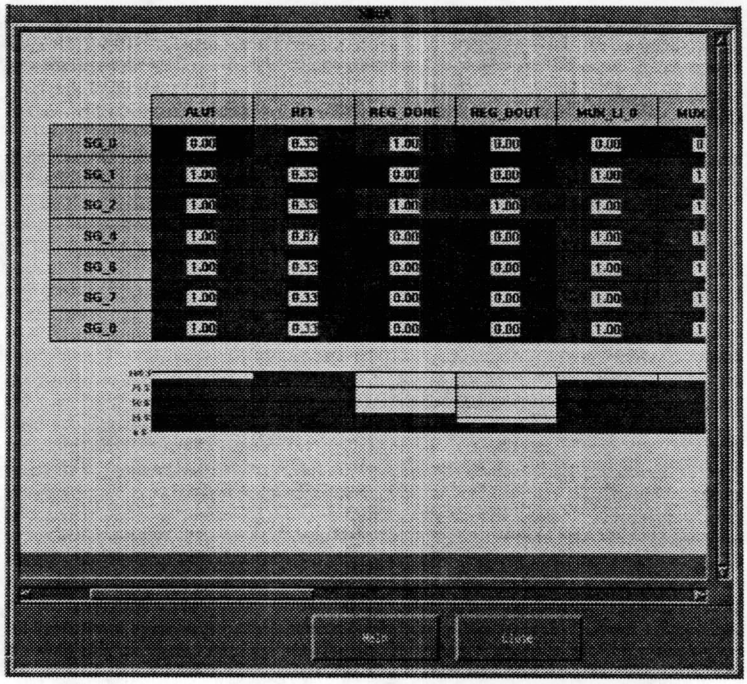


Figure 10: Component Utilization

button to dismiss the area utilization display.

- **Netlists ⇒ Datapath**

  This command brings up a netlist display tool, which displays the datapath of the synthesized design. Each component in the datapath is shown as rectangle, and the control ports are shown on the left. The control lines displayed can be turned off by selecting *CONTROLLINES ⇒ OFF*. They can be turned back on by selecting *CONTROLLINES ⇒ ON*.

  The netlist display tool provides zoom in and zoom out capabilities. In order to zoom into a specific region of the netlist, place the cursor at the north-west corner of the region and press the third mouse button <m3>. Drag the mouse to the south-east corner of the region. and release the mouse button <m3>. Repeating the same actions, while moving the mouse from the south-east corner to the north-west corner will cause a zoom out action to occur. Use the middle mouse button <m2> to refit the picture. The **File ⇒ Quit** option in the netlist display dismisses the netlist display tool.

- **Netlists ⇒ ControlUnit**

  Use this command to show the netlist of the controller, after controller synthesis.

- **ShowVHDL ⇒ CU-DP Netlist**

  This command displays the textual netlist (structural VHDL) consisting of the Control unit and

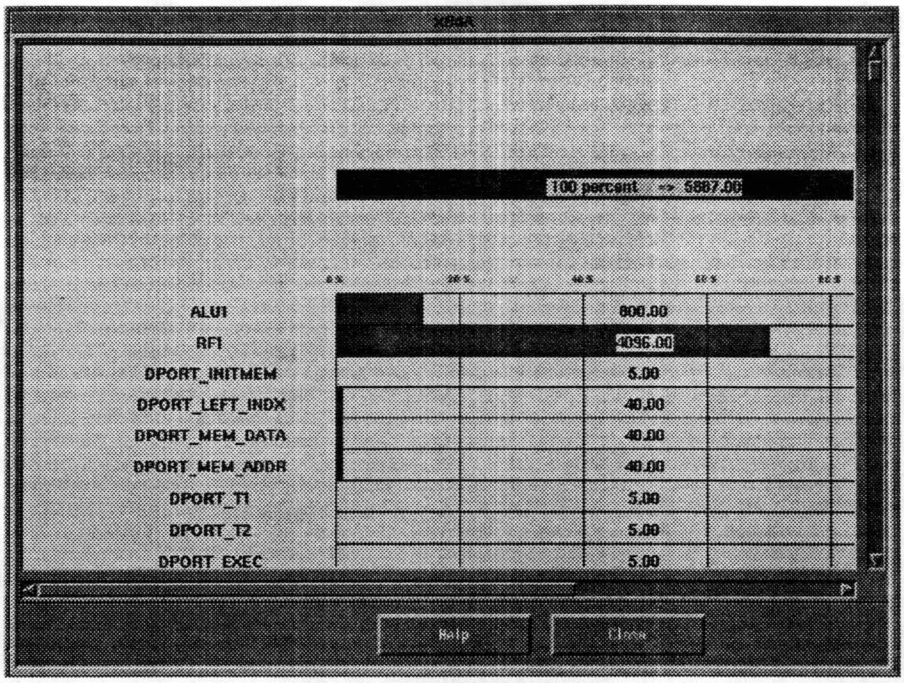


Figure 11: Area Utilization

the Datapath and their interconnections. Figure 13 shows BdA's textual display tool, which is used for displaying all the textual VHDL structures.

- **ShowVHDL ⇒ DP Netlist**
  This command displays the structural netlist of the datapath expressed as VHDL structure.

- **ShowVHDL ⇒ CU Process**
  This command displays the behavioral description of the synthesized control unit.

- **ShowVHDL ⇒ CU Netlist**
  This command displays the structural netlist of the control unit, expressed as VHDL structure.

- **ShowVHDL ⇒ Components**
  This command displays the VHDL description of the components used in the datapath netlist.

4.4 Shell Mode

It is possible to use the essential features of BdA without the X windows interface. The synthesis commands can be directly invoked from the UNIX shell. We divide the commands in the shell mode are divided into three categories. (i) commands to set constraints (ii) commands to invoke tools in the Synthesis Manager, and (iii) commands for results display.

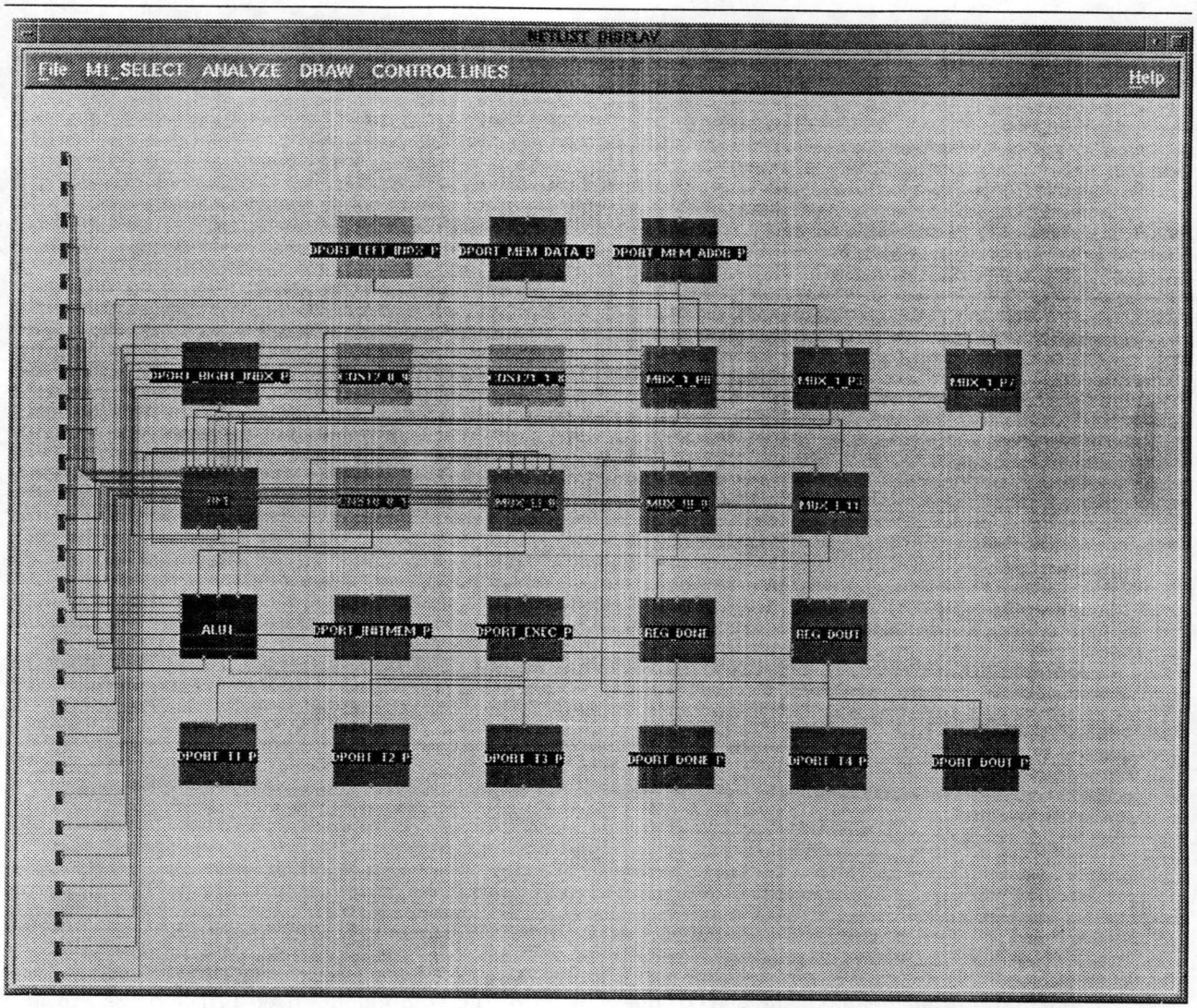


Figure 12: Netlist Display tool

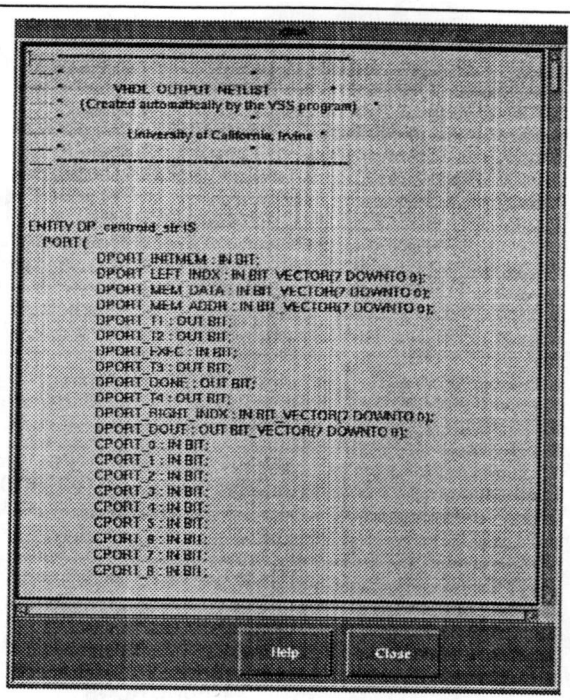


Figure 13: VHDL Display tool

4.4.1 Commands to set constraints

- Resource Constraints

  Specify the resource constraints a file called *UNIT_CONSTRAINT*. A sample UNIT_CONSTRAINT file is provided in $RELEASE_DIR/sample/UNIT_CONSTRAINT (Figure 14).

  The sample file shown contains four components from the GENUS library [1, 7]. Each functional unit definition is demarcated with a line containing a '#' sign. The first functional unit called ADDER1 describes a 16-bit ALU, which performs the ADD function. The second component, called COMP1 is another ALU that performs two comparison functions (i.e., LEQ and LT). The third component is a 16-bit multiplier called MULT_1, and the fourth component is a 1K by 8 REG_FILE called MY_RF containing 1 write port and 2 read ports [1].

  Copy the sample UNIT_CONSTRAINT file to the directory where the behavioral description resides and edit it to reflect the appropriate constraints. Here are some of the ways the file can be modified.

  - Changing the number of instances - Copy the definition of the instance and change the GC_INSTANCE_NAME of the second instance.
  - Changing the bit width of components - Change the *GC_INPUT_WIDTH: 10* line to the desired value (e.g., *GC_INPUT_WIDTH: 16*).

```
GC_COMPILER_NAME: ALU
GC_INSTANCE_NAME: ADDER1
GC_INPUT_WIDTH: 16
GC_NUM_FUNCTIONS:1
GC_FUNCTION_LIST: GC_ADD
GC_STYLE: GC_RIPPLE_CARRY
AREA: 100.0
DELAY: 40.0
#
GC_COMPILER_NAME: ALU
GC_INSTANCE_NAME: COMP1
GC_INPUT_WIDTH: 16
GC_NUM_FUNCTIONS:2
GC_FUNCTION_LIST: GC_LEQ,GC_LT
GC_STYLE: GC_RIPPLE_CARRY
AREA: 100.0
DELAY: 40.0
#
GC_COMPILER_NAME : MULT
GC_INSTANCE_NAME : MULT_1
GC_LEFT_INPUT_WIDTH : 16
GC_RIGHT_INPUT_WIDTH : 16
GC_STYLE : GC_ARRAY
DELAY : 60.0
AREA : 3000.0
#
GC_COMPILER_NAME: REG_FILE
GC_INSTANCE_NAME: MY_RF
GC_INPUT_WIDTH: 8
GC_NUM_WORDS: 1024
GC_NUM_INPUT_PORTS: 1
GC_NUM_INOUT_PORTS: 16
GC_NUM_OUTPUT_PORTS: 2
AREA: 100.0
DELAY: 10.0
#
```

Figure 14: Sample UNIT_CONSTRAINT file

- Deleting a component - Remove the component specification including the '#' sign.
- Creating a new type of component - The ALU in GENUS can perform many different functions. The possible functions are explained in the GENUS manual [1]. To create an ALU that performs the increment and decrement operations copy the lines (including the # line) that represent the ALU component, and change the GC_NUM_FUNCTIONS and GC_FUNCTION_LIST parameters. In other words, the following lines would get appended to the UNIT_CONSTRAINT file.

```
GC_COMPILER_NAME: ALU
GC_INSTANCE_NAME: MY_INC_DEC
GC_INPUT_WIDTH: 16
GC_NUM_FUNCTIONS: 2
GC_FUNCTION_LIST: GC_INC,GC_DEC
GC_STYLE: GC_RIPPLE_CARRY
NUM_INSTANCES: 1
AREA: 100.0
DELAY: 80.0
#
```

- Changing Technology Specific Parameters - The AREA and the DELAY specification provides hints to the synthesis program. This can be changed to reflect the area and delay values of components in your technology library.

- Variable mapping

  BdA reads a file called *STORAGE_MAP* to determine the mapping of variables to the REG_FILEs. A sample STORAGE_MAP is available in $RELEASE_DIR/samples/STORAGE_MAP. Each line of the file specifies the instance name for the register file and the variables that are to be stored in that register file. The sample STORAGE_MAP file shown in Figure 15 contains two lines. The first line indicates that the variables *a, b, c* and *d* are to be stored in the first register file called MY_RF1. The second line indicates that the variables *e, f* and *g* are to be stored in the second register file called MY_RF2. Note that MY_RF1 and MY_RF2 are the instance names of the register files that were allocated in the UNIT_CONSTRAINT file.

  Copy this file to the directory where the behavioral description resides and edit the mapping. It is necessary to specify the mapping for all the array variables in the STORAGE_MAP file.

4.4.2 Commands to invoke synthesis tools

BdA has two separate commands based on the type of architecture that you would like to select.

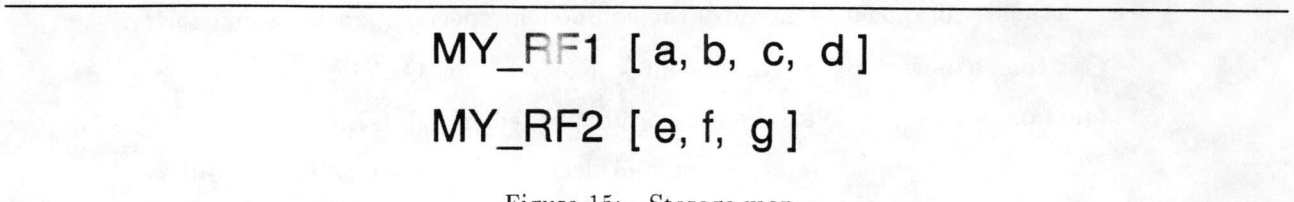


Figure 15: Storage map

- Datapath and Control Synthesis (MuxBased Architecture):
  Use the command
  % BdA_mux [-cp_style <CntrlPipelining>] [-clock <period>] <input_file >
  The <CntrlPipelining> has to be chosen from one of the following: (a) **non_pipelined** (b) **status_pipelined** (c) **control_status_pipelined.** If the -cp_style is not specified, the default is assumed to be non_pipelined. The <period> can be a number of floating point type. If the -clock option is not specified then the default is assumed to be 100.0.

  For example, the command
  % BdA_mux -cp_style non_pipelined -clock 200.0 idct.vhdl
  synthesizes a non_pipelined design from the behavioral description in the file idct.vhdl. The clock period assumed during synthesis is 200ns.

  BdA creates the controller specification for Mux-based architecture in a directory RESULTS_MUX. In order to run Control synthesis for the Mux-based architecture use the command:
  % cd RESULTS_MUX; state_syn < design_name >

  For example, the command
  % cd RESULTS_MUX; state_syn idct
  synthesizes a control unit for the idct design.

- Datapath and Control Synthesis (BusBased Architecture):
  Use the command
  % BdA_bus [-cp_style <CntrlPipelining>] [-clock <period>] <input_file >
  The <CntrlPipelining> has to be chosen from one of the following: (a) non_pipelined (b) status_pipelined (c) control_status_pipelined The default is assume to be non_pipelined.

  The <period> can be a number of floating point type. The default is assumed to be 100.0.

  For example, the command
  % BdA_bus -cp_style status_pipelined -clock 170.0 dct.vhdl
  synthesizes a status_pipelined bus-based design from the behavioral description in the file dct.vhdl. The clock period assumed during synthesis is 170ns.

BdA creates the controller specification for BusBased architecture in a directory RESULTS_BUS.

In order to run Control synthesis for the BusBased Architecture use the command:

% cd RESULTS_BUS; state_syn < design_name >

For example, the command

% cd RESULTS_BUS; state_syn dct synthesizes a control unit for the dct design.

4.4.3 Commands to view synthesis results

The VHDL files created after synthesis reside in the directory RESULTS_MUX for mux-based architectures and in the directory RESULTS_BUS for bus-based architectures. The files that are produced in the directory and their functionality is shown in Figure 16.

Filename	Description
<design>_struct.vhdl	VHDL structural description of design
<design>_dp_struct.vhdl	VHDL structural description of datapath
<design>_cu_struct.vhdl	VHDL structural description of control unit
<design>_cu_process.vhdl	VHDL behavioral description of control unit
STATE_INFO	State transitions and actions

Figure 16: Results Directory

The VHDL files for the components are created in the directory that is pointed to by the environmental variable $GENUS_VHDL_DIR. The default script file that was released sets the environmental variable to the directory "/home/BdA/BdA_1.0/misc/genus_models. ".

5 Simulating the output

All the results of BdA are created in the results subdirectory which is either RESULTS_MUX or RESULTS_BUS based on the architecture being synthesized. The VHDL models for the individual components are be created in the directory pointed to by the environmental variable GENUS_VHDL_DIR.

In order to verify the results of synthesis, simulate the files in the following order.

- types.vhd - the file is available in the directory /home/Bda/BdA_1.0/genus/vhdl_package

- resMVL4_functions.vhd - the file is available in the directory /home/Bda/BdA_1.0/genus/vhdl_package
- all files with a *.vhdl* extension in the directory pointed to by the environmental variable **GENUS_VHDL**
- the datapath structure netlist - the file <design_name>_dp_struct.vhdl in the results directory.
- the control unit process - the file <design_name>_cu_proess.vhdl in the results directory.
- the overall structure consisting of the CU and DP - the file <design_name>_struct.vhdl in the results

After compiling the files in the above order, test vectors can be applied to the overall structure.

6 Tutorial

In this section we will walkthrough an example to illustrate the use of BdA on a real life design. We choose a centroid computation example, which was part of a fuzzy logic controller design [8]).

Figure 17 shows the behavioral description of the circuit. The design computes the centroid of a set of values stored in a array variable *s*. The centroid is defined as that index of the memory which has the sum of the values stored on the left of it equal to the sum of the values stored to the right of it. We will synthesize this design using the interactive and the shell mode in BdA.

6.1 Interactive Mode

Let us go through a step-by-step procedure for synthesizing the centroid design, using the Interactive mode.

1. Source the vss_script file

   % source /home/BdA/BdA_1.0/scripts/BdA_script

2. Create a new working directory and move to this new directory:

   % mkdir $HOME/test_bda; cd $HOME/test_bda

3. Copy the VHDL file from the examples directory

   % cp <RELEASE_DIR>/examples/centroid/centroid.vhdl .

4. Invoke XBdA

   % XBdA .

```
entity CompCentroid is
   port (
        exec:  in bit;
        initmem:  in bit;
        mem_data:  in bit_vector(7 downto 0);
        mem_addr:  in bit_vector(7 downto 0);
        left_indx   : in bit_vector(7 downto 0);
        right_indx   : in bit_vector(7 downto 0);
        dout : out bit_vector(7 downto 0);
        done : out bit;
end CompCentroid;


architecture ArchCompCentroid of CompCentroid is

begin
   P0: process
   type Memory is array (integer range <>) of bit_vector  7 downto 0);
   variable s: Memory (255 downto 0
   variable lsum, rsum : bit_vector(7 downto 0
   variable i,j   : bit_vector(7 downto 0
   begin
     if exec = '1' then
          done <= 0;
          i  := left_indx;
          j  := right_indx;
          while i < j loop
               if rsum < lsum then
                    rsum :=  rsum + s(j
                    j :=  j-1;
               else
                    lsum :=  lsum + s(i
                    i :=  i+1;
               end if;
          end loop;
          dout <= j;
          done <= 1;
     elsif initmem = '1' then
          s(mem_addr) = mem_data;
     end if;
   end process;
 end ArchCompCentroid;
```

Figure 17: Centroid - Behavioral Description

5. Load the VHDL file by selecting **File** $\Rightarrow$ **ReadVHDL** on the main window and clicking on centroid.vhdl in the file selection window.

6. Set clock to 50.0 by editing the clock information in the constraints area.

7. Set the architecture to Mux-based by clicking on the MuxBased item in the *Architecture* box.

8. Set the ControlPipelining style to Nonpipelined by clicking on the NonPipelined item in the *Controlpipelining* box.

9. Modify the allocated resources, by clicking on the *Modify* button in the Resources box. This brings up the ResourceEditor (shown in Figure 6). Let us first allocate one 8-bit ALU to perform the addition/subtraction operation. In order to do this click on the *Add Comp* button on the resource editor and select *ALU* as the type of component to be added. You will be queried for the ALU's parameters. Enter the responses shown in Figure 18.

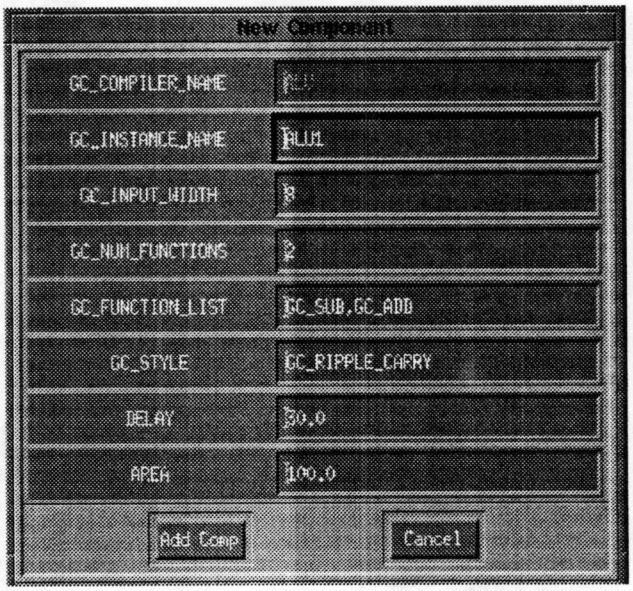


Figure 18: Allocating Adder Subtractor

Let us add another ALU which can perform the comparison operations, LT and LEQ. In order to do this click on the *Add Comp* button on the resource editor again and specify *ALU* as the type of component to be added. Enter the following responses as shown in Figure 19.

Finally let us add a Register File with 1 read port and 1 write port to store the array variable *s*. Click on the *Add Comp* button on the resource editor again and specify *REG_FILE* as the type of component to be added. Enter the following responses (shown in Figure 20) for the REG_FILE.

The Resource Editor displays each resource on the screen as it is being allocated. Figure 21 shows the Resource Editor after all the three resources are added.

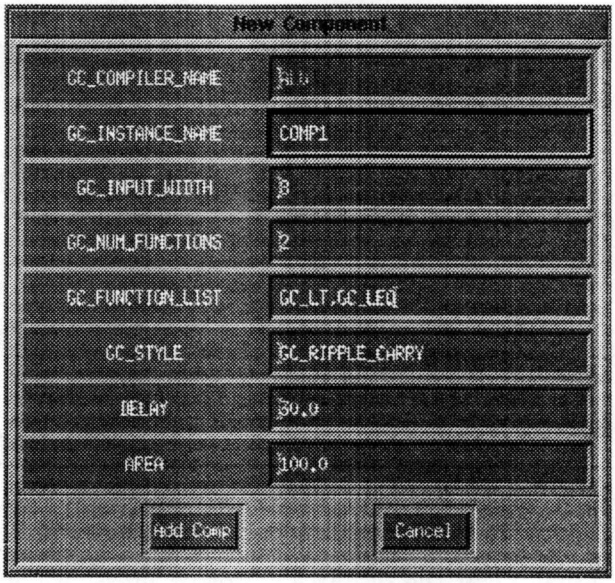


Figure 19: Allocating Comparator

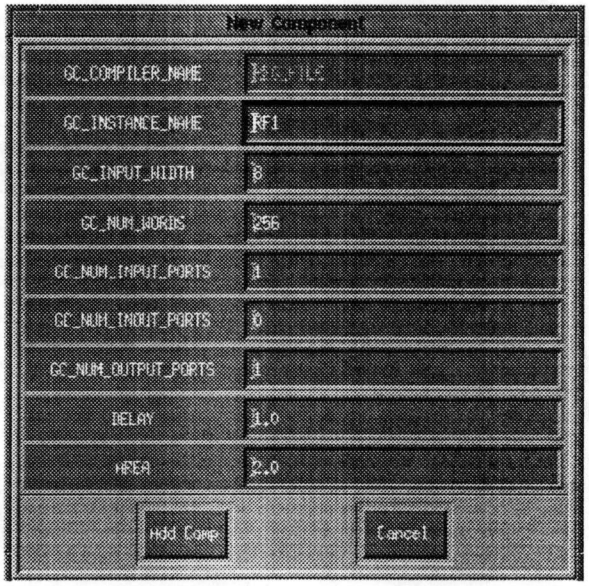


Figure 20: Allocating Register File

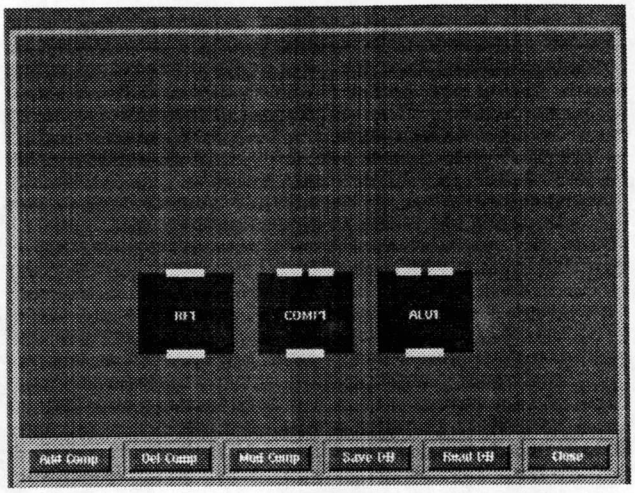


Figure 21: Resource Editor

Save the three components to the allocation database by clicking on the *Save_DB* button in the Resource Editor. Finally exit the resource editor by pressing *Quit.*

10. The next step is to map the array variable *s* to the allocated register file (MY_RF). This is done by clicking on the *EDIT* button in the VariableMapping. This brings up a variable mapping window, which shows all the allocated register files and memories. In this example since only one register file (My_RF1) was allocated. Let us map variable *s* to this register file (Figure 22).

    Save the variable map to the database by clicking on the *Save* button in the Variable Mapping editor. Finally exit the resource editor by pressing *Cancel.*

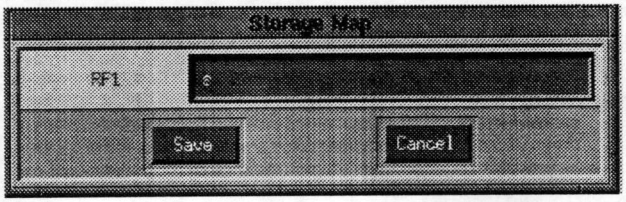


Figure 22: Variable Mapping Editor

11. Synthesize the datapath by selecting **Synthesis ⇒ Datapath.** After synthesis is completed an information window announcing *DP Synthesis Completed* is displayed. Dismiss this window by pressing *Cancel.*

12. Synthesize the control unit by selecting **Synthesis ⇒ ControlUnit.** After synthesis is completed an information window announcing *CU Synthesis Completed* is displayed. Dismiss this window by pressing *Cancel.*

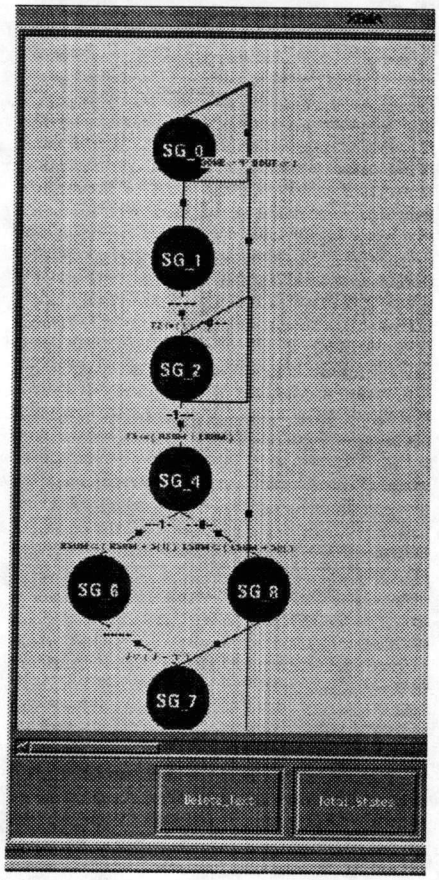


Figure 23: Centroid - State Diagram

13. View the following results that are available after synthesis. (a) Results ⇒ State Diagram (Figure 23). (b) Select Results ⇒ Comp. Utilization (Figure 24), (c) Results ⇒ Design Area displays Figure 25, and (d) Results ⇒ Datapath displays Figure 26,

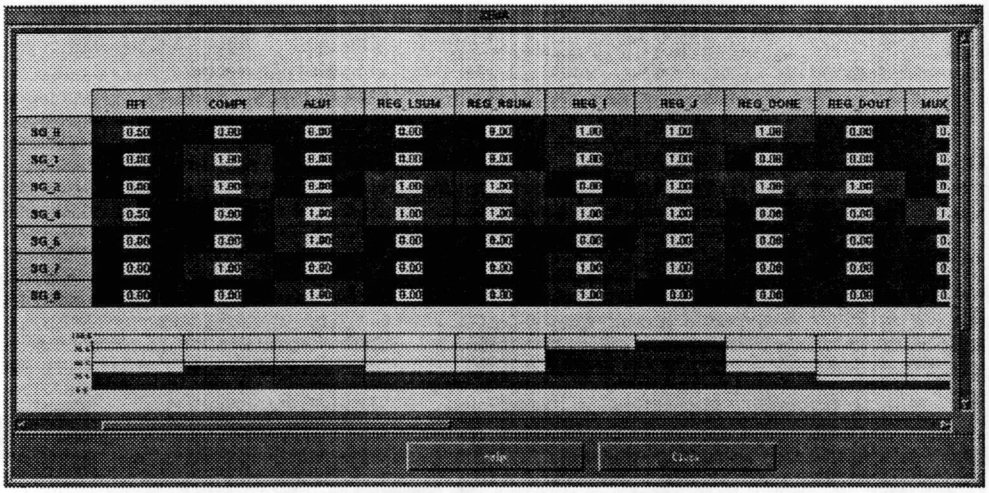


Figure 24: Centroid - Component Utilization

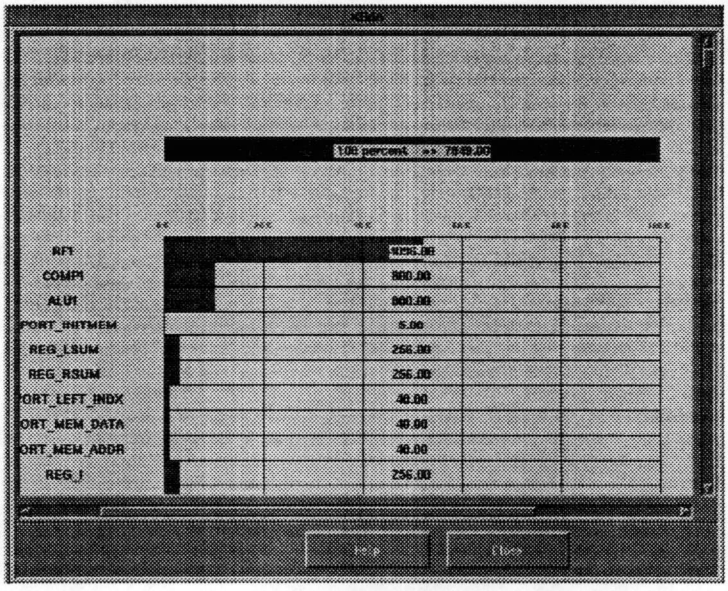


Figure 25: Centroid - Design Area

6.2 Shell Mode

The centroid example can also be synthesized with the Shell mode in BdA. The important steps for the synthesis include:

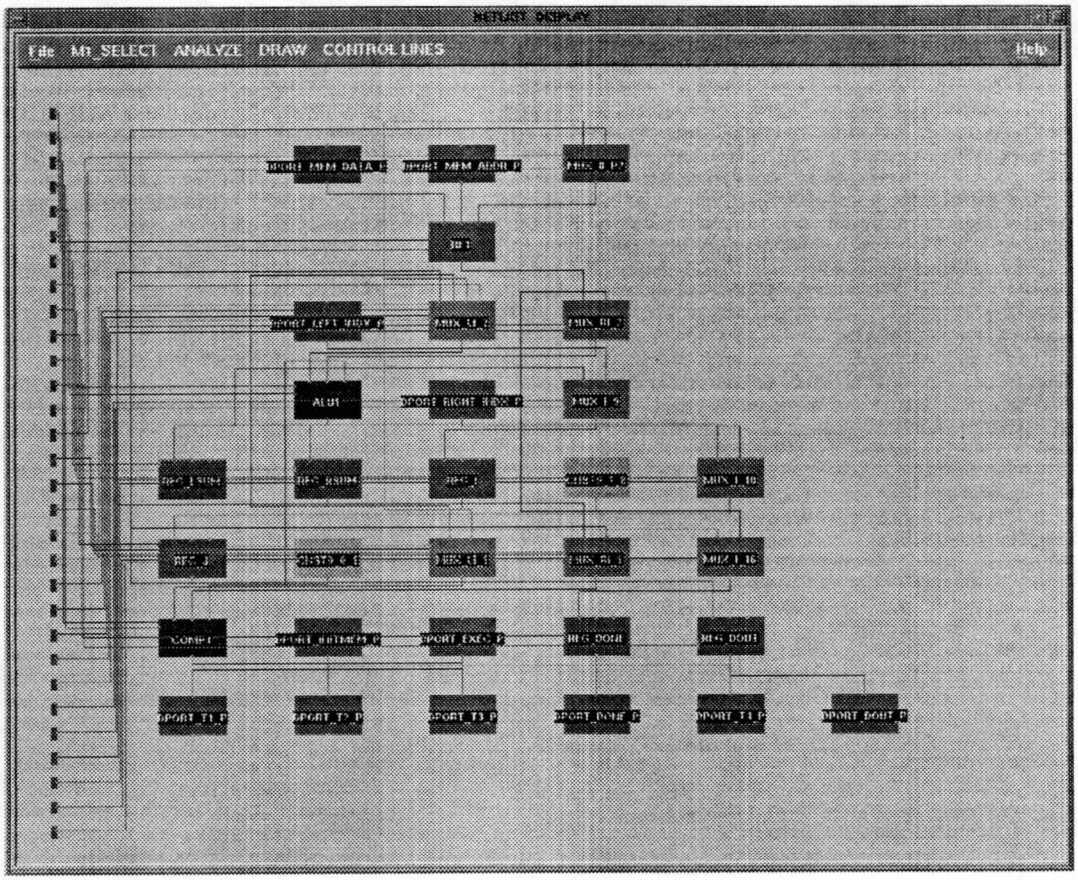


Figure 26: Centroid - Datapath

1. Source the vss_script file

   % source /home/BdA/BdA_1.0/scripts/BdA_script

2. Create a new working directory and move to this new directory:

   % mkdir $HOME/test_bda; cd $HOME/test_bda

3. Copy the VHDL file from the examples directory

   % cp <RELEASE_DIR>/examples/centroid/centroid.vhdl .

4. Create a file called UNIT_CONSTRAINT as shown in Figure 27. As shown in the figure, the UNIT constraint file allocates one 8-bit ALU for performing addition and subtraction, one 8-bit ALU for performing the comparison operations and one Register File with 1 read and 1 write ports.

```
GC_COMPILER_NAME : REG_FILE
GC_INSTANCE_NAME : RF1
GC_INPUT_WIDTH : 8
GC_NUM_WORDS : 256
GC_NUM_INPUT_PORTS : 1
GC_NUM_INOUT_PORTS : 0
GC_NUM_OUTPUT_PORTS : 1
DELAY : 1.0
AREA : 2.0
#
GC_COMPILER_NAME : ALU
GC_INSTANCE_NAME : COMP1
GC_INPUT_WIDTH : 8
GC_NUM_FUNCTIONS : 2
GC_FUNCTION_LIST : GC_LT,GC_LEQ
GC_STYLE : GC_RIPPLE_CARRY
DELAY : 30.0
AREA : 100.0
#
GC_COMPILER_NAME : ALU
GC_INSTANCE_NAME : ALU1
GC_INPUT_WIDTH : 8
GC_NUM_FUNCTIONS : 2
GC_FUNCTION_LIST : GC_SUB,GC_ADD
GC_STYLE : GC_RIPPLE_CARRY
DELAY : 30.0
AREA : 100.0
#
```

Figure 27: Centroid - UNIT_CONSTRAINT

5. Create a file called STORAGE_MAP as shown in Figure 28. This file maps the only array variable in the description *s* to the allocated register file.

```
RF1    [s]
```

Figure 28: Centroid - STORAGE_MAP

6. Synthesize the datapath for a mux based architecture with a clock period of 50.0 ns and a non-pipelined style for control pipelining. This is possible by executing the following command

   % BdA_mux -cp_style non_pipelined -clock 50.0 centroid.vhdl

7. Synthesize the control unit with the following command:

   % cd RESULTS_MUX; state_syn centroid

8. View the following results that are available after synthesis. (a) Datapath Structure

   % edit RESULTS_MUX/centroid_dp_struct.vhdl

   (b) Control Unit Structure

   % edit RESULTS_MUX/centroid_cu_struct.vhdl

   (c) Control Unit Behavior

   % edit RESULTS_MUX/centroid_cu_process.vhdl

   (d) Overall structure

   % edit RESULTS_MUX/centroid.vhdl

7 Appendix A

In this appendix we list some of the features of the VHDL language that are currently not supported by the BdA tool.

BdA does not support the following constructs in VHDL

- Procedure and Function Calls
- Enumerated Types
- Record Structures
- CONSTANT declarations
- Null statements
- Exit statements
- Return Satements
- loop with no iteration scheme
- Arrays of more than 2 dimensions
- Inout Ports

8 References

[1] P. Jha, T. Hadley, and N. Dutt, "The GENUS User Manual and C Programming Library," Tech Rep: 93-32, ICS Dept, University of California, Irvine, CA 92717, 1993.

[2] R. Brayton, R. Rudell, A. SangiovanniVincentelli, and A. Wang, "Mis: A multiple level logic optimization system," in *Transactions on CAD*, 1987.

[3] D. Gajski and L. Ramachandran, "Introduction to High Level Synthesis," in *submitted to IEEE Design and Test*, 1994.

[4] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High Level Synthesis.* Boston, Massachusetts: Kluwer Academic Publishers, 1992.

[5] L. Ramachandran, D. D. Gajski, and V. Chaiyakul, "An algorithm for array variable clustering," in *Proc. of the EDAC94 Conference*, Feb 1994.

[6] L. Ramachandran and D. D. Gajski, "Architectural Tradeoffs in Synthesis of Pipelined Controls," in *Proc. of the European Design Automation Conference*, September 1993.

[7] N. Dutt and P. Jha, "Rt component sets for high-level design applications," in *Proc. 1st Asia Pacific Conf. on HDL Standards and Applications*, 1993.

[8] D. Gajski, L. Ramachandran, P.Fung, S. Narayan, and F. Vahid, "100 hr design methodology: A Test Case," in *Proc. EURO-DAC, Hamburg*, 1994.