AFRL-IF-WP-TR-2002-1524

MULTICOMPONENT SYNTHESIS THROUGH ARCHITECTURAL PARTITIONING

Dr. David Springer
Dr. Elizabeth Lagnese

DASYS, Inc.
3547 Shadeland Avenue
Pittsburgh, PA 15212

DECEMBER 2001

Final Report for 27 March 1992 – 10 June 1996

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

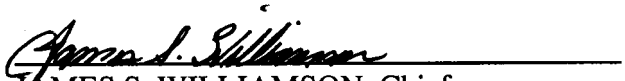
NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE US GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

DARRELL BARKER
Project Engineer
Embedded Info Sys Engineering Branch
Information Technology Division



JAMES S. WILLIAMSON, Chief
Embedded Info Sys Engineering Branch
Information Technology Division
Information Directorate

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
December 2001	Final	03/27/1992 – 06/10/1996

4. TITLE AND SUBTITLE
MULTICOMPONENT SYNTHESIS THROUGH ARCHITECTURAL PARTITIONING

5a. CONTRACT NUMBER
F33615-92-C-1030

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62204F

6. AUTHOR(S)
Dr. David Springer
Dr. Elizabeth Lagnese

5d. PROJECT NUMBER
6096

5e. TASK NUMBER
20

5f. WORK UNIT NUMBER
22

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DASYS, Inc.
3547 Shadeland Avenue
Pittsburgh, PA 15212

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Information Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson Air Force Base, OH 45433-7334

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/IFTA

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-IF-WP-TR-2002-1524

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A system synthesis tool allows a designer to synthesize designs onto different types of system implementations from a single system specification. This may include boards of FPGAs for prototyping, a low cost initial design using smaller ASICs on a board, and a final high-performance implementation on an MCM or single chip. System synthesis tools allow a designer to synthesize each different stage of a system's lifetime from early prototype to final high-volume design as well as explore possible design tradeoffs between cost and performance.

In this project, DASYS has developed a first system synthesis tool, Dasynth, that takes behavioral system specifications and synthesizes RTL structural specifications suitable for current commercial synthesis tools. Dasynth supports a design flow that partitions the design's behavior before high-level synthesis. This early partitioning both helps high level synthesis tools to meet design constraints and decomposes the synthesis problem, thereby improving the synthesis tool's performance.

15. SUBJECT TERMS
system synthesis, design partitioning, electronic computer-aided design, VHDL

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Darrell Barker
Unclassified	Unclassified	Unclassified	SAR	24	19b. TELEPHONE NUMBER *(Include Area Code)* (937) 255-6548 x3605

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

Table of Contents

List of Figures

1.0 Introduction

Highly complex integrated circuit systems are generally composed of multiple components which may be modules, or packages, or areas on a large chip. A system synthesis tool allows a designer to synthesize designs for different types of system implementations from a single system specification. A design life for an architecture may involve multiple instantiations using different technologies, as shown in Figure 1. The various instantiations require different design cycles, and may include boards of Field Programmable Gate Arrays (FPGAs) and Programmable Logic Devices (PLDs) for prototyping, a low cost initial design using smaller Application Specific Integrated Circuits (ASICs) on a board, and a final high performance implementation on a Multi-Chip Module (MCM) or single chip. System synthesis tools allow a designer to synthesize each different stage of a system's life cycle from early prototype to final high volume design, and beyond for model year updates. Each of these of implementations may require a different physical partitioning of the system's behavior, and consequently a different structural implementation of each partition. In addition to supporting the different stages of a system's life cycle, system synthesis allows design space exploration of cost/performance trade-offs. DASYS' new tool, called Dasynth, supports the interactive partitioning of a design's behavior across multiple physical partitions and the interactive or automatic synthesis of those partitions and their interconnect to a target technology.

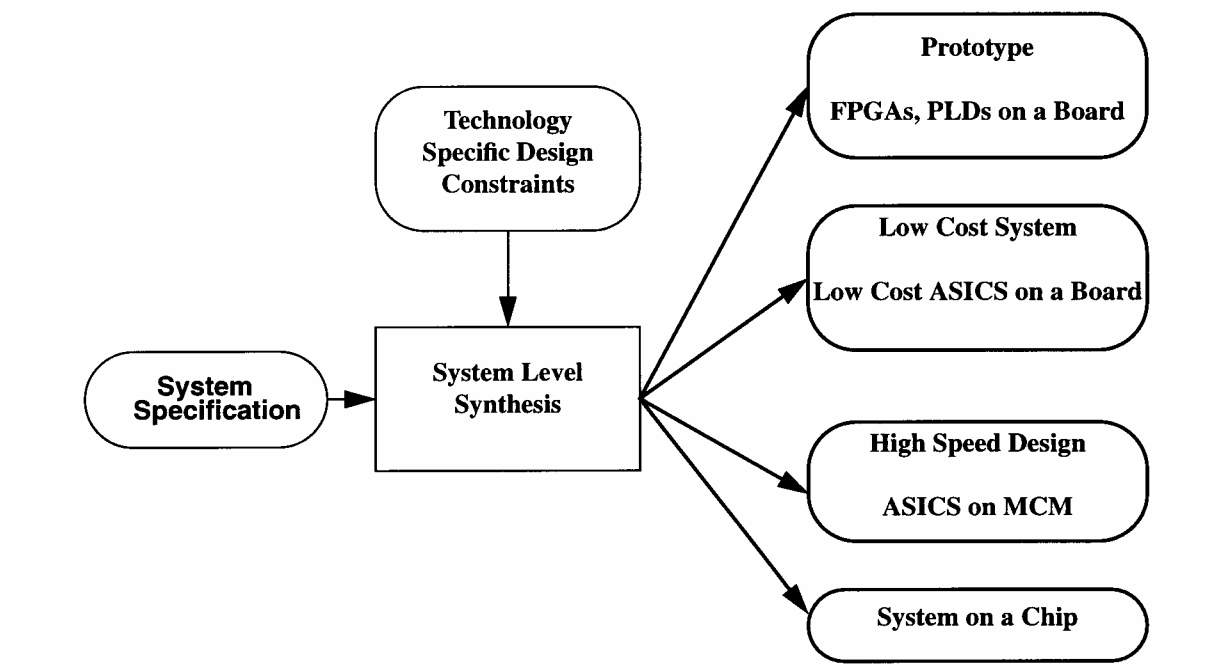


Figure 1. System Synthesis

Dasynth was developed to demonstrate an approach to system synthesis based on architectural partitioning. In this approach, a system is described using a behavioral system specification, which is then partitioned and synthesized to create a structural register transfer level (RTL) description for each behavioral partition.

2.0 System Synthesis Using Architectural Partitioning

Architectural partitioning operates on a behavioral description, partitioning the behavior onto physical packages before high level synthesis (HLS) is begun. This early partitioning is imperative since it allows early introduction of packaging information into the design process, ensuring that design constraints can be met. Early partitioning also provides an early decomposition of the synthesis problem. Each partition can be synthesized separately reducing the time needed to synthesize the system. This brings about shorter design times for rapid prototyping or time for each module to be better optimized for high performance applications.

DASYS' approach employs three major new concepts: abstract behavioral specifications, architectural partitioning, and high level synthesis. Each will be described in this section.

2.1 Behavioral Specification

A behavioral system specification is composed of two major parts, a description of the system's behavior and system constraints.

A system's behavior specifies the actions which must be accomplished to perform the system's function. The system's behavior can span a range of levels of abstraction. The lowest level of behavioral abstraction is a cycle-by-cycle behavior. In this case the system's state sequence is defined. However the description of each state's behavior does not imply specific hardware such as Arithmetic Logic Units (ALUs), registers or busses. The most abstract behavioral description is an algorithmic behavior wherein the system's state sequence is not specified.

The Hardware Description Language (HDL) fragments shown in Figure 2 demonstrate these two extremes. Both describe a matrix cross product. In either of the cases shown in Figure 2, a variable in the HDL (e.g., crossx or P1) does not imply a specific register, nor do the operations * or - imply a specific multiplier or subtracter. During system synthesis the exact register and functional units for these constructs will be defined. The difference between the two descriptions is that in part b of the figure, clock edges are defined, specifying that the operations between the clock edges must be executed during the given clock cycle.

In addition to a system's behavior, a system specification defines the system's design constraints. The most common types of constraints are clock period, timing, area, and power.

***a.)* Algorithmic Behavior**

Constraints: Time = 280ns
crossx = A[l]*B[3] - A[3]*B[2]
crossy = A[3]*B[l] - A[l]*B[3]
crossz = A[l]*B[2] - A[2]*B[l]

***b.)* Cycle-By Cycle-Behavior**

Constraint: clock = 25Mz

at posedge(clock)	at posedge(clock)
Pl = A[I]*B[3]	crossy = Pl - P2
at posedge(clock)	Pl = A[l]*B[2]
P2 = A[3]*N[2]	at posedge(clock)
at posedge(clock)	P2 = A[2]*B[l]
crossx = Pl - P2	at posedge(clock)
pl = A[3]*B[2]	crossx = Pl - P2
at posedge(clock)	
P2 = A[l]B[3]	

Figure 2. Example Behavioral Specification

Timing constraints specify the time between any two points in the systems behavior. Timing constraints might be defined in terms of clock cycles or absolute time. For example, two operators may be constrained to be exactly one cycle apart or 15ns apart.

Area constraints limit the amount of board or chip area which maybe used by the system. This constrains the resources which may be used to implement the systems behavior. Area constraints can also be defined indirectly in terms of allowed resources. For example instead of providing a board area, a designer may specify a number of specific packages. Instead of specifying a chip's area, the designer may specify a number of specific components that may be used.

Power constraints specify the amount of power which may be consumed by a system or how the power dissipation must be distributed across the design.

2.2 Architectural Partitioning

Architectural partitioning is engaged before synthesis, operating directly on the behavioral specification. Architectural partitioning determines the number of chips or areas on a chip to used for the design and the subset of the behavior that will be implemented in each partition.

In current design synthesis methodologies, partitioning is left to the final design stages (e.g., gate level). However, this may result in irreconcilable constraints; structural implementation decisions made without regard to partitioning issues may result in a design that cannot be partitioned according to requirements. Consequently, partitioning should be one of the first stages in the design process, so that later synthesis stages can take advantage of partitioning information and so that desirable partitioning moves are not blocked.

Behavioral partitioning requires criteria that are different from those used for structural or physical partitioning. For any partitioning, the goal is to optimize certain physical criteria or meet physical constraints, but in behavioral partitioning, without the knowledge of physical characteristics, these criteria and constraints can not be directly measured. Instead, behavioral criteria are used which strongly influence our ability to meet the physical criteria. The following is a partial list of behavioral partitioning criteria and how they effect the design's physical implementation.

- Minimizing the communications between partitions minimizes the need for pins and interconnect between partitions. This maximizes the chance of the HLS tools to meet pin constraints or board interconnect constraints.

- Minimizing the number of data transfers between partitions that lie on the design's critical path minimizes performance degradation due to off chip data transfers.

- Maximizing the locality of control within partitions minimizes the synchronization that must be performed between chips, and minimizes the size of the controller on each chip.

- Maximizing the similarity of the functionality within partitions maximizes resource sharing and therefore minimizes the size of chips needed to implement each partition.

2.3 High Level Synthesis

After the system's behavior has been partitioned, high level synthesis tools are used to create an RTL implementation of the architecture. This RTL representation must be suitable for input to current logic synthesis tools, data path compilers, or for use with RTL cell libraries. High Level Synthesis performs two major tasks: it creates the a datapath that implements the desired behavior and it defines a state machine to control the datapath. Synthesis of a partitioned design can take advantage of the partitioning information if the tools know about resource requirements and the difference between inter and intra chip interconnect and pin limitations.

3.0 Dasynth

Dasynth is a system synthesis tool being developed by DASYS, which takes behavioral system specifications and synthesizes Register Transfer Level (RTL) structural specifications suitable for current synthesis tools. Dasynth performs two major design steps, as shown in Figure 3. First it creates a high level partitioning (or floor plan) for the design before synthesis, and then it synthesizes an RTL structural description for each partition as well as the interconnect between partitions. It is important to understand that the tool partitions the behavior before synthesis. This early partitioning is imperative, since it allows early introduction of packaging information into the design process, ensuring that design constraints can be met. Early partitioning also provides an early decomposition of the synthesis problem. Each partition can be synthesized separately reducing the time needed to synthesize the system allowing shorter design times for rapid prototyping or time for each module to be better optimized for high performance applications.

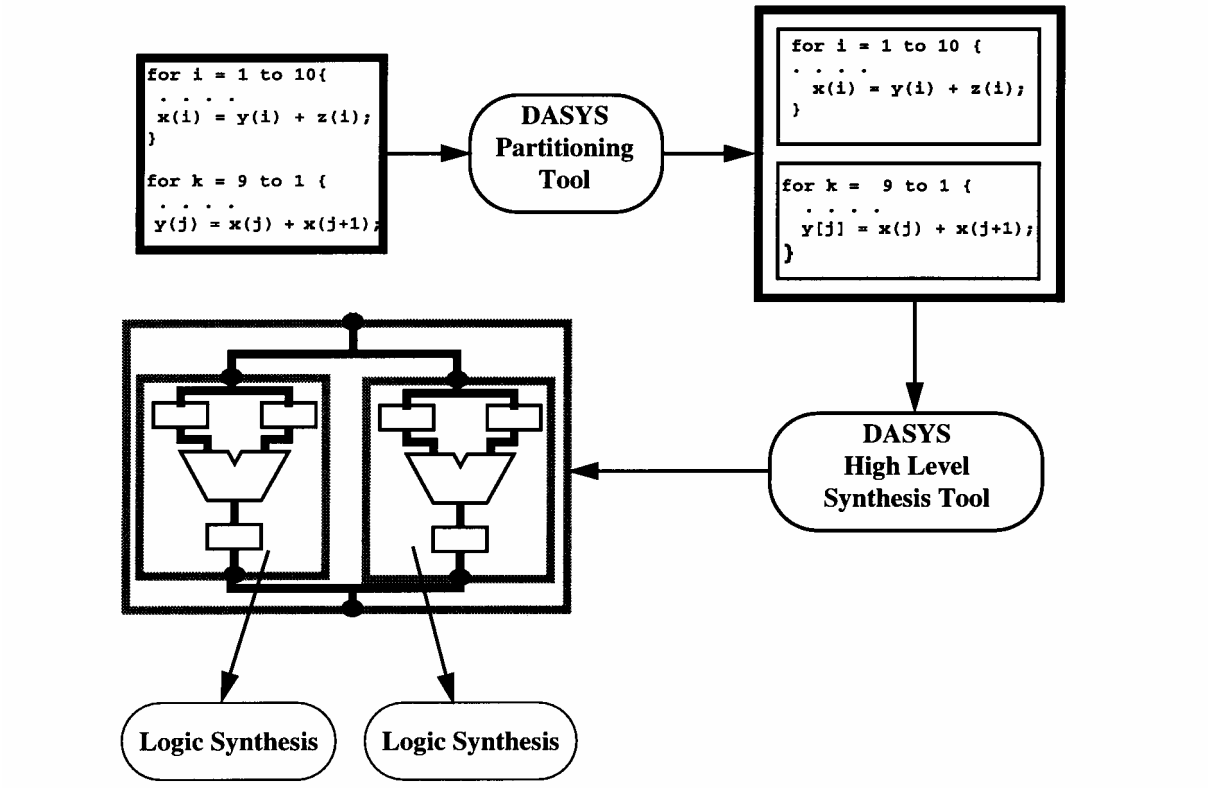


Figure 3. Dasynth Design Flow

3.1 Dasynth Input and Output

Figure 4 shows the input/output requirements for Dasynth. Dasynth takes four types of inputs: a system's behavior, design constraints, an RTL component library, and an opcode table. It produces an RTL structure of the synthesized design. Each of the input and output files is described in greater detail below.

3.1.1 Behavior

Dasynth takes as input a description of a system's behavior. A system's behavior specifies the actions that must be accomplished to perform the system's function. The system's behavior can span a range of levels of abstraction. Dasynth version 0.0 takes as input the most abstract behavioral description, an algorithmic behavior, written in the hardware description language, VHDL. In this case the system's state sequence is not specified.

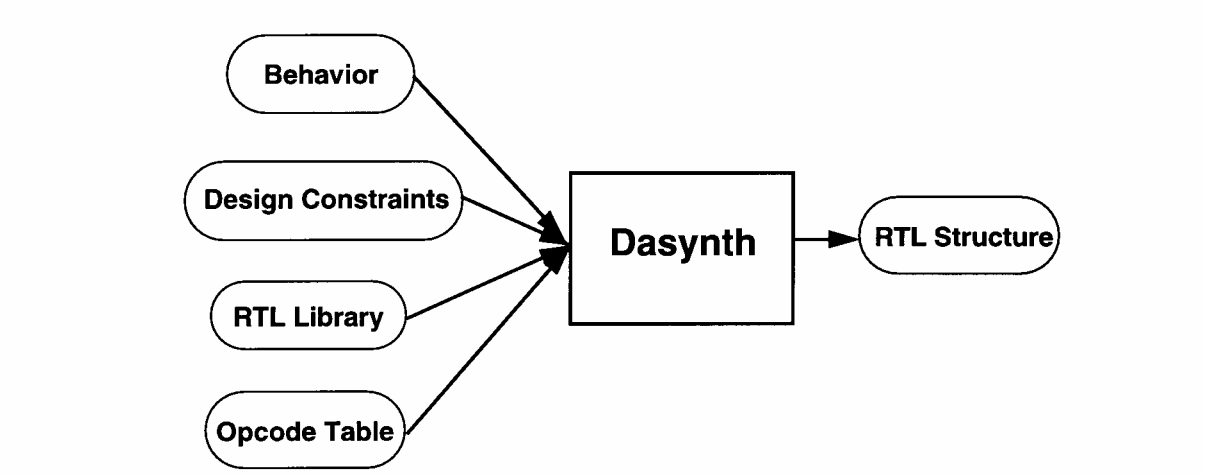


Figure 4. Dasynth Input/Output Description

3.2 Design Constraints

Dasynth supports three types of design constraints: cycle length, timing constraints, and resource constraints. The cycle length specifies the clock period. Timing constraints can be placed between any two operators in the behavior. Timing constraints can be defined in terms of clock cycles or absolute time. For example, two operators may be constrained to be exactly one cycle apart or 15ns apart. The types of timing constraints supported are at least n, at most n, exactly n, more than n, less than n, and not n, where n is a number of cycles or nanoseconds between the constrained events.

Resource constraints limit the amount of hardware used in the data path. Dasynth version 0.0 allows the user to specify resource constraints in terms of the number of RTL components in each partition.

3.2.1 Opcode Table

Dasynth reads a set of files that define an opcode table. This table defines the legal operations in the behavior and assigns a symbolic opcode to each operation. These opcodes are used to map VHDL operations onto components in the RTL library. The opcode table also allows the user to specify additional operations not supported by the language, for example an FFT operator. To define a new operator, the user writes a function describing the new operator and the opcode table will map calls to this function into an opcode. Dasynth treats the new operator like any other, meaning it will be partitioned, scheduled and mapped onto cells in the library. The ability to define new operations allows easy integration of conversion functions for packages like standard logic in VHDL, as well as the use of off the shelf components such as a FIFO.

3.2.2 RTL Component Library

Dasynth employs an RTL component library. This library contains RTL parts for building the design's datapaths. Such parts include functional units (ALUS, adders, shifters, etc.), registers, buffers and muxes. Current standards or pending standards for writing libraries at this level are Xilinx Blox for FPGAS or Synopsis DesignWare.

Each library entry is composed of a simulatable HDL description of a cell plus additional information needed for synthesis. The synthesis information provided by the library includes the cell's area, a description of the operations performed by the cell (using the opcodes from the opcode table), the ports used by each operator, and a worst case time for each operator.

The library also contains a template for the controller. This template defines the specific requirements of the logic synthesis tool which will be used to synthesize the controller.

3.2.3 RTL Structure

Dasynth produces an RTL structural specification of the system. A structural specification of the system is composed of two components: a description of the system's controlling state machine and instantiations of RTL library modules which define the data path. The description of the state machine is a case statement format suitable for input to current logic synthesis tools. Figure 5 shows an example of the structural specification for the cross product of Figure 2.

```
case state_variable

000  Mult_control <= 1
     A_address <= 2
     B_address <= 3
     P1_enable <= 1
     P2_enable <= 0
     crossx_enable <= 0
     crossy_enable <= 0
     crossz_enable <= 0
     state_variable <= 001
001  Mult_control <= 1

     ...
```

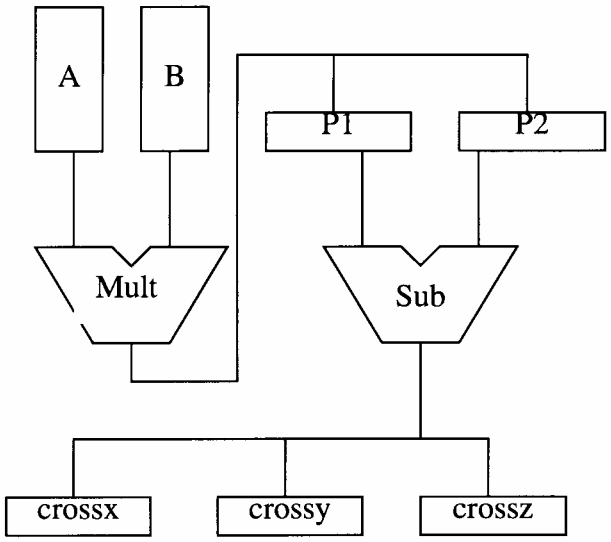


Figure 5. Example Structural Description

3.3 Dasynth Organization

As shown in Figure 6, Dasynth is comprised of the Graphical User Interface (GUI), the Synthesis Tools, the VHDL Interface, and the Library Manager. Each of these components is described in the following sections.

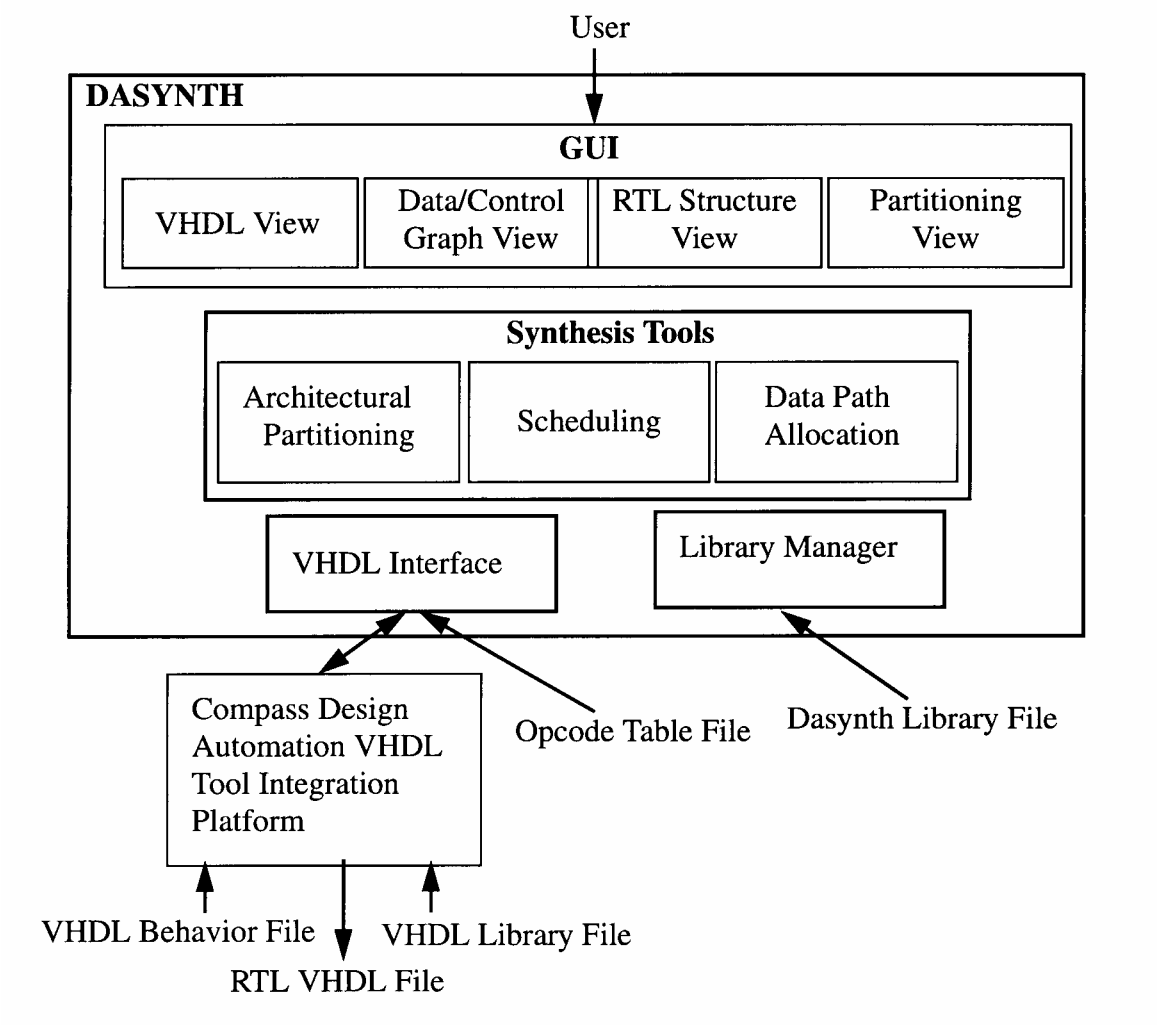


Figure 6. Dasynth Origanization

3.3.1 GUI

Dasynth's Graphical User Interface provides four views of the design: VHDL, Data/Control Graph, RTL Structure, and Partitioning. The VHDL window displays the VHDL input. The Data/Control Graph window displays a hierarchical description of the design's data and control flow. The RTL structure window shows a schematic of the design's RTL structure. The partitioning window shows graphical representations of design partitions and their relationship to each other based on select partitioning criteria.

One of Dasynth's most powerful features is the ability to graphically display the correlation between each view of the design. For example if a designer selects an operator in the HDL window, the system can highlight that operator in the state machine view, the RTL cell

implementing that operator in the schematic and the partition in the partitioning window. Another example: a designer can select a partition in the partitioning window and see all operators in the HDL window or in the control flow graph window which are in the selected partition. This ability to correlate the different design views allows easy understanding of the system's actions and the relationship between the behavioral input and structural RTL output.

3.3.2 Synthesis Tools

Dasynth's synthesis tools are: an interactive architectural partitioning tool, an automatic scheduler, and a data path allocator. The partitioning tool allows the designer to partition based on a number of criteria, such as data similarity, position on the critical paths, operator similarity, and locality of control.

The scheduler defines the state sequence for the design's controller. The scheduler is constraint driven and will seek to maximize performance while meeting both resource and timing constraints. The scheduler supports iterative refinement of the system's state machine definition by allowing constraints to be changed and the design or part of the design to be rescheduled. As the design is scheduled the control/data flow view of the design is modified to represent the current state sequence.

The hardware allocator creates a new module for each partition. It allocates ports for each module and the interconnect between them. Within each module, it allocates and connects cells from the RTL library for the data path and defines the interconnect to the system's controller.

3.3.3 VHDL Front End

The VHDL front end provides an interface between Compass Design Automations VHDL Tool Integration Platform (VTIP), and DASYS' internal data structures. The interface translates between VTIP's VHDL data structures and Dasys internal format. This includes translating the VHDL input into the internal data structures before synthesis, translating the VHDL descriptions of library components into the library manager's data structures, and translating intemal data structures back to VHDL after synthesis. VTIP is responsible for analyzing and producing valid VHDL.

3.3.4 Library Manager

Dasynth has an independent library manager that provides a uniform functional interface to replacable component libraries. The library manager allows the synthesis tools to use the library in a technology-independent way. This allows the library to be radically changed for new technologies with no affect on the rest of the system.

4.0 Tool Evaluation

Dasynth has been evaluated internally by DASYS personnel, using multiple small descriptions, and externally by Raytheon personnel. Feedback from Raytheon indicates that the tool, with appropriate enhancements (see Section 5.0), would be useful in reducing their current design cycle time by approximately 50 percent.

5.0 Lessons Learned

This section outlines the major lessons learned during completion of this project. These represent both technical requirements and feedback from designers at Raytheon.

5.1 Constraints

When we first began the project we asked designers at Raytheon and elsewhere how they wanted to specify design constraints. Since they knew what data path elements were needed when they designed a chip they suggested that resource constraints be in terms of the number of specific cells in the data path. However after trying the tool they decided that specifying such constraints for each partition was not desirable. They preferred the tool determine the resource constraints for each partition given the timing constraints. This was more flexible, allowing easier exploration of the design space.

5.2 Scheduling

The above change of view on constraints requires a change in how scheduling is performed. The scheduler was designed to fit a design onto a specified set of data path elements while meeting timing constraints. With the new view of scheduling the scheduler needs to be driven primarily by the timing constraints.

5.3 Using RTL Cell Libraries

Dasynth currently requires an RTL cell library for scheduling and allocation. The final design's data path is composed of instantiations from this cell library. This was a major departure from previous high level synthesis systems which allowed almost arbitrary combination of functionality into functional units. Designers at Raytheon suggested that there might be situations where the cell library approach is too restrictive. This might be the case during very early design stages where the main task is design space exploration, or in an environment where all functional units are hand crafted. In either of these cases a more general mechanism for choosing components might be desirable.

5.4 Allocation

In the past, architectural partitioning information was only used as a guide for HLS tasks by including the partitions in the cost function during allocation and scheduling. Even though the design was partitioned behaviorally, structurally it was still treated as a single design. In this project, the goal was to create physical MCM partitions, so we viewed the partitions as distinct pieces of hardware represented as a separate VHDL modules. As a result, we needed to specify ports for each partition and the interconnect between them. The assignment of these ports can have significant affect on both the interpartition and intrapartition interconnect. As a result, port assignment is a major new task during allocation of multi component systems.

5.5 Behavioral Descriptions

On starting this project, we knew that most commercial design sites do not write abstract behavioral descriptions of their designs, since their design tools cannot make use of them. However, we did not anticipate the difficulty we would have in defining such a description such that a commercial designer could write one. Our work to date has provided us with the prototype needed to introduce high level concepts (abstract behavioral descriptions, architectural partitioning, high level synthesis) in a mutually understandable language.

5.6 MCM Design

At the abstract behavioral level at which Dasynth does partitioning, MCMs resemble most other partitioned designs. However, Raytheon personnel pointed out the importance of addressing varying die sizes within a single design, and power dissipation issues that do not arise in, for example, FPGA designs.

5.7 VHDL Signals

VHDL's timing model specifies that signals change only at specific points in the description and that their value is present between those points. To match this timing model during synthesis, signals must be stored to assure that their value does not change inappropriately. In a partitioned design this means deciding in which partition to store the signal. When partitions were only suggestions this was not necessary.

5.8 Targeting Downstream Tools

The VHDL subset accepted by downstream synthesis tools was more restrictive and varied than anticipated. As a result, significantly different output modules must be used for each down stream tool. It is not clear that we have sufficient control over the VHDL output using a third party VHDL tool. While Compass Design Automation's tools create nice VHDL output, it is not legal for all downstream tools. As a result, a manual edit or post-processing stage is needed to manipulate the VHDL into its final form.

6.0 Conclusion

A system synthesis tool allows a designer to synthesize designs onto different types of system implementations from a single system specification. This may include boards of FPGAs for prototyping, a low cost initial design using smaller ASICs on a board, and a final high-performance implementation on an MCM or single chip. System synthesis tools allow a designer to synthesize each different stage of a system's lifetime from early prototype to final high volume design as well as explore possible design trade-offs between cost and performance.

In this project, DASYS has developed a first system synthesis tool, Dasynth, that takes behavioral system specifications and synthesizes RTL structural specifications suitable for current commercial synthesis tools. Dasynth supports a design flow that partitions the design's behavior before high level synthesis. This early partitioning both helps high level synthesis tools to meet design constraints and decomposes the synthesis problem, thereby improving the synthesis tool's performance.