The Design Process for Behavioral Synthesis from VHDL

Loganath Ramachandran
Nancy D. Holmes
Daniel D. Gajski

Technical Report #94-04
February 14, 1994

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

Email : ramachan@ics.uci.edu
Email : nholmes@ics.uci.edu

Abstract

In this report we describe the design process for behavioral synthesis from VHDL descriptions. The design process can be divided into two parts, (a) the architectural allocator (AA), which derives the appropriate type and number of resources (storage, functional units, and buses) that can satisfy the area and performance constraints imposed by the designer, and (b) the VHDL Synthesis System (VSS), which synthesizes a RT-level netlist based on the allocation constraints. The human interface for all the tasks in the design process is also described in this report.

Contents

1 Introduction **2**

1.1 Top-Down Design Methodology 2

1.2 Chip-Level Synthesis in **AA-VSS** 4

2 The Chip-Level Synthesis Model **6**

3 Chip-Level Synthesis: Input and Tasks **8**

3.1 Input Description 8

3.2 Allocation 10

3.3 Scheduling 12

3.4 Binding 14

4 AA-VSS : I/O Specifications **16**

5 AA : Block Diagram, Human Interface, and Design Scenarios **18**

5.1 **AA** : Block Diagram 18

5.2 Shape Function Display 20

5.3 Allocation Display 22

5.4 Quality Measure Display 24

5.5 **AA** : Design Methodology 26

5.6 **AA** : Possible I/O Scenarios 28

5.7 **AA** : Typical Design Scenario, Steps 1 and 2 30

5.8 **AA** : Typical Design Scenario, Steps 2 and 3 32

5.9 **AA** : Typical Design Scenario, Final Allocation 34

6 VSS : Block Diagram, Human Interface, and Design Scenarios **36**

6.1 **VSS** : Block Diagram 36

6.2 Control Pipelining 38

6.3 Array Variable Clustering 40

6.4 **VSS** : User Interface 42

6.5 Quality Measure Display 44

6.6 XDISP_NL: Netlist display tool 46

6.7 **VSS** : Typical Design Scenario 48

6.8 **VSS** : Typical Design Scenario (2) 50

6.9 **VSS** : Typical Design Scenario (3) 52

7 AA-VSS : Strengths and Weaknesses **54**

7.1 Strengths 54

7.2 Weaknesses 54

8 References **56**

1 Introduction

1.1 Top-Down Design Methodology

AA-VSS is based on a top-down design methodology which is divided into three domains, (a) System-Level Synthesis Domain, (b) Chip-Level Synthesis Domain, and (c) Logic/Layout-Level Synthesis Domain. Since **AA-VSS** performs chip-level synthesis, we concentrate primarily on that aspect of the methodology.

- **System-Level Synthesis Domain** - System-level synthesis tools (such as SpecSyn) partition the system description into hardware and software pieces based on the constraints specified by the designer. Interface synthesis is also performed to ensure that appropriate communication protocols are incorporated into the hardware and software portions. More details of the system-level specification and synthesis process are available in [1].

- **Chip-Level Synthesis Domain** - Each of the chips derived from system-level synthesis must be synthesized into hardware. This is the function of the chip-level synthesis domain. The input to this domain is the behavioral specification of each chip.

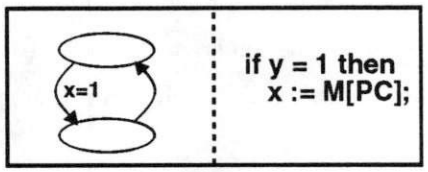
  The important tasks in the chip level synthesis domain are (i) **Allocation** - deriving the appropriate type and number of resources (storage, functional units, and buses) that can satisfy the area and performance constraints imposed by the designer, (ii) **Scheduling** - partitioning the various operations in the behavioral description into states such that all operations in a state are performed during a single clock cycle, and the resources required in each clock cycle do not exceed the resources allocated during the allocation step, and (iii) **Binding** - mapping each of the operations in the behavior to appropriate hardware units - the *arithmetic* and *logic operations* are bound to functional units, the *data storage operations* are bound to appropriate registers or memories, and the *data transfer operations* are bound to buses and muxes in the design.

  The output of the chip-level synthesis domain is a RT-level netlist, containing RT components such as ALUs, registers, register files, memories, and buses. The output netlist is specified using VHDL, which can again be simulated with commercial simulators to verify the correctness of the chip-level synthesis process.

- **Logic/Layout-Level Synthesis Domain** - The RT level netlist that is output from chip-level synthesis can be further synthesized using some of the commercially available logic and layout synthesis tools.

Top Down Design Methodology

Abstract Specification

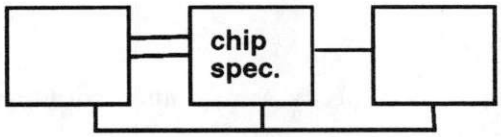


states, protocols, hierarchy, concurrency, programming statements

partitioning, interface synthesis, bus merging, estimation

SpecSyn

Chips

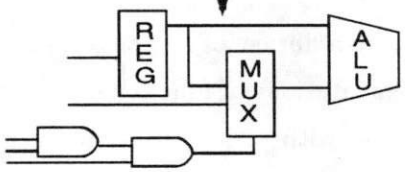


processes, programming statements

VSS

scheduling, allocation, binding

Structure



RTL components, control logic

Layout

Layout, Logic synthesis

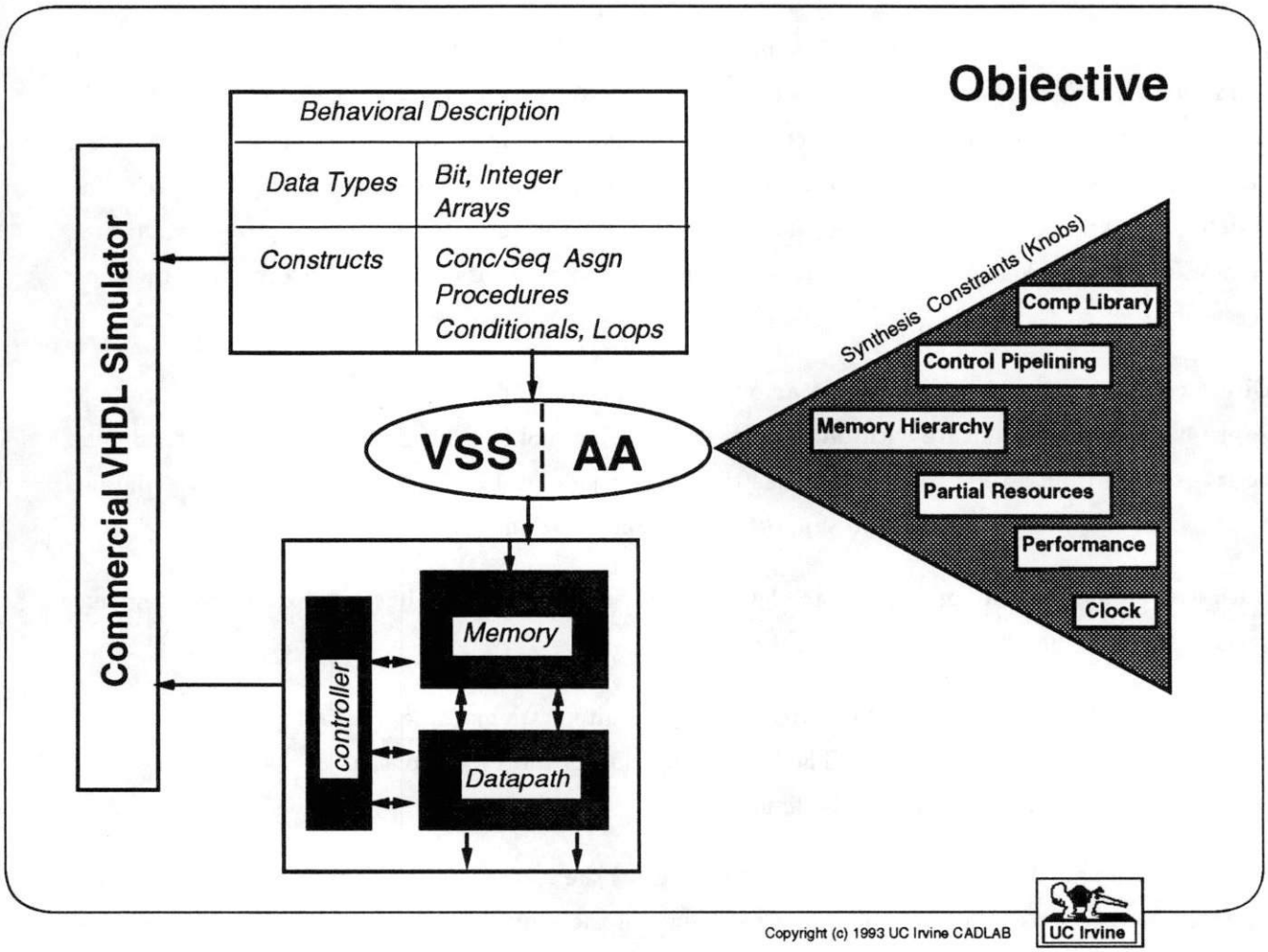
UC Irvine

1.2 Chip-Level Synthesis in AA-VSS

AA-VSS can be used to explore the design space to produce efficient chip designs. The input to the system is a behavioral description of a design written in VHDL. The **AA-VSS** system works on this input description and synthesizes a RT-level netlist, expressed using structural VHDL statements. This enables easy verification of the output since the behavioral input and the structural output can be simulated with a commercially available VHDL simulator using the same set of test vectors.

In order to facilitate the exploration of a large design space, many *knobs* are available on the system. The designer can control the synthesis process by changing the settings on the knobs, which are listed below.

- **Partial Resources** - There are three kinds of resources in the synthesized netlist: (i) functional units such as ALUs and multipliers, (ii) storage units such as RAMs, registers and register files, and (iii) communication units such as muxes and buses. The designer can specify a list of partial resources to be used during synthesis
- **Clock Period** - The designer can select the clock period for the synthesis. This clock period indicates the clock frequency at which the chip will driven and is usually determined by the clock period on the rest of the system.
- **Performance Constraint** - The designer can specify a maximum execution time constraint for the design. The synthesis process will allocate sufficient resources to meet the specified performance constraint.
- **Component Library** - The designer can provide a library of components for the synthesis process. This library could contain multiple implementations of functional units each with different area and delay characteristics, as well as memories with various sizes, access delays, and number of ports.
- **Control Pipelining** - The **AA-VSS** system allows the designer to select among three different control pipelining architectures. The control pipelining architecture determines the pipelining style between the controller and the datapath.
- **Memory Hierarchy** - The storage part of the synthesized netlist may contain many types of storage units (*ie.* registers, register files, and different kinds of memories) arranged in a hierarchical fashion. The designer may specify the number of levels in the memory hierarchy.


Objective
Behavioral Description
Data Types
Bit, Integer
Arrays
Constructs
Conc/Seq Asgn
Procedures
Conditionals, Loops
Commercial VHDL Simulator
VSS
AA
Synthesis Constraints (Knobs)
Comp Library
Control Pipelining
Memory Hierarchy
Partial Resources
Performance
Clock
Memory
controller
Datapath
Copyright (c) 1993 UC Irvine CADLAB
UC Irvine

2 The Chip-Level Synthesis Model

The designs synthesized by the **AA-VSS** system can be represented using the FSMD model of hardware. In the FSMD model, the synthesized design consists of two parts (a) the datapath, which performs the storage of the data and computations on the stored data values, and (b) the controller, which performs the sequencing of various states, and controls the datapath operations.

Datapath - The datapath model consists of functional units, memory components, and buses arranged hierarchically. Functional units are always at the lowest level (level 0) of the hierarchy, while the remaining levels (levels 1 through n) contain the memory components. Memories are arranged in increasing order of access delay. Storage components with small access delays such as registers and register files are at low levels of the hierarchy, while slower components such as large, multi-port memories are at higher levels. We assume that memory components at the same level have the same access delay.

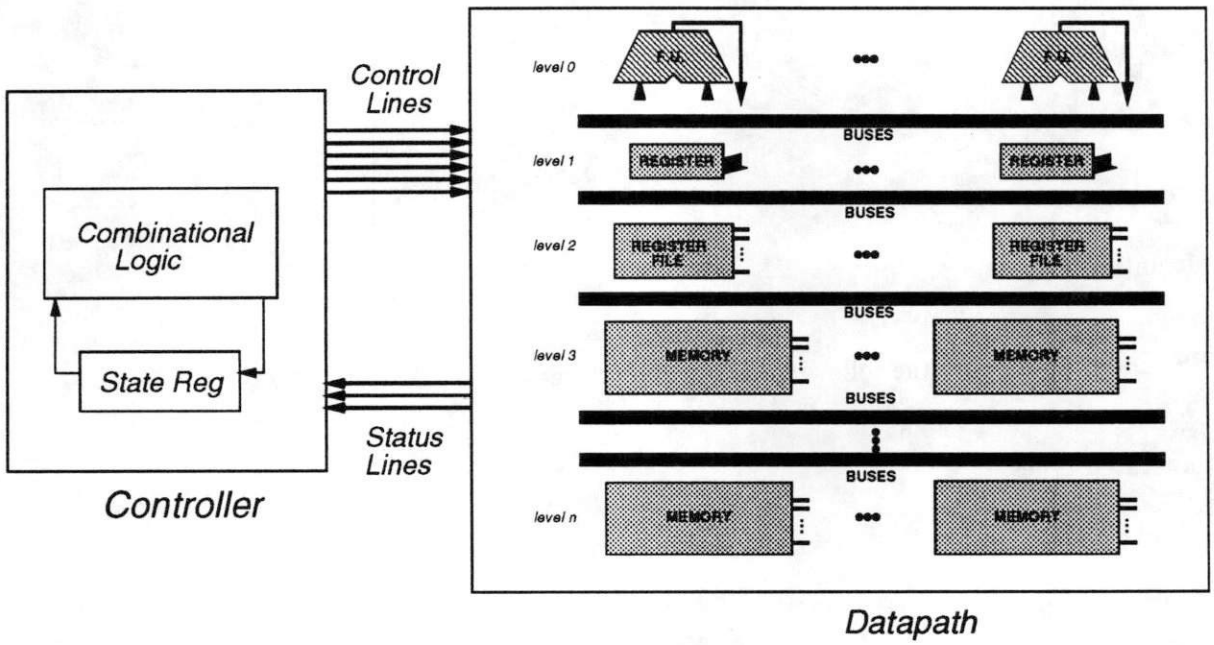
Direct communication is only possible among components at adjacent levels. For instance, units at level i can only communicate with units at levels $(i-1), i$, and $(i+1)$. So, for example, if data must be transferred from level 4 to level 2, it has to pass through level 3. Note that the datapath contains a set of buses between each level to support this communication.

Both the number of components at each level as well as the number of buses between the levels may vary.

Controller - The controller can be represented by a finite state machine, consisting of a set of states and transitions between the states. The implementation of the FSM, consists of a state register and a combinational block as shown in the figure.

The datapath and the controller communicate using (a) the status lines and (b) the control lines. The control lines are set by the controller and the value on these lines determines the operation that will be performed by the datapath. The status lines are set by the datapath to indicate the status of a given computation.

FSMD Model



UC Irvine

3 Chip-Level Synthesis: Input and Tasks

3.1 Input Description

AA-VSS accepts behavioral descriptions in VHDL [2] as input to the system. An example of such a description is shown in in the figure. In general, VHDL descriptions may contain sequential, process-level statements such as (a) sequential assignment statements, (b) conditional statements, (c) loop statements, and (d) wait statements. However, the current implementation of the **AA-VSS** system supports only the following subset of statements.

- Assignment statements
- If statement
- Case statement
- while loops
- wait until statements

AA-VSS does not support the following language features in VHDL.

- enumerated types
- aliases
- CONSTANT declarations
- null statements
- procedures and functions
- exit statements
- return statements
- loop statements with no iteration schemes

Input Description (Behavioral VHDL)

```
entity CompCentroid is
      port ( i , j  : in integer);
             dout   :  out integer);
end CompCentroid;

architecture A of CompCentroid is
begin
    P0: process
        type Memory is array (integer range <>) of integer;
        variable s: Memory;
        variable rs , ls : integer;
    begin
        while (i < j) loop
            if (rs < ls) then
                rs := rs + s(j);      j := j – 1;
            else
                ls := ls + s(i);      i := i + 1;
            end if;
        end loop;
        dout <= j;
    end process P0;
end architecture A;
```


3.2 Allocation

The first step in high-level synthesis is *architecture allocation.* Architecture allocation refers to the process of selecting the functional units, storage elements, and interconnects used to implement the datapath. In addition, the architecture allocation defines the way in which data may be transfered between the datapath functional and storage units. For instance, storage elements may be arranged in a hierarchy such that data flows from register files to functional units and from memories to register files but not from memories to functional units.

More specifically, an architecture allocation consists of the following information.

1. **Level Allocation:** This refers to grouping the storage elements into different levels (thereby imposing a memory hierarchy) such that data may only be transfered between memories at adjacent levels and between functional units and the first level of memory.

2. **Storage Allocation:** This defines the number and types of memories used in the datapath. A storage allocation specifies (1) the number of memories, (2) the delay per memory, and (3) the number of ports and size per memory.

3. **Functional Unit Allocation:** A FU allocation specifies (1) the types of FUs used (where type is defined by the functions performed and the bitwidth), (2) the number of FUs of each type, and (3) the delay and data initiation rate of each FU.

4. **Interconnect Allocation:** An interconnect allocation defines (1) the number of buses in the design, and (2) the delay per bus.

A sample design description and a corresponding architecture allocation are shown in the figure. It is assumed that there is only one level of memory hierarchy; so, data may be transferred between any two storage elements or between any storage element and any FU.

Design Description and Allocation

High–Level Design Description

```
while (i < j) loop
    if (rs < ls) then
        rs := rs + s(j);     j := j – 1;
    else
        ls := ls + s(i);     i := i + 1;
    end if;
end loop;
dout <= j;
```

Datapath Unit Allocation

unit	no.	delay	ports	size	stages	functions
ALU	1	20 ns	–	16 bit	1	add, sub, cmp
BUS	3	5 ns	–	16 bit	–	–
REG	7	5 ns	1r, 1w	16 bit	–	read, write
MEM	1	15 ns	1rw	1K w X 16 bit	1	read, write

3.3 Scheduling

Scheduling partitions all the operations in the CDFG into different subgraphs such that each subgraph is executed in one control step. However, the scheduling process must ensure that sufficient resources are available in each clock cycle to execute all the operations assigned to that control step. If more functional units, memory ports and buses are available, then it is possible to assign more operations to a given control step, thereby reducing the total number of clock steps required to execute the design. On the other hand, if fewer resources are allocated, then the scheduling process can only assign fewer operations in each clock cycle, thereby increasing the total number of steps required to complete the schedule. The main goal of the scheduling is to maximize the utilization of the allocated resources.

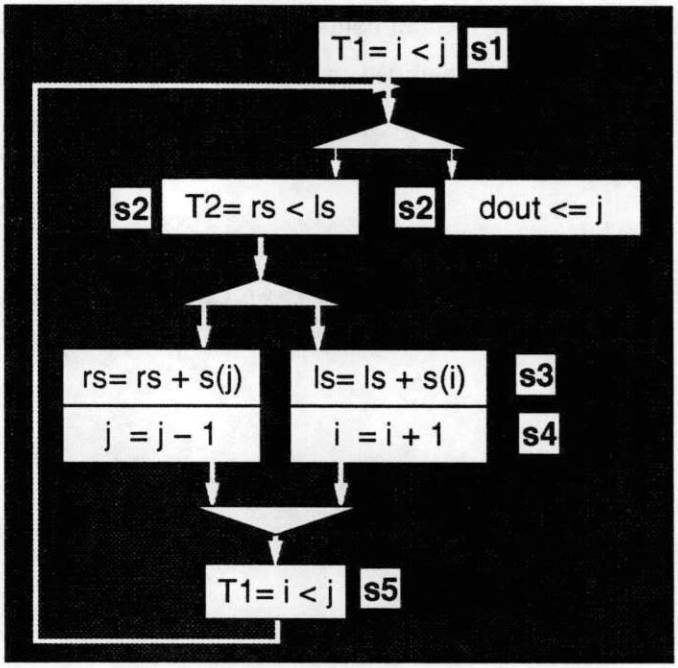
The diagram shows the results of the scheduling process on the input description from Section 3.1. The allocation constraints provided to the scheduler are shown on the left part of the figure. Since a single ALU was allocated, it was only possible to schedule one add or subtract operation every clock cycle. Five states (s1, s2 .. s5) were required to schedule the design with the allocated resources. The state assignments for each of the operations are shown in the figure. However, two mutually exclusive operations can be scheduled on the same state, since by definition only one of them would get executed during any given instance. For example the operations (i := i + 1) and (j := j -1) are scheduled into the same clock cycle (i.e., s4) because they are mutually exclusive.

VSS uses a list based scheduling algorithm [3] for scheduling the operations in a basic block.

Scheduling (with Resource Constraints)

Resource Constraints

unit	no.	delay
ALU	1	20 ns
BUS	3	5 ns
REG	7	5 ns
MEM	1	15 ns



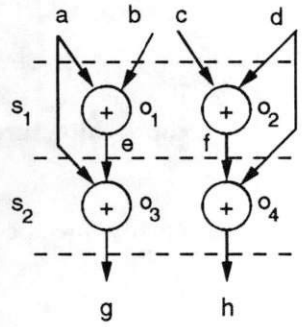
UC Irvine

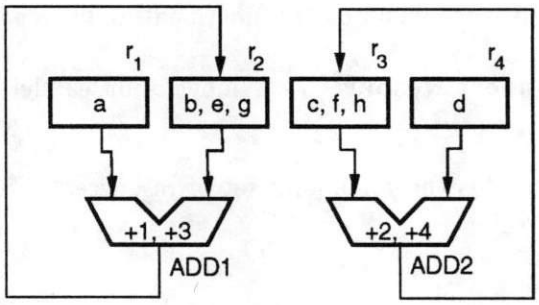
3.4 Binding

Binding maps the variables and operations in the scheduled CDFG onto specific instances of the allocated functional units, storage components, and interconnect units while ensuring that the design behavior operates correctly. Suppose we are given an allocation of two adders and four registers and the schedule shown in the left-hand side of the figure. Operations o_1 and o_2 cannot be mapped to the same adder since they are scheduled in the same control step. On the other hand, operations o_1 and o_3 can share an adder because they and executed during different control steps. Similarly, variables whose values are needed concurrently may not share the same register.

Note that there are several different ways of performing binding. For example, we can map o_2 and o_3 to ADD1 and o_1 and o_4 to ADD2. Or, we can map o_1 and o_3 to ADD1 and o_2 and o_4 to ADD2. The best binding is the one which minimizes design quality metrics such as area, delay, and power dissipation.

Binding





4 AA-VSS : I/O Specifications

The designer, the architecture allocator (**AA**), and the synthesizer (**VSS**) interact as shown in the diagram.

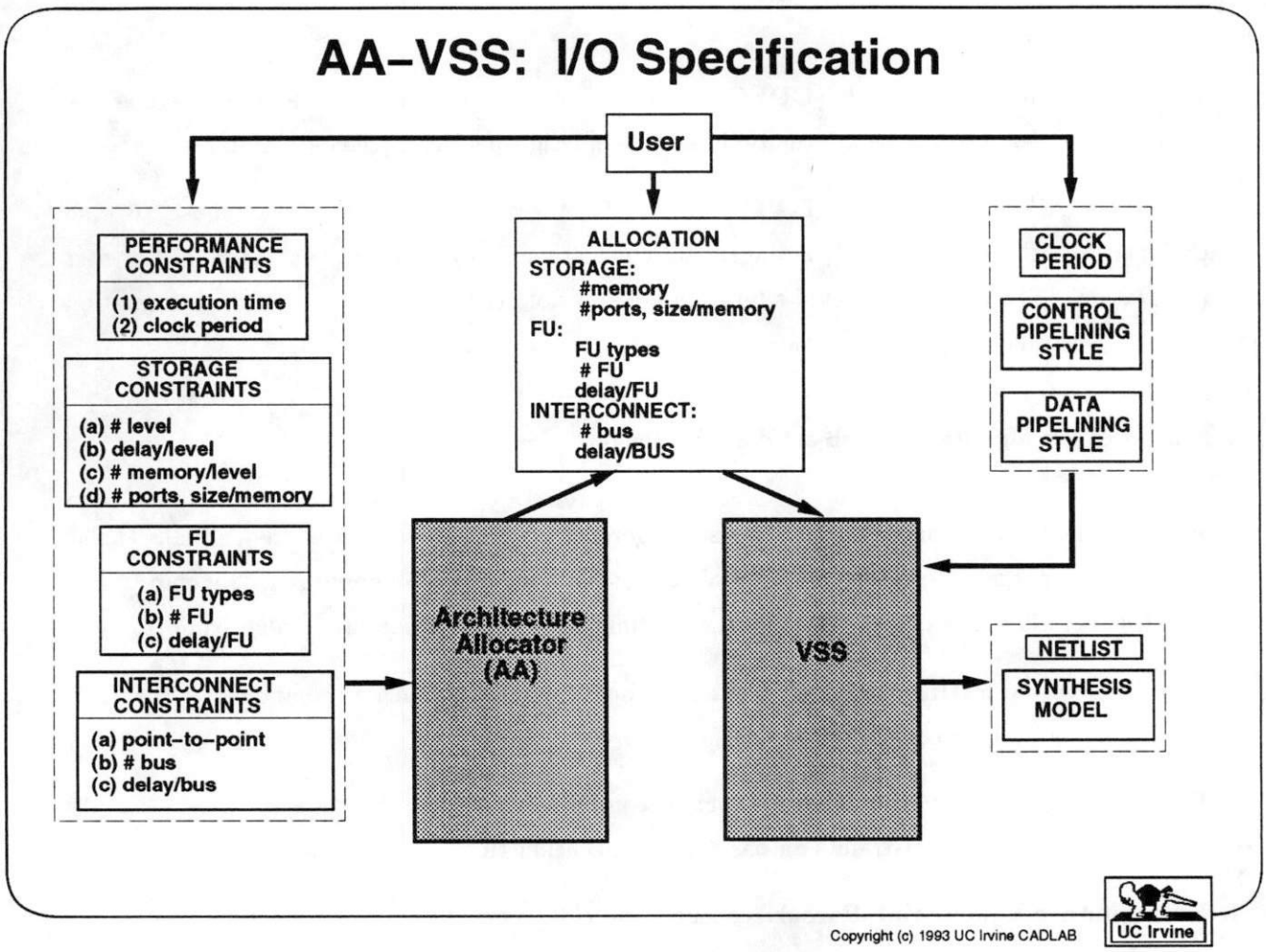
AA selects the optimal mix of library components (not pictured) used during the synthesis process. The designer interacts with **AA** primarily by constraining some or all of the following design parameters.

- **Performance** - clock period, execution time
- **Storage Resources** - number of levels of memory hierarchy, number of memory modules in each level, delay of memories at each level, size of each memory module, number of ports in each memory module
- **FU Resources** - number of functional units, type of each functional unit, delay of each unit
- **Interconnect Resources** - number of buses, delay per bus, point-to-point architecture

AA then computes the remaining set of resources to 'match' the resources already specified by the designer.

The allocation provided by **AA** is then fed into the **VSS** system which uses the resource constraints to synthesize the design. In addition, the designer interacts with **VSS** by specifying the following synthesis parameters.

- **Clock Period** - **VSS** allows the clock period to be set to any value. However the resultant design may look entirely different for two different clock period settings.
- **Control Pipelining Style** - **VSS** can synthesize designs with three different control pipelining (CP) styles. The control pipelining style determines how the microactions in each state are parallelized.
- **Datapath Pipelining Style** - **VSS** can synthesize designs with pipelined functional units.


AA–VSS: I/O Specification
User
PERFORMANCE CONSTRAINTS
(1) execution time
(2) clock period
STORAGE CONSTRAINTS
(a) # level
(b) delay/level
(c) # memory/level
(d) # ports, size/memory
FU CONSTRAINTS
(a) FU types
(b) # FU
(c) delay/FU
INTERCONNECT CONSTRAINTS
(a) point-to-point
(b) # bus
(c) delay/bus
ALLOCATION
STORAGE:
#memory
#ports, size/memory
FU:
FU types
FU
delay/FU
INTERCONNECT:
bus
delay/BUS
CLOCK PERIOD
CONTROL PIPELINING STYLE
DATA PIPELINING STYLE
Architecture Allocator (AA)
VSS
NETLIST
SYNTHESIS MODEL
Copyright (c) 1993 UC Irvine CADLAB
UC Irvine

5 AA : Block Diagram, Human Interface, and Design Scenarios

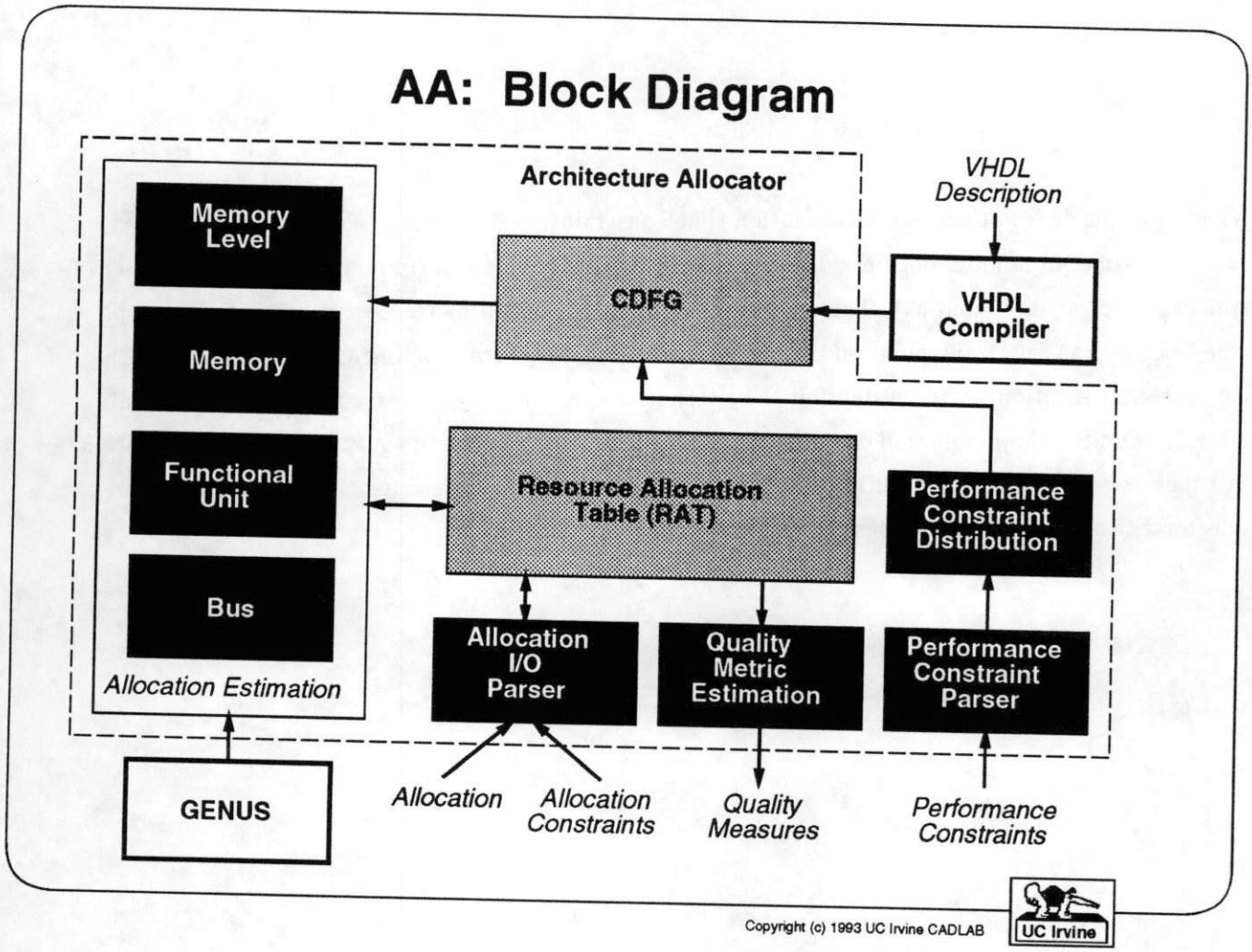
5.1 AA : Block Diagram

AA works off of two basic data structures, the control data flow graph and the resource allocation table.

1. **Control Data Flow Graph (CDFG):** indicates control and data dependencies among the operations and also contains information about the design performance constraints.
2. **Resource Allocation Table (RAT):** contains (1) the allocation constraints imposed by the designer, (2) the expected FU, storage, interconnect, and level allocations, and (3) the cost (utilization/dollar) of the expected allocation. In addition, the RAT stores all of the information needed to estimate quality metrics.

AA consists of five algorithms which are described below.

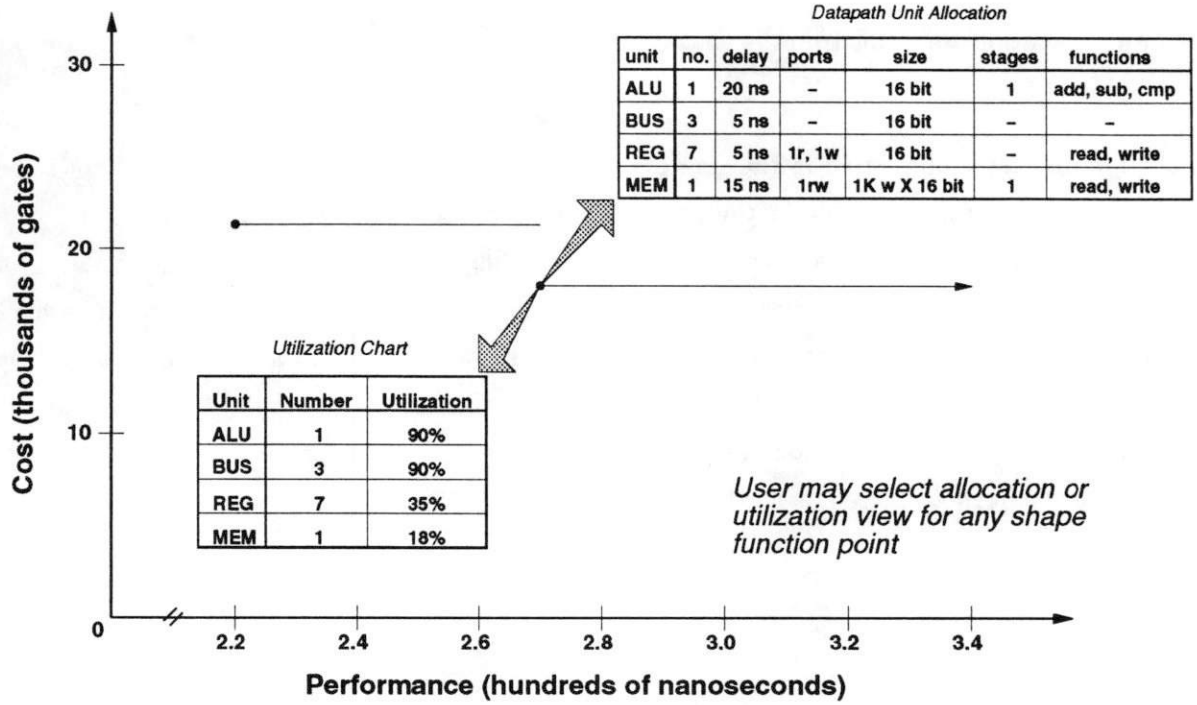
1. **Allocation Estimator:** contains two sub-algorithms which are executed sequentially. The first sub-algorithm determines the level allocation and the second computes FU, memory, and interconnect allocations. The **AA** allocation estimator is an extension of the algorithm from [4].
2. **Quality Metric Estimator:** computes design quality measures such as component utilization, total area, and power consumption.
3. **Performance Constraint Distributor:** decomposes the execution time constraint for the behavior into smaller constraints on basic block execution times.
4. **Performance Constraint Parser:** captures the clock period and execution time constraints specified by the designer and generates the performance constraint input display.
5. **Allocation I/O Parser:** produces the allocation displays viewed by the designer. Note that the same display is used for both capturing allocation constraints and displaying allocation output.


AA: Block Diagram
Architecture Allocator
VHDL Description
Memory Level
Memory
Functional Unit
Bus
Allocation Estimation
CDFG
VHDL Compiler
Resource Allocation Table (RAT)
Performance Constraint Distribution
Allocation I/O Parser
Quality Metric Estimation
Performance Constraint Parser
GENUS
Allocation
Allocation Constraints
Quality Measures
Performance Constraints
Copyright (c) 1993 UC Irvine CADLAB
UC Irvine

5.2 Shape Function Display

If the designer does not provide an execution time constraint, **AA** outputs a shape function displaying execution time (in number of clocks or nanoseconds) versus design cost (in number of gates, number of transistors, or square microns). The diagram illustrates the performance/cost shape function display. The designer can select any point on the function and view the corresponding architecture allocation or component utilization. After evaluating the alternatives, he/she may choose an architecture allocation directly from the shape function or refine the design manually by placing constraints on the allocation. In other words, the designer can use the shape function display to determine a final allocation or as a feedback tool to 'suggest' design alternatives.

Shape Function Display



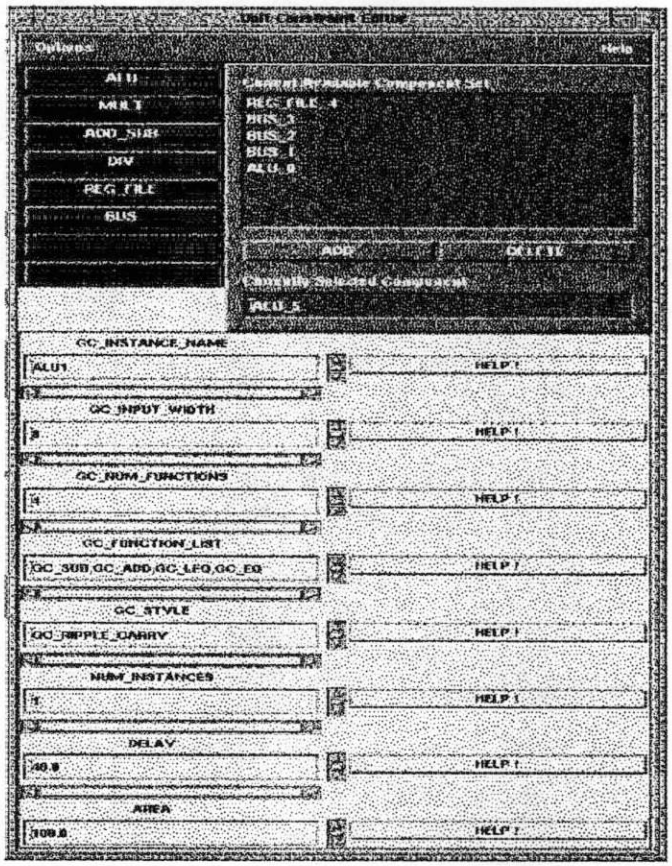
UC Irvine

5.3 Allocation Display

The allocation display serves two purposes: (1) to capture allocation constraints, and (2) to display the allocation output of **AA** . It should be noted that the allocation display is used to depict output only when an execution time constraint is placed on the design; otherwise, the shape function format is used.

The allocation display is illustrated in the diagram. The menu on the upper left-hand side shows the component types currently available in the library, while the display on the upper right-hand side shows the current allocation constraints or allocation depending on whether the user is looking at an input or output display. To input a new allocation constraint, the designer must select a component type from the library and fill out the information display shown on the lower left-hand side of the diagram. He/she then clicks on the ADD button to add the constraint. It should be noted that the allocation output (or the allocation constraints) can be dumped to a text file for viewing by the user. The text file output format is used as the allocation input to **VSS** .

Resource Allocation Tool: User Interface



UC Irvine

5.4 Quality Measure Display

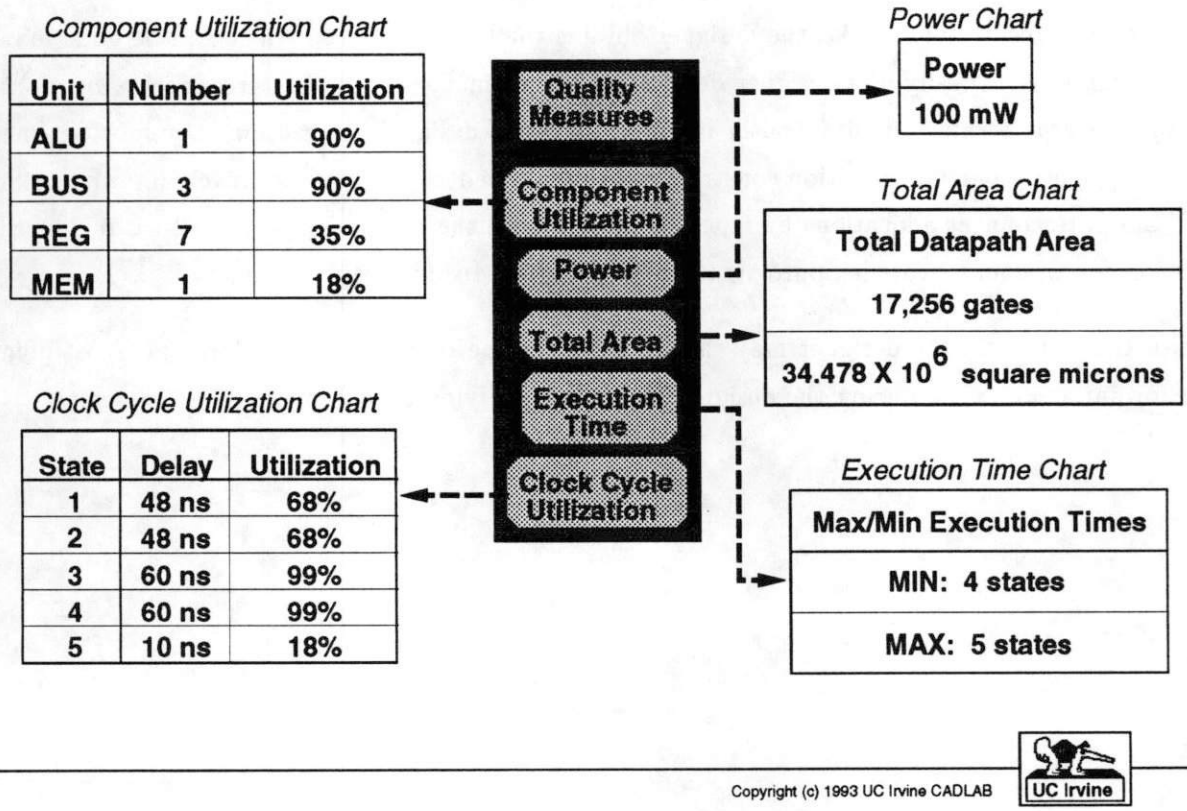
AA estimates several design quality measures including component utilization, total area, and power consumption. The quality measure display consists of a pull-down menu from which the designer may select the metric that he/she wishes to view. The three **AA** quality metrics are described briefly below. (The remaining two metrics shown in the figure are for **VSS** .)

1. **Component Utilization:** The utilization for a component instance is the amount of time that the component is used divided by the total execution time. Since **AA** does neither scheduling nor binding, the utilization value cannot be derived exactly. Instead, it must be *estimated.* Furthermore, **AA** computes utilization per *component type* instead of per *component instance* due to the fact that an exact binding is not available.

2. **Total Area:** In **AA** , total area refers to the sum of the functional unit and memory areas for the datapath components. Note that interconnect area is not included since a floorplan is not available to **AA** , and controller area is not included since **AA** does not perform control synthesis.

3. **Power Consumption:** The power consumption metric is estimated by summing an average power consumption value (determined empirically) for the library components multiplied by the expected switching frequency of the component.

Design Quality Measures



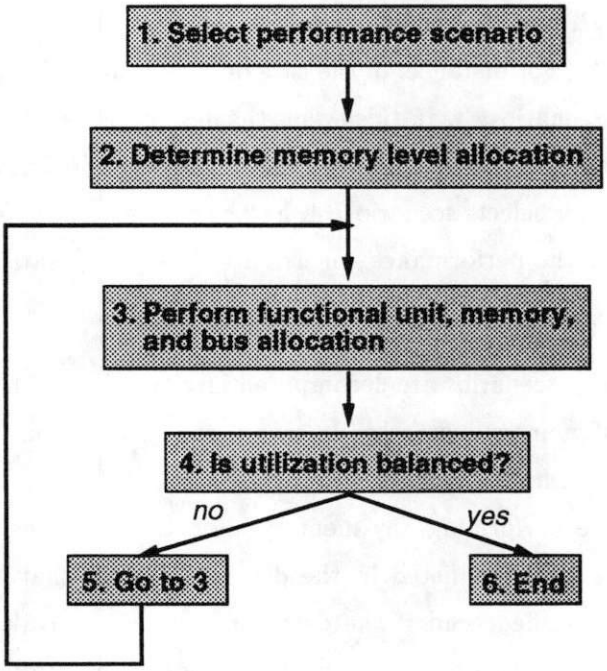
UC Irvine

5.5 AA : Design Methodology

A flow chart of the **AA** design methodology is shown in the figure. The first step is to select a performance scenario. This determines whether the **AA** output in subsequent steps is a shape function or a performance/cost point. Next, the designer obtains the level allocation which can be done manually or using **AA** . Thirdly, the designer determines the memory, FU, and interconnect allocations. These tasks are performed simultaneously in **AA** ; however, designer interaction is supported since the user may specify partial allocation constraints. Finally, the designer may iteratively improve memory, FU and interconnect allocations by repeating step 3 until the component utilization is 'balanced' (utilization for all components is approximately the same) and sufficiently high.

Note also that, in **AA** , the designer may change from a shape function to a performance/cost point output format at any time during the design process by specifying an execution time constraint.

AA Design Methodology



UC Irvine

5.6 AA : Possible I/O Scenarios

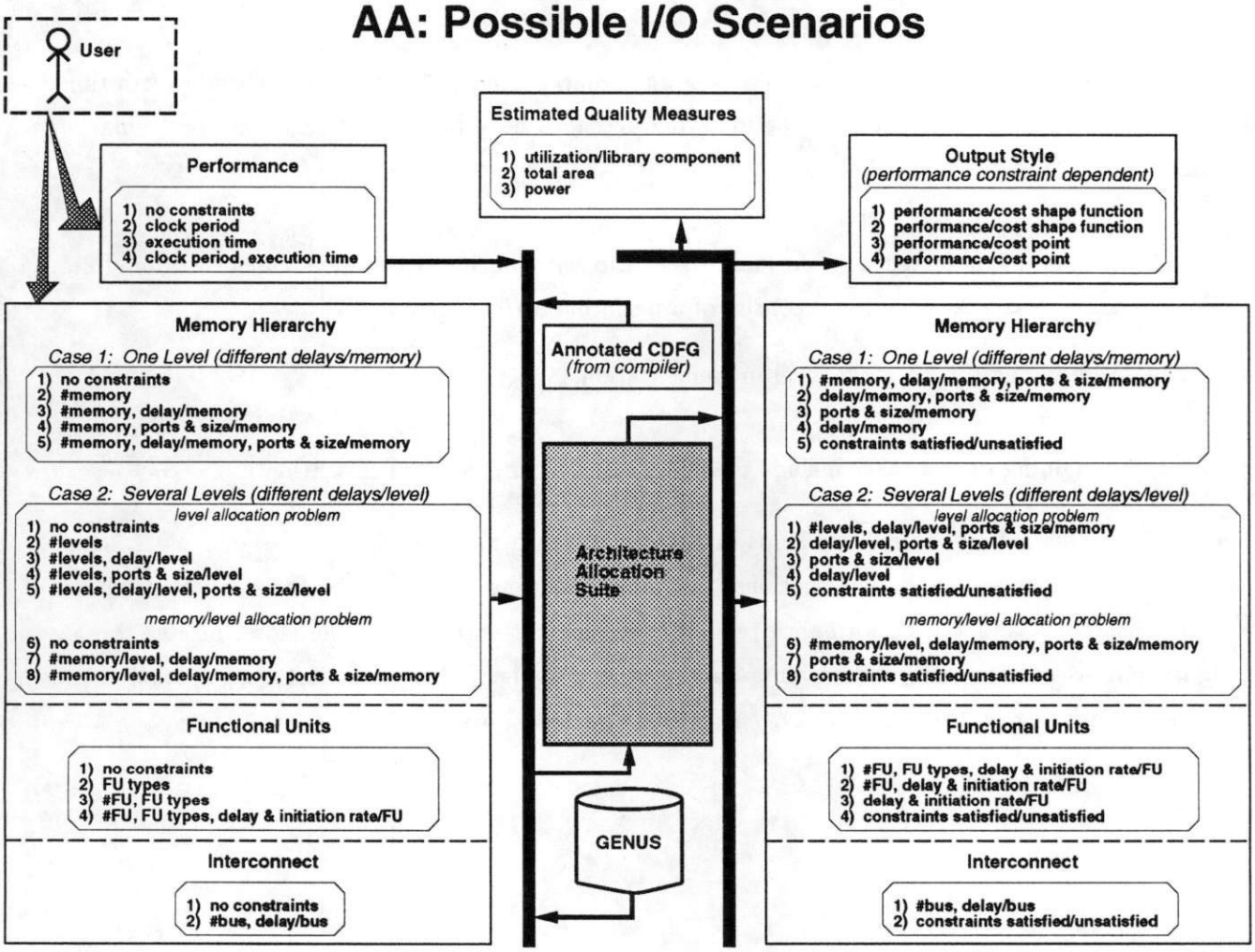
All of the possible input/output scenarios for the architecture allocator are listed in the diagram. Input scenarios are shown on the left-hand side, and the corresponding output scenarios are listed on the right-hand side. To use **AA** , the designer must select performance, memory, functional unit, and interconnect input scenarios from the alternatives listed in the diagram.

The performance constraint scenario determines whether the output style is a performance/cost shape function or a performance/cost point. Specifically, if an execution time is given, then the output is a point; otherwise, it is a shape function.

Allocation scenarios (memory, FU, and interconnect) define a partial allocation, and the remaining parameters are determined by **AA** . For instance, in the case of FUs, four input scenarios are possible. Suppose that the designer selects scenario #3. In this event, the designer must specify both the number of functional units allocated and their types, while **AA** determines only the delay and initiation rates of the units. However, if the designer selects scenario #4, he/she specifies the complete FU allocation, and **AA** only determines whether the performance constraints (if any) are satisfiable with the given FU allocation.

It should be noted that the memory scenarios are decomposed into two cases: (1) one level of memory hierarchy, and (2) several levels of memory hierarchy. In case 1, the designer may select any FU, memory, and interconnect scenarios that he/she wishes. However, in case 2, the designer must determine a complete level allocation **before** performing any memory allocation. In other words, the designer must first select one of the level scenarios listed in the diagram. (Note that FU and interconnect allocations must be completely specified, and if the designer does not provide them, **AA** will use default allocations.) Then, after deciding on a level allocation, he/she determines memory, FU, and interconnect allocations by selecting appropriate scenarios as in case 1.

Finally, **AA** also estimates several design quality measures which may be viewed by the user at any time.


AA: Possible I/O Scenarios
User
Performance
1) no constraints
2) clock period
3) execution time
4) clock period, execution time
Estimated Quality Measures
1) utilization/library component
2) total area
3) power
Output Style
(performance constraint dependent)
1) performance/cost shape function
2) performance/cost shape function
3) performance/cost point
4) performance/cost point
Memory Hierarchy
Case 1: One Level (different delays/memory)
1) no constraints
2) #memory
3) #memory, delay/memory
4) #memory, ports & size/memory
5) #memory, delay/memory, ports & size/memory
Case 2: Several Levels (different delays/level)
level allocation problem
1) no constraints
2) #levels
3) #levels, delay/level
4) #levels, ports & size/level
5) #levels, delay/level, ports & size/level
memory/level allocation problem
6) no constraints
7) #memory/level, delay/memory
8) #memory/level, delay/memory, ports & size/memory
Functional Units
1) no constraints
2) FU types
3) #FU, FU types
4) #FU, FU types, delay & initiation rate/FU
Interconnect
1) no constraints
2) #bus, delay/bus
Annotated CDFG
(from compiler)
Architecture
Allocation
Suite
GENUS
Memory Hierarchy
Case 1: One Level (different delays/memory)
1) #memory, delay/memory, ports & size/memory
2) delay/memory, ports & size/memory
3) ports & size/memory
4) delay/memory
5) constraints satisfied/unsatisfied
Case 2: Several Levels (different delays/level)
level allocation problem
1) #levels, delay/level, ports & size/memory
2) delay/level, ports & size/level
3) ports & size/level
4) delay/level
5) constraints satisfied/unsatisfied
memory/level allocation problem
6) #memory/level, delay/memory, ports & size/memory
7) ports & size/memory
8) constraints satisfied/unsatisfied
Functional Units
1) #FU, FU types, delay & initiation rate/FU
2) #FU, delay & initiation rate/FU
3) delay & initiation rate/FU
4) constraints satisfied/unsatisfied
Interconnect
1) #bus, delay/bus
2) constraints satisfied/unsatisfied

5.7 AA : Typical Design Scenario, Steps 1 and 2

Section 5.7 and Section 5.8 describe a typical design scenario using **AA** for the behavioral description listed in Section 3.1. We assume that the following decisions have been made regarding performance scenarios and level allocation.

1. The designer has selected a performance scenario with both clock period and execution time constraints. So, **AA** output will consist of a performance/cost point.
2. The designer has chosen a one-level implementation.

Hence, the remaining design tasks include memory, FU, and interconnect allocation.

The diagram illustrates the first two steps in the design scenario. Initially, the designer places constraints on the memory and FU allocations, but after executing **AA** , he/she finds that the design is over-constrained and the performance requirements are not satisfiable. Therefore, in step 2, the designer removes the memory constraints and runs **AA** again. This time, **AA** responds with an allocation consisting of 2 ALUs, 6 buses, 7 registers, and 1 memory.

Typical Design Scenario

(1) Designer provides initial allocation constraints

Performance:
clock: 60ns
execution time: 5 clock cycles
Memory Levels:
one memory level
Memory:
number, delay: 6, 5 ns
number, delay: 1, 15 ns
Functional Unit:
number, type: 2 ALU
Interconnect:
no constraints

AA

***Design is overconstrained – cannot satisfy performance requirements**

(2) Designer removes memory constraints and estimates allocation

Memory:
no constraints

AA

Datapath Unit Allocation and Utilization

Unit	Number	Utilization
ALU	2	46%
BUS	6	46%
REG	7	35%
MEM	1	18%

UC Irvine

5.8 AA : Typical Design Scenario, Steps 2 and 3

The figure shows steps 2 and 3 of the design scenario. Recall that, at the end of step 2, **AA** had computed memory, FU, and interconnect allocations, but the component utilizations were very low. In step 3, the designer attempts to improve utilization by reducing the number of ALUs from 2 to 1. He/she then runs **AA** and obtains a new allocation with 1 ALU, 3 buses, 7 registers, and 1 memory. Note that the ALU and bus utilizations have both improved from 46 % to 90 %. As a result, the designer is satisfied and proceeds with scheduling and binding.

Typical Design Scenario

(2) Designer removes memory constraints and estimates allocation

Performance:
clock: 60ns
execution time: 5 clock cycles
Memory Levels:
one memory level
Memory:
no constraints
Functional Unit:
number, type: 2 ALU
Interconnect:
no constraints

AA

Datapath Unit Allocation and Utilization

Unit	Number	Utilization
ALU	2	46%
BUS	6	46%
REG	7	35%
MEM	1	18%

(3) Designer observes low utilization and reduces FU allocation

Functional Unit:
number, type: 1 ALU

AA

ALU	1	90%
BUS	3	90%

UC Irvine

5.9 AA : Typical Design Scenario, Final Allocation

The diagram shows the final allocation from the **AA** design scenario. The details of each component, such as delay, ALU functions, memory ports and sizes, etc . . ., are listed in the diagram. For example, the allocation contains 1 16-bit, non-pipelined ALU. The ALU has a 20 *ns* delay and is capable of performing the functions add, subtract, and compare. When the designer has approved the final allocation, it is fed into a text file and used as a resource constraint for scheduling and binding in **VSS** .

Typical Design Scenario

FINAL ALLOCATION

unit	no.	delay	ports	size	stages	functions
ALU	1	20 ns	–	16 bit	1	add, sub, cmp
BUS	3	5 ns	–	16 bit	–	–
REG	7	5 ns	1r, 1w	16 bit	–	read, write
MEM	1	15 ns	1rw	1K w X 16 bit	1	read, write

6 VSS : Block Diagram, Human Interface, and Design Scenarios

6.1 VSS : Block Diagram

The **VSS** system is composed of a set of modules. The input to **VSS** is a set of resource constraints, and the output is a simulatable VHDL netlist at the RT level and the quality measures for the netlist. **VSS** interfaces to the GENUS library to get the necessary components when synthesizing the netlist.

The block diagram indicating all the **VSS** modules is shown in the figure. These blocks are:

1. **VHDL Compiler** compiles the VHDL description into an internal control data flow graph (CDFG).

2. **Graph Critic** applies transformations on the CDFG to remove VHDL specific portions in the flowgraph to more hardware oriented attributes. For example the CDFG for the VHDL statement *if (not(clk'stable) and (clock = 1))* is transformed to imply a rising clock edge.

3. **Scheduler** partitions the CDFG into control steps (Section 3.3).

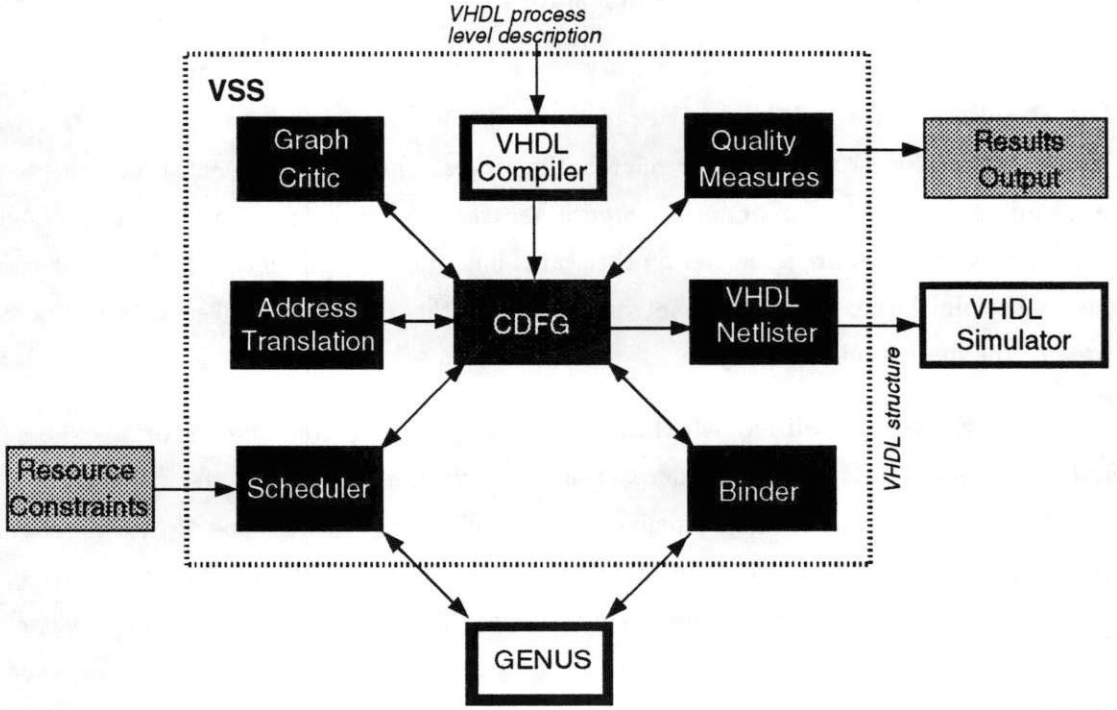
4. **Binder** determines a mapping between the operations in the flowgraph to the allocated hardware components. (Section 3.4).

5. **VHDL Netlister** generates the output VHDL netlist and creates necessary hooks for simulating the design with a commercial VHDL simulator.

6. **Quality Estimator** estimates the quality measures shown in Section 5.4.

7. **Address Translation** performs the required address translations, when several array variables share the same memory module.

VSS : Block Diagram



UC Irvine

6.2 Control Pipelining

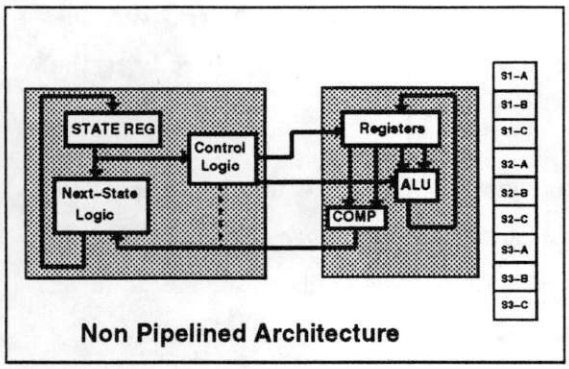
In an FSMD model, the design performs three microactions during a state: (1) set control lines, (2) perform datapath operations, and (3) compute next state value. It is possible to execute these three micro-actions in a pipelined fashion. We refer to this pipelining methodology as *control pipelining* since we are actually pipelining the micro-actions in a state. **VSS** supports three different control pipelining methodologies [5].

1. **Non-Pipelined Methodology:** In this model, the three microactions in a state are performed serially. Since all the three microactions in a state are executed sequentially, the clock period is quite large.

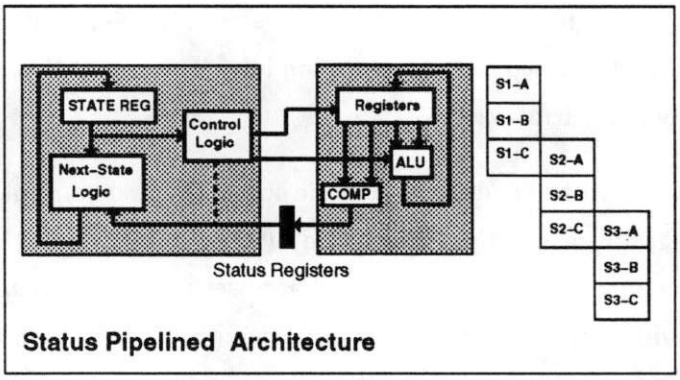
2. **Status Pipelined Methodology:** We can think of this model as a two stage pipeline, where the first stage performs the first two microactions (*set_control* and *exec_dp*) and the second stage performs the third microaction (*compute_next_state*). In order to achieve this pipelined performance, we now require a register on the status lines. The length of the clock cycle decreases because of the pipelining; however, the introduction of a status register on all status lines increases the area of the design substantially.

3. **Control and Status Pipelined Methodology:** In this model, the number of pipeline stages is further increased. All the three microactions are executed in a pipelined fashion. We can think of this model as a three-stage pipeline, where each stage performs one of the microactions, *set_control*, *exec_dp* and *next_state*. In order to implement this pipelining style, we have to now introduce registers on both the status and control lines. The introduction of pipeline registers on all status and control signals further decreases the clock period of this architecture; however, additional no-op states may be required during scheduling to accommodate these pipeline registers. The area penalty is also quite high.

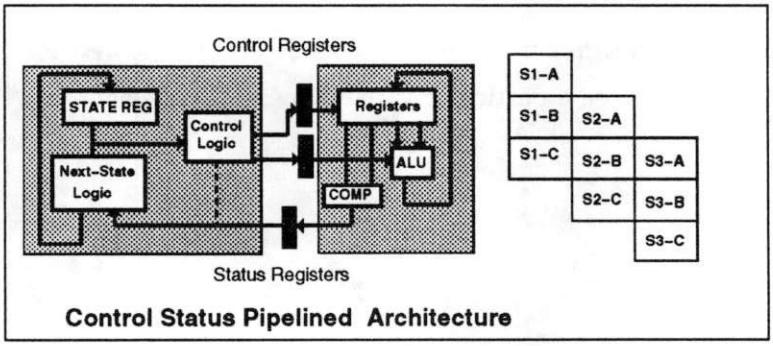
Control Pipelining



Non Pipelined Architecture



Status Pipelined Architecture



Control Status Pipelined Architecture

UC Irvine

6.3 Array Variable Clustering

By using *array variable clustering* techniques it is possible to store more than one array variable in a given memory module [6].

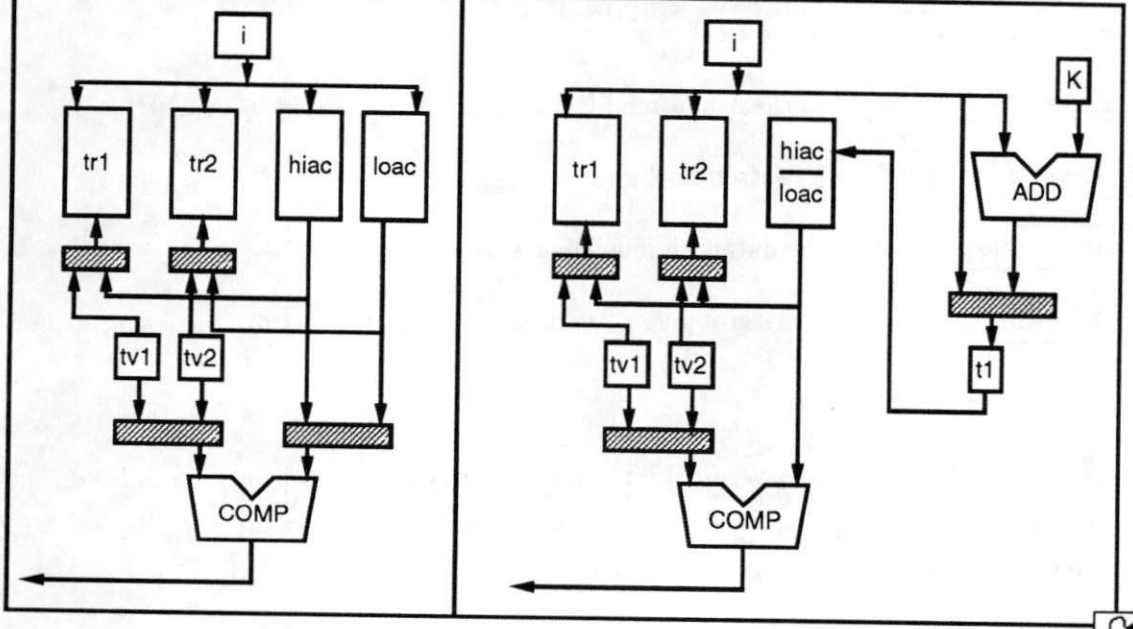
In the figure, the variables $tr1$, $tr2$, *hiac* and *loac* are array variables. The results of synthesis using a simple memory allocation scheme is shown in the left-hand side of the figure. Here, each array variable is stored in a separate memory module. The size of these memory modules correspond directly to the size of the array variables.

On the other hand, it is possible for both array variables to share the same memory module. This is shown on the right-hand side of the figure, where the variables *hiac* and *loac* are stored in the same memory module. Now, all accesses to variable *loac* require an address translation. An adder is required for this purpose. If an adder has already been allocated for doing datapath operations, the address translation may also be done using the same adder.

VSS uses a memory merging algorithm called MeSA [6] to determine the best merging of variables in the design. The user can also impose a particular variable merging scheme.

Array Variable Clustering

```
begin
    if (tv1 < hiac(i))  then  tr1(i) = tv1;  else  tr1(i) = hiac(i);  endif;
    if (tv2 < loac(i))  then  tr2(i) = tv2;  else  tr2(i) = loac(i);  endif;
end
```



UC Irvine

6.4 VSS : User Interface

The **VSS** User Interface (UI) provides users with a simple and elegant mechanism to control the synthesis process. The UI displays a top level view of the design to the user. This top level view shows the individual blocks and processes and their interconnections with global signals.

Using this top level view, designers can select individual blocks or processes to be synthesized. The various options available during synthesis can be specified interactively. Some of the commands available on the UI include:

[a] **Constraints ⇒ Clock** to set the clock period

[b] **Constraints ⇒ Resource** to check and modify the resources allocated by AA.

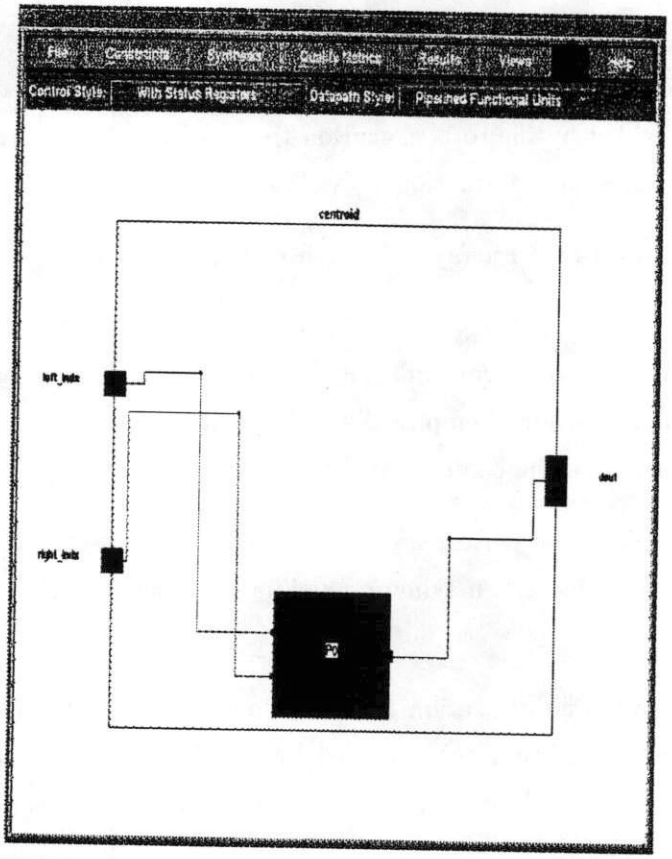
[c] **Control Style** to specify the control pipelining style.

[d] **Datapath Style** to specify the datapath pipelining stages.

[e] **Array Grouping** to specify the grouping of variables that is to be stored in each memory module.

The interface also provides buttons to verify the results of synthesis. The quality measure display consists of a pull down menu from which the designer may select the metric that he/she wishes to view. The five quality metrics are described in Section 6.5.

VSS : User Interface



UC Irvine

6.5 Quality Measure Display

VSS estimates several design quality measures including component utilization, total area, power consumption, execution time, and clock cycle utilization. The quality measure display consists of a pull down menu from which the designer may select the metric that he/she wishes to view. The five quality metrics are described briefly below.

1. **Component Utilization:** The utilization for a component instance is the amount of time that the component is used divided by the total execution time. **VSS** can compute this exactly since the exact scheduling and binding information is available after synthesis.

2. **Total Area:** In **VSS** , total area refers to the sum of functional unit, memory, interconnect, and controller area.

3. **Power Consumption:** The power consumption metric is estimated by summing an average power consumption value (determined empirically) for the library components multiplied by the expected switching frequency of the component.

4. **Performance:** **VSS** displays the performance of the synthesized design in terms of the clock period and the minimum possible and maximum possible execution times for the design with the given set of resources.

5. **Clock Cycle Utilization:** The utilization for each clock cycle is defined as the critical path delay for the events scheduled in that cycle divided by the clock period. Clock cycle utilization is also referred to as *clock slack*. More information about clock slack is available in [7].

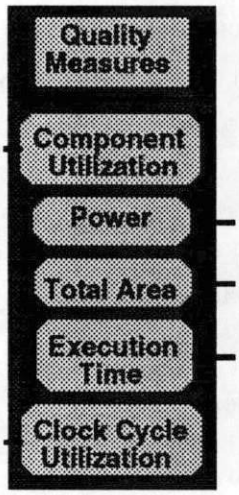
DESIGN QUALITY MEASURES

Component Utilization Chart

Unit	Number	Utilization
ALU	1	90%
BUS	3	90%
REG	7	35%
MEM	1	18%

Clock Cycle Utilization Chart

State	Delay	Utilization
1	48 ns	68%
2	48 ns	68%
3	60 ns	99%
4	60 ns	99%
5	10 ns	18%



Power Chart

Power
100 mW

Total Area Chart

Total Datapath Area
17,256 gates
34.478 X 10^6 square microns

Execution Time Chart

Max/Min Execution Times
MIN: 4 states
MAX: 5 states

UC Irvine

6.6 XDISP_NL: Netlist display tool

One of the important display tools that has been developed along with **VSS** is the netlist display tool **xdisp_nl**. This tool accepts the VHDL files generated by **VSS** and displays the schematics. All components are shown as boxes, but each component type appears in a user-definable color, making it easy to identify important components.

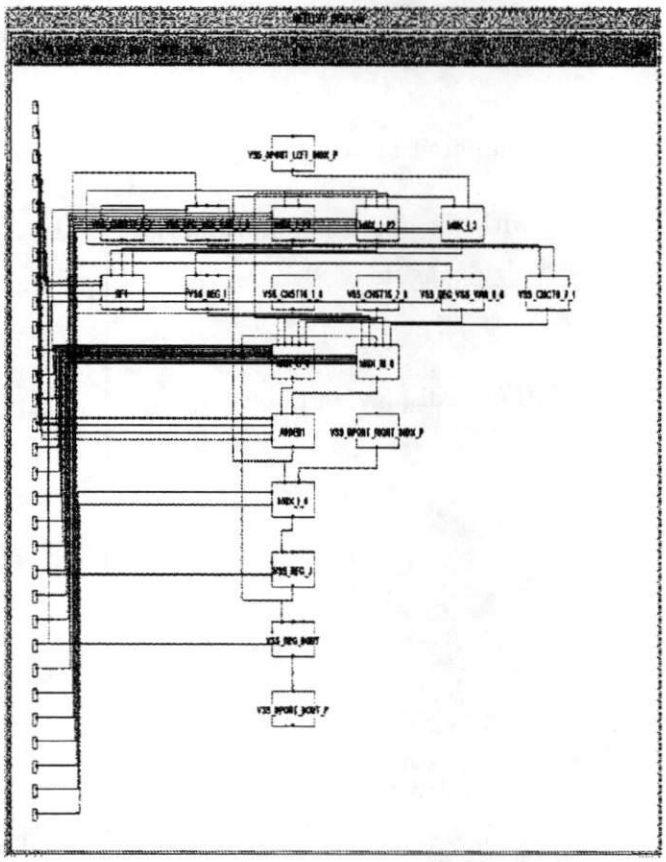
The display tool provides many interactive features that facilitate the examination of the netlist by the designer. Some of the important capabilities of the display tool include

[a] Zooming in and out, to examine portions of the netlist

[b] Turning off the control lines display to examine the datapath.

[c] Highlighting the sources or the destinations of a component, for netlist traversal purposes.

Netlist Display Tool: User Interface



6.7 VSS : Typical Design Scenario

VSS was invoked using the allocation computed by AA. The clock was set to 100 ns.

VSS performs scheduling and binding with the allocation and the clock constraint. The synthesized datapath is shown in the figure. The design consists of 8 states. The datapath consists of 5 muxes in addition to the allocated components.

Then the designer invokes the *clock utilization* display on the design. He/she finds that the design has very poor (30%) clock utilization.

VSS – Scenario 1

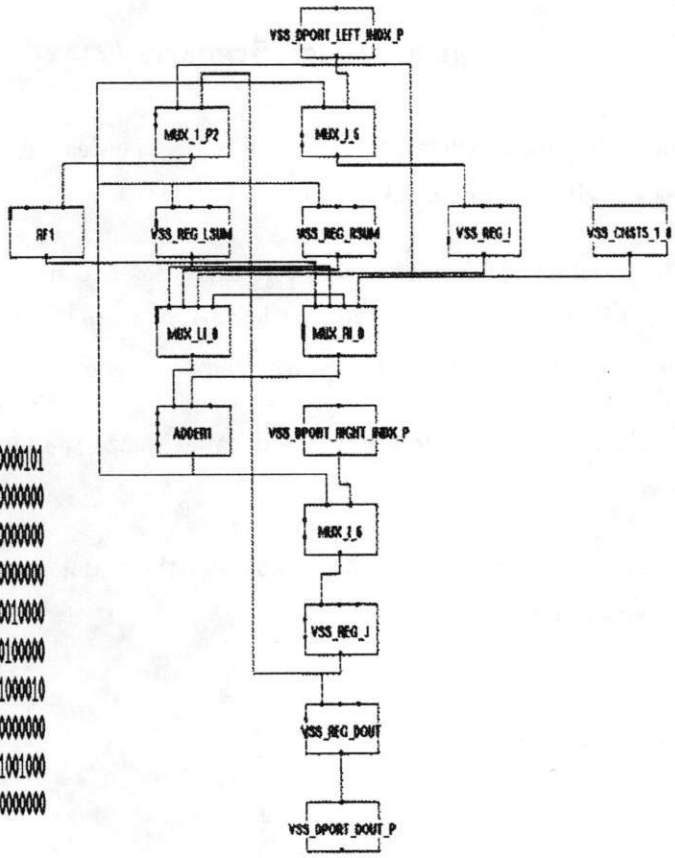
Designer synthesizes for given Allocation Constraints

RESOURCE ALLOCATION CONSTRAINTS

unit	no.	delay	ports	size	stages	functions
ALU	1	20 ns	-	16 bit	1	add, sub, cmp
BUS	3	5 ns	-	16 bit	-	-
REG	7	5 ns	1r, 1w	16 bit	-	read, write
MEM	1	15 ns	1rw	1K w X 16 bit	1	read, write

CLOCK CONSTRAINT = 100 ns

```
--  000  001  00000000110000000000000101
--  001  010  00010000000010001000000000
1-  010  011  00010000000010001000000000
0-  010  000  00000000001000000000000000
-1  011  101  10000101000010010000010000
-0  011  111  10000110000100010001000000
--  101  110  01000000010000100010000010
--  110  010  00010000000010001000000000
--  111  000  10000001000010000010010000
--  000  010  00010000000010001000000000
```



Designer observes poor clock utilization

6.8 VSS : Typical Design Scenario (2)

Since the clock utilization was very poor, the designer changes the clock period. He/she attempts a design with a 10 ns clock period.

The number of temporary registers increases because it is not possible to directly store the output of an ALU into the Register file. It has to be stored into a temporary register before being loaded onto the Register file. Similarly, values read from the register file require additional temporary registers.

In addition to the increase in the datapath, size the number of states in the controller has gone up by three times.

The designer tries to now merge some of the scalar variables *lsum* and *rsum* into the register file along with the array variables.

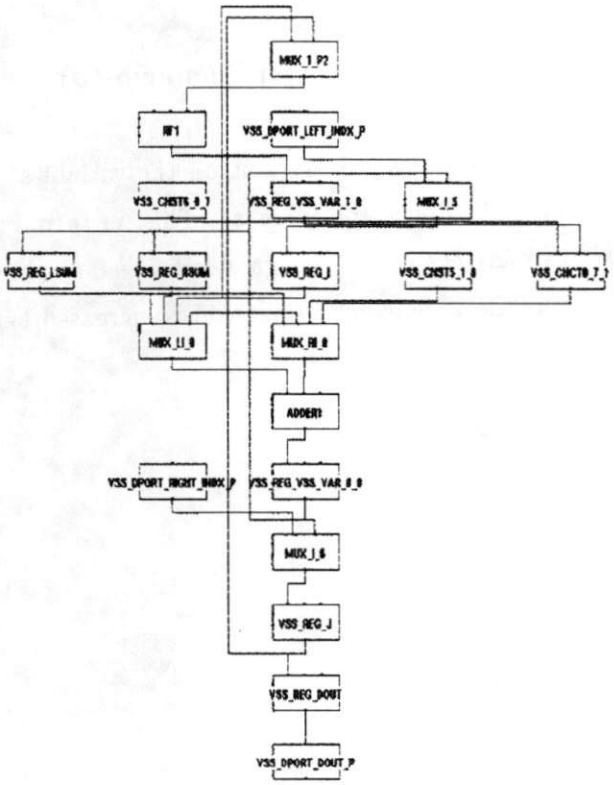
VSS – Scenario 2

Designer changes clock to 10 ns

RESOURCE ALLOCATION CONSTRAINTS (Same as in Scenario 1)

CLOCK CONSTRAINT = 10 ns

--	00000	00001	00000000110000000000001010
--	00001	00010	00000000000000000000000000
--	00010	00011	00010000000000100100000000
--	00011	00100	00010000000000100100000000
1-	00100	00101	00000000000000000000000000
0-	00100	00000	00000000010000000000000000
--	00101	00110	00010000000001001000000000
--	00110	00111	00010000000001001000000000
-1	00111	01001	00000000000000000000000000
-0	00111	01111	00000000000000000000000000
--	01001	01010	00000100000000000000010000
--	01010	01011	10000100000100100000101000
--	01011	01100	10000001000010100000100000
--	01100	01101	01000000000000010010000000
--	01101	01110	01000000010010001001000001
--	01110	10101	00000000000000000000000000
--	01111	10000	00000100000000000000100000
--	10000	10001	10000100000101000000110000
--	10001	10010	10000010000011000000100000
--	10010	10011	10000000000000100010000000
--	10011	10100	10000000100010001000010000
--	10100	10101	00000000000000000000000000
--	10101	10110	00010000000000100100000000
--	10110	00100	00010000000000100100000000



Designer attempts to reduce number of registers

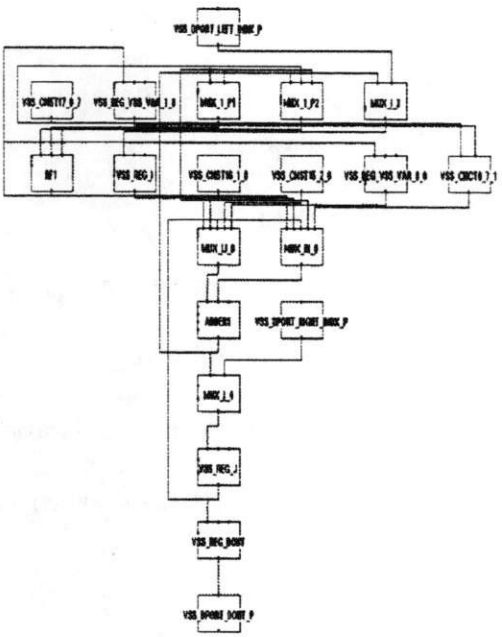
6.9 VSS : Typical Design Scenario (3)

VSS performs address translations on the variables accessed from the memory. The number of read operations in the flowgraph increase. This, in turn, increases the number of states in the design. The number of registers in the design decrease since the variables lsum and rsum are stored inside the memory module. The clock utilization is increased because the address translations occupy the ALU.

VSS – Scenario 3

Designer changes clock back to 100 ns and constrains Lsum, rsum to be merged in along with s

--	0000	0001	00000011000000000000000000101
--	0001	0010	00010000000100000100000000000
1-	0010	0011	00000100010000000000000010000
0-	0010	0000	00000000100000000000000000000
--	0011	0100	00010100000000110000000100000
-1	0100	0110	10000100001000100010000001000000
-0	0100	1001	10000100001000100100000001000000
--	0110	0111	10001100000100000000101001000000
--	0111	1000	01000001000001000001000000010
--	1000	0010	00010000000100000100000000000
--	1001	1010	10001100000100000000110010000000
--	1010	1011	10000010000010000010000001000
--	1011	0010	00010000000100000100000000000



7 AA-VSS : Strengths and Weaknesses

Our synthesis approach with the **AA-VSS** system has several advantages over other comparable approaches. We list some of the important ones in the figure.

7.1 Strengths

The strengths of **AA-VSS** are that it:

- accepts VHDL inputs and outputs so it is easy to verify both the input and output with commercial VHDL simulators.
- is geared for industrial designs.
- supports multiple architectural styles with respect to control pipelining.
- supports multiple memory hierarchy levels.
- uses area, delay, and utilization considerations to make synthesis decisions.
- supports interactivity so that the designer can make some of the decisions himself/herself.

7.2 Weaknesses

The weaknesses of **AA-VSS** are that it:

- does not provide full VHDL capability.
- does not allow users to specify partial structures or partial interconnections - only resource constraints are allowed.

AA-VSS : Strengths and Weaknesses

Strengths

- Simulatable VHDL Input/Output
- Geared for Industrial Descriptions
- Architectural styles (Control and Data Pipelining)
- Memory Hierarchy
- Physical design cost considerations
- User Interactivity

Weaknesses

- VHDL subset
- No partial structures

8 References

[1] D. Gajski, F. Vahid, and S. Narayan, "A System-Design Methodology: Executable-Specification Refinement," in *European Conference on Design Automation*, 1994.

[2] *IEEE Standard VHDL Language Reference Manual*, 1988.

[3] B. Pangrle and D. Gajski, "State Synthesis and Connectivity Binding for Microarchitecture Compilation," in *Proc. of the IEEE Conf. on Computer Aided Design.*, pp. 210–213, IEEE, November 1986.

[4] N. D. Holmes and D. D. Gajski, "An Algorithm for Generation of Behavioral Shape Functions," in *European Conference on Design Automation*, 1994.

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

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

[7] S. Narayan and D. Gajski, "System Clock Estimation based on Clock Slack Minimization," in *Proc. 1st EURO-DAC, Hamburg*, 1992.